Trees Assignment

20BD1A1290-Jakkanagari Harika

Program 1:

Given the roots of two binary trees p and q, write a function to check if they are the same or not. Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Solution:

import java.util.\*;

class Tree{

public static void main(String args[]){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

int b[]=new int[n];

for(int i=0;i<n;i++){

b[i]=sc.nextInt();

}

Tree t=new Tree();

Node r1=new Node(a[0]);

for(int i=1;i<n;i++){

t.insert(r1,a[i]);

}

Node r2=new Node(b[0]);

for(int i=1;i<n;i++){

t.insert(r2,b[i]);

}

System.out.println(t.method(r1,r2));

}

static class Node {

int value;

Node left, right;

Node(int value){

this.value = value;

left = null;

right = null;

}

}

public void insert(Node node, int value) {

if (value < node.value) {

if (node.left != null) {

insert(node.left, value);

}

else {

node.left = new Node(value); } } else if (value > node.value) {

if (node.right != null) {

insert(node.right, value);

} else {

node.right = new Node(value);

}

}

}

public boolean method(Node a,Node b){

if(a==null && b==null){

return true;

}

if(a!=null && b!=null){

return (a.value==b.value && method(a.left,b.left)&&method(a.right,b.right));

}

return false;

}

}

Input: p = [1,2,3], q = [1,2,3]Output: true

Program 2:

Symmetric Tree Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Solution:

import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

String a[]=new String[n];

for(int i=0;i<n;i++){

a[i]=sc.next();

}

root=new TreeNode(Integer.parseInt(a[0]));

for(int i=1;i<a.length;i++){

String s=a[i];

// int j;

if(s.equals("null")){

insert(root,-1);

}

else{

insert(root,Integer.parseInt(a[i]));

}

}

System.out.println(checking());

}

public static boolean checking(){

return check(root,root);

}

public static boolean check(TreeNode left,TreeNode right){

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

return true;

}

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

return (check(left.left,right.right)&& check(left.right,right.left));

}

return false;

}

}

Input: root = [1,2,2,3,4,4,3]Output: true

3. Invert Binary Tree Given the root of a binary tree, invert the tree, and return its root.

Solution:  
import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

String a[]=new String[n];

for(int i=0;i<n;i++){

a[i]=sc.next();

}

root=new TreeNode(Integer.parseInt(a[0]));

for(int i=1;i<a.length;i++){

String s=a[i];

// int j;

if(s.equals("null")){

insert(root,-1);

}

else{

insert(root,Integer.parseInt(a[i]));

}

}

method(root);

print(root);

}

public static void print(TreeNode root){

if(root==null){

return;

}

System.out.print(root.data);

print(root.left);

print(root.right);

}

public static void method(TreeNode root){

if(root==null){

return;

}

Queue<TreeNode> q=new LinkedList<>();

q.add(root);

while(q.size()>0){

TreeNode cur=q.peek();

q.remove();

TreeNode temp=cur.left;

cur.left=cur.right;

cur.right=temp;

if(cur.left!=null){

q.add(cur.left);

}

if(cur.right!=null){

q.add(cur.right);

}

}

}

}

Input: root = [4,2,7,1,3,6,9]Output: [4,7,2,9,6,3,1]

4. Balanced Binary Tree Given a binary tree, determine if it is height-balanced

Solution:  
import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

String a[]=new String[n];

for(int i=0;i<n;i++){

a[i]=sc.next();

}

root=new TreeNode(Integer.parseInt(a[0]));

for(int i=1;i<a.length;i++){

String s=a[i];

// int j;

if(s.equals("null")){

insert(root,-1);

}

else{

insert(root,Integer.parseInt(a[i]));

}

}

System.out.println(method(root));

print(root);

}

static boolean method(TreeNode node)

{

int lh;

int rh;

if (node == null)

return true;

lh = height(node.left);

rh = height(node.right);

if (Math.abs(lh - rh) <= 1 &&method(node.left)

&& method(node.right))

return true;

return false;

}

public static void print(TreeNode root){

if(root==null){

return;

}

System.out.print(root.data);

print(root.left);

print(root.right);

}

static int height(TreeNode node)

{

if (node == null)

return 0;

return 1+ Math.max(height(node.left),height(node.right));

}

}

Input: root = [3,9,20,null,null,15,7]Output: true

5. Merge Two Binary Trees You are given two binary trees root1 and root2. Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree. Return the merged tree. Note: The merging process must start from the root nodes of both trees.

