

ESTRUCTURAS DE DATOS Y ALGORITMOS

CURSO 2009 - PRÁCTICO 8

Nota: al igual que en los prácticos 6 y 7, en los problemas que siguen se usarán los tipos *lista* y *árbol binario*, como *tipos abstractos*, cada uno de ellos definido por un conjunto mínimo de funciones (constructoras, predicados y selectoras).

Pruebe estos ejercicios en máquina, importando los módulos de lista y árbol binario, luego pruébelos con las diferentes implementaciones de los tipos abstractos.

1-

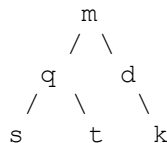
- Escribir en C/C++ un módulo de definición para el Tipo Abstracto de Datos Árbol Binario de Naturales. Dar una especificación funcional mínima de constructores, predicados y selectores.
- Escribir en C/C++ un módulo de implementación para el TAD Árbol Binario de Naturales. Utilice la representación clásica de punteros para Árboles Binarios e implemente las operaciones.

2-

- Demostrar que un árbol binario con todos los niveles completos y con n nodos internos (nodos que no son hojas) tiene $n+1$ nodos externos (nodos que son hojas).
- Demostrar que un árbol de n nodos tiene exactamente $n-1$ aristas.

3-

- Si cada nodo de un Árbol Binario de Caracteres contiene una letra, la concatenación de las mismas en cada camino que va desde la raíz a una hoja representa una palabra. Escribir una función que devuelva una lista con todas las palabras almacenadas en un árbol binario.
- Escribir una función que dados: un Árbol Binario de Caracteres y un camino expresado en forma de Lista de Caracteres, determine si existe dicho camino en el árbol teniendo en cuenta que el camino debe comenzar en la raíz y terminar en una hoja. Por ejemplo, para el árbol que sigue existen los caminos $m-q-t$ y $m-d-k$, pero no existen los caminos $m-d$, $r-q-t$ ni $d-k$:



4-

- Escribir en C/C++ un módulo de definición para el Tipo Abstracto de Datos Expresiones Aritméticas de números enteros. Las operaciones permitidas son: el producto, la división, la suma y la resta. La especificación funcional mínima de constructores, predicados y selectores debe constar de las siguientes operaciones:

```
Expre* Constante(int x);  
/* Devuelve una expresión constante, es decir, que sólo contiene un valor  
* numérico  
*/  
  
Expre* ConsExpre(char o, Expre *l, Expre *r);  
/* Crea una expresión con el operador o y las subexpresiones l y r  
*/  
  
bool EsConstante(Expre *e);  
/* Determina si una expresión es constante  
*/  
  
char OperadorExpre(Expre *e);  
/* Devuelve el operador de una expresión que no es constante  
*/  
  
int ValorExpre(Expre *e);  
/* Devuelve el valor de una expresión constante  
*/  
  
Expre* IzquierdaExpre(Expre *e);  
/* Devuelve la subexpresión izquierda de una expresión que no es  
constante  
*/  
  
Expre* DerechaExpre(Expre *e);  
/*Devuelve la subexpresión derecha de una expresión que no es constante  
*/
```

- b) Escribir en C/C++ un módulo de implementación para el TAD Expresiones Aritméticas. Utilice la representación clásica de punteros para Árboles Binarios e implemente las operaciones.
- c) Implementar la operación de evaluación de Expresiones Aritméticas, usando sólo las operaciones definidas en la parte a):

```
int EvaluarExpre(Expre *e);  
/* Devuelve el valor de una expresión aritmética  
*/
```

