TEMA: **1a**
Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

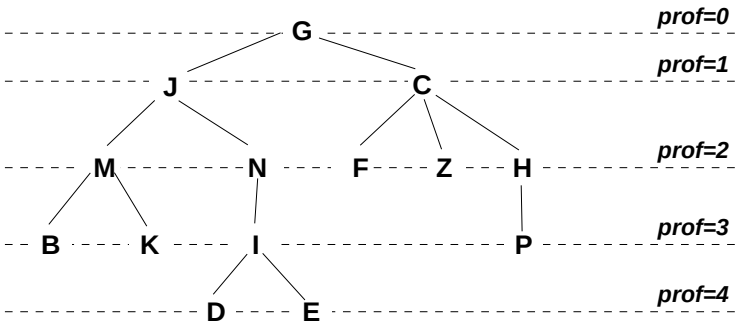
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. Tema 1a. [11 de Julio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado mediante vectores de bits. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias. (a) `ANULA`, (b) `UNION`, (c) `INTERSECCION`, (d) `MIEMBRO`, (e) `MIN`, (f) `INSERTA` y (g) `SUPRIME`.

Ej. 2.- Escribir un función `function CUENTA_PROF(n:nodo; m:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raíz es `n` que están a profundidad `m` o menor (con respecto a `n`). Por ejemplo, para el árbol de la figura debe retornar

```
CUENTA_PROF(G,2,A) -> 8
CUENTA_PROF(J,1,A) -> 3
CUENTA_PROF(N,3,A) -> 4
```



Usar las primitivas de árbol ordenado orientado siguientes:
`HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. *Sugerencia:* Hacer la función *recursiva*. Notar que, por ejemplo:

```
CUENTA_PROF(G,2,A) = 1 + CUENTA_PROF(J,1,A) + CUENTA_PROF(C,1,A)
```

La recursividad de la función debe cortar cuando `n`$= \Lambda$ o $m < 0$.

Ej. 3.- Uso básico de TAD's:

(a) Escribir un procedimiento `procedure ROTA(var L:lista; n:integer);` que manda los primeros `n` elementos de la lista al final de la misma. Por ejemplo, si $L = \{1, 3, 5, 4, 2, 6\}$, entonces `ROTA(L,2)` la deja en $L = \{5, 4, 2, 6, 1, 3\}$.

(b) Escribir una función `CUENTA(var P:pila) : integer;` que cuenta los elementos de una pila. *Ayuda: utilizar una pila auxiliar.*

Ej. 4.- [LIBRES] Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Q, R, A, B, L, M, N\}$,
- `ORD_POST` $=\{A, B, R, M, N, L, Q\}$.

(b) **Clasificación por montículos:** Dados los enteros $\{30, 10, 12, 15, 7, 2\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

Ej. 5.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) La ventaja de la representación del `TAD COLA` por *arreglo circular* con respecto a la representación por celdas enlazadas es que ...

- ☐ ... requiere menos memoria ya que no necesita los punteros para enlazar celdas.
- ☐ ... es más simple.
- ☐ ... permite inserciones/supresiones en el medio de la cola.
- ☐ ... el orden de las operaciones `FRENTE, QUITA, PONE` es $O(\log n)$.

(b) Una ventaja del método de clasificación por selección, en comparación con otros algoritmos *lentos*, es que realiza sólo n *intercambios*...

- ☐ ... cuando el vector está ordenado.
- ☐ ... cuando el vector está desordenado.
- ☐ ... siempre.
- ☐ ... a veces.

(c) Una de las desventajas de implementar el `TAD LISTA` con arreglos es que ...

- ☐ ... las inserciones en el medio de la lista son $O(n)$.
- ☐ ... las inserciones en el medio de la lista son $O(1)$.
- ☐ ... ocupa más memoria.
- ☐ ... no se pueden insertar/suprimir elementos.

(d) La búsqueda *exhaustiva* del camino de longitud mínima para el *"Problema del Agente Viajante"* (PAV) tiene un tiempo de ejecución (n es el número de ciudades) ...

- ☐ $O(n!)$
- ☐ $O(1)$
- ☐ $O(n)$
- ☐ $O(n^2)$

TEMA: **1b**
Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

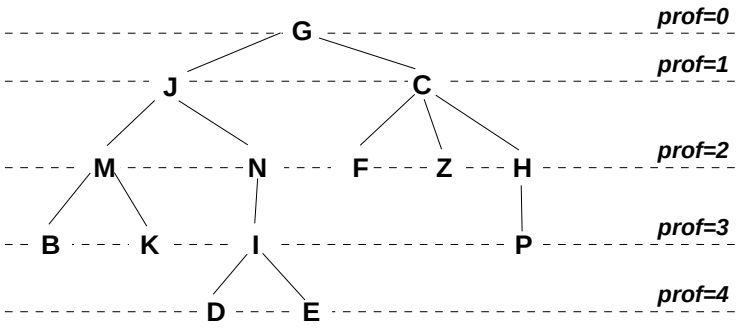
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. Tema 1b. [11 de Julio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado mediante vectores de bits. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias. (a) `ANULA`, (b) `UNION`, (c) `INTERSECCION`, (d) `MIEMBRO`, (e) `MIN`, (f) `INSERTA` y (g) `SUPRIME`.

Ej. 2.- Escribir un función `function CUENTA_PROF(n:nodo; m:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raíz es `n` que están a profundidad `m` o menor (con respecto a `n`). Por ejemplo, para el árbol de la figura debe retornar

```
CUENTA_PROF(G,2,A) -> 8
CUENTA_PROF(J,1,A) -> 3
CUENTA_PROF(N,3,A) -> 4
```



Usar las primitivas de árbol ordenado orientado siguientes:
`HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. *Sugerencia:* Hacer la función *recursiva*. Notar que, por ejemplo:

```
CUENTA_PROF(G,2,A) = 1 + CUENTA_PROF(J,1,A) + CUENTA_PROF(C,1,A)
```

La recursividad de la función debe cortar cuando `n`$= \Lambda$ o $m < 0$.

Ej. 3.- Uso básico de TAD's:

(a) Escribir un procedimiento `procedure ROTA(var L:lista; n:integer);` que manda los primeros `n` elementos de la lista al final de la misma. Por ejemplo, si $L = \{1, 3, 5, 4, 2, 6\}$, entonces `ROTA(L,2)` la deja en $L = \{5, 4, 2, 6, 1, 3\}$.

(b) Escribir una función `CUENTA(var P:pila) : integer;` que cuenta los elementos de una pila. *Ayuda: utilizar una pila auxiliar.*

Ej. 4.- [LIBRES] Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, T, Q, L, M, N, P\}$,
- `ORD_POST` $=\{Q, M, N, P, L, T, Z\}$.

(b) **Clasificación por montículos:** Dados los enteros $\{31, 11, 13, 16, 8, 8\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

Ej. 5.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) La ventaja de la representación del `TAD COLA` por *arreglo circular* con respecto a la representación por celdas enlazadas es que ...

- ☐ ... el orden de las operaciones `FRENTE, QUITA, PONE` es $O(\log n)$.
- ☐ ... es más simple.
- ☐ ... requiere menos memoria ya que no necesita los punteros para enlazar celdas.
- ☐ ... permite inserciones/supresiones en el medio de la cola.

(b) Una ventaja del método de clasificación por selección, en comparación con otros algoritmos *lentos*, es que realiza sólo n *intercambios*...

- ☐ ... a veces.
- ☐ ... cuando el vector está ordenado.
- ☐ ... cuando el vector está desordenado.
- ☐ ... siempre.

(c) Una de las desventajas de implementar el `TAD LISTA` con arreglos es que ...

- ☐ ... no se pueden insertar/suprimir elementos.
- ☐ ... las inserciones en el medio de la lista son $O(1)$.
- ☐ ... las inserciones en el medio de la lista son $O(n)$.
- ☐ ... ocupa más memoria.

(d) La búsqueda *exhaustiva* del camino de longitud mínima para el *"Problema del Agente Viajante"* (PAV) tiene un tiempo de ejecución (n es el número de ciudades) ...

- ☐ $O(1)$
- ☐ $O(n)$
- ☐ $O(n^2)$
- ☐ $O(n!)$

TEMA: **1c**
Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

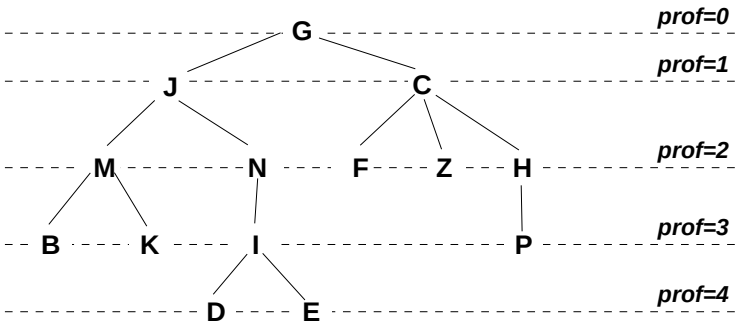
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. Tema 1c. [11 de Julio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado mediante vectores de bits. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias. (a) `ANULA`, (b) `UNION`, (c) `INTERSECCION`, (d) `MIEMBRO`, (e) `MIN`, (f) `INSERTA` y (g) `SUPRIME`.

Ej. 2.- Escribir un función `function CUENTA_PROF(n:nodo; m:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raíz es `n` que están a profundidad `m` o menor (con respecto a `n`). Por ejemplo, para el árbol de la figura debe retornar

```
CUENTA_PROF(G,2,A) -> 8
CUENTA_PROF(J,1,A) -> 3
CUENTA_PROF(N,3,A) -> 4
```



Usar las primitivas de árbol ordenado orientado siguientes:
`HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. *Sugerencia:* Hacer la función *recursiva*. Notar que, por ejemplo:

```
CUENTA_PROF(G,2,A) = 1 + CUENTA_PROF(J,1,A) + CUENTA_PROF(C,1,A)
```

La recursividad de la función debe cortar cuando `n`$= \Lambda$ o $m < 0$.

Ej. 3.- Uso básico de TAD's:

(a) Escribir un procedimiento `procedure ROTA(var L:lista; n:integer);` que manda los primeros `n` elementos de la lista al final de la misma. Por ejemplo, si $L = \{1, 3, 5, 4, 2, 6\}$, entonces `ROTA(L,2)` la deja en $L = \{5, 4, 2, 6, 1, 3\}$.

(b) Escribir una función `CUENTA(var P:pila) : integer;` que cuenta los elementos de una pila. *Ayuda: utilizar una pila auxiliar.*

Ej. 4.- [LIBRES] Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Z, Q, L, R, P, D, N\}$,
- `ORD_POST` $=\{Z, L, Q, P, R, N, D, A\}$.

(b) **Clasificación por montículos:** Dados los enteros $\{32, 12, 14, 17, 9, 4\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

Ej. 5.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) La ventaja de la representación del `TAD COLA` por *arreglo circular* con respecto a la representación por celdas enlazadas es que ...

- ☐ ... es más simple.
- ☐ ... permite inserciones/supresiones en el medio de la cola.
- ☐ ... el orden de las operaciones `FRENTE, QUITA, PONE` es $O(\log n)$.
- ☐ ... requiere menos memoria ya que no necesita los punteros para enlazar celdas.

(b) Una ventaja del método de clasificación por selección, en comparación con otros algoritmos *lentos*, es que realiza sólo n *intercambios*...

- ☐ ... a veces.
- ☐ ... cuando el vector está ordenado.
- ☐ ... cuando el vector está desordenado.
- ☐ ... siempre.

(c) Una de las desventajas de implementar el `TAD LISTA` con arreglos es que ...

- ☐ ... las inserciones en el medio de la lista son $O(1)$.
- ☐ ... no se pueden insertar/suprimir elementos.
- ☐ ... las inserciones en el medio de la lista son $O(n)$.
- ☐ ... ocupa más memoria.

(d) La búsqueda *exhaustiva* del camino de longitud mínima para el *"Problema del Agente Viajante"* (PAV) tiene un tiempo de ejecución (n es el número de ciudades) ...

- ☐ $O(1)$
- ☐ $O(n!)$
- ☐ $O(n^2)$
- ☐ $O(n)$

Apellido y Nombre: _____

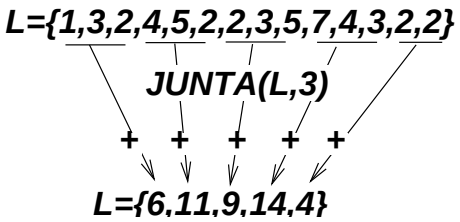
Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
Examen Final. [1 de Agosto de 2002]

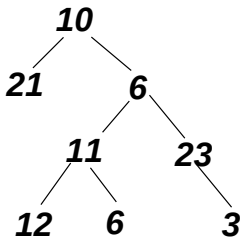
Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado *listas enlazadas clasificadas*. Es decir, implementar en Pascal los siguientes procedimientos/funciones listados abajo. Incluir todas las definiciones de tipo necesarias. (a) `ANULA(A)`, (b) `UNION(A,B,C)`, (c) `INTERSECCION(A,B,C)`, (d) `DIFERENCIA(A,B,C)`, (e) `MIEMBRO(x,A)`, (f) `MIN(A)`, (g) `INSERTA(x,A)` y (h) `SUPRIME(x,A)`.

Ej. 2.- Escribir un procedimiento `procedure JUNTA(var L: lista; n:integer);` que dada una lista `L`, agrupa de a `m` elementos dejando su suma (ver figura). Usar las siguientes primitivas del `TAD LISTA`: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



Ej. 3.- Uso básico de TAD's:

(a) Escribir una función `function MAXIMO_PAR(n:nodo; A:arbol):integer;` que retorna el máximo de las etiquetas **pares** de un árbol binario. En el caso del árbol de la figura debe retornar 12. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQUIERDO(n,A)`, `HIJO_DERECHO(n,A)` y `ETIQUETA(n,A)`,



(b) Escribir un procedimiento `procedure ELIMINA_VALOR(var C:cola; n: integer);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C = {1,3,5,4,2,3,7,3,5}`, después de `ELIMINA_VALOR(C,3)` debe quedar `C = {1,5,4,2,7,5}`. *Sugerencia: Usar una estructura auxiliar lista o cola.* Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

Ej. 4.- [LIBRES] Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $=\{C, D, E, R, S, Q, A, B\}$,
- `ORD_POST` $=\{D, Q, R, A, B, S, E, C\}$.

(b) **Árboles de Huffman:** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CORRALITO`
$P(C) = 0.2, P(O) = 0.2, P(R) = 0.2, P(A) = 0.1, P(L) = 0.1, P(I) = 0.1, P(T) = 0.1$
Calcular la longitud promedio del código obtenido.

Ej. 5.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución para el algoritmo de clasificación rápida (*"quicksort"*) **en el peor caso** es
- $O(\log n)$
- $O(n)$
- $O(n^2)$
- $O(n \log n)$

(b) El tiempo de ejecución para el algoritmo de clasificación por montículos es
- $O(n \log n)$ en el mejor caso, $O(n^2)$ en promedio
- siempre $O(n \log n)$
- siempre $O(n^2)$
- a veces $O(n^2)$

(c) Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es
- $O(n^2/B)$
- $O((n/B)^2)$
- $O(n + B)$
- $O(n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
Examen Final. [3 de Octubre de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD ARBOL BINARIO` listadas a continuación, con celdas enlazadas por punteros **o** cursores: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`.

Ej. 2.- Escribir un procedimiento `procedure KRONECKER(LA,LB: lista; var L:lista);` que, dadas dos listas `LA`=$\{a_1, a_2, ..., a_N\}$ y `LB`=$\{b_1, b_2, ..., b_M\}$ retorna otra lista `L` con $N \times M$ valores de la siguiente forma:

$$\begin{aligned} L = \{&a_1b_1, a_1b_2, ..., a_1b_M, \\ &a_2b_1, a_2b_2, ..., a_2b_M, \\ &... \\ &a_Nb_1, a_Nb_2, ..., a_Nb_M\} \end{aligned} \tag{1}$$

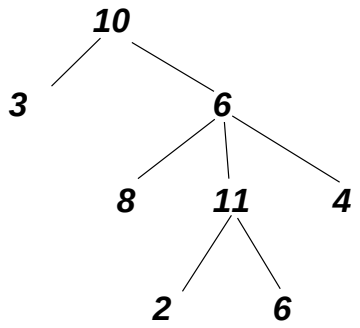
Por ejemplo, si `LA`=$\{1, 3, 5, 0\}$ y `LB`=$\{1, 2, 3, 4\}$ entonces `KRONECKER(LA,LB,L)` debe hacer que

$$L = \{1, 2, 3, 4, 3, 6, 9, 12, 5, 10, 15, 20, 0, 0, 0, 0\}. \tag{2}$$

Usar las siguientes primitivas del `TAD LISTA`: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. **N**o usar ninguna estructura auxiliar.

Ej. 3.- Escribir una función `function MAXCOTA(n:nodo;A:arbol;cota:integer):integer;` que retorna el máximo de las etiquetas de un árbol binario tales que son menores o iguales que la cota `c`. Por ejemplo, si las etiquetas de un árbol `A` son $\{1, 3, 7, 4, 2, 10, 13\}$ y `cota`=8, entonces `MAXCOTA(raiz(A),A,8)` debe retornar 7. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)` y `raiz(A)`.

Ej. 4.- Escribir una función `function MAXHOJA(n:nodo;A:arbol):integer;` que retorna el máximo de las etiquetas *de las hojas* un árbol ordenado orientado. Por ejemplo, en el siguiente árbol debe retornar 8.



Usar las primitivas del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 5.- **[LIBRES] Ejercicios operativos:**

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, Q, T, R, S, U, A, L\}$,
- `ORD_POST` $=\{T, Q, S, U, L, A, R, C\}$.

(b) **Clasificación por montículos:** Dados los enteros $\{15, 7, 3, 8, 4, 10, 2\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

Ej. 6.- **[LIBRES] Preguntas:** [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) Dadas las funciones $T_1(n) = 2n^3 + \log n$, $T_2(n) = 4 + \log n$, $T_3(n) = 2^n + n!$ y $T_4(n) = 4^n + n^2$ decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_1 < T_2 < T_3 < T_4$
- ☐ $T_4 < T_3 < T_2 < T_1$
- ☐ $T_2 < T_1 < T_4 < T_3$
- ☐ $T_3 < T_4 < T_1 < T_2$

(b) ¿Cuál de los siguientes algoritmos de clasificación es el más rápido *en el caso promedio*?

- ☐ Burbuja (*"Bubble-sort"*)
- ☐ Clasificación rápida (*"Quick-sort"*)
- ☐ Clasificación óptima (*"Holy grail sort"*)
- ☐ Selección

(c) Una ventaja del método de clasificación por selección, en comparación con otros algoritmos *lentos*, es que realiza sólo n *intercambios*...

- ☐ ... a veces.
- ☐ ... cuando el vector está ordenado.
- ☐ ... siempre.
- ☐ ... cuando el vector está desordenado.

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

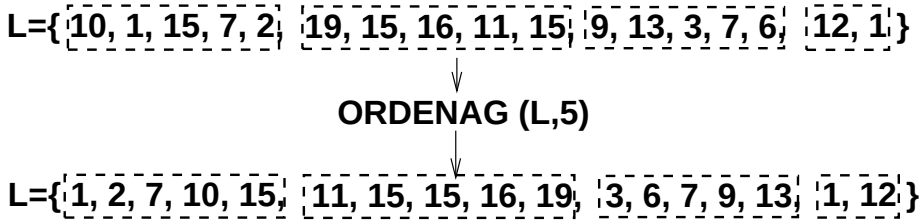
Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [5 de Diciembre de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con celdas enlazadas por punteros **o** cursores: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`.

Ej. 2.- Escribir un procedimiento `procedure ORDENAG(var L: lista; m:integer);` que, dada una lista `L` va ordenando sus elementos de a grupos de m elementos. Por ejemplo si $m = 5$ `ORDENAG` ordena los primeros 5 elementos entre sí, después los siguientes 5 elementos, y así siguiendo. Si la longitud N de la lista no es un múltiplo exacto de m elementos, entonces los últimos N `mod` m elementos *también* deben ser ordenados entre sí. En la figura se muestra el resultado de aplicar `ORDENAG(L,5)` a una lista de 17 elementos.



Usar las siguientes primitivas del `TAD LISTA`: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)` y `FIN(L)`. Sugerencia: Usar una lista auxiliar. Para ordenar los elementos dentro de esta lista auxiliar puede usarse cualquier algoritmo de ordenamiento. Se sugiere usar inserción o similar.

Ej. 3.- Escribir una función `function SUMAPAR(n: nodo; A: arbol) : integer` que retorna la suma de las etiquetas pares de un árbol binario. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 4.- Escribir una función `function MIN(n:nodo; A:arbol):integer;` que retorna el mínimo de las etiquetas de un árbol ordenado orientado. Usar las primitivas del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 5.- **[LIBRES] Ejercicios operativos:**

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Q, R, S, L, W, M, Z, T\}$,
- `ORD_POST` $=\{R, W, L, Z, M, S, T, Q\}$.

(b) **[LIBRES]** Dados los enteros $\{12, 11, 14, 8, 16, 13, 9, 7, 15, 21, 4\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 12, 9 y 13.

Ej. 6.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) Dadas las funciones $T_1(n) = 5n + \log n$, $T_2(n) = 4n^2 + \sqrt{n}$, $T_3(n) = 2^n + n!$ y $T_4(n) = \sqrt{(}n) + \log n$ decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_4 < T_3 < T_2 < T_1$
- ☐ $T_2 < T_1 < T_4 < T_3$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_3 < T_4 < T_1 < T_2$

(b) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log n)$ *en el caso promedio*?

- ☐ Clasificación secuencial (*"Sequential sort"*)
- ☐ Clasificación rápida (*"Quick-sort"*)
- ☐ Selección
- ☐ Burbuja (*"Bubble-sort"*)

(c) La búsqueda *exhaustiva* del camino de longitud mínima para el *"Problema del Agente Viajante"* (PAV) tiene un tiempo de ejecución... (n es el número de ciudades)

- ☐ $O(n!)$
- ☐ $O(1)$
- ☐ $O(n)$
- ☐ $O(n^2)$

(d) Una de las desventajas de implementar el `TAD LISTA` con arreglos es que las inserciones/supresiones en el medio de la lista son...

- ☐ ... $O(1)$.
- ☐ ... $O(n^2)$.
- ☐ ... $O(n)$.
- ☐ ... $O(\log n)$.

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [19 de Diciembre de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias. `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 2.- Escribir una función `function INCLUIDO(nb,na: nodo; B,A: arbol) : boolean`, la cual retorna verdadero si la *estructura* del subárbol del nodo `nb` en el árbol ordenado orientado `B` está incluida dentro de la del subárbol del nodo `na` en el árbol `A`, independientemente de las etiquetas de los nodos correspondientes. Definiéndolo en forma recursiva, el subárbol del nodo `nb` está incluido en el de otro nodo `na` si:

- `nb`$= \Lambda$, **o**
- `nb`$<> \Lambda$, `na`$<> \Lambda$ y los subárboles de los hijos de `nb` están incluidos en los subárboles de los hijos de `na`. (Notar la *recursividad* de la definición).

Utilizar las funciones `PADRE (n,A)`, `HIJO_MAS_IZQUIERDO (n,A)`, `HERMANO_DERECHO (n,A)`, y `ETIQUETA (n,A)`.

Ej. 3.- Ejercicios simples sobre TAD's

(a) Escribir una función `function DMINMAX(L:lista):integer;` que, dada una lista `L` retorna la distancia entre las posiciones del mínimo y del máximo de la lista. La distancia debe ser positiva si el mínimo está antes del máximo y negativa en caso contrario. Por ejemplo, si `L`$=\{5,1,3,2,4,7,6\}$ entonces `DMINMAX(L)` debe retornar 4, ya que las posiciones del mínimo y máximo son 2 y 6, respectivamente. Para `L`$=\{5,9,3,2,4,1,6\}$ debe retornar -4. Si el valor del mínimo aparece repetido se debe tomar la primera posición, mientras que para el máximo se debe tomar la última. Por ejemplo, para `L`$=\{7,5,1,3,2,1,4,7,6\}$ entonces debe retornar 5. Usar las primitivas del `TAD LISTA:RECUPERA(p,L)`, `SIGUIENTE(p,L)`, `PRIMERO(L)`, y `FIN(L)`. No utilizar estructuras auxiliares.

(b) Escribir un procedimiento `procedure JUNTA(var P:pila; m:integer);` que reemplaza los primeros `m` elementos de la pila P por su suma. Si hay menos de `m` elementos en P entonces debe reemplazar todos por su suma. Por ejemplo, si `P`$=\{1,3,2,5,4,3,2\}$, entonces `JUNTA(P,4)` debe dejar `P`$=\{11,4,3,2\}$. Si `P`$=\{1,3,2\}$ entonces `JUNTA(P,4)` debe dejar `P`$=\{6\}$. Usar las primitivas del `TAD PILA`: **Pila:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 4.- [LIBRES] Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $=\{Z, Q, R, T, U, V, W, O\}$.
- `ORD_POST` $=\{Q, T, V, O, W, U, R, Z\}$.

(b) **[LIBRES] Clasificación por montículos.** Dados los enteros $\{12, 11, 14, 8, 16, 3, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

Ej. 5.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) Dadas las funciones $T_1(n) = 0.5n + \sqrt{n}$, $T_2(n) = 0.2n^2 + 2.1 \log n$, $T_3(n) = 3n! + 5.4n^3$ y $T_4(n) = n^{1/2}$ decir cuál de los siguientes ordenamientos es el correcto

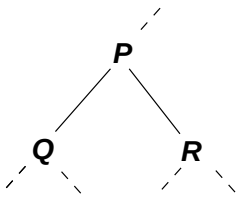
- ☐ $T_4 < T_3 < T_2 < T_1$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_2 < T_1 < T_4 < T_3$
- ☐ $T_3 < T_4 < T_1 < T_2$

(b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"heapsort"*) es $O(n \log(n)$ (n es el número de elementos a ordenar) ...

- ☐ ... siempre.
- ☐ ... a veces.
- ☐ ... nunca.
- ☐ ... en el mejor caso.

(c) El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si P, Q, R son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ $Q + R \leq \infty$.
- ☐ $Q \leq P \leq R$.
- ☐ $P \leq 1/2(Q + R)$.
- ☐ $P \leq Q, R$.



(d) ¿Cuál es el tiempo de ejecución del procedimiento de clasificación por *incrementos decrecientes* (*shell-sort*) en el caso promedio?

- ☐ $O(n^{1.3})$
- ☐ $O(n^{1.5})$
- ☐ $O(\log n)$
- ☐ $O(n!)$

Algoritmos y Estructuras de Datos.
Parcial 1. Tema 1a. [16 de abril de 2002]

Ej. 1.- Ordenar por velocidad de crecimiento **de menor a mayor** las siguientes funciones

(a) $T_1(n) = 4.3n^2 + 2n + 0.3\log(n)$, $T_2(n) = 3.5 \times 2^n + 5n^3 + 2\sqrt{n}$, $T_3(n) = 0.3n$, $T_4(n) = 23$.

(b) $T_5(n) = 2n$, $T_6(n) = \sqrt{n}$, $T_7(n) = 0.3\log n + 5$

(c) $T_8(n) = n^3 + \sqrt{n} + 2^n$, $T_9(n) = 0.3n + 5n^3$, $T_{10}(n) = 23 + 2n^2$, $T_{11}(n) = n!$

Ej. 2.- Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por punteros. Es decir, implementar en Pascal los siguientes procedimientos/funciones:

(a) `INSERTA(x,p,L)`,

(b) `LOCALIZA(x,L)`,

(c) `RECUPERA(p,L)`,

(d) `SUPRIME(p,L)`,

(e) `SIGUIENTE(p,L)`,

(f) `ANULA(L)`,

(g) `PRIMERO(L)`, y

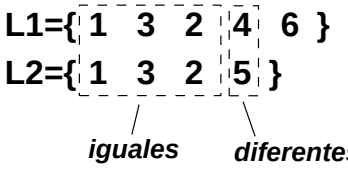
(h) `FIN(L)`.

[*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

Ej. 3.- Dadas dos listas de enteros `L1` y `L2` escribir una función

```
function LEXICORD(L1,L2:lista) : boolean;
```

que retorna `true` si la lista `L1` es mayor que la `L2` en sentido *"lexicográfico"*, y `false` en caso contrario. El orden lexicográfico es el orden alfabético usado para ordenar las palabras en el diccionario, si consideramos que la lista de enteros es una *"palabra"* cuyas *"letras"* son cada uno de los enteros de la lista. Luego, se van comparando los números en las posiciones correspondientes hasta encontrar uno diferente. La lista que tiene el número (*"letra"*) mayor es la lista *"mayor"* en el sentido lexicográfico. Asi, por ejemplo, si `L1=(1 3 2 4 6)`, `L2=(1 3 2 5)`, entonces `L2 > L1` ya que los primeros elementos diferentes son 5 y 4 en la cuarta posición, de los cuales es mayor el 5 que está en `L2` (ver figura).



Por lo tanto `LEXICORD(L1,L2)` debe retornar `false`. Además si una lista esta totalmente contenida en el principio de la otra, es *"mayor"* la lista más larga. Por ejemplo, si `L1=(1 3 3 4 5) L2=(1 3 3)`, entonces `L1 > L2` y `LEXICORD(L1,L2)` retorna `true`. Dos listas son iguales, si y sólo si tienen la misma cantidad de elementos, y los elementos en posiciones correspondientes son iguales. Por ejemplo, si `L1=(1 3 3 4)` y `L2=(1 3 3 4)`, entonces `L1 = L2` y `LEXICORD(L1,L2)` retorna `false`.

El orden lexicográfico entre dos listas `L1` y `L2` puede definirse, en forma más precisa, de la siguiente manera,

(a) Si las dos listas son vacías, entonces `L1 = L2` (valor de retorno `false`)

(b) Si una de las listas es vacía (digamos `L2`) y la otra (`L1`) no lo es, entonces `L1 > L2` (retorna `true`). Si la que esta vacía es `L1` entonces `L1 < L2` y retorna `false`.

(c) Si las dos listas no son vacías, y los primeros elementos son diferentes (digamos a_1 en `L1` y b_1 en `L2`), entonces el valor de retorno es el valor lógico de $a_1 > b_1$.

(d) Finalmente, si los dos primeros elementos son iguales, entonces el valor de retorno es el que corresponde a las listas que se obtienen eliminando los primeros elementos.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 4.- Escribir los siguientes procedimientos/funciones

(a) Escribir un procedimiento `ROTACION(var C: cola)` que saca una cierta cantidad de enteros del frente de la cola `C` y los vuelve a insertar en fin de cola, de tal manera que quede en el frente de cola un número par. Por ejemplo, si `C`=$\{1, 3, 5, 2, 4\}$ entonces, despues de `ROTACION(C)` debe quedar `C`=$\{2, 4, 1, 3, 5\}$
Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) Escribir un procedimiento `SACA_FONDO(var P: pila);` que elimina el último elemento de una pila `P` dejando los demás inalterados, usando exclusivamente una pila auxiliar.
Utilizar las primitivas del **TAD PILA:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Algoritmos y Estructuras de Datos.
Parcial 1. Tema 1b. [16 de abril de 2002]

Ej. 1.- Ordenar por velocidad de crecimiento **de menor a mayor** las siguientes funciones

(a) $T_1(n) = 2n + n^2 + 3.3\log(n)$, $T_2(n) = 1.2 \times 3^n + 2n^4 + 3.2\sqrt{n}$, $T_3(n) = 0.1n$, $T_4(n) = 35$

(b) $T_5(n) = 2n^2$, $T_6(n) = \sqrt{n}$, $T_7(n) = 1.4\log n + 2$

(c) $T_8(n) = n^2 + \sqrt{n} + 3^n$, $T_9(n) = 1.2n + 2n^3$, $T_{10}(n) = 2.3 + 0.1n^2$, $T_{11}(n) = 2n!$

Ej. 2.- Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por punteros. Es decir, implementar en Pascal los siguientes procedimientos/funciones:

(a) `INSERTA(x,p,L)`,

(b) `LOCALIZA(x,L)`,

(c) `RECUPERA(p,L)`,

(d) `SUPRIME(p,L)`,

(e) `SIGUIENTE(p,L)`,

(f) `ANULA(L)`,

(g) `PRIMERO(L)`, y

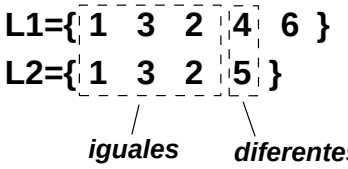
(h) `FIN(L)`.

[*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

Ej. 3.- Dadas dos listas de enteros `L1` y `L2` escribir una función

```
function LEXICORD(L1,L2:lista) : boolean;
```

que retorna `true` si la lista `L1` es mayor que la `L2` en sentido *"lexicográfico"*, y `false` en caso contrario. El orden lexicográfico es el orden alfabético usado para ordenar las palabras en el diccionario, si consideramos que la lista de enteros es una *"palabra"* cuyas *"letras"* son cada uno de los enteros de la lista. Luego, se van comparando los números en las posiciones correspondientes hasta encontrar uno diferente. La lista que tiene el número (*"letra"*) mayor es la lista *"mayor"* en el sentido lexicográfico. Asi, por ejemplo, si `L1=(1 3 2 4 6)`, `L2=(1 3 2 5)`, entonces `L2 > L1` ya que los primeros elementos diferentes son 5 y 4 en la cuarta posición, de los cuales es mayor el 5 que está en `L2` (ver figura).



Por lo tanto `LEXICORD(L1,L2)` debe retornar `false`. Además si una lista esta totalmente contenida en el principio de la otra, es *"mayor"* la lista más larga. Por ejemplo, si `L1=(1 3 3 4 5) L2=(1 3 3)`, entonces `L1 > L2` y `LEXICORD(L1,L2)` retorna `true`. Dos listas son iguales, si y sólo si tienen la misma cantidad de elementos, y los elementos en posiciones correspondientes son iguales. Por ejemplo, si `L1=(1 3 3 4)` y `L2=(1 3 3 4)`, entonces `L1 = L2` y `LEXICORD(L1,L2)` retorna `false`.

El orden lexicográfico entre dos listas `L1` y `L2` puede definirse, en forma más precisa, de la siguiente manera,

(a) Si las dos listas son vacías, entonces `L1 = L2` (valor de retorno `false`)

(b) Si una de las listas es vacía (digamos `L2`) y la otra (`L1`) no lo es, entonces `L1 > L2` (retorna `true`). Si la que esta vacía es `L1` entonces `L1 < L2` y retorna `false`.

(c) Si las dos listas no son vacías, y los primeros elementos son diferentes (digamos a_1 en `L1` y b_1 en `L2`), entonces el valor de retorno es el valor lógico de $a_1 > b_1$.

(d) Finalmente, si los dos primeros elementos son iguales, entonces el valor de retorno es el que corresponde a las listas que se obtienen eliminando los primeros elementos.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 4.- Escribir los siguientes procedimientos/funciones

(a) Escribir un procedimiento `ROTACION(var C: cola)` que saca una cierta cantidad de enteros del frente de la cola `C` y los vuelve a insertar en fin de cola, de tal manera que quede en el frente de cola un número par. Por ejemplo, si `C`=$\{1, 3, 5, 2, 4\}$ entonces, despues de `ROTACION(C)` debe quedar `C`=$\{2, 4, 1, 3, 5\}$
Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) Escribir un procedimiento `SACA_FONDO(var P: pila);` que elimina el último elemento de una pila `P` dejando los demás inalterados, usando exclusivamente una pila auxiliar.
Utilizar las primitivas del **TAD PILA:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Algoritmos y Estructuras de Datos.
Parcial 1. Tema 1c. [16 de abril de 2002]

Ej. 1.- Ordenar por velocidad de crecimiento **de menor a mayor** las siguientes funciones

(a) $T_1(n) = 0.2n + 2n^2 + \log(n)$, $T_2(n) = 0.2 \times 4^n + 2n^4 + 0.3\sqrt{n}$, $T_3(n) = 0.5n$, $T_4(n) = 12$

(b) $T_5(n) = 4n^2$, $T_6(n) = \sqrt{n}$, $T_7(n) = 1.1 \log n + 5$

(c) $T_8(n) = 3n^3 + 2.4\sqrt{n} + 4^n$, $T_9(n) = 6.2n + 5n^3$, $T_{10}(n) = 4.1 + 0.3n^2$, $T_{11}(n) = 4n!$

Ej. 2.- Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por punteros. Es decir, implementar en Pascal los siguientes procedimientos/funciones:

(a) `INSERTA(x,p,L)`,

(b) `LOCALIZA(x,L)`,

(c) `RECUPERA(p,L)`,

(d) `SUPRIME(p,L)`,

(e) `SIGUIENTE(p,L)`,

(f) `ANULA(L)`,

(g) `PRIMERO(L)`, y

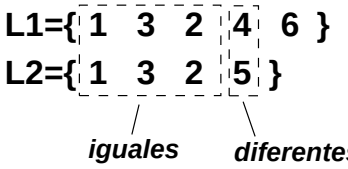
(h) `FIN(L)`.

[*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

Ej. 3.- Dadas dos listas de enteros `L1` y `L2` escribir una función

```
function LEXICORD(L1,L2:lista) : boolean;
```

que retorna `true` si la lista `L1` es mayor que la `L2` en sentido *"lexicográfico"*, y `false` en caso contrario. El orden lexicográfico es el orden alfabético usado para ordenar las palabras en el diccionario, si consideramos que la lista de enteros es una *"palabra"* cuyas *"letras"* son cada uno de los enteros de la lista. Luego, se van comparando los números en las posiciones correspondientes hasta encontrar uno diferente. La lista que tiene el número (*"letra"*) mayor es la lista *"mayor"* en el sentido lexicográfico. Asi, por ejemplo, si `L1=(1 3 2 4 6)`, `L2=(1 3 2 5)`, entonces `L2 > L1` ya que los primeros elementos diferentes son 5 y 4 en la cuarta posición, de los cuales es mayor el 5 que está en `L2` (ver figura).



Por lo tanto `LEXICORD(L1,L2)` debe retornar `false`. Además si una lista esta totalmente contenida en el principio de la otra, es *"mayor"* la lista más larga. Por ejemplo, si `L1=(1 3 3 4 5) L2=(1 3 3)`, entonces `L1 > L2` y `LEXICORD(L1,L2)` retorna `true`. Dos listas son iguales, si y sólo si tienen la misma cantidad de elementos, y los elementos en posiciones correspondientes son iguales. Por ejemplo, si `L1=(1 3 3 4)` y `L2=(1 3 3 4)`, entonces `L1 = L2` y `LEXICORD(L1,L2)` retorna `false`.

El orden lexicográfico entre dos listas `L1` y `L2` puede definirse, en forma más precisa, de la siguiente manera,

(a) Si las dos listas son vacías, entonces `L1 = L2` (valor de retorno `false`)

(b) Si una de las listas es vacía (digamos `L2`) y la otra (`L1`) no lo es, entonces `L1 > L2` (retorna `true`). Si la que esta vacía es `L1` entonces `L1 < L2` y retorna `false`.

(c) Si las dos listas no son vacías, y los primeros elementos son diferentes (digamos a_1 en `L1` y b_1 en `L2`), entonces el valor de retorno es el valor lógico de $a_1 > b_1$.

(d) Finalmente, si los dos primeros elementos son iguales, entonces el valor de retorno es el que corresponde a las listas que se obtienen eliminando los primeros elementos.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 4.- Escribir los siguientes procedimientos/funciones

(a) Escribir un procedimiento `ROTACION(var C: cola)` que saca una cierta cantidad de enteros del frente de la cola `C` y los vuelve a insertar en fin de cola, de tal manera que quede en el frente de cola un número par. Por ejemplo, si `C`=$\{1, 3, 5, 2, 4\}$ entonces, despues de `ROTACION(C)` debe quedar `C`=$\{2, 4, 1, 3, 5\}$
Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) Escribir un procedimiento `SACA_FONDO(var P: pila);` que elimina el último elemento de una pila `P` dejando los demás inalterados, usando exclusivamente una pila auxiliar.
Utilizar las primitivas del **TAD PILA:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Algoritmos y Estructuras de Datos.
Parcial 1. Tema 2a. [16 de abril de 2002]

Ej. 1.- Ordenar por velocidad de crecimiento **de menor a mayor** las siguientes funciones

(a) $T_1(n) = 4.3n^2 + 2n + 0.3\log(n)$, $T_2(n) = 3.5 \times 2^n + 5n^3 + 2\sqrt{n}$, $T_3(n) = 0.3n$, $T_4(n) = 23$.

(b) $T_5(n) = 2n$, $T_6(n) = \sqrt{n}$, $T_7(n) = 0.3\log n + 5$

(c) $T_8(n) = n^3 + \sqrt{n} + 2^n$, $T_9(n) = 0.3n + 5n^3$, $T_{10}(n) = 23 + 2n^2$, $T_{11}(n) = n!$

Ej. 2.- Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por punteros. Es decir, implementar en Pascal los siguientes procedimientos/funciones:

(a) `INSERTA(x,p,L)`,

(b) `LOCALIZA(x,L)`,

(c) `RECUPERA(p,L)`,

(d) `SUPRIME(p,L)`,

(e) `SIGUIENTE(p,L)`,

(f) `ANULA(L)`,

(g) `PRIMERO(L)`, y

(h) `FIN(L)`.

[*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

Ej. 3.- Dada una lista con enteros `L` y dos listas `SEQ` y `REEMP` escribir un procedimiento `REEMPLAZA(var L : lista; SEQ,REEMP: lista )` que busca todas las secuencias de `SEQ` en `L` y las reemplaza por `REEMP`. Por ejemplo, si `L`=$\{1,2,3,4,5,1,2,3,4,5,1,2,3,4,5\}$, `SEQ`=$\{4,5,1)$ y `REEMP`=$\{9,7,3\}$, entonces después de llamar a REEMPLAZA debe quedar `L`=$\{1,2,3,9,7,3,2,3,9,7,3,2,3,4,5\}$. Este procedimiento tiene un efecto equivalente a la función `REEMPLAZAR` de los editores de texto.

Se sugiere el siguiente algoritmo. Se recorre la lista `L` con una posición `p`. Si la secuencia de `L` a partir de `p` es igual a `SEQ` entonces se eliminan los elementos de `L` correspondientes y se inserta `REEMP`. Si no, se avanza `p`. Para esto realizar las siguientes tareas,

(a) Escribir una `function LONGITUD(L:lista):integer` que cuenta los elementos de una lista.

(b) Escribir una `function COMPARA(SEQ, L: lista; p: posicion) : boolean;` que compara la secuencia de elementos de `L` a partir de la posición `p` con la lista `SEQ`. Devuelve verdadero si `SEQ` esta contenido en `L` y en caso contrario falso. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5\}$ y `SEQ`=$\{3, 4\}$, entonces `COMPARA(SEQ,L,p)` retorna `true` si p es la tercera posición y `false` en cualquier otro caso.

(c) Escribir un procedimiento `procedure SUBST(var L : lista; p:posicion; n:integer; REEMP:lista);` que elimina n elementos de `L` a partir de la posición `p` e inserta la lista `SEQ` en esa posición. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5\}$, `REEMP`=$\{7, 8\}$, y `p` es la segunda posición, entonces después de la `SUBST(L,p,3,REEMP)`, la lista `L` queda como `L`=$\{1, 7, 8, 5\}$

(d) Escribir el procedimiento `procedure REEMPLAZA(var l: lista; SEQ, REEMP: lista );` utilizando las funciones/procedimientos `LONGITUD, COMPARA y SUBST`.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L), RECUPERA(p,L), SUPRIME(p,L), SIGUIENTE(p,L), ANULA(L), PRIMERO(L),` y `FIN(L).`

Ej. 4.- Escribir los siguientes procedimientos/funciones

(a) Escribir un procedimiento `ROTACION(var C: cola)` que saca una cierta cantidad de enteros del frente de la cola `C` y los vuelve a insertar en fin de cola, de tal manera que quede en el frente de cola un número par. Por ejemplo, si `C`=$\{1, 3, 5, 2, 4\}$ entonces, despues de `ROTACION(C)` debe quedar `C`=$\{2, 4, 1, 3, 5\}$
Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) Escribir un procedimiento `SACA_FONDO(var P: pila);` que elimina el último elemento de una pila `P` dejando los demás inalterados, usando exclusivamente una pila auxiliar.
Utilizar las primitivas del **TAD PILA:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Algoritmos y Estructuras de Datos.
Parcial 1. Tema 2b. [16 de abril de 2002]

Ej. 1.- Ordenar por velocidad de crecimiento **de menor a mayor** las siguientes funciones

(a) $T_1(n) = 2n + n^2 + 3.3\log(n)$, $T_2(n) = 1.2 \times 3^n + 2n^4 + 3.2\sqrt{n}$, $T_3(n) = 0.1n$, $T_4(n) = 35$

(b) $T_5(n) = 2n^2$, $T_6(n) = \sqrt{n}$, $T_7(n) = 1.4\log n + 2$

(c) $T_8(n) = n^2 + \sqrt{n} + 3^n$, $T_9(n) = 1.2n + 2n^3$, $T_{10}(n) = 2.3 + 0.1n^2$, $T_{11}(n) = 2n!$

Ej. 2.- Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por punteros. Es decir, implementar en Pascal los siguientes procedimientos/funciones:

(a) `INSERTA(x,p,L)`,

(b) `LOCALIZA(x,L)`,

(c) `RECUPERA(p,L)`,

(d) `SUPRIME(p,L)`,

(e) `SIGUIENTE(p,L)`,

(f) `ANULA(L)`,

(g) `PRIMERO(L)`, y

(h) `FIN(L)`.

[*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

Ej. 3.- Dada una lista con enteros `L` y dos listas `SEQ` y `REEMP` escribir un procedimiento `REEMPLAZA(var L : lista; SEQ,REEMP: lista )` que busca todas las secuencias de `SEQ` en `L` y las reemplaza por `REEMP`. Por ejemplo, si `L`=$\{1,2,3,4,5,1,2,3,4,5,1,2,3,4,5\}$, `SEQ`=$\{4,5,1)$ y `REEMP`=$\{9,7,3\}$, entonces después de llamar a REEMPLAZA debe quedar `L`=$\{1,2,3,9,7,3,2,3,9,7,3,2,3,4,5\}$. Este procedimiento tiene un efecto equivalente a la función `REEMPLAZAR` de los editores de texto.

Se sugiere el siguiente algoritmo. Se recorre la lista `L` con una posición `p`. Si la secuencia de `L` a partir de `p` es igual a `SEQ` entonces se eliminan los elementos de `L` correspondientes y se inserta `REEMP`. Si no, se avanza `p`. Para esto realizar las siguientes tareas,

(a) Escribir una `function LONGITUD(L:lista):integer` que cuenta los elementos de una lista.

(b) Escribir una `function COMPARA(SEQ, L: lista; p: posicion) : boolean;` que compara la secuencia de elementos de `L` a partir de la posición `p` con la lista `SEQ`. Devuelve verdadero si `SEQ` esta contenido en `L` y en caso contrario falso. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5\}$ y `SEQ`=$\{3, 4\}$, entonces `COMPARA(SEQ,L,p)` retorna `true` si p es la tercera posición y `false` en cualquier otro caso.

(c) Escribir un procedimiento `procedure SUBST(var L : lista; p:posicion; n:integer; REEMP:lista);` que elimina n elementos de `L` a partir de la posición `p` e inserta la lista `SEQ` en esa posición. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5\}$, `REEMP`=$\{7, 8\}$, y `p` es la segunda posición, entonces después de la `SUBST(L,p,3,REEMP)`, la lista `L` queda como `L`=$\{1, 7, 8, 5\}$

(d) Escribir el procedimiento `procedure REEMPLAZA(var l: lista; SEQ, REEMP: lista );` utilizando las funciones/procedimientos `LONGITUD, COMPARA y SUBST`.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L), RECUPERA(p,L), SUPRIME(p,L), SIGUIENTE(p,L), ANULA(L), PRIMERO(L),` y `FIN(L).`

Ej. 4.- Escribir los siguientes procedimientos/funciones

(a) Escribir un procedimiento `ROTACION(var C: cola)` que saca una cierta cantidad de enteros del frente de la cola `C` y los vuelve a insertar en fin de cola, de tal manera que quede en el frente de cola un número par. Por ejemplo, si `C`=$\{1, 3, 5, 2, 4\}$ entonces, despues de `ROTACION(C)` debe quedar `C`=$\{2, 4, 1, 3, 5\}$
Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) Escribir un procedimiento `SACA_FONDO(var P: pila);` que elimina el último elemento de una pila `P` dejando los demás inalterados, usando exclusivamente una pila auxiliar.
Utilizar las primitivas del **TAD PILA:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Algoritmos y Estructuras de Datos.
Parcial 1. Tema 2c. [16 de abril de 2002]

Ej. 1.- Ordenar por velocidad de crecimiento **de menor a mayor** las siguientes funciones

(a) $T_1(n) = 0.2n + 2n^2 + \log(n)$, $T_2(n) = 0.2 \times 4^n + 2n^4 + 0.3\sqrt{n}$, $T_3(n) = 0.5n$, $T_4(n) = 12$

(b) $T_5(n) = 4n^2$, $T_6(n) = \sqrt{n}$, $T_7(n) = 1.1 \log n + 5$

(c) $T_8(n) = 3n^3 + 2.4\sqrt{n} + 4^n$, $T_9(n) = 6.2n + 5n^3$, $T_{10}(n) = 4.1 + 0.3n^2$, $T_{11}(n) = 4n!$

Ej. 2.- Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por punteros. Es decir, implementar en Pascal los siguientes procedimientos/funciones:

(a) `INSERTA(x,p,L)`,

(b) `LOCALIZA(x,L)`,

(c) `RECUPERA(p,L)`,

(d) `SUPRIME(p,L)`,

(e) `SIGUIENTE(p,L)`,

(f) `ANULA(L)`,

(g) `PRIMERO(L)`, y

(h) `FIN(L)`.

[*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

Ej. 3.- Dada una lista con enteros `L` y dos listas `SEQ` y `REEMP` escribir un procedimiento `REEMPLAZA(var L : lista; SEQ,REEMP: lista )` que busca todas las secuencias de `SEQ` en `L` y las reemplaza por `REEMP`. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5\}$, `SEQ`=$\{4, 5, 1)$ y `REEMP`=$\{9, 7, 3\}$, entonces después de llamar a REEMPLAZA debe quedar `L`=$\{1, 2, 3, 9, 7, 3, 2, 3, 9, 7, 3, 2, 3, 4, 5\}$. Este procedimiento tiene un efecto equivalente a la función `REEMPLAZAR` de los editores de texto.

Se sugiere el siguiente algoritmo. Se recorre la lista `L` con una posición `p`. Si la secuencia de `L` a partir de `p` es igual a `SEQ` entonces se eliminan los elementos de `L` correspondientes y se inserta `REEMP`. Si no, se avanza `p`. Para esto realizar las siguientes tareas,

(a) Escribir una `function LONGITUD(L:lista):integer` que cuenta los elementos de una lista.

(b) Escribir una `function COMPARA(SEQ, L: lista; p: posicion) : boolean;` que compara la secuencia de elementos de `L` a partir de la posición `p` con la lista `SEQ`. Devuelve verdadero si `SEQ` esta contenido en `L` y en caso contrario falso. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5\}$ y `SEQ`=$\{3, 4\}$, entonces `COMPARA(SEQ,L,p)` retorna `true` si p es la tercera posición y `false` en cualquier otro caso.

(c) Escribir un procedimiento `procedure SUBST(var L : lista; p:posicion; n:integer; REEMP:lista);` que elimina n elementos de `L` a partir de la posición `p` e inserta la lista `SEQ` en esa posición. Por ejemplo, si `L`=$\{1, 2, 3, 4, 5\}$, `REEMP`=$\{7, 8\}$, y `p` es la segunda posición, entonces después de la `SUBST(L,p,3,REEMP)`, la lista `L` queda como `L`=$\{1, 7, 8, 5\}$

(d) Escribir el procedimiento `procedure REEMPLAZA(var l: lista; SEQ, REEMP: lista );` utilizando las funciones/procedimientos `LONGITUD, COMPARA y SUBST`.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L), RECUPERA(p,L), SUPRIME(p,L), SIGUIENTE(p,L), ANULA(L), PRIMERO(L),` y `FIN(L).`

Ej. 4.- Escribir los siguientes procedimientos/funciones

(a) Escribir un procedimiento `ROTACION(var C: cola)` que saca una cierta cantidad de enteros del frente de la cola `C` y los vuelve a insertar en fin de cola, de tal manera que quede en el frente de cola un número par. Por ejemplo, si `C`=$\{1, 3, 5, 2, 4\}$ entonces, despues de `ROTACION(C)` debe quedar `C`=$\{2, 4, 1, 3, 5\}$
Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) Escribir un procedimiento `SACA_FONDO(var P: pila);` que elimina el último elemento de una pila `P` dejando los demás inalterados, usando exclusivamente una pila auxiliar.
Utilizar las primitivas del **TAD PILA:** `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Algoritmos y Estructuras de Datos.
Parcial 2. Tema 1a. [28 de Mayo de 2002]

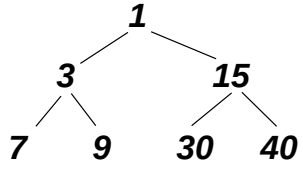
Ej. 1.- Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{U, T, Q, S, R, D, M, E\}$,
- `ORD_POST` $=\{T, R, D, S, M, E, Q, U\}$.

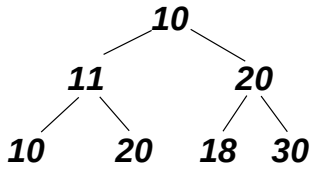
Ej. 2.- Escribir las funciones primitivas del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

(a) `PADRE(n,A)`
(b) `HIJO_MAS_IZQ(n,A)`
(c) `HERMANO_DER(n,A)`
(d) `ETIQUETA(n,A)`
(e) `CREA2(v, A1, A2)`
(f) `ANULA(A)`

Ej. 3.- Se dice que un árbol binario es *"parcialmente ordenado"* (PO) si la etiqueta de cada nodo es menor o igual que las etiquetas de sus hijos. Así por ejemplo de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



SI es PO.



NO es PO.

Escribir una función `VERIFICA_PO(n:nodo; A: Arbol) : boolean` que retorna `true` si el subárbol que cuelga del nodo `n` es parcialmente ordenado y `false` en caso contrario. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 4.- **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CRISIS`
$P(c) = 0.3, P(r) = 0.2, P(s) = 0.2, P(i) = 0.1,$
$P(z) = 0.05, P(u) = 0.05, P(v) = 0.05, P(t) = 0.05.$
Calcular la longitud promedio del código obtenido.

Ej. 5.- Insertar los números $28, 11, 38, 48, 23, 58, 68, 32$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Algoritmos y Estructuras de Datos.
Parcial 2. Tema 1b. [28 de Mayo de 2002]

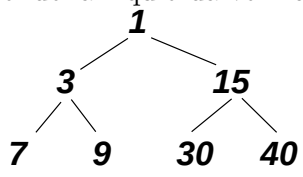
Ej. 1.- Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Y, A, W, Z, R, Q, S, B\}$,
- `ORD_POST` $=\{W, A, R, Q, S, Z, B, Y\}$.

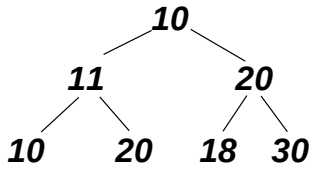
Ej. 2.- Escribir las funciones primitivas del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

(a) `PADRE(n,A)`
(b) `HIJO_MAS_IZQ(n,A)`
(c) `HERMANO_DER(n,A)`
(d) `ETIQUETA(n,A)`
(e) `CREA2(v, A1, A2)`
(f) `ANULA(A)`

Ej. 3.- Se dice que un árbol binario es *"parcialmente ordenado"* (PO) si la etiqueta de cada nodo es menor o igual que las etiquetas de sus hijos. Así por ejemplo de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



SI es PO.



NO es PO.

Escribir una función `VERIFICA_PO(n:nodo; A: Arbol) : boolean` que retorna `true` si el subárbol que cuelga del nodo `n` es parcialmente ordenado y `false` en caso contrario. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 4.- **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CRISIS`
$P(c) = 0.1, P(r) = 0.3, P(s) = 0.2, P(i) = 0.2,$
$P(z) = 0.05, P(u) = 0.05, P(v) = 0.05, P(t) = 0.05.$
Calcular la longitud promedio del código obtenido.

Ej. 5.- Insertar los números $29, 12, 39, 49, 24, 59, 69, 33$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Algoritmos y Estructuras de Datos.
Parcial 2. Tema 1c. [28 de Mayo de 2002]

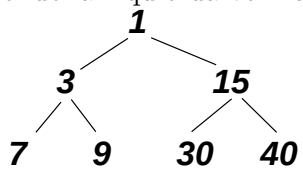
Ej. 1.- Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{W, B, R, S, Q, Z, A, B\}$,
- `ORD_POST` $=\{R, S, B, Q, A, B, Z, W\}$.

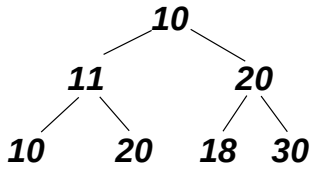
Ej. 2.- Escribir las funciones primitivas del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

(a) `PADRE(n,A)`
(b) `HIJO_MAS_IZQ(n,A)`
(c) `HERMANO_DER(n,A)`
(d) `ETIQUETA(n,A)`
(e) `CREA2(v, A1, A2)`
(f) `ANULA(A)`

Ej. 3.- Se dice que un árbol binario es *"parcialmente ordenado"* (PO) si la etiqueta de cada nodo es menor o igual que las etiquetas de sus hijos. Así por ejemplo de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



SI es PO.



NO es PO.

Escribir una función `VERIFICA_PO(n:nodo; A: Arbol) : boolean` que retorna `true` si el subárbol que cuelga del nodo `n` es parcialmente ordenado y `false` en caso contrario. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 4.- **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CRISIS`
$P(c) = 0.2, P(r) = 0.2, P(s) = 0.1, P(i) = 0.3,$
$P(z) = 0.05, P(u) = 0.05, P(v) = 0.05, P(t) = 0.05.$
Calcular la longitud promedio del código obtenido.

Ej. 5.- Insertar los números $27, 10, 37, 47, 22, 57, 67, 31$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Algoritmos y Estructuras de Datos.
Parcial 2. Tema 2a. [28 de Mayo de 2002]

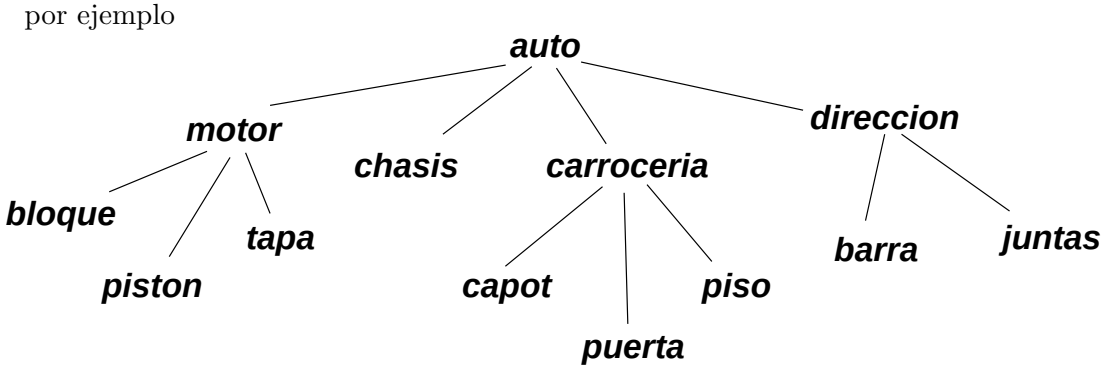
Ej. 1.- Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{U, T, Q, S, R, D, M, E\}$,
- `ORD_POST` $=\{T, R, D, S, M, E, Q, U\}$.

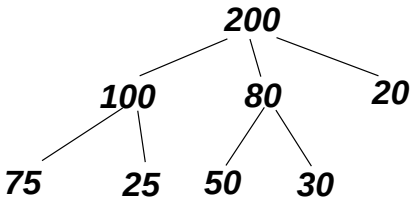
Ej. 2.- Escribir las funciones primitivas del `TAD ARBOL BINARIO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

(a) `PADRE(n,A)`
(b) `HIJO_IZQ(n,A)`
(c) `HIJO_DER(n,A)`
(d) `ETIQUETA(n,A)`
(e) `CREA2(v, A1, A2)`
(f) `ANULA(A)`

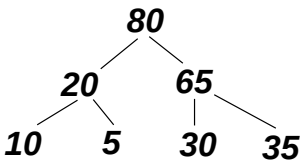
Ej. 3.- En un programa de diseño asistido por computadora, (tipo AutoCAD) se desea mantener las piezas de un sistema (por ejemplo un auto) clasificando sus partes en forma de árbol, por ejemplo



Se quiere mantener en cada hoja el peso (en Kilogramos) del componente, y en los nodos interiores el peso total de todos los componentes del subárbol que cuelga de ese nodo. Periódicamente se quiere verificar que efectivamente las etiquetas del árbol verifican esta propiedad, es decir *"que la etiqueta de los nodos interiores es la suma de las hojas del subárbol que cuelga de él. "* Así, por ejemplo, de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



Verifica



No verifica

Escribir una función `VERIFICA_SUMA(n:nodo; A: Arbol) : boolean;` que retorna `true` si el subárbol que cuelga del nodo verifica la condición y `false` en caso contrario. Usar las primitivas del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`

Ej. 4.- **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CRISIS`
$P(c) = 0.3, P(r) = 0.2, P(s) = 0.2, P(i) = 0.1,$
$P(z) = 0.05, P(u) = 0.05, P(v) = 0.05, P(t) = 0.05.$
Calcular la longitud promedio del código obtenido.

Ej. 5.- Insertar los números $28, 11, 38, 48, 23, 58, 68, 32$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Algoritmos y Estructuras de Datos.
Parcial 2. Tema 2b. [28 de Mayo de 2002]

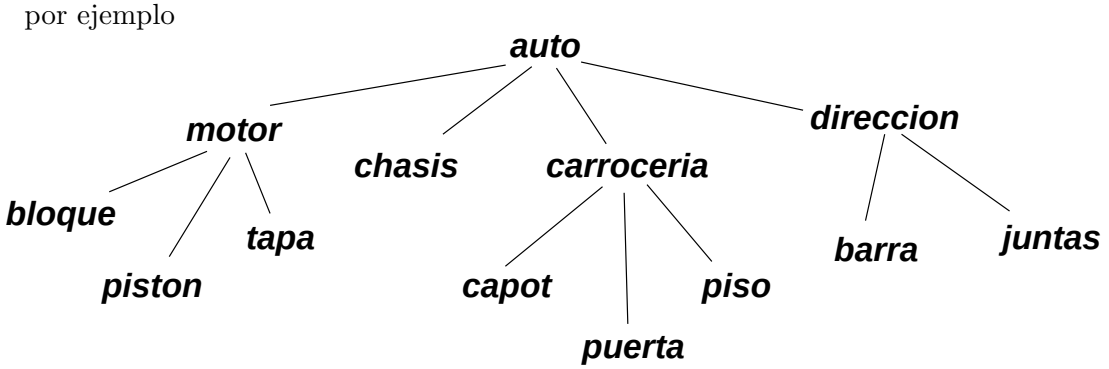
Ej. 1.- Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Y, A, W, Z, R, Q, S, B\}$,
- `ORD_POST` $=\{W, A, R, Q, S, Z, B, Y\}$.

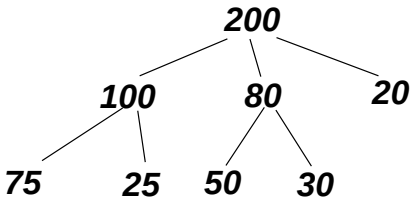
Ej. 2.- Escribir las funciones primitivas del `TAD ARBOL BINARIO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

(a) `PADRE(n,A)`
(b) `HIJO_IZQ(n,A)`
(c) `HIJO_DER(n,A)`
(d) `ETIQUETA(n,A)`
(e) `CREA2(v, A1, A2)`
(f) `ANULA(A)`

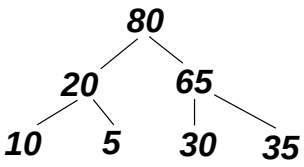
Ej. 3.- En un programa de diseño asistido por computadora, (tipo AutoCAD) se desea mantener las piezas de un sistema (por ejemplo un auto) clasificando sus partes en forma de árbol, por ejemplo



Se quiere mantener en cada hoja el peso (en Kilogramos) del componente, y en los nodos interiores el peso total de todos los componentes del subárbol que cuelga de ese nodo. Periódicamente se quiere verificar que efectivamente las etiquetas del árbol verifican esta propiedad, es decir *"que la etiqueta de los nodos interiores es la suma de las hojas del subárbol que cuelga de él. "* Así, por ejemplo, de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



Verifica



No verifica

Escribir una función `VERIFICA_SUMA(n:nodo; A: Arbol) : boolean;` que retorna `true` si el subárbol que cuelga del nodo verifica la condición y `false` en caso contrario. Usar las primitivas del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`

Ej. 4.- **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CRISIS`
$P(c) = 0.1, P(r) = 0.3, P(s) = 0.2, P(i) = 0.2,$
$P(z) = 0.05, P(u) = 0.05, P(v) = 0.05, P(t) = 0.05.$
Calcular la longitud promedio del código obtenido.

Ej. 5.- Insertar los números $29, 12, 39, 49, 24, 59, 69, 33$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Algoritmos y Estructuras de Datos.
Parcial 2. Tema 2c. [28 de Mayo de 2002]

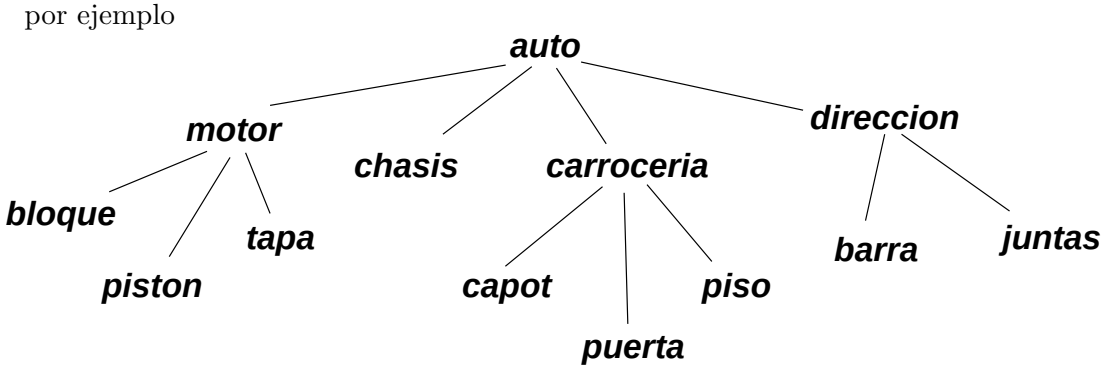
Ej. 1.- Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{W, B, R, S, Q, Z, A, B\}$,
- `ORD_POST` $=\{R, S, B, Q, A, B, Z, W\}$.

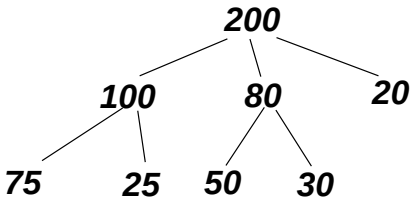
Ej. 2.- Escribir las funciones primitivas del `TAD ARBOL BINARIO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

(a) `PADRE(n,A)`
(b) `HIJO_IZQ(n,A)`
(c) `HIJO_DER(n,A)`
(d) `ETIQUETA(n,A)`
(e) `CREA2(v, A1, A2)`
(f) `ANULA(A)`

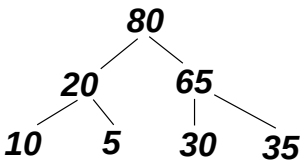
Ej. 3.- En un programa de diseño asistido por computadora, (tipo AutoCAD) se desea mantener las piezas de un sistema (por ejemplo un auto) clasificando sus partes en forma de árbol, por ejemplo



Se quiere mantener en cada hoja el peso (en Kilogramos) del componente, y en los nodos interiores el peso total de todos los componentes del subárbol que cuelga de ese nodo. Periódicamente se quiere verificar que efectivamente las etiquetas del árbol verifican esta propiedad, es decir *"que la etiqueta de los nodos interiores es la suma de las hojas del subárbol que cuelga de él. "* Así, por ejemplo, de los dos árboles siguientes, el de la izquierda verifica la condición, mientras que el de la derecha no.



Verifica



No verifica

Escribir una función `VERIFICA_SUMA(n:nodo; A: Arbol) : boolean;` que retorna `true` si el subárbol que cuelga del nodo verifica la condición y `false` en caso contrario. Usar las primitivas del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`

Ej. 4.- **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CRISIS`
$P(c) = 0.2, P(r) = 0.2, P(s) = 0.1, P(i) = 0.3,$
$P(z) = 0.05, P(u) = 0.05, P(v) = 0.05, P(t) = 0.05.$
Calcular la longitud promedio del código obtenido.

Ej. 5.- Insertar los números $27, 10, 37, 47, 22, 57, 67, 31$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Algoritmos y Estructuras de Datos.
Parcial 3. Tema 1a. [25 de Junio de 2002]

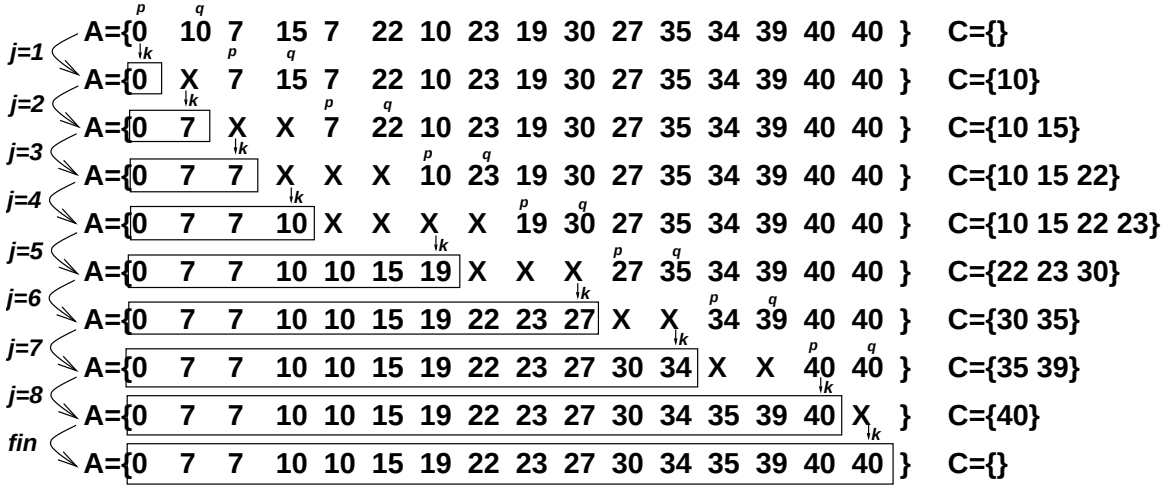
Ej. 1.- Escribir las funciones primitivas del `TAD COLA DE PRIORIDAD` representada por montículos. Incluir todas las definiciones de tipo necesarias.

(a) `ANULA`

(b) `INSERTA`

(c) `SUPRIME_MIN`

Ej. 2.- **El algoritmo mergesort modificado:** El procedimiento Mergesort se basa en la estrategia *"dividir para vencer"*. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer *"in place"*).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es *"in place"*). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j - 1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a k, marcados en la figura con una caja rectangular) y en la cola C. Entre la posición $k + 1$

y la posición $2j-1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
   p := A[2j-1]; q := A[2j]; r := min(p,q);
   pone 'max(p,q)' en C;
   pone todos los elementos de 'C' menores o iguales que 'r' en
                la cabecera de 'A' (despues de la posicion 'k');
   pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C. Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es *"in place"* el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento `MERGEQ` que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del `TAD COLA`:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) Todos los algoritmos de clasificación genérica (es decir, que se pueden aplicar a conjuntos infinitos de claves, como los enteros o los reales) tienen un número de operaciones $T(n)$ que es...

- ☐ $T(n) > O(\sqrt{n})$
- ☐ $T(n) > O(n \log(n))$

- ☐ $T(n) > O(1/n)$
- ☐ $T(n) > O(n^2)$

(b) Las operaciones `INSERTA` y `SUPRIME_MIN` sobre el `TAD COLA DE PRIORIDAD` representado por montículos requiere un número de operaciones que está relacionado con ...

- ☐ ... el número de elementos en el montículo n
- ☐ ... el cuadrado del número de elementos en el montículo n
- ☐ ... la longitud del camino en el àrbol desde la raíz hasta el punto de inserción o supresión.
- ☐ ... el logaritmo de la semilla.

(c) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log(n))$ en *promedio* y $O(n^2)$ *en el peor caso*?

- ☐ Inserción
- ☐ Clasificación rápida (*"quick-sort"*)
- ☐ Clasificación súbita (*"snake-sort"*)
- ☐ Clasificación por montículos (*"heap-sort"*)

(d) ¿Cuál de los siguientes algoritmos *"lentos"* realiza *siempre* exactamente n intercambios?

- ☐ Inserción
- ☐ Selección
- ☐ Burbuja (*"bubble-sort"*)
- ☐ Repetición (*"humble-sort"*)

Ej. 4.- Realizar las siguientes tareas

(a) Dados los enteros $\{72, 65, 65, 49, 48, 44, 36, 22, 17, 1\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

(b) Dados los enteros $\{811, 685, 405, 520, 51\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.

(c) Dados las cadenas de caracteres `{Europa, Almatea, Leda, Calixto, Metis, Ganímedes,Io}` insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Lugo, mostrar las operaciones necesarias para eliminar los elementos `{Almatea, Ganímedes}`

Algoritmos y Estructuras de Datos.
Parcial 3. Tema 1b. [25 de Junio de 2002]

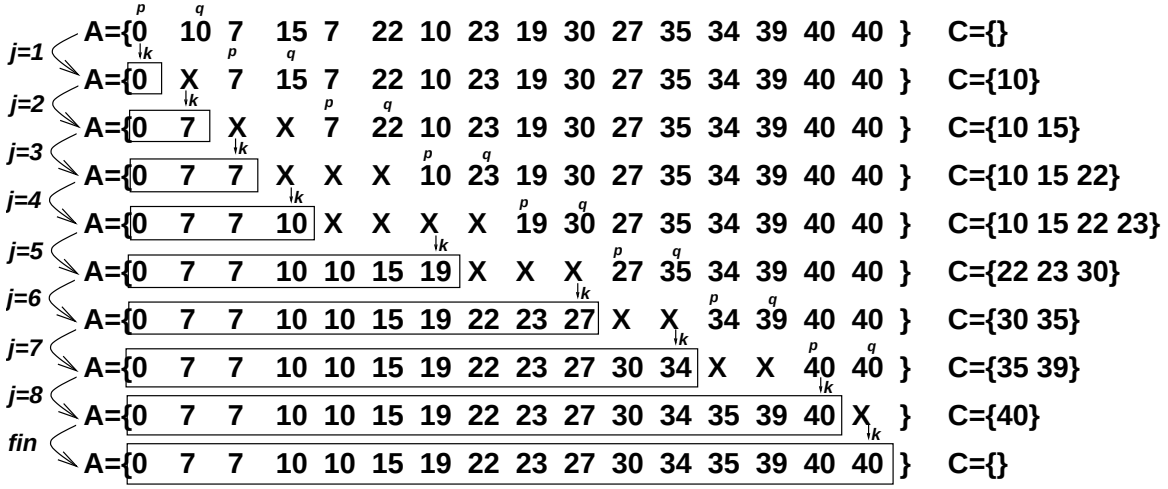
Ej. 1.- Escribir las funciones primitivas del `TAD COLA DE PRIORIDAD` representada por montículos. Incluir todas las definiciones de tipo necesarias.

(a) `ANULA`

(b) `INSERTA`

(c) `SUPRIME_MIN`

Ej. 2.- **El algoritmo mergesort modificado:** El procedimiento Mergesort se basa en la estrategia *"dividir para vencer"*. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer *"in place"*).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es *"in place"*). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j - 1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a k, marcados en la figura con una caja rectangular) y en la cola C. Entre la posición $k + 1$

y la posición $2j-1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
   p := A[2j-1]; q := A[2j]; r := min(p,q);
   pone 'max(p,q)' en C;
   pone todos los elementos de 'C' menores o iguales que 'r' en
                la cabecera de 'A' (despues de la posicion 'k');
   pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C. Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es *"in place"* el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento `MERGEQ` que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del `TAD COLA`:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) Todos los algoritmos de clasificación genérica (es decir, que se pueden aplicar a conjuntos infinitos de claves, como los enteros o los reales) tienen un número de operaciones $T(n)$ que es...

- ☐ $T(n) > O(n^2)$
- ☐ $T(n) > O(1/n)$

- ☐ $T(n) > O(n\log(n))$
- ☐ $T(n) > O(\sqrt{n})$

(b) Las operaciones `INSERTA` y `SUPRIME_MIN` sobre el `TAD COLA DE PRIORIDAD` representado por montículos requiere un número de operaciones que está relacionado con ...

- ☐ ... el cuadrado del número de elementos en el montículo n
- ☐ ... el logaritmo de la semilla.
- ☐ ... la longitud del camino en el àrbol desde la raíz hasta el punto de inserción o supresión.
- ☐ ... el número de elementos en el montículo n

(c) ¿Cuál de los siguientes algoritmos de clasificación es $O(n\log(n))$ en *promedio* y $O(n^2)$ *en el peor caso*?

- ☐ Clasificación por montículos (*"heap-sort"*)
- ☐ Inserción
- ☐ Clasificación rápida (*"quick-sort"*)
- ☐ Clasificación súbita (*"snake-sort"*)

(d) ¿Cuál de los siguientes algoritmos *"lentos"* realiza *siempre* exactamente n intercambios?

- ☐ Inserción
- ☐ Selección
- ☐ Repetición (*"humble-sort"*)
- ☐ Burbuja (*"bubble-sort"*)

Ej. 4.- Realizar las siguientes tareas

(a) Dados los enteros $\{97, 85, 78, 63, 62, 58, 50, 33, 19, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

(b) Dados los enteros $\{685, 405, 520, 51, 811\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.

(c) Dados las cadenas de caracteres `{Borodin, Vivaldi, Mendelsohn, Strauss, Beethoven, Mozart, Litz}` insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Lugo, mostrar las operaciones necesarias para eliminar los elementos `{Vivaldi, Mozart}`

Algoritmos y Estructuras de Datos.
Parcial 3. Tema 1c. [25 de Junio de 2002]

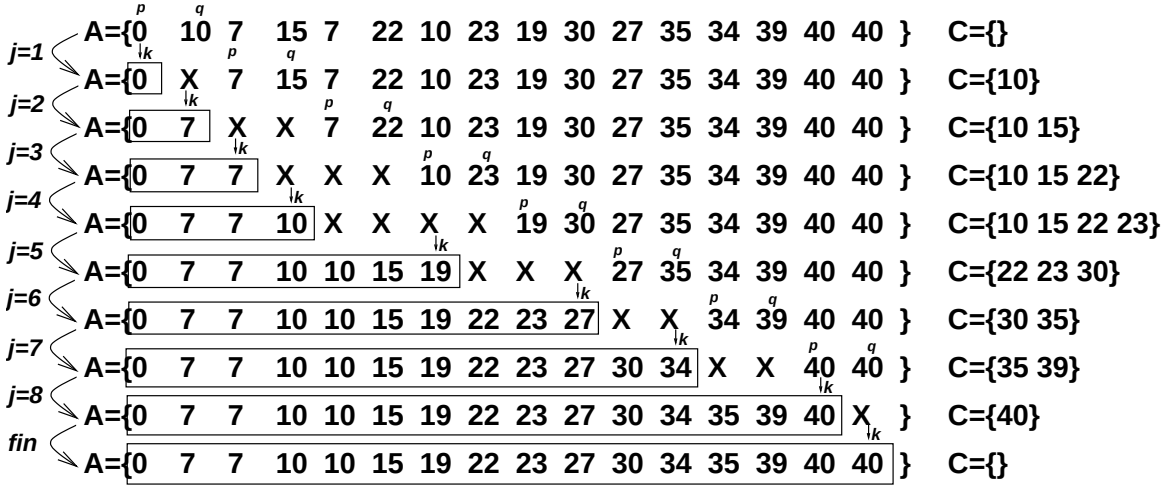
Ej. 1.- Escribir las funciones primitivas del `TAD COLA DE PRIORIDAD` representada por montículos. Incluir todas las definiciones de tipo necesarias.

(a) `ANULA`

(b) `INSERTA`

(c) `SUPRIME_MIN`

Ej. 2.- **El algoritmo mergesort modificado:** El procedimiento Mergesort se basa en la estrategia *"dividir para vencer"*. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer *"in place"*).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es *"in place"*). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j - 1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a k, marcados en la figura con una caja rectangular) y en la cola C. Entre la posición $k + 1$

y la posición $2j-1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
   p := A[2j-1]; q := A[2j]; r := min(p,q);
   pone 'max(p,q)' en C;
   pone todos los elementos de 'C' menores o iguales que 'r' en
                la cabecera de 'A' (despues de la posicion 'k');
   pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C. Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es *"in place"* el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento `MERGEQ` que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del `TAD COLA`:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) Todos los algoritmos de clasificación genérica (es decir, que se pueden aplicar a conjuntos infinitos de claves, como los enteros o los reales) tienen un número de operaciones $T(n)$ que es...

- ☐ $T(n) > O(n^2)$
- ☐ $T(n) > O(n \log(n))$

- ☐ $T(n) > O(\sqrt{n})$
- ☐ $T(n) > O(1/n)$

(b) Las operaciones `INSERTA` y `SUPRIME_MIN` sobre el `TAD COLA DE PRIORIDAD` representado por montículos requiere un número de operaciones que está relacionado con ...

- ☐ ... el número de elementos en el montículo n
- ☐ ... el logaritmo de la semilla.
- ☐ ... el cuadrado del número de elementos en el montículo n
- ☐ ... la longitud del camino en el àrbol desde la raíz hasta el punto de inserción o supresión.

(c) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log(n))$ en *promedio* y $O(n^2)$ *en el peor caso*?

- ☐ Inserción
- ☐ Clasificación rápida (*"quick-sort"*)
- ☐ Clasificación súbita (*"snake-sort"*)
- ☐ Clasificación por montículos (*"heap-sort"*)

(d) ¿Cuál de los siguientes algoritmos *"lentos"* realiza *siempre* exactamente n intercambios?

- ☐ Burbuja (*"bubble-sort"*)
- ☐ Inserción
- ☐ Selección
- ☐ Repetición (*"humble-sort"*)

Ej. 4.- Realizar las siguientes tareas

(a) Dados los enteros $\{99, 96, 84, 83, 67, 48, 40, 33, 27, 8\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

(b) Dados los enteros $\{405, 520, 51, 811, 685\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.

(c) Dados las cadenas de caracteres `{Julieta, Rosalinda, Ariel, Titania, Oberon, Belinda, Miranda}` insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Lugo, mostrar las operaciones necesarias para eliminar los elementos `{Rosalinda, Belinda}`

Algoritmos y Estructuras de Datos.
Parcial 3. Tema 2a. [25 de Junio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` listadas a continuación, para árboles binarios de búsqueda:

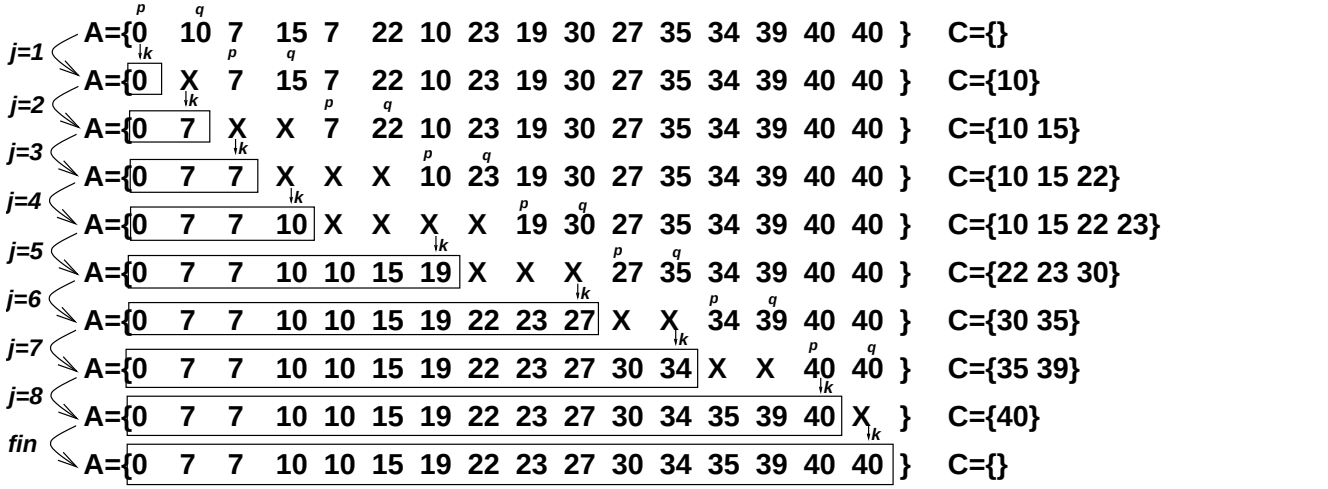
(a) `ANULA(A)`

(b) `INSERTA(x,A)`

(c) `SUPRIME(x,A)`

(d) `MIEMBRO(X,A)`

Ej. 2.- **El algoritmo mergesort modificado:** El procedimiento Mergesort se basa en la estrategia *"dividir para vencer"*. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer *"in place"*).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es *"in place"*). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j-1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a

k, marcados en la figura con una caja rectangular) y en la cola C. Entre la posición $k+1$ y la posición $2j-1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
   p := A[2j-1]; q := A[2j]; r := min(p,q);
   pone 'max(p,q)' en C;
   pone todos los elementos de 'C' menores o iguales que 'r' en
                la cabecera de 'A' (despues de la posicion 'k');
   pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C. Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es *"in place"* el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento `MERGEQ` que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del `TAD COLA`:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log(n))$?

- ☐ Burbuja (*"bubble-sort"*)
- ☐ Clasificación por montículos (*"heap-sort"*)
- ☐ Reducción óptima (*"indian-tonic-sort"*)

☐ Inserción

(b) El mejor caso para el algoritmo de clasificación rápida (*"quick-sort"*) es cuando el pivote v particiona la secuencia en dos secuencias de longitud n_1 y n_2 tales que

☐ ... $n_1 \ll n_2$
☐ ... $n_1 \approx n_2$
☐ ... $n_1 \gg n_2$
☐ ... $n_1 \approx \sqrt{n_2}$

(c) En un *montículo* con n elementos, el número de niveles es ...

☐ ... $O(2^n)$
☐ ... $O(\log(n))$
☐ ... $O(n)$
☐ ... $O(n^2)$

(d) El algoritmo de clasificación por urnas tiene un tiempo de ejecución $T(n) = O(n)$ pero está restringido a que el número de claves posibles ...

☐ ... sea una potencia de 2.
☐ ... sea finito.
☐ ... sea delgado.
☐ ... sea infinito.

Ej. 4.- Realizar las siguientes tareas

(a) Dados los enteros $\{89, 87, 48, 44, 42, 40, 40, 36, 33, 27\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

(b) Dados los enteros $\{685, 520, 811, 405, 71\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.

(c) Dados las cadenas de caracteres `{Picasso, Escher, van Gogh, Miro, Monet, Da Vinci, Rembrandt}` insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Lugo, mostrar las operaciones necesarias para eliminar los elementos `{Escher, van Gogh}`

Algoritmos y Estructuras de Datos.
Parcial 3. Tema 2b. [25 de Junio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` listadas a continuación, para árboles binarios de búsqueda:

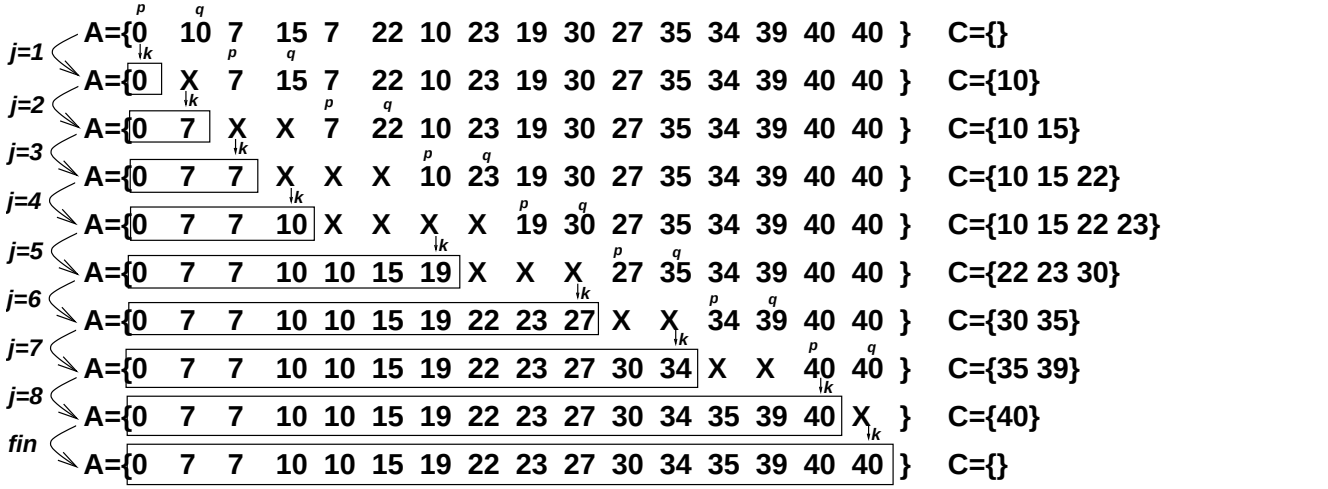
(a) `ANULA(A)`

(b) `INSERTA(x,A)`

(c) `SUPRIME(x,A)`

(d) `MIEMBRO(X,A)`

Ej. 2.- **El algoritmo mergesort modificado:** El procedimiento Mergesort se basa en la estrategia *"dividir para vencer"*. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer *"in place"*).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es *"in place"*). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j - 1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a

k, marcados en la figura con una caja rectangular) y en la cola C. Entre la posición $k+1$ y la posición $2j-1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
   p := A[2j-1]; q := A[2j]; r := min(p,q);
   pone 'max(p,q)' en C;
   pone todos los elementos de 'C' menores o iguales que 'r' en
                la cabecera de 'A' (despues de la posicion 'k');
   pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C. Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es *"in place"* el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento `MERGEQ` que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del `TAD COLA`:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log(n))$?

- ☐ Burbuja (*"bubble-sort"*)
- ☐ Inserción
- ☐ Reducción óptima (*"indian-tonic-sort"*)

☐ Clasificación por montículos (*"heap-sort"*)

(b) El mejor caso para el algoritmo de clasificación rápida (*"quick-sort"*) es cuando el pivote v particiona la secuencia en dos secuencias de longitud n_1 y n_2 tales que

☐ ... $n_1 \approx \sqrt{n_2}$
☐ ... $n_1 \ll n_2$
☐ ... $n_1 \gg n_2$
☐ ... $n_1 \approx n_2$

(c) En un *montículo* con n elementos, el número de niveles es ...

☐ ... $O(2^n)$
☐ ... $O(n)$
☐ ... $O(\log(n))$
☐ ... $O(n^2)$

(d) El algoritmo de clasificación por urnas tiene un tiempo de ejecución $T(n) = O(n)$ pero está restringido a que el número de claves posibles ...

☐ ... sea una potencia de 2.
☐ ... sea finito.
☐ ... sea delgado.
☐ ... sea infinito.

Ej. 4.- Realizar las siguientes tareas

(a) Dados los enteros $\{93, 87, 63, 61, 32, 24, 17, 16, 10, 2\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

(b) Dados los enteros $\{520, 811, 405, 71, 685\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.

(c) Dados las cadenas de caracteres `{Pandora, Hiperon, Pan, Mimas, Atlas, Helena, Calipso}` insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Lugo, mostrar las operaciones necesarias para eliminar los elementos `{Hiperon, Pan}`

Algoritmos y Estructuras de Datos.
Parcial 3. Tema 2c. [25 de Junio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` listadas a continuación, para árboles binarios de búsqueda:

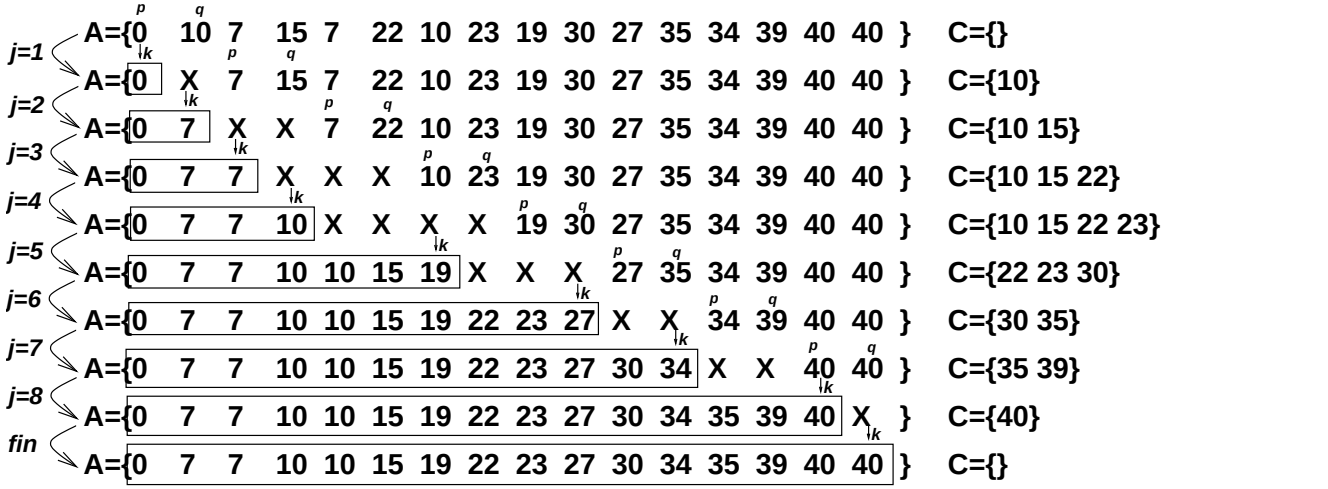
(a) `ANULA(A)`

(b) `INSERTA(x,A)`

(c) `SUPRIME(x,A)`

(d) `MIEMBRO(X,A)`

Ej. 2.- **El algoritmo mergesort modificado:** El procedimiento Mergesort se basa en la estrategia *"dividir para vencer"*. Primero se ordenan las posiciones impares entre sí (recursivamente), luego las pares, y luego se intercalan ambas subsecuencias. Si el intercalamiento se puede hacer en tiempo $O(n)$ entonces puede demostrarse que el número de operaciones total es $O(n \log(n))$ en el peor caso. Ya hemos visto para la representación de conjuntos como listas enlazadas clasificadas que el intercalamiento de dos listas enlazadas clasificadas se puede hacer en tiempo $O(n)$. Para vectores no es tan sencillo, y se conjetura que no se puede hacer tal operación en tiempo $O(n)$ sin uso de memoria adicional (es decir, no se puede hacer *"in place"*).



Se propone el siguiente algoritmo de intercalación para vectores que utiliza una cola auxiliar C (y por lo tanto no es *"in place"*). Por simplicidad asumimos que el número de elementos en el arreglo n es par, que las posiciones impares y pares están clasificadas entre sí, e inicialmente la cola C está vacía. Considerando el arreglo A de longitud $n = 16$ de la figura, el algoritmo lo ordena en $n/2 = 8$ pasos. En cada paso el algoritmo ordena los elementos p, q en las posiciones $2j-1$ y $2j$. Después del paso j los primeros $2j$ elementos están ordenados y ubicados en parte en la cabecera del vector (posiciones 1 a

k, marcados en la figura con una caja rectangular) y en la cola C. Entre la posición $k+1$ y la posición $2j-1$ hay una serie de elementos (tantos como hay en C) cuyos valores son irrelevantes (en la figura están marcados con una X). El algoritmo es el siguiente

```
for j:=1 to n div 2 do begin
   p := A[2j-1]; q := A[2j]; r := min(p,q);
   pone 'max(p,q)' en C;
   pone todos los elementos de 'C' menores o iguales que 'r' en
                la cabecera de 'A' (despues de la posicion 'k');
   pone 'r' en la cabecera de 'A';
end;
pone todos los elementos restantes de 'C' en la cabecera de 'A';
```

Así, por ejemplo, en el paso 6 del ejemplo de la figura consideramos los elementos $p = 27$, $q = 35$ en las posiciones 11 y 12. El mínimo de ambos es $r = 27$ de manera que insertamos 35 en C. Tomamos los elementos de C menores que r (22 y 23), y los ponemos en la cabecera de A y finalmente ponemos r en la cabecera. Durante este paso el número de elementos ordenados en la cabecera se incrementó en 3 y la longitud de la cola C se redujo en 1. Si bien este algoritmo no es *"in place"* el tamaño de la cola C es en promedio de longitud $O(\sqrt{n})$ (puede ser n en el peor caso).

Consigna: Escribir un procedimiento `MERGEQ` que intercala dos secuencias de acuerdo al algoritmo descripto previamente.

```
tipo_elemento = integer;
type vector = array[1..N] of tipo_elemento;
procedure MERGEQ(var A:vector; N:tipo_elemento);
```

Usar las primitivas del `TAD COLA`:

```
procedure ANULA(var C:cola);
procedure PONE(x:tipo_elemento; var C:cola);
procedure QUITA(var C:cola);
function FRENTE(C:cola): tipo_elemento;
function VACIA(C:cola): boolean;
```

Ej. 3.- Preguntas sobre clasificación: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) ¿Cuál de los siguientes algoritmos de clasificación es $O(n \log(n))$?

- ☐ Inserción
- ☐ Burbuja (*"bubble-sort"*)
- ☐ Clasificación por montículos (*"heap-sort"*)

☐ Reducción óptima (*"indian-tonic-sort"*)

(b) El mejor caso para el algoritmo de clasificación rápida (*"quick-sort"*) es cuando el pivote v particiona la secuencia en dos secuencias de longitud n_1 y n_2 tales que

☐ ... $n_1 \approx \sqrt{n_2}$
☐ ... $n_1 \gg n_2$
☐ ... $n_1 \ll n_2$
☐ ... $n_1 \approx n_2$

(c) En un *montículo* con n elementos, el número de niveles es ...

☐ ... $O(n)$
☐ ... $O(2^n)$
☐ ... $O(\log(n))$
☐ ... $O(n^2)$

(d) El algoritmo de clasificación por urnas tiene un tiempo de ejecución $T(n) = O(n)$ pero está restringido a que el número de claves posibles ...

☐ ... sea infinito.
☐ ... sea una potencia de 2.
☐ ... sea delgado.
☐ ... sea finito.

Ej. 4.- Realizar las siguientes tareas

(a) Dados los enteros $\{89, 67, 59, 53, 51, 43, 20, 12, 9, 8\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

(b) Dados los enteros $\{811, 405, 71, 685, 520\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se debe utilizar para ubicar los elementos en las urnas durante cada pasada.

(c) Dados las cadenas de caracteres `{Nereida, Triton, Larisa, Galatea, Bianca, Talasa, Despina}` insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Lugo, mostrar las operaciones necesarias para eliminar los elementos `{Triton, Despina}`

TEMA: **1a**
Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio Globalizador. Tema 1a. [2 de Julio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias.

(a) `INSERTA(x,p,L)`,
(b) `LOCALIZA(x,L)`,
(c) `RECUPERA(p,L)`,
(d) `SUPRIME(p,L)`,
(e) `SIGUIENTE(p,L)`,
(f) `ANULA(L)`,
(g) `PRIMERO(L)`, y
(h) `FIN(L)`.

Ej. 2.- Dos árboles ordenados T y T' se dicen *"semejantes"* si tienen la misma estructura. Formalmente esto significa que

- ambos son vacíos, **ó**
- ambos son no vacíos y los subárboles de sus hijos correspondientes son semejantes.

Informalmente decimos que T y T' tienen ambos la misma *"forma"*. **Consigna:** Escriba una función

```
function SEMEJANTE(T,T': nodo) : boolean;
```

que retorna `true` si los árboles son semejantes y `false` caso contrario. Usar las primitivas de árbol ordenado orientado siguientes:

(a) `PADRE(n,A)`
(b) `HIJO_MAS_IZQ(n,A)`
(c) `HERMANO_DER(n,A)`

Ej. 3.- Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{S,T,A,V,F,X,G,Z,H\}$,
- `ORD_POST` $=\{A,T,F,G,X,H,Z,V,S\}$.

(b) **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `MUNDIAL`
$P(d) = 0.2, P(i) = 0.25, P(a) = 0.25, P(l) = 0.2,$
$P(m) = 0.03, P(u) = 0.03, P(n) = 0.02, P(f) = 0.02.$
Calcular la longitud promedio del código obtenido.

(c) **Tablas de dispersión:** Insertar los números $9, 22, 32, 15, 14, 42, 24, 11, 34$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Ej. 4.- Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) El tiempo de inserción y supresión de un elemento para el `TAD LISTA` implementado con listas simplemente enlazadas por cursores, en cualquier posición dentro de la misma es (n es el número de elementos en la lista.)

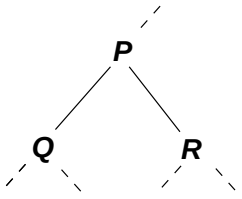
- ☐ $T(n) = O(1)$
- ☐ $T(n) = O(n)$
- ☐ $T(n) = O(\sqrt{n})$
- ☐ $T(n) = O(1/n)$

(b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"heapsort"*) es $O(n \log(n)$ (n es el número de elementos a ordenar) ...

- ☐ ... siempre.
- ☐ ... a veces.
- ☐ ... nunca.
- ☐ ... en el mejor caso.

(c) Una de las representaciones más eficientes para el `TAD COLA DE PRIORIDAD` es la implementación por montículos. El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si P, Q, R son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ $Q + R \leq \infty$.
- ☐ $Q \leq P \leq R$.
- ☐ $P \leq \frac{1}{2}(Q + R)$.
- ☐ $P \leq leQ, R$.



(d) ¿Cuál de los siguientes algoritmos de clasificación es el más rápido *en el caso promedio*?

- ☐ Clasificación rápida (*"Quick-sort"*)
- ☐ Selección

☐ Burbuja (*"bubble-sort"*)
☐ Clasificación telescópica (*"hubble-sort"*)

TEMA: **1b**
Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio Globalizador. Tema 1b. [2 de Julio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias.

(a) `INSERTA(x,p,L)`,
(b) `LOCALIZA(x,L)`,
(c) `RECUPERA(p,L)`,
(d) `SUPRIME(p,L)`,
(e) `SIGUIENTE(p,L)`,
(f) `ANULA(L)`,
(g) `PRIMERO(L)`, y
(h) `FIN(L)`.

Ej. 2.- Dos árboles ordenados T y T' se dicen *"semejantes"* si tienen la misma estructura. Formalmente esto significa que

- ambos son vacíos, **ó**
- ambos son no vacíos y los subárboles de sus hijos correspondientes son semejantes.

Informalmente decimos que T y T' tienen ambos la misma *"forma"*. **Consigna:** Escriba una función

```
function SEMEJANTE(T,T': nodo) : boolean;
```

que retorna `true` si los árboles son semejantes y `false` caso contrario. Usar las primitivas de árbol ordenado orientado siguientes:

(a) `PADRE(n,A)`
(b) `HIJO_MAS_IZQ(n,A)`
(c) `HERMANO_DER(n,A)`

Ej. 3.- Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{W, A, C, D, X, T, U, B\}$,
- `ORD_POST` $=\{A, X, T, U, D, C, B, W\}$.

(b) **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `MUNDIAL`
$P(i) = 0.2, P(a) = 0.25, P(l) = 0.25, P(d) = 0.2,$
$P(m) = 0.03, P(u) = 0.03, P(n) = 0.02, P(f) = 0.02.$
Calcular la longitud promedio del código obtenido.

(c) **Tablas de dispersión:** Insertar los números $8, 21, 31, 14, 13, 41, 23, 10, 33$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Ej. 4.- Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) El tiempo de inserción y supresión de un elemento para el `TAD LISTA` implementado con listas simplemente enlazadas por cursores, en cualquier posición dentro de la misma es (n es el número de elementos en la lista.)

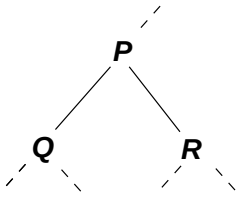
- ☐ $T(n) = O(1/n)$
- ☐ $T(n) = O(1)$
- ☐ $T(n) = O(n)$
- ☐ $T(n) = O(\sqrt{n})$

(b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"heapsort"*) es $O(n \log(n)$ (n es el número de elementos a ordenar) ...

- ☐ ... siempre.
- ☐ ... a veces.
- ☐ ... en el mejor caso.
- ☐ ... nunca.

(c) Una de las representaciones más eficientes para el `TAD COLA DE PRIORIDAD` es la implementación por montículos. El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si P, Q, R son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ $P \leq leQ, R.$
- ☐ $Q + R \leq \infty.$
- ☐ $P \leq \frac{1}{2}(Q + R).$
- ☐ $Q \leq P \leq R.$



(d) ¿Cuál de los siguientes algoritmos de clasificación es el más rápido *en el caso promedio*?

- ☐ Burbuja (*"bubble-sort"*)
- ☐ Selección

- ☐ Clasificación rápida (*"Quick-sort"*)
- ☐ Clasificación telescópica (*"hubble-sort"*)

TEMA: **1c**
Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio Globalizador. Tema 1c. [2 de Julio de 2002]

Ej. 1.- Escribir las funciones primitivas del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias.

(a) `INSERTA(x,p,L)`,
(b) `LOCALIZA(x,L)`,
(c) `RECUPERA(p,L)`,
(d) `SUPRIME(p,L)`,
(e) `SIGUIENTE(p,L)`,
(f) `ANULA(L)`,
(g) `PRIMERO(L)`, y
(h) `FIN(L)`.

Ej. 2.- Dos árboles ordenados T y T' se dicen *"semejantes"* si tienen la misma estructura. Formalmente esto significa que

- ambos son vacíos, **ó**
- ambos son no vacíos y los subárboles de sus hijos correspondientes son semejantes.

Informalmente decimos que T y T' tienen ambos la misma *"forma"*. **Consigna:** Escriba una función

```
function SEMEJANTE(T,T': nodo) : boolean;
```

que retorna `true` si los árboles son semejantes y `false` caso contrario. Usar las primitivas de árbol ordenado orientado siguientes:

(a) `PADRE(n,A)`
(b) `HIJO_MAS_IZQ(n,A)`
(c) `HERMANO_DER(n,A)`

Ej. 3.- Ejercicios operativos:

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{H, T, Q, C, D, R, B, A\}$,
- `ORD_POST` $=\{C, D, Q, B, R, A, T, H\}$.

(b) **Árboles de Huffman:** Dados los carácteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `MUNDIAL`
$P(a) = 0.2, P(l) = 0.25, P(d) = 0.25, P(i) = 0.2,$
$P(m) = 0.03, P(u) = 0.03, P(n) = 0.02, P(f) = 0.02.$
Calcular la longitud promedio del código obtenido.

(c) **Tablas de dispersión:** Insertar los números $7, 20, 30, 13, 12, 40, 22, 19, 32$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Ej. 4.- Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!]

(a) El tiempo de inserción y supresión de un elemento para el `TAD LISTA` implementado con listas simplemente enlazadas por cursores, en cualquier posición dentro de la misma es (n es el número de elementos en la lista.)

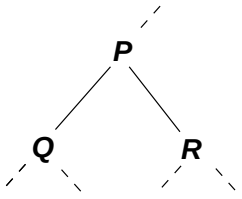
- ☐ $T(n) = O(1)$
- ☐ $T(n) = O(1/n)$
- ☐ $T(n) = O(n)$
- ☐ $T(n) = O(\sqrt{n})$

(b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"heapsort"*) es $O(n \log(n)$ (n es el número de elementos a ordenar) ...

- ☐ ... nunca.
- ☐ ... a veces.
- ☐ ... siempre.
- ☐ ... en el mejor caso.

(c) Una de las representaciones más eficientes para el `TAD COLA DE PRIORIDAD` es la implementación por montículos. El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si P, Q, R son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ $P \leq leQ, R.$
- ☐ $Q \leq P \leq R.$
- ☐ $P \leq \frac{1}{2}(Q + R).$
- ☐ $Q + R \leq \infty.$



(d) ¿Cuál de los siguientes algoritmos de clasificación es el más rápido *en el caso promedio*?

- ☐ Clasificación rápida (*"Quick-sort"*)
- ☐ Burbuja (*"bubble-sort"*)

- ☐ Clasificación telescópica (*"hubble-sort"*)
- ☐ Selección

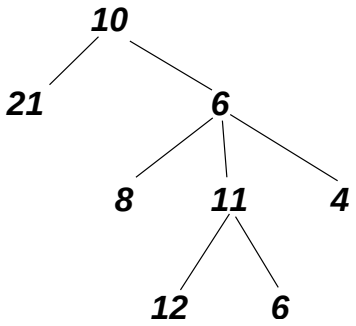
Algoritmos y Estructuras de Datos. Examen Final. [9 de mayo de 2002]

Ej. 1.- Escriba una implementación para el TAD CONJUNTO con listas simplemente enlazadas clasificadas implementadas con punteros o cursores. Las operaciones a implementar son: `ANULA`, `INTERSECCION`, `UNION`, `DIFERENCIA`, `MIEMBRO`, `INSERTA` y `SUPRIME`. Incluir los tipos y definiciones necesarios.

Ej. 2.- Escribir un procedimiento `procedure INTERCAMBIA(var L:lista);` que intercambia los elementos de `L` de tal forma que aquellos que están en una posición impar son intercambiados con el elemento siguiente. Es decir el primero es intercambiado con el segundo, el tercero con el cuarto y así siguiendo. Si el número de elementos es impar, entonces el último elemento queda inalterado. Así por ejemplo, si `L`=$\{10, 1, 15, 7, 2, 19, 15, 16, 1\}$ entonces despues de intercambia debemos tener `L`=$\{1, 10, 7, 15, 19, 2, 16, 15, 1\}$.

Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 3.- Escribir una función `function SUMA_PAR_ANTEC(n:nodo;a:arbol) : integer;` que devuelve la suma de las etiquetas de los nodos tales que su etiqueta y la etiqueta de todos los antecesores es par. Se sugiere hacerlo en forma recursiva, notando que si la etiqueta de un nodo es impar, entonces `SUMA_PAR_ANTEC` de todo el subárbol correspondiente es nulo. Por, ejemplo para el árbol de la figura, `SUMA_PAR_ANTEC` da 28. Notar que las etiquetas 6 y 12 no son incluidas ya que su antecesor 11 es impar.

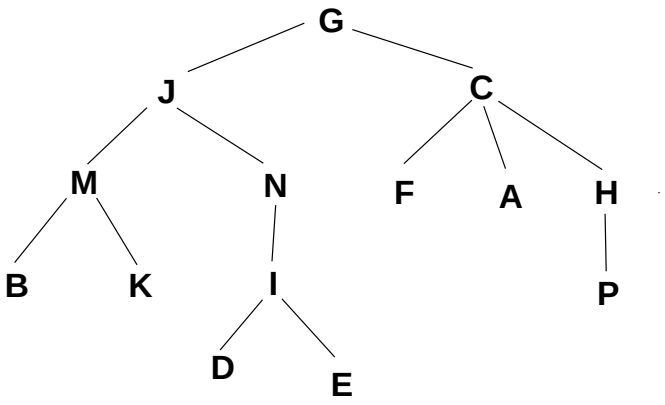


Utilizar las primitivas del **TAD ARBOL ORDENADO**: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`, `RAIZ(n,A)`, `ANULA(L)`.

Ej. 4.- [**LIBRES**] Dados los enteros $\{15, 3, 10, 9, 7, 4, 2\}$ ordenarlos por el métodos de *"montículos"*. Mostrar el montículo antes y después de cada inserción.

Ej. 5.- [**LIBRES**] Dados los enteros $\{10, 9, 12, 6, 14, 11, 7, 5, 13, 19, 2\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 10, 7 y 11.

Ej. 6.- [**LIBRES**] Dado el siguiente árbol



(a) Listar en orden previo, posterior y simétrico las etiquetas.

(b) ¿ Cuáles son las profundidades y alturas de los nodos G, N, E y C?

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayuscula de imprenta GRANDE]

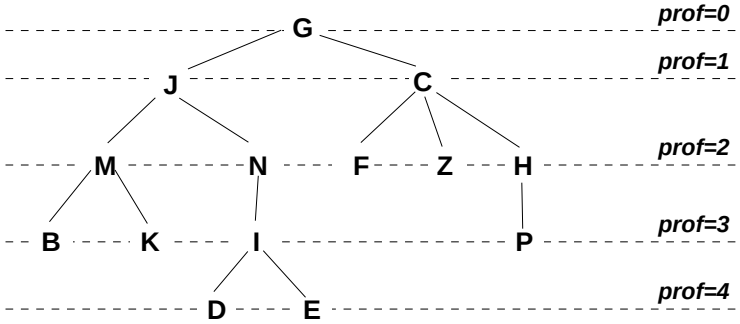
Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [13 de Febrero de 2003]

Ej. 1.- Escribir las funciones primitivas del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. Es decir, implementar en Pascal los siguientes procedimientos/funciones listadas abajo. Incluir todas las definiciones de tipo necesarias. `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 2.- Escribir un función `function CUENTA_PROF(n:nodo; m:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raíz es `n` que están a profundidad `m` o menor (con respecto a `n`). Por ejemplo, para el árbol de la figura debe retornar

```
CUENTA_PROF(G,2,G) -> 8
CUENTA_PROF(J,1,G) -> 3
CUENTA_PROF(N,3,G) -> 4
```



Usar las primitivas de árbol ordenado orientado siguientes:
`HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. *Sugerencia:* Hacer la función *recursiva*. Notar que, por ejemplo:

```
CUENTA_PROF(G,2,G) = 1 + CUENTA_PROF(J,1,G) + CUENTA_PROF(C,1,G)
```

La recursividad de la función debe cortar cuando `n`$= \Lambda$ o $m < 0$.

Ej. 3.- Ejercicios básicos sobre TAD's

(a) Escribir un procedimiento `procedure SACAPAR(var L:lista; C:cola);` que apendiza a la lista `L` todos los elementos de `C` que son pares, los cuales a su vez deben ser removidos de `C`. Se puede usar una estructura auxiliar (cola o lista). Por ejemplo, si inicialmente `L`=$\{2,3,4\}$ y `C`=$\{1,6,3,5,2,8\}$ entonces, después de hacer `SACAPAR(L,C)` debe quedar `L`=$\{2,3,4,6,2,8\}$ y `C`=$\{1,3,5\}$. Utilizar las siguientes primitivas:

- **TAD LISTA:** `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.
- **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`

(b) Escribir un procedimiento `procedure ELIMINA_VALOR(var C:cola; n: integer);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C =` {1,3,5,4,2,3,7,3,5}, después de `ELIMINA_VALOR(C,3)` debe quedar `C =` {1,5,4,2,7,5}. *Sugerencia: Usar una estructura auxiliar lista o cola.* El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original. Utilizar las primitivas del **TAD COLA** listadas en el ejemplo anterior.

Ej. 4.- [**LIBRES**] **Ejercicios operativos:**

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Z, S, T, U, W, Q, R, B\}$.
- `ORD_POST` $=\{T, W, U, S, Z, R, B, Q, A\}$.

(b) [**LIBRES**] Dados los enteros $\{9, 8, 11, 5, 13, 10, 6, 4, 12, 18, 1\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 9, 6 y 10.

Ej. 5.- [**LIBRES**] **Preguntas:** [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) Dadas las funciones $T_1(n) = 5n + \log n$, $T_2(n) = 4n^2 + \sqrt{n}$, $T_3(n) = 2^n + n!$ y $T_4(n) = \sqrt(n) + \log n$ decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_3 < T_4 < T_1 < T_2$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_2 < T_1 < T_4 < T_3$
- ☐ $T_4 < T_3 < T_2 < T_1$

(b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"quicksort"*) es $O(n \log(n)$ (n es el número de elementos a ordenar) ...

- ☐ ... siempre.
- ☐ ... cuando el vector ya está ordenado.
- ☐ ... nunca.
- ☐ ... en el caso promedio.

(c) Una ventaja del método de clasificación por selección, en comparación con otros algoritmos *lentos*, es que realiza sólo n *intercambios*...

- ☐ ... a veces.
- ☐ ... cuando el vector está ordenado.
- ☐ ... siempre.
- ☐ ... cuando el vector está desordenado.

(d) ¿Cuál es el tiempo de ejecución del procedimiento de clasificación por *incrementos decrecientes* (*shell-sort*) en el caso promedio?

- ☐ $O(n^{1.3})$
- ☐ $O(n^{1.5})$
- ☐ $O(\log n)$
- ☐ $O(n!)$

Apellido y Nombre: _____

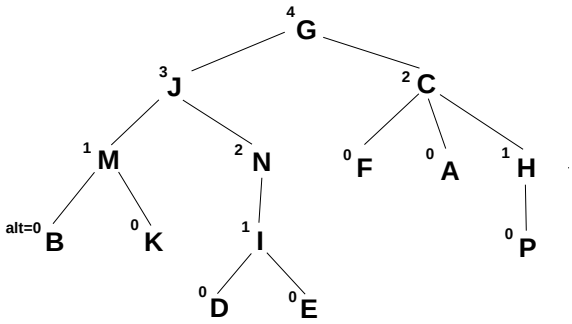
Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos. Examen Final. [27 de Febrero de 2003]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado mediante *listas enlazadas clasificadas*. Es decir, implementar en Pascal los siguientes procedimientos/funciones listados abajo. Incluir todas las definiciones de tipo necesarias. `ANULA(A)`, `UNION(A,B,C)`, `INTERSECCION(A,B,C)`, `DIFERENCIA(A,B,C)`, `MIEMBRO(x,A)`, `MIN(A)`, `INSERTA(x,A)` y `SUPRIME(x,A)`.

Ej. 2.- Recordemos que la *altura* de un nodo en un árbol, es la máxima longitud de los caminos que van desde el nodo a una hoja descendiente del mismo. Por ejemplo, consideremos el árbol de la figura. El número que está arriba y a la izquierda de cada nodo es la altura del nodo.



Escribir una función `function CUENTA_ALT(n:nodo; m:integer; var altura:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raiz es `n` tales que su altura es menor o igual que `m`. Además, en la variable `altura` debe retornar la altura del nodo `n`. Por lo tanto las siguientes llamadas deben retornar, para el árbol del ejemplo,

```
CUENTA_ALT(G,0,altura,G) -> 7
CUENTA_ALT(G,1,altura,G) -> 10
CUENTA_ALT(G,2,altura,G) -> 12
CUENTA_ALT(G,3,altura,G) -> 13
CUENTA_ALT(G,4,altura,G) -> 14
```

En todos estos casos, la variable `altura` debe retornar `altura=4`. Por otra parte, si nos referimos al nodo `N`, entonces debe retornar

```
CUENTA_ALT(N,0,altura,G) -> 2
CUENTA_ALT(N,1,altura,G) -> 3
CUENTA_ALT(N,2,altura,G) -> 4
```

y `altura=2`.

Usar las primitivas de árbol ordenado orientado siguientes: `HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. Hacer la función *recursiva*. Notar que el conteo de nodos que cumple la condición para un nodo dado es igual a la suma sobre los hijos más 1 o 0, dependiendo de si la altura del nodo es menor que `m` o no.

