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)]

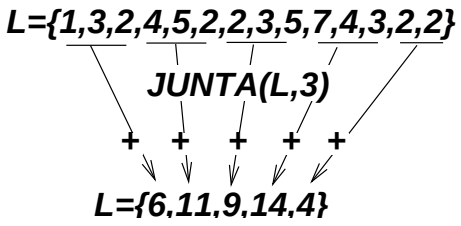
Escibir las primitivas que se indican del TAD `set` por ABB.

```
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` $=\{D, Q, R, A, B, S, E, C\}$.

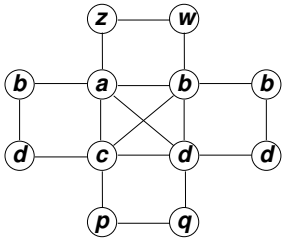
b) **[huffman (5 ptos)]** Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `PIQUETERO`
$P(P) = 0{,}2, P(I) = 0{,}2, P(Q) = 0{,}2, P(U) = 0{,}1, P(E) = 0{,}1, P(T) = 0{,}1, P(R) = 0{,}05, P(O) = 0{,}05$
Calcular la longitud promedio 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.