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. [2011-10-27]

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 ("AVANZADA").**

a) **[AOO (10pt)]** Para el TAD Arbol Ordenado Orientado (AOO):
- Declarar las clases `tree`, `cell`, `iterator`, con su correspondiente anidamiento, incluyendo las declaraciones de datos miembros.
- Implementar los métodos
  - `tree<T>::iterator tree<T>::insert(tree<T>::iterator n, const T& x);`
  - `tree<T>::iterator tree<T>::begin();`

b) **[AB (10pt)]** Para el TAD Arbol Binario (AB):
- Declarar las clases `btree`, `cell`, `iterator`, con su correspondiente anidamiento, incluyendo las declaraciones de datos miembros.
- Implementar los métodos
  - `btree<T>::iterator btree<T>::erase(btree<T>::iterator n);`,
  - `btree<T>::iterator btree<T>::iterator::right();`.

[Ej. 2] **[Programación (total = 40pt)] Insistimos: deben usar la interfaz STL.**
Atención: Hay 3 ejercicios cuya suma es **50**, pero el total de la sección es **40**. (O sea, la nota final en esta sección es `min(40,n1+n2+n3)`).

a) **[make-mirror (20pt)]** Escribir una función `void make_mirror(tree<int> &T);` que convierte **in-place** al árbol `T` en su espejo. Por ejemplo si `T=(1 (5 3 2 1) (10 9 7))` entonces después de hacer `make_mirror(T)` debe quedar `T=(1 (10 7 9) (5 1 2 3))`. **Ayuda:** Debe ir invirtiendo la lista de hijos para cada nodo `n` y aplicar la función recursivamente sobre los hijos. Para invertir los hijos se puede usar alguna de las siguientes opciones:
- Usar un árbol auxiliar `Taux`, insertar un elemento cualquiera (p.ej. 0) en la raíz y hacer *splice* para ir moviendo cada uno de de los hijos de `n` a `Taux`. Luego volver a mover (con *splice*) todos los hijos de `Taux` en `n`, pero de forma que queden invertidos.
- Usar una lista de árboles `list<tree<int>> L`. Mover los hijos de `n` sucesivamente en nuevos árboles en la lista. Para insertar un nuevo árbol en la lista simplemente hacemos `L.push_back(tree<int>())`.
- No usar ningún contenedor auxiliar, ir recorriendo los hijos de `n` y moverlos en `n.lchild()`. En este caso tener cuidado con los iteradores.

Curiosidad: Notar que `ordprev(make_mirror(T))=reverse(ordpost(T))` y `ordpost(make_mirror(T))=reverse(ordprev(T))`.

b) **[is-balanced (20pt)]** Un árbol binario (AB) es balanceado si
- Es el árbol vacío ó,
- Sus subárboles derecho e izquierdo son balanceados, y sus alturas difieren a lo sumo en 1, o sea $|h_L - h_R| \leq 1$.

Por ejemplo `(1 (2 (3 (4 5 6) 7) (13 8 9)) (15 10 (16 11 12)))` es un árbol balanceado (notar que no necesariamente las profundidades de las hojas difieren en ± 1).
Consigna: Escribir una función `bool is_balanced(btree<int> &T);` que retorna `true` si el árbol está balanceado.
Ayuda: En la función auxiliar retornar el resultado de si el árbol es balanceado o no, pero además retornar (con un valor pasado por referencia, o con un `pair`) la altura del árbol correspondiente.

c) [**decomp-int (10pt)**] A partir de un número entero `m` escribir una función `void decomp_int(int m, btree<int> &T);` que construye el árbol binario `T` de la siguiente forma:

- En la raíz contiene `m`
- En los hijos izquierdo y derecho contiene los valores `ml` y `mr` computados como `int mr=m/2, ml=m-mr`. Si `mr` ó `ml` son nulos el nodo no es insertado.
- Propaga recursivamente la decomposición a los nodos.

Por ejemplo si `m=5` entonces el árbol generado es `(5 (3 (2 1 1) 1) (2 1 1))`.

[Ej. 3] [operativos (total 20pt)]

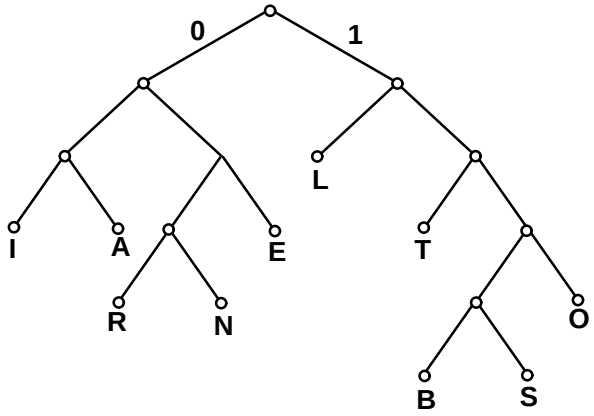
a) [**rec-arbol (8pt)**] Dibujar el AOO cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Z, D, F, G, X, Y, W, Q, \}$,
- `ORD_POST` $=\{Z, F, X, Y, Q, W, G, D, A, \}$.

b) [**huffman (6pt)**] Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario utilizando el algoritmo de Hufmann y encodar la palabra `LIOMESSI`
$P(O) = 0.10, P(M) = 0.10, P(S) = 0.10, P(E) = 0.30, P(I) = 0.05, P(L) = 0.35$. Calcular la longitud promedio del código obtenido.

c) [**hf-decode (6pt)**]

Utilizando el código de la derecha desencodar el mensaje
`10001111000110100000101110111111101`



[Ej. 4] [Preguntas (total = 20pt, 4pt por pregunta)]

a) Realice los pasos necesarios (sentencias de C/C++) para construir el siguiente AOO
`T=(5 (7 9 8 (2 4 3)) (3 4))`

b) ¿Como se define (en forma abstracta, no pedimos el código) la notación Lisp para árboles?

c) ¿Se puede hacer insert en iteradores no dereferenciables en un árbol ordenado orientado (AOO)? ¿Y en los dereferenciables? ¿Y para árbol binario (AB)?

d) Dado el árbol `T=(1 4 (5 3 2 1) (10 9 7))` listar todos los nodos `L` que están a la izquierda, a la derecha `R`, antecesores propios `A`, y descendientes propios `D`, del nodo 5.

e) ¿Cómo se define el iterator para AB? Describa los miembros de la clase y que significan con un ejemplo.