Ej. 3.- Ejercicios básicos sobre TAD's

(a) Escribir un procedimiento `procedure STRIDE(var L1:lista; L:lista; c, f, i: integer);` que retorna en `L1` todos le elementos en las posiciones desde el comienzo `c` hasta el fin `f` (pero sin incluirlo) a intervalos `i`. Es decir, debe retornar los elementos en las posiciones `c, c+i, c+2*i, ...`, mientras éstas sean menores que `f` y esten dentro del rango de posiciones válidas en la lista. Por ejemplo, si `L=(3,2,1,5,4,2,3,2,6)` y llamamos `STRIDE(L1,L,2,9,3)`, entonces debe retornar `L1=(2,4,2)`, mientras que si hacemos `STRIDE(L1,L,3,20,4)` debe retornar `L1=(1,3)` Utilizar las siguientes primitivas:

- **TAD LISTA:** `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) Escribir una función `function SUMA_COLA(C:cola): integer;` que calcula la suma de los elementos de una cola de enteros. La cola original debe quedar inalterada. *Sugerencia:* usar una variable auxiliar (cola o lista). Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

Ej. 4.- [**LIBRES**] **Ejercicios operativos:**

(a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{M, G, Z, R, N, A, B, D, E\}$.
- `ORD_POST` $=\{Z, R, G, A, D, E, B, N, M\}$.

(b) [**LIBRES**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PACIENCIA` $P(P) = 0.3, P(A) = 0.1, P(C) = 0.3, P(I) = 0.05, P(E) = 0.05, P(N) = 0.2$ Calcular la longitud promedio del código obtenido.

Ej. 5.- [**LIBRES**] **Preguntas:** [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) Dadas las funciones $T_1(n) = 2\sqrt{n} + 0.5n!$, $T_2(n) = n^2 + 5n^3$, $T_3(n) = \log n + 2n$ y $T_4(n) = n^2 + 3\sqrt{n}$ decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_3 < T_4 < T_1 < T_2$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_3 < T_4 < T_2 < T_1$
- ☐ $T_4 < T_3 < T_2 < T_1$

(b) Sea una lista `L=(1,3,5,4)` simplemente enlazada por punteros o cursores, y sea `p` la posición correspondiente al elemento 3. Despues de hacer `SUPRIME(p,L)`, ¿cual es el resultado de hacer `x = RECUPERA(p,L)`? ...

- ☐ ... retorna el elemento 1
- ☐ ... produce un error
- ☐ ... retorna el elemento 5
- ☐ ... retorna el elemento 3

(c) ¿Cuál es el criterio para elegir una buena función de dispersión? ...

- ☐ Debe tratar de concentrar los elementos en pocas cubetas.
- ☐ Debe tratar de concentrar los elementos en una sóla cubeta.
- ☐ Debe tratar de concentrar los elementos en la primera cubeta.
- ☐ Debe distribuir los elementos en la forma más uniforme posible entre las cubetas.

(d) ¿Cuál de los siguientes algoritmos de clasificación es el más rápido *en el caso promedio*?

- ☐ Burbuja (*"Bubble-sort"*)
- ☐ Clasificación por *incrementos decrecientes* (*shell-sort*)
- ☐ Selección
- ☐ Clasificación rápida (*"Quick-sort"*)

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [8 de Mayo de 2003]

Ej. 1.- Escribir las funciones funciones del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con celdas enlazadas por punteros **o** cursores:

a) `PADRE(n,A)`

b) `HIJO_MAS_IZQ(n,A)`

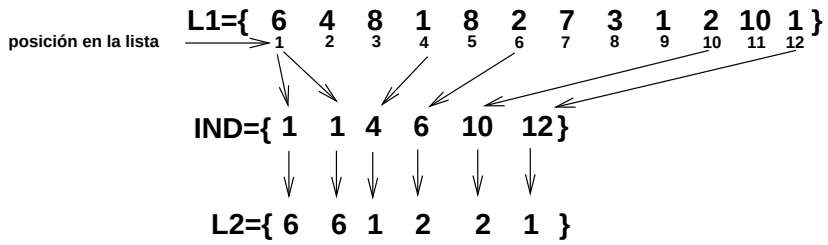
c) `HERMANO_DER(n,A)`

d) `ETIQUETA(n,A)`

e) `CREA2(v, A1, A2)`

f) `ANULA(A)`

Ej. 2.- Escribir un procedimiento `procedure MULTI_REC(L1,IND: lista; var L2: lista);` que, dados una lista `L1` y otra lista `IND` de indices enteros, retorna los elementos de la lista `L1` que están en las posiciones indicadas por los números en `IND`. Se asume que los índices en `IND` son crecientes, es decir $\texttt{IND}_j <= \texttt{IND}_{j+1}$, para todo j. Por ejemplo,



Usar las funciones del **TAD LISTA:** `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

Ej. 3.- Ejercicios básicos sobre TAD's

a) Escribir una función "`ACOTADO(n: nodo; A:arbol; max:integer): boolean;`" que retorna `true` si todas las etiquetas del árbol binario `A` están por debajo de `max` y `false` en caso contrario. Usar las funciones del **TAD ARBOL BINARIO:** `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

b) Escribir un procedimiento `procedure SEPARA(C: cola; var CPAR,CIMPAR: cola);` que separa los elementos de la cola `C` poniéndo los pares en `CPAR` y los impares en `CIMPAR`. Utilizar las funciones del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

Ej. 4.- [LIBRES] Ejercicios operativos:

a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, Z, Q, U, V, W, X, R, T\}$.

- `ORD_POST` $=\{Z, U, X, W, C, Q, T, R, C\}$.

b) **[LIBRES]** Dados los enteros $\{1, 12, 8, 5, 7, 2, 3\}$ ordenarlos por el métodos de *"montículos"*. Mostrar el montículo antes y después de cada inserción.

c) **[LIBRES] Preguntas:** [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

1) Dadas las funciones
   - $T_1(n) = 5\sqrt{n} + 2n^2$,
   - $T_2(n) = n^3 + \log n$,
   - $T_3(n) = \log n + 0.5\, 2^n$ y
   - $T_4(n) = 0.3 \log n + 7$

   decir cuál de los siguientes ordenamientos es el correcto

   ☐ $T_2 < T_1 < T_4 < T_3$
   ☐ $T_4 < T_1 < T_2 < T_3$
   ☐ $T_2 < T_3 < T_4 < T_1$
   ☐ $T_1 < T_3 < T_2 < T_4$

2) ¿Cuál es la ventaja de las listas doblemente enlazadas con respecto a las simplemente enlazadas?

   ☐ `ANTERIOR(p,L)` es $O(n^2)$.
   ☐ `SIGUIENTE(p,L)` es $O(n)$.
   ☐ `ANTERIOR(p,L)` es $O(1)$.
   ☐ `LOCALIZA(x,L)` es $O(1)$.

3) ¿Cuál es el tiempo de ejecución del algoritmo de clasificación por montículos en el peor caso?

   ☐ $O(n \log n)$.
   ☐ $O(n^2)$.
   ☐ $O(1)$.
   ☐ $O(n^3)$.

4) El algoritmo de Huffmann permite obtener códigos binarios para encodar mensajes utilizando árboles binarios. La longitud del código asignado a un caracter, en bits está dado por...

   ☐ ... la profundidad del nodo correspondiente en el árbol.
   ☐ ... la altura del nodo correspondiente en el árbol.
   ☐ ... la etiqueta del nodo correspondiente en el árbol.
   ☐ ... el número de nodos en el subárbol que cuelga del nodo correspondiente.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final y Recuperatorio. [17 de Julio de 2003]

Ej. 1.- [**Primitivas (Recup=10 puntos, Final=20 puntos)**] En todos los casos *escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

(a) [**Alumnos que recuperan 1er parcial**] Escribir las funciones del `TAD COLA` implementadas por **arreglo circular**, a saber: `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

(b) [**FINAL+Alumnos que recuperan 2do parcial**] Escribir las funciones del `TAD ARBOL BINARIO` con celdas enlazadas por **punteros ó cursores** a saber: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v,A1,A2)` y `ANULA(A)`.

(c) [**Alumnos que recuperan 3er parcial**] Escribir las funciones del `TAD CONJUNTO` implementado mediante **vectores de bits** a saber: `ANULA`, `UNION`, `INTERSECCION`, `MIEMBRO`, `MIN`, `INSERTA` y `SUPRIME`.

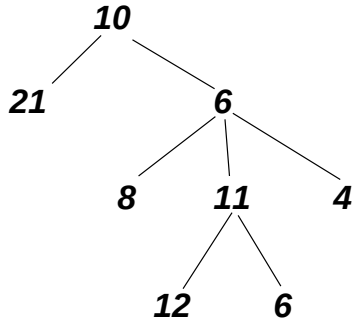
Ej. 2.- [**TODOS: Ejercicios de programación**]

(a) [**Expande (Recup=20 puntos, Final=35 puntos)**] Escribir un procedimiento `procedure EXPANDE(var L:lista);` que inserta una sucesión de números enteros a partir de la posición que esta siendo visitada. La cantidad de números a insertar debe ser igual al número entero que esta en esa posición. La sucesión de enteros debe comenzar con el entero con el cual se va a expandir y debe ser creciente. Por ejemplo, si inicialmente `L=`$\{1,2,3,4\}$, después de hacer `EXPANDE(L)` debe quedar `L=`$\{1,2,2,3,4,3,4,5,6,4,5,6,7,8\}$. Utilizar las primitivas del TAD LISTA: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

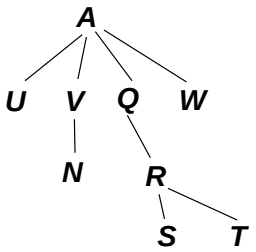
(b) [**Verifica ABB (Recup=30 puntos, Final=45 puntos)**] Escribir una función `VERIFICA_ABB(n: nodo; A: arbol): boolean` que verifica si el subárbol de un nodo `n` verifica la condición de *árbol binario de búsqueda.*

Ej. 3.- [**LIBRES y RECUPERATORIO. Ejercicios operativos**]

(a) [**Crea (Recup=10 puntos, Final=25 puntos)**] Escribir una función `function CREA_ARBOL(A:arbol): nodo` que, usando las funciones del `TAD ARBOL ORDENADO ORIENTADO` `CREA0(v)`, `CREA1(v, A1)`, ... `CREAn(v, A1, A2, ..., An)` crea el siguiente árbol



(b) **[Particionar (Recup=10 puntos, Final=25 puntos)]** Considerando el árbol de la figura, decir cuál son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



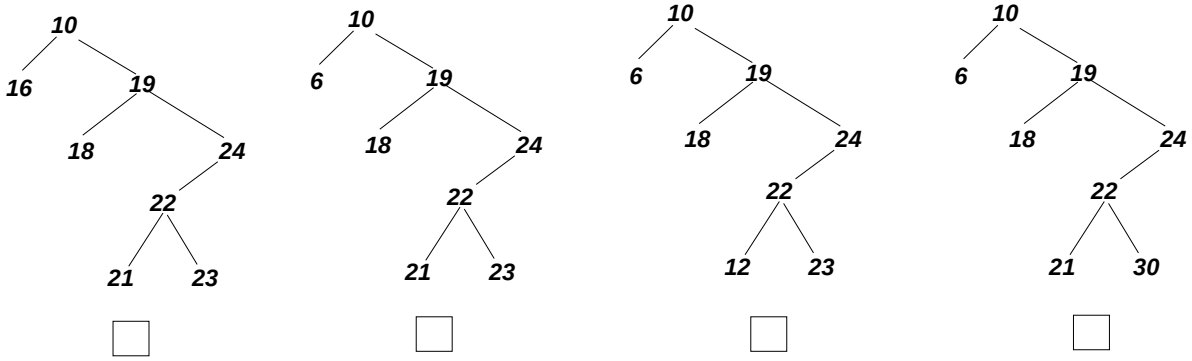
(c) **[Heap-sort (Recup=10 puntos, Final=25 puntos)]** Dados los enteros $\{3, 5, 8, 1, 2, 6, 3, 4, 7\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

Ej. 4.- **[LIBRES(total=25 ptos) y RECUPERATORIO(total=10 ptos)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) ¿Cómo es el tiempo de ejecución para intercalar dos listas clasificadas de n elementos?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(b) ¿Cuál de los siguientes árboles es un árbol binario de búsqueda?



☐ ☐ ☐ ☐

(c) El tiempo de ejecución para el algoritmo de *clasificación rápida* (*quick-sort*) en el peor caso es:

☐ ... $O(n \log n)$ ☐ ... $O(1)$ ☐ ... $O(n^2)$ ☐ ... $O(n)$

(d) El tiempo de ejecución para la función `SUPRIME_MIN` en el `TAD COLA DE PRIORIDAD` implementado por montículos es:

☐ ... $O(n)$ ☐ ... $O(n \log n)$ ☐ ... $O(\log n)$ ☐ ... $O(1)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [7 de Agosto de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD ARBOL ORDENADO ORIENTADO` con celdas enlazadas por **punteros ó cursores** a saber: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERM_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v,A1,A2)` y `ANULA(A)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

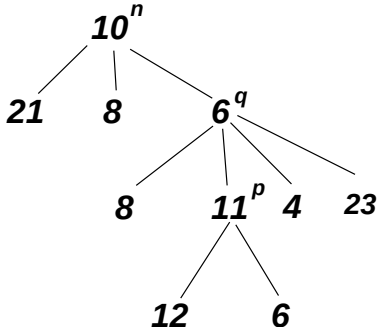
Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

(a) [**Encuentra (35 puntos)**] Escribir una función
`function ENCUENTRA(L1,L2: lista; var INDX:lista): boolean` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `INDX` los índices de los elementos de `L2` que forman `L1`, si no `INDX` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `ENCUENTRA` debe retornar `true`, y `INDX`=$\{1, 2, 5, 6, 8, 10\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `INDX`=$\{\}$. Nota: Los índices en `INDX` deben ser estrictamente crecientes. Utilizar las primitivas del TAD LISTA: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) [**Contiene hijos (35 puntos)**] Escribir una función
`function CONT_HIJOS(L:lista; n:nodo; A:arbol): nodo` que retorna el nodo `m` del subárbol de `n` tal que las etiquetas de sus hijos corresponden exactamente con los enteros en la lista L. Si ningún nodo cumple esta condición entonces debe retornar Λ. Por ejemplo, en el árbol de la figura si `L`=$\{12, 6\}$ entonces `CONT_HIJOS(L,n,A)` debe retornar `p` mientras que si `L`=$\{8, 11, 4, 23\}$ debe retornar `q`. Por otra parte si `L`=$\{21, 8\}$ entonces debe retornar Λ ya que si bien `n` tiene como hijos a $\{21, 8\}$ también tiene al 6, el cual no esta en `L`. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)` y las del `TAD LISTA` `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.



(c) [**Invierte cola (10 puntos)**] Escribir un procedimiento
`procedure INVIERTE(var C:cola);` que invierte los elementos de una cola, usando una pila auxiliar. Por ejemplo, si `C`=$\{12, 4, 23, 10, 1\}$ entonces `INVIERTE(C)` debe dejar `C`=$\{1, 10, 23, 4, 12\}$. Utilizar las funciones del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA(x,C)`,

`QUITA_DE_COLA (C)`, `VACIA(C)`, `FRENTE_DE_COLA(C)` y del `TAD PILA ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 3.- [**LIBRES. Ejercicios operativos (total 80 puntos)**] **Atención!! Alumnos libres deben completar un mínimo de 70% en cada uno de los ítems**

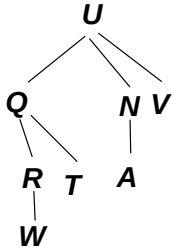
(a) [**Reconstruir árbol (25 puntos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Q, P, L, C, M, D, F, G, H\}$,
- `ORD_POST` $=\{P, C, M, F, G, H, D, L, Q\}$.

(b) [**Árboles de Huffman (20 puntos)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PINGUINO`
$P(P) = 0.1, P(I) = 0.1, P(N) = 0.1, P(G) = 0.1, P(U) = 0.1, P(O) = 0.25, P(Z) = 0.25$
Calcular la longitud promedio del código obtenido.

(c) [**Heap-sort (25 puntos)**] Dados los enteros $\{3, 5, 2, 1, 6, 8, 3, 7\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(d) [**Particionar árbol (10 puntos)**] Considerando el árbol de la figura, decir cuales son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



Ej. 4.- [**LIBRES(20 ptos, 5 por pregunta)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!

(a) ¿Cuál es el tiempo de ejecución para la función `ANTERIOR` para listas *doblemente enlazadas* en el peor caso? (n es el número de elementos en la lista)

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(b) ¿Cuál es el número de niveles en un árbol binario lleno (todos sus niveles están completos) en función del número n de nodos en el árbol?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(c) ¿Cuál es el número de *intercambios* en el método de clasificación por seleccón?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(d) ¿Cuál es el tiempo de ejecución del algoritmo de clasificación por montículos *en el peor caso*?

☐ ... $O(\log n)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(n \log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [7 de Agosto de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD ARBOL ORDENADO ORIENTADO` con celdas enlazadas por **punteros ó cursores** a saber: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERM_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v,A1,A2)` y `ANULA(A)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

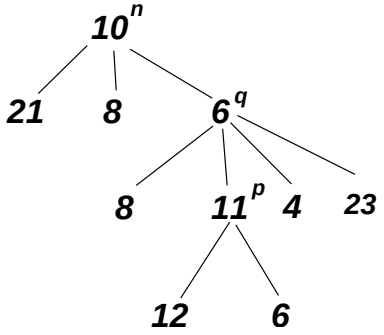
Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

(a) [**Encuentra (35 puntos)**] Escribir una función
`function ENCUENTRA(L1,L2: lista; var INDX:lista): boolean` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `INDX` los índices de los elementos de `L2` que forman `L1`, si no `INDX` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `ENCUENTRA` debe retornar `true`, y `INDX`=$\{1, 2, 5, 6, 8, 10\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `INDX`=$\{\}$. Nota: Los índices en `INDX` deben ser estrictamente crecientes. Utilizar las primitivas del TAD LISTA: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) [**Contiene hijos (35 puntos)**] Escribir una función
`function CONT_HIJOS(L:lista; n:nodo; A:arbol): nodo` que retorna el nodo `m` del subárbol de `n` tal que las etiquetas de sus hijos corresponden exactamente con los enteros en la lista L. Si ningún nodo cumple esta condición entonces debe retornar Λ. Por ejemplo, en el árbol de la figura si `L`=$\{12, 6\}$ entonces `CONT_HIJOS(L,n,A)` debe retornar `p` mientras que si `L`=$\{8, 11, 4, 23\}$ debe retornar `q`. Por otra parte si `L`=$\{21, 8\}$ entonces debe retornar Λ ya que si bien `n` tiene como hijos a $\{21, 8\}$ también tiene al 6, el cual no esta en `L`. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)` y las del `TAD LISTA` `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.



(c) [**Invierte cola (10 puntos)**] Escribir un procedimiento
`procedure INVIERTE(var C:cola);` que invierte los elementos de una cola, usando una pila auxiliar. Por ejemplo, si `C`=$\{12, 4, 23, 10, 1\}$ entonces `INVIERTE(C)` debe dejar `C`=$\{1, 10, 23, 4, 12\}$. Utilizar las funciones del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA(x,C)`,

`QUITA_DE_COLA (C)`, `VACIA(C)`, `FRENTE_DE_COLA(C)` y del `TAD PILA ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 3.- **[LIBRES. Ejercicios operativos (total 80 puntos)] Atención!! Alumnos libres deben completar un mínimo de 70% en cada uno de los ítems**

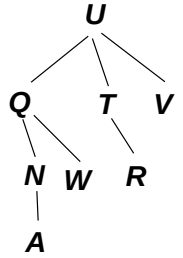
(a) **[Reconstruir árbol (25 puntos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{R, S, M, D, U, L, V, N, H\}$,
- `ORD_POST` $=\{D, L, U, V, M, N, S, H, R\}$.

(b) **[Árboles de Huffman (20 puntos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CATEDRA`
$P(C) = 0.1, P(A) = 0.1, P(T) = 0.1, P(E) = 0.1, P(D) = 0.1, P(R) = 0.25, P(M) = 0.25$
Calcular la longitud promedio del código obtenido.

(c) **[Heap-sort (25 puntos)]** Dados los enteros $\{5, 7, 4, 3, 8, 10, 5, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(d) **[Particionar árbol (10 puntos)]** Considerando el árbol de la figura, decir cuales son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



Ej. 4.- **[LIBRES(20 ptos, 5 por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!

(a) ¿Cuál es el tiempo de ejecución para la función `ANTERIOR` para listas *doblemente enlazadas* en el peor caso? (n es el número de elementos en la lista)

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(b) ¿Cuál es el número de niveles en un árbol binario lleno (todos sus niveles están completos) en función del número n de nodos en el árbol?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(c) ¿Cuál es el número de *intercambios* en el método de clasificación por selecćon?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(d) ¿Cuál es el tiempo de ejecución del algoritmo de clasificación por montículos *en el peor caso*?

☐ ... $O(\log n)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(n \log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [7 de Agosto de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD ARBOL ORDENADO ORIENTADO` con celdas enlazadas por **punteros ó cursores** a saber: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERM_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v,A1,A2)` y `ANULA(A)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

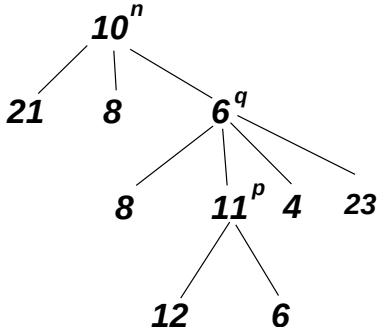
Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

(a) [**Encuentra (35 puntos)**] Escribir una función
`function ENCUENTRA(L1,L2: lista; var INDX:lista): boolean` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `INDX` los índices de los elementos de `L2` que forman `L1`, si no `INDX` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `ENCUENTRA` debe retornar `true`, y `INDX`=$\{1, 2, 5, 6, 8, 10\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `INDX`=$\{\}$. Nota: Los índices en `INDX` deben ser estrictamente crecientes. Utilizar las primitivas del TAD LISTA: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) [**Contiene hijos (35 puntos)**] Escribir una función
`function CONT_HIJOS(L:lista; n:nodo; A:arbol): nodo` que retorna el nodo `m` del subárbol de `n` tal que las etiquetas de sus hijos corresponden exactamente con los enteros en la lista L. Si ningún nodo cumple esta condición entonces debe retornar Λ. Por ejemplo, en el árbol de la figura si `L`=$\{12, 6\}$ entonces `CONT_HIJOS(L,n,A)` debe retornar `p` mientras que si `L`=$\{8, 11, 4, 23\}$ debe retornar `q`. Por otra parte si `L`=$\{21, 8\}$ entonces debe retornar Λ ya que si bien `n` tiene como hijos a $\{21, 8\}$ también tiene al 6, el cual no esta en `L`. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)` y las del `TAD LISTA` `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.



(c) [**Invierte cola (10 puntos)**] Escribir un procedimiento
`procedure INVIERTE(var C:cola);` que invierte los elementos de una cola, usando una pila auxiliar. Por ejemplo, si `C`=$\{12, 4, 23, 10, 1\}$ entonces `INVIERTE(C)` debe dejar `C`=$\{1, 10, 23, 4, 12\}$. Utilizar las funciones del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA(x,C)`,

`QUITA_DE_COLA (C)`, `VACIA(C)`, `FRENTE_DE_COLA(C)` y del `TAD PILA ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 3.- [**LIBRES. Ejercicios operativos (total 80 puntos)**] **Atención!! Alumnos libres deben completar un mínimo de 70% en cada uno de los ítems**

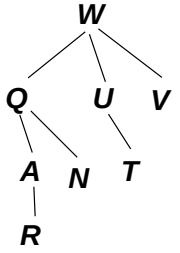
(a) [**Reconstruir árbol (25 puntos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Q, R, Z, T, W, B, D, U\}$,
- `ORD_POST` $=\{R, Q, W, T, Z, D, U, B, A\}$.

(b) [**Árboles de Huffman (20 puntos)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CALAMAR`
$P(C) = 0.1, P(A) = 0.1, P(L) = 0.1, P(R) = 0.1, P(M) = 0.1, P(Q) = 0.25, P(P) = 0.25$
Calcular la longitud promedio del código obtenido.

(c) [**Heap-sort (25 puntos)**] Dados los enteros $\{7, 9, 6, 5, 10, 12, 7, 11\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(d) [**Particionar árbol (10 puntos)**] Considerando el árbol de la figura, decir cuales son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



Ej. 4.- [**LIBRES(20 ptos, 5 por pregunta)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!

(a) ¿Cuál es el tiempo de ejecución para la función `ANTERIOR` para listas *doblemente enlazadas* en el peor caso? (n es el número de elementos en la lista)

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(b) ¿Cuál es el número de niveles en un árbol binario lleno (todos sus niveles están completos) en función del número n de nodos en el árbol?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(c) ¿Cuál es el número de *intercambios* en el método de clasificación por selecćon?

☐ ... $O(1)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(\log n)$

(d) ¿Cuál es el tiempo de ejecución del algoritmo de clasificación por montículos *en el peor caso*?

☐ ... $O(\log n)$ ☐ ... $O(n)$ ☐ ... $O(n^2)$ ☐ ... $O(n \log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [2 de Octubre de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD COLA DE PRIORIDAD` listadas a continuación, implementado por montículos: `ANULA (C)`, `INSERTA (x,C)`, `SUPRIME_MIN (C)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

(a) [**Intercambia secuencia (35 puntos)**] Escribir un procedimiento `procedure INTERCAMBIA_SEC(var L: lista);` que intercambia el grupo de los primeros elementos consecutivos impares por el siguiente grupo de elementos consecutivos pares y así sucesivamente. Por ejemplo si `L={1,2,4,5,6,8,7,9,13,2}` despues de llamar a `INTERCAMBIA_SEC(L)` debe quedar `L={2,4,1,6,8,5,2,7,9,13}` Restricciones:

- Usar **sólo** una estructura auxiliar. Puede ser una cola o una lista (se sugiere una cola).
- El algoritmo debe ser $O(n)$
- Utilizar las primitivas del TAD LISTA: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. y del `TAD COLA: ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA (C)`, y `FRENTE_DE_COLA (C)`.

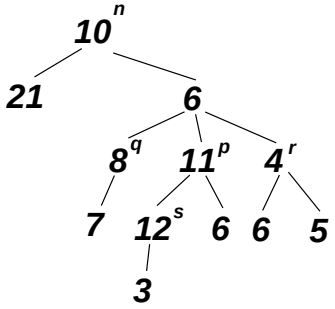
(b) [**Encuentra suma (35 puntos)**] Escribir una función `function ENCUENTRA_SUMA(s:real; n:nodo; A:arbol): nodo` que retorna el nodo `m` del árbol orientado `A` tal que la suma de las etiquetas de todos los nodos descendientes de `m` es `s`. Si no existe un tal nodo, entonces debe retornar Λ. Si hay varios nodos que cumplen la condición debe retornar *el primero en orden previo*. Por ejemplo, para un árbol como el de la figura debe darse

`ENCUENTRA_SUMA(32,n,A)` retorna p

`ENCUENTRA_SUMA(15,n,A)` retorna $\rightarrow q$ (notar que r y s también dan la suma, pero q está primero en orden previo).

`ENCUENTRA_SUMA(15,n,A)` retorna $\rightarrow \Lambda$

El algoritmo debe ser $O(n)$ donde n es el número de nodos en el árbol. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)` y `ETIQUETA(n,A)`.



(c) [**Reordena pila (10 puntos)**] Escribir un procedimiento `procedure REORDENA(var P:pila);` que reordena los elementos de una pila de tal forma que quedan los impares en el fondo y los pares arriba. Los pares e impares deben quedar en el mismo orden relativo entre sí. Por ejemplo si $P = \{1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9\}$ entonces debe quedar $P = \{4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9\}$. Restricciones:

- Se puede usar sólo dos pilas auxiliares.
- El algoritmo debe ser $O(n)$
- Utilizar las funciones del `TAD PILA ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 3.- [**LIBRES. Ejercicios operativos (total 80 puntos)**] **Atención!! Alumnos libres deben completar un mínimo de 70% en cada uno de los ítems**

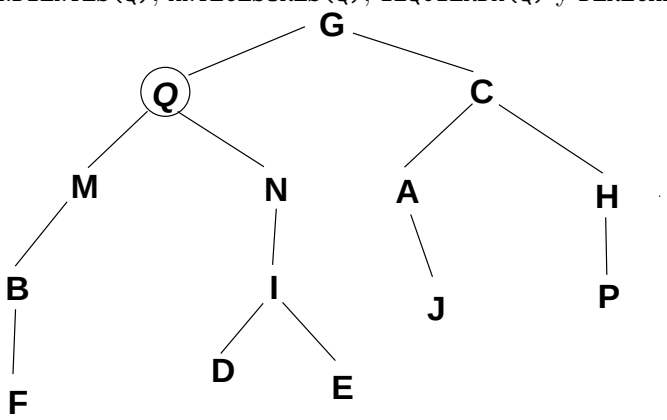
(a) [**Reconstruir árbol (25 puntos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE`=$\{C, A, R, S, H, Q, T, J, U\}$,
- `ORD_POST`=$\{R, H, S, A, J, T, U, Q, C\}$.

(b) [**Crea (20 puntos)**] Escribir una función `function CREA_ARBOL(A:arbol): nodo` que, usando las funciones del `TAD ARBOL ORDENADO ORIENTADO CREA0(v)`, `CREA1(v, A1)`, ... `CREAn(v, A1, A2, ..., An)` crea el árbol A de la figura más arriba.

(c) [**Heap-sort (25 puntos)**] Dados los enteros $\{3, 7, 1, 2, 5, 4, 3, 6\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(d) [**Particionar árbol (10 puntos)**] Considerando el árbol de la figura, decir cuales son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



Ej. 4.- [**LIBRES (20 ptos, 5 por pregunta)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!

(a) ¿Cuál es el tiempo de ejecución para el algoritmo de clasificación por montículos (*heap-sort*) en el peor caso? (n es el número de elementos a clasificar)

☐ ...$O(n \log n)$ ☐ ...$O(1)$ ☐ ...$O(n)$ ☐ ...$O(n^2)$

(b) ¿Cuál es, en promedio, el número de intentos necesarios para insertar un *nuevo* elemento en una tabla de dispersión cerrada? ($0 < \alpha < 1$ es la *"tasa de llenado"* de la tabla, α = (ocupados+suprimidos)/(total de cubetas).

☐ ...$\log \alpha$ ☐ ...α ☐ ...$1/\alpha$ ☐ ...$1/(1-\alpha)$

(c) ¿Cuál es el tiempo de ejecución de la función `ANTERIOR` para listas *simplemente* enlazadas?

☐ ...$O(n^2)$ ☐ ...$O(n)$ ☐ ...$O(\log n)$ ☐ ...$O(1)$

(d) ¿Cuántos elementos, en promedio, hay en cada cubeta en una tabla de dispersión abierta o cerrada con N elementos y B cubetas?

☐ ...$O(N/B)$ ☐ ...$O(N+B)$ ☐ ...$O(B/N)$ ☐ ...$O((N/B)^2)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [2 de Octubre de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD COLA DE PRIORIDAD` listadas a continuación, implementado por montículos: `ANULA (C)`, `INSERTA (x,C)`, `SUPRIME_MIN (C)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

(a) [**Intercambia secuencia (35 puntos)**] Escribir un procedimiento `procedure INTERCAMBIA_SEC(var L: lista);` que intercambia el grupo de los primeros elementos consecutivos impares por el siguiente grupo de elementos consecutivos pares y así sucesivamente. Por ejemplo si `L={1,2,4,5,6,8,7,9,13,2}` despues de llamar a `INTERCAMBIA_SEC(L)` debe quedar `L={2,4,1,6,8,5,2,7,9,13}` Restricciones:

- Usar **sólo** una estructura auxiliar. Puede ser una cola o una lista (se sugiere una cola).
- El algoritmo debe ser $O(n)$
- Utilizar las primitivas del TAD LISTA: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. y del `TAD COLA: ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA (C)`, y `FRENTE_DE_COLA (C)`.

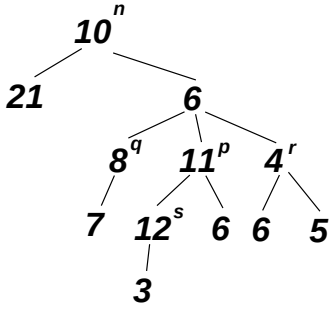
(b) [**Encuentra suma (35 puntos)**] Escribir una función `function ENCUENTRA_SUMA(s:real; n:nodo; A:arbol): nodo` que retorna el nodo `m` del árbol orientado `A` tal que la suma de las etiquetas de todos los nodos descendientes de `m` es `s`. Si no existe un tal nodo, entonces debe retornar Λ. Si hay varios nodos que cumplen la condición debe retornar *el primero en orden previo*. Por ejemplo, para un árbol como el de la figura debe darse

`ENCUENTRA_SUMA(32,n,A)` retorna p

`ENCUENTRA_SUMA(15,n,A)` retorna $\rightarrow q$ (notar que r y s también dan la suma, pero q está primero en orden previo).

`ENCUENTRA_SUMA(15,n,A)` retorna $\rightarrow \Lambda$

El algoritmo debe ser $O(n)$ donde n es el número de nodos en el árbol. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)` y `ETIQUETA(n,A)`.



(c) **[Reordena pila (10 puntos)]** Escribir un procedimiento `procedure REORDENA(var P:pila);` que reordena los elementos de una pila de tal forma que quedan los impares en el fondo y los pares arriba. Los pares e impares deben quedar en el mismo orden relativo entre sí. Por ejemplo si $P = \{1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9\}$ entonces debe quedar $P = \{4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9\}$. Restricciones:

- Se puede usar sólo dos pilas auxiliares.
- El algoritmo debe ser $O(n)$
- Utilizar las funciones del `TAD PILA ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 3.- [LIBRES. Ejercicios operativos (total 80 puntos)] Atención!! Alumnos libres deben completar un mínimo de 70% en cada uno de los ítems

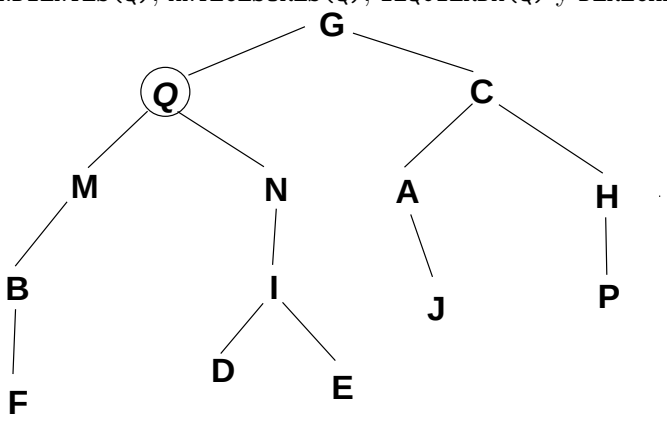
(a) **[Reconstruir árbol (25 puntos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, D, R, T, U, L, V, J, K\}$,
- `ORD_POST` $=\{L, U, K, J, V, T, R, D, C\}$.

(b) **[Crea (20 puntos)]** Escribir una función `function CREA_ARBOL(A:arbol): nodo` que, usando las funciones del `TAD ARBOL ORDENADO ORIENTADO CREA0(v)`, `CREA1(v, A1)`, ... `CREAn(v, A1, A2, ..., An)` crea el árbol A de la figura más arriba.

(c) **[Heap-sort (25 puntos)]** Dados los enteros $\{5, 9, 3, 4, 7, 6, 5, 8\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(d) **[Particionar árbol (10 puntos)]** Considerando el árbol de la figura, decir cuales son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



Ej. 4.- [LIBRES (20 ptos, 5 por pregunta)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!

(a) ¿Cuál es el tiempo de ejecución para el algoritmo de clasificación por montículos (*heap-sort*) en el peor caso? (n es el número de elementos a clasificar)

☐ ...$O(n \log n)$ ☐ ...$O(n)$ ☐ ...$O(1)$ ☐ ...$O(n^2)$

(b) ¿Cuál es, en promedio, el número de intentos necesarios para insertar un *nuevo* elemento en una tabla de dispersión cerrada? ($0 < \alpha < 1$ es la *"tasa de llenado"* de la tabla, α = (ocupados+suprimidos)/(total de cubetas).

☐ ...$\log \alpha$ ☐ ...$1/(1-\alpha)$ ☐ ...$1/\alpha$ ☐ ...α

(c) ¿Cuál es el tiempo de ejecución de la función `ANTERIOR` para listas *simplemente* enlazadas?

☐ ...$O(\log n)$ ☐ ...$O(n^2)$ ☐ ...$O(n)$ ☐ ...$O(1)$

(d) ¿Cuántos elementos, en promedio, hay en cada cubeta en una tabla de dispersión abierta o cerrada con N elementos y B cubetas?

☐ ...$O(B/N)$ ☐ ...$O(N+B)$ ☐ ...$O(N/B)$ ☐ ...$O((N/B)^2)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

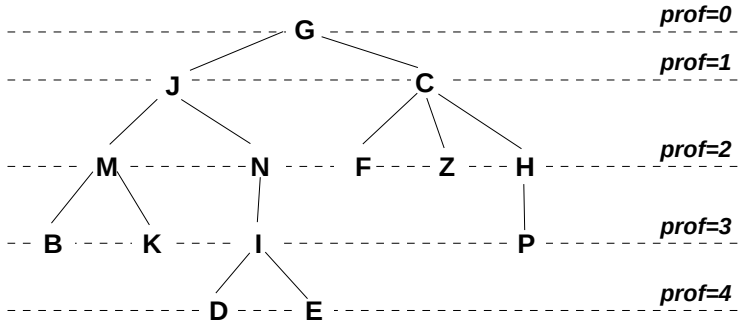
Algoritmos y Estructuras de Datos. Examen Final. [11 de Diciembre de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD PILA` listadas a continuación, implementado por punteros o cursores `ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

a) [**cuenta-prof (35 puntos)**] Escribir un función `function CUENTA_PROF(n:nodo; m:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raíz es `n` que están a profundidad `m` o menor (con respecto a `n`). Por ejemplo, para el árbol de la figura debe retornar

```
CUENTA_PROF(G,2,A) -> 8
CUENTA_PROF(J,1,A) -> 3
CUENTA_PROF(N,3,A) -> 4
```



Usar las primitivas de árbol ordenado orientado siguientes:
`HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. Hacer la función *recursiva*. Notar que, por ejemplo:

```
CUENTA_PROF(G,2,A) = 1 + CUENTA_PROF(J,1,A) + CUENTA_PROF(C,1,A)
```

La recursividad de la función debe cortar cuando `n`$= \Lambda$ o $m < 0$. El tiempo de ejecución del algoritmo debe ser $O(n)$, donde n es el número de nodos en el árbol.

b) [**semejante (10 puntos)**] Dos árboles ordenados T y T' se dicen *"semejantes"* si tienen la misma estructura. Formalmente esto significa que

- ambos son vacíos, **ó**
- ambos son no vacíos y los subárboles de sus hijos correspondientes son semejantes.

Informalmente decimos que T y T' tienen ambos la misma *"forma"*. **Consigna:** Escriba una función

```
function SEMEJANTE(T,T': nodo) : boolean;
```

que retorna `true` si los árboles son semejantes y `false` caso contrario. Usar las primitivas de árbol ordenado orientado siguientes:

1) `PADRE(n,A)`

2) `HIJO_MAS_IZQ(n,A)`
3) `HERMANO_DER(n,A)`

c) [**junta2 (35 puntos)**] Escribir un procedimiento `procedure JUNTA2(var P: pila);` que modifica la pila `P` sumando de a dos los elementos en la pila. Si el número de elementos es impar, entonces el último queda inalterado. Por ejemplo, si `P={tope->1,3,5,2,4,1,2,3,9}` entonces después de hacer `JUNTA2(P)` debe quedar `P={tope->4,7,5,5,9}`. Utilizar las funciones del `TAD PILA ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

Ej. 3.- [**LIBRES. Ejercicios operativos (total 80 puntos)**] **Atención!! Alumnos libres deben completar un mínimo de 70 % en cada uno de los ítems**

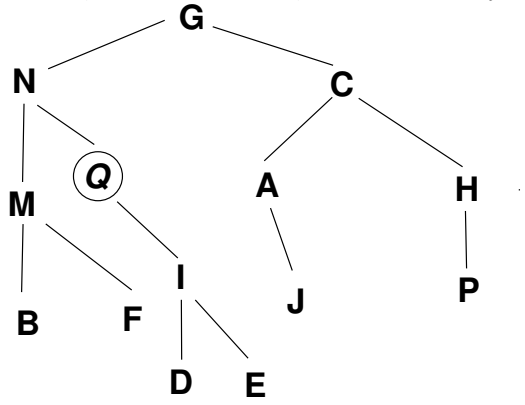
a) [**reconstruir-arbol (20 puntos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE`=$\{Z, A, B, J, D, E, K, L, M\}$,
- `ORD_POST`=$\{A, D, K, M, L, E, J, B, Z\}$.

b) [**abb (20 ptos)**] Dados los enteros $\{1, 5, 8, 3, 2, 10, 20\}$ insertarlos, en ese orden, en un `TAD ARBOL BINARIO DE BUSQUEDA`. Mostrar las operaciones necesarias para eliminar los elementos 5, 6 y 3, en ese orden.

c) [**huffman (20 puntos)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario según el algortimo de *Huffman* y encodar la palabra `TRANSVERSAL` $P(A) = P(N) = P(S) = P(L) = 0{,}2, P(R) = P(T) = P(V) = P(E) = 0{,}05$. Calcular la longitud promedio del código obtenido.

d) [**particionar-arbol (10 puntos)**] Considerando el árbol de la figura, decir cuales son los nodos `DESCENDIENTES(Q)`, `ANTECESORES(Q)`, `IZQUIERDA(Q)` y `DERECHA(Q)`.



Ej. 4.- [**LIBRES (20 ptos, 5 por pregunta)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!

a) Dadas las funciones

☐ $...T_1(n) = \sqrt{n} + 0{,}5\, n^{1,7} + 2{,}3\, \log_2 n$
☐ $...T_2(n) = 3\, 2^n + 5\, n^2 + 6\, \sqrt{n}$
☐ $...T_3(n) = 5n! + 2\, 5^n + 4\, n^5$
☐ $...T_4(n) = 3\sqrt{n} + 2\, n^2 + 4\, n^3$

decir cuál de los iguientes ordenamientos es el correcto

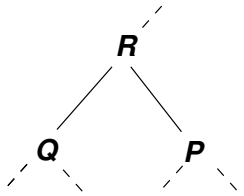
- ☐ ...$T_2 < T_1 < T_4 < T_3$
- ☐ ...$T_4 < T_1 < T_2 < T_3$
- ☐ ...$T_1 < T_4 < T_2 < T_3$
- ☐ ...$T_3 < T_4 < T_1 < T_2$

b) El mejor caso para el algoritmo de clasificación rápida (*"quick-sort"*) es cuando el pivote v particiona la secuencia en dos secuencias de longitud n_1 y n_2 tales que

- ☐ ..... $n_1 \approx n_2$
- ☐ ..... $n_1 \gg n_2$
- ☐ ..... $n_1 \ll n_2$
- ☐ ..... $n_1 \approx \sqrt{n_2}$

c) El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si R, P, Q son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ ...$Q + P \leq R$.
- ☐ ...$R \leq Q, P$.
- ☐ ...$Q \leq R \leq P$.
- ☐ ...$P \leq R \leq Q$.



d) ¿Cuál es el tiempo de ejecución en promedio para inserción de un elemento una tabla de dispersión abierta con N elementos y B cubetas?

- ☐ ...$O(N/B)$
- ☐ ...$O(N)$
- ☐ ...$O(B)$
- ☐ ...$O(NB)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

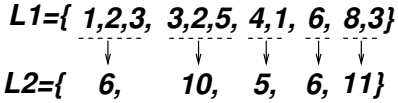
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [30 de Diciembre de 2003]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

a) [**check-sum (35 puntos)**] Dadas dos listas de enteros `L1` y `L2` ecribir una función `function CHECK_SUM(L1,L2: lista): boolean;` que retorna verdadero si los elementos de `L1` pueden agruparse sumando de manera de obtener los elementos de `L2` *sin alterar el orden de los elementos*. Por ejemplo, en el caso de la figura `CHECK_SUM(L1,L2)` debe retornar verdadero ya que los agrupamientos mostrados reducen la lista `L1` a la `L2`.

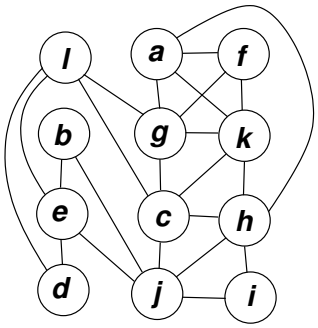


b) [**saca-par (20 puntos)**] Escribir un procedimiento `procedure SACA_PAR(var L:lista; C:cola);` que apendiza a la lista `L` todos los elementos de `C` que son pares, los cuales a su vez deben ser removidos de `C`. Por ejemplo, si inicialmente `L`$\{2,3,4\}$ y `C`$\{1,6,3,5,2,8\}$ entonces, después de hacer `SACA_PAR(L,C)` debe quedar `L`$\{2,3,4,6,2,8\}$ y `C`$=\{1,3,5\}$. No usar ninguna estructura auxiliar más que una cola. Usar las funciones del `TAD LISTA: INSERTA(x,p,L), RECUPERA(p,L), SUPRIME(p,L), SIGUIENTE(p,L), ANULA(L), PRIMERO(L), FIN(L)` y del `TAD COLA: ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

c) [**suma-par (25 puntos)**] Escribir una función `function SUMA_PAR(n: nodo; A: arbol) : integer` que retorna la suma de las etiquetas pares de un árbol binario. Usar las funciones del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)}`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 3.- [**LIBRES**] **Ejercicios operativos (total 80 puntos)**

a) [**colorear-grafo (20 puntos)**] Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



b) [**reconstruir-arbol (20 puntos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{P,Q,R,T,U,S,V,W,Z\}$.

- `ORD_POST` $=\{T, U, R, V, Z, W, S, Q, P\}$.

c) **[abb (20 puntos)]** Dados los enteros $\{12, 11, 14, 8, 16, 13, 9, 7, 15, 21, 4\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 12, 9 y 13.

d) **[heap-sort (20 puntos)]** Dados los enteros $\{3, 7, 1, 2, 5, 4, 3, 6\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

Ej. 4.- [LIBRES] preguntas (total 20ptos, 5ptos/preg): [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Dadas las funciones

- $T_1(n) = 5n + \log n$,
- $T_2(n) = 4n^2 + \sqrt{n}$,
- $T_3(n) = 2^n + n!$ y
- $T_4(n) = \sqrt{(n)} + \log n$

decir cuál de los siguientes ordenamientos es el correcto

☐ $T_3 < T_4 < T_1 < T_2$
☐ $T_4 < T_1 < T_2 < T_3$
☐ $T_2 < T_1 < T_4 < T_3$
☐ $T_4 < T_3 < T_2 < T_1$

b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"quicksort"*) es $O(n \log(n)$ (n es el número de elementos a ordenar) ...

☐ ... siempre.
☐ ... cuando el vector ya está ordenado.
☐ ... nunca.
☐ ... en el caso promedio.

c) Una ventaja del método de clasificación por selección, en comparación con otros algoritmos *lentos*, es que realiza sólo n *intercambios*...

☐ ... a veces.
☐ ... cuando el vector está ordenado.
☐ ... siempre.
☐ ... cuando el vector está desordenado.

d) ¿Cuál es el tiempo de ejecución del procedimiento de clasificación por *incrementos decrecientes* (*shell-sort*) en el caso promedio?

☐ $O(n^{1,3})$
☐ $O(n^{1,5})$
☐ $O(\log n)$
☐ $O(n!)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: **1a.** [22 de Abril de 2003]

[Ej. 1] **[Tiempos de ejecución (10 puntos)]** Dadas las funciones

- $T_1(n) = 0.3 \log_2(n) + 3n^2$
- $T_2(n) = n! + 2^n$
- $T_3(n) = 3^n + n^2$
- $T_4(n) = 0.5\sqrt{n} + 2n^{0.9}$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_3 < T_4 < T_2 < T_1$
- ☐ $T_3 < T_4 < T_1 < T_2$
- ☐ $T_2 < T_3 < T_4 < T_1$
- ☐ $T_4 < T_1 < T_3 < T_2$

[Ej. 2] **[Primitivas (15 puntos)]** Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por *cursores*. Es decir, implementar en Pascal los siguientes procedimientos/funciones: `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. [*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

[Ej. 3] **[Programación (total = 45 puntos)]** Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$.

(a) **[35 puntos]** Escribir una función "`function MAX_DEV_M(L:lista; m:integer) : integer;`" que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

$$\begin{aligned}\text{max_dev_m(L)} = \ & \max\{\text{max_dev}(a_1, a_2, \ldots, a_m), \\ & \text{max_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max_dev}(a_{n-m+1}, \ldots, a_n)\} \quad (1)\end{aligned}$$

Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces `MAX_DEV_N(L,3)` debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones. Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) **[5 puntos]** Cual es el tiempo de ejecución, en el peor caso, si `m=2`, como función de `n`,

(c) **[5 puntos]** Cual es el tiempo de ejecución en el peor caso, si `m=n/2` (asumimos que `n` es par), como función de `n`.

[Ej. 4] **[Programación básica de pilas y colas (total = 20 puntos)]** Escribir los siguientes procedimientos/funciones

(a) **[10 puntos]** Escribir una función "`function MAXPILA(C:pila) : integer`" que retorna el máximo de los elementos de una pila usando una pila auxiliar. Finalmente la pila debe quedar en el mismo estado que originalmente. Usar las primitivas del `TAD PILA: ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

(b) **[10 puntos]** Escribir un procedimiento "`procedure X2(var C: cola);`" que invierte los elementos de una cola de a 2, es decir el primero con el 2do, el 3ro con el 4to, etc... usando una cola auxiliar. Si la cola tiene un número impar de elementos entonces el último queda inalterado. Por ejemplo, si `C=(frente=2,4,3,2,5,6,7)` entonces después de `X2(C);` debe quedar `C=(frente=4,2,2,3,6,5,7)`.

[Ej. 5] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución de la función `ANTERIOR` para listas *simplemente* enlazadas es ... (n es el número de elementos en la lista)

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$

(b) El tiempo de ejecución de la función `PONE_EN_COLA` para colas implementadas por arreglos circulares es ... (n es el número de elementos en la cola)

- ☐ ... $O(n^2)$
- ☐ ... $O(n)$
- ☐ ... $O(\log n)$
- ☐ ... $O(1)$

(c) La desventaja de la implementación del `TAD CORRESPONDENCIA` por arreglos es que ... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... el tamaño de la correspondencia es $O(n)$.
- ☐ ... el tamaño de la correspondencia es $O(N_d)$.
- ☐ ... el tiempo de ejecución de `CALCULA(M,d,r)` es $O(N_d)$.
- ☐ ... el tiempo de ejecución de `CALCULA(M,d,r)` es $O(n)$.

(d) El tiempo de ejecución de la función `CALCULA(M,d,r)` para el `TAD CORRESPONDENCIA` implementado por listas, en el peor caso es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(N_d)$
- ☐ ... $O(n^2)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: **1b.** [22 de Abril de 2003]

[Ej. 1] **[Tiempos de ejecución (10 puntos)]** Dadas las funciones

- $T_1(n) = 0.3 \log_2(n) + 3n^2$
- $T_2(n) = n! + 2^n$
- $T_3(n) = 3^n + n^2$
- $T_4(n) = 0.5\sqrt{n} + 2n^{0.9}$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_2 < T_3 < T_4 < T_1$
- ☐ $T_3 < T_4 < T_1 < T_2$
- ☐ $T_3 < T_4 < T_2 < T_1$
- ☐ $T_4 < T_1 < T_3 < T_2$

[Ej. 2] **[Primitivas (15 puntos)]** Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por *cursores*. Es decir, implementar en Pascal los siguientes procedimientos/funciones: `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. [*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

[Ej. 3] **[Programación (total = 45 puntos)]** Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$.

(a) **[35 puntos]** Escribir una función "`function MAX_DEV_M(L:lista; m:integer) : integer;`" que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

$$\begin{aligned}\text{max_dev_m(L)} = \ & \max\{\text{max_dev}(a_1, a_2, \ldots, a_m), \\ & \text{max_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max_dev}(a_{n-m+1}, \ldots, a_n)\} \quad (1)\end{aligned}$$

Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces `MAX_DEV_N(L,3)` debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones. Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) **[5 puntos]** Cual es el tiempo de ejecución, en el peor caso, si `m=2`, como función de `n`,

(c) **[5 puntos]** Cual es el tiempo de ejecución en el peor caso, si `m=n/2` (asumimos que `n` es par), como función de `n`.

[Ej. 4] **[Programación básica de pilas y colas (total = 20 puntos)]** Escribir los siguientes procedimientos/funciones

(a) **[10 puntos]** Escribir una función "`function MAXPILA(C:pila) : integer`" que retorna el máximo de los elementos de una pila usando una pila auxiliar. Finalmente la pila debe quedar en el mismo estado que originalmente. Usar las primitivas del `TAD PILA: ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

(b) **[10 puntos]** Escribir un procedimiento "`procedure X2(var C: cola);`" que invierte los elementos de una cola de a 2, es decir el primero con el 2do, el 3ro con el 4to, etc... usando una cola auxiliar. Si la cola tiene un número impar de elementos entonces el último queda inalterado. Por ejemplo, si `C=(frente=2,4,3,2,5,6,7)` entonces después de `X2(C);` debe quedar `C=(frente=4,2,2,3,6,5,7)`.

[Ej. 5] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución de la función `ANTERIOR` para listas *simplemente* enlazadas es ... (n es el número de elementos en la lista)

- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(n)$

(b) El tiempo de ejecución de la función `PONE_EN_COLA` para colas implementadas por arreglos circulares es ... (n es el número de elementos en la cola)

- ☐ ... $O(n)$
- ☐ ... $O(\log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$

(c) La desventaja de la implementación del `TAD CORRESPONDENCIA` por arreglos es que ... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... el tamaño de la correspondencia es $O(N_d)$.
- ☐ ... el tiempo de ejecución de `CALCULA(M,d,r)` es $O(n)$.
- ☐ ... el tamaño de la correspondencia es $O(n)$.
- ☐ ... el tiempo de ejecución de `CALCULA(M,d,r)` es $O(N_d)$.

(d) El tiempo de ejecución de la función `CALCULA(M,d,r)` para el `TAD CORRESPONDENCIA` implementado por listas, en el peor caso es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(n^2)$
- ☐ ... $O(n)$
- ☐ ... $O(N_d)$
- ☐ ... $O(1)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: **1c.** [22 de Abril de 2003]

[Ej. 1] **[Tiempos de ejecución (10 puntos)]** Dadas las funciones

- $T_1(n) = 0.3 \log_2(n) + 3n^2$
- $T_2(n) = n! + 2^n$
- $T_3(n) = 3^n + n^2$
- $T_4(n) = 0.5\sqrt{n} + 2n^{0.9}$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_3 < T_4 < T_2 < T_1$
- ☐ $T_2 < T_3 < T_4 < T_1$
- ☐ $T_4 < T_1 < T_3 < T_2$
- ☐ $T_3 < T_4 < T_1 < T_2$

[Ej. 2] **[Primitivas (15 puntos)]** Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por *cursores*. Es decir, implementar en Pascal los siguientes procedimientos/funciones: `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. [*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

[Ej. 3] **[Programación (total = 45 puntos)]** Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$.

(a) **[35 puntos]** Escribir una función "`function MAX_DEV_M(L:lista; m:integer) : integer;`" que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

$$\begin{aligned}\text{max_dev_m(L)} = \ & \max\{\text{max_dev}(a_1, a_2, \ldots, a_m), \\ & \text{max_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max_dev}(a_{n-m+1}, \ldots, a_n)\} \quad (1)\end{aligned}$$

Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces `MAX_DEV_N(L,3)` debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones. Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) **[5 puntos]** Cual es el tiempo de ejecución, en el peor caso, si `m=2`, como función de `n`,

(c) **[5 puntos]** Cual es el tiempo de ejecución en el peor caso, si `m=n/2` (asumimos que `n` es par), como función de `n`.

[Ej. 4] **[Programación básica de pilas y colas (total = 20 puntos)]** Escribir los siguientes procedimientos/funciones

(a) **[10 puntos]** Escribir una función "`function MAXPILA(C:pila) : integer`" que retorna el máximo de los elementos de una pila usando una pila auxiliar. Finalmente la pila debe quedar en el mismo estado que originalmente. Usar las primitivas del `TAD PILA: ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

(b) **[10 puntos]** Escribir un procedimiento "`procedure X2(var C: cola);`" que invierte los elementos de una cola de a 2, es decir el primero con el 2do, el 3ro con el 4to, etc... usando una cola auxiliar. Si la cola tiene un número impar de elementos entonces el último queda inalterado. Por ejemplo, si `C=(frente=2,4,3,2,5,6,7)` entonces después de `X2(C);` debe quedar `C=(frente=4,2,2,3,6,5,7)`.

[Ej. 5] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución de la función `ANTERIOR` para listas *simplemente* enlazadas es ... (n es el número de elementos en la lista)

- ☐ ... $O(n^2)$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n)$

(b) El tiempo de ejecución de la función `PONE_EN_COLA` para colas implementadas por arreglos circulares es ... (n es el número de elementos en la cola)

- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(n)$

(c) La desventaja de la implementación del `TAD CORRESPONDENCIA` por arreglos es que ... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... el tamaño de la correspondencia es $O(N_d)$.
- ☐ ... el tamaño de la correspondencia es $O(n)$.
- ☐ ... el tiempo de ejecución de `CALCULA(M,d,r)` es $O(N_d)$.
- ☐ ... el tiempo de ejecución de `CALCULA(M,d,r)` es $O(n)$.

(d) El tiempo de ejecución de la función `CALCULA(M,d,r)` para el `TAD CORRESPONDENCIA` implementado por listas, en el peor caso es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(n)$
- ☐ ... $O(N_d)$
- ☐ ... $O(n^2)$
- ☐ ... $O(1)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 1er Parcial. Tema: **2a.** [22 de Abril de 2003]

[Ej. 1] **[Tiempos de ejecución (10 puntos)]** Dadas las funciones

- $T_1(n) = 2^n + n^2$
- $T_2(n) = 3^n + n^3$
- $T_3(n) = \sqrt{n} + \log n$
- $T_4(n) = \sqrt{n} + n!$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_3 < T_1 < T_2 < T_4$
- ☐ $T_4 < T_1 < T_3 < T_2$
- ☐ $T_1 < T_4 < T_2 < T_3$
- ☐ $T_2 < T_3 < T_4 < T_1$

[Ej. 2] **[Primitivas (15 puntos)]** Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por *cursores*. Es decir, implementar en Pascal los siguientes procedimientos/funciones: `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. [*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

[Ej. 3] **[Programación (total = 45 puntos)]** Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$.

(a) **[35 puntos]** Escribir un procedimiento "`procedure SUAVIZA_M(var L:lista; m,maxdif:integer) : integer;`" que elimina la mínima cantidad de elementos de `L` de tal manera que la máxima desviación de una subsecuencia de `m` elementos consecutivos es `maxdif`. Por ejemplo, si `L=(1,3,5,4,2,3,7,4)` entonces `SUAVIZA_M(L,3,3)` debe retornar `L=(1,3,4,2,3,4)`, habiéndose eliminado los elementos 5 y 7. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista recorrer los `m-1` elementos siguientes a `p`, removiendo aquellos elementos que tienen una diferencia con el elemento `p` mayor a `maxdif`. Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) **[5 puntos]** Cual es el tiempo de ejecución, en el peor caso, si `m=2`, como función de `n`,

(c) **[5 puntos]** Cual es el tiempo de ejecución en el peor caso, si `m=n/2` (asumimos que `n` es par), como función de `n`.

[Ej. 4] **[Programación básica de pilas y colas (total = 20 puntos)]** Escribir los siguientes procedimientos/funciones

(a) **[10 puntos]** Escribir un procedimiento "`procedure DEJAPAR(var P:pila)`" que elimina de la pila `P` todos los elementos impares usando una pila auxiliar. Los elementos deben quedar en el mismo orden en el que estaban. Por ejemplo, si `P=(tope=6,1,2,4,3,5,6)` entonces después de `DEJAPAR(P)`, debe quedar `P=(tope=6,2,4,6)`. Usar las primitivas del `TAD PILA: ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

(b) **[10 puntos]** Escribir una función "`function MAXCOLA(C:pila) : integer`" que retorna el máximo de los elementos de una cola usando una cola auxiliar. Finalmente la cola debe quedar en el mismo estado que originalmente. Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

[Ej. 5] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución de la función `LOCALIZA` para listas *simplemente* enlazadas es $O(1)$... (n es el número de elementos en la lista)

- ☐ ... en el peor caso.
- ☐ ... cuando el elemento no está en la lista.
- ☐ ... siempre.
- ☐ ... en el mejor caso.

(b) El tiempo de ejecución de la función `CALCULA(M,d,r)` para el `TAD CORRESPONDENCIA` implementado por arreglos es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(N_d)$
- ☐ ... $O(n^2)$

(c) El requerimiento de memoria para el `TAD CORRESPONDENCIA` implementado por listas es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(1)$
- ☐ ... $O(N_d^2)$
- ☐ ... $O(N_d)$
- ☐ ... $O(n)$

(d) ¿Cuál de las siguientes expresiones booleanas indica si una lista `L` esta vacía?

- ☐ ... `PRIMERO(L) = FIN(L)`
- ☐ ... `PRIMERO(L) <> FIN(L)`
- ☐ ... `SIGUIENTE(PRIMERO(L),L) = FIN(L)`
- ☐ ... `PRIMERO(L)+1 = FIN(L)`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 1er Parcial. Tema: **2b.** [22 de Abril de 2003]

[Ej. 1] **[Tiempos de ejecución (10 puntos)]** Dadas las funciones

- $T_1(n) = 2^n + n^2$
- $T_2(n) = 3^n + n^3$
- $T_3(n) = \sqrt{n} + \log n$
- $T_4(n) = \sqrt{n} + n!$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_1 < T_4 < T_2 < T_3$
- ☐ $T_2 < T_3 < T_4 < T_1$
- ☐ $T_4 < T_1 < T_3 < T_2$
- ☐ $T_3 < T_1 < T_2 < T_4$

[Ej. 2] **[Primitivas (15 puntos)]** Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por *cursores*. Es decir, implementar en Pascal los siguientes procedimientos/funciones: `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. [*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

[Ej. 3] **[Programación (total = 45 puntos)]** Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$.

(a) **[35 puntos]** Escribir un procedimiento "`procedure SUAVIZA_M(var L:lista; m,maxdif:integer) : integer;`" que elimina la mínima cantidad de elementos de `L` de tal manera que la máxima desviación de una subsecuencia de `m` elementos consecutivos es `maxdif`. Por ejemplo, si `L=(1,3,5,4,2,3,7,4)` entonces `SUAVIZA_M(L,3,3)` debe retornar `L=(1,3,4,2,3,4)`, habiéndose eliminado los elementos 5 y 7. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista recorrer los `m-1` elementos siguientes a `p`, removiendo aquellos elementos que tienen una diferencia con el elemento `p` mayor a `maxdif`. Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) **[5 puntos]** Cual es el tiempo de ejecución, en el peor caso, si `m=2`, como función de `n`,

(c) **[5 puntos]** Cual es el tiempo de ejecución en el peor caso, si `m=n/2` (asumimos que `n` es par), como función de `n`.

[Ej. 4] **[Programación básica de pilas y colas (total = 20 puntos)]** Escribir los siguientes procedimientos/funciones

(a) **[10 puntos]** Escribir un procedimiento "`procedure DEJAPAR(var P:pila)`" que elimina de la pila `P` todos los elementos impares usando una pila auxiliar. Los elementos deben quedar en el mismo orden en el que estaban. Por ejemplo, si `P=(tope=6,1,2,4,3,5,6)` entonces después de `DEJAPAR(P)`, debe quedar `P=(tope=6,2,4,6)`. Usar las primitivas del `TAD PILA: ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

(b) **[10 puntos]** Escribir una función "`function MAXCOLA(C:pila) : integer`" que retorna el máximo de los elementos de una cola usando una cola auxiliar. Finalmente la cola debe quedar en el mismo estado que originalmente. Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

[Ej. 5] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución de la función `LOCALIZA` para listas *simplemente* enlazadas es $O(1)$... (n es el número de elementos en la lista)

- ☐ ... en el peor caso.
- ☐ ... en el mejor caso.
- ☐ ... cuando el elemento no está en la lista.
- ☐ ... siempre.

(b) El tiempo de ejecución de la función `CALCULA(M,d,r)` para el `TAD CORRESPONDENCIA` implementado por arreglos es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(n^2)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(N_d)$

(c) El requerimiento de memoria para el `TAD CORRESPONDENCIA` implementado por listas es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(N_d^2)$
- ☐ ... $O(N_d)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$

(d) ¿Cuál de las siguientes expresiones booleanas indica si una lista `L` esta vacía?

- ☐ ... `PRIMERO(L) <> FIN(L)`
- ☐ ... `PRIMERO(L) = FIN(L)`
- ☐ ... `PRIMERO(L)+1 = FIN(L)`
- ☐ ... `SIGUIENTE(PRIMERO(L),L) = FIN(L)`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 1er Parcial. Tema: **2c.** [22 de Abril de 2003]

[Ej. 1] **[Tiempos de ejecución (10 puntos)]** Dadas las funciones

- $T_1(n) = 2^n + n^2$
- $T_2(n) = 3^n + n^3$
- $T_3(n) = \sqrt{n} + \log n$
- $T_4(n) = \sqrt{n} + n!$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_4 < T_1 < T_3 < T_2$
- ☐ $T_1 < T_4 < T_2 < T_3$
- ☐ $T_2 < T_3 < T_4 < T_1$
- ☐ $T_3 < T_1 < T_2 < T_4$

[Ej. 2] **[Primitivas (15 puntos)]** Escribir las funciones primitivas del TAD Lista con celdas simplemente enlazadas por *cursores*. Es decir, implementar en Pascal los siguientes procedimientos/funciones: `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. [*Nota: Se recomienda utilizar celda de encabezamiento. Puede usarse puntero a la última celda o no.*]

[Ej. 3] **[Programación (total = 45 puntos)]** Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$.

(a) **[35 puntos]** Escribir un procedimiento "`procedure SUAVIZA_M(var L:lista; m,maxdif:integer) : integer;`" que elimina la mínima cantidad de elementos de `L` de tal manera que la máxima desviación de una subsecuencia de `m` elementos consecutivos es `maxdif`. Por ejemplo, si `L=(1,3,5,4,2,3,7,4)` entonces `SUAVIZA_M(L,3,3)` debe retornar `L=(1,3,4,2,3,4)`, habiéndose eliminado los elementos 5 y 7. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista recorrer los `m-1` elementos siguientes a `p`, removiendo aquellos elementos que tienen una diferencia con el elemento `p` mayor a `maxdif`. Utilizar las primitivas del **TAD LISTA**: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

(b) **[5 puntos]** Cual es el tiempo de ejecución, en el peor caso, si `m=2`, como función de `n`,

(c) **[5 puntos]** Cual es el tiempo de ejecución en el peor caso, si `m=n/2` (asumimos que `n` es par), como función de `n`.

[Ej. 4] **[Programación básica de pilas y colas (total = 20 puntos)]** Escribir los siguientes procedimientos/funciones

(a) **[10 puntos]** Escribir un procedimiento "`procedure DEJAPAR(var P:pila)`" que elimina de la pila `P` todos los elementos impares usando una pila auxiliar. Los elementos deben quedar en el mismo orden en el que estaban. Por ejemplo, si `P=(tope=6,1,2,4,3,5,6)` entonces después de `DEJAPAR(P)`, debe quedar `P=(tope=6,2,4,6)`. Usar las primitivas del `TAD PILA: ANULA(P)`, `METE(x,P)`, `SACA(P)`, `TOPE(P)` y `VACIA(P)`.

(b) **[10 puntos]** Escribir una función "`function MAXCOLA(C:pila) : integer`" que retorna el máximo de los elementos de una cola usando una cola auxiliar. Finalmente la cola debe quedar en el mismo estado que originalmente. Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

[Ej. 5] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

(a) El tiempo de ejecución de la función `LOCALIZA` para listas *simplemente* enlazadas es $O(1)$... (n es el número de elementos en la lista)

- ☐ ... siempre.
- ☐ ... en el mejor caso.
- ☐ ... en el peor caso.
- ☐ ... cuando el elemento no está en la lista.

(b) El tiempo de ejecución de la función `CALCULA(M,d,r)` para el `TAD CORRESPONDENCIA` implementado por arreglos es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(N_d)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$

(c) El requerimiento de memoria para el `TAD CORRESPONDENCIA` implementado por listas es... (N_d es el número de elementos en el dominio, n es el número de elementos del dominio que tienen valores asignados).

- ☐ ... $O(n)$
- ☐ ... $O(N_d^2)$
- ☐ ... $O(1)$
- ☐ ... $O(N_d)$

(d) ¿Cuál de las siguientes expresiones booleanas indica si una lista `L` esta vacía?

- ☐ ... `PRIMERO(L)+1 = FIN(L)`
- ☐ ... `SIGUIENTE(PRIMERO(L),L) = FIN(L)`
- ☐ ... `PRIMERO(L) = FIN(L)`
- ☐ ... `PRIMERO(L) <> FIN(L)`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: **1a.** [3 de junio de 2003]

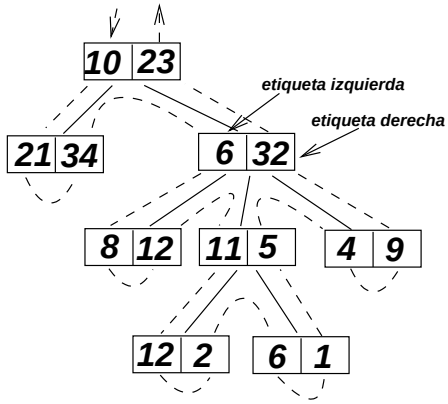
[Ej. 1] **[Primitivas (10 puntos)]** Escribir las funciones del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[Bilistado AOO (30 puntos)]**
El *"bi-listado"* de un árbol ordenado orientado puede pensarse como una combinación de los listados previo y posterior. Asumamos que las etiquetas tienen dos partes, una etiqueta *"derecha"* y una *"izquierda"*,

```
type tipo_etiqueta = record
    derecha, izquierda: integer
end;
```

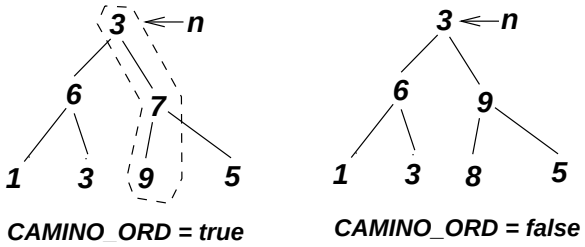
entonces el bi-listado de un nodo n se define como la lista vacía si el nodo es Λ y, si no, recursivamente como la etiqueta izquierda de n, seguida del bi-listado de los hijos, seguido de la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el bi-listado sigue la línea de puntos y resulta en la siguiente lista:
`BI-LISTADO`$=\{10, 21, 34, 6, 8, 12, 11, 12, 2, 6, 1, 5, 4, 9, 32, 23\}$. Notar que si consideramos sólo las etiquetas izuierdas, entonces el resultado es el *orden previo*, mientras que si consideramos sólo las derechas obtenemos el *orden posterior*. *Consigna:* Escribir un procedimiento `procedure BI_LISTADO(n: nodo; A: arbol);` que imprime el bi-listado de un nodo n en un árbol A. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

(b) **[Camino ordenado AB (30 puntos)]** Dado un árbol ordenado orientado `A` y un nodo `n` en él escribir una función `function CAMINO_ORD(n: nodo, A:arbol): boolean;` que retorna verdadero si existe algún camino ordenado desde `n` a una hoja en el subárbol del nodo `n`. Por ejemplo en el caso del árbol de la izquierda en la figura de abajo debe retornar `true` ya que el

camino marcado con líneas de puntos esta ordenado de menor a mayor yendo desde la raíz a la hoja, mientras que para el de la derecha debe retornar `false` ya que no existe ningún camino con esas características. Se sugiere el siguiente algoritmo: Un dado nodo `n` debe retornar `true` si, o bien es una hoja, o bien alguno de sus hijos `c` contiene un camino ordenado y `ETIQUETA(c)>ETIQUETA(n)`.



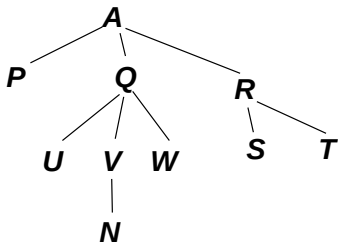
[Ej. 3] [Operativos (total = 30 puntos)]

(a) **[Árboles de Huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `MAREJADA`. $P(M) = 0.4, P(A) = 0.4, P(R) = 0.1, P(J) = 0.025, P(D) = 0.025, P(Z) = 0.025, P(W) = 0.025$. Calcular la longitud promedio del código obtenido.

(b) **[Reconstuir árbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, Q, W, P, R, M, A, B, D\}$,
- `ORD_POST` $=\{P, R, W, Q, A, D, B, M, Z\}$.

(c) **[Particionar árbol (10 ptos)]** Considerando el árbol de la figura, decir cuál son los nodos descendientes `DESC`, antecesores `A`, izquierda `I` y derecha `DER` del nodo Q.



Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: **1b.** [3 de junio de 2003]

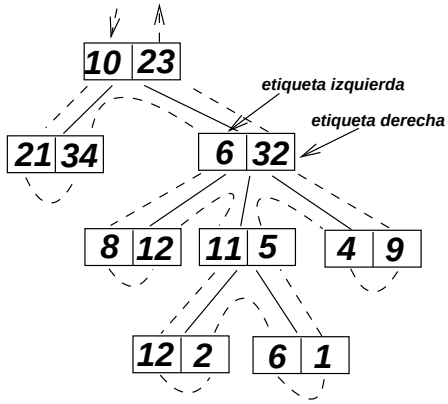
[Ej. 1] **[Primitivas (10 puntos)]** Escribir las funciones del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[Bilistado AOO (30 puntos)]**
El *"bi-listado"* de un árbol ordenado orientado puede pensarse como una combinación de los listados previo y posterior. Asumamos que las etiquetas tienen dos partes, una etiqueta *"derecha"* y una *"izquierda"*,

```
type tipo_etiqueta = record
    derecha, izquierda: integer
end;
```

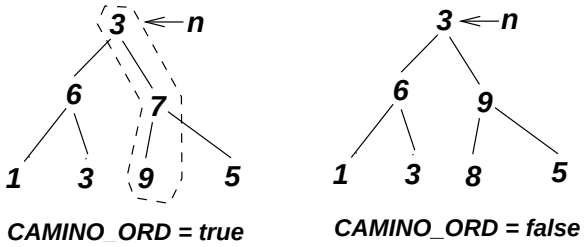
entonces el bi-listado de un nodo n se define como la lista vacía si el nodo es Λ y, si no, recursivamente como la etiqueta izquierda de n, seguida del bi-listado de los hijos, seguido de la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el bi-listado sigue la línea de puntos y resulta en la siguiente lista:
`BI-LISTADO`$=\{10, 21, 34, 6, 8, 12, 11, 12, 2, 6, 1, 5, 4, 9, 32, 23\}$. Notar que si consideramos sólo las etiquetas izuierdas, entonces el resultado es el *orden previo*, mientras que si consideramos sólo las derechas obtenemos el *orden posterior*. *Consigna:* Escribir un procedimiento `procedure BI_LISTADO(n: nodo; A: arbol);` que imprime el bi-listado de un nodo n en un árbol A. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

(b) **[Camino ordenado AB (30 puntos)]** Dado un árbol ordenado orientado `A` y un nodo `n` en él escribir una función `function CAMINO_ORD(n: nodo, A:arbol): boolean;` que retorna verdadero si existe algún camino ordenado desde `n` a una hoja en el subárbol del nodo `n`. Por ejemplo en el caso del árbol de la izquierda en la figura de abajo debe retornar `true` ya que el

camino marcado con líneas de puntos esta ordenado de menor a mayor yendo desde la raíz a la hoja, mientras que para el de la derecha debe retornar `false` ya que no existe ningún camino con esas características. Se sugiere el siguiente algoritmo: Un dado nodo `n` debe retornar `true` si, o bien es una hoja, o bien alguno de sus hijos `c` contiene un camino ordenado y `ETIQUETA(c)>ETIQUETA(n)`.



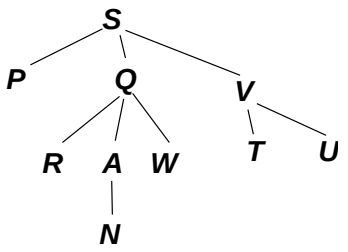
[Ej. 3] [Operativos (total = 30 puntos)]

(a) **[Árboles de Huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `MAREJADA`. $P(M) = 0.025, P(A) = 0.1, P(R) = 0.4, P(J) = 0.025, P(D) = 0.4, P(Z) = 0.025, P(W) = 0.025$. Calcular la longitud promedio del código obtenido.

(b) **[Reconstuir árbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, W, Z, T, Q, R, S, M, N\}$,
- `ORD_POST` $=\{Z, T, W, M, N, S, R, Q, A\}$.

(c) **[Particionar árbol (10 ptos)]** Considerando el árbol de la figura, decir cuál son los nodos descendientes `DESC`, antecesores `A`, izquierda `I` y derecha `DER` del nodo Q.



Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: **1c.** [3 de junio de 2003]

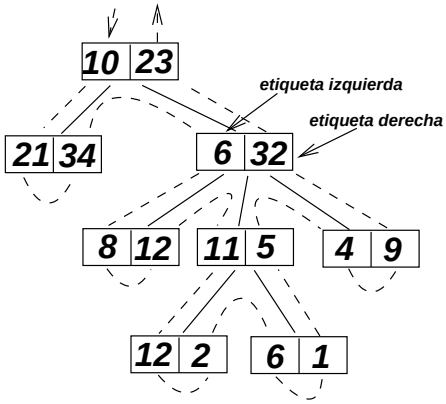
[Ej. 1] **[Primitivas (10 puntos)]** Escribir las funciones del `TAD ARBOL ORDENADO ORIENTADO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[Bilistado AOO (30 puntos)]**
El *"bi-listado"* de un árbol ordenado orientado puede pensarse como una combinación de los listados previo y posterior. Asumamos que las etiquetas tienen dos partes, una etiqueta *"derecha"* y una *"izquierda"*,

```
type tipo_etiqueta = record
    derecha, izquierda: integer
end;
```

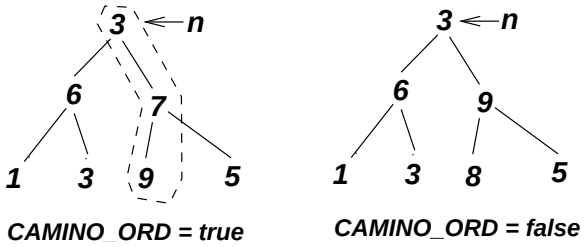
entonces el bi-listado de un nodo n se define como la lista vacía si el nodo es Λ y, si no, recursivamente como la etiqueta izquierda de n, seguida del bi-listado de los hijos, seguido de la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el bi-listado sigue la línea de puntos y resulta en la siguiente lista:
`BI-LISTADO`$=\{10, 21, 34, 6, 8, 12, 11, 12, 2, 6, 1, 5, 4, 9, 32, 23\}$. Notar que si consideramos sólo las etiquetas izuierdas, entonces el resultado es el *orden previo*, mientras que si consideramos sólo las derechas obtenemos el *orden posterior*. *Consigna:* Escribir un procedimiento `procedure BI_LISTADO(n: nodo; A: arbol);` que imprime el bi-listado de un nodo n en un árbol A. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

(b) **[Camino ordenado AB (30 puntos)]** Dado un árbol ordenado orientado `A` y un nodo `n` en él escribir una función `function CAMINO_ORD(n: nodo, A:arbol): boolean;` que retorna verdadero si existe algún camino ordenado desde `n` a una hoja en el subárbol del nodo `n`. Por ejemplo en el caso del árbol de la izquierda en la figura de abajo debe retornar `true` ya que el

camino marcado con líneas de puntos esta ordenado de menor a mayor yendo desde la raíz a la hoja, mientras que para el de la derecha debe retornar `false` ya que no existe ningún camino con esas características. Se sugiere el siguiente algoritmo: Un dado nodo `n` debe retornar `true` si, o bien es una hoja, o bien alguno de sus hijos `c` contiene un camino ordenado y `ETIQUETA(c)>ETIQUETA(n)`.



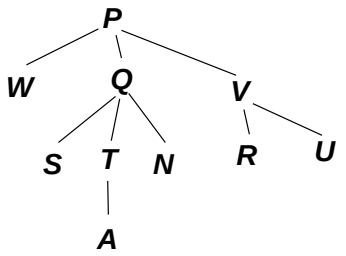
[Ej. 3] [Operativos (total = 30 puntos)]

(a) **[Árboles de Huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `MAREJADA`. $P(M) = 0.1, P(A) = 0.4, P(R) = 0.4, P(J) = 0.025, P(D) = 0.025, P(Z) = 0.025, P(W) = 0.025$. Calcular la longitud promedio del código obtenido.

(b) **[Reconstuir árbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{W, C, D, B, A, T, R, S\}$,
- `ORD_POST` $=\{C, D, B, T, R, S, A, W\}$.

(c) **[Particionar árbol (10 ptos)]** Considerando el árbol de la figura, decir cuál son los nodos descendientes `DESC`, antecesores `A`, izquierda `I` y derecha `DER` del nodo Q.



Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: **2a.** [3 de junio de 2003]

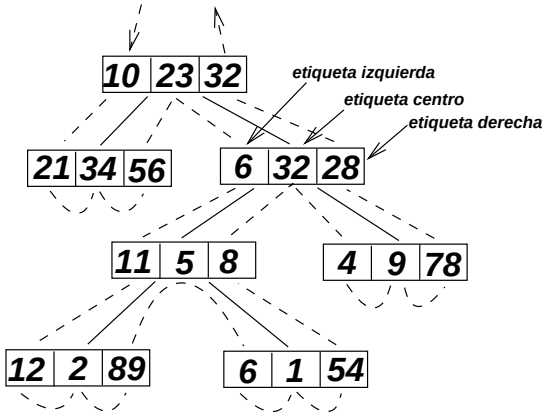
[Ej. 1] **[Primitivas (10 puntos)]** Escribir las funciones del `TAD ARBOL BINARIO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[Trilistado AB (30 puntos)]**
El *"tri-listado"* de un árbol binario puede pensarse como una combinación de los listados previo, posterior y simétrico. Asumamos que las etiquetas tienen tres partes *"derecha"*, *"centro"* e *"izquierda"*,

```
type tipo_etiqueta = record
    derecha, centro, izquierda: integer
end;
```

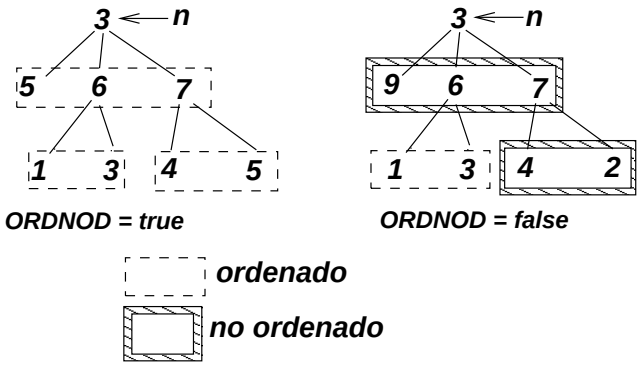
entonces el tri-listado de un nodo n se define como la lista vacía si el nodo es Λ y, si no, recursivamente como la etiqueta izquierda de n, seguida del tri-listado del hijo izquierdo, la etiqueta central, el tri-listado del hijo derecho y finalmente la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el tri-listado sigue la línea de puntos y resulta en la siguiente lista:
`TRI_LISTADO`=$\{10, 21, 34, 56, 23, 6, 11, 12, 2, 89, 5, 6, 1, 54, 8, 32, 4, 9, 78, 28, 32\}$. Notar que si consideramos sólo las etiquetas izquierdas, entonces el resultado es el *orden previo*, mientras que si consideramos sólo las derechas obtenemos el *orden posterior* y si consideramos las del centro, entonces obtenemos el *orden simétrico*. *Consigna:* Escribir un procedimiento `procedure TRI_LISTADO(n: nodo; A: arbol);` que imprime el tri-listado del subárbol de un nodo n en un árbol binario A. Usar las funciones del `TAD ARBOL BINARIO`: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

(b) **[Verifica orden AOO (30 puntos)]** Escribir una función `function ORDNODO(n: nodo, A:arbol): boolean;` que verifica si cada secuencia de

hermanos del subárbol del nodo `n` de un árbol ordenado orientado `A` están ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol de la izquierda debería retornar `true` mientras que para el de la derecha debería retornar `false` ya que las secuencias de hermanos indicados en las *dobles cajas* rayadas NO están ordenados. Se sugiere el siguiente algoritmo. Para un dado nodo retornar `false` si: 1) sus hijos no están ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad). Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.



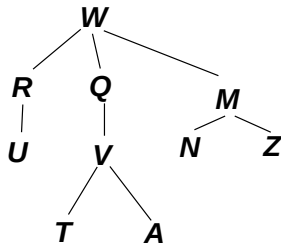
[Ej. 3] [Operativos (total = 30 puntos)]

(a) **[Árboles de Huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `EPIDEMIA`. $P(E) = 0.4, P(P) = 0.4, P(I) = 0.1, P(D) = 0.025, P(M) = 0.025, P(A) = 0.025, P(W) = 0.025$. Calcular la longitud promedio del código obtenido.

(b) **[Reconstuir árbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{D, Q, R, S, T, U, V, W\}$,
- `ORD_POST` $=\{S, T, U, V, W, R, Q, D\}$.

(c) **[Particionar árbol (10 ptos)]** Considerando el árbol de la figura, decir cuál son los nodos descendientes `DESC`, antecesores `A`, izquierda `I` y derecha `DER` del nodo Q.



Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: **2b.** [3 de junio de 2003]

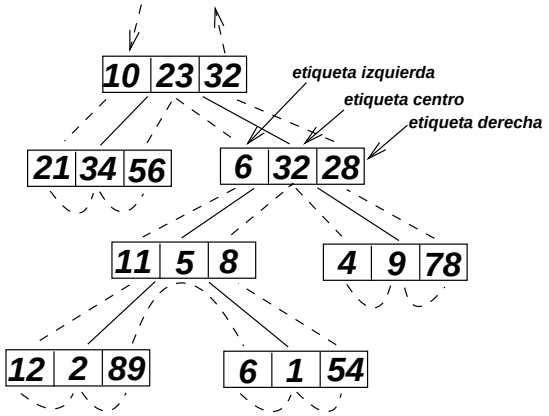
[Ej. 1] **[Primitivas (10 puntos)]** Escribir las funciones del `TAD ARBOL BINARIO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[Trilistado AB (30 puntos)]**
El *"tri-listado"* de un árbol binario puede pensarse como una combinación de los listados previo, posterior y simétrico. Asumamos que las etiquetas tienen tres partes *"derecha"*, *"centro"* e *"izquierda"*,

```
type tipo_etiqueta = record
    derecha, centro, izquierda: integer
end;
```

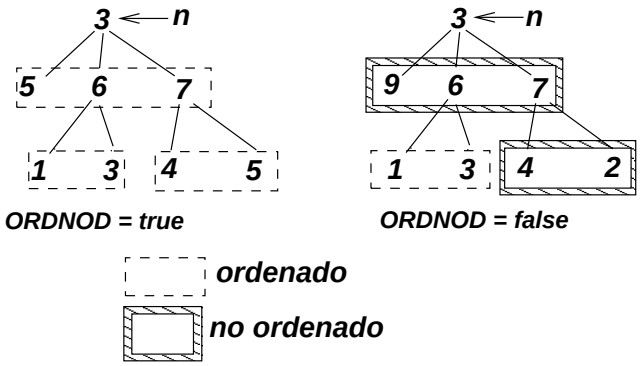
entonces el tri-listado de un nodo n se define como la lista vacía si el nodo es Λ y, si no, recursivamente como la etiqueta izquierda de n, seguida del tri-listado del hijo izquierdo, la etiqueta central, el tri-listado del hijo derecho y finalmente la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el tri-listado sigue la línea de puntos y resulta en la siguiente lista:
`TRI_LISTADO`=$\{10, 21, 34, 56, 23, 6, 11, 12, 2, 89, 5, 6, 1, 54, 8, 32, 4, 9, 78, 28, 32\}$. Notar que si consideramos sólo las etiquetas izquierdas, entonces el resultado es el *orden previo*, mientras que si consideramos sólo las derechas obtenemos el *orden posterior* y si consideramos las del centro, entonces obtenemos el *orden simétrico*. *Consigna:* Escribir un procedimiento `procedure TRI_LISTADO(n: nodo; A: arbol);` que imprime el tri-listado del subárbol de un nodo n en un árbol binario A. Usar las funciones del `TAD ARBOL BINARIO`: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

(b) **[Verifica orden AOO (30 puntos)]** Escribir una función `function ORDNODO(n: nodo, A:arbol): boolean;` que verifica si cada secuencia de

hermanos del subárbol del nodo `n` de un árbol ordenado orientado `A` están ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol de la izquierda debería retornar `true` mientras que para el de la derecha debería retornar `false` ya que las secuencias de hermanos indicados en las *dobles cajas* rayadas NO están ordenados. Se sugiere el siguiente algoritmo. Para un dado nodo retornar `false` si: 1) sus hijos no están ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad). Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.



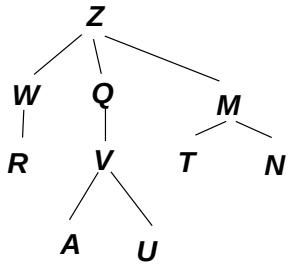
[Ej. 3] [Operativos (total = 30 puntos)]

(a) **[Árboles de Huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `EPIDEMIA`. $P(E) = 0.1, P(P) = 0.4, P(I) = 0.4, P(D) = 0.025, P(M) = 0.025, P(A) = 0.025, P(W) = 0.025$. Calcular la longitud promedio del código obtenido.

(b) **[Reconstuir árbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{M, N, P, T, Q, R, S, A, B\}$,
- `ORD_POST` $=\{N, T, R, A, B, S, Q, P, M\}$.

(c) **[Particionar árbol (10 ptos)]** Considerando el árbol de la figura, decir cuál son los nodos descendientes `DESC`, antecesores `A`, izquierda `I` y derecha `DER` del nodo Q.



Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: **2c.** [3 de junio de 2003]

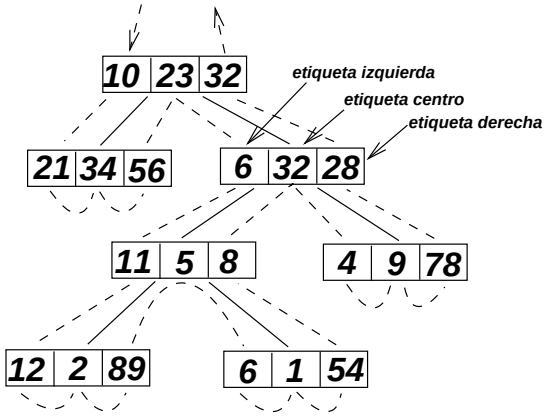
[Ej. 1] **[Primitivas (10 puntos)]** Escribir las funciones del `TAD ARBOL BINARIO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[Trilistado AB (30 puntos)]**
El *"tri-listado"* de un árbol binario puede pensarse como una combinación de los listados previo, posterior y simétrico. Asumamos que las etiquetas tienen tres partes *"derecha"*, *"centro"* e *"izquierda"*,

```
type tipo_etiqueta = record
    derecha, centro, izquierda: integer
end;
```

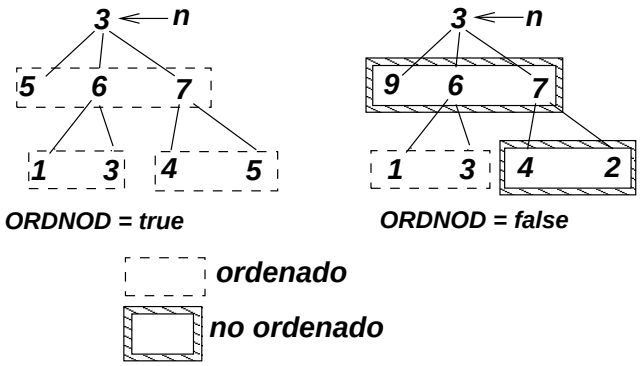
entonces el tri-listado de un nodo n se define como la lista vacía si el nodo es Λ y, si no, recursivamente como la etiqueta izquierda de n, seguida del tri-listado del hijo izquierdo, la etiqueta central, el tri-listado del hijo derecho y finalmente la etiqueta derecha de n. Por ejemplo, para el árbol de la figura



el tri-listado sigue la línea de puntos y resulta en la siguiente lista:
`TRI_LISTADO`=$\{10, 21, 34, 56, 23, 6, 11, 12, 2, 89, 5, 6, 1, 54, 8, 32, 4, 9, 78, 28, 32\}$. Notar que si consideramos sólo las etiquetas izquierdas, entonces el resultado es el *orden previo*, mientras que si consideramos sólo las derechas obtenemos el *orden posterior* y si consideramos las del centro, entonces obtenemos el *orden simétrico*. *Consigna:* Escribir un procedimiento `procedure TRI_LISTADO(n: nodo; A: arbol);` que imprime el tri-listado del subárbol de un nodo n en un árbol binario A. Usar las funciones del `TAD ARBOL BINARIO`: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`.

(b) **[Verifica orden AOO (30 puntos)]** Escribir una función `function ORDNODO(n: nodo, A:arbol): boolean;` que verifica si cada secuencia de

hermanos del subárbol del nodo `n` de un árbol ordenado orientado `A` están ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol de la izquierda debería retornar `true` mientras que para el de la derecha debería retornar `false` ya que las secuencias de hermanos indicados en las *dobles cajas* rayadas NO están ordenados. Se sugiere el siguiente algoritmo. Para un dado nodo retornar `false` si: 1) sus hijos no están ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad). Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `PADRE(n,A)`, `HIJO_MAS_IZQ(n,A)`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.



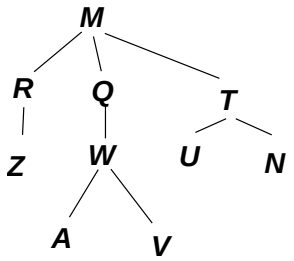
[Ej. 3] [Operativos (total = 30 puntos)]

(a) **[Árboles de Huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `EPIDEMIA`. $P(E) = 0.4, P(P) = 0.025, P(I) = 0.025, P(D) = 0.4, P(M) = 0.025, P(A) = 0.1, P(W) = 0.025$. Calcular la longitud promedio del código obtenido.

(b) **[Reconstuir árbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{P, Q, R, T, S, U, V, W, M\}$,
- `ORD_POST` $=\{Q, R, T, M, W, V, U, S, P\}$.

(c) **[Particionar árbol (10 ptos)]** Considerando el árbol de la figura, decir cuál son los nodos descendientes `DESC`, antecesores `A`, izquierda `I` y derecha `DER` del nodo Q.



Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Recup. Especial. [18 de junio de 2003]

[Ej. 1] [**Primitivas**] Escribir las funciones del `TAD ARBOL BINARIO` listadas a continuación, con *celdas enlazadas por punteros* **ó** *cursores*, a saber: `PADRE(n,A)`, `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)`, `CREA2(v, A1, A2)` y `ANULA(A)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] [**Programación**]

(a) [**MAXCOTA**] Escribir una función `function MAXCOTA(n:nodo;A:arbol;cota:integer):integer;` que retorna el máximo de las etiquetas de un árbol binario tales que son menores o iguales que la cota `c`. Por ejemplo, si las etiquetas de un árbol `A` son $\{1, 3, 7, 4, 2, 10, 13\}$ y `cota`=8, entonces `MAXCOTA(raiz(A),A,8)` debe retornar 7. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQ(n,A)`, `HIJO_DER(n,A)`, `ETIQUETA(n,A)` y `raiz(A)`.

(b) [**INCLUIDO**] Escribir una función `function INCLUIDO(nb,na: nodo; B,A: arbol) : boolean`, la cual retorna verdadero si la *estructura* del subárbol del nodo `nb` en el árbol ordenado orientado `B` está incluida dentro de la del subárbol del nodo `na` en el árbol `A`, independientemente de las etiquetas de los nodos correspondientes. Definiéndolo en forma recursiva, el subárbol del nodo `nb` está incluido en el de otro nodo `na` si:

- `nb`$= \Lambda$, **o**
- `nb`$<> \Lambda$, `na`$<> \Lambda$ y los subárboles de los hijos de `nb` están incluidos en los subárboles de los hijos de `na`. (Notar la *recursividad* de la definición).

Utilizar las funciones `PADRE (n,A), HIJO_MAS_IZQUIERDO (n,A), HERMANO_DERECHO (n,A),` y `ETIQUETA (n,A)`.

(c) [**CUENTA_COTA**] Escribir una funcion `function CUENTA_COTA(n:nodo; A:arbol; cota:integer):integer` que retorna el numero de nodos de un árbol orientado que tienen etiqueta menor o igual que cota. Utilizar las funciones `PADRE (n,A), HIJO_MAS_IZQUIERDO (n,A), HERMANO_DERECHO (n,A),` y `ETIQUETA (n,A)`.

[Ej. 3] [**Operativos**]

(a) [**Reconstruir árbol**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, T, U, V, X, Y, W, A, B\}$,
- `ORD_POST` $=\{X, Y, V, A, B, W, U, T, Z\}$.

(b) [**Ubicar nodos**] Dado el arbol siquiente

i. Ubicar los nodos que estén a la izquierda de A y a la derecha de I

ii. Ubicar los nodos que estén a la derecha de D y sea descendiente de J

(c) [**Arbol de expresiones**] Dada la expresión siguiente

$$(a + d * e - f/g) * (h/p - r) \tag{1}$$

encontrar el árbol binario correspondiente. Escribir la expresión en notación prefija y postfija.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 3er Parcial. Tema: **1A.** [8 de julio de 2003]

[Ej. 1] **[Primitivas (20 puntos)]** Escribir las funciones del `TAD DICCIONARIO` listadas a continuación, implementado por tabla de dispersión cerrada con resolución lineal de colisiones `ANULA(D)`, `INSERTA(x,D)`, `MIEMBRO(x,D)` y `SUPRIME(x,D)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[MergeSort-Listas (30 puntos)]** Escribir un procedimiento `procedure MERGESORT(var L:lista);` que ordena los elementos de una lista usando el método de clasificación por intercalación. Es decir, divide la lista en dos sublistas auxiliares `L1` y `L2` de tamaño similar, las ordena aplicando `MERGESORT` recursivamente y luego intercala las sublistas:

- dividir `L` en dos sublistas `L1`, `L2`
- clasificar `L1` y `L2` por `MERGESORT`
- intercalar `L1` y `L2` en `L`

Usar las primitivas del `TAD LISTA`: `RECUPERA (p,L)`, `SIGUIENTE (p,L)`, `PRIMERO (L)`, y `FIN (L)` que considere necesarias.

(b) **[Selección para colas (30 puntos)]** Escribir un procedimiento `procedure SELECCION (var C:cola);` que ordena los elementos de una cola, usando una cola auxiliar `CAUX`. El procedimiento va extrayendo sucesivamente el mínimo de los valores de la cola `C` y los va insertando en la cola `CAUX` hasta que `C` este vacía. Finalmente pasa todo el contenido de `CAUX` a `C`. Para extraer el mínimo de una cola escribir una función auxiliar `function SUPRIME_MIN (var C:cola): integer;` que retorna el mínimo de los elementos de una cola, suprimiéndolo, en donde se usa otra cola auxiliar `C2`. Usar las primitivas del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA(C)`, y `FRENTE_DE_COLA (C)` que considere necesarias.

[Ej. 3] **[Operativos (total = 20 puntos)]**

(a) **[Heap-sort (10 ptos)]** Dados los enteros $\{1, 5, 8, 2, 3, 0, 10, 4\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(b) **[Bin-sort (10 ptos)]** Dados los enteros $\{520, 811, 405, 71, 685, 23, 442, 536\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se puede utilizar para ubicar los elementos en las urnas durante cada pasada.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 3er Parcial. Tema: **1B.** [8 de julio de 2003]

[Ej. 1] **[Primitivas (20 puntos)]** Escribir las funciones del `TAD DICCIONARIO` listadas a continuación, implementado por tabla de dispersión cerrada con resolución lineal de colisiones `ANULA(D)`, `INSERTA(x,D)`, `MIEMBRO(x,D)` y `SUPRIME(x,D)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[MergeSort-Listas (30 puntos)]** Escribir un procedimiento `procedure MERGESORT(var L:lista);` que ordena los elementos de una lista usando el método de clasificación por intercalación. Es decir, divide la lista en dos sublistas auxiliares `L1` y `L2` de tamaño similar, las ordena aplicando `MERGESORT` recursivamente y luego intercala las sublistas:

- dividir `L` en dos sublistas `L1`, `L2`
- clasificar `L1` y `L2` por `MERGESORT`
- intercalar `L1` y `L2` en `L`

Usar las primitivas del `TAD LISTA`: `RECUPERA (p,L)`, `SIGUIENTE (p,L)`, `PRIMERO (L)`, y `FIN (L)` que considere necesarias.

(b) **[Selección para colas (30 puntos)]** Escribir un procedimiento `procedure SELECCION (var C:cola);` que ordena los elementos de una cola, usando una cola auxiliar `CAUX`. El procedimiento va extrayendo sucesivamente el mínimo de los valores de la cola `C` y los va insertando en la cola `CAUX` hasta que `C` este vacía. Finalmente pasa todo el contenido de `CAUX` a `C`. Para extraer el mínimo de una cola escribir una función auxiliar `function SUPRIME_MIN (var C:cola): integer;` que retorna el mínimo de los elementos de una cola, suprimiéndolo, en donde se usa otra cola auxiliar `C2`. Usar las primitivas del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA(C)`, y `FRENTE_DE_COLA (C)` que considere necesarias.

[Ej. 3] **[Operativos (total = 20 puntos)]**

(a) **[Heap-sort (10 ptos)]** Dados los enteros $\{5, 9, 12, 6, 7, 4, 14, 8\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(b) **[Bin-sort (10 ptos)]** Dados los enteros $\{685, 23, 442, 536, 520, 811, 405, 71\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se puede utilizar para ubicar los elementos en las urnas durante cada pasada.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 3er Parcial. Tema: **1C.** [8 de julio de 2003]

[Ej. 1] **[Primitivas (20 puntos)]** Escribir las funciones del `TAD DICCIONARIO` listadas a continuación, implementado por tabla de dispersión cerrada con resolución lineal de colisiones `ANULA(D)`, `INSERTA(x,D)`, `MIEMBRO(x,D)` y `SUPRIME(x,D)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[MergeSort-Listas (30 puntos)]** Escribir un procedimiento `procedure MERGESORT(var L:lista);` que ordena los elementos de una lista usando el método de clasificación por intercalación. Es decir, divide la lista en dos sublistas auxiliares `L1` y `L2` de tamaño similar, las ordena aplicando `MERGESORT` recursivamente y luego intercala las sublistas:

- dividir `L` en dos sublistas `L1`, `L2`
- clasificar `L1` y `L2` por `MERGESORT`
- intercalar `L1` y `L2` en `L`

Usar las primitivas del `TAD LISTA`: `RECUPERA (p,L)`, `SIGUIENTE (p,L)`, `PRIMERO (L)`, y `FIN (L)` que considere necesarias.

(b) **[Selección para colas (30 puntos)]** Escribir un procedimiento `procedure SELECCION (var C:cola);` que ordena los elementos de una cola, usando una cola auxiliar `CAUX`. El procedimiento va extrayendo sucesivamente el mínimo de los valores de la cola `C` y los va insertando en la cola `CAUX` hasta que `C` este vacía. Finalmente pasa todo el contenido de `CAUX` a `C`. Para extraer el mínimo de una cola escribir una función auxiliar `function SUPRIME_MIN (var C:cola): integer;` que retorna el mínimo de los elementos de una cola, suprimiéndolo, en donde se usa otra cola auxiliar `C2`. Usar las primitivas del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA(C)`, y `FRENTE_DE_COLA (C)` que considere necesarias.

[Ej. 3] **[Operativos (total = 20 puntos)]**

(a) **[Heap-sort (10 ptos)]** Dados los enteros $\{3, 7, 10, 4, 5, 2, 12, 6\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

(b) **[Bin-sort (10 ptos)]** Dados los enteros $\{405, 71, 685, 23, 442, 536, 520, 811\}$ ordenarlos por el método de urnas (*"bin-sort"*), con $B = 10$ urnas, y tantas pasadas como sea necesario. Mostrar la tabla de urnas y su contenido después de cada pasada. Especificar cuál es la instrucción Pascal que se puede utilizar para ubicar los elementos en las urnas durante cada pasada.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 3er Parcial. Tema: **2A.** [8 de julio de 2003]

[Ej. 1] **[Primitivas (20 puntos)]** Escribir las funciones del `TAD COLA DE PRIORIDAD` listadas a continuación, implementado por montículos: `ANULA (C)`, `INSERTA (x,C);`, `SUPRIME_MIN (C)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[MergeSort-Colas (30 puntos)]** Escribir un procedimiento `procedure MERGESORT (var C:cola);` que ordena los elementos de una cola `C` usando el método de clasificación por intercalación. Es decir, divide la cola en dos subcolas auxiliares `C1` y `C2` de tamaño similar, las ordena aplicando `MERGESORT` recursivamente y luego intercala las subcolas:

- dividir `C` en dos subcolas `C1`, `C2`
- clasificar `C1` y `C2` por `MERGESORT`
- intercalar `C1` y `C2` en `C`

Usar las primitivas del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA (C)`, y `FRENTE_DE_COLA (C)` que considere necesarias.

(b) **[Burbuja para listas (30 puntos)]** Escribir un procedimiento `procedure BURBUJA(var L:lista);` que ordena los elementos de una lista `L` mediante el método de clasificación por burbujas. Como en listas simplemente enlazadas no es conveniente avanzar hacia el principio de la lista por una cuestión de eficiencia, el algoritmo de burbuja se modifica de manera de que los elementos más grandes viajan hacia el fin. Para arreglos, sería

```
for  i := n downto 2 do
   for  j := 1 to (i-1) do
      if  A[j].clave > A[j+1].clave then
         intercambia (A[j].clave, A[j+1].clave);
```

Usar las primitivas del `TAD LISTA`: `RECUPERA (p,L)`, `SIGUIENTE (p,L)`, `PRIMERO (L)`, y `FIN (L)` que considere necesarias.

[Ej. 3] **[Operativos (total = 20 puntos)]**

(a) **[Quick-sort (10 ptos)]** Dados los enteros $\{6, 10, 5, 2, 6, 11, 9, 4, 4, 3, 12, 7\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.

(b) **[Heap-sort (10 ptos)]** Dados los enteros $\{2, 6, 9, 3, 4, 1, 11, 5\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 3er Parcial. Tema: **2B.** [8 de julio de 2003]

[Ej. 1] **[Primitivas (20 puntos)]** Escribir las funciones del `TAD COLA DE PRIORIDAD` listadas a continuación, implementado por montículos: `ANULA (C)`, `INSERTA (x,C);`, `SUPRIME_MIN (C)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[MergeSort-Colas (30 puntos)]** Escribir un procedimiento `procedure MERGESORT (var C:cola);` que ordena los elementos de una cola `C` usando el método de clasificación por intercalación. Es decir, divide la cola en dos subcolas auxiliares `C1` y `C2` de tamaño similar, las ordena aplicando `MERGESORT` recursivamente y luego intercala las subcolas:

- dividir `C` en dos subcolas `C1`, `C2`
- clasificar `C1` y `C2` por `MERGESORT`
- intercalar `C1` y `C2` en `C`

Usar las primitivas del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA (C)`, y `FRENTE_DE_COLA (C)` que considere necesarias.

(b) **[Burbuja para listas (30 puntos)]** Escribir un procedimiento `procedure BURBUJA(var L:lista);` que ordena los elementos de una lista `L` mediante el método de clasificación por burbujas. Como en listas simplemente enlazadas no es conveniente avanzar hacia el principio de la lista por una cuestión de eficiencia, el algoritmo de burbuja se modifica de manera de que los elementos más grandes viajan hacia el fin. Para arreglos, sería

```
for  i := n downto 2 do
   for  j := 1 to (i-1) do
      if  A[j].clave > A[j+1].clave then
         intercambia (A[j].clave, A[j+1].clave);
```

Usar las primitivas del `TAD LISTA`: `RECUPERA (p,L)`, `SIGUIENTE (p,L)`, `PRIMERO (L)`, y `FIN (L)` que considere necesarias.

[Ej. 3] **[Operativos (total = 20 puntos)]**

(a) **[Quick-sort (10 ptos)]** Dados los enteros $\{5, 9, 1, 2, 6, 7, 11, 2, 7, 11, 12, 8\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.

(b) **[Heap-sort (10 ptos)]** Dados los enteros $\{4, 8, 11, 5, 6, 3, 13, 7\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. 3er Parcial. Tema: **2C.** [8 de julio de 2003]

[Ej. 1] **[Primitivas (20 puntos)]** Escribir las funciones del `TAD COLA DE PRIORIDAD` listadas a continuación, implementado por montículos: `ANULA (C)`, `INSERTA (x,C);`, `SUPRIME_MIN (C)`. **Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.**

[Ej. 2] **[Programación (total = 60 puntos)]**

(a) **[MergeSort-Colas (30 puntos)]** Escribir un procedimiento `procedure MERGESORT (var C:cola);` que ordena los elementos de una cola `C` usando el método de clasificación por intercalación. Es decir, divide la cola en dos subcolas auxiliares `C1` y `C2` de tamaño similar, las ordena aplicando `MERGESORT` recursivamente y luego intercala las subcolas:

- dividir `C` en dos subcolas `C1`, `C2`
- clasificar `C1` y `C2` por `MERGESORT`
- intercalar `C1` y `C2` en `C`

Usar las primitivas del `TAD COLA`: `ANULA (C)`, `PONE_EN_COLA (x,C)`, `QUITA_DE_COLA (C)`, `VACIA (C)`, y `FRENTE_DE_COLA (C)` que considere necesarias.

(b) **[Burbuja para listas (30 puntos)]** Escribir un procedimiento `procedure BURBUJA(var L:lista);` que ordena los elementos de una lista `L` mediante el método de clasificación por burbujas. Como en listas simplemente enlazadas no es conveniente avanzar hacia el principio de la lista por una cuestión de eficiencia, el algoritmo de burbuja se modifica de manera de que los elementos más grandes viajan hacia el fin. Para arreglos, sería

```
for  i := n downto 2 do
   for  j := 1 to (i-1) do
      if  A[j].clave > A[j+1].clave then
         intercambia (A[j].clave, A[j+1].clave);
```

Usar las primitivas del `TAD LISTA`: `RECUPERA (p,L)`, `SIGUIENTE (p,L)`, `PRIMERO (L)`, y `FIN (L)` que considere necesarias.

[Ej. 3] **[Operativos (total = 20 puntos)]**

(a) **[Quick-sort (10 ptos)]** Dados los enteros $\{1, 1, 11, 2, 7, 12, 6, 2, 6, 5, 11, 6\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.

(b) **[Heap-sort (10 ptos)]** Dados los enteros $\{6, 11, 14, 8, 9, 6, 16, 10\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [12 de Febrero de 2004]

Ej. 1.- [**Primitivas (20 puntos)**] Escribir las funciones del `TAD DICCIONARIO` listadas a continuación, implementado por tabla de dispersión cerrada con resolución lineal de colisiones `ANULA(D)`, `INSERTA(x,D)`, `MIEMBRO(x,D)` y `SUPRIME(x,D)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

Ej. 2.- [**Ejercicios de programación (total 80 puntos)**]

a) Escribir un procedimiento `procedure KRONECKER(LA,LB: lista; var L:lista);` que, dadas dos listas `LA`=$\{a_1, a_2, ..., a_N\}$ y `LB`=$\{b_1, b_2, ..., b_M\}$ retorna otra lista `L` con $N \times M$ valores de la siguiente forma:

$$\begin{aligned} L = \{&a_1b_1, a_1b_2, ..., a_1b_M, \\ &a_2b_1, a_2b_2, ..., a_2b_M, \\ &... \\ &a_Nb_1, a_Nb_2, ..., a_Nb_M\} \end{aligned} \tag{1}$$

Por ejemplo, si `LA`=$\{1, 3, 5, 0\}$ y `LB`=$\{1, 2, 3, 4\}$ entonces `KRONECKER(LA,LB,L)` debe hacer que

$$L = \{1, 2, 3, 4, 3, 6, 9, 12, 5, 10, 15, 20, 0, 0, 0, 0\}. \tag{2}$$

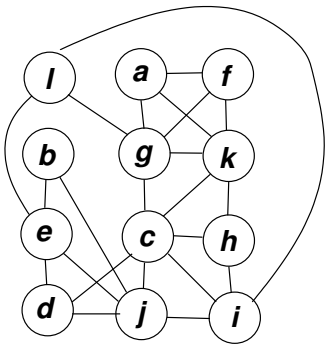
Usar las siguientes primitivas del `TAD LISTA`: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. **N**o usar ninguna estructura auxiliar.

b) [**cant-hojas (20 puntos)**] Escribir una función recursiva `function CANT_HOJAS(a:nodo; A:arbol):integer` que retorna la cantidad de hojas del árbol cuya raíz es *a*. Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)}`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

c) [**iguales (35 puntos)**] Escribir una función recursiva `function IGUALES(n,m: nodo; A: arbol) : boolean` que retorna verdadero (true) si dos árboles ordenados y orientados son iguales y falso (false) si no lo son. Dos árboles son iguales si tienen la misma estructura y contenido (etiquetas). Usar las funciones del `TAD ARBOL ORDENADO ORIENTADO`: `HIJO_MAS_IZQ(n,A)}`, `HERMANO_DER(n,A)`, `ETIQUETA(n,A)`.

Ej. 3.- [**LIBRES**] **Ejercicios operativos (total 80 puntos)**

a) [**colorear-grafo (20 puntos)**] Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



b) **[reconstruir-arbol (20 puntos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{15, 22, 3, 14, 6, 7, 81, 9, 11, 12, 10, 5\}$.
- `ORD_POST` $=\{22, 81, 7, 6, 12, 11, 9, 10, 14, 5, 3, 15\}$.

c) **[abb (20 puntos)]** Dados los caracteres $\{z, t, a, l, r, k, b, h, w, x, n\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos z, k y l.

d) **[tablas de dispersión (20 puntos)]** Insertar los números $9, 22, 32, 15, 14, 42, 24, 11, 34$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

Ej. 4.- [LIBRES] preguntas (total 20ptos, 5ptos/preg): [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Las operaciones `INSERTA` y `SUPRIME_MIN` sobre el `TAD COLA DE PRIORIDAD` representado por montículos requiere un número de operaciones que está relacionado con ...

- ☐ ... la longitud del camino en el àrbol desde la raíz hasta el punto de inserción o supresión.
- ☐ ... el número de elementos en el montículo n
- ☐ ... el cuadrado del número de elementos en el montículo n
- ☐ ... el logaritmo de la semilla.

b) El algoritmo de clasificación por urnas tiene un tiempo de ejecución $T(n) = O(n)$ pero está restringido a que el número de claves posibles ...

- ☐ ... sea finito.
- ☐ ... sea delgado.
- ☐ ... sea una potencia de 2.
- ☐ ... sea infinito.

c) La ventaja de la representación del `TAD COLA` por *arreglo circular* con respecto a la representación por celdas enlazadas es que ...

- ☐ ... requiere menos memoria ya que no necesita los punteros para enlazar celdas.
- ☐ ... el orden de las operaciones `FRENTE, QUITA, PONE` es $O(\log n)$.
- ☐ ... permite inserciones/supresiones en el medio de la cola.
- ☐ ... es más simple.

d) La búsqueda *exhaustiva* del camino de longitud mínima para el *"Problema del Agente Viajante"* (PAV) tiene un tiempo de ejecución... (n es el número de ciudades)

- ☐ $O(n)$
- ☐ $O(1)$
- ☐ $O(n\$\$\mathrm{O}(\mathrm{n}^2)$

Apellido y Nombre: _____

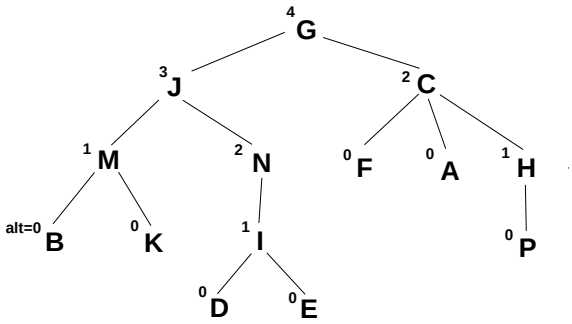
Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [26 de Febrero de 2004]

Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado mediante *listas enlazadas clasificadas*. Es decir, implementar en Pascal los siguientes procedimientos/funciones listados abajo. Incluir todas las definiciones de tipo necesarias. `ANULA(A)`, `UNION(A,B,C)`, `INTERSECCION(A,B,C)`, `DIFERENCIA(A,B,C)`, `MIEMBRO(x,A)`, `MIN(A)`, `INSERTA(x,A)` y `SUPRIME(x,A)`.

Ej. 2.- Recordemos que la *altura* de un nodo en un árbol, es la máxima longitud de los caminos que van desde el nodo a una hoja descendiente del mismo. Por ejemplo, consideremos el árbol de la figura. El número que está arriba y a la izquierda de cada nodo es la altura del nodo.



Escribir una función `function CUENTA_ALT(n:nodo; m:integer; var altura:integer; A:arbol) : integer;` que dado un nodo `n` en un árbol `A` cuenta el número de nodos del subárbol de `A` cuya raiz es `n` tales que su altura es menor o igual que `m`. Además, en la variable `altura` debe retornar la altura del nodo `n`. Por lo tanto las siguientes llamadas deben retornar, para el árbol del ejemplo,

```
CUENTA_ALT(G,0,altura,G) -> 7
CUENTA_ALT(G,1,altura,G) -> 10
CUENTA_ALT(G,2,altura,G) -> 12
CUENTA_ALT(G,3,altura,G) -> 13
CUENTA_ALT(G,4,altura,G) -> 14
```

En todos estos casos, la variable `altura` debe retornar `altura=4`. Por otra parte, si nos referimos al nodo `N`, entonces debe retornar

```
CUENTA_ALT(N,0,altura,G) -> 2
CUENTA_ALT(N,1,altura,G) -> 3
CUENTA_ALT(N,2,altura,G) -> 4
```

y `altura=2`.

Usar las primitivas de árbol ordenado orientado siguientes: `HIJO_MAS_IZQ(n,A)`,`HERMANO_DER(n,A)`. Hacer la función *recursiva*. Notar que el conteo de nodos que cumple la condición para un nodo dado es igual a la suma sobre los hijos más 1 o 0, dependiendo de si la altura del nodo es menor que `m` o no.

Ej. 3.- Ejercicios básicos sobre TAD's

a) Escribir un procedimiento `procedure STRIDE(var L1:lista; L:lista; c, f, i: integer);` que retorna en `L1` todos le elementos en las posiciones desde el comienzo `c` hasta el fin `f` (pero sin incluirlo) a intervalos `i`. Es decir, debe retornar los elementos en las posiciones `c, c+i, c+2*i, ...`, mientras éstas sean menores que `f` y esten dentro del rango de posiciones válidas en la lista. Por ejemplo, si `L=(3,2,1,5,4,2,3,2,6)` y llamamos `STRIDE(L1,L,2,9,3)`, entonces debe retornar `L1=(2,4,2)`, mientras que si hacemos `STRIDE(L1,L,3,20,4)` debe retornar `L1=(1,3)` Utilizar las siguientes primitivas:

- **TAD LISTA:** `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`.

b) Escribir una función `function SUMA_COLA(C:cola): integer;` que calcula la suma de los elementos de una cola de enteros. La cola original debe quedar inalterada. *Sugerencia:* usar una variable auxiliar (cola o lista). Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`.

