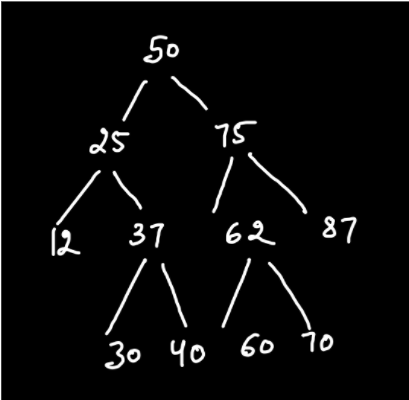
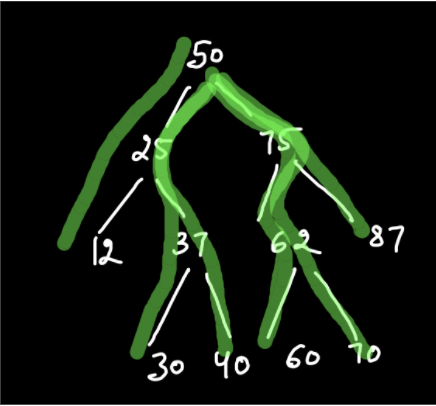
**1. Problem Discussion :**

You are given a value lo and a value hi. You are required to complete the body of pathToLeafFromRoot function which is expected to print all paths from root to leaves which have sum of nodes in range from lo to hi (both inclusive). The elements in the path should be separated by spaces. Each path should be in a separate line. Say, we are given the input binary tree as shown in figure 1.

****

Then, the total paths from root to leaf are 6 which is equal to the total number of leaf nodes. These paths can be depicted by figure 2.

****

We wish to print those paths whose sum of nodes ranges between lo and hi (both inclusive). Let, the "lo" and "hi" values be 150 and 250 respectively. Then the required paths will be: 50-25-37-40 with sum=152 ; 150<=152<=250 50-75-62-60 with sum=247 ; 150<=247<=250 50-75-87 with sum=212 ; 150<=212<=250

**2. Approach :**

Since you have done many questions on Binary Trees, we directly write the code and then simultaneously discuss it. The code is as follows:

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static class Node {

int data;

Node left;

Node right;

Node(int data, Node left, Node right) {

this.data = data;

this.left = left;

this.right = right;

}

}

public static class Pair {

Node node;

int state;

Pair(Node node, int state) {

this.node = node;

this.state = state;

}

}

public static Node construct(Integer[] arr) {

Node root = new Node(arr[0], null, null);

Pair rtp = new Pair(root, 1);

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

st.push(rtp);

int idx = 0;

while (st.size() > 0) {

Pair top = st.peek();

if (top.state == 1) {

idx++;

if (arr[idx] != null) {

top.node.left = new Node(arr[idx], null, null);

Pair lp = new Pair(top.node.left, 1);

st.push(lp);

} else {

top.node.left = null;

}

top.state++;

} else if (top.state == 2) {

idx++;

if (arr[idx] != null) {

top.node.right = new Node(arr[idx], null, null);

Pair rp = new Pair(top.node.right, 1);

st.push(rp);

} else {

top.node.right = null;

}

top.state++;

} else {

st.pop();

}

}

return root;

}

public static void display(Node node) {

if (node == null) {

return;

}

String str = "";

str += node.left == null ? "." : node.left.data + "";

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

str += node.right == null ? "." : node.right.data + "";

System.out.println(str);

display(node.left);

display(node.right);

}

public static void pathToLeafFromRoot(Node node, String path, int sum, int lo, int hi) {

if (node == null) { //1

return;

}

if (node.left == null && node.right == null) { //2

sum += node.data; //3

if (sum >= lo && sum <= hi) { //4

System.out.println(path + node.data);

}

return;

}

pathToLeafFromRoot(node.left, path + node.data + " ", sum + node.data, lo, hi); //5

pathToLeafFromRoot(node.right, path + node.data + " ", sum + node.data, lo, hi);

}

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

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

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

Integer[] arr = new Integer[n];

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

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

if (values[i].equals("n") == false) {

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

} else {

arr[i] = null;

}

}

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

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

Node root = construct(arr);

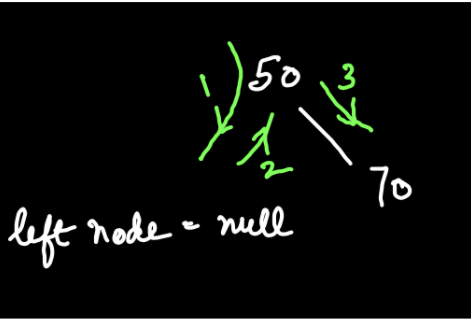
pathToLeafFromRoot(root, "", 0, lo, hi);

}

}

**3. Code Discussion:**

1• BASE CASE : If the present node is null then we return the function. It is possible in the case shown in Figure 3.

****

Here 50 has only one child where right node= 70. Since the left node is null, therefore we return to the previous parent node so that it can traverse to the right node.

2• We check whether we have reached the leaf node by checking if both its children are null.

3• We need to add the current node to the sum because it wasn't included in the previous recursion call.

4• Now we check whether the sum of the given path lies in the given range between lo and hi (both inclusive). If it does then the path is printed by adding the current node to the path because we didn't include that node when we made our previous recursion call.

5• We call the recursive function on the left node of the current node. The data of that node is included in the path so far and arithmetically added to the sum so far. Similarly , it is called on the right node of the current node. The data of that node is included in the path so far and arithmetically added to the sum so far.