AcWing-4742:电⚡

某城市有 N 个电力节点，编号 1 ~ N。

这些电力节点形成的电力网络，可以看作一个 N 个节点 N − 1 条边的连通图。

每个电力节点都有一个固定的电容，其中第 i 个节点的电容为 A_i。

现在，可以选择其中一个节点进行供电，其它节点也可以根据实际连接以及具体电容情况接收电力。

具体来说，如果第 i 个节点通电，那么它也可以将电力传输给其它所有与它**直接连接**且电容**严格小于** A_i 的节点。

我们希望通过合理选择初始供电节点，从而使得尽可能多的节点能够通电。

请你计算并输出可以通电的最大节点数量。

输入格式

第一行包含整数 T，表示共有 T 组测试数据。

每组数据第一行包含整数 N。

第二行包含 N 个整数 $A_1, A_2, \ldots, A_N$。

接下来 N − 1 行，每行包含两个整数 X_i, Y_i，表示节点 X_i 和 Y_i 之间存在直接连接。

输出格式

每组数据输出一个结果，每个结果占一行。

结果表示为 `Case #x: y`，其中 x 为组别编号（从 1 开始），y 为可以通电的最大节点数量。

数据范围

$1 \leqslant T \leqslant 100$,
$1 \leqslant A_i \leqslant 10^9$,
$1 \leqslant X_i, Y_i \leqslant N$,
一个测试点内最多 15 组数据满足 $1 \leqslant N \leqslant 2 \times 10^5$，其余数据满足 $1 \leqslant N \leqslant 10^3$。

输入样例：

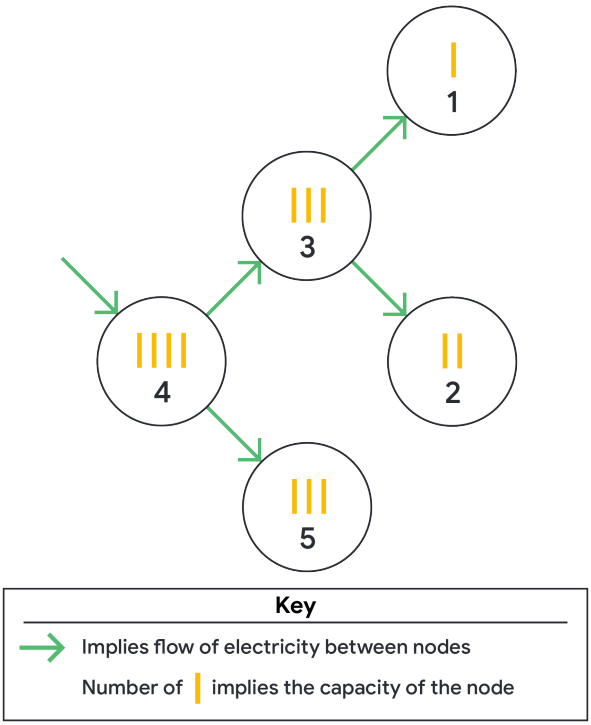
```
2
5
1 2 3 4 3
1 3
2 3
4 3
4 5
6
1 2 3 3 1 4
3 1
3 2
3 4
```

```
4 5
1 6
```