Ej. 4.- **[LIBRES] Ejercicios operativos:**

a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{M, G, Z, R, N, A, B, D, E\}$.
- `ORD_POST` $=\{Z, R, G, A, D, E, B, N, M\}$.

b) **[LIBRES]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PACIENCIA` $P(P) = 0{,}3, P(A) = 0{,}1, P(C) = 0{,}3, P(I) = 0{,}05, P(E) = 0{,}05, P(N) = 0{,}2$ Calcular la longitud promedio del código obtenido.

Ej. 5.- **[LIBRES] Preguntas:** [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Dadas las funciones $T_1(n) = 2\sqrt{n} + 0{,}5n!$, $T_2(n) = n^2 + 5n^3$, $T_3(n) = \log n + 2n$ y $T_4(n) = n^2 + 3\sqrt{n}$ decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_3 < T_4 < T_1 < T_2$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_3 < T_4 < T_2 < T_1$
- ☐ $T_4 < T_3 < T_2 < T_1$

b) Sea una lista `L=(1,3,5,4)` simplemente enlazada por punteros o cursores, y sea `p` la posición correspondiente al elemento 3. Despues de hacer `SUPRIME(p,L)`, ¿cual es el resultado de hacer `x = RECUPERA(p,L)`? ...

- ☐ ... retorna el elemento 1
- ☐ ... produce un error
- ☐ ... retorna el elemento 5
- ☐ ... retorna el elemento 3

c) ¿Cuál es el criterio para elegir una buena función de dispersión? ...

- ☐ Debe tratar de concentrar los elementos en pocas cubetas.
- ☐ Debe tratar de concentrar los elementos en una sóla cubeta.
- ☐ Debe tratar de concentrar los elementos en la primera cubeta.
- ☐ Debe distribuir los elementos en la forma más uniforme posible entre las cubetas.

d) ¿Cuál de los siguientes algoritmos de clasificación es el más rápido *en el caso promedio*?

- ☐ Burbuja (*"Bubble-sort"*)
- ☐ Clasificación por *incrementos decrecientes* (*shell-sort*)
- ☐ Selección
- ☐ Clasificación rápida (*"Quick-sort"*)

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

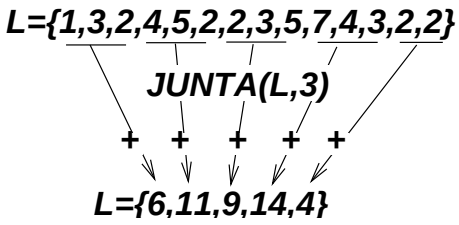
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [6 de Mayo de 2004]

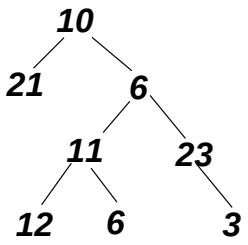
Ej. 1.- [**primi (20 puntos)**] Escribir las funciones del `TAD LISTA` con celdas simplemente enlazadas por punteros **ó** cursores. `INSERTA(x,p,L)`, `LOCALIZA(x,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. *Escribir todos los tipos, definiciones, funciones y procedimientos auxiliares necesarios.*

Ej. 2.- [**Programacion (total 80 puntos)**]

a) [**junta (40 puntos)**] Escribir un procedimiento `procedure JUNTA(var L: lista; n:integer);` que dada una lista `L`, agrupa de a `m` elementos dejando su suma (ver figura). Usar las siguientes primitivas del `TAD LISTA`: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



b) [**maximo-par (20 puntos)**] Escribir una función `function MAXIMO_PAR(n:nodo; A:arbol):integer;` que retorna el máximo de las etiquetas **pares** de un árbol binario. En el caso del árbol de la figura debe retornar 12. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQUIERDO(n,A)`, `HIJO_DERECHO(n,A)` y `ETIQUETA(n,A)`,



c) [**elimina-valor (20 puntos)**] Escribir un procedimiento `procedure ELIMINA_VALOR(var C:cola; n: integer);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C = {1,3,5,4,2,3,7,3,5}`, después de `ELIMINA_VALOR(C,3)` debe quedar `C = {1,5,4,2,7,5}`. *Sugerencia: Usar una estructura auxiliar lista o cola.* Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

Ej. 3.- [**LIBRES**] **Ejercicios operativos (total 80 puntos):**

a) [**arboles (30 puntos)** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Q, T, R, S, V, W, Y, M, N\}$,
- `ORD_POST` $=\{T, S, M, N, Y, W, V, R, Q\}$.

b) **[huffman (30 puntos)]:** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `TRANSVERSAL` $P(T) = 0.4, P(R) = 0.2, P(A) = 0.1, P(N) = 0.1, P(S) = 0.05, P(V) = 0.05, P(E) = 0.05, P(L) = 0.05$ Calcular la longitud promedio del código obtenido.

Ej. 4.- **[abb (20 ptos)]** Dados los enteros $\{4, 8, 11, 6, 5, 13, 23\}$ insertarlos, en ese orden, en un `TAD ARBOL BINARIO DE BUSQUEDA`. Mostrar las operaciones necesarias para eliminar los elementos 8, 11 y 6, en ese orden.

Ej. 5.- **[LIBRES] [preguntas (total 20ptos, 5 ptos c/una)]** [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- $O(n^2/B)$
- $O((n/B)^2)$
- $O(n + B)$
- $O(n/B)$

---

b) Dadas las funciones

- $T_1(n) = 3n^3 + 7n!$,
- $T_2(n) = 2n^2 + 0.5\sqrt{n}$,
- $T_3(n) = 7n^3 + 8!$ y
- $T_4(n) = 4^3 + 7^n$

decir cuál de los siguientes ordenamientos es el correcto

- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_3 < T_4 < T_2 < T_1$
- ☐ $T_2 < T_3 < T_4 < T_1$
- ☐ $T_4 < T_3 < T_2 < T_1$

---

c) Sea una lista `L=(1,3,5,7)` simplemente enlazada por punteros o cursores. Despues de hacer

```
p = PRIMERO(L);
p = SIGUIENTE(p,L);
p = SIGUIENTE(p,L);
SUPRIME(p,L);
p = SIGUIENTE(p,L);
```

¿Que retorna `RECUPERA(p,L)` ?

- ☐ ... retorna 1
- ☐ ... produce un error
- ☐ ... retorna 7
- ☐ ... retorna 3

d) ¿Cuál es el criterio para elegir una buena función de dispersión? ...

- ☐ Debe tratar de concentrar los elementos en pocas cubetas.
- ☐ Debe tratar de concentrar los elementos en una sóla cubeta.
- ☐ Debe tratar de concentrar los elementos en la primera cubeta.
- ☐ Debe distribuir los elementos en la forma más uniforme posible entre las cubetas.

Apellido y Nombre: _____

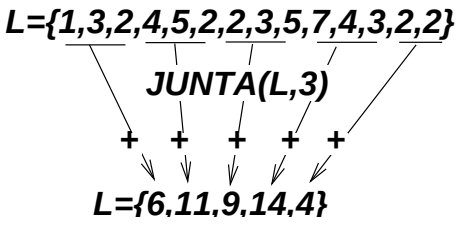
Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
Examen Final. [Pascal - 08 de Julio de 2004]

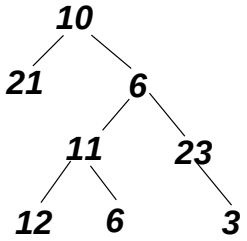
Ej. 1.- Escribir las funciones primitivas del `TAD CONJUNTO` implementado *listas enlazadas clasificadas*. Es decir, implementar en Pascal los siguientes procedimientos/funciones listados abajo. Incluir todas las definiciones de tipo necesarias. (a) `ANULA(A)`, (b) `UNION(A,B,C)`, (c) `INTERSECCION(A,B,C)`, (d) `DIFERENCIA(A,B,C)`, (e) `MIEMBRO(x,A)`, (f) `MIN(A)`, (g) `INSERTA(x,A)` y (h) `SUPRIME(x,A)`.

Ej. 2.- Escribir un procedimiento `procedure JUNTA(var L: lista; n:integer);` que dada una lista `L`, agrupa de a `m` elementos dejando su suma (ver figura). Usar las siguientes primitivas del `TAD LISTA`: `INSERTA(x,p,L)`, `RECUPERA(p,L)`, `SUPRIME(p,L)`, `SIGUIENTE(p,L)`, `ANULA(L)`, `PRIMERO(L)`, y `FIN(L)`. **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



Ej. 3.- Uso básico de TAD's:

a) Escribir una función `function MAXIMO_PAR(n:nodo; A:arbol):integer;` que retorna el máximo de las etiquetas **pares** de un árbol binario. En el caso del árbol de la figura debe retornar 12. Usar las primitivas del `TAD ARBOL BINARIO`: `HIJO_IZQUIERDO(n,A)`, `HIJO_DERECHO(n,A)` y `ETIQUETA(n,A)`,



b) Escribir un procedimiento `procedure ELIMINA_VALOR(var C:cola; n: integer);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C =` {`1,3,5,4,2,3,7,3,5`}, después de `ELIMINA_VALOR(C,3)` debe quedar `C =` {`1,5,4,2,7,5`}. *Sugerencia: Usar una estructura auxiliar lista o cola.* Utilizar las primitivas del **TAD COLA:** `ANULA(C)`, `PONE_EN_COLA(x,C)`, `QUITA_DE_COLA(C)`, `VACIA(C)`, y `FRENTE_DE_COLA(C)`. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

Ej. 4.- [LIBRES] Ejercicios operativos:

a) **Árboles:** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, D, E, R, S, Q, A, B\}$,
- `ORD_POST` $=\{D, Q, R, A, B, S, E, C\}$.

b) **Árboles de Huffman:** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PIQUETERO`
$P(P) = 0{,}2, P(I) = 0{,}2, P(Q) = 0{,}2, P(U) = 0{,}1, P(E) = 0{,}1, P(T) = 0{,}1, P(R) = 0{,}05, P(O) = 0{,}05$ Calcular la longitud promedio del código obtenido.

Ej. 5.- [LIBRES] Preguntas: [Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) El tiempo de ejecución para el algoritmo de clasificación rápida (*"quicksort"*) **en el peor caso** es

- $O(\log n)$
- $O(n)$
- $O(n^2)$
- $O(n \log n)$

b) El tiempo de ejecución para el algoritmo de clasificación por montículos es

- $O(n \log n)$ en el mejor caso, $O(n^2)$ en promedio
- siempre $O(n \log n)$
- siempre $O(n^2)$
- a veces $O(n^2)$

c) Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- $O(n^2/B)$
- $O((n/B)^2)$
- $O(n + B)$

- $O(n/B)$

d) ¿Cuál es el tiempo de ejecución para la función `p=ANTERIOR(L,q)` en listas *doblemente* enlazadas por punteros o cursores?

- $O(n)$
- $O(1)$
- $O(n^2)$
- $O(\log_2 n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [8 de Julio de 2004]

[Ej. 1] [Clases (20 puntos)]

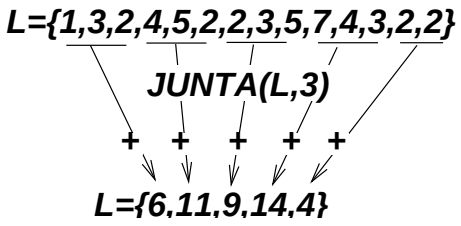
Escibir las primitivas que se indican del TAD `set` por ABB.

```
1  // -*- C++ -*-
2
3  //
4  // Conjunto - implementaci\'on por ABB
5  // ----------------------------------
6  //
7  // Dado el siguiente archivo de cabecera 'abb_set.h' se pide escribir
8  // el correspondiente archivo 'abb_set.cpp' con la implementaci\'on
9  // completa de la clase 'set' declarada a continuaci\'on.
10 //
11 // NOTA: asumir que los bloques marcados con  { /* --- */ }'
12 // significan que la funci\'on ya est\'a implementada en el archivo
13 // de cabecera (por lo tanto, no es necesario escribirla).
14 //
15
16 #ifndef ABB_SET_H
17 #define ABB_SET_H
18
19
20 #include <cstddef>
21 #include <pair.h>
22
23 #include <btree.h>
24
25 typedef int elem_t;
26
27 typedef btree<elem_t>    tree_t;
28 typedef tree_t::iterator node_t;
29
30 class set;
31
32 class iterator {
33 private:
34   friend class set;
35   node_t  node;   // nodo en que se encuentra el elemento
36   tree_t* bstree; // puntero al ABB del conjunto
37   iterator(node_t n, tree_t& t)
38     : node(n), bstree(&t) { }
39 public:
40   iterator()
41     : bstree(NULL) { }
42   iterator(const iterator& p)
43     : node(p.node), bstree(p.bstree) { }
44   bool operator==(const iterator& p) { return node==p.node; }
45   bool operator!=(const iterator& p) { return node!=p.node; }
46   iterator& operator=(const iterator& p) {
47     node   = p.node; bstree = p.bstree;
48   }
49   node_t next(node_t n) { /* --- */ } // siguiente elemento
```

```
50  };
51
52  typedef std::pair<iterator,bool> pair_t;
53
54  class set {
55  private:
56    tree_t bstree;        // ABB del conjunto;
57    node_t min(node_t n); // encuentra el minimo a partir de un nodo
58    node_t max(node_t n); // encuentra el maximo a partir de un nodo
59  public:
60    set() : { }
61    set(const set& A) : bstree(A.bstree) { }
62    ~set() { }
63    pair_t insert(const elem_t& x);
64    void erase(iterator p) { /* --- */}
65    int erase(const elem_t& x);
66    void clear();
67    iterator find(const elem_t& x);
68    elem_t& retrieve(iterator p);
69    iterator begin();
70    iterator end();
71    int size();
72  };
73
74  #endif // ABB_SET_H
```

[Ej. 2] [Programación (total = 50 puntos)]

a) **[junta (30 puntos)]**
Escribir una función `void junta(list<int>&L,int n);` que dada una lista `L`, agrupa de a `n` elementos dejando su suma (ver figura).
Restricciones: **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



b) **[maximo-par (10 puntos)]**
Escribir una función `int maximo_par(btree<int>&T);` que retorna el máximo de las etiquetas **pares** de un árbol binario. En el caso del árbol `(10 21 (6 (11 12 6) (23 . 3)))` debe retornar 12.
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en el árbol.

c) **[elimina-valor (10 puntos)]**
Escribir una función `voi elimina_valor(queue<int>&C, int);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C = {1,3,5,4,2,3,7,3,5}`, después de `elimina_valor(C,3)` debe quedar `C = {1,5,4,2,7,5}`. *Sugerencia: Usar una estructura auxiliar lista o cola.*
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

[Ej. 3] [operativos (total = 20 puntos)]

a) **[rec-arbol (5 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, D, E, R, S, Q, A, B\}$,
- `ORD_POST` $=\{D, Q, R, A, B, S, E, C\}$.

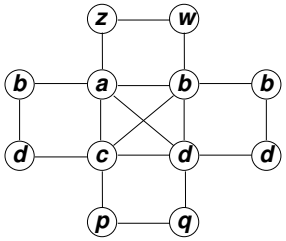
b) **[huffman (5 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PIQUETERO`
$P(P) = 0{,}2, P(I) = 0{,}2, P(Q) = 0{,}2, P(U) = 0{,}1, P(E) = 0{,}1, P(T) = 0{,}1, P(R) = 0{,}05, P(O) = 0{,}05$
Calcular la longitud promedio del código obtenido.

c) **[misc-arbol (5pt)]:** Dado el árbol `(p (q r s) (t (u v)))`,

1) Cuál es el nodo que está a la vez a la izquierda de `v` y a la derecha de `r`?
2) Particione el árbol con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda

d) **[colorear-grafo (5 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n+B)$
- ☐ $O(n/B)$

¿Cuál es el tiempo de ejecución para la función `p--` en listas *doblemente* enlazadas por punteros o cursores?

- ☐ $O(n)$
- ☐ $O(1)$
- ☐ $O(n^2)$
- ☐ $O(\log_2 n)$

Sea `L` una lista conteniendo los elementos $(4, 7, 6, 5, 3,)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = ++q;
p = L.erase(p);
```

¿Cuál de las opciones es verdadera?

- ☐ `*p=*q=7`.
- ☐ `*p=5`, `*q=7`.
- ☐ `*p=5`, `q` es inválido.
- ☐ `p` y `q` son inválidos.

¿Como afectan los elementos suprimidos (`deleted`) a la eficiencia en una tabla de dispersión cerrada? Es decir, ?Como es la eficiencia de una tabla A con 50 % indefinidos y 40 % suprimidos con respecto a una tabla B con 50 % indefinidos y sin suprimidos?

- ☐ Equivalen a un elemento ocupado (por lo B es más eficiente que A).
- ☐ Equivalen a un elemento indefinido (`undef`) (por lo tanto B es menos eficiente que A).
- ☐ La eficiencia de A es igual a la de B.
- ☐ No puede haber elementos suprimidos en una tabla de dispersión.

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [29 de Julio de 2004]

[Ej. 1] [Clases (20 puntos)]

Escribir las primitivas que se indican del TAD `list<>` `insert(x,p)`, `erase(p)`, `retrieve(p)/*p`, `next()/p++`, `list()` y `~list()`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [Programación (total = 80 puntos)]

a) **[corrida (40 puntos)]**
En ciertas aplicaciones interesa detectar las corridas ascendentes en una lista de números $L = [\ a_1\ a_2\ ,\ ...,\ a_n\ ]$, donde cada corrida ascendente es una sublista de números consecutivos a_i, a_{i+1}, ..., a_{i+k} de modo tal que $a_j \leq a_{j+1}$ para $j = i, ..., i+k-1$, y es máxima en el sentido que si $i > 0$ entonces $a_{i-1} > a_i$, y si $i + k < n$ entonces $a_{i+k} > a_{i+k+1}$. Por ejemplo, si $n = 10$ y la lista es L = [0,5,5,9,4,3,9,6,5,5,2], entonces hay $h = 6$ corridas ascendentes, a saber, [0,5,5,9], [4], [3,9], [6], [5,5] y [2]. Usando las operaciones del TAD lista, escribir una función
`int ascendente(list<t> &l,vector< list<t> > &v)` en la cual, dada una lista de enteros `l`, almacene cada carrera ascendente como una componente del vector de listas `vector<list <t> > &v`, devolviendo además el número h de carreras ascendentes halladas.

b) **[ord-nodo (20 puntos)]**
Escribir una función predicado `bool ordnodo(tree<int> &A);` que verifica si cada secuencia de hermanos del subárbol del nodo `n` (perteneciente al árbol ordenado orientado `A`) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol `(3 5 (6 1 3) (7 4 5))` debería retornar `true`, mientras que para `(3 9 (6 1 3) (7 4 2))` debería retornar `false`, ya que las secuencias de hermanos `(9 6 7)` y `(4 2)` NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

c) **[es-completo (20 puntos)]**
Escribir una función predicado `bool es_completo(btree<int> &)`, la cual retorna verdadero si el árbol binario es completo.

[Ej. 3] [operativos (total = 80 puntos)]

a) **[rec-arbol (30 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, Z, Q, R, A, M, P, K, L, T\}$,
- `ORD_POST` $=\{Z, A, P, M, K, L, R, T, Q, C\}$.

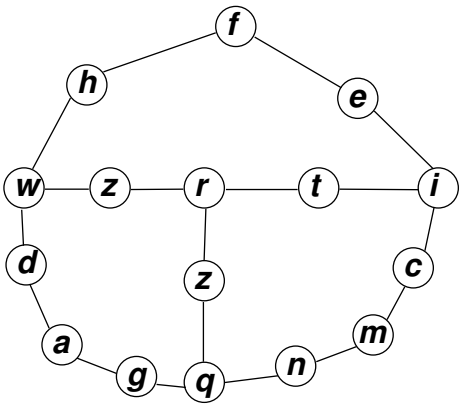
b) **[huffman (30 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario y encodar la palabra `PAPAFRITA`
$P(P) = 0{,}05, P(A) = 0{,}2, P(F) = 0{,}1, P(R) = 0{,}05, P(I) = 0{,}2, P(T) = 0{,}15, P(Q) = 0{,}15, P(S) = 0{,}1$
Calcular la longitud promedio del código obtenido.

c) **[misc-arbol (10pt)]:** Dado el árbol `(c q (t (r u (v z))))`,

1) Cuál es el nodo que está a la vez a la izquierda de `v` y no es descendiente de `r`?
2) Particione el árbol con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

d) **[colorear-grafo (10 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] [**Preguntas (total = 20 puntos, 5puntos por pregunta)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(a b (c d e))`. ¿Cuál de las opciones es verdadera?

- ☐ Tiene dos raíces y altura 2.
- ☐ Tiene 3 hojas y altura 3.
- ☐ Tiene 3 hojas y altura 2.
- ☐ Tiene 4 nodos a profundidad 1 y 3 hojas.

¿Cuál es el tiempo de ejecución para la función `find(x)` en conjuntos implementados con árboles binarios de búsqueda?

- ☐ Caso promedio $O(n)$, peor caso $O(n)$.
- ☐ Siempre $O(n \log n)$.
- ☐ Caso promedio $O(n \log n)$, peor caso $O(n^2)$.
- ☐ Caso promedio $O(n \log n)$, peor caso $O(n)$.

Sea `L` una lista conteniendo los elementos $(1, 3, 4, 2, 5, 6)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = L.erase(q);
p++;
q = p;
q++;
```

¿Cuál de las siguientes opciones es verdadera?

- ☐ `*p=2, *q=5`.
- ☐ `*p=2`, `q` es inválido.
- ☐ `*p=4`, `*q=2`
- ☐ `*p=4`, `q` es inválido.

¿Cuál es el tiempo de ejecución para intersección de conjuntos por vectores de bits? (N es el número de elementos en el conjunto universal, n el número de elementos en el conjunto dado.)

- ☐ $O(N \log N)$
- ☐ $O(n)$
- ☐ $O(N)$
- ☐ $O(n \log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [1 de Octubre de 2004]

[Ej. 1] [Clases (20 puntos)]

Escribir los siguientes métodos del TAD `btree`: `insert(p,x)`, `erase(p)`, `find(x)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [programacion (total = 80 puntos)]

a) **[encuentra (35 puntos)]** Escribir una función
`bool encuentra(list<int> &L1, list<int> &L2, list<int> &indx);` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `indx` los índices de los elementos de `L2` que forman `L1`, si no `indx` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `encuentra` debe retornar `true`, e `indx`=$\{0, 1, 4, 5, 7, 9\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `indx`=$\{\}$. Nota: Los índices en `indx` deben ser estrictamente crecientes. *Restricciones:* No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$.

b) **[encuentra-suma (35 puntos)]** Escribir una función
`tree<double>::iterator encuentra_suma(double s, tree<double> &A);` que retorna el nodo `m` del árbol orientado `A` tal que la suma de las etiquetas de todos los nodos descendientes de `m` (incluyendo a `m`) es `s`. Si no existe un tal nodo, entonces debe retornar Λ. Si hay varios nodos que cumplen la condición basta con que retorne uno de ellos. Por ejemplo, para `A=(10 21 (6 (8 7) (11 (12 3) 6) (4 6 5)))`

- `encuentra_suma(32,A)` retorna el nodo 11,
- `encuentra_suma(15,A)` puede retornar el nodo 8, el 4 o el 12.
- `encuentra_suma(9,A)` retorna Λ

Restricciones: El algoritmo debe ser $O(n)$ donde n es el número de nodos en el árbol.

c) **[reordena-pila (10 puntos)]** Escribir una función
`void reordena(stack<int> &P,bool (*pred)(int));` que reordena los elementos de una pila de tal forma que quedan los que no satisfacen el predicado `pred()` en el fondo y los que si lo satisfacen arriba. Por ejemplo si $P = \{1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9\}$ y `pred()` es `bool par(int x) { return x%2==0; }` entonces debe quedar $P = \{4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9\}$. *Restricciones:* el algoritmo debe ser *estable*, es decir los elementos que satisfacen `pred()` deben quedar en el mismo orden entre sí, y lo mismo para los que no satisfacen `pred()`. Se pueden usar dos estructuras auxiliares (listas, pilas o colas).

[Ej. 3] [operativos (total = 80 puntos)]

a) **[rec-arbol (40 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

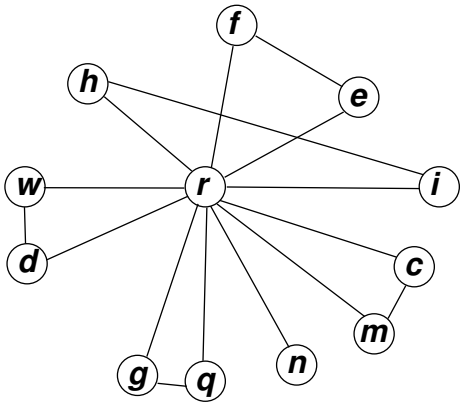
- `ORD_PRE` $=\{A, B, C, E, F, Z, Y, W, G, D\}$,
- `ORD_POST` $=\{B, E, Z, Y, W, F, G, C, D, A\}$.

b) **[misc-arbol (20pt)]:** Dado el árbol `(b t (u (z x y) w))`,

1) Cuál es el nodo de profundidad 2 que está a la izquierda de `w`?
2) Particione el árbol con respecto al nodo `u`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

c) **[colorear-grafo (20 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] **[Preguntas (total = 20 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(q e (r w) s t)`. ¿Cuál de las opciones es verdadera?

- ☐ Tiene 4 nodos a profundidad 1 y 5 hojas.
- ☐ Tiene dos raíces y 4 hojas.
- ☐ Tiene 4 nodos a profundidad 1 y 4 hojas.
- ☐ Tiene 5 nodos a profundidad 1 y 4 hojas.

¿Cuál es el tiempo de ejecución de la operación de *re-heap* en montículos?

- ☐ Caso promedio $O(\log n)$.
- ☐ Siempre $O(\log n)$.
- ☐ Caso promedio $O(n \log n)$.
- ☐ Siempre $O(n \log n)$.

Dado el árbol binario `(z (a b q) r)`, ¿cuál de las siguientes opciones es verdadera?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n + B)$
- ☐ $O(n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [2 de Diciembre de 2004]

[Ej. 1] [**clases (20 puntos)**] Escribir los siguientes métodos del TAD Conjunto, implementado con árboles binarios de búsqueda `set<>`: `erase(n)`, `find(x)`, `insert(x)` (donde `n` es nodo y `x` elemento). Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [**programacion (total = 80 puntos)**]

a) [**merge-dict (total = 40 puntos)**] Implemente una función

```
void merge_map(const map< string, list<int> > &A,
               const map< string, list<int> > &B,
                     map< string, list<int> > &C)
```

que a partir de los diccionarios A y B construya un diccionario C de acuerdo a las siguientes reglas:

- Si una clave `key` esta contenida en A o B pero no en ambos, entonces C debe contener dicha clave y su valor asociado debe ser la lista asociada a `key` en A o B pero sin repeticiones.
- Si una clave `key` esta contenida en A y B a la vez, entonces C debe contener dicha clave y su valor asociado debe ser una lista que contenga todos los elementos de las listas asociadas a `key` en A y B, pero sin repeticiones (es decir, su unión como conjunto, $C[key] \leftarrow A[key] \bigcup B[key]$).

Por ejemplo, dados:

```
A = { 'XX' : [3,3,1,2,2],       B = { 'YY' : [3,3,4,5,8],
      'YY' : [5,5,4]     }             'ZZ' : [1,1,9]     }
```

se debe obtener:

```
C = { 'XX' : [3,1,2],
      'YY' : [5,4,3,8]
      'ZZ' : [1,9]        }
```

b) [**intersect-tree (total = 40 puntos)**] Dados dos árboles binarios `A` y `B`, escribir una función `void intersect_tree(btree<int>&A, btree<int>&B, btree<int>&B)` que devuelve en `C` la `intersección` entre los dos árboles, es decir áquellos nodos que (estructuralmente) están en los dos árboles y que contienen el mismo valor. Por ejemplo, si `A=(8 (3 6 (7 10 11)) (5 4 .))` y `B=(8 (3 6 (9 . 3)) (5 7 .))` entonces `C=(8 (3 6.) 5)`.

[Ej. 3] [**operativos (total=80pts) - LIBRES**]

a) [**abb (30 ptos)**] Dados los enteros $\{6, 5, 8, 2, 10, 7, 3, 1, 9, 15, 0\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 6, 3 y 7.

b) [**misc-arbol (20pt)]:** Dado el árbol binario `(z (w (x k .) (y m n)) p)`,

1) Dibuje el árbol correspondiente.
2) De los nodos que están a la izquierda de *y* ¿Cuál es el que está a mayor profundidad?
3) Particione el árbol con respecto al nodo `w`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.
4) ¿ Es completo?

c) [**heap-sort (30 ptos)**] Dados los enteros $\{15, 14, 17, 11, 19, 6, 12\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 4] [**Preguntas (total = 20 puntos, 5puntos por pregunta) - LIBRES**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(5 7 (8 6 9))`. Después de hacer:

```
n = D.find(7);
n++;
n = n.lchild();
n = n.lchild();
n = D.insert(n,2);
```

¿Cuál de las opciones es verdadera?

- ☐ `(5 7 (8 2 6 9))`
- ☐ Da un error.
- ☐ `(5 7 (8 6 2 9))`
- ☐ `(5 7 (8 (6 2) 9))`

¿Cuál es el tiempo de ejecución de la operación de sort en montículos?

- ☐ $O(n \log n)$ sólo en el caso promedio.
- ☐ Siempre $O(n \log n)$.
- ☐ $O(\log n)$ sólo en el caso promedio.
- ☐ Siempre $O(\log n)$.

¿Cuál es el tiempo de ejecución de `splice(p,q)` para árboles?

- ☐ ...$O(n)$
- ☐ ...$O(1/n)$
- ☐ ...$O(1)$
- ☐ ...$O(\log_2 n)$

Sea un tabla de dispersión cerrada con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de una inserción exitosa ($\alpha = n/B$) en promedio

- ☐ $\alpha/(1-\alpha)$
- ☐ $1/(1-\alpha)$
- ☐ $\log \alpha/(1-\alpha)$
- ☐ cte

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [16 de Diciembre de 2004]

[Ej. 1] **[clases (20 puntos)]** Escribir los siguientes métodos del TAD Conjunto, implementado con listas ordenadas `set<>`: `erase(n)`, `find(x)`, `insert(x)` (donde `n` es nodo y `x` elemento). Escribir la función `set_difference(set<T> &A,set<T> &B,set<T> &C);`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] **[programacion (total = 80 puntos)]**

a) **[separa (total = 40 puntos)]** ... Implemente una función

```
void separa(const map<string,int> &M, list<string> &L,
            queue<string> &Q, stack< pair<string,int> > S&);
```

que a partir del diccionario $M : key \rightarrow value$ construya una lista L, una cola Q y una pila S de acuerdo a las siguientes reglas:

- Si el valor `value` (*int*) es **negativo**, insertar la clave `key` (*string*) de la lista L en alguna posición (por ejemplo, `end()`).
- Si el valor `value` es **cero**, insertar la clave `key` en la cola Q.
- Si el valor `value` (*int*) es **positivo**, insertar un par (`pair<string,int>(key,value)`) en la pila S.

Por ejemplo, dado:

```
M = { 'A' : -2,   'B' : -1,   'C' :  0,   'D' :  0, 'E' :  1,
      'F' :  2,   'G' :  3}
```

se debe obtener:

```
L = ['A','B'],    Q = ['C','D'],   S = [('G', 3),('F', 2),('E',1)]
```

b) **[es-camino (total = 40 puntos)]** Implemente una función

```
bool es_camino(tree<int> &t, tree<int>::iterator n,
       list<int> &l, list<int>::iterator q);
bool es_camino(tree<int> &t, list<int> &l); // wrapper
```

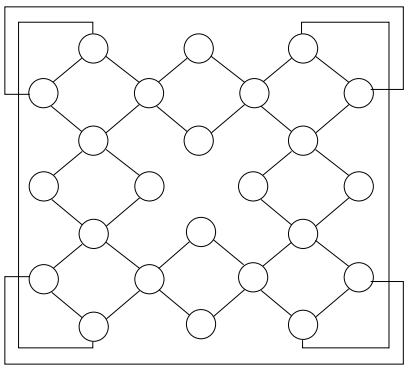
que determina si los elementos en la lista `l` son un camino en el árbol `t`. Por ejemplo,

```
t=(5 7 (8 12 (12 16 11 7) 15) 9) ,
l1={5,8,12,11},      es_camino(t,l1) -> true
l2={5,8,12,11,3},    es_camino(t,l2) -> false
l3={5,8},            es_camino(t,l3) -> true
```

[Ej. 3] **[operativos (total=80pts) - LIBRES]**

a) **[colorear-grafo (30 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



b) **[reconstruir-arbol (30 puntos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, W, T, S, X, L, M, N, P, Q\}$.
- `ORD_POST` $=\{S, T, W, N, P, Q, M, L, X, Z\}$.

c) **[arbol-expr (20 ptos)]** Escribir el árbol que corresponde a la expresión matemática siguiente. Dar la notación prefija y postfija.

$$\frac{(e - hp)(q + r/s)}{t + u + v} \tag{1}$$

[Ej. 4] [Preguntas (total = 20 puntos, 5puntos por pregunta) - LIBRES] Responder según el sistema "*multiple choice*", es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) Dadas las funciones

- $T_1(n) = 3n^2 + 7n!$,
- $T_2(n) = 8n^2 + 0.5\sqrt{n}$,
- $T_3(n) = 0.3 \log n + 2!$ y
- $T_4(n) = 4^n + 2/n!$

decir cuál de los siguientes ordenamientos es el correcto

☐ $T_4 < T_1 < T_2 < T_3$
☐ $T_3 < T_2 < T_4 < T_1$
☐ $T_2 < T_3 < T_4 < T_1$
☐ $T_4 < T_3 < T_2 < T_1$

---

b) Sea una lista `L=(6,8,10,12)`. Después de hacer

```
p = L.begin(); p++; p++;
p = L.erase(p); p++;
```

¿Que retorna `*p` ?

☐ ... retorna 6
☐ ... produce un error
☐ ... retorna 12
☐ ... retorna 8

---

c) ¿Cuál es el número de niveles en un árbol binario lleno (todos sus niveles están completos) en función del número n de nodos en el árbol?

☐ ... $O(1)$
☐ ... $O(\log n)$
☐ ... $O(n)$
☐ ... $O(n \log n)$

---

d) ¿Cuál es el número de *intercambios* en el método de clasificación por selección?

☐ ... $O(1)$
☐ ... $O(n)$
☐ ... $O(n^2)$
☐ ... $O(\log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 1A. [27 de abril de 2004]

[Ej. 1] **[Clases (20 puntos)]** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores. Las funciones a implementar son `insert(p,x)`, `erase(p)`, `erase(p,q)`, `clear()`, `begin()`, `end()`, `next(p)`, `retrieve(p)`. Observaciones:

- Escribir tanto las declaraciones como las funciones (archivos `.h` y `.cpp`).
- Incluir las definiciones de tipo (`typedef`) y clases auxiliares necesarias.
- Se puede escribir la interface avanzada (con templates, clases anidadas, sobrecarga de operadores).

[Ej. 2] **[Programación (total = 60 puntos)]**

a) **[anagrama (40 puntos)]** Escribir una función `bool anagrama_p(list<int> &L1,list<int> &L2);` que retorna verdadero sólo si `L1` y `L2` tienen la misma cantidad de elementos y los elementos de `L1` son una permutación de los de `L2`. Por ejemplo, si `L1=`$(1, 23, 21, 4, 2, 3, 0)$ y `L2=`$(21, 1, 3, 2, 4, 23, 0)$ entonces `anagrama_p(L1,L2)` debe retornar `true`, mientras que si `L1=`$(1, 3, 5)$ y `L2=`$(4, 5, 4)$ debe retornar `false`. En el caso de que haya elementos repetidos estos tienen que estar el mismo número de veces en las dos listas. Se sugiere el siguiente algoritmo: Para cada elemento de `L1` se recorre `L2`. Si el elemento está en `L2` entonces se suprime y se continúa con el siguiente elemento, hasta recorrer todos los elementos de `L1`. Si el elemento no está en `L2` entonces directamente se retorna `false`.
Restricciones:

- Usar la interfase STL para listas.
- No usar el operador `--`.
- No usar ninguna estructura auxiliar ni otros algoritmos de STL.
- El algoritmo debe ser $O(n^2)$.
- El algoritmo puede ser *"destructivo"* (elimina elementos de los contenedores).
- Prestar a no usar posiciones inválidas al iterar sobre las listas.

b) **[flota-pares (15 puntos)]** Escribir una función `void flota_pares(stack<int> &P);` que reordena los elementos de una pila `P` de tal manera que los pares quedan arriba de los impares. Los elementos pares deben quedar en el mismo orden entre sí, y lo mismo para los impares. Por ejemplo, si `P=`$(4, 17, 9, 7, 4, 2, 0, 9, 2, 17)$ entonces después de hacer `flota_pares(P)` debe quedar `P=`$(4, 4, 2, 0, 2, 17, 9, 7, 9, 17)$. Usar dos pilas auxiliares.
Restricciones:

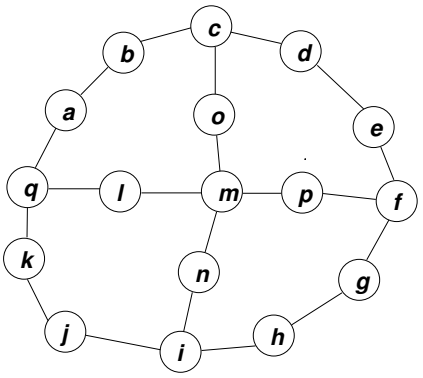
- Usar la interfase STL para pilas.
- No usar más estructuras auxiliares que los indicados ni otros algoritmos de STL.
- El algoritmo debe ser $O(n)$.

c) **[cum-sum-cola (15 puntos)]** Escribir una función `void cum_sum(queue<int> &Q)` que modifica a `Q` dejando la suma acumulada de los elementos, es decir, si los elementos de `Q` antes de llamar a `cum_sum(Q)` son $(a_0, a_1, \ldots, a_{n-1})$, entonces después de llamar a `cum_sum(Q)` debe quedar `Q=`$(a_0, a_0 + a_1, \ldots, a_0 + a_1 + \ldots + a_n)$. Por ejemplo, si `Q=`$(1, 3, 2, 4, 2)$ entonces después de hacer `cumsum(Q)` debe quedar `Q=`$(1, 4, 6, 10, 12)$. Usar una cola auxiliar.
Restricciones:

- Usar la interfase STL para colas (`clear()`, `front()`, `pop()`, `push(T x)`, `size()`, `empty()`).
- No usar más estructuras auxiliares que la indicada ni otros algoritmos de STL.

- El algoritmo debe ser $O(n)$.

[Ej. 3] **[color-grafo (10 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido.



[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) El tiempo de ejecución del operador `--` para listas *simplemente* enlazadas es ... (n es el número de elementos en la lista)

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$

---

b) El tiempo de ejecución de la función `clear()` para listas implementadas por punteros es ... (n es el número de elementos en la cola)

- ☐ ... $O(\log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(n)$

---

c) Sea `L` una lista conteniendo los elementos $(1, 3, 5)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q,r;
p=L.begin();
q=p++;
r=L.erase(p);
```

¿Cuál de las opciones es verdadera?

- ☐ `*p` es 3, `*r` es 5, `*q` es 1.
- ☐ `p` es inválido, `*r` es 5, `*q` es 3.
- ☐ `p` es inválido, `*r` es 5, `*q` es 1.
- ☐ `p` es inválido, `*r` es 5, `q` es inválido.

---

d) Considere el siguiente código:

```
map<int,int>M;
M[3] = 7;
map<int,int>::iterator p = M.find(3);
p->second = 9;
```

¿Cuál es el valor de `M[3]`?

- ☐ 0. (El valor por defecto generado por una asignación accidental).
- ☐ Da un error.
- ☐ 9
- ☐ 7

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 2A. [27 de abril de 2004]

[Ej. 1] **[Clases (20 puntos)]** Escribir la implementación en `C++` del `TAD CORRESPONDENCIA` (clase `map`) implementado por contenedores lineales (listas o vectores, ordenados o no ordenados). Las funciones a implementar son `find(key)`, `insert(key,val)`, `retrieve(key)`, `erase(p)`, `key(p)`, `value(p)`, `begin()`, `end()`, `clear()`. Observaciones:

- Escribir tanto las declaraciones como las funciones (archivos `.h` y `.cpp`).
- Incluir las definiciones de tipo (`typedef`) y clases auxiliares necesarias.
- Se puede escribir la interface avanzada (con templates, clases anidadas, sobrecarga de operadores).

[Ej. 2] **[Programación (total = 60 puntos)]**

a) **[mayorar (30 puntos)]** Escribir una función `void mayorar(list<int> &L1,list<int> &L2);` que modifica las listas `L1` `L2` de tal manera que si a_{1j}, a_{2j} son los elementos de `L1` y `L2` *antes de aplicar la función* y a'_{1j}, a'_{2j} los elementos *después* de aplicar la función, entonces $a'_{1j} = \text{máx}(a_{1j}, a_{2j})$, $a'_{2j} = \text{mín}(a_{1j}, a_{2j})$. Si las listas no tienen la misma longitud, entonces los elementos restantes quedan inalterados. Ejemplo: si `L1=`$(14, 0, 6, 13, 11, 12, 3, 17, 14, 18)$ y `L2=`$(6, 4, 4, 11, 12, 15, 8, 17, 18, 11, 23, 1, 2, 5, 15)$ entonces después de hacer `mayorar(L1,L2)` debe quedar `L1=`$(14, 4, 6, 13, 12, 15, 8, 17, 18, 18)$ y `L2=`$(6, 0, 4, 11, 11, 12, 3, 17, 14, 11, 1, 2, 5, 15)$. Se sugiere el siguiente algoritmo: Recorrer ambas listas con dos posiciones e ir intercambiando los elementos si $a_{1j} < a_{2j}$.
Restricciones:

- Usar la interfase STL para listas.
- No usar el operador `--`.
- No usar ninguna estructura auxiliar.
- El algoritmo debe ser $O(n)$.
- Prestar a no usar posiciones inválidas al iterar sobre las listas.

b) **[creciente (15 puntos)]** Escribir una función `void creciente(queue<int> &Q)` que elimina elementos de `Q` de tal manera de que los elementos que quedan estén ordenados en forma creciente. Por ejemplo, si `P=`$(5, 5, 9, 13, 19, 17, 16, 20, 19, 21)$, entonces después de hacer `creciente(Q)` debe quedar `P=`$(5, 5, 9, 13, 19, 20, 21)$. Usar una cola auxiliar. Se sugiere utilizar el siquiente algoritmo: Ir eliminando los elementos de la pila `Q` y ponerlos en la cola auxiliar `Q2` sólo si el elemento es mayor o igual que el máximo actual. Finalmente volver a pasar todos los elementos de `Q2` a `Q`. **Restricciones:**

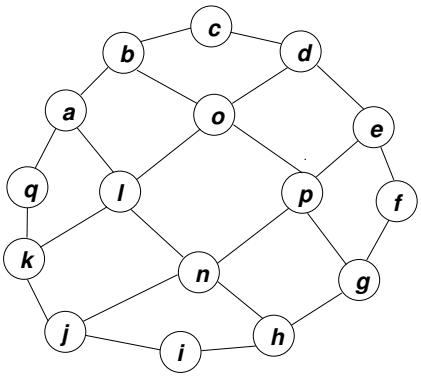
- Usar la interfase STL para colas.
- No usar más estructuras auxiliares que la indicada ni otros algoritmos de STL.
- El algoritmo debe ser $O(n)$.

c) **[cum-sum-pila (15 puntos)]** Escribir una función `void cum_sum(stack<int> &P)` que modifica a `P` dejando la suma acumulada de los elementos, es decir, si los elementos de `P` antes de llamar a `cum_sum(P)` son $(a_0, a_1, \ldots, a_{n-1})$, entonces después de llamar a `cum_sum(P)` debe quedar `P=`$(a_0, a_0 + a_1, \ldots, a_0 + a_1 + \ldots + a_n)$. Por ejemplo, si `P=`$(1, 3, 2, 4, 2)$ entonces después de hacer `cumsum(P)` debe quedar `P=`$(1, 4, 6, 10, 12)$. Usar una pila auxiliar.
Restricciones:

- Usar la interfase STL para pilas (`clear()`, `top()`, `pop()`, `push(T x)`, `size()`, `empty()`).

- No usar más estructuras auxiliares que la indicada ni otros algoritmos de STL.
- El algoritmo debe ser $O(n)$.

[Ej. 3] **[color-grafo (10 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido.



[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Sea `L` una lista conteniendo los elementos $(5, 2, 4)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q,r;
p=L.begin();
q=++p;
r=L.erase(p);
```

¿Cuál de las opciones es verdadera?

- ☐ `p` es inválido, `*r` es 2, `q` es inválido.
- ☐ `*p` es 5, `*r` es 4, `*q` es inválido.
- ☐ `p` es inválido, `*r` es 4, `*q` es 5.
- ☐ `p` es inválido, `*r` es 4, `q` es inválido.

---

b) Considere el siguiente código:

```
map<int,int>M;
M[4] = 3;
M[3] = 4;
M.erase(4);
map<int,int>::iterator p = M.find(3);
int val = p->second;
```

¿Cuál es el valor de `val`?

- ☐ Da un error.
- ☐ 4
- ☐ 0. (El valor por defecto generado por una asignación accidental).
- ☐ 3

---

c) El tiempo de ejecución de la función `M.find(key)` para el `TAD CORRESPONDENCIA` (clase `map`) implementado por vectores ordenados con búsqueda binaria es (asumimos que `key` tiene un valor asignado):

- ☐ ... $O(\log_2 n)$
- ☐ ... $O(1)$
- ☐ ... $O(n)$
- ☐ ... $O(n^2)$

---

d) ¿Cuál es el tiempo de ejecución, en el peor caso, para la función `insert(p,x)` en la implementación de listas por arreglos?

- ☐ ... $O(n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log_2 n)$
- ☐ ... $O(1)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

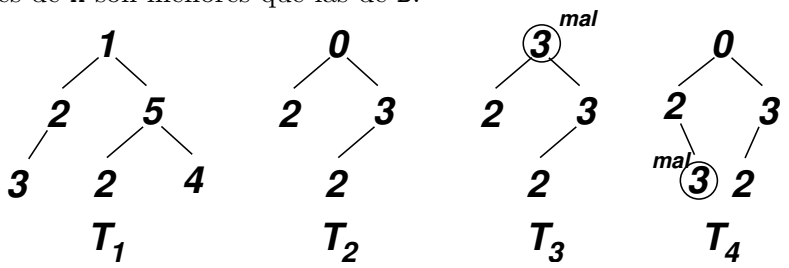
Algoritmos y Estructuras de Datos.
2do Parcial. Tema: 1A. [27 de Mayo de 2004]

[Ej. 1] **[Clases (20 puntos)]** Escribir la implementación en `C++` del `TAD Arbol binario` (clase `btree`) implementado con celdas enlazadas por punteros. Las funciones a implementar son `insert(key,val)`, `retrieve(key)`, `erase(p)`, `begin()`, `end()`, `clear()` además de las clases `iterator` y `cell`. Observaciones:

- Incluir las definiciones de tipo (`typedef`) y clases auxiliares necesarias.
- Se puede escribir la interface avanzada (con templates, clases anidadas, sobrecarga de operadores).

[Ej. 2] **[Programación (total = 60 puntos)]**

a) **[contenido (30 puntos)]**
Escribir una función predicado `bool contenido(btree<int> &A, btree<int> &B);` que retorna verdadero si la estructura del árbol binario `A` esta *"contenido"* dentro de la de `B` y las etiquetas correspondientes de `A` son menores que las de `B`.



Para los árboles de la figura debemos tener .

- `contenido(T2,T1)` retorna `true`
- `contenido(T3,T1)` retorna `false`
- `contenido(T3,T1)` retorna `false`

En los casos que debe retornar false, el nodo que viola la condición está marcado. Se sugiere escribir una función auxiliar recursiva.

b) **[nodos-n (15 puntos)]** Escribir una función `int nodos_n(tree<int> &A, int n);` que cuenta el número de nodos de un árbol ordenado orientado cuya etiqueta es exactamente igual a `n`.

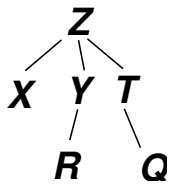
c) **[verifica-par (15 puntos)]** Escribir una función predicado `bool verifica_par(tree<int> &A);` que verifica si las etiquetas de todos los nodos de un árbol ordenado orientado `A` son pares.

[Ej. 3] **[operativos (total = 10 puntos)]**

- **[rec-arbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
  - `ORD_PRE` $=\{Z, A, X, U, V, Q, W, B, R, T\}$,
  - `ORD_POST` $=\{X, U, Q, V, W, A, T, R, B, Z\}$.
- **[arbol-expr (10 ptos)]** Escribir el árbol que corresponde a la expresión $a * (b + c)/(d + z) - x * (y - w)$

[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

¿Cuál es el nodo que está a la izquierda de Q y no es antecesor de R



- ☐ ... Z
- ☐ ... X
- ☐ ... Y
- ☐ ... R

Sea el árbol ordenado orientado `A=(4 7 6 (2 3))`. Si hacemos las operaciones

```
n = A.find(7);
n++; n++;
n = A.insert(n,1);
```

¿Cuál de las siguientes afirmaciones es verdadera?

- ☐ ... Da un error
- ☐ ... Queda `A=(4 7 6 (2 1 3))`
- ☐ ... Queda `A=(4 7 6 (2 3) 1)`
- ☐ ... Queda `A=(4 7 6 1 (2 3))`

Si `A=(3 5 4 (6 7 8))` y `B=(5 3)` entonces ¿cómo quedan `A` y `B` después de hacer...?

```
n=A.find(6);
m=B.begin(); m=m.lchild();
m=m.lchild();
B.splice(m,n);
```

- ☐ ... Da un error.
- ☐ ... `A=(3 5 4)` y `B=(5 (3 (6 7 8)))`
- ☐ ... `A=(3 5 4 6)` y `B=(5 (3 7 8))`
- ☐ ... `A=(3 5 4)` y `B=(5 (3 (6 7 8)))`

¿Cuál de los siguientes árboles binarios es *"completo"*?

- ☐ ... `(1 2 (3 . 7))`
- ☐ ... `(1 2 (3 5 7))`
- ☐ ... `(1 2 (3 5 .))`
- ☐ ... `(1 (2 . 4) (3 5 7))`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

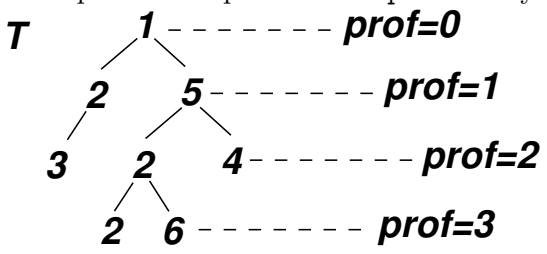
Algoritmos y Estructuras de Datos.
2do Parcial. Tema: 2A. [27 de Mayo de 2004]

[Ej. 1] **[Clases (20 puntos)]** Escribir la implementación en `C++` del `TAD Arbol Ordenado Orientado` (clase `tree`) implementado con celdas enlazadas por punteros. Las funciones a implementar son `insert(key,val)`, `retrieve(key)`, `erase(p)`, `begin()`, `end()`, `clear()` además de las clases `iterator` y `cell`. Observaciones:

- Incluir las definiciones de tipo (`typedef`) y clases auxiliares necesarias.
- Se puede escribir la interface avanzada (con templates, clases anidadas, sobrecarga de operadores).

[Ej. 2] **[Programación (total = 60 puntos)]**

a) **[cant-nodos-prof (30 puntos)]**
Escribir una función `int cant_nodos_prof(btree<int> &A, int prof);` que retorna el número de nodos de una árbol binario `A` que están a profundidad `prof` o mayor.



Para el árbol T de la figura debemos tener

- `cant_nodos_prof(T,0)` retorna 8
- `cant_nodos_prof(T,1)` retorna 7
- `cant_nodos_prof(T,2)` retorna 5
- `cant_nodos_prof(T,3)` retorna 2
- `cant_nodos_prof(T,4)` retorna 0
- `cant_nodos_prof(T,5)` retorna 0

Se sugiere escribir una función auxiliar recursiva.

b) **[algun-par (15 puntos)]** Escribir una función predicado `bool algun_par(tree<int> &A);` que verifica si *al menos una* de las etiquetas de un árbol ordenado orientado `A` es par.

c) **[nodos-mayores-que (15 puntos)]** Escribir una función `int nodos_mayores_que(tree<int> &A, int m);` que cuenta el número de nodos de un árbol ordenado orientado cuya etiqueta es mayor o igual que `m`.

[Ej. 3] **[operativos (total = 10 puntos)]**

- **[rec-arbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
  - `ORD_PRE` $=\{W, A, B, D, E, H, J, C, F, G\}$,
  - `ORD_POST` $=\{D, J, H, E, B, F, G, C, A, W\}$.
- **[arbol-expr (10 ptos)]** Escribir el árbol que corresponde a la expresión $(a - q) * (r + z)/(w - f) + d * e/(x - y)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

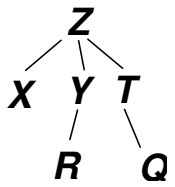
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

[Ej. 4] [Preguntas (total = 10 puntos, 2.5puntos por pregunta)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

¿Cuál es el nodo que está a la derecha de X y es antecesor propio de Q?



- ☐ ... X
- ☐ ... Z
- ☐ ... Y
- ☐ ... R

Sea el árbol ordenado orientado `A=(5 3 (4 7 6) 2)`. Si hacemos las operaciones

```
n = A.find(3);
n++;
n = A.insert(n,5);
```

¿Cuál de las siguientes afirmaciones es verdadera?

- ☐ ... Da un error
- ☐ ... Queda `A=(5 3 (4 5 7 6) 2)`
- ☐ ... Queda `A=(5 3 5 (4 7 6) 2)`
- ☐ ... Queda `A=(5 3 (4 7 6) 5 2)`

Si `A=(2 (3 (6 4 2)) 5)` y `B=(1 7 9)` entonces ¿cómo quedan `A` y `B` después de hacer...?

```
n=A.find(6);
m=B.begin(); m=m.lchild();
m++;
m=m.lchild();
B.splice(m,n);
```

- ☐ ... `A=(2 (3 (6 4 2)) 5)` y `B=(1 7 (9 (6 4 2)))`
- ☐ ... `A=(2 3 (5 6))` y `B=(1 7 (9 4 2)))`
- ☐ ... `A=(2 3 5)` y `B=(1 7 (9 (6 4 2)))`
- ☐ ... Da un error.

¿Cuál de los siguientes árboles binarios es *"completo"*?

- ☐ ... `(5 (6 7 8) .)`
- ☐ ... `(5 (6 7 8) 3)`
- ☐ ... `(5 (6 7 8) (3 . 1))`
- ☐ ... `(5 (6 7 8) (3 4 .))`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. Tema: 1A. [24 de Junio de 2004]

[Ej. 1] [Clases (20 puntos)]

Dado el siguiente archivo de cabecera `set.h` para el TAD conjunto por vectores de bits, se pide escribir la implementación de las siguientes funciones del correspondiente archivo `set.cpp`

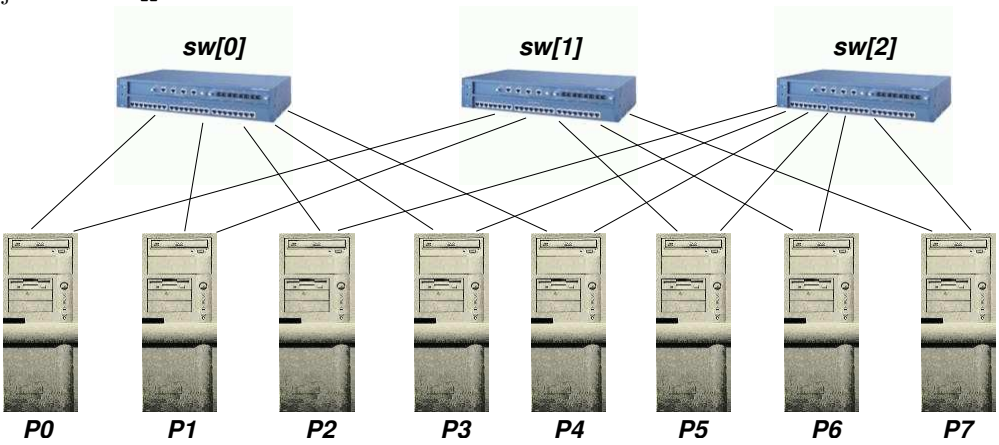
- `insert(const elem_t& x)`
- `find(const elem_t& x)`
- `erase(const elem_t& x)`
- `set_intersection(set &A,set &B,set &C)`
- `set_difference(set &A,set &B,set &C)`

Notas: `N` es la cantidad de elementos del conjunto universal, asúmase que es una variable global ya definida.

```
1  #ifndef SET_H
2  #define SET_H
3
4  #include <vector>
5  #include <pair.h>
6
7  typedef int elem_t;
8  typedef int    (*index_fun)(const elem_t& e); // index   <-- element
9  typedef elem_t (*element_fun)(int i);         // element <-- index
10
11 typedef int iterator_t;
12
13 class set {
14 private:
15   std::vector<bool> v; // vector de bits
16   index_fun   index;   // index   <-- element
17   element_fun element; // element <-- index
18 public:
19   set(index_fun ifun, element_fun efun)
20     : v(N,false), index(ifun), element(efun) { }
21   set(const set& S)
22     : v(S.v), index(S.index), element(S.element) { }
23   iterator_t begin();
24   iterator_t end();
25   iterator_t next(iterator_t p);
26   elem_t retrieve(iterator_t p);
27   std::pair<iterator_t,bool> insert(const elem_t& x);
28   iterator_t find(const elem_t& x);
29   int erase(const elem_t& x);
30   void erase(iterator_t p);
31   void clear();
32   int size();
33
34   friend void set_union(const set &A, const set &B, set &C);
35   friend void set_intersection(const set &A, const set &B, set &C);
36   friend void set_difference(const set &A, const set &B, set &C);
37 };
38 #endif
```

[Ej. 2] [Programación (total = 50 puntos)]

a) **[flat (30 puntos)]**
Se está diseñando una red inerconectada por switches y se desea, para reducir lo más posible la *latencia* entre nodos, que cada par de nodos esté conectado en forma directa por al menos un switch. Sabemos que el número de nodos es `n` y tenemos un `vector< set<int> > sw` que contiene para cada switch el conjunto de los nodos conectados por ese switch, es decir `sw[j]` es un conjunto de enteros que representa el conjunto de nodos inteconectados por el switch j.
Consigna: Escribir una función predicado `bool flat(vector< set<int> > &sw, int n);` que retorna verdadero si cada par de enteros (j, k) con $0 \leq j, k < n$ está contenido en al menos uno de los conjunto en `sw[]`.



En el ejemplo de la figura tenemos 8 nodos conectados via 3 switches y puede verficarse que cualquier par de nodos está conectado en forma directa a través de al menos un switch. Para este ejemplo el vector `sw` sería

$$\texttt{sw[0]} = \{0,1,2,3,4\}, \quad \texttt{sw[1]} = \{0,1,5,6,7\}, \quad \texttt{sw[2]} = \{2,3,4,5,6,7\} \tag{1}$$

Por lo tanto `flat(sw,8)` debe retornar `true`. Por otra parte si tenemos

$$\texttt{sw[0]} = \{0,2,3,4\}, \quad \texttt{sw[1]} = \{0,1,5,7\}, \quad \texttt{sw[2]} = \{2,3,5,6,7\} \tag{2}$$

entonces los pares $(0,6)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,6)$, $(4,5)$, $(4,6)$ y $(4,7)$ no están conectados en forma directa y `flat(sw,8)` debe retornar `false`.
Sugerencia 1: Recorrer todos los pares de valores (j, k) y para cada par recorrer todos los conjuntos en `sw[]` hasta encontrar uno que contenga al par.
Sugerencia 2: Puede ser de ayuda el escribir una función auxiliar `bool estan_conectados(sw,j,k)`.

b) **[es-neg (20 puntos)]**
Escribir una función predicado `bool es_neg(set<int> &A,set<int> &B);` que retorna verdadero si el conjunto `B` contiene los elementos de `A`, pero cambiados de signo. Por ejemplo, si $A = \{-5,-3,5,10\}$ y $B = \{-10,-5,3,5\}$ entonces `es_neg(A,B)` debe retornar `true`. Mientras que si $A = \{-5,-3,2,10\}$ y $B = \{-10,-5,4,5\}$, entonces debe retornar `false` ya que el elemento 2 de A y el 4 de B no tienen su negativo en el otro conjunto.
Estrategia 1: Crear un conjunto temporario con los negativos de A y compararlo con B.
Estrategia 2: Recorrer los elementos de A y verificar que su negativo esté en B y viceversa.

[Ej. 3] [operativos (total = 20 puntos)]

a) **[huffman (8 puntos)]:** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `TEMPLATE`
$P(T) = 0{,}3, P(E) = 0{,}2, P(M) = 0{,}15, P(P) = 0{,}15, P(L) = 0{,}05, P(A) = 0{,}05, P(W) = 0{,}05, P(Q) = 0{,}05$
Calcular la longitud promedio del código obtenido.

b) **[abb (8 ptos)]** Dados los enteros $\{13, 7, 20, 2, 3, 10, 5, 4, 1, 12\}$
insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 7, 5 y 4 en ese orden.

c) **[ohash (4 ptos)]** Insertar los enteros (10,5,4,14,4,25,23,12,41) en una tabla de dispersión abierta con función de dispersión `h(x) = x % B` con `B`=10 cubetas.

[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

¿Cuál es el tiempo de ejecución de `insert(x)` en el TAD diccionario por tablas de dispersión abiertas, en el caso promedio?

- ☐ ... $\log(n/B)$
- ☐ ... $O(1 + B/n)$
- ☐ ... $O(1 + n^2/B)$
- ☐ ... $O(1 + n/B)$

¿Cuál de los siguientes árboles *es* un árbol binario de búsqueda?

- ☐ ... `(3 1 (5 4 .))`
- ☐ ... `(3 (1 2 .) (6 5 .))`
- ☐ ... `(3 1 (5 2 .))`
- ☐ ... `(3 (1 . 4) (6 5 .))`

¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por árbol binario de búsqueda, en el peor caso?

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

¿Donde se encuentra el máximo en un árbol binario de búsqueda?

- ☐ ... En un nodo ubicado lo más a la izquierda posible.
- ☐ ... En la raíz
- ☐ ... En un nodo ubicado lo más a la derecha posible.
- ☐ ... En el nodo lo más profundo posible.

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. Tema: 2A. [24 de Junio de 2004]

[Ej. 1] [Clases (20 puntos)]

Dado el siguiente archivo de cabecera `hash_set.h` para el TAD diccionario con tablas de dispersión abiertas, se pide escribir la implementación de las siguientes funciones del correspondiente archivo `set.cpp`

- `erase(iterator_t p)`
- `clear()`
- `insert(const key_t& x)`
- `find(const key_t& x)`
- `erase(const key_t& x)`

```
#ifndef HASH_SET_H
#define HASH_SET_H

#include <list>
#include <vector>
#include <pair.h>

typedef int key_t;
typedef int (*hash_fun)(key_t x);

class hash_set;

class iterator_t {
  friend class hash_set;
 private:
  int bucket;                    // numero de cubeta
  std::list<key_t>::iterator p; // posicion en la lista
  iterator_t(int bucket_a, std::list<key_t>::iterator p_a)
    : bucket(bucket_a), p(p_a) { }
 public:
  iterator_t() { };
  bool operator==(iterator_t q);
  bool operator!=(iterator_t q);
};


class hash_set {
 private:
  int B;       // cantidad de cubetas
  int count;   // cantidad de elementos en el conjunto
  hash_fun h;  // puntero a la funcion de hash
  std::vector< std::list<key_t> > v; // vector de cubetas
 public:
  hash_set(const hash_set& s);
  hash_set(int B_a,hash_fun h_a) : B(B_a), v(B), h(h_a), count(0) { }
  iterator_t begin();
  iterator_t end();
  iterator_t next(iterator_t p);
  key_t retrieve(iterator_t p);
  std::pair<iterator_t,bool> insert(const key_t& x);
  iterator_t find(const key_t& x);
  int erase(const key_t& x);
```

```
43    void erase(iterator_t p);
44    void clear();
45    int size();
46  };
47
48  #endif /* HASH_SET_H */
```

[Ej. 2] [Programación (total = 50 puntos)]

a) **[en-todos (30 puntos)]**
Escribir una función predicado `bool en_todos(vector< set<int> > &v);` que retorna verdadero si existe al menos un elemento que pertenece a todos los conjuntos `v[j]`. Por ejemplo, si

$$\texttt{v[0]} = \{0,2,3,4,5\}, \quad \texttt{v[1]} = \{0,1,5,7\}, \quad \texttt{v[2]} = \{2,3,5,6,7\} \tag{1}$$

entonces `en_todos(v)` debe retornar `true` ya que 5 está en los tres conjuntos. Por el contrario, si

$$\texttt{v[0]} = \{0,2,3,4,5\}, \quad \texttt{v[1]} = \{0,1,7\}, \quad \texttt{v[2]} = \{2,3,5,6,7\} \tag{2}$$

entonces `en_todos(v)` debe retornar `false`.
Sugerencia: generar el conjunto que es la intersección de todos los `v[j]` y finalmente verificar si es vacío o no.

b) **[mediana (20 puntos)]**
Escribir una función `int mediana(list<int> &L);` que retorna la mediana de los valores contenidos en la lista `L`. Recordemos que la mediana de una serie de n valores consiste en el valor que queda en la posicion $n/2$ después de ordenarlos. Por ejemplo, si $L = (3,2,4,-1,0)$ la mediana es 2. Asumir que todos los elementos en L son distintos.
Sugerencia: Insertar los elementos en un conjunto temporario A y después buscar la posición apropiada, recorriéndolo con un iterator. Recordemos que al iterar sobre un conjunto los elementos aparecen en forma ordenada de menor a mayor.

[Ej. 3] [operativos (total = 20 puntos)]

a) **[huffman (8 puntos)]:** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `TEMPLATE`
$P(T) = 0{,}3, P(E) = 0{,}2, P(M) = 0{,}15, P(P) = 0{,}15, P(L) = 0{,}05, P(A) = 0{,}05, P(W) = 0{,}05, P(Q) = 0{,}05$
Calcular la longitud promedio del código obtenido.

b) **[abb (8 ptos)]** Dados los enteros $\{12, 18, 5, 23, 22, 15, 20, 21, 24, 13\}$
insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 18, 20 y 23 en ese orden.

c) **[chash (4 ptos)]** Insertar los enteros (14,27,24,15,34,41,57,67,55,27) en una tabla de dispersión cerrada con `B`=10 cubetas con función de dispersión `h(x) = x % B` y redispersión lineal. Mostrar como queda la tabla después de realizar las inserciones.

[Ej. 4] [Preguntas (total = 10 puntos, 2.5puntos por pregunta)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

¿Donde se encuentra el mínimo en un árbol binario de búsqueda?

- ☐ ... En el nodo lo más profundo posible.
- ☐ ... En un nodo ubicado lo más a la izquierda posible.
- ☐ ... En un nodo ubicado lo más a la derecha posible.
- ☐ ... En la raíz

¿Cuál de los siguientes árboles *es* un árbol binario de búsqueda?

- ☐ ... `(10 (5 . 8) (12 . 9))`
- ☐ ... `(10 (5 . 8) (12 9 .))`
- ☐ ... `(10 (5 . 11) 12)`
- ☐ ... `(10 (5 . 8) 12)`

¿Cuál es el tiempo de ejecución de `find(x)` en el TAD diccionario por tablas de dispersión abiertas, en el caso promedio?

- ☐ ... $O(1 + n/B)$
- ☐ ... $O(1 + B/n)$
- ☐ ... $\log(n/B)$
- ☐ ... $O(1 + n^2/B)$

¿Cual es el tiempo de ejecución para `erase(x)` en el TAD conjunto por árbol binario de búsqueda, en el peor caso?

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [10 de Febrero de 2005]

[Ej. 1] [**clases (20 puntos)**] Escribir el constructor por copia del TAD Arbol Ordenado Orientado (`class tree`) `tree<T>(const tree<T> &TT)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [**programacion (total = 80 puntos)**]

a) [**filter (total = 40 puntos)**] ... Escriba una función
`void filter(list<int> &L,bool (*pred)(int x));`
que elimina de la lista `L` aquellos elementos `x` que no satisfacen el predicado `pred(x)`. Por ejemplo, si `L=`$\{1,3,2,5,4,3,6,7\}$ y `bool impar(int x) { return x%2; }` , entonces `filter(L,impar)` debe dejar sólo los elementos impares, es decir `L=`$\{1,3,5,3,7\}$. *Restricciones:* Los elementos restantes deben quedar en el orden en el que estaban originalmente. El algoritmo debe ser $O(n)$, donde n es el número de elementos en la lista. El algoritmo debe ser *"in-place"*, es decir, no se debe usar ninguna estructura auxiliar.

b) [**cond-depth (total = 40 puntos)**] Implemente una función
`int cond_depth(tree<int> &T, ,bool (*pred)(int x));` que calcula la máxima profundidad de los elementos del árbol que satisfacen `pred()`. Por ejemplo, si `T=(1 (5 2 (9 6)) 3 (7 8))`, y `pred()` es `bool impar(int x) { return x%2; }` , entonces `cond_depth(T,impar)` debe retornar 2, ya que el elemento impar a mayor profundidad es el 9, que está a profundidad 2.

[Ej. 3] [**operativos (total=80pts) - LIBRES**]

a) [**ohash (20 ptos)**] Insertar los enteros (10,5,4,14,4,25,23,12,41,12,32,46) en una tabla de dispersión abierta con función de dispersión `h(x) = x % B` con `B`=10 cubetas. Mostrar como queda la tabla después de realizar las inserciones.

b) [**huffman (20 ptos)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `TSUNAMI`
$P(T) = 0.2, P(S) = 0.2, P(U) = 0.2, P(N) = 0.1, P(A) = 0.1, P(M) = 0.1, P(I) = 0.05, P(Z) = 0.05$
Calcular la longitud promedio del código obtenido.

c) [**heap (20 ptos)**] Dados los enteros {`7,1,6,2,5,3,4,10,12`} ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

d) [**reconstruir-arbol (20 puntos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Z, W, X, Y, T, R, Q, V, L\}$.
- `ORD_POST` $=\{Z, W, X, Y, R, Q, V, L, T, A\}$.

[Ej. 4] [**Preguntas (total = 20 puntos, 5puntos por pregunta) - LIBRES**] Algunas preguntas son según el sistema el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) Dadas las funciones

- $T_1(n) = 8n^3 + 2n + 3^{1/3}$, ‘
- $T_2(n) = \log 2 + 5n^4 + n!$,
- $T_3(n) = \sqrt{n} + n^{1/2} + n^{1/4} + \sqrt{5}n$ y
- $T_4(n) = (n+1)^2 + \log n$

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$

---

b) Sea el AOO `A=(5 7 (9 3 4) 6)`. Después de hacer las operaciones

```
p = A.find(9);
p = p.lchild(); p++;
p = A.insert(p,10);
```

¿ Cuál de las siguientes opciones es verdadera?

- ☐ Da un error al insertar.
- ☐ `A=(5 7 (9 3 4 10) 6)`, `*p=10`.
- ☐ `A=(5 7 (9 3  10 4) 6)`, `*p=10`.
- ☐ `A=(5 7 (9 3  10 4) 6)`, `*p=4`.

---

c) El tiempo de ejecución para el algoritmo de clasificación por montículos es

- $O(n \log n)$ en el mejor caso, $O(n^2)$ en promedio
- siempre $O(n \log n)$
- siempre $O(n^2)$
- a veces $O(n^2)$

---

d) Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- $O(n^2/B)$
- $O((n/B)^2)$
- $O(n + B)$
- $O(n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

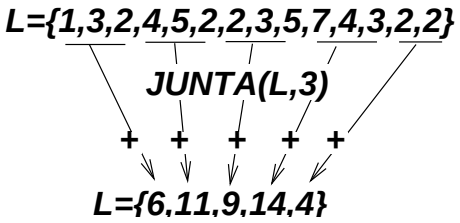
Algoritmos y Estructuras de Datos.
Examen Final. [24 de Febrero de 2005]

[Ej. 1] [Clases (20 puntos)]

Escibir los métodos `insert(x)`, `erase(iterator p)`, `erase(x)` de correspondientes al TAD conjunto (`class set`) implementado por ABB.

[Ej. 2] [Programación (total = 50 puntos)]

a) **[junta (30 puntos)]**
Escribir una función `void junta(list<int>&L,int n);` que dada una lista `L`, agrupa de a `n` elementos dejando su suma (ver figura).
Restricciones: **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



b) **[maximo-par (10 puntos)]**
Escribir una función `int maximo_par(btree<int>&T);` que retorna el máximo de las etiquetas **pares** de un árbol binario. En el caso del árbol `(10 21 (6 (11 12 6) (23 . 3)))` debe retornar 12.
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en el árbol.

c) **[elimina-valor (10 puntos)]**
Escribir una función `voi elimina_valor(queue<int>&C, int);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C = {1,3,5,4,2,3,7,3,5}`, después de `elimina_valor(C,3)` debe quedar `C = {1,5,4,2,7,5}`. *Sugerencia: Usar una estructura auxiliar lista o cola.*
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

[Ej. 3] [operativos (total = 20 puntos)]

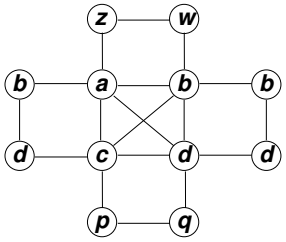
a) **[rec-arbol (5 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $=\{C, D, E, R, S, Q, A, B\}$,
- `ORD_POST` $=\{D, Q, R, A, B, S, E, C\}$.

b) **[huffman (5 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PIQUETERO`
$P(P) = 0{,}2, P(I) = 0{,}2, P(Q) = 0{,}2, P(U) = 0{,}1, P(E) = 0{,}1, P(T) = 0{,}1, P(R) = 0{,}05, P(O) = 0{,}05$
Calcular la longitud promedio del código obtenido.

c) **[misc-arbol (5pt)]:** Dado el árbol `(p (q r s) (t (u v)))`,
1) Cuál es el nodo que está a la vez a la izquierda de `v` y a la derecha de `r`?
2) Particione el árbol con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda

d) **[colorear-grafo (5 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] **[Preguntas (total = 10 puntos, 2.5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n+B)$
- ☐ $O(n/B)$

¿Cuál es el tiempo de ejecución para la función `p--` en listas *doblemente* enlazadas por punteros o cursores?

- ☐ $O(n)$
- ☐ $O(1)$
- ☐ $O(n^2)$
- ☐ $O(\log_2 n)$

Sea `L` una lista conteniendo los elementos $(4, 7, 6, 5, 3,)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = ++q;
p = L.erase(p);
```

¿Cuál de las opciones es verdadera?

- ☐ `*p=*q=7`.
- ☐ `*p=5`, `*q=7`.
- ☐ `*p=5`, `q` es inválido.
- ☐ `p` y `q` son inválidos.

¿Como afectan los elementos suprimidos (`deleted`) a la eficiencia en una tabla de dispersión cerrada? Es decir, ?Como es la eficiencia de una tabla A con 50 % indefinidos y 40 % suprimidos con respecto a una tabla B con 50 % indefinidos y sin suprimidos?

- ☐ Equivalen a un elemento ocupado (por lo B es más eficiente que A).
- ☐ Equivalen a un elemento indefinido (`undef`) (por lo tanto B es menos eficiente que A).
- ☐ La eficiencia de A es igual a la de B.
- ☐ No puede haber elementos suprimidos en una tabla de dispersión.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
Examen Final. [05 de Mayo de 2005]

[Ej. 1] [Clases (30 puntos)]

Escribir los siguientes métodos del TAD `btree`: `insert(p,x)`, `erase(p)`, `find(x)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [Programación (total = 70 puntos)]

a) **[incluido (45 puntos)]**
Escribir un predicado `bool incluido(tree<int>&A,tree<int>&A);`, el cual retorna verdadero si la *estructura* del árbol del nodo `B` está *"incluida"* dentro del árbol `A`, independientemente de las etiquetas de los nodos correspondientes. Por ejemplo, si tenemos los árboles `T1=(z q r (t u v))`, `T2=(x y (p w s))`, `T3=(a c d (e j))`, entonces tenemos que `T3` está incluido en `T1`, pero `T2` no está incluido en `T1`.

b) **[separa (25 puntos)]**
Escribir una función `void separa(queue<int>&Q, queue<int>&Qt,queue<int>&Qf,bool (*pre)(int));` que separa los valores de la cola `Q` en dos colas `Qf`, `Qt`, tal que los valores que satisfacen el predicado `pred()` van a la cola `Qt` mientras que los que no lo satisfacen van a la cola `Qf`. Por ejemplo, si `Q={1,3,2,4,3,2,5}` y el predicado es `bool impar(x) {return x%2; }`, entonces después de `separa(Q,Qt,Qf)` debe quedar `Qt={1,3,3,5}`, `Qf={2,4,2}`.
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original. No se deben usar estructuras auxiliares. El algoritmo debe ser *"estable"*.

[Ej. 3] [LIBRES - operativos (total = 80 puntos)]

a) **[rec-arbol (20 pt)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, A, B, C, J, M, Q, N, P, K, D\}$,
- `ORD_POST` $=\{A, Q, M, P, N, J, K, C, D, B, Z\}$.

b) **[huffman (20 pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `BENEDICTO` $P(B) = 0.2, P(O) = 0.2, P(N) = 0.2, P(E) = 0.1, P(D) = 0.1, P(I) = 0.1, P(C) = 0.05, P(T) = 0.05$ Calcular la longitud promedio del código obtenido.

c) **[misc-arbol (20 pt)]:** Dado el árbol `(z q (r t s) p)`,

1) Cuál es el nodo que está a la vez a la izquierda de `p` y a la derecha de `q` y es antecesor propio de `t`?
2) Particione el árbol con respecto al nodo `t`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda

d) **[heap-sort (20 pt)]** Dados los enteros $\{12, 11, 14, 8, 16, 3, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 4] [LIBRES - preguntas (total = 20 pt, 5pt/preg)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Dadas las funciones $T_1(n) = 0.5n + \sqrt{n}$, $T_2(n) = 0.2n^2 + 2.1 \log n$, $T_3(n) = 3n! + 5.4n^3$ y $T_4(n) = n^{1/2}$ decir cuál de los siguientes ordenamientos es el correcto

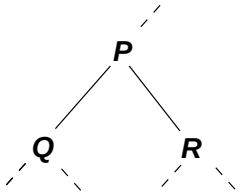
- ☐ $T_4 < T_3 < T_2 < T_1$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_2 < T_1 < T_4 < T_3$
- ☐ $T_3 < T_4 < T_1 < T_2$

b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"heapsort"*) es $O(n \log n)$ (n es el número de elementos a ordenar) ...

- ☐ ... siempre.
- ☐ ... a veces.
- ☐ ... nunca.
- ☐ ... en el mejor caso.

c) El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si P, Q, R son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ $Q + R \leq \infty$.
- ☐ $Q \leq P \leq R$.
- ☐ $P \leq 1/2(Q + R)$.
- ☐ $P \leq Q, R$.



d) ¿Cuál es el tiempo de ejecución del procedimiento de clasificación por *incrementos decrecientes* (*shell-sort*) en el caso promedio?

- ☐ $O(n^{1.3})$
- ☐ $O(n^{1.5})$
- ☐ $O(\log n)$
- ☐ $O(n!)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Final. [7 de Julio de 2005]

[Ej. 1] [clases (30 pts)]

a) **[aoo (30 pts)]** Escribir los siguientes métodos del **TAD árbol ordenado orientado**: `insert(p,x)`, `erase(p)`, `splice(to,from)`.

[Ej. 2] [programacion (46 pts)]

a) **[parc-ord (34 pts)]** Recordemos que un árbol es parcialmente ordenado si dados dos nodos m, n tal que m es hijo de n, entonces el valor de m es mayor o igual al de n.
Consigna: Ecribir un predicado
`bool es_parcialmente_ordenado(tree<int> &T,bool (comp*)(int,int))`, que verifica si el árbol ordenado orientado T es parcialmente ordenado con respecto a la función de comparación `comp()`.

b) **[proper-subset (12 pts)]** Dada una lista de árboles `list< set<int> > L`, escribir una función predicado `bool proper_subset(list< set<int> > &L)`, que determina si los conjuntos de la lista `L` son subconjuntos propios en forma consecutiva. Es decir, si $L = (A_0, A_1, ..., A_{n-1})$, determinar si $A_j \subset A_{j+1}$ para $j = 0, ..., n-2$. (Recordar que $A \subset B$ indica inclusión propia, es decir $A \subseteq B$ y $A \neq B$.)

[Ej. 3] [operativos (16 pts)]

- **[t-exec (4 pts)]**
  Dadas las funciones
  - $T_1(n) = 3\,n^2 + 2\,n^3 + 0.5 \cdot 3^n$,
  - $T_2(n) = 4^5 + 10\,n + 0.2\,n \log_2 n$,
  - $T_3(n) = 3\,n + 5 + 6\sqrt{n}$,
  - $T_4(n) = n^{1.5} + 7! + 4\,\sqrt{n}$.

  ordenarlas de menor a mayor.

  $T_{\square} < T_{\square} < T_{\square} < T_{\square}$
- **[huffmann (4 pts)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `SKYWALKER`: $P(S) = 0.1, P(K) = 0.2, P(Y) = 0.05, P(W) = 0.05, P(A) = 0.05, P(L) = 0.05, P(E) = 0.2, P(R) = 0.3$ Calcular la longitud promedio del código obtenido.
- **[heap-sort (4 pts)]** Dados los enteros $\{23, 15, 12, 6, 8, 13, 10, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
- **[quick-sort (4 pts)]** Dados los enteros $\{3, 1, 11, 14, 18, 17, 7, 2, 11, 20\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.

[Ej. 4] [preguntas (8 pts, 2 por pregunta)]

a) ¿Cómo es el tiempo de ejecución para intercalar dos listas clasificadas de n elementos?

☐ ... $O(1)$
☐ ... $O(n)$

- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

b) ¿Cuál de los siguientes árboles es un árbol binario de búsqueda?

- ☐ ... `(6 (5 (3 . 4 ) .) (9 8 10))`
- ☐ ... `(6 (5 (3 . 4 ) .) (9 (8 2 .) 10))`
- ☐ ... `(6 (5 (3 4 .) .) (9 8 10))`
- ☐ ... `(6 (5 (3 . 4 ) 7) (9 8 10))`

c) ¿Cuál es el tiempo de ejecución para el algoritmo de ordenamiento pot montículos en el peor caso?

- ☐ ... $O(n)$
- ☐ ... $O(\log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n \log n)$

d) ¿Cuál es el tiempo de ejecución para la función `push()` del TAD COLA, implementado sobre listas (asumiendo que el frente de la cola es el principio de la lista)?

- ☐ ... $O(n)$
- ☐ ... $O(\log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n \log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

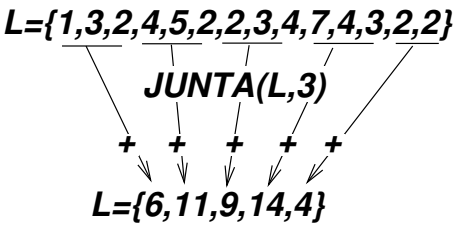
Algoritmos y Estructuras de Datos.
Examen Final. [28 de Julio de 2005]

[Ej. 1] [Clases (30 puntos)]

Escribir las primitivas que se indican del TAD `list<>` `insert(x,p)`, `erase(p)`, `retrieve(p)/*p`, `next()/p++`, `list()` y `~list()`.

[Ej. 2] [Programación (total = 70 puntos)]

a) **[junta (25 puntos)]**
Escribir una función `void junta(list<int>&L,int n);` que dada una lista `L`, agrupa de a `n` elementos dejando su suma (ver figura).
Restricciones: **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



b) **[ord-nodo (25 puntos)]**
Escribir una función predicado `bool ordnodo(tree<int> &A);` que verifica si cada secuencia de hermanos del subárbol del nodo `n` (perteneciente al árbol ordenado orientado `A`) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol `(3 5 (6 1 3) (7 4 5))` debería retornar `true`, mientras que para `(3 9 (6 1 3) (7 4 2))` debería retornar `false`, ya que las secuencias de hermanos `(9 6 7)` y `(4 2)` NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

c) **[es-completo (20 puntos)]**
Escribir una función predicado `bool es_completo(btree<int> &A);` la cual retorna verdadero si el árbol binario es completo.

[Ej. 3] [operativos (total = 80 puntos)]

a) **[rec-arbol (30 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, Z, Q, R, A, M, P, K, L, T\}$,
- `ORD_POST` $=\{Z, A, P, M, K, L, R, T, Q, C\}$.

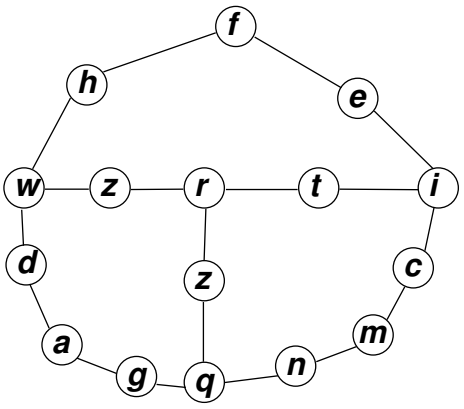
b) **[huffman (30 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario y encodar la palabra `VELOCIRAPTOR` $P(R) = 0{,}05, P(V) = 0{,}2, P(E) = 0{,}1, P(L) = 0{,}05, P(O) = 0{,}2, P(T) = 0{,}15, P(C) = 0{,}1, P(P) = 0{,}05, P(I) = 0{,}05, P(A) = 0{,}05$. Calcular la longitud promedio del código obtenido.

c) **[misc-arbol (10pt)]:** Dado el árbol `(c q (t (r u (v z))))`,

1) Cuál es el nodo que está a la vez a la izquierda de `v` y no es descendiente de `r`?
2) Particione el árbol con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

d) **[colorear-grafo (10 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] [**Preguntas (total = 20 puntos, 5puntos por pregunta)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(a b (c d e))`. ¿Cuál de las opciones es verdadera?

- ☐ Tiene dos raíces y altura 2.
- ☐ Tiene 3 hojas y altura 3.
- ☐ Tiene 3 hojas y altura 2.
- ☐ Tiene 4 nodos a profundidad 1 y 3 hojas.

¿Cuál es el tiempo de ejecución para la función `find(x)` en conjuntos implementados con árboles binarios de búsqueda?

- ☐ Caso promedio $O(n)$, peor caso $O(n)$.
- ☐ Siempre $O(n \log n)$.
- ☐ Caso promedio $O(n \log n)$, peor caso $O(n^2)$.
- ☐ Caso promedio $O(\log n)$, peor caso $O(n)$.

Sea `L` una lista conteniendo los elementos $(1, 3, 4, 2, 5, 6)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = L.erase(q);
p++;
q = p;
q++;
```

¿Cuál de las siguientes opciones es verdadera?

- ☐ `*p=2, *q=5`.
- ☐ `*p=2`, `q` es inválido.
- ☐ `*p=4`, `*q=2`
- ☐ `*p=4`, `q` es inválido.

¿Cuál es el tiempo de ejecución para intersección de conjuntos por vectores de bits? (N es el número de elementos en el conjunto universal, n el número de elementos en el conjunto dado.)

- ☐ $O(N \log N)$
- ☐ $O(n)$
- ☐ $O(N)$
- ☐ $O(n \log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [6 de Octubre de 2005]

El examen se compone de dos partes.

- **Clases y programación: TODOS.**
- **Operativos y preguntas: Sólo LIBRES.**

Los libres deben cumplir con al menos un 70 % del porcentaje de la segunda parte.

Ej. 1.- [**todos: queue (20 pt)**] Escribir las funciones del `TAD COLA` (`queue<>`). Incluir el header y todas las declaraciones necesarias.

Ej. 2.- [**todos: Ejercicios de programación**]

a) [**verifica-abb (50 pt)**] Escribir una función predicado `bool verifica_abb(tree<int> &A,bool (*less)(int,int));` que verifica si el subárbol de un nodo `n` verifica la condición de *árbol binario de búsqueda* con respecto a la relación de orden `less()`.

b) [**elimina-valor (30 puntos)**]
Escribir una función `voi elimina_valor(queue<int>&C, int);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C =` {`1,3,5,4,2,3,7,3,5`}, después de `elimina_valor(C,3)` debe quedar `C =` {`1,5,4,2,7,5`}. *Sugerencia: Usar una estructura auxiliar lista o cola.*
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

Ej. 3.- [**libres: Ejercicios operativos. (total 80pt)**]

a) [**particionar (20 pt)**] Considerando el árbol `(z q (r a (b l)) (t n m))`, decir cuál son los nodos `descendientes(q)`, `antecesores(q)`, `izquierda(q)` y `derecha(q)`.

b) [**heap-sort (20 pt)**] Dados los enteros $\{6, 8, 11, 4, 5, 9, 6, 7, 10\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

c) [**huffman (20 ptos)**] Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario y encodar la palabra `KATHRINA`
$P(K) = 0{,}05, P(H) = 0{,}2, P(T) = 0{,}1, P(R) = 0{,}05, P(A) = 0{,}2, P(N) = 0{,}15, P(I) = 0{,}1, P(P) = 0{,}05, P(S) = 0{,}05, P(B) = 0{,}05$. Calcular la longitud promedio del código obtenido.

d) [**rec-arbol (20 ptos)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{W, X, Y, Z, R, P, Q, T, V, U\}$,
- `ORD_POST` $=\{X, Y, R, P, T, V, U, Q, Z, W\}$.

Ej. 4.- [**libres: preguntas (20 pt)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) ¿Cómo es el tiempo de ejecución para intercalar dos listas clasificadas de n elementos?

- ☐ ... $O(1)$
- ☐ ... $O(n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

b) ¿Cuál de los siguientes árboles es un árbol binario de búsqueda?

- ☐ ... `(7 5 (9 6 (10 . 12)))`
- ☐ ... `(7 5 (9 (8 6 .) (13 . 12)))`
- ☐ ... `(7 5 (9 8 (10 . 12)))`
- ☐ ... `(7 5 (9 8 (10 12)))`

c) El tiempo de ejecución para el algoritmo de *clasificación rápida* (*quick-sort*) en el peor caso es:

- ☐ ... $O(n \log n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(1)$
- ☐ ... $O(n)$

d) ¿Cuál es el tiempo de ejecución de la operación de *re-heap* en montículos?

- ☐ ... Caso promedio $O(\log n)$.
- ☐ ... Siempre $O(\log n)$.
- ☐ ... Caso promedio $O(n \log n)$.
- ☐ ... Siempre $O(n \log n)$.

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos
Examen Final. [15 de Diciembre de 2005]

El examen se compone de dos partes.

- **Clases y programación: TODOS.**
- **Operativos y preguntas: Sólo LIBRES.**

Los libres deben cumplir con al menos un 70 % del porcentaje de la segunda parte.

1. [**todos: clases (20 puntos)**]. Escribir la implementación en `C++` del `TAD ARBOL BINARIO` (clase `btree`). Las funciones a implementar son `insert(p,x)`, `erase(p)` y `find(x)`. Observaciones:
   - En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
   - En caso de optar por la interfase *"avanzada"*, **no es necesario** declarar e implementar las clases `iterator` y `cell`.

2. [**todos: Ejercicios de programación**].

   a) [**nilpot (50 puntos)**]. Dadas dos correspondencias M_1 y M_2 la *"composición"* de ambas es la correspondencia $M = M_2 \circ M_1$ tal que si $M_1[a] = b$ y $M_2[b] = c$, entonces $M[a] = c$. Por ejemplo, si `M1={(0,1),(1,2),(2,0),(3,4),(4,3)}`, y `M2={(0,1),(1,0),(2,3),(3,4),(4,2)}`, entonces $M = M_1 \circ M_2$ `={(0,0),(1,3),(2,1),(3,2),(4,4)}`. Notemos que para que sea posible componer las dos correspondencias es necesario que los valores del contradominio de M_1 estén incluidos en las claves de M_2. Si el conjunto de valores del contradominio de una correspondencia M está incluido en el conjunto de sus claves, entonces podemos componer a M consigo misma, es decir $M^2 = M \circ M$. Por ejemplo, $M_1^2 = M_1 \circ M_1$`= {(0,2),(1,0),(2,1),(3,3),(4,4)}`. De la misma manera puede definirse, $M^3, ..., M^n$, componiendo sucesivamente. Puede demostrarse que, para algún n debe ser $M^n = I$, donde I es la *"correspondencia identidad"*, es decir aquella tal que $I[x] = x$. Por ejemplo, si `M = {(0,1),(1,2),(2,0)}`, entonces para $n = 3$, $M^n = M^3 = I$.
   Consigna: Escribir una función `int nilpot(map<int,int> &M);` que dada una correspondencia `M` retorna el mínimo entero `n` tal que $M^n = I$.
   Sugerencia: Escribir dos funciones auxiliares:
   - `void compose(map<int,int> &M1,map<int,int> &M2,map<int,int> &M);` que dadas dos correspondencias `M1, M2`, calcula la composición $M = M_2 \circ M_1$, devolviéndola en el argumento `M`,
   - `bool is_identity(map<int,int> &M);` que dada una correspondencia `M`, retorna `true` si `M` es la identidad, y `false` en caso contrario.

   b) [**elimina-valor (30 puntos)**].
   Escribir una función `void elimina_valor(queue<int>&C, int);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C =` {`1,3,5,4,2,3,7,3,5`}, después de `elimina_valor(C,3)` debe quedar `C =` {`1,5,4,2,7,5`}. *Sugerencia: Usar una estructura auxiliar lista o cola.*
   Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

3. [**libres: Ejercicios operativos. (total 80 puntos)**].

   a) [**particionar (20 puntos)**]. Considerando el árbol `(f (a c e (h b)) (d g))` decir cuál son los nodos `descendientes(b)`, `antecesores(b)`, `izquierda(b)` y `derecha(b)`.

b) [**heap-sort (20 puntos)**]. Dados los enteros $\{5, 0, 8, 2, 4, 5, 2, 4, 1, 3\}$. ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) **antes** y **después** de **cada** inserción/supresión.

c) [**huffman (20 puntos)**]. Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario (para **todos** los caracteres) y encodar la palabra `FAVALORO`: $P(A) = 0.24$, $P(F) = 0.12$, $P(G) = 0.10$, $P(L) = 0.10$, $P(M) = 0.10$, $P(O) = 0.14$, $P(R) = 0.08$, $P(S) = 0.06$, $P(V) = 0.06$. Indicar el número de nivel de cada caracter y calcular la longitud promedio del código obtenido.

d) [**rec-arbol (20 puntos)**]. Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE`=$\{B, F, E, H, I, D, J, A, C, G\}$,
- `ORD_POST`=$\{E, F, I, D, A, J, C, H, G, B\}$.

4. [**libres: preguntas (20 puntos)**]. Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Dado el árbol binario `(z (a b q) r)`, ¿cuál de las siguientes opciones es verdadera ?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

b) ¿Cuál es el tiempo de ejecución para intersección de conjuntos por vectores de bits? (N es el número de elementos en el conjunto universal, n el número de elementos en el conjunto dado)

- ☐ $O(N \log N)$
- ☐ $O(n)$
- ☐ $O(N)$
- ☐ $O(n \log n)$

c) Sea el árbol `(5 7 (8 6 9))`. Después de hacer:

```
n = D.find(7);
n++;
n = n.lchild();
n = n.lchild();
n = D.insert(n,2);
```

¿Cuál de las opciones es verdadera?

- ☐ `(5 7 (8 2 6 9))`
- ☐ Da un error.
- ☐ `(5 7 (8 6 2 9))`
- ☐ `(5 7 (8 (6 2) 9))`

d) ¿Cómo es el tiempo de ejecución para intercalar dos listas clasificadas de n elementos?

- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
- ☐ ... $O(n!)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [29 de Diciembre de 2005]

[Ej. 1] [Clases (20 puntos)]

Escribir los siguientes métodos del TAD `btree`: `insert(p,x)`, `erase(p)`, `find(x)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [programacion (total = 80 puntos)]

a) **[encuentra (35 puntos)]** Escribir una función `bool encuentra(list<int> &L1, list<int> &L2, list<int> &indx);` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `indx` los índices de los elementos de `L2` que forman `L1`, si no `indx` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `encuentra` debe retornar `true`, e `indx`=$\{0, 1, 4, 5, 7, 9\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `indx`=$\{\}$. Nota: Los índices en `indx` deben ser estrictamente crecientes. *Restricciones:* No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$.

b) **[encuentra-suma (35 puntos)]** Escribir una función `tree<double>::iterator encuentra_suma(double s, tree<double> &A);` que retorna el nodo `m` del árbol orientado `A` tal que la suma de las etiquetas de todos los nodos descendientes de `m` (incluyendo a `m`) es `s`. Si no existe un tal nodo, entonces debe retornar Λ. Si hay varios nodos que cumplen la condición basta con que retorne uno de ellos. Por ejemplo, para `A=(10 21 (6 (8 7) (11 (12 3) 6) (4 6 5)))`

- `encuentra_suma(32,A)` retorna el nodo 11,
- `encuentra_suma(15,A)` puede retornar el nodo 8, el 4 o el 12.
- `encuentra_suma(9,A)` retorna Λ

Restricciones: El algoritmo debe ser $O(n)$ donde n es el número de nodos en el árbol.

c) **[reordena-pila (10 puntos)]** Escribir una función `void reordena(stack<int> &P,bool (*pred)(int));` que reordena los elementos de una pila de tal forma que quedan los que no satisfacen el predicado `pred()` en el fondo y los que si lo satisfacen arriba. Por ejemplo si $P = \{1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9\}$ y `pred()` es `bool par(int x) { return x%2==0; }` entonces debe quedar $P = \{4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9\}$. *Restricciones:* el algoritmo debe ser *estable*, es decir los elementos que satisfacen `pred()` deben quedar en el mismo orden entre sí, y lo mismo para los que no satisfacen `pred()`. Se pueden usar dos estructuras auxiliares (listas, pilas o colas).

[Ej. 3] [operativos (total = 80 puntos)]

a) **[rec-arbol (40 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

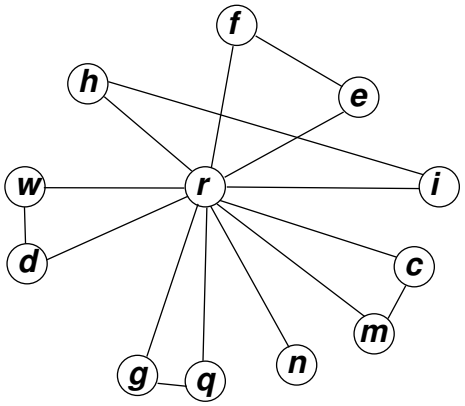
- `ORD_PRE` $=\{A, B, C, E, F, Z, Y, W, G, D\}$,
- `ORD_POST` $=\{B, E, Z, Y, W, F, G, C, D, A\}$.

b) **[misc-arbol (20pt)]:** Dado el árbol `(b t (u (z x y) w))`,

1) Cuál es el nodo de profundidad 2 que está a la izquierda de `w`?
2) Particione el árbol con respecto al nodo `u`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

c) **[colorear-grafo (20 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] **[Preguntas (total = 20 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(q e (r w) s t)`. ¿Cuál de las opciones es verdadera?

- ☐ Tiene 4 nodos a profundidad 1 y 5 hojas.
- ☐ Tiene dos raíces y 4 hojas.
- ☐ Tiene 4 nodos a profundidad 1 y 4 hojas.
- ☐ Tiene 5 nodos a profundidad 1 y 4 hojas.

¿Cuál es el tiempo de ejecución de la operación de *re-heap* en montículos?

- ☐ Caso promedio $O(\log n)$.
- ☐ Siempre $O(\log n)$.
- ☐ Caso promedio $O(n \log n)$.
- ☐ Siempre $O(n \log n)$.

Dado el árbol binario `(z (a b q) r)`, ¿cuál de las siguientes opciones es verdadera?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n + B)$
- ☐ $O(n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 1A. [21 de abril de 2005]

[Ej. 1] **[Clases (30 puntos)]** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores **ó** arreglos. Las funciones a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `clear()`. Observaciones:

- En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase *"avanzada"*, debe declarar e implementar completamente las partes privadas de la clase `list` e `iterator`.

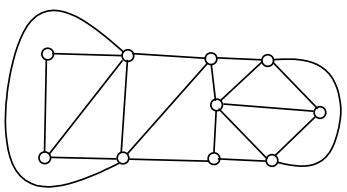
[Ej. 2] **[Programación (total = 50 puntos)]**

a) **[nilpot (25 puntos)]** Dadas dos correspondencias M_1 y M_2 la *"composición"* de ambas es la correspondencia $M = M_2 \circ M_1$ tal que si $M_1[a] = b$ y $M_2[b] = c$, entonces $M[a] = c$. Por ejemplo, si `M1={(0,1),(1,2),(2,0),(3,4),(4,3)}`, y `M2={(0,1),(1,0),(2,3),(3,4),(4,2)}`, entonces $M = M_1 \circ M_2$ `={(0,0),(1,3),(2,1),(3,2),(4,4)}`. Notemos que para que sea posible componer las dos correspondencias es necesario que los valores del contradominio de M_1 estén incluidos en las claves de M_2. Si el conjunto de valores del contradominio de una correspondencia M está incluido en el conjunto de sus claves, entonces podemos componer a M consigo misma, es decir $M^2 = M \circ M$. Por ejemplo, $M_1^2 = M_1 \circ M_1$`= {(0,2),(1,0),(2,1),(3,3),(4,4)}`. De la misma manera puede definirse, $M^3, ..., M^n$, componiendo sucesivamente. Puede demostrarse que, para algún n debe ser $M^n = I$, donde I es la *"correspondencia identidad"*, es decir aquella tal que $I[x] = x$. Por ejemplo, si `M = {(0,1),(1,2),(2,0)}`, entonces para $n = 3$, $M^n = M^3 = I$.
Consigna: Escribir una función `int nilpot(map<int,int> &M);` que dada una correspondencia `M` retorna el mínimo entero `n` tal que $M^n = I$.
Sugerencia: Escribir dos funciones auxiliares:

- `void compose(map<int,int> &M1,map<int,int> &M2,map<int,int> &M);` que dadas dos correspondencias `M1, M2`, calcula la composición $M = M_2 \circ M_1$, devolviéndola en el argumento `M`,
- `bool is_identity(map<int,int> &M);` que dada una correspondencia `M`, retorna `true` si `M` es la identidad, y `false` en caso contrario.

b) **[chunk-revert (25 puntos)]** Escribir una función `void chunk_revert(list<int> &L,int n);` que dada una lista `L` y un entero `n`, invierte los elementos de la lista tomados de a `n`. Si la longitud de la lista no es múltiplo de `n` entonces se invierte el resto también. Por ejemplo, si `L={1,3,2,5,4,6,2,7}` entonces después de hacer `chunk_revert(L,3)` debe quedar `L={2,3,1,6,4,5,7,2}`. *Restricciones:* Usar a lo sumo una estructura auxiliar. (En tal caso debe ser lista, pila o cola).

[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. ¿La coloración obtenida es óptima? Justifique.



[Ej. 4] **[Preguntas (total = 15 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Considere la función:

```
bool is_mapped(map<int,int> &M,int key,int val) {
  return /* Esta key -> val en M? ... */; }
```

que debe retornar `true` si el par de asignación `(key,val)` está en la correspondencia `M`. ¿Cuál es la expresión correcta que refina el seudocódigo?

- ☐ ... `M[key]==val`
- ☐ ... `M[key]==val && (M.find(key)!=M.end())`
- ☐ ... `M.find(key)->second==val`
- ☐ ... `(M.find(key)!=M.end()) && M[key]==val`

---

b) El tiempo de ejecución de la función `find(key)` para correspondencias implementadas por vectores ordenados es ... (n es el número de asignaciones en la correspondencia)

- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
- ☐ ... $O(n^2)$

---

c) Dadas las funciones

- $T_1(n) = 3n^3 + 2n! + \log n$,
- $T_2(n) = 3 \cdot 2^3 + n^2 + n^{1.5}$,
- $T_3(n) = 5! + 6 \cdot 2^n + 20 \cdot n^2$ y
- $T_4(n) = 2^{10} + 20n + \log_2 40$

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 2A. [21 de abril de 2005]

[Ej. 1] **[Clases (30 puntos)]** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores **ó** arreglos. Las funciones a implementar son `insert(p,x)`, `erase(p,q)`, `begin()`, `end()`, `retrieve()/operator*()`, `~list()`. Observaciones:

- En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase *"avanzada"*, debe declarar e implementar completamente las partes privadas de la clase `list` e `iterator`.

[Ej. 2] **[Programación (total = 50 puntos)]**

a) **[lexico-stack (25 puntos)]** Considere el problema de generar todas las subsecuencias ordenadas de la secuencia $X = (1, 2, ..., n)$.

Por ejemplo, si $n = 4$ las subsecuencias ordenadas de $X = (1, 2, 3, 4)$ son: (1), (12), (123), (124), (13), (134), (14) (2), (23), (234) (24), (3), (34) y (4).

Esta construcción se puede implementar mediante el uso de una pila S bajo las siguientes reglas:

- Inicializar la pila con el elemento 1.
- Si el tope t de la pila verifica $t < n$ entonces apilamos $t + 1$.
- Si $t = n$, entonces lo desapilamos y, a continuación, si la pila no quedara vacía incrementamos el nuevo tope de la misma.
- El algoritmo termina cuando la pila queda vacía.

Ejemplo:

```
          4
        3 3 4     4         4
      2 2 2 2 3 3 4     3 3 4     4
    1 1 1 1 1 1 1 1 2 2 2 2 3 3 4
```

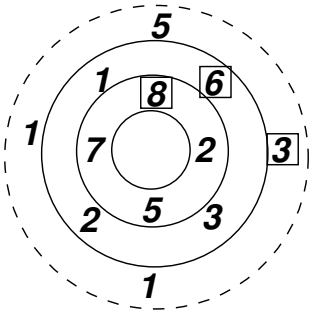
Consigna: Escriba un procedimiento `void lexico_stack(int & n);` el cual, ingresado el número natural n imprime todos los conjuntos ordenados de $(1, 2, ..., n)$. *Sugerencia:* Implementar el algoritmo descripto, llamando a una función auxiliar `void imprime_pila(stack<int> &S)` (implementarla!!) que imprime la pila `S` en forma no-destructiva.

Restricciones:

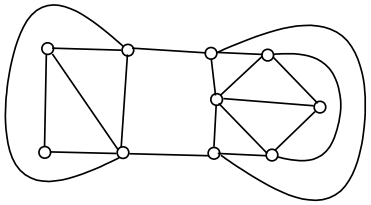
- Usar la interfase STL para pilas.
- En `lexico_stack()`: usar una sola estructura auxiliar.
- En `imprime_pila()`: usar una sola estructura auxiliar.
- No usar otros algoritmos de STL.

b) **[circulo (25 puntos)]** Coloquemos n números enteros positivos alrededor de una circunferencia inicial. Construyamos ahora sucesivas circunferencias concéntricas *hacia el exterior*, de igual cantidad de elementos, los cuales son obtenidos restando (en valor absoluto) pares consecutivos de la última circunferencia exterior. Entonces, puede verificarse que si $n = 2^k$ en alguna iteración p aparecerán n números iguales. En ese momento se detiene la iteración.

Por ejemplo, supongamos $k = 2$, $(n = 4)$ y que la circunferencia "inicial" sea $C_0 = (8, 2, 5, 7)$, entonces iteramos y obtendremos sucesivamente, $C_1 = (6, 3, 2, 1)$, $C_2 = (3, 1, 1, 5)$, $C_3 = (2, 0, 4, 2)$, $C_4 = (2, 4, 2, 0)$ y $C_5 = (2, 2, 2, 2)$, por lo que el número de circunferencias iteradas es $p = 5$. Entonces, dada una lista $L = [x_0, x_1, ..., x_{n-1}]$ de n números enteros que representan los valores alrededor de la circunferencia inicial, escribir una función `int circulo(list<int>& L);` que ejecuta esta tarea y devuelva además el número de circunferencias iteradas p.
Restricción: el algoritmo debe ser *in place*.
Ayuda: Pensar a la lista en un *"sentido circular"*. Tener cuidado al generar la diferencia correspondiente al extremo.



[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. ¿La coloración obtenida es óptima? Justifique.



[Ej. 4] **[Preguntas (total = 15 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) La función

```
void swap(list<int> &L,list<int>::iterator p) {
  int x; list<int>::iterator q;
  /* ... */;  }
```

debe intercambiar el elemento en la posición `p` con el de la siguiente `q` (asumiendo que son dereferenciables). ¿Cuál de los siguientes es el fragmento faltante correcto?

- ☐ `q = p; x = *++q; L.erase(q); L.insert(p,x);`
- ☐ `x = *++p; p = L.erase(p); L.insert(p,x);`
- ☐ `x = *++p; L.erase(p); L.insert(p,x);`
- ☐ `x = *p; p = L.erase(p); L.insert(p,x);`

---

b) El tiempo de ejecución de la función `find(key)` para correspondencias implementadas por listas ordenadas es (en el caso promedio)... (n es el número de asignaciones en la correspondencia)

- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(1)$
- ☐ ... $O(n)$

---

c) Dadas las funciones

- $T_1(n) = 3n^3 + 3 \cdot 2^3 + 5! + 2^{10}$,
- $T_2(n) = 2n! + n^2 + 6 \cdot 2^n + 20n$,
- $T_3(n) = \log n + n^{1.5} + 20n^2 + \log_2 40$ y
- $T_4(n) = 10n^3 + n^4 + 6 \cdot 2^n + 0.0001n!$

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [26 de mayo de 2005]

[Ej. 1] **[clases (20 puntos)]** Escribir la implementación en `C++` del `TAD ARBOL ORDENADO ORIENTADO` (clase `tree`). Las funciones a implementar son `insert(p,x)`, `erase(p)`, `find(x)` y `clear()`. Observaciones:

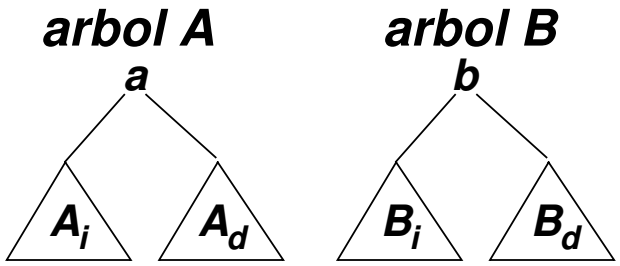
- En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase *"avanzada"*, debe declarar e implementar completamente las partes privadas de la clase `tree`, `iterator` y `cell`.

[Ej. 2] **[programacion (50 pts)]**

a) **[es-menor (30 pts)]** Se define una relación de orden entre árboles binarios de enteros de la siguiente forma:

$$A < B \text{ si } \begin{cases} a < b; \\ a = b \ \text{ y } \ A_i < B_i \\ a = b \ \text{ y } \ A_i = B_i \ \text{ y } \ A_d < B_d \end{cases} \qquad \text{(si } a \text{ o } b \text{ son } \Lambda \text{ entonces se considera } -\infty)$$

donde a, b son las raíces y A_i, A_d, B_i, B_d son los subárboles izquierdos y derechos.
Consigna: Escribir una función `bool es_menor(tree<int>&A,tree<int>&B)` que retorna verdadero si $A < B$.



b) **[count-if (10 pts)]** Escribir una función `int count_if(tree<int>&T,bool (*pred)(int x));` que retorna el número de nodos del árbol `T` que satisfacen el predicado `pred`. Por ejemplo, si `T=(1 2 (3 5 7 6) 4)`, entonces `count_if(T,odd)` debe retornar 4. Escribir el predicado `bool odd(int x)` que determina si un entero es impar.

c) **[list-if (10 pts)]** Escribir una función `void list_if(tree<int>&T,list<int>&L,bool (*pred)(int x));` que retorna en `L` la lista de valores nodales en orden previo de un árbol ordenado orientado `T` que satisfacen el predicado `pred`. Por ejemplo, si `T=(1 (-2 7 (8 -7) (3 -5 -6)))`, entonces después de `list_if(T,L,positive)`, debe quedar `L={1,7,8,3}`. Escribir el predicado `bool positive(int x)` que determina si un entero es mayor que 0.

[Ej. 3] **[operativos (20 pts)]**

- **[rec-arbol (8 pts)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
  - `ORD_PRE` $=\{Z, A, R, Q, L, M, N, T, S, W, Q\}$,
  - `ORD_POST` $=\{Q, L, R, N, M, A, W, Q, S, T, Z\}$.

- **[huffman (8 pts)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CONCLAVE`
  $P(C) = 0.05, P(O) = 0.05, P(N) = 0.1, P(A) = 0.2, P(L) = 0.2, P(V) = 0.1, P(E) = 0.1, P(Q) = 0.2$
  Calcular la longitud promedio del código obtenido.
- **[part-arbol (4 pts)]** Particione el árbol AOO `(p (q (s x y) t u) (r v w))` con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

[Ej. 4] **[preguntas (10 pts, 2.5 por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) ¿Como es el número de niveles l para un árbol binario lleno, en función del número de nodos n?

- ☐ ... $l = O(2^n)$
- ☐ ... $l = O(n!)$
- ☐ ... $l = O(\log_2 n)$
- ☐ ... $l = O(n^2)$

---

b) En la codificación de mensajes en secuencias de bits, la condición de prefijos asegura que...

- ☐ ... el código es óptimo.
- ☐ ... el mensaje codificado se puede *decodificar* en forma única.
- ☐ ... el mensaje se puede *codificar* en forma única.
- ☐ ... el código no es redundante.

---

c) El algoritmo de Huffmann para compresión se basa en ...

- ☐ ... utilizar códigos más cortos para los caracteres con *mayor* probabilidad.
- ☐ ... utilizar códigos más cortos para los caracteres con *menor* probabilidad.
- ☐ ... utilizar códigos que generen longitudes promedio lo más largas posibles.
- ☐ ... utilizar códigos con la misma longitud en bits para cada carácter.

---

d) Considere el AOO `Q=(3 5 8 (7 8 6) 9)`, después de hacer las siguientes operaciones

```
tree<int>::iterator n = Q.find(5);
n++;
n = Q.erase(n);
n++;
n = Q.erase(n);
```

- ☐ ... queda `(3 5 (7 8 6))`, `*n=7`
- ☐ ... queda `(3 5 9)`, `n=end()`
- ☐ ... da un error.
- ☐ ... queda `(3 5 (7 8 6))`, `n=end()`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
3er Parcial. Tema: 1A. [23 de Junio de 2005]

[Ej. 1] [clases (total 20 pts)]

a) [**vecbit (total 5 pts)**] El archivo de cabecera siguiente declara la clase conjunto para rangos contiguos de enteros utilizando vectores de bits.

- El tamaño del conjunto universal es `N`, es decir, los enteros que guarda el conjunto pertenecen al intervalo `[0, N)`.
- El constructor inicializa el vector de bits con `N` elementos en `false` (es decir, el conjunto esta inicialmente vacío).
- El valor de `N` siempre puede recuperarse con `vecbit.size()`.

Implemente el método `size()` y la operación binaria `set_difference(A,B,C)` de tal forma que su complejidad sea $O(N)$. Asuma que los conjuntos `A`, `B` y `C` fueron todos creados con el mismo valor de `N`.

```
1  #ifndef SET_VECBIT
2  #define SET_VECBIT
3
4  #include <vector>
5
6  class set {
7  private:
8    std::vector<bool> vecbit;
9    /* ... */
10 public:
11   set(int N) : vecbit(N, false) { }
12   /* ... */
13   int size();
14   friend void set_difference(set &A, set &B, set &C);
15 };
16 /* ... */
17 void set_difference(set &A, set &B, set &C); // C = A - B
18
19 #endif
```

b) [**bstree (total 10 pts)**] El archivo de cabecera siguiente declara la clase conjunto para una implementacion por árbol binario de búsqueda.

- Los conjuntos contienen un árbol binario `bstree` como miembro privado.
- Los iteradores de conjuntos contienen un puntero `bstree` al árbol binario del conjunto y también un iterador de árbol binario `node` que indica la posición del elemento en el árbol binario de búsqueda..

Implemente las funciones `clear()` e `insert(x)`. Para esta última utilice como ayuda el método `find(x)` y el constructor `iterator(n,bst)`, ambos ya implementados.

```
1  #ifndef SET_BSTREE
2  #define SET_BSTREE
```

```
3  #include <pair>
4  #include <btree.h>
5
6  template<typename T>
7  class set {
8  private:
9    btree<T> bstree;
10   /* ... */
11 public:
12   class iterator {
13     friend class set;
14   private:
15     btree<T>::iterator node;
16     btree<T>           *bstree;
17     iterator(btree<T>::iterator n, btree<T> &bst)
18       : node(n), bstree(&bst) { }
19     /* ... */
20   public:
21     /* ... */
22   }; // end class iterator
23   iterator find(T x) {
24     btree<T>::iterator m = bstree.begin();
25     while(true) {
26       if (m == bstree.end())
27         return iterator(m, bstree);
28       if      (x < *m) m = m.left();
29       else if (x > *m) m = m.right();
30       else  return iterator(m, bstree);
31     }
32   }
33   /* ... */
34   std::pair<iterator, bool> insert(T x);
35   void clear();
36   /* ... */
37 }; // end class set
38
39 #endif // SET_BSTREE
```

c) [**merge-sort (total 5 pts)**] Implemente el algoritmo de ordenamiento por fusión para listas `merge_sort(L,comp)`. Se sugiere implementar dos funciones auxiliares:

- `split(L,L1,L2)` que separe una lista `L` en dos listas `L1` y `L2`, dejando a `L` vacía.
- `merge(L1,L2,L,comp)` que fusione las listas **ordenadas** `L1` y `L2` en una lista `ordenada` `L` utilizando la *función de comparación* `comp`, dejando a `L1` y `L2` vacías.

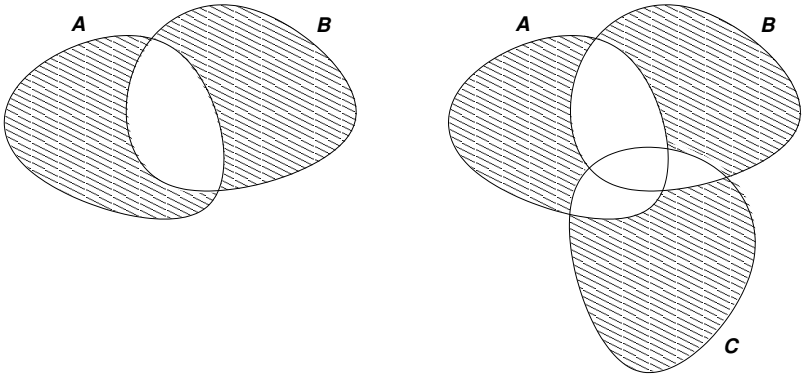
```
1  #include <list>
2
3  template<typename T>
4  void merge_sort(std::list<T> &L, bool (*comp)(T&, T&));
5
6  template<typename T>
7  void split(std::list<T> &L, std::list<T> &L1, std::list<T> &L2);
```

```
8
9  template<typename T>
10 void merge(std::list<T> &L1, std::list<T> &L2,
11            std::list<T> &L, bool (*comp)(T&, T&));
```

[Ej. 2] [programacion (50 pts)]

a) **[diff-sym (30 pts)]** Para dos conjuntos A, B, la *"diferencia simétrica"* se define como

$$\begin{aligned}\text{diff_sym}(A,B) &= (A-B)\cup(B-A), \quad \text{o también}\\ &= (A\cup B)-(A\cap B)\end{aligned}$$



En general, definimos la diferencia simétrica de varios conjuntos como el conjunto de todos los elementos que pertenecen a uno y sólo uno de los conjuntos. En las figuras vemos en sombreado la diferencia simétrica para dos y tres conjuntos. Por ejemplo, si $A = \{1, 2, 5\}$, $B = \{2, 3, 6\}$ y $C = \{4, 6, 9\}$ entonces $\text{diff_sym}(A, B, C) = \{1, 3, 4, 5, 9\}$.
Consigna: Escribir una función `void diff_sym(list<set<int> > &l,set<int>&s);` que retorna en `s` la diferencia simétrica de los conjuntos en `l`.
Ayuda: La solución se puede encarar con alguna de las dos estrategias siguientes:

1) Escribir una función `int cuenta(list<set<int> > &l,int x);` que retorna el número de conjuntos de `l` en los cuales x está incluido. Recorrer todos los elementos de todos los conjuntos de `l`, e insertar el elemento en `s` sólo si `cuenta` retorna exactamente 1.
2) Notar que en el caso de tres conjuntos si $S = \text{diff_sym}(A, B)$ y $U = A \cup B$, entonces $\text{diff_sym}(A, B, C) = (S - C) \cup (C - U)$. Esto vale en general para cualquier número de conjuntos, de manera que podemos utilizar el siguiente lazo

   l = lista de conjuntos, $S = \emptyset, U = \emptyset$;
   for Q = en la lista de conjuntos l **do**
   $S = (S - Q) \cup (Q - U)$;
   $U = U \cup Q$;
   end for

   Al terminar el lazo, S es la diferencia simétrica buscada.

b) **[incluido (20 pts)]** Escribir un predicado `bool incluido(set<int> &A, set<int> &B);` que retorna verdadero si y solo si $A \subset B$.

