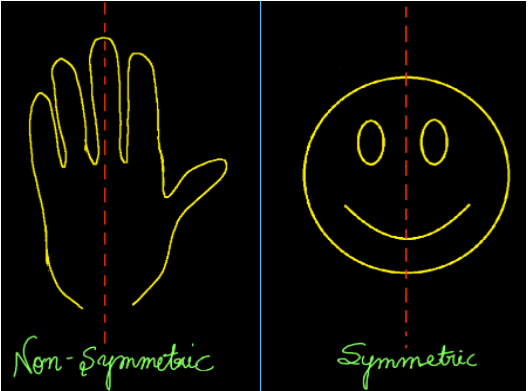
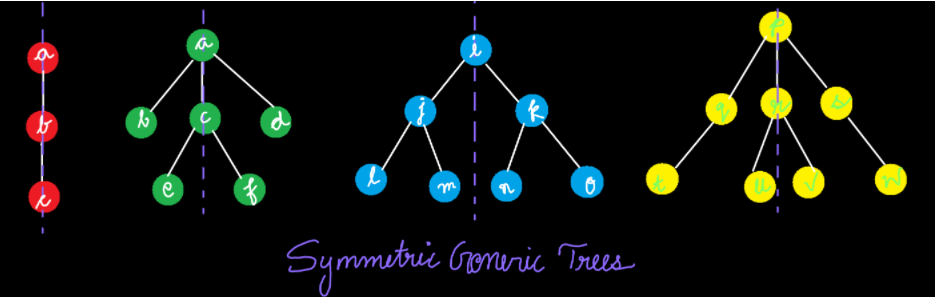
**1. Problem Discussion :**

In this problem you are given a partially written GenericTree class. All you need to do is to complete the body of the IsSymmetric function. Input and Output is managed for you. The function is expected to check if the tree is symmetric, if it is symmetric then return true otherwise return false. For knowing symmetricity, think of face and palm. Face is symmetric while the palm is not.

****

Also, we check only the symmetricity of shape and not content in the tree. It means that nodes may have different data but the shape of the tree should be such that if a line is drawn through the centre of the tree then it should act like a mirror. Some examples of symmetric generic trees are shown in the figure given below.

****

Check the question video for clarity.

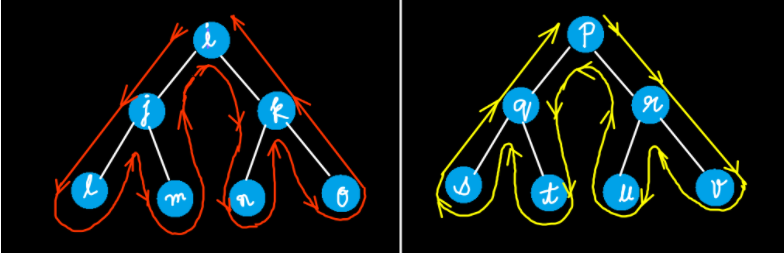
For more clarity of the question, watch the question video

Play Video

**2. Approach :**

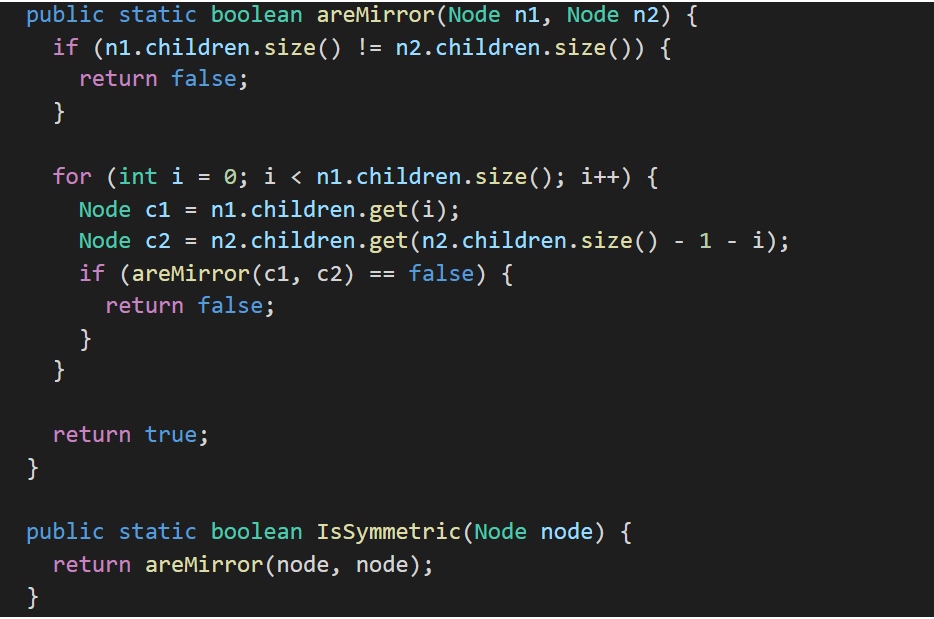
The first and foremost question that we need to tackle here is, how do we check symmetricity?

When we fold the tree along its vertical axis passing through the root node, if the left half overlaps the right half then the tree is said to be symmetric. If we carefully observe then alternatively we can also say that a tree is symmetric whenever the left half of the tree is the mirror image of the right half of the tree. This happens since when we say that we fold the tree, this means that we are inverting the tree of the left half and then superimposing on the right half, which is the same as having a mirror image superimposed. Hence our problem is now reduced to only finding whether our left subtree is the mirror image of the right subtree or not, alternatively, we can also check for whether our tree is the mirror image of itself or not. If a tree is a mirror image of itself then it will surely be symmetric.

****

How to check whether a tree is a mirror image of itself or not? We have already discussed that in a previous problem, Are Trees Mirror. Check it out for any kind of difficulty.

Let's try to code this.

****

What did we even do?

Well we simply wrote the code of the areMirrors() function as it is, that we understood in the previous problem. For any doubt or revision, it is advised that you watch the video lecture on " Are Trees Mirror".

For more clarity of the question, watch the question video

Play Video

In IsSymmetric(Node node), a single node is taken as a parameter (basically the root of the input generic tree). Inside this function we return the function areMirror(). We do so because the answer that areMirror() returns is our final answer also. If areMirror() returns true, then the tree is Mirror image of itself and therefore symmetric too. So IsSymmetric() will also return true. And if areMirror() returns false, then the tree is not Mirror image of itself and therefore not symmetric. So IsSymmetric() will also return false. Since we have to check whether the given tree is a mirror image of itself or not, we pass the same node in both the arguments of areMirror(node, node).

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 boolean IsSymmetric(Node node) {

return areMirror(node, node);

}

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);

boolean sym = IsSymmetric(root);

System.out.println(sym);

// display(root);

}

}

**3. Analysis:**

Time Complexity: O(n) We travel to all the elements of the tree. Every node is processed individually.

Space Complexity: O(1) No extra space is used therefore space complexity remains constant. But when the function runs at that time, space acquired by the memory stack will be O(n).

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 video lecture of this problem.