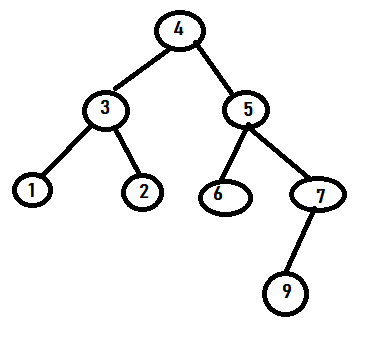
Name: **ZOHAIB HASSAN SOOMRO**

RollNo#: **19SW42**

Subject: **DSA(Practical)**



MY ASSUMED BINARY TREE FOR FOLLOWING CODE



**Task 1: Implement the binary tree with following fields:**

**Object root;**

**BinaryTree left, BinaryTree right;**

**• Create the following overloaded constructors:**

**◦ BinaryTree(Object root);**

**◦ BinaryTree(Object root, BinaryTree left, BinaryTree right);**

**• Create getter & setter methods for root, left and right.**

class BinaryTree{

Object root;

BinaryTree left,right;

BinaryTree(Object r){

root=r;

}

BinaryTree(Object r,BinaryTree l,BinaryTree ri){

root=r; left=l; right=ri;

}

public void setRoot(Object r){

root=r;

}

public void setLeft(BinaryTree l){

left=l;

}

public void setRight(BinaryTree ri){

right=ri;

}

public Object getRoot(){

return root;

}

public BinaryTree getLeft(){

return left;

}

public BinaryTree getRight(){

return right;

}

}

public class Task1{

public static void main(String[] args) {

BinaryTree tree=new BinaryTree(45);

BinaryTree tree2=new BinaryTree(45);

BinaryTree tree3=new BinaryTree(45);

}

}

**Task 2: Add the following methods to your class:**

**1. public String toString()**

**2. public boolean isLeaf()**

**3. public boolean isFull()**

**4. public boolean isComplete()**

**5. public boolean isBalanced()**

**6. public int size()**

**7. public int height()**

**8. public int degree()**

**9. public boolean contains(Object object)**

**10. public int numOfLeaves()**

**11. public int level(Object object)**

class BinaryTree{

Object root;

BinaryTree left,right;

BinaryTree(Object r){

root=r;

}

BinaryTree(Object r,BinaryTree l,BinaryTree ri){

root=r; left=l; right=ri;

}

public void setRoot(Object r){

root=r;

}

public void setLeft(BinaryTree l){

left=l;

}

public void setRight(BinaryTree ri){

right=ri;

}

public Object getRoot(){

return root;

}

public BinaryTree getLeft(){

return left;

}

public BinaryTree getRight(){

return right;

}

public String toString(){

if(root==null)

return "";

String buf="";

if(left!=null)

buf+=left+",";

buf+=root;

if(right!=null)

buf+=","+right;

return buf;

}

public boolean isLeaf(){

if(left==null && right==null)

return true;

return false;

}

public boolean isFull(){

if (root==null) {

return false;

}

if(left==null && right==null)

return true;

if(left!=null && right!=null)

return (left.isFull() && right.isFull());

return false;

}

public boolean isComplete(){

if (root==null)

return false;

if (left==null && right==null)

return true;

if (left==null && right!=null)

return false;

if (left!=null && right==null)

return left.isComplete();

if(left!=null && right!=null)

return (left.isComplete() && right.isComplete());

return true;

}

public boolean isBalanced(){

if(root==null)

return false;

if(left==null && right==null)

return true;

if((left.height()-right.height())==0 || (left.height()-right.height())==1 ||(left.height()-right.height())==-1)

return true;

return false;

}

public int size(){

int size=0;

if(root!=null)

size=1;

if(left!=null)

size+=left.size();

if(right!=null)

size+=right.size();

return size;

}

public int height(){

int rights=0,lefts=0;

if(left!=null)

lefts=1+left.height();

if(right!=null)

rights=1+right.height();

return Math.max(lefts,rights);

}

public int degree(){

int leftDegree=0,rightDegree=0;

if(left==null && right==null)

return 0;

if(left!=null && right==null)

leftDegree=(int)Math.max(1,left.degree());

if(right!=null && left==null)

rightDegree=(int)Math.max(1,right.degree());

if(left!=null && right!=null){

leftDegree=Math.max(2,left.degree());

rightDegree=Math.max(2,right.degree());

}

return Math.max(leftDegree,rightDegree);

