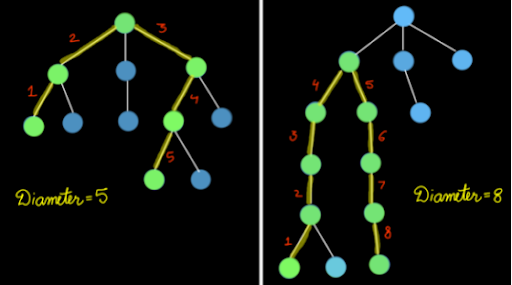
**1. Problem Statement:**

In this problem you are given a partially written GenericTree class. All you need to do is to find and print the diameter of the tree. Input is managed for you.

****

The diameter is defined as the maximum number of edges between any two nodes in the tree. In the left tree, diameter is 5 and in the right tree, the diameter is 8. To be noted that the diameter may not necessarily include the root node as shown in the right tree. Check the question video for more clarity.

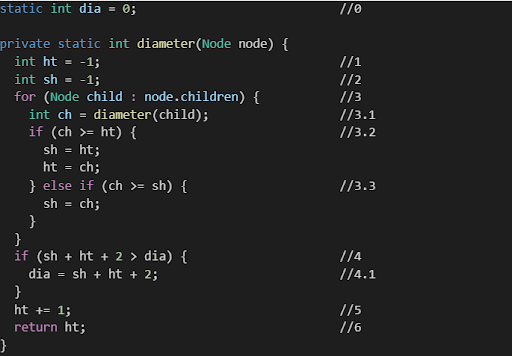
For more clarity of the question, watch the question video

Play Video

Approach:

We need to maximize the number of edges between any two nodes to calculate diameter. To be noted that to maximize the number of edges we have to always consider the leaf nodes. Now we wish to find a diameter that passes through our current node. This can be found by adding the deepest subtree and second deepest subtree and adding 2 to their sum. Getting the deepest and second deepest subtree ensures that we are taking the maximum possible edges from the current node and 2 is added to link both the leaves. Now we can recurse this approach for every node in our tree as our diameter need not always pass through the root node. So at each node, we calculate the diameter from the current node and compare it with the global maximum and then we return the height of our subtree which can, later on, be used by any ancestor nodes to calculate their diameter and height.

Let's try to Code this!

****

1 - We define a static variable dia, which stores the value of greatest diameter found so far. Inside the function diameter we basically calculate the diameter and update it if required but return the height. And as a parameter, a Node is passed. 2 - To begin with, we initialize a variable ht (height) to -1, which will further get updated with the largest value of height present in a node. 3 - In addition, we initialize another variable sh (second height) to -1, which will further get updated with the second largest value of height present in a node. 3.1 - Furthermore, a for loop is run to access all children of the node. Inside this for loop, we calculate the height of the child node at which the "for" loop is iterating and store it in variable ch (current height). 3.2 - After that we compare this ch with ht, if ch is greater than ht then we update sh with ht and then ht with ch. 3.3 - In case, "if" condition is false then we further check if ch is greater than at least sh, if this condition is true we just update sh with ch. 4 - Once we are out of the for loop, we compare values of dia (largest diameter) and diameter of current node (ht + sh + 2). If the value of diameter of the current node is greater than the value of dia, then we update dia with (ht + sh + 2). 5 - Furthermore we update ht by adding 1 to it, which gives us the height of the node. 6 - Finally we return the updated height.

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

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 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 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);

IterativePreandPostOrder(root);

}

}

We hope that this article was helpful. If somehow you are finding it difficult to understand this problem then we advise you to watch our Lecture video :-

For more clarity of the question, watch the question video

Play Video

Trust me it will get much easier to understand this problem once you watch the solution video.

You can contact us via our website. Doubts, suggestions and feedback are always welcomed. It is also advised that you follow the sequence of modules and questions which is there on our website.

Keep practicing more and more problems daily. Meditate enough on each problem and each step. All the best for an adventurous future! Happy Coding!

Analysis :

Time Complexity: O(n)

The time complexity for the function is linear as tree traversal is involved which is linear in terms of time complexity.

Space Complexity: O(h)

The space complexity for the function is proportional to the height of the generic tree due to the recursion stack.