Code : 1  
package Topic\_15\_Trees;

import java.io.\*;

import java.util.\*;

public class DiamterTree {

private static class Node {

int data;

ArrayList<Node> children = new ArrayList<>();

}

public static void display(Node node) {

String str = node.data + " -> ";

for (Node child : node.children) {

str += child.data + ", ";

}

str += ".";

System.out.println(str);

for (Node child : node.children) {

display(child);

}

}

public static Node construct(int[] arr) {

Node root = null;

Stack<Node> st = new Stack<>();

for (int i = 0; i < arr.length; i++) {

if (arr[i] == -1) {

st.pop();

} else {

Node t = new Node();

t.data = arr[i];

if (st.size() > 0) {

st.peek().children.add(t);

} else {

root = t;

}

st.push(t);

}

}

return root;

}

static int dia = 0;

public static int diameter(Node node) {

// write your code here

int dch = -1;

int sdch = -1;

for (Node n : node.children) {

int ch = diameter(n);

if (ch >= dch) {

sdch = dch;

dch = ch;

} else if (ch >= sdch) {

sdch = ch;

}

}

int cand = dch + sdch + 2;

if (cand > dia) {

dia = cand;

}

dch += 1;

return dch;

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

// int n = Integer.parseInt(br.readLine());

// String[] values = "10 20 50 -1 60 -1 -1 30 70 -1 80 110 130 150 170 -1 -1 -1 -1 120 140

// 160 180 190 -1 -1 -1 -1 -1 -1 90 -1 -1 40 100 -1 -1 -1".split(" ");

String[] values =

"10 20 50 -1 60 -1 -1 30 70 -1 80 110 130 150 170 -1 -1 -1 -1 120 140 160 180 190 -1 -1 -1 -1 -1 -1 90 -1 -1 40 100 -1 -1 -1"

.split(" ");

int[] arr = new int[values.length];

for (int i = 0; i < arr.length; i++) {

arr[i] = Integer.parseInt(values[i]);

}

Node root = construct(arr);

// write your code here

int d = diameter(root);

System.out.println(dia);

}

}

Code : 2  
package Topic\_15\_Trees;

import java.io.\*;

import java.util.\*;

public class F\_GenericTree {

private static class Node {

int data;

ArrayList<Node> children = new ArrayList<>();

}

public static void display(Node node) {

String str = node.data + " -> ";

for (Node child : node.children) {

str += child.data + ", ";

}

str += ".";

System.out.println(str);

for (Node child : node.children) {

display(child);

}

}

public static int size(Node node) {

int s = 0;

for (Node child : node.children) {

s += size(child);

}

return s;

}

public static int max(Node node) {

int m = Integer.MIN\_VALUE;

for (Node child : node.children) {

int cm = max(child);

m = Math.max(m, cm);

}

m = Math.max(m, node.data);

return m;

}

public static int height(Node node) {

int h = -1;

for (Node child : node.children) {

int ch = height(child);

h = Math.max(h, ch);

}

h += 1;

return h;

}

// Generic Tree - Traversals (pre-order, Post-order)

public static void traversals(Node node) {

System.out.println("Node Pre " + node.data);

for (Node child : node.children) {

System.out.println("Edge Pre " + node.data + "--" + child.data);

traversals(child);

System.out.println("Edge Post " + node.data + "--" + child.data);

}

System.out.println("Node Post " + node.data);

}

public static void levelOrder(Node root) {

Queue<Node> queue = new ArrayDeque<Node>();

queue.add(root);

while (queue.size() > 0) {

// r,p,a

Node temp = queue.remove();

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

for (Node child : temp.children) {

queue.add(child);

}

}

System.out.println(".");

}

public static void levelOrderLinewise(Node root) {

Queue<Node> queue = new ArrayDeque<Node>();

Queue<Node> cqueue = new ArrayDeque<Node>();

queue.add(root);

while (queue.size() > 0) {

Node temp = queue.remove();

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

for (Node child : temp.children) {

cqueue.add(child);

}

if (queue.size() == 0) {

queue = cqueue;

cqueue = new ArrayDeque<>();

System.out.println("");

}

}

}

// Extra question

public static void levelOrderLinewiseZZ(Node node) {

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

stack.add(node);

Stack<Node> cstack = new Stack<>();

int level = 0;

while (stack.size() > 0) {

node = stack.pop();

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

if (level % 2 == 0) {

for (int i = 0; i < node.children.size(); i++) {

Node child = node.children.get(i);

cstack.push(child);

}

} else {

for (int i = node.children.size() - 1; i >= 0; i--) {

Node child = node.children.get(i);

cstack.push(child);

