https://www.luogu.com.cn/problem/P5666

给定一树，求删掉每条边之后分裂成的两棵树的重心编号之和

以树的重心为根

断边(u,v)，会将树分为v的子树和剩余部分

v子树的重心为内重心，剩余部分的重心为外重心，考虑如何分别求出两类重心

先考虑x的内重心：已知所有$son[x]$的内重心，求x的内重心。

考虑一个“合并”的过程：一开始只有x，依次将每个儿子“合并”进来，每次“合并”是由原先的树与新的子树合并，显然新的重心在两者中更重的一者内部，只需要从这一者原先的重心向上跳若干步即可

这样的复杂度是$O(n)$的，考虑每次都会往上跳，所以每条边只会被跳一次

再考虑x的外重心：已知所有$son[x]$的外重心，求x的外重心。

因为根节点是整棵树的重心，而x的子树被砍掉，所以外重心不可能在被砍掉x的子树的这边，所有$son[x]$的外重心在根节点向下的某一条链上(不断向重儿子方向跑，否则重儿子会更大)，x的外重心也在这条链上，且是最深的一个点

所以我们找到$son[x]$中深度最深的外重心，继承这个点，并尝试向下移动即可找到x的外重心；向下移动只可能进入重儿子，预处理一下；一种特殊情况是继承了根节点并且x就在其重儿子里，这时我们只能尝试向次重儿子移动

复杂度同上

```
//大体思路是以整棵树重心为根 对每个点求内重心和外重心
//内重心考虑一个一个子树合并 合并后重心一定是原来两重心的连线 所以一定是较大子树重心向上直到now
//因为整棵树重心为根 外重心一定在根的另一棵子树中 且k几个儿子的外重心一定在一条链上(不断向重儿子方向跑 否则重儿子会变得更大) 则k的重心在该链下面
#include <iostream>
#include <cstdio>
#include <cstring>
#define int long long
using namespace std;
const int MAX=3e5+5;
const int inf=9999999999999999;
struct edge{
    int next,to;
}h[MAX<<1];
int head[MAX],cnt;
void add(int x,int y){
    h[++cnt]=edge{head[x],y};
    head[x]=cnt;
    h[++cnt]=edge{head[y],x};
    head[y]=cnt;
}
int t,n,ans,tmp;
int root,amax[MAX],siz[MAX],in[MAX][2],out[MAX][2],fa[MAX];
//可能有两个重心 都对答案有贡献
int ci,que[MAX];   //root的次大儿子 求外重心的可能链
void getroot(int now,int father){
    siz[now]=1,amax[now]=0;
    for(int aim,i=head[now];i;i=h[i].next){
        aim=h[i].to;
        if(aim!=father){
            getroot(aim,now);
            siz[now]+=siz[aim];
            amax[now]=max(amax[now],siz[aim]);
        }
```

```
    }
    amax[now]=max(amax[now],n-siz[now]);
    if(amax[root]>amax[now])root=now;
}
void dfs1(int now,int father){
    in[now][0]=in[now][1]=now,siz[now]=1,amax[now]=0;      //内重
心;子树大小;重儿子
    fa[now]=father;
    for(int aim,i=head[now];i;i=h[i].next){
        aim=h[i].to;
        if(aim!=father) {
            dfs1(aim, now);
            if(siz[aim]>siz[amax[now]]){
                if(now==root)ci=amax[now];
                amax[now]=aim;
            }else if(now==root && siz[aim]>siz[ci])ci=aim;

            if(siz[now]==siz[aim])in[now][0]=now,in[now]
[1]=aim,siz[now]+=siz[aim];
            else{
                if(siz[aim]>siz[now])in[now][0]=in[now]
[1]=in[aim][0];
                siz[now]+=siz[aim];
                while(in[now][0]!=now && siz[now]-siz[in[now]
[0]]>siz[now]/2)in[now][0]=fa[in[now][0]];
                in[now][1]=in[now][0];
                if(in[now][1]!=now && siz[amax[fa[in[now][1]]]]
<=siz[now]/2)in[now][1]=fa[in[now][1]];
            }
        }
    }
}
void dfs2(int now,int father){
    out[now][0]=out[now][1]=1;
    for(int aim,i=head[now];i;i=h[i].next){
        aim=h[i].to;
```

```
        if(aim!=father){
            dfs2(aim,now);
            if(out[aim][0]>out[now][0])out[now][0]=out[now]
[1]=out[aim][0];
        }
    }
    while(out[now][0]+1<=que[0] && siz[que[out[now][0]+1]]>(n-
siz[now])/2)out[now][0]++;
    out[now][1]=out[now][0];
    if(out[now][1]+1<=que[0] && n-siz[que[out[now][1]+1]]-
siz[now]<=(n-siz[now])/2)out[now][1]++;
    ans+=que[out[now][0]];
    if(out[now][1]!=out[now][0])ans+=que[out[now][1]];
}
signed main(){
    scanf("%lld",&t);
    while(t--) {
        memset(head, 0, sizeof(head));
        cnt = root=0;

        scanf("%lld", &n);
        for (int x, y, i = 1; i < n; i++)scanf("%lld %lld", &x,
&y), add(x, y);
        amax[0] = inf;
        getroot(1, 1);
        amax[0] = ans = ci = 0;
        dfs1(root, root);
        for (int aim, i = head[root]; i; i = h[i].next) {
            aim = h[i].to;
            if (aim == amax[root]) {
                que[0] = 0, que[++que[0]] = root, que[++que[0]] =
ci;
                while (amax[que[que[0]]] != 0) {
                    tmp = amax[que[que[0]]];
                    que[++que[0]] = tmp;
                }
```

```
                dfs2(aim, root);
            } else {
                que[0] = 0, que[++que[0]] = root, que[++que[0]] =
amax[root];
                while (amax[que[que[0]]] != 0) {
                    tmp = amax[que[que[0]]];
                    que[++que[0]] = tmp;
                }
                dfs2(aim, root);
            }
        }
        for (int i = 1; i <= n; i++) {
            if (i != root) {
                if (in[i][0] == in[i][1])ans += in[i][0];
                else ans += in[i][1] + in[i][0];
            }
        }
        printf("%lld\n", ans);
    }
    return 0;
}
```