小团惹小美生气了，小美要去找小团“讲道理”。小团望风而逃，他们住的地方可以抽象成一棵有 n 个结点的树，小美位于 x 位置，小团位于 y 位置。小团和小美每个单位时间内都可以选择不动或者向相邻的位置转移，很显然最终小团会无路可逃，只能延缓一下被“讲道理”的时间，请问最多经过多少个单位时间后，小团会被追上。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

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 x = Integer.parseInt(params[1]);
        int y = Integer.parseInt(params[2]);
        if(x == y){
            // 直接被逮到，GG
            System.out.println(0);
        }else{
            // 否则还能挣扎一下，用邻接表构建无向图
            Node[] graph = new Node[n + 1];
            for(int i = 1; i <= n; i++)
                graph[i] = new Node();
            for(int i = 0; i < n - 1; i++){
                params = br.readLine().trim().split(" ");
                int u = Integer.parseInt(params[0]);
                int v = Integer.parseInt(params[1]);
                graph[u].neighbor.add(v);
                graph[v].neighbor.add(u);
            }
            int[] disX = new int[n + 1];
            int[] disY = new int[n + 1];
            Arrays.fill(disX, -1);
            Arrays.fill(disY, -1);
            // 用 bfs 求小美和小团到其他节点的时间
            Queue<int[]> queue = new LinkedList<>();
            queue.offer(new int[]{x, 0});  // 自己和自己的距离为 0
            while(!queue.isEmpty()){
                int[] cur = queue.poll();
```

```
            disX[cur[0]] = cur[1];
            // 遍历节点 cur[0]的所有邻居
            for(int i = 0; i < graph[cur[0]].neighbor.size(); i++){
                int node = graph[cur[0]].neighbor.get(i);
                if(disX[node] == -1){
                    // 当前节点的邻居在它的基础上距离要增加 1
                    int time = cur[1] + 1;
                    queue.offer(new int[]{node, time});
                }
            }
        }
        queue = new LinkedList<>();
        queue.offer(new int[]{y, 0});
        while(!queue.isEmpty()){
            int[] cur = queue.poll();
            disY[cur[0]] = cur[1];
            for(int i = 0; i < graph[cur[0]].neighbor.size(); i++){
                int node = graph[cur[0]].neighbor.get(i);
                if(disY[node] == -1){
                    int time = cur[1] + 1;
                    queue.offer(new int[]{node, time});
                }
            }
        }
        int maxTime = Integer.MIN_VALUE;
        for(int i = 1; i <= n; i++){
            if(disX[i] > disY[i])
                maxTime = Math.max(maxTime, disX[i]);
        }
        System.out.println(maxTime);
    }
  }
}

class Node {
  public ArrayList<Integer> neighbor;
  public Node() {
    neighbor = new ArrayList<>();
  }
}
```

小团深谙保密工作的重要性，因此在某些明文的传输中会使用一种加密策略，小团如果需要传输一个字符串 S，则他会为这个字符串添加一个头部字符串和一个尾部字符串。头部字符串满足至少包含一个“MT”子序列，且以 T 结尾。尾部字符串需要满足至少包含一个“MT”子序列，且以 M 开头。例如 AAAMT 和 MAAAT

都是一个合法的头部字符串，而 MTAAA 就不是合法的头部字符串。很显然这样的头尾字符串并不一定是唯一的，因此我们还有一个约束，就是 S 是满足头尾字符串合法的情况下的最长的字符串。

很显然这样的加密策略是支持解码的，给出你一个加密后的字符串，请你找出中间被加密的字符串 S。

