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`).

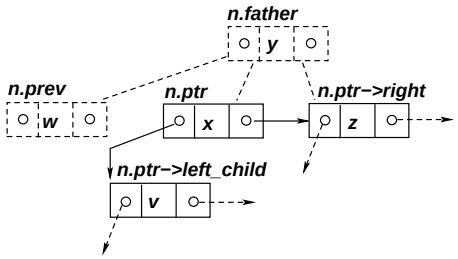

Figura 1: Entorno local de un iterator para árboles ordenados orientados.

[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)))`