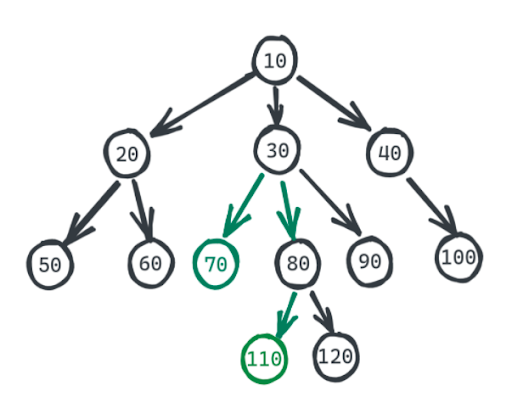
**1. Problem Discussion:**

1• You are given a partially written GenericTree class.

2• You are required to complete the body of distanceBetweenNodes function. The function is expected to return the distance (in terms of number of edges) between two nodes in a generic tree.

****

If we are asked to find the distance between 70 and 110 we will calculate the number of edges between them i.e 3

**2. Approach :**

Let's understand what distance means in a Tree. In a tree, distance refers to the edges between them. Now in a tree, there is always one path from one node to another.

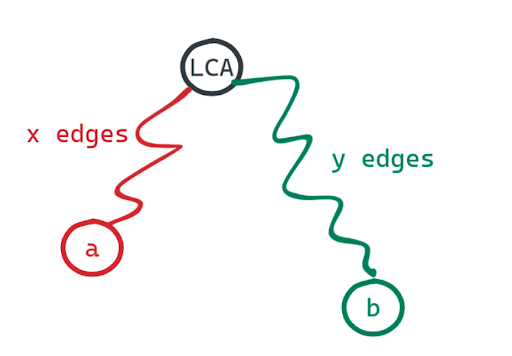
Also, the path will have the LCA of those two nodes.

1• In the above example, the LCA of 70 and 110 is 30. If you don't know what an LCA is then I suggest you solve that problem first. As a small recap, LCA is the lowest common ancestor. In the above example, 10 is also a common ancestor but not the lowest.

2• So, for this problem too we will re-use a function we discussed previously nodeToRootPath.

3• The nodeToRootPath for 70 is [70, 30, 10]. Let this array be p1[]. The nodeToRootPath for 110 is [110, 80, 30, 10]. Let this array be p2[].

4• We will find the largest smallest i and j such that p1[i] == p2[j]. In our case i=1 and j=2. Now can you tell what is the significance of p1[i] or p2[j]? Yes, it is the LCA.

****

5• Now if one node is x edges away from the LCA and the other one is y edges away from LCA then the distance between these nodes will be x+y. We know in the case of a Generic Tree there exists a unique LCA for every two nodes. And hence the distance between those two nodes will also be unique.

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 ArrayList<Integer> nodeToRootPath(Node node, int data) {

if (node.data == data) {

ArrayList<Integer> path = new ArrayList<>();

path.add(node.data);

return path;

}

for (Node child : node.children) {

ArrayList<Integer> ptc = nodeToRootPath(child, data);

if (ptc.size() > 0) {

ptc.add(node.data);

return ptc;

}

}

return new ArrayList<>();

}

public static int lca(Node node, int d1, int d2) {

ArrayList<Integer> p1 = nodeToRootPath(node, d1);

ArrayList<Integer> p2 = nodeToRootPath(node, d2);

int i = p1.size() - 1;

int j = p2.size() - 1;

while(i >= 0 && j >= 0 && p1.get(i) == p2.get(j)){

i--;

j--;

}

return p1.get(i + 1);

}

public static int distanceBetweenNodes(Node node, int d1, int d2){

// write your code here

ArrayList<Integer> a1=nodeToRootPath(node,d1);

ArrayList<Integer> a2=nodeToRootPath(node,d2);

int i=a1.size()-1;

int j=a2.size()-1;

while(i>=0 && j>=0 && a1.get(i)==a2.get(j))

{

i--;

j--;

}

i++;

j++;

return i+j;

}

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]);

}

int d1 = Integer.parseInt(br.readLine());

int d2 = Integer.parseInt(br.readLine());

Node root = construct(arr);

int dist = distanceBetweenNodes(root, d1, d2);

System.out.println(dist);

// display(root);

}

}

**3. Pseudocode:**

1• Calculate node to root path for both the nodes.

2• Keep iterating from the end and the last equal items in the path of both the nodes becomes the LCA.

3• Now the distance between two nodes will be the distance of node 1 from LCA in addition to the distance of node 2 from LCA.

**4. Analysis:**

Time Complexity: O(n)

Finding the node in the entire tree to get the node to the root path takes O(n). Then, just traversing the node-to-root path (arrays) takes O(d) where d = depth of the node. In the worst case, d can be equal to n, hence total time complexity will be O(n) only.

Space Complexity: O(n)

We are storing the node to the root path so the total auxiliary size is O(n).