}

public boolean contains(Object object){

if(toString().contains(object+""))

return true;

return false;

}

public int numOfLeaves(){

int leaves=0;

if(left==null && right==null)

return 1;

if(right!=null)

leaves+=right.numOfLeaves();

if(left!=null)

leaves+=left.numOfLeaves();

return leaves;

}

public int level(Object object){

if(root==null)

return 0;

if(root.equals(object))

return 1;

if(left!=null && right!=null)

if(left.root.equals(object) || right.root.equals(object))

return 2;

else

return 1+Math.max(left.level(object),right.level(object));

return -1;

}

}

public class Task2{

public static void main(String[] args) {

BinaryTree tree1=new BinaryTree(1);

BinaryTree tree6=new BinaryTree(6);

BinaryTree tree9=new BinaryTree(9);

BinaryTree tree7=new BinaryTree(7,null,tree9);

BinaryTree tree2=new BinaryTree(2);

BinaryTree tree5=new BinaryTree(5,tree6,tree7);

BinaryTree tree3=new BinaryTree(3,tree1,tree2);

BinaryTree tree4=new BinaryTree(4,tree3,tree5);

System.out.println("tree4.toString() :"+tree4.toString());

System.out.println("tree9.isLeaf() :"+tree9.isLeaf());

System.out.println("tree4.size() :"+tree4.size());

System.out.println("tree4.height() :"+tree4.height());

System.out.println("tree4.degree() :"+tree4.degree());

System.out.println("tree4.contains(34) :"+tree4.contains(34));

System.out.println("tree4.numOfLeaves():"+tree4.numOfLeaves());

System.out.println("tree4.level(7) :"+tree4.level(7));

System.out.println("tree4.isBalanced() :"+tree4.isBalanced());

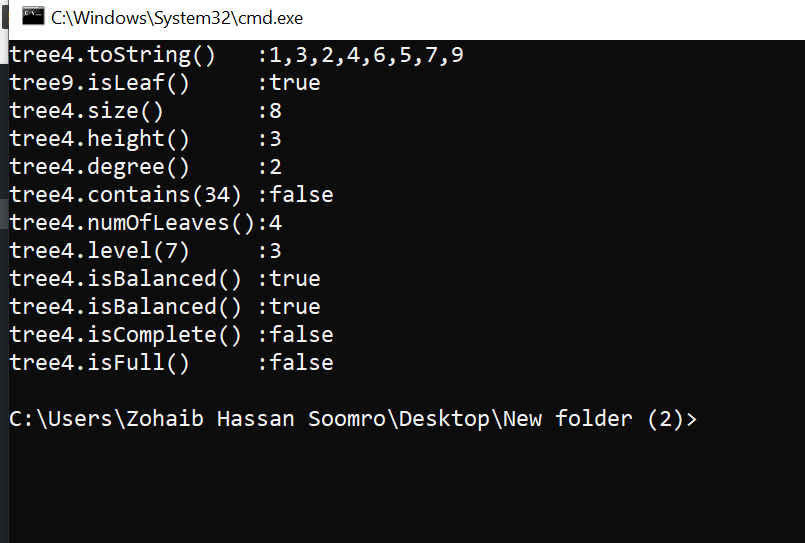
System.out.println("tree4.isBalanced() :"+tree4.isBalanced());

System.out.println("tree4.isComplete() :"+tree4.isComplete());

System.out.println("tree4.isFull() :"+tree4.isFull());

}

}



**Task 3: Implement all the traversal algorithms in your class.**

class BinaryTree{

Object root;

BinaryTree left,right;

BinaryTree(Object r){

root=r;

}

BinaryTree(Object r,BinaryTree l,BinaryTree ri){

root=r; left=l; right=ri;

}

public void setRoot(Object r){

root=r;

}

public void setLeft(BinaryTree l){

left=l;

}

public void setRight(BinaryTree ri){

right=ri;

}

public Object getRoot(){

return root;

}

public BinaryTree getLeft(){

return left;

}

public BinaryTree getRight(){

return right;

}

public String toString(){

if(root==null)

return "";

String buf="";

if(left!=null)

buf+=left+",";

buf+=root;

if(right!=null)

buf+=","+right;

return buf;

}

public boolean isLeaf(){

if(left==null && right==null)

return true;

return false;

}

public boolean isFull(){

if (root==null) {

return false;

}

if(left==null && right==null)

return true;

if(left!=null && right!=null)

return (left.isFull() && right.isFull());

return false;