Solution:

import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

static TreeNode root1;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

String a[]=new String[n];

for(int i=0;i<n;i++){

a[i]=sc.next();

}

root=new TreeNode(Integer.parseInt(a[0]));

for(int i=1;i<a.length;i++){

String s=a[i];

// int j;

if(s.equals("null")){

insert(root,-1);

}

else{

insert(root,Integer.parseInt(a[i]));

}

}

int n1=sc.nextInt();

String b[]=new String[n1];

for(int i=0;i<n1;i++){

b[i]=sc.next();

}

root1=new TreeNode(Integer.parseInt(b[0]));

for(int i=1;i<n1;i++){

String s=b[i];

// int j;

if(s.equals("null")){

insert(root1,-1);

}

else{

insert(root1,Integer.parseInt(b[i]));

}

}

TreeNode root3=mergeTrees(root,root1);

print(root3);

}

public static TreeNode mergeTrees(TreeNode r1, TreeNode r2) {

if (r1 == null) return r2;

if (r2 == null) return r1;

if(r1.data==-1||r2.data==-1){

r1.data=r1.data+r2.data+1;

}

else{

r1.data=r1.data+r2.data;

}

r1.left = mergeTrees(r1.left, r2.left);

r1.right = mergeTrees(r1.right, r2.right);

return r1;

}

public static void print(TreeNode r){

if(r==null){

return;

}

if(r.data==-1){

System.out.print("null");

return;

}

else if(r.data!=-1){

System.out.print(r.data);

}

print(r.left);

print(r.right);

}

}

Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]Output: [3,4,5,5,4,null,7]

6.Validate Binary Search Tree Given the root of a binary tree, determine if it is a valid binary search tree (BST)

Solution:

import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

// static TreeNode root1;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

String a[]=new String[n];

for(int i=0;i<n;i++){

a[i]=sc.next();

}

root=new TreeNode(Integer.parseInt(a[0]));

for(int i=1;i<a.length;i++){

String s=a[i];

// int j;

if(s.equals("null")){

insert(root,-1);

}

else{

insert(root,Integer.parseInt(a[i]));

}

}

System.out.println(isValidBST(root));

}

public static boolean isValidBST(TreeNode root) {

if (root == null) return true;

Stack<TreeNode> stack = new Stack<>();

TreeNode pre = null;

while (root != null || !stack.isEmpty()) {

while (root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

if(pre != null && root.data <= pre.data) return false;

pre = root;

root = root.right;

}

return true;

}

public static void print(TreeNode r){

if(r==null){

return;

}

if(r.data==-1){

System.out.print("null");

return;

}

else if(r.data!=-1){

System.out.print(r.data);

}

print(r.left);

print(r.right);

}

}

Input: root = [2,1,3]Output: true

7. Kth Smallest Element in a BST Given the root of a binary search tree, and an integer k, return the k th smallest value (1-indexed) of all the values of the nodes in the tree.

Solution:  
import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

// static TreeNode root1;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

String a[]=new String[n];

for(int i=0;i<n;i++){

a[i]=sc.next();

}

root=new TreeNode(Integer.parseInt(a[0]));

for(int i=1;i<a.length;i++){

String s=a[i];

// int j;

if(s.equals("null")){

insert(root,-1);

}

else{

insert(root,Integer.parseInt(a[i]));

}

}

int k=sc.nextInt();

System.out.println(kthSmallest(root,k));

}

public static int kthSmallest(TreeNode root, int k) {

Stack<TreeNode> stack = new Stack<>();

while(root != null || !stack.isEmpty()) {

while(root != null) {

stack.push(root);

root = root.left;

