Practico 1 - Árboles

Ejercicio 1

A. Defina la clase **NodoAB** para utilizar en un árbol binario, que tenga los siguientes atributos:

```
private int dato;
private NodoAB der;
private NodoAB izq;
```

Definir los métodos de acceso y modificación y los constructores que considere necesarios.

B. Defina la clase **AB** con un atributo raíz del tipo **NodoAB**.

Para poder crear el árbol agregar el constructor que reciba la raíz.

Ejercicio 2

Implemente las siguientes funciones sobre árboles binarios.

A. `public int cantNodos();`

Pos: Retorna la cantidad de nodos del AB.

B. `public int cantHojas();`

Pos: Retorna la cantidad de nodos hoja del AB.

C. `public int altura();`

Pos: Retorna la altura del AB.

D. `public boolean todosPares();`

Pos: Retorna true si y solo si todos los elementos del AB son pares.

E. `public boolean iguales(AB a);`

Pos: Retorna true si y solo si es igual al AB pasado por parámetro. Dos árboles son iguales si son vacíos o si tienen los mismos elementos y en el mismo orden.

F. `public boolean equilibrado();`

Pos: Dado un árbol binario retorna true si y solo si el árbol es equilibrado.

G. `boolean pertenece(int x);`

Pos: Retorna true si y solo si el dato pasado como parámetro pertenece al AB.

Ejercicio 3

A. `public AB clon();`

Pos: Retorna un nuevo AB resultado de copiar todos los elementos del AB actual en el nuevo árbol.

Nota: Se debe crear un AB nuevo e independiente en memoria.

B. `public AB espejo();`

Pos: Retorna un nuevo AB resultado de copiar espejados todos los elementos del AB actual en el nuevo árbol.

Nota: Se debe crear un AB nuevo e independiente en memoria.

Ejercicio 4

A. Realice un dibujo del Árbol Binario de Búsqueda (ABB) resultante de insertar los siguientes elementos

a. 4, 2, 6, 1, 3, 5, 7
b. 1, 2, 3, 4, 5, 6, 7
c. 2, 1, 3, 4, 6, 5, 7

B. ¿Puede afirmar que los árboles binarios de búsqueda sin balancear SIEMPRE tienen mejor eficiencia que los árboles binarios? ¿Por qué?

Ejercicio 5

Implemente las siguientes funciones para la estructura de **Árbol Binario de Búsqueda** de enteros.

A. `void insertar(int x);`

Pos: Inserta el dato pasado como parámetro en el árbol manteniéndolo ordenado.

B. `boolean pertenece(int x);`

Pos: Retorna true si y solo si el dato pasado como parámetro pertenece al ABB.

C. `void listarAscendente();`

Pos: Lista en pantalla los elementos del ABB ordenados de menor a mayor.

D. `void listarDescendente();`

Pos: Lista en pantalla los elementos del ABB ordenados de mayor a menor.

E. `int borrarMinimo();`

Pos: Elimina el menor elemento del ABB y lo retorna.

Ejercicio 6

Nota: Considerar que el nivel de la raíz es 0.

A. Desarrolle un algoritmo que, recibiendo un valor entero k, retorne la cantidad de elementos que son mayores a k.

B. Desarrolle un algoritmo que retorne una lista con sus elementos ordenados de forma ascendente.

C. Desarrolle un algoritmo que, recibiendo un valor entero k, retorne una lista con los elementos que son mayores a k.

D. Desarrolle un algoritmo que, recibiendo un valor entero, retorne el nivel en que se encuentra dicho valor o -1 si no se encuentra.

E. Desarrolle un algoritmo que, recibiendo un valor entero k, imprima los elementos del nivel k.

F. Desarrolle un algoritmo que imprima el ABB en orden por niveles.

G. Desarrolle un algoritmo que retorne una lista de tuplas con todos los elementos del ABB y el número del nivel en el que se encuentra cada uno.

H. Desarrolle un algoritmo que, recibiendo un valor entero k, imprima la cantidad de elementos del nivel k.

I. Desarrolle un algoritmo que, recibiendo un valor de tipo String, imprima el ABB en orden por niveles, separando los niveles con el valor pasado por parámetro.

J. Desarrolle un algoritmo que, recibiendo un valor entero k, retorne el número de nivel del árbol que tiene mayor cantidad de nodos, considerando sólo hasta nivel k.
Nota: Recorrer el árbol una sola vez. Considerar utilizar un array auxiliar para guardar la cantidad de nodos por nivel.

Ejercicio 7

A. Crear la clase **Estudiante** con los siguientes atributos:

- **numero**: int.
- **nombre**: String.
- **edad**: int.
- **ci**: String.

Agregar a dicha clase:

- La implementación de la interfaz comparable.
- El método **equals()** comparando por el número de estudiante.
- El método **toString()**.

B. Implementar las siguientes funciones para la estructura de **ABB** de **Estudiante**.

- `void insertar(Estudiante estudiante);`
  Pos: Inserta el estudiante pasado como parámetro en el árbol manteniéndolo ordenado.

- `boolean pertenece(Estudiante estudiante);`
  Pos: Retorna true si y solo si el estudiante pasado como parámetro pertenece al ABB.
  Nota: El Estudiante pasado como parámetro puede tener seteados solo los atributos utilizados para comparar la igualdad (número de estudiante).

- `Estudiante obtener(Estudiante estudiante);`
  Pre: El estudiante pertenece al ABB.
  Pos: Retorna el estudiante con todos los atributos seteados.
  Nota: El Estudiante pasado como parámetro tiene seteados solo los atributos utilizados para comparar la igualdad (número de estudiante).

- `void listarAscendente();`
  Pos: Lista en pantalla los elementos del ABB ordenados de menor a mayor.

Ejercicio 8

Implementar las siguientes funciones para la estructura de un ABB genérico.

- `void` **`insertar`**`(T dato);`
  Pos: Inserta el dato pasado como parámetro en el árbol manteniéndolo ordenado.

- `boolean` **`pertenece`**`(T dato);`
  Pos: Retorna true si y solo si el dato pasado como parámetro pertenece al ABB.
  Nota: El dato pasado como parámetro puede tener seteados solo los atributos utilizados para comparar la igualdad.

- `Estudiante` **`obtener`**`(T dato);`
  Pre: El dato pertenece al ABB.
  Pos: Retorna el dato con todos los atributos seteados.
  Nota: El dato pasado como parámetro tiene seteados solo los atributos utilizados para comparar la igualdad.

- `void` **`listarAscendente`**`();`
  Pos: Lista en pantalla los elementos del ABB ordenados de menor a mayor.