[Ej. 3] [operativos (20 pts)]

- **[heap-sort (5 pts)]** Dados los enteros $\{0, 4, 7, 1, 2, 12, 9, 3\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
- **[quick-sort (5 pts)]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.
- **[abb (5 pts)]** Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 0, 11\}$
  insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 7, 5 y 4 en ese orden.
- **[hash-dict (5 pts)]** Insertar los números $0, 13, 23, 6, 5, 33, 15, 2, 25$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

[Ej. 4] [preguntas (10 pts, 2.5 por pregunta)]

---

a) ¿Cuál de los siguientes es el resultado correcto de ordenar la secuencia $\{-3, 5, 3, 2, -1, -2, 1, 2, -3\}$ por la relación de orden débil $|a| < |b|$?

- ☐ ... $\{1, 2, -2, 2, -3, 3, -3, 5\}$
- ☐ ... $\{-3, -3, -3, 1, 2, 2, 3, 5\}$
- ☐ ... $\{1, -2, 2, 2, -3, -3, 3, 5\}$
- ☐ ... $\{1, 2, 2, -2, 3, -3, -3, 5\}$

---

b) ¿Cuál es el número de *intercambios* en el método de clasificación por selección?

- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$

---

c) El tiempo de ejecución de *"quick-sort"* en el peor caso es

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

---

d) Recordemos que el procedimiento *"re-heap"* es áquel que, mediante una serie de intercambios restituye la propiedad de montículo a un árbol binario el cuál satisface la propiedad de parcialmente ordenado en todos sus nodos menos, eventualmente, en la raíz. El tiempo de ejecución de *re-heap* es...

- ☐ ... $O(n^2)$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio. [30 de Junio de 2005]

[Ej. 1] [clases (20 pts)]

a) [**parcial-1 lista (20 pts)**] Escribir los siguientes métodos del **TAD lista**: `insert(p,x)`, `erase(p)`, `begin()`.

b) [**parcial-2 arbol-bin (20 pts)**] Escribir los siguientes métodos del **TAD árbol binario**: `find(x,p)`, `insert(p,x)`, `clear()`.

c) [**parcial-3 set (20 pts)**] Escribir los siguientes métodos del **TAD conjunto** por listas ordenadas: `insert(x)`, `find(x)`, `clear()`.

[Ej. 2] [programacion (50 pts)]

a) [**map-pre-post (30 pts)**] Escribir una función `void map_pre_post(tree<int> &T,list<int> &L, int (*fpre)(int),int (*fpost)(int))` que lista los valores nodales del árbol ordenado orientado `T` en una mezcla de orden previo y posterior, de acuerdo a la siguiente definición

$$\text{mpp}(\Lambda, f, g) = \text{lista vacía}$$
$$\text{mpp}(n, f, g) = f(n), \text{mpp}(n_1, f, g), ..., \text{mpp}(n_m, f, g), g(n)$$

donde $n_1 ... n_m$ son los hijos del nodo n. Por ejemplo, si `T=(1 3 (5 6 7 8))`, $f(x) = x$ y $g(x) = x + 1000$, entonces `map_pre_post(T,L,f,g)` debe dar `L=(1,3,1003,5,6,1006,7,1007,8,1008,1005,1001)`.

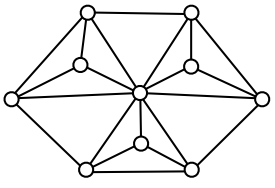
b) [**purge (10 pts)**] Escribir una función `void purge(list<int> & L)` que purga elementos repetidos de una lista usando un conjunto como estructura auxiliar y con una implementación tal que sea $O(n \log n)$.

c) [**cum-sum-pila (10 pts)**] Escribir una función `void cum_sum_pila(stack<int>&S)` que modifica a la pila `S` dejando la suma acumulada de sus elementos, es decir, si los elementos de `S` antes de llamar a `cum_sum_pila (S)` son $S = (a_0, a_1, \ldots, a_{n-1})$, entonces después de llamar a `cum_sum_pila (S)` debe quedar $S = (a_0, a_0 + a_1, \ldots, a_0 + a_1 + \ldots + a_{n-1})$. Por ejemplo, si `S = (1,3,2,4,2)` entonces después de hacer `cum_sum_pila (S)` debe quedar `S = (1,4,6,10,12)`. Restricciones: (i) usar una pila auxiliar; (ii) NO usar más estructuras auxiliares que la indicada ni otros algoritmos de STL; y (iii) el algoritmo debe ser $O(n)$.

[Ej. 3] [operativos (20 pts)]

- [**parcial-1 color-grafo (10 pts)**]

  Colorear el grafo de la figura usando un algoritmo heurístico ávido para obtener el mínimo número de colores posible. ¿La coloración obtenida es óptima? Justifique.



- **[parcial-1 t-exec (10 pts)]**
  Dadas las funciones
  - $T_1(n) = 5n^3 + 2n! + \log n$,
  - $T_2(n) = 2^{15} + 2 \cdot 5^n + 3 \cdot n^2$,
  - $T_3(n) = 6! + n^2 + n^{1.7}$,
  - $T_4(n) = 1.3 \cdot 2^3 + 20n + \log_2 10$.

  ordenarlas de menor a mayor.

  $T_{\square} < T_{\square} < T_{\square} < T_{\square}$
- **[parcial-2 rec-arbol (8 pts)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
  - `ORD_PRE` $=\{A, Z, W, D, E, X, Y, P, Q, R, S\}$,
  - `ORD_POST` $=\{E, D, W, X, Z, P, S, R, Q, Y, A\}$.
- **[parcial-2 huffman (8 pts)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CONEXA`
  $P(C) = 0.2, P(O) = 0.1, P(N) = 0.1, P(A) = 0.05, P(L) = 0.05, P(X) = 0.1, P(E) = 0.1, P(Q) = 0.3$
  Calcular la longitud promedio del código obtenido.
- **[parcial-2 part-arbol (4 pts)]** Particione el árbol AOO `(z (q w y l) (r t u))` con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.
- **[parcial-3 heap-sort (5 pts)]** Dados los enteros $\{21, 12, 9, 3, 5, 10, 7, 6\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
- **[parcial-3 quick-sort (5 pts)]** Dados los enteros $\{17, 19, 9, 6, 2, 3, 13, 18, 9, 0\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.
- **[parcial-3 abb (5 pts)]** Dados los enteros $\{10, 12, 15, 11, 7, 6, 4, 5, 8, 9\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 7, 12 y 8, en ese orden.
- **[parcial-3 hash-dict (5 pts)]** Insertar los números $5, 18, 28, 11, 10, 38, 23, 7, 30$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x\,\%10$ y estrategia de redispersión lineal.

[Ej. 4] [preguntas (10 pts, 2.5 por pregunta)]

a) ¿Cuál es el criterio para elegir una buena función de dispersión? ...

- ☐ Debe tratar de concentrar los elementos en pocas cubetas.
- ☐ Debe tratar de concentrar los elementos en una sóla cubeta.
- ☐ Debe tratar de concentrar los elementos en la primera cubeta.
- ☐ Debe distribuir los elementos en la forma más uniforme posible entre las cubetas.

---

b) Dado el árbol binario `(x e (d f g))`, ¿cuál de las siguientes opciones es verdadera?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

---

c) ¿Cuál es el número de niveles en un árbol binario lleno en función del número n de nodos en el árbol?

- ☐ ... $O(n \log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n)$
- ☐ ... $O(\log n)$

---

d) Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n + B)$
- ☐ $O(1 + n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [06 de Julio de 2006]

[Ej. 1] [Clases (20 puntos)]

Escribir los siguientes métodos del TAD `btree`: `insert(p,x)`, `erase(p)`, `find(x)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [programacion (total = 80 puntos)]

a) **[encuentra (35 puntos)]** Escribir una función `bool encuentra(list<int> &L1, list<int> &L2, list<int> &indx);` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `indx` los índices de los elementos de `L2` que forman `L1`, si no `indx` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `encuentra` debe retornar `true`, e `indx`=$\{0, 1, 4, 5, 7, 9\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `indx`=$\{\}$. Nota: Los índices en `indx` deben ser estrictamente crecientes. *Restricciones:* No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$.

b) **[encuentra-suma (35 puntos)]** Escribir una función `tree<double>::iterator encuentra_suma(double s, tree<double> &A);` que retorna el nodo `m` del árbol orientado `A` tal que la suma de las etiquetas de todos los nodos descendientes de `m` (incluyendo a `m`) es `s`. Si no existe un tal nodo, entonces debe retornar Λ. Si hay varios nodos que cumplen la condición basta con que retorne uno de ellos. Por ejemplo, para `A=(10 21 (6 (8 7) (11 (12 3) 6) (4 6 5)))`

- `encuentra_suma(32,A)` retorna el nodo 11,
- `encuentra_suma(15,A)` puede retornar el nodo 8, el 4 o el 12.
- `encuentra_suma(9,A)` retorna Λ

Restricciones: El algoritmo debe ser $O(n)$ donde n es el número de nodos en el árbol.

c) **[reordena-pila (10 puntos)]** Escribir una función `void reordena(stack<int> &P,bool (*pred)(int));` que reordena los elementos de una pila de tal forma que quedan los que no satisfacen el predicado `pred()` en el fondo y los que si lo satisfacen arriba. Por ejemplo si $P = \{1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9\}$ y `pred()` es `bool par(int x) { return x%2==0; }` entonces debe quedar $P = \{4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9\}$. *Restricciones:* el algoritmo debe ser *estable*, es decir los elementos que satisfacen `pred()` deben quedar en el mismo orden entre sí, y lo mismo para los que no satisfacen `pred()`. Se pueden usar dos estructuras auxiliares (listas, pilas o colas).

[Ej. 3] [operativos (total = 80 puntos)]

a) **[rec-arbol (40 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

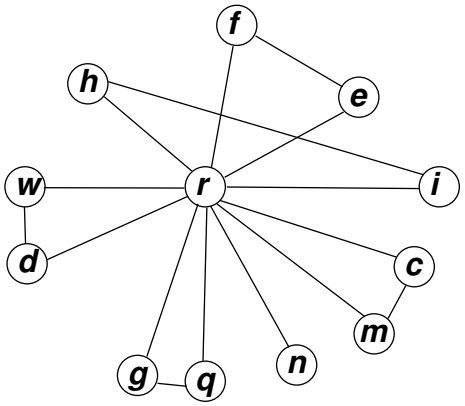
- `ORD_PRE` $=\{A, B, C, E, F, Z, Y, W, G, D\}$,
- `ORD_POST` $=\{B, E, Z, Y, W, F, G, C, D, A\}$.

b) **[misc-arbol (20pt)]:** Dado el árbol `(b t (u (z x y) w))`,

1) Cuál es el nodo de profundidad 2 que está a la izquierda de `w`?
2) Particione el árbol con respecto al nodo `u`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

c) **[colorear-grafo (20 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] **[Preguntas (total = 20 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(q e (r w) s t)`. ¿Cuál de las opciones es verdadera?

- ☐ Tiene 4 nodos a profundidad 1 y 5 hojas.
- ☐ Tiene dos raíces y 4 hojas.
- ☐ Tiene 4 nodos a profundidad 1 y 4 hojas.
- ☐ Tiene 5 nodos a profundidad 1 y 4 hojas.

¿Cuál es el tiempo de ejecución de la operación de *re-heap* en montículos?

- ☐ Caso promedio $O(\log n)$.
- ☐ Siempre $O(\log n)$.
- ☐ Caso promedio $O(n \log n)$.
- ☐ Siempre $O(n \log n)$.

Dado el árbol binario `(z (a b q) r)`, ¿cuál de las siguientes opciones es verdadera?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n + B)$
- ☐ $O(n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Examen Final. [27 de Julio de 2006]

[Ej. 1] [Clases (30 puntos)]

Escribir los siguientes métodos del TAD `btree`: `insert(p,x)`, `erase(p)`, `find(x)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [Programación (total = 70 puntos)]

a) **[incluido (45 puntos)]**
Escribir un predicado `bool incluido(tree<int>&A,tree<int>&A);`, el cual retorna verdadero si la *estructura* del árbol del nodo `B` está *"incluida"* dentro del árbol `A`, independientemente de las etiquetas de los nodos correspondientes. Por ejemplo, si tenemos los árboles `T1=(z q r (t u v))`, `T2=(x y (p w s))`, `T3=(a c d (e j))`, entonces tenemos que `T3` está incluido en `T1`, pero `T2` no está incluido en `T1`.

b) **[separa (25 puntos)]**
Escribir una función `void separa(queue<int>&Q, queue<int>&Qt,queue<int>&Qf,bool (*pre)(int));` que separa los valores de la cola `Q` en dos colas `Qf`, `Qt`, tal que los valores que satisfacen el predicado `pred()` van a la cola `Qt` mientras que los que no lo satisfacen van a la cola `Qf`. Por ejemplo, si `Q={1,3,2,4,3,2,5}` y el predicado es `bool impar(x) {return x%2; }`, entonces después de `separa(Q,Qt,Qf)` debe quedar `Qt={1,3,3,5}`, `Qf={2,4,2}`.
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original. No se deben usar estructuras auxiliares. El algoritmo debe ser *"estable"*.

[Ej. 3] [LIBRES - operativos (total = 80 puntos)]

a) **[rec-arbol (20 pt)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, A, B, C, J, M, Q, N, P, K, D\}$,
- `ORD_POST` $=\{A, Q, M, P, N, J, K, C, D, B, Z\}$.

b) **[huffman (20 pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `BENEDICTO` $P(B) = 0.2, P(O) = 0.2, P(N) = 0.2, P(E) = 0.1, P(D) = 0.1, P(I) = 0.1, P(C) = 0.05, P(T) = 0.05$ Calcular la longitud promedio del código obtenido.

c) **[misc-arbol (20 pt)]:** Dado el árbol `(z q (r t s) p)`,

1) Cuál es el nodo que está a la vez a la izquierda de `p` y a la derecha de `q` y es antecesor propio de `t`?
2) Particione el árbol con respecto al nodo `t`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda

d) **[heap-sort (20 pt)]** Dados los enteros $\{12, 11, 14, 8, 16, 3, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 4] [LIBRES - preguntas (total = 20 pt, 5pt/preg)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Dadas las funciones $T_1(n) = 0.5n + \sqrt{n}$, $T_2(n) = 0.2n^2 + 2.1 \log n$, $T_3(n) = 3n! + 5.4n^3$ y $T_4(n) = n^{1/2}$ decir cuál de los siguientes ordenamientos es el correcto

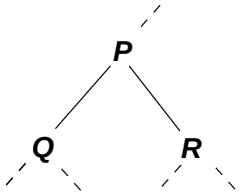
- ☐ $T_4 < T_3 < T_2 < T_1$
- ☐ $T_4 < T_1 < T_2 < T_3$
- ☐ $T_2 < T_1 < T_4 < T_3$
- ☐ $T_3 < T_4 < T_1 < T_2$

b) El tiempo de ejecución para el algoritmo de clasificación por montículos (*"heapsort"*) es $O(n \log n)$ (n es el número de elementos a ordenar) ...

- ☐ ... siempre.
- ☐ ... a veces.
- ☐ ... nunca.
- ☐ ... en el mejor caso.

c) El montículo es un árbol binario que satisface la condición de ser *"parcialmente ordenado"*. Si P, Q, R son las etiquetas del nodo y sus dos hijos, la condición de parcialmente ordenado se expresa como (*Nota: Consideramos un montículo "minimal"*):

- ☐ $Q + R \leq \infty$.
- ☐ $Q \leq P \leq R$.
- ☐ $P \leq 1/2(Q + R)$.
- ☐ $P \leq Q, R$.



d) ¿Cuál es el tiempo de ejecución del procedimiento de clasificación por *incrementos decrecientes* (*shell-sort*) en el caso promedio?

- ☐ $O(n^{1.3})$
- ☐ $O(n^{1.5})$
- ☐ $O(\log n)$
- ☐ $O(n!)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 1A. [20 de abril de 2006]

[Ej. 1] [Clases (30 puntos)]

a) Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores **ó** arreglos. Las funciones a implementar son `insert(p,x)`, `erase(p)`. Observaciones:

- En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase *"avanzada"*, **debe** declarar completamente las partes privadas de la clase `list`. **NO** es necesario implementar las clases `cell` ni `iterator`.

b) Escribir la implementación en `C++` del `TAD PILA` o del `TAD COLA` (a elección).

[Ej. 2] [Programación (total = 45 puntos)]

a) **[ascendente (15 puntos)]**
En ciertas aplicaciones interesa separar las corridas ascendentes en una lista de números $L = (\ a_1,\ a_2,\ ...,\ a_n\ )$, donde cada corrida ascendente es una sublista de números consecutivos a_i, a_{i+1}, ..., a_{i+k}, la cual termina cuando $a_{i+k} > a_{i+k+1}$, y es ascendente en el sentido que $a_i \leq a_{i+1} \leq ... \leq a_{i+k}$. Por ejemplo, si $n = 10$ y la lista es L = (0,5,6,9,4,3,9,6,5,5,2,3,7), entonces hay 6 corridas ascendentes, a saber: (0,5,6,9), (4), (3,9), (6), (5,5) y (2,3,7).
Consigna: usando las operaciones de la clase lista, escribir una función
`int ascendente(list<int> &L,list<list<int> > &LL)` en la cual, dada una lista de enteros `L`, almacena cada corrida ascendente como una sublista en la lista de listas `LL`, devolviendo además el número z de corridas ascendentes halladas. *Restricciones:* a) El tiempo de ejecución del algoritmo debe ser $O(n)$, b) La lista de listas `LL` inicialmente está vacía, c) No usar otras estructuras auxiliares.

b) **[apply-map (15 puntos)]** Escribir una funcion
`void apply_map(list<int> &L, map<int,int> &M, list<int> &ML)` que, dada una lista `L` y una correspondencia `M` retorna por `ML` una lista con los resultados de aplicar `M` a los elementos de `L`. Si algún elemento de `L` no está en el dominio de `M` entonces el elemento correspondiente de `ML` no es incluido. Por ejemplo, si

$$\texttt{L} = (1, 2, 3, 4, 5, 6, 7, 1, 2, 3)$$
$$\texttt{M} = \{(1, 2), (2, 3), (3, 4), (4, 5), (7, 8)\}$$

entonces después de hacer `apply_map(L,M,ML)`, debe quedar

$$\texttt{ML} = (2, 3, 4, 5, 8, 2, 3, 4)$$

Restricciones: No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$, donde n es el número de elementos en la lista (asumiendo que las operaciones usadas de correspondencia son $O(1)$).

c) **[inverse (15 puntos)]** Dada una correspondencia `M` y asumiendo que es *invertible* o *biunívoca* (esto es, todos los valores del contradominio son distintos), la correspondencia *"inversa"* `N` es aquella tal que, si `y=M[x]`, entonces `x=N[y]`. Por ejemplo, si `M={(0,1),(1,2),(2,0)}`, entonces la inversa es `N={(1,0),(2,1,(0,2))}`. *Consigna:* Escribir una función
`bool inverse(map<int,int> &M,map<int,int> &N)` tal que, si `M` es invertible, entonces retorna true y `N` es su inversa. En caso contrario retorna falso y `N` es la correspondencia *"vacía"* (sin asignaciones).

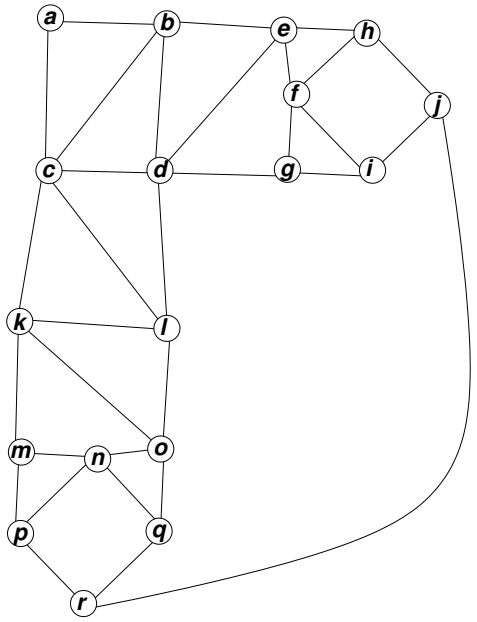
Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos



[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico partiendo del vértice a y recorriendo los vértices en orden alfabético $b, c, d, ...$ ¿La coloración obtenida es óptima? Justifique.

[Ej. 4] **[Preguntas (total = 20 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) ¿Cuál es el tiempo de ejecución de `x=M[key]` para correspondencias implementadas con vectores ordenados?

- ☐ ... $O(n)$
- ☐ ... $O(\sqrt{n})$
- ☐ ... $O(1)$
- ☐ ... $O(\log n)$

---

b) ¿Cuál es el tiempo de ejecución para `L.insert(p,x)` en listas implementadas por arreglos?

- ☐ ... $O(\sqrt{n})$
- ☐ ... $O(\log n)$
- ☐ ... $O(1)$
- ☐ ... $O(n)$

---

c) Dado el siguiente fragmento de código

```
map<int,int>M;
M[0]=1;
int i = M[3];
```

¿Cuál de las siguientes aseveraciones es verdadera?

- ☐ ... `M` no se modifica y devuelve `i=3`.
- ☐ ... `M` se modifica y devuelve `i=0`.
- ☐ ... Da un error.
- ☐ ... `M` se modifica y devuelve `i=1`.

---

d) Dadas las funciones

- $T_1(n) = 2n^2 + 1000\,n + \sqrt{5n} + 2^{20}$,
- $T_2(n) = n! + 2n^5 + 5n^2 + 3n^{5/2}$,
- $T_3(n) = 3n^2 + 2n^3 + n^{3/2} + 3\,\log n$,
- $T_4(n) = 4n^{1.5} + 3n^{2.5} + n\log n + 6\sqrt{n}$.

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: 1A. [23 de mayo de 2006]

[Ej. 1] **[clases (20 puntos)]** Escribir la implementación en `C++` del `TAD Arbol Ordenado Orientado` (clase `tree`). Para la clase `tree` implemente: `insert(p,x)`, `find(x)` y `clear()`. Para la clase `iterator` implemente `operator!=`, `operator==`, `lchild()` y `right()` (u `operator++`). Declare además la clase `cell`. Observaciones:

- **Debe** declarar los miembros privados de las clases a declarar o implementar. Ayuda: use la figura 1.
- Si opta por la interfase *"estilo"* STL, implemente la forma **prefija** del operador `operator++` (`++p`).

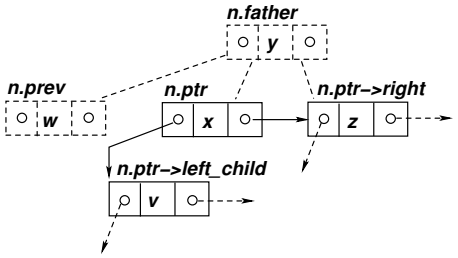


Figura 1: Entorno local de un iterator sobre árboles.

[Ej. 2] **[programación (total = 45 puntos)]**

a) **[longest-path (25 puntos)]**
Escribir una función `void longest_path(btree<int> &A,list<int> &L);` que, dado un árbol binario `A` retorna en `L` la lista de valores en el camino más largo en el árbol. Por ejemplo, si `A=(3 (4 2 .) (6 7 (8 9 5)))`, entonces `longest_path` debe retornar `L=(3 6 8 9)`. Si hay más de un camino con la longitud más larga, entonces puede retornar cualquiera de ellos. Por ejemplo, en el caso anterior puede retornar también `L=(3 6 8 5)`. *Notas:* se puede usar `list<int>::size()` y todas las funciones de lista en STL sin restricciones (asignación, constructor por copia...).

b) **[path-of-largest (20 puntos)]**
Escribir una función `void path_of_largest(tree<int> &A,list<int> &L);` que, dado un árbol ordenado orientado `A`, retorna en `L` el camino que se obtiene recorriendo el árbol hacia abajo, *siempre por el hijo más grande*. Por ejemplo, si `A=(3 (4 6 (7 8 9)) (5 2 4 6))` entonces `path_of_largest` debe retornar `L=(3 5 6)`. Si para un nodo el valor más grande de los hijos está repetido, entonces el camino puede seguir por cualquiera de ellos. Por ejemplo si `A=(3 (4 5 6) 4 2)` entonces `path_of_largest()` puede retornar `L=(3 4 6)` o `L=(3 4)`.

[Ej. 3] **[operativos (total = 25 puntos)]**

a) **[huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PAPELERAS`
$P(P) = 0.1, P(A) = 0.1, P(L) = 0.3, P(E) = 0.1, P(R) = 0.2, P(B) = 0.05, P(Q) = 0.05, P(S) = 0.1$
Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.

b) **[rec-arbol (5 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, K, Y, W, Q, M, N, T, U, V\}$;
- `ORD_POST` $=\{Y, W, K, M, V, U, T, N, Q, Z\}$.

c) [**construir-arbol (5 ptos)**] Escribir la secuencia de sentencias necesarias para construir el arbol binario `(3 (4 . 2) (1 . 7))`. (*Restricciones:* No usar `find()`).

d) [**modificar-arbol (5 ptos)**] Escribir la secuencia de secuencia necesarias para modificar el árbol ordenado orientado `A = (3 (4 5 6) (7 8 9))` de manera que se convierta en `A = (3 (4 5 6) (7 8 10 9))`. (Es decir, insertar el 10 entre el 8 y el 9). (*Restricciones:* No usar `find()`).

[Ej. 4] [Preguntas (total = 10 puntos, 2.5 puntos por pregunta)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) Dado el árbol `(a (b e f) (c (g h)) d))`, marcar cuáles de los siguientes son *caminos* válidos en el árbol.

- ☐ ... `(e b f)`
- ☐ ... `(f b a)`
- ☐ ... `(c g h)`
- ☐ ... `(a c g h)`
- ☐ ... `(d)`

---

b) ¿Cuál es el tiempo de ejecución para `splice(to,from)` en árboles binarios?

- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
- ☐ ... $O(\sqrt{n})$
- ☐ ... $O(1)$

---

c) Dado el árbol `a=(1 3 (5 7 (9 11)))`, después de aplicar las siguientes sentencias:

```
n = a.find(5);
a.erase(n);
```

¿Como queda el árbol?

- ☐ ... `a=(1 3 (9 7 11))`
- ☐ ... `a=(1 3 (. 7 (9 11)))`
- ☐ ... `a=(1 3)`
- ☐ ... `a=(1 3 (7 (9 11)))`

---

d) Marcar cuáles de los siguientes son *árboles binarios completos* (ABC):

- ☐ ... `(1 2 (3 . 5))`
- ☐ ... `(1 2 3)`
- ☐ ... `(1 . (3 4 5))`
- ☐ ... `(1 2 (3 4 5))`
- ☐ ... `(1 (7 8 9) (3 4 5))`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litora
Facultad de Ingeniería y Ciencias Hídrica
Departamento de Informátic
Algoritmos y Estructuras de Dato

Algoritmos y Estructuras de Datos.
3er Parcial. Tema: 1A. [22 de junio de 2006]

[Ej. 1] [clases (20 puntos)]

a) **[merge-sort (5pts)]** Implemente el algoritmo de ordenamiento por fusión para listas `void merge_sort(list<int> &L,bool (*comp)(int,int))`. Asuma que dispone de las funciones auxiliares (*NO es necesario implementarlas*):

- `split(L,L1,L2)` que separe una lista `L` en dos listas `L1` y `L2`, dejando a `L` vacía.
- `join(L1,L2,L,comp)` que fusiona las listas **ordenadas** `L1` y `L2` en una lista `ordenada L` utilizando la *función de comparación* `comp`, dejando a `L1` y `L2` vacías.

b) **[set-abb (15pts)]** Implemente los siguientes métodos de la clase `set<>` por árbol binario de búsqueda: `begin()` `find(x)`, `insert(x)`.

[Ej. 2] [programación (total = 45 puntos)]

a) **[es-simétrica (15 puntos)]**. Considere la correspondencia de adyacencia simétrica `map<int,set<int>> G`, la cual asigna a cada vértice de un grafo, el subconjunto de sus vértices adyacentes (o primera capa de vecinos). Por ejemplo, la correspondencia de adyacencias G_1 para el grafo G de 6 vértices y 7 aristas mostrado en la Fig. es la indicada.

```
0  --> {1, 4, 5}                          0---1   2
1  --> {0, 4}                         G   |\  |  /|
2  --> {3, 4}                             | \ | / |
3  --> {2, 4}                             |  \|/  |
4  --> {0, 1, 2, 3}                       5   4---3
5  --> {0}
```

Al decir que G es una correspondencia simétrica, significa decir que para cada arista `(i,j)` presente en G (con `i` distinto de `j`), entonces también debería estar presente la arista `(j,i)`, para todo $i, j \in V$, donde V es el conjunto de vértices del grafo. Pero, podría suceder, que la correspondencia ingresada no sea completamente simétrica, por ejemplo, la correspondencia de adyacencias G_2 no es completamente simétrica.

```
0  --> {1, 4}
1  --> {0, 4}
2  --> {4}
3  --> {2, 4}
4  --> {0, 2, 3}
5  --> {0}
```

Consigna: escribir la función `bool es_simetrica (map<int,set<int> > &G)` que devuelva `true` si G es una correspondencia de adyacencia simétrica y falso en caso contrario.

b) **[de-arista-a-adyacencia (15 puntos)]**. Suponga que se ingresan las aristas de un grafo G mediante un `list<pair<int,int>> L` de pares de enteros, donde cada par representa los vértices de cada arista. Por ejemplo el grafo F lo describimos mediante la lista de aristas
$L = \{(0,2),(2,5),(3,4),(4,1),(0,3),(2,4),(5,1),(0,4),(5,4),(6,5),(6,1)\}$, por lo que la correspondencia de adyacencias simétrica `map<int,set<int>> F` estará dada por

```
0  --> {2, 3, 4}
1  --> {4, 5, 6}              0---2---5---6
2  --> {0, 4, 5}              |\  |  /|  /
3  --> {0, 4}                 | \ | / | /  F
4  --> {0, 1, 2, 3, 5}        |  \|/  |/
5  --> {1, 2, 4, 6}           3---4---1
6  --> {1, 5}
```

Consigna: escribir la función
`void de_arista_a_adyacencia (list<pair<int,int>> &L, map<int,set<int>> &F)` la cual, dada la lista `L` de aristas, devuelva la correspondencia de adyacencia *simétrica* `F`.

c) [**cuenta-aristas (15 puntos)**]. Considere la correspondencia de adyacencia simétrica `G <int,set<int,int>>`, la cual asigna a cada vértice de un grafo, el subconjunto de sus vértices adyacentes (o primera capa de vecinos). **Consigna**: escribir la función `int cuenta_aristas(map<int,set<int> > &G)` que devuelve el número de aristas presentes en la correspondencia de adyacencia simétrica G. Por ejemplo, para el grafo G debe retornar 7 y para el grafo F debe retornar 11.

[Ej. 3] [**operativos (total = 25 puntos)**]

a) [**heap-sort (10 pts)**] Dados los enteros $\{0, 4, 7, 1, 2, 12, 9\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

b) [**open-hash (5 pts)**] Considere una tabla de dispersión **abierta** con 4 cubetas para almacenar strings con la función de dispersión `int hash(std::string &s) { return s.size(); }`. Inserte las palabras de la frase `TRES GRANDES TIGRES TRAGABAN TRES GRANDES PLATOS DE TRIGO`.

c) [**abb (5 pts)**] Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 0, 11\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar lo elementos 12, 11. en ese orden.

d) [**hash-dict (5 pts)**] Insertar los números $0, 13, 23, 6, 5, 33, 15, 2, 25$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ y estrategia de redispersión lineal.

[Ej. 4] [**Preguntas (total = 10 puntos)**] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) ¿Cuál de los siguientes es el resultado correcto de ordenar la secuencia $\{-3, 5, 3, 2, -2, 1, 2, -3\}$ por la relación d orden débil $|a| < |b|$?

- ☐ ... $\{1, 2, -2, 2, -3, 3, -3, 5\}$
- ☐ ... $\{-3, -3, -3, 1, 2, 2, 3, 5\}$
- ☐ ... $\{1, 2, 2, -2, 3, -3, -3, 5\}$
- ☐ ... $\{1, -2, 2, 2, -3, -3, 3, 5\}$

---

b) ¿Cuál es el número de *intercambios* en el método de clasificación por selección?

- ☐ ... $O(\log n)$
- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$

---

c) El tiempo de ejecución de *"quick-sort"* en el peor caso es

- ☐ ... $O(n)$
- ☐ ... $O(1)$
- ☐ ... $O(n^2)$
- ☐ ... $O(\log n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio Globalizador. [29 de Junio de 2006]

[Ej. 1] [Clases (20 pts)]

- **TAD's 1er Parcial:** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores. Las funciones a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `clear()`.
- **TAD's 2do Parcial:** Escribir la implementación en `C++` del `TAD Arbol Ordenado Orientado` (clase `tree`) implementado con celdas enlazadas por punteros. Las funciones a implementar son `insert(p,x)`, `retrieve(p)`, `erase(p)`, `begin()`, `end()`, `clear()`. No es necesario implementar las clases `iterator` y `cell`.
- **TAD's 3er Parcial:** Escribir la implementación en `C++` de las siguientes funciones del *TAD conjunto por vectores de bits* (clase `set`): `insert(x)`, `find(x)`, `erase(x)`. `set_difference(set &A,set &B,set &C)`

[Ej. 2] [Programación (total = 50 pts)]

a) **[purga (10 pts)]**
Escribir una función `void purga(list <T> & L)` que purga los elementos repetidos de una lista usando un conjunto como estructura auxiliar. Si un elemento está repetido varias veces, entonces en la lista purgada *debe quedar sólo la primera ocurrencia*. Por ejemplo, si L =(1,5,7,3,2,4,3,1,7,1,3,9) entonces, después de hacer `purga(L)` debe quedar L =(1,5,7,5,2,4,9). *Restricciones:* El algoritmo debe ser *estable* es decir que los elementos que quedan deben mantener la misma posición relativa. Además, el algoritmo debe ser $O(n)$.

b) **[es-completo (20 pts)]**
Escribir una función predicado `bool es_completo(btree<int> &)`, la cual retorna verdadero si el árbol binario es completo.

c) **[grado (20 pts)]**
Escribir una función `void degree(map<string,set<string> > &G,map<string,int> &dgr);` que dado un grafo simétrico G cuyos vértices están identificados por strings retorna por `dgr` el *grado* de cada nodo (número de aristas conectadas que contienen al vértice).

[Ej. 3] [operativos (total = 20 pts)]

a) **[lisp (5 pts)]:** Dibujar los árboles cuyas notaciones Lisp son:
- Arbol ordenado orientado: `(a (b d (e f g h)) (c z))`
- Arbol binario: `(z d (k . (j (l m n) .)))`

b) **[arbol-expr (5 pts)]** Escribir el árbol que corresponde a la expresión `(a+b)/(x*y)+z-d/r`. Dar la notación prefija y postfija.

c) **[ohash (5 pts)]** Insertar los enteros (15,10,9,19,9,29,28,17,46) en una tabla de dispersión abierta con función de dispersión `h(x) = x % B` con `B`=10 cubetas.

d) **[rec-arbol (5 pts)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $=\{A, Z, X, Y, M, W, R, T, U, V\}$,
- `ORD_POST` $=\{Y, M, X, Z, T, U, V, R, W, A\}$.

[Ej. 4] [Preguntas (total = 10 pts, 2.5pts por pregunta)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Dada la lista `L=(1,5,7,9,4)`. Cuál de las proposiciones es válida después de hacer las operaciones

```
q = L.begin();
p=q++; p++; q++;
p = L.erase(p);
```

- ☐ ... `*p=1` y `q` es inválido.
- ☐ ... `*p=7` y `*q=7`.
- ☐ ... `*p=7` y `q` es inválido.
- ☐ ... `*p=9` y `*q=5`.

b) Dado el AOO `A=(1 5 (3 4 2 7))`. Cuál de las proposiciones es válida después de hacer las operaciones

```
p = A.find(3);
q = A.find(7);
p = A.erase(p);
p = A.insert(p,8);
```

- ☐ ... `A=(1 5 (8 4 2 7))` y `q` es inválido.
- ☐ ... `A=(1 5 8)` y `*q=7`.
- ☐ ... `A=(1 5 8)` y `q` es inválido.
- ☐ ... `A=(1 5 (8 4 2 7))` y `*q=7`.

c) ¿Cuál listado aplicado a un árbol binario de búsqueda da los elementos ordenados?

- ☐ ... Orden previo.
- ☐ ... Orden simétrico.
- ☐ ... Orden posterior.
- ☐ ... Orden antisimétrico.

d) ¿Cuál es el tiempo de ejecución de `splice(p,q)` para árboles?

- ☐ ... $O(1)$
- ☐ ... $O(n)$
- ☐ ... $O(1/n)$
- ☐ ... $O(\log_2 n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [06 de Diciembre de 2007]

[Ej. 1] [Clases (20 puntos)]

Escribir los siguientes métodos del TAD `btree`: `insert(p,x)`, `erase(p)`, `find(x)`. Escribir las declaraciones de la clase y los componentes necesarios para implementar las funciones indicadas.

[Ej. 2] [programacion (total = 80 puntos)]

a) **[encuentra (35 puntos)]** Escribir una función `bool encuentra(list<int> &L1, list<int> &L2, list<int> &indx);` que,

- Retorna `true` o `false` dependiendo de si `L1` es una sublista o no de `L2`.
- En caso de que si lo sea, retorna en `indx` los índices de los elementos de `L2` que forman `L1`, si no `indx` debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si `L2`=$\{13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10\}$ y `L1`=$\{13, 9, 9, 6, 2, 14\}$ entonces `encuentra` debe retornar `true`, e `indx`=$\{0, 1, 4, 5, 7, 9\}$. Si `L1`=$\{8, 9, 13\}$ debe retornar `false` e `indx`=$\{\}$. Nota: Los índices en `indx` deben ser estrictamente crecientes. *Restricciones:* No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$.

b) **[encuentra-suma (35 puntos)]** Escribir una función `tree<double>::iterator encuentra_suma(double s, tree<double> &A);` que retorna el nodo `m` del árbol orientado `A` tal que la suma de las etiquetas de todos los nodos descendientes de `m` (incluyendo a `m`) es `s`. Si no existe un tal nodo, entonces debe retornar Λ. Si hay varios nodos que cumplen la condición basta con que retorne uno de ellos. Por ejemplo, para `A=(10 21 (6 (8 7) (11 (12 3) 6) (4 6 5)))`

- `encuentra_suma(32,A)` retorna el nodo 11,
- `encuentra_suma(15,A)` puede retornar el nodo 8, el 4 o el 12.
- `encuentra_suma(9,A)` retorna Λ

Restricciones: El algoritmo debe ser $O(n)$ donde n es el número de nodos en el árbol.

c) **[reordena-pila (10 puntos)]** Escribir una función `void reordena(stack<int> &P,bool (*pred)(int));` que reordena los elementos de una pila de tal forma que quedan los que no satisfacen el predicado `pred()` en el fondo y los que si lo satisfacen arriba. Por ejemplo si $P = \{1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9\}$ y `pred()` es `bool par(int x) { return x%2==0; }` entonces debe quedar $P = \{4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9\}$. *Restricciones:* el algoritmo debe ser *estable*, es decir los elementos que satisfacen `pred()` deben quedar en el mismo orden entre sí, y lo mismo para los que no satisfacen `pred()`. Se pueden usar dos estructuras auxiliares (listas, pilas o colas).

[Ej. 3] [operativos (total = 80 puntos)]

a) **[rec-arbol (40 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

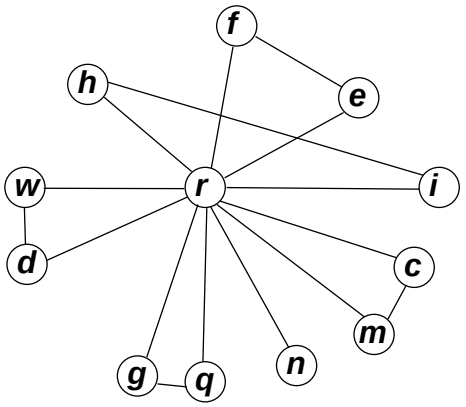
- `ORD_PRE` $=\{A, B, C, E, F, Z, Y, W, G, D\}$,
- `ORD_POST` $=\{B, E, Z, Y, W, F, G, C, D, A\}$.

b) **[misc-arbol (20pt)]:** Dado el árbol `(b t (u (z x y) w))`,

1) Cuál es el nodo de profundidad 2 que está a la izquierda de `w`?
2) Particione el árbol con respecto al nodo `u`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

c) **[colorear-grafo (20 ptos)]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] **[Preguntas (total = 20 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

Sea el árbol `(q e (r w) s t)`. ¿Cuál de las opciones es verdadera?

- ☐ Tiene 4 nodos a profundidad 1 y 5 hojas.
- ☐ Tiene dos raíces y 4 hojas.
- ☐ Tiene 4 nodos a profundidad 1 y 4 hojas.
- ☐ Tiene 5 nodos a profundidad 1 y 4 hojas.

¿Cuál es el tiempo de ejecución de la operación de *re-heap* en montículos?

- ☐ Caso promedio $O(\log n)$.
- ☐ Siempre $O(\log n)$.
- ☐ Caso promedio $O(n \log n)$.
- ☐ Siempre $O(n \log n)$.

Dado el árbol binario `(z (a b q) r)`, ¿cuál de las siguientes opciones es verdadera?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

- ☐ $O(n^2/B)$
- ☐ $O((n/B)^2)$
- ☐ $O(n + B)$
- ☐ $O(n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

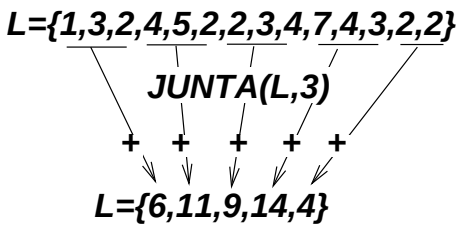
Algoritmos y Estructuras de Datos.
Examen Final. [20 de Diciembre de 2007]

[Ej. 1] [Clases (mínimo 60 %)]

Escribir los métodos que se indican del TAD `list<>` `insert(x,p)`, `erase(p)`, `retrieve(p)/*p`, `next()/p++`, `list()` y `~list()` implementados por celdas enlazadas por punteros o cursores.

[Ej. 2] [Programación (mínimo 40 %)]

a) **[junta]**
Escribir una función `void junta(list<int>&L,int n);` que dada una lista `L`, agrupa de a `n` elementos dejando su suma (ver figura).
Restricciones: **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.



b) **[ord-nodo]**
Escribir una función predicado `bool ordnodo(tree<int> &A);` que verifica si cada secuencia de hermanos del subárbol del nodo `n` (perteneciente al árbol ordenado orientado `A`) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol `(3 5 (6 1 3) (7 4 5))` debería retornar `true`, mientras que para `(3 9 (6 1 3) (7 4 2))` debería retornar `false`, ya que las secuencias de hermanos `(9 6 7)` y `(4 2)` NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

c) **[es-completo]**
Escribir una función predicado `bool es_completo(btree<int> &A);` la cual retorna verdadero si el árbol binario es completo.

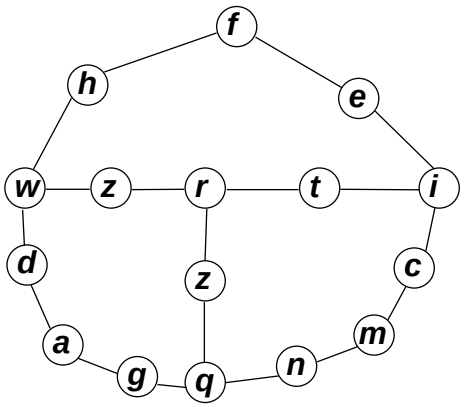
[Ej. 3] [operativos (mínimo 70 %)]

a) **[rec-arbol]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $=\{C, Z, Q, R, A, M, P, K, L, T\}$,
- `ORD_POST` $=\{Z, A, P, M, K, L, R, T, Q, C\}$.

b) **[misc-arbol]:** Dado el árbol `(c q (t (r u (v z))))`,
1) Cuál es el nodo que está a la vez a la izquierda de `v` y no es descendiente de `r`?
2) Particione el árbol con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

c) **[colorear-grafo]**

Colorear el siguiente grafo, utilizando una estrategia heurística para tratar de usar el menor número de colores posibles.



[Ej. 4] [Preguntas (mínimo 70 %)]

a) Sea `L` una lista conteniendo los elementos $(1, 3, 4, 2, 5, 6)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = L.erase(q);
p++;
q = p;
q++;
```

indique para cada uno de los iterators `p`, `q` si es válido o inválido y en el primer caso el valor que almacena (es decir `*p` y `*q`).

b) Explique que quiere decir la propiedad de "Transitividad" de $O()$.

c) ¿Cuál es la "Regla del producto" para $O()$?.

d) ¿Cuándo se dice que dos tasas de crecimiento del orden del tiempo de ejecución son equivalentes?

e) ¿Cual es la signatura de la función `insert()` en listas STL?. Diga que es cada una las variables y cual es el tipo de retorno?. En caso de que haya varias signaturas explique una de ellas.

f) Explique con un ejemplo la condición de "prefijo" de un código.

g) ¿Cuál es el costo de inserción en tablas de dispersión abiertas con listas desordenadas en el peor caso?. Diga cual es ese caso.

h) ¿Que condiciones debe satisfacer un árbol binario para ser "árbol binario de búsqueda"?

i) ¿Que quiere decir que un algoritmo de ordenamiento sea "estable"?.

j) ¿Por qué se les dice "lentos" a ciertos algoritmos de ordenamiento?. Enumere los algoritmos lentos que conoce. Enumere los algoritmos rápidos que conoce.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. Tema: 1A. [21 de abril de 2005]

[Ej. 1] **[Clases (30 puntos)]** Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por punteros **ó** cursores **ó** arreglos. Las funciones a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `clear()`. Observaciones:

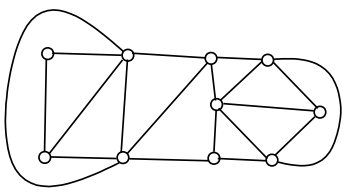
- En caso de optar por escribir la interfase *"básica"*, debe escribir todas las declaraciones necesarias de la clase, tanto en la parte privada como pública.
- En caso de optar por la interfase *"avanzada"*, debe declarar e implementar completamente las partes privadas de la clase `list` e `iterator`.

[Ej. 2] **[Programación (total = 50 puntos)]**

a) **[es-permutacion (25 puntos)]**
Una correspondencia es una *"permutacion"* si el conjunto de los elementos del dominio (las claves) es igual al del contradominio (los valores). De esta manera se puede interpretar a la correspondencia como una operación de permutación de las claves (de ahí su nombre). Por ejemplo `M={1->5,3->7,9->9,7->1,5->3}` es una permutación ya que tanto las claves como los valores son el conjunto `{1,3,5,7,9}`. Para determinar si una correspondencia es una permutación (y sin usar contenedores de tipo *"conjunto"*, que serán vistos más adelante), se puede utilizar el siguiente procedimiento. Construir una correspondencia `M2` que mapea los valores del contradominio a las claves, es decir, para cada par de asignación `(k->v)` tal que `M[k] = v`, entonces se debe asignar el par `(v->k)` en `M2`. Notar que si la correspondencia no es biunívoca, es decir si varias claves son asignadas a un mismo valor, entonces a `v` se le asignará alguna de esas claves. Cual de ellas es asignada dependerá del orden en que se recorren las asignaciones de `M`. Notar que si `M` es biunívoca, entonces `M2` es la inversa de `M`. Entonces, si la composición de `M` con `M2` es la identidad es decir para cada `k` en las claves de `M`, si `M2[M[k]] = k`, entonces `M` es una permutación. Consigna: escribir una función predicado `bool es_permutacion(map<int,int> &M)` que retorna `true` si `M` es una permutación y `false` si no lo es.

b) **[list-sort (25 puntos)]**
Escribir una función `void sort(list<int> &L);`, que ordena los elementos de `L` de menor a mayor. Para ello utilizar el siguiente algoritmo simple, utilizando una lista auxiliar `L2`: ir tomando el menor elemento de `L`, eliminarlo de `L` e insertarlo al final de `L2` hasta que `L` este vacía. Cual es el tiempo de ejecución en función de la cantidad de elementos `n` de `L`?

[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. ¿La coloración obtenida es óptima? Justifique.



[Ej. 4] **[Preguntas (total = 15 puntos, 5puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

a) Considere la función:

```
bool is_mapped(map<int,int> &M,int key,int val) {
  return /* Esta key -> val en M? ... */; }
```

que debe retornar `true` si el par de asignación `(key,val)` está en la correspondencia `M`. ¿Cuál es la expresión correcta que refina el seudocódigo?

- ☐ ... `M[key]==val`
- ☐ ... `(M.find(key)!=M.end()) && M[key]==val`
- ☐ ... `M[key]==val && (M.find(key)!=M.end())`
- ☐ ... `M.find(key)->second==val`

---

b) El tiempo de ejecución de la función `find(key)` para correspondencias implementadas por vectores ordenados es ... (n es el número de asignaciones en la correspondencia)

- ☐ ... $O(1)$
- ☐ ... $O(\log n)$
- ☐ ... $O(n^2)$
- ☐ ... $O(n)$

---

c) Dadas las funciones

- $T_1(n) = 3n^3 + 2n! + \log n$,
- $T_2(n) = 3 \cdot 2^3 + n^2 + n^{1.5}$,
- $T_3(n) = 5! + 6 \cdot 2^n + 20 \cdot n^2$ y
- $T_4(n) = 2^{10} + 20n + \log_2 40$

ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. Tema: 1A. [1 de noviembre de 2007]

[Ej. 1] **[clases (20 puntos)]** Escribir la implementación en `C++` del `TAD Arbol Ordenado Orientado` (clase `tree`). Para la clase `tree` implemente: `insert(p,x)`, `find(x)` y `clear()`. Para la clase `iterator` implemente `lchild()` y `right()` (u `operator++`). Observaciones:

- **Debe declarar** los **miembros privados** de las clases a declarar o implementar. Ayuda: use la figura 1.
- Si opta por la interfase *"estilo"* STL, implemente la forma **prefija** del operador `operator++` (`++p`).

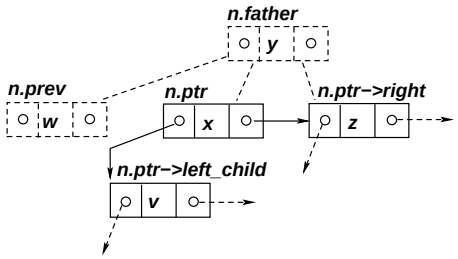


Figura 1: Entorno local de un iterator para árboles ordenados orientados.

[Ej. 2] **[programación (total = 50 puntos)]**

a) **[check-ordprev (25 puntos)]**
Escribir una función `bool check_ordprev(tree<int> &T,list<int> &L);` que, dado un árbol ordenado orientado `T` retorna `true` si la lista `L` contiene al listado en orden previo de `T`. Por ejemplo, si `T=(3 (4 2 1) 0 (6 7 (8 9 5)))` y `L={3,4,2,1,0,6,7,8,9,5}` entonces `check_ordprev` debe retornar `true` y si por ejemplo `L={3,4,2,1,0,6,7,8,9,5,1000}` o `L={3,4,2,1,0,6,7000,8,9,5}` `check_ordprev` debe retornar `false`. Una manera simple de hacerlo es iterar en el árbol en la manera habitual e ir sacando los elementos de la lista `L` si coinciden con el contenido del nodo o posición actual. Una vez que se recorrió todo el árbol se verifica en la función *"wrapper"* que la lista haya quedado vacía. Se debe retornar `true` si este es el caso. La función `check_ordprev` está definida tal que:

- si el árbol es vacío y la lista no lo es (o viceversa) `check_ordprev` es `false`,
- si lo anterior no ocurre y si el árbol es vacío `check_ordprev` es `true`,
- si no ocurre lo anterior y los contenidos del nodo y la posición de la lista actual no coinciden `check_ordprev` es `false`,
- si no ocurre lo anterior, elimino el elemento actual de la lista,
- llamo recursivamente a la función verificando que no haya llegado al fin de lista.

Nota: se pueden usar todas las funciones de lista de STL sin restricciones.

b) **[remove-leafs (25 puntos)]**
Se desea escribir una función `int remove_leafs(btree<int>&T, btree<int>::iterator n, int min_leaf_val)` que transforma un subárbol de un árbol binario, eliminando las hojas cuyo valor es menor que un cierto umbral `min_leaf_val` . A medida que las hojas son eliminadas, su valor debe ser acumulado en el padre, de manera que la suma de los valores en los nodos del subárbol se mantiene. La aplicación es recursiva. Por ejemplo si `T = (8 (7 9 2) (3 (9 (6 1 .) .) 1))` y hacemos `remove_leafs(T,T.begin(),10)` debe quedar T=(8 18 (4 16 .)). Puede pasar que si la suma de los valores del subárbol es menor que `min_leaf_val` entonces el subárbol completo es eliminado, en ese caso el valor de retorno es la suma de los valores del subárbol eliminado. Si el árbol no es eliminado entonces el valor de retorno es nulo. En resumen

- Si $n = \Lambda$, `remove_leafs` retorna 0.
- En caso contrario, se aplica `remove_leafs` a los hijos izquierdo y derecho de `n`. (Tener en cuenta que estas llamadas pueden eliminar el nodo correspondiente, por lo tanto debe prestarse atención a los iterators inválidos). El valor retornado por los hijos debe acumularse en `n`.
- Después de la aplicación recursiva de `remove_leafs` a los hijos de `n`:
  - Si `n` **no es hoja**, retornar 0.
  - Si el valor de `n` **no es menor** que el umbral `min_leaf_val`, retornar 0.
  - Finalmente, si lo anterior no se cumple, se debe **eliminar** `n` y retornar su valor.

[Ej. 3] [operativos (total = 20 puntos)]

a) **[huffman (10 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `TOMATE`
$P(T) = 0.1, P(A) = 0.1, P(M) = 0.3, P(O) = 0.1, P(E) = 0.2, P(B) = 0.05, P(Q) = 0.05, P(S) = 0.1$ Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.

b) **[rec-arbol (10 ptos)]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{P, Q, R, U, S, V, X, Y, T, W, Z, A\}$;
- `ORD_POST` $=\{Q, U, R, X, Y, V, S, A, Z, W, T, P\}$.

[Ej. 4] [Preguntas (total = 10 puntos, 3.33 puntos por pregunta)] Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) El algoritmo de Huffmann permite obtener códigos binarios para encodar mensajes utilizando árboles binarios. La longitud del código asignado a un caracter, en bits está dado por...

- ☐ ... la profundidad del nodo correspondiente en el árbol.
- ☐ ... la altura del nodo correspondiente en el árbol.
- ☐ ... el número de nodos en el subárbol que cuelga del nodo correspondiente.
- ☐ ... la etiqueta del nodo correspondiente en el árbol.

b) Dado el árbol `a=(2 1 (15 8 (4 11)))`, después de aplicar las siguientes sentencias:

```
n = a.find(15);
a.erase(n);
```

¿Como queda el árbol?

- ☐ ... `a=(2 1 (8 (4 11)))`
- ☐ ... `a=(2 1)`
- ☐ ... `a=(2 1 (4 8 11))`
- ☐ ... `a=(2 1 (. 8 (4 11)))`

c) Si `A=(2 (3 (6 4 2)) 5)` y `B=(1 7 9)` entonces ¿cómo quedan `A` y `B` después de hacer...?

```
n=A.find(6);

m=B.begin();
m=m.lchild();

m++;

m=m.lchild();

B.splice(m,n);
```

- ☐ ... `A=(2 3 5)` y `B=(1 7 (9 (6 4 2)))`
- ☐ ... `A=(2 3 (5 6))` y `B=(1 7 (9 4 2))`
- ☐ ... Da un error.
- ☐ ... `A=(2 (3 (6 4 2)) 5)` y `B=(1 7 (9 (6 4 2)))`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. Tema: 1A. [22 de noviembre de 2007]

[Ej. 1] [clases (25 puntos)]

- Escribir la implementación en `C++` del `TAD Conjunto` (clase `set`). Implemente: `insert(x)`, `find(x)`, `clear()` y `empty()`. **Observaciones**: **Debe declarar** los **miembros privados** de las clases a declarar o implementar. También debe implemetar todo método o función auxiliar que necesite. Puede elegir entre implementaciones basadas en *vectores de bits*, *listas ordenadas*, *arboles binarios de búsqueda y tablas de dispersión*.
- Implementar una función
`btree<int>::iterator abb_find(btree<int> &T, btree<int>::iterator n, int x)`
que busca en el ABB `T` el valor `x` devolviendo su posición o `T.end()` si no se encuentra.

[Ej. 2] [programación (total = 50 puntos)]

a) **[includes-all (20 puntos)]**

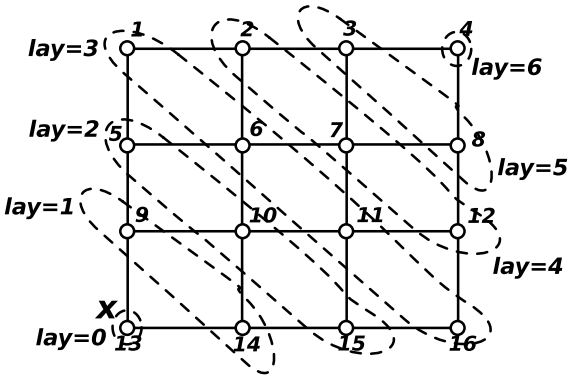
Dados n conjuntos de vector de conjuntos `vector<set<int>> A` se pide escribir una función `int includes_all(vector< set<int> > &setv);` tal que si alguno de esos conjuntos incluye a todos los otros, es decir existe algún j tal que $A_{[k]} \subset A_{[j]}$, para todo k, entonces debe retornar j. En ese caso contrario, debe retornar -1. Por ejemplo, si $A_{[0]} = \{1, 2, 3\}$, $A_{[1]} = \{1, 2, 3, 4, 5\}$, $A_{[2]} = \{3, 4, 5\}$, entonces debe retornar 1. Si $A_{[0]} = \{1, 2, 3\}$, $A_{[1]} = \{3, 4, 5\}$ entonces debe retornar -1. *Nota:* Si existen varios $A_{[j]}$ que satisfacen la condición entonces debe retorar uno cualquiera de ellos. *Sugerencia:* Notar que, si alguno de los $A_{[j]}$ satisface la condición entonces debe ser el mayor de todos (en cantidad de elementos).

b) **[mklayers (30 puntos)]**

Escribir una función
`void mklayers(vector<set<int> > &G, int x, vector<set<int> > &layers)` que dado un grafo `G` y un vértice de partida `x` determina la estructuras de capas de vecinos `layers` de `x` definida de la siguiente manera:

- La capa 0 es el conjunto `layers[0]={x}` .
- La capa 1 es el conjunto de los vecinos de `x`.
- Para `l>1` la capa `l` es el conjunto de los vecinos de los nodos en la capa `l-1` que no están en capas anteriores (0 a `l-1`). Notar que en realidad sólo hace falta verificar que no estén en las capas `l-1` y `l-2`.

Puede demostrarse que los vértices en la capa `l` son los que están a distancia `l` de `x`. Por ejemplo, dado el grafo de la figura, y partiendo del nodo `x=0` las capas son `layers[0]={0}`, `layers[1]={9,14}`, `layers[2]={5,10,15}`, `layers[3]={1,6,11,16}`, `layers[4]={2,7,12}`, `layers[5]={3,8}`, `layers[6]={4}`.



[Ej. 3] **[operativos (total = 15 puntos)]**

a) **[abb (7.5 pts)]** Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 0, 11\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 12, 4 y 11 en ese orden.

b) **[hash-dict (7.5 pts)]** Insertar los números $2, 15, 25, 8, 7, 35, 17, 4, 27$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x$ `mod` 8 y estrategia de redispersión lineal.

[Ej. 4] **[Preguntas (total = 10 puntos, 3.33 puntos por pregunta)]** Responder según el sistema *"multiple choice"*, es decir marcar con una cruz el casillero apropiado. **Atención:** Algunas respuestas son intencionalmente "descabelladas" y tienen puntajes **negativos**!!]

---

a) ¿Cuál es el tiempo de ejecución de `find(x)` en el TAD diccionario por tablas de dispersión abiertas, en el caso promedio?

☐ ... $\log(n/B)$
☐ ... $O(1 + n^2/B)$
☐ ... $O(1 + n/B)$
☐ ... $O(1 + B/n)$

---

b) ¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por árbol binario de búsqueda, en el peor caso?

☐ ... $O(n)$
☐ ... $O(n^2)$
☐ ... $O(1)$
☐ ... $O(\log n)$

---

c) ¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por tabla de dispersión cerrada, en el caso promedio?

☐ ... $O(n^2)$
☐ ... $O(1)$
☐ ... $O(\log n)$
☐ ... $O(n)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio. [29 de Noviembre de 2007]

[Ej. 1] [clases (15 pts)]

a) **[parcial-1 lista (15 pts)]** Escribir los siguientes métodos del **TAD lista**: `insert(p,x)`, `erase(p)`, `begin()`.

b) **[parcial-2 arbol-bin (15 pts)]** Escribir los siguientes métodos del **TAD árbol binario**: `find(x,p)`, `insert(p,x)`, `clear()`.

c) **[parcial-3 set (15 pts)]** Escribir los siguientes métodos del **TAD conjunto** por listas ordenadas: `insert(x)`, `find(x)`, `clear()`.

[Ej. 2] [programacion (50 pts)]

a) **[map-pre-post (15 pts)]** Escribir una función
`void map_pre_post(tree<int> &T,list<int> &L, int (*fpre)(int),int (*fpost)(int))`
que lista los valores nodales del árbol ordenado orientado `T` en una mezcla de orden previo y posterior, de acuerdo a la siguiente definición

$$\text{mpp}(\Lambda, f, g) = \text{lista vacía}$$
$$\text{mpp}(n, f, g) = f(n), \text{mpp}(n_1, f, g), ..., \text{mpp}(n_m, f, g), g(n)$$

donde $n_1 ... n_m$ son los hijos del nodo n. Por ejemplo, si `T=(1 3 (5 6 7 8))`, $f(x) = x$ y $g(x) = x + 1000$, entonces `map_pre_post(T,L,f,g)` debe dar `L=(1,3,1003,5,6,1006,7,1007,8,1008,1005,1001)`.

b) **[maxdev (20 pts)]**
Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\text{máx}_{j=1}^{n} a_j) - (\text{mín}_{j=1}^{n} a_j)$. Escribir una función `int max_dev_m(list<int> &l, int m);` que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

$$\begin{aligned}\text{max_dev_m(L)} = \ & \text{máx}\{\text{max_dev}(a_1, a_2, \ldots, a_m), \\ & \text{max_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max_dev}(a_{n-m+1}, \ldots, a_n)\} \quad (1)\end{aligned}$$

Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces `max_dev_m(L,3)` debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones.

c) **[is-mapped-set (15 pts)]** Escribir una función predicado
`bool is_mapped_set(set<int> &A,set<int> &B,int (*mapfun)(int));` que retorna verdadero si el conjunto `B` contiene los elementos de `A`, mapeados vía la función `mapfun`. Por ejemplo, si $A = \{-5, -3, 5, 10\}$ y $B = \{9, 25, 100\}$ entonces `is_mapped_set(A,B,sq)`

debe retornar `true`, donde `sq(x)` es la función que eleva al cuadrado, es decir `sq(x)=x*x;` . Mientras que si $A = \{-5, -3, 2, 10\}$ y $B = \{4, 9, 100, 130\}$, entonces debe retornar `false` ya que el $25 = (-5)^2$ no está en B y 130 no es el cuadrado de ningún elemento de A.
Sugerencia: Crear un conjunto temporario conteniendo las imágenes de los elementos de A y compararlo con B.

[Ej. 3] [operativos (25 pts)]

- **[t-exec (5 pts)]**
  Dadas las funciones
  - $T_1(n) = 5n^3 + 2n! + \log n$,
  - $T_2(n) = 2^{15} + 2 \cdot 5^n + 3 \cdot n^2$,
  - $T_3(n) = 6! + n^2 + n^{1.7}$,
  - $T_4(n) = 1.3 \cdot 2^3 + 20n + \log_2 10$.

  ordenarlas de menor a mayor.

  $T_{\square} < T_{\square} < T_{\square} < T_{\square}$

- **[huffman (5 pts)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CONEXA`
  $P(C) = 0.2, P(O) = 0.1, P(N) = 0.1, P(A) = 0.05, P(L) = 0.05, P(X) = 0.1, P(E) = 0.1, P(Q) = 0.3$ Calcular la longitud promedio del código obtenido.
- **[abb (10 pts)]** Dados los enteros $\{7, 9, 12, 8, 4, 3, 1, 2, 5, 6\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 4, 9 y 5, en ese orden.
- **[hash-dict (5 pts)]** Insertar los números $5, 18, 28, 11, 10, 38, 3, 2$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x\,\%8$ y estrategia de redispersión lineal.

[Ej. 4] [preguntas (10 pts, 2.5 por pregunta)]

a) ¿Cuál es el criterio para elegir una buena función de dispersión? ...

- ☐ Debe tratar de concentrar los elementos en pocas cubetas.
- ☐ Debe tratar de concentrar los elementos en una sóla cubeta.
- ☐ Debe tratar de concentrar los elementos en la primera cubeta.
- ☐ Debe distribuir los elementos en la forma más uniforme posible entre las cubetas.

---

b) Dado el árbol binario `(x e (d f g))`, ¿cuál de las siguientes opciones es verdadera?

- ☐ Es completo y es lleno.
- ☐ Es completo pero no lleno.
- ☐ Es lleno pero no completo.
- ☐ Ni es completo ni es lleno.

---

c) ¿Cuál es el número de niveles en un árbol binario lleno en función del número n de nodos en el árbol?

- ☐ ... $O(n \log n)$

☐ ... $O(1)$
☐ ... $O(n)$
☐ ... $O(\log n)$

---

d) Sea un tabla de dispersión abierta con B cubetas y n elementos. Asumiendo que la función de dispersión es lo suficientemente buena como para distribuir los elementos en forma uniforme entre las cubetas, el costo medio de inserción de un nuevo elemento es

☐ $O(n^2/B)$
☐ $O((n/B)^2)$
☐ $O(n + B)$
☐ $O(1 + n/B)$

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [14 de Febrero de 2008]

[Ej. 1] [Clases (mínimo 60 %)]

a) Escribir las funciones de los TAD cola y pila (clases `queue<>` y `stack<>`). Incluir el header y todas las declaraciones necesarias.

b) Escribir la función de ordenamiento por fusión `void merge_sort(list<int> &L)` para listas de enteros. Los enteros se comparan por el operador `<`.

[Ej. 2] [Programación (mínimo 40 %)]

a) [**verifica-abb**] Escribir una función predicado
`bool verifica_abb(tree<int> &A,bool (*less)(int,int));` que verifica si el subárbol de un nodo `n` verifica la condición de *árbol binario de búsqueda* con respecto a la relación de orden `less()`.

b) [**elimina-valor**]
Escribir una función `void elimina_valor(queue<int>&C, int);` que elimina todas las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C = {1,3,5,4,2,3,7,3,5}`, después de `elimina_valor(C,3)` debe quedar `C = {1,5,4,2,7,5}`. *Sugerencia: Usar una estructura auxiliar lista o cola.*
Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

[Ej. 3] [operativos (mínimo 70 %)]

a) [**heap-sort**] Dados los enteros $\{6, 8, 11, 4, 5, 9, 6, 7, 10\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.

b) [**huffman**] Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario y encodar la palabra `KATHRINA` $P(K) = 0{,}05, P(H) = 0{,}2, P(T) = 0{,}1, P(R) = 0{,}05, P(A) = 0{,}2, P(N) = 0{,}15, P(I) = 0{,}1, P(P) = 0{,}05, P(S) = 0{,}05, P(B) = 0{,}05$. Calcular la longitud promedio del código obtenido.

[Ej. 4] [Preguntas (mínimo 70 %)]

a) ¿Puede tener una correspondencia varios valores iguales del dominio, o sea varias claves iguales? (Por ejemplo `M1={(1,2),(1,3)}`) ¿Y varios valores iguales del contradominio? (Por ejemplo `M2={(1,2),(3,2)}`)

b) Que retorna la dereferenciación de un iterator sobre correspondencias (clase `map`).

c) ¿Qué ocurre si se invoca el `operator[]` sobre una correspondencia con una clave que no tiene asignación. Por ejemplo `M={(1,2),(3,4)}` y hacemos `x = M[5]`. ¿Da un error? ¿Qué valor toma `x`?

d) ¿Cuál es la condición de Arbol Binario de Búsqueda?

e) ¿Cómo se encuentra el mínimo y el máximo de los valores en un árbol binario de búsqueda? ¿Cuál es la complejidad algorítmica de esta operación (caso promedio, mejor y peor)?

f) ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si siempre es $(n+1)^2 > n^2$?

g) Discuta si es posible insertar en una posición dereferenciable en árboles binarios.

h) Discuta el valor de retorno del método `pair<set::iterator,bool> set::insert(T x)`.

i) Defina árbol binario completo y árbol binario lleno. ¿Es verdad que todo árbol binario completo es lleno? ¿Y viceversa?

j) ¿Cuál es el número de intercambios (en notación $O()$) que requieren los algoritmos de ordenamiento lentos en el peor caso?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [25 de setiembre de 2008]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] [clases (30pt)]

a) **[lista (20pt)]** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `begin()`, `end()`.

b) **[pila-cola (10pt)]** Escribir la implementación en `C++` de los métodos `push`, `pop`,`front` y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

[Ej. 2] [Programación (total = 50pt)]

a) **[is-cyclic (30pt)]**

Dada una correspondencia M de enteros a enteros y un elemento x_0 podemos construir una secuencia $(x_0, x_1, ...)$ haciendo sucesivamente $x_1 = M(x_0)$, $x_2 = M(x_1)$,...,$x_{k+1} = M(x_k)$. La secuencia se detiene cuando el valor x_k no pertenece a las claves de M. Por ejemplo si $M = \{(1,2),(2,3),(3,4)\}$ entonces dado $x_0 = 1$ se genera la secuencia $(1,2,3,4)$. La secuencia se detiene en 4 ya que 4 no es una clave de M. En un caso así decimos que la secuencia generada es de longitud **finita**. Ahora bien, si alguno de los elementos de la secuencia, se repiten (digamos $x_{k+m} = x_k$, para algún $m > 0$) entonces a partir de allí es obvio que la secuencia se repetirá indefinidamente. Por ejemplo, si $M = \{(1,2),(2,3),(3,4),(4,2)\}$ y $x_0 = 1$ entonces la secuencia generada será $(1,2,3,4,2,3,4,2,3,4,...)$ es decir se genera una secuencia **cíclica infinita**.

Consigna: Escribir una función `void cyclic(map<int,int> &M,list<int> &L);` que extrae en `L` todas aquellas claves de `M` que generan una secuencia cíclica infinita. Por ejemplo, si $M = \{(1,2),(2,5),(3,4),(4,6),(5,2)\}$ entonces `cyclic(M,L)` debe retornar `L=(1,2,5)`.

Sugerencia: Escrbir primero un *predicado* `int is_cyclic(map<int,int> &M,int x);` que determina si el elemento `x` genera una secuencia cíclica o no. Para ello se van guardando los elementos de la secuencia generada en una lista `Mx`. Cada nuevo elemento que se genera es comparado con los elementos de `Mx`, si el nuevo elemento generado está en `Mx` la secuencia es cíclica. Si la secuencia no es cíclica, entonces se detiene cuando algún elemento no pertence a las claves de `M`. *Nota:* Puede ser útil implementar y utilizar una función predicado `bool contains(list<int> &L,int x);` que devuelve `true` si el elemento `x` está en la lista `L`.

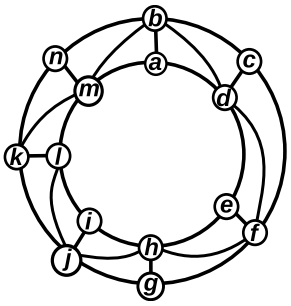
b) **[merge-map (20pt)]**

Dadas dos correspondencias A y B, que asocian enteros con listas ordenada de enteros, escribir una función `void merge_map(map<int,list<int>> &A, map<int,list<int>> &B, map<int,list<int>> &C)` que devuelve en `C` una correspondencia que asigna al elemento `x` la fusión ordenada de las dos listas `A[x]` y `B[x]`. Si `x` no es clave de `A`, entonces `C[x]` debe ser `B[x]` y viceversa. Por ejemplo:

$$A = \begin{cases} 1 \to (2,3) \\ 3 \to (5,7,10) \end{cases} \quad B = \begin{cases} 3 \to (7,8,9,11) \\ 5 \to (7,10) \end{cases} \quad C = \begin{cases} 1 \to (2,3) \\ 3 \to (5,7,7,8,9,10,11) \\ 5 \to (7,10) \end{cases}$$

Sugerencia: Implementar y utilizar una función auxiliar `void merge(list<int> &L1,list<int> &L2,list<int> &L);` que devuelve en `L` la fusión ordenada de `L1` y `L2`.

[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. (*Nota:* debe recorrer los vértices en el orden que se indica, a, b, c, ...) ¿Puede indicar si la coloración obtenida es óptima? Justifique.



[Ej. 4] **[Preguntas (total = 15pt, 5pt por pregunta)]**

a) Ordenar las siguientes funciones por tiempo de ejecución:

$$\begin{aligned} T_1 &= \log_{100} n + \sqrt[3]{n} + 1.2 \cdot 10^n \\ T_2 &= 10\,2^n + 4\,n! + n^2 \\ T_3 &= \sqrt{n} + 3\log_2 n + 4\,n^2 + 2\,n^4 \\ T_4 &= 2^5 + 20 + 10\log_2 n + 2\log_{10} n \\ T_5 &= n^5 + 10\,2^n + 10^{10} \end{aligned} \tag{1}$$

b) Comente una ventaja y una desventaja del uso de listas doblemente enlazadas con respecto a simplemente enlazadas.

c) Si la correspondencia `M={(1->2),(5->8)}` y ejecutamos el código `int x= M[5]`. ¿Que ocurre? ¿Que valores toman `x` y `M`? ¿Y si hacemos `x = M[3]`?

d) Cual es la complejidad algorítmica (mejor/promedio/peor) de las siguientes funciones:

1) `list<T>::begin()`,
2) `list<T>::insert(p,x)`,
3) `list<T>::clear(p,x)`,
4) `map<K,V>::find(x)`, para la implementación con vectores ordenados,
5) `map<K,V>::find(x)`, para la implementación con listas ordenadas.

e) Discuta si los algoritmos de búsqueda exhaustiva y heurístico para la coloración de grafos dan la solución óptima al problema. Señale la complejidad algorítmica de los mismos. ¿Cuál de los dos elegiría para un grafo con 100 vértices y 500 aristas? Justifique.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [23 de octubre de 2008]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] **[clases (20pt)]** Escribir la implementación en `C++` del `TAD Arbol Ordenado Orientado` (clase `tree`). Para la clase `tree` implemente: `insert(p,x)`, `erase(p)` y `retrieve(p)` (u `operator*`). Para la clase `iterator` implemente `lchild()` y `right()` (u `operator++`). Observaciones:

- **Debe declarar** los **miembros privados** de las clases a declarar o implementar. Ayuda: use la figura 1.
- Si opta por la interfase *"estilo"* STL, implemente la forma **prefija** del operador `operator++` (`++p`).

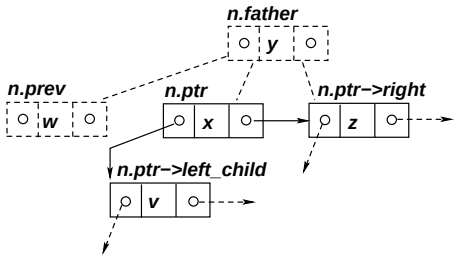


Figura 1: Entorno local de un iterator para árboles ordenados orientados.

[Ej. 2] **[Programación (total = 40pt)]**

a) **[is-full (20pt)]** Recordemos que un árbol binario (AB) es *"lleno"* si todos sus *niveles están completos (atención, no confundir con árbol completo)*. Se puede definir en forma recursiva de la siguiente manera

- El árbol vacío es lleno
- Un árbol no vacío es lleno si sus dos hijos son llenos y tienen la misma altura.

Consigna: Escribir una función predicado `bool is_full(btree<int> &T);`.
Sugerencia: Escribir una función auxiliar
`bool is_full(btree<int> &T,btree<int>::iterator n, int &height);` que retorna por `height` la altura del árbol.

b) **[max-sons-count-node (20pt)]**
Escribir una función
`tree<int>::iterator max_sons_count_node(tree<int> &T, int &nsonmax);` que, dado un árbol ordenado orientado `T` retorna la posición (o nodo o iterator) del nodo cuya cantidad de hijos es la máxima y en `nsonmax` ese número máximo de hijos. Por ejemplo, si `T=(3 (4 2 10) (6 7 (8 9 5)) (11 12 13 14 15 16))`, entonces `max_sons_count_node` debe retornar la posición (o iterator) del nodo cuyo contenido es `11` y `nsonmax=5`. Si hay más de un nodo con la cantidad máxima de hijos, entonces puede retornar cualquiera de ellos. *Sugerencia: dado un nodo debe ir recorriendo los hijos contandolos y al mismo tiempo ir reteniendo un iterator al nodo con mayor cantidad de hijos.*

[Ej. 3] [operativos (total 20pt)]

a) [**rec-arbol (10pt)**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, B, L, C, D, H, J, O, K, P, E, I, F, G, M\}$;
- `ORD_POST` $=\{L, B, C, O, J, P, K, H, D, I, E, F, M, G, A\}$.

b) [**huffman (10pt)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `WALLSTREET` $P(W) = 0.10, P(A) = 0.15, P(L) = 0.05, P(S) = 0.10, P(T) = 0.20, P(R) = 0.10, P(E) = 0.25, P(T) = 0.05$ Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.

b) Defina en forma recursiva el listado en orden previo y el listado en orden posterior de un arbol ordenado orientado con raiz `t` y sub-árboles hijos `h_1,h_2,...,h_n`.

c) Cual es el orden del tiempo de ejecución (en promedio) en función del número de elementos (n) que posee el árbol ordenado orientado implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos

- `insert(p,x)`
- `begin()`
- `lchild()`
- `operador ++ (prefijo o postfijo)`
- `find(x)`
- `erase(p)`
- `*n`

d) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.

e) Para el arbol binario `(1 . (2 (3 5 .) 6))`

- como queda el árbol y que sucede si hacemos
```
btree<int>::iterator p=T.begin();
p=T.erase(p.left());
```
- y como queda el árbol asi hacemos por otro lado
```
btree<int>::iterator p=T.begin(),n;
n=p;
n=n.right();
p=T.splice(p.left(),n);
```

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
3er Parcial. [21 de Noviembre de 2008]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] [clases (30pt), mínimo 50 %]

a) Escribir una función
`btree<int>::iterator abb_insert(btree<int> &T,int x);` que realiza la inserción de `x` en un ABB `T`. La función retorna el iterator donde está el elemento después de la inserción (sea esta exitosa o no).

b) Escribir una función
`bool open_hash_erase(vector<list<int>> &H,int (*h)(int),int x);` que elimina el elemento `x` de la tabla de dispersión abierta `H`, representada como un vector de listas **ordenadas**. El valor de retorno indica si la eliminación fue exitosa.

c) Escribir una función
`void vbset_difference(vector<bool> &A,vector<bool> &B,vector<bool> &C);` que implementa la operación binaria diferencia de conjuntos ($C = A - B$), para la representación por vectores de bits. Asuma que `A`, `B`, `C` tienen la misma longitud.

[Ej. 2] [programacion (total = 30pt, mínimo 40 %)]

a) **[connected-to (15 pt)]** Recordemos que dado un grafo no dirigido $G = (V, E)$ donde V son los vértices y E las aristas del grafo, una componente conexa de G es un subconjunto V_i de V tal que cualquier par de nodos en V están conectados entre sí, es decir existe un camino entre los dos vértices.
Consigna: Escribir una función
`void connected_to(map<int,set<int>> &G,int x,set<int> &W);` que calcula la componente conexa `W` del nodo `x`, es decir el conjunto de nodos `y`, para los cuales existe un camino en `G` desde `x` hasta `y`. *Por ejemplo,* si `G={1->{2},2->{1},3->{4},4->{3}}`, entonces si `x=1`, `W={1,2}` y si `x=4` entonces `W={3,4}`.
Sugerencia: Mantendremos dos conjuntos `W` (los vértices ya visitados) y `F` el frente de `W` que esta avanzando. Inicialmente `W=F={x}`. Ahora para calcular el siguiente frente hacemos

$$\begin{aligned} \texttt{Q} &= \bigcup_{n \in \texttt{F}} G[n], \quad \text{(vecinos de F)} \\ \texttt{F} &= \texttt{Q} - \texttt{W}, \qquad \text{(nuevo frente)} \\ \texttt{W} &= \texttt{W} \cup \texttt{F}, \qquad \text{(actualiza Q)} \end{aligned} \tag{1}$$

El algoritmo termina cuando `F` es el conjunto vacío.

b) **[eq-class (15 pt)]**
Recordemos que dada una relación de order estricta y débil $<$ en un conjunto universal U, dos elementos $a, b \in U$ son equivalentes ($a \equiv b$) si no es verdad que $a < b$ ni $b < a$. Esto separa a cualquier conjunto $S \subset U$ en conjuntos disjuntos $E_i \subset U$ llamados *"clases de*

equivalencia" tales que a, b son equivalentes entre sí, si y solo si a, b pertencen al mismo E_i.
Consigna: Escribir una función
`void eqclass(set<int> &S, bool (*comp)(int x,int y), list<set<int>> &L);` que dado un conjunto `S` y una relación de orden `comp()` devuelve sus clases de equivalencia en `L`. *Por ejemplo,* si `S={-2,-1,0,1,2}` y `comp()` es la relación *"menor en valor absoluto"* entonces debe retornar `L=({0},{-1,1},{-2,2})`.
Sugerencia: Para cada elemento `x` de `S`, recorrer los conjuntos en `L` hasta encontrar la posición donde está o debería estar su clase de equivalencia. (Este algoritmo es similar a `lower_bound(x)`). Si la clase de equivalencia ya está, entonces simplemente se agrega `x` a esta clase. Si no, se inserta en la lista un nuevo conjunto con el elemento `x`.

[Ej. 3] [operativos (total 20 pt), mínimo 25 %]

a) **[abb (5 pts)]** Dados los enteros $\{13, 7, 20, 2, 3, 10, 5, 4, 1, 12\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 13, 5 y 2 en ese orden.

b) **[hash-dict (5 pts)]** Insertar los números $3, 16, 26, 9, 8, 36, 18, 5, 28$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x$ `mod` 9 y estrategia de redispersión lineal.

c) **[heap-sort (5 pts)]** Dados los enteros $\{1, 5, 8, 2, 3, 13, 10\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

d) **[quick-sort (5 pts)]** Dados los enteros $\{5, 9, 4, 1, 5, 10, 8, 3, 3, 2, 11, 6\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 4] [Preguntas (total = 20 pt, mínimo 60 %)]

a) ¿Cuáles son las 3 etapas básicas en el ordenamiento por fusión (o *"merge sort"* o por intercalamiento)?. Explique de manera breve y simple cada una de ellas.

b) Explique una estrategia para elegir el pivot en quicksort.

c) ¿Cuál es el tiempo de ejecución para `insert(x)` en el TAD conjunto por tabla de dispersión cerrada, en el caso promedio?. ¿Cuál es el tiempo de ejecución de `find(x)` en el TAD diccionario por tablas de dispersión abiertas, en el caso promedio?

d) ¿Cuál es la función del procedimiento *"re-heap"* en montículos?.

e) Explique el concepto de estabilidad para algoritmos de ordenamiento. De ejemplos de algoritmos de ordenamiento estables y no estables.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio. [28 de Noviembre de 2008]

- **[CLASES-P1]**

  Escribir la implementación en `C++` del `TAD LISTA` (clase `list`) implementado por celdas (simple o doblemente) enlazadas por punteros . Las funciones a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `clear()`.

- **[CLASES-P2]** Escribir la implementación en `C++` del `TAD ARBOL ORDENADO ORIENTADO` (clase `tree`). Las funciones a implementar son `insert(p,x)`, `erase(p)`, `find(x)` y `clear()`.

- **[CLASES-P3]**

  1. Escribir una función
     `bool abb_erase(btree<int> &T,int x);`
     que elimina un elemento `x` de un ABB y retorna `true` si la eliminación fue exitosa y `false` en caso contrario. Si necesita una función auxiliar (por ejemplo `min()`) debe implementarla.
  2. Escribir una función
     `void set_intersection(vector<bool> &A,vector<bool> &B);`
     que realiza la operación $A \leftarrow A \cap B$, es decir **in-place**.

- **[PROG-P1]**

  1. **[nilpot]**. Dadas dos correspondencias M_1 y M_2 la *"composición"* de ambas es la correspondencia $M = M_2 \circ M_1$ tal que si $M_1[a] = b$ y $M_2[b] = c$, entonces $M[a] = c$. Por ejemplo, si `M1={(0,1),(1,2),(2,0),(3,4),(4,3)}`, y `M2={(0,1),(1,0),(2,3),(3,4),(4,2)}`, entonces $M = M_1 \circ M_2$ `={(0,0),(1,3),(2,1),(3,2),(4,4)}`. Notemos que para que sea posible componer las dos correspondencias es necesario que los valores del contradominio de M_1 estén incluidos en las claves de M_2. Si el conjunto de valores del contradominio de una correspondencia M está incluido en el conjunto de sus claves, entonces podemos componer a M consigo misma, es decir $M^2 = M \circ M$. Por ejemplo, $M_1^2 = M_1 \circ M_1$`= {(0,2),(1,0),(2,1),(3,3),(4,4)}`. De la misma manera puede definirse, $M^3, ..., M^n$, componiendo sucesivamente. Puede demostrarse que, para algún n debe ser $M^n = I$, donde I es la *"correspondencia identidad"*, es decir aquella tal que $I[x] = x$. Por ejemplo, si `M = {(0,1),(1,2),(2,0)}`, entonces para $n = 3$, $M^n = M^3 = I$.
     Consigna: Escribir una función `int nilpot(map<int,int> &M);` que dada una correspondencia `M` retorna el mínimo entero `n` tal que $M^n = I$.
     Sugerencia: Escribir dos funciones auxiliares:
     - `void compose(map<int,int> &M1,map<int,int> &M2,map<int,int> &M);` que dadas dos correspondencias `M1, M2`, calcula la composición $M = M_2 \circ M_1$, devolviéndola en el argumento `M`,
     - `bool is_identity(map<int,int> &M);` que dada una correspondencia `M`, retorna `true` si `M` es la identidad, y `false` en caso contrario.

2. [**apply-map**] Escribir una funcion
`void apply_map(list<int> &L, map<int,int> &M, list<int> &ML)` que, dada una lista `L` y una correspondencia `M` retorna por `ML` una lista con los resultados de aplicar `M` a los elementos de `L`. Si algún elemento de `L` no está en el dominio de `M` entonces el elemento correspondiente de `ML` no es incluido. Por ejemplo, si

$$\texttt{L} = (1, 2, 3, 4, 5, 6, 7, 1, 2, 3)$$
$$\texttt{M} = \{(1, 2), (2, 3), (3, 4), (4, 5), (7, 8)\}$$

entonces después de hacer `apply_map(L,M,ML)`, debe quedar

$$\texttt{ML} = (2, 3, 4, 5, 8, 2, 3, 4)$$

Restricciones: No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$, donde n es el número de elementos en la lista (asumiendo que las operaciones usadas de correspondencia son $O(1)$).

- [**PROG-P2**]

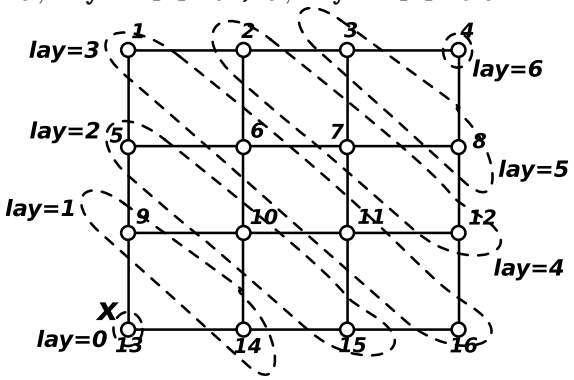
1. [**comptree**] Se define una relación de orden entre árboles binarios de enteros de la siguiente forma: `A<B` si `a<b`, o bien (`a=b` y `Ai < Bi`), o bien (`a=b` y `Ai=Bi` y `Ad<Bd`), donde `a`, `b` son las raíces y `Ai`, `Ad`, `Bi`, `Bd` son los subárboles izquierdos y derechos de `A` y `B`. Un nodo 'lambda' equivale a $-\infty$. **Consigna:** Escribir una función
`bool es_menor(tree<int> &A,tree<int> &B)` que retorna verdadero si `A < B`.
2. [**ord-nodo**]
Escribir una función predicado `bool ordnodo(tree<int> &A);` que verifica si cada secuencia de hermanos del subárbol del nodo `n` (perteneciente al árbol ordenado orientado `A`) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol `(3 5 (6 1 3) (7 4 5))` debería retornar `true`, mientras que para `(3 9 (6 1 3) (7 4 2))` debería retornar `false`, ya que las secuencias de hermanos `(9 6 7)` y `(4 2)` NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

- [**PROG-P3**]

1. [**flat**]
Se está diseñando una red inerconectada por switches y se desea, para reducir lo más posible la *latencia* entre nodos, que cada par de nodos esté conectado en forma directa por al menos un switch. Sabemos que el número de nodos es `n` y tenemos un `vector< set<int> > sw` que contiene para cada switch el conjunto de los nodos conectados por ese switch, es decir `sw[j]` es un conjunto de enteros que representa el conjunto de nodos inteconectados por el switch j.
Consigna: Escribir una función predicado `bool flat(vector< set<int> > &sw, int n);` que retorna verdadero si cada par de enteros (j, k) con $0 \leq j, k < n$ está contenido en al menos uno de los conjunto en `sw[]`.
2. [**mklayers**]
Escribir una función
`void mklayers(vector<set<int> > &G, int x, vector<set<int> > &layers)` que dado un grafo `G` y un vértice de partida `x` determina la estructuras de capas de vecinos `layers` de `x` definida de la siguiente manera:

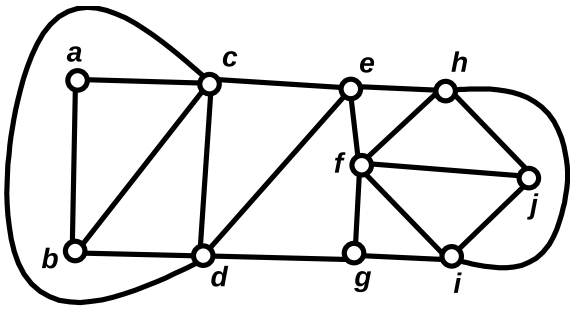
- La capa 0 es el conjunto `layers[0]={x}` .
- La capa 1 es el conjunto de los vecinos de `x`.
- Para `l>1` la capa `l` es el conjunto de los vecinos de los nodos en la capa `l-1` que no están en capas anteriores (0 a `l-1`). Notar que en realidad sólo hace falta verificar que no estén en las capas `l-1` y `l-2`.

Puede demostrarse que los vértices en la capa `l` son los que están a distancia `l` de `x`. Por ejemplo, dado el grafo de la figura, y partiendo del nodo `x=0` las capas son `layers[0]={0}`, `layers[1]={9,14}`, `layers[2]={5,10,15}`, `layers[3]={1,6,11,16}`, `layers[4]={2,7,12}`, `layers[5]={3,8}`, `layers[6]={4}`.



- **[OPER-P1]**

1. **[color-grafo]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido siguiendo los nodos en el orden indicado $(a, b, c, ...)$. ¿La coloración obtenida es óptima? Justifique.



- **[OPER-P2]**

  - **[rec-arbol]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
    - `ORD_PRE` $=\{Z, A, R, Q, L, M, N, T, S, W, Q\}$,
    - `ORD_POST` $=\{Q, L, R, N, M, A, W, Q, S, T, Z\}$.
  - **[huffman]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `CONCLAVE` $P(C) = 0.05, P(O) = 0.05, P(N) = 0.1, P(A) = 0.2, P(L) = 0.2, P(V) = 0.1, P(E) = 0.1, P(Q) = 0.2$ Calcular la longitud promedio del código obtenido.
  - **[part-arbol]** Particione el árbol AOO `(p (q (s x y) t u) (r v w))` con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

- **[OPER-P3]**

- [**heap-sort**] Dados los enteros $\{0, 4, 7, 1, 2, 12, 9, 3\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
- [**quick-sort**] Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.
- [**abb**] Dados los enteros $\{16, 10, 23, 5, 6, 13, 8, 7, 4, 15\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 12, 10 y 7 en ese orden.
- [**hash-dict**] Insertar los números $5, 18, 28, 11, 10, 38, 20, 7, 30$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

- [**PREG-P1**]

1. Explique que quiere decir la propiedad de *"Transitividad"* de $O()$.
2. ¿Puede tener una correspondencia varios valores iguales del dominio, o sea varias claves iguales? (Por ejemplo `M1={(1,2),(1,3)}`) ¿Y varios valores iguales del contradominio? (Por ejemplo `M2={(1,2),(3,2)}`)
3. ¿Qué retorna la dereferenciación de un iterator sobre correspondencias (clase `map`)?
4. ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si siempre es $(n+1)^2 > n^2$?
5. ¿Qué ocurre si se invoca el `operator[]` sobre una correspondencia con una clave que no tiene asignación? Por ejemplo `M={(1,2),(3,4)}` y hacemos `x = M[5]`. ¿Da un error? ¿Qué valor toma `x`?

- [**PREG-P2**]

1. ¿Por qué no es útil definir a un "iterator"sobre un árbol ordenado orientado como `typedef *cell` como se hace en listas simplemente enlazadas?
2. Discuta si es posible insertar en una posición dereferenciable en ÁRBOLES BINARIOS.
3. ¿Para qué sirve el algoritmo de Huffmann para generar códigos? ¿En qué casos es bueno? ¿Da una solución *"heurística"* o da el código óptimo.
4. ¿Cómo se traduce la *condición de prefijos* para códigos representados por árboles? El árbol `(. (. a b) (c . d))`, ¿lo satisface?
5. Códigos de Hufmann: ¿De qué dependen las profundidades de los nodos en el árbol?

- [**PREG-P3**]

1. ¿Defina claramente cuál es al condición de *"Árbol Binario de Búsqueda"*?
2. ¿Por qué se les dice *"lentos"* a ciertos algoritmos de ordenamiento? Enumere los algortimos lentos que conoce.
3. ¿Cómo se encuentra el mínimo y el máximo de los valores en un árbol binario de búsqueda? ¿Cuál es la complejidad algorítmica de esta operación (caso promedio, mejor y peor)?
4. Discuta el valor de retorno de `insert(x)` para conjuntos.
5. Discuta el número de inserciones que requieren los algoritmos de ordenamiento lentos en el peor caso.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [2009-09-17]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] [clases (30pt)]

a) **[lista (20pt)]** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()/iterator::operator++(int)`, `list()`, `begin()`, `end()`.

b) **[pila-cola (10pt)]** Escribir la implementación en `C++` de los métodos `push`, `pop`,`front` y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

[Ej. 2] [Programación (total = 50pt)]

a) **[intersect-map (30pt)]** Implemente una función `void intersect_map(const map< string, list<int> > &A, const map< string, list<int> > &B, map< string, list<int> > &C)` que a partir de los diccionarios `A` y `B` construya un diccionario `C` de acuerdo a las siguientes reglas:

- Si una clave `key` esta contenida en A o B pero **no en ambos**, entonces C **no debe contener** dicha clave.
- Si una clave `key` esta contenida en A y B a la vez, entonces C debe contener dicha clave y su valor asociado debe ser una lista que contenga todos los **elementos comunes y sin repetición** de las listas asociadas a `key` en A y B (es decir, la intersección como conjunto, $C[key] \leftarrow A[key] \bigcap B[key]$).

Por ejemplo, dados:

```
A = { 'XX' : [3,3,1,2,2,7],     B = { 'YY' : [3,3,4,5,8,1],
      'YY' : [7,1,5,5,4,1] }           'ZZ' : [1,1,9]       }
```

se debe obtener:

```
C = { 'YY' : [1,5,4]}
```

Sugerencia: implementar una función auxiliar `bool contains(list<int>&L, int x)` que devuelve `true` si la lista `L` contiene el valor `x`; y `false` en caso contrario. Utilize esta funciń como ayuda para contruir la lista de elementos comunes y sin repetición $C[key] \leftarrow A[key] \bigcap B[key]$

b) **[are-inverses (10pt)]** Dos correspondencias `M1` y `M2` son **inversas** una de la otra si tienen el mismo número de asignaciones y para cada par de asignación `x->y` en `M1` existe el par `y->x` en `M2`.
Consigna: Escribir una función **predicado**
`bool areinverse(map<int,int> &M1,map<int,int> &M2);` que determina si las correspondencias `M1`, `M2` son una la inversa de la otra o no.

c) **[map2list (10pt)]** Escribir las funciones `map2list()` y `list2map()` de acuerdo a las siguientes especificaciones.

- `void map2list(map<int,int> &M,list<int> &keys,list<int> &vals);` dado un map `M` retorna las listas de claves y valores. Es decir, si `M={(1->2,3->5,8->20)}`, entonces debe retornar `keys=(1,3,8)`, `vals=(2,5,20)`.
  Nota: `keys` debe quedar ordenada de menor a mayor. Asumir que la implementación de `map<>` es STL, es decir, tal que al recorrer las asignaciones las claves resultan ordenadas.
- `void list2map(list<int> &keys,list<int> &vals,map<int,int> &M);` dadas las listas de claves `(k1,k2,k3...)` y valores `(v1,v2,v3...)` retorna el map `M` con las asignaciones correspondientes `{(k1,v1),(k2,v2),(k3,v3),...}`. (*Nota:* Si hay **claves repetidas**, sólo debe quedar la asignación correspondiente a la **última** clave en la lista. Si hay menos valores que claves, utilizar cero como valor. Si hay más valores que claves, ignorarlos).

- *Pregunta 1:* Si ejecutamos el siguiente código

```
list2map(keys,vals,M);
map2list(M,keys2,vals2);
```

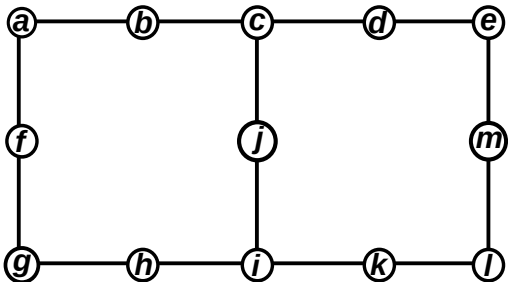
¿Cuáles son las condiciones sobre `keys` y `vals` para que resulte ser `keys2==keys`, `vals2==vals`? Justifique.
Ayuda: Tener en cuenta qué ocurre cuando 1) hay claves repetidas, 2) si `keys` y `vals` están ordenadas o no, y 3) las longitudes de ambas son iguales o no.

- *Pregunta 1:* Si ejecutamos el siguiente código

```
map2list(M,keys,vals);
list2map(keys,vals,M2);
```

¿Resulta ser `M2==M`? ¿Depende la respuesta de la implementación de `map<>`? Justifique.

[Ej. 3] **[color-grafo (5 ptos)]** Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido. (*Nota:* debe recorrer los vértices **en el orden que se indica**, *a*, *b*, *c*, ...) ¿Puede indicar si la coloración obtenida es óptima? Justifique.



[Ej. 4] **[Preguntas (total = 15pt, 3pt por pregunta)]**

a) Ordenar las siguientes funciones por tiempo de ejecución:

$$\begin{aligned} T_1 &= \sqrt{n} + 2\log_4 n + 2\,n^3 + 2\,n^5 \\ T_2 &= n^3 + 3\cdot 4^n + 2^{20} \\ T_3 &= 4^3 + 10 + 2\log_{10} n + 4\log_2 n \\ T_4 &= 2\cdot 4^n + 5\,n! + 3n^2 \\ T_5 &= \log_{16} n + \sqrt{5}\cdot n + 5\cdot 100^n \end{aligned} \tag{1}$$

b) ¿Porqué se dice que la pila es una estructura *FIFO ("First in, First Out")*? ¿Porqué se dice que la cola es una estructura *LIFO ("Last In, First Out")*?

c) Si tenemos la correspondencia `M={(5->6),(9->12)}` y ejecutamos el código `int x= M[9]`. ¿Que ocurre? ¿Que valores toman `x` y `M`? ¿Y si hacemos `x = M[7]`?

d) Cual es la complejidad algorítmica (mejor/promedio/peor) de las función `lower_bound()` para correspondencias implementadas por

1) listas ordenadas,
2) vectores ordenados.

e) Considerando la implementación de pilas con **listas simplemente enlazadas**. ¿Cuál es la diferencia entre elegir como tope de la pila el comienzo o fin de la lista?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos.
2do Parcial. [27 de octubre de 2009]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] [clases (20pt)]

a) **AOO:** declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`

b) **AB:** declarar las clases `btree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`

[Ej. 2] [programación (total 50pt)]

a) [**split-down (25pt)**] Dados dos enteros `M` y `n>0`, crear un AOO `T` tal que:

- 1) La suma de los contenidos de las hojas es `M`, pero cada una de ellas es `h<=n`.
- 2) Se satisface que para cada nodo `p` la suma de sus hijos es `*p`.
- 3) Cada nodo tiene a lo sumo `g` hijos, con `g>1` una constante dada.

Por ejemplo dados `M=20`, `n=4`, y `g=3`, un posible árbol (la solución no es única, por supuesto) es `(20 (7 3 2 2) (7 3 2 2) (6 2 2 2))`.
Consigna: Escribir una función `void split_down(tree<int> &T,int n,int g);` que realiza la tarea indicada.
Ayuda: El arbol se puede construir poniendo inicialmente `M` en la raíz, y dividiendo el contenido `*n` de cada nodo en `g` valores aprox iguales hasta obtener valores `<=n`. Para ello escribir una función recursiva auxiliar
`int depth_if(btree<int> &T,btree<int>::iterator n, pred_t pred);`. Asumir que existe una función `void distrib(int m,int g, vector<int> &v);` que distribuye el entero `m` en a lo sumo `g` partes, tratando de que difieran a lo sumo en una unidad, por ejemplo `distrib(20,6,v);` hace `v=[4,4,3,3,3,3]`.

b) [**depth-if (25pt)**]
Dado un AB `T` encontrar, la profundidad máxima de los nodos que satisfacen un cierto predicado (*profundidad condicionada*). Por ejemplo, si `T=(6 (7 9 (3 1)) 2)`, entonces `depth_if(T,even)` debe retornar 1, ya que el nodo par a mayor profundidad es 2, mientras que `depth_if(T,odd)` debe retornar 3, ya que el nodo impar a máxima profundidad es 1.
Consigna: Escribir una función `int depth_if(btree<int> &T,bool (*pred)(int));` que realiza la tarea indicada.
Ayuda: La función recursiva auxiliar debe retornar la máxima profundidad de un nodo que satisface el predicado o -1 si el árbol está vacío o ningún nodo satisface el predicado.
Entonces, dadas las profundidades condicionadas d_l, d_r de los hijos la profundidad se puede expresar en forma recursiva como

$$\text{depth}(n) = \begin{cases} -1, & \text{si } n \text{ es } \Lambda, \\ \max(d_r, d_l) + 1, & \text{si } \max(d_r, d_l) \geq 0, \\ 0, & \text{si } n \text{ satisface el predicado,} \\ -1, & \text{si } n \text{ no satisface el predicado.} \end{cases}$$

[Ej. 3] [operativos (total 20pt)]

a) [**rec-arbol (10pt)**] Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $= \{W, X, Y, A, B, C, D, F, G, H\}$;
- `ORD_POST` $= \{X, Y, B, F, G, H, D, C, A, W\}$.

b) [**huffman (10pt)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `SUDAFRICA` $P(S) = 0.15, P(U) = 0.15, P(D) = 0.05, P(A) = 0.05, P(F) = 0.20, P(R) = 0.10, P(I) = 0.20, P(C) = 0.10$. Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Defina que es un **camino** en un árbol. Dado el AOO `(a b (c e (f g)) d))`, cuáles de los siguientes son caminos?
1) `(a,c,f,g)`,
2) `(e,c,f,g)`,
3) `(e,c,a)`,
4) `(c,f,g)`.

b) Defina qué es un **AB lleno**. ¿Cuál es el **número de nodos** en el nivel l de un árbol lleno? ¿Cuál es el **número total de nodos** en un árbol lleno de n niveles? ¿Cuantas **hojas** contiene?

c) ¿Es verdad que si dos nodos están en el **mismo nivel** de un árbol, entonces son **hermanos**? ¿Y la recíproca? De ejemplos.

d) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un AB? ¿Y en un AOO? Discuta y de ejemplos.

e) ¿Porqué el AB correspondiente a la **codificación binaria** de una serie de caracteres debe ser un **ABC (AB completo)**?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. [27 de Noviembre de 2009]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 25 % en operativos (Ej 3) y un 60 % sobre las preguntas de teoría (Ej 4).

[Ej. 1] [clases (20pt)]

a) Implementar `set::iterator set::find(T x)` para conjuntos implementados por ABB. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.

b) Implementar `int set::erase(T x)` para conjuntos implementados por listas ordenadas. Si se utiliza algún método auxiliar, también implementarlo. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.

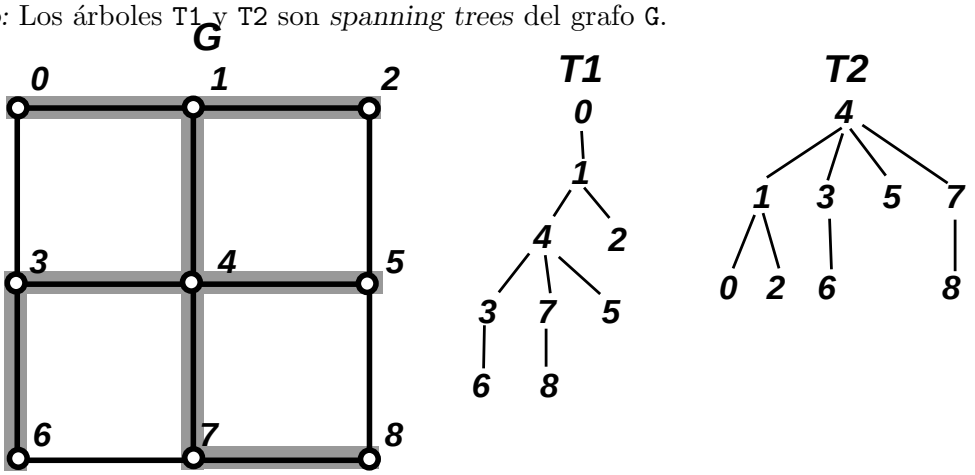
c) Implementar una función `bool openhashtable_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T), T x)` que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y retorna un booleano indicando si la inserción fue o no exitosa.

d) implementar una función `void vecbit_difference(vector<bool>&, A, vector<bool>& B, vector<bool>& C);` que devuelve `C=A-B` con `A,B,C` vectores de bits que representan conjuntos de un rango contiguo de enteros `[0, N)`, donde `N` es el tamaño de los vectores `A,B,C` (asumir el mismo tamaño para los tres, es decir, haciendo `int N=A.size();` es suficiente).

[Ej. 2] [programación (total 40pt)] Dado un grafo conexo `map<int, set<int> > G` y un árbol AOO `tree<int> T` decimos que `T` es un **spanning tree** de `G` si, se puede llegar desde

- El conjunto de nodos de `T` es igual al conjunto de vértices de `G`.
- Los nodos de `T` no están repetidos.
- Los caminos de `T` son caminos en el grafo y por lo tanto permiten llegar desde la raíz a cualquier otro nodo.

Podemos pensar a `T` como un subgrafo de `G`, pero que cumple las propiedades para ser un árbol.
Ejemplo: Los árboles `T1` y `T2` son *spanning trees* del grafo `G`.



Consigna: Escribir una función `bool is_spng_tree(map<int,set<int>> &G,tree<int> &T);` que retorna `true` sii `T` es *spanning tree* de `G`.
Ayuda:

a) [**tree-nodeset (10pt)**] Escribir una función `bool tree_nodeset(tree<int> &T,set<int> &nodeset);` que retorna `true` si todos los nodos de `T` son distintos y si ese es el caso retorna por `nodeset` el conjunto de nodos de `T`. *Ayuda:* Probablemente esto requiere hacerlo en forma recursiva sobre el árbol.

b) [**graph-nodeset (10pt)**] Escribir una función `void graph_vrtxset(map<int,set<int>> &G,set<int> &vrtxset);` que retorna por `vrtxset` el conjunto de vértices de `G`.

c) [**isspngtree (20pt)**] Con la ayuda de las dos funciones auxiliares previas:

- Comprobar que los nodos de `T` son únicos.
- Comprobar que el conjunto de nodos de `T` es igual al de vértices de `G`.
- Verificar que las *aristas* de `T` (es decir los pares padre-hijo) están contenidos en `G`. (**Ayuda:** escribir una función recursiva sobre el árbol).

[Ej. 3] [operativos (total 20pt)]

a) [**abb (5 pts)**] Dados los enteros $\{17, 11, 24, 6, 7, 14, 9, 8, 5, 16\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 17, 9 y 6 en ese orden.

b) [**hash-dict (5 pts)**] Insertar los números $4, 20, 30, 13, 12, 40, 22, 9$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x$ `mod` 8 y estrategia de redispersión lineal.

c) [**heap-sort (5 pts)**] Dados los enteros $\{5, 9, 12, 6, 7, 17, 14\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

d) [**quick-sort (5 pts)**] Dados los enteros $\{9, 13, 8, 5, 9, 14, 12, 7, 7, 6, 15, 10\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 4] [Preguntas (total = 10pt, 4pt por pregunta)]

a) Defina la propiedad de **transitividad** para las relaciones de orden. Si $f(x)$ es una función sobre los reales y defino $a <_f b$ sii $f(a) < f(b)$, es $<_f$ transitiva?

b) Nombre diversas **relaciones de orden** que se pueden usar con los algoritmos de ordenamiento.

c) Discuta el valor de retorno de `insert(x)` para conjuntos.

d) Discuta el **número de inserciones** que requieren los algoritmos de ordenamiento lentos en el peor caso.

e) ¿Cuál es el costo de **inserción** en tablas de dispersión cerradas?

Apellido y Nombre: _____

Carrera: _____ DNI: _____
[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio. [3 de Diciembre de 2009]

- [**CLASES-P1**]

  1. Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `begin()`, `end()`.
  2. Escribir la implementación en `C++` de los métodos `push`, `pop`,`front` y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

- [**CLASES-P2**]

  1. **AOO:** declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`
  2. **AB:** declarar las clases `btree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`

- [**CLASES-P3**]

  1. Implementar `set::iterator set::find(T x)` para conjuntos implementados por ABB. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.
  2. Implementar `int set::erase(T x)` para conjuntos implementados por listas ordenadas. Si se utiliza algún método auxiliar, también implementarlo. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.
  3. Implementar una función `bool openhashtable_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T), T x)` que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y retorna un booleano indicando si la inserción fue o no exitosa.
  4. implementar una función `void vecbit_difference(vector<bool>&, A, vector<bool>& B, vector<bool>& C);` que devuelve `C=A-B` con `A,B,C` vectores de bits que representan conjuntos de un rango contiguo de enteros `[0, N)`, donde `N` es el tamaño de los vectores `A,B,C` (asumir el mismo tamaño para los tres, es decir, haciendo `int N=A.size();` es suficiente).

- [**PROG-P1**]

  1. [**junta**] Escribir una función `void junta(list<int> &L,int n);` que dada una lista `L`, agrupa de a `n` elementos dejando su suma. Por ejemplo, si la lista `L` contiene L=(1,3,2,4,5,2,2,3,5,7,4,3,2,2), entonces depués de `junta (L,3)` debe quedar L=(6,11,10,14,4).
     Restricciones: **N**o usar ninguna estructura auxiliar. Prestar atención a no usar posiciones inválidas después de una supresión. El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la lista original.
  2. [**nilpot**]. Dadas dos correspondencias M_1 y M_2 la *"composición"* de ambas es la correspondencia $M = M_2 \circ M_1$ tal que si $M_1[a] = b$ y $M_2[b] = c$, entonces $M[a] = c$. Por ejemplo, si `M1={(0,1),(1,2),(2,0),(3,4),(4,3)}`, y `M2={(0,1),(1,0),(2,3),(3,4),(4,2)}`, entonces $M = M_1 \circ M_2$ `={(0,0),(1,3),(2,1),(3,2),(4,4)}`. Notemos que para que sea posible componer las dos correspondencias es necesario que los valores del contradominio de M_1 estén incluidos en las claves de M_2. Si el conjunto de valores del contradominio de una correspondencia M está incluido en el conjunto de sus claves, entonces podemos componer a M consigo misma, es decir $M^2 = M \circ M$. Por ejemplo, $M_1^2 = M_1 \circ M_1$`= {(0,2),(1,0),(2,1),(3,3),(4,4)}`. De la misma manera puede definirse, $M^3, ..., M^n$, componiendo sucesivamente. Puede demostrarse que, para algún n debe ser $M^n = I$,

donde I es la *"correspondencia identidad"*, es decir aquella tal que $I[x] = x$. Por ejemplo, si `M = {(0,1),(1,2),(2,0)}`, entonces para $n = 3$, $M^n = M^3 = I$.
Consigna: Escribir una función `int nilpot(map<int,int> &M);` que dada una correspondencia `M` retorna el mínimo entero `n` tal que $M^n = I$.
Sugerencia: Escribir dos funciones auxiliares:

- `void compose(map<int,int> &M1,map<int,int> &M2,map<int,int> &M);` que dadas dos correspondencias `M1, M2`, calcula la composición $M = M_2 \circ M_1$, devolviéndola en el argumento `M`,
- `bool is_identity(map<int,int> &M);` que dada una correspondencia `M`, retorna `true` si `M` es la identidad, y `false` en caso contrario.

- **[PROG-P2]**

1. **[path-of-largest]**
   Escribir una función `void path_of_largest(tree<int> &A,list<int> &L);` que, dado un árbol ordenado orientado `A`, retorna en `L` el camino que se obtiene recorriendo el árbol hacia abajo, *siempre por el hijo más grande*. Por ejemplo, si `A=(3 (4 6 (7 8 9)) (5 2 4 6))` entonces `path_of_largest` debe retornar `L=(3 5 6)`. Si para un nodo el valor más grande de los hijos está repetido, entonces el camino puede seguir por cualquiera de ellos. Por ejemplo si `A=(3 (4 5 6) 4 2)` entonces `path_of_largest()` puede retornar `L=(3 4 6)` o `L=(3 4)`.
2. **[cumsum-down]**
   El `cumsum(v)` de un vector `v` es la suma acumulada, es decir en la posicion `v[j]` debe quedar la suma de los elementos de `v[0..j]`. Para un árbol lo podemos extender diciendo que en cada nodo del árbol queda la suma de los valores de los nodos desde la raíz hasta el nodo en cuestión ANTES de la operación. Por ejemplo si `T=(1 (3 2 6) (5 4 8))` entonces después de `cumsum(T)` debe quedar `T=(1 (4 6 10) (6 10 14))`.
   Consigna: Escribir una función `void cumsum(btree<int> &T);` que realiza la tarea indicada para un **árbol binario (AB).**

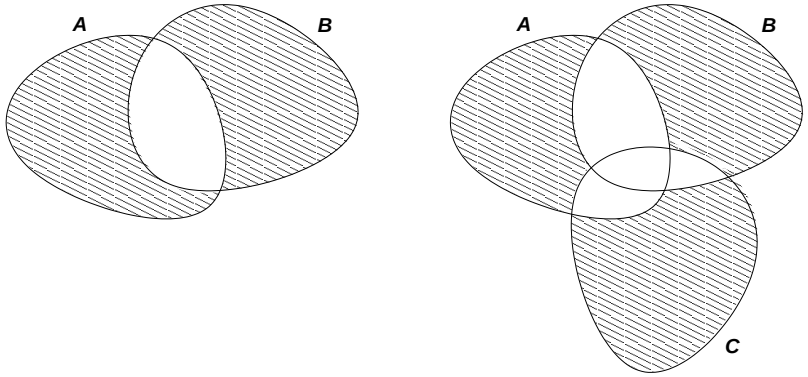
- **[PROG-P3]**

1. **[is-connected]**
   Se dice que un grafo `G` es **conexo** si para cada par de vértices `a`, `b` existe un camino que lleva desde un vértice al otro recorriendo las aristas del grafo.
   Consigna: Escribir una función bool `is_connected(map<int,set<int>> &G);` que determina si el grafo `G` es conexo o no.
   Ayuda: Escribir una función auxiliar `void connected_set(map<int,set<int>> &G,int x,set<int> &S);` que calcula la componente conexa `S` de `x`. Para ello recorre los vértices del grafo en forma *breadth-first* es decir usa una cola Q y ejecuta el siguiente algoritmo.

   `Q=` cola vacía, `S=`$\emptyset$;
   Pone `x` en `Q`
   while `Q` no esta vacía **do**
   Toma el primer elemento `y` de `Q`
   Pone `y` en `S`
   for `z` nodo vecino de `y` **do**
   Si `z` no fue visitado lo pone en `Q`.
   end for
   end while

   Cuando termina el lazo en `S` tenemos la componente conexa de `x` en `G`. Una vez escrita esta función sólo basta con tomar cualquier vértice de `G`, calcular su componente conexa y verificar si esta coincide con todo el conjunto de vértices del grafo o no.
2. **[diffsym]** Para dos conjuntos A, B, la *"diferencia simétrica"* se define como

$$\begin{aligned} \text{diff_sym}(A,B) &= (A - B) \cup (B - A), \quad \text{o también} \\ &= (A \cup B) - (A \cap B) \end{aligned}$$



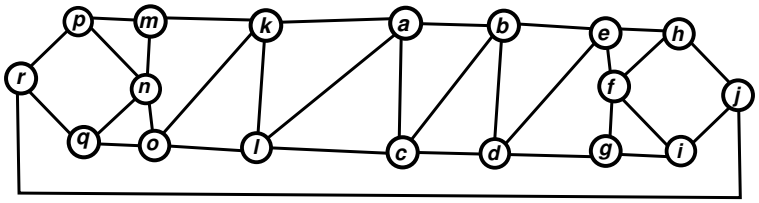
En general, definimos la diferencia simétrica de varios conjuntos como el conjunto de todos los elementos que pertenecen a uno y sólo uno de los conjuntos. En las figuras vemos en sombreado la diferencia simétrica para dos y tres conjuntos. Por ejemplo, si $A = \{1, 2, 5\}$, $B = \{2, 3, 6\}$ y $C = \{4, 6, 9\}$ entonces $\text{diff_sym}(A, B, C) = \{1, 3, 4, 5, 9\}$.
Consigna: Escribir una función `void diff_sym(list<set<int>> &l,set<int> &s);` que retorna en `s` la diferencia simétrica de los conjuntos en `l`.
Ayuda: Notar que en el caso de tres conjuntos si $S = \text{diff_sym}(A, B)$ y $U = A \cup B$, entonces $\text{diff_sym}(A, B, C) = (S - C) \cup (C - U)$. Esto vale en general para cualquier número de conjuntos, de manera que podemos utilizar el siguiente lazo

$l =$ lista de conjuntos, $S = \emptyset, U = \emptyset$;
for $Q =$ en la lista de conjuntos l **do**
  $S = (S - Q) \cup (Q - U)$;
  $U = U \cup Q$;
end for

Al terminar el lazo, S es la diferencia simétrica buscada.

- [**OPER-P1**]

1. [**color-grafo**] Colorear el grafo de la figura usando el mínimo número de colores posible. Usar el algoritmo heurístico ávido siguiendo los nodos en el orden indicado $(a, b, c, ...)$. ¿La coloración obtenida es óptima? Justifique.



- [**OPER-P2**]

1. [**huffman**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PAPELERAS`
   $P(P) = 0.1, P(A) = 0.1, P(L) = 0.3, P(E) = 0.1, P(R) = 0.2, P(B) = 0.05, P(Q) = 0.05, P(S) = 0.1$
   Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.
2. [**rec-arbol**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
   - `ORD_PRE` $=\{Z, K, Y, W, Q, M, N, T, U, V\}$;
   - `ORD_POST` $=\{Y, W, K, M, V, U, T, N, Q, Z\}$.
3. [**make-tree**] Escribir la secuencia de sentencias necesarias para construir el arbol binario `(3 (4 . 2) (1 . 7))`. (*Restricciones:* No usar `find()`).
4. [**modif-tree**] Escribir la secuencia de secuencia necesarias para modificar el árbol ordenado orientado `A = (3 (4 5 6) (7 8 9))` de manera que se convierta en `A = (3 (4 5 6) (7 8 10 9))`. (Es decir, insertar el 10 entre el 8 y el 9). (*Restricciones:* No usar `find()`).

- [**OPER-P3**]

  - [**heap-sort**] Dados los enteros $\{0, 4, 7, 1, 2, 12, 9, 3\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
  - [**quick-sort**] Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.
  - [**abb**] Dados los enteros $\{16, 10, 23, 5, 6, 13, 8, 7, 4, 15\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 16, 10 y 7 en ese orden.
  - [**hash-dict**] Insertar los números $5, 18, 28, 11, 10, 38, 20, 7, 30$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

- **[PREG-P1]**

1. ¿Cuál es el tiempo de ejecución de `x=M[key]` para correspondencias implementadas con vectores ordenados?
2. ¿Cuál es el tiempo de ejecución para `L.insert(p,x)` en listas implementadas por arreglos?
3. Dado el siguiente fragmento de código