```
public class Main {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      int n = Integer.parseInt(br.readLine().trim());
      char[] str = br.readLine().trim().toCharArray();
      int left = 0, right = n - 1;
      // 先保证头部出现过 M
      for(left = 0; left < str.length; left ++) {
         if(str[left] == 'M'){
            left ++;
            break;
         }
      }
      // 再保证尾部出现过 T
      for(right = n - 1; right >= 0; right --) {
         if(str[right] == 'T'){
            right --;
            break;
         }
      }
      // 最后双指针夹逼
      while(!((str[left] == 'T') && (str[right] == 'M'))){
         if(str[left] != 'T')
            left ++;
         if(str[right] != 'M')
            right --;
      }
      System.out.println(new String(str).substring(left + 1, right));
   }
}
```

美团打算选调 n 名业务骨干到 n 个不同的业务区域，本着能者优先的原则，公司将这 n 个人按照业务能力从高到底编号为 1~n。编号靠前的人具有优先选择的权力，每一个人都会填写一个意向，这个意向是一个 1~n 的排列，表示一个人希望的去的业务区域顺序，如果有两个人同时希望去某一个业务区域则优先满足编号小的人，每个人最终只能去一个业务区域。

例如 3 个人的意向顺序都是 1 2 3，则第一个人去 1 号区域，第二个人由于 1 号区域被选择了，所以只能选择 2 号区域，同理第三个人只能选择 3 号区域。

最终请你输出每个人最终去的区域。

```
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));
      int n = Integer.parseInt(br.readLine().trim());
      boolean[] used = new boolean[n + 1];
      for(int i = 0; i < n; i++){
         String[] choice = br.readLine().trim().split(" ");
         for(int j = 0; j < choice.length; j++){
            // 依次遍历第 i 位骨干的选择，遇到没有选过的就选
            if(!used[Integer.parseInt(choice[j])]){
               used[Integer.parseInt(choice[j])] = true;  // 将这个区域标记为选择过
               System.out.print(choice[j] + " ");
               break;
            }
         }
      }
   }
}
```

小团从某不知名论坛上突然得到了一个测试默契度的游戏，想和小美玩一次来检验两人的默契程度。游戏规则十分简单，首先有给出一个长度为 n 的序列，最大值不超过 m。

小团和小美各自选择一个[1,m]之间的整数，设小美选择的是 l，小团选择的是 r，我们认为两个人是默契的需要满足以下条件:

1. l 小于等于 r。
2. 对于序列中的元素 x，如果 0<x<l,或 r<x<m+1,则 x 按其顺序保留下来，要求保留下来的子序列单调不下降。

小团为了表现出与小美最大的默契，因此事先做了功课，他想知道能够使得两人默契的二元组<l,r>一共有多少种。

我们称一个序列 A 为单调不下降的，当且仅当对于任意的 i>j,满足 A_i>=A_j。

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

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 m = Integer.parseInt(params[0]);
      int n = Integer.parseInt(params[1]);
```

```
    String[] strArr = br.readLine().trim().split(" ");
    int[] arr = new int[n];
    for(int i = 0; i < n; i++) arr[i] = Integer.parseInt(strArr[i]);
    int count = 0;  // 符合默契的数对数
    int r = m, l;
    while(r > 0){
        ArrayList<Integer> temp = new ArrayList<>();
        for(int i = 0; i < n; i++){
            if(arr[i] > r)
                temp.add(arr[i]);
        }
        if(!isIncrease(temp)) break;
        int lowerBound = 1, upperBound = r;
        // 固定 r 然后二分法确定 l
        int curCnt = 0;  // 在当前 r 下能获得的符合题意的数对数
        while(lowerBound <= upperBound){
            l = (upperBound + lowerBound) / 2;
            // 保留数组中小于 l 或大于 r 的数
            temp = new ArrayList<>();
            for(int i = 0; i < n; i++){
                if(arr[i] < l || arr[i] > r)
                    temp.add(arr[i]);
            }
            // 检查一下 temp 是否满足单调不减
            if(isIncrease(temp)){
                // 如果满足，l 就可以取小于等于当前 l 的数，一共有(l,r)对 l 个
                curCnt = l;
                lowerBound = l + 1;    // 小的肯定满足，尝试更大的 l
            }else{
                upperBound = l - 1;
            }
        }
        r --;
        count += curCnt;
    }
    System.out.println(count);
}

// 判断序列的单调不减性
private static boolean isIncrease(ArrayList<Integer> seq) {
    for(int i = 0; i < seq.size() - 1; i++)
        if(seq.get(i) > seq.get(i + 1)) return false;
    return true;
}
```

```
}
```