小美将自己朋友的名字写在了一块，惊讶地发现她写出的那个字符串 S 有一个惊人的性质：一个人是小美的朋友当且仅当她/他的名字是那个字符串的子序列。现在小团想根据那个字符串判断一个人是不是小美的朋友。

*子序列：一个字符串 A 是另外一个字符串 B 的子序列，当且仅当可以通过在 B 中删除若干个字符（也可能一个都不删），其他字符保留原来顺序，使得形成的新字符串 B’与 A 串相等。例如，ABC 是 AABDDC 的子序列，而 ACB 不是 AABDDC 的子序列。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

public class Main {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      String[] params = br.readLine().trim().split(" ");
      int n = Integer.parseInt(params[0]);
      int m = Integer.parseInt(params[1]);
      String S = br.readLine().trim();
      String T = br.readLine().trim();
      int j = 0;
      long posSum = 0;
      for(int i = 0; i < n; i++){
         if(S.charAt(i) == T.charAt(j)){
            j++;
            posSum += i + 1;
         }
      }
      if(j == T.length()){
         System.out.println("Yes");
         System.out.println(posSum);
      }else{
         System.out.println("No");
      }
   }
}
```

小美在观察一棵美丽的无根树。

小团问小美：“小美，我考考你，如果我选一个点为根，你能不能找出子树大小不超过 K 的前提下，子树内最大值和最小值差最大的子树的根是哪个点？多个点的话你给我编号最小的那个点就行了。”

小美思索一番，说这个问题难不倒他。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.HashMap;
```

```
import java.util.ArrayList;

public class Main {
    static boolean[] visited;  // 标记节点是否已经被访问
    static int[] weight;       // 节点权值
    static int[] childNum;     // 存储以节点 i 为根的树有多少个节点
    static int[] max, min;     // 存储以节点 i 为根的子树下的最大值和最小值
    // 节点间的最大差值
    static int maxDiff = -1;
    // 待求节点
    static int node = -1;
    // 邻接表
    static HashMap<Integer, ArrayList<Integer>> tree;
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String[] temp = br.readLine().trim().split(" ");
        int n = Integer.parseInt(temp[0]);
        int k = Integer.parseInt(temp[1]);
        temp = br.readLine().trim().split(" ");
        weight = new int[n + 1];
        for(int i = 1; i <= n; i++) weight[i] = Integer.parseInt(temp[i - 1]);
        int x, y;
        // 构建树图的邻接表
        tree = new HashMap<>();
        ArrayList<Integer> list;
        for(int i = 1; i <= n - 1; i++){
            temp = br.readLine().trim().split(" ");
            x = Integer.parseInt(temp[0]);
            y = Integer.parseInt(temp[1]);
            if(tree.containsKey(x)){
                list = tree.get(x);
                list.add(y);
                tree.put(x, list);
            }else{
                list = new ArrayList<>();
                list.add(y);
                tree.put(x, list);
            }
            if(tree.containsKey(y)){
                list = tree.get(y);
                list.add(x);
                tree.put(y, list);
            }else{
                list = new ArrayList<>();
```

```
            list.add(x);
            tree.put(y, list);
        }
    }
    int root = Integer.parseInt(br.readLine().trim());
    visited = new boolean[n + 1];
    max = new int[n + 1];
    min = new int[n + 1];
    childNum = new int[n + 1];
    dfs(root, k);
    System.out.println(node);
  }

  // 求取节点 parent 下子节点的最值
  private static void dfs(int parent, int k) {
    visited[parent] = true;
    // 初始化 parent 下的最值为 parent 的节点权重
    max[parent] = weight[parent];
    min[parent] = weight[parent];
    // 初始情况下，该子树只有一个节点
    childNum[parent] = 1;
    for(int i = 0; i < tree.get(parent).size(); i++){
      int child = tree.get(parent).get(i);
      if(!visited[child]){
        // 没访问过这个孩子节点就进行深搜
        dfs(child, k);
        max[parent] = Math.max(max[parent], max[child]);
        min[parent] = Math.min(min[parent], min[child]);
        childNum[parent] += childNum[child];
      }
    }
    if(childNum[parent] <= k && max[parent] - min[parent] >= maxDiff){
      // 以 parent 为根节点的子树满足节点数小于等于 k，且最大差值大于等于目前最大
      if(max[parent] - min[parent] > maxDiff){
        // 大于了直接更新，等于的话需要考虑哪个根节点的编号小
        node = parent;
        maxDiff = max[parent] - min[parent];
      }else{
        // 如果 node 还没有赋值，就直接赋值为当前节点，否则取满足要求的节点中编号最小的
        node = node == -1? parent: Math.min(node, parent);
      }
    }
```

```
    }
}
```