package com.company;

//Problem 4 Assignment 2

//Time complexity – O(n^2) Space complexity – O(n)

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

import java.util.Stack;

public class Main {

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

Scanner scan = new Scanner(System.in);

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

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

String[] s = new String[p.length];

//Taking input

for (int i = 0; i < p.length; i++) {

if(p[i].equals("N")||p[i].equals("n")){ //for null cases

s[i]="-1";

}

else {

s[i]=p[i]; //all data in s

}

//System.out.print(s[i]);

}

Tree tree = new Tree(s); //Tree constructed by passing s

int informant = scan.nextInt();

//Doing dfs from index informant in the data structure

tree.dodfs(informant);

//dfs on node informant and tree and count max size of queue-- answer

}

}

class Node{

public int data;

public Node link1;

public Node link2;

}

class Tree {

static int max = 0;

static int count = 0;

public Node[] nodes;

Stack<Integer> stack = new Stack<Integer>();

//constructor

public Tree(String[] s) {

nodes = new Node[s.length];

for (int i = 0; i < s.length; i++) {

nodes[i] = new Node(); //Initialising nodes with all data from s

nodes[i].data = Integer.parseInt(s[i]);

nodes[i].link1 = null;

nodes[i].link2 = null;

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

if (nodes[j].data == -1 && j == 0) continue; //continue when ip “N” encounters

else { //Linking nodes to one another

if (nodes[j].link1 == null && nodes[j].data != -1) {

nodes[j].link1 = nodes[i];

//System.out.println("Parent of"+nodes[i].data+" is "+nodes[j].data+"-Left child");

break;

} else {

if (nodes[j].link2 == null && nodes[j].data != -1) {

nodes[j].link2 = nodes[i];

//System.out.println("Parent of"+nodes[i].data+" is "+nodes[j].data+"-right child");

break;

}

}

}

}

}

}

public void dodfs(int informant) {

//dfs algorithm but only calculating count of deepest branch

Boolean[] visited = new Boolean[nodes.length];

for (int i = 0; i < nodes.length; i++) {

if (nodes[i].data == informant) {

//stack[top++] = nodes[i];

//if(top+1>max)max = top+1;

dfsperform(i, visited);

}

}

}

public void dfsperform(int nodeindex, Boolean[] visited) {

visited[nodeindex] = true;

count++;

stack.push(nodeindex);

if (count > max) max = count;

while (!stack.isEmpty()) {

int n = stack.pop();

if (visited[n] == false) {

dfsperform(n, visited);

count++;

}

}

System.out.println(max);

}

}

Approach – All the data is stored in individual nodes with data field as value of nodes and all links link1 and link2 as null initially, After creating all nodes with input values given by user, tree is constructed. Tree is constructed from element by element filling links of each node from first to last and filling link1 then link2 to maintain order of tree. Next the informant variable is checked against all data values of nodes[] to find the node for dfs. dfs is run on node informant and calculated max depth as it will be the max number of hours we have to wait. Other nodes will end data parallely to it