```
map<int,int> M;
M[0]=1;
int i = M[3];
```

   ¿Que valor obtiene `i`? ¿Se modifica `M`?
4. Dadas las funciones
   - $T_1(n) = 2n^2 + 1000\,n + \sqrt{5n} + 2^{20}$,
   - $T_2(n) = n! + 2n^5 + 5n^2 + 3n^{5/2}$,
   - $T_3(n) = 3n^2 + 2n^3 + n^{3/2} + 3\,\log n$,
   - $T_4(n) = 4n^{1.5} + 3n^{2.5} + n\log n + 6\sqrt{n}$.

   ordenarlas de menor a mayor.

$$T_{\square} < T_{\square} < T_{\square} < T_{\square}$$

- **[PREG-P2]**

1. Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.
2. Defina en forma recursiva el listado en orden previo y el listado en orden posterior de un arbol ordenado orientado con raiz `t` y sub-árboles hijos `h_1,h_2,...,h_n`.
3. Cual es el orden del tiempo de ejecución (en promedio) en función del número de elementos (n) que posee el árbol ordenado orientado implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos
   - `insert(p,x)`
   - `begin()`
   - `lchild()`
   - `operador ++ (prefijo o postfijo)`
   - `find(x)`
   - `erase(p)`
   - `*n`
4. Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.
5. Para el arbol binario `(1 . (2 (3 5 .) 6))`
   - como queda el árbol y que sucede si hacemos

```
btree<int>::iterator p=T.begin();
p=T.erase(p.left());
```

   - y como queda el árbol asi hacemos por otro lado

```
btree<int>::iterator p=T.begin(),n;
n=p;
n=n.right();
p=T.splice(p.left(),n);
```

- **[PREG-P3]**

  1. ¿Cuál es el tiempo de ejecución de `find(x)` en el TAD diccionario por tablas de dispersión **abiertas**, en el caso promedio?
  2. ¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por árbol binario de búsqueda, en el peor caso?
  3. ¿Cual es el tiempo de ejecución para `insert(x)` en el TAD conjunto por tabla de dispersión **cerrada**, en el caso promedio?
  4. ¿Cuál es el número de *intercambios* en el método de clasificación por selección?
  5. Recordemos que el procedimiento *"re-heap"* es áquel que, mediante una serie de intercambios restituye la propiedad de montículo a un árbol binario el cuál satisface la propiedad de parcialmente ordenado en todos sus nodos menos, eventualmente, en la raíz. ¿Cuál es el tiempo de ejecución de *re-heap*?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [2010-09-16]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), y un 60 % sobre las preguntas de teoría (Ej 3).

[Ej. 1] **[clases (30pt)]**

a) **[lista (10pt)]** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`.

b) **[pila-cola (10pt)]** Escribir la implementación en `C++` de los métodos `push`, `pop`, `front`, y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

c) **[map (10pt)]** Escribir la implementación en `C++` del TAD correspondencia (clase `map`) implementado con listas ordenadas. Métodos a implementar: `find(key)`.

[Ej. 2] **[Programación (total = 50pt)]** Recordar que en los ejercicios de programación **deben usar la interfaz STL**.

a) **[cutoff-map (15pt)]** *Consigna:* Implemente una función `void cutoff_map(map<int, list<int> > &M,int p,int q);` que elimina todas las claves que NO están en el rango `[p,q)`. En las asignaciones que quedan también debe eliminar los elementos de la lista que no están en el rango. Si la lista queda vacía entonces la asignación debe ser eliminada. *Por ejemplo:* Si `M={1->(2,3,4), 5->(6,7,8), 8->(4,5), 3->(1,3,7)}`, entonces `cutoff_map(M,1,6)` debe dejar `M={1->(2,3,4), 3->(1,3)}`. Notar que la clave 5 ha sido eliminada si bien está dentro del rango porque su lista quedaría vacía.
Restricciones: El programa no debe usar contenedores auxiliares.

b) **[compacta (20pt)]** *Consigna:* Escribir una función `void compacta(list<int> &L,stack<int> &S);` que va tomando un elemento entero `n` de `S` y, si es positivo, saca `n` elementos de `L` y los reemplaza por su suma. Esto ocurre con todos los elementos de `S` hasta que se acaben, o bien se acaben los elementos de `L`.
Por ejemplo: Si `L=(1,3,2,1,4,5,3,2,4,1)` y `S=(3,2,-1,0,2,5,2,-3)` entonces `L` debe quedar así `L=(6,5,8,7)`, y `S=(2,-3)` (es decir, sobran elementos de `S`).
Otro ejemplo: Si `L=(1,3,2,1,4,5,3,2,4,1,3,2,1,4,7)` y `S=(3,2,-1,0,2,5)` entonces `L` debe quedar así `L=(6,5,8,12,1,4,7)`, y `S=()` (es decir, sobran elementos de `L`).
Restricciones: El programa no debe usar contenedores auxiliares.

c) **[concat-map (15pt)]** *Consigna:* Escribir una función `void concat_map(map<int,list<int> >& M, list<int>& L);` tal que reemplaza los elementos de `L` por su imagen en `M`. Si un elemento de `L` no es clave de `M` entonces se asume que su imagen es la lista vacía.
Por ejemplo: Si `M={5->(3,2,5),2->(4,1),7->(2,1,1)}` y `L=(1,5,7,2,3)`, entonces debe quedar `L=(3,2,5,2,1,1,4,1)`.
Restricciones: El programa no debe usar contenedores auxiliares.

[Ej. 3] **[Preguntas (total = 20pt, 4pt por pregunta)]**

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = 5 \cdot 2^n + 2n^3 + 3\,n!+$$
$$T_2 = 2 \cdot 10^n + \sqrt{3} \cdot n + \log_8 n+$$
$$T_3 = n^2 + 5 \cdot 3^n + 4^{10}$$
$$T_4 = 2.3 \log_8 n + \sqrt{n} + 5\,n^2 + 2\,n^5$$
$$T_5 = 1000 + 3 \log_4 n + 8^2 + 5 \log_{10} n$$

b) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `map<>` implementada con **listas desordenadas** en el caso promedio?
Métodos: `find(key)`, `M[key]`, `erase(key)`, `erase(p)`, `begin()`, `end()`, `clear()`).

c) ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si en realidad es siempre verdad que $(n+1)^2 > n^2$?

d) Discuta las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas.

e) ¿Qué ventajas o desventajas tendría implementar la clase *pila* en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [2010-10-28]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] **[clases (20pt)]** **Insistimos: deben usar la interfaz STL**.

a) **[AOO (10pt)]** Escribir la implementación en `C++` del TAD árbol ordenado orientado (clase `tree`). Los métodos a implementar son
`tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x);`
`tree<T>::iterator tree<T>::erase(tree<T>::iterator n); tree<T>::iterator begin();`
`tree<T>::iterator end();`.

b) **[AB (10pt)]** Par el TAD Arbol Binario (AB): declarar las clases `btree`, `cell`, `iterator`, incluyendo las declaraciones de datos miembros. Implementar el método
`btree<T>::iterator btree<T>::find(const T& x);`. Si utiliza alguna función auxiliar impleméntela.

[Ej. 2] **[Programación (total = 40pt)]** **Insistimos: deben usar la interfaz STL**.

a) **[list2tree (20pt)]** Recordemos que **serializar** una estructura de datos consiste en convertir esa estructura en una secuencia de bytes, tal que de esa forma pueda almacenarse en disco o tansmitirse por red para después ser reconstruida. Hemos visto que para serializar un árbol podemos usar la **notación LISP** o bien la combinación de listados en orden previo y posterior del árbol. Otra posibilidad consiste en generar una lista donde por cada nodo n listamos, en orden previo, el valor contenido en el mismo y su cantidad de hijos del nodo. Por ejemplo, si el árbol es `T=(6 4 8 (5 4 4) 7 9)` entonces la lista generada sería `L=(6 5 4 0 8 0 5 2 4 0 4 0 7 0 9 0)`.
Consigna: Escribir una función `void list2tree(tree<int> &T,list<int> &L);` que dado una lista `L` reconstruye el **árbol ordenado orientado (AOO)** `T` de acuerdo a la serialización descripta previamente.
Ayuda: Escribir una función auxiliar recursiva `tree<int>::iterator list2tree(tree<int> &T,tree<int>::iterator n, list<int> &L,list<int>::iterator &p);` tal que contruye en el nodo `n` el subárbol que comienza en la lista en la posición `p`. El nodo `n` es inicialmente no dereferenciable (Λ). Para ello

- Extrae de la lista el valor `*n` almacenado en el nodo y la cantidad de hijos `nchild`.
- Inserta en `n` el valor.
- Va aplicando recursivamente `list2tree()` sobre los hijos, hasta completar `nchild`.

`list2tree` retorna el iterator *refrescado* al nodo `n`.

b) **[cumsum (20pt)]** *Consigna:* Escribir una función `void cumsum(btree<int> &T);` (por **suma acumulada**) que dado un **árbol binario (AB)** `T` modifica el valor de los nodos interiores, de manera que resulta ser la suma de los valores de sus hijos **más** el valor que había en el nodo antes de llamar a `cumsum()`. Los valores de las hojas no son modificados, y los valores de los nodos interiores resultan ser la suma de todos los valores del subárbol del nodo **antes** de llamar a `cumsum`.
Ejemplo: Si `T=(1 5 (2 3 7 (11 4 2)))` entonces después de llamar `cumsum(T)` debe quedar `T=(35 5 (29 3 7 (17 4 2)))`.

[Ej. 3] [operativos (total 20pt)]

a) **[rec-arbol (10pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
   - `ORD_PRE` $=\{Z, X, A, B, D, E, H, K, C, \}$,
   - `ORD_POST` $=\{X, D, H, K, E, B, C, A, Z, \}$,

b) **[huffman (10pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `CATUPECU`
$P(C) = 0.10, P(A) = 0.10, P(T) = 0.10, P(U) = 0.30, P(P) = 0.02, P(E) = 0.03, P(U) = 0.35$. Calcular la longitud promedio del código obtenido.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Realice los pasos necesarios (sentencias de C/C++) para construir el siguiente árbol binario
`T=(1 (3 . (2 10 20)) (5 7 .))`

b) Escriba la *"definición recursiva"* de la función ALTURA de un AOO.

c) Comente puntos a favor y puntos en contra del método de **Huffman** para la codificación de palabras/caracteres.

d) Explique porqué el método de Huffman cumple con la *"condición de prefijos"*.

e) Cual es el **orden del tiempo de ejecución** (en promedio) en función del número de elementos (n) que posee el **árbol ordenado orientado (AOO)** implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos

1) `*n`
2) `begin()`
3) `lchild()`
4) `insert(p,x)`
5) `operador++` (prefijo o postfijo)
6) `erase(p)`
7) `find(x)`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. [2010-11-25]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)] Insistimos: deben usar la interfaz STL.

a) Escribir la función `btree<int>::iterator abb_find(btree<int> &T,int x);` que devuelve el iterator en el ABB `T`, donde se encuentra el elemento `x`, o caso contrario `T.end()`.
Restricción: La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.

b) Escribir la función `pair<int,int> minmax(btree<int> &T);` que devuelve el mínimo y el máximo de el ABB `T`. Si el árbol está vacío debe devolver las constantes `(INT_MAX,INT_MIN)`.
Restricción: La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.

c) Escribir una función `void sort(vector<int> &v);` que ordena los elementos de `v` utilizando el operador `<` como función de comparación. Puede elegir cualquier algoritmo de ordenamiento rápido o lento visto en el curso. **Especifique cuál es el algoritmo que está usando.**
Restricción: El algoritmo debe ser *in-place*, es decir no debe usar contenedores auxiliares.

d) Implementar una función `bool openhashtable_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T), T x)` que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y retorna un booleano indicando si la inserción fue o no exitosa.

[Ej. 2] [Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.

a) **[ident-set (20pt)]**
Dado una serie de n conjuntos distintos S_j para $j = 0, ..., n-1$, y un conjunto S que es alguno de los S_j queremos identificar cuál de ellos es, es decir encontrar k tal que $S_j = S$.
Consigna: *Escribir una función* `int ident_set(vector<set<int>> vecset,set<int> s);` *que rertorna el índice* `k` *tal que* `vecset[k]=s`.
Para ello proponemos el siguiente algoritmo:

1) Construimos un `vector<set<int>> vecset` tal que `vecset[j]`=S_j, y un vector de enteros `vector<int> indx(n)`, que inicialmente es la identidad `indx[j]=j`.
2) Escribir una función `void split_vecset(vector<set<int>> &vecset,vector<int> &indx,int x);` que, dado un entero `x` deja en `vecset` los conjuntos que contienen a `x` y en `indx` los índices correspondientes. Así por ejemplo si `vecset[0]`$= (0, 1, 3, 8)$, `vecset[1]`$=(1, 3, 5, 9)$, `vecset[2]`$=(2, 4, 7, 8)$, y `indx=(1,5,6)` entonces después de hacer `split_vecset(vecset,indx,8);` debe quedar `vecset[0]`$= (0, 1, 3, 8)$, `vecset[1]`$=(2, 4, 7, 8)$, y `indx=(1,6)`. En todo momento el tamaño de `vecset` y `indx` debe ser el mismo.
3) Para cada `x`$\in S$, aplicar la función anterior `split_vecset` para extraer los conjuntos que contienen a ese x en particular, hasta que quede un solo conjunto. El índice correspondiente en `indx` es el `k` buscado.

b) **[contract-edge (20pt)]** Escribir una función `void contract_edge(map<int, set<int> > &G, pair<int,int> &e);` que remueve del grafo simple G la arista e y uno de sus vértices, dejando en G el vértice de menor etiqueta.

Sea $e = (u, v)$ con $u < v$, entonces los pasos a realizar por la función son:
1) **Actualizar la lista de adyacencia** de u incorporando la adyacencia de v (tener cuidado de no agregar u en la nueva adyacencia de u para no generar lazos),
2) Para cada vértice x de la adyacencia de v, **remover** v **de la adyacencia**de x
3) Para cada vértice x de la adyacencia de v, si $x \neq u$ entonces **agregar** u **a la adyacencia de** x**,**
4) **Remover el vértice** v y su adyacencia del grafo G.
Ejemplo: Sea

```
G=[1 -> {2,3,4},
  2 -> {1,3,5,6},
  3 -> {1,2},
  4 -> {1,5},
  5 -> {2,4},
  6 -> {2}]
```

y `e=(1,2)`. Entonces despues de hacer `remove_edge(G,e); G` tiene que quedar:

```
G=[1 -> {3,4,5,6},
     3 -> {1},
     4 -> {1,5},
     5 -> {1,4},
     6 -> {1}]
```

[Ej. 3] [operativos (total 20pt)]

a) **[abb (5 pts)]** Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 2, 11\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 17, 9 y 6 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $1, 17, 27, 10, 9, 37, 19, 6$ en una **tabla de dispersión cerrada** con $B = 8$ cubetas, con función de dispersión `h(x) = x % 8` y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{2, 6, 9, 3, 4, 14, 11\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
d) **[quick-sort (5 pts)]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Escriba el código para ordenar un vector de enteros `v` por **valor absoluto**, es decir escriba la función de comparación correspondiente y la llamada a `sort()`.
Nota: recordar que la llamada a `sort()` es de la forma `sort(p,q,comp)` donde `[p,q)` es el rango de iteradores a ordenar y `comp(x,y)` es la función de comparación.
b) Cual es el tiempo de ejecución en el caso **promedio** para el método `insert(x)` de la clase **diccionario** (`hash_set`) implementado por **tablas de dispersión cerradas**, en función de la `tasa de llenado` $\alpha = n/B$, para el caso de inserción exitosa y no exitosa.
c) Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**.
d) Se quiere representar el conjunto de enteros múltiplos de 3 entre 30 y 99 (o sea $U = \{30, 33, 36, ..., 99\}$) por **vectores de bits**, escribir las funciones `indx()` y `element()` correspondientes.
e) Discuta la complejidad algorítmica de las operaciones binarias `set_union(A,B,C)`, `set_intersection(A,B,C)`, y `set_difference(A,B,C)` para conjuntos implementados por vectores de bits, donde `A`, `B`, y `C` son subconjuntos de tamaño n_A, n_B, y n_C respectivamente, de un conjunto universal `U` de tamaño N.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos. Recuperatorio. [2010-12-02]

ATENCIÓN: Recordar que tanto en las clases como en los ejercicios de programación **deben usar la interfaz STL**.

- **[CLASES-P1]**

  1. Escribir la implementación en `C++` del **TAD lista** (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `begin()`, `end()`.
  2. Escribir la implementación en `C++` de los métodos `push`, y `pop` de los **TAD pila y cola** (clases `stack` y `queue`).
  3. Escribir la implementación en `C++` del método `find(x)` para el **TAD correspondencia** (clase `map`) implementado con los listas ordenadas.

- **[CLASES-P2]**

  1. **AOO:** declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `tree<T>::iterator tree<T>::erase(tree<T>::iterator n);`
  2. **AB:** declarar las clases `btree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `btree<T>::iterator btree<T>::insert(btree<T>::iterator n, const T& x)`

- **[CLASES-P3]**

  1. Implementar `set::iterator set::find(T x)` para conjuntos implementados por ABB. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.
  2. Implementar `int set::erase(T x)` para conjuntos implementados por listas ordenadas. Si se utiliza algún método auxiliar, también implementarlo. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.
  3. Escribir la función de ordenamiento por fusión `void merge_sort(list<int> &L)` para listas de enteros. Los enteros se comparan por el operador `<`.

- **[PROG-P1]**

  1. [max-dev-n] Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: $\text{max_dev}(a_1, a_2, ..., a_n) = (\text{máx}_{j=1}^{n} a_j) - (\text{mín}_{j=1}^{n} a_j)$. Escribir una función `int max_dev_m(list<int> &l, int m);` que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

  $$\begin{aligned}\text{max_dev_m(L)} = \ &\text{máx}\{\text{max_dev}(a_1, a_2, \ldots, a_m),\\ &\text{max_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max_dev}(a_{n-m+1}, \ldots, a_n)\}\end{aligned} \quad (1)$$

  Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces `max_dev_m(L,3)` debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones.
  2. [chunk-revert] Escribir una función `void chunk_revert(list<int> &L,int n);` que dada una lista `L` y un entero `n`, invierte los elementos de la lista tomados de a `n`. Si la longitud de la lista no es múltiplo de `n` entonces se invierte el resto también. Por ejemplo, si `L={1,3,2,5,4,6,2,7}` entonces después de hacer `chunk_revert(L,3)` debe quedar `L={2,3,1,6,4,5,7,2}`. **Restricciones:** Usar a lo sumo una estructura auxiliar. (En tal caso debe ser lista, pila o cola).

3. [cum-sum-pila] Escribir una función `void cum_sum(stack<int> &P)` que modifica a `P` dejando la suma acumulada de los elementos, es decir, si los elementos de `P` antes de llamar a `cum_sum(P)` son $(a_0, a_1, \ldots, a_{n-1})$, entonces después de llamar a `cum_sum(P)` debe quedar `P`=$(a_0, a_0 + a_1, \ldots, a_0 + a_1 + \ldots + a_n)$. Por ejemplo, si `P`=$(1, 3, 2, 4, 2)$ entonces después de hacer `cumsum(P)` debe quedar `P`=$(1, 4, 6, 10, 12)$. Usar una pila auxiliar.
Restricciones:
   - Usar la interfase STL para pilas (`clear()`, `top()`, `pop()`, `push(T x)`, `size()`, `empty()`).
   - No usar más estructuras auxiliares que la indicada ni otros algoritmos de STL.
   - El algoritmo debe ser $O(n)$.

- **[PROG-P2]**

1. **[check-ordprev]** (AOO)
Escribir una función `bool check_ordprev(tree<int> &T,list<int> &L);` que, dado un árbol ordenado orientado `T` retorna `true` si la lista `L` contiene al listado en orden previo de `T`. Por ejemplo, si `T=(3 (4 2 1) 0 (6 7 (8 9 5)))` y `L={3,4,2,1,0,6,7,8,9,5}` entonces `check_ordprev` debe retornar `true` y si por ejemplo `L={3,4,2,1,0,6,7,8,9,5,1000}` o `L={3,4,2,1,0,6,7000,8,9,5}` `check_ordprev` debe retornar `false`. Una manera simple de hacerlo es iterar en el árbol en la manera habitual e ir sacando los elementos de la lista `L` si coinciden con el contenido del nodo o posición actual. Una vez que se recorrió todo el árbol se verifica en la función *"wrapper"* que la lista haya quedado vacía. Se debe retornar `true` si este es el caso. La función `check_ordprev` está definida tal que:
   - si el árbol es vacío y la lista no lo es (o viceversa) `check_ordprev` es `false`,
   - si lo anterior no ocurre y si el árbol es vacío `check_ordprev` es `true`,
   - si no ocurre lo anterior y los contenidos del nodo y la posición de la lista actual no coinciden `check_ordprev` es `false`,
   - si no ocurre lo anterior, elimino el elemento actual de la lista,
   - llamo recursivamente a la función verificando que no haya llegado al fin de lista.

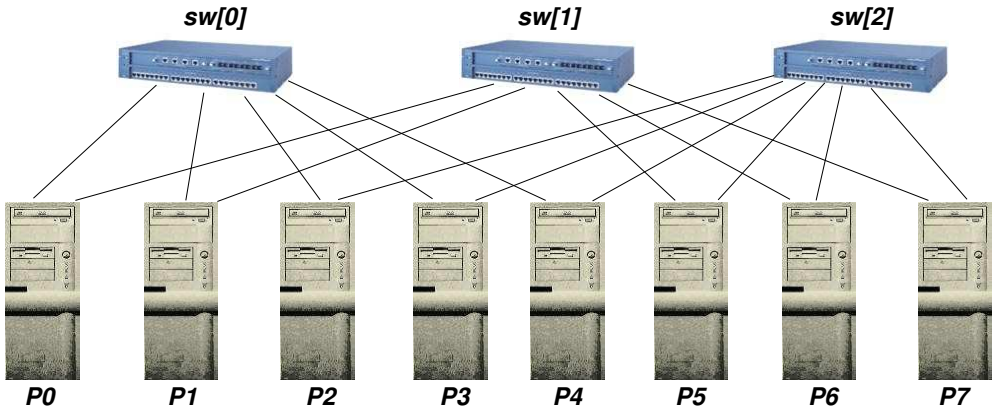
   Nota: se pueden usar todas las funciones de lista de STL sin restricciones.

2. **[depth-if]** (AB) Dado un AB `T` encontrar, la profundidad máxima de los nodos que satisfacen un cierto predicado (*profundidad condicionada*). Por ejemplo, si `T=(6 (7 9 (3 1)) 2)`, entonces `depth_if(T,even)` debe retornar 1, ya que el nodo par a mayor profundidad es 2, mientras que `depth_if(T,odd)` debe retornar 3, ya que el nodo impar a máxima profundidad es 1.
Consigna: Escribir una función `int depth_if(btree<int> &T,bool (*pred)(int));` que realiza la tarea indicada.
Ayuda: La función recursiva auxiliar debe retornar la máxima profundidad de un nodo que satisface el predicado o -1 si el árbol está vacío o ningún nodo satisface el predicado. Entonces, dadas las profundidades condicionadas d_l, d_r de los hijos la profundidad se puede expresar en forma recursiva como

$$\text{depth}(n) = \begin{cases} -1, & \text{si } n \text{ es } \Lambda, \\ \max(d_r, d_l) + 1, & \text{si } \max(d_r, d_l) \geq 0, \\ 0, & \text{si } n \text{ satisface el predicado,} \\ -1, & \text{si } n \text{ no satisface el predicado.} \end{cases}$$

- **[PROG-P3]**

1. **[flat]** Se está diseñando una red inerconectada por switches y se desea, para reducir lo más posible la *latencia* entre nodos, que cada par de nodos esté conectado en forma directa por al menos un switch. Sabemos que el número de nodos es `n` y tenemos un `vector< set<int> > sw` que contiene para cada switch el conjunto de los nodos conectados por ese switch, es decir `sw[j]` es un conjunto de enteros que representa el conjunto de nodos inteconectados por el switch j.
Consigna: Escribir una función predicado `bool flat(vector< set<int> > &sw, int n);` que retorna verdadero si cada par de enteros (j, k) con $0 \leq j, k < n$ está contenido en al menos uno de los conjunto en `sw[]`.



En el ejemplo de la figura tenemos 8 nodos conectados via 3 switches y puede verficarse que cualquier par de nodos está conectado en forma directa a través de al menos un switch. Para este ejemplo el vector `sw` sería

$$\texttt{sw[0]} = \{0,1,2,3,4\}, \quad \texttt{sw[1]} = \{0,1,5,6,7\}, \quad \texttt{sw[2]} = \{2,3,4,5,6,7\} \tag{2}$$

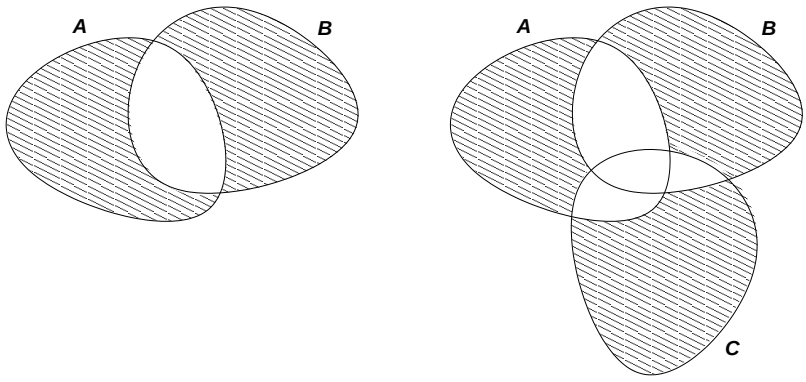
Por lo tanto `flat(sw,8)` debe retornar `true`. Por otra parte si tenemos

$$\texttt{sw[0]} = \{0,2,3,4\}, \quad \texttt{sw[1]} = \{0,1,5,7\}, \quad \texttt{sw[2]} = \{2,3,5,6,7\} \tag{3}$$

entonces los pares $(0,6)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,6)$, $(4,5)$, $(4,6)$ y $(4,7)$ no están conectados en forma directa y `flat(sw,8)` debe retornar `false`.
Sugerencia 1: Recorrer todos los pares de valores (j,k) y para cada par recorrer todos los conjuntos en `sw[]` hasta encontrar uno que contenga al par.
Sugerencia 2: Puede ser de ayuda el escribir una función auxiliar `bool estan_conectados(sw,j,k)`.

2. **[diff-sym]** Para dos conjuntos A, B, la *"diferencia simétrica"* se define como

$$\begin{aligned} \text{diff_sym}(A,B) &= (A-B)\cup(B-A), \quad \text{o también} \\ &= (A\cup B)-(A\cap B) \end{aligned}$$



En general, definimos la diferencia simétrica de varios conjuntos como el conjunto de todos los elementos que pertenecen a uno y sólo uno de los conjuntos. En las figuras vemos en sombreado la diferencia simétrica para dos y tres conjuntos. Por ejemplo, si $A = \{1,2,5\}$, $B = \{2,3,6\}$ y $C = \{4,6,9\}$ entonces $\text{diff_sym}(A,B,C) = \{1,3,4,5,9\}$.
Consigna: Escribir una función `void diff_sym(list<set<int> > &l,set<int>&s);` que retorna en `s` la diferencia simétrica de los conjuntos en `l`.
Ayuda: La solución se puede encarar con alguna de las dos estrategias siguientes:

a) Escribir una función `int cuenta(list<set<int> > &l,int x);` que retorna el número de conjuntos de `l` en los cuales x está incluido. Recorrer todos los elementos de todos los conjuntos de `l`, e insertar el elemento en `s` sólo si `cuenta` retorna exactamente 1.

b) Notar que en el caso de tres conjuntos si $S = \text{diff_sym}(A, B)$ y $U = A \cup B$, entonces $\text{diff_sym}(A, B, C) = (S - C) \cup (C - U)$. Esto vale en general para cualquier número de conjuntos, de manera que podemos utilizar el siguiente lazo

l = lista de conjuntos, $S = \emptyset, U = \emptyset$;
for Q en la lista de conjuntos l **do**
  $S = (S - Q) \cup (Q - U)$;
  $U = U \cup Q$;
end for

Al terminar el lazo, S es la diferencia simétrica buscada.

Nota: Recordar que para las operaciones binarias (por ej. `set_union(A,B,C);`), los conjuntos deben ser distintos, es decir no se puede hacer `set_union(A,B,A);`, en ese caso debe hacerse `set_union(A,B,tmp); tmp=A;`

Nota: Para usar las operaciones binarias se puede usar tanto la versión simplificada que hemos utilizado en el curso, `void set_union(set &A,set &B,set &C);` como la versión de las STL `set_union(A.begin(),A.end(),B.begin(),B.end(),inserter(C,C.begin()));`

- [**OPER-P2**]

  1. [**huffman**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `WIKILEAK`
     $P(W) = 0.1, P(I) = 0.1, P(K) = 0.3, P(L) = 0.1, P(E) = 0.2, P(A) = 0.05, P(Z) = 0.05, P(T) = 0.1$
     Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.
  2. [**rec-arbol**] Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son
     - `ORD_PRE` $=\{Q, R, T, Y, L, W, A, B, C, Z, \}$,
     - `ORD_POST` $=\{R, L, A, B, C, W, Y, Z, T, Q, \}$,
  3. [**make-tree**] Escribir la secuencia de sentencias necesarias para construir el arbol binario `(5 (2 . 3) (7 6 8))`. (**Restricciones:** No usar `find()`).
  4. [**modif-tree**] Escribir la secuencia de secuencia necesarias para modificar el árbol ordenado orientado `A = (2 3 (5 6 7))` de manera que se convierta en `A = (2 (3 (5 6 7)))`, usando el método `splice()`.

- [**OPER-P3**]

  1. [**heap-sort**] Dados los enteros $\{2, 6, 9, 3, 4, 14, 5\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de **cada** inserción/supresión.
  2. [**quick-sort**] Dados los enteros $\{7, 11, 6, 3, 7, 12, 10, 5, 5, 4, 13, 8\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición.
  3. [**abb**] Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 0, 11\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 12, 6 y 3 en ese orden.
  4. [**hash-dict**] Insertar los números $2, 15, 25, 8, 7, 35, 17, 4, 27$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$ `mod` 10 y estrategia de redispersión lineal.

- **[PREG-P1]**

1. Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = 5 \cdot 2^n + 2n^3 + 3\,n!+$$
$$T_2 = 2 \cdot 10^n + \sqrt{3} \cdot n + \log_8 n+$$
$$T_3 = n^2 + 5 \cdot 3^n + 4^{10}$$
$$T_4 = 2.3 \log_8 n + \sqrt{n} + 5\,n^2 + 2\,n^5$$
$$T_5 = 1000 + 3 \log_4 n + 8^2 + 5 \log_{10} n$$

2. ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `map<>` implementada con **listas desordenadas** en el caso promedio?
   Métodos: `find(key)`, `M[key]`, `erase(key)`, `erase(p)`, `begin()`, `end()`, `clear()`).

3. ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si en realidad es siempre verdad que $(n+1)^2 > n^2$?
4. Discuta las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas.
5. ¿Qué ventajas o desventajas tendría implementar la clase *pila* en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista?

- **[PREG-P2]**

1. Realice los pasos necesarios (sentencias de C/C++) para construir el siguiente árbol binario `T=(1 (3 . (2 10 20)) (5 7 .))`
2. Escriba la *"definición recursiva"* de la función ALTURA de un AOO.
3. Comente puntos a favor y puntos en contra del método de **Huffman** para la codificación de palabras/caracteres.
4. Explique porqué el método de Huffman cumple con la *"condición de prefijos"*.
5. Cual es el **orden del tiempo de ejecución** (en promedio) en función del número de elementos (n) que posee el **árbol ordenado orientado (AOO)** implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos
   a) `*n`
   b) `begin()`
   c) `lchild()`
   d) `insert(p,x)`
   e) `operador++` (prefijo o postfijo)
   f) `erase(p)`
   g) `find(x)`

- **[PREG-P3]**

1. Escriba el código para ordenar un vector de enteros `v` por **valor absoluto**, es decir escriba la función de comparación correspondiente y la llamada a `sort()`.
   Nota: recordar que la llamada a `sort()` es de la forma `sort(p,q,comp)` donde `[p,q)` es el rango de iteradores a ordenar y `comp(x,y)` es la función de comparación.
2. Cual es el tiempo de ejecución en el caso **promedio** para el método `insert(x)` de la clase **diccionario** (`hash_set`) implementado por **tablas de dispersión cerradas**, en función de la `tasa de llenado` $\alpha = n/B$, para el caso de inserción exitosa y no exitosa.
3. Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**.
4. Se quiere representar el conjunto de enteros múltiplos de 3 entre 30 y 99 (o sea $U = \{30, 33, 36, ..., 99\}$) por **vectores de bits**, escribir las funciones `indx()` y `element()` correspondientes.
5. Discuta la complejidad algorítmica de las operaciones binarias `set_union(A,B,C)`, `set_intersection(A,B,C)`, y `set_difference(A,B,C)` para conjuntos implementados por vectores de bits, donde `A`, `B`, y `C` son subconjuntos de tamaño n_A, n_B, y n_C respectivamente, de un conjunto universal `U` de tamaño N.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [2011-12-22]

ATENCIÓN: Recordar que tanto en las clases como en los ejercicios de programación **deben usar la interfaz STL**.

1. **[clases (20pt,min 50 %)]**.

   a) **[stack (6pt)]** Escribir la implementación del TAD pila (clase `stack`) usando listas.
   b) **[oht-erase (7pt)]** Escribir una función `bool oht_erase(vector<list<T>> &V, bool (*equal)(T,T), unsigned int hash(T), T x);` que elimina el elemento `x` de la tabla de dispersión abierta `V`, utilizando la función de hash `hash()` y la relación de equivalencia dada por `equal()`.
   c) **[btree (7pt)]**. Escribir la implementación en C++ del `TAD ARBOL BINARIO` (clase `btree`). Las funciones a implementar son `erase(p)` y `find(x)`.

2. **[programacion (40pt,min 50 %)]**.

   a) **[nilpot (25 puntos)]**. Dadas dos correspondencias M_1 y M_2 la *"composición"* de ambas es la correspondencia $M = M_2 \circ M_1$ tal que si $M_1[a] = b$ y $M_2[b] = c$, entonces $M[a] = c$. Por ejemplo, si `M1={(0,1),(1,2),(2,0),(3,4),(4,3)}`, y `M2={(0,1),(1,0),(2,3),(3,4),(4,2)}`, entonces $M = M_1 \circ M_2$ `={(0,0),(1,3),(2,1),(3,2),(4,4)}`. Notemos que para que sea posible componer las dos correspondencias es necesario que los valores del contradominio de M_1 estén incluidos en las claves de M_2. Si el conjunto de valores del contradominio de una correspondencia M está incluido en el conjunto de sus claves, entonces podemos componer a M consigo misma, es decir $M^2 = M \circ M$. Por ejemplo, $M_1^2 = M_1 \circ M_1$`= {(0,2),(1,0),(2,1),(3,3),(4,4)}`. De la misma manera puede definirse, $M^3, ..., M^n$, componiendo sucesivamente. Puede demostrarse que, para algún n debe ser $M^n = I$, donde I es la *"correspondencia identidad"*, es decir aquella tal que $I[x] = x$. Por ejemplo, si `M = {(0,1),(1,2),(2,0)}`, entonces para $n = 3$, $M^n = M^3 = I$.
   Consigna: Escribir una función `int nilpot(map<int,int> &M);` que dada una correspondencia `M` retorna el mínimo entero `n` tal que $M^n = I$.
   Sugerencia: Escribir dos funciones auxiliares:

   - `void compose(map<int,int> &M1,map<int,int> &M2,map<int,int> &M);` que dadas dos correspondencias `M1, M2`, calcula la composición $M = M_2 \circ M_1$, devolviéndola en el argumento `M`,
   - `bool is_identity(map<int,int> &M);` que dada una correspondencia `M`, retorna `true` si `M` es la identidad, y `false` en caso contrario.

   b) **[elimina-valor (15 puntos)]**.
   Escribir una función `void elimina_valor(queue<int>&C, int);` que elimina todos las ocurrencias del valor `n` en la cola `C`. Por ejemplo, si `C = {1,3,5,4,2,3,7,3,5}`, después de `elimina_valor(C,3)` debe quedar `C = {1,5,4,2,7,5}`. *Sugerencia: Usar una estructura auxiliar lista o cola.*
   Restricciones: El algoritmo debe tener un tiempo de ejecución $O(n)$, donde n es el número de elementos en la cola original.

3. **[operativos (20pt,min 50 %)]**.

   a) **[particionar (5pt)]**. Considerando el árbol `(f (a c e (h b)) (d g))` decir cuál son los nodos `descendientes(b)`, `antecesores(b)`, `izquierda(b)` y `derecha(b)`.

   b) **[heap-sort (5pt)]**. Dados los enteros $\{5, 0, 8, 2, 4, 5, 2, 4, 1, 3\}$. ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) **antes** y **después** de **cada** inserción/supresión.

   c) **[huffman (5pt)]**. Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario (para **todos** los caracteres) y encodar la palabra `FAVALORO`: $P(A) = 0.24$, $P(F) = 0.12$, $P(G) = 0.10$, $P(L) = 0.10$, $P(M) = 0.10$, $P(O) = 0.14$, $P(R) = 0.08$, $P(S) = 0.06$, $P(V) = 0.06$. Indicar el número de nivel de cada caracter y calcular la longitud promedio del código obtenido.

   d) **[rec-arbol (5pt)]**. Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- ORD_PRE=$\{B, F, E, H, I, D, J, A, C, G\}$,
- ORD_POST=$\{E, F, I, D, A, J, C, H, G, B\}$.

4. **[preguntas (20pt,min 60 %)]**.

   a) Dado el árbol binario `(z (a b q) r)`, ¿Es completo? ¿Es lleno? Justifique

   b) ¿ Cuál es el tiempo de ejecución para intersección de conjuntos por vectores de bits?

   c) Sea el árbol `(5 7 (8 6 9))`. Después de hacer:

   ```
   n = D.find(7);
   n++;
   n = n.lchild();
   n = n.lchild();
   n = D.insert(n,2);
   ```

   ¿Cómo queda el árbol? ¿Se produce un error?

   d) ¿Cómo es el tiempo de ejecución para intercalar dos listas clasificadas de n elementos?

   e) ¿Cuál es el tiempo de ejecución de la función `find(key)` para correspondencias implementadas por vectores ordenados?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [2011-09-15]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), y un 60 % sobre las preguntas de teoría (Ej 3).

[Ej. 1] [clases (30pt)]

a) **[lista (20pt)]** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por celdas (simple o doblemente) enlazadas por punteros **ó** cursores. (**Indicar claramente qué implementación elige.**) Los métodos a implementar son `insert(p,x)`, `erase(p)`, `iterator::operator++(int)` (postfijo), `iterator::operator++()` (prefijo).

b) **[pila-cola (10pt)]** Escribir la implementación en `C++` de los métodos `push`, `pop`, `front`, y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

[Ej. 2] [Programación (total = 50pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL**.

a) **[expand (20pt)]** **Consigna:** Escribir una función `void expand(list<int>  &L,int m);` que transforma los elementos de una lista `L` de tal forma que todos los elementos de `L` resulten ser menores o igual que `m`, pero de tal forma que su suma se mantenga inalterada. Para esto, si un elemento `x` es mayor que `m` entonces, lo divide en tantas partes como haga falta para satisfacer la condición; por ejemplo si `m=3` podemos dividir a `10` en `3,3,3,1`. Es decir si `L=(7,2,3,1,4,5)`, entonces después de hacer `expand(L,2)` debe quedar `L=(2,2,2,1,2,2,1,1,2,2,2,2,1)`.

b) **[rota (10pt)]** Escribir una función `void rota(stack<int> &S, queue<int> &Q);` que realiza las siguientes operaciones (sea `m` el número de elementos en la pila):
1) Pone todos los elementos de la pila `S` en la cola `Q`;
2) Saca `m` elementos de `Q` y los pone en `S`.
Así por ejemplo si `S=(1,3,2,4)` y `Q=(6,2,5)` entonces después de llamar `rota(S,Q)` debe quedar `S=(1,5,2,6)` y `Q=(3,2,4)`.

c) **[is-rotation (20pt)]** Determinar si una correspondencia `map<int,int> M` es una *"rotación"* es decir, una permutación tal que cada elemento del conjunto de las claves es asignado al siguiente elemento, en cierto orden. Por ejemplo `M={1->3,2->8,3->5,4->2,5->4,8->1}` corresponde a una rotación en orden `(1,3,5,4,2,8)`.
Consigna: escribir una función `bool is_rotation(map<int,int> &M,list<int> &order);` que determinar si `M` es una rotación y en ese caso devuelve en `L` la lista de los elementos de `M` en el orden dado por la correspondencia.
Ayuda: Generar la lista `L` aplicando sucesivamente $x_{i+1} = M(x_i)$ a partir de cualquier clave inicial x_0. La correspondencia es una rotación si
- El valor del contradominio x_{i+1} siempre es una clave.
- Los elementos de la secuencia son todos distintos, salvo para el último elemento, cuando $x_n = x_0$.

(**Nota:** Utilizar una correspondencia auxiliar para saber cuales claves ya fueron visitadas.)

[Ej. 3] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$T_1 = 10 + 2\log_3 n^{1.2} + 5^4 + 3.14159\log_2 n$$
$$T_2 = n^3 + 2 \cdot 2^n + 3^5$$
$$T_3 = 2 \cdot 5^n + \sqrt{5} \cdot n + \log_3 n$$
$$T_4 = 2 \cdot 3^n + 4n^5 + 6\,n!$$
$$T_5 = 5\log_2 n + \sqrt{n} + 6\,n^4 + 5\,n^2$$

b) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `lista<>` implementada con **celdas simplemente enlazadas** (por punteros o cursores) en el caso promedio?
Métodos: `insert(p,x)`, `*p`, `erase(p)`, `clear()`, `begin()`, `end()`.

c) ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si en realidad es siempre verdad que $(n+1)^2 > n^2$?

d) ¿Qué ocurre si ejecutamos el siguiente código? `S` es una pila de enteros que puede estar vacía o no.

```
S.push(3); S.push(5); S.top(); S.top(); x=S.top();
```

¿Puede dar un error porque `S` se queda vacía? ¿Que valor toma `x`?

e) Discuta ventajas y desventajas de usar contenedores lineales **ordenados** o **desordenados** para representar correspondencias.

[Ej. 4] **[lab (bool)]** Dado un `map<int, list<bool> >` `M`, verificar que para todas las claves pares, la lista correspondiente tenga todos sus elementos en true o bien que sea vacía. Si el map no tiene elementos, la salida debe ser true.

Ejemplos:

```
M = {2 -> [true, true], 4 ->[]} -> returns true
M = {1 -> [false]} -> returns true
M = {2 -> [true, true, true], 3 -> [true, false]} -> returns true
M = {} -> returns true
M = {6 -> [false, true], 4 -> [true, true]} -> returns false
```

La función debe tener la siguiente signatura `bool mostrar_map(map<int, list<bool> >&);`

Instrucciones para la autoverificación: Junto se provee un archivo llamado `evaluar.hpp` que contiene una clase llamada `Evaluar` dentro de un namespace llamado `aed`. Esta clase la deben utilizar para comprobar si lo que hicieron es correcto. Lo único que tienen que hacer es incluir el archivo en su programa, y en el main crear un objeto de esta clase. Como parámetro al constructor deben pasarle un puntero a la función que resuelve el ejercicio (simplemente escriben el nombre de la función). Por ejemplo: `aed::Evaluar obj(mostrar_map);` en el caso de que su función se llame `mostrar_map`. Si el programa les muestra **todo** `BIEN`, es porque está bien. Si en algún caso en particular hay algún error, se les indicará cual era la respuesta esperada (la correcta) y cual es la que ustedes obtuvieron.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [2011-10-27]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] **[clases (20pt)] Insistimos: deben usar la interfaz STL ("AVANZADA").**

a) **[AOO (10pt)]** Para el TAD Arbol Ordenado Orientado (AOO):
- Declarar las clases `tree`, `cell`, `iterator`, con su correspondiente anidamiento, incluyendo las declaraciones de datos miembros.
- Implementar los métodos
  - `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x);`
  - `tree<T>::iterator tree<T>::begin();`

b) **[AB (10pt)]** Para el TAD Arbol Binario (AB):
- Declarar las clases `btree`, `cell`, `iterator`, con su correspondiente anidamiento, incluyendo las declaraciones de datos miembros.
- Implementar los métodos
  - `btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`,
  - `btree<T>::iterator btree<T>::iterator::right();`.

[Ej. 2] **[Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.**
Atención: Hay 3 ejercicios cuya suma es **50**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,n1+n2+n3)`).

a) **[make-mirror (20pt)]** Escribir una función `void make_mirror(tree<int> &T);` que convierte **in-place** al árbol `T` en su espejo. Por ejemplo si `T=(1 (5 3 2 1) (10 9 7))` entonces después de hacer `make_mirror(T)` debe quedar `T=(1 (10 7 9) (5 1 2 3))`. **Ayuda:** Debe ir invirtiendo la lista de hijos para cada nodo `n` y aplicar la función recursivamente sobre los hijos. Para invertir los hijos se puede usar alguna de las siguientes opciones:
- Usar un árbol auxiliar `Taux`, insertar un elemento cualquiera (p.ej. 0) en la raíz y hacer *splice* para ir moviendo cada uno de de los hijos de `n` a `Taux`. Luego volver a mover (con *splice*) todos los hijos de `Taux` en `n`, pero de forma que queden invertidos.
- Usar una lista de árboles `list<tree<int>> L`. Mover los hijos de `n` sucesivamente en nuevos árboles en la lista. Para insertar un nuevo árbol en la lista simplemente hacemos `L.push_back(tree<int>())`.
- No usar ningún contenedor auxiliar, ir recorriendo los hijos de `n` y moverlos en `n.lchild()`. En este caso tener cuidado con los iteradores.

Curiosidad: Notar que `ordprev(make_mirror(T))=reverse(ordpost(T))` y `ordpost(make_mirror(T))=reverse(ordprev(T))`.

b) **[is-balanced (20pt)]** Un árbol binario (AB) es balanceado si
- Es el árbol vacío ó,
- Sus subárboles derecho e izquierdo son balanceados, y sus alturas difieren a lo sumo en 1, o sea $|h_L - h_R| \leq 1$.

Por ejemplo `(1 (2 (3 (4 5 6) 7) (13 8 9)) (15 10 (16 11 12)))` es un árbol balanceado (notar que no necesariamente las profundidades de las hojas difieren en ± 1).
Consigna: Escribir una función `bool is_balanced(btree<int> &T);` que retorna `true` si el árbol está balanceado.
Ayuda: En la función auxiliar retornar el resultado de si el árbol es balanceado o no, pero además retornar (con un valor pasado por referencia, o con un `pair`) la altura del árbol correspondiente.

c) [**decomp-int (10pt)**] A partir de un número entero `m` escribir una función `void decomp_int(int m, btree<int> &T);` que construye el árbol binario `T` de la siguiente forma:

- En la raíz contiene `m`
- En los hijos izquierdo y derecho contiene los valores `ml` y `mr` computados como `int mr=m/2, ml=m-mr`. Si `mr` ó `ml` son nulos el nodo no es insertado.
- Propaga recursivamente la decomposición a los nodos.

Por ejemplo si `m=5` entonces el árbol generado es `(5 (3 (2 1 1) 1) (2 1 1))`.

[Ej. 3] [operativos (total 20pt)]

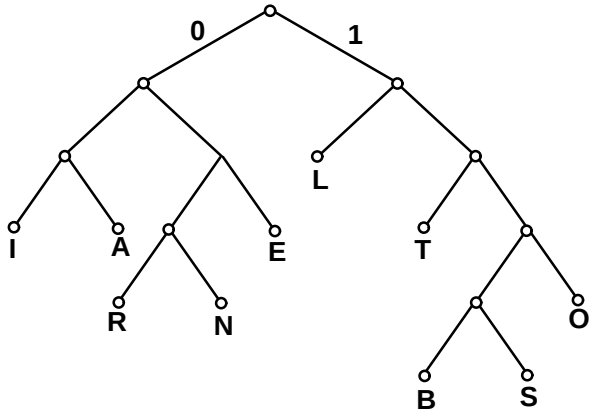
a) [**rec-arbol (8pt)**] Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Z, D, F, G, X, Y, W, Q, \}$,
- `ORD_POST` $=\{Z, F, X, Y, Q, W, G, D, A, \}$.

b) [**huffman (6pt)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `LIOMESSI`
$P(O) = 0.10, P(M) = 0.10, P(S) = 0.10, P(E) = 0.30, P(I) = 0.05, P(L) = 0.35$. Calcular la longitud promedio del código obtenido.

c) [**hf-decode (6pt)**]

Utilizando el código de la derecha desencodar el mensaje
`10001111000110100000101110111111101`



[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Realice los pasos necesarios (sentencias de C/C++) para construir el siguiente AOO
`T=(5 (7 9 8 (2 4 3)) (3 4))`

b) ¿Como se define (en forma abstracta, no pedimos el código) la notación Lisp para árboles?

c) ¿Se puede hacer insert en iteradores no dereferenciables en un árbol ordenado orientado (AOO)? ¿Y en los dereferenciables? ¿Y para árbol binario (AB)?

d) Dado el árbol `T=(1 4 (5 3 2 1) (10 9 7))` listar todos los nodos `L` que están a la izquierda, a la derecha `R`, antecesores propios `A`, y descendientes propios `D`, del nodo 5.

e) ¿Cómo se define el iterator para AB? Describa los miembros de la clase y que significan con un ejemplo.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. [2011-11-24]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)] Insistimos: deben usar la interfaz STL.

a) **[oht-erase (7pt)]** Escribir una función `bool oht_erase(vector<list<T>> &V, bool (*equal)(T,T), unsigned int hash(T), T x);` que elimina el elemento `x` de la tabla de dispersión abierta `V`, utilizando la función de hash `hash()` y la relación de equivalencia dada por `equal()`.

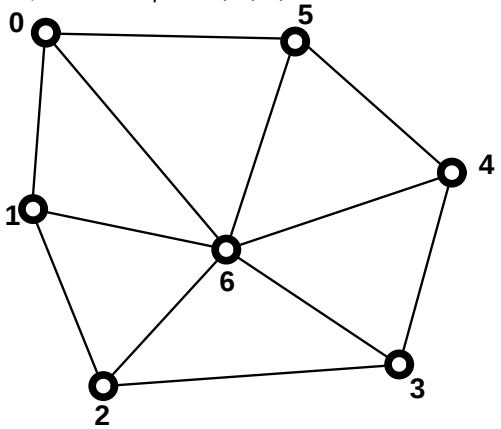
b) **[bst-find (7pt)]** Escribir la función `bool bst_find(btree<string> &T, bool (*less)(string,string), string x);` que devuelve `true` en caso que `x` esté en el ABB y `false` en caso contrario, utilizando `less()` como relación de orden.

c) **[bst-max (6pt)]** Escribir una función `string bst_max(btree<string> &T)` que devuelve el mínimo elemento de un árbol binario de búsqueda (ABB) `T`, siendo los valores contenidos en los nodos del árbol de tipo `string`. Si `T` es el árbol vacío, entonces debe retornar el string vacío (`""`).

[Ej. 2] [Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.

Atención: Hay 3 ejercicios cuya suma es **50**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,n1+n2+n3)`).
Nota: En los ejercicios con grafos siguientes, los grafos son simples y se representan como un `map<int, set<int> >G`, donde `G[j]` es el conjunto de vértices adyacentes (esto es, conectados por una arista) con el vértice `j`.

a) **[is-indep (20pt)]** Dado un grafo `map<int, set<int> >G` y un conjunto `set<int> S` de vértices del grafo, escribir un predicado `bool is_indep(map<int, set<int> >&G,set<int>&S);` que determina si `S` es un conjunto **independiente** de `G`. Se dice que `S` es un conjunto independiente de `G`, si para cada par de vértices de `S`, NO existe una arista que los une en `G`. Por ejemplo, en la figura de abajo los conjuntos `{0,2,4}` y `{1,3,5}` son independientes, mientras que `{0,2,4,6}` no lo es.



b) **[is-hamilt (20pt)]** Dado un grafo `map<int, set<int> >G` y una lista de vértices `list<int> L` determinar si `L` es un **camino hamiltoniano** en `G`.

Ayuda: Mantener un `set<int> visited` con todos los vértices visitados. Para dos enteros `x`, `y` sucesivos en la lista `L`, verificar si son vértices de `G` y si son adyacentes. Verificar que el nuevo vértice `y` no fuera visitado previamente e insertarlo en `visited`. Al final, chequear que todos los vértices fueron visitados.
Nota: Recordamos que un camino Hamiltoniano en un grafo es un camino en el mismo que pasa por todos los nodos sin repetir ninguno. Por ejemplo en el grafo de arriba el camino `{0,1,2,3,4,5,6}` es Hamiltoniano.

c) **[verif-color (10pt)]** Dado un grafo `map<int, set<int> >G` y una coloración `map<int,string> C` escribir un predicado `bool verif_color(map<int, set<int> >&G,map<int,string> C);` que determina si `C` es una coloración válida, es decir si dados dos vértices adyacentes `x`,`y` de `G` sus colores son diferentes.

[Ej. 3] [operativos (total 20pt)]

a) **[abb (5 pts)]** Dados los enteros $\{14, 8, 21, 3, 4, 11, 6, 5, 4, 13\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 14, 8 y 3 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $\{4, 20, 30, 13, 12, 40, 20, 22, 9\}$ en una **tabla de dispersión cerrada** con $B = 8$ cubetas, con función de dispersión `h(x) = 3*x-2` y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{5, 9, 12, 6, 7, 17, 14\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
d) **[quick-sort (5 pts)]** Dados los enteros $\{7, 11, 6, 3, 7, 12, 10, 5, 5, 4, 13, 8\}$ ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Cual es el costo de una **inserción exitosa** en una tabla de dispersión cerrada en función de su **tasa de ocupación**.
b) Discuta cuál es la influencia de las cubetas con valores `undef` y `deleted` en las **tablas de dispersión cerradas**.
c) Defina el concepto de **estabilidad** para algoritmos de ordenamiento. De un ejemplo.
d) Se quiere representar subconjuntos del conjunto universal de pares de enteros

$$U = \{(j, k) \ / \ 0 \leq j, k < 10\} \tag{1}$$

como un conjunto con **vectores de bits**.

1) ¿Cuál es el tamaño del conjunto universal?
2) Escribir las funciones `int indx(pair<int,int>)` y `pair<int,int> element(int)` correspondientes, que mapean el espacio de elementos del conjunto universal a índices y viceversa.

e) Discuta la cantidad de **intercambios** que realizan los 3 **algoritmos de ordenamiento lentos** que fueron estudiados.

mstorti@galileo/aed-2.0.4-53-g51a1e41/Thu Nov 24 13:08:17 2011 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2011-12-01]

ATENCIÓN: Recordar que tanto en las clases como en los ejercicios de programación **deben usar la interfaz STL**.

1. **[CLASES-P1]**

   a) Escribir la implementación en C++ del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)`, `list()`, `begin()`, `end()`.
   b) Escribir la implementación en C++ de los métodos `push`, `pop`,`front` y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

2. **[CLASES-P2]**

   a) **AOO:** declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
   `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`
   b) **AB:** declarar las clases `btree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
   `btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`

3. **[CLASES-P3]**

   a) Implementar `bool bst_erase(btree<T> t,T x,bool (*less)(T,T))` que elimina el elemento `x` del ABB T, retornando true si la eliminación fue exitosa, y false caso contrario.
   b) Implementar una función `pair<int,bool> oht_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T),bool (*equal)(T,T),T x)` que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y la relación de equivalencia `equal` retornando un par de `int` (que indica la cubeta) y `bool` que indica si la inserción fue exitosa.
   c) Implementar una función `void vecbit_difference(vector<bool>&, A, vector<bool>& B, vector<bool>& C);` que devuelve `C=A-B` con `A,B,C` vectores de bits que representan conjuntos de un rango contiguo de enteros `[0, N)`, donde `N` es el tamaño de los vectores `A,B,C` (asumir el mismo tamaño para los tres, es decir, haciendo `int N=A.size();` es suficiente).

4. **[PROG-P1]**

   a) **[es-permutacion (25 puntos)]** Una correspondencia es una *"permutacion"* si el conjunto de los elementos del dominio (las claves) es igual al del contradominio (los valores). De esta manera se puede interpretar a la correspondencia como una operación de permutación de las claves (de ahí su nombre). Por ejemplo `M={1->5,3->7,9->9,7->1,5->3}` es una permutación ya que tanto las claves como los valores son el conjunto `{1,3,5,7,9}`. Para determinar si una correspondencia es una permutación (y sin usar contenedores de tipo *"conjunto"*, que serán vistos más adelante), se puede utilizar el siguiente procedimiento. Construir una correspondencia `M2` que mapea los valores del contradominio a las claves, es decir, para cada par de asignación `(k->v)` tal que `M[k] = v`, entonces se debe asignar el par `(v->k)` en `M2`. Notar que si la correspondencia no es biunívoca, es decir si varias claves son asignadas a un mismo valor, entonces a `v` se le asignará alguna de esas claves. Cual de ellas es asignada dependerá del orden en que se recorren las asignaciones de `M`. Notar que si `M` es biunívoca, entonces `M2` es la inversa de `M`. Entonces, si la composición de `M` con `M2` es la identidad es decir para cada `k` en las claves de `M`, si `M2[M[k]] = k`, entonces `M` es una permutación. Consigna: escribir una función predicado `bool es_permutacion(map<int,int> &M)` que retorna `true` si `M` es una permutación y `false` si no lo es.
   b) **[list-sort (25 puntos)]** Escribir una función `void sort(list<int> &L);`, que ordena los elementos de `L` de menor a mayor. Para ello utilizar el siguiente algoritmo simple, utilizando una lista auxiliar `L2`: ir tomando el menor elemento de `L`, eliminarlo de `L` e insertarlo al final de `L2` hasta que `L` este vacía. Cual es el tiempo de ejecución en función de la cantidad de elementos `n` de `L`?

5. **[PROG-P2]**

   a) **[is-full (20pt)]** Recordemos que un árbol binario (AB) es *"lleno"* si todos sus *niveles están completos (atención, no confundir con árbol completo)*. Se puede definir en forma recursiva de la siguiente manera

   - El árbol vacío es lleno
   - Un árbol no vacío es lleno si sus dos hijos son llenos y tienen la misma altura.

   b) **[max-sons-count-node (20pt)]**
   Escribir una función `tree<int>::iterator max_sons_count_node(tree<int> &T, int &nsonmax);` que, dado un árbol ordenado orientado `T` retorna la posición (o nodo o iterator) del nodo cuya cantidad de hijos es la máxima y en `nsonmax` ese número máximo de hijos. Por ejemplo, si `T=(3 (4 2 10) (6 7 (8 9 5)) (11 12 13 14 15 16))`, entonces `max_sons_count_node` debe retornar la posición (o iterator) del nodo `11` y `nsonmax=5`. Si hay más de un nodo con la cantidad máxima de hijos, entonces puede retornar cualquiera de ellos.
   Sugerencia: dado un nodo debe ir recorriendo los hijos contándolos y al mismo tiempo ir reteniendo un iterator al nodo con mayor cantidad de hijos.

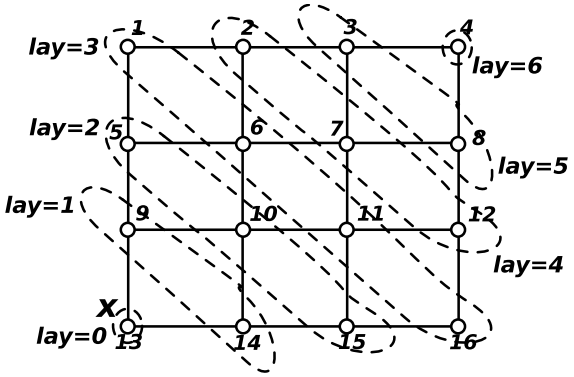
6. **[PROG-P3]**

   a) **[is-simple-graph (15 puntos)]**. Escribir una función `bool is_simple_graph(map<int,set<int> > &G)` que determinar si `G` es un grafo simple, es decir que no contiene blucles (`j->j`) y es no orientado (`i->j` $\iff$ `j->i`).

   b) **[mklayers (30 puntos)]** Escribir una función
   `void mklayers(vector<set<int> > &G, int x, vector<set<int> > &layers)` que dado un grafo `G` y un vértice de partida `x` determina la estructuras de capas de vecinos `layers` de `x` definida de la siguiente manera:

   - La capa 0 es el conjunto `layers[0]={x}` .
   - La capa 1 es el conjunto de los vecinos de `x`.
   - Para `l>1` la capa `l` es el conjunto de los vecinos de los nodos en la capa `l-1` que no están en capas anteriores (0 a `l-1`). Notar que en realidad sólo hace falta verificar que no estén en las capas `l-1` y `l-2`.

   Puede demostrarse que los vértices en la capa `l` son los que están a distancia `l` de `x`. Por ejemplo, dado el grafo de la figura, y partiendo del nodo `x=0` las capas son `layers[0]={0}`, `layers[1]={9,14}`, `layers[2]={5,10,15}`, `layers[3]={1,6,11,16}`, `layers[4]={2,7,12}`, `layers[5]={3,8}`, `layers[6]={4}`.



7. **[OPER-P2]**

   a) **[rec-arbol (10pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

   - `ORD_PRE` $= \{W, X, Y, A, B, C, D, F, G, H\}$;
   - `ORD_POST` $= \{X, Y, B, F, G, H, D, C, A, W\}$.

b) **[huffman (10pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `SUDAFRICA` $P(S) = 0.15, P(U) = 0.15, P(D) = 0.05, P(A) = 0.05, P(F) = 0.20, P(R) = 0.10, P(I) = 0.20, P(C) = 0.10$. Calcular la longitud promedio del código obtenido. Justificar si cumple o no la condición de prefijos.

8. **[OPER-P3]**

a) **[abb (5 pts)]** Dados los enteros $\{13, 7, 20, 2, 3, 10, 5, 4, 1, 12\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 13, 5 y 2 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $3, 16, 26, 9, 8, 36, 18, 5, 28$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x \bmod 9$ y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{1, 5, 8, 2, 3, 13, 10\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
d) **[quick-sort (5 pts)]** Dados los enteros $\{5, 9, 4, 1, 5, 10, 8, 3, 3, 2, 11, 6\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

9. **[PREG-P1]**

a) Considere la función:

```
bool is_mapped(map<int,int> &M,int key,int val) {
  return /* Esta key -> val en M? ... */; }
```

que debe retornar `true` si el par de asignación `(key,val)` está en la correspondencia `M`. ¿Cuál es la expresión correcta que refina el seudocódigo?
b) ¿Cuál es el tiempo de ejecución de la función `find(key)` para correspondencias implementadas por vectores ordenados en función de n, el número de asignaciones en la correspondencia?
c) Dadas las funciones
- $T_1(n) = 3n^3 + 2n! + \log n$,
- $T_2(n) = 3 \cdot 2^3 + n^2 + n^{1.5}$,
- $T_3(n) = 5! + 6 \cdot 2^n + 20 \cdot n^2$ y
- $T_4(n) = 2^{10} + 20n + \log_2 40$

ordenarlas de menor a mayor orden de tiempo de ejecución.
d) Discuta si los algoritmos de búsqueda **exhaustiva** y **heurístico** para la coloración de grafos dan la solución **óptima** al problema. Señale la **complejidad algorítmica** de los mismos. ¿Cuál de los dos elegiría para un grafo con 100 vértices y 500 aristas? Justifique.
e) Si la correspondencia `M={(1->2),(5->8)}` y ejecutamos el código `int x= M[5]`. ¿Que ocurre? ¿Que valores toman `x` y `M`? ¿Y si hacemos `x = M[3]`?

10. **[PREG-P2]**

a) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.
b) Defina en forma recursiva el listado en orden previo y el listado en orden posterior de un arbol ordenado orientado con raiz `t` y sub-árboles hijos `h_1,h_2,...,h_n`.
c) Cual es el orden del tiempo de ejecución (en promedio) en función del número de elementos (n) que posee el árbol ordenado orientado implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos
- `insert(p,x)`
- `begin()`

- `lchild()`
- `operador ++ (prefijo o postfijo)`
- `find(x)`
- `erase(p)`
- `*n`

d) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.

e) Para el arbol binario `(1 . (2 (3 5 .) 6))`

- como queda el árbol y que sucede si hacemos
```
btree<int>::iterator p=T.begin();
p=T.erase(p.left());
```
- y como queda el árbol asi hacemos por otro lado
```
btree<int>::iterator p=T.begin(),n;
n=p;
n=n.right();
p=T.splice(p.left(),n);
```

11. **[PREG-P3]**

a) Defina la propiedad de **transitividad** para las relaciones de orden. Si $f(x)$ es una función sobre los reales y defino $a <_f b$ sii $f(a) < f(b)$, es $<_f$ transitiva?

b) Nombre diversas **relaciones de orden** que se pueden usar con los algoritmos de ordenamiento.

c) Discuta el valor de retorno de `insert(x)` para conjuntos.

d) Discuta el **número de intercambios** que requieren los algoritmos de ordenamiento lentos en el peor caso.

e) ¿Cuál es el costo de **inserción exitosa** en tablas de dispersión abiertas?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [2012-12-06]

ATENCIÓN: Recordar que tanto en las clases como en los ejercicios de programación **deben usar la interfaz STL**.

1. **[clases (20pt,min 50 %)]**.

   a) **[stack (6pt)]** Escribir la implementación del TAD pila (clase `stack`) usando listas.
   b) **[oht-erase (7pt)]** Escribir una función `bool oht_erase(vector<list<T>> &V, bool (*equal)(T,T), unsigned int hash(T), T x);` que elimina el elemento `x` de la tabla de dispersión abierta `V`, utilizando la función de hash `hash()` y la relación de equivalencia dada por `equal()`.
   c) **[btree (7pt)]**. Escribir la implementación en C++ del `TAD ARBOL BINARIO` (clase `btree`). Las funciones a implementar son `erase(p)` y `find(x)`.

2. **[programacion (40pt,min 50 %)]**.

   a) **[are-inverses (10pt)]** Dos correspondencias `M1` y `M2` son **inversas** una de la otra si tienen el mismo número de asignaciones y para cada par de asignación `x->y` en `M1` existe el par `y->x` en `M2`.
   Consigna: Escribir una función **predicado** `bool areinverse(map<int,int> &M1,map<int,int> &M2);` que determina si las correspondencias `M1`, `M2` son una la inversa de la otra o no.
   b) **[intersect-map (30pt)]** Implemente una función `void intersect_map(const map< string, list<int> > &A, const map< string, list<int> > &B, map< string, list<int> > &C)` que a partir de los diccionarios `A` y `B` construya un diccionario `C` de acuerdo a las siguientes reglas:
      - Si una clave `key` esta contenida en A o B pero **no en ambos**, entonces C **no debe contener** dicha clave.
      - Si una clave `key` esta contenida en A y B a la vez, entonces C debe contener dicha clave y su valor asociado debe ser una lista que contenga todos los **elementos comunes y sin repetición** de las listas asociadas a `key` en A y B (es decir, la intersección como conjunto, $C[key] \leftarrow A[key] \bigcap B[key]$).

      Por ejemplo, dados:

      ```
      A = { 'XX' : [3,3,1,2,2,7],     B = { 'YY' : [3,3,4,5,8,1],
            'YY' : [7,1,5,5,4,1] }          'ZZ' : [1,1,9]       }
      ```

      se debe obtener:

      ```
      C = { 'YY' : [1,5,4]}
      ```

      Sugerencia: implementar una función auxiliar `bool contains(list<int>&L, int x)` que devuelve `true` si la lista `L` contiene el valor `x`; y `false` en caso contrario. Utilize esta funciń como ayuda para contruir la lista de elementos comunes y sin repetición $C[key] \leftarrow A[key] \bigcap B[key]$
   c) **[map2count (15 puntos)]**. Escribir una función `void map2count(tree<int> &A,tree<int> &B);` que construye un árbol `B` a partir de otro dado `A` tal que `B` tiene la misma estructura que `A`, y el valor en el nodo `nB` de `B` es la cantidad de hojas en el subárbol del nodo correspondiente `nA` en `A`. Por ejemplo si `A=(5 1 (3 0 2 (6 (7 8 9))))`, entonces debe ser `B=(5 1 (4 1 1 (2 (2 1 1))))`.

3. **[operativos (20pt,min 50 %)]**.

   a) **[particionar (5pt)]**. Considerando el árbol `(z (a (d x y)) (b p q r))` decir cuál son los nodos `descendientes(b)`, `antecesores(b)`, `izquierda(b)` y `derecha(b)`.
   b) **[heap-sort (5pt)]**. Dados los enteros $\{8, 3, 11, 5, 7, 8, 5, 7, 4, 6\}$. ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) **antes** y **después** de **cada** inserción/supresión.
   c) **[huffman (5pt)]**. Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario (para **todos** los caracteres) y encodar la palabra "`GANGNAM STYLE`":
   $P(G) = 0.24$, $P(N) = 0.12$, $P(A) = 0.10$, $P(M) = 0.10$, $P(S) = 0.10$, $P(T) = 0.14$, $P(Y) = 0.08$, $P(L) = 0.06$, $P(E) = 0.06$. Indicar el número de nivel de cada caracter y calcular la longitud promedio del código obtenido.

d) **[rec-arbol (5pt)]**. Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{Z, A, D, X, Y, B, P, Q, R\}$,
- `ORD_POST` $=\{X, Y, D, A, P, Q, R, B, Z\}$.

4. **[preguntas (20pt,min 60 %)]**.

a) ¿Qué ocurre si ejecutamos el siguiente código? `S` es una pila de enteros que puede estar vacía o no.

```
S.push(3); S.push(5); S.top(); S.top(); x=S.top();
```

¿Puede dar un error porque `S` se queda vacía? ¿Que valor toma `x`?

b) Discuta ventajas y desventajas de usar contenedores lineales **ordenados** o **desordenados** para representar correspondencias.

c) Explique porqué el método de Huffman cumple con la *"condición de prefijos"*.

d) Discuta el valor de retorno de `insert(x)` para **conjuntos**.

e) ¿Pueden ser los siguientes **predicados binarios** una **relación de orden**?

```
bool comp1(int x,int y) {
  return true;
}
bool comp2(int x,int y) {
  return false;
}
```

Algoritmos y Estructuras de Datos.
1er Parcial. [2012-09-13]

1. Instrucciones

- El examen consiste en escriban las dos funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **debe compilar y correr correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Junto tendrán que programar algunos casos de prueba, que les daremos como ejemplo, y la salida debe corresponder a la indicada. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.

El ejercicio consiste de dos partes.

1.1. Problema del rango de suma nula en una lista

Escribir una función `bool nullsum(list<int> &L, list<int> &L2);` que dada una lista `L`, determina si hay un **rango de iteradores** `[p,q)`) tal que su suma sea 0. En caso afirmativo debe retornar además el rango (uno de ellos, porque puede no ser único) por `L2`. En caso negativo `L2` debe quedar vacía. El algoritmo debe ser $O(n^2)$. Por ejemplo si `L=(-10 14 -2 7 -19 -3 2 17 -8 8)` entonces debe retornar `true` y `L2=(14 -2 7 -19)`, o también `L2=(-8 8)`. Si `L=(1,3,-5)` entonces debe retornar `false` y `L2=()`.

1.2. Filtrar con nullsum las entradas de un map

Escribir una funcion `void nullsum_filt(map<int,list<int>> &M, map<int,list<int>> &M2);` que extrae de todos los pares de asignación `(k,L)` de `M` aquellos para los cuales `L` tiene un rango de suma zero, e inserta en `M2` los pares `(k,L')` donde `L'` es el rango de `L` que tiene suma 0.
Por ejemplo si

```
M[0] -> (-7 -4 7 5 3 5 -4 2 -1 -9)
M[1] -> (-8 -3 0 9 -7 -4 -10 -4 2 6)
M[2] -> (1 -2 -3 -1 -8 0 -8 -7 -3 5)
M[3] -> (-1 -8 -8 8 -1 -3 3 6 1 -8)
M[4] -> (1 3 5 -2)
M[5] -> (3 -4 1 -10 6 3 -8 0 6 -9)
M[6] -> (-5 -5 -6 -3 6 -5 -4 -1 3 7)
M[7] -> (-6 5 -8 -5 4 -3 4 -6 -7 0)
M[8] -> (1 5 4)
M[9] -> (2 -10 6 -2 9 2 -4 -4 4 9)
```

entonces debe retornar por `M2` (recordad que como los rangos de suma nula pueden no ser únicos el valor (contradominio) de la correspondencia puede

```
M2[1] -> (0)
M2[2] -> (0)
M2[3] -> (-8 8)
M2[5] -> (3 -4 1)
M2[6] -> (-5 -4 -1 3 7)
M2[7] -> (-5 4 -3 4)
M2[9] -> (-4 4)
```

2. SOLUCION

Se adjunta aquí la solución al problema:

- Decargar `nullsum.cpp`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [2012-09-13]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 50 % en clases (Ej 1), 40 % en programación (Ej 2), y un 60 % sobre las preguntas de teoría (Ej 3).

[Ej. 1] **[clases (30pt)]**

a) **[lista (20pt)]** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por celdas (simple o doblemente) enlazadas por punteros **ó** cursores. (**Indicar claramente qué implementación elige.**) Los métodos a implementar son `insert(p,x)`, `erase(p)`, `iterator::operator++(int)` (postfijo), `iterator::operator++()` (prefijo). **Atención:** deben declarar los datos miembros de las clases a declarar o implementar.

b) **[pila-cola (10pt)]** Escribir la implementación en `C++` de los métodos `push`, `pop`, `front`, y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.

[Ej. 2] **[Programación (total = 50pt)]** Recordar que en los ejercicios de programación **deben usar la interfaz STL**.

a) **[is-sublist (20 puntos)]**. Escribir una función `bool is_sublist(list<int> &L1, list<int> &L2, list<list<int>::iterator> &iters)` que dadas dos listas listas de enteros `L1,L2` determina si la lista `L2` es una sublista de `L1`, es decir que `L2` se puede obtener a partir de `L1` sólo eliminando algunos de sus elementos. Por ejemplo si `L1=(3,2,6,5,4,1,8)` y `L2=(3,6,5,1)`, entonces `is_sublist(L1,L2,iters)` debe retornar `true`, mientras que si `L2=(3,6,1,5)` entonces debe retornar `false` ya que los elementos 1 y 5 están invertidos con respecto a su posición en `L1`. Adicionalmente, en el caso que SI sea sublista debe retornar por `iters` las **posiciones** (iteradores) que corresponden a los elementos de `L1` qye están en `L2`. En el ejemplo `L1=(3,2,6,5,4,1,8)` y `L2=(3,6,5,1)` debe retornar `true` y por `iters` 4 posiciones en la lista `L1` correspondientes a los elementos 3,6,5,1.
RESTRICCIONES: el algoritmo debe ser $O(n)$
AYUDA: Para cada elemento de `L1` verificar si se encuentra en `L2` siempre partiendo desde la última posición en que se encontró el elemento anterior.

b) **[son-iguales (10 puntos)]**. Escribir una función `bool son_iguales(stack<int> &S, queue<int> &Q);` que determina si el contenido de los contenedores `S` y `Q` son iguales (los elementos son idénticos). Después de terminar, los contenedores deben quedar en el mismo estado. Se pueden utilizar contenedores auxiliares.

c) **[submap (20 puntos)]**. Dada una correspondencia `map<string,string> M`, y una clave `k0`, se desea extraer todos los pares de asignación correspondientes a claves **conectadas** con `k0`. Es decir, a partir de `k0` se puede generar una secuencia `L=(k0,k1,....)` de la siguiente forma `k1 = M(k0), k2=M(k1)`, ... hasta que o bien `k_j` no pertenece a las claves de `M` o bien `k_j` ya fue visitado. Se quiere extraer de la correspondencia aquellos pares de asignación cuyas claves están en la secuencia `L`. Por ejemplo si `M1={(a,e),(c,a),(d,h), (e,g),(f,c),(g,d)}` y `k0=a` entonces debe dejar `M2={(a,e),(d,h),(e,g),(g,d)}`
CONSIGNA: Escribir una funcion `void submap(map<string,string> &M1, map<string,string> &M2,string k0);` que deja en `M2` los pares de asignación de aquellas claves que están en la secuencia que se genera por el algoritmo previo.

[Ej. 3] **[Preguntas (total = 20pt, 4pt por pregunta)]**

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = 10n^3 + 2n! + 3\log_2 n + 4,$$
$$T_2 = 23! + 5n^4 + 2^n,$$
$$T_3 = n^3 + 3^n + 10,$$
$$T_4 = 2 + 5\log n,$$
$$T_5 = n + 2n^2 + 5n^3,$$

b) Defina que entiende por un **algoritmo de búsqueda exhaustiva** y un **algoritmo ávido**. Explique como se aplicarían estas estrategias a un problema como el de coloración de grafos o el del agente viajero (TSP).

c) ¿Como se define la **notación asintótica** $T(n) = O(f(n))$? ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si siempre es $(n+1)^2 > n^2$?

d) Si se implementa una **pila** en términos de **listas simplemente enlazadas**: ¿Dónde ubicaría el **tope** de la pila? ¿En el comienzo o en el fin de la lista? ¿Porqué?

e) Discuta como se calcula el tiempo de ejecución de un bloque `if` en términos de los tiempos de ejecución del condicional T_{cond} y los bloques `true` y `false`: T_{true} y T_{false}.

```
if (COND) {
  BODY_TRUE
} else {
  BODY_FALSE
}
```

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [2012-10-25]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] **[clases (20pt)] Insistimos: deben usar la interfaz STL ("AVANZADA").**

a) **[AOO (10pt)]** Para el TAD Arbol Ordenado Orientado (AOO):
- Declarar las clases `tree`, `cell`, `iterator`, con su correspondiente anidamiento, incluyendo las declaraciones de datos miembros.
- Implementar los métodos
  - `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x);`
  - `tree<T>::iterator tree<T>::begin();`

b) **[AB (10pt)]** Para el TAD Arbol Binario (AB):
- Declarar las clases `btree`, `cell`, `iterator`, con su correspondiente anidamiento, incluyendo las declaraciones de datos miembros.
- Implementar los métodos
  - `btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`,
  - `btree<T>::iterator btree<T>::iterator::right();`.

[Ej. 2] **[Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.**
Atención: Hay 3 ejercicios cuya suma es **60**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,n1+n2+n3)`).

a) **[bin2ord (20pt)]** Escribir una funcion `void bin2ord(btree<int> &B, tree<int> &T,int LAMBDA);` que dado un árbol binario (AB) `B` de enteros **positivos** lo convierte a un árbol ordenado orientado (AOO) `T` con la siguiente convención. En caso de que uno de los nodos del AB tenga uno solo de los hijos reemplazamos el valor por un valor especial `LAMBDA` (en este caso puede ser `LAMBDA=-1`, ya que todos los valores nodales son `>0`). Por ejemplo, si `B=(1 (2 . (3 . 4)) .)`, y `LAMBDA=-1` entonces debe dar `T=(1 (2 -1 (3 -1 4)) -1)`. Notar que en notación Lisp esto equivale a reemplazar los puntos por `LAMBDA`.

b) **[ord2bin (20pt)]** Escribir la funcion inversa `void ord2bin(tree<int> &T, btree<int> &B, int LAMBDA);` que dado un AOO (se asume que fue generado por `bin2ord`) lo convierte de nuevo a AB. (Debería ser siempre `B==ord2bin(bin2ord(B))`).

c) **[lesser (20pt)]** Se define una **relación de orden** entre AOO de enteros de la siguiente forma: `A<B` si `(na,c0a,c1a,c2a...)<(nb,c0b,c1b,c2b...)`, donde `na` es la raíz de `A`, `h0a` el **subárbol** del primer hijo de `A` y así siguiendo. En la expresión anterior se toma el orden lexicográfico para listas y se asume que en caso de tener longitudes diferentes se completa con $-\infty$.
Consigna: Escribir un predicado binario `bool lesser(tree<int> &A,tree<int> &B);` que determina si `A<B` en el sentido indicado previamente.
Nota: puede demostrarse que efectivamente este predicado binario es una **relación de orden total**.

[Ej. 3] **[operativos (total 20pt)]**

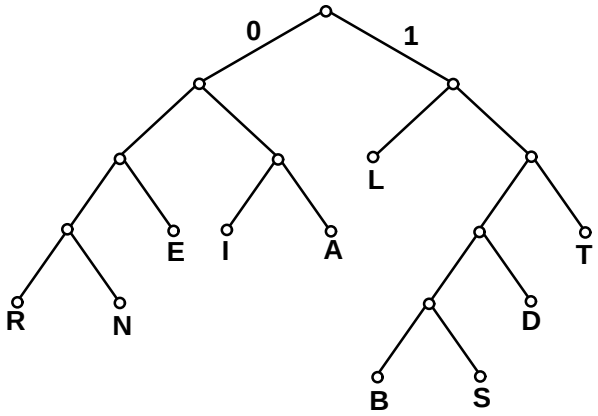
a) **[rec-arbol (8pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{W, X, Y, Z, T, R, S, Q\}$,
- `ORD_POST` $=\{X, Z, T, R, Q, S, Y, W\}$,

b) **[huffman (6pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `FRAGATA`
$P(G) = 0.10, P(F) = 0.10, P(R) = 0.10, P(T) = 0.30, P(A) = 0.05, P(S) = 0.35$. Calcular la longitud promedio del código obtenido.

c) **[hf-decode (6pt)]**

Utilizando el código de la derecha desencodar el mensaje `10010110000010000111010111101`



[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Dado el árbol ordenado orientado (AOO) `(W X Y (A B (C (D F G H))))` determinar cuales son los nodos **antecesores, descendientes, izquierda** y **derecha** del nodo `A`. ¿Son **disjuntos**? Justifique.

b) Si queremos generar un código binario de igual longitud para un conjunto de 26 caracteres. ¿Cuantos bits tendrá, como **mínimo**, cada caracter?

c) ¿Como se define la **altura** de un nodo en un árbol? ¿Cuál es la altura de los nodos `A`, `B`, y `C` en `(W X Y (A B (C (D F G H))))`?

d) ¿Cómo se define la clase **iterator** para árboles binarios? Enumere sus campos dato y discuta la utilidad de cada uno de ellos. De ejemplos.

e) ¿Cómo se define en forma recursiva el **orden posterior** para un árbol ordenado orientado (AOO)?

mstorti@galileo/aed-2.0.4-158-g13abc45/Thu Oct 25 17:48:34 2012 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
3er Parcial. [2012-11-22]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)] Insistimos: deben usar la interfaz STL.

a) **[set-vbit (6pt)]**
- Escribir una función `void vbset_difference(vector<bool> &A,vector<bool> &B,vector<bool> &C);` que implementa la operación binaria diferencia de conjuntos ($C = A - B$), para la representación por vectores de bits. Asuma que `A`, `B`, `C` tienen la misma longitud.
- Escribir una función `void vbset_clear(vector<bool> &A);` que elimina todos los elementos de un conjunto `A` representado con vectores de bits.

b) **[set-oht (7pt)]** Implementar una función `pair<int,bool> oht_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T),bool (*equal)(T,T),T x)` que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y la relación de equivalencia `equal` retornando un par de `int` (que indica la cubeta) y `bool` que indica si la inserción fue exitosa.

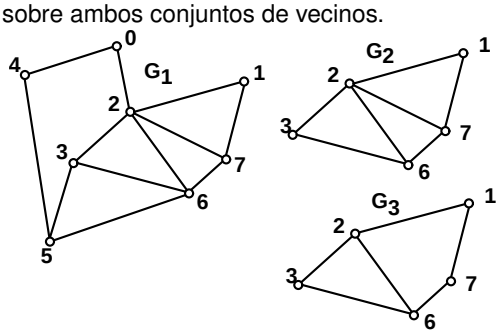
c) **[set-abb (7pt)]**
- Escribir la función `btree<int>::iterator abb_find(btree<int> &T,int x);` que devuelve el iterator en el ABB `T`, donde se encuentra el elemento `x`, o caso contrario `T.end()`. **Restricción:** La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.
- Escribir la función `pair<int,int> minmax(btree<int> &T);` que devuelve el mínimo y el máximo de el ABB `T`. Si el árbol está vacío debe devolver las constantes `(INT_MAX,INT_MIN)`. **Restricción:** La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.

[Ej. 2] [Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.

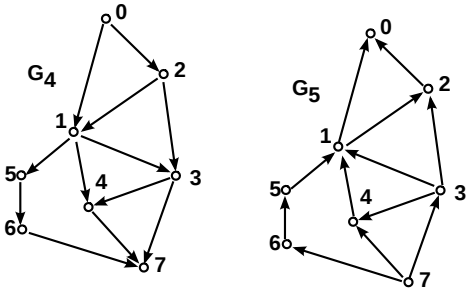
Atención: Hay 4 ejercicios cuya suma es **60**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,a+b+c+d)`).
Nota: En los ejercicios con grafos siguientes, los grafos no tienen bucles y se representan como un `map<int, set<int> >G`, donde `G[j]` es el conjunto de vértices adyacentes (esto es, conectados por una arista) con el vértice `j`. A continuación el tipo `graph_t` está definido como `typedef map<int, set<int> > graph_t`.

a) **[is-subgraph (20pt)]** Dados dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ determinar si G_2 es un subgrafo que se obtiene dejando un cierto conjunto de vértices $V_2 \subseteq V_1$ y **todas** las aristas de E_1 que conectan nodos de V_2.
Consigna: Escribir una función `bool issubgraph(graph_t &G1,graph_t &G2);` que realiza la tarea indicada.
Si lo aplicamos a los grafos de abajo entonces debemos tener `issubgraph(G1,G2) -> T` y `issubgraph(G1,G3) -> F` ya que la arista $(2, 7)$ está en `G1` pero no en `G3`.
Ayuda: Primero chequear que los vértices de `G2` están incluidos en `G1`. Después chequear que para vértice en `G2` sus vecinos son exactamente la intersección de sus vecinos en `G1` y el conjunto de vértices de `G2`. Esta

última operación se puede hacer o bien utilizando las operaciones binarias de conjuntos (con conjuntos auxiliares) o bien iterando sobre ambos conjuntos de vecinos.



b) **[dag-descendants (20pt)]** Dados un **Grafo Dirigido Acíclico (DAG)** $G = (V, E)$ y un subconjunto de vértices $W \subseteq V$, determinar el conjunto $D \subseteq V$ de **descendientes** de W, es decir $d \in D$ si y sólo si existe un camino que va de algún nodo $w \in W$ a d. En el grafo de abajo los descendientes de $W = \{5, 3\}$ en el grafo G_4 son $D = \{5, 3, 4, 6, 7\}$.
Consigna: Escribir una función `void dag_desc(graph_t &G,set<int> &W, set<int> &D);` que realiza la tarea indicada.



c) **[reverse-dag (10pt)]** Dados un **Grafo Dirigido Acíclico (DAG)** $G = (V, E)$ obtener el DAG $G' = (V, E')$ tal que si $(v, w) \in E$ entonces $(w, v) \in E'$. (Es decir, equivale a invertir el sentido de las flechas en el dibujo). Por ejemplo en los grafos de arriba aplicamos a G_4 debe dar G_5 y vicersa. **Consigna:** Escribir una función `void reverse_dag(graph_t &Gin,graph_t &Gout);` que realiza la tarea indicada.

d) **[is-mapped (10pt)]** Dados dos conjuntos `set<int> A,B` y una función **biyectiva** `int f(int)` determinar si es `B=f(A)` es decir **todos** los elementos de `B` se obtienen como imagen de `A` aplicando `f`.
Consigna: Escribir una función `bool ismapped(set<int> &A,set<int> &B,int (*f)(int));`
Restricción: el algoritmo debe ser **in-place** (no usar estructuras auxiliares) y **no-destructivo**, de hecho, no debe modificar ni `A` ni `B`.
Ayuda: Como `f()` es biyectiva, basta con determinar si el tamaño de ambos conjuntos es el mismo, y aplicar la función a los elementos de `A` y chequear que estén en `B`.

[Ej. 3] [operativos (total 20pt)]

a) **[abb (5 pts)]** Dados los enteros $\{12, 6, 19, 1, 2, 9, 4, 3, 2, 11\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 12, 6 y 1 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $\{0, 16, 26, 9, 8, 36, 16, 18, 5\}$ en una **tabla de dispersión cerrada** con $B = 8$ cubetas, con función de dispersión `h(x) = 2*x+3` y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{0, 4, 7, 1, 2, 12, 9\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
d) **[quick-sort (5 pts)]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Discuta la **estabilidad** del algoritmo de ordenamiento de **listas por fusión** (`merge-sort`). ¿Es estable? ¿Bajo que condiciones?

b) ¿Cuál es el tiempo de ejecución del algoritmo de **ordenamiento rápido** (`quick-sort`), en el caso promedio y en el peor caso? ¿Cuando se produce el peor caso?

c) ¿Cuáles son las condiciones que debe satisfacer un predicado binario para ser una **relación de orden débil**?

d) Se quiere representar subconjuntos del conjunto universal de los cubos perfectos en $[-1000, 1000]$, o sea

$$\begin{aligned} U &= \{-1000, -729, -512, \ldots, -27, -8, 0, 8, 27, \ldots, 512, 729, 1000\}, \\ &= \{j \in \mathbb{Z} \,/\, \exists\, k \in \mathbb{Z} \text{ con } j = k^3\}, \end{aligned} \tag{1}$$

como un conjunto con **vectores de bits**.

1) ¿Cuál es el tamaño del conjunto universal?
2) Escribir las funciones `int indx(int)` y `int element(int)` correspondientes, que mapean el espacio de elementos del conjunto universal a índices y viceversa.

Ayuda: asumir la existencia de una función `int cbrt(int)` que calcula la raíz cúbica entera de un número entero (si es un cubo perfecto). Si el argumento no es un cubo perfecto, retorna un valor indefinido.
Nota: al escribir la función `index()` se debe chequear que el argumento sea un cubo perfecto. Es decir, si no es un cubo perfecto en el rango indicado debe dar un error (llamando a `abort()` o lanzando una excepción).

e) ¿Cuál es el resultado de aplicar la función `l=particiona(w,j,k,v)` ? (**Nota:** No se pide el algoritmo, sino cuál es el efecto de aplicar tal función, independientemende de como se programe). Explique los argumentos de la función. ¿Cuál debe ser el tiempo de ejecución para `particiona()` si queremos que `quick_sort()` sea un algoritmo **rápido**?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2012-11-29]

ATENCIÓN (1): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases (Ej 1),
- **50 %** en programación (Ej 2),
- **50 %** en operativos (Ej 3) y
- **60 %** sobre las preguntas de teoría (Ej 4).

ATENCIÓN (2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [CLASES-P1 (30pt)]

a) Escribir la implementación en ***C++*** del TAD lista (clase ***list***) implementado por punteros **ó** cursores. Los métodos a implementar son ***insert(p,x)***, ***erase(p)***, ***next()***/***iterator::operator++(int)***, ***list()***, ***begin()***, ***end()***.

b) Escribir la implementación en ***C++*** de los métodos ***push***, ***pop***,***front*** y ***top*** de los TAD pila y cola (clases ***stack*** y ***queue***), según corresponda.

[Ej. 2] [CLASES-P2 (20pt)]

a) **AOO:** declarar las clases ***tree***, ***cell***, ***iterator***, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)

b) **AB:** declarar las clases ***btree***, ***cell***, ***iterator***, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
btree<T>::iterator btree<T>::erase(btree<T>::iterator n);

[Ej. 3] [CLASES-P3 (20pt)]

a) Implementar ***bool bst_erase(btree<T> t,T x,bool (*less)(T,T))*** que elimina el elemento ***x*** del ABB T, retornando true si la eliminación fue exitosa, y false caso contrario.

b) Implementar una función ***pair<int,bool> oht_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T),bool (*equal)(T,T),T x)*** que inserta el elemento ***x*** en la tabla de dispersión abierta ***table*** utilizando la función de dispersión ***hashfunc*** y la relación de equivalencia ***equal*** retornando un par de ***int*** (que indica la cubeta) y ***bool*** que indica si la inserción fue exitosa.

c) Implementar una función ***void vecbit_diffsym(vector<bool>&, A, vector<bool>& B, vector<bool>& C);*** que devuelve la **diferencia simétrica** definida como $C = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$ con ***A,B,C*** vectores de bits que representan conjuntos de un rango contiguo de enteros ***[0, N)***, donde ***N*** es el tamaño de los vectores ***A,B,C*** (asumir el mismo tamaño para los tres, es decir, haciendo ***int N=A.size();*** es suficiente).

[Ej. 4] [PROG-P1 (50pt)]

a) **[ascendente (25pt)]** En ciertas aplicaciones interesa detectar las corridas ascendentes en una lista de números $L = [\, a_1\ a_2\ ,\ ...,\ a_n\,]$, donde cada corrida ascendente es una sublista de números consecutivos a_i, a_{i+1}, ..., a_{i+k} de modo tal que $a_j \leq a_{j+1}$ para $j = i, ..., i+k-1$, y es máxima en el sentido que si $i > 0$ entonces $a_{i-1} > a_i$, y si $i + k < n$ entonces $a_{i+k} > a_{i+k+1}$. Por ejemplo, si $n = 10$ y la lista es L = [0,5,5,9,4,3,9,6,5,5,2], entonces hay $h = 6$ corridas ascendentes, a saber, [0,5,5,9], [4], [3,9], [6], [5,5] y [2]. Usando las operaciones del TAD lista, escribir una función
int ascendente(list<t> &l,vector< list<t> > &v) en la cual, dada una lista de enteros ***l***, almacene cada carrera ascendente como una componente del vector de listas ***vector<list <t> > &v***, devolviendo además el número h de carreras ascendentes halladas.

b) **[concatmap (25pt)]** *Consigna:* Escribir una función
`void concat_map(map<int,list<int> >& M, list<int>& L);` tal que reemplaza los elementos de ***`L`*** por su imagen en ***`M`***. Si un elemento de ***`L`*** no es clave de ***`M`*** entonces se asume que su imagen es la lista vacía.
Por ejemplo: Si
`M={5->(3,2,5),2->(4,1),7->(2,1,1)}` y ***`L=(1,5,7,2,3)`***, entonces debe quedar
`L=(3,2,5,2,1,1,4,1)`.
Restricciones: El programa no debe usar contenedores auxiliares.

[Ej. 5] [PROG-P2 (40pt)]

a) **[ord-nodo (20pt)]**. Escribir una función predicado ***`bool ordnodo(tree<int> &A);`*** que verifica si cada secuencia de hermanos del subárbol del nodo ***`n`*** (perteneciente al árbol ordenado orientado ***`A`***) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol ***`(3 5 (6 1 3) (7 4 5))`*** debería retornar ***`true`***, mientras que para ***`(3 9 (6 1 3) (7 4 2))`*** debería retornar ***`false`***, ya que las secuencias de hermanos ***`(9 6 7)`*** y ***`(4 2)`*** NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

b) **[depth-if (20pt)]**
Dado un AB ***`T`*** encontrar, la profundidad máxima de los nodos que satisfacen un cierto predicado (*profundidad condicionada*). Por ejemplo, si ***`T=(6 (7 9 (3 1)) 2)`***, entonces ***`depth_if(T,even)`*** debe retornar 1, ya que el nodo par de mayor profundidad es 2, mientras que ***`depth_if(T,odd)`*** debe retornar 3, ya que el nodo impar a máxima profundidad es 1.
Consigna: Escribir una función ***`int depth_if(btree<int> &T,bool (*pred)(int));`*** que realiza la tarea indicada.
Ayuda: La función recursiva auxiliar debe retornar la máxima profundidad de un nodo que satisface el predicado o -1 si el árbol está vacío o ningún nodo satisface el predicado. Entonces, dadas las profundidades condicionadas d_l, d_r de los hijos la profundidad se puede expresar en forma recursiva como

$$\mathrm{depth}(n) = \begin{cases} -1, & \text{si } n \text{ es } \Lambda, \\ \max(d_r, d_l) + 1, & \text{si } \max(d_r, d_l) \geq 0, \\ 0, & \text{si } n \text{ satisface el predicado,} \\ -1, & \text{si } n \text{ no satisface el predicado.} \end{cases}$$

[Ej. 6] [PROG-P3 (40pt)]

a) **[is-hamilt (20pt)]** Dado un grafo ***`map<int, set<int> >G`*** y una lista de vértices ***`list<int> L`*** determinar si ***`L`*** es un **camino hamiltoniano** en ***`G`***.
Ayuda: Mantener un ***`set<int> visited`*** con todos los vértices visitados. Para dos enteros ***`x`***, ***`y`*** sucesivos en la lista ***`L`***, verificar si son vértices de ***`G`*** y si son adyacentes. Verificar que el nuevo vértice ***`y`*** no fuera visitado previamente e insertarlo en ***`visited`***. Al final, chequear que todos los vértices fueron visitados.
Nota: Recordamos que un camino Hamiltoniano en un grafo es un camino en el mismo que pasa por todos los nodos sin repetir ninguno. Por ejemplo en el grafo de arriba el camino ***`{0,1,2,3,4,5,6}`*** es Hamiltoniano.

b) **[en-todos (20pt)]**
Escribir una función predicado ***`bool en_todos(vector< set<int> > &v);`*** que retorna verdadero si existe al menos un elemento que pertenece a todos los conjuntos ***`v[j]`***. Por ejemplo, si

$$\mathrm{v[0]} = \{0,2,3,4,5\},\ \ \mathrm{v[1]} = \{0,1,5,7\},\ \ \mathrm{v[2]} = \{2,3,5,6,7\} \tag{1}$$

entonces ***`en_todos(v)`*** debe retornar ***`true`*** ya que 5 está en los tres conjuntos. Por el contrario, si

$$\mathrm{v[0]} = \{0,2,3,4,5\},\ \ \mathrm{v[1]} = \{0,1,7\},\ \ \mathrm{v[2]} = \{2,3,5,6,7\} \tag{2}$$

entonces ***`en_todos(v)`*** debe retornar ***`false`***.
Sugerencia: generar el conjunto que es la intersección de todos los ***`v[j]`*** y finalmente verificar si es vacío o no.

[Ej. 7] [OPER-P2 (20pt)]

a) **[rec-arbol (7pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
- ***ORD_PRE***=$\{A, R, B, O, L, I, T, O, \}$,
- ***ORD_POST***=$\{R, O, O, T, I, L, B, A, \}$,

b) **[huffman (7pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra ***BUITRE***
$P(B) = 0.10, P(M) = 0.10, P(U) = 0.10, P(E) = 0.30, P(I) = 0.05, P(T) = 0.3, (R) = 0.05$ Calcular la longitud promedio del código obtenido.

c) **[hf-decode (6pt)]** Utilizando el código del siguiente árbol binario completo ***(F (L (* R N) (* D O)))*** a derecha desencodar el mensaje `10000001000010000`

[Ej. 8] [OPER-P3 (20pt)]

a) **[abb (5 pts)]** Dados los enteros $\{15, 9, 22, 4, 5, 12, 7, 6, 3, 14\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 15, 7 y 4 en ese orden.
b) **[hash-dict (5 pts)]** Insertar los números $0, 13, 23, 6, 5, 33, 15, 2, 25$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x \bmod 9$ y estrategia de redispersión lineal.
c) **[heap-sort (5 pts)]** Dados los enteros $\{10, 13, 3, 2, 8, 5, 1\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.
d) **[quick-sort (5 pts)]** Dados los enteros $\{8, 12, 7, 4, 8, 13, 11, 6, 6, 5, 14, 9\}$ ordenarlos por el método de *"clasficación rápida"* (*"quick-sort"*). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 9] [PREG-P1 (20pt)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$T_1 = 5 \cdot 2^n + 2n^3 + 3\,n!+$$
$$T_2 = 2 \cdot 10^n + \sqrt{3} \cdot n + \log_8 n+$$
$$T_3 = n^2 + 5 \cdot 3^n + 4^{10}$$
$$T_4 = 2.3 \log_8 n + \sqrt{n} + 5\,n^2 + 2\,n^5$$
$$T_5 = 1000 + 3 \log_4 n + 8^2 + 5 \log_{10} n$$

b) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase ***map<>*** implementada con **listas desordenadas** en el caso promedio?
Métodos: ***find(key)***, ***M[key]***, ***erase(key)***, ***erase(p)***, ***begin()***, ***end()***, ***clear()***).

c) ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si en realidad es siempre verdad que $(n+1)^2 > n^2$?
d) Discuta las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas.
e) ¿Qué ventajas o desventajas tendría implementar la clase *pila* en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista?

[Ej. 10] [PREG-P2 (20pt)]

a) Defina que es un **camino** en un árbol. Dado el AOO ***(a b (c e (f g)) d))***, cuáles de los siguientes son caminos?

1) ***(a,c,f,g)***,
2) ***(e,c,f,g)***,
3) ***(e,c,a)***,
4) ***(c,f,g)***.

b) Defina qué es un **AB lleno**. ¿Cuál es el **número de nodos** en el nivel l de un árbol lleno? ¿Cuál es el **número total de nodos** en un árbol lleno de n niveles? ¿Cuantas **hojas** contiene?

c) ¿Es verdad que si dos nodos están en el **mismo nivel** de un árbol, entonces son **hermanos**? ¿Y la recíproca? De ejemplos.

d) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un AB? ¿Y en un AOO? Discuta y de ejemplos.

e) ¿Porqué el AB correspondiente a la **codificación binaria** de una serie de caracteres debe ser un **ABC (AB completo)**?

[Ej. 11] [PREG-P3 (20pt)]

a) Escriba el código para ordenar un vector de enteros ***v*** por **valor absoluto**, es decir escriba la función de comparación correspondiente y la llamada a ***sort()***.
Nota: recordar que la llamada a ***sort()*** es de la forma ***sort(p,q,comp)*** donde ***[p,q)*** es el rango de iteradores a ordenar y ***comp(x,y)*** es la función de comparación.

b) Cual es el tiempo de ejecución en el caso **promedio** para el método ***insert(x)*** de la clase **diccionario** (***hash_set***) implementado por **tablas de dispersión cerradas**, en función de la ***tasa de llenado*** $\alpha = n/B$, para el caso de inserción exitosa y no exitosa.

c) Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**.

d) Se quiere representar el conjunto de enteros múltiplos de 3 entre 30 y 99 (o sea $U = \{30, 33, 36, ..., 99\}$) por **vectores de bits**, escribir las funciones ***indx()*** y ***element()*** correspondientes.

e) Discuta la complejidad algorítmica de las operaciones binarias ***set_union(A,B,C)***, ***set_intersection(A,B,C)***, y ***set_difference(A,B,C)*** para conjuntos implementados por vectores de bits, donde ***A***, ***B***, y ***C*** son subconjuntos de tamaño n_A, n_B, y n_C respectivamente, de un conjunto universal ***U*** de tamaño N.

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
RECUPARATORIO LAB [2012-10-17]

Instrucciones

- El examen consiste en escriban la función descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **debe compilar y correr correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Junto tendrán que programar algunos casos de prueba, que les daremos como ejemplo, y la salida debe corresponder a la indicada. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.

Consigna

Escribir una función `void make_map_odd_even(list<int> &L,map_t &M);` que dada una lista de enteros `L`, construye la correspondencia `M` que mapea cada elemento **impar** de la lista al mayor rango de elementos contigous **pares** que sigue al elemento impar, por ejemplo si `L=(9,10,6,7,6,8,6,10,2,7)`, entonces debe ser `M=[9->(10,6),7->(6,8,6,10,2)]`.

Nota 1: Notar que si bien `7` aparece dos veces en la lista, el rango correspondiente para la última instancia es el rango vacío, por lo tanto queda el primer rango `(6,8,6,10,2)`.

Nota 2: Si hay varios rangos con la longitud mínima, entonces puede queadar cualquiera de ellos.

Restricción: El algoritmo debe ser $O(n)$, donde n es la longitud de la lista.

Ayuda: Se sugiere el siguiente algoritmo, para cada posición `p`, si el elemento es impar recorrer los siguientes elementos pares poniéndolos en una lista `tmp`. Cuando se llega al final o a un elemento impar se asigna la lista `tmp` all valor almacenado en `p` a condición de que no tenga ya un valor asignado, o si su longitud es menor que la de `tmp`.

mstorti@galileo/aed-2.0.4-135-ge732569/Mon Oct 15 15:22:45 2012 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er Parcial. [2013-10-03]

ATENCIÓN(1): Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 4) y 50 % en las restantes secciones.
ATENCIÓN(2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)]

a) **[lista (10pt)].** Escribir la implementación en `C++` del TAD lista (clase `list`) implementado por celdas (simple o doblemente) enlazadas por punteros **ó** cursores. (**Indicar claramente qué implementación elige.**) Los métodos a implementar son `insert(p,x)`, `erase(p)`, `iterator::operator++(int)` (postfijo), `iterator::operator++()` (prefijo).

b) **[AOO (10pt)].** Declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`

[Ej. 2] [Programación (40pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL**.

a) **[nchild (20pt)]**. Escribir una función predicado `bool nchild(tree<int> &T,int n);` que dado un árbol ordenado orientado (AOO) `T` chequea si cada hijo tiene a lo sumo `n` hijos. Por ejemplo si `T=(a (b f g h) (c z (d e)))`. Entonces `nchild(T,3) -> true` ya que el máximo número de hijos es 3 para el nodo `b` mientras que `nchild(T,2) -> false`.

b) **[evencount (20pt)]**. Escribir una función predicado `bool evencount(map<int,list<int>> &M);` que chequea que para cada par `(k,L)` en `M` se verifique que la lista `L` contiene exactamente `k` elementos pares. Por ejemplo, si

```
M1[0] = (5,7,9)
M1[3] = (5,2,3,4,6)
```

entonces `evencount(M1)->true`. Mientras que si