输出样例：

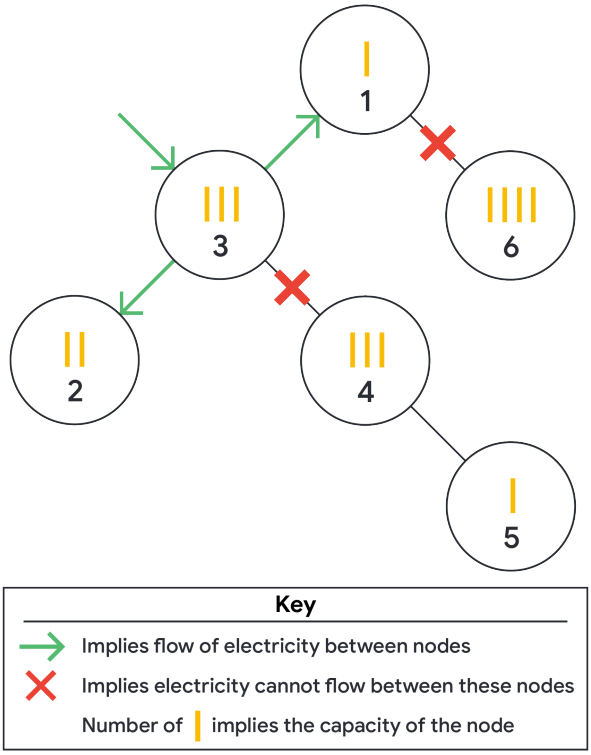
```
Case #1: 5
Case #2: 3
```

样例解释



在 Case 1 中，最佳方案是给第 4 个节点供电，这样可以将电力传输到所有节点。

注意，如果给第 3 个节点供电，则电力只会传输至第 1,2 个节点，而无法传输至第 4 个节点，这样只有三个节点可以通电。



在 Case 2 中，最佳方案是给第 3 个节点供电，这样可以将电力传输至第 1,2 个节点，但是无法传输至第 4 个节点，因为 A4 并不严格小于 A3。

注意，如果给第 6 个节点供电，则电力只会传输至第 1 个节点，如果给第 4 个节点供电，则电力只会传输至第 5 个节点。

题目要求说n个节点n-1条边，故一定为一棵树，且不可能有环（否则环状结构会导致边数过多or不满足单向传递条件）

本题思路为使用**dp+记忆化存储**，设`val[i]`为如果让`i`作为手动点亮的节点，则一共可以点亮多少个节点。这个值是固定的。

此外，使用邻接表存储图，接着遍历每一个节点求点亮它的最多亮灯数。使用记忆化存储减少重复计算。

```
75  const int N = int(2e5+5);
76
77  struct Node{
78      int num;     //电容值
79      vector<int> neigh;   //电容小于该节点且相连的节点
80  }list[N];
81
82  int val[N];      //记忆化数组
```

此外，可以只存储满足单向调节的边：

```
110             if(list[x1].num<list[x2].num)
111                 list[x2].neigh.push_back(x1);
112             else if(list[x1].num>list[x2].num)
113                 list[x1].neigh.push_back(x2);    //只存储有效边
```

Code：

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <unordered_map>
#include <cstring>
using namespace std;

const int N = int(2e5+5);

struct Node{
    int num;    //电容值
    vector<int> neigh;  //电容小于该节点且相连的节点
}list[N];

int val[N];     //记忆化数组

int dfs(int index){
    //计算index的总亮数
    if(val[index]!=0) return val[index];

    int temp=1;
    for(int i=0;i<list[index].neigh.size();i++){
        temp+=dfs(list[index].neigh[i]);
    }
    val[index]=temp;
    return temp;
}

int main() {
    int T;
    scanf("%d",&T);
    for(int index=1;index<=T;index++){
        memset(list,0,sizeof(list));
        memset(val,0,sizeof(val));
        int N;
        scanf("%d",&N);
        for(int i=1;i<=N;i++){
            scanf("%d",&list[i].num);   //录入电容值
        }
        for(int i=0;i<N-1;i++){
            int x1,x2;
            scanf("%d%d",&x1,&x2);
            if(list[x1].num<list[x2].num)
                list[x2].neigh.push_back(x1);
            else if(list[x1].num>list[x2].num)
                list[x1].neigh.push_back(x2);   //只存储有效边
        }
        int ans=0;      //结果
        for(int i=1;i<=N;i++){
```

```
            ans=max(ans,dfs(i));
        }
        printf("Case #%d: %d\n",index,ans);
    }
}
```

代码提交状态：Accepted

这题时间卡得很死，使用`cin`代替`scanf`就T了，如果先加个排序再从底层开始遍历也会TLE，以下代码就不符合要求：

```
//#include <iostream>
//#include <algorithm>
//#include <vector>
//#include <unordered_map>
//#include <cstring>
//using namespace std;
//
//const int N = int(2e5+5);
//
//struct Node{
//    int lstn;   //节点编号
//    int num;    //电容值
//    int tog;    //连通电压的总节点
//    vector<int> neigh;  //电容小于该节点且相连的节点
//}list[N];
//
//bool cmp(Node& p1,Node& p2){
//    return p1.num<p2.num;
//}
//
//int main() {
//    int T;
//    scanf("%d",&T);
//    for(int index=1;index<=T;index++){
//        memset(list,0,sizeof(list));
//        int N;
//        scanf("%d",&N);
//        for(int i=1;i<=N;i++){
//            list[i].lstn=i;
//            scanf("%d",&list[i].num);   //录入电容值
//        }
//        for(int i=0;i<N-1;i++){
//            int x1,x2;
//            scanf("%d%d",&x1,&x2);
//            if(list[x1].num<list[x2].num)
```

```
//                 list[x2].neigh.push_back(x1);
//             else if(list[x1].num>list[x2].num)
//                 list[x1].neigh.push_back(x2);
//         }
//         //按照电容值排序
//         int ans=0;
//         sort(list+1,list+N+1,cmp);
//         int Maps[N];
//         for(int i=1;i<=N;i++){
//             Maps[list[i].lstn]=i;       //录入转换后的节点编号和位置对应关系
//         }
//         for(int i=1;i<=N;i++){
//             int nodeLight=0;
//             for(int ii=0;ii<list[i].neigh.size();ii++){
//                 nodeLight+=list[Maps[list[i].neigh[ii]]].tog;
//             }
//             list[i].tog=nodeLight+1;
//             if(list[i].tog>ans)
//                 ans=list[i].tog;
//         }
//         printf("Case #%d: %d\n",index,ans);
//     }
//}
```