}

}

if (stack.size() == 0) {

stack = cstack;

cstack = new Stack<>();

level++;

System.out.println();

}

}

}

public static void mirror(Node node) {

for (Node child : node.children) {

mirror(child);

}

Collections.reverse(node.children);

}

public static void removeLeaves(Node node) {

for (int i = node.children.size() - 1; i >= 0; i--) {

Node child = node.children.get(i);

if (child.children.size() == 0) {

node.children.remove(i);

}

}

for (Node child : node.children) {

removeLeaves(child);

}

}

private static Node getTail(Node node) {

while (node.children.size() == 1) {

node = node.children.get(0);

}

return node;

}

public static void linearize(Node node) {

for (Node child : node.children) {

linearize(child);

}

while (node.children.size() > 1) {

Node lc = node.children.remove(node.children.size() - 1);

Node sl = node.children.get(node.children.size() - 1);

Node slt = getTail(sl);

slt.children.add(lc);

}

}

public static Node linearizeEfficient(Node node) {

if (node.children.size() == 0) {

return node;

}

Node lastChild = node.children.get(node.children.size() - 1);

Node lastKiTail = linearizeEfficient(lastChild);

while (node.children.size() > 1) {

Node slastChild = node.children.get(node.children.size() - 2);

Node slastKiTail = linearizeEfficient(slastChild);

slastKiTail.children.add(lastChild);

node.children.remove(node.children.size() - 1);

lastChild = slastChild;

}

return lastKiTail;

}

public static boolean findANodeInTree(Node node, int data) {

if (node.data == data) {

return true;

}

for (Node child : node.children) {

boolean fic = findANodeInTree(child, data);

if (fic == true) {

return true;

}

}

return false;

}

public static ArrayList<Integer> nodeToRootPath(Node node, int data) {

if (node.data == data) {

ArrayList<Integer> bres = new ArrayList<>();

bres.add(node.data);

return bres;

}

for (Node child : node.children) {

ArrayList<Integer> nodeToChildPath = nodeToRootPath(child, data);

if (nodeToChildPath.size() > 0) {

nodeToChildPath.add(node.data);

return nodeToChildPath;

}

}

return new ArrayList<>();

}

public static int LowestCommonAncestor(Node node, int d1, int d2) {

ArrayList<Integer> path1 = nodeToRootPath(node, d1);

ArrayList<Integer> path2 = nodeToRootPath(node, d2);

int i = path1.size() - 1;

int j = path2.size() - 1;

while (i >= 0 && j >= 0) {

if (path1.get(i) == path2.get(j)) {

i--;

j--;

} else {

break;

}

}

int lca = path1.get(i + 1);

return lca;

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

String[] values = br.readLine().split(" ");

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

arr[i] = Integer.parseInt(values[i]);

}

Node root = construct(arr);

linearize(root);

display(root);

}

public static Node construct(int[] arr) {

Node root = null;

Stack<Node> st = new Stack<>();

for (int i = 0; i < arr.length; i++) {

if (arr[i] == -1) {

st.pop();

} else {

Node t = new Node();

t.data = arr[i];

if (st.size() > 0) {

st.peek().children.add(t);

} else {

root = t;

}

st.push(t);

}

}

return root;

}

}

Code : 3  
package Topic\_15\_Trees;

import java.util.\*;

public class PracticeGenericTree {

public static class Node {

Node(int data) {

this.data = data;

}

public Node() {}

int data;

ArrayList<Node> children = new ArrayList<>();

}

private static Node construct(int[] arr) {

Stack<Node> s = new Stack<>();

Node root = new Node();

for (int data : arr) {

if (data == -1) {

s.pop();

} else {

Node n = new Node(data);

if (s.size() == 0) {

root = n;

s.push(n);

} else {

Node temp = s.peek();

temp.children.add(n);

s.push(n);

}

}

}

return root;

}

public static void display(Node node) {

// Print self and its children data

String str = node.data + " -> ";

for (Node child : node.children) {

str += child.data + ", ";

}

str += ".";

System.out.println(str);

// Then make a call for display each children

// faith is children know how to display of its children

for (Node child : node.children) {

display(child);

}

}

public static int size(Node node) {

// write your code here

int sz = 1;

for (Node n : node.children) {

sz += size(n);

}

return sz;

}

public static int max(Node node) {

int max = node.data;

for (Node n : node.children) {

int data = max(n);

if (data > max)

max = data;

}

return max;

}

public static int height(Node node) {

// write your code here

int height = -1;

for (Node n : node.children) {

int res = height(n);

if (res > height) {

height = res;

}

}

return height + 1;