```
M2[2] = (5,7,8)
M2[4] = (5,2,3,4,6,8)
```

debe retornar `evencount(M2)->false` ya que para la clave 2, la lista correspondiente tiene un solo elemento par.
Ayuda: Escribir una función `int evencount(list<int> &L);` que cuenta cuantos pares hay en la lista `L`.

[Ej. 3] [operativos (20pt)]

a) **[rec-arbol (10pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
- `ORD_PRE` $=\{A, Z, W, B, C, D, Q, R, L, F\}$,
- `ORD_POST` $=\{B, C, D, W, L, R, Q, Z, F, A\}$,

b) **[particionar (10pt)]**. Considerando el árbol `(a (b f g h) (c z (d e)))` decir cuál son los nodos `descendientes(z)`, `antecesores(z)`, `izquierda(z)` y `derecha(z)`. (**Nota:** se refiere a antecesores y descendientes **propios**).

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = n + 5\log n,$$
$$T_2 = 10n^3 + 2n! + 5n^4 + 2^n,$$
$$T_3 = 23! + 3\log_2 n + 4,$$
$$T_4 = 2 + 2n^2 + 5n^3,$$
$$T_5 = n^3 + 3^n + 10.$$

b) ¿Es la **pila** un contenedor **FIFO** o **LIFO**? ¿Que significa? Responda la misma pregunta para el TAD **cola**.

c) Enuncie la **regla de la suma** para la notación asintótica $O(\ )$. De ejemplos.

d) ¿Qué ventajas o desventajas tendría implementar la clase *pila* en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista?

e) ¿Cual es la principal ventaja de implementar correspondencias con **vectores ordenados**? ¿Se obtiene la misma ventaja con **listas ordenadas**?

mstorti@galileo/aed-2.0.4-208-gda743fd/Tue Oct 1 10:04:27 2013 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do Parcial. [2013-11-14]

ATENCIÓN(1): Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 4) y 50 % en las restantes secciones.
ATENCIÓN(2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)]

a) **[AB (5pt)]** Par el TAD Arbol Binario (AB): declarar las clases `btree`, `cell`, `iterator`, incluyendo las declaraciones de datos miembros. Implementar el método `btree<T>::iterator btree<T>::find(const T& x);`. Si utiliza alguna función auxiliar impleméntela.

b) **[ABB (5pt)]** Escribir la función `pair<int,int> minmax(btree<int> &T);` que devuelve el mínimo y el máximo de el ABB `T`. Si el árbol está vacío debe devolver las constantes `(INT_MAX,INT_MIN)`.
Restricción: La complejidad algorítmica debe ser $O(l)$ donde l es la profundidad del árbol.

c) **[sort (10pt)]** Escribir una función `void sort(vector<int> &v);` que ordena los elementos de `v` utilizando el operador `<` como función de comparación. Puede elegir cualquier algoritmo de ordenamiento rápido o lento visto en el curso. **Especifique cuál es el algoritmo que está usando.**
Restricción: El algoritmo debe ser *in-place*, es decir no debe usar contenedores auxiliares.

[Ej. 2] [Programación (40pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL**.
Atención: Notar que hay 2 ejercicios cuya suma es **50**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,n1+n2)`, donde `n1,n2` son las notas de cada ejercicio).

a) **[count-if (20pt]** Escribir una función `int countif(btree<int> &T,bool (*pred)(int x));` que retorna el número de nodos del árbol `T` que satisfacen el predicado `pred`. Por ejemplo, si `T=(7 2 (4 (3 8 4) (5 . 6)))`, entonces `countif(T,odd)` debe retornar 3 y `countif(T,even)` debe retornar 5. Escribir las funciones predicados `odd` y `even`.

b) **[powerset (30pt)]**. Escribir una función `void powerset(set<int> &S,list<set<int> > &PS);` que dado un conjunto `S` devuelve el **conjunto potencia** de `S`. Por ejemplo, si `S={2,5}` entonces debe quedar `PS={{},{2},{5},{2,5}}`.
Ayuda: Escribir el algoritmo en forma **recursiva**.

- Si el conjunto es vacío, el único subconjunto es el conjunto vacío.
- Si no tomar un elemento cualquiera `x` del conjunto y tomar `W=S-{x}`.
- Formar `PW` todos los subconjuntos de `W` llamando recursivamente a la función.
- Formar `PS` tomando todos los elementos de `PW` y todos los de `PW` agregandole `x`. O sea `PS.size()=2*PW.size()`.

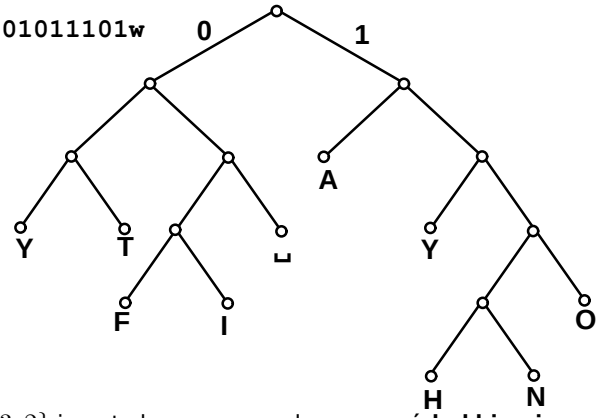
Por ejemplo, si `S={2,5}` entonces

- Sacamos cualquier elemento, por ejemplo `x=5`.
- Queda el conjunto `W={2}`.
- Los subconjuntos de `W` son `PW={{},{5}}`.
- Entonces `PS={{},{2},{5},{2,5}}`

[Ej. 3] [operativos (20pt)]

a) **[huffman (4pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `TIFON`
$P(F) = 0.10, P(N) = 0.10, P(O) = 0.10, P(T) = 0.30, P(i) = 0.05, P(C) = 0.35$. Calcular la longitud promedio del código obtenido.

b) **[hf-decode (4pt)]**
Utilizando el código de la derecha desencodar el mensaje
`00101010100111111101011111001001011101011101w`



c) **[abb (4pt)]** Dados los enteros $\{14, 8, 21, 3, 4, 11, 6, 5, 4, 13, 2\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 14, 8 y 3 en ese orden.

d) **[hash-dict (4pt)]** Insertar los números $\{3, 19, 29, 12, 11, 39, 21, 8\}$ en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión `h(x) = x`.

e) **[heap-sort (4pt)]** Dados los enteros $\{4, 8, 11, 5, 6, 16, 13\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Explique porqué el método de Huffman cumple con la *"condición de prefijos"*.

b) Defina qué es un **ABC (Árbol Binario Completo)**. De ejemplos de AB que son ABC, y que no lo son.

c) Escriba el código para ordenar un vector de enteros `v` por **valor absoluto**, es decir escriba la función de comparación correspondiente y la llamada a `sort()`.
Nota: recordar que la llamada a `sort()` es de la forma `sort(p,q,comp)` donde `[p,q)` es el rango de iteradores a ordenar y `comp(x,y)` es la función de comparación.

d) Se quiere representar el conjunto de enteros múltiplos de 3 entre 30 y 99 (o sea $U = \{30, 33, 36, ..., 99\}$) por **vectores de bits**, escribir las funciones `indx()` y `element()` correspondientes.

e) Discuta el **número de inserciones** que requieren los algoritmos de ordenamiento lentos en el peor caso.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2013-11-28]

ATENCIÓN (1): Escribir cada sección **hojas independientes**, poniendo claramente el nombre en la parte superior derecha de la misma.
ATENCIÓN (2): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases.
- **50 %** en programación.
- **50 %** en operativos.
- **60 %** sobre las preguntas de teoría.

ATENCIÓN (3): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [CLAS1 (W=20)]

a) Escribir la implementación en **C++** del TAD lista (clase **`list`**) implementado por punteros **ó** cursores. Los métodos a implementar son **`insert(p,x)`**, **`erase(p)`**, **`next()`**/**`iterator::operator++(int)`** (postfijo), **`list()`**, **`begin()`**, **`end()`**.
b) Escribir la implementación en **C++** de los métodos **`push`**, **`pop`**,**`front`** y **`top`** de los TAD pila y cola (clases **`stack`** y **`queue`**), según corresponda.
c) **AOO:** declarar las clases **`tree`**, **`cell`**, **`iterator`**, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
`tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`

[Ej. 2] [CLAS2 (W=20)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

a) **AB:** declarar las clases **`btree`**, **`cell`**, **`iterator`**, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
`btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`
b) Implementar **`bool bst_erase(btree<T> t,T x,bool (*less)(T,T))`** que elimina el elemento **`x`** del ABB T, retornando true si la eliminación fue exitosa, y false caso contrario.
c) Implementar una función **`pair<int,bool> oht_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T),bool (*equal)(T,T),T x)`** que inserta el elemento **`x`** en la tabla de dispersión abierta **`table`** utilizando la función de dispersión **`hashfunc`** y la relación de equivalencia **`equal`** retornando un par de **`int`** (que indica la cubeta) y **`bool`** que indica si la inserción fue exitosa.
d) Implementar una función **`void vecbit_diffsym(vector<bool>&, A, vector<bool>& B, vector<bool>& C);`** que devuelve la **diferencia simétrica** definida como $C = (A - B) \cup (B - A) = (A \cup B) - (A \cap B)$ con **`A,B,C`** vectores de bits que representan conjuntos de un rango contiguo de enteros **`[0, N)`**, donde **`N`** es el tamaño de los vectores **`A,B,C`** (asumir el mismo tamaño para los tres, es decir, haciendo **`int N=A.size();`** es suficiente).

[Ej. 3] [PROG1 (W=40)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

a) **[are-inverses (W=20)]** Dos correspondencias **`M1`** y **`M2`** son **inversas** una de la otra si tienen el mismo número de asignaciones y para cada par de asignación **`x->y`** en **`M1`** existe el par **`y->x`** en **`M2`**.
Consigna: Escribir una función **predicado**
`bool areinverse(map<int,int> &M1,map<int,int> &M2);` que determina si las correspondencias **`M1`**, **`M2`** son una la inversa de la otra o no.

b) **[ord-nodo (W=20)]**. Escribir una función predicado `bool ordnodo(tree<int> &A);` que verifica si cada secuencia de hermanos del subárbol del nodo `n` (perteneciente al árbol ordenado orientado `A`) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol `(3 5 (6 1 3) (7 4 5))` debería retornar `true`, mientras que para `(3 9 (6 1 3) (7 4 2))` debería retornar `false`, ya que las secuencias de hermanos `(9 6 7)` y `(4 2)` NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

[Ej. 4] [PROG2 (W=40)]

Insistimos: **deben usar la interfaz avanzada (STL)**.

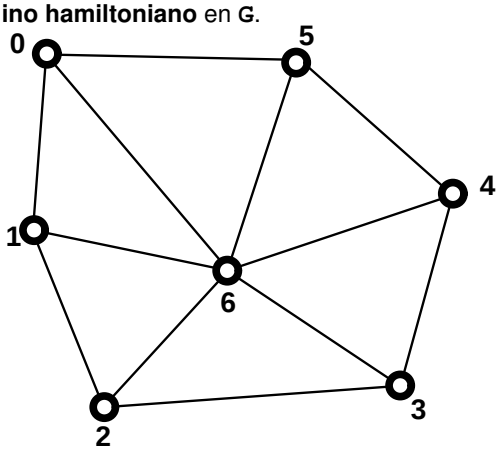
a) **[cumsum (W=20)]** *Consigna:* Escribir una función `void cumsum(btree<int> &T);` (por **suma acumulada**) que dado un **árbol binario (AB)** `T` modifica el valor de los nodos interiores, de manera que resulta ser la suma de los valores de sus hijos **más** el valor que había en el nodo antes de llamar a `cumsum()`. Los valores de las hojas no son modificados, y los valores de los nodos interiores resultan ser la suma de todos los valores del subárbol del nodo **antes** de llamar a `cumsum`.

Ejemplo: Si `T=(1 5 (2 3 7 (11 4 2)))` entonces después de llamar `cumsum(T)` debe quedar `T=(35 5 (29 3 7 (17 4 2)))`.

Ayuda:

- Si el nodo es Λ no debe hacer nada.
- Si no es Λ debe primero aplicar recursivamente la función a todos los hijos. Luego poner en el nodo `n` la suma de su valor más la de todos sus hijos.

b) **[is-hamilt (W=20)]** Dado un grafo `map<int, set<int> >G` y una lista de vértices `list<int> L` determinar si `L` es un **camino hamiltoniano** en `G`.



Consigna: Escribir una función
`bool ishamilt(map<int, set<int> > &G,list<int> &L);` que realiza la tarea solicitada.

Ayuda: Mantener un `set<int> visited` con todos los vértices visitados. Para dos enteros `x`, `y` sucesivos en la lista `L`, verificar si son vértices de `G` y si son adyacentes. Verificar que el nuevo vértice `y` no fuera visitado previamente e insertarlo en `visited`. Al final, chequear que todos los vértices fueron visitados.

Nota: Recordamos que un camino Hamiltoniano en un grafo es un camino en el mismo que pasa por todos los nodos sin repetir ninguno. Por ejemplo en el grafo de arriba el camino `{0,1,2,3,4,5,6}` es Hamiltoniano.

[Ej. 5] [OPER1 (W=20)]

a) **[rec-arbol (W=10)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE={Z,X,R,T,Q,P,A,W}`
- `ORD_POST={X,T,W,A,P,Q,R,Z}`

b) **[particionar (W=10)]**. Considerando el árbol `(Z (T Q R) (A (B (D E (F Y ))))))` decir cuál son los nodos `descendientes(F)`, `antecesores(F)`, `izquierda(F)` y `derecha(F)`.
(**Nota:** se refiere a antecesores y descendientes **propios**).

[Ej. 6] [OPER2 (W=20)]

a) **[huffman (W=5)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`NALBANDIAN`**.
$P(A) = 0.10, P(N) = 0.10, P(L) = 0.10, P(I) = 0.30, P(D) = 0.05, P(B) = 0.3, (C) = 0.05$
Calcular la **longitud promedio** del código obtenido.

b) **[abb (W=5)]** Dados los enteros $\{13, 7, 20, 2, 3, 10, 5, 4, 3, 12, 1\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 13, 5 y 2 en ese orden.

c) **[hash-dict (W=5)]** Insertar los números $\{1, 14, 24, 7, 6, 34, 16, 3\}$ en una tabla de dispersión cerrada con $B = 8$ cubetas, con función de dispersión $h(x) = x$.

d) **[heap-sort (W=5)]** Dados los enteros $\{11, 14, 4, 3, 9, 6, 2\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 7] [PREG1 (W=20)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$
\begin{aligned}
T_1 &= 5n^4 + 2 \cdot 3^n + 25\, n!. \\
T_2 &= \sqrt{2} \cdot n + 5 \cdot 2^n + \log_{10} n, \\
T_3 &= 2^{20} + n^2 + 3 \cdot 5^n, \\
T_4 &= \sqrt{n} + 5 \log_8 n + 2\, n^2 + 3\, n^5, \\
T_5 &= 2 \log_2 n + 5^3 + 3 \log_4 n + 200,
\end{aligned}
$$

b) Si la correspondencia **`M={(4->3),(2->8)}`** y ejecutamos el código **`int x= M[4]`**. ¿Que valor toma **`x`**? ¿Cómo queda **`M`**? Si ahora hacemos **`x = M[5]`**, ¿cómo quedan **`x`** y **`M`**?

c) Defina que es un **camino** en un árbol. Dado el AOO **`(b c (d f (g h)) e)`**, cuáles de los siguientes son caminos?
1) **`(f,d,b)`**,
2) **`(b,d,g,h)`**,
3) **`(d,g,h)`**,
4) **`(f,d,g,h)`**.

d) Discuta si los algoritmos de **búsqueda exhaustiva** y **heurístico** para la coloración de grafos dan la solución **óptima** al problema. Señale la complejidad algorítmica de los mismos. ¿Cuál de los dos elegiría para un grafo con 100 vértices y 500 aristas? Justifique.

e) Defina en forma **recursiva** el listado en **orden previo** y el listado en **orden posterior** de un arbol ordenado orientado con raiz **`n`** y sub-árboles hijos **`h_1,h_2,...,h_n`**.

[Ej. 8] [PREG2 (W=20)]

a) ¿Es posible insertar en una posición **no-dereferenciable** (Λ) en un AB? ¿Y en una posición dereferenciable?

b) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.

c) Defina la propiedad de **transitividad** para las relaciones de orden. Si $f(x)$ es una función sobre los reales y defino $a <_f b$ sii $f(a) < f(b)$, ¿es $<_f$ transitiva?

d) Discuta el valor de retorno de **`insert(x)`** para conjuntos.

e) ¿Cuál es el costo de **inserción** en tablas de dispersión cerradas?

`mstorti@galileo/aed-2.0.5-17-gcd7e7ca/Mon Dec 2 09:05:11 2013 -0300`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio 2. [2013-12-19]

ATENCIÓN (1): Escribir cada sección en **hojas independientes**, poniendo claramente el nombre en la parte superior derecha de la misma.
ATENCIÓN (2): Para aprobar deben obtener un **puntaje mínimo** de

- **50 %** en clases.
- **50 %** en programación.
- **50 %** en operativos.
- **60 %** sobre las preguntas de teoría.

ATENCIÓN (3): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [CLAS1 (W=20)]

a) **[list]** Escribir la implementación en **C++** del TAD lista (clase `list`) implementado por punteros **ó** cursores. Los métodos a implementar son `insert(p,x)`, `erase(p)`, `next()`/`iterator::operator++(int)` (postfijo), `list()`, `begin()`, `end()`.
b) **[stack]** Escribir la implementación en **C++** de los métodos `push`, `pop`,`front` y `top` de los TAD pila y cola (clases `stack` y `queue`), según corresponda.
c) **[AOO]** Para la clase `tree<>` (Arbol Ordenado Orientado) implementado por celdas enlazadas por punteros, declarar las clases `tree`, `cell`, `iterator`, (preferentemente respetando el anidamiento), incluyendo las declaraciones de datos miembros. Implementar el método
`tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x)`

[Ej. 2] [CLAS2 (W=20)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

a) **[AB]** Par el TAD Arbol Binario (AB): declarar las clases `btree`, `cell`, `iterator`, incluyendo las declaraciones de datos miembros. Implementar el método
`btree<T>::iterator btree<T>::find(const T& x);`. Si utiliza alguna función auxiliar impleméntela.
b) **[set-list]** Escribir los siguientes métodos del **TAD conjunto** por listas ordenadas: `insert(x)`, `find(x)`, `clear()`.
c) **[set-vecbit]** Escribir las siguientes funciones para conjuntos representados con vectores de bit:
`void set_difference(vector<bool>&A, vector<bool>& B, vector<bool>& C);`
(Recordar que `set_difference(A,B,C)` hace `C=A-B`).

[Ej. 3] [PROG1 (W=40)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

a) **[es-permutacion]** Una correspondencia es una *"permutacion"* si el conjunto de los elementos del dominio (las claves) es igual al del contradominio (los valores). De esta manera se puede interpretar a la correspondencia como una operación de permutación de las claves (de ahí su nombre). Por ejemplo
`M={1->5,3->7,9->9,7->1,5->3}` es una permutación ya que tanto las claves como los valores son el conjunto `{1,3,5,7,9}`.
Consigna: escribir una función predicado `bool es_permutacion(map<int,int> &M)` que retorna `true` si `M` es una permutación y `false` si no lo es.
Ayuda: Construir una correspondencia `M2` que mapea los valores del contradominio a las claves, es decir, para cada par de asignación `(k->v)` tal que `M[k] = v`, entonces se debe asignar el par `(v->k)` en `M2`. Para que `M` sea una permutación el conjunto de las claves de `M` debe ser igual al conjunto de las claves de `M2`.

b) **[check-ordprev]**
Escribir una función **`bool check_ordprev(tree<int> &T,list<int> &L);`** que, dado un árbol ordenado orientado **`T`** retorna **`true`** si la lista **`L`** contiene al listado en orden previo de **`T`**. Por ejemplo, si **`T=(3 (4 2 1) 0 (6 7 (8 9 5)))`** y **`L={3,4,2,1,0,6,7,8,9,5}`** entonces **`check_ordprev`** debe retornar **`true`** y si por ejemplo **`L={3,4,2,1,0,6,7,8,9,5,1000}`** o **`L={3,4,2,1,0,6,7000,8,9,5}`** **`check_ordprev`** debe retornar **`false`**. Una manera simple de hacerlo es iterar en el árbol en la manera habitual e ir sacando los elementos de la lista **`L`** si coinciden con el contenido del nodo o posición actual. Una vez que se recorrió todo el árbol se verifica en la función *"wrapper"* que la lista haya quedado vacía. Se debe retornar **`true`** si este es el caso. La función **`check_ordprev`** está definida tal que:

- si el árbol es vacío y la lista no lo es (o viceversa) **`check_ordprev`** es **`false`**,
- si lo anterior no ocurre y si el árbol es vacío **`check_ordprev`** es **`true`**,
- si no ocurre lo anterior y los contenidos del nodo y la posición de la lista actual no coinciden **`check_ordprev`** es **`false`**,
- si no ocurre lo anterior, elimino el elemento actual de la lista,
- llamo recursivamente a la función verificando que no haya llegado al fin de lista.

[Ej. 4] [PROG2 (W=40)]
Insistimos: **deben usar la interfaz avanzada (STL)**.

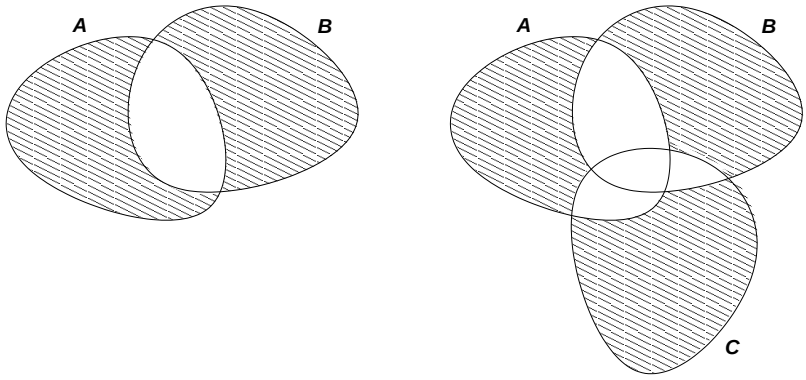
a) **[is-balanced]** Un árbol binario (AB) es balanceado si

- Es el árbol vacío ó,
- Sus subárboles derecho e izquierdo son balanceados, y sus alturas difieren a lo sumo en 1, o sea $|h_L - h_R| \leq 1$.

Por ejemplo **`(1 (2 (3 (4 5 6) 7) (13 8 9)) (15 10 (16 11 12)))`** es un árbol balanceado (notar que no necesariamente las profundidades de las hojas difieren en ± 1).
Consigna: Escribir una función **`bool is_balanced(btree<int> &T);`** que retorna **`true`** si el árbol está balanceado.
Ayuda: En la función auxiliar retornar el resultado de si el árbol es balanceado o no, pero además retornar (con un valor pasado por referencia, o con un **`pair`**) la altura del árbol correspondiente.

b) **[diff-sym]** Para dos conjuntos A, B, la *"diferencia simétrica"* se define como

$$\begin{aligned} \text{diff_sym}(A,B) &= (A-B) \cup (B-A), \quad \text{o también} \\ &= (A \cup B) - (A \cap B) \end{aligned}$$



En general, definimos la diferencia simétrica de varios conjuntos como el conjunto de todos los elementos que pertenecen a uno y sólo uno de los conjuntos. En las figuras vemos en sombreado la diferencia simétrica para dos y tres conjuntos. Por ejemplo, si $A = \{1,2,5\}$, $B = \{2,3,6\}$ y $C = \{4,6,9\}$ entonces $\text{diff_sym}(A,B,C) = \{1,3,4,5,9\}$.
Consigna: Escribir una función **`void diff_sym(list<set<intd> > &l,set<int> &s);`** que retorna en **`s`** la diferencia simétrica de los conjuntos en **`l`**.

Ayuda: Notar que en el caso de tres conjuntos si $S = \text{diff_sym}(A, B)$ y $U = A \cup B$, entonces $\text{diff_sym}(A, B, C) = (S - C) \cup (C - U)$. Esto vale en general para cualquier número de conjuntos, de manera que podemos utilizar el siguiente lazo

$l =$ lista de conjuntos, $S = \emptyset, U = \emptyset$;
for Q en la lista de conjuntos l **do**
  $S = (S - Q) \cup (Q - U)$;
  $U = U \cup Q$;
end for

Al terminar el lazo, S es la diferencia simétrica buscada.
Nota: Recordar que para las operaciones binarias (por ej. `set_union(A,B,C);`), los conjuntos deben ser distintos, es decir no se puede hacer `set_union(A,B,A);`, en ese caso debe hacerse
`set_union(A,B,tmp); tmp=A;`
Nota: Para usar las operaciones binarias se puede usar tanto la versión simplificada que hemos utilizado en el curso, `void set_union(set &A,set &B,set &C);` como la versión de las STL
`set_union(A.begin(),A.end(),B.begin(),B.end(),inserter(C,C.begin()));`

[Ej. 5] [OPER1 (W=20)]

a) **[rec-arbol (W=10)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE={Y,W,Q,S,P,O,B,V}`,
- `ORD_POST={W,S,V,B,O,P,Q,Y}`.

b) **[particionar (W=10)]**. Considerando el árbol `(Y (S P Q) (B (A (C D (E X)))))` decir cuál son los nodos `descendientes(S)`, `antecesores(S)`, `izquierda(S)` y `derecha(S)`.
(**Nota:** se refiere a antecesores y descendientes **propios**).

[Ej. 6] [OPER2 (W=20)]

a) **[huffman (W=5)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`MANDELA`**.
$P(M) = 0.10, P(N) = 0.10, P(L) = 0.10, P(D) = 0.30, P(A) = 0.05, P(E) = 0.3, (C) = 0.05$
Calcular la **longitud promedio** del código obtenido.

b) **[abb (W=5)]** Dados los enteros $\{14, 8, 21, 3, 4, 11, 6, 5, 13, 2\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 14, 8 y 3 en ese orden.

c) **[hash-dict (W=5)]** Insertar los números $\{2, 15, 25, 8, 7, 35, 17, 4\}$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x$.

d) **[heap-sort (W=5)]** Dados los enteros $\{12, 15, 5, 4, 10, 7, 3\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 7] [PREG1 (W=20)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = \sqrt{n} + 3\log_{10} n + 4\,n^2 + 2\,n^5,$$
$$T_2 = \sqrt{5} \cdot n + 3 \cdot 4^n + \log_2 n,$$
$$T_3 = 3n^5 + 6 \cdot 2^n + 5\,n!.$$
$$T_4 = 3\log_2 n + 3^5 + 125.$$
$$T_5 = 20^2 + 5n^2 + 2 \cdot 3^n,$$

b) ¿Porqué se dice que la pila es una estructura *FIFO ("First in, First Out")*? ¿Porqué se dice que la cola es una estructura *LIFO ("Last In, First Out")*?

c) Cual es la complejidad algorítmica (mejor/promedio/peor) de la función **`lower_bound()`** para correspondencias implementadas por
   1) listas ordenadas,
   2) vectores ordenados.

d) Considerando la implementación de pilas con **listas simplemente enlazadas**. ¿Cuál es la diferencia entre elegir como tope de la pila el comienzo o fin de la lista?

e) ¿Como se define la **altura** de un nodo en un árbol? ¿Cuál es la altura de los nodos **`A`**, **`B`**, y **`C`** en **`(W X Y (A B (C (D F G H))))`**?

[Ej. 8] [PREG2 (W=20)]

a) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.

b) Escriba el código para ordenar un vector de enteros **`v`** por **valor absoluto**, es decir escriba la función de comparación correspondiente y la llamada a **`sort()`**.
Nota: recordar que la llamada a **`sort()`** es de la forma **`sort(p,q,comp)`** donde **`[p,q)`** es el rango de iteradores a ordenar y **`comp(x,y)`** es la función de comparación.

c) Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**.

d) Se quiere representar el conjunto de enteros múltiplos de 3 entre 30 y 99 (o sea $U = \{30, 33, 36, ..., 99\}$) por **vectores de bits**, escribir las funciones **`indx()`** y **`element()`** correspondientes.

e) Discuta la complejidad algorítmica de las operaciones binarias **`set_union(A,B,C)`**, **`set_intersection(A,B,C)`**, y **`set_difference(A,B,C)`** para conjuntos implementados por vectores de bits, donde **`A`**, **`B`**, y **`C`** son subconjuntos de tamaño n_A, n_B, y n_C respectivamente, de un conjunto universal **`U`** de tamaño N.

Algoritmos y Estructuras de Datos.
TPL1. 1er Trábajo Práctico de Laboratorio. [2013-08-31]

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.

1. split-mod

Escribir una función `void split_mod(list<int> &L, int m, vector<list<int> > &VL);` que dada una lista `L`, y un entero `m` divide a la lista en las sublistas de los elementos que son congruentes entre sí módulo `m`. Es decir, si `m=2` debe dejar en `VL[0]` los elementos pares y en `VL[1]` los elementos impares. Si `m=3` debe dejar en `VL[0]` los múltiplos de 3, en `VL[1]` los `x` tales que `x%3==1` y así siguiendo. En general en `VL[j]` deben quedar los elementos tales que `x%m==j`. Por ejemplo si `L=(1,3,2,1,5,6,0)` y `m=3`, debe quedar

```
VL[0] = (3,6,0)
VL[1] = (1,1)
VL[2] = (2,5)
```

Los elementos deben quedar en las listas `VL[j]` en el mismo orden que estaban en `L`.

2. is-sublist

Escribir una función predicado `bool is_sublist(list<int> &L1, list<int> &L2);` que determina si `L2` es una sublista de `L1` es decir si `L2` se puede obtener de `L1` sólo borrando elementos (pero sin insertar). Por ejemplo

```
L1=(1 3 2 5 6) L2=(3 5) -> true
L1=(1 3 2 5 6) L2=(3 5 2) -> false
```

Ayuda: Tomar dos posiciones `p1` y `p2`. Avanzar `p1` hasta encontrar el elemento en `p2`. Si no lo encuentra retorna falso, si lo encuentra avanza ambas posiciones.

3. max-sublist

Escribir una función `void max_sublist(list<int> &L, list<int> &subl);` que dada una lista `L` retorna la máxima sublista contigua de `L` con elementos pares `subl`. Por ejemplo si

```
L=(5,4,5,8,6,2,10,9,3,1) -> subl=(8,6,2,10)
```

Si hay varias sublistas con la misma longitud debe retornar la primera. Por ejemplo

```
L=(1,2,4,6,5,0,2,4) -> subl=(2,4,6)
```

Ayuda: Guardar la máxima lista actual en `subl`. Recorrer la lista `L` y cada vez que se encuentra un elemento par, copiarlo junto todos los elementos pares siguientes en una lista temporaria `tmp`. Si la lista `tmp` es más larga que `subl` reemplazarla, si no seguir adelante.

Algoritmos y Estructuras de Datos.
TPL1. 1er Trábajo Práctico de Laboratorio. [2013-08-31]

PASSWD PARA EL ZIP: **`ARV5671MJA`**

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.

- **merge-kway**
  Dado un vector de **listas ordenadas** hacer una **fusión K-way**, que hace una fusión de todas las listas en una sola, también ordenada. **`void mergekw(vector<list<int>> &VL, list<int> &L)`**
  Ayuda: El algoritmo K-way consiste en recorrer los primeros elementos de las listas (atención que algunas pueden estar vacías), tomar el menor e insertarlo al fin de la lista L Esto se realiza hasta que todas las listas esten vacías. El algoritmo es similar a la fusión de dos listas, pero generalizado a K listas. Por ejemplo, si $K = 3$ y

  ```
  VL[0]=(1 4 6)
  VL[1]=(1 3 4 6 10)
  VL[2]=(6 8 )
  ```

  Entonces

  ```
  L=(1 1 3 4 4 6 6 6 8 10)
  ```

- **split-list** Dado un vector de listas **`VL1`** y un entero **`M`** debe devolver otro vector de listas **`VL2`** donde las listas tienen a lo sumo M elementos.
  `void split_list(vector<list<int>> &VL1, int M,vector<list<int>> &VL2);`
  Para eso toma cada una de las listas y si su longitud es mayor que **`M`** la va dividiendo en pedazos de longitud **`M`** y la va poniendo en **`VL2`**. Por ejemplo:

  ```
  VL1[0]=(4 6 8 6 4 6 1 3 5 5)
  VL1[1]=(6 4 3)
  VL1[2]=(1 10 3 7 8 1 2 3 8)
  VL1[3]=(6 2 8 8)
  ```

  Entonces si hacemos **`split_list(VL1,3,VL2);`**

  ```
  VL2[0]=(4 6 8)
  VL2[1]=(6 4 6)
  VL2[2]=(1 3 5)
  VL2[3]=(5)
  VL2[4]=(6 4 3)
  VL2[5]=(1 10 3)
  VL2[6]=(7 8 1)
  VL2[7]=(2 3 8)
  VL2[8]=(6 2 8)
  VL2[9]=(8)
  ```

- **join-list** Dado un vector de listas **VL1** escribir una función
  `void join_list(vector<list<int>> &VL1, int M,vector<list<int>> &VL2);`
  que va juntando listas de **VL1** hasta que todas tengan longitud **>=M**.

  ```
  VL1[0]=(2 4 10)
  VL1[1]=(4 8 5)
  VL1[2]=(7 5 7)
  VL1[3]=(2 9)
  VL1[4]=(1 7 5)
  VL1[5]=(3 8 5)
  VL1[6]=(6 3 10)
  VL1[7]=(3 8 4)
  VL1[8]=(2)
  ```

  Entonces si hacemos `join_list(VL1,5,VL2);`

  ```
  VL2[0]=(2 4 10 4 8 5)
  VL2[1]=(7 5 7 2 9)
  VL2[2]=(1 7 5 3 8 5)
  VL2[3]=(6 3 10 3 8 4)
  VL2[4]=(2)
  ```

Algoritmos y Estructuras de Datos.
TPL2. 1er Trábajo Práctico de Laboratorio. [2013-10-05]

PASSWD PARA EL ZIP: `6DG1ZG7IH6`

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las tres funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- También se incluyen con el paquete los utilitarios **`tree.h`** y **`util_tree.h`**. Estos contienen diferentes funciones utilitarias que pueden servir para debuggear el programa.

Ejercicios

[Ej. 1] **[count-level]** Escribir una función **`void count_level(tree<int> &T, int l)`**, que cuenta cuantos nodos hay en el nivel **`l`** del árbol **`T`**.
Ayuda: La definición recursiva es:

$$\texttt{count_level}(n,l) = \begin{cases} 0; & \text{si } n = \mathbf{\Lambda} \text{ o } l < 0, \\ 1; & \text{si } l = 0, \\ \sum_{c=\text{hijo den}} \texttt{count_level}(c, l-1). & \end{cases} \quad (1)$$

Por ejemplo, para el árbol **`T=(a (b c d e) (f (g h i)))`** debe dar

```
count_level(T,1) -> 2
count_level(T,2) -> 4
```

[Ej. 2] **[is-shift]** Dados dos grafos **`G1,G2`** (como **`typedef map<int,list<int>> graph_t;`**), escribir una función **`bool is_shift(graph_t &G1,graph_t &G2,int m);`** que determina si **`G2`** es un 'shift' del **`G1`**, es decir la arista **`(x,y)`** está en **`G1`** si y solo si **`(x+m,y+m)`** está en **`G2`**.
Por ejemplo, si

```
G1 = [1->{2,4}, 2->{1,3,4}, 3->{2,4}, 4->{1,2,3}]
G2 = [4->{5,7}, 5->{4,6,7}, 6->{5,7}, 7->{4,5,6}]
```

entonces **`is_shift(G1,G2,3) -> true`**.

[Ej. 3] **[odd2even]** Dada una lista **`L`**, escribir una función
`void odd2even(list<int> &L,map<int,list<int>> &M);` que mapea cada elemento impar de **`L`** a la siguiente subsecuencia de elementos pares. Por ejemplo si **`L=(1,2,4,3,2,5,2,7,1,6,9,2,14)`** entonces debe dejar **`M={1->(2,4,6),3->{2},5->{2},7->{},9->(2,14)}`**. Si un numero impar aparece seguido de otro impar entonces el valor correspondiente debe ser la lista vacia (por ejemplo el 7 arriba). Si un numero impar aparece mas de una vez, entonces el valor correspondiente debe ser la concatenacion de todas las subsecuencias correspondiente (por ejemplo el 1 arriba).

Algoritmos y Estructuras de Datos.
RECUP-TPL2. Recuperatorio Trábajo Práctico de Laboratorio 2. [2013-10-22]

PASSWD PARA EL ZIP: `OZU5DAUTYFIB`

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las tres funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- También se incluyen con el paquete los utilitarios **`tree.h`** y **`util_tree.h`**. Estos contienen diferentes funciones utilitarias que pueden servir para debuggear el programa.
- Algunas funciones utilitarias que pueden servir

```
void printmap(map<int,int> &M);
void printmap(map<int, list<int> >& M);
void printl(list<int> &L);
T.lisp_print();
```

Ejercicios

[Ej. 1] **[evpath]** Escribir una función **`int evpath(tree<int> &T),`** que devuelve la longitud del máximo camino donde todos los elementos de los nodos son pares. Por ejemplo, para el árbol **`T=(0 (1 2 3) (2 (4 5 7) 3))`** debe dar 2, ya que el máximo camino de pares es **`(0 2 4)`** de longitud 2.
Ayuda: La definición recursiva es:

$$\texttt{evpath}(T,n) = \begin{cases} -1; & \text{si } n = \mathbf{\Lambda}, \\ 0; & \text{si } n \text{ es impar}, \\ 1 + \max_{c=\text{hijo den}} \texttt{evpath}(T,c); & \text{caso contrario .} \end{cases}$$

[Ej. 2] **[checksz]** Dados dos grafos **`G1,G2`** (como **`typedef map<int,list<int>> graph_t`**), escribir una función **`bool checksz(graph_t &G1,graph_t &G2);`** que determina si un dado nodo tiene la misma cantidad de vecinos en **`G1`** y en **`G2`**. Es decir las claves de **`G1`** y **`G2`** deben ser las mismas y dada una clave **`k`**, las imágenes **`G1[k]`** y **`G2[k]`** deben tener el mismo tamaño.
Ejemplo 1:

```
G1 = [1->{2,4}, 2->{1,3,4}]
G2 = [1->{3,4}, 2->{5,6,7}]
checksz(G1,G2) -> true
```

Ejemplo 2:

```
G1 = [1->{2,4}, 2->{1,3}]
G2 = [1->{3,4}, 2->{5,6,7},3->{6}]
checksz(G1,G2) -> false
```

ya que los tamaños de `G1[2]`, `G2[2]` son diferentes y además la clave 3 aparece sólo en `G2`.
Ayuda: Chequear que los tamaños de ambos grafos (o sea la cantidad de claves) sea la misma. Recorrer las claves de `G1` y verificar que la clave en `G2` exista y los tamaños de las imágenes sean iguales.
Alternativa: Para cada uno de los grafos construir un mapa `map<int,int> G1S,G2S` que mapean las claves de `G1`, `G2` en los tamaños de las listas. Luego verificar que `G1S,G2S` sean iguales.

[Ej. 3] **[maxodd]** Dada una lista `L`, escribir una función
`void maxodd(list<int> &L,map<int,int> &M);` que mapea cada elemento par de `L` al máximo de la siguiente subsecuencia de elementos impares. Por ejemplo si `L=(1,2,4,3,2,5,8,2,7,1,6,9,2,14)` entonces debe dejar `M={2->7,4->3,6->9}`.

- Si un numero par aparece seguido de otro par entonces la subsecuencia de impares está vacía y en el mapa final no debe asignarle ningún valor. (Como en el ejemplo de arriba donde el 8 no recibe ningún valor.)
- Si un numero par aparece mas de una vez, entonces el valor correspondiente debe ser el máximo de todas las subsecuencias correspondientes.
- Ignorar elementos impares que estén al principio de la lista (en el ejemplo de arriba el 1).

Algoritmos y Estructuras de Datos.
TPL3. Trábajo Práctico de Laboratorio 3. [2013-11-09]

PASSWD PARA EL ZIP: `FW4R92XA5BK2`

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las tres funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- También se incluyen con el paquete la clase de arbol binario **`btree.h`**.
- Algunas funciones utilitarias que pueden servir

```
void print(set<int> &S);
void print(list<set<int> > &LS);
void print(vector<set<int> > &VS);
T.lisp_print();
```

Ejercicios

[Ej. 1] **[gatherset]** Dado una serie de conjuntos de enteros S_j, con $j \in [0, N_S)$ juntarlos entre sí aquellos que tienen al menos un elemento en común. Es decir debemos encontrar otro grupo de conjuntos W_j, con $j \in [0, N_W)$ tal que

- Los W_j son disjuntos entre sí, esto es $W_j \cap W_k = \emptyset$ si $j \neq k$
- Dos enteros están en el mismo W_k (sólo en uno porque son disjuntos) si y sólo si comparten alguno de los S_j.

Por ejemplo, si **`S0={5,7,11}`**, **`S1={0,1,3}`**, **`S2={1,2,10}`**, y **`S3={7,13,22}`**. Entonces tenemos **`W0={5,7,11,13,22}`**, **`W1={0,1,2,3,10}`**.

Consigna: Escribir una función
`void gatherset(list< set<int<> > &S, list< set<int<> > &W);` que realiza la tarea descripta.

Ayuda:

- Inicializar **`W`** como una lista vacía.
- Mientras que **`S`** no este vacío):
  - Hacer **`tmp=`** algún elemento de **`S`** y eliminarlo de **`S`**
  - Para cada **`w`** en **`W`**:
    - Si la intersección de **`w`** y **`tmp`** no es nula
      - Hacer **`tmp=tmp`** $\cup$ **`w`**.
      - Eliminar **`w`** de **`W`**.

- Insertar **`tmp`** en **`W`**

[Ej. 2] **[maxshare]** Dado una serie de conjuntos de enteros S_j, con $j \in [0, N_S)$ y otro conjunto W encontrar aquel S_k cuya intersección con W tiene el máximo tamaño, es decir el conjunto tal que

$$|S_k \cap W| \leq |S_j \cap W|, \ \forall j \in [0, N_s)$$

donde $|\ .\ |$ denota el tamaño del conjunto. Por ejemplo, si **`W={1,3,5}`** y **`S0={0,2,4,8}`**, **`S1={0,1,4,9}`**, **`S2={5,10}`**, y **`S3={1,3,6}`**. Entonces tenemos que el elegido debe ser **`S3={1,3,6}`** ya que el tamaño de la intersección con **`W`** es 2 y para los otros es menor.

Consigna: Escribir una función **`set<int> maxshare(vector< set<int> > &S, set<int> &W);`** que realiza la tarea descripta. Si hay varios conjuntos que tienen el misma tamaño de intersección debe retornar el primero de ellos.

Ayuda: Ir recorriendo **`S`** y mantener unas variables auxiliares **`set<int> Smax; int nsmax;`** que son el conjunto de **`S`** que tiene la máxima intersección hasta el momento, y el tamaño de su intersección con **`W`**. Para cada conjunto de **`S`** calcular la intersección con **`W`**, y si es estrictamente mayor que **`nsmax`** reemplazar los mínimos actuales.

[Ej. 3] **[sccount]** Dado una árbol binario (AB) **`T`**, determinar cuantos nodos de **`T`** tienen exactamente un solo hijo (*single child count*).

Ejemplos:
Para **`T=(8 (7 9 2) (3 . (9 . 1)));`** debe retornar **`2`**
Para **`T=(2 8 (1 (2 . 6) .));`** debe retornar **`2`**
Para **`T=(4 (9 (2 6 .) .) 8);`** debe retornar **`2`**
Para **`T=(8 (6 (5 . (4 3 (4 . (9 . 6)))) .) 3);`** debe retornar **`4`**

Definición recursiva:

$$\texttt{sccount}(n) = \begin{cases} 0; & \text{si } n = \Lambda, \\ s + \texttt{sccount}(l) + \texttt{sccount}(r); & \text{caso contrario,} \end{cases} \quad (1)$$

donde $s = 1$, si n tiene exactamente un sólo hijo y 0 en caso contrario. l, r son los hijos izquierdo y derecho de n.

Consigna: Escribir una función **`int sccount(btree<int> &T);`** que realiza la tarea descripta.

Curiosidad: Coincide con la cantidad de puntos que aparecen en notación Lisp, pero no recomendamos encarar esa estrategia.

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
RECUP TPL3. Recuperatorio del Trábajo Práctico de Laboratorio 3. [2013-11-21]

PASSWD PARA EL ZIP: `XPXC LWIK 7MUL`
(Son 12 letras mayúsculas y números, sin espacios)

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las tres funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- También se incluyen con el paquete la clase de arbol binario **`btree.h`**.
- Algunas funciones utilitarias que pueden servir

```
void print(set<int> &S);
void print(list<set<int> > &LS);
T.lisp_print();
```

- Las funciones **`evaluar`** ahora tienen un segundo argumento **`int vrbs`** (*verbosity*). Si **`vrbs==1`** entonces la clase evaluadora imprime los argumentos de entrada, valores retornados por la función del usuario (*user*) y esperados (*reference*). Por ejemplo en el caso de **`sum_sim`** si hacemos

  ```
  ev.evaluar3(sum_sim,1);
  ```

  entonces la clase evaluadora imprime algo así como

  ```
  A: (3 (9 5 (9 . 9)) 7)
  B: (3 7 (9 (9 9 .) 5))
  retval: user 0, ref 1
  ```

  `A,B` son los árboles datos y el valor espera de retorno es 1, mientras que la función escrita por el usuario retorno 0 (incorrecto).

Ejercicios

[Ej. 1] **[subsup]** Dado una lista de conjuntos de enteros **`LS`**, y un conjunto **`W`** encontrar la lista **`LSUB`** de aquellos conjuntos S_j de **`LS`** tales que son subconjuntos de **`W`** ($S_j \subseteq W_j$). También debe devolver los conjuntos **`LSUP`** que son supraconjuntos de **`W`**, es decir los ($S_j \supseteq W_j$).
Ejemplo: Si **`LS={{1,2},{2},{2,3,4},{2,4,6},{1,3,5})`** y **`W={2,4}`**, entonces debe retornar **`LSUB={{2})`** y **`LSUP={{2,3,4},{2,4,6})`**.

Consigna: Escribir una función

```
typedef list<set<int>> ls_t;
void subsup(ls_t &LS,set<int> &W,ls_t &LSUB,ls_t &LSUP);
```

que realiza la tarea indicada. Los conjuntos en **`LSUB`** y **`LSUP`** deben quedar en el mismo orden que en el conjunto original.

Ayuda1: Recordar que para determinar si $A \subseteq B$ basta con hacer la diferencia $A - B = \emptyset$. Recomendamos escribir una función auxiliar **`bool issubset(set<int> &A,set<int> &B)`**.

Ayuda2: Recorrer los conjuntos en **`LS`** y verificar si incluyen a **`W`** o son incluidos por **`W`**. Dependiendo de esto incluirlo en **`LSUB`** o **`LSUP`**. **OJO** que puede ocurrir que un S_j esté en ambos (cuando ocurre?).

[Ej. 2] **[inall]** Dado una lista de conjuntos **`LS`** determinar si alguno de ellos está incluido en todos los otros, es decir si existe un S_j tal que $S_j \subset S_k$ para todo k.

Consigna: Escribir una función

```
typedef list<set<int>> ls_t;
bool inall(ls_t &LS,set<int> &S);
```

tal que

- Si alguno de los S_j satisface las condiciones entonces devuelve **`true`** y lo retorna por **`S`**.
- Caso contrario devuelve **`false`** (**`S`** queda ideterminado).

Ejemplos:

- **`LS={{1,2,3},{2,3,4}}`** retorna **`false`**
- **`LS={{1,2,3},{1,2,3,4},{1,2,3}}`** retorna **`true`** y **`S={1,2,3}`**.
- **`LS={{1,2,3},{1},{1,2}}`** retorna **`true`** y **`S={1}`**

Ayuda: Si existe el tal conjunto debe ser el de menor tamaño. Por lo tanto

- Buscar el **`S`** de menor tamaño.
- Verificar si **`S`** es subconjunto de todos los otros.

[Ej. 3] **[sum-sim]** Dados dos AB, **`T1`** y **`T2`**, escribir una funcion que verifique que la cantidad de nodos en el nivel **`l`** en **`T1`** y **`T2`** sea la misma, y que además la suma de los nodos de ese nivel coincida.

Consigna: Sea **`AB_t`** el tipo de árbol binario

```
typedef btree<int> AB_t;
```

Escribir una función
`bool sum_sim(AB_t & T1, AB_t & T2, int l);`
que realiza la tarea indicada.

Ayuda: Notar que la cantidad de nodos en el nivel **`L`** se puede escribir recursivamente como

$$\texttt{nodecount}(\texttt{n}, L) = \begin{cases} 0; & \text{si } \texttt{n} = \Lambda, \\ 1 + \texttt{nodecount}(\texttt{l}, L-1) + \texttt{nodecount}(\texttt{r}, L-1); & \text{caso contrario,} \end{cases} \tag{1}$$

Para la suma de los valores de los nodos tenemos algo similar

$$\texttt{nodesum}(\texttt{n}, L) = \begin{cases} 0; & \text{si } \texttt{n} = \Lambda, \\ \texttt{*n} + \texttt{nodesum}(\texttt{l}, L-1) + \texttt{nodesum}(\texttt{r}, L-1); & \text{caso contrario,} \end{cases} \tag{2}$$

Se sugiere encarar algunas de las siguientes estrategias,

a) Escribir las funciones auxiliares **`nodesum`** y **`nodecount`** y aplicarlas a la raíz de **`T1`** y **`T2`** y chequear los valores retornados.
b) Escribir una función que calcule las dos cosas al mismo tiempo, retornándolas por un **`pair<int,int>`**.
c) Escribir una función que calcule las dos cosas al mismo tiempo retornándolas por referencia.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2014-09-18]

ATENCIÓN(1): Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 4) y 50 % en las restantes secciones.
ATENCIÓN(2): Recordar que tanto en las clases (Ej. 1) como en los ejercicios de programación (Ej 2.) **deben usar la interfaz STL**.

[Ej. 1] [clases (20pt)]

a) **[list (10pt)]** Escribir la implementación en **C++** del TAD lista (clase **`list`**) implementado por punteros **ó** cursores. Los métodos a implementar son **`insert(p,x)`**, **`erase(p)`**, **`iterator::operator++(int)`**.
b) **[stack-queue (10pt)]** Escribir la implementación en **C++** de los métodos **`push`**, **`pop`**, **`front`**, y **`top`** de los TAD pila y cola (clases **`stack`** y **`queue`**), según corresponda.
c) **[map (10pt)]** Escribir la implementación en **C++** del TAD correspondencia (clase **`map`**) implementado con listas ordenadas. Métodos a implementar: **`find(key)`**.

[Ej. 2] [Programación (50pt)] Recordar que en los ejercicios de programación **deben usar la interfaz STL**.

a) **[stack-sep (10pt)]**. Dada una pila **`S`**, escribir una función que deja en el tope los elementos pares y en el fondo los impares. El algoritmo debe ser **estable** es decir que los pares deben estar en el mismo orden entre si y los impares también. Por ejemplo si **`S=(0,1,2,3,4,5,6)`** entonces debe quedar **`S=(0,2,4,6,1,3,5)`**.
Consigna: Escribir una función **`void stacksep(stack<int> &S);`** que realiza la tarea indicada.
Restricciones: Como estructuras auxiliares sólo pueden usar pilas. El algoritmo no debe alocar memoria adicional, es decir la memoria adicional alocada debe ser $O(1)$.

b) **[exists-path (20pt)]**. Sebastián un buen día queda en juntarse para comer un asado en la casa de Santiago. Pero resulta que si bien son buenos para organizar reuniones, no son buenos para recordar fechas, y olvidaron que ese día habrán varios desfiles por el carnaval, quedando solo algunas calles de un único sentido habilitadas para los vehículos.

Sebastián es un conductor muy prudente, y no se arriesgará a que lo multen, por lo que solo tomará los caminos en el sentido correcto.

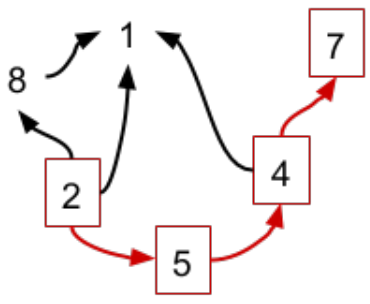
Entonces con rapidez arma un mapa (grafo) de las calles habilitadas que tendrá a las intersecciones de las calles como nodos y a las calles como aristas. Seba le pide a usted (ya que él está muy ocupado haciendo la ensalada para la cena) que escriba una función
`bool exists_path (map <int, list <int> > &M, int m, int n);` que dado un mapa **`M`**, le haga saber si existe al menos un camino que él pueda hacer desde su casa (ubicada en la intersección **`m`**) hacia la casa de Santiago, que está en la intersección **`n`**.

¿Podrán juntarse, o deberán posponer la reunión?

Ejemplo: Dado un mapa **`M`**

Entonces si **`m=2`**, y **`n=7`** el resultado debe ser verdadero.

```
1 >( )
2 >( 1, 5, 8 )
4 >( 1, 7 )
5 >( 4 )
7 >( )
8 >( 1 )
```



c) **[extractm (20pt)]**.
Dada una lista **L**, y un entero **m**, escribir una función
`void extractm(list<int> &L,int m,list<int> &Lm);` que genere una lista **Lm** con todos los elementos que aparecen exactamente **m** veces en **L**. Por ejemplo, si **`L=(0,1,2,3,4,5,1,2,4,2)`** entonces **`extractm(L,2,Lm)`** debe dejar **`Lm=(1,4)`** ya que son los únicos dos elementos que están exactamente 2 veces. **`extractm(L,1,Lm)`** debe retornar **`Lm=(0,3,5)`**.
Restricciones: Construir una correspondencia **`map<int,int> F`** cuyas claves son los elementos de **L** y la imagen la **frecuencia**, es decir cuántas veces aparece en **L**. Luego se recorre **F** y se insertan en **Lm** sólo las que tienen frecuencia **m**.

[Ej. 3] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$
\begin{aligned}
T_1 &= 3n^4 + 4n^3 + 20, \\
T_2 &= 4^n + 5n! + 3n^{10} + 4n^5, \\
T_3 &= 10 + 2^n + n^2, \\
T_4 &= 4 + 2\log_{10} n + 50!, \\
T_5 &= 2\log_3 n + 5n.
\end{aligned}
$$

b) Sea la correspondencia **`M={(2->1),(8->5)}`** y ejecutamos el código int **`x= M[8]`**. ¿Qué ocurre? ¿Qué valores toman **x** y **M**? ¿Y si hacemos **`x = M[1]`**?

c) ¿Cuál es la complejidad algorítmica (mejor/promedio/peor) de las siguientes funciones? (asumir listas implementadas con celdas enlazadas por punteros o cursores)

1) **`list<T>::begin()`**,
2) **`list<T>::insert(p,x)`**,
3) **`list<T>::clear(p,x)`**,
4) **`map<K,V>::find(x)`**, para la implementación con vectores ordenados,
5) **`map<K,V>::find(x)`**, para la implementación con listas ordenadas.

d) Discuta las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas. ¿Cuáles son los métodos cuyo tiempo de ejecución cambia y por qué?

e) ¿Qué ventajas o desventajas tendría implementar la clase **stack** (pila) en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista? ¿Cuáles serían los tiempos de ejecución para **`top`**, **`pop`**, **`push`**?
Lo mismo para cola: ¿Qué ventajas o desventajas tendría implementar la clase **queue** (cola) en términos de **lista simplemente enlazada** poniendo el frente de la pila en el fin de la lista? ¿Cuáles serían los tiempos de ejecución para **`front`**, **`pop`**, **`push`**?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 2. [2014-11-13]

ATENCIÓN: Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 4) y 50 % en las restantes secciones.

[Ej. 1] [clases (W=20pt)]

Recordar que **deben usar la interfaz STL**.

a) **[aoo (5pt)]**
Arbol Ordenado Orientado: Implementar (haciendo uso de la interfaz STL) los métodos

- `iterator lchild();`
- `iterator right ();`
- `iterator insert (iterator p, T t);`

considerando que se debe escribir la clase `iterator` (en el scope que corresponda) además de aquellas estructuras sobre las cuales esta se construye (a saber, la clase `cell`).

b) **[hash-dict (5pt)]**
Diccionario por tablas de dispersión cerrada: Implementar la clase `iterator_t` en conjunto con aquello que considere pertinente dentro de la parte privada de la clase `hash_set`. Finalmente, implemente el método `iterator_t find (key_t & x)`

c) **[vbitset (5pt)]**
Conjuntos por vectores de bits: Implementar los métodos

- `pair_t insert (elem_t x);`
- `elem_t retrieve (iterator_t p);`
- `void set_difference (set & A, set & B, set & C);`

para ello deberá además implementar la parte privada de la clase.

d) **[abb (5pt)]**
Conjuntos por árbol binario de búsqueda: Implementar (haciendo uso de la interfaz STL) el método

- `void erase (iterator_m)`

definiendo para ello (y respetando el anidamiento pertinente) la clase iterator.

[Ej. 2] [operativos (W=20pt)]

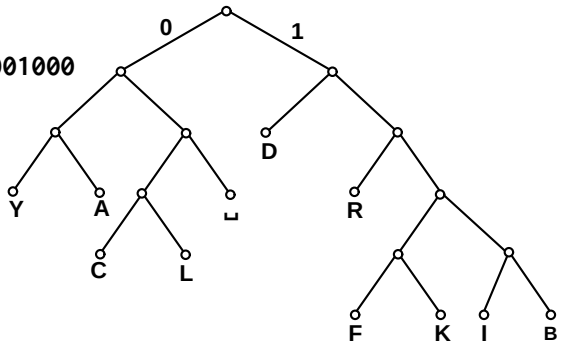
a) **[rec-arbol (3pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD-PRE =(Z,A,B,C,D,E,F,T,G,H)`
- `ORD-POST =(A,E,F,D,G,H,T,C,B,Z)`

b) **[part-arbol (2pt)]** Dado el árbol ordenado orientado (AOO): `(A (R J L) (T (V (Z D) Q)))` determinar cuales son los nodos **antecesores, descendientes, izquierda** y **derecha** del nodo `V`. ¿Son **disjuntos**? Justifique.

c) **[huffman (3pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `CYBERMONDAY`,
$P(O) = 0.10$, $P(D) = 0.10$, $P(B) = 0.10$, $P(N) = 0.30$, $P(C) = 0.10$, $P(A) = 0.02$, $P(E) = 0.02$, $P(R) = 0.02$, $P(Y) = 0.04$, $P(M) = 0.20$.
Calcular la longitud promedio del código obtenido.

d) **[hf-decode (2pt)]**
Utilizando el código de la derecha desencodar el mensaje
`11111010100101001110101111100110111101000100 0`



e) **[abb (2pt)]** Dados los enteros $\{14, 8, 21, 3, 4, 11, 6, 5, 4, 13, 2\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 14, 8 y 3 en ese orden.

f) **[hash-dict (3pt)]** Insertar los números $\{2, 18, 28, 11, 10, 38, 20, 7\}$ en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión **`h(x) = x`**.

g) **[heap-sort (3pt)]** Dados los enteros $\{3, 12, 10, 4, 5, 15, 7\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

h) **[quick-sort (2pts)]** Dados los enteros $\{8, 2, 9, 4, 1, 3, 10, 2, 7, 5, 4, 0\}$ ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 3] [Preguntas (W=20pt, 4pt por pregunta)]

a) Explique el concepto de **estabilidad** para algoritmos de ordenamiento. De ejemplos de algoritmos de ordenamiento estables y no estables.

b) Explique cuáles son las condiciones que debe satisfacer un predicado binario para ser una **relación de orden fuerte**.

c) Explique que las operaciones que realizan (semántica) las dos versiones de **`erase`** para conjuntos.

```
1 void erase(iterator p);
2 int erase(key_t x);
```

Explicar que son los valores de retorno, y porqué una retorna un entero y la otra no.

d) ¿Cuál es el tiempo de ejecución de **`find(x)`** para conjunto por árbol binario de búsqueda (mejor/promedio/peor)? Justifique.

e) ¿Porqué el árbol binario de una códificación binaria para compresión debe ser un árbol binario lleno (FBT)? (Recordemos que un FBT es áquel cuyos hijos son hojas o nodos interiores con sus dos hijos.)

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2014-11-21]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 3 y 6) y 50 % en las restantes secciones.
[ATENCIÓN 2] Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo CLAS1 en una o más hojas, PROG1 en una o más hojas separadas, etc...
[ATENCIÓN 3] Encabezar las hojas con **sección, Nro de hoja, apellido, y nombre, ASI**:

CLAS1, Hoja #2/3 LOVELACE, ADA

[Ej. 1] [CLAS1 (W=30pt)]
Recordar que **deben usar la interfaz STL**.

a) **[list (10pt)]** Escribir la implementación en **C++** del TAD lista (clase **`list`**) implementado por punteros **ó** cursores. Los métodos a implementar son **`insert(p,x)`**, **`erase(p)`**, **`iterator::operator++(int)`**.

b) **[stack-queue (10pt)]** Escribir la implementación en **C++** de los métodos **push**, **pop**, **`front`**, y **top** de los TAD pila y cola (clases **`stack`** y **queue**), según corresponda.

c) **[map (10pt)]** Escribir la implementación en **C++** del TAD correspondencia (clase **`map`**) implementado con **vectores ordenados**. Métodos a implementar: **`find(key)`** (en caso de utilizar la función privada **`lower_bound()`** también implementarla).

[Ej. 2] [PROG1 (W=50pt)]

a) **[qsortl (20pt)]**
Escribir una función **`int qsortl(list<int> &L, bool (*comp)(int,int));`** que implementa el algoritmo de ordenamiento **quick-sort** para listas de la siguiente manera

- Toma como pivote **`v`** el primer elemento de la lista.
- Separa **`L`** en tres listas: **`L1,L2,L3`** los estrictamente menores a **`v`**, equivalentes a **`v`**, y mayores a **`v`**.
- Aplica recursivamente **`qsortl()`** a **`L1`** y **`L3`**.
- Concatena **`L1,L2,L3`** en **`L`**, en ese orden.

Nota: El concepto de particionamiento es el mismo que el que hemos visto en el capítulo de ordenamiento para vectores, sin embargo acá como es una lista no hace falta hacer intercambios de elementos, simplemente se van tomando los elementos de la lista y se van insertando en la lista auxiliar apropiada.

Restricciones: Usar sólo listas. Hacer la operación in place (usar sólo $n + O(1)$ celdas), donde n es el tamaño de la lista original.

b) **[apply-map (15pt)]**
Escribir una funcion **`void apply_map(list<int> &L, map<int,int> &M, list<int> &ML)`** que, dada una lista **`L`** y una correspondencia **`M`** retorna por **`ML`** una lista con los resultados de aplicar **`M`** a los elementos de **`L`**. Si algún elemento de **`L`** no está en el dominio de **`M`** entonces el elemento correspondiente de **`ML`** no es incluido. Por ejemplo, si

`L= (1,2,3,4,5,6,7,1,2,3)`, y `M = { (1,2),(2,3),(3,4),(4,5),(7,8) }` entonces después de hacer `apply_map(L,M,ML)`, debe quedar `ML = (2,3,4,5,8,2,3,4)`

Restricciones: No usar estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser $O(n)$, donde n es el número de elementos en la lista (asumiendo que las operaciones usadas de correspondencia son $O(1)$).

c) **[gather (15pt)]**
Escribir una función `void gather(list<int> &L, int M);` que, dada una lista de enteros `L`, agrupa elementos consecutivos de tal manera que en la lista queden solo elementos mayores o iguales que `M`. El algoritmo recorre la lista y, cuando encuentra un elemento menor, empieza a agrupar el elemento con los siguientes hasta llegar a `M` o hasta que se acabe la lista. Por ejemplo, si `L=[3,4,2,4,1,4,4,3,2,2,4,1,4,1,4,4,1,4,4,2]`, entonces `gather(L,10);` da `L=[13,12,13,10,10]`. En la lista final NO deben quedar elementos menores que `M` salvo eventualmente el último elemento.
Restricciones: Usar sólo listas y hacerlo *in-place* es decir usar sólo $n + O(1)$ celdas.

[Ej. 3] [PREG1 (W=20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$
\begin{aligned}
T_1 &= n^2 + 5 \cdot 3^n + 4^{10}, \\
T_2 &= 2 \cdot 10^n + \sqrt{3} \cdot n + \log_8 n, \\
T_3 &= 1000 + 3 \log_4 n + 8^2 + 5 \log_{10} n, \\
T_4 &= 5 \cdot 2^n + 2n^3 + 3\, n! \\
T_5 &= 2.3 \log_8 n + \sqrt{n} + 5\, n^2 + 2\, n^5,
\end{aligned}
$$

b) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase `list<>` implementada con **celdas simplemente enlazadas** (por punteros o cursores) en el caso promedio? *Métodos:* `insert(p,x)`, `*p`, `erase(p)`, `clear()`, `begin()`, `end()`.

c) Explique la propiedad de **transitividad** de la notación asintótica $O(\,\cdot\,)$. De un ejemplo.

d) ¿Qué ocurre si ejecutamos el siguiente código? `S` es una pila de enteros que puede estar vacía o no.

```
S.push(3); S.push(5); S.top(); S.top(); x=S.top();
```

¿Puede dar un error porque `S` se queda vacía? ¿Que valor toma `x`?

e) Discuta ventajas y desventajas de usar contenedores lineales **ordenados** o **desordenados** para representar correspondencias.

[Ej. 4] [CLAS2 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[aoo (5pt)]**
Arbol Ordenado Orientado: Implementar (haciendo uso de la interfaz STL) los métodos

- `iterator lchild();`
- `iterator right ();`

- `iterator insert (iterator p, T t);`

considerando que se debe escribir la clase **`iterator`** (en el scope que corresponda) además de aquellas estructuras sobre las cuales esta se construye (a saber, la clase **`cell`**).

b) **[hash-dict (5pt)]**
Diccionario por tablas de dispersión abierta: Implementar la clase **`iterator_t`** en conjunto con aquello que considere pertinente dentro de la parte privada de la clase **`hash_set`**. Finalmente, implemente el método **`iterator_t find (key_t & x)`**

c) **[vbitset (5pt)]**
Conjuntos por vectores de bits: Implementar los métodos

- `pair_t insert (elem_t x);`
- `elem_t retrieve (iterator_t p);`
- `void set_difference (set & A, set & B, set & C);`

para ello deberá además implementar la parte privada de la clase.

d) **[abb (5pt)]**
Conjuntos por árbol binario de búsqueda: Implementar (haciendo uso de la interfaz STL) el método

- `void erase (iterator_m)`

definiendo para ello (y respetando el anidamiento pertinente) la clase iterator.

[Ej. 5] [OPER2 (W=20pt)]

a) **[rec-arbol (2.5pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- **`ORD-PRE=(Q,R,S,U,V,W,X,Y,T)`**,
- **`ORD-POST=(R,V,W,Y,X,U,S,T,Q)`**,

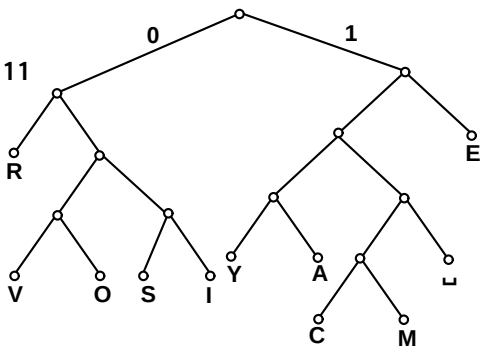
b) **[part-arbol (2.5pt)]** Dado el árbol ordenado orientado (AOO): **`(Z (Q T) (R W) (Y (Q A C)))`** determinar cuales son los nodos **antecesores, descendientes, izquierda** y **derecha** del nodo **`R`**. ¿Son **disjuntos**? Justifique.

c) **[huffman (2.5pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`LOLLAPALOOZA`**, $P(L) = 0.10$, $P(O) = 0.10$, $P(A) = 0.10$, $P(P) = 0.30$, $P(Z) = 0.10$, $P(Q) = 0.10$, $P(M) = 0.20$.

Calcular la longitud promedio del código obtenido.

d) **[hf-decode (2.5pt)]**
Utilizando el código de la derecha desencodar el mensaje

`01101010010010010001011101010101000011111`



e) **[abb (2.5pt)]** Dados los enteros $\{16, 10, 23, 5, 6, 13, 8, 7, 6, 15, 4\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 16, 10, y 5 en ese orden.

f) **[hash-dict (2.5pt)]** Insertar los números $\{9, 15, 25, 8, 7, 35, 17, 4\}$ en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión `h(x) = x`.

g) **[heap-sort (2.5pt)]** Dados los enteros $\{1, 10, 8, 2, 3, 13, 5\}$ ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

h) **[quick-sort (2.5pts)]** Dados los enteros $\{10, 4, 11, 6, 3, 5, 12, 4, 9, 7, 6, 2\}$ ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 6] [PREG2 (W=20pt, 4pt por pregunta)]

a) Si queremos generar un código binario de igual longitud para un conjunto de 26 caracteres. ¿Cuantos bits tendrá, como **mínimo**, cada caracter?

b) ¿Como se define la **altura** de un nodo en un árbol? ¿Cuál es la altura de los nodos `A`, `B`, y `C` en `(W X Y (A B (C (D  F G H))))`?

c) ¿Cómo se define la clase **iterator** para árboles binarios? Enumere sus campos dato y discuta la utilidad de cada uno de ellos. De ejemplos.

d) ¿Cómo se define en forma recursiva el **orden posterior** para un árbol ordenado orientado (AOO)?

e) ¿Cuál es el resultado de aplicar la función `l=particiona(w,j,k,v)`? (**Nota:** No se pide el algoritmo, sino cuál es el efecto de aplicar tal función, independientemende de como se programe). Explique los argumentos de la función. ¿Cuál debe ser el tiempo de ejecución para `particiona()` si queremos que `quick_sort()` sea un algoritmo **rápido**?

Algoritmos y Estructuras de Datos.
TPL1. Trábajo Práctico de Laboratorio 1. [2014-08-30]

PASSWD PARA EL ZIP: `IBALB792CRZ1`

Ejercicios

[Ej. 1] **[large-even-list]** Dado un `vector<list<int>> &VL` buscar aquella lista `VL[j]` que contiene la máxima cantidad de pares y retornar la sublista de pares correspondientes (en el mismo orden que están en `VL[j]`). Por ejemplo, si

```
VL[0]: 0 1 2 3 4 5 7
VL[1]: 0 1 2 3
VL[2]: 2 2 2 1 0
```

entonces vemos que las listas contienen 3, 2, y 4 elementos pares. De tal forma que la función debe retornar los pares de la última `VL[2]`, es decir `(2 2 2 0)`.

Consigna: Escribir la función
`void large_even_list(vector< list<int> >&VL,list<int>&L);`

Ayuda: Mantener una lista `Lmax` que tiene la lista con los elementos pares del máximo actual. Para `VL[j]` calcular la lista `tmp` de los elementos pares correspondientes y si su longitud es mayor que la de `Lmax` reemplazarla.

```
Lmax = lista vacia;
for (Lj in VL) {
   tmp = lista con elementos pares de Lj;
   if (tamaño de tmp>tamaño de Lmax) Lmax = tmp
}
```

[Ej. 2] **[interlaced-split]** Dada una lista de enteros `L` y un entero positivo `m` dividir a `L` en `m` sublistas (en un vector de listas `vector< list<int> > VL`) en forma **entrelazada** es decir $a_0, a_1, a_2 ...$ van correspondientemente a las listas `VL[0]`, `VL[1]`, `VL[m-1]`, `VL[0]`, `VL[1]`... Es decir, el elemento a_j va a la lista `VL[k]` donde `k=j%m`.

Consigna: Escribir la función
`void interlaced_split(list<int>&L, int m,vector< list<int> >&VL);`

Por ejemplo, si `L=(0 1 2 3 4 5 6 7 8)` y `m=4` entonces debemos tener

```
VL[0]: 0 4 8
VL[1]: 1 5
VL[2]: 2 6
VL[3]: 3 7
```

[Ej. 3] **[interlaced-join]** Es la inversa de `interlaced_split()`. Dado un `vector< list<int> > VL` juntarlos en una lista `L` de uno a la vez. Es decir, primero los primeros elementos de `VL[0]`, `VL[1]`, `...`, `VL[m-1]`, después los segundos elementos, hasta que se acaben todas las listas.

Consigna: Escribir la función
`void interlaced_join(vector< list<int> >&VL, list<int>&L);`

Ejemplo 1: si `VL` es

```
VL[0]: 0 4 8
VL[1]: 1 5
VL[2]: 2 6
VL[3]: 3 7
```

entonces debe dar **`L=(0 1 2 3 4 5 6 7 8)`** (la inversa del ejemplo presentado en la definición de **`interlaced_split()`**.

Nota: Si **`VL`** es el resultado de un **`interlaced_split()`** entonces las diferentes listas de **`VL`** difieren en longitud en a lo sumo 1, y las primeras son más largas que las últimas. En el ejemplo anterior la primera **`VL[0]`** tiene 3 elementos y las demás tienen 2. Pero en general no podemos asumir que esto sea así, por ejemplo

```
VL[0]: 0 1
VL[1]: 2
VL[2]: 3 4
VL[3]: 5 6
VL[4]: 7
```

En cuyo caso el resultado esperado es **`L=(0 2 3 5 7 1 4 6)`**.

Ayuda: Utilizar el siguiente seudocódigo

```
L = lista vacía;
while (1) {
  if (todas las listas están vacías) break;
  for j in [0,N) {
     if (VL[j] no es vacía) {
       sacar el primer elemento de VL[j] e insertarlo al final de L;
     }
  }
}
```

Nota: Como mencionamos, la composición de **`interlaced_split()`** con **`interlaced_join()`** (en ese orden) es la identidad, es decir:

```
interlaced_split(L1,m,VL);
interlaced_join(VL,L2);
```

da siempre **`L1==l2`**. Mientras que

```
interlaced_join(VL1,L);
m = VL1.size();
interlaced_split(L,m,VL2);
```

no necesariamente da **`VL1==VL2`**. Esto sólo ocurre si las longitudes de los **`VL[j]`** difieren entre sí en a lo sumo 1, y son decrecientes, es decir **`VL[j+1]<=VL[j]`**.

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Funciones utilitarias que pueden servir

  ```
  void dump(list<int> &L,string s="");
  void dump(vector< list<int> > &VL,string s="");
  ```

  Imprimen por pantalla los contenedores. Ambas son métodos en el namespace **`aed`**, de manera que hay que usarlos así:

  ```
  aed::dump(VL);
  aed::dump(L);
  ```

- El TPL se autoevalúa, cada una de las funciones **`Eval::eval1()`** a **`eval3()`** reporta una serie de diagnósticos. Deben dar todos **`OK`**, por ejemplo

  ```
  Evaluando ejercicio 3
  EJ3|Caso0. Estado: OK
  EJ3|Caso1. Estado: OK
  EJ3|Caso2. Estado: OK
  EJ3|Caso3. Estado: OK
  ESTADO EJ3: total 4, ok 4, mal 0. Todos bien? OK
  ```

- Las funciones **`eval()`** tienen un segundo argumento **`int vrbs`** que es la **verbosidad** con la cual queremos que la clase evaluadora reporte errores o información adicional. Una primera forma de debug es poner el nivel máximo de verbosidad **`vrbs=2`**.
- Para cada ejercicio hay un segundo control que llamamos **seed**/**hash**, la función es del tipo
  `eval1r(function f,int X,int vrbs);`
  donde **`f`** es la función del usuario a evaluar, **`X`** una semilla y **`vrbs`** la verbosidad.

  La clase evaluadora genera una serie de contenedores a partir de la semilla, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`X=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la clase evaluadora con un valor determinado de la semilla **`X`** y se comprobará que genere el valor correcto del checksum **`H`**.

  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`X`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.

mstorti@galileo/aed-2.0.5-67-gcc0cf57/Tue Sep 23 12:39:24 2014 -0300

Algoritmos y Estructuras de Datos.
RTPL1. Recup Trábajo Práctico de Laboratorio 1. [2014-09-23]

Ejercicios

[Ej. 1] **[expand]** Escribir una función `void expand(list<int>  &L,int m);` que transforma los elementos de una lista `L` de tal forma que todos los elementos de `L` resulten ser menores o igual que `m`, pero de tal forma que su suma se mantenga inalterada. por ejemplo si `m=3` podemos dividir a `10` en `3,3,3,1`. Es decir si `L=(7,2,3,1,4,5)`, entonces despues de hacer `expand(L,2)` debe quedar `L=(2,2,2,1,2,2,1,1,2,2,2,2,1)`.

[Ej. 2] **[ascendente]** Escribir una función `int ascendente(list <int> &L, list<list<int> > &LL)` que, dada una lista `L`, genera una lista de listas `LL` de tal forma de que cada sublista es ascendente. Por ejemplo, si la lista es `L=(0,5,6,9,4,3,9,6,5,5,2,3,7)`, entonces hay 6 corridas ascendentes, a saber: `(0,5,6,9)`, `(4)`, `(3,9)`, `(6)`, `(5,5)` y `(2,3,7)`. Por lo tanto la función debe retornar `LL=((0,5,6,9,4,3,9,6,5,5,2,3,7), (0,5,6,9), (4), (3,9), (5,5), (2,3,7))`.

[Ej. 3] **[deja1solo]** Escribir una función `void deja1solo(list<int> &L);` de tal forma que. si vamos dividiendo a `L` en rangos de pares e impares consecutivos, deja el primero de cada rango, por ejemplo si `L=(5 4 5 3 4 9 10 4 9 3 2 10 3 7 5 3)` entonces debe dejar `L=(5 4 5 4 9 10 4 9 3 2 3)` Para `L=(10 10 1 9 7 4 4 10 7 1 1 4 10 1 1 7)` debe dejar `L=(10 1 4 7 4 1)`. El algoritmo debe ser in-place, sólo debe borrar elementos.

mstorti@galileo/aed-2.0.5-67-gcc0cf57/Tue Sep 23 12:39:33 2014 -0300

Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio 2. [2014-10-04]

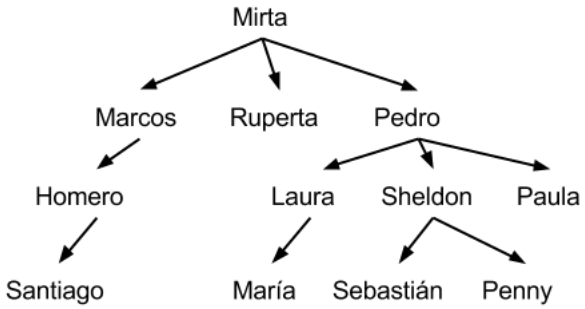
Ejercicios

[Ej. 1] **[common-relative]:** Durante la juntada que hicieron Santi y Seba para comer el asado, comenzaron a preguntarse sobre sus familias y de quienes eran descendientes. Fue por esto que se les ocurrió hacer a cada uno su árbol genealógico. Para ello buscaron en Internet a sus parientes e hicieron numerosas llamadas a familiares y a cada persona que lleva su apellido en la guía telefónica (más de uno le cortó la llamada porque pensaron que era una encuesta de una importante empresa nacional de telefonía). Una vez que reunieron suficiente información y vieron el árbol genealógico que formaron, quedaron estupefactos al ver cuán grande les quedó.

Cansados por toda la tarea de inteligencia, nuevamente recurren a usted para que implemente una función

```
string common_relative( tree <string> &T, string name1, string name2);
```

que dados un árbol **T** y dos personas que pertenecen al mismo (**name1** y **name2**), retorne el ancestro más próximo que tengan en común.



En este ejemplo, si preguntamos cual representa el pariente en común más cercano entre María y Sheldon, éste será Pedrito. Por otro lado, si nos hacemos esta misma pregunta pero con Santiago y Laura, su respuesta deberá ser Mirta.

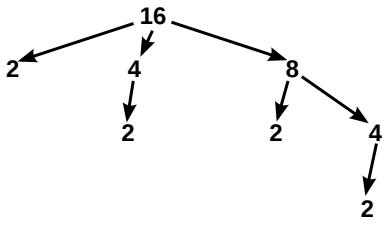
Si la pregunta es entre Santiago y Homero, la respuesta debe ser Homero.

Ayuda: Para cada uno de los nombres **name1** y **name2** encontrar el camino que va desde la raíz al nombre, y luego recorrer los caminos hasta el primer nodo cuyo valor es diferente. Por ejemplo en el caso anterior el camino que va a María es **L1=(Mirta,Pedro,Laura,María)** mientras que el que va a Sheldon es **L1=(Mirta,Pedro,Sheldon)** por lo tanto las recorriendo las listas vemos que el único camino en comun es Mirta. Para el par Santiagp, Homero en cambio los caminos son **L1=(Mirta,Marcos,Homero,Santiago)** y **L1=(Mirta,Marcos,Homero)**, por lo tanto el último antecesor común es Homero.

[Ej. 2] **[div-tree]** Dado un número natural **n**, se busca generar un árbol con todos sus divisores como hijos (sin incluir los divisores triviales 1 y **n**). A su vez, cada uno de sus hijos deberán tener como descendientes a sus correspondientes divisores. Y así sucesivamente hasta que éstos resulten ser números primos. Los números que sean primos, representarán hojas en el árbol. (Recordar que los números primos sólo tienen como divisores al número 1 y a sí mismos.)

La función a implementar será void **div_tree(tree<int> &T, int N);**

Por ejemplo, si a la función se le pasa como argumento al número 16, deberá retornar en **T** el siguiente árbol:



Restricciones: Los hijos deberán encontrarse en orden creciente.

Ayuda: Para obtener los divisores de un número **N**, basta con utilizar un contador **I** que recorra desde el número 2 hasta **N/2** inclusive. Si se verifica que (**N% I == 0**), entonces **I** es divisor de **N**.

[Ej. 3] [bordering-countries]:

Un licenciado en geografía tiene dos estructuras de tipo correspondencias. Una de ellas posee como claves a diferentes países, y como valor, las ciudades que tienen cada uno de ellos.

La otra estructura tiene como ciudades como claves, y como valor a las ciudades vecinas.

A este entusiasta licenciado se le ocurrió preguntarse, para un determinado país, qué países limítrofes tiene, en base a las ciudades que tiene conectadas.

El licenciado nos dió un ejemplo de posibles correspondencias:

```
countries["Francia"] = {"Calais", "París", "Perpignan"};
countries["Gran Bretaña"] = {"Dover", "Londres"};
countries["España"] = { "Barcelona", "Madrid"};

neighbours["Paris"]={"Lyon", "Calais"}
neighbours["Calais"]={"Lyon", "Ginebra","París"}
neighbours["Perpignan"]={"París", "Barcelona"}
neighbours["Dover"]={"Londres"}
neighbours["Londres"]={"Dover"}
neighbours["Barcelona"]={"Madrid", "Perpignan"}
neighbours["Madrid"]={"Barcelona"}
```

Entonces para el ejemplo, si queremos saber ¿qué países limitan con Francia?, la respuesta será **{"España"}**.

Su trabajo consiste en definir una función

```
list<string> bordering_countries(map<string,list<string> >
   &countries, map<string,list<string> > &neighbours, string   country);
```

que retorne una lista con los países limítrofes al país **country**. La lista deberá estar ordenada alfabéticamente y no deberá tener elementos repetidos.

[Ej. 4] [is-palindrome]: Dada una pila **S** determinar si es un **palíndromo**, es decir si al invertir los elementos queda igual.

```
bool is_palindrome(stack<int> &S);
```

Por ejemplo, para **P1=(0 1 2 3 2 1 0)** debe retornar **true** y para **P2=(0 1 2 3 4 3)** debe retornar **false**.

Restricciones: solo se pueden usar pilas auxiliares y no se debe usar espacio adicional, es decir la suma de todas las longitudes de las pilas debe ser $n + O(1)$. Puede ser destructiva.

Ayuda: Pasar los elementos de **S** a una pila auxiliar **S2** y contarlos. Sabiendo el numero de elementos **n**, pasar **[n/2]** elementos de **S2** a **S**. Si **n** es impar sacar un elemento de **S2**. Ahora **S** y **S2** deben ser iguales. Ahora se debe verificar si **S** es igual a **S2**.

En el ejemplo anterior **P1** debe quedar **S=S2=(2 1 0)**, mientras que para **P2** debe quedar **S=(3 4 3)**, y **S2=(2 1 0)**.

Instrucciones

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header **`tree.h`**.

mstorti@galileo/aed-2.0.5-79-gd0712f5/Thu Oct 2 22:39:25 2014 -0300

Algoritmos y Estructuras de Datos.
TPL2R. Recup Trabajo Práctico de Laboratorio 2. [2014-10-16]

PASSWD PARA EL ZIP: `NPZ PZ4 ESB V51`

Ejercicios

[Ej. 1] **[mkmtree]** Escribir una función `void mkmtree(tree<int> &T,int m,int k);` que dados dos enteros positivos `m`, `k`, (con `0<k<m`) construye un **árbol ordenado orientado (AOO)** `T` de la siguiente forma:

- En la raíz tiene el valor `m`
- Los hijos de `m` son: `m-k`,..., `m-j*k`, mientras los elementos sean **no negativos** `m-j*k>=0`.
- La definición se propaga **recursivamente**, es decir si un nodo `n` tiene el valor `*n`, los hijos son `(*n)-k`,..., `(*n)-j*k`,... mientras `(*n)-j*k>=0`.

Ejemplos:

- `m=10,k=3, -> T=(10 (7 (4 1) 1) (4 1) 1)`
- `m=10,k=2, -> T=(10 (8 (6 (4 2) 2) (4 2) 2) (6 (4 2) 2) (4 2) 2)`
- `m=20,k=7, -> T=(20 (13 6) 6)`
- `m=23,k=6, -> T=(23 (17 (11 5) 5) (11 5) 5)`

[Ej. 2] **[has-equal-path]** Dado un arbol ordenado orientado (AOO) `T` escribir una funcion predicado que determina si contiene un camino de valores iguales que va desde la raíz a una hoja con todos elementos iguales. Por ejemplo

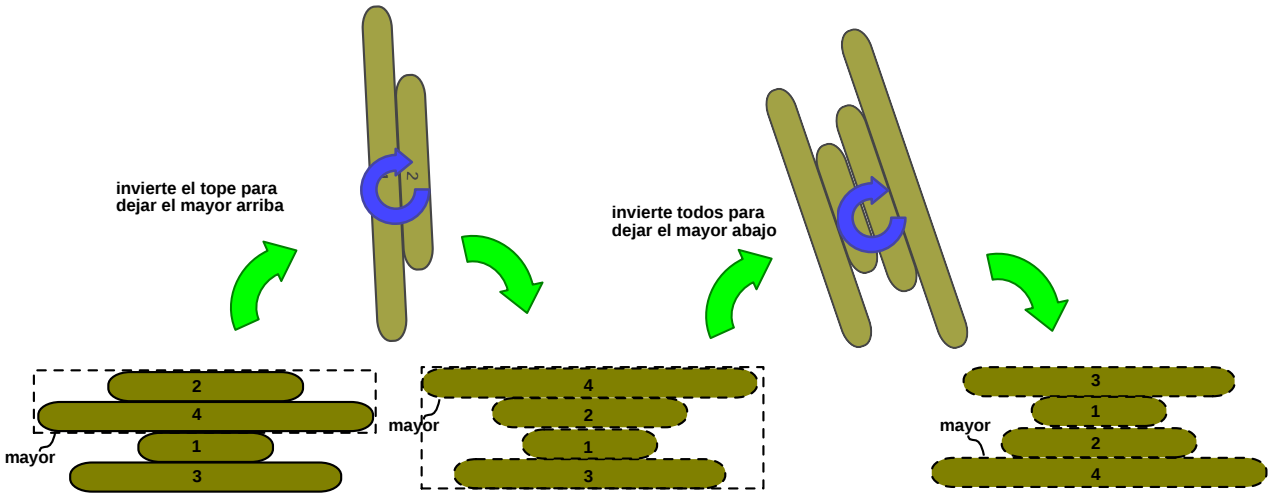
- `T=(1 (1 2 3) (1 2 4)) -> false`
- `T=(1 (1 2 3) (1 2 1)) -> true` (el camino es `(1,1,1)`)
- `T=(6 (6 2 6) (1 2 4)) -> true` (el camino es `(6,6,6)`)
- `T=(1 (2 2 1) (1 2 4)) -> false`

Consigna: Escribir una función `bool has_equal_path(tree<int> &T);` que realiza la tarea indicada.

Ayuda: Escribir una función recursiva
`bool has_equal_path(tree<int> &T,tree<int>::iterator n);` que

- Si `n` es Λ o una hoja retorna `true`.
- Caso contrario recorre los hijos `c` de `n` y retorna `true` sí y sólo si alguno de ellos es tal que su valor al del padre (es decir `*c==*n`) y tiene un camino de valores iguales a una hoja (**recursividad**) .

[Ej. 3] **[pancake-sort]:** Dada una pila de números `stack<int> &S`, implementar el algoritmo **Pancake Sort** para ordenar la misma. El algoritmo se basa en buscar el elemento mayor de la pila y colocarlo al fondo de la misma. Luego buscar el segundo mayor elemento y colocarlo en el fondo-1 de la pila. Y así sucesivamente hasta que quede completamente ordenada.



Sin embargo, hay una restricción para efectuar estas operaciones. Sólo se pueden realizar operaciones en las cuales se **invierte** una cierta cantidad de elementos en el tope de la pila, como si fuera una pila de panqueques (de ahí el nombre del algoritmo). Al encontrar el elemento mayor, se debe invertir toda la pila hasta la posición en que se encuentra el elemento mayor, y luego volver a invertir toda la pila hasta que el elemento mayor quede en el fondo de la pila. Luego repetir estos pasos con el resto de los elementos.

Consigna: Escribir la función `void pancake_sort(stack<int> &S);` que debe retornar por referencia, la pila ordenada.

Restricciones: Solo se deben utilizar como estructuras de datos auxiliares, pilas o colas.

Ayuda: Considerar el siguiente algoritmo:

Comenzar desde el tamaño actual de la pila e ir decrementando en 1 hasta llegar a 1. Llamar a este tamaño `POSITION`, y luego hacer lo siguiente para cada uno de los valores de `POSITION`:

- Encontrar el índice al máximo elemento `MAX_ELEM` en la pila desde el tope de la pila hasta la posición `POSITION`.
- Llamar a una función `FLIP(stack,i)` que se encargue de invertir la pila hasta la posición `i`-ésima con `i=MAX_ELEM`.
- Llamar a `FLIP(stack,i)` con `i=POSITION`.

[Ej. 4] **[count-cycles]** Dado un `map<int,int> &M` que representa una **permutación** (es decir tal que el conjunto de las claves es igual al conjunto de las imágenes) se deben contar cuantos **ciclos** tiene presente. Un ciclo se corresponde con una secuencia de pares `M[clave] = valor` tal que `M[X1] = X2`, `M[X2] = X3`, `M[X3] = ... , M[...] = X1`. Se debe considerar que `M[X1] = X1` es un ciclo.

Ejemplo:

Suponiendo el mapa siguiente:

```
M[1] = 2;
M[2] = 4;
M[3] = 6;
M[4] = 9;
M[6] = 3;
M[9] = 1;
M[10] = 10;
```

La función retornará 3. Los ciclos son

- `1->2->4->9->1`,

- `3->6->3`,
- `10->10`,

Consigna: Escribir la función `int count_cycles(map<int,int> &M);` que debe retornar la cantidad de ciclos que posee el mapa.

Restricción: los ciclos se deben contar una sola vez, ya que por ejemplo, el ciclo `1->2->4->9->1` es el mismo que `2->4->9->1->2` o `4->9->1->2->4`.

Ayuda:

- A partir de cualquier clave de la correspondencia `kini` hacer `k=kini` y aplicar repetidamente `k=M[k]` hasta volver a `k==kini`. Por ser una permutación está garantizado que a lo sumo en `M.size()` iteraciones esto va a ocurrir.
- Ir **eliminando** los pares de asignación por los cuales se va pasando, para no contar los ciclos dos veces (el algoritmo es **destructivo**).
- Mantener un **contador** de los ciclos.
- Prestar especial atención a los casos de ciclos de **un sólo elemento** (`k==M[k]`)

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header `tree.h`.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.evalj(f,vrbs);
h1 = ev.evaljr(f,seed); // para SEED=123 debe dar H1=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval1` y `eval1r`). La primera `ev.evalj(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- La segunda función `evaljr` es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla `seed`, se los pasa a la función del usuario `f()`. Las respuestas de la `f()` van siendo procesadas por la función interna de hash que genera un **checksum** `H` de las respuestas. Por ejemplo para el primer ejercicio si `X=123` entonces el checksum es `H=523`. Una vez que el alumno termina su tarea se le pedirá que corra la clase evaluadora con un valor determinado de la semilla `X` y se comprobará que genere el valor correcto del checksum `H`.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro `X` con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de `H`.
- En la clase evaluadora cuentan con las siguientes funciones utilitarias:
  - `void dump(map<int,int> &M,string s="")`: Imprime un mapa entero/entero. Nota: Es un método de la clase `Eval` es decir que hay que hacer `Eval ev; ev.dump(M);`. El string `s` es un label opcional.

- **`void dump(stack<int> S,string s="")`**: Imprime una lista de enteros (primero el tope). Uso: **`Eval ev; ev.dump(S);`**
- **`void dump(queue<int> S,string s="")`**: igual que **`dump(stack<int>)`**
- **`tree<int>::lisp_print()`**: Lisp print de un árbol. Nota: esta pertenece a la clase **`tree`**. Uso: **`tree<int> T; T.lisp_print();`**

Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio 3. [2014-11-01]

PASSWD PARA EL ZIP: `X35 926 RWF NG9`

Ejercicios

[Ej. 1] **[bijsubset]** Dado un conjunto `S` y una función de **mapeo** `T(*f)(T)` determinar el conjunto `S1` de elementos `x` de `S1` tales que no existe otro elemento `z` con `f(x)=f(z)`. Es decir, para los elementos de `S1` y su imagen, la relación es biyectiva.

Por ejemplo si `S={-1,1,-2,2,3,4,5}` y `f(x)=x*x` entonces debe ser `S1={3,4,5}` ya que los elementos `(-1,1)`, y `(-2,2)` los elementos de la imagen son los mismos.

Consigna: Escribir una función
`void bijsubset(set<int> &S, map_fun_t f, set<int> &S1);` que realiza la tarea indicada.
Donde
`typedef int(*map_fun_t)(int);`

Ayuda: Recorrer los elementos `x` de `S`, aplicarles la función `y=f(x)` y almacenarlos `y` en un conjunto `Y`. Al mismo tiempo si se detecta que el elemento `y` ya estaba en `Y` almacenarlo en otro conjunto `Ydup`. Por ejemplo en el caso anterior deberíamos terminar con `Y={1,4,16,25}` e `Ydup={1,4}`. Ahora volver a recorrer los elementos de `S` e insertarlos en `S1` sólo si su imagen no está en `Ydup`.

[Ej. 2] **[preimage]** Dados un `vector<set<int> > VX` y un `set<int> Y`, y una función de mapeo `T(*f)(T)` encontrar el conjunto de `VX[j]` **preimagen** de `Y` tal que si se le aplica `f` a sus elementos, se obtiene `Y`.

Por ejemplo si `VX=[{2,3,4},{-1,1,2},{5,6,7},{-1,-2}]` y `Y={1,4}`, y `f(x)=x*x` entonces debe retornar `true`, y `preY=VX[1]`.

Consigna: escribir la función
`bool preimage(vector< set<int> >&VX,map_fun_t f, set<int> &Y,set<int> &preY);`
Donde:
`typedef int(*map_fun_t)(int);`

Nota: Si hay varios `VX[j]` que satisfacen la condición entonces debe retornar el primero. En el ejemplo `{-1,-2}` tambien es preimagen de `{1,4}`, pero `{-1,1,2}` está primero.

Nota 2: Si no hay ninguno que sea la preimagen entonces debe retornar `false` y `preY={}`.

Ayuda: Escribir una función
`bool ismapped(set<int> &X,map_fun_t f,set<int> &Y)` que determina si `X` es la preimagen de `Y` por `f`. Luego recorrer los elementos de `VX` y aplicarles `ismapped()`.

[Ej. 3] **[is-recurrent-tree]** Dado un árbol binario lleno (todos sus nodos interiores tienen los dos hijos), verificar si es un "árbol recurrente". Un árbol binario se considera recurrente si cada uno de sus nodos interiores es la suma de sus 2 nodos hijos.

Por ejemplo

- `T = (5 (2 1 1) (3 1 (2 1 1))) -> true`
- `T = (5 (3 (2 1 1) 1) (3 1 (2 1 1))) -> false` (la raíz no es la suma de sus hijos)
- `T = (5 (2 1 1) (3 1 (1 1 1))) -> false` (el nodo con valor 3 no es la suma de sus hijos, al igual que uno de sus hijos tampoco es la suma de sus hojas).

Además lo generalizaremos en el estilo de **programación funcional**, reemplazando la suma por cualquier función binaria asociativa, de la siguiente forma:

Consigna: Escribir la función
`bool is_recurrent_tree(btree< int > &T,assoc_fun_t g);` donde:
`typedef int(*assoc_fun_t)(int, int);` que debe retornar `true` en caso que el árbol binario `T` sea recurrente (con respecto a `g(.,.)`), y false en caso contrario.

[Ej. 4] **[ordered-tree]** Dado un vector de números enteros positivos, se deberá armar un árbol binario de manera que la posición i-ésima del vector verifica si es mayor que la raíz. En caso que sea mayor, intentará insertarse en el subárbol derecho de la misma, y sino en el subárbol izquierdo. Si el hijo correspondiente está vacío entonces lo inserta allí, si no lo compara con el elemento, y vuelve a aplicar la regla recursivamente.

Consigna: Escribir la función
`void ordered_tree(vector<int> &V, btree< int > &T);` que debe retornar por referencia el árbol binario `T` que cumpla con el enunciado.

Ejemplos:

`V = [1 5 10 8 7 2] -> T = (1 . (5 2 (10 (8 7 .) .)))`

Se puede ver que el primer elemento del vector será la raíz del árbol. Luego el siguiente elemento, al ser mayor que la raíz se inserta a la derecha debido a que no existía este hijo derecho previamente. Después el siguiente elemento correspondiente al 10, es mayor que la raíz, pero como existe este hijo derecho, ahora comprueba si es mayor o menor que 5 y al ser mayor, inserta a la derecha nuevamente.

`V = [5 4 2 6 8 7 3] T = (5 (4 (2 . 3) .) (6 . (8 7 .)))`

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header `tree.h`.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.evalj(f,vrbs);
h1 = ev.evaljr(f,seed); // para SEED=123 debe dar H1=170
```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval1` y `eval1r`). La primera `ev.evalj(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- La segunda función `evaljr` es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla `seed`, se los pasa a la función del usuario `f()`. Las respuestas de la `f()` van siendo procesadas por la función interna de hash que genera un **checksum** `H` de las respuestas. Por ejemplo para el primer ejercicio si `seed=123` entonces el checksum es `H=523`. Una vez que el alumno termina su tarea se le pedirá que corra la clase evaluadora con un valor determinado de la semilla `seed` y se comprobará que genere el valor correcto del checksum `H`.

Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.

- En la clase evaluadora cuentan con las siguientes funciones utilitarias:
  - **`void dump(vector<set<int> > &VX,string s="")`**: Imprime un mapa entero/entero. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval ev; ev.dump(VX);`**. El string **`s`** es un label opcional.
  - Análogamente está **`void dump(set<int> S,string s="")`**.
  - **`btree<int>::lisp_print()`**: Lisp print de un árbol. Nota: esta pertenece a la clase **`tree`**. Uso: **`btree<int> T; T.lisp_print();`**

Algoritmos y Estructuras de Datos.
TPL3R. Recuperatorio Trabajo Práctico de Laboratorio 3. [2014-11-06]

Ejercicios

[Ej. 1] **[only1]** Dado un vector de conjuntos **`VS`**, determinar el conjunto **`S1`** de todos aquellos elementos que están en uno y sólo uno de ellos. Por ejemplo, si
`VS=[{0,1,2,3},{2,3,4,5}]`, entonces **`S1={0,1,4,5}`**.

Consigna: Escribir una función **`void only1(vector<set<intd> > &VS,set<int> &S1);`** **Ayuda:** Recorrer los elementos de **`VS`**. Mantener un conjunto **`X`** que contiene todos los elementos de la union de los **`VS[j]`** y otro **`Xdup`** que contienen los elementos duplicados. Para esto realiza el siguiente seudocódigo

```
for S in VS {
  for x in S {
     si x esta en X agregarlo a Xdup;
     agregar x a X;
  }
}
```

Al final de este recorrido debemos tener en **`X`** todos los elementos de los **`VS[j]`** y en **`Xdup`** los duplicados. Sólo basta con hacer una operación de diferencia.

[Ej. 2] **[included]** Dada un vector de conjuntos **`vector< set<int> > VS`**, escribir una función predicado **`bool included(vector< set<int> > &VS)`**, que determina si los conjuntos del vector **`VS`** son subconjuntos propios en forma consecutiva. Es decir, si $S_j \subset S_{j+1}$ para $j = 0, ..., n-2$. (Recordar que $A \subset B$ indica inclusión propia, es decir $A \subseteq B$ y $A \neq B$.)

[Ej. 3] **[diffh]** Un árbol binario (AB) es balanceado si

- Es el árbol vacío ó,
- Sus subárboles derecho e izquierdo son balanceados, y sus alturas difieren a lo sumo en 1, o sea $|h_L - h_R| \leq 1$.

Por ejemplo **`(1 (2 (3 (4 5 6) 7) (13 8 9)) (15 10 (16 11 12)))`** es un árbol balanceado (notar que no necesariamente las profundidades de las hojas difieren en ± 1).
Consigna: Escribir una función **`bool diffh(btree<int> &T);`** que retorna **`true`** si el árbol está balanceado.
Ayuda: En la función auxiliar retornar el resultado de si el árbol es balanceado o no, pero además retornar (con un valor pasado por referencia, o con un **`pair`**) la altura del árbol correspondiente.

[Ej. 4] **[onechild]** Dado una árbol binario (AB) **`T`**, determinar cuantos nodos de **`T`** tienen exactamente un solo hijo (*single child count*).

Ejemplos:
Para **`T=(8 (7 9 2) (3 . (9 . 1)));`** debe retornar **`2`**
Para **`T=(2 8 (1 (2 . 6) .));`** debe retornar **`2`**
Para **`T=(4 (9 (2 6 .) .) 8);`** debe retornar **`2`**
Para **`T=(8 (6 (5 . (4 3 (4 . (9 . 6)))) .) 3);`** debe retornar **`4`**

Definición recursiva:

$$\texttt{onec}(n) = \begin{cases} 0; & \text{si } n = \Lambda, \\ s + \texttt{onec}(l) + \texttt{onec}(r); & \text{caso contrario,} \end{cases} \quad (1)$$

donde $s = 1$, si n tiene exactamente un sólo hijo y 0 en caso contrario. l, r son los hijos izquierdo y derecho de n.

Consigna: Escribir una función `int onechild(btree<int> &T);` que realiza la tarea descripta.

Curiosidad: Coincide con la cantidad de puntos que aparecen en notación Lisp, pero no recomendamos encarar esa estrategia.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.evalj(f,vrbs);
  h1 = ev.evaljr(f,seed); // para SEED=123 debe dar H1=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval1`** y **`eval1r`**). La primera **`ev.evalj(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- La segunda función **`evaljr`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con las siguientes funciones utilitarias:
  - **`void dump(vector<set<int> > &VX,string s="")`**: Imprime un mapa entero/entero. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval ev; ev.dump(VX);`**. El string **`s`** es un label opcional.
  - Análogamente está **`void dump(set<int> S,string s="")`**.
  - **`btree<int>::lisp_print()`**: Lisp print de un árbol. Nota: esta pertenece a la clase **`tree`**. Uso: **`btree<int> T; T.lisp_print();`**

mstorti@galileo/aed-2.0.6-11-g19662d4/Thu Nov 6 13:07:51 2014 -0300

Algoritmos y Estructuras de Datos.
TPLSR. Recuperatorio de Trabajo Práctico de Laboratorio. [2014-11-07]

PASSWD PARA EL ZIP: **FK5 FV4 AFF K53**

Ejercicios

[Ej. 1] **[isomorph]** Dos árboles binarios `B1`, `B2` son **isomorfos** si se pueden aplicar una serie de inversiones entre los hijos derechos e izquierdos de los nodos de `B2` de manera que quede un árbol semejante a `B1`, es decir que tiene la misma estructura. Por ejemplo `(1 . (2 3 4))` es isomorfo a `(1 (3 4 2) .)`. En particular si dos árboles son isomorfos la cantidad de nodos que tienen una profundidad y altura dadas es la misma. Por ejemplo en el caso anterior ambos árboles tienen 4 nodos, de los cuales dos a 2 a profundidad 2, 1 a profundidad 1 y por supuesto 1 a profundidad 0 (raíz).

Concretamente `B1` es isomorfo a `B2` si

- Ambos son Λ
- Los subárboles de los hijos izquierdos de `B1` y `B2` son isomorfos entre sí y los derechos también
  ó
  el subárbol del hijo izquierdo de `B1` es isomorfo al derecho de `B2` y el derecho de `B1` es isomorfo al izquierdo de `B2`.

Nota: Notar que sólo se mira la **estructura** del árbol, no los valores de los nodos.

Consigna: Escribir una función
`bool btisomorph(btree<int> &B1,btree<int> &B2);`

[Ej. 2] Dado un `set<int>` y un entero `M` determinar si existe un subconjunto de `S` que sume exactamenet `M`.

Consigna: Escribir la función
`bool has_sum(set<int> &S, int M);`

Ayuda:

- Si hay un elemento en `S` que es `M` ya está.
- Si todos los elementos de `S` son mayores que `M` ya está.
- Si no probar (recursivamente) **para cada uno** de los elementos `x` de `S` que son `x<M` para ver si `S-{x}` tiene un subconjunto `S'` con suma `M-x`. Si esto ocurre entonces `S'+{x}` tiene suma `M`.

[Ej. 3] Sean `L1`, `L2` de longitud `N` dos listas de enteros tales que `L2` es una permutación cíclica de `L1`, es decir `L2[k] = L1[(k+m)%N]` para un cierto entero `m<N`. Escribir una función
`int find_shift(list<int> &L1,list<int> &L2);` que retorna el shift `m`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header `tree.h`.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.evalj(f,vrbs);
hj = ev.evaljr(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval1` y `eval1r`). La primera `ev.evalj(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- La segunda función `evaljr` es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla `seed`, se los pasa a la función del usuario `f()`. Las respuestas de la `f()` van siendo procesadas por la función interna de hash que genera un **checksum** `H` de las respuestas. Por ejemplo para el primer ejercicio si `seed=123` entonces el checksum es `H=523`. Una vez que el alumno termina su tarea se le pedirá que corra la clase evaluadora con un valor determinado de la semilla `seed` y se comprobará que genere el valor correcto del checksum `H`.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro `seed` con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de `H`.
- En la clase evaluadora cuentan con las siguientes funciones utilitarias:
  - `void dump(vector<set<int> > &VX,string s="")`: Imprime un mapa entero/entero. Nota: Es un método de la clase `Eval` es decir que hay que hacer `Eval ev; ev.dump(VX);`. El string `s` es un label opcional.
  - Análogamente está `void dump(set<int> S,string s="")`.
  - `btree<int>::lisp_print()`: Lisp print de un árbol. Nota: esta pertenece a la clase `tree`. Uso: `btree<int> T; T.lisp_print();`

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2015-09-17]

ATENCIÓN 1: Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría (Ej 2) y 50 % en clases.
ATENCIÓN 2: Hacer cada ejercicio en una **hoja separada**.

[Ej. 1] [clases (W=20pt)]

a) **[list]**
Siendo la siguiente una posible implementación del TAD Lista mediante celdas simplemente enlazadas por punteros:

```
1  class cell;
2  typedef cell *iterator_t;
3
4  class list {
5  private:
6    cell *first;
7    int _size;
8  public:
9    list();
10   ~list();
11   iterator_t insert(iterator_t p,elem_t j);
12   iterator_t erase(iterator_t p);
13   iterator_t erase(iterator_t p,iterator_t q);
14   void clear();
15   iterator_t begin();
16   iterator_t end();
17   iterator_t next(iterator_t p);
18   iterator_t prev(iterator_t p);
19   elem_t & retrieve(iterator_t p);
20   int size();
21 };
22
23 class cell {
24   friend class list;
25   elem_t elem;
26   cell *next;
27   cell() : next(NULL) {}
28 };
```

1) Implemente el método **`iterator_t insert(iterator_t p,elem_t j);`** que inserta en la lista un elemento **`j`** en la posición **`p`** (debe ser $O(1)$)
2) Implemente el método **`iterator_t erase(iterator_t p);`** que elimina de la lista el elemento ubicado en la posición **`p`** (debe ser $O(1)$)
3) Implemente el método **`int size();`** que retorna el número de elementos de la lista (debe ser $O(1)$)

b) **[map]** Siendo la siguiente una posible implementación del TAD Correspondencia mediante vectores ordenados:

```
1  class elem_t {
2  private:
3    friend class map;
4    domain_t first;
5    range_t second;
6  };
7
8  typedef elem_t* iterator_t;
9
10 class map {
11 private:
12   vector<elem_t> v;
13   iterator_t lower_bound(domain_t key);
```

```
14 public:
15   map():
16     iterator_t find(domain_t key);
17   iterator_t insert(domain_t key,range_t val);
18   range_t &retrieve(domain_t key);
19   bool empty();
20   void erase(iterator_t p);
21   int erase(domain_t key);
22   iterator_t begin();
23   iterator_t end();
24   void clear();
25   int size();
26   domain_t key(iterator_t p);
27   range_t &value(iterator_t p);
28 };
```

1) Implemente los métodos
   - **`iterator_t lower_bound(domain_t key);`** es la función auxiliar que retorna la primera posición donde el elemento podría ser insertado de manera que se mantenga ordenado.
   - **`iterator_t find(domain_t key);`**

   La implementación de ambos debe ser tal que **`find()`** es $O(\log(n))$

[Ej. 2] [pregs (W=20pt)]

a) Considere el siguiente algoritmo. Comente brevemente qué es lo que resuelve el mismo.

```
1    bool algoritmo(list<int> &L, list<int>::iterator p, int x) {
2       if(p==L.end()) return false;
3       if(*p==x) return true;
4       return algoritmo(L,++p,x);
5    }
```

b) Determine el tiempo de ejecución $T(n)$ del algoritmo previo para:
1) El peor caso $T_{\text{peor}}(n)$
2) El mejor caso $T_{\text{mejor}}(n)$
3) El caso promedio $T_{\text{prom}}(n)$

c) ¿De qué complejidad algorítmica $O(n)$ es?

d) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$
\begin{aligned}
T_1 &= n^2 + 2 \cdot 4^n + 5^3 \\
T_2 &= 3 \cdot 2^n + 5n^4 + 2\,n! \\
T_3 &= 2 \log_5 n + \sqrt{n} + 3\,n^2 + 2\,n^5 \\
T_4 &= 4^5 + 2.7 \log_4 n + \log_2(5) n^{1.5} \\
T_5 &= \log_2 n + 5
\end{aligned}
$$

e) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase **`map<>`** implementada con **listas ordenadas** y **vectores ordenados** en el caso promedio? Métodos: **`find(key)`**, **`M[key]`**, **`erase(key)`**, **`erase(p)`**, **`begin()`**, **`end()`**, **`clear()`**.

f) Discuta las ventajas y desventajas de utilizar listas **doblemente enlazadas** con respecto a las **simplemente enlazadas**. ¿Cuáles son los métodos cuyo **tiempo de ejecución cambia** y por qué?

mstorti@galileo/parc1-2015-8-g5ff6a2a/Wed Sep 23 14:30:34 2015 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2015-11-17]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en las restantes secciones.
[ATENCIÓN 2] Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
[ATENCIÓN 3] Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS1, Hoja #2/3 LOVELACE, ADA

[Ej. 1] [CLAS1 (W=20pt)]

a) **[pila]** Sea la siguiente una posible implementación del **TAD Pila** basada en una lista:

```
class stack : private list {
private:
  int size_m;
public:
  stack();
  void clear();
  T& top();
  void pop();
  void push(T x);
  int size();
  bool empty();
};
```

Implemente los 6 métodos de su interfase pública.

b) **[list]** Implemente los métodos:

```
iterator_t insert(T& x);
iterator_t erase(iterator_t p);
iterator_t erase(iterator_t p, iterator_t q);
```

de una **lista simplemente enlazada por punteros**, siendo:

```
class cell{
  cell *next;
  T elem;
  cell() : next(NULL) {}
}
typedef cell *iterator_t;
```

[Ej. 2] [PREG1 (W=20pt, 4pt por pregunta)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$
\begin{aligned}
T_1 &= 3^{10} + n^3 + 2 \cdot 3^n, \\
T_2 &= 5 \log_2 n + 3^5 + 2 \log_4 n + 100, \\
T_3 &= \sqrt{3} \cdot n + 2 \cdot 5^n + \log_{10} n, \\
T_4 &= 4n^5 + 3 \cdot 2^n + 52\, n!. \\
T_5 &= \sqrt{n} + 3 \log_4 n + 3\, n^3 + 5\, n^4,
\end{aligned}
$$

b) Si la correspondencia `M={(3->8),(5->2)}` y ejecutamos el código `int x= M[3]`. ¿Que valor toma `x`? ¿Cómo queda `M`? Si ahora hacemos `x = M[4]`, ¿cómo quedan `x` y `M`?

c) Defina que es un **camino** en un árbol. Dado el AOO `(Z (Q R) (Y W V) T)`, cuáles de los siguientes son caminos?

1) **(R Q Z)**,
2) **(Z Y V)**,
3) **(Q Z Y W)**,
4) **(Z Q R)**.

d) ¿Cuáles son los tiempos de ejecución para los siguientes métodos de la clase **map<>** implementada con listas desordenadas en el caso promedio? Métodos: **find(key)**, **M[key]**, **erase(key)**, **erase(p)**, **begin()**, **end()**, **clear()**.

e) Explique que quiere decir la propiedad de **transitividad** de la notación $O()$.

[Ej. 3] [CLAS2 (W=20pt)]

a) **[setvb]** Siendo la siguiente una posible implementación del **TAD Conjunto (set)** para el conjunto universal **U=[0,50)** mediante un vector de bits:

```
const int N=50;
typedef int elem_t;
int indx(elem_t t) { return t; }
elem_t element(int j) { return j; }

typedef int iterator_t;
typedef pair<iterator_t,bool> pair_t;
class set {
private:
  vector<bool> v;
  iterator_t next_aux(iterator_t p) { /* ... */ };
public:
  set() : v(N,0) { }
  set(const set &A) : v(A.v) {}
  ~set() {}
  iterator_t lower_bound(elem_t x) {
    return next_aux(indx(x));
  }
  pair_t insert(elem_t x) { /* ... */ }
  elem_t retrieve(iterator_t p) { ... }
  void erase(iterator_t p) { /* ... */ }
  bool erase(elem_t x) { /* ... */ }
  void clear() { for(int j=0; j<N; j++) v[j]=false; }
  iterator_t find(elem_t x) { return (v[indx(x)] ?  indx(x) : N);
  iterator_t begin() { return next_aux(0); }
  iterator_t end() { return N; }
  iterator_t next(iterator_t p) { return next_aux(++p);}
  int size() { /* ... */ }
  friend void set_union(set &A,set &B,set &C);
  friend void set_intersection(set &A,set &B,set &C);
  friend void set_difference(set &A,set &B,set &C);
};
```

- Implemente el método **iterator_t next_aux(iterator_t p)** que dado un **iterator_t p**, retorne en **iterator_t** el siguiente elemento válido del set.
- Implemente el método **bool erase(elem_t x)** que elimina el elemento **x** del conjunto y retorna **true** si efectivamente lo eliminó o **false** si no existía en el conjunto.
- Implemente los métodos **set_union**, **set_intersection** y **set_difference** que realizan las operaciones binarias correspondientes entre los sets **A** y **B**, almacenando el resultado en **C**.

b) **[AB]** Siendo la siguiente una posible implementación del **TAD Árbol Binario (AB)** mediante punteros:

```
class btree;
class iterator_t;
class cell {
  friend class btree;
  friend class iterator_t;
  elem_t t;
  cell *right,*left;
  cell() : right(NULL), left(NULL) {}
};

class iterator_t {
private:
  friend class btree;
  cell *ptr,*father;
  enum side_t {NONE,R,L};
  side_t side;
  iterator_t(cell *p,side_t side_a,cell *f_a)
    : ptr(p), side(side_a), father(f_a) { }
public:
  iterator_t(const iterator_t &q) {
    ptr = q.ptr;
    side = q.side;
    father = q.father;
  }
  bool operator!=(iterator_t q) { return ptr!=q.ptr; }
  bool operator==(iterator_t q) { return ptr==q.ptr; }
  iterator_t left() { /* ... */ }
  iterator_t right() { /* ... */ }
};

class btree {
private:
  cell *header;
  tree(const tree &T) {}
public:
  btree() {
    header = new cell;
    header->right = NULL;
```

```
    header->left = NULL;
  }

  ~btree() { clear(); delete header; }
  iterator_t insert(iterator_t p,elem_t t)
    iterator_t erase(iterator_t p){ /* ... */ }
  ...
  void clear() { erase(begin()); }
```

```
  iterator_t begin() {
    return iterator_t(header->left,
           iterator_t::L,header);}
  iterator_t end() { return iterator_t(); }

  bool empty() { return begin()==end(); }
  int size() { /* ... */ }
};
```

- Implemente los métodos **`iterator_t iterator_t::right()`** y **`iterator_t iterator_t::left()`** que retornan el **`iterator_t`** al hijo derecho y el hijo izquierdo respectivamente.
- Implemente el método **`iterator_t btree::insert(iterator_t p,elem_t elem)`** que inserta el elemento **`elem`** en la posición **`p`** y retorna el **`iterator_t`** actualizado.
- Implemente el método **`iterator_t btree::erase(iterator_t p)`** que elimina el elemento en la posición **`p`** (y todo su subárbol) y retorna el **`iterator_t`** al siguiente hermano a la derecha.

[Ej. 4] [OPER2 (W=20pt)]

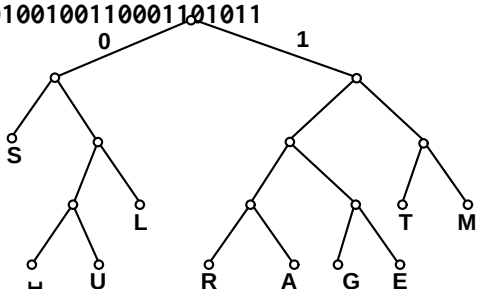
a) **[rec-arbol (2.5pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son
- **`ORD-PRE=(B,Q,R,Z,Y,W,T,Q,O)`**,
- **`ORD-POST=(Z,Y,R,Q,W,O,Q,T,B)`**,

b) **[part-arbol (2.5pt)]** Dado el árbol ordenado orientado (AOO): **`(T Z (Q (R W X (Y F))))`** determinar cuales son los nodos **antecesores, descendientes, izquierda** y **derecha** del nodo **`Q`**. ¿Son **disjuntos**? Justifique.

c) **[huffman (2.5pt)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`BIGBROTHER`**, $P(B) = 0.05$, $P(H) = 0.05$, $P(R) = 0.05$, $P(T) = 0.10$, $P(O) = 0.20$, $P(Z) = 0.15$, $P(I) = 0.10$, $P(G) = 0.10$, $P(X) = 0.10$, $P(E) = 0.10$. Calcular la longitud promedio del código obtenido.

d) **[hf-decode (2.5pt)]**
Utilizando el código de la derecha desencodar el mensaje

`1111011010010100101001101001010010110110100100110001101011`



e) **[abb (2.5pt)]** Dados los enteros **`(13,7,20,2,3,10,5,4,3,12,1)`** insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 13, 7, y 2 en ese orden.

f) **[hash-dict (2.5pt)]** Insertar los números **`(7,13,23,6,5,33,15,2)`** en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión **`h(x) = x`**.

g) **[heap-sort (2.5pt)]** Dados los enteros **`(3,12,10,4,5,15,7)`** ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

h) **[quick-sort (2.5pts)]** Dados los enteros **`(8,2,9,4,1,3,10,2,7,5,4,0)`** ordenarlos por el método de **clasficación rápida** (**quick-sort**). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 5] [PREG2 (W=20pt, 4pt por pregunta)]

a) Si queremos generar un código binario de igual longitud para un conjunto de 26 caracteres. ¿Cuantos bits tendrá, como mínimo, cada caracter?

b) ¿Como se define la altura de un nodo en un árbol? ¿Cuál es la altura de los nodos B, C, y D en **`(X Y Z (B C (D (E G H I))))`**?

c) Escriba la **definición recursiva** de la función **altura** de un **AOO**. (**Nota:** No pedimos código.)

d) Discuta la **estabilidad** del algoritmo de ordenamiento de **listas por fusión** (**merge-sort**). ¿Es estable? ¿Bajo que condiciones?

e) ¿Cuál es el tiempo de ejecución del algoritmo de **ordenamiento rápido** (**quick-sort**), en el caso promedio y en el peor caso? ¿Cuando se produce el peor caso?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Super R-Recuperatorio. [2016-02-10]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en las restantes secciones.
[ATENCIÓN 2] Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
[ATENCIÓN 3] Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS2, Hoja #2/3 LOVELACE, ADA`

[Ej. 1] [CLAS1 (W=20pt)]

a) **[list]** Siendo la siguiente una posible implementación del TAD **Lista mediante celdas simplemente enlazadas por punteros**:

```
class cell;
typedef cell *iterator_t;
class list {
  private:
  cell *first;
  public:
  list();
  ~list();
  iterator_t insert(iterator_t p,elem_t j);
  iterator_t erase(iterator_t p);
  iterator_t erase(iterator_t p,iterator_t q);
  void clear();
  iterator_t begin();
  iterator_t end();
  iterator_t next(iterator_t p);
  iterator_t prev(iterator_t p);
  elem_t & retrieve(iterator_t p);
  int size();
};

class cell { ... };
```

Sin modificar la definición de la clase **`list`**,

- Complete la definición de la clase **`cell`**.
- Implemente el constructor y los métodos **`prev`** y **`size`**.
- Determine y justifique los órdenes de crecimiento $O(?)$ para los métodos implementados en el punto 2.

b) **[map]** Defina una clase **`map`** para modelar una correspondencia entre strings y doubles e implemente todos los métodos y auxiliares que considere necesarios para utilizarla en el siguiente ejemplo:

```
map nums;
nums.insert("golden ratio", 1.618 );
nums.insert("pi", 3.14159 );
nums.insert("ultimate answer", 42 );
nums.insert("speed of light", 299e8 );

string name;
cin >> name;
iterator_t it = nums.find(name);
if (it == nums.end()) cout << "Error: Number not found";
else cout << "Your number is " << nums.value(it);
```

Ayudas: se sugiere implementarla mediante un vector ordenado de structs (definir **`elem_t`**) , y definir (implementar) el método auxiliar **`lower_bound(...)`**. Nota: no hace falta declarar ni definir métodos u operadores de **`map`** que no se requieran para este ejemplo.

[Ej. 2] [CLAS2 (W=20pt)]

a) **[ab]** Para la siguiente implementación del **TAD Arbol Binario (AB)**:

```
class cell { /*...*/ };
class iterator_t {
/*...*/
public:
    iterator_t left() { /* ... */ }
    iterator_t right() { /* ... */ }
};
class btree {
private:
    cell *header;
    btree(const btree &T);
public:
    btree();
    ~btree();
    iterator_t insert(iterator_t p,elem_t t);
    iterator_t erase(iterator_t p);
    void clear();
    iterator_t begin();
    iterator_t end();
    bool empty();
    int size();
};
```

Complete las definiciones de las clases **`cell`** e **`iterator_t`** e implemente el constructor de la clase **`btree`** y el método **`insert`**.

b) **[hashset]** Para la siguiente implementación de un **diccionario por tabla de dispersión abiertas**:

```
typedef int key_t;
typedef int (*hash_fun)(key_t x);
class iterator_t {
    friend class hash_set;
private:
    int bucket;
    std::list<key_t>::iterator p;
    iterator_t(int b,std::list<key_t>::iterator q);
public:
    bool operator==(iterator_t q);
    bool operator!=(iterator_t q);
    iterator_t();
};
class hash_set {
private:
    typedef std::list<key_t> list_t;
    typedef list_t::iterator listit_t;
    typedef std::pair<iterator_t,bool> pair_t;
    hash_set(const hash_set&) {}
    hash_set& operator=(const hash_set&) {}
    hash_fun h;
    int B;
    int count;
    std::vector<list_t> v;
    iterator_t next_aux(iterator_t p);
public:
    hash_set(int B_a,hash_fun h_a)
        : B(B_a), v(B), h(h_a), count(0) {}
    iterator_t begin();
    iterator_t end();
    iterator_t next(iterator_t p);
    key_t retrieve(iterator_t p);
    pair_t insert(const key_t& x);
    iterator_t find(const key_t& x);
    int erase(const key_t& x);
    void erase(iterator_t p);
    void clear();
    int size();
};
```

Implemente los métodos **`insert(const key_t&)`** y **`erase(const key_t&)`**.

[Ej. 3] [PREG2 (W=20pt, 4pt por pregunta)]

a) ¿Es verdad que si dos nodos están en el **mismo nivel** de un árbol, entonces son **hermanos**? ¿Y la recíproca? De ejemplos.

b) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un Árbol Binario (AB)? ¿Y en un Árbol Ordenado Orientado (AOO)? Discuta y de ejemplos.

c) Explique cual es la **condición de prefijos** para códigos binarios. De un ejemplo de códigos que cumplen con la condicion de prefijos y que no cumplen para un conjuntos de 3 caracteres.

d) Defina en forma recursiva el listado en **orden previo** y el listado en **orden posterior** de un **Árbol Ordenado Orientado (AOO)** con raíz **`t`** y sub-árboles hijos **`h_1,h_2,...,h_n`**.

e) Cual es el tiempo de ejecución (en notación $O(\cdot)$, en promedio) en función del número de elementos (n) que posee el árbol ordenado orientado implementado con celdas encadenadas por punteros de las siguientes operaciones/funciones/métodos

- **`insert(p,x)`**
- **`begin()`**
- **`lchild()`**
- **`operator++`** (prefijo/postfijo)

- `find(x)`
- `erase(p)`
- `*n`

f) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.

g) Para el arbol binario `(1 . (2 (3 5 .) 6))`

- como queda el árbol y que sucede si hacemos
```
btree<int>::iterator p=T.begin();
p=T.erase(p.left());
```
- y como queda el árbol asi hacemos por otro lado
```
btree<int>::iterator p=T.begin(),n;
n=p;
n=n.right();
p=T.splice(p.left(),n);
```

h) Discuta el **número de intercambios** que requieren los algoritmos de ordenamiento **lentos** en el peor caso.

i) ¿Cuál es el costo de **inserción** en tablas de dispersión cerradas?

j) ¿Cuál es el resultado de aplicar la función `l=particiona(w,j,k,v)`? (**Nota:** No se pide el algoritmo, sino cuál es el efecto de aplicar tal función, independientemende de como se programe). Explique los argumentos de la función. ¿Cuál debe ser el tiempo de ejecución para `particiona()` si queremos que `quick_sort()` sea un algoritmo **rápido**?

Algoritmos y Estructuras de Datos.
TPL1. Trabajo Práctico de Laboratorio. [2015-08-29]

PASSWD PARA EL ZIP: ZF9 K22 3RU Y8M

Ejercicios

[Ej. 1] **[iscomb]** Dado un vector de listas `VL` y una lista `L`, determinar si `L` es una combinación de las listas de `VL` en alguna permutación dada. Cada una de las `VL[j]` debe aparecer una y sólo una vez en `L`. Por ejemplo si `VL=[(1,2,3),(4,5,6),(7,8)]`, y `L=(7,8,4,5,6,1,2,3)` entonces `iscomb(VL,L)->true`. Pero si `L=(3,2,1,7,8,4,5,6)->false`.

Ayuda:

- Escribir una funcion `bool isprefix(Lpref,L);` que retorna `true` si `Lpref` es prefijo de `L`.
- Primero chequear que la longitud de `L` debe ser igual a la suma de las longitudes de los `VL[j]`.
- Si pasa este test se procede en forma recursiva. Para cada `VL[j]` se determina si es prefijo de `L`. Si ninguna de ellas es prefijo entonces retorna `false`. Si una (o varias de ellas) lo es entonces se aplica recursivamente `iscomb()` para determinar si `L-VL[j]` es una combinación de `VL-VL[j]`. Es decir se quita `VL[j]` del comienzo de `L` y la posicion `j` de `VL`. La recursión se corta cuando `L` es vacía.

[Ej. 2] **[max-sublist]** Programar una función `list<int> max_sublist(list<int> &L);` la cual recibe una lista de enteros y encuentra y retorna la sublista consecutiva `Lmax` que obtenga la mayor suma entre todos sus elementos. Notar que debido a que algunos elementos pueden ser negativos el problema no se resuelve simplemente tomando todos los elementos. También es posible que la sublista resultado no contenga ningún elemento, en el caso de que todos los elementos de `L` sean negativos.

Por ejemplo:

```
(1, 2, -5, 4, -3, 2) -> (4)
(5, -3, -5, 1, 7, -2) -> (1,7)
(4, -3, 11, -2) -> (4, -3, 11)
(4  -4  2 2) -> (4)
```

Si hay varias sublistas que den la misma suma mínima debe retornar la primera y la más corta. En el último ejemplo de arriba hay tres sublistas que dan la máxima suma de 4: a saber `(4)`, `(4 -4 2 2)`, `(2 2)`. Por este criterio debe retornar `(4)` ya que es la que empieza primero y la más corta.

[Ej. 3] **[mergesort]** Programar una funcion `void mergesort(list<int> &L);` que reciba una lista `L` desordenada y la ordene en forma ascendente mediante la siguiente estrategia recursiva:

- Si la lista está vacía o de un elemento ya está ordenada
- Sino se parte la lista en dos sublistas y se las ordena a cada una de forma recursiva. Luego se mezclan (fusionan) cada una de las sublistas ya ordenadas.
- Para partir una lista puede utilizarse el método de la clase list `void splice (iterator position, list& x, iterator first, iterator last);` el cual transfiere los elementos desde la lista `x` al contenedor que realiza la llamada en la posición position.
- Se sugiere escribir una función `void merge(list<int> &L1, list<int> &L2, list<int>& L);` la cual recibe dos listas ordenadas de forma ascendente `L1` y `L2` y retorna una lista `L`, pasada como parámetro, con los elementos de ambas ordenados tambien en forma ascendente. Por ejemplo si `L1=(1, 3, 6, 11)` y `L2=(2, 4, 6, 10)`, la lista `L` debe quedar como `(1, 2, 3, 4, 6, 6, 10, 11)`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay una función de evaluación, por ejemplo si **`fj`** son las funciones a evaluar tenemos

```
Eval ev;
int vrbs = 0;
ev.eval1(f1,vrbs);
ev.eval2(f2,vrbs);
ev.eval3(f3,vrbs);
```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las función (**`eval1`**). La primera **`ev.evalj(fj,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`fj`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval ev; ev.dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
  - **`void Eval::dump(vector< list<int> > VL,string s="")`**.
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.pdf`**). Primero el apellido.

