## Binary Tree : Swap Nodes

We define depth of a node as follow:

* Root node is at depth *1*.
* If the depth of parent node is d, then the depth of current node wll be d+1.

**Swapping:** Swapping subtrees of a node means that if initially node has left subtree L and right subtree R, then after swapping left subtree will be R and right subtree L.

Eg. In the following tree, we swap children of node 1.

Depth

1 1 [1]

/ \ / \

2 3 -> 3 2 [2]

\ \ \ \

4 5 5 4 [3]

Inorder traversal of left tree is 2 4 1 3 5 and of right tree is 3 5 1 2 4.

**Swap operation**: Given a tree and a integer, K, we have to swap the subtrees of all the nodes who are at depth h, where h ∈ [K, 2K, 3K,...].

You are given a tree of N nodes where nodes are indexed from [1..N] and it is rooted at 1. You have to performT swap operations on it, and after each swap operation print the inorder traversal of the current state of the tree.

**Input Format**   
First line of input contains N, number of nodes in tree. Then N lines follow. Here each of *ith* line (1 <= i <= N) contains two integers, a b, where a is the index of left child, and b is the index of right child of *ith* node. -1 is used to represent null node.   
Next line contain an integer, T. Then again T lines follows. Each of these line contains an integer K.

**Output Format**   
For each K, perform swap operation as mentioned above and print the inorder traversal of the current state of tree.

**Constraints**   
1 <= N <= 1024   
1 <= T <= 100   
1 <= K <= N   
Either a = -1 or 2 <= a <= N   
Either b = -1 or 2 <= b <= N   
Index of (non-null) child will always be greater than that of parent.

**Sample Input #00**

3

2 3

-1 -1

-1 -1

2

1

1

**Sample Output #00**

3 1 2

2 1 3

**Explanation**

\*\*[s] represents swap operation is done at this depth.

Test Case #00: As node 2 and 3 has no child, swapping will not have any effect on it. We only have to swap the child nodes of root node.

1 [s] 1 [s] 1

/ \ -> / \ -> / \

2 3 [s] 3 2 [s] 2 3

Solution :

import java.io.\*;

import java.util.\*;

class LL {

Node root;

class Node {

Node left;

Node right;

int data;

public Node(int d)

{

data = d;

left = null;

right = null;

}

}

public Node insert(int n,Scanner sc)

{

Queue q = new LinkedList();

root = new Node(1);

q.add(root);

Node curr;

String s1 = null;

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

{

curr = (Node)q.poll();

s1 = sc.nextLine();

String[] nd = s1.split(" ");

int nd1 = Integer.parseInt(nd[0]);

int nd2 = Integer.parseInt(nd[1]);

if(nd1 != -1)

{

Node new1 = new Node(nd1);

curr.left = new1;

q.add(new1);

}

else

{

curr.left = null;

}

if(nd2 != -1)

{

Node new2 = new Node(nd2);

curr.right = new2;

q.add(new2);

}

else

{

curr.right = null;

}

}

return root;

}

public void inorder(Node root)

{

if(root == null)

{

return;

}

inorder(root.left);

System.out.print(root.data + " ");

inorder(root.right);

}

public void swap(Node root,int t,Scanner sc)

{

Node curr = root;

Node tmp;

for(int i=0;i<t;i++)

{

String k1 = sc.nextLine();

int k = Integer.parseInt(k1);

//write a function for finding the nodes at k dept

depthK(root,k,1);

System.out.println();

}

}

public void depthK(Node root,int k1,int currL)

{

if(root == null){ return; }

if(currL % k1 == 0)

{

Node tmp = root.left;

root.left = root.right;

root.right = tmp;

}

depthK(root.left,k1,currL+1);

System.out.print(root.data + " ");

depthK(root.right,k1,currL+1);

}

}

public class Swap\_tree\_nodes {

public static void main(String[] args) {

LL l1 = new LL();

Scanner sc = new Scanner(System.in);

String n1 = sc.nextLine();

int n = Integer.parseInt(n1);

l1.root = l1.insert(n,sc);

String t1 = sc.nextLine();

int t = Integer.parseInt(t1);

l1.swap(l1.root,t,sc);

}

}