C infected tree（简单树形dp难度应该是提高组普及加。）

ad

> 对于一颗二叉树：
> 引入感染的概念：根节点1被感染。
> 现在有如下过程，
> 1.选择一个未被感染的点。删除该节点
> 2.感染的点继续相邻的节点感染。
> 求最终保存节点的最大值。

$20min$

- 计算出所有节点向下的节点数。
- 每一步做出最优策略。就是选择节点数更大的节点。
- 递归函数的终点设计
  - 当前感染的点子节点小于2.

- 最后发现想法应该错了。
  - 并不是每一次都选择节点数量更大的那一个点。
  - 要最终影响的量也有另外一颗子树。例如一颗子树深度非常大，一颗子树深度比较小。

及时更新思路：

- 定义f_i为选择染节点i之后的操作。
- 两个节点的情况之下：$f_i = max(s_u + f_v, s_v + f_u)$
- 一个节点的情况之下：$f_i = s_u$
- 零个节点的情况在之下：$f_i = 0$

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int maxn = 3e5 + 10;
struct tree{
    int no;
    int to;
} e[maxn << 1];
int tot = 0;
int head[maxn];
int f[maxn];
int dp[maxn];
ll ans = 0;
void add(int x, int y){
    e[++tot].to = head[x];
    e[tot].no = y;
    head[x] = tot;
}
int cu(int now, int fa){
    for (int t = head[now]; t; t = e[t].to)
        if (e[t].no != fa)
```

```
            f[now] += cu(e[t].no, now);
    return f[now] + 1;
}
void dfs(int now, int fa){
    int temp[3]{};
    int s = 0;
    for (int t = head[now]; t; t = e[t].to)
        if (e[t].no != fa)
            dfs(e[t].no, now), temp[++s] = e[t].no; //保证下面的点已经被解决了。
    if (s == 0)
        dp[now] = 0;
    if (s == 1)
        dp[now] = f[temp[1]];
    if (s == 2)
        dp[now] = max(f[temp[1]] + dp[temp[2]], f[temp[2]] + dp[temp[1]]);
}
void solve(){
    int n;
    cin >> n;
    ans = 0;
    // memset(f, 0, n + 1);
    // memset(dp, 0, n + 1);
    // memset(head, 0, (n + 1) << 1);
    //按照字节数进行的一个变化。
    //最近的这一些函数导致了的bug不少。
    for (int i = 0; i <= n; i++)
        head[i] = f[i] = head[i] = 0;
    for (int i = 1; i < n; i++)
    {

        int x, y;
        cin >> x >> y;
        add(x, y);
        add(y, x);
    }
    cu(1, 0);
    dfs(1, 0);
    cout << dp[1] << '\n';
}
int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    int t;
    cin >> t;
    while (t--)
        solve();
}
```