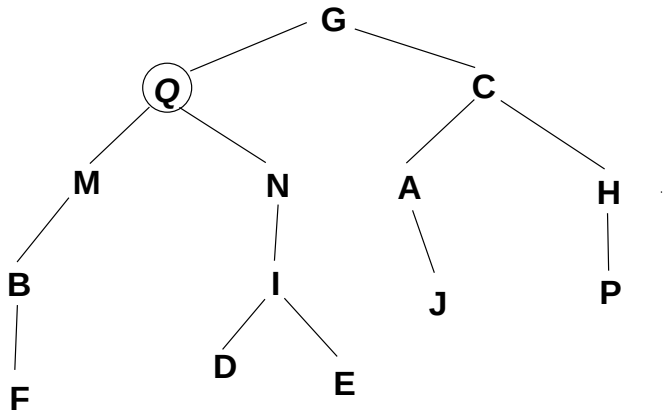
Guía de Trabajos Prácticos número 3
Árboles

[1] [Teoría y Operativos]

a) Dado el nodo `Q` del árbol de la figura, decir cuales son los nodos:
`descendientes_propios(Q)`, `antecesores_propios(Q)`, `izquierda(Q)` y `derecha(Q)`.



b) Sea el árbol `(a b (c d e))`. ¿ Cuál de las opciones es verdadera?

- ☐ Tiene dos raíces y altura 2.
- ☐ Tiene 3 hojas y altura 3.
- ☐ Tiene 3 hojas y altura 2.
- ☐ Tiene 4 nodos a profundidad 1 y 3 hojas.

c) ¿Es verdad que si dos nodos están en el **mismo nivel** de un árbol, entonces son **hermanos**? ¿Y la recíproca? De ejemplos.

d) Dado el árbol `(c q (t (r u (v z))))`,

1) ¿Cuál es el nodo que está a la vez a la izquierda de `v` y no es descendiente de `r`?
2) Particione el árbol con respecto al nodo `q`, es decir indique cuales son sus antecesores y descendientes propios, derecha e izquierda.

e) Dado el árbol binario `(z (a b q) r)`, ¿Es completo? ¿Es lleno?. Justifique

f) Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{C, Z, Q, R, A, M, P, K, L, T\}$,
- `ORD_POST` $=\{Z, A, P, M, K, L, R, T, Q, C\}$.

g) Dibujar el árbol ordenado orientado cuyos nodos, listados en orden previo y posterior son

- `ORD_PRE` $=\{A, Z, W, X, Y, T, R, Q, V, L\}$.
- `ORD_POST` $=\{Z, W, X, Y, R, Q, V, L, T, A\}$.

h) Sea el AOO `A=(5 7 (9 3 4) 6)`. Después de hacer las operaciones

```
p = A.find(9);
p = p.lchild(); p++;
p = A.insert(p,10);
```

¿Cuál de las siguientes opciones es verdadera?

- ☐ Da un error al insertar.
- ☐ `A=(5 7 (9 3 4 10) 6)`, `*p=10`.
- ☐ `A=(5 7 (9 3  10 4) 6)`, `*p=10`.

☐ `A=(5 7 (9 3  10 4) 6)`, `*p=4`.

i) Sea el árbol `(5 7 (8 6 9))`. Después de hacer:

```
n = D.find(7);
n++;
n = n.lchild();
n = n.lchild();
n = D.insert(n,2);
```

¿Cómo queda el árbol? ¿Se produce un error?

j) Defina que es un **camino** en un árbol. Dado el AOO `(a b (c e (f g)) d))`, ¿cuáles de los siguientes son caminos?

1) `(a,c,f,g)`,
2) `(e,c,f,g)`,
3) `(e,c,a)`,
4) `(c,f,g)`.

k) ¿Es posible **insertar** en una posición `no-dereferenciable` (Λ) en un AB? ¿Y en un AOO? Discuta y dé ejemplos.

l) Explique porqué el método de Huffman cumple con la *"condición de prefijos"*.

m) Utilizando el código del siguiente árbol binario `(* (* A (* O (* P .))) (* U T))` a derecha desencodar el mensaje `010101101101011`

n) Dados los caracteres siguientes con sus correspondientes probabilidades, construir el código binario y encodar la palabra `PAPAFRITA`
$P(P) = 0{,}05, P(A) = 0{,}2, P(F) = 0{,}1, P(R) = 0{,}05, P(I) = 0{,}2, P(T) = 0{,}15, P(Q) = 0{,}15, P(S) = 0{,}1$
Calcular la longitud promedio del código obtenido.