}

public static void traversals(Node node) {

System.out.println("Node Pre " + node.data);

for (Node n : node.children) {

System.out.println("Edge Pre " + node.data + "--" + n.data);

traversals(n);

System.out.println("Edge Post " + node.data + "--" + n.data);

}

System.out.println("Node Post " + node.data);

}

public static void levelOrder(Node root) {

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

q.addLast(root);

while (q.size() > 0) {

Node first = q.removeFirst();

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

for (Node ch : first.children) {

q.addLast(ch);

}

}

System.out.print(".");

System.out.println(".");

}

public static void levelOrderLinewise(Node root) {

Queue<Node> queue = new ArrayDeque<Node>();

Queue<Node> cqueue = new ArrayDeque<Node>();

queue.add(root);

while (queue.size() > 0) {

Node temp = queue.remove();

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

for (Node child : temp.children) {

cqueue.add(child);

}

if (queue.size() == 0) {

queue = cqueue;

cqueue = new ArrayDeque<>();

System.out.println("");

}

}

}

public static void levelOrderZigZag(Node node) {

Stack<Node> ms = new Stack<>();

Stack<Node> cs = new Stack<>();

ms.add(node);

int level = 1;

while (ms.size() >= 0) {

node = ms.pop();

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

if (level % 2 == 1) {

for (int i = 0; i < node.children.size(); i++) {

cs.push(node.children.get(i));

}

} else {

for (int i = node.children.size() - 1; i >= 0; i--) {

cs.push(node.children.get(i));

}

}

if (ms.size() == 0) {

ms = cs;

cs = new Stack<>();

level++;

System.out.println();

}

}

}

public static void mirror(Node node) {

// write your code here

for (Node child : node.children) {

mirror(child);

}

Collections.reverse(node.children);

}

public static void removeLeaves(Node node) {

// write your code here

for (int i = node.children.size() - 1; i >= 0; i--) {

Node child = node.children.get(i);

if (child.children.size() == 0) {

node.children.remove(i);

}

}

for (Node i : node.children) {

removeLeaves(i);

}

}

public static void linearize(Node node) {

for (int i = node.children.size() - 1; i >= 1; i--) {

Node last = node.children.remove(i);

Node secondLast = node.children.get(i - 1);

secondLast.children.add(last);

}

for (Node n : node.children) {

linearize(n);

}

}

public static boolean find(Node node, int data) {

// write your code here

if (node == null) {

return false;

}

boolean result = false;

if (node.data == data) {

return true;

}

for (Node item : node.children) {

result = find(item, data);

if (result)

break;

}

return result;

}

public static ArrayList<Integer> nodeToRootPath(Node node, int data) {

if (node.data == data) {

ArrayList<Integer> list = new ArrayList<>();

list.add(node.data);

return list;

}

for (Node item : node.children) {

ArrayList<Integer> list = nodeToRootPath(item, data);

if (list.size() > 0) {

list.add(node.data);

return list;

}

}

return new ArrayList<Integer>();

}

public static int lca(Node node, int d1, int d2) {

// write your code here

ArrayList<Integer> path1 = nodeToRootPath(node, d1);

ArrayList<Integer> path2 = nodeToRootPath(node, d2);

int i = path1.size() - 1;

int j = path2.size() - 1;

while (i >= 0 && j >= 0) {

if (path1.get(i) == path2.get(j)) {

i--;

j--;

} else {

break;

}

}

int lca = path1.get(i + 1);

return lca;

}

public static int distanceBetweenNodes(Node node, int d1, int d2) {

ArrayList<Integer> one = nodeToRootPath(node, d1);

ArrayList<Integer> two = nodeToRootPath(node, d2);

int i = one.size() - 1, j = two.size() - 1;

while (i >= 0 && j >= 0 && one.get(i) == two.get(j)) {

i--;

j--;

}

i++;

j++;

return i + j;

}

public static boolean IsSymmetric(Node node) {

// write your code here

return areMirror(node, node);

}

private static boolean areMirror(Node node1, Node node2) {

if (node1.children.size() != node2.children.size()) {

return false;

}

boolean res = true;

for (int i = 0, j = node2.children.size() - 1; i < node1.children.size()

&& j >= 0; i++, j--) {

res = areMirror(node1.children.get(i), node2.children.get(j));

if (res == false) {

break;

}

}

return res;

}

static int max = Integer.MIN\_VALUE, min = Integer.MAX\_VALUE, size = 0, height = -1;

public static void multiSolver(Node node, int depth) {

size++;

if (node.data > max) {

max = node.data;

}

if (node.data < min) {

min = node.data;

}

if (depth > height) {

height = depth;

