**1. Problem Statement:**

1. You are given a partially written GenericTree class. 2. You are required to complete the body of areMirror function. The function is expected to check if the two trees passed to it are mirror images of each other in shape (data not to be checked, just the shape of the tree). 3. Input and Output is managed for you.

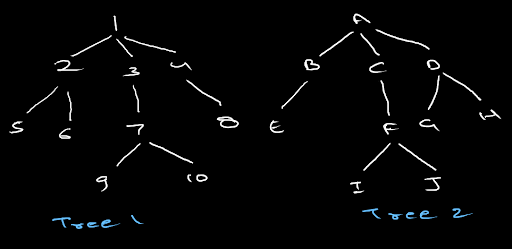
For more clarity of the question, watch the question video

Play Video

Note: If you have not tried enough to come up with logic, then we recommend you to first spend an hour or so doing it, else read only the logic used, take it as a hint and try the problem again with the same logic.

**2. Problem Discussion:**

We have to write a function that is static and two root nodes of the generic trees are passed as a parameter.we have to return true if both the generic trees are mirror in shape and return false if they are not mirror in shape. First of all we have to make in mind that we have to check only the shape of the tree, not the data of specific nodes. For eg- The image shown below represents the two generic trees we have to check these two generic trees are mirror in shape or not.

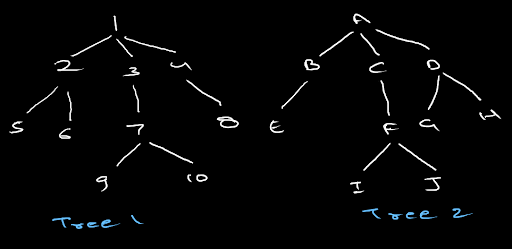
****

You may refer to the question video if you have any doubts regarding the question. We recommend you try to solve this problem on your own first and then go to the solution.

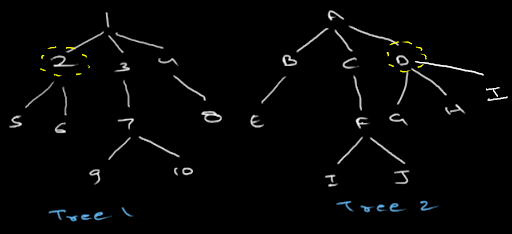
**3. Approach:**

Checking the Child Nodes of the Root Node:

Consider the example of the below image.

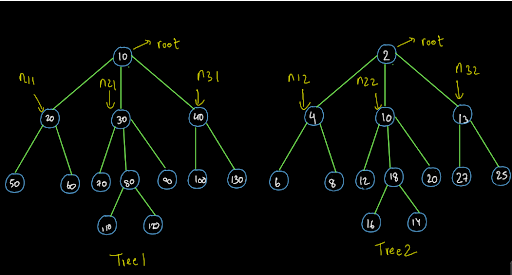
****

At first is it necessary to start checking whether these trees are mirror in shape or not? No because, root of both these trees should have an equal number of child nodes.so the very first step would be to check whether the number of child nodes of both the root nodes of both the trees are same or not.But is it valid for only the root node? Well, this is valid for every node of tree 1 and tree 2 in reverse order. For eg: In the image below, node 2 and node D do not have an equal number of children so tree 1 and tree 2 are not symmetric.

****

High-Level Thinking:

Let us have a look at the diagram below:

****

We have already established that the number of child nodes of the root node should be the same for us to begin the procedure. Now, after that we can break our problem into smaller problems and solve it using recursion. As shown above we have mentioned the 3 child nodes of tree1 as n11 (node one of tree 1), n21 (node two of tree 1) and n31 (node 3 of tree 1) respectively. Similarly we have named the corresponding nodes in the second tree as n12, n22 and n32.

The trees will be the mirror in shape if the sub-trees from n11 and n21 are the same and the sub-trees from n32 and n22 are the same respectively. So, let us now break this procedure down to:

Expectation: We expect that the function areMirror(node1,node2) will return true if the trees are mirror in shape and it will return false if they are not mirror in shape. Faith: We have a faith on the recursion that recursion can tell us whether the sub-trees of the same given trees are mirror in shape or not. Relation: We will first find whether the number of child nodes of the root nodes of both the trees is same or not. If it is then we will check whether the sub-trees of both the trees in reverse order are mirror in shape or not. If we find at any node that the shape is not mirror, we will return false otherwise if we complete the entire procedure and find mirror shape for every respective node then we will return true. Now that we have done the high level thinking, let us write the code for the above procedure.

**4. Pseudo Code:**

1• Check Number of Child Nodes: For the trees to be of the mirror shape the number of child nodes of the root nodes of both the trees must be the same. If they are not the same, return false otherwise continue the procedure. 2• High Level Thinking: After checking the number of nodes, we must ensure that the subtree which is starting from first child node must be the mirror in shape as the last child node subtree of the other tree. So, we will use our faith in recursion to achieve it by calling the function areMirror(node1,node2) for all the children of both the trees. 3• Low Level Thinking:We don't have to find a base case as such. We will do the low level thinking for understanding the working of the code that we wrote using the high level thinking in a better way.