5-

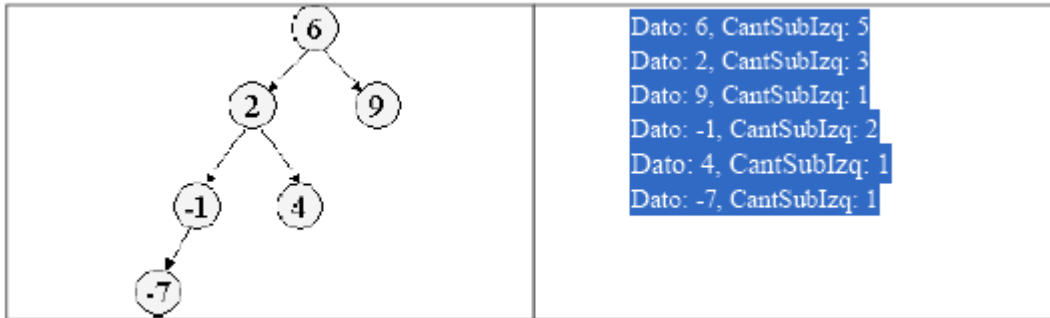
- a) Escribir en C/C++ un módulo de definición para el Tipo Abstracto de Datos Árbol Binario de Búsqueda de Naturales. Dar una especificación funcional mínima de constructores, predicados y selectores.
- b) Escribir en C/C++ un módulo de implementación para el TAD Árbol Binario de Búsqueda de Naturales. Utilice la representación clásica de punteros de Árbol Binario de Búsqueda e implemente las operaciones.

6-

- a) Agregar a la especificación de Árbol Binario de Búsqueda de Naturales del ejercicio 5 la siguiente operación:
kMenor, que dado un natural k retorna el subárbol que tiene al k -ésimo menor elemento del árbol como raíz, si éste existe. Si no hay al menos k elementos en el árbol o k es cero, la función debe retornar el árbol vacío. Si k es 1, nos referimos al menor elemento del árbol, si k es 2 al segundo elemento más pequeño del árbol y así sucesivamente.

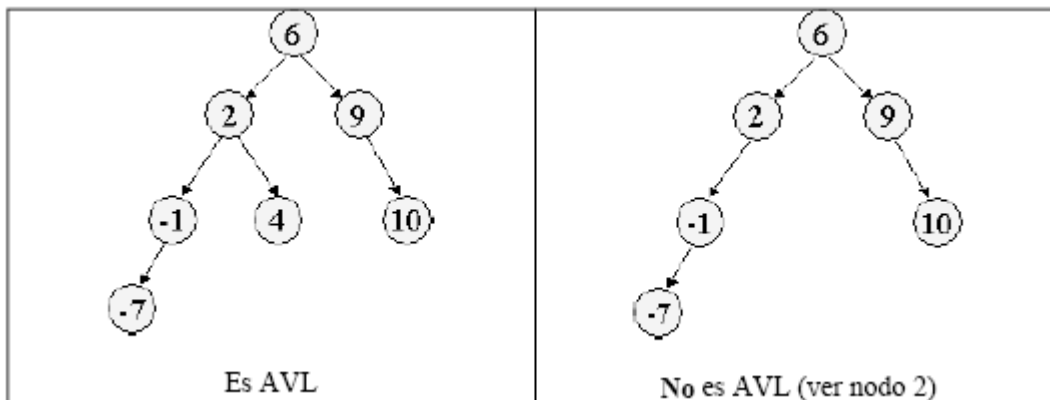
- b) Implementar el TAD **Árbol Binario de Búsqueda** extendido de tal manera que la operación de inserción y la función **kMenor** recorran a lo sumo un camino del árbol (desde la raíz a una hoja). Para esto considere una representación donde cada nodo del árbol almacene adicionalmente al dato la cantidad de elementos en su subárbol izquierdo más 1.

Ejemplo:



7-

- a) Definir una función que dado un **Árbol Binario** retorne *TRUE* si, y solamente si, el árbol es un **Árbol Binario de Búsqueda**.
- b) Un **Árbol Binario de Búsqueda** es un **AVL** si para cada nodo del árbol se cumple que las alturas de sus subárboles izquierdo y derecho difieren a lo sumo en uno. El árbol vacío es un AVL. Definir una función que dado un **Árbol Binario de Búsqueda de Naturales** retorne *TRUE* si, y solamente si, el árbol es un AVL.