ñ) Dados los caracteres siguientes con sus correspondientes probabilidades, contruir el código binario y encodar la palabra `TSUNAMI`
$P(T) = 0{,}2, P(S) = 0{,}2, P(U) = 0{,}2, P(N) = 0{,}1, P(A) = 0{,}1, P(M) = 0{,}1, P(I) = 0{,}05, P(Z) = 0{,}05$
Calcular la longitud promedio del código obtenido.

[2] [Programación]

a) **[Árboles Ordenados Orientados]**

1) *ListarAOO*. Escribir procedimientos para listar un árbol ordenado orientado siguiendo los órdenes prefijo y postfijo.
2) *VariosAOO*. Escribir procedimientos para:
   - a' Calcular la altura de un AOO.
   - b' Contar el número de hojas.
   - c' Obtener la máxima etiqueta de todo el árbol.
   - d' Obtener la máxima etiqueta par del árbol.
   - e' Obtener la máxima etiqueta de sólo las hojas.
   - f' Obtener la suma de todas las etiquetas.
   - g' Eliminar todas las etiquetas impares y sus subárboles.
3) *Level* Escribir una función `void count_level(tree<int> &T, int l)`, que cuente cuantos nodos hay en el nivel `l` del árbol `T`.
4) *levelList* El listado en órden de nivel de los nodos de un árbol lista primero la raíz, luego todos los nodos de profundidad 1, después todos los de profundidad 2, y así sucesivamente. Los nodos que estén en la misma profundidad se listan en órden de izquierda a derecha. Escribir una función `void orden_de_nivel (tree <int> &T)` para listar los nodos de un árbol en orden de nivel.

5) *ContarSi*. Escribir una función `int count_if(tree<int> &T,bool (*pred)(int x))` que retorna el número de nodos del árbol `T` que satisfacen el predicado `pred`. Por ejemplo, si `T=(1 2 (3 5 7 6) 4)`, entonces `count_if(T,odd)` debe retornar `4`. Escribir el predicado `bool odd(int x)` que determina si un entero es impar.
6) *ListarSi*. Escribir una función `void list_if(tree<int> &T,list<int> &L, bool (*pred)(int x))` que retorna en `L` la lista de valores nodales en orden previo de un árbol ordenado orientado `T` que satisfacen el predicado `pred`. Por ejemplo, si `T=(1 (-2 7 (8 -7) (3 -5 -6)))`, entonces después de `list_if(T,L,positive)`, debe quedar `L={1,7,8,3}`. Escribir el predicado `bool positive(int x)` que determina si un entero es mayor que `0`.
7) *list2tree* Escribir una función `void list2tree(tree<int> &T,list<int> &L)` que dada una lista `L` con el orden previo de `T` y el tamaño de sus subárboles reconstruye `T`. La forma de almacenar el árbol en `T` es la siguiente: se almacena en orden previo 2 valores enteros por cada nodo, a saber el contenido del nodo y el número de hijos. Por ejemplo para el árbol `(6 4 8 (5 4 4) 7 9)` tenemos `L=(6 5 4 0 8 0 5 2 4 0 4 0 7 0 9 0)`. Escribir también la función inversa `void tree2list(tree<int> &T,list<int> &L)`.
8) *deExpansión* Dado un grafo `G`, y un árbol `T`, decimos que `T` *expande* a `G` si la raiz `n` de `T` es un vértice de `G`, y los caminos de `T` permiten llegar desde `n` hasta cualquier otro nodo de `G`. Escribir una función `bool es_arbol_de_expansion(graph G, tree<int> T)` que determina si `T` expande a `G`.
9) *creaAOO* Escribir una función `tree<double> crea(double M, int n)` que dados un real M y un entero n, tales que $n < M$ crea un AOO `T` tal que:
   a' La suma de las hojas es M, pero cada una de ellas es $h \leq n$.
   b' Se satisface que para cada nodo `p` la suma de sus hijos es `*p`.
   c' Cada nodo tiene a lo sumo `g` hijos, con `g`> 1 una constante dada.
   Ayuda: El árbol se puede construir poniendo inicialmente M en la raíz, y dividiendo el contenido `*n` de cada nodo en `g` valores iguales.
10) *autoCad*. En un programa de diseño asistido por computadora (tipo AutoCAD) se desea mantener las piezas de un sistema (por ejemplo un auto) clasificando sus partes en la forma de un árbol, donde sus nodos interiores representan sistemas del auto (planta motriz, dirección, carrocería), sus hijos subs-sistemas y así siguiendo hasta las hojas que son los componentes indivisibles del automóvil (por ej. el tornillo de fijación del espejo retrovisor izquierdo). Se quiere mantener en cada hoja el peso (en Kilogramos) del componente, y en los nodos interiores el peso total de todos los componentes del subárbol que cuelga de ese nodo. Periódicamente se quiere verificar que efectivamente las etiquetas del árbol verifican esta propiedad, es decir que la etiqueta de los nodos interiores es la suma de las hojas del subárbol que cuelga de él. Escribir una función `bool verif_sum (tree <int> &T, node_t n)` que retorna true si el subárbol que cuelga del nodo `n` verifica la condición dada.
11) *Prim*. Programe `int Prim(graphW& G, tree<int>&T)` que dado el grafo no dirigido ponderado `G` definido como `typedef map<int,map<int,int>> graphW;`, implemente el algoritmo de Prim para hallar y retornar el árbol de expansión de costo mínimo.
12) *Kruskal*. Programe `int Kruskal(graphW& G, tree<int>&T)` que dado el grafo no dirigido ponderado `G` definido como `typedef map<int,map<int,int>> graphW;`, implemente el algoritmo de Kruskal para hallar y retornar el árbol de expansión de costo mínimo.

