

## 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

```

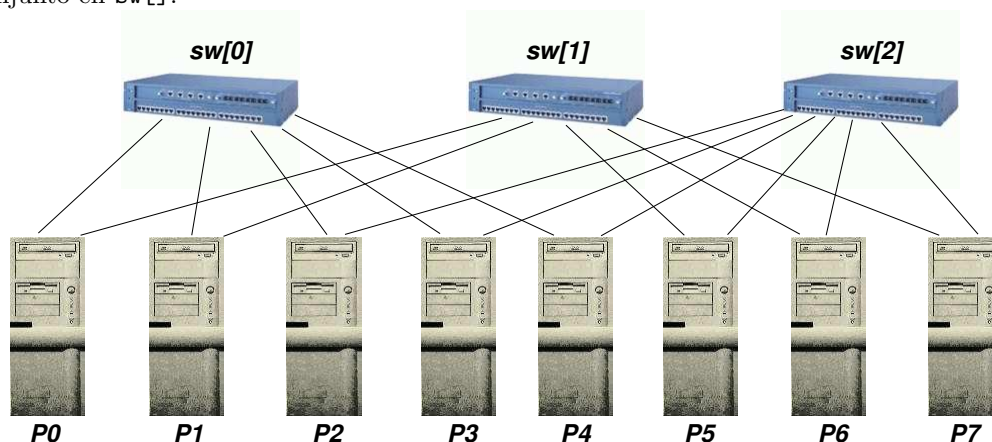
---

### [Ej. 2] [Programación (total = 50 puntos)]

## a) [flat (30 puntos)]

Se está diseñando una red interconectada 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 interconectados 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 conjuntos en  $sw[]$ .



En el ejemplo de la figura tenemos 8 nodos conectados via 3 switches y puede verificarse que cualquier par de nodos está conectado en forma directa a través de al menos un switch. Para este ejemplo el vector  $sw$  sería

$$sw[0] = \{0, 1, 2, 3, 4\}, \quad sw[1] = \{0, 1, 5, 6, 7\}, \quad sw[2] = \{2, 3, 4, 5, 6, 7\} \quad (1)$$

Por lo tanto `flat(sw,8)` debe retornar `true`. Por otra parte si tenemos

$$sw[0] = \{0, 2, 3, 4\}, \quad sw[1] = \{0, 1, 5, 7\}, \quad sw[2] = \{2, 3, 5, 6, 7\} \quad (2)$$

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.

Apellido y Nombre: \_\_\_\_\_

Carrera: \_\_\_\_\_ DNI: \_\_\_\_\_

[Llenar con letra mayúscula de imprenta GRANDE]

- 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.