

## 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 %)].

- [stack (6pt)]** Escribir la implementación del TAD pila (clase stack) usando listas.
- [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()`.
- [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 %)].

- [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.
- [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,  $C[key] \leftarrow A[key] \cap B[key]$ ).

Por ejemplo, dados:

`A = { 'XX' : [3,3,1,2,2,7], 'YY' : [7,1,5,5,4,1] }`      `B = { 'YY' : [3,3,4,5,8,1], '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. Utilice esta función como ayuda para contruir la lista de elementos comunes y sin repetición  $C[key] \leftarrow A[key] \cap B[key]$

- [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 %)].

- [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`).
- [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.
- [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;  
}
```