Algoritmos y Estructuras de Datos.
TPL1R. Recuperatorio Trabajo Práctico de Laboratorio. [2015-09-12]

PASSWD PARA EL ZIP: `Y69 8LP BPM EP9`

Ejercicios

[Ej. 1] **[mochila]** Se necesita llenar una mochila, incapaz de soportar más de un peso determinado `K`, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo. En esta versión simplificada, se supone que cada objeto tiene el mismo valor, pero pueden tener diferentes pesos. Por lo tanto se trata de **elegir la mayor cantidad de objetos sin exceder** `K`. El peso de los objetos (enteros positivos) está en una lista `P`.

Consigna: Escribir una función `list<int> mochila(list<int> &P, int K);` que reciba una lista `P` con el peso de cada uno de los objetos que se podrían llevar en la mochila y un peso máximo permitido `K` y que retorne una lista `M` con una lista con un subconjunto de los elementos de `P` cuya suma es `<=K` y tiene la máxima cantidad de elementos. Si hay varias posibilidades debe retornar cualquiera de ellas.

Ejemplos:

- Si `P=(3,2,7,11,2,5,9,4), K=11 -> M=(3,2,2,4)`
- Si `K=10 -> M=(3,2,2)` o `M=(4,2,2)` o `M=(5,2,2)`
- Si `K=12 -> M=(2,2,3,4)`

Ayuda:

- Escribir el algoritmo en forma recursiva. Si `K<=0` o `P=()` entonces la solución es la lista vacía.
- Tomar el primer elemento `a0` de `P` y formar la lista `Pm0=P-(a0)` es decir `P` pero sin el elemento `a0`.
- Considerar dos casos, que `a0` esté o no en `M`. Si `a0>K` no hace falta considerar la posibilidad de que `a0` esté en `M`.
- En el caso que `a0` **NO** esté en `M` llamar a `mochila(Pm0,K)`.
- En el caso que `a0` **SI** esté en `M` la suma de los elementos de `Pm0` debe sumar `K-a0` de manera que se debe llamar a `mochila(Pm0,K-a0)`. A la `M` resultante debe agregársele `a0`.
- De las dos posibilidades quedarse con la que tiene la mochila con más elementos.

Notas:

- Los elementos en `M` pueden estar en cualquier orden.
- Puede haber elementos iguales en `P`.
- Puede darse `K==0` en ese caso debe retornar `M=()`.

[Ej. 2] **[enunosolo]** Dada una lista de listas `LL` y una lista `L`, buscar que cada elemento de `L` esté contenido en una y sólo una lista de `LL`. La función propuesta es
`bool contenido(list <list<int> > &LL, list <int> &L);` Si el elemento de `L` no se encuentra en ninguna lista contenida en `LL`, deberá retornar `false`.

Ejemplos: Considerando `LL=((1 2 3 4) (1 2 3 5) (1 2 3 6))`.

- Si **L=(6 5 4) -> true** ya que 6 está sólo en la lista **L2**, 5 en **L1** y **4** en **L0**
- Si **L=(6 5) -> true** ya que 6 está sólo en la lista **L2**, 5 en **L1**. No hay ningún elemento único para la lista **L0** pero no importa.
- Si **L=(6 5 1) -> false**, ya que 1 está en todas las listas.
- Si **L=(6 5 3) -> false**, ya que 3 está en todas las listas.
- Si **L=(6 5 3 7) -> false**, ya que 3 está en todas las listas y 7 en ninguna.

Ayuda:

- Escribir una función auxiliar
  bool contiene(list<int>&L,int x), que retorna **true** si **x** está en **L**.
- Para cada elemento **x** de **L** recorrer todas las listas de **LL** y contar en cuantas de ellas está **x**. Si no está exactamente una vez retornar **false**.
- Si todos los elementos de **L** pasan la condición entonces retorna **true**.

[Ej. 3] **[ppt]** Escriba una función
int ppt(list<int> &H, int n); que reciba una lista de enteros que representan todas las elecciones previas del jugador oponente en el juego **piedra-papel-tijera** y devuelve la siguiente más probable **nextplay**. La lista solo tiene los enteros 1, 2 y 3, que se corresponden con las elecciones de “piedra”, “papel” y “tijera” respectivamente. El último elemento de la lista corresponde a la última jugada. El objetivo es tratar de predecir la próxima jugada **nextplay** del oponente. Para ello debe buscar en su historial **H** todas las veces que el jugador jugó la misma secuencia que en las últimas **n** partidas y retornar el valor siguiente a dicha vez que apareció la secuencia.

Por ejemplo, si el historial es **H=(1,1,2,2,1,2,3,3,2,1,2,3,1,2)** y **n=2** la función debería retornar 3. Las últimas dos jugadas fueron **last=(1,2)**, y la anterior vez que jugó **(1,2)** eligió a continuación **nextplay=3**.

En caso de que la secuencia **last** no aparezca debe retornar **nextplay=0**.

Ayuda:

- Copie en una lista auxiliar **last** las últimas **n** jugadas.
- Recorriendo **H** para cada posición **p** verificar si a partir de **p** se encuentra una copia de **last**.
- Si se encuentra y hay al menos un elemento adicional en **H** quedarse con ese candidato **nextplay**.
- De todos los posibles **nextplay** quedarse con el último.

Notas:

- Tener en cuenta que dos apariciones de la secuencia buscada pueden solaparse, por ejemplo si **H=(1 2 1 2 1 2 3 1 2 1 2)** y **n=4** tenemos las secuencias **(1 2 1 2 1)** y **(1 2 1 2 3)**. Se recomienda utilizad un iterador **p** que va recorriendo la lista y para cada **p** recorrer con otro iterador **q** desde **p** hacia atrás.
- Tener en cuenta que al final de **H** aparece la secuencia buscada, pero no debe tenerse en cuenta porque no tiene un elemento siguiente.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.evalj(f,vrbs);
  hj = ev.evalrj(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval1`** y **`evalr1`**). La primera **`ev.evalj(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- La segunda función **`evalrj`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum `H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalrj()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**. Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
  - **`void Eval::dump(list< list<int> > &LL,string s="")`**.
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.pdf`**). Primero el apellido.

Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio. [2015-10-03]

PASSWD PARA EL ZIP: K69X H8G3 LP9X

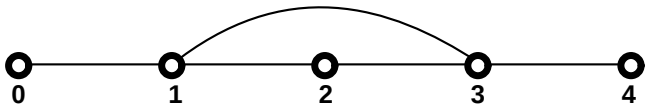
Ejercicios

[Ej. 1] **[getpath (25pt)]** Escribir una función
`void getpath(map<int,list<int> > &M, int vrtx, list<int> &path);` que retorna un camino `path` en el grafo simple definido por `M` que comienza en `vrtx` y no repite ningún vértice. El algoritmo debe utilizar una estrategia ávida (no recursiva) de la siguiente manera:

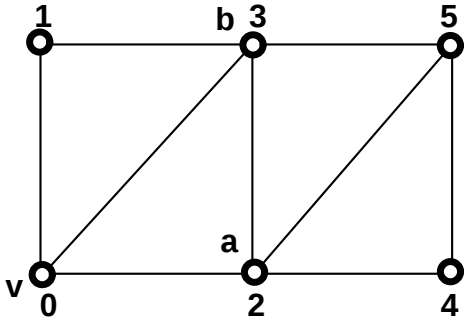
- Insertar `vrtx` en el `path`.
- Buscar entre los vecino del último nodo insertado (y que a su vez es el último en `path`), el primero que aún no se encuentre en `path`, e insertarlo.
- Repetir el paso previos hasta que no se encuentre ningún vecino que aún no haya sido insertado.

Ayuda: Es conveniente gestionar un `vector<bool> visited` que indique si el vértice `i` ya fue incluído en `path` o no (`visited[i]` es `true` si ya fue visitado o viceversa).
Por ejemplo, para el grafo de la figura, representado de la siguiente forma
`M={0->(1),1->(3,0,2),2->(1,3),3->(1,4,2),4->(3)}`
entonces si `vrtx=3` debe retornar `path=(3,1,0)`, y (para el mismo grafo)

- `vrtx=0, => path=(0 1 3 4)`
- `vrtx=1, => path=(1 3 4)`
- `vrtx=2, => path=(2 1 3 4)`
- `vrtx=4, => path=(4 3 1 0)`



[Ej. 2] **[tricount (35pt)]** Escribir una función
`int tricount(map<int,list<int> > &M);` que retorna la cantidad de triángulos en un grafo simple `M`. Es decir los vértices del grafo representan puntos en una hoja de papel, y las aristas segmentos que unen pares de puntos. Escribir una función para determinar cuantos triángulos hay dibujados. Por ejemplo en el grafo de la figura debe retornar 4.
`M={0->(3,1,2),1->(3,0),2->(5,0,4,3),3->(2,1,0,5),4->(2,5),5->(2,3,4)}`



Ayuda: Para cada vértice `v` del grafo tomar la lista de vecinos `v` y

- Inicializar un contador a `0`.

- Con dos iteradores recorrer todos los pares posibles **{a,b}** de la lista **v**.
- Para cada par **{a,b}**, verificar si **a** y **b** son vecinos entre sí. En caso afirmativo, incrementar el contador.

Al terminar de recorrer todos los vértices el número final debe ser 6 veces el número de triángulos, ya que cada triángulo tiene 3 aristas y cada arista es contada dos veces.
Para chequear si el vértice **b** es vecino de **a** se sugiere se sugiere escribir una función auxiliar **`bool contiene(list<int> &ngbrs,int b);`** que determina si el vértice **vrtx** está en la lista de vecinos **ngbrs**.
Nota sobre el recorrido de los pares: Para recorrer todos los pares de enteros **j,k** con valores entre **[0,M)** lo podemos hacer de dos formas.

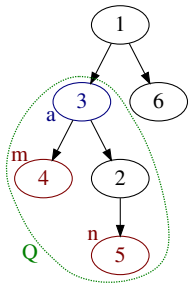
```
for (int j=0; j<M; j++) { // VERSION 1
  for (int k=0; k<M; k++) {
    // Inspecciona el par j,k...
  }
}
```

```
for (int j=0; j<M-1; j++) {  // VERSION 2
  for (int k=j+1; k<M; k++) {
    // Inspecciona el par j,k...
  }
}
```

La versión 2 garantiza que **j!=k** y que se pasa una sola vez por el mismo par, por ejemplo se pasa por **(2,5)** pero no por **(5,2)**. Tener en cuenta que si se recorre de la forma 2 se cuenta 1 sola vez cada arista, de manera que el contador final será **3 veces el número de triángulos y no 6**.

[Ej. 3] **[min-com-subtree (35pt)]** Dados dos nodos **m,n** de un arbol **T**, extraer el mínimo subárbol **Q** que contiene a **m**, y a **n**, es decir el árbol del primer antecesor común de ambos **a**. Por ejemplo en el caso que **T=(1 (3 4 (2 5)) 6)**, y ***m=4,*n=5** debe retornar el subárbol de 3, ya que es el primer antecesor común de ambos. Es decir debe retornar **Q=(3 4 (2 5))**.
Nota 1: Asumimos **`typedef tree<int> tree_t; typedef tree<int>::iterator node_t;`**
Nota 2: Se garantiza que **m** y **n** **no tienen relación de antecesor/descendiente** entre ellos, es decir **a!=m** y **a!=n**.
Consigna: Escribir
`void min_com_subtree(tree_t &T,node_t m,node_t n, tree_t &Q);`
Ayuda:

- Escribir una función auxiliar recursiva **`list<node_t> nodepath(tree_t &T,node_t n,node_t q);`** que retorna una lista de nodos que corresponde al camino que va desde **n** hasta el nodo **q** si **q** está en el subárbol de **n**. Si el nodo **q** no está en el subárbol entonces debe retornar la lista vacía. Hacerlo en forma recursiva.
  - Si **n** es Λ entonces debe retornar la lista vacía.
  - Si **n==q** debe retornar simplemente una lista conteniendo al nodo **n** (cortar recursión).
  - Si no es, llamar recursivamente a **nodepath(T,c,q)** para todos los hijos **c** de **n**. Si alguno de ellos retorna un camino no vacío entonces debe retornar esa lista, prependizando el nodo **n**.
- Usar **nodepath()** para hallar los caminos **mpath** y **npath** a **m,n**.
- Buscar el primer antecesor **a** común de **m,n** (en el ejemplo ***a=3**). Para esto ir recorriendo ambas listas con iteradores **am,an** hasta encontrar el primer par que es diferente ***am=*an**. El anterior es **a**. Como **m** y **n** no tienen relación de antecesor/descendiente está garantizado que se encontrará este antecesor común **a**, y que no es ni **a=m**, ni **a=n**.

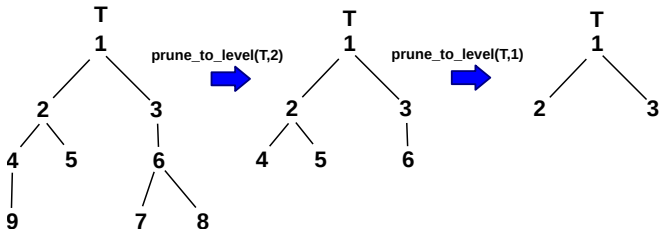


- Mover a **Q** el subárbol de **a** usando **`tree::splice()`**.

Notas:

- El algorimo descripto en la ayuda modifica `T` (es **destructivo**). (Notar que está OK, ya que la consigna no indica lo contrario).
- Notar que los caminos son **listas de nodos** en el árbol, por lo tanto si `am` es un iterator en `mpath`, `*am` es un nodo en el árbol `T` y `**am` el valor contenido en el árbol. Para la consigna pedida no hace falta inspeccionar los valores contenidos en los nodos. Por ejemplo al buscar el antecesor común sólo se deben comparar los nodos (`*am,*an`, no sus contenidos (`**am,**an`).

[Ej. 4] **[prune-to-level (25pt)]** Escribir una función `void prune_to_level(T,lev);` que poda todas los nodos de un árbol por debajo del nivel `lev` *in-place*. Por ejemplo si al árbol `T` de la izquierda le aplicamos `prune_to_level(T,2)` debe quedar el del centro y si le aplicamos `prune_to_level(T,1)` debe quedar el de la derecha.



Más ejemplos:

- `T=(4 (4 1 5) (0 (2 4) 2)), lev=4, => T=(4 (4 1 5) (0 (2 4) 2))`
- `T=(4 (4 1 5) (0 (2 4) 2)), lev=3, => T=(4 (4 1 5) (0 (2 4) 2))`
- `T=(4 (4 1 5) (0 (2 4) 2)), lev=2, => T=(4 (4 1 5) (0 2 2))`
- `T=(4 (4 1 5) (0 (2 4) 2)), lev=1, => T=(4 4 0)`
- `T=(4 (4 1 5) (0 (2 4) 2)), lev=0, => T=(4)`

Ayuda:

- Escribir una función auxiliar recursiva `void prune_to_level(T,n,lev);`
- Si `n` es Λ no debe hacer nada.
- Si `lev>0` debe aplicar recursivamente la función a cada uno de los hijos con `lev-1`
- Si `lev==0` debe eliminar todos los hijos.

Nota: La función debe operar correctamente para cualquier `lev`. Si es negativo debe dejar el árbol vacío, si es `lev==0` debe dejar sólo la raíz, y si es mayor que la profundidad del árbol no debe hacer nada.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header `tree.h`.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.evalj(f,vrbs);
hj = ev.evalrj(f,seed); // para SEED=123 debe dar Hj=170
```

j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval1** y **evalr1**). La primera **ev.evalj(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- La segunda función **evalrj** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalrj()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**. Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
  - **void Eval::dump(list< list<int> > &LL,string s="")**.
  - **tree<int>::lisp_print()**: Lisp print de un árbol ordenado orientado (AOO). Nota: esta pertenece a la clase **tree**. Uso: **tree<int> T; T.lisp_print();**
  - Idem para AB: **btree<int>::lisp_print()**: Lisp print de un árbol binario (AB). Nota: esta pertenece a la clase **btree**. Uso: **btree<int> T; T.lisp_print();**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.pdf**). Primero el apellido.
- **Puntos:** Notar que la suma de los puntos es 120. La nota del parcial **min(sum(Zj),100)** es decir que si haciendo los tres primeros ejercicios se obtienen 95/100 pts. Para aprobar basta hacer 50/100 pts de manera que basta con cualquiera dos de los cuatro.
- **usercase**: Ahora las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbx,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Torneo de programación:**
  - Los ejercicios 2 (**tricount**) y 3 (**min_com_subtree**) participan del **Torneo**. La tabla de posiciones para el torneo se determina a partir del tiempo que tarda en correr el programa en **50 casos muy grandes**, (por ejemplo 50 grafos aleatorios de 250.000 vértices en el caso de **tricount**).
  - Para ello la función de evaluación por **seed/hash evalrj()** tiene un parámetro adicional **int hardness** (dificultad) que puede tomar valores **0<=hardnes<=5**, **Eval::evalr2(tricount,seed,vrbs,hardness);**.

- **`hardness=0`** (valor por defecto) corresponde a lo que se toma en el TPL para determinar el funcionamiento correcto, y por lo tanto los contenedores son pequeños.
- Para **`hardness>0`** el tamaño va creciendo geométricamente y la función reporta el tiempo que tardó en realizar la tarea. Para **`hardness=5`** corre el tamaño máximo de los contenedores. Por ejemplo la corrida para los 5 valores de **`hardness`** reportan

  ```
  EJ2. Hardness 1, elapsed 0.00811529[s]
  EJ2. Hardness 2, elapsed 0.0350215[s]
  EJ2. Hardness 3, elapsed 0.239452[s]
  EJ2. Hardness 4, elapsed 1.57176[s]
  EJ2. Hardness 5, elapsed 10.8449[s] (mips 1.08925, relative 9.95625[s])
  ```

  `elapsed` es el tiempo transcurrido. De esta forma Uds pueden probar diferentes alternativas de programación y ver si reducen o no el tiempo.
- Para definir los resultados del torneo se correrán todos los programas de los participantes en un **servidor dedicado**, por lo tanto no habrá tiempos de diferencia debido al procesador.
- Para **`hardness=5`** se reportan también los **`mips`** de la máquina (una medida de su eficiencia) y el tiempo relativo (o sea el tiempo real dividido los mips). De esta forma se tiene una medida de la eficiencia del algoritmo independiente de la rapidez del procesador.
- Como referencia, Para los ejercicios 2 y 3 de este TPL los valores obtenidos con un código implementado en base a la ayuda que hemos dado da los siguientes tiempos.

  ```
  EJ2. Hardness 5, elapsed 11.1808[s] (mips 1.06821, relative 10.4668[s])
  EJ3. Hardness 5, elapsed 5.57989[s] (mips 1.08289, relative 5.15276[s])
  ```

`mstorti@galileo/parc1-2015-76-g3da893d/Fri Oct 9 09:15:11 2015 -0300`

Algoritmos y Estructuras de Datos.
TPL2R. Recup Trabajo Práctico de Laboratorio. [2015-10-22]

PASSWD PARA EL ZIP: `P3EJ KPW8 DW9H`

Ejercicios

ATENCION/ERRATA: En **`program.cpp`** el orden de los templates para **`tree2count`** y **`count2tree`** están invertidos.

[Ej. 1] **[tree2count (25pt)]** Escribir una función **`void tree2count(tree<int> &T,list<int> &L)`** que, dado un Arbol Ordenado Orientado (AOO) **`T`** retorna una lista que contiene para cada nodo **`n`**, en preorden, la cantidad de nodos del subárbol de **`n`**. Por ejemplo,

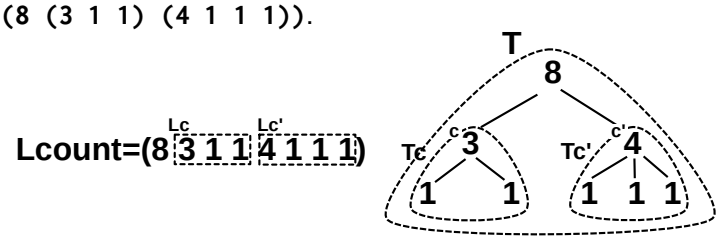
```
T=(4 5 3)  => L=(3 1 1);
T=(6 (7 8 9) (1 2 3 4))  => L=(8 3 1 1 4 1 1 1);
```

Ayuda:

- Conviene hacerlo en dos pasos **`mkcount(T)`** y aplicar luego **`preorder()`**.
- Escribir una función auxiliar recursiva **`mkcount(tree<int> &T,node_t n)`** que reemplaza los valores de los nodos de **`T`** por el conteo de nodos de su subárbol. Si el nodo es Λ no hace nada. Si no es Λ aplica primero **`mkcount`** a los hijos y después reemplaza el valor del nodo por 1+ los valores de los hijos (que a esa altura contendrán el conteo de nodos de sus subárboles).

ATENCION: En **`program.cpp`** la función recursiva auxiliar **`void tree2count(tree<int> &T, node_t n,list<int> &Ln);`** no coincide con la ayuda que les damos. Se podría hacer de esa forma pero no es como le estamos diciendo en la ayuda previa. Si van a seguir la ayuda indiciada aquí conviene eliminarla.

[Ej. 2] **[count2tree (35pt)]** Sea un árbol **`T`** tal que en sus nodos contiene el número de nodos que contiene su subárbol. Por ejemplo **`T=(8 (3 1 1) (4 1 1 1))`**.



Entonces, dada la lista **`Lcount=(8 3 1 1 4 1 1 1)`** de los valores (que son la cantidad de nodos en el subárbol) podemos reconstruir el árbol con el siguiente algoritmo. Sea **`n`** la raíz del árbol.

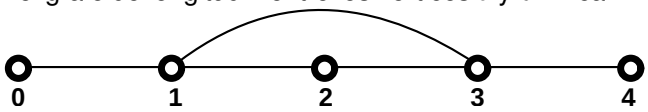
- El primer elemento **`count=8`** es la cantidad de nodos del árbol.
- El siguiente elemento **`3`** es la cantidad de nodos del subárbol del primer hijo **`c`** de **`n`**. Extrae **`3`** elementos **`Lc=(3 1 1)`** y esa es el **`Lcount`** del subárbol de **`c`**.
- El siguiente elemento **`4`** es la cantidad de elementos del segundo subárbol por lo tanto el **`Lcount`** del siguiente nodo **`c'`** es **`Lcount=(4 1 1 1)`**.
- El proceso se detiene cuando se extraen tantos elementos como **`count-1`** (el 1 corresponde a la raíz **`n`**).

Consigna: Escribir una función **`void count2tree(list<int> &L,tree<int> &T);`**

Ayuda: Escribir una función auxiliar recursiva
`node_t count2tree(list<int> &L,tree<int> &T,node_t n);`. Al entrar a la función **n** debe ser un Λ.
La función construye el subárbol en **n** y retorna el iterator refrescado.

- Extrae el primer elemento **count** de **L**, esa es la cantidad de nodos total del árbol, y lo inserta en **n**.
- Hace **count--** ya que insertó un nodo. El valor de **count** es ahora la cantidad total de nodos en los subárboles.
- Recursivamente va a aplicando el algoritmo a **Lcount** y va extrayendo subárboles en los hijos de **c**, **c'**... de **n** hasta que se hayan extraído **count** elementos de la lista.
- Prestar atención con retornar el nodo refrescado ya que la función **count2tree** toma un nodo vacío y inserta algo allí, por lo tanto el iterador cambia.
- Prestar atención a refrescar el iterador y avanzarlo al recorrer los hijos **c**, **c'** ...
- Mantener correctamente el conteo de los nodos que se van extrayendo de la lista. Cada vez que se aplica **count2tree** a **Lcount** la cantidad de nodos está en la primera posición; deben ir acumulando ese valor.

[Ej. 3] **[hasnpath (25pt)]** Dado un grafo simple `map<int,list<int>> G` y dos vértices **a** y **b**, implementar una función `bool hasnpath(graph&G, int a, int b, int n);` que retorne **true** si existe al menos un camino en el grafo de longitud **n** entre los vértices **a** y **b**. El camino puede repetir nodos.



Por ejemplo en este caso tenemos que para los nodos **a=0**, **b=4** debe dar:

```
a=0, b=4, n=1 -> 0
a=0, b=4, n=2 -> 0
a=0, b=4, n=3 -> 1
a=0, b=4, n=4 -> 1
a=0, b=4, n=5 -> 1
```

Ayuda: Utilizar una estrategia recursiva:

- Si la longitud **n** es 0, entonces debe ser **a==b**.
- Si no, verificar si existe un camino de longitud **n-1** entre alguno de los vecinos de **a** (la adyacencia de **a**) y **b** (o viceversa, la adyacencia de **b** con **a**).

[Ej. 4] **[key2abbrevs (35pt)]** Dada una lista de strings todos diferentes entre sí encontrar el menor prefijo de cada uno (abbrev) lo mas corto posible pero de tal forma que las abreviaciones sean diferentes entre si. La abreviación debe tener al menos longitud 1. Por ejemplo

```
# key -> abbrev
abcb -> a
acaa -> ac
badc -> b
bbad -> bb
bbbd -> bbb
bbcd -> bbc
bdca -> bd
```

Consigna: escribir una función `void key2abbrevs(map<string,string> &abbrevs);` que realiza la tarea indicada. Inicialmente **abbrevs** contiene como claves las keys y las imágenes son todas el string vacío. A la salida de la función la imagen debe ser la abreviación.

Ayuda: Utilizar un map auxiliar **`map<string,string> abb2key`** que contendrá la inversa de **`abbrevs`** (es decir **`abbrev->key`**). Inicialmente está vacío. Para cada key de **`abbrevs`**, probar con el prefijo de longitud 1. Si es clave de inverse **`abb2key`** ya tiene otra key asignada y esa abreviación no puede usarse. Probar con un prefijo de longitud 2 y así siguiendo hasta encontrar un prefijo que no tenga valor asignado en **`abb2key`**. Cuando encuentra el prefijo asignarlo en **`abbrevs`** y (la asignación inversa) en **`abb2key`**.
Ayuda memoria para strings: **`s,s1,s2`** son string:

- **`s.substr(pos,len)`** retorna el substring de longitud **`len`** que empieza en **`pos`**. Por ej. **`s="abcd"; s.susbtr(0,3) -> "abc"`**
- **`s.size()`** longitud del string.
- **`s[j]`** char en la posición **`j`**
- **`s=s1+s2`** concatenación.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.evalj(f,vrbs);
  hj = ev.evalrj(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval1`** y **`evalr1`**). La primera **`ev.evalj(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- La segunda función **`evalrj`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalrj()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**. Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es

un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.

- **void Eval::dump(list <int> &L,string s="")**
- **void Eval::dump(list< list<int> > &LL,string s="")**.
- **tree<int>::lisp_print()**: Lisp print de un árbol ordenado orientado (AOO). Nota: esta pertenece a la clase **tree**. Uso: **tree<int> T; T.lisp_print();**
- Idem para AB: **btree<int>::lisp_print()**: Lisp print de un árbol binario (AB). Nota: esta pertenece a la clase **btree**. Uso: **btree<int> T; T.lisp_print();**

- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.pdf**). Primero el apellido.
- **Puntos:** Notar que la suma de los puntos es 120. La nota del parcial **min(sum(Zj),100)** es decir que si haciendo los tres primeros ejercicios se obtienen 95/100 pts. Para aprobar basta hacer 50/100 pts de manera que basta con cualquiera dos de los cuatro.
- **usercase**: Ahora las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbx,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Torneo de programación:** Los ejercicios de este Recup **no participan** del Torneo.

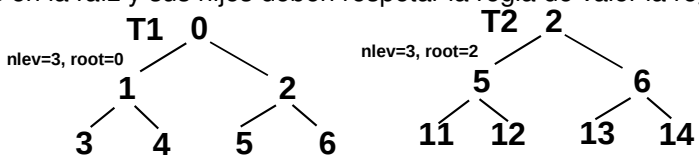
Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio. [2015-11-05]

PASSWD PARA EL ZIP: `H49G IKLJ IU6K`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

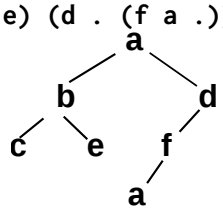
[Ej. 1] **[mkheaptree (35pt)]** Escribir una función `void mkheaptree(btree<int> &T,int nlev,int root);` que construye un árbol binario de `nlev` niveles donde los valores de los nodos corresponde a la numeración por niveles. Si `root=0` entonces la raíz contiene el valor `0`, el primer nivel `1`, y`2`, y así siguiendo. Para ello notar que si el padre es `j` entonces los hijos deben contener `2*j+1` y `2*j+2`. Si `root=0` y `nlev=3` debe retornar el árbol `T1` de la figura. Si `root>0` entonces el árbol debe contener ser construído con `root` en la raíz y sus hijos deben respetar la regla de valer la regla `2*j+1` y `2*j+2`.



Ayuda: Escribir una función recursiva
`void mkheaptree(btree<int> &T,btree<int>::iterator n,int nlev,int root);`

- Si `nlev=0` no debe hacer nada.
- Si no, debe insertar `root` en el `n`, llamar a la función sobre los hijos pasando el valor apropiado de acuerdo a la regla mencionada anterior y el nivel decrementado.

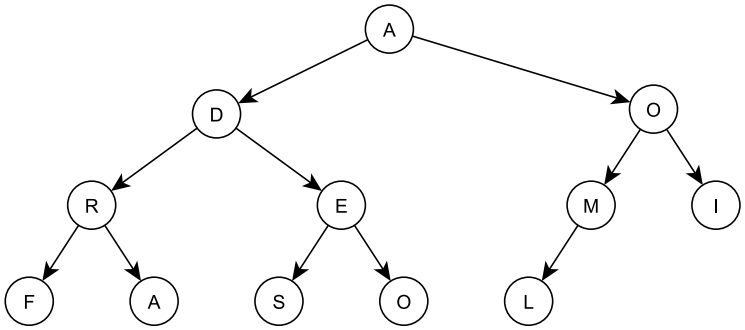
[Ej. 2] **[find-word (35pt)]** Dado un árbol de caracteres, hay que buscar una palabra dentro de lo niveles del árbol. Debe retornar el nivel de menor profundidad en el cual se encuentra la palabra, caso contrario retornar -1. Además la palabra no necesariamente debe tener el largo de nodos del nivel, sino que puede ser un substring del nivel entero.
Consigna: Escribir una función `int find_word( btree<char> &t, string s);`
Ejemplo1: Tenemos el árbol binario `(a (b c e) (d . (f a .)))`



entonces

- si queremos buscar la palabra `fa`, el agoritmo retornará -1;
- si queremos buscar la palabra `a` el algoritmo retornará 0 (si bien la letra también se encuentra como un nodo en el nivel 3, la primera ocurrencia de la misma está en el nivel 0;
- si queremos buscar la palabra `bd`, el algoritmo retornará 1;
- si queremos buscar la palabra `ef`, el algoritmo retornará 2; y así sucesivamente.

Ejemplo2:



- `find_word(T,"do") => 1`
- `find_word(T,"mi") => 2`
- `find_word(T,"la") => -1`

Ayuda:

- Escribir una función recursiva
  ```
  void fill_vector(btree<char> &t, vector<string> &v,
                   btree<char>::iterator it, int actual_level);
  ```
  que llena el `vector<string> v` con las niveles del árbol.
- Recorrer los strings y verificar si el string buscado está contenido en el mismo.
- Es muy conveniente utilizar la función `string::find()`. La expresión `w.find(s)` busca el string `s` dentro del string `w`. Si no lo encuentra retorna el valor especial `string::npos`. Por lo tanto para chequear si contiene al string deben usar la expresión
  ```
  if (w.find(s)!=string::npos) { /* w contiene a s */ }
  ```

[Ej. 3] **[filtcp (25pt)]** Hannibal quiere invitar a su novia a cenar a un restaurante elegante. A lo hora de elegir la cena (compuesta por 1 bebida + 1 plato principal, que por suerte vienen para dos personas), quiere conocer de antemano cuales son las opciones que caben dentro de su presupuesto. Para ayudarlo en su decisión, Hannibal quiere que usted le programe una función `set<pair_t> filtcp(set<int>& S1, set<int>& S2, func_t f);` que dados dos conjuntos de enteros `S1` y `S2` con los precios de cada una de las opciones en bebidas y platos principales respectivamente, retorne un conjunto de pares de enteros `set<pair_t>` con todas las opciones posibles de cena que le es posible elegir de acuerdo a una función predicado f que determina si el par (bebida,plato principal) se encuentra dentro del gasto admitido. Además, Hannibal le solicita que programe a modo de ejemplo una función predicado `bool affordable(pair_t p);` (*accesible*) la cual retorna `true` si el costo de la opción de cena (`pair_t p`) está dentro de su presupuesto `gmin<=costo<=gmax` (con `gmin=5`, `gmax=10`).
Ayuda: Hacer un doble lazo para generar los pares de valores, aplicar el predicado para cada uno de ellos y si lo satisface insertar en el conjunto de salida.
Nota: `filtcp` viene de **filter cartesian product**.

[Ej. 4] **[greatest-subset (25pt)]** Se debe implementar una función
`int greatest_subset(vector<set<char> > &vs,string s);`
que, dado un vector de conjuntos de caracteres `vs` y una palabra `s` retorna la posición del conjunto que más letras de la palabra contenga. En caso de que haya varios que tienen el mismo máximo debe retornar el primero. Si la palabra tiene letras repetidas cuentan una sóla vez.
Ejemplo: `vs= [{a,b},{c}]`, y `s="abccc"` debe retorar `0` ya que si bien `c` aparece tres veces en `s` hay que contarla una vez sóla.

Ayuda:

- Armar un **set<char>** con los caracteres de **s**.
- Recorrer los conjuntos de **vs** y usar la función de intersección de las STL.
- Recordar que el uso de **set_intersection()** es
  set_intersection(A.begin(),A.end(),B.begin(),B.end(),inserter(C.begin()));

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **program.cpp**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header **tree.h**.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **f** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(set<int> S,string s="")**.
  - **void Eval::dump(list <int> &L,string s="")**

  - `void Eval::dump(list< list<int> > &LL,string s="")`.
  - `tree<int>::lisp_print()`: Lisp print de un árbol ordenado orientado (AOO). Nota: esta pertenece a la clase `tree`. Uso: `tree<int> T; T.lisp_print();`
  - Idem para AB: `btree<int>::lisp_print()`: Lisp print de un árbol binario (AB). Nota: esta pertenece a la clase `btree`. Uso: `btree<int> T; T.lisp_print();`

- Después del parcial deben entregar el programa fuente (sólo el `program.cpp`) renombrado con su apellido y nombre (por ejemplo `messilionel.pdf`). Primero el apellido.
- **Puntos:** Notar que la suma de los puntos es 120. La nota del parcial `min(sum(Zj),100)` es decir que si haciendo los tres primeros ejercicios se obtienen 95/100 pts. Para aprobar basta hacer 50/100 pts de manera que basta con cualquiera dos de los cuatro.
- `usercase`: Ahora las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbx,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere "hacer todos". Por ejemplo `ev.eval4(prune_to_level,1,51);` corre sólo el caso 51.

  - Los ejercicios 2 (`tricount`) y 3 (`min_com_subtree`) participan del **Torneo**. La tabla de posiciones para el torneo se determina a partir del tiempo que tarda en correr el programa en **50 casos muy grandes**, (por ejemplo 50 grafos aleatorios de 250.000 vértices en el caso de `tricount`).
  - Para ello la función de evaluación por **seed/hash** `evalrj()` tiene un parámetro adicional `int hardness` (dificultad) que puede tomar valores `0<=hardnes<=5`, `Eval::evalr2(tricount,seed,vrbs,hardness);`.
  - `hardness=0` (valor por defecto) corresponde a lo que se toma en el TPL para determinar el funcionamiento correcto, y por lo tanto los contenedores son pequeños.
  - Para `hardness>0` el tamaño va creciendo geométricamente y la función reporta el tiempo que tardó en realizar la tarea. Para `hardness=5` corre el tamaño máximo de los contenedores. Por ejemplo la corrida para los 5 valores de `hardness` reportan

    ```
    EJ2. Hardness 1, elapsed 0.00811529[s]
    EJ2. Hardness 2, elapsed 0.0350215[s]
    EJ2. Hardness 3, elapsed 0.239452[s]
    EJ2. Hardness 4, elapsed 1.57176[s]
    EJ2. Hardness 5, elapsed 10.8449[s] (mips 1.08925, relative 9.95625[s])
    ```

    `elapsed` es el tiempo transcurrido. De esta forma Uds pueden probar diferentes alternativas de programación y ver si reducen o no el tiempo.
  - Para definir los resultados del torneo se correrán todos los programas de los participantes en un **servidor dedicado**, por lo tanto no habrá tiempos de diferencia debido al procesador.
  - Para `hardness=5` se reportan también los `mips` de la máquina (una medida de su eficiencia) y el tiempo relativo (o sea el tiempo real dividido los mips). De esta forma se tiene una medida de la eficiencia del algoritmo independiente de la rapidez del procesador.
  - Como referencia, Para los ejercicios 2 y 3 de este TPL los valores obtenidos con un código implementado en base a la ayuda que hemos dado da los siguientes tiempos.

    ```
    EJ2. Hardness 5, elapsed 11.1808[s] (mips 1.06821, relative 10.4668[s])
    EJ3. Hardness 5, elapsed 5.57989[s] (mips 1.08289, relative 5.15276[s])
    ```

Algoritmos y Estructuras de Datos.
TPLSR. Super Recup de TPLs. [2015-11-19]

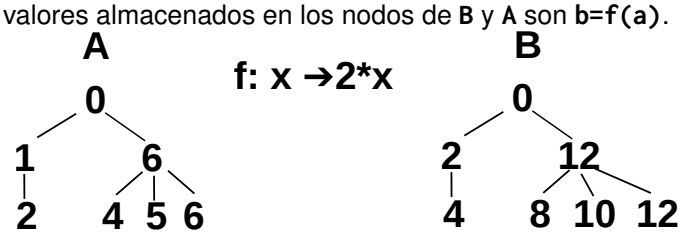
PASSWD PARA EL ZIP: 76DS YVK1 T8NH

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[mkstats (25pt)]** Dado un vector de conjuntos `vector<set<int> > V`, escribir una función `void mkstats(vector<set<int> > &V, map<int,int> &count);` de tal forma que `counts[j]` es la cantidad de conjuntos que contienen a `j`. Si un entero no está en ningún conjunto NO debe estar como clave en `counts` o, dicho de otra manera, los valores en `counts` deben ser todos `count[j]>0`. Por ejemplo, si `V=[{1,2,3},{2,3,4},{3,4,5,9}]` entonces debe dar `M=(1->1,2->2,3->3,4->2,5->1,9->1)`.

[Ej. 2] **[ismaptree (35pt)]**: Dado dos árboles ordenados orientados (AOO) `tree<int> A,B`, y una función de mapeo `mapfunc_t f` (donde `typedef int (*mapfunc_t)(int);`) determinar si `B=f(A)`, es decir si `B` es **semejante** a `A` y los valores almacenados en los nodos de `B` y `A` son `b=f(a)`.



Es decir `B` debe ser semejante a `A` y cada valor de nodo en B surge de aplicar la función `f` al nodo de la posición equivalente en `A`.

Ejemplos: Asumiendo en todos los casos que la función de mapeo es `f(x)=2*x`

```
A=(0 (1 2) (3 4 5 6)), B=(0 (2 4) (6 8 10 12))   -> true
A=(0 (1 2) (3 4 5 6)), B=(0 (2 4) (6 8 10 13))   -> false (mal el valor 13)
A=(0 (1 2) (3 4 5 6)), B=(0 (2 4 5) (6 8 10 12)) -> false (el nodo 5 está de mas)
A=(0 (1 2) (3 4 5 6)), B=(0 (2 4) (6 8 12))      -> false (falta el nodo 10)
```

Ayuda (versión 1): Escribir una función recursiva auxiliar que itera sobre `A` y `B` (como en el `operator==()` de AOO) pero verificando que `*nb=f(*na)` para los nodos `na` en `A` y `nb` en `B`.
Ayuda (versión 2): Escribir una función `void apply(tree<int> &A,mapfunc_t f,tree<int> &fA);` que construye a `fA=f(A)` y después utilizar el operador de igualdad entre árboles.

[Ej. 3] **[foodplan (35pt)]** La princesa Leia se cansó de tener que pensar todas las noches la comida para ella y Hans Solo, así que le pidó a Hans que escriba un programa para que la computadora de a bordo del Halcón Milenario genere cronogramas de comidas para el resto del año. Hans escribió un programa que genera vectores con listas de 7 comidas (para cada día de la semana, para varias semanas). Pero su programa no respeta los caprichos de la princesa. A la princesa no le gusta, por ejemplo, comer pastas dos veces seguidas. Escriba una función `foodplan` que reciba el vector de listas con los cronogramas de varias semanas, una función `f` que dadas dos comidas retorna verdadero si la princesa acepta que se programen consecutivas en la misma semana, y un conjunto con comidas alternativas. Se debe buscar en cada lista pares de comidas consecutivas que no cumplan con el requisito, y corregirlo. Para corregirlo, cuando dos comidas no cumplen el requisito, debe intercambiar la segunda de ellas con la

primer opción compatible que encuentre en el conjunto de alternativas, y eliminar esa opción del conjunto. Las comidas se representan con `std::string`
Por ejemplo, si:

- El vector (para solo una semana) es `[("Pollo","Pollo","Vegetales","Pizza","Pastas","Pastas","Hamburguesas")]`,
- La función `f` retorna falso cuando recibe dos comidas iguales,
- y el conjunto de alternativas es `{"Empanadas", "Pescado"}`, la función debe corregirlo para que quede:
  `[("Pollo","Empanadas","Vegetales","Pizza","Pastas","Pescado","Hamburguesas")]`

Explicación: primero cambia la segunda aparición de `"Pollo"` por `"Empanadas"`, y alternativas queda `{"Pescado", "Pollo"}` ... Luego cambia la segunda aparición de `"Pastas"` por `"Pescado"`.
Consigna: escribir la función `bool foodplan(vector<list<string>> &cronograma, bool (*f)(string,string), set<string> &alternativas);`. Nota: la función retorna `true` si logra arreglar el cronograma, `false` si no encuentra una alternativa factible para alguna de las correcciones.
Ayuda:

- En `foodplan`:
  Para cada lista del vector
  Por cada par de elementos de la lista consecutivos `(c1,c2)`
  Aplicarles la función `F`, si retorna `false`
  Invocar a una función auxiliar `Corregir` pasandole `c1, c2, F` y `alternativas`
  Si `Corregir` retornó `false`, terminar retornando `false`
  Retornar true
- En la función auxiliar `Corregir:`
  Para cada elemento `(a)` del conjunto `alternativas`
  Invocar a `F` con `c1` y `a`.
  Si retorna true
  remover `a` del conjunto e insertar `c2`
  asignar `a` en `c2`, y retornar `true`
  Retornar false

Nota: no se debe considerar el caso en que la última comida de una semana coincide con la primera de la siguiente, ya que las restricciones aplican solo a comidas "de la misma semana"

[Ej. 4] **[minsublist (25pt)]** Escribir una función `int minsum(list<int> &L);` que dada una lista de enteros `L`, se debe retornar la suma total de la sublista de menor suma.
Ejemplos:

```
L=[1 -2 3 -3 -1 2 5 ]; Retorna: -4 (corresponde a sublista [-3 -1])
L=[1 -2 1 -3 -1 2 5 ]; Retorna: -5 (corresponde a sublista [-2 1 -3 -1])
L=[1 2];               Retorna = 0 (corresponde a la sublista vacía [])
```

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más abajo; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.

- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas. El paquete ya incluye el header **`tree.h`**.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Ahora las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbx,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(set<int> S,string s="")`**.
  - **`void Eval::dump(list <int> &L,string s="")`**

  - `void Eval::dump(list< list<int> > &LL,string s="")`.
  - `tree<int>::lisp_print()`: Lisp print de un árbol ordenado orientado (AOO). Nota: esta pertenece a la clase `tree`. Uso: `tree<int> T; T.lisp_print();`
  - Idem para AB: `btree<int>::lisp_print()`: Lisp print de un árbol binario (AB). Nota: esta pertenece a la clase `btree`. Uso: `btree<int> T; T.lisp_print();`
- Después del parcial deben entregar el programa fuente (sólo el `program.cpp`) renombrado con su apellido y nombre (por ejemplo `messilionel.pdf`). Primero el apellido.
- **Puntos:** Los puntos de los ejercicios son variables (hay de 25pt y 35pt). Los puntos conseguidos son **comodines**, se pueden agregar a cualquiera de los TPLs hasta que sature en 100pt.
- **Torneo de programación:** Los ejercicios de este TPL **no participan** del Torneo.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [2016-08-04]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en las restantes secciones.
[ATENCIÓN 2] Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo CLAS1 en una o más hojas, PROG1 en una o más hojas separadas, etc...
[ATENCIÓN 3] Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS1, Hoja #2/3 LOVELACE, ADA

[Ej. 1] [Preguntas (mínimo 70 %)]

a) Sea L una lista conteniendo los elementos $(1, 3, 4, 2, 5, 6)$. Después de aplicar las siguientes líneas

```
list<int>::iterator p,q;
p = L.begin();
q = ++p;
p = L.erase(q);
p++;
q = p;
q++;
```

indique para cada uno de los iterators **p**, **q** si es válido o inválido y en el primer caso el valor que almacena (es decir `*p` y `*q`).

b) Explique que quiere decir la propiedad de "Transitividad" de $O()$.

c) Cuál es la "Regla del producto" para $O()$?.

d) Cual es la signatura de la función `insert()` en listas STL?. Diga que es cada una las variables y cual es el tipo de retorno?. En caso de que haya varias signaturas explique una de ellas.

e) Explique con un ejemplo la condición de "prefijo" de un código.

f) Cuál es el costo de inserción en tablas de dispersión abiertas con listas desordenadas en el peor caso?. Diga cual es ese caso.

g) Que condiciones debe satisfacer un árbol binario para ser "árbol binario de búsqueda"?

h) Que quiere decir que un algoritmo de ordenamiento sea "estable"?. Dé ejemplos de algoritmos estables y no estables.

i) Por qué se les dice "lentos" a ciertos algoritmos de ordenamiento?. Liste los algoritmos lentos que conoce. Liste los algoritmos rápidos que conoce.

j) Discuta el número de inserciones que requieren los algoritmos de ordenamiento lentos en el peor caso.

[Ej. 2] [Clases (mínimo 60 %)]

a) Escribir los métodos que se indican del TAD `list<>` `insert(x,p)`, `erase(p)`, `retrieve(p)/*p`, `next()/p++`, `list()` y `~list()` implementados por celdas enlazadas por punteros o cursores. Defina también los miembros privados de la clase.

b) Implementar `set::iterator set::find(T x)` para conjuntos implementados por ABB. **No es necesario** escribir las declaraciones auxiliares de los miembros privados de la clase.

[Ej. 3] [Programación (mínimo 40 %)]

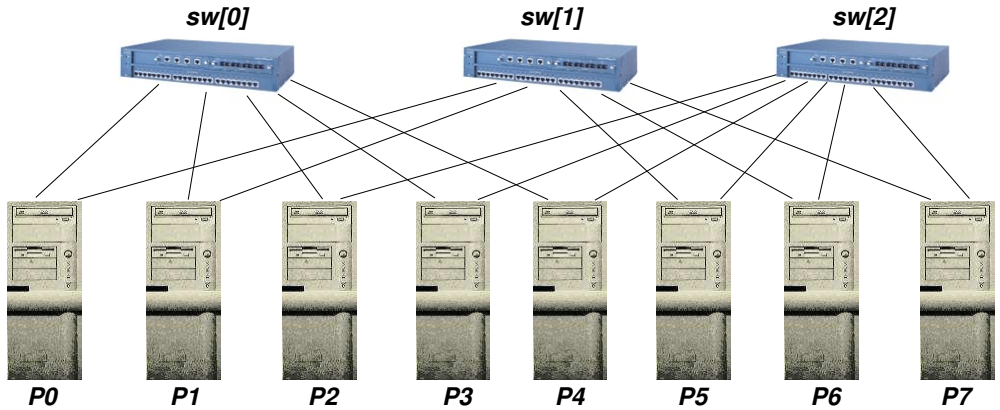
a) [max-dev-n] Dada una secuencia de números $\{a_1, a_2, ..., a_n\}$, vamos a decir que su *"máxima desviación"*, es la máxima diferencia (en valor absoluto) entre todos sus números: max_dev$(a_1, a_2, ..., a_n) = (\max_{j=1}^{n} a_j) - (\min_{j=1}^{n} a_j)$. Escribir una función **`int max_dev_m(list<int> &l, int m);`** que retorna el máximo de las máximas desviaciones de las subsecuencias de L de longitud m, es decir

$$\text{max_dev_m(L)} = \max\{\text{max_dev}(a_1, a_2, \ldots, a_m), \text{max_dev}(a_2, a_3, \ldots, a_{m+1}), \text{max_dev}(a_3, \ldots, a_{m+2}), \ldots, \text{max_dev}(a_{n-m+1}, \ldots, a_n)\} \quad (1)$$

Por ejemplo, si `L=(1,3,5,4,3,5)`, entonces **`max_dev_m(L,3)`** debe retornar 4 ya que la máxima desviación se da en la primera subsecuencia `(1,3,5)` y es 4. Se sugiere el siguiente algoritmo, para cada posición `p` en la lista hallar la máxima desviación de los `m` elementos siguientes (incluyendo a `p`). Hallar la máxima de estas desviaciones.

b) **[ord-nodo]**
Escribir una función predicado **`bool ordnodo(tree<int> &A);`** que verifica si cada secuencia de hermanos del subárbol del nodo **`n`** (perteneciente al árbol ordenado orientado **`A`**) estan ordenadas entre sí, de izquierda a derecha. Por ejemplo, para el árbol **`(3 5 (6 1 3) (7 4 5))`** debería retornar **`true`**, mientras que para **`(3 9 (6 1 3) (7 4 2))`** debería retornar **`false`**, ya que las secuencias de hermanos **`(9 6 7)`** y **`(4 2)`** NO están ordenados. Se sugiere el siguiente algoritmo: para un dado nodo retornar false si: 1) sus hijos no estan ordenados o 2) algunos de sus hijos contiene en su subárbol una secuencia de hermanos no ordenada (recursividad).

c) **[flat]** Se está diseñando una red inerconectada por switches y se desea, para reducir lo más posible la *latencia* entre nodos, que cada par de nodos esté conectado en forma directa por al menos un switch. Sabemos que el número de nodos es **`n`** y tenemos un **`vector< set<int> > sw`** que contiene para cada switch el conjunto de los nodos conectados por ese switch, es decir **`sw[j]`** es un conjunto de enteros que representa el conjunto de nodos inteconectados por el switch j.
Consigna: Escribir una función predicado **`bool flat(vector< set<int> > &sw, int n);`** que retorna verdadero si cada par de enteros (j, k) con $0 \leq j, k < n$ está contenido en al menos uno de los conjunto en **`sw[]`**.



En el ejemplo de la figura tenemos 8 nodos conectados via 3 switches y puede verficarse que cualquier par de nodos está conectado en forma directa a través de al menos un switch. Para este ejemplo el vector **`sw`** sería

$$\texttt{sw[0]} = \{0, 1, 2, 3, 4\}, \quad \texttt{sw[1]} = \{0, 1, 5, 6, 7\}, \quad \texttt{sw[2]} = \{2, 3, 4, 5, 6, 7\} \quad (2)$$

Por lo tanto **`flat(sw,8)`** debe retornar **`true`**. Por otra parte si tenemos

$$\texttt{sw[0]} = \{0,2,3,4\},\ \ \texttt{sw[1]} = \{0,1,5,7\},\ \ \texttt{sw[2]} = \{2,3,5,6,7\} \tag{3}$$

entonces los pares $(0,6)$, $(1,2)$, $(1,3)$, $(1,4)$, $(1,6)$, $(4,5)$, $(4,6)$ y $(4,7)$ no están conectados en forma directa y **`flat(sw,8)`** debe retornar **`false`**.
Sugerencia 1: Recorrer todos los pares de valores (j,k) y para cada par recorrer todos los conjuntos en **`sw[]`** hasta encontrar uno que contenga al par.
Sugerencia 2: Puede ser de ayuda el escribir una función auxiliar
`bool estan_conectados(sw,j,k)`.

[Ej. 4] [operativos (mínimo 70 %)]

a) **[rec-arbol]** Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- ORD_PRE $=\{C,Z,Q,R,A,M,P,K,L,T\}$,
- ORD_POST $=\{Z,A,P,M,K,L,R,T,Q,C\}$.

b) **[make-tree]** Escribir la secuencia de sentencias necesarias para construir el arbol binario **`(5 (2 . 3) (7 6 8))`**. (**Restricciones:** No usar **`find()`**).

c) **[quick-sort]:** Dados los enteros 3, 7, 9, 11, 10, 12, 6, 5, 7, 4 ordenarlos por el método de clasificación rápida (quick-sort). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

d) **[huffman]:** Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario (para todos los caracteres) y encodar la palabra ARGENTINA: $P(A) = 0.24$, $P(R) = 0.12$, $P(G) = 0.10$, $P(E) = 0.10$, $P(N) = 0.10$, $P(T) = 0.14$, $P(I) = 0.08$, $P(S) = 0.06$, $P(V) = 0.06$. Calcular la longitud promedio del código obtenido.

e) **[hash-dict]** Insertar los números $2, 15, 25, 8, 7, 35, 17, 4, 27$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x \bmod 10$ y estrategia de redispersión lineal.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2016-10-06]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS1` en una o más hojas **separadas**, `OPER1` en una o más hojas **separadas**, `PREG1` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 LOVELACE, ADA`

[Ej. 1] [CLAS1 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[lista]** Implemente los métodos:

```
1       iterator_t insert(T& x);
2       iterator_t erase(iterator_t p);
3       iterator_t erase(iterator_t p, iterator_t q);
```

de una lista doblemente enlazada por punteros, siendo:

```
1       class cell{
2         cell *next, *prev;
3         T elem;
4         cell() : next(NULL) {}
5         friend class list<T>;
6       }
7       typedef cell *iterator_t;
```

b) Para garantizar $T(n) = O(\log(n))$ en las búsquedas en un mapa dada una clave es necesario mantener ordenadas las mismas. Utilizando la implementación de mapa basada en vectores, implemente el método:

```
1       iterator insert(T_k key, T_v value)
```

Para ello utilice e implemente el método

```
1       iterator lower_bound(T_k key)
```

que retorna un iterador a:

- Si existe la clave, la posición en la que ya se encuentra.
- Si no existe la clave, la primera posición en la cual puede insertarse la nueva entrada, preservando el orden de las claves.

Considere: `typedef int iterator`

[Ej. 2] [OPER1 (W=20pt)]

a) **[rec-arbol (XXpt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD-PRE=(Z,Q,R,T,W,A,X,F,Y,U)`,
- `ORD-POST=(Q,R,A,W,F,X,U,Y,T,Z)`.

b) **[part-arbol (XXpt)]** Dado el árbol ordenado orientado (AOO):
`(Z Q (R (T U (V X) (W Y))))`
determinar cuales son los nodos **antecesores propios, descendientes propios, izquierda** y **derecha** del nodo `V`. ¿Son **disjuntos**? Justifique.

c) Un zoológico quiere construir hábitats naturales en los que exhibir los animales que dispone. Por desgracia, algunos animales se comerían a otros si se les diese la oportunidad. ¿Cómo utilizar un grafo para modelar esta situación? ¿Cómo utilizar una coloración del grafo para determinar el número de hábitats diferentes que se necesitan y la distribución de los animales en esos hábitats?

d) Utilizando sólo métodos **`insert, lchild, right`** e iteradores del AOO, complete el siguiente código que arma el árbol **`T=(3 (2 7 5) 9)`**

```
1       tree<int> T;
2       tree<int>::iterator n = T.insert(T.begin(),3);
3       ...
4       COMPLETAR
5       ...
```

e) Considere el siguiente algoritmo:

```
1       bool a(int n) {
2         if(n==0) return 0;
3         if(n==1) return 1;
4         return a(n-1)+2*a(n-2);
5       }
```

Halle la expresión recursiva de los términos de la sucesión que se implementa. Determine la complejidad algorítmica en notación $O(\cdot)$

[Ej. 3] [PREG1 (W=20pt)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = 5\log_2 n + \sqrt[3]{n} + 2\,n^3 + 5\,n^2$$
$$T_2 = 2n^2 + 2 \cdot 3^n + 7^4$$
$$T_3 = 3 \cdot 3^n + 4n^5 + 6\,n!$$
$$T_4 = \log_{10} n + 3$$
$$T_5 = 5^6 + 3.4\log_2 n + \log_4(16)n^{2.5}$$

b) Se puede insertar un elemento en una posición **dereferenciable** en una lista? Y en una **no-dereferenciable**?

c) Enuncie y justifique la **regla de la suma** para la notación $O(\cdot)$. De un ejemplo.

d) En una **correspondencia**: ¿puede haber una **misma clave** con **diferentes valores**? Por ej. **`M=(1->2, 1->3)`** ¿Puede haber **diferentes claves** con el **mismo valor**? Por ej. **`M=(1->2, 3->2)`**

e) En un árbol, ¿puede un nodo tener **más de un padre**? ¿Puede un nodo **no tener padre**?

f) En un Arbol Ordenado Orientado (AOO): ¿importa el **orden de los nodos**? Por ejemplo, ¿es lo mismo el árbol **`(a b c)`** que el árbol **`(a c b)`**?

g) ¿Cuantas posiciones **no dereferenciables** puede haber en una **lista**? ¿Y en un **árbol**?

h) Dado dos nodos de un Arbol Ordenado Orientado (AOO) **`a,b`**, ¿Puede ser **b descendiente** y **antecesor** de **a** al mismo tiempo? ¿Puede ser **b descendiente propio** y **antecesor propio** de **a** al mismo tiempo?

i) Supongamos que tenemos un árbol (AOO) **`T=(1 3 (5 6 8) 7)`** y un iterator **`p`** apuntando al 5. ¿Cómo queda **`T`** si hacemos una inserción: **`T.insert(p,4);`**

j) Supongamos que tenemos un árbol (AOO) **`T=(2 (4 6 8) (7 10))`** y un iterator **`p`** apuntando al 4. ¿Cómo queda **`T`** si hacemos una supresión: **`T.erase(p);`**

mstorti@galileo/aed-3.0-36-g76cec07/Wed Sep 28 14:15:44 2016 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do parcial. [17 de Noviembre de 2016]

[Ej. 1] [Clases (mínimo 50 %)]

a) **[set-vb]:** Dada la siguiente interfaz para un conjunto (set) por vector de bits que contiene letras mayúsculas (de `A` a `Z`):

```
const int N=26;
typedef char elem_t;
int index(elem_t t);
elem_t element(int j);
typedef int iterator_t;
typedef pair<iterator_t,bool> pair_t;

class set {
private:
    vector<bool> v;
    iterator_t next_aux(iterator_t p);
public:
    set();
    pair_t insert(elem_t x);
    elem_t retrieve(iterator_t p);
    void erase(iterator_t p);
    bool erase(elem_t x);
    iterator_t find(elem_t x);
    iterator_t begin();
    iterator_t end();
    iterator_t next(iterator_t p);
};
```

implemente los métodos: `begin`, `next` e `insert`, y las funciones `index` y `element` (si dentro de alguno de ellos utiliza otro método de set, deberá implementarlo también).

b) **[set-abb]:** Dada la siguiente interfaz para un conjunto (set) por vector árbol binario de búsqueda:

```
template<class T>
class set {
private:
    typedef btree<T> tree_t;
    typedef typename tree_t::iterator node_t;
    tree_t bstree;
    ...
public:
    class iterator {
        note_t node;
        tree_t &bstree;
    private:
            friend class set;
            iterator(tree_t &t, node_t n) : bstree(t), node(n) {}
            ...
    };
    typedef pair<iterator,bool> pair_t;
```

```
    pair_t insert(T x);
    void erase(iterator m);
    int erase(T x);
    iterator find(T x);
    iterator begin();
    iterator end();
    ...
};
```

implemente los métodos `find`, `insert`, y `begin` (si dentro de alguno de ellos utiliza otro método de set, deberá implementarlo también).

[Ej. 2] [Operativos (mínimo 50 %)]

a) **[Huffman]** Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario siguiendo el algoritmo de Huffman y encodar la palabra `ACLAMADA`.
P(A)=0.25, P(L)=0.1, P(R)=0.1, P(O)=0.15, P(M)=0.05, P(D)=0.05, P(S)=0.1, P(C)=0.15, P(E)=0.05
Calcular la longitud promedio del código obtenido. ¿Cual debería ser la probabilidad de cada caracter para minimizar la longitud del código de la palabra a encodar en el ejercicio?

b) **[quick-sort]:** Dados los enteros `{6, 10, 5, 2, 6, 11, 9, 4, 4, 3, 12, 7}` ordenarlos por el método de *clasificación rápida* (*quick-sort*). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

c) **[abb]:** Dados los enteros `{14, 8, 21, 3, 4, 11, 6, 5, 13, 2}` insertarlos, en ese orden, en un *árbol binario de búsqueda*. Mostrar las operaciones necesarias para eliminar los elementos 14, 8 y 3 en ese orden.

d) **[hash]:** Insertar los enteros `{0, 19, 25, 8, 7, 35, 17, 4}` en una tabla de dispersión cerrada con `B=10` cubetas, con función de dispersión `h(x) = (x+1)%B`. Luego eliminar el elemento `7` e insertar el elemento `25`, en ese orden. Mostrar la tabla resultante.

[Ej. 3] [Preguntas (mínimo 60 %)]

a) Si queremos generar un código binario de longitud fija para el conjunto de las letras mayúsculas y minúsculas (52 caracteres). ¿Cuantos bits tendrá, como mínimo, cada caracter?

b) ¿Cuantos nodos tiene un árbol binario completo de 4 niveles?

c) Explique que las operaciones que realizan (semántica) las dos versiones de erase para conjuntos.

1) `void erase(iterator p);`
2) `int erase(key_t x);`

Explicar que son los valores de retorno, y porqué una retorna un entero y la otra no.

d) Explique el concepto de estabilidad para algoritmos de ordenamiento.

e) ¿Cuál es el resultado de aplicar la función `l=particiona(w,j,k,v)`? (Nota: No se pide el algoritmo, sino cuál es el efecto de aplicar tal función, independientemende de como se programe). Explique los argumentos de la función. ¿Cuál debe ser el tiempo de ejecución para particiona() si queremos que quick_sort() sea un algoritmo rápido?

f) Discuta el número de inserciones que requieren los algoritmos de ordenamiento lentos en el peor caso.

g) Discuta el tiempo de ejecución de los algoritmos de ordenamiento.

h) ¿Cuál es el tiempo de ejecución del algoritmo de ordenamiento rápido (quick-sort), en el caso promedio y en el peor caso? ¿Cuando se produce el peor caso?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

FICH

UNL

Algoritmos y Estructuras de Datos.
Recuperatorio. [2016-11-24]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS1` en una o más hojas **separadas**, `OPER1` en una o más hojas **separadas**, `PREG1` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS1, Hoja #2/3 TORVALDS, LINUS

[Ej. 1] [CLAS1 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[map-vo]** Considerando la siguiente definición parcial de una clase map que mapea enteros a strings:

```
class map {
  vector<pair<int,string>> v;
  ...
  public:
  ...
  Iterator find(int x);
  pair<bool,iterator> insert(int x);
  int erase(int x);
  ...
};
```

defina el tipo `iterator`, e implemente los métodos `find`, `insert` y `erase` de manera tal que todas estas operaciones sean $O(\log(n))$. Si dentro de alguno de ellos utiliza otro método de `map`, deberá implementarlo también.

b) **[aoo]** Dada la siguiente definición de una clase para representar un árbol ordenado orientado:

```
class tree {
  cell *header;
  ...
  public:
  tree();
  ~tree();
  elem_t &retrieve(iterator_t p);
  iterator_t insert(iterator_t p,elem_t elem);
  iterator_t erase(iterator_t p);
  iterator_t splice(iterator_t to,iterator_t from);
  iterator_t find(elem_t elem);
  void clear();
  iterator_t begin();
  iterator_t end();
  ...
};
```

defina las clases `cell` e `iterator_t` e implemente todos sus métodos (`lchild`, `right` y operadores `!=` y ==).

[Ej. 2] [OPER1 (W=20pt)]

a) **[particiona]** Dado el AOO cuyo listado previo es `(L M R W P J S H B K N)` y posterior es `(R P W M H B K S N J L)`, determine los antecesores propios, descendientes propios, izquierda y derecha del nodo `S`. ¿Son conjuntos disjuntos? Justifique.

b) **[coloreo]** El departamento informático de la universidad se compone de los 6 comités que se Enumeran a continuación:

```
C1 = {Pablo, Mario}
C2 = {Romina, Jose, Pablo}
C3 = {Marcos, Mario}
C4 = {Laura, Juan}
C5 = {Jose, Mario}
C6 = {Pedro, Juan, Marcos}
```

Estos comités se reúnen un día específico al mes. ¿Existe una mejor solución que plantear 6 horarios diferentes para cada reunión de tal manera de que nadie tenga programada dos reuniones en simultáneo? Encuentre una mejor propuesta por medio de una coloración del grafo que modela el problema.

c) **[aoo]** Utilizando sólo los métodos **`insert()`**, **`right()`** y **`lchild()`** del TAD AOO y sus iteradores, complete el siguiente código para armar el árbol **`T=(4 9 (7 -1 2) 11)`**.

```
tree<int> T;
tree<int>::iterator node = T.insert(T.begin(), 4);
. . .
```

d) **[orden]** La siguiente función evalúa un polinomio cuyos coeficientes son **a** es decir $p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ y posee un método **`evalua`** el cual retorna el valor del polinomio evaluado con **`x=c`**, esto es **`p(c)`**;

```
double evalua_1 (vector<int> &a,double c) {
  double resultado= 0.0;
  for (int termino= 0; termino < a.size(); termino++) {
    double xn= 1.0;
    for (int j=0; j < termino; j++)
      xn *= c;          // x elevado a n
    resultado += a[termino] * xn;
  }
  return resultado;
}
```

Responda:

1) De qué orden algorítmico es el método **`evalua_1`**?
2) Proponga un método **`evalua_2`** que sea más eficiente reduciendo la complejidad algorítmica.

[Ej. 3] [PREG1 (W=20pt)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$T_1 = 2\log_{10} n + \sqrt[3]{n} + 4\,n^4 + 6\,n^2$$
$$T_2 = 2\,n^4 + 2\,n! + 3 \cdot 4^n$$
$$T_3 = 4 \cdot 4^n + 3^3 + 4n^3$$
$$T_4 = \log_2 n + 3$$
$$T_5 = 3.3\log n + 4^4 + \log(25)n^{2.5}$$

b) Explique la propiedad de transitividad de la notación asintótica $O(\,\cdot\,)$. De un ejemplo.

c) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase **`list<>`** implementada con **celdas simplemente enlazadas** (por punteros o cursores) en el caso promedio? *Métodos:* **`insert(p,x)`**, **`*p`**, **`erase(p)`**, **`clear()`**, **`begin()`**, **`end()`**.

d) ¿Es la **pila** un contenedor **FIFO** o **LIFO**? ¿Que significa? Responda la misma pregunta para el TAD **cola**.

e) ¿Cual es la principal ventaja de implementar correspondencias con **vectores ordenados**? ¿Se obtiene la misma ventaja con **listas ordenadas**?

f) Discuta si los algoritmos de **búsqueda exhaustiva** y **heurístico** para la **coloración de grafos** dan la solución **óptima** al problema. Señale la complejidad algorítmica de los mismos. ¿Cuál de los dos elegiría para un grafo con 100 vértices y 500 aristas? Justifique.

g) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase **`map<>`** implementada con **listas ordenadas** y **vectores ordenados** en el caso promedio? Métodos: **`find(key)`**, **`M[key]`**, **`erase(key)`**, **`erase(p)`**, **`begin()`**, **`end()`**, **`clear()`**.

h) Defina que es un **camino** en un árbol. Dado el AOO **`(z x (y (d e f)))`**, cuáles de los siguientes son caminos?

1) **`(y,d,e)`**,
2) **`(d,y,z)`**,
3) **`(e,d,f)`**,
4) **`(d,f)`**.

i) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un árbol ordenado orientado (AOO)? ¿Y en una posición **dereferenciable**?

j) ¿Como se define la **altura** de un nodo en un árbol? ¿Cuál es la altura de los nodos **`A`**, **`B`**, y **`C`** en el siguiente AOO: **`(C (B (T X Y)) (Z A (D P)))`**?

[Ej. 4] [CLAS2 (W=20pt)]

Recordar que **deben usar la interfaz STL**.

a) **[ab]** Siendo la siguiente una posible implementación de la clase btree, que representa un árbol binario:

```
class cell {
  friend class btree;
  friend class iterator_t;
  elem_t t;
  cell *right,*left;
};
class iterator_t {
  friend class btree;
  cell *ptr,*father;
  enum side_t {NONE,R,L};
  side_t side;
  iterator_t(cell *p,side_t side_a,cell *f_a);
public:
  iterator_t();
  bool operator!=(iterator_t q);
  bool operator==(iterator_t q);
  iterator_t left();
  iterator_t right();
};
class btree {
private:
  cell *header;
  ...
  public:
  btree();
  ~btree();
  elem_t & retrieve(iterator_t p);
  iterator_t insert(iterator_t p,elem_t t);
  iterator_t erase(iterator_t p);
  void clear();
  iterator_t begin();
  iterator_t end();
  ...
}
```

implemente el constructor de la clase **`btree`** y los métodos **`insert`** y **`erase`**.

b) **[set-lo]** Dada la siguiente interfaz de una clase set que implementa un conjunto mediante una lista ordenada:

```
class set {
  list<elem_t> L;
public:
  elem_t retrieve(iterator_t p);
  iterator_t lower_bound(elem_t t);
  iterator_t next(iterator_t p);
  pair<iterator_t,bool> insert(elem_t x);
  void erase(iterator_t p);
  void erase(elem_t x);
  void clear();
  iterator_t find(elem_t x);
  iterator_t begin();
```

```
    iterator_t end();
    int size();
    friend void set_union(set &A,set &B,set &C);
    friend void set_intersection(set &A,set &B,set &C);
    friend void set_difference(set &A,set &B,set &C);
};
```

Implementar las funciones **set_union**, **set_intersection** y **set_difference** de forma que sean todas $O(n)$.

[Ej. 5] [OPER2 (W=20pt)]

a) **[huffman]** Al procesar un archivo de texto de 500 caracteres se contabilizaron 100 "**A**", 50 "**I**", 50 "␣" (espacios o caracteres en blanco), 75 "**R**", 25 "**P**", 100 "**O**", 50 "**E**", 25 "**S**" y 25 "**L**". Se pide que se construya el árbol de la codificación de Huffman óptima para ese texto y se codifique la frase **EL␣PERRO**. ¿Cuál es la longitud promedio de la codificación obtenida?

b) **[hash]** Insertar los enteros **{3, 7, 13, 25, 9, 11, 7, 17}** en una tabla de dispersión cerrada con **B=12** cubetas, con función de dispersión **h(x) = x%B** y función de redispersión **d(x) = x-1**. Luego elimine el elemento 25 e inserte el elemento 4. Muestre la tabla resultante en cada paso.

c) **[abb]** Dados los enteros **{15, 7, 11, 23,18, 19, 9, 30, 1}** insertarlos en ese orden en un árbol binario de búsqueda. Mostrar las operaciones necesarias para eliminar los elementos 23 y 7 en ese orden.

d) **[quick-sort]** Dados los enteros **{17, 6, 5, 9, 11, 16, 22, 1, 11, 18}** ordenarlos por el método de clasificación rápida (quick-sort). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor entre los dos primeros elementos distintos.

[Ej. 6] [PREG2 (W=20pt)]

a) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.

b) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.

c) Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**.

d) Se quiere representar el conjunto de enteros múltiplos de 3 entre 30 y 99 (o sea $U = \{30, 33, 36, ..., 99\}$) por **vectores de bits**, escribir las funciones **indx()** y **element()** correspondientes.

e) Discuta la complejidad algorítmica de las operaciones binarias **set_union(A,B,C)**, **set_intersection(A,B,C)**, y **set_difference(A,B,C)** para conjuntos implementados por vectores de bits, donde **A**, **B**, y **C** son subconjuntos de tamaño n_A, n_B, y n_C respectivamente, de un conjunto universal **U** de tamaño N.

f) ¿Cuál es el costo de **inserción exitosa** en tablas de dispersión abiertas?

g) Se quiere representar conjuntos de pares de enteros en el rango **[0,10)** es decir por ejemplo **(3,7)**, **(9,0)**, por **vectores de bits**. ¿Cuál es el tamaño **N** del conjunto universal? Completar las funciones **indx()** y **element()** correspondientes:

```
1       int indx(pair<int,int> p) { /* ... */ }
2       pair<int,int> element(int z) { /* ... */ }
```

h) Si la correspondencia **map<int,string> M** contiene **M={1->"Romero",6->"Biglia"}** y ejecutamos el código **string z= M[6]**. ¿Que valor toma **z**? ¿Cómo queda **M**? Si ahora hacemos **z = M[10]**, ¿cómo quedan **z** y **M**?

i) Si queremos generar un código binario de longitud fija para el conjunto de letras minúsculas, mayúsculas, y dígitos (en total 26+26+10=62 caracteres. ¿Cuantos bits tendrá, como **mínimo**, la representación de cada caracter? ¿Y para el conjunto de caracteres mayúsculas y dígitos (26+10=36)?

j) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un árbol binario (AB)? ¿Y en una posición **dereferenciable**?

Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio. [2016-10-13]

PASSWD PARA EL ZIP: `LYW2 P51S EWPB`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[unordered-equal (33pt)]** Escribir una función bool que reciba dos Arboles Ordenados Orientados (AOO) y retorne **`true`** si son "desordenadmente" iguales. Se dice que dos árboles son "desordenadamente" iguales si:

(a) las raíces de ambos son iguales,
(b) el conjunto de hijos de la raíz de uno es igual al conjunto de hijos de la raíz del otro, y
(c) la condición se cumple recursivamente para cada par de subarboles de cada par de nodos equivalentes (uno de cada conjunto).

Notar que en un "conjunto" no importa el orden, entonces la condición (b) implica que las raíces tienen los mismo hijos, aunque podrían estar en diferente orden. Por ejemplo, los árboles **`(1 (2 5 6) 3 (4 7 8))`** y **`(1 (4 7 8) 3 (2 6 5))`** son desordenadamente iguales, ya que en ambos árboles los hijos de 1 son 2, 3 y 4 (aunque en diferente orden), los hijos de 2 son 5 y 6 (en dif orden), los hijos de 4 son 7 y 8, y el resto no tiene hijos.
NOTA: Como prerequisito se asume que todos los hermanos son diferentes entre sí. Es decir el árbol **`(1 2 3 2)`** no satisface el prerequisito ya que hay dos elementos **`2`** entre los hijos de **`1`**.
Algoritmo sugerido: sean **`A`** y **`B`** dos árboles de enteros (**`tree<int>`**), dados dos iteradores **`itA`** e **`itB`** apuntando a las raíces de **`A`** y **`B`** respectivamente:

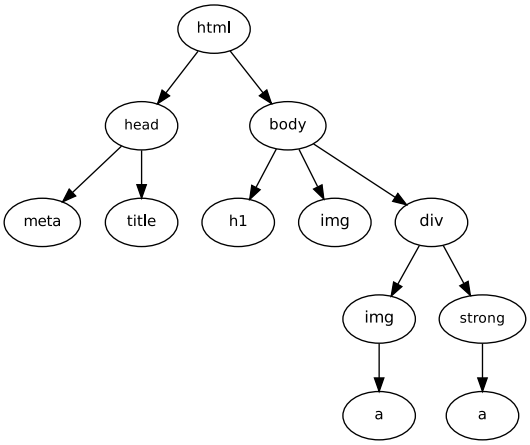
- Colocar todos los hijos del nodo **`itA`** en un mapa **`MA`**, donde las claves sean la etiquetas de dichos hijos, y los valores sean iteradores apuntando a los mismos.
- Colocar todos los hijos del nodo **`itB`** en un mapa **`MB`**, donde las claves sean la etiquetas de dichos hijos, y los valores sean iteradores apuntando a los mismos.
- Si los mapas **`MA`** y **`MB`** tienen diferente tamaño retornar **`false`**.
- Recorrer en simultáneo **`MA`** y **`MB`**, y por cada par de pares clave-valor (uno de **`MA`** y otro de **`MB`**):
  - Si las etiquetas (claves) son diferentes retornar **`false`**
  - Aplicar el algoritmo recursivamente utilizando los dos iteradores (valores), y si retorna falso, retornar falso.
- Retornar verdadero

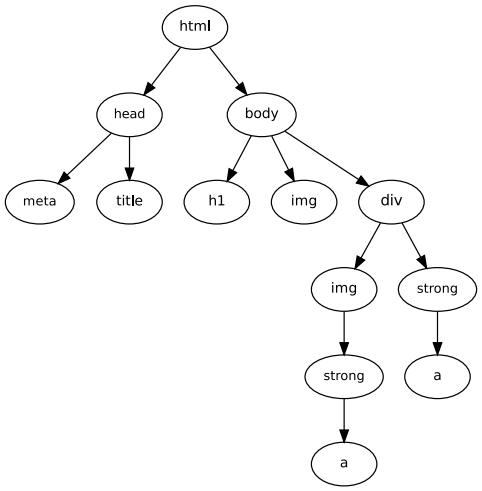
[Ej. 2] **[hay-camino (33pt)]** Un viajante quiere viajar desde una ciudad a otra siguiendo algún camino del grafo conexo de rutas **`M`**. Lamentablemente se tiene la información de que en algunas ciudades hay piquetes y es imposible pasar por ellas. Para determinar si es posible realizar el viaje se debe implementar una función,
`bool hay_camino(map<string,list<string>>&M, list<string>&P, string cini, string cend);`
que recibe en **`M`** el mapa de rutas disponibles (cada arista del grafo representa una ruta directa disponible entre las ciudades de los vértices que conecta), y en **`P`** la lista de ciudades con piquetes. La función debe retornar verdadero si existe alguna ruta que comience en la ciudad **`cini`** y finalice en **`cend`** sin pasar por ninguna de las ciudades con piquetes.

[Ej. 3] **[enhance-html (33pt)]** Los desarrolladores de un sitio web desean resaltar los links que aparecen dentro de cada página del sitio. Para ello es necesario que cada link (tag **<a>** en HTML) se encuentre dentro de un tag ****. Para resolver este problema ya contamos con un parser del código HTML que lo representa en un **tree<string>**. Por ejemplo, para el codigo HTML:

```
<html>
<head>
  <meta charset="UTF-8">
  <title>Ejemplo</title>
</head>
<body>
  <h1> Página de Ejemplo </h1>
  <img src="ejemplo.jpeg">
  <div>
    <a href="www.google.com.ar">
    <strong><a href="www.facebook.com.ar"></strong>
  <div>
</body>
</html>
```

el AOO (**tree<string>**) asociado es
T=(html (head meta title) (body h1 img (div (a img a) (strong a)))).





Se pide entonces implementar la función
void enhance_HTML(tree<string>&T); que recibe un AOO **T** con el parsing de alguna página web y lo manipula para obtener la estructura con el diseño modificado, es decir, todo tag **a** debe ser hijo de un tag **strong**. Notar que un link ya puede estar resaltado y que los links no pueden contener otros tags dentro suyo.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.

- Se incluye un template llamado **program.cpp**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **f** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

mstorti@galileo/aed-3.0-45-g19b81ae/Thu Oct 13 08:38:45 2016 -0300

TPL2. Trabajo Práctico de Laboratorio. [2016-10-13]. TABLA SEED/HASH

```
S=123 -> H1=457 H2=026 H3=261
S=577 -> H1=835 H2=342 H3=197
S=393 -> H1=622 H2=562 H3=619
S=686 -> H1=920 H2=965 H3=857
S=349 -> H1=633 H2=842 H3=204
S=762 -> H1=099 H2=959 H3=686
S=690 -> H1=663 H2=718 H3=729
S=663 -> H1=313 H2=577 H3=238
S=340 -> H1=857 H2=027 H3=103
S=872 -> H1=106 H2=597 H3=206
S=711 -> H1=801 H2=540 H3=307
S=367 -> H1=563 H2=226 H3=454
S=882 -> H1=895 H2=553 H3=151
S=162 -> H1=680 H2=410 H3=297
S=767 -> H1=227 H2=693 H3=931
S=429 -> H1=937 H2=308 H3=953
S=622 -> H1=123 H2=822 H3=341
S=969 -> H1=405 H2=183 H3=330
S=893 -> H1=622 H2=256 H3=917
S=311 -> H1=440 H2=932 H3=079
S=529 -> H1=607 H2=428 H3=712
S=721 -> H1=765 H2=336 H3=669
S=384 -> H1=012 H2=284 H3=811
S=798 -> H1=225 H2=492 H3=553
S=615 -> H1=661 H2=708 H3=520
```

```
S=386 -> H1=890 H2=687 H3=826
S=215 -> H1=288 H2=391 H3=933
S=935 -> H1=348 H2=194 H3=993
S=292 -> H1=463 H2=730 H3=436
S=821 -> H1=070 H2=662 H3=761
S=527 -> H1=374 H2=545 H3=866
S=359 -> H1=730 H2=175 H3=355
S=626 -> H1=259 H2=652 H3=150
S=226 -> H1=378 H2=052 H3=482
S=236 -> H1=939 H2=371 H3=944
S=468 -> H1=162 H2=188 H3=518
S=529 -> H1=607 H2=428 H3=712
S=630 -> H1=228 H2=989 H3=545
S=923 -> H1=973 H2=697 H3=430
S=335 -> H1=526 H2=427 H3=397
S=802 -> H1=768 H2=227 H3=594
S=958 -> H1=923 H2=868 H3=865
S=967 -> H1=285 H2=409 H3=215
S=656 -> H1=424 H2=547 H3=128
S=242 -> H1=383 H2=008 H3=390
S=973 -> H1=373 H2=027 H3=567
S=219 -> H1=123 H2=190 H3=517
S=437 -> H1=098 H2=955 H3=885
S=624 -> H1=324 H2=985 H3=789
S=670 -> H1=865 H2=177 H3=038
```

Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio. [2016-11-03]

PASSWD PARA EL ZIP: 6N4V CLD8 2E43

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[puede-simplificar (33pt)]** Se utiliza un árbol binario `btree<string> T` para representar una expresión matemática, donde los nodos interiores representan operadores binarios, y los nodos hoja operandos (variables o constantes).
Por ejemplo, la expresión `3*q+3*(f-1)` se representa con el árbol `(+ (* 3 q) (* 3 (- f 1)))`".
Escriba una función `bool puede_simplificar(btree<string> T, string fc);` para determinar si la expresión que representa el árbol puede simplificarse extrayendo `fc` como factor común.
Notar que: `fc` es un factor común válido si: o bien a) el nodo raíz del árbol es un producto, y uno de sus dos hijos es el valor `fc`, o bien b) la etiqueta del nodo raíz es `+` o `-` y `fc` es un factor común válido para los dos subárboles que cuelgan a derecha e izquierda del mismo.
Algoritmo sugerido:
Dado un árbol `T`, un iterador `it`, y un factor `fc`:

- Si `it` apunta a una hoja,
  - Si su etiqueta es `fc` retornar verdadero
  - Sino retornar falso
- Si la etiqueta de `it` es `*`
  - Invocar la función recursivamente para ambos hijos, y
  - Si alguna de las llamadas recursivas retorna verdadero $\Longrightarrow$ retornar verdadero
- Si la etiqueta de `it` es `+` o `-`
  - Invocar la función recursivamente para ambos hijos, y
  - si ambas llamadas recursivas retornan verdadero $\Longrightarrow$ retornar verdadero
- Retornar falso

[Ej. 2] **[prune-and-return (33pt)]**
Implemente la función `void prune_and_return(btree<int>&T, int v, list<int>& L);` que dado un árbol binario `T` busque el nodo cuya etiqueta sea `v` y:

- Retorne en `L` la lista en preorden de todos los nodos del subárbol que lo tiene a él como raíz.
- Elimine el nodo del árbol.

Notar que si existen varios nodos con la misma etiqueta, el proceso debe hacerse sobre el primero de estos nodos al listar el árbol en orden prefijo. Si el nodo no existe se retorna la lista L vacía.
Hints:

- Utilice una función auxiliar `void preorden(btree<int>&T, btree<int>::iterator it, list<int>& L);` que almacene la etiqueta de los nodos del subárbol de `T` con raíz en `it` en la lista `L`.
  El algoritmo recursivo es el siguiente:
  Si el subárbol no es vacío:
  - Si la lista a completar aún está vacía

    - Si la etiqueta es la buscada
        - Listar el subárbol en preorden
        - Eliminar el subarbol
  - Sino
    - Verificar el subárbol izquierdo
    - Verificar el subárbol derecho

Ejemplos:

- Si recibe `T= (8 (10 4 2) (5 . (7 9 10)))` con `v=5`, retorna `T = (8 (10 4 2) .), L = (5 7 9 10)`
- Si recibe `T= (8 (10 4 2) (5 . (7 9 10)))` con `v=15`, retorna `T = (8 (10 4 2) (5 . (7 9 10))), L = ()`
- Si recibe `T= (8 (10 4 2) (5 . (7 9 10)))` con `v=8`, retorna `T = (), L = (8 10 4 2 5 7 9 10)`

[Ej. 3] [gather-sets (33pt)]

Dado un vector de conjuntos `vector<set<int>> VS;` y un predicado `bool pred(int);` retornar la unión `set<int> S` de todos los conjuntos que contienen al menos un elemento que satisface el predicado.
`void gather_sets(vector<set<int>> &VS,bool (*pred)(int),set<int> &S);`
Ayuda:

- Limpiar `S`
- Para conjunto `R` de `VS`, verificar si `R` contiene algún elemento que satisface el predicado.
- Si contiene tal elemento hacer `S = S` $\cup$ `R` (Acordarse de que se en este caso se debe usar un temporario en las operaciones binarias).

Ejemplo:
Si `VS=[{1,3}{1,2,3},{2},{2,3,4},{2,4,6,8},{3,5,7}]` entonces

- `gather_sets(VS,even,S) => S={1,2,3,4,6,8}`
- `gather_sets(VS,odd,S) => S={1,2,3,4,5,7}`

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el

argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
   "T1": "( 0 (1 2) (3 4 5 6) )",
   "T2": "( 0 (2 4) (6 8 10 12) )",
   "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

mstorti@galileo/aed-3.0-58-g66c7314/Wed Nov 2 14:14:06 2016 -0300

TPL3. Trabajo Práctico de Laboratorio. [2016-11-03]. TABLA SEED/HASH

```
S=123 -> H1=961 H2=891 H3=140
S=577 -> H1=802 H2=161 H3=073
S=393 -> H1=493 H2=850 H3=726
S=686 -> H1=677 H2=068 H3=527
S=349 -> H1=764 H2=586 H3=094
S=762 -> H1=662 H2=724 H3=605
S=690 -> H1=262 H2=686 H3=198
S=663 -> H1=560 H2=392 H3=178
S=340 -> H1=741 H2=765 H3=439
S=872 -> H1=512 H2=785 H3=651
S=711 -> H1=672 H2=043 H3=912
S=367 -> H1=501 H2=497 H3=498
S=882 -> H1=505 H2=198 H3=602
S=162 -> H1=524 H2=815 H3=734
S=767 -> H1=357 H2=766 H3=725
S=429 -> H1=727 H2=734 H3=453
S=622 -> H1=317 H2=279 H3=870
S=969 -> H1=463 H2=413 H3=328
S=893 -> H1=504 H2=253 H3=871
S=311 -> H1=334 H2=373 H3=195
S=529 -> H1=097 H2=428 H3=338
S=721 -> H1=295 H2=875 H3=372
S=384 -> H1=877 H2=778 H3=479
S=798 -> H1=141 H2=952 H3=650
S=615 -> H1=377 H2=849 H3=096
S=386 -> H1=626 H2=046 H3=324
S=215 -> H1=317 H2=507 H3=361
S=935 -> H1=106 H2=260 H3=285
S=292 -> H1=685 H2=918 H3=178
S=821 -> H1=156 H2=863 H3=884
S=527 -> H1=017 H2=709 H3=557
S=359 -> H1=851 H2=797 H3=847
S=626 -> H1=719 H2=119 H3=618
S=226 -> H1=978 H2=893 H3=823
S=236 -> H1=755 H2=306 H3=748
S=468 -> H1=300 H2=191 H3=308
S=529 -> H1=097 H2=428 H3=338
S=630 -> H1=939 H2=208 H3=956
S=923 -> H1=533 H2=709 H3=323
S=335 -> H1=364 H2=155 H3=784
S=802 -> H1=568 H2=778 H3=859
S=958 -> H1=067 H2=936 H3=424
S=967 -> H1=915 H2=338 H3=882
S=656 -> H1=950 H2=354 H3=546
S=242 -> H1=331 H2=261 H3=679
S=973 -> H1=274 H2=680 H3=658
S=219 -> H1=400 H2=620 H3=268
S=437 -> H1=391 H2=673 H3=449
S=624 -> H1=900 H2=712 H3=185
S=670 -> H1=231 H2=947 H3=166
```

Algoritmos y Estructuras de Datos.
Recuperatorio de Trabajos Prácticos de Laboratorio. [2016-11-10]

PASSWD PARA EL ZIP: `N28G 02VV FQFD`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[max_sublist_m (tpl1 - 25pts)]** Programar una función `int max_sublist_m(list<int> &L, int m)` que reciba una lista de enteros `L` y un entero `m` tal que `0<m<=L.size()`, y encuentre y retorne el valor de la mayor suma de entre todas las posibles sumas de sublistas de longitud m.
Por ejemplo: Si `L: {1,2,-5,4,-3,2}`

- `m=1` ⇒ `4` (por `{4}`)
- `m=2` ⇒ `3` (por `{1,2}`)
- `m=3` ⇒ `3` (por `{4,-3,2}`)
- `m=4` ⇒ `2` (por `{1,2,-5,4}`)
- `m=5` ⇒ `0` (por `{2,-5,4,-3,2}`)
- `m=6` ⇒ `1` (por `{1,2,-5,4,-3,2}`)

[Ej. 2] **[remove_max_sibling (tpl2 - 25pts)]** Basado en la historia bíblica de las 7 plagas de Egipto. En la última plaga Dios ordenó matar todo primogénito Egipcio. Suponga ahora que recibe un árbol `T` con las edades de una familia egipcia, dado un grupo de hermanos, elimine al de mayor edad. Al finalizar el algoritmo, en `T` debe quedar las edades de los sobrevivientes de la familia. Implemente la función `void remove_max_sibling(tree<int>&T)` que dado un AOO `T`, por cada grupo de hermanos elimine aquel de etiqueta mayor que no tenga descendencia. La eliminación debe realizarse en post-orden.
Ayuda:

- Dado un nodo `it`
  - Buscar la etiqueta máxima entre sus hijos sin descendencia (si los hay)
  - Si había hijos sin descendencia, eliminar el de la etiqueta máxima
  - Aplicar recursivamente sobre los descendientes de los hijos no eliminados

Ejemplos:

- `(50 (25 7 4) 35 20)` ⇒ `(50 (25 4) 20)`
- `(50 25 (35 7 4) 20)` ⇒ `(50 (35 4) 20)`

[Ej. 3] **[max_siblings_sum (tpl1/tpl2 - 50pts)]** Programar una función `void max_siblings_sum(tree<int>&T,list<int>&L)` que reciba un AOO y retorne la lista de nodos hermanos (ordenados desde el más izquierdo) que obtenga la mayor suma entre todas sus etiquetas. Si hay varias listas que den la misma suma, se debe retornar la primera en preorden.
Por ejemplo:

- `T = (2 (5 7 2 3)(-1 9))` ⇒ `L = {7,2,3}`
- `T = (2 (5 -7 2 3)(-1 9))` ⇒ `L = {9}`
- `T = (2 (5 -7 2 3)(-1 -9))` ⇒ `L = {5,-1}`
- `T = (2 (5 7 2 3)(-1 12))` ⇒ `L = {7,2,3}`

[Ej. 4] **[max_valid_path (tpl2/tpl3 - 50pts)]** Implementar la función `int max_valid_path(map<int,set<int>>& G, bool (*pred)(int))` que recibe un grafo no dirigido `G` y retorna la longitud del camino más largo (sin repetir vértices) tal que cada uno de los nodos que lo compone satisface el predicado `pred`. Recordar que la longitud de un camino es el número de aristas que lo compone. Si el grafo no tiene ningún nodo que cumpla con el predicado, la función debe retornar `-1`. Por ejemplo, si `G` es:

```
1 -> {2, 4}
2 -> {1, 3, 4}
3 -> {2, 4}
4 -> {1, 2, 3}
```

entonces:

- Predicado `es_impar` ⇒ `0` (desde un nodo impar no hay camino hacia otro impar)
- Predicado `es_par` ⇒ `1` (el camino 2 - 4 o viceversa)
- Predicado `es_menor_a_5` ⇒ `3` (por ejemplo el camino 1 - 2 - 3 - 4)

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- `ucase`: Además las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere "hacer todos". Por ejemplo `ev.eval4(prune_to_level,1,51);` corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función `eval<j>` están almacenados en un archivo `test1.json` o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo.

datain son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
  "T1": "( 0 (1 2) (3 4 5 6) )",
  "T2": "( 0 (2 4) (6 8 10 12) )",
  "func": "doble" },
 "output": { "retval": true },
 "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

Algoritmos y Estructuras de Datos.
TPLSR16. Recuperatorio de Trabajos Prácticos de Laboratorio. [2017-02-09]

PASSWD PARA EL ZIP: X5NQ HJ6Q F6KD

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[filter-sets]** Dada una lista de conjuntos `LS_IN` de enteros, y una función `int f(int)`, copiar en otra lista de conjuntos `LS_OUT` solo aquellos conjuntos para los cuales la función `f` es inyectiva (esto es, que genera imágenes diferentes para todos los elementos del conjunto).
Ejemplo: Si `LS_IN=[ (1,2,3), (2,-2,10), (0,5) ]` y `f` es `f(x)=x^2` entonces `LS_OUT` debería quedar `[ (1,2,3), (0,5) ]`, ya que para el segundo conjunto de la lista de entrada hay dos elementos (`2` y `-2`) que generan la misma imagen (`4`), razón por la cual ese conjunto no se omite.
Ayuda:

- Por cada conjunto de `S` de `LS_IN`
  - Generar un conjunto auxiliar con las imágenes de la función aplicada a sus elementos
  - Comparar los tamaños de ambos conjuntos para determinar si es debe insertar `S` en `LS_OUT`.

[Ej. 2] **[distancia]** Dado un grafo no ponderado `G`, un vértice de partida `x` y un vértice de llegada `y`, implementar el método `int distancia(map<int,list<int>>& G, int x, int y)` que retorna la distancia entre los vértices pasados como parámetros. Recordar que la distancia entre dos vértices de un grafo no ponderado se puede definir como el número de aristas del camino más corto que los une. Se asegura que los vértices pasados como argumentos pertenecen al grafo.
Ayuda:

- Chequear si el vértice actual es el vértice de llegada
- En caso positivo retornar la distancia acumulada hasta el momento
- En caso negativo:
  - Marcar el vértice como visitado
  - Lanzar la recursion a cada uno de los vértices vecinos aún no visitados incrementando la distancia actual en 1. Retornar la menor distancia hallada.
  - Si todos los vecinos ya fueron visitados retornar un valor de distancia muy grande (1000) que indique que no se pudo hallar el vértice de llegada por este camino.

Por ejemplo: Si `G[1] -> {2 ,3}, G[2] -> {1 ,3}, G[3] -> {1 ,4}, G[4] -> {3}` debe dar,

```
distancia(G,1,1) -> 0
distancia(G,2,4) -> 2
distancia(G,1,4) -> 2
distancia(G,4,3) -> 1
```

[Ej. 3] **[suma-en-la-hoja]** Implemente el método `void suma_en_la_hoja(tree<int>& T)` que agregue un hijo a cada hoja del árbol ordenado orientado `T` cuya etiqueta sea la suma de las etiquetas de los vértices que componen el camino desde la raíz hasta dicha hoja. Un árbol vacío retorna un árbol vacío.

Ejemplo: Si `T = (5 (2 1 3) 7)` , luego de `suma_en_la_hoja(T)` el árbol debe quedar:
`(5 (2 (1 8) (3 10)) (7 12))`
Ayuda:

- Implementar una función recursiva que reciba el árbol `T`, un iterador `it`, y un entero `s` que deberá contener la suma de las etiquetas del camino desde la raíz hasta el padre del nodo apuntado por `it`.
  - Si `it` apunta a un nodo válido, la función debe sumar a `s` la etiqueta de `it` e invocarse recursivamente para cada hijo.
- Si `it` apunta a un nodo lambda, la función debe insertar la suma en dicha posición. La recursión comienza con `it` apuntando a la raíz y `s=0`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- `ucase`: Además las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere "hacer todos". Por ejemplo `ev.eval4(prune_to_level,1,51);` corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función `eval<j>` están almacenados en un archivo `test1.json` o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. `datain` son los datos pasados a la función y `output` la salida producida por la función de usuario. `ucase` es el número de caso.

```
{ "datain": {
  "T1": "( 0 (1 2) (3 4 5 6) )",
  "T2": "( 0 (2 4) (6 8 10 12) )",
  "func": "doble" },
```

```
"output": { "retval": true },
"ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2017-10-05]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS1` en una o más hojas **separadas**, `OPER1` en una o más hojas **separadas**, `PREG1` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS1 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) Dada la siguiente implementación de celdas para una lista simplemente enlazada:

```
template<class T> class cell{
  friend class list;
  T elem;
  cell* next;
  cell() : next(NULL) {}
};
typedef cell *iterator;
```

Implementar los métodos:

```
iterator insert(iterator p, T elem);
iterator erase(iterator p);
iterator erase(iterator p, iterator q);
```

Nota: La implementación debe ser tal que las dos primeras (`insert(p,x)` y `erase(p)`) deben ser $O(1)$.

b) En una implementación de Árbol Ordenado Orientado AOO en donde cada nodo del árbol es representado por una estructura como la siguiente:

```
template<class T> struct cell{
  cell * left_child, right;
  T elem;
}
```

- ¿Por qué es insuficiente utilizar iteradores que sean `typedef cell* iterator`? Presente un ejemplo que lo demuestre.
- Proponga una definición de iterador que solucione el/los problemas mencionados en el punto anterior.

[Ej. 2] [OPER1 (W=20pt)]

a) **[rec-arbol (5pt)]** Dibujar el Arbol Ordenado Orientado AOO cuyos nodos, listados en orden previo y posterior son

- `ORD-PRE=(W,A,B,C,T,U,M,N)`,
- `ORD-POST=(A,T,M,N,U,C,B,W)`,

b) Dada la siguiente función:

```
map<int,string>::iterator
        f1(map<int,string>&M, list<int>&L){
  auto it = L.begin();
  while(it!=L.end()){
    auto m = M.find(*it);
    if(m!=M.end())
    return m;
    it++;
  }
  return M.end();
}
```

Determine el tiempo de ejecución **`T(n)`** para el peor caso, mejor caso y el caso promedio.

Nota: Asumir que la correspondencia está implementada con **vectores ordenados.**

c) **[part-arbol (5pt)]** Dado el Árbol Ordenado Orientado (AOO):
`(A B C (D F (G H)) (E I))`
determinar cuales son los nodos **antecesores propios, descendientes propios, izquierda** y **derecha** del nodo **`D`**. ¿Son **disjuntos**? Justifique.

d) **[hacer-arbol (5pt)]** Utilizando sólo métodos **`insert, lchild, right`** e iteradores del Arbol Ordenado Orientado AOO, complete el siguiente código que arma el árbol **`T=(7 9 (8 3))`**

```
tree<int> T;
tree<int>::iterator n = T.insert(T.begin(),7);
...
COMPLETAR
...
```

[Ej. 3] [PREG1 (W=20pt)]

a) Notación $O(\cdot)$.

- Para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$).

$$T_1 = 6\,n^4 + 3\log_{10} n + \sqrt[4]{n} + 4\,n^3,$$
$$T_2 = 3 \cdot 2^n + 4\,n^3 + 2\,n!,$$
$$T_3 = 4^4 + 2 \cdot 4^n + 2n^3,$$
$$T_4 = \log_{10} n + 3,$$
$$T_5 = \log(15)n^{1.5} + 3^2 + 2.4\log n.$$

- Ordenar las funciones por tiempo de ejecución, es decir ponerlas en orden de la siguiente forma: $T_x < T_y < T_z$....

b) ¿Cuál es el tiempo de ejecución (mejor/promedio/peor) para el algoritmo de **búsqueda binaria** en un vector ordenado?

c) Considerando la implementación de pila con listas simplemente enlazadas, ¿Cuál es el tiempo de ejecución (mejor/promedio/peor) de **pop**, **push** y **`top`** cuando el tope de la pila está en el comienzo de la lista? ¿Y cuando el tope está al final?

d) Sea el árbol **`(z (q p r) (d (e a b)))`**. Cuáles de los siguientes son **caminos**?

- **`(z d e)`**
- **`(e d z)`**
- **`(q r)`**

- `(a e b)`

e) ¿Existe una relación biunívoca entre un Arbol Ordenado Orientado (AOO) y su **orden previo**? ¿Y con el **orden posterior**? ¿Y con ambos a la vez? De ejemplos.

f) Explique qué se entiende por algoritmos de **búsqueda exhaustiva** y **heurístico**. Discuta cuales son las ventajas y desventajas.

g) Exprese la **regla del producto** para la notación $O(.)$. De un ejemplo.

h) Dado el siguiente AOO `(15 (2 (7 11 20 (24 1))) (8 9))`

- ¿Donde queda apuntado **n** después de las siguientes operaciones?

```
1          n = T.find(2);
2          n = n.lchild();
3          n = n.lchild();
4          n++;n++;
5          n = n.lchild();
```

- ¿Como queda el árbol al hacer `T.erase(T.find(7));` ?

i) Comente ventajas y desventajas del uso de listas **doblemente enlazadas** con respecto a **simplemente enlazadas**.

j) Discuta si es posible insertar en una **posición dereferenciable** en Arbol Ordenado Orientado (AOO). ¿Y en una **no dereferenciable**?

mstorti@galileo/aed-3.1-55-g2e9e4612/Fri Oct 6 07:28:42 2017 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
2do parcial. [16 de Noviembre de 2017]

[Ej. 1] [Clases (mínimo 50 %)]

a) **[ab]:** Dado el siguiente resumen de la implementación de árbol binario

```
class btree{
  class cell {
    elem_t t;
    cell *right,*left;
    cell() : right(NULL), left(NULL) {}
  };
  class iterator {
  private:
    cell *ptr,*father;
    side_t side;
    iterator(cell *p,side_t side_a,cell *f_a)
    : ptr(p), side(side_a), father(f_a) { }
  public:
    iterator(const iterator_t &q) { ... }
    bool operator!=(iterator_t q) { ... }
    bool operator==(iterator_t q) { ... }
    iterator() : ptr(NULL), side(NONE),
    father(NULL) { }
    iterator left() { ... }
    iterator right() { ... }  //-->implementar
  }
  ...
  iterator splice(iterator to, iterator from); //-->implementar
  iterator insert(iterator it, elem_t e); //-->implementar
  ...
}
```

implementar los métodos `btree::splice`, `iterator::right` y `btree::insert`.

b) **[set-vb]:** Dada la siguiente interfaz para un conjunto (set) por vector de bits que debe contener todos los números pares desde 1 a 100:

```
const int N=50;
typedef char elem_t;
int index(elem_t t);
elem_t element(int j);
typedef int iterator_t;
typedef pair<iterator_t,bool> pair_t;

class set {
private:
    vector<bool> v;
    iterator_t next_aux(iterator_t p);
public:
    set();
    pair_t insert(elem_t x);
    elem_t retrieve(iterator_t p);
```

```
    void erase(iterator_t p);
    bool erase(elem_t x);
    iterator_t find(elem_t x);
    iterator_t begin();
    iterator_t end();
    iterator_t next(iterator_t p);
};
```

implemente los métodos: `begin`, `next` e `insert`, y las funciones `index` y `element` (si dentro de alguno de ellos utiliza otro método de la misma clase set, deberá implementarlo también).

[Ej. 2] [Operativos (mínimo 50 %)]

a) **[Huffman]** Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario siguiendo el algoritmo de Huffman y encodar la palabra `MESSI`.
P(A)=0.25, P(M)=0.10, P(S)=0.20, P(O)=0.10, P(I)=0.05, P(L)=0.05, P(E)=0.10, P(C)=0.10, P(U)=0.05
Calcular la longitud promedio del código obtenido.

b) **[heap-sort]:** Dados los enteros `{6, 10, 5, 2, 6, 11, 9, 4, 4, 3, 12, 7}` ordenarlos por el método de *ordenamiento por montículos*. Mostrar el montículo inicial y el resultante luego de extraer cada mínimo parcial.

c) **[abb]:** Dados los enteros `{4,8,23,4,42,15,37,16}` insertarlos, en ese orden, en un *árbol binario de búsqueda*. Mostrar las operaciones necesarias para eliminar los elementos 14, 5 y 23 en ese orden.

d) **[hash]:** Insertar los enteros `{-2, 7, 5, -12, 7, 33, 12}` en una tabla de dispersión cerrada con `B=9` cubetas, con función de dispersión `h(x) = x%B` y redispersión linear. Luego eliminar el elemento `-2` e insertar el elemento `25`, en ese orden. Mostrar la tabla resultante.

[Ej. 3] [Preguntas (mínimo 60 %)]

a) Explique que es la *condición de prefijos* en codificación. ¿Cómo se representa ésta condición por medio de árboles?

b) Comente acerca del número de intercambios realizados por los algoritmos de ordenamiento lentos.

c) ¿Qué dos condiciones debe cumplir un montículo? ¿Qué nos garantiza cada una de ellas?.

d) Explique los tiempos de ejecución de quick-sort y heap-sort para el peor caso, mejor caso y caso promedio. ¿Cuando es conveniente utilizar uno u otro?

e) Explique el concepto de estabilidad de un algoritmo de ordenamiento y clasifique los algoritmos que conoce.

f) En el desarrollo de videojuegos suele ser importante resolver de forma eficiente la colisión entre dos objetos. En los juegos actuales las escenas constan de realmente muchos objetos (>10000) distribuidos en todo el espacio y para aportar realismo se debe generar cada cuadro muy rápido (mínimo 60fps). ¿Por qué no es una buena idea simplemente detectar si un elemento colisiona con cada uno de los restantes?. Proponga una estrategia basada en las estructura de datos estudiadas durante el curso para obtener una implementación más eficiente. Base su respuesta en el analisis de órdenes de ejecución.

Algoritmos y Estructuras de Datos.
TPL1. Trabajo Práctico de Laboratorio 1. [2017-09-07]

PASSWD PARA EL ZIP: `6KLD W3AN CC1U`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[extract-range]** Dada una lista de enteros `L1` y dos iteradores en la misma `p,q`, extraer el rango `[p,q)` de `L1` en la lista `L2`. Por ejemplo si `L1=(0,4,3,5,2,7,1,8)` y `*p=4`, `*q=1`, entonces `extract_range(L1,p,q,L2)` debe dejar `L1=(0,1,8)`, `L2=(4,3,5,2,7)`.
Nota1: Ambos `p,q` pueden ser `end()`
Nota2: Puede ocurrir que `q` esté antes de `p` en cuyo caso `L2` debe retornar vacío. Por ej.: si `L1=(0,4,3,5,2,7,1,8)` y `*p=7`, `*q=5`, entonces debe quedar: `L1=(0,4,3,5,2,7,1,8)`, `L2=()`.
Consigna: Escribir una función

```
void extract_range(list<int> &L1, list<int>::iterator p,
list<int>::iterator q, list<int> &L2);
```

que realiza la tarea indicada.

[Ej. 2] **[add-elements]** Insertar cada uno de los elementos de la pila `S` en la lista ordenada `L`, la cual debe permanecer ordenada luego de la inserción. La función debe retornar la cantidad de números repetidos en la lista `L` luego de la inserción. Tener en cuenta que si hay más de dos ocurrencias del mismo número, dicho número cuenta una única vez en la suma de elementos repetidos.
Ejemplos:
(ej1): `S=[-3,4], L=[-1,4,4,7] --> L = [-3,-1,4,4,4,7], ret=1`
(ej2): `S=[-3,4], L=[-3,4,7] --> L = [-3,-3,4,4,7], ret=2`.
(ej3): `S=[-3,4], L=[-3,4,7,10,10] --> L = [-3,-3,4,4,7,10,10], ret=3`.
Nota: La cantidad de elementos distintos es independiente de si estos elementos duplicados se originaron al insertar los nuevos elementos o existían previamente o ambas cosas a la vez.
Consigna: Escribir una función `int addElements(list<int>& L, stack<int> &S);`
que realiza la tarea indicada.

[Ej. 3] **[coprimos]** Escribir una función `bool coprimos(list<int> &L);` que retorna `true` si todos los elementos de `L` son coprimos entre sí. Recordemos que dos enteros son coprimos entre sí el único entero que divide a ambos es 1.
Por ejemplo:

- 12,15 no son coprimos ya que 3 es divisor de ambos.
- 10,21 son coprimos ya que no tienen ningún divisor (`>1`) común.

Nota: Todos los enteros `x` en `L` son `x>1`.
Ayuda:

- Escribir una función auxiliar
  `void divisors(int x,list<int> &divs);` que retorna en `divs` los divisores del entero `x`. Para ello recorre todos los enteros en `[2,x]` y los incluye en `divs` si dividen a `x`.
- Escribir una función `bool coprimos(int x,int y);` que chequea si `x,y` son coprimos. Para ello genera los divisores de `x` y los de `y` y verificar si no hay elementos comunes en ambos.
- Recorrer todos los pares de valores `xi,xj` en `L` (con `i!=j`) chequear si son coprimos entre sí.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum `H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra

verificará el valor de salida de **H**.

- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

TPL1. Trabajo Práctico de Laboratorio 1. [2017-09-07]. TABLA SEED/HASH

```
S=123 -> H1=630 H2=107 H3=357
S=577 -> H1=414 H2=204 H3=666
S=393 -> H1=870 H2=425 H3=146
S=686 -> H1=383 H2=944 H3=115
S=349 -> H1=388 H2=724 H3=228
S=762 -> H1=215 H2=704 H3=577
S=690 -> H1=511 H2=264 H3=327
S=663 -> H1=868 H2=194 H3=527
S=340 -> H1=035 H2=720 H3=284
S=872 -> H1=402 H2=739 H3=291
S=711 -> H1=351 H2=245 H3=126
S=367 -> H1=119 H2=353 H3=844
S=882 -> H1=895 H2=756 H3=214
S=162 -> H1=101 H2=719 H3=994
S=767 -> H1=226 H2=390 H3=939
S=429 -> H1=524 H2=513 H3=953
S=622 -> H1=753 H2=088 H3=058
S=969 -> H1=869 H2=887 H3=129
S=893 -> H1=253 H2=392 H3=873
S=311 -> H1=480 H2=806 H3=888
S=529 -> H1=594 H2=108 H3=369
S=721 -> H1=879 H2=507 H3=863
S=384 -> H1=679 H2=052 H3=799
S=798 -> H1=441 H2=024 H3=698
S=615 -> H1=618 H2=518 H3=663
S=386 -> H1=077 H2=086 H3=675
S=215 -> H1=464 H2=745 H3=970
S=935 -> H1=104 H2=681 H3=952
S=292 -> H1=441 H2=428 H3=802
S=821 -> H1=406 H2=484 H3=505
S=527 -> H1=329 H2=587 H3=514
S=359 -> H1=044 H2=274 H3=560
S=626 -> H1=882 H2=816 H3=364
S=226 -> H1=128 H2=646 H3=713
S=236 -> H1=313 H2=753 H3=622
S=468 -> H1=040 H2=125 H3=952
S=529 -> H1=594 H2=108 H3=369
S=630 -> H1=076 H2=251 H3=917
S=923 -> H1=896 H2=739 H3=952
S=335 -> H1=929 H2=082 H3=874
S=802 -> H1=577 H2=532 H3=644
S=958 -> H1=605 H2=236 H3=072
S=967 -> H1=275 H2=397 H3=530
S=656 -> H1=670 H2=894 H3=633
S=242 -> H1=469 H2=421 H3=427
S=973 -> H1=257 H2=187 H3=244
S=219 -> H1=726 H2=852 H3=765
S=437 -> H1=990 H2=796 H3=528
S=624 -> H1=108 H2=586 H3=753
S=670 -> H1=950 H2=535 H3=718
```

Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio 2. [2017-10-12]

PASSWD PARA EL ZIP: MPBR PZXR 5QPP

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] [prom-nivel]
Dado un árbol `tree<int> T`, generar mediante la función
`void prom_nivel(tree<int> &T, list<float> &P);` una lista de reales `P`, donde el primer elemento de la lista sea el promedio de los nodos del árbol de nivel 0, el segundo sea el promedio de los de nivel 1, el tercero el promedio de los de nivel 2, y así sucesivamente. Es decir, que si el árbol tiene profundidad `N`, la lista tendrá `N+1` elementos de tipo `float`.
Ejemplos:

- `T= (5 (10 8 9) (7 3))  ->  P=(5, 8.5, 6.66666666)`
- `T= (23 78 52 (69 50) 28 (79 13)) -> P=(23, 61.2000007629395, 31.5)`

Nota: recuerde que en C++ la división entre enteros siempre arroja un resultado entero (por ejemplo, 7/3 da 2). Para lograr un resultado real puede castear uno o ambos operandos a float: float(7)/3 da 2.33333.
Ayuda: utilizar una o dos estructuras auxiliares donde guardar las sumatorias y los conteos de nodos por nivel (por ejemplo dos mapas, uno de nivel a sumatoria, otro de nivel a cantidad de nodos).

[Ej. 2] **[es-circuito]** Dado un grafo G representado por un map de nodos a lista de vecinos (`map<int,list<int>>`), y una lista de nodos `list<int> L` , escriba una función `bool esCircuito(map<int,list<int>>&G,list<int>&L)` que determine si la secuencia de nodos `L` es un camino dentro del grafo `G`.
Ejemplos:

- `G=(1->{2,4},2->{1,3},3->{2,4},4->{1,3}), L={1,2,3,4} -> true`.
- `G=(1->{2},2->{1,3},3->{2,4},4->{3}), L={1,2,3,4} -> false`.

Ayuda: para cada par de elementos consecutivos en la lista, y también para el par formado por el último y el primero, verificar que exista en el grafo una arista que una a ambos.

[Ej. 3] **[map-graph]** Dado un grafo

```
typedef map<int,vector<int>> graph_t;
graph_t Gin;
```

y una permutación `map<int,int> P`, encontrar el grafo `Gout` que resulta de permutar los vértices de `Gin` por la pertmutación `P`, es decir si la arista `(j,k)` está en `Gin`, entonces la arista `(P[j],P[k])` debe estar en `Gout`.
Consigna: Escribir una función
`void map_graph(graph_t &Gin,map<int,int> &P,graph_t &Gout);`
Ejemplos:

- `Gin= (0->{1}, 1->{0,2,3},2->{1,4},3->{1,4},4->{2,3}),`
  `P=(0->1,1->2,2->3,3->4,4->0)`
  Debe quedar:
  `Gout= (0->{3,4},1->{2},2->{1,3,4},3->{2,0},4->{2,0}),`

- `Gin= (0->{1}, 1->{0,2,3},2->{1,4},3->{1,4},4->{2,3}),`
  `P=(0->1,1->0,2->2,3->4,4->3)`
  Debe quedar:
  `Gout= (0->{1,2,4},1->{0},2->{0,3},3->{2,4},4->{0,3}),`

Nota: En `Gout` los vecinos del grafo deben quedar ordenados de menor a mayor.
Ayuda: Con un doble lazo recorrer las aristas de `Gin`. Para cada arista `(i,j)` en `Gin` agregar la arista `(P[i],P[j])` en `Gout`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- `ucase`: Además las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere "hacer todos". Por ejemplo `ev.eval4(prune_to_level,1,51);` corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función `eval<j>` están almacenados en un archivo `test1.json` o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. `datain` son los datos pasados a la función y `output` la salida producida por la función de usuario. `ucase` es el número de caso.

```
{ "datain": {
  "T1": "( 0 (1 2) (3 4 5 6) )",
  "T2": "( 0 (2 4) (6 8 10 12) )",
  "func": "doble" },
 "output": { "retval": true },
 "ucase": 0 },
```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

mstorti@galileo/aed-3.1-84-g1d9b1d7b/Thu Oct 12 09:06:40 2017 -0300

Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio 3. [2017-11-02]

PASSWD PARA EL ZIP: JZQT WKBA ZLYZ

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[isBST]** Dado un árbol binario `T`, generar la función `bool isBST(btree<int>&T);` que devuelve `true` si `T` es un árbol binario de búsqueda y `false` en caso de que no lo sea.
Recordar que un árbol binario es un **ABB** (o **BST** por sus siglas en inglés) si:

- Está vacío.
- Para cualquier nodo `n` del árbol, todos los valores del subárbol izquierdo del nodo `n` son menores al valor de `n`.
- Para cualquier nodo `n` del árbol, todos los valores del subárbol derecho del nodo `n` son mayores al valor de `n`.

Ejemplos:

- `T=(10 (5 (3 (7 6 9))) (18 15 (20 19 30)))` debe retornar `true`.
- `T=(10 (5 (3 (7 6 9))) (18 15 (20 19 3)))` debe retornar `false`.
- `T=(8 (5 4 7) (15 13 20))` debe retornar `true`.

