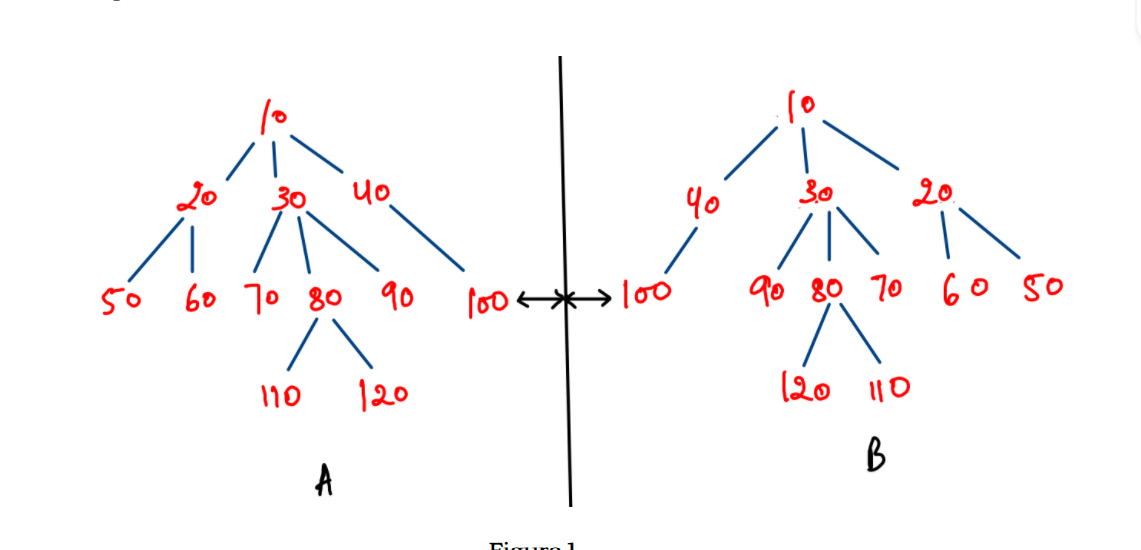
**1. Problem Discussion :**

You are required to complete the body of the mirror function. The function is expected to create a mirror image of the tree. You can see in figure 1, if the given tree is denoted by A then, its mirror image is depicted as B.

****

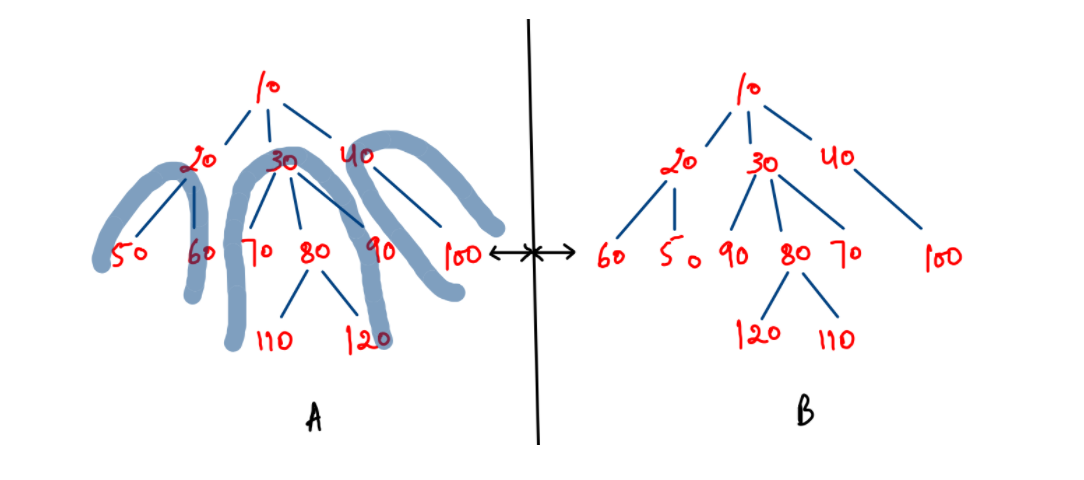
Each element of the mirror image is the same distance behind the mirror as each element of the object is in front of the mirror.

**2. Approach :**

We go through 3 steps of High Level Thinking.

● Set Expectation : We expect that if we input a generic tree, then the output should be its mirror tree as shown in figure 1 .

● Build Faith : We must have faith that if our code can give us an output for the entire generic tree, then it can definitely give as an output its subtree.Those subtrees of that generic tree are considered where each root of that sub tree is a child of the root of the original generic tree. As shown in figure 2, we have faith that we can make mirror trees for the 3 highlighted sub trees with roots 20, 30 and 40.

****

The output of those sub trees is seen in Figure 2(B). You just need to believe. Don't focus on "HOW" that will happen.

● Expectation meets Faith : We see that figure 2 is not yet complete. The final output tree must look like Figure 1(B). For that, we need to change the positions of the children of the root node in Figure 2(B). The families of these sub trees should get reversed for getting the desired output. We had already mirrored 20, 30 and 40. By reversing their families, the family of 10 will get mirrored too.

We'll now see how our above discussion could be coded. It is fairly simple and easy to understand.

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 void mirror(Node node) {

for (Node child : node.children) { //1

mirror(child);

}

Collections.reverse(node.children); //2

}

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);

display(root);

mirror(root);

display(root);

}

}

**3. CODE DISCUSSION**

● We recursively call the mirror function on all the children of the root of the generic tree which would give us an output as shown in figure 2(B). This causes the families of 20, 30 and 40 to be mirrored.

● Since, the family of root 10 needs to be mirrored too, we reverse the families of its children 20,30 and 40 by using Collections.reverse() function.

This will give our desired output of Figure 1(B). To see the dry run of this code, we highly request you to watch the video, "Mirror of Generic Tree".

**4. Analysis:**

Time Complexity: O(n) The time complexity for the function is linear as we post traversing the tree.

Space Complexity: O(nlogn) The space complexity for the function is equal to the height of the tree due to the recursion stack.

With this we conclude this problem. See you in the next question.