DAY 17

2)class Node {

int data;

Node left, right;

Node(int d) {

data = d;

left = null;

right = null;

}

}

public class Main {

static int countleft(Node root){

int count=0;

Node current=root.left;

if(root!=null){

if(isleaf(current)){

return 1;

}

}

return countleft(root.left) + countleft(root.right);

}

static int countright(Node root){

int count=0;

Node current=root.right;

if(root!=null){

if(isleaf(current)){

return 1;

}

}

return countright(root.left) + countright(root.right);

}

static boolean isleaf(Node root){

return root.left==null && root.right==null;

}

public static void main(String[] args) {

Node root = new Node(2);

Node left = new Node(3);

Node right = new Node(4);

root.left = left;

root.right = right;

root.left.left = new Node(5);

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

root.right.left = new Node(8);

root.right.right = new Node(9);

root.right.right = new Node(9);

int m= countleft(root);

int n= countright(root);

System.out.println(m+n);

}

}

1)

import java.util.\*;

class Node {

String val;

Node left, right;

Node(String v) { val = v; }

}

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String[] tokens = sc.nextLine().trim().split("\\s+");

sc.close();

Node root = buildTree(tokens);

printLevelOrder(root);

}

// Build tree from level-order String tokens

static Node buildTree(String[] arr) {

if (arr.length == 0 || arr[0].equals("null")) return null;

Node root = new Node(arr[0]);

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

q.offer(root);

int i = 1;

while (i < arr.length) {

Node cur = q.poll();

String s = arr[i++];

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

cur.left = new Node(s);

q.offer(cur.left);

}

if (i < arr.length) {

s = arr[i++];

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

cur.right = new Node(s);

q.offer(cur.right);

}

}

}

return root;

}

// Print level-order traversal

static void printLevelOrder(Node root) {

if (root == null) return;

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

q.offer(root);

while (!q.isEmpty()) {

Node cur = q.poll();

System.out.print(cur.val + " ");

if (cur.left != null) q.offer(cur.left);

if (cur.right != null) q.offer(cur.right);

}

System.out.println();

}

}

3)

class Node {

int val;

Node left, right;

Node(int v) { val = v; }

}

public class Main {

// Recursive method to find tree height

static int height(Node root) {

if (root == null) {

return -1;

// empty tree has height -1, single-node tree is 0

}

int leftH = height(root.left);

int rightH = height(root.right);

return Math.max(leftH, rightH) + 1;

}

public static void main(String[] args) {

Node root = new Node(12);

root.left = new Node(8);

root.right = new Node(18);

root.left.left = new Node(5);

root.left.right = new Node(11);

System.out.println("Height = " + height(root));

}

}