**COMP 203 DATA STRUCTURES AND ALGORITHMS**

**HOMEWORK 3 (Total=100 points)**

**Deadline: 31.12.2023 23:59**

**Read the questions and rules carefully. They are clear and well defined.**

**Rules:**

**1. No Cheating:** You are not allowed to collaborate with your friends and use any kind of websites or AI. If your homework gives a sign of any of them, **directly it will be graded as zero**.

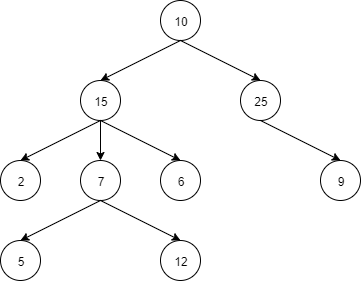
**2. Goal:** Please do your homework alone. Our main aim is to **learn** whatever we cover so far.

**3. Submission:** Submit your homework in **2 java files.** **No other file types will be accepted. You will submit only 2 java files. DON’T USE ZIP/RAR etc. In these cases, your points will be deducted by 30%.**

**4. Coding policy:** Explain your code in comments. **This is a must!**

**5. Latency policy:** A 10% deduction will be applied for each day of late submission.

**SOLUTION**



**Include comments of your code for each method and class.**

**Submit QTree.java to Canvas.**

**1. total 55 pt**

import java.util.LinkedList; //5pt usage of linkedlist

import java.util.Queue;

class Node<E> { //2pt

E data;

Queue<Node<E>> childrenList;

Node<E> parent;

public Node(E data) {//3pt

this.data = data;

this.childrenList = new LinkedList<>();

this.parent = null;

}

}

public class QTree<E> {//5pt

private Node<E> root;

public QTree() {

this.root = null;

}

public void deleteNode(Node<E> root, E deletedValue) { //10pt

if (root == null) {

return;

}

if (root.data.equals(deletedValue)) {

if (root.parent != null) {

root.parent.childrenList.remove(root);

} else {

// Deleting the root node

this.root = null;

}

return;

}

for (Node<E> child : root.childrenList) {

deleteNode(child, deletedValue);

}

}

public Node<E> find(Node<E> root, E value) { //10pt

if (root == null) {

return null;

}

if (root.data.equals(value)) {

return root;

}

for (Node<E> child : root.childrenList) {

Node<E> foundNode = find(child, value);

if (foundNode != null) {

return foundNode;

}

}

return null;

}

public static void main(String[] args) {

// Creating the given tree //10pt

QTree<Integer> tree = new QTree<>();

Node<Integer> node10 = new Node<>(10);

Node<Integer> node15 = new Node<>(15);

Node<Integer> node25 = new Node<>(25);

Node<Integer> node2 = new Node<>(2);

Node<Integer> node7 = new Node<>(7);

Node<Integer> node6 = new Node<>(6);

Node<Integer> node5 = new Node<>(5);

Node<Integer> node12= new Node<>(12);

Node<Integer> node9 = new Node<>(9);

tree.root = node10;

node10.childrenList.add(node15);

node10.childrenList.add( node25);

node15.childrenList.add(node2);

node15.childrenList.add(node7);

node15.childrenList.add(node6);

node7.childrenList.add(node5);

node7.childrenList.add(node12);

node25.childrenList.add(node9);

// Testing methods

System.out.println("Before deletion:");

tree.printTree(tree.root);

tree.deleteNode(tree.root, 4); //5pt

System.out.println("\nAfter deletion of node with value 4:");

tree.printTree(tree.root);

Node<Integer> foundNode = tree.find(tree.root, 5); //5pt

System.out.println("\nNode with value 5 found: " + (foundNode != null));

}

private void printTree(Node<E> root) {

if (root != null) {

System.out.print(root.data + " -> ");

for (Node<E> child : root.childrenList) {

System.out.print(child.data + " ");

}

System.out.println();

for (Node<E> child : root.childrenList) {

printTree(child);

}

}

}

}

2.

A diagram of a tree

Description automatically generated

**Include comments of your code for each method and class.**

**Submit BinarySearchTree.java to Canvas.**

**Total 45 pt**

class Node<E> { **//2pt**

E data;

Node<E> left;

Node<E> right;

public Node(E data) {**//3pt**

this.data = data;

this.left = null;

this.right = null;

}

}

public class BinarySearchTree<E extends Comparable<E>> { **//5pt**

private Node<E> root;

public BinarySearchTree() {

this.root = null;

}

public void insert(Node<E> root, E value) { **//10pt**

if (this.root == null) {

this.root = new Node<>(value);

} else {

if (value.compareTo(root.data) < 0) {

if (root.left == null) {

root.left = new Node<>(value);

} else {

insert(root.left, value);

}

} else if (value.compareTo(root.data) > 0) {

if (root.right == null) {

root.right = new Node<>(value);

} else {

insert(root.right, value);

}

}

// Duplicate values are not considered in this implementation

}

}

public Node<E> delete(Node<E> root, E value) { **//10pt**

if (root == null) {

return root;

}

if (value.compareTo(root.data) < 0) {

root.left = delete(root.left, value);

} else if (value.compareTo(root.data) > 0) {

root.right = delete(root.right, value);

} else {

// Node with only one child or no child

if (root.left == null) {

return root.right;

} else if (root.right == null) {

return root.left;

}

// Node with two children

root.data = minValue(root.right);

root.right = delete(root.right, root.data);

}

return root;

}

private E minValue(Node<E> root) {

E minValue = root.data;

while (root.left != null) {

minValue = root.left.data;

root = root.left;

}

return minValue;

}

public static void main(String[] args) {

BinarySearchTree<Integer> bst = new BinarySearchTree<>();

// Creating the given binary search tree **//5pt**

bst.insert(bst.root, 40);

bst.insert(bst.root, 30);

bst.insert(bst.root, 50);

bst.insert(bst.root, 25);

bst.insert(bst.root, 35);

bst.insert(bst.root, 45);

bst.insert(bst.root, 60); **//5pt**

// Testing methods

System.out.println("Before deletion:");

bst.PreOrderTraversal(bst.root);

bst.delete(bst.root, 60); **//5pt**

System.out.println("\nAfter deletion of node with value 60:");

bst.PreOrderTraversal(bst.root);

}

private void PreOrderTraversal(Node<E> root) { // to print the tree

if (root != null) {

System.out.print(root.data + " ");

PreOrderTraversal(root.left);

PreOrderTraversal(root.right);

}

}

}