b) [**Árboles Binarios**]

1) *ListarAB*. Escribir procedimientos para listar un árbol ordenado orientado siguiendo los órdenes prefijo, infijo y postfijo.
2) *VariosAB*. Escribir procedimientos para:
   a' Determinar si dos árboles tienen la misma estructura.
   b' Determinar si la estructura de un árbol es la espejada de otro.
   c' Determina si dos árboles son iguales, en estructura y contenido.

d' Copiar un árbol en otro en forma espejada.

3) *Profundidad.* Escribir una función `int cant_nodos_prof(btree<int> &A, int prof)` que retorna el número de nodos de un árbol binario `A` que están a profundidad `prof` o mayor.
4) *bin2ord.* Escribir una función `void bin2ord(btree<int> &B, tree<int> &T)` que dado un árbol binario `B` de enteros positivos lo convierte a un árbol ordenado orientado con la siguiente convención: En caso de que uno de los nodos del AB tenga uno sólo de los hijos reemplazamos el valor por un valor λ (en este caso puede ser $\lambda = -1$).
5) *ord2bin.* Escribir la función inversa `void ord2bin(tree<int> &T, btree<int> &B)` que dado un AOO (que supuestamente fue generado por `bin2ord`) lo convierte de nuevo a AB. (Debería ser siempre `B=ord2bin(bin2ord(B))`).
6) *compbtree.* Se define una relación de orden entre árboles binarios de enteros de la siguiente forma: $A < B$ si $a < b$, o bien ($a = b$ y $A_i < B_i$), o bien ($a = b$ y $A_i = B_i$ y $A_d < B_d$), dónde a, b son las raíces y A_i, A_d, B_i, B_d son los subárboles izquierdos y derechos de A y B. Consigna: Escribir una función `bool es_menor(tree<int> &A,tree<int> &B)` que retorna verdadero si $A < B$.
7) *contenido.* Escribir una función predicado `bool contenido(btree<int> &A, btree<int> &B)` que retorna verdadero si la estructura del árbol binario `A` esta contenido dentro de la de `B` y las etiquetas correspondientes de `A` son menores que las de `B`.
8) *balanceado.* Un árbol binario es balanceado si

   a' Es el arbol vacío o,

   b' Sus subárboles derecho e izquierdo son balanceados, y sus alturas difieren a lo sumo en 1.

   Escribir una función `bool is_balanced(btree<int> &T)` que determina si el árbol binario `T` está balanceado.
9) *Huffman.* Escribir una función `btree<char> make_huffman_tree(map<char,float> &A)` que dados un alfabeto `A` donde las clave son los caracteres a codificar, y los con valores las probabilidades de ocurrencia de cada uno, construya y retorna el árbol de la codificación de Huffman.