**5. Implementation:**

Note: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code

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;

}

public static int size(Node node) {

int s = 0;

for (Node child : node.children) {

s += size(child);

}

s += 1;

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;

}

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

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

return false;

}

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

Node c1 = n1.children.get(i);

Node c2 = n2.children.get(n2.children.size() - 1 - i);

if (areMirror(c1, c2) == false) {

return false;

}

}

return true;

}

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

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

int n1 = Integer.parseInt(br.readLine());

int[] arr1 = new int[n1];

String[] values1 = br.readLine().split(" ");

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

arr1[i] = Integer.parseInt(values1[i]);

}

Node root1 = construct(arr1);

int n2 = Integer.parseInt(br.readLine());

int[] arr2 = new int[n2];

String[] values2 = br.readLine().split(" ");

for (int i = 0; i < n2; i++) {

arr2[i] = Integer.parseInt(values2[i]);

}

Node root2 = construct(arr2);

boolean mirror = areMirror(root1, root2);

System.out.println(mirror);

}

}

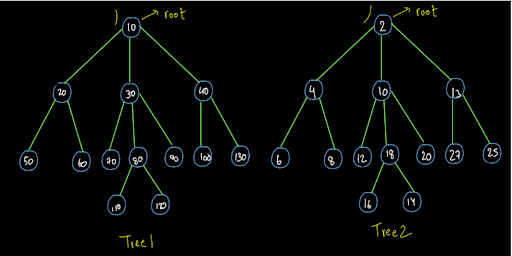
Dear reader, we hope that you got the procedure as well as the code. If you have any doubts regarding the above procedure or the code, you may refer to the solution video to understand the procedure and the code in detail. Now let us do the low-level analysis of this code and understand how this code is working.

For more clarity of the question, watch the question video

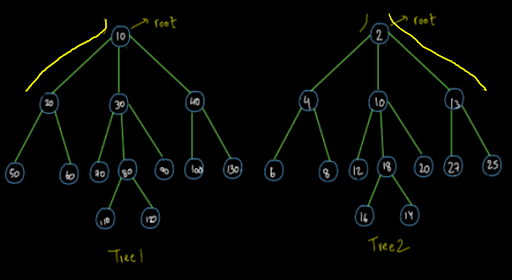
Play Video

**6. Low-Level-Analysis:**

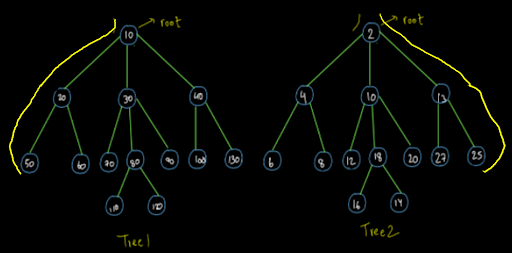
We recommend you have the code by your side to understand this low level analysis. We are initially at the root nodes of both the trees. We check whether the number of child nodes of the root nodes of both the trees is the same or not.

****

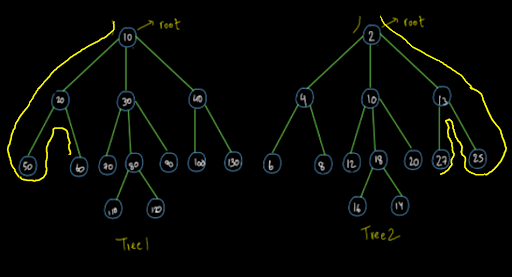
Since they were the same, we entered the loop inside the code. Here we move to the first child of tree 1 and the last child of tree 2.

****

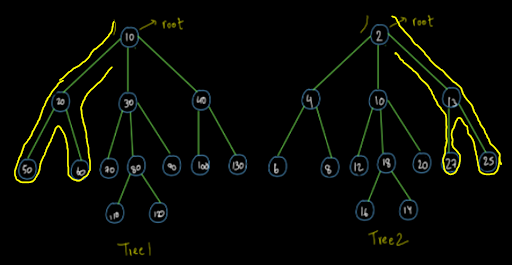
Since both of them also have the same number of nodes therefore we enter the loop in the code and we move to the first child of node of tree1 and the last child of node of tree 2.

****

Since both of them are leaf nodes, the number of nodes i.e. 0 is the same for both of them. Now we will not enter the loop as there is no child node for these nodes. Therefore, their code is completed. So, they will wipe out from the recursion stack and we will return to node (20) and node(13) respectively. Since we are still inside the loop for both these nodes, the call will be made for their second child nodes and we will head there in our Euler paths.

****

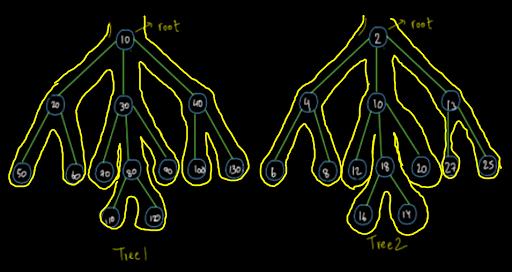
Again, they both are leaf nodes. So, we will return to node (20) and node (13) respectively. Since all its child nodes have been traversed, we will return back to the root nodes of both the trees.

****

So, dear reader, we now request you to complete this procedure further in the same way as taught above. If you still have any doubts regarding this procedure, you may refer to the solution video to clear all your doubts. For the above trees, you should have completed the Euler path and found that both of them are same in shape.

For more clarity of the question, watch the question video

Play Video

****

Now let us analyze the time and space complexity of the above procedure.

**7. Analysis:**

Time Complexity:

O(n) The time complexity will be O(n) as we have traversed every node of both the trees if they are in the mirror shape.

Space Complexity:

The space complexity is O(1) as we have not used any extra data structure. As usual if we consider the recursion space then the space complexity will be O(logn) as the max height of the stack at any point will be the same as the height of the tree i.e. O(logn).