

## 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 implementela.

[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 ( $\Lambda$ ). 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)