}

public boolean isComplete(){

if (root==null)

return false;

if (left==null && right==null)

return true;

if (left==null && right!=null)

return false;

if (left!=null && right==null)

return left.isComplete();

if(left!=null && right!=null)

return (left.isComplete() && right.isComplete());

return true;

}

public boolean isBalanced(){

if(root==null)

return false;

if(left==null && right==null)

return true;

if((left.height()-right.height())==0 || (left.height()-right.height())==1 ||(left.height()-right.height())==-1)

return true;

return false;

}

public int size(){

int size=0;

if(root!=null)

size=1;

if(left!=null)

size+=left.size();

if(right!=null)

size+=right.size();

return size;

}

public int height(){

int rights=0,lefts=0;

if(left!=null)

lefts=1+left.height();

if(right!=null)

rights=1+right.height();

return Math.max(lefts,rights);

}

public int degree(){

int leftDegree=0,rightDegree=0;

if(left==null && right==null)

return 0;

if(left!=null && right==null)

leftDegree=(int)Math.max(1,left.degree());

if(right!=null && left==null)

rightDegree=(int)Math.max(1,right.degree());

if(left!=null && right!=null){

leftDegree=Math.max(2,left.degree());

rightDegree=Math.max(2,right.degree());

}

return Math.max(leftDegree,rightDegree);

}

public boolean contains(Object object){

if(toString().contains(object+""))

return true;

return false;

}

public int numOfLeaves(){

int leaves=0;

if(left==null && right==null)

return 1;

if(right!=null)

leaves+=right.numOfLeaves();

if(left!=null)

leaves+=left.numOfLeaves();

return leaves;

}

public int level(Object object){

if(root==null)

return 0;

if(root.equals(object))

return 1;

if(left!=null && right!=null)

if(left.root.equals(object) || right.root.equals(object))

return 2;

else

return 1+Math.max(left.level(object),right.level(object));

return -1;

}

public void preOrderTraversal(){

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

if(left!=null)

left.preOrderTraversal();

if(right!=null)

right.preOrderTraversal();

}

public void postOrderTraversal(){

if(left!=null)

left.postOrderTraversal();

if(right!=null)

right.postOrderTraversal();

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

}

public void inOrderTraversal(){

if(left!=null)

left.inOrderTraversal();

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

if(right!=null)

right.inOrderTraversal();

}

public void levelOrderTraversal(){

if(root==null) return;

LinkedQueue queue=new LinkedQueue();

queue.add(this);

BinaryTree currentTree;

while(!queue.isEmpty()){

currentTree=queue.remove();

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

if(currentTree.left!=null)

queue.add(currentTree.left);

if(currentTree.right!=null)

queue.add(currentTree.right);

}

}

}

public class Task3{

public static void main(String[] args){

BinaryTree tree1=new BinaryTree(1);

BinaryTree tree6=new BinaryTree(6);

BinaryTree tree9=new BinaryTree(9);

BinaryTree tree7=new BinaryTree(7,null,tree9);

BinaryTree tree2=new BinaryTree(2);

BinaryTree tree5=new BinaryTree(5,tree6,tree7);

BinaryTree tree3=new BinaryTree(3,tree1,tree2);

BinaryTree tree4=new BinaryTree(4,tree3,tree5);

System.out.println("preOrderTraversal() : ");

tree4.preOrderTraversal();

System.out.println("\n\npostOrderTraversal() : ");

tree4.postOrderTraversal();

System.out.println("\n\ninOrderTraversal() : ");

tree4.inOrderTraversal();

System.out.println("\n\nlevelOrderTraversal() : ");

tree4.levelOrderTraversal();

}

}

**LINKEDQUEUE CLASS:**

public class LinkedQueue{

int size;

Node head=new Node(null);

private class Node{

BinaryTree data;

Node next,prev;

Node(BinaryTree d){

data=d;

next=prev=this;

}

Node(BinaryTree d,Node p,Node n){

data=d;

next=n;

prev=p;

}

}

public void add(BinaryTree data){

head.prev=head.prev.next=new Node(data,head.prev,head);

size++;

}

public BinaryTree first(){

if(size==0) return null;

return head.next.data;

}

public BinaryTree remove(){

if(size==0) return null;

BinaryTree obj=head.next.data;

head.next=head.next.next;

head.next.prev=head;

size--;

return obj;

}

public boolean isEmpty(){

return size==0;

}

public int size(){

return size;

}

}

