

### Practico N° 11 – Árboles

#### Notas:

- Previo a la realización de este práctico es necesario leer el siguiente material sobre:  
- [Árboles](#)

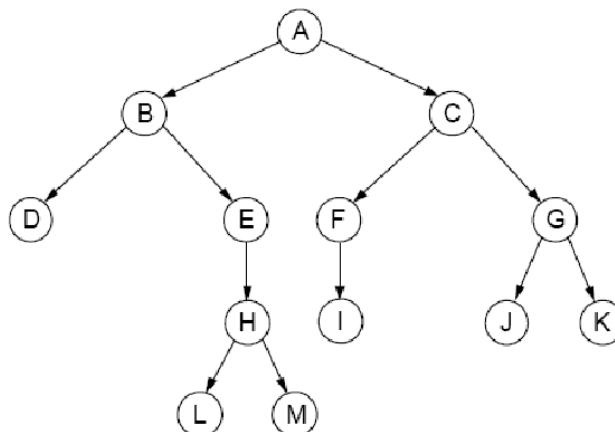
1. Para el árbol binario representado por el siguiente diagrama:

a) Listar los nodos del árbol anterior en:

- Preorden
- Enorden
- Postorden

b) Definir funciones que recorran un árbol binario en:

- Preorden
- Enorden
- Postorden



2. Definir un tipo de datos árbol que contiene un carácter y un entero en cada nodo, y exactamente tres subárboles.

3. Definir un tipo de datos árbol que contiene un entero en cada nodo, y que permite a cada nodo tener cualquier número de subárboles.

4. Defina el tipo de datos Tree que representa árboles binarios de elementos de un tipo genérico que sólo guarda información en los nodos hojas (nodos externos). Los nodos internos no guardan información. El árbol más pequeño es una hoja.

- Defina una función mapTree que dado un árbol de tipo (Tree A), para un tipo genérico A, y una función  $f:A \rightarrow B$ , con B un conjunto dado, retorne un árbol de tipo (Tree B) obtenido por la aplicación de la función f a cada uno de los nodos hojas del árbol parámetro.
- Defina una función que cuente la cantidad de nodos hojas que posee un árbol de tipo (Tree A), para un tipo genérico A.
- Defina una función hojas que retorne una lista con las hojas de un árbol de tipo (Tree A), para un tipo genérico A.

5. Defina el tipo de datos (BinTree A) de árboles binarios con nodos internos de un tipo genérico A y nodos externos (hojas) de un tipo genérico B. El árbol más pequeño es una hoja.

- Defina una función que cuente la cantidad de nodos externos de un árbol binario de tipo (BinTree A).
- Defina una función que cuente la cantidad de nodos internos de un árbol binario de tipo BinTree.

6. Definir un tipo de datos que implemente un árbol binario en Haskell de cualquier tipo.

7. Dados un árbol binario de naturales (Naturales.hs) y una función f de N en N definir una función **mapear** que devuelva el árbol resultante de aplicar f a todos los elementos del árbol dado.

8. Escribir en Haskell un ejemplo de cálculo de mapear aplicado a un árbol de 4 nodos y a la función (suma 3).

9. Definir una función **insertar** que dado un ABB de enteros y un entero inserte el entero dado en el árbol devolviendo un ABB.

## Árboles:

Recordar la definición del tipo genérico de un árbol:

```
Data Arbol a = Vacio | Nodo a [Arbol a] deriving Show
```

10. Definir una función en Haskell, **raiz**::Arbol a → Integer que devuelva la raíz del mismo.
11. El tamaño de un árbol es el número de elementos que almacena. Defina en Haskell una función **tamaño**::Arbol a → Integer, que calcule el tamaño de un árbol.
12. Los distintos elementos de un árbol están agrupados por niveles de modo que se considera que la raíz del árbol se encuentra en el nivel cero, las raíces de los subárboles del nodo raíz están en el nivel 1 y así sucesivamente. Se define profundidad de un árbol como el máximo nivel del árbol más uno.  
Defina en Haskell una función **profundidad**::Arbol a → Integer, que calcule la profundidad de un árbol.
13. Los nodos sin subárboles, se llaman nodos hoja y se caracterizan porque su lista de subárboles es vacía. Definir en Haskell una función **esHoja**::Arbol a → Bool, que devuelva True si el nodo es hoja y False en caso contrario.
14. Defina una función **sumArbol** que calcule la suma de los valores almacenados en un árbol de números enteros.
15. Defina una función **maxArbol** que calcule el máximo valor almacenado en un árbol.
16. Defina una función **algunoArbol**:: (a → Bool) → Arbol a → Bool, que compruebe si algún elemento de un árbol cumple un condición.
17. Defina una función **ocurrencias**::Eq a ⇒ a → Arbol a → Integer, que calcule el número de veces que aparece un dato en un árbol.

## Árboles Binarios:

Recordar la definición del tipo genérico de un árbol binario:

```
Data ArbolB a = VacioB | NodoB (ArbolB a) a (ArbolB a) deriving Show
```

Consideramos que las tres componentes del constructor **NodoB** son el subárbol izquierdo, el dato raíz y el subárbol derecho respectivamente. Los árboles binarios suelen ser utilizados como *contenedores* de datos.

18. Definir un función **perteneceB**:: Eq a ⇒ a → ArbolB a → Integer, que compruebe si un dato pertenece a un árbol binario.

### Árboles Binarios de búsqueda:

19. Definir una función **esVacioB**::  $\text{ArbolB } a \rightarrow \text{Bool}$  que compruebe si un árbol binario de búsqueda es vacío o no.
20. Definir en Haskell una función **esArbolBB**::  $\text{Ord } a \Rightarrow \text{ArbolB } a \rightarrow \text{Bool}$ , que compruebe si un árbol es o no un árbol binario de búsqueda.
21. Definir en Haskell una función:  
**eliminarBB**::  $\text{Ord } a \Rightarrow a \rightarrow \text{ArbolB } a \rightarrow \text{ArbolB } a$ , que elimine un dato de un nodo de un árbol binario de búsqueda.
22. Definir una función para insertar un dato dentro de un árbol de búsqueda de modo que el resultado sea también un árbol de búsqueda.  
**insertarBB**::  $\text{Ord } a \Rightarrow a \rightarrow \text{ArbolB } a \rightarrow \text{ArbolB } a$
23. El recorrido de **en orden** de un árbol binario dará como resultado una lista con los datos de un árbol de modo que primero se recorre el subárbol izquierdo, luego la raíz y por último el subárbol derecho. Definir una función **enOrden**::  $\text{Arbol } a \rightarrow [a]$  que realice este procedimiento.

### Ejercicios Complementarios:

24. Definir en Haskell las siguientes funciones que recorren un árbol:
  - **preOrden**::  $\text{BinTree } a \rightarrow [a]$
  - **enOrden**::  $\text{BinTree } a \rightarrow [a]$
  - **postOrden**::  $\text{BinTree } a \rightarrow [a]$
25. Sea el siguiente tipo para representar árboles binarios que almacenan datos tan sólo en sus hojas:  
**data**  $\text{ArbolH } a = \text{HojaH } a \mid \text{NodoH } (\text{ArbolH } a) (\text{ArbolH } a) \text{ deriving Show}$

Defina las siguientes funciones:

- a) **profundidadH** que calcule la profundidad de un árbol.
- b) **tamañoH** que calcule el tamaño de un árbol.
- c) **perteneceH** que compruebe si un dato pertenece a un árbol.
- d) **todosArbolH** que compruebe si todos los datos almacenados en un árbol cumplen un condición.
- e) **fmap** que aplique una función a todos los elementos de un árbol.