- HARD
- Max Score: 50 Points
- https://course.acciojob.com/idle?question=775f2cc3-6262-45bf-b2ba-a92d983b4b01

Kth Ancestor of a Tree Node

You are given a tree with `n` nodes numbered from `0` to `n - 1` in the form of a parent array `parent` where `parent[i]` is the parent of `ith` node. The root of the tree is node `0`. Find the `kth` ancestor of a given node.

Return the `kth` ancestor of a tree `node` or if no such ancestor is present return -1. It is the `kth` node in the path from that `node` to the root node.

Input Format

First line contains `n`.

Next line contains n spaced integers of `parent` array.

Next line contains `node` and `k`.

Output Format

Return the `kth` ancestor of a tree `node` is the `kth` node in the path from that `node` to the root node or if no such ancestor is present return -1.

Example 1

Input

- `7`
- `-1 0 0 1 1 2 2`
- `5 2`

Output

- `0`

Explanation

- `      0`
- `   1      2`
- `  3   4   5   6`
-
- `Observe from the tree that the node 5''''s 2nd ancestor is the node 0.`

Example 2

Input

- `7`
- `-1 0 0 1 1 2 2`
- `6 3`

Output

- `-1`

Explanation

- `      0`
- `   1      2`

- `3 4 5 6`
-
- `There is no 3rd ancestor of the node 6 so output is -1.`

Constraints

1 <= k <= n <= 5 * 10^4

parent.length == n

parent[0] == -1

0 <= parent[i] < n for all 0 < i < n

0 <= node < n

Topic Tags

- Trees
- DFS
- Binary Search
-

My code

```
import java.util.*;

class Solution {
    //  public int kthAncestor(int n, int[] parent, int node, int k) {
        //write code here
                    public int kthAncestor(int n, int[] parent, int node, int k) {
        //write code here
                    if(node < 0) return -1;
```

```
                    if(k == 0) return node;

                    return kthAncestor(n, parent, parent[node], k-1);
  //  }

    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int parent[] = new int[n];
        for (int i = 0; i < n; i++)
            parent[i] = sc.nextInt();
        int node = sc.nextInt();
        int k = sc.nextInt();
        Solution obj = new Solution();
        System.out.println(obj.kthAncestor(n, parent, node, k));
        sc.close();
    }
}
```