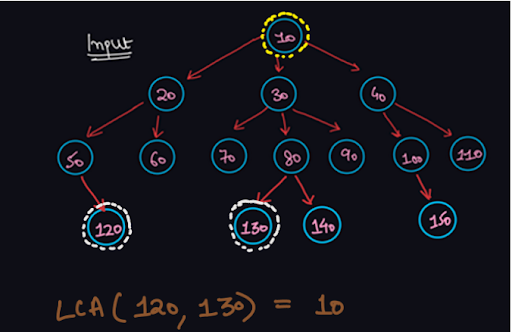
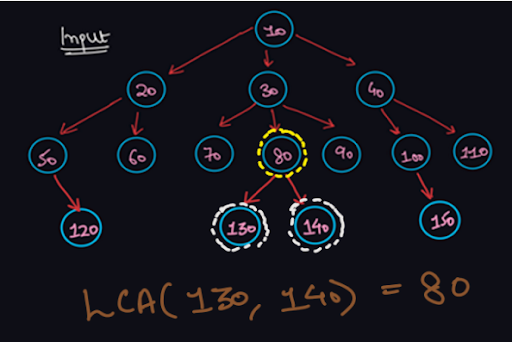
**1. Problem Discussion :**

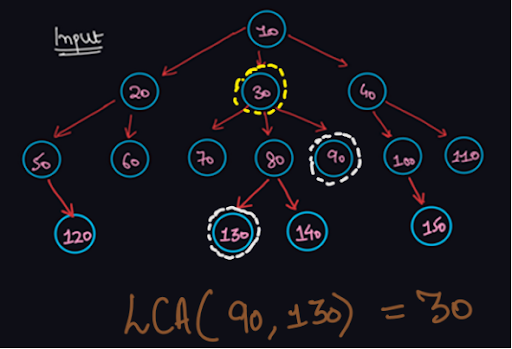
You are given a partially written GenericTree class. (Input and Output is managed for you.) You are required to complete the body of the lca function. The function is expected to return the lowest common ancestor of two data values that are passed to it. Lowest Common Ancestor is the node which is the common ancestor for both the nodes and is the closest to them (or farthest from root). It is guaranteed that both the nodes are present in the tree. Hence, there will always be the lowest common ancestor possible. Also, since the nodes of the tree have distinct values, there will always be a unique LCA. (Although the common ancestors are many, but the lowest of them will be only one). Examples:

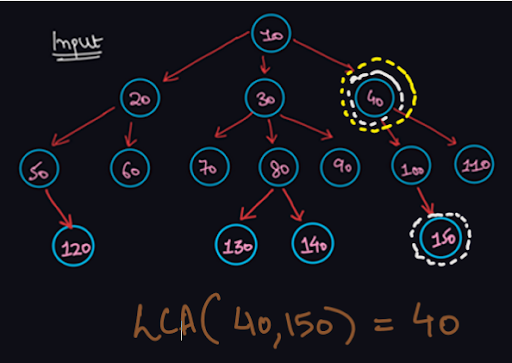
****

Some LCA queries:

****

****

****

****

Please watch the question video to understand what lca is.

For more clarity of the question, watch the question video

Play Video

Approach :

If we can get the node to the root path of both the elements in an array, then finding the lowest common ancestor will be really easy for us. By definition, the lowest common ancestor is the common ancestor between both nodes which is as far as possible from the root. Since the paths are stored in node-to-root order, i.e. root is at the last index of the array, hence we will start two pointers (i and j) from the last index of the two arrays. We will keep on decreasing i and j until we get an unequal element. Hence, the last equal element (arr[i + 1] or arr[j + 1]) in the node-to-root path will be our lowest common ancestor. Note: This solution can handle corner cases such as when one node is the ancestor of another node, or both nodes are same (equal), one node is the root node, etc.

Pseudo Code

Store node-to-root path of the first node in array p1. Store node-to-root path of the second node in array p2. Initialize two pointers i and j as p1.length - 1 and p2.length - 1 respectively. Decrement i and j until arr[i] and arr[j] become unequal. The lowest common ancestor will be arr[i + 1] (or arr[j + 1]). 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 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 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 lca = lca(root, d1, d2);

System.out.println(lca);

// display(root);

}

}

This code is written and explained by our team in the solution video. Do check it out to understand the concept completely.

For more clarity of the question, watch the question video

Play Video

Analysis:

Time Complexity:

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

Space Complexity:

We are storing node-to-root paths in arraylists. This will take O(n) auxiliary space.

Follow Up

Q) What if it was allowed that one of the nodes may not exist in the tree. We should return -1 as lca in this node. How will you modify the code? R) Getting a node-to-root path for a node which does not exist in the tree is not possible. It will be an empty array in this case. Hence, if we get an empty path for any node, we simply return -1 as lca does not exist in this case. Hope that you liked the article on Lowest Common Ancestor - Generic Tree. Subscribe to Pepcoding's youtube channel for more such amazing video content on Data Structures & Algorithms. You can suggest any improvements to the article on our telegram channel, or on the youtube channel's comment section.