8-

- a) Escribir en C/C++ un módulo de definición para el Tipo Abstracto de Datos **Árbol Finitario de Naturales**. La especificación funcional mínima de constructores, predicados y selectores debe constar de las siguientes operaciones:

```

FTree* NullFTree();
/* Devuelve el árbol vacío
*/

FTree* ConsFTree(unsigned int k, ListFTree* l);
/* Crea un árbol no vacío a partir de un natural y una lista de hijos
* (subárboles)
*/
    
```

```
bool IsEmptyFTree (FTree *t);  
/* Determina si un árbol dado es o no vacío  
*/  
  
unsigned int RootFTree (FTree *t);  
/* Devuelve el valor en la raíz de un árbol no vacío  
*/  
  
ListFTree* OffspringFTree (FTree *t);  
/* Devuelve la lista de hijos de un árbol no vacío  
*/  
  
ListFTree* SiblingsFTree (FTree *t);  
/* Devuelve la lista de hermanos de un árbol no vacío  
*/
```

Nota: se puede considerar que el TAD **ListFTree** está definido e implementado.

- b) Escribir en C/C++ un módulo de implementación para el TAD **Árbol Finitario de Naturales** e implementar las operaciones. Utilice la representación de "Lista de hijos, Lista de hermanos" donde cada nodo tiene 2 punteros uno de los cuales indica el siguiente hermano en la lista de hermanos y otro que indica el comienzo de la lista de hijos, esta representación es similar a la representación clásica de punteros de **Árbol Binario** (debido a que cada nodo tiene información y 2 punteros) pero difiere en el uso que se les da a los punteros.
- c) Agregar las siguientes operaciones al TAD **Árbol Finitario de Naturales**:

```
FTree* CreateFTree(unsigned inr x );  
/* Crea un árbol no vacío, que no tiene hijos, con x en la raíz  
*/  
  
FTree* AddOffspringFTree( FTree *t, FTree *s);  
/* Agrega a t un hijo s  
*/  
  
FTree* DeleteOffspringFTree( FTree *t,unsigned int x);  
/* Elimina de t el único hijo que contiene en la raíz a x  
*/
```

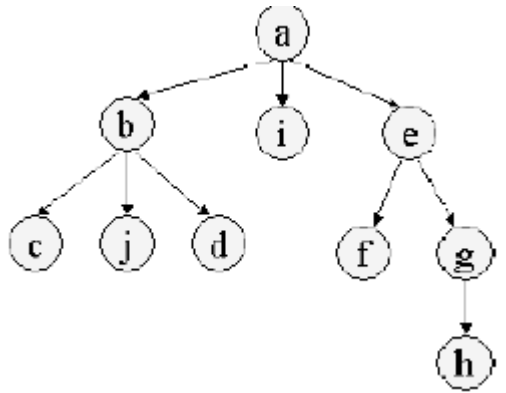
Nota: implementar estas operaciones, basándose en la representación dada en la parte b. Considerar que no se repite la información de los nodos.

9-

Se definen las recorridas en InOrden, PreOrden y PostOrden para **Árboles Finitarios** como sigue:

- **InOrden**: visita el hijo más a la izquierda, luego a la raíz y luego al resto de los hijos de izquierda a derecha.
- **PreOrden**: visita primero la raíz y luego los hijos de izquierda a derecha.
- **PostOrden**: visita de izquierda a derecha a los hijos y luego a la raíz.

a) Mostrar los resultados de las tres recorridas sobre el siguiente árbol:



- b) Escribir un procedimiento que realice la recorrida en InOrden de un Árbol Finitario.
- c) Escribir un procedimiento que realice la recorrida en PreOrden de un Árbol Finitario.
- d) Escribir un procedimiento que realice la recorrida en PostOrden de un Árbol Finitario.

10-

Se quiere representar los descendientes de una persona. Esto es, los hijos de la persona, los descendientes de sus hijos (nietos) y así sucesivamente. Cada persona está identificada por su nombre, es decir que los nombres no se repiten.

- a) Justificar porque un Árbol Binario **no** modela adecuadamente este tipo de información y un Árbol Finitario **sí** lo hace. Explicar que dato o datos debe contener dicho Árbol.
- b) Dada la estructura que representa los descendientes de una persona implementar las siguientes operaciones:
 - i. Dado el nombre de un descendiente indicar cuantas generaciones los separan.
 - ii. Imprimir los nombres de sus descendientes por generaciones.
 - iii. Indicar la cantidad de generaciones máxima que lo separan de un descendiente.
 - iv. Devolver la cantidad de descendientes que no tienen descendencia propia.

Nota: para la parte b. debe utilizar el TAD auxiliar Cola de Árboles Finitarios especificando sus operaciones. Puede suponerlo implementado.