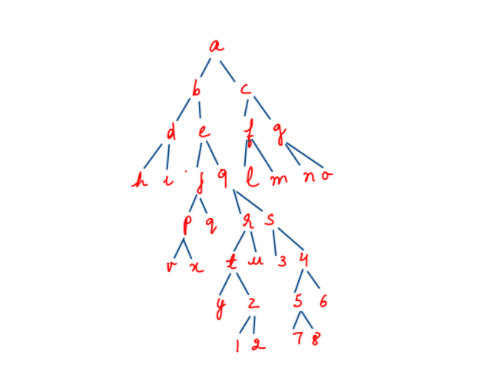
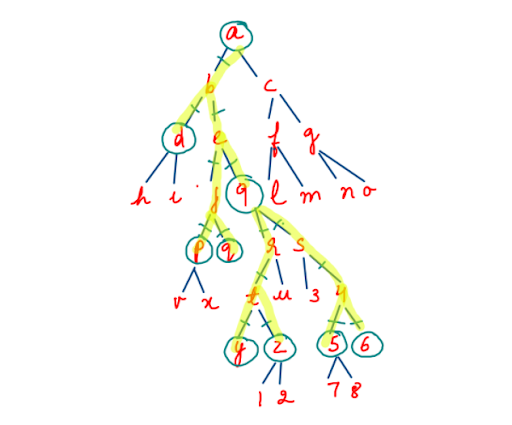
**1. Problem Discussion :**

You are given a value data and a value k. You are required to complete the body of printKNodesFar function. The function is expected to print all nodes which are k distance away in any direction from nodes with value equal to data. Say, we are given a tree as shown in figure 1.

****

Here, if the data=9 and k=3 then the paths shown in figure 2 are k=3 distance away from data=9.

****

Before starting the discussion, if you haven't watched the videos for "Find and Node To Root Path in Binary Tree '' and "Print K levels down", we suggest you watch them now.

For more clarity of the question, watch the question video

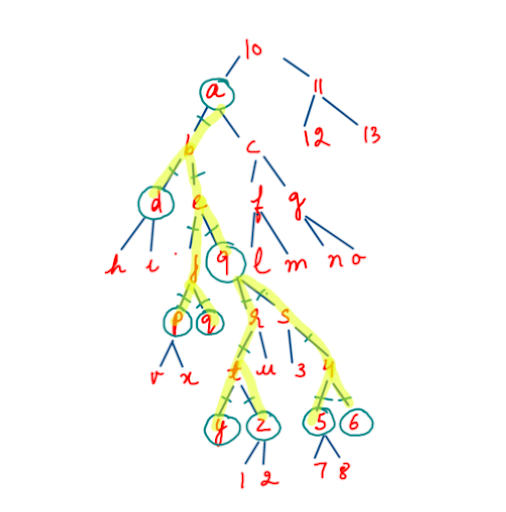
Play Video

For more clarity of the question, watch the question video

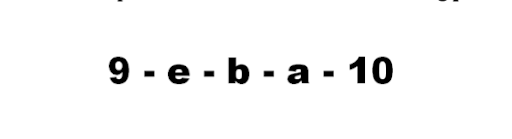
Play Video

Approach :

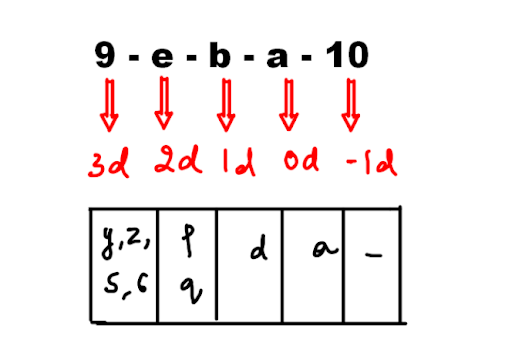
Let's make an addition in the given tree for explanation purposes.

****

Now 10 is the root of the tree. First we call the "Nodetorootpath" function which has been discussed previously in one of the questions and obtain the following path.

****

Then we go 3 down, 2 down, 1 down, 0 down and -1 down from the elements of this path respectively using the "Print k levels down" function which was discussed in the previous question. Meditate on the figure given below.

****

When we go 3 steps down from 9, we reach y, z, 5 and 6 nodes. When we go 2 steps down from e, we reach p and q. When we go 1 step down from b, we reach d. When we go 0 steps down from a, we reach the node itself i.e. a. When we go -1 steps down from 10, we reach nowhere. The above nodes can provide us the desired output. BUT. Notice that if we go 2 steps down from e, then we also reach r and s nodes which isn't desirable. Similarly, when we go 1 step down from b, then we also reach node e which again isn't desirable. How can we prevent these unwanted cases from getting printed? Answer: By using a "blocker" variable. We'll discuss it more after writing its code. Look at the code and its discussion given below to understand the solution better.

ConsoleJava

import java.io.\*;

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);

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRINT K NODES FAR\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static void printKNodesFar(Node node, int data, int k) {

ArrayList< Node> path = find(node, data); //1

for (int i = 0; i < path.size(); i++) { //2

printKLevelsDown(path.get(i), k - i, i == 0 ? null : path.get(i - 1)); //3

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FIND FUNCTION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

static ArrayList< Node> path;

public static boolean find(Node node, int data) {

if (node == null)

return false;

if (node.data == data) {

path.add(node);

return true;

}

boolean filc = find(node.left, data);

if (filc) {

path.add(node);

return true;

}

boolean firc = find(node.right, data);

if (firc) {

path.add(node);

return true;

}

return false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*PRINT K LEVELS DOWN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static void printKLevelsDown(Node node, int k, Node blocker) {

if (node == null || k < 0 || node == blocker)

return;

if (k == 0)

System.out.println(node.data);

printKLevelsDown(node.left, k - 1, blocker);

printKLevelsDown(node.right, k - 1, blocker);

}

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 data = Integer.parseInt(br.readLine());

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

Node root = construct(arr);

printKNodesFar(root, data, k);

}

}

You have already studied the code for "Node to Root Path" and "Print K Levels Down", so we expect that you have a good understanding of them. Now we start discussing the function "printKNodesFar". We first store the path from node to root in an arraylist "path" by calling the "find" function. Next we start a loop for all the nodes in the "path". We call the function "printKLevelsDown" for all the elements in "path". This function takes three parameters, "node", "k" and "blocker".

****

Our "node" is the ith element in the loop and is obtained by path.get(i) . Then, "k" or the steps we wish to go down is k-i. We see from figure 5 that, for i=0th element, we go 3=3-0=3-i steps down, for i=1th element, we go 2=3-1=3-i steps down, for i=2ndelement, we go 1=3-2=3-I steps down, and so on. Here 3=k, so we realize that for every element we go k-i steps down. Now, we discuss the "blocker" variable like we promised to in the "WHAT" section. Here, we wish to remove unwanted paths as discussed earlier. > When we call 2 down from e, we don't wish to include 'r' and 's' nodes in the paths. Hence, here "9" should appear as the "blocker". Now, we won't be able to traverse below 9 as it is an obstacle. > Similarly, when we call 1 down from b, we don't wish to include an 'e' node in the path. Hence, we put "e" as the blocker, so we won't traverse to "e" or any element below it. >Reader, notice that the first element needs no "blocker". > So if i=0, then blocker= "null" else it is equal to the previous element or path.get(i-1). > We realize that for every node, the node before it in the arraylist is the blocker.

Analysis :

Time Complexity:

O(n) With this we conclude our discussion of this problem. We highly recommend you to watch the solution video of this question.