DataStructures and algorithms

Lab 10

Name:Pranjal bajaj

Cms:023-21-0022

Task 1

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) {

        this.val = val;

        this.left = this.right = null;

    }

}

public class tree {

    public boolean isBalanced(TreeNode root) {

        return checkHeight(root) != -1;

    }

    private int checkHeight(TreeNode node) {

        if (node == null) return 0;

        int leftHeight = checkHeight(node.left);

        if (leftHeight == -1) return -1;

        int rightHeight = checkHeight(node.right);

        if (rightHeight == -1) return -1;

        if (Math.abs(leftHeight - rightHeight) > 1) return -1;

        return Math.max(leftHeight, rightHeight) + 1;

    }

    public static void main(String[] args) {

        tree tree = new tree();

        TreeNode root = new TreeNode(1);

        root.right = new TreeNode(2);

        root.right = new TreeNode(3);

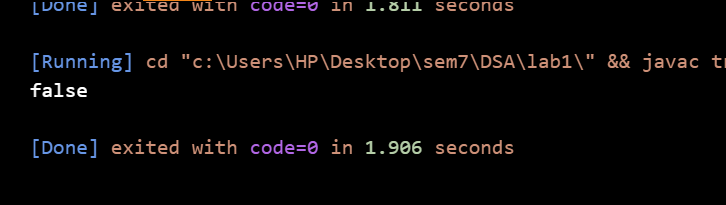
        root.right.left = new TreeNode(4);

        root.right.left = new TreeNode(5);

        System.out.println(tree.isBalanced(root));

    }

}



Task 2

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) {

        this.val = val;

        this.left = this.right = null;

    }

}

public class tree {

    public int findMin(TreeNode root) {

        if (root == null) throw new IllegalArgumentException("Tree is empty");

        return root.val;

    }

    public int findMax(TreeNode root) {

        if (root == null) throw new IllegalArgumentException("Tree is empty");

        while (root.right != null) root = root.right;

        return root.val;

    }

    public int rangeSumBST(TreeNode root, int low, int high) {

        return calculateSum(root, low, high);

    }

    private int calculateSum(TreeNode node, int low, int high) {

        if (node == null) return 0;

        if (node.val < low) return calculateSum(node.right, low, high);

        if (node.val > high) return calculateSum(node.left, low, high);

        return node.val + calculateSum(node.left, low, high) + calculateSum(node.right, low, high);

    }

    public static void main(String[] args) {

        tree bst = new tree();

        TreeNode root = new TreeNode(10);

        root.left = new TreeNode(5);

        root.right = new TreeNode(15);

        root.left.left = new TreeNode(3);

        root.left.right = new TreeNode(7);

        root.right.right = new TreeNode(25);

        int low = bst.findMin(root);

        int high = bst.findMax(root);

        System.out.println("Lowest value in tree: " + low);

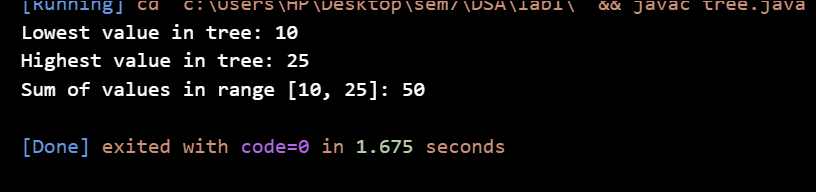
        System.out.println("Highest value in tree: " + high);

        int result = bst.rangeSumBST(root, low, high);

        System.out.println("Sum of values in range [" + low + ", " + high + "]: " + result);

    }

}



Task 3

import java.util.ArrayList;

import java.util.List;

class TreeNode {

    int val;

    TreeNode left, right;

    TreeNode(int val) {

        this.val = val;

        this.left = this.right = null;

    }

}

public class tree {

    public boolean findTarget(TreeNode root, int k) {

        List<Integer> sortedList = new ArrayList<>();

        inOrderTraversal(root, sortedList);

        int left = 0, right = sortedList.size() - 1;

        while (left < right) {

            int sum = sortedList.get(left) + sortedList.get(right);

            if (sum == k) return true;

            if (sum < k) left++;

            else right--;

        }

        return false;

    }

    private void inOrderTraversal(TreeNode node, List<Integer> list) {

        if (node == null) return;

        inOrderTraversal(node.left, list);

        list.add(node.val);

        inOrderTraversal(node.right, list);

    }

    public static void main(String[] args) {

        tree bst = new tree();

        TreeNode root = new TreeNode(10);

        root.left = new TreeNode(6);

        root.right = new TreeNode(15);

        root.left.left = new TreeNode(3);

        root.left.right = new TreeNode(7);

        int k = 15;

        System.out.println("Does the tree have two nodes with sum " + k + "? " + bst.findTarget(root, k));

    }

}