[Ej. 2] **[fillBST]** Dada una lista de enteros `L` y un árbol binario `T`, generar la función `void fillBST(btree<int>&T,list<int>& L);` que inserta los elementos de `L` en `T` formando un árbol binario de búsqueda siguiendo el orden original de `L`. Tal como en la implementación de `set`, en `T` no pueden quedar elementos repetidos (pese a que `L` los tenga).
Nota: No se puede usar `unique`.
Ejemplos:

```
L=[11,3,7,5,22,15,7] => debe retornar T = (11 (3 . (7 5 .)) (22 15 .))
L=[3,3,3,3]          => debe retornar T = (3)
L=[45,34,23,9,89,12] => debe retornar T = (45 (34 (23 (9 . 12) .) .) 89)
```

[Ej. 3] **[eqsumplit]** Escribir un predicado `bool eqsumsplit(set<int> &S);` que retorna `true` sii el conjunto `S` se puede descomponer en dos conjuntos disjuntos `S1` y `S2` tales que la suma de los elementos de `S1` es igual a la suma de `S2`.
Restricción: no usar otros contenedores que `set`.
Ejemplos:

```
S={1,2,3}         => debe retornar true (S1={1,2},S2={3})
S={1,2,3,4}       => debe retornar true (S1={1,4},S2={2,3})
S={1,2,3,4,5}     => debe retornar false
S={1,2,3,4,5,6}   => debe retornar false
S={1,2,3,4,5,6,7} => debe retornar true (S1={3,5,6},S2={1,2,4,7}
```

Ayuda:

- Se sugiere proceder recursivamente. Primero con **S1=S**. Después con todos los subconjuntos **S1** que provienen de tomar **S1** y extraer un elemento, y asi siguiendo.
- Escribir una función auxiliar **bool eqsumsplit(set<int> &S,set<int> &W);** que realiza la tarea consignada, con la restricción de que **S1** debe ser un subconjunto de **W**.
- **eqsumsplit(S,W)** debe primero chequear si **S1=W**, **S2=S-S1** es una solución. Si es solución debe retornar **true**.
- Si no debe formar todos los subconjuntos de **S1** de **W**, que consisten de eliminar un elemento de **W**. Es decir, para todo **x** en **W**, formar **Wx=W-{x}** y llamar recursivamente a **eqsumsplit(S,Wx)**, si en algún caso retorna true debe retornar **true**.
- Si para ningún **Wx** retorna **true**, entonces debe retornar **false**.
- El proceso se inicializa con **W=S**.
- El seudocodigo es asi:

```
bool eqsumsplit(S,W) {
  S1 = W; S2=S-S1;
  Si el par S1,S2 satisface la condicion retornar true
  para cada elemento x de W {
    Hacer W1 = W-{x}
    si eqsumsplit(S,W1) retorna verdadero, retornar verdadero
  }
  retornar falso;
}
```

- Escribir el wrapper **bool eqsumsplit(S);**

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **program.cpp**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **f** es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
   "T1": "( 0 (1 2) (3 4 5 6) )",
   "T2": "( 0 (2 4) (6 8 10 12) )",
   "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

Algoritmos y Estructuras de Datos.
Recuperatorio de Trabajos Prácticos de Laboratorio. [2017-11-09]

PASSWD PARA EL ZIP: T11A P23E L58D

Ejercicios

[Ej. 1] **[tpl1 - sum_sublist]**
Implemente la función `list<int> sum_sublist(list<int>& L,int S)` que dada una lista de enteros `L` y un entero `S`, encuentre y retorne una sublista cuya suma sea `S`. Si no existe ninguna sublista con dicha suma, retorne una lista vacía. En caso de haber varias listas que cumplan retorne la primera y la más corta.
Ejemplos:

- Para `L: [1,2,6,5,8,3,4,6]` y `S=13` debe retornar `[2,6,5]`
- Para `L: [1,2,6,5,8,3,4,6]` y `S=15` debe retornar `[8,3,4]`
- Para `L: [1,2,6,5,8,3,4,6]` y `S=17` debe retornar `[]`

[Ej. 2] **[tpl1 - discrete_moving_mean]** Implemente la función `list<int> discrete_moving_mean(list<int>& L, int w)` que dada una lista de enteros `L` y un entero `w`, retorne una lista con los valores de la media móvil con ventana fija de tamaño `w`. El primer elemento será el promedio (en división entera) de los primeros `w` elementos de `L`; el segundo será el promedio desde el `2` elemento al `w+1`; y en general, el elemento en la posición `N` de la lista resultado, será el promedio entre los elementos `[N,w+N)` de `L`.
Ejemplos:

- Para `L: [1,2,6,5,8,3,4,6]` y `w=2` debe retornar `[1,4,5,6,5,3,5]`
- Para `L: [1,2,6,5,8,3,4,6]` y `w=3` debe retornar `[3,4,6,5,5,4]`
- Para `L: [1,2,6,5,8,3,4,6]` y `w=4` debe retornar `[3,5,5,5,5]`

[Ej. 3] **[tpl2 - d10s]** Implemente la función `list<int> d10s(tree<int> &T)` que reciba un AOO `T` y retorne el camino (list<int>) desde la raíz hasta el nodo con la etiqueta `10`. Se garantiza que todo árbol tiene una sola etiqueta con el número `10`.
Ejemplos:

- Para `T = (1 (3 4 (5 10)) 2)` debe retornar `[1,3,5,10]`
- Para `T = (1 (3 4 5 (8 9 (15 10))) 2)`, debe retornar `[1,3,8,15,10]`

[Ej. 4] **[tpl2 - is_cycle]** Implemente una función bool `is_cycle(graph_t &G)` que determine si el grafo `G` es un ciclo dirigido o no. Un grafo ciclo dirigido cumple dos condiciones:

- cada vértice sea de grado 2 (una arista de entrada y otra de salida).
- es posible hallar un circuito Hamiltoniano.

Nota: `graph_t` es un `typedef` de `map<int,list<int>>`.
Ejemplos:

- Para `G = {0=>[1],1=>[2],2=>[3],3=>[4],4=>[5],5=>[0]}` debe retornar `true`.
- Para `G = {0=>[1],1=>[2],2=>[3],3=>[4],4=>[5],5=>[0],6=>[4]}` debe retornar `false`.

[Ej. 5] **[tpl3 - existe_subc]** Implementar una función `bool existe_subc(set<int> &S, int n)` que dado un conjunto `S` y un valor `n`, determine si existe un subconjunto de `S` para el cual la suma de sus elementos sea `n`.
Ejemplos:

- Para `S = {-1,3,5,8,13}, n=5` debe retornar `true`, el sub-conjunto sería `{5}`
- Para `S = {-1,3,5,8,13}, n=10` debe retornar `true`, el sub-conjunto sería `{-1,3,8}`
- Para `S = {-1,3,5,8,13}, n=6` debe retornar `false`
- Para `S = {-1,3,5,8,13}, n=0` debe retornar `true`, el sub-conjunto sería `{ }`

Ayuda:

- Escribir primero un algoritmo recursivo para generar todos los subconjuntos de `S` posibles.
- Modificarlo luego para que cada vez que se genera un subconjunto, se verifique su suma, y se corte la recursión si coincide con `n`.

[Ej. 6] **[tpl3 - replace_btree]** Implemente la función `void replace_btree(btree<int>&T, pt_fun_t pred)` que busque los nodos que cumplan con la condición `pred` y los reemplace. Cada nodo que cumple la condición debe ser reemplazado por el primero de sus descendientes (según el pre-orden) que no la cumpla. En caso de no encontrar ningún descendiente que no la cumpla, todo el nodo (junto con todo su subárbol) se elimina.
Ejemplos:

- Para `T = (2 (4 6 8) (10 12 14)), f = es_par,` debe retornar `T = ()`
- Para `T = (2 (4 6 8) (10 12 13)), f = es_par,` debe retornar `T = (13 . (13 . 13))`
- Para `T = (8 (5 . 7) (6 . 5)), f = es_par,` debe retornar `T = (5 (5 . 7) (5 . 5))`
- Para `T = (7 (5 3 (0 1 .)) 8), f = es_par,` debe retornar `T = (7 (5 3 (1 1 .)) .)`

Ayuda: Escribir una funcion recursiva `replace_btree(T,n,pred)`:

- Si `n` satisface el predicado entonces buscar en su subarbol un elemento `q` que no lo satisfaga.
  - Si `n` es hoja o no hay ningún `q` que no satisfaga el predicado entonces directamente eliminar todo el subárbol de `n`
  - En caso contrario, reemplaza el valor de `n` por el de `q`.
- Aplicar recursivamente a los hijos de `n`

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
   "T1": "( 0 (1 2) (3 4 5 6) )",
   "T2": "( 0 (2 4) (6 8 10 12) )",
   "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Examen Final. [08-02-2018]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 70 % de la sección PREG. y 50 % en las restantes secciones.

[Ej. 1] [PREG (min 70 %)]

a) Explique qué quiere decir la propiedad de "Transitividad" de **`O()`**
b) ¿Cuál es la "Regla del producto" para **`O()`**?
c) Discuta las diferencias en la implementación y las ventajas y desventajas de utilizar listas doblemente enlazadas con respecto a las simplemente enlazadas. ¿Cuáles son los métodos cuyo tiempo de ejecución cambia y por qué?
d) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase **`map<T>`** implementada con listas ordenadas y vectores ordenados en el caso promedio? Métodos: **`find(key)`**, **`M[key]`**, **`erase(key)`**, **`erase(p)`**, **`begin()`**, **`end()`**, **`clear()`**.
e) Indique los tiempos de ejecución de los algoritmos de ordenamiento rápidos para los tres casos (peor, mejor y promedio). ¿Cuando es conveniente utilizar uno u otro?
f) Explique el concepto de "Estabilidad" de un algoritmo de ordenamiento. Mencione dos ejemplos: un algoritmo estable y otro no estable.
g) Explique que es la condición de prefijos. ¿Cómo se representa ésta condición en términos del árbol de la codificación?
h) ¿Es posible "insertar" en una posición "dereferenciable" (Λ) en un AB? ¿Y en un AOO? Discuta y de ejemplos.
i) ¿Cómo se define la tasa de ocupación de una tabla de dispersión cerrada? ¿Qué implicancias tiene en el funcionamiento de la estructura de datos el hecho de que dicha tasa sea cercana a cero? ¿y cercana a uno?.
j) Mencione las dos condiciones que debe cumplir un montículo y describa cual es la utilidad en términos algorítmicos y/o almacenamiento de datos de dichas estructuras.

[Ej. 2] [CLASES (min 50 %)]

a) **[dlist]** Implemente los métodos **`insert`** y **`erase`** para una lista doblemente enlazada:

```
class cell {
    cell *next, *prev;
    elem_t elem;
    cell() : next(NULL) {}
    friend class list;
}
typedef cell *iterator_t;
class list {
    cell *first;
    public:
    ...
    iterator_t insert(iterator_t p, elem_t e);
    iterator_t erase(iterator_t p);
    ...
};
```

- ¿Es necesario utilizar un iterador atrasado en este tipo de lista? ¿Por qué?

b) **[aoo]** En una implementación de Árbol Ordenado Orientado AOO en donde cada nodo del árbol es representado por una estructura como la siguiente:

```
template<class T> struct cell{
    cell * left_child, right;
    T elem;
}
```

- ¿Por qué es insuficiente utilizar iteradores que sean **`typedef cell* iterator`**? Presente un ejemplo que lo demuestre.
- Proponga una definición de iterador que solucione el/los problemas mencionados en el punto anterior.

c) **[ab]** Defina los structs/clases **`iterator`** y **`cell`** para un Árbol Binario, e implemente el método **`iterator btree<T>::find(T x)`**.

[Ej. 3] [PROG (min 50 %)]

a) **[remp-seq]** Dada una lista de enteros **`L`** y dos listas **`SEQ`** y **`REEMP`**, posiblemente de distintas longitudes, escribir una función **`void reemplaza(list<int> &L, list<int>& SEQ, list<int> &REEMP)`**, que busca todas las secuencias de **`SEQ`** en **`L`** y las reemplaza por **`REEMP`**. Por ejemplo, si **`L=1,2,3,4,5,1,2,3,4,5,1,2,3,4,5)`**, **`SEQ=(4,5,1)`** y **`REEMP=(9,7,3)`**, entonces después de llamar a **`reemplaza(L,SEQ,REEMP)`**, debe quedar **`L=(1,2,3,9,7,3,2,3,9,7,3,2,3,4,5)`**. Para implementar este algoritmo primero buscar desde el principio la secuencia **`SEQ`**, al encontrarla, reemplazar por **`REEMP`**, luego seguir buscando a partir del siguiente elemento al último de **`REEMP`**.

b) **[hay-camino]** Escriba una función **`bool hay_camino(map<int,list<int>> &G,int i,int j)`** que dado un Grafo dirigido **`G`**, y dos nodos del mismo, retorne verdadero si y solo si existe un camino dentro del grafo que vaya desde el primer al segundo nodo.

[Ej. 4] [OPER (min 50 %)]

a) **[rec-arbol]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- **`ORD-PRE`**=$\{P, F, H, S, B, N, Q, G, K, T, E, D, W, O\}$,
- **`ORD-POST`**=$\{Q, G, N, B, S, H, D, W, E, T, O, K, F, P\}$,

b) **[huffman]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra **`EXAMEN`**
$P(X) = 0.10, P(M) = 0.10, P(U) = 0.10, P(E) = 0.30, P(I) = 0.05, P(A) = 0.3, (N) = 0.05$
Calcular la longitud promedio del código obtenido.

c) **[hf-decode]** Utilizando el código del siguiente árbol binario **`(* (* (* U V) (* (* W X) Y)) Z)`**, desencodar el mensaje `010101111011`

d) **[abb]** Dados los enteros $\{15, 9, 22, 4, 5, 12, 7, 6, 3, 14\}$ insertarlos, en ese orden, en un *"árbol binario de búsqueda"*. Mostrar las operaciones necesarias para eliminar los elementos 15, 7 y 4 en ese orden.

e) **[hash-dict]** Insertar los números $9, 2, 0, 19, 32, 14, 41, 58$ en una tabla de dispersión cerrada con $B = 10$ cubetas, con función de dispersión $h(x) = x \bmod 9$ y estrategia de redispersión lineal.

f) **[heap-sort]** Dados los enteros $\{10, 13, 3, 2, 8, 5, 1\}$ ordenarlos por el método de *"montículos"* (*"heap-sort"*). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

mstorti@galileo/TPL-SUPREC-2015-15-ge3b3752/Thu Nov 19 12:34:07 2015 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniera y Ciencias Hdricas
Departamento de Informtica
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
1er parcial. [04-10-2018]

[ATENCIÓN 1] Para aprobar deben obtener un **puntaje mínimo** del 60 % de la sección PREG. y 50 % en las restantes secciones.

[Ej. 1] [PREG (min 60 %)]

a) Explique qué quiere decir la propiedad de "Transitividad" de **`O()`**
b) Presente las diferencias entre las estrategias heurísticas y exhaustivas. Dé un ejemplo.
c) Comente ventajas y desventajas del uso de listas doblemente enlazadas con respecto a simplemente enlazadas. ¿Cuáles son los métodos cuyo tiempo de ejecución cambia y por qué?
d) ¿Por qué en un mapa en el que el método **`find`** es **`O(log(n))`**, el método **`retrieve(x)`** puede ser **`O(n)`**? ¿En qué casos ocurre?
e) Dentro de la clase **`tree`** que define un AOO, ¿por qué no es suficiente definir al iterador como **`typedef iterator *cell`**? Explique
f) Sea el AOO **`T=(A B C (D E))`**, y tomando dos iteradores **`it1 = T.find(B).lchild()`** e **`it2 = T.find(D).right()`**. Grafique el árbol y ubique los iteradores mencionados presentando claramente los punteros que contienen. ¿Por qué a pesar de las diferencias obtengo verdadero al hacer **`it1==it2`**?.

[Ej. 2] [CLASES (min 50 %)]

a) **[s-list]** Dada la siguiente implementación para una lista simplemente enlazada:

```
class cell {
    int elem;
    cell *next;
    friend class list;
};
using iterator = cell*;
class list {
    cell *first, *last;
public:
    list();
    ~list();
    iterator insert(iterator p, int x);
    iterator erase(iterator p);
    void clear();
    iterator begin();
    iterator end();
    iterator prev(iterator p);
    iterator next(iterator p);
    int retrieve(iterator p);
    int size();
    void splice(iterator p, list &l2, iterator from, iterator to);
    friend void swap(list &l1, list &l2);
};
```

implementar las funciones: **`list::retrieve(...)`**, **`list::size()`**, **`list::splice(...)`**, y **`swap(...)`** de forma tal que **`list::splice`** y **`swap`** sean **`O(1)`**. Nota: recuerde que splice mueve una parte de **`l2`**, definida por **`[from,to)`**, hacia la posición **`p`** de la lista que hace la llamada.

b) **[map-vo]** Considerando la siguiente definición de una clase **`map`** que mapea enteros a strings utilizando un vector ordenado:

```
class map {
    vector<pair<int,string>> v;
    iterator lower_bound(int x);
    ...
public:
```

```
    ...
    iterator find(int x);
    pair<bool,iterator> insert(int x);
    string retrieve(int x);
    ...
};
```

defina el tipo iterator, e implemente el método **lower_bound** de manera tal que sea **O(log(n))**; y utilícelo luego para implementar también **find**, **insert** y **retrieve**.

[Ej. 3] [OPER (min 50 %)]

a) **[coloreo]** Un paseador de perros tiene 10 clientes: Huesos, Coraje, Procer, Oddie, Scooby, Pluto, Brian, Snoopy, Bolt y Goofy. Pero no puede pasear a los 10 perros a la vez porque hay perros que se sabe que se pelean entre sí. Huesos siempre quiere morder a Procer y a Oddie. Scooby odia a Pluto y a Goofy. Coraje les tiene mucho miedo a Goofy, Oddie y Snoopy. Brian no quiere pasear ni con Bolt ni con Oddie. Y finalmente, se sabe que Bolt tampoco se lleva bien con Procer. Cómo utilizaría coloreado de grafos para modelar este problema y armar grupos de paseo. Utilizando la estrategia que proponga, genere una coloración válida y explique qué significa el resultado.

b) **[draw]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- **ORD-PRE = (C B F C I H G A J D E)**,
- **ORD-POST = (F I C B H A D J E G C)**,

luego particionar el conjunto de nodos respecto al nodo **J**.

c) **[orden]** Ordenar las siguientes funciones por tiempo de ejecución:

$$T_1(n) = 5\sqrt{n^3} + \log(5n)$$
$$T_2(n) = 3n^2 + 10^4$$
$$T_3(n) = 3^n + 3n^3$$
$$T_4(n) = 5n^2 \log(n)$$

Además, para cada una de las funciones **T1 , . . . , T4** determinar su velocidad de crecimiento.

d) **[function-f]** Dada la siguiente función y su *wrapper*:

```
void f(int n, vector<bool>& VL){
    if(n==0){
        for(auto x:VL) cout<<x<<" ";
        cout<<endl;
    }else{
        auto VL_0 = VL;
        auto VL_1 = VL;
        VL_0.push_back(0);
        VL_1.push_back(1);
        f(n-1,VL_0);
        f(n-1,VL_1);
    }
}
void f(int n){
    vector<bool> VL;
    f(n,VL);
}
```

- Renombrar la función con un nombre alusivo a la tarea que efectúa.
- Determinar el orden algorítmico de **f** respecto al parámetro **n**.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

FICH
UNL

Algoritmos y Estructuras de Datos.
Parcial 2. [2018-11-13]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS1, Hoja #2/3 TORVALDS, LINUS

[Ej. 1] [CLAS2 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[set-vb]:** Dada la siguiente interfaz para un conjunto (set) por vector de bits que contiene pares de letras mayúsculas (de `"AA"` a `"ZZ"`, representadas como strings de longitud 2):

```
typedef string elem_t;
int index(elem_t t);
elem_t element(int j);
typedef int iterator_t;

class set {
private:
    vector<bool> v;
    iterator_t next_aux(iterator_t p);
public:
    set();
    pair<iterator_t,bool> insert(elem_t x);
    elem_t retrieve(iterator_t p);
    void erase(iterator_t p);
    bool erase(elem_t x);
    iterator_t find(elem_t x);
    iterator_t begin();
    iterator_t end();
    iterator_t next(iterator_t p);
};
```

implemente los métodos **`begin`**, **`end`**, **`next`** e **`insert`** (si dentro de alguno de ellos utiliza otro método de set, también deberá implementarlo) y las funciones **`index`** y **`element`**.

b) **[set-abb]:** Dada la siguiente interfaz para un conjunto (set) por árbol binario de búsqueda:

```
template<class T>
class set {
private:
    typedef btree<T> tree_t;
    typedef typename tree_t::iterator node_t;
    tree_t bstree;
    ...
public:
    class iterator {
            note_t node;
            tree_t &bstree;
    private:
                friend class set;
                iterator(tree_t &t, node_t n) : bstree(t), node(n) {}
        ...
```

```
16      };
17      ...
18      pair<iterator,bool> insert(T x);
19      iterator find(T x);
20      int erase(T x);         //<- implementar
21      iterator begin();       //<- implementar
22      iterator end();         //<- implementar
23      node_t next(node_t  ); //<- implementar
24      ...
25  };
```

implemente los métodos **begin**, **end**, **next** y **erase** (si dentro de alguno de ellos utiliza otro método de set, también deberá implementarlo).

[Ej. 2] [OPER2 (W=20pt)]

a) **[heap-sort (5pt)]** [heap-sort]: Dados los enteros { 7, 11, 5, 13, 9, 4, 1, 4 } ordenarlos por el método de ordenamiento por montículos. Mostrar el montículo inicial y el resultante luego de extraer cada mínimo parcial.

b) **[quick-sort (5pt)]** Dados los enteros { 6, 1, 5, -2, 6, 11, -9, 4, -4, 3, 12, -7 } ordenarlos por el método de ordenamiento rápido (quick-sort). En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber, el mayor de los dos primeros elementos distintos.

c) **[hash (5pt)]** Se implementa un diccionario a través de una tabla de dispersión cerrada con B=10 cubetas, con función de dispersión h(x) = x/(10 B) y redispersión lineal. Insertar los enteros { 710, 193, 225, 818, 27, 353, 179 } . Luego eliminar el elemento 353 e insertar el elemento 179, en ese orden. Mostrar la tabla resultante. ¿Que rango de números enteros es capaz de gestionar el diccionario propuesto?

d) **[2d-game (5pt)]** En el desarrollo de videojuegos suele ser importante determinar de forma eficiente la colisión entre dos objetos. En los juegos actuales las escenas constan de realmente muchos objetos (>10000) distribuidos en todo el espacio y para aportar realismo se debe generar cada cuadro muy rápido (mínimo 60fps). Una estrategia para evitar comprobar la colisión *todos contra todos* consiste en particionar el espacio de forma recursiva, tal como presenta la Figura 1, de tal manera de obtener una distribución de objetos casi constante por región. ¿Qué estructura de datos utilizaría para contener los objetos? Presentelá para el caso de la Figura y mencione las ventajas de utilizar esta estrategia en términos de sus tiempos de ejecución.

[Ej. 3] [PREG2 (W=20pt)]

a) En el proceso de construcción de una condificación binaria con árboles de Hufmann se van tomando dos árboles **T1** y **T2** y se combinan en uno nuevo donde **T1** y **T2** quedan como hijos derecho e izquierdo de la raíz, es decir **(Z T1 T2)**, donde **Z** es cualquier valor. Discuta si es relevante cuál de ellos se pone como hijo izquierdo e hijo derecho. Es decir, ¿da lo mismo reemplazar por **(z T1 T2)** ó **(z T2 T1)**? Justifique.

b) ¿Cuál es el tiempo de ejecución **T(n)** de la operación **splice** par árboles binarios?

c) Explique el proceso necesario para borrar un elemento en una tabla de dispersión cerrada. ¿Se puede reemplazar el elemento en la cubeta por el elemento indefinido **undef**? Justifique.

d) ¿Cuál es el tiempo de ejecución para las operaciones binarias de conjunto en la representación de conjuntos con listas ordenadas?

e) ¿Se puede definir un árbol binario simplemente como un subconjunto de los árboles ordenados orientados limitando a que la cantidad de hijos sea a lo sumo 2? Justifique.

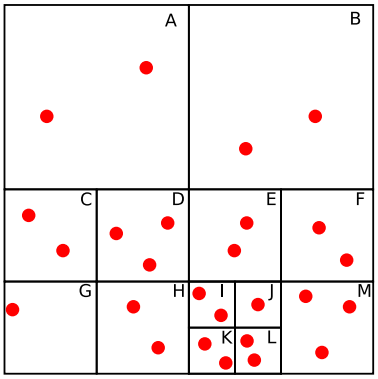


Figura 1: Escena 2d particionada en subregiones. Los puntos indican la ubicación de cada objeto. Las letras son las etiquetas de cada subregión.

f) Defina las propiedades para que un árbol sea un **montículo**. Indique que tipo de árbol se utiliza (AB o AOO) y que propiedades debe tener.

g) Discuta las propiedades de estabilidad de los diferentes métodos de ordenamiento. Indique si son estables o no y justifique brevemente porqué.

mstorti@galileo/aed-3.1-308-gc31be5e3/Thu Nov 15 13:08:58 2018 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Recuperatorio. [2018-11-20]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.

2. **[ATENCIÓN 2]**

   - Escribir cada ejercicio (sección) en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
   - Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**:
     `CLAS1, Hoja #2/3` `TORVALDS, LINUS`
   - En caso de no entregar una sección, agregar una hoja vacía con el nombre.
   - Al entregar, verificar con el docente que recibe que todas las secciones estén entregadas en forma correcta.

[Ej. 1] [CLAS1 (W=20pt)]
Recordar que **deben usar la interfaz STL**.

a) **[s-list]** Dada la siguiente definición para una lista simplemente enlazada:

```
class cell {
    int elem;
    cell *next;
    friend class list;
};
using iterator = cell*;
class list {
    cell *first, *last;
public:
    list();
    ~list();
    iterator begin();
    iterator end();
    iterator prev(iterator p);
    iterator next(iterator p);
    int retrieve(iterator p);            //<implementar
    iterator insert(iterator p, int x);  //<implementar
    iterator erase(iterator p);
    void clear();
    int size();
    void splice(iterator p, list &l2, iterator from, iterator to); //<implementar
};
```

implementar los método: `retrieve(...)`, `insert(...)` y `splice(...)` (de forma tal que `splice` sea `O(1)`). Nota: recuerde que `splice` mueve una parte de `l2`, definida por `[from,to)`, hacia la posición `p` de la lista que hace la llamada.

b) **[AOO]:** Dada la siguiente implementación de árbol ordenado orientado:

```
class cell { ... };         //< definir

class iterator_t { ... };   //< definir

class tree {
  private:
   cell *header;
   tree(const tree &T);
  public:
   tree();
   ~tree();
   elem_t &retrive(iterator_t p);
   iterator_t insert(iterator_t p,elem_t elem);  //< implementar
```

```
14    iterator_t erase(iterator_t p);              //< implementar
15    void clear();
16    iterator_t begin();
17    iterator_t end();
18    ...
19 }
```

Definir las clases **cell** e **iterator** e implementar los métodos **insert(...)**, **erase** y **find(...)** de la clase **tree**.

[Ej. 2] [OPER1 (W=20pt)]

a) **[AOO (5pt)]** Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- **ORD-PRE = (C B F C I H G A J D E)**
- **ORD-POST = (F I C B H A D J E G C)**

luego particionar el conjunto de nodos respecto al nodo **J**.

b) **[orden (5pt)]** Dada la siguiente función y su *wrapper*:

```
1 void f(int n, vector<bool>& VL){
2     if(n==0){
3         for(auto x:VL) cout<<x<<" ";
4         cout<<endl;
5     }else{
6         auto VL_0 = VL;
7         auto VL_1 = VL;
8         VL_0.push_back(0);
9         VL_1.push_back(1);
10        f(n-1,VL_0);
11        f(n-1,VL_1);
12    }
13 }
14 void f(int n){
15    vector<bool> VL;
16    f(n,VL);
17 }
```

- Renombrar la función con un nombre alusivo a la tarea que efectúa.
- Determinar el orden algorítmico de **f** respecto al parámetro **n**.

c) **[coloreo (5pt)]** Estamos organizando un congreso de informáticos y cualquier hotel de la ciudad tiene capacidad para alojar a todos los informáticos, pero hay algunos que están peleados a tal punto que no pueden alojarse en el mismo hotel. Mencione como modelaría el problema de asignar hotel a cada participante por medio de un problema de coloreo de grafos. Indique que representan los vértices, las aristas y los colores.

d) **[size (5pt)]** Suponga que cierto procesador tiene una frecuencia de reloj de 1 Ghz, y simplifique el análisis suponiendo que por cada diez ciclos de reloj se realiza una operación en particular. Estime de qué tamaño puede ser el problema que se puede resolver en diez segundos usando un algoritmo que requiere $T(n)$ operaciones, con los siguientes valores de $T(n)$: $\log(n)$, n y 2^n.

[Ej. 3] [PREG1 (W=20pt)]

a) Ordenar las siguientes funciones por tiempo de ejecución. Además, para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\,\cdot\,)$).

$$T_1 = 3\,n^4 + 2\log_{10} n + \sqrt[4]{n}$$
$$T_2 = 2 \cdot 3^n + 3\,n! + 4\,n^2$$
$$T_3 = 3^2 + 3n^2 + 3 \cdot 3^n$$
$$T_4 = \log_{10} n + 4$$
$$T_5 = \log(13) n^{3.5} + 3.4 \log n + 2^4$$

b) ¿Como se define la altura de un nodo en un árbol? ¿Cuál es la altura de los nodos **o**, **p**, y **q** en el AOO siguiente: **(k (o p (q (r t u v))) l m)**?

c) Explique la propiedad de transitividad de la notación asintótica $O(\cdot)$. De un ejemplo.

d) ¿Cuáles son los tiempos de ejecución para los diferentes métodos de la clase **map<>** implementada con **listas ordenadas** y **vectores ordenados** en el caso promedio? Métodos: **find(key)**, **M[key]**, **erase(key)**, **erase(p)**, **begin()**, **end()**, **clear()**.

e) Sea el árbol **(a b q (z r (p t)))**. Cuáles de los siguientes son **caminos**?

- **(a b)**
- **(b a)**
- **(z p t)**
- **(r z p)**

f) ¿Existe una relación biunívoca entre un Arbol Ordenado Orientado (AOO) y su **orden previo**? ¿Y con el **orden posterior**? ¿Y con ambos a la vez? De ejemplos.

g) Explique qué se entiende por algoritmos de **búsqueda exhaustiva** y **heurístico**. Discuta cuales son las ventajas y desventajas.

h) Exprese la **regla del producto** para la notación $O(.)$. De un ejemplo.

[Ej. 4] [CLAS2 (W=20pt)]

a) **[set-tda]** Dada la siguiente base para una clase que representa un conjunto mediante una tabla de disperción abierta:

```
class iterator_t { ... };                  //< definir
typedef int (*hash_func_t)(key_t x);
class hash_set {
   vector<list<key_t>> v;
   hash_func_t h;
   int B;
public:
   hash_set(int B, hash_func_t f);
   iterator_t  begin();                    //< implementar
   iterator_t  end();                      //< implementar
   iterator_t  next(iterator_t p);         //< implementar
   key_t retrieve(iterator_t p);
   pair<iterator_t,bool> insert(key_t x); //< implementar
   iterator_t find(key_t x);
   int erase(key_t x);
   void erase(iterator_t p);
};
```

Definir la clase **iterator_t**. e implementar los métodos **begin**, **end**, **next** e **insert** de la clase **hash_set** (si dentro de estos métodos utiliza algún otro método o función auxiliar, también deberá implementarlo/a).

b) **[AB]:** Dada la siguiente implementación de árbol binario:

```
class cell {
    friend class btree;
    friend class iterator;
    elem_t elem;
    cell *right, *left;
};

class iterator {
    friend class btree;
    cell *ptr, *father;
    enum side_t (NONE,R,L);
    side_t side;
    iterator(cell *p, side_t side_a, cell *f_a);
  public:
    ...
};
```

```
class btree {
  private:
    cell *header;
    ...
  public:
    btree();
    ~btree();
    elem_t & retrieve(iterator p);
    iterator find(iterator p, elem_t e);   //<
    iterator insert(iterator p, elem_t e); //<
    iterator erase(iterator p);            //<
    void clear();
    iterator begin();
    iterator end();
    ...
};
```

- Implementar los métodos `insert(...)`, `erase(...)` y `find(...)`.

[Ej. 5] [OPER2 (W=20pt)]

a) **[Huffman (5pt)]** Se procesa un texto y se extrae la siguiente probabilidad de ocurrencia de cada caracter: `P({char,prob}) = [{'A', 0.15}, {'C', 0.1}, {'E', 0.1}, {'H', 0.05}, {'L', 0.1}, {'M', 0.05}, {'N', 0.1}, {'O', 0.15}, {' ', 0.2}]`. Hallar una codificación Huffman e indicar su longitud promedio. Luego encodar: `'EL CLAN'`.

b) **[quick-sort (5pt)]** Ordenar los enteros `{3, 1, -5 -2, 6, 9, 4, -1}` utilizando el algoritmo quick-sort. Indicar claramente las iteraciones del algoritmo (seleccion de pivot y partición en listas).

c) **[hash (5pt)]** Insertar los enteros `{-2, 7, 5, -12, 7, 33, 12}` en una tabla de dispersión cerrada con B=9 cubetas, con función de dispersión `h(x) = x%B` y redispersión linear. Luego eliminar el elemento `-2` e insertar el elemento `25`, en ese orden. Mostrar la tabla resultante.

d) **[abb (5pt)]** Dados los enteros `{4,-8,-23,6,42,-15,37,16}` insertarlos, en ese orden, en un árbol binario de búsqueda. Mostrar las operaciones necesarias para eliminar los elementos 4, -15 y -23 en ese orden.

[Ej. 6] [PREG2 (W=20pt)]

a) Explique cual es la condición de códigos prefijos. De un ejemplo de códigos que cumplen con la condicion de prefijo y que no cumplen para un conjuntos de 3 caracteres.
b) Exprese como se calcula la longitud promedio de un código de Huffman en función de las probabilidades de cada unos de los caracteres P_i, de la longitud de cada caracter L_i para un número N_c de caracteres a codificar.
c) Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**.
d) Explique las operaciones que realizan (semántica) las dos versiones de erase para conjuntos.
   1) `void erase(iterator p);`
   2) `int erase(key_t x);`

   Explicar que son los valores de retorno, y porqué una retorna un entero y la otra no.
e) Explique el concepto de estabilidad para algoritmos de ordenamiento.
f) ¿Cómo se define en forma recursiva el **orden posterior** para un **árbol ordenado orientado (AOO)**?
g) Escriba las instrucciones necesarias para crear el siguiente **árbol binario (AB)**: `T=(1 (2 . 3) .)`
h) ¿Porqué el árbol binario de una códificación binaria para compresión debe ser un **árbol binario lleno (FBT)**? (Recordemos que un FBT es áquel cuyos hijos son hojas o nodos interiores con sus dos hijos.)

Algoritmos y Estructuras de Datos.
TPL1. Trabajo Práctico de Laboratorio 1. [2018-09-06]

PASSWD PARA EL ZIP: X5QV 296R F6V8

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[sign-split]** Implemente una función
`void sign_split(list<int> &L,vector< list<int> > &VL);` que dada una lista `L`, retornar el un vector de listas `VL` las sublistas contiguas del mismo signo (el caso 0 es considerado junto con los positivos, es decir separa negativos de no-negativos).
Ejemplos:

```
L:[4,-3,-5,-4,-5,-1,4,-1,-5,-5,1,-5,3,3,0,2,2],
=> VL:[[4],[-3,-5,-4,-5,-1],[4],[-1,-5,-5],[1],[-5],[3,3,0,2,2]],

L:[0,4,-2,4,1,-1,-4,-4,-3,-1,-4,4,1],
=> VL:[[0,4],[-2],[4,1],[-1,-4,-4,-3,-1,-4],[4,1]],

L:[2,-1,3,-3,3,-3,0,-1,-2,0],
=> VL:[[2],[-1],[3],[-3],[3],[-3],[0],[-1,-2],[0]],
```

Ayuda:

- Chequear el signo `s` del primer elemento y extraer tantos elementos de ese signo como se pueda para ponerlos en `VL[0]`.
- **Invertir** el signo (`s=-s`) y repetir, para poner la siguiente sublista en `VL[1]`.
- Seguir así, invirtiendo `s` en cada iteración hasta que se acaben los elementos de `L`.
- El algoritmo puede ser **destructivo** sobre `L` es decir puede modificar a `L` o no, da igual. Probablemente la solución más simple es la destructiva, extrayendo los elementos de `L` a medida que se insertan en los `VL[j]`.
- Tal vez convenga escribir una función `int sign(int x);` que retorna el "signo" de acuerdo a la convención establecida, es decir `sign(0)` debe retornar `1`.

[Ej. 2] **[sign-join]** Implemente una función
`void sign_join(vector< list<int> > &VL,list<int> &L);` que, dado un vector de listas `VL` generar una lista `L` donde están

- Primero concatenados todos la primera subsecuencia de no-negativos de `VL[0]`, `VL[1]`, ...
- Después los negativos.
- Después nuevamente los no-negativos.
- Asi siguiendo hasta que se acaban todos los `VL`.

Ejemplos:

```
VL: [[1,2,-1,-2,3,4,-3,-4],[5,6,-5,-6,7,8,-7,-8],[-9,-10]],
=> L: [1,2,5,6,-1,-2,-5,-6,-9,-10,3,4,7,8,-3,-4,-7,-8],
```

```
VL: [[0,4,-2,4,1],[-1,-4,-4,-3,-1],[-4,4,1,3,-2],[0,3,-2,4,-5],[-5,-5,-1,-3,-2]],
=> L:[0,4,0,3,-2,-1,-4,-4,-3,-1,-4,-2,-5,-5,-1,-3,-2,4,1,4,1,3,4,-2,-5]

VL:[[-1,-1,-1,3],[-5,3,1,2],[0,2,4,-3],[-3,0,4,-3]],
=> L:[0,2,4,-1,-1,-1,-5,-3,-3,3,3,1,2,0,4,-3],
```

Ayuda:

- Igual que en el ejercicio anterior recomendamos escribir una función `int sign(int x);` que retorna el "signo" de acuerdo a la convención establecida, es decir `sign(0)` debe retornar `1`.
- Extraer todos los que tienen signo `s=1` (o sea los no-negativos) de `VL[0]` y ponerlos en `L`, después los de `VL[1]` y así siguiendo para todos los `VL[j]`.
- Cambiar el signo (`s=-s`) y realizar la misma operación.
- Repetir hasta que todos los `VL[j]` estén vacíos.
- El algoritmo puede ser **destructivo** sobre `VL` es decir puede modificar a `VL` o no, da igual. Probablemente la solución más simple es la destructiva, extrayendo los elementos de los `VL[j]` a medida que se insertan en `L`.

[Ej. 3] **[reverse-stack]** Escribir un programa que invierta una pila `S` utilizando recursión. No está permitido el uso de constructores iterativos (`while`, `for`, etc) ni tampoco el uso estructuras de datos adicionales. Sólo pueden utilizarse métodos de la pila, a saber:

- `S.empty()`
- `S.top()`
- `S.push(s)`
- `S.pop()`

Se propone el siguiente algoritmo recursivo:

- Si la pila está vacia finalizar
- Si no:
  - Guardar una copia del elemento al tope
  - Quitarlo de la pila
  - Invertir el resto de la pila
  - Insertar el elemento guardado al final de la lista invertida

Para el último paso se requiere un método auxiliar, también recursivo, que reciba una lista e inserte un elemento en la base de la misma, siguiendo el siguiente algoritmo:

- Si la pila está vacía, insertar el elemento.
- Si no:
  - Almacenar una copia del elemento al tope
  - Quitarlo de la pila
  - Insertar el elemento recibido al final de la pila restante
  - Apilar el temporal en la pila resultado

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.

- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.

  - **`void Eval::dump(list <int> &L,string s="")`**

- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

TPL1. Trabajo Práctico de Laboratorio 1. [2018-09-06]. TABLA SEED/HASH

```
S=123 -> H1=694 H2=052 H3=382
S=577 -> H1=535 H2=818 H3=993
S=393 -> H1=170 H2=362 H3=339
S=686 -> H1=091 H2=073 H3=821
S=349 -> H1=990 H2=141 H3=103
S=762 -> H1=606 H2=793 H3=379
S=690 -> H1=799 H2=917 H3=108
S=663 -> H1=111 H2=637 H3=098
S=340 -> H1=749 H2=277 H3=666
S=872 -> H1=177 H2=838 H3=753
S=711 -> H1=718 H2=734 H3=561
S=367 -> H1=314 H2=180 H3=026
S=882 -> H1=191 H2=162 H3=362
S=162 -> H1=773 H2=662 H3=833
S=767 -> H1=037 H2=640 H3=463
S=429 -> H1=428 H2=753 H3=567
S=622 -> H1=470 H2=889 H3=948
S=969 -> H1=945 H2=516 H3=027
S=893 -> H1=512 H2=397 H3=903
S=311 -> H1=311 H2=856 H3=032
S=529 -> H1=410 H2=349 H3=191
S=721 -> H1=523 H2=349 H3=906
S=384 -> H1=924 H2=074 H3=755
S=798 -> H1=703 H2=013 H3=175
S=615 -> H1=724 H2=832 H3=202
S=386 -> H1=103 H2=017 H3=751
S=215 -> H1=616 H2=358 H3=165
S=935 -> H1=205 H2=038 H3=267
S=292 -> H1=725 H2=070 H3=450
S=821 -> H1=220 H2=561 H3=598
S=527 -> H1=974 H2=931 H3=921
S=359 -> H1=367 H2=889 H3=479
S=626 -> H1=970 H2=531 H3=405
S=226 -> H1=506 H2=860 H3=796
S=236 -> H1=911 H2=256 H3=523
S=468 -> H1=416 H2=960 H3=436
S=529 -> H1=410 H2=349 H3=191
S=630 -> H1=768 H2=518 H3=773
S=923 -> H1=465 H2=190 H3=256
S=335 -> H1=869 H2=132 H3=653
S=802 -> H1=394 H2=118 H3=715
S=958 -> H1=055 H2=533 H3=510
S=967 -> H1=654 H2=474 H3=448
S=656 -> H1=205 H2=899 H3=791
S=242 -> H1=343 H2=243 H3=953
S=973 -> H1=498 H2=825 H3=398
S=219 -> H1=629 H2=173 H3=949
S=437 -> H1=311 H2=323 H3=598
S=624 -> H1=492 H2=622 H3=763
S=670 -> H1=779 H2=401 H3=206
```

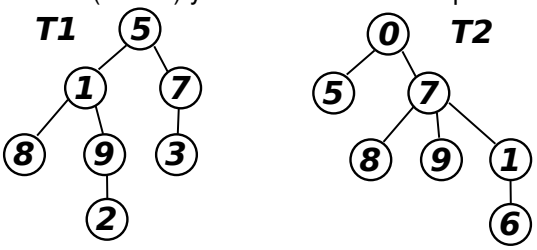
Algoritmos y Estructuras de Datos.
TPL2. Trabajo Práctico de Laboratorio 2. [2018-10-11]

PASSWD PARA EL ZIP: `5F4V GW7C 3P7H`

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[classif]** Implemente una función
`void classify_relative(tree<int> &T,int n1,int n2,int &m1, int&m2);` que dados dos valores nodales `n1` y `n2` en un árbol `T` retorna las distancias `m1` y `m2` de ambos nodos al antecesor común más cercano. Por ejemplo si `T1=(5 (1 8 (9 2)) (7 3))`, `n1=8` y `n2=7` entonces el antecesor común es el nodo `5` (la raíz) y las distancias correspondientes son `m1=2` y `m2=1`.



Más ejemplos:

```
T1=(5 (1 8 (9 2)) (7 3)),    n1=1,n2=1   =>    m1=0,m2=0,
T1=(5 (1 8 (9 2)) (7 3)),    n1=9,n2=5   =>    m1=2,m2=0,
T2=(0 5 (7  8 9 (1 6))),     n1=8,n2=6   =>    m1=1,m2=2,
T2=(0 5 (7  8 9 (1 6))),     n1=6,n2=7   =>    m1=2,m2=0,
```

Nota: Con esta función es muy fácil de determinar el grado de parentezco de los nodos,

- Si `m1=m2=0` los nodos son iguales (o sea `n1=n2`)
- Si `m1=0` y `m2>0`: `n2` es antecesor de `n1` y viceversa
- Si `m1=m2=1` son hermanos
- Si `m1=m2=2` son *primos*
- Si `m1=1 m2=2` entonces `n1` es *tío* de `n2`

Nota: Los árboles son generados tal que los valores almacenados en los nodos son únicos. Por lo tanto dado un valor nodal que se pasa a la función (`n1` y `n2`), está garantizado que están en el árbol y son únicos.

Ayuda:

- Escribir una función
  `void get_path(tree<int> &T,int n,list<int> &L);` que retorna en `L` la lista de valores nodales enel camino que va desde la raíz hasta el nodo que contiene al valor `n`. Por ejemplo si `T1=(5 (1 8 (9 2)) (7 3))`, y `n=8` entonces debe retornar `L=(5,1,8)`.
- Para ello escribir la función recursiva
  `void get_path(tree<int> &T,tree<int>::iterator q,int n,list<int> &L);` que busca el camino pero a partir del nodo `q`. Para ello
  - Si el nodo `q`=Λ retorna la lista vacía.
  - Si el nodo contiene el valor buscado retorna una lista con sólo ese elemento.

- Si ninguna de esas opciones es verdadera recorre los hijos de **q** aplicando recursivamente la función. Si alguno de ellos retorna una lista no vacía **L** entonces prependiza el valor del nodo a esa lista.

- Una vez elaborada la función **get_path()** utilizarla para obtener los caminos **L1** y **L2** a los nodos que contienen los valores **n1** y **n2**. Por ejemplo si **T=(5 (1 8 (9 2)) (7 3))**, **n1=8** y **n2=7** entonces debemos tener **L1=(5,1,8)** y **L2=(5,7)**.
- Estas listas deben empezar en el valor de la raíz y terminar en los valores buscados **n1** y **n2**.
- Recorrer estas listas (que son los caminos) a partir de la raíz hasta el antecesor común, es decir el último valor nodal que es igual, en este caso el 5.
- A partir de allí sólo basta contar los nodos desde ese antecesor común hasta el fin de cada una de las listas. En el ejemplo **m1=2** para ir desde el 5 hasta el 8 y **m2=1** para ir desde el 5 hasta el 7.
- Sugerimos concentrarse primero en escribir correctamente la función **get_path()** y después pasar a **classify_relative()**.

[Ej. 2] **[prom-path]** Dado un árbol **T**, escribir una función **float prom_path(tree<int> &T);** que retorne la longitud promedio de los caminos desde la raíz a las hojas. Por ejemplo, para el árbol **T=(1 (2 3 4 (5 6)) 7 8)**, los 5 posibles caminos desde la raiz a una hoja son: **{1,2,3}, {1,2,4}, {1,2,5,6}, {1,7}, {1,8}**; cuyas longitudes son 2, 2, 3, 1 y 1; lo cual da un promedio de (2+2+3+1+1)/5 = 9/5 = 1.8.

[Ej. 3] **[filtra-deps]** Se tiene un **map<string,list<string>>** que representa un grafo dirigido donde los nodos son nombres paquetes de software en un repositorio, y los arcos dependencias entre paquetes.
Por ejemplo, en:

```
zinjai->{gdb,gcc,gtk,xterm}
pseint->{gtk,glut}
gtk->{Xorg}
glut->{mesa}
mesa->{Xorg}
xterm->{Xorg}
Xorg->{}
gcc->{}
gdb->{}
```

pseint->{gtk,glut} indica que para usar el paquete **pseint** es necesario instalar antes los paquetes **gtk** y **glut**; pero a su vez, para instalar estos paquetes, antes se deben instalar **mesa** y **Xorg** por sus respectivas dependencias.
Debe implementar una función
void filtra_deps(map<string,list<string>> &G, list<string> &L); que reciba el mapa del repositorio completo y una **list<string>** con nombres de paquetes; y recorte el mapa de forma que solo incluya a los paquetes a instalar y sus dependencias (directas o indirectas).
Por ejemplo, si la lista **L** es **{xterm,pseint}**, debe retornar el grafo:

```
pseint->{xterm,gtk,glut}
gtk->{Xorg}
glut->{mesa,Xorg}
mesa->{Xorg}
xterm->{Xorg}
Xorg->{}
```

Ayuda: Se sugiere utilizar el siguiente algoritmo:

- Por **cada paquete** `P` de la lista:
  a) Marcar el paquete `P`, y recursivamente todas sus dependencias, como "necesario". ("Necesarios"será un conjunto de paquetes a implementar de forma similar al conjunto de "visitados.[en] otros ejercicios.)
- Por **cada nodo** `N` del grafo:
  a) Si no es "necesario", borrar el nodo `N` del grafo.
  b) Si es “necesario”, por cada dependencia `D`:
    1) Si no es "necesaria", borrar `D` de la lista de dependencias del nodo `N`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- `ucase`: Además las funciones `eval()` tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento ‘ucase’ (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en `ucase=-1` que quiere “hacer todos”. Por ejemplo `ev.eval4(prune_to_level,1,51);` corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función `eval<j>` están almacenados en un archivo `test1.json` o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. `datain` son los datos pasados a la función y `output` la salida producida por la función de usuario. `ucase` es el número de caso.

```
{ "datain": {
  "T1": "( 0 (1 2) (3 4 5 6) )",
  "T2": "( 0 (2 4) (6 8 10 12) )",
  "func": "doble" },
```

```
"output": { "retval": true },
"ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

TPL2. Trabajo Práctico de Laboratorio 2. [2018-10-11]. TABLA SEED/HASH

```
S=123 -> H1=323 H2=517 H3=090
S=577 -> H1=266 H2=672 H3=435
S=393 -> H1=888 H2=905 H3=840
S=686 -> H1=387 H2=046 H3=588
S=349 -> H1=680 H2=327 H3=451
S=762 -> H1=612 H2=218 H3=777
S=690 -> H1=785 H2=722 H3=807
S=663 -> H1=443 H2=705 H3=693
S=340 -> H1=835 H2=464 H3=817
S=872 -> H1=760 H2=279 H3=613
S=711 -> H1=793 H2=867 H3=294
S=367 -> H1=246 H2=870 H3=345
S=882 -> H1=311 H2=021 H3=077
S=162 -> H1=187 H2=196 H3=279
S=767 -> H1=678 H2=213 H3=023
S=429 -> H1=548 H2=545 H3=489
S=622 -> H1=395 H2=076 H3=382
S=969 -> H1=672 H2=651 H3=940
S=893 -> H1=541 H2=213 H3=118
S=311 -> H1=578 H2=792 H3=066
S=529 -> H1=275 H2=597 H3=157
S=721 -> H1=005 H2=607 H3=259
S=384 -> H1=942 H2=732 H3=681
S=798 -> H1=523 H2=089 H3=821
S=615 -> H1=605 H2=718 H3=314
S=386 -> H1=138 H2=650 H3=928
S=215 -> H1=938 H2=113 H3=343
S=935 -> H1=103 H2=809 H3=668
S=292 -> H1=341 H2=459 H3=088
S=821 -> H1=374 H2=976 H3=644
S=527 -> H1=959 H2=425 H3=505
S=359 -> H1=465 H2=978 H3=297
S=626 -> H1=242 H2=178 H3=861
S=226 -> H1=977 H2=055 H3=829
S=236 -> H1=610 H2=671 H3=451
S=468 -> H1=902 H2=883 H3=264
S=529 -> H1=275 H2=597 H3=157
S=630 -> H1=507 H2=718 H3=456
S=923 -> H1=806 H2=729 H3=788
S=335 -> H1=628 H2=258 H3=288
S=802 -> H1=397 H2=950 H3=557
S=958 -> H1=497 H2=958 H3=992
S=967 -> H1=013 H2=377 H3=578
S=656 -> H1=806 H2=172 H3=844
S=242 -> H1=184 H2=894 H3=857
S=973 -> H1=037 H2=840 H3=958
S=219 -> H1=293 H2=663 H3=592
S=437 -> H1=174 H2=582 H3=806
S=624 -> H1=623 H2=194 H3=759
S=670 -> H1=266 H2=092 H3=878
```

Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio 3. [2018-11-01]

PASSWD PARA EL ZIP: 8QGV JQHW XGP7

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[make-full]** Implementar una función
`void make_full(btree<int> &T);` que elimina (in-place) los nodos interiores de un árbol binario que tienen un sólo hijo, de manera que el árbol resultante es un árbol binario lleno (Full Binary Tree). En el árbol resultante sólo deben quedar hojas o árboles interiores con dos hijos.
Ejemplos:

```
T(input): (5 . (7 . (3 2 4))) => T(output): (3 2 4)
T(input): (5 (6 8 9) .)       => T(output): (6 8 9)
T(input): (5 . (6 8 9))       => T(output): (6 8 9)
T(input): (1 (2 . 4) (3 . 5)) => T(output): (1 4 5)
```

Ayuda:
Escribir una función recursiva
`void make_full(btree<int> &T, btree<int>::iterator n);` que haga lleno el subárbol del nodo `n` de la siguiente manera:

- Sean `cl` y `cr` los hijos izquierdo y derecho del nodo `n`.
- Si solo uno de ellos (`cl` ó `cr`) es Λ:
  - Crear un árbol vacío `T2`.
  - Mover(`splice`) el subárbol de `cl`/`cr` a `T2`.
  - Eliminar el nodo `n` de `T`.
  - Mover(`splice`) `T2` de vuelta a `n`.
  - Aplicar recursivamente `make_full` al nuevo `it`
- Si no:
  - Si `cl` no es Λ, aplicar `make_full` recursivamente al subarbol izquierdo.
  - Si `cr` no es Λ, aplicar `make_full` recursivamente al subarbol derecho.

[Ej. 2] **[max-subtree]** Escribir una función,
`int max_subbtree(btree<int>&T);` que retorna la suma de etiquetas máxima de entre todos los posibles subárboles del árbol binario T.
Nota: Las etiquetas de los nodos pueden ser negativas y positivas (el subárbol vacío puede ser el de mayor suma, en este caso 0).
Ejemplos:

```
T=(1 2 3)                 => 6
T=(1 -1 1)                => 1
T=(1 -2 -4)               => 0
T=(-2 (2 2 -1) (2 3 -1))  => 5
T=(-2 (2 2 -1) (2 -3 -1)) => 3
```

Ayuda:

- Puede ser útil realizar una doble recursión.
- Escribir una función recursiva auxiliar que calcula la suma del subárbol de un nodo dato.
- En la función principal una primera recursión para recorrer todo el árbol, y una interna para obtener la suma de todo subárbol con raíz en el nodo que se está analizando.

[Ej. 3] **[most-connected]** Implementar una función `int most_connected(vector< set<int> > &VS);` que retorna el índice `j` tal que el conjunto `VS[j]` es el que está *conectado* con un mayor número de otros conjuntos de `VS`. Decimos que dos conjuntos están conectados si no son disjuntos, es decir, si tienen intersección común no vacía.
Nota: En el caso de haber varios conjuntos con el mismo número de conexiones debe retornar el primero de ellos.
Ejemplos:

```
VS=[{0},{1},{2},{0,1,2}],                       retorna 3
VS=[{0,1,2},{0},{1},{2}],                       retorna 0
VS=[{0,6,9},{5,6,9},{5},{1},{5,9},{5},{1,5,7}], retorna 1
```

Ayuda:

- Escribir una función auxiliar `bool connected(set<int> &s1,set<int> &s2);` que retorna `true` si los conjuntos no son disjuntos.
- Hacer un doble lazo sobre los conjuntos de `VS`.
- Para cada conjunto en la iteración exterior `VS[j]` determinar en el lazo interior cuantos conjuntos `VS[k]` están conectados.
- Mantener variables `jmax`, `nmax`, que contienen el índice correspondiente al máximo actual, y el número de conexiones del máximo actual.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado `program.cpp`. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si `f` es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (`eval<1>` y `evalr<1>`). La primera `ev.eval<j>(f,vrbs);` toma una serie de casos de prueba de entrada, le aplica la función del usuario `f` y compara la salida del usuario (`user`) con respecto a la esperada (`ref`). Si la verbosidad (el argumento `vrbs`) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
   "T1": "( 0 (1 2) (3 4 5 6) )",
   "T2": "( 0 (2 4) (6 8 10 12) )",
   "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

mstorti@galileo/aed-3.1-302-gf1b8e1bc/Thu Nov 1 21:35:31 2018 -0300

Algoritmos y Estructuras de Datos.
Recuperatorio de Trabajos Prácticos de Laboratorio. [2018-11-22]

PASSWD PARA EL ZIP: FRP5 2PF7 XR7J

Ejercicios

[Ej. 1] [tpl2 - tree-comp]

Queremos escribir una función predicado **`bool tree_less(tree<int> &T1,tree<int> &T2)`** que sea una relación de orden fuerte para AOO. Debe retornar **`true`** si **T1**<**T2**. Recordemos que una relación de orden fuerte debe ser transitiva y también debe satisfacer que si **T1**<**T2** y **T2**<**T1** son falsos debe ser **T1**=**T2**.

Recordemos cómo procedemos con objetos que tiene varios campos miembro **`class A { public: int j; double z; string s; }`**. Si tenemos dos objetos **a1** y **a2**, comparamos primero el campo **j**, es decir si **`a1.j<a2.j`** retornamos **`true`**, si **`a2.j<a1.j`** retornamos **`false`**, y si ninguno de los dos es cierto continuamos con el siguiente campo **z** y después **s**. Si todos los campos son iguales debemos retornar **`false`** ya que quiere decir que todos los campos son iguales.

En el caso de un AOO podemos pensar que es un objeto compuesto del elemento contenido en la raíz **r** y de una lista de objetos que son los subárboles de los hijos de la raíz, es decir **`T=(r (a...) (b...) (c...) ...)`** donde **a** es el hijo más izquierdo de **r**, **b** el segundo, y así siguiendo. Entonces dados dos árboles, **`T1=(r1 (a1...) (b1...) (c1...) ...)`**, y **`T2=(r2 (a2...) (b2...) (c2...) ...)`**.

Además el subárbol vacío Λ equivale a $-\infty$.

Entonces la función de comparación debe hacer lo siguiente:

- Si **T1**==Λ y **T2**==Λ retornar **`false`** (son iguales)
- Si **T1**!=Λ y **T2**==Λ retornar **`false`** (**T2** es $-\infty$ y nadie puede ser menor que $-\infty$)
- Si **T1**==Λ y **T2**!=Λ retornar **`true`** **T1** es $-\infty$ y **T2** no $\Rightarrow$ **T1**<**T2**)

Pasado este punto sabemos que ninguno de los dos es Λ y debemos comparar dos árboles que no son vacíos:

- Si **r1**<**r2** retornar **`true`**, si **r2**<**r1** retornar **`false`**.
- Si **`(a1...)`**<**`(a2...)`** retornar **`true`** si **`(a2...)`**<**`(a1...)`** retornar **`false`**.
- Idem para **`(b1...) (b2...)`**
- Idem para **`(c1...) y (c2...)`** hasta recorrer todos los hijos.

Tener en cuenta que para la raíz se deben comparar los valores almacenados en los nodos, mientras que para los hijos se deben comparar los subárboles, aplicando la función recursivamente.

Si al recorrer las listas de hijos, son todos iguales y algunas de ellas se acaba entonces hay que pensar como que los Λ equivalen a $-\infty$ es decir,

- Si se termina la lista de hijos de **T1** pero no la de **T2** retornar **`true`** (ya que **T1**<**T2**)
- Si termina la lista de hijos de **T2** pero no la de **T1** retornar **`false`** (ya que **T1**>**T2**)
- Si ambas listas se terminaron retornar **`false.`** (ya que **T1**=**T2**)

[Ej. 2] [tpl1 - xcommon]

Implemente una función `void xcommon(list<int> &L1,list<int> &L2,list<int> &common)` tal que dadas dos listas `L1` y `L2` extraiga la parte común del comienzo de `Lcommon`, de tal forma que `L2=(Lcommon,Ltail1)` y `L2=(Lcommon,Ltail2)`. Adicionalmente, los argumentos `L1`, `L2` deben quedar con las "colas" correspondientes `Ltail1` y `Ltail2`.

Ejemplo: si `L1=(0,3,2,1,5,6)` y `L2=(0,3,2,4)` ⇒ `Lcommon=(0,3,2)`, `L1=(1,5,6)`, `L2=(4)`

[Ej. 3] [tpl3 - xsubtrees]

Implemente la función `void xsubtrees(btree<int> &T, int depth, list<btree<int>> &Lout)` que dado un árbol binario `T` y un entero `depth` extraer del árbol `T` todos los subárboles de nodos que estén a profundidad `depth` (moverlos hacia la lista de subárboles `Lout`).

[Ej. 4] [tpl3 - maxsubK]

Escriba una función int `maxsubk(set<int> &S, int k)` que devuelva la máxima suma en valor absoluto de todos los subconjuntos posibles del conjunto `S` tomados de a `k`.

Ejemplos: Para `S={2,-3,5,-1,4}`:

- `maxsubk(S,1)` ⇒ `5`
- `maxsubk(S,2)` ⇒ `9`
- `maxsubk(S,3)` ⇒ `12`
- `maxsubk(S,4)` ⇒ `14`

Ayuda

- Implemente una función auxiliar que sume, en valor absoluto, todos los elementos de un conjunto
- Una propuesta de algoritmo recursivo para generar todos los k-subconjuntos realizaría lo siguiente:
  - Si el conjunto parcial ya tiene `k` elementos, retorna su suma
  - Si no hay más elementos en el conjunto total, retorna `-1`
  - Sino:
    - Toma el primer elemento del conjunto total y lo elimina
    - Recursión sin incluir el elemento en el conjunto parcial
    - Recursión incluyendo el elemento en el conjunto parcial
    - Retorna la mayor de las sumas absolutas obtenidas en recursión

[Ej. 5] [tpl2/tpl3 - num-path]

Escriba una función `int num_path(map<int,set<int>>& G, int i, int j)` que retorne el número de caminos (sin ciclos) en el grafo no dirigido `G`, partiendo desde el vértice `i` y finalizando en el vértice `j`. Se garantiza que los nodos `i` y `j` están en el grafo.

Ejemplos: Para `G` = `{ 1→{2,3,4}, 2→{1,3}, 3→{1,2}, 4→{1} }`:

- `num_path(G,1,4)` ⇒ `1`
- `num_path(G,1,3)` ⇒ `2`
- `num_path(G,1,2)` ⇒ `2`
- `num_path(G,3,4)` ⇒ `2`

[Ej. 6] [tpl1 - super-stable-partition]

Escribir una función **`super_stable_partition`** que reciba una lista con enteros **`L`** y dos listas vacías **`L_low`** y **`L_geq`**, y:

- determine si existe alguna posición para particionar la lista en dos sublistas según el valor de dicha posición, sin reordenarla. Es decir, debe encontrar una posición tal que todos los elementos previos a la misma sean menores al valor que hay en dicha posición, y todos los elementos posteriores sean mayores o iguales.
- si existe tal posición (si hay más de una, tome la primera en la secuencia), mueva (splice) ambas partes a las listas **`L_low`** (menores) y **`L_geq`** (mayores o iguales) y retorne **`true`**; si no existe retorne **`false`**.

Ejemplos

- `L = {5,3,2,7,9,7,10}` ⇒ `true, L = {} L_low = {5,3,2}, L_geq = {7,9,7,10}`
- `L = {1,3,2,7,9,7,1}` ⇒ `true, L = {}, L_low = {}, L_geq = {1,3,2,7,9,7,1}`
- `L = {10,3,2,7,9,7,11}` ⇒ `true, L = {}, L_low = {10,3,2,7,9,7}, L_geq = {11}`
- `L = {5,3,2,7,9,7,1}` ⇒ `false, L = {1,3,2,7,9,7,1}, L_low = {}, L_geq = {}`

Ayuda

- Escribir una función auxiliar que dada una lista y una posición, determine si dicha posición es váida para generar la partición
- Recorrer la lista **`L`** determinando en cada posición si es o no válida con la función auxiliar:
  - Si una posición es válida:
    - Mover (splice) a **`L_low`** desde el comienza hasta esa posición
    - Mover (splice) a **`L_geq`** todo lo que queda en **`L`**
    - Finalizar retornando **`true`**
- Si no se encontró posición válida, retornar **`false`**

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del

usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
   "T1": "( 0 (1 2) (3 4 5 6) )",
   "T2": "( 0 (2 4) (6 8 10 12) )",
   "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **seed** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

Algoritmos y Estructuras de Datos.
TPLSR18. Recuperatorio de Trabajos Prácticos de Laboratorio. [2019-02-28]

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[filtra-conjuntos]** Dada una lista de conjuntos `LS_IN` de enteros, y una función `int f(int)`, copiar en otra lista de conjuntos `LS_OUT` solo aquellos conjuntos para los cuales la función `f` es inyectiva (esto es, que genera imágenes diferentes para todos los elementos del conjunto).
Ejemplo: Si `LS_IN=[ (1,2,3), (2,-2,10), (0,5) ]` y `f` es `f(x)=x^2` entonces `LS_OUT` debería quedar `[ (1,2,3), (0,5) ]`, ya que para el segundo conjunto de la lista de entrada hay dos elementos (`2` y `-2`) que generan la misma imagen (`4`), razón por la cual ese conjunto no se omite.

[Ej. 2] **[distancia]** Dado un grafo no ponderado `G`, un vértice de partida `x` y un vértice de llegada `y`, implementar el método `int distancia(map<int,list<int>>& G, int x, int y)` que retorna la distancia entre los vértices pasados como parámetros. Recordar que la distancia entre dos vértices de un grafo no ponderado se puede definir como el número de aristas del camino más corto que los une. Se asegura que los vértices pasados como argumentos pertenecen al grafo, pero puede no existir un camino entre ellos, en cuyo caso hay que devolver un valor grande (1000).
Ayuda:

- Chequear si el vértice actual es el vértice de llegada
- En caso positivo retornar la distancia acumulada hasta el momento
- En caso negativo:
  - Marcar el vértice como visitado
  - Lanzar la recursion a cada uno de los vértices vecinos aún no visitados incrementando la distancia actual en 1. Retornar la menor distancia hallada.
  - Si todos los vecinos ya fueron visitados retornar un valor de distancia muy grande (1000) que indique que no se pudo hallar el vértice de llegada por este camino.

Por ejemplo: Si `G[1] -> {2 ,3}, G[2] -> {1 ,3}, G[3] -> {1 ,4}, G[4] -> {3}` debe dar,

```
distancia(G,1,1) -> 0
distancia(G,2,4) -> 2
distancia(G,1,4) -> 2
distancia(G,4,3) -> 1
```

[Ej. 3] **[suma-en-la-hoja]** Implemente el método `void suma_en_la_hoja(tree<int>& T)` que agregue un hijo a cada hoja del árbol ordenado orientado `T` cuya etiqueta sea la suma de las etiquetas de los vértices que componen el camino desde la raíz hasta dicha hoja. Un árbol vacío retorna un árbol vacío.
Ejemplo: Si `T = (5 (2 1 3) 7)` , luego de `suma_en_la_hoja(T)` el árbol debe quedar:
`(5 (2 (1 8) (3 10)) (7 12))`
Ayuda:

- Implementar una función recursiva que reciba el árbol `T`, un iterador `it`, y un entero `s` que deberá contener la suma de las etiquetas del camino desde la raíz hasta el padre del nodo apuntado por `it`.
- La recursión comienza con `it` apuntando a la raíz y `s=0`.

- Si **it** apunta a una hoja, la función debe insertar la suma final como hijo más izquierdo de dicha posición.
- Si **it** apunta a un nodo interior, la función debe sumar a **s** la etiqueta de **it** e invocarse recursivamente para cada hijo.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **program.cpp**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **f** es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

  j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

```
{ "datain": {
   "T1": "( 0 (1 2) (3 4 5 6) )",
   "T2": "( 0 (2 4) (6 8 10 12) )",
   "func": "doble" },
  "output": { "retval": true },
  "ucase": 0 },
```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una

vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.

- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2019-10-03]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.

2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...

3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS1 (W=20pt)]

a) Defina la estructura de la clase celda de un AOO (`tree<T>::cell`). Presente los atributos y el prototipo de los métodos principales.

b) Defina la estructura de la clase iterador de un AOO (`tree<T>::iterator`). Presente los atributos y el prototipo de los métodos principales.

c) Dado el siguiente fragmento de código:

```
1 tree<char> T;
2 lisp2tree("(a (h (k s)))", T);
3 tree<char>::iterator p = T.begin().lchild().right();
```

En base a *a)* y *b)*, presente un croquis de todas las estructuras `tree<T>::cell` e `tree<T>::iterator` presentes en memoria luego de la ejecución y los punteros que las relacionan.

d) Defina la estructura de la clase celda de una lista simplemente enlazada (`list<T>::cell`). Presente los atributos y el prototipo de los métodos principales.

e) Implemente el constructor por defecto de la clase `list` (simplemente enlazada)

f) Implemente el método

```
1   typedef list<T>::iterator it_t;
2   it_t list<T>::splice(it_t it, list<T>& L,it_t first,it_t last);
```

el cual inserta el rango de elementos `[first, last)` de la lista `L` en la lista `this` utilizando un procedimiento $\mathcal{O}(1)$. Considere listas simplemente enlazadas.

[Ej. 2] [OPER1 (W=20pt)]

a) **[draw]**: Dibujar el `AOO` cuyos nodos, listados en orden previo y posterior son

```
1       ORD-PRE = (C M J K R L A N P F) ,
2       ORD-POST = (J M L N A R F P K C) ,
```

luego particionar el conjunto de nodos respecto al nodo `R`.

b) Un profesor de historia de una carrera a distancia quiere evaluar a sus alumnos presentándole a cada uno de ellos un texto para analizar para que luego respondan un cuestionario de tipo *multiple choice*. En un intento por evitar que dos alumnos se copien, el profesor ha investigado a través de las redes sociales qué alumnos se conocen entre sí; y ahora le pide ayuda a usted para no tener que hacer un enunciado diferente por cada alumno. Modele este problema con un grafo de forma tal que pueda aplicar coloreado de grafos para determinar un número razonable de enunciados diferentes a preparar y cómo distribuirlos entre los alumnos.

c) **[operaciones]**: Sea el árbol `D=(5 (4 3 6) (1 9 8 7) (4 6))`. Después de hacer:

```
auto n = D.find(1);
n++;
n = n.lchild();
n = n.lchild();
n = D.insert(n,2);
```

¿Cómo queda el árbol? ¿Se produce un error?

d) **[Notación $O(\cdot)$]**. cada una de las funciones $T_1, \ldots, T_4$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$) y ordenarlas de forma creciente.

$$T_1 = n^2\,(2n + 4\,n^2) + 2\log_{10} n + \sqrt[2]{n}$$
$$T_2 = 4\,n! + 2\,n^4 + 4 \cdot 3^n$$
$$T_3 = 4 \cdot 4^n + 4^4 + 2n^4$$
$$T_4 = 4^4 + 4.2\log n + \log(15)n^{3.5}$$

e) **[function-f]**: Renombrar la siguiente función con un nombre alusivo a la tarea que efectúa y determinar su orden algorítmico respecto al tamaño del vector **v**.

```
int f(vector<int>& v, int x){
  int i = 0;
  int j = v.size()-1;
  while(i<=j){
    int r = (i+j)/2;
    if(x==v[r]) return r;
    if(x>v[r]){
      i = r+1;
    }else{
      j = r-1;
    }
  }
  return -1;
}
```

[Ej. 3] [PREG1 (W=20pt)]

a) Sea el árbol **(a (b t) (e (q r z) f))**.
- Cuáles de los siguientes son **caminos**?
  - **(b t)**
  - **(r q z)**
  - **(e q z)**
  - **(f e a)**
- Liste las **hojas** del árbol y los nodos a **profundidad** 2.
- ¿Cuál es la **altura** del árbol?

b) Sea la **correspondencia** **M={(6->5),(12->9)}** y ejecutamos el código int **x= M[12]**. ¿Qué ocurre? ¿Qué valores toman **x** y **M**? ¿Y si hacemos **x = M[5]**?

c) Discuta las ventajas y desventajas de utilizar listas **doblemente enlazadas** con respecto a las **simplemente enlazadas**. ¿Cuáles son los métodos cuyo tiempo de ejecución cambia y por qué?

d) ¿Cuál es la **complejidad algorítmica** (mejor/promedio/peor) de las siguientes funciones? (asumir listas implementadas con celdas enlazadas por punteros o cursores)

1) **list<T>::end(),**
2) **list<T>::erase(p),**
3) **list<T>::clear(),**
4) **map<K,V>::find(x),** para la implementación con vectores ordenados,
5) **map<K,V>::find(x),** para la implementación con listas ordenadas.

e) ¿Qué ventajas o desventajas tendría implementar la clase **stack** (pila) en términos de **lista simplemente enlazada** poniendo el tope de la pila en el fin de la lista? ¿Cuáles serían los tiempos de ejecución para **`top`**, **pop**, **`push`**? Lo mismo para cola: ¿Qué ventajas o desventajas tendría implementar la clase **queue** (cola) en términos de **lista simplemente enlazada** poniendo el frente de la cola en el fin de la lista? ¿Cuáles serían los tiempos de ejecución para **`front`**, **pop**, **`push`**?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 2. [2019-11-14]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.

2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...

3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: CLAS2, Hoja #2/3 TORVALDS, LINUS

[Ej. 1] [CLAS2 (W=20pt)]

a) Defina la estructura de la clase celda de un AB (`btree<T>::cell`). Presente los atributos y el prototipo de los métodos principales.

b) Defina la estructura de la clase iterador de un AB (`btree<T>::iterator`). Presente los atributos y el prototipo de los métodos principales.

c) Dado el siguiente fragmento de código:

```
1 btree<char> T;
2 lisp2btree("(a (h k .) m)", T);
3 btree<char>::iterator p = T.begin().left().right();
```

En base a *a)* y *b)*, presente un croquis de todas las estructuras `btree<T>::cell` e `btree<T>::iterator` presentes en memoria luego de la ejecución y los punteros que las relacionan.

d) Implementar una función
`bool openhashtable_insert(vector<list<T> >& table, unsigned int (*hashfunc)(T), T x)`
que inserta el elemento `x` en la tabla de dispersión abierta `table` utilizando la función de dispersión `hashfunc` y retorna un booleano indicando si la inserción fue o no exitosa.

[Ej. 2] [OPER2 (W=20pt)]

a) **[huffman]**: Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario utilizando el algoritmo de Huffman y encodar la palabra `CYBERMONDAY`, $P(O) = 0.10, P(D) = 0.10, P(B) = 0.10, P(N) = 0.30, P(C) = 0.10, P(A) = 0.02, P(E) = 0.02, P(R) = 0.02, P(Y) = 0.04, P(M) = 0.20$. Calcular la longitud promedio del código obtenido.

b) **[hf-decode]** Utilizando el código Lisp
`(. (. (. Y A) (. (. C L) ␣)) (. D (. R (. (. F K) (. I B)))))`
desencodar el mensaje:
`11111010100101001110101111100110111101000 1000`.

c) **[abb]** Dados los enteros $\{14, 8, 21, 3, 4, 11, 6, 5, 4, 13, 2\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 14, 8 y 3 en ese orden.

d) **[hash-dict]** Insertar los números $\{200, 180, 280, 110, 100, 380, 200, 700\}$ en una **tabla de dispersión cerrada** con $B = 10$ cubetas, con función de dispersión **h(x) = x/(B*10)** y redispersión lineal. Muestre la tabla resultante. Luego elimine el elemento 110 y muestre la tabla final.

e) **[quick-sort]** Dados los enteros $\{6, 10, 5, 2, 6, 11, 9, 4, 4, 3, 12, 7\}$ ordenarlos por el método de *clasificación rápida (quick-sort)*. En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos.

[Ej. 3] [PREG2 (W=20pt)]

a) Comente ventajas y desventajas de las **tablas de dispersión abiertas y cerradas**. Además analice costos de inserción y remoción en cada una de ellas.

b) Se quiere representar conjuntos de **cuadrados perfectos** en el rango **`[0,100]`** es decir por ejemplo **`0,4,9,16,25,...,100`**, por **vectores de bits**. ¿Cuál es el tamaño **`N`** del conjunto universal? Completar las funciones **`indx()`** y **`element()`** correspondientes:

```
1       int indx(int p) { /* ... */ }
2       int element(int z) { /* ... */ }
```

c) Si queremos generar un código binario de **longitud fija** para el conjunto de pares de letras (por ejemplo **`df`**, **`qz`**...), o sea en total **`26*26=676`** "caracteres". ¿Cuantos bits tendrá, como **mínimo**, la representación de cada caracter? ¿Y para el conjunto de pares de caracteres mayúsculas y dígitos (**`52*52=2704`** caracteres)?

d) Discuta la **estabilidad** del algoritmo de ordenamiento de **listas por fusión** (**merge-sort**). ¿Es estable? ¿Bajo que condiciones?

e) ¿Cuál es el tiempo de ejecución del algoritmo de **ordenamiento rápido** (**quick-sort**), en el caso promedio y en el peor caso? ¿Cuando se produce el peor caso?

f) ¿Qué dos condiciones cumple un montículo? Explique la utilidad de cada condición.

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

FICH
UNL

Algoritmos y Estructuras de Datos.
Recuperatorio Parciales. [2019-11-21]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS1 (W=20pt)]

a) Dado el siguiente fragmento de código:

```
1 tree<char> T;
2 lisp2tree("(m (j k) r t))", T);
3 tree<char>::iterator p = T.begin().lchild().right().right();
```

Presente un croquis de todas las estructuras **`tree<T>::cell`** e **`tree<T>::iterator`** presentes en memoria luego de la ejecución y los punteros que las relacionan.

b) Demuestre a partir de un ejemplo porqué no es posible utilizar un puntero a celda **`cell*`** como iterador de un AOO.

c) Implemente los métodos **`iterator_t erase(iterator_t p)`** y **`iterator_t erase(iterator_t p, iterator_t q)`** de la clase lista doblemente enlazada.

[Ej. 2] [OPER1 (W=20pt)]

a) **[rec-arbol]**: Dibujar el **`AOO`** cuyos nodos, listados en orden previo y posterior son

```
1         ORD-PRE = (A B F G H C D E J),
2         ORD-POST = (G H F B C J E D A),
```

luego particionar el conjunto de nodos respecto al nodo **`C`**.

b) **[operaciones]**: Sea el árbol **`G=(5 (3 4 6) (2 7) 1 (9 8 10))`**. Después de hacer:

```
1         auto n = G.find(1);
2         n++;
3         n = n.lchild();
4         n = n.lchild();
5         n = D.insert(n,11);
```

¿Cómo queda el árbol? ¿Se produce un error?

c) **[Notación $O(\cdot)$]**. Para cada una de las funciones $T_1, \ldots, T_5$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$) y ordenarlas de forma creciente.

$$T_1 = n^3\,(2n^2 + 5\,n^3) + 2\log_{10} n + \sqrt[2]{n}$$
$$T_2 = 4n! + 2^n + 5 \cdot 2^n$$
$$T_3 = 7 \cdot 4^n + 4^4 + 2n^4$$
$$T_4 = 4.2 \log n + 5^n + 10n!$$
$$T_5 = 40^9 + 8 \log n + \log(50) n^2$$

d) **[size]**: Suponga que cierto procesador tiene una frecuencia de reloj de 1 Ghz, y simplifique el análisis suponiendo que por cada diez ciclos de reloj se realiza una operación en particular. Estime de qué tamaño puede ser el problema que se puede resolver en diez segundos usando un algoritmo que requiere $T(n)$ operaciones, con los siguientes valores de $T(n)$: $log(n)$, n y $2n$.

e) **[orden]**: Dada la siguiente función:

```
1     set<pair<int,int>> f(set<int>& A,set<int>& B){
2         set<pair<int,int>> C;
3         for(int a:A)
4            for(int b:B)
5                C.insert(pair(a,b));
6        return C;
7     }
```

- Renombrar la función con un nombre alusivo a la tarea que efectúa.
- Determinar el orden algorítmico de **f** respecto al tamaño de los conjuntos **A** y **B**.

[Ej. 3] [PREG1 (W=20pt)]

a) ¿Cuantas posiciones **no dereferenciables** puede haber en una **lista**? ¿Y en un **árbol**?

b) Supongamos que tenemos un árbol (AOO) `T=(6 (0 1 3) 2 7)` y un iterator `p` apuntando al 2. ¿Cómo queda `T` si hacemos una inserción: `T.insert(p,9);`

c) En una **correspondencia**: ¿puede haber una **misma clave** con **diferentes valores**? Por ej. `M=(5->8, 5->9)` ¿Puede haber **diferentes claves** con el **mismo valor**? Por ej. `M=(4->3, 8->3)`

d) Sea el árbol `(r (a (p t u)) (b q z))`. Cuáles de los siguientes son **caminos**?

- `(p a r)`
- `(r a p)`
- `(a r b z)`
- `(a p u)`

e) ¿Cuál es el **tiempo de ejecución** (mejor/promedio/peor) de las siguientes funciones? (asumir listas implementadas con **celdas enlazadas por punteros o cursores**)

1) `list<T>::begin(),`
2) `list<T>::insert(p,x),`
3) `list<T>::clear(p,x),`
4) `map<K,V>::find(x),` para la implementación con vectores ordenados,
5) `map<K,V>::find(x),` para la implementación con listas ordenadas.

f) ¿Como se define la **notación asintótica** $T(n) = O(f(n))$? ¿Porqué decimos que $(n+1)^2 = O(n^2)$ si siempre es $(n+1)^2 > n^2$?

[Ej. 4] [CLAS2 (W=20pt)]

a) Implementar una función
`bool closedhashtable_insert(vector<T>& table, unsigned int (*hashfunc)(T), T x)`
que inserta el elemento `x` en la tabla de dispersión cerrada `table` utilizando la función de dispersión `hashfunc` y redispersión lineal, retornando un booleano indicando si la inserción fue o no exitosa.

b) Implemente el método `iterator_t splice(iterator_t to, iterator_t from)` de la clase árbol binario.

c) ¿Cómo se define un `iterator_t` de la estructura conjunto (`set<T>`) implementado a partir de un árbol binario de búsqueda?

[Ej. 5] [OPER2 (W=20pt)]

a) **[hf-decode]** Utilizando el código Lisp
`(. (. E O) (. (. (. (. K M) L) (. W (. F S))) (. B U)))`
desencodar el mensaje: `11000011010111100110110`.

b) **[huffman]**: Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario utilizando el algoritmo de Huffman y encodar la palabra **PARALELISMO**, $P(P) = 0.01, P(A) = 0.05, P(R) = 0.04, P(M) = 0.10, P(U) = 0.20, P(O) = 0.20, P(E) = 0.05, P(S) = 0.20, P(I) = 0.05, P(L) = 0.10$. Calcular la longitud promedio del código obtenido.

c) **[hash]**: Insertar los enteros **{2, 7, 5, 12, 7, 33, 9}** en una tabla de dispersión cerrada con B=9 cubetas, con función de dispersión **h(x) = x%B** y redispersión linear. Luego eliminar el elemento **2** e insertar el elemento **38**, en ese orden. Mostrar la tabla resultante.

d) **[heap-sort]**: Dados los enteros **(3,12,10,4,5,15,7)** ordenarlos por el método de **montículos** (**heap-sort**). Mostrar el montículo (minimal) antes y después de cada inserción/supresión.

[Ej. 6] [PREG2 (W=20pt)]

a) Explique el concepto de estrategia **por búsqueda exhaustiva**. Si le piden colorear un grafo de 200 vértices, ¿utilizaría una estrategia exhaustiva para resolver el problema? Justifique.

b) Si tenemos un iterator **q** dereferenciable en un **set<>** ¿es posible asignarle un valor? Es decir, dado el siguiente código, ¿puede falla línea **(1)**? ¿porqué?

```
set<string> S;
//... llena S
auto q = S.begin();
// (Asumir que q es dereferenciable)
*q = "Spiderman"; // (1)
```

c) Escriba las instrucciones necesarias para crear el siguiente **árbol binario (AB)**: **T=(3 . (4 2 .))**

d) Explique que operaciones realizan (semántica) las dos versiones de **erase** para conjuntos.

```
1  void erase(iterator p);   // (1)
2  int erase(key_t x);       // (2)
```

Explicar qué son los valores de retorno, y porqué una retorna un entero y la otra no. ¿Cuando puede fallar la versión de iterator (1)? ¿Puede fallar la versión (2) si la clave **x** no tiene asignación?

e) Discuta el número de **intercambios** que realizan los algoritmos de ordenamiento lentos en el peor caso.

f) ¿Porqué el árbol binario de una códificación binaria para compresión debe ser un **árbol binario lleno (FBT)**? (Recordemos que un FBT es áquel cuyos hijos son hojas o nodos interiores con sus dos hijos.)

mstorti@galileo/tpl2-2019-88-gff840e36/Sun Nov 24 18:14:16 2019 -0300

Algoritmos y Estructuras de Datos.
Trabajo Práctico de Laboratorio 1. [2019-09-05]

PASSWD PARA EL ZIP: 364C 29H6 RGV8

Ejercicios

[Ej. 1] [Pareto] (lista)

Se dice que un punto $x = (x_1, x_2, .., x_n)$ n-dimensional domina a otro punto $y = (y_1, y_2, .., y_n)$ si se cumple que:

$$\forall k \in [1, n] : x_k \leq y_k, \text{ y } \exists k_0 \ \in \ [1, n] \ / \ x_{k_0} < y_{k_0} \tag{1}$$

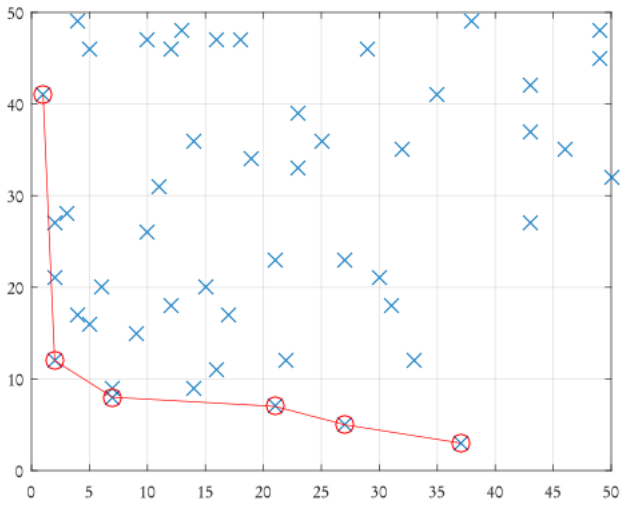
Por ejemplo **x=[2,1,5]** domina a **y=[2,2,5]**, pero dado **z=[1,2,5]** no es cierto ni que **x** domina a **z** ni **z** domina a **x**.

Entonces, dada una lista de puntos de coordenadas enteras positivas **`list<vector<int>> L`**, implemente una función **`list<vector<int>> Pareto(list<vector<int>>&L);`** que retorne la lista de puntos no-dominados (aquellos que no son dominados por ningún otro punto). Se garantiza que cada **`vector<int>`** tiene **n** coordenadas. La lista de retorno debe devolver los vectores en el mismo orden en que se encontraban en **L**.

Nota: Formalmente, lo que se solicita es que retorne los puntos que pertenecen a la frontera de Pareto, muy utilizada en problemas de optimización con múltiples objetivos.

Ejemplo: Si **L=[[2,1,5],[2,2,5],[1,2,5]]** en tonces **pareto(L)** debe retornar **[[2,1,5],[1,2,5]]**, ya que **[2,2,5]** es dominado por **[2,1,5]** pero **[2,1,5]** y +[1,2,5]+ no se dominan entre sí.

Ayuda: Se sugiere implementar una función auxiliar **`bool domina(vector<int> x, vector<int> y);`** que reciba dos puntos n-dimensionales y que retorne verdadero en caso de que **x** domine a **y**.



[Ej. 2] [merge-kway] (cola)

Implementar una función void **`merge_kway(vector< queue<int> > &qords,queue<int> &merged)`** que dado un vector de colas ordenadas **ordqs**, obtener la cola **merged** resultante de la fusión de todas las colas en una sola, de forma de que los elementos siguen ordenadas. Por ejemplo si **`ordqs=((1,3,5,6),(0,3,5,8),(6,9,10))`** entonces debe dar **`merged=(0,1,3,3,5,5,6,6,8,9,10)`**. El algoritmo puede ser destructivo sobre **ordqs**.

Ayuda: Se sugiere seguir el siguiente algoritmo:

- Eliminar todas las colas vacías del vector.
- Si no quedaron colas (estaban todas vacías) el proceso terminó.
- Recorrer todos los frentes de la colas, buscando el frente menor. Retener el índice de la cola que contiene el frente menor. Por ejemplo en el caso anterior daría el índice 1 ya que el frente menor es el elemento 0 que está en la cola 1.
- Sacar el elemento de la cola correspondiente e insertarlo en **`merged`**.
- Repetir hasta que todas las colas estén vacías.

[Ej. 3] **[is-balanced] (stack)** Dado un string **`expr`**, implementar una función **`bool is_balanced(string &expr);`** que determine si el par y orden de **`{`**, **`}`**, **`(`**, **`)`**, **`[`**, **`]`** está **balanceado**. Ejemplos:

- **`[ ( ) ] { } { [ ( ) ( ) ] ( ) }`** está **balanceado**
- **`[ ( ] )`** no está **balanceado**.

Ayuda: Se propone el siguiente algoritmo:

- Declarar una pila de **`char S`**.
- Recorrer el string **`expr`**.
- Si el caracter actual es un símbolo de apertura, entonces apilarlo.
- Si el caracter actual es un símbolo de cierre y matchea con su símbolo de apertura, entonces quitarlo de la pila. Si no retornar falso. Atención: Chequear que la pila no esté vacía.
- Una vez recorrido todo el string, si la pila está vacía, entonces la expresión está balanceada.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

```
ev.eval<j>(f,vrbs);
hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
```

`j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

```
m: 10, k: 3
T(ref): (10 (7 (4 1) 1) (4 1) 1)
T(user): (10 (7 (4 1) 1) (4 1) 1)
EJ1|Caso0. Estado: OK
```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

```
{ "datain": {
  "T1": "( 0 (1 2) (3 4 5 6) )",
  "T2": "( 0 (2 4) (6 8 10 12) )",
  "func": "doble" },
 "output": { "retval": true },
 "ucase": 0 },
```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum `H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

Algoritmos y Estructuras de Datos.
Trabajo Práctico de Laboratorio 2. [2019-10-10]

PASSWD PARA EL ZIP: `R4FC J3PR VQM6`

Ejercicios

[Ej. 1] [fill-oprev] (AOO)
Dado un AOO `T` y una lista `L`, escribir una función `void fill_ordprev(tree<int> &T,list<int> &L);` que reemplaza los valores de `T` por los valores en `L`, en orden previo. Si la lista `L` tiene menos elementos que nodos en `T` entonces los nodos restantes de `T` quedan en su valor original. Recíprocamente, si `L` tiene más valores que `T` entonces los valores restantes de `L` son descartados.
Ejemplos:

```
L:[6,7,8],T:(1 2 3 4 5)          =>      T:(6 7 8 4 5),
L:[6,7,8,9,10,11],T:(1 2 3 4 5) =>      T:(6 7 8 9 10),
L:[6,7,8],T:(1 (2 3 4) 5)        =>      T:(6 (7 8 4) 5),
L:[6,7,8,9,10,11,12],T:(1 (2 3 4) 5) => T:(6 (7 8 9) 10),
```

Nota: la función puede ser *destructiva* sobre `L`.
Ayuda:

- Considerar una función recursiva.
- La función debe chequear si el nodo no es Λ y si la lista no está vacía.
- Extraer un valor de la lista y ponerlo en la raíz.
- Aplicar recursivamente a los hijos del árbol.

[Ej. 2] [a-lo-ancho] (grafos, AOO) Implemente la función `void ancho(graph& G, tree<char>& T)` que reciba un grafo no dirigido `G` y genere un árbol de expansión `T` de la componente conexa a la que pertenece el primer vértice del grafo (que será la raíz del árbol), a partir del algoritmo de búsqueda a lo ancho. Recordar que en la búsqueda en anchura o a lo ancho:

- Se toma el primer vértice de `G` como raíz del árbol `T`.
- Añadimos al árbol todos los vértices adyacentes (forman el nivel 1 del árbol) en orden alfabético (se garantiza que la lista de adyacentes está ordenada de este modo)
- Por cada vértice del nivel 1, añadimos a `T` todos los vértices incidentes en él, siempre que no formen ciclo (no hayan sido previamente visitados). Se conforma el nivel 2.
- Repetimos el procedimiento hasta que no haya vértices en el nivel `j` actual.

[Ej. 3] [intersect-map] (correspondencia) Implemente una función `void intersect_map(map<int,list<int» &A, map<int, list<int» &B,map<int, list<int» &C)` que a partir de las correspondencias `A` y `B` construye una correspondencia `C` de manera que las claves de `C` son la intersección de las claves de `A` y `B` y para cada clave `k` en `C` la imagen `C[k]` es la intersección ordenada de las listas ordenadas `A[k]` y `B[k]`. Si ésta intersección de listas es nula, no debe incluirse la entrada de clave `k` en `C`. **Ejemplo:**

```
A:[0->{}, 3->{0,1}],
B:[0->{0,1,2}, 1->{0,1,2,3}, 3->{0,2}]
=>
C:[3->{0}]
```

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum `H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es

un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.

- **`void Eval::dump(list <int> &L,string s="")`**

- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

Algoritmos y Estructuras de Datos.
TPL3. Trabajo Práctico de Laboratorio 3. [2019-10-31]

PASSWD PARA EL ZIP: 2GQ5 3M67 RG27

Ejercicios

[Ej. 1] [spaceship] (AB)

Recordemos que una relación de orden puede definirse a través de un predicado binario `bool operator<(T a,T b);`. Vimos también que, en realidad, puede definirse a partir de cualquiera de los operadores <, <=, >, >=. Alternativamente puede definirse el operador "nave espacial" (*spaceship* operator) introducido en la norma **C++20**, definido de la siguiente forma: **a<=>b** debe retornar **+1** si **a>b**, 0 si **a==b**, -1 si **a<b**. La ventaja del operador <=> es que cualquiera de los otros operadores puede escribirse en términos de <=> con una sola llamada al mismo.
En realidad no es necesario que spaceship retorne exactamente **-1,0,1** sino que basta con que retorne un número positivo o negativo. Por lo tanto para enteros basta con que **spacehip(a,b)** retorne **a-b**.
Consigna: Escribir una función `int spaceship(btree<int> &T1,btree<int>&T2);` que define el operador **spaceship** para árboles binarios (AB).
[**Nota:** Como el operador <=> no puede ser sobrecargado en las versiones actuales del compilador lo implementaremos en términos de una función **spaceship(.,.)**.]
Para el árbol binario podemos definir **spaceship** en forma recursiva de la siguiente forma. Básicamente se compara primero la raíz, después los hijos izquierdos, despues los derechos. En cada caso si el resultado de **spaceship** de la comparación no es nulo, ya se puede retornar ese valor. Básicamente es lo siguiente,

- Si **spaceship(*n1,*n2)!=0** retornar **spaceship(*n1,*n2)**
- Si **spaceship(l1,l2)!=0** retornar **spaceship(l1,l2)**
- Si **spaceship(r1,r2)!=0** retornar **spaceship(r1,r2)**
- Retornar 0

Sin embargo, a estas instrucciones hay que modificarlas para considerar el caso de que los nodos **n1** o **n2** sean **Lambda**. Lo más simple es considerar que, si un nodo es **Lambda** entonces debe considerarse como $-\infty$ (como hacemos en la comparación lexicográfica). Entonces,

- Si **n1==n2==Lambda** entonces retorna **0**.
- Si **n1==Lambda** y **n2!=Lambda**, retorna **-1**.
- Si **n1!=Lambda** y **n2==Lambda**, retorna **+1**.
- Si **spaceship(*n1,*n2)!=0** retornar **spaceship(*n1,*n2)**
- Si **spaceship(l1,l2)!=0** retornar **spaceship(l1,l2)**
- Si **spaceship(r1,r2)!=0** retornar **spaceship(r1,r2)**
- Si llegamos aquí los árboles son iguales y por lo tanto debe retornar **0**

Notar que en la línea donde dice **spaceship(*n1,*n2)** debe aplicarse el spaceship de enteros, como se describió más arriba.

[Ej. 2] [mkhuffman] (AB)

Implemente la función **btree<char>** `makeHuffmannTree(list<pair<btree<char>,float>> & bosque);` que reciba una lista de pares **pair<btree<char>, float> bosque** con los árboles iniciales y su peso respectivo (procentaje de ocurrencia) del proceso de Huffmann y retorne el árbol de la codificación resultante.

Ayuda:

- Mientras haya más de un árbol en el bosque:
  - Seleccionar los dos árboles de peso mínimo `Tl` y `Tr`. Para ello deberá utilizar la función auxiliar **`getMin`** provista que retorna un iterador al árbol de peso mínimo.
  - Remover del bosque los pares que involucran dichos árboles
  - Generar un par con los datos `T = ("-" Tl Tr)` y `w = wl+wr` y almacenarlo en la última posición del bosque. Notar que se inserta un char guión "`-`" como raíz.
- Retornar el árbol resultante

[Ej. 3] [min-cost-sets] (Conjuntos)

Dado un conjunto universal `U` de n elementos y una lista de subconjuntos de `U` denominada `S = (S1, S2,...,Sm)` en donde cada subconjunto `Si` tiene un costo asociado, se pide implementar una función `int minCostSets(set<int>& U, vector<set<int>>& S, map<set<int>, int>& costos);` que retorna el costo de la sublista de `S` de costo mínimo y que cubra todos los elementos de `U`. Si no se puede cubrir el conjunto `U` se debe retornar `INT_MAX`.

Ejemplo 1: Retorna costo mínimo 13 correspondiente a la sublista `(S2, S3)`.

```
U = {1,2,3,4,5}
S = {S1,S2,S3}

S1 = {4,1,3},    Cost(S1) = 5
S2 = {2,5},      Cost(S2) = 10
S3 = {1,4,3,2},  Cost(S3) = 3
```

Ejemplo 2: Retorna costo mínimo `INT_MAX` ya que no se puede cubrir `U`.

```
U = {1,6}
S = {S1,S2}
S1 = {1},    Cost(S1) = 2
S2 = {},      Cost(S2) = 4
```

Ayuda:

- Escribir una función auxiliar recursiva `int minCostSets(set<int>& U, vector<set<int>> S, map<set<int>, int>& costos, list<set<int>>& L, int pos);` donde `L` es la lista de conjuntos que participan de la solución actual.
- Utilizar el índice `pos` para procesar el vector de conjuntos `S` y poder desplazarse sobre él en cada llamada a la recursión.
- Cuando se inserta un conjunto en la lista de conjuntos `L`, quitar todos sus elementos del conjunto `U` para reducir el problema y llegar a una solución. Recordar que se busca el listado de subconjuntos tal que su costo sea mínimo y que su unión sea igual al conjunto `U`.
- Por ejemplo, dado el conjunto `U = {1, 2, 3, 4, 5}`, si se insertó el conjunto `Si = {1, 2, 3}` en `L` entonces, al quitar los elementos de `Si`, el conjunto queda como `U = {4, 5}`.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.

- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum `H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.
  - **`void Eval::dump(list <int> &L,string s="")`**
- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

mstorti@galileo/tpl2-2019-52-g9b8020c4/Thu Oct 31 17:34:09 2019 -0300

TPL3. Trabajo Práctico de Laboratorio 3. [2019-10-31]. TABLA SEED/HASH

```
S=123 -> H1=431 H2=882 H3=507
S=577 -> H1=703 H2=452 H3=981
S=393 -> H1=183 H2=648 H3=795
S=686 -> H1=165 H2=115 H3=421
S=349 -> H1=309 H2=104 H3=045
S=762 -> H1=681 H2=933 H3=407
S=690 -> H1=622 H2=926 H3=371
S=663 -> H1=176 H2=497 H3=689
S=340 -> H1=884 H2=139 H3=155
S=872 -> H1=430 H2=041 H3=076
S=711 -> H1=360 H2=758 H3=705
S=367 -> H1=162 H2=811 H3=411
S=882 -> H1=855 H2=298 H3=671
S=162 -> H1=434 H2=944 H3=769
S=767 -> H1=300 H2=392 H3=493
S=429 -> H1=446 H2=836 H3=099
S=622 -> H1=584 H2=228 H3=197
S=969 -> H1=554 H2=510 H3=520
S=893 -> H1=366 H2=895 H3=759
S=311 -> H1=554 H2=851 H3=330
S=529 -> H1=125 H2=475 H3=419
S=721 -> H1=070 H2=396 H3=487
S=384 -> H1=501 H2=393 H3=726
S=798 -> H1=730 H2=466 H3=485
S=615 -> H1=824 H2=988 H3=486

S=386 -> H1=878 H2=876 H3=477
S=215 -> H1=540 H2=226 H3=694
S=935 -> H1=777 H2=905 H3=245
S=292 -> H1=852 H2=693 H3=696
S=821 -> H1=393 H2=287 H3=556
S=527 -> H1=319 H2=266 H3=109
S=359 -> H1=280 H2=544 H3=855
S=626 -> H1=808 H2=371 H3=060
S=226 -> H1=429 H2=546 H3=495
S=236 -> H1=372 H2=104 H3=780
S=468 -> H1=481 H2=461 H3=560
S=529 -> H1=125 H2=475 H3=419
S=630 -> H1=596 H2=481 H3=101
S=923 -> H1=898 H2=785 H3=687
S=335 -> H1=204 H2=685 H3=523
S=802 -> H1=636 H2=183 H3=963
S=958 -> H1=230 H2=942 H3=389
S=967 -> H1=847 H2=207 H3=352
S=656 -> H1=468 H2=611 H3=503
S=242 -> H1=799 H2=586 H3=588
S=973 -> H1=215 H2=540 H3=045
S=219 -> H1=343 H2=977 H3=413
S=437 -> H1=194 H2=187 H3=319
S=624 -> H1=294 H2=040 H3=985
S=670 -> H1=985 H2=307 H3=605
```

Algoritmos y Estructuras de Datos.
TPLR. Recuperatorio de Trabajos Prácticos de Laboratorio. [2019-11-07]

PASSWD PARA EL ZIP: `4GW3 P267 4QCM`

Ejercicios

[Ej. 1] **[media-movil-entera] (TPL1)** Implemente la función `list<int> mediaMovilEntera(list<int>& L, int v)` que dada una lista de enteros `L`, retorne una lista con los valores de la media móvil con ventana fija de tamaño `v`.
El primer elemento de la lista resultado será el promedio (en división entera) de los primeros `v` elementos de `L`, el segundo será el promedio desde el 2do elemento al `v`+1. En general, el elemento en la posición `j` de la lista resultado será el promedio entre los elementos `[j,j+v)` de `L`
Ejemplos:

- Para `L=[1,2,6,5,8,3,4,6]` y `v=2` debe retornar `[1,4,5,6,5,3,5]`
- Para `L=[1,2,6,5,8,3,4,6]` y `v=3` debe retornar `[3,4,6,5,5,4]`
- Para `L=[1,2,6,5,8,3,4,6]` y `w=4` debe retornar `[3,5,5,5,5]`

[Ej. 2] **[nSumaK] (TPL3)** Implementar la función `bool nSumaK(set<int> &S, int k)` que dado un conjunto `S` y un valor `k`, retorne el número de subconjuntos de `S` para los cuales la suma de sus elementos sea `k`.
Ejemplos:

- Para `S={-5,2,7}`, `k=4` debe retornar `1` (subconjunto `{-5,2,4}`)
- Para `S={-5,2,7}`, `k=2` debe retornar `2` (subconjuntos `{-5,7}, {2}`)
- Para `S={-5,2,7}`, `k=1` debe retornar `0`

Ayuda:

- Escribir primero un algoritmo recursivo para generar todos los subconjuntos de `S` posibles.
- Modificarlo para que cada vez que se genera un subconjunto, se verifique su suma, y se acumule uno si coincide con `k`.

[Ej. 3] **[nCaminosK] (TPL2,TPL3)** Dado un grafo simple `map<int,set<int>> G` y dos vértices `a` y `b`, implementar una función `int nCaminosK(graph& G, int a, int b, int k)` que retorne el número de caminos de longitud `k` entre los vértices `a` y `b`. El camino puede repetir nodos.
Ayuda. Utilizar una estrategia recursiva:

- Si la longitud `k=0`, se habrá encontrado un camino si `a==b`.
- Si no, contar la cantidad de caminos de longitud `k-1` entre alguno de los vecinos de `a` y `b`.

[Ej. 4] **[make-heap] (TPL3)** Escribir una función `void make_heap(btree<int> &T);` que convierte *in-place* el árbol binario en parcialmente ordenado, aplicando el algoritmo *make-heap*. (**Nota:** esto lo vemos en el curso para montículos representados como vectores, aquí pedimos hacerlo en un árbol binario, que puede no ser parcialmente completo). Por ejemplo,

```
(5 (3 2 1) (2 1 0))       => (0 (1 2 3) (1 5 2))
(4 (9 0 .) (4 8 .))       => (0 (4 9 .) (4 8 .))
(5 (5 . 5) (4 9 (0 . 7))) => (0 (5 . 5) (4 9 (5 . 7)))
```

El algoritmo se basa en escribir una función recursiva **`make_heap(T,n)`** de acuerdo a las siguientes instrucciones,

- Si el nodo es Λ entonces no hace nada.
- Caso contrario aplicar recursivamente **`make_heap`** a los hijos.
- Aplicar **`re_heap`** en el nodo, intercambiando el nodo con su hijo menor siempre que el hijo sea mayor que **`n`**. Repetir esto hasta llegar a una hoja o a un nodo tal que ambos hijos con mayores o iguales que el nodo.

[Ej. 5] **[sort-stack](TPL1)** Escribir un programa que ordene una pila **`S`** utilizando recursión de forma tal que los elementos de mayor valor se encuentren en el tope. No está permitido el uso de constructores iterativos (**`while`**, **`for`**, etc) ni tampoco el uso estructuras de datos adicionales. Sólo pueden utilizarse métodos de la pila, a saber:

- **`S.empty()`**
- **`S.top()`**
- **`S.push(s)`**
- **`S.pop()`**

Se propone el siguiente algoritmo recursivo:

- Si la pila está vacía finalizar
- Si no:
  - Guardar una copia del elemento al tope.
  - Quitarlo de la pila.
  - Ordenar el resto de la pila.
  - Insertar el elemento guardado al tope de la pila ordenada.

Para el último paso se requiere un método auxiliar, también recursivo, que reciba un elemento y una pila e inserte dicho elemento en forma ordenada, siguiendo el siguiente algoritmo:

- Si la pila está vacía o si el elemento a insertar es mayor al tope, insertar el elemento.
- Si no:
  - Almacenar una copia temporal del elemento al tope.
  - Quitarlo de la pila.
  - Insertar el elemento recibido en forma ordenada recursivamente.
  - Apilar el elemento temporal en la pila resultado.

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.
- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.

- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **f** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  j es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**eval<1>** y **evalr<1>**). La primera **ev.eval<j>(f,vrbs);** toma una serie de casos de prueba de entrada, le aplica la función del usuario **f** y compara la salida del usuario (**user**) con respecto a la esperada (**ref**). Si la verbosidad (el argumento **vrbs**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **ucase**: Además las funciones **eval()** tienen dos parámetros adicionales:
  Eval::eval(func_t func,int vrbs,int ucase);
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **ucase=-1** que quiere "hacer todos". Por ejemplo **ev.eval4(prune_to_level,1,51);** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **eval<j>** están almacenados en un archivo **test1.json** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **datain** son los datos pasados a la función y **output** la salida producida por la función de usuario. **ucase** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **evalr<j>** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **seed**, se los pasa a la función del usuario **f()**. Las respuestas de la **f()** van siendo procesadas por la función interna de hash que genera un **checksum H** de las respuestas. Por ejemplo para el primer ejercicio si **seed=123** entonces el checksum es **H=523**. Una vez que el alumno termina su tarea se le pedirá que corra la función **evalr<j>()** de la clase evaluadora con un valor determinado de la semilla **seed** y se comprobará que genere el valor correcto del checksum **H**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  void Eval::dump(list <int> &L,string s=""): Imprime una lista de enteros por **stdout**. Nota: Es un método de la clase **Eval** es decir que hay que hacer **Eval::dump(VX);**. El string **s** es un label opcional.
  - **void Eval::dump(list <int> &L,string s="")**
- Después del parcial deben entregar el programa fuente (sólo el **program.cpp**) renombrado con su apellido y nombre (por ejemplo **messilionel.cpp**). Primero el apellido.

mstorti@galileo/tpl2-2019-61-g71f85448/Sat Nov 9 01:20:51 2019 -0300

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

FICH

UNL

Algoritmos y Estructuras de Datos.
Parcial 1. [2022-10-06]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.
2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...
3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS1 (W=20pt)]

a) **[list]**.
- Indique un prototipo (definición de atributos y métodos públicos y privados) de la clase lista implementada como celdas simplemente enlazadas.
- Implemente los métodos **`begin`** y **`end`** (deben ser $O(1)$)
- Implemente el metodo **`erase(iterator_t p,iterator_t q)`** que elimina el rango entre los elementos indicados por **`p`** y **`q`** respectivamente.

b) **[stack]**.
- Indique un prototipo de una implementación del TAD Pila
- Implemente los métodos
  - **`T max(stack::<T> &S);`** que devuelva el máximo valor almacenado en la pila
  - **`void erase(stack::<T> &S, T t)`**, que elimina de la pila los objetos **`t`**.

c) **[AOO]**.
- Indique un prototipo de implementación de un árbol binario.
- Implemente un iterador adecuado para dicho prototipo de TAD.

[Ej. 2] [OPER1 (W=20pt)]

a) **[rec-arbol]**: Dibujar el **`AOO`** cuyos nodos, listados en orden previo y posterior son

```
1       ORD-PRE = (Z W A B C D X Y T)
2       ORD-POST = (W B A X T Y D C Z)
```

luego particionar el conjunto de nodos respecto al nodo **`A`**.

b) **[operaciones]**: Sea el árbol **`D=(6 (5 4 3) (2 7 9 8) (1 0))`**. Después de hacer:

```
1 auto n = D.find(5);
2 n++;
3 n = n.lchild();
4 n++;
5 n = n.lchild();
6 n = D.insert(n,2);
```

¿Cómo queda el árbol? ¿Se produce un error?

c) **[Notación $O(\cdot)$]**. cada una de las funciones $T_1, \ldots, T_4$ determinar su velocidad de crecimiento (expresarlo con la notación $O(\cdot)$) y ordenarlas de forma creciente.

$$T_1 = 2\,n^2 + 2\log_{10} n + 4\,n^4 + \sqrt[2]{n}$$
$$T_2 = 4\,n^3 + 3\,n! + 2 \cdot 3^n$$
$$T_3 = 2n^4 + 3 \cdot 3^n + 3^4$$
$$T_4 = \log_2 n + 4$$
$$T_5 = \log(8) n^{1.5} + 3.2 \log n + 4^4$$

d) **[hacer-arbol (5pt)]** Utilizando sólo métodos **`insert, lchild, n++`**, e iteradores del Arbol Ordenado Orientado AOO, complete el siguiente código que arma el árbol **`T=(3 (5 6 8) 4)`**

```
1      tree<int> T;
2      tree<int>::iterator n = T.insert(T.begin(),3);
3      ...
4      COMPLETAR
5      ...
```

[Ej. 3] [PREG1 (W=20pt)]

a) Se puede insertar un elemento en una posición **dereferenciable** en una lista? Y en una **no-dereferenciable**?
b) Enuncie y justifique la **regla de transitividad** para la notación $O(\cdot)$. De un ejemplo.
c) En una **correspondencia**: ¿puede haber una **misma clave** con **diferentes valores**? Por ej. **`M=(5->4, 5->2)`** ¿Puede haber **diferentes claves** con el **mismo valor**? Por ej. **`M=(3->5, 4->5)`**
d) En un árbol, ¿puede un nodo tener **más de un padre**? ¿Puede un nodo **no tener padre**?
e) Sea el árbol **`(z r (p (q n) (t m)))`**.
   - Cuáles de los siguientes son **caminos**?
     - **`(p t)`**
     - **`(n q p)`**
     - **`(t p q)`**
     - **`(z p q)`**
   - Liste las **hojas** del árbol y los nodos a **profundidad** 2.
   - ¿Cuál es la **altura** del árbol?
f) Defina que entiende por un **algoritmo de búsqueda exhaustiva** y un **algoritmo ávido**. Explique como se aplicarían estas estrategias a un problema como el de coloración de grafos o el del agente viajero (TSP).
g) Si se implementa una **pila** en términos de **listas simplemente enlazadas**: ¿Dónde ubicaría el **tope** de la pila? ¿En el comienzo o en el fin de la lista? ¿Porqué?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 2. [2022-11-17]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.

2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...

3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS2, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS2 (W=20pt)]

a) **[set-abb]** A partir del siguiente bloque de código:

```
1  set<int> S;
2  S.insert(3);
3  S.insert(-1);
4  S.insert(2);
5  S.insert(6);
6  S.insert(4);
7  S.insert(7);
8  auto p = S.find(3);
```

Presente mediante un croquis las estructuras de datos presente en memoria luego de su ejecución. Considere la implementación de conjuntos por medio de árboles binarios de búsqueda.

b) **[ab]** Dado el siguiente resumen de la implementacicen de árbol binario

```
1  class btree{
2    class cell {
3      elem_t t;
4      cell *right,*left;
5      cell() : right(NULL), left(NULL) {}
6    };
7    class iterator {
8      private:
9      cell *ptr,*father;
10     side_t side;
11     iterator(cell *p,side_t side_a,cell *f_a)
12     : ptr(p), side(side_a), father(f_a) { }
13     public:
14     iterator(const iterator_t &q) { ... }
15     bool operator==(iterator_t q) { ... }
16     bool operator!=(iterator_t q) { ... }
17     iterator() : ptr(NULL), side(NONE),
18     father(NULL) { }
19     iterator left() { ... }
20     iterator right() { ... } //-->implementar
21   }
22   ...
23   iterator splice(iterator to, iterator from); //-->implementar
24   iterator insert(iterator it, elem_t e); //-->implementar
25   ...
26 }
```

implementar los métodos `btree::splice`, `iterator::right` y `btree::insert`.

[Ej. 2] [OPER2 (W=20pt)]

a) **[huffman]**: Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario utilizando el algoritmo de Huffman y encodar la palabra **SCALONETA**, $P(S) = 0.07, P(C) = 0.06, P(A) = 0.07, P(L) = 0.20, P(O) = 0.10, P(N) = 0.05, P(E) = 0.05, P(T) = 0.1, P(Y) = 0.07, P(Z) = 0.23$. Calcular la longitud promedio del código obtenido.

b) **[hf-decode]** Utilizando la codificación correspondiente al árbol binario representado por el siguiente código Lisp
(. (. (. G ␣) (. (. (. B R) (. T H)) E)) (. O I))";
desencodar el mensaje:
0100011000001010000100110010100101101101001.

c) **[abb]** Dados los enteros $\{8, 14, 1, 19, 18, 11, 16, 17, 18, 9, 20\}$ insertarlos, en ese orden, en un **árbol binario de búsqueda (ABB)**. Mostrar las operaciones necesarias para eliminar los elementos 8, 14 y 19 en ese orden.

d) **[quick-sort]** Dados los enteros $\{4, 8, 3, 0, 4, 9, 7, 2, 2, 1, 10, 5\}$ ordenarlos por el método de *clasificación rápida (quick-sort)*. En cada iteración indicar el pivote y mostrar el resultado de la partición. Utilizar la estrategia de elección del pivote discutida en el curso, a saber el mayor de los dos primeros elementos distintos. **Nota:** La partición se puede realizar con cualquier estrategia.

e) **[hash]:** Insertar los enteros **{3, 22, 28, 11, 10, 38, 20, 7}** en una tabla de dispersión cerrada con **B=10** cubetas, con función de dispersión **h(x) = x % B**. Luego eliminar el elemento **10** e insertar el elemento **27**, en ese orden. Mostrar la tabla resultante.

[Ej. 3] [PREG2 (W=20pt)]

a) Realice los pasos necesarios (sentencias de C/C++) para construir el siguiente árbol binario **T=(1 (3 . (2 10 20)) (5 7 .))**
b) Escriba la *"definición recursiva"* de la función **ALTURA** de un Ábrol Ordenado Orientado (AOO).
c) ¿Cuál es la altura de los nodos **A**, **B**, y **C** en el siguiente AOO: **(C (B (T X Y)) (Z A (D P)))**?
d) Explique porqué el método de Huffman produce una codificación que cumple con la *"condición de prefijos"*.
e) Cual es el **orden del tiempo de ejecución** (en promedio) en función del número de elementos (n) que posee el **conjunto por árbol binario de búsqueda (ABB)** de las siguientes operaciones/funciones/métodos. Asuma que el árbol se mantiene bien balanceado.

1) **insert(x)**
2) **erase(p)**
3) **erase(x)**
4) ***p**
5) **set_union(A,B,C)**

f) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un árbol ordenado orientado (AOO)? ¿Y en una posición **dereferenciable**?
g) ¿Es posible **insertar** en una posición **no-dereferenciable** (Λ) en un árbol binario (AB)? ¿Y en una posición **dereferenciable**?

Apellido y Nombre: _____

Carrera: _____ DNI: _____

[Llenar con letra mayúscula de imprenta GRANDE]

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática
Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos.
Parcial 1. [2023-10-05]

1. **[ATENCIÓN 1]** Para aprobar deben obtener un **puntaje mínimo** del 60 % en las preguntas de teoría y 50 % en clases y operativos.

2. **[ATENCIÓN 2]** Escribir cada ejercicio en **hoja(s) separada(s)**. Es decir todo `CLAS2` en una o más hojas **separadas**, `OPER2` en una o más hojas **separadas**, `PREG2` en una más hojas **separadas**, etc...

3. **[ATENCIÓN 3]** Encabezar las hojas con **sección, Nro de hoja (relativo a la sección), apellido, y nombre, ASI**: `CLAS1, Hoja #2/3 TORVALDS, LINUS`

[Ej. 1] [CLAS1 (W=20pt)]

a) **[list]**.

1) Indique las diferencias entre los prototipos (definición de atributos y métodos públicos y privados) de la clase lista implementada como celdas simplemente y doblemente enlazadas.
2) Implemente el método **`list<T>::iterator erase(list<T>::iterator p)`**, elimina el elemento indicado por **p**para ambos prototipos.
3) Considere la función function indicada:
4) Explique la función de dicho algoritmo.
5) Indique su complejidad en notación $O(\cdot)$. Justifique.
6) Implemente una versión de dicho algoritmo que sea más eficiente. Justifique.

```
void function (list<T> L, list<T>::iterator a, list<T>::iterator b)
{
  while( a != b && a != L.end())
    {
      a = L.erase(a);
    }
}
```

b) **[stack]**.

1) Explique brevemente los siguientes métodos del TAD pila.

```
  elem_t top();
  void pop();
  void push(elem_t x);
```

2) Implemente los métodos
   i) **`void clear():`** remueve todos los elementos de la pila.
   ii) **`int size():`** devuelve el número de elementos en la pila.
   iii) **`bool empty():`** retorna verdadero si la pila está vacía, falso en caso contrario.
   iv) **`T min():`** que devuelva el min valor almacenado en la pila
   v) **`void bound(T t):`**, que asigna **`t`** a los elementos de la pila cuyo valor sea mayor a **`t`**.

[Ej. 2] [OPER1 (W=20pt)]

a) **[Notación $O(\cdot)$]**. cada una de las funciones $T_1, \ldots, T_4$ determinar su velocidad de crecimiento

(expresarlo con la notación $O(\cdot)$) y ordenarlas de forma creciente.

$$T_1 = 4\,n! + 3\,n^3 + 2 \cdot 2^n$$
$$T_2 = \log_{10} n + 4$$
$$T_3 = 3n^3 + 4^2 + 3 \cdot 4^n$$
$$T_4 = 4^4 + 3.4 \log n + \log(27) n^{3.5}$$
$$T_5 = 4\,n^4 + 3 \log_2 n + 2\,n^3 + \sqrt[3]{n}$$

b) **[operaciones]**. Sea el árbol `D=(5 (4 3 2) (1 9 8 7) (6 5))`. Después de hacer:

```
1  auto n = D.find(1);
2  n++;
3  n = n.lchild();
4  n = n.lchild();
5  n = D.insert(n,4);
```

Dibuje como queda el arbol. En caso que se produzca un error, por favor detalle el mismo.

c) **[rec-arbol]**. Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

```
1       ORD-PRE = (A B F G H C D E J),
2       ORD-POST = (G H F B C J E D A)
```

Luego particionar el conjunto de nodos respecto al nodo `B`.

d) **[hacer-arbol]** Utilizando sólo métodos `begin, insert, lchild, n++`, e iteradores del Arbol Ordenado Orientado AOO, complete el siguiente código que arma el árbol `T=(2 3 (4 5 (6 8)))`

```
1      tree<int> T;
2      tree<int>::iterator n = T.insert(T.begin(),2);
3      ...
4      COMPLETAR
5      ...
```

[Ej. 3] [PREG1 (W=20pt)]

a) ¿Qué ocurre si se invoca el `operator[]` sobre una correspondencia con una clave que no tiene asignación? Por ejemplo `M={(5,7),(8,9)}` y hacemos `x = M[4]`. ¿Da un error? ¿Qué valor toma `x`? ¿Cómo queda la correspondencia?

b) Discuta las ventajas y desventajas de utilizar **listas doblemente enlazadas** con respecto a las simplemente enlazadas. ¿En cuál caso es necesario tener posiciones adelantadas? ¿Cuál es el *overhead* (bytes adicionales que no representan datos) en cada caso? ¿Cual es la eficiencia al recorrer la lista en uno y otro sentido?

c) ¿Que quieren decir los términos **FIFO** y **LIFO** en contenedores? Menciones los contenedores que son de uno y otro tipo.

d) En algunos contenedores(por ejemplo `list` y `tree`) el operador `*` de `iterator` retorna una **referencia** al dato. ¿Cuál es la ventaja de esto, con respecto a retornarlo por **copia**?

e) Defina cuándo un nodo n es **antecesor** de un nodo m en un AOO. Ídem para **descendiente**, **derecha** e **izquierda**.

f) ¿Es válido insertar elementos en una posición **dereferenciable** en un árbol ordenado orientado (AOO)? ¿Y en una posición **no dereferenciable**? ¿Y para una lista?

g) Explique la propiedad de **transitividad** de la notación asintótica $O(\cdot)$. De un ejemplo. Ídem para la regla de la suma.