Guía de Trabajos Prácticos

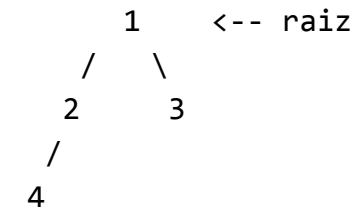
Unidad 05: Búsqueda

Ejercicio 5.01

Crear un programa que implemente búsqueda binaria en un arreglo

Ejercicio 5.02

Escribí un programa que defina la estructura necesaria para representar un árbol binario y cree un árbol como el siguiente.



Ejercicio 5.03

Escribí un programa que permita crear una clase que represente un árbol binario de búsqueda ABB e insertar nodos en el árbol.
Escribí el código para realizar una prueba, creando una o más instancias de la clase e insertando nodos.
Agregá un destructor para la clase ABB

Ejercicio 5.04

Implementá el método buscar(x) que busque un nodo en un árbol binario por su valor.

Ejercicio 5.05

Implementá los métodos para imprimir el árbol recorriendolo en preorden, inorden y postorden

Ejercicio 5.06

Implementá los métodos para recorrer el árbol en amplitud

Ejercicio 5.07

El recorrido en postorden de un ABB que contiene caracteres es:
DMLCTAISRUNOKB
Y en inorden es:
DMATLCBIKUSRON

Dibujá el árbol binario
Escribí una función que reconstruya el árbol original a partir de sus recorridos postorden e inorden

Ejercicio 5.08

Implementá una función que permita obtener la altura de un nodo.

Ejercicio 5.09

Implementar el método que permita determinar si un árbol es abb de acuerdo al procedimiento recursivo cuyo pseudocódigo está a continuación:

```
bool esAbb(nodo)
{
      si nodo es NULL retornar verdadero

            si (esSubarbolMenor(nodo.izquierdo, nodo.dato)
                  && esSubarbolMayor(nodo.derecho, nodo.dato)
                  && esAbb(nodo.izquierdo)
                  && esAbb(nodo.derecho))
                  retornar verdadero
      sino
            retornar falso
}
```

Ejercicio 5.10

Implementá el método que permita determinar si un árbol es abb de acuerdo al algoritmo con eficiencia O(n) que consiste en verificar los límites superior e inferior entre los que debe estar cada nodo.

Ejercicio 5.11

Implementá el método que permita determinar si un árbol es abb usando el recorrido inorden.

Ejercicio 5.12

Implementá el método eliminar que permite eliminar un nodo determinado a partir del valor provisto

Ejercicio 5.13

Implementá el método eliminar que permite mediante un algoritmo iterativo permite eliminar un nodo determinado a partir del valor provisto

Ejercicio 5.14

Escribí una función que encuentre el segundo mayor valor de un abb

Ejercicio 5.15

Dados dos árboles ABB: listar todos los elementos comunes a ambos árboles.

(nota: hay una solución trivial a este problema que es recorrer uno de los árboles e ir buscando cada elemento en el otro árbol. Esta solución es O(n1 x h2) donde n1 es la cantidad de elementos del primer árbol y h2 es la altura del segundo árbol. Tenés que encontrar una solución más eficiente)

Ejercicio 5.16

Creá un programa que imprima un abb en la consola

Ejercicio 5.17

Determiná los estados intermedios y el estado final del árbol AVL en el que se inserta la siguiente secuencia de valores:

43, 18, 22, 9, 21, 6, 8, 20, 63, 50, 62, 51

Ejercicio 5.18

Determiná los estados intermedios y el estado final del árbol AVL en el que se inserta la siguiente secuencia de valores:

21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

Ejercicio 5.19

Implementá una clase que modele el comportamiento del árbol AVL

Ejercicio 5.20

Implementá una tabla hash que utilice exploración lineal como mecanismo de resolución de colisiones

Ejercicio 5.21

Realizá las modificaciones necesarias a la implementación de la tabla hash para que almacene nombres de personas. Comprobá su correcto funcionamiento.

Ejercicio 5.22

Implementá método que cuente la cantidad de hojas de un árbol de búsqueda binaria

Ejercicio 5.23

Implementá un método que muestre en qué nivel (profundidad) está cada nodo de un árbol binario de búsqueda

Ejercicio 5.24

Implementá un método que verifique si un árbol de búsqueda binaria está o no balanceado en altura

Ejercicio 5.25

Implementá un método que retorne la distancia entre dos nodos cualesquiera de un árbol de búsqueda binaria