}

root = stack.pop();

if(--k == 0) break;

root = root.right;

}

return root.data;

}

public static void print(TreeNode r){

if(r==null){

return;

}

if(r.data==-1){

System.out.print("null");

return;

}

else if(r.data!=-1){

System.out.print(r.data);

}

print(r.left);

print(r.right);

}

}

Input: root = [3,1,4,null,2], k = 1Output: 1

8. Convert Sorted Array to Balanced Binary Search Tree Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

Solution:  
import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

TreeNode n1=sortedArrayToBST(a,0,n-1);

print(n1);

}

public static TreeNode sortedArrayToBST(int a[],int s,int e){

if(s>e){

return null;

}

int mid=s+(e-s)/2;

TreeNode node=new TreeNode(a[mid]);

node.left=sortedArrayToBST(a,s,mid-1);

node.right=sortedArrayToBST(a,mid+1,e);

return node;

}

public static void print(TreeNode r){

if(r==null){

return;

}

System.out.print(r.data);

print(r.left);

print(r.right);

}

}

Input: nums = [-10,-3,0,5,9]Output: [0,-3,9,-10,null,5]

Q8 Range Sum of BST Google Facebook Amazon Given the root node of a binary search tree and two integers low and high, return the sum of values of all nodes with a value in the inclusive range [low, high].

Solution:  
import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

// static TreeNode root1;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

root=new TreeNode(a[0]);

for(int i=1;i<a.length;i++){

insert(root,a[i]);

}

int low=sc.nextInt();

int high=sc.nextInt();

traverse(root,low,high);

System.out.println(sum);

}

static int sum;

public static void traverse(TreeNode root,int low,int high)

{

if (root==null)

return;

else

{

if (root.data>low)

traverse(root.left,low,high);

if (root.data<high)

traverse(root.right,low,high);

if(root.data>=low && root.data<=high)

sum+=root.data;

}

}

public static void print(TreeNode r){

if(r==null){

return;

}

if(r.data==-1){

System.out.print("null");

return;

}

else if(r.data!=-1){

System.out.print(r.data);

}

print(r.left);

print(r.right);

}

}

Input: root = [10,5,15,3,7,null,18], low = 7, high = 15

Q9 Search in a Binary Search Tree You are given the root of a binary search tree (BST) and an integer val. Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null

Solution:  
import java.util.\*;

class TreeNode{

int data;

TreeNode left,right;

TreeNode(int node){

this.data=node;

this.left=null;

this.right=null;

}

}

class Test{

static TreeNode root;

// static TreeNode root1;

static void insert(TreeNode temp,int key){

if(temp==null){

temp=new TreeNode(key);

return;

}

Queue<TreeNode> qu=new LinkedList<TreeNode>();

qu.add(temp);

while(!qu.isEmpty()){

temp=qu.peek();

qu.remove();

if(temp.left==null){

temp.left=new TreeNode(key);

break;

}

else{

qu.add(temp.left);

}

if(temp.right==null){

temp.right=new TreeNode(key);

break;

}

else{

qu.add(temp.right);

}

}

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int a[]=new int[n];

for(int i=0;i<n;i++){

a[i]=sc.nextInt();

}

root=new TreeNode(a[0]);

for(int i=1;i<a.length;i++){

insert(root,a[i]);

}

int l=sc.nextInt();

TreeNode r1=searchBST(root,l);

print(r1);

}

public static TreeNode searchBST(TreeNode root, int val) {

if (root == null || root.data== val) {

return root;

}

if (val > root.data) {

return searchBST(root.right, val);

} else {

return searchBST(root.left, val);

}

}

public static void print(TreeNode r){

if(r==null){

return;

}

if(r.data==-1){

System.out.print("null");

return;

}

else if(r.data!=-1){

System.out.print(r.data);

}

print(r.left);

print(r.right);

}

}

Input: root = [4,2,7,1,3], val = 2