}

for (Node item : node.children) {

multiSolver(item, depth + 1);

}

// height += 1;

}

static Node predecessor;

static Node successor;

static int state = 0;

public static void predecessorAndSuccessor(Node node, int data) {

// write your code here

if (state == 0) {

if (node.data == data) {

state = 1;

} else {

predecessor = node;

}

} else if (state == 1) {

successor = node;

state = 2;

}

for (Node c : node.children) {

predecessorAndSuccessor(c, data);

}

}

static int ceil;

static int floor;

public static void ceilAndFloor(Node node, int data) {

if (node.data > data) {

if (node.data < ceil) {

ceil = node.data;

}

}

if (node.data < data) {

if (node.data > floor) {

floor = node.data;

}

}

for (Node child : node.children) {

ceilAndFloor(child, data);

}

}

public static int kthLargest(Node node, int k) {

int i = 0;

floor = Integer.MIN\_VALUE;

int factor = Integer.MAX\_VALUE;

while (i < k) {

ceilAndFloor(node, factor);

factor = floor;

floor = Integer.MIN\_VALUE;

i++;

}

return factor;

}

static int mSum = Integer.MIN\_VALUE;

static int mSumNode = Integer.MIN\_VALUE;

public static int nodeWithMaximumSubtreeSum(Node node) {

int sum = node.data;

for (Node child : node.children) {

int cstSum = nodeWithMaximumSubtreeSum(child);

sum += cstSum;

}

if (sum > mSum) {

mSum = sum;

mSumNode = node.data;

}

return sum;

}

static int dia = 0;

public static int diameter(Node node) {

// write your code here

int dch = -1;

int sdch = -1;

for (Node n : node.children) {

int ch = diameter(n);

if (ch >= dch) {

sdch = dch;

dch = ch;

} else if (ch >= sdch) {

sdch = ch;

}

}

int cand = dch + sdch + 2;

if (cand > dia) {

dia = cand;

}

dch += 1;

return dch;

}

static class Pair {

Node node;

int state;

Pair(Node node, int state) {

this.node = node;

this.state = state;

}

}

public static void IterativePreandPostOrder(Node node) {

Pair p = new Pair(node, -1);

Stack<Pair> st = new Stack<>();

st.push(p);

String preOrder = "";

String postOrder = "";

while (st.size() > 0) {

Pair top = st.peek();

if (top.state == -1) {

preOrder += top.node.data + " ";

top.state++;

} else if (top.state >= 0 && top.state < top.node.children.size()) {

Pair cp = new Pair(top.node.children.get(top.state), -1);

st.push(cp);

top.state++;

} else {

postOrder += top.node.data + " ";

st.pop();

}

}

System.out.println(preOrder);

System.out.println(postOrder);

}

public static boolean areSimilar(Node n1, Node n2) {

// write your code here

if (n1.children.size() != n2.children.size()) {

return false;

}

boolean res = true;

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

res = areSimilar(n1.children.get(i), n2.children.get(i));

if (!res) {

break;

}

}

return res;

}

public static void main(String[] args) {

String[] values =

"10 20 50 -1 60 -1 -1 30 70 -1 80 110 130 150 170 -1 -1 -1 -1 120 140 160 180 190 -1 -1 -1 -1 -1 -1 90 -1 -1 40 100 -1 -1 -1"

.split(" ");

int[] arr = new int[values.length];

for (int i = 0; i < arr.length; i++) {

arr[i] = Integer.parseInt(values[i]);

}

int arr1[] = {10, 20, 50, -1, 60, -1, -1, 30, 70, -1, 80, 110, -1, 120, -1, -1, 90, -1, -1,

40, 100, -1, -1, -1};

Node root = construct(arr1);

// display(root);

// int size = size(root);

// System.out.println("Size : " + size);

// int max = max(root);

// System.out.println("Maximum : " + max);

// int height = height(root);

// System.out.println("Height : " + height);

// System.out.println("Level Order");

// levelOrder(root);

// System.out.println("Level Order Line Wise");

// levelOrderLinewise(root);

// int d = diameter(root);

// System.out.println("Diameter :" + dia);

// predecessorAndSuccessor(root, 90);

// System.out.println("Predecessor : " + predecessor.data + " Successor : " +

// successor.data);

// multiSolver(root, 0);

// System.out.println("Min: " + min + " Max: " + max + " Height: " + height + " Size: " +

// size);

// ceilAndFloor(root, 65);

// System.out.println(ceil + " " + floor);

IterativePreandPostOrder(root);

boolean res = areSimilar(root, root);

System.out.println(res);

System.out.println(kthLargest(root, 3));

}

}