Swap Nodes [Algo]

A binary tree is a tree which is characterized by any one of the following properties:

- It can be an empty (null).
- It contains a root node and two subtrees, left subtree and right subtree. These subtrees are also binary tree.

Inorder traversal is performed as

1. Traverse the left subtree.
2. Visit root (print it).
3. Traverse the right subtree.

(For an Inorder traversal, start from the root and keep visiting the left subtree recursively until you reach the leaf,then you print the node at which you are and then you visit the right subtree.)

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 perform `T` 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 i^{th} 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 i^{th} 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
```

Sample Input #01

```
5
2 3
-1 4
-1 5
-1 -1
-1 -1
1
2
```

Sample Output #01

```
4 2 1 5 3
```

Sample Input #02

```
11
2 3
4 -1
5 -1
6 -1
7 8
-1 9
-1 -1
10 11
-1 -1
-1 -1
-1 -1
2
2
4
```

Sample Output #02

```
2 9 6 4 1 3 7 5 11 8 10
2 6 9 4 1 3 7 5 10 8 11
```

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
```

Test Case #01: Swapping child nodes of node *2* and *3* we get

```
   1          1
  /\         /\
 2  3  [s] ->  2  3
  \  \       /  /
   4  5      4  5
```

Test Case #02: Here we perform swap operations at the nodes whose depth is either *2* and *4* and then at nodes whose depth is *4*.

```
     1           1             1
    /\          /\            /\
   /  \        /  \          /  \
  2   3  [s]   2   3          2   3
 /   /        \   \          \   \
/   /          \   \          \   \
4   5     ->   4   5     ->   4   5
 /   /\         /   /\         /   /\
/   /  \       /   /  \       /   /  \
6   7   8 [s]  6   7   8 [s]  6   7   8
 \     /\      /     /\       \     /\
  \   /  \    /     /  \       \   /  \
   9  10  11  9     11  10      9  10  11
```