

题面

有一个 n 个节点的树，根节点为 1，令叶子节点数为 m ，叶子节点的权值为一个 1 到 m 的排列。
Alice 和 Bob 在树上玩游戏，两人从根节点开始，Alice 先手的轮流行走 $u \rightarrow \text{son}(u)$ 的路径直到抵达叶子节点。叶子的权值为本次游戏的得分。Alice 希望最大化得分，Bob 希望最小化得分。在某种叶子权值的排列下，得分的最大值和最小值分别是多少。

数据范围：

$$1 \leq n \leq 2 \times 10^5$$

题解

容易发现两种情况对称，先考虑求得分的最大值。

状态定义：

定义状态 $f_{u,0/1}$ ， $f_{u,0}$ 表示在 u 节点的人希望最大化得分时能得到子树第 $f_{u,0}$ 大的权值； $f_{u,1}$ 表示在 u 节点的人希望最小化得分时能得到子树第 $f_{u,1}$ 大的权值。

边界条件：

对于所有叶子节点有 $f_{u,0} = f_{u,1} = 1$ 。

状态转移方程：

以下令 $v \in \text{son}(u)$ 。

考虑 $f_{u,0}$ 的转移，因为要令得分最大，所以最优策略是把 $f_{v,1}$ 中的最小值 x 取出来，然后把所有权值中最大的 x 个点全部放入使 $f_{v,1}$ 最小的 v 的子树中，此时就可以直接走到节点 v 并取到最大值，所以有：

$$f_{u,0} = \min_{v \in \text{son}(u)} f_{v,1}$$

考虑 $f_{u,1}$ 的转移，因为要令得分最大，所以最优策略是把 $f_{v,0}$ 中所有的数和 x 算出来，然后把所有权值中最大的 x 个点在每个 v 的子树中放 $f_{v,0}$ 个，这样可以使 u 在行动时无论如何走都只能使最小值 \geq 第 x 大的数，所以有：

$$f_{u,1} = \sum_{v \in \text{son}(u)} f_{v,0}$$

最终答案：

最终得分的最大值为 $m - f_{1,0} + 1$ 。考虑得分的最小值相当于所有权值反转 $x \leftarrow m - x + 1$ 后得分的最大值且先手是想要得分最小的人，得分为 $m - f_{1,1} + 1$ ，再反转回去可以得到得分的最小值 $f_{1,1}$ 。

。

总复杂度 $\Theta(n)$

代码

```
#include<cstdio>
#define Min(x,y) ((y)<(x)&&((x)=(y)))
const int N=200005, INF=0x3f3f3f3f;
int n, head[N], pedge, f[N][2], cnt;
struct Edge{
    int to, next;
}edge[N<<1];
inline void Ins(int u, int v){
    edge[++pedge]={v, head[u]}, head[u]=pedge;
}
inline void ins(int u, int v){
    Ins(u, v), Ins(v, u);
}
void dfs(int u, int father){
```

```

bool flag=1;
f[u][0]=INF, f[u][1]=0;
for(int e=head[u]; e; e=edge[e].next){
    int v=edge[e].to;
    if(v==father)continue;
    dfs(v,u), flag=0;
    Min(f[u][0], f[v][1]), f[u][1]+=f[v][0];
}
if(flag)f[u][0]=f[u][1]=1, cnt++;
}
int main(){
    scanf("%d", &n);
    for(int i=1, u, v; i<n; i++)scanf("%d%d", &u, &v), ins(u, v);
    dfs(1, 0), printf("%d %d\n", cnt-f[1][0]+1, f[1][1]);
    return 0;
}

```

T2

题面

接下来的树指非空、有根、区分左右孩子的二叉树。

定义：

1. 称 T 能单步替换成为 T' 表示存在 T'' 和 T 的一个叶子节点 u 使得将 u 替换成 T'' 后 $T = T'$ ，记做 $T \rightarrow T'$ 。
2. 称 T 能替换成为 T' 表示存在正整数 n ，使得 $T = T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n = T'$ ，记做 $T \rightarrow^* T'$ 。
3. 定义 $\text{grow}(T) = \{T' | T \rightarrow T'\}$ 。
4. 对森林 $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ ，定义 $\text{grow}(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} \text{grow}(T)$ 。

有 N 组询问，每组询问有 m 个树组成森林 $\mathcal{T} = \{T_1, T_2, \dots, T_m\}$ ，判断是否有有限个树 T 满足 $T \notin \text{grow}(\mathcal{T})$ 。

数据范围：

令每个树的大小为 n ， $N \leq 100$ ， $1 \leq \sum n, \sum m \leq 2 \times 10^6$

题解

令森林 \mathcal{T} 中深度最大的树的深度为 d_{\max} 。令不存在的节点的 $size$ 为 0。

定理 1：

存在深度为 $d > d_{\max}$ 的树不能被替换得到当且仅当存在深度为 $d - 1$ 的树不能被替换得到。

反证，所有的深度为 $d - 1$ 的树能被替换得到当且仅当所有深度为 d 的树能被替换得到。所以只用考虑深度为 d_{\max} 的树是否能被替换得到。如果全部满足，那么只有有限个树 T 满足 $T \notin \text{grow}(\mathcal{T})$ ，因为只有深度 $< d_{\max}$ 的树才可能不能被替换得到。

定理 2：

所有深度为 d_{\max} 的树不能被替换得到当且仅当所有深度为 d_{\max} 的好的树不能被替换得到。其中好的树定义为对于树上的每一个节点 u ，有 $\min(size_{ls_u}, size_{rs_u}) \leq 1$ 。

对深度为 d_{\max} 的树，选择一条从根到叶子的长度为 d_{\max} 的链，可以构造一颗好树同样拥有这条链，且好树的每个点是否有不在链上的儿子取决于原树是否有不在链上的儿子。这样子构造的好树可以通过替换得到原树。这样只需要判断是否能替换得到所有的深度为 d_{\max} 的好树。

定理 3：

不是好树的树不能替换成好树。

对任意不是好树的树，存在节点 u 使得 $\min(size_{ls_u}, size_{rs_u}) > 1$ 替换后 $size$ 一定不小于原来的值，新树仍然不是好树。这样子就只用考虑 \mathcal{T} 中所有是好树的树，删去不是好树的树。

定理 4：

任何一个深度为 d 的好树，可以对应一个或两个长度为 $d-1$ 的 4 进制数。

对于任何一个好树的链上的非叶子节点 u 可以对应四种状态：

1. $size_{ls_u} = 0$ ， u 对应数位上的数为 0。
2. $size_{rs_u} = 0$ ， u 对应数位上的数为 1。
3. $size_{ls_u} = 1$ ， u 对应数位上的数为 2。
4. $size_{rs_u} = 1$ ， u 对应数位上的数为 3。

这条链上的每个节点依次替换成四种数字，再建立 trie 树，就可以将问题简化。若有两个深度为 d 的儿子，则可能同时对应 2 和 3，整个好树可能对应两个四进制树。

定理 5：

对任意一个 \mathcal{T} 中的好树 T ，如果 T' 对应的四进制数的一个前缀是 T ， T' 能被替换得到。

这样子就可以将需要在 trie 树上建立的点的个数从 $O(4^{d_{\max}})$ 变成 $O(\sum n)$ 个了。

根据定理得到一个做法：将 \mathcal{T} 中所有的好树取出来，转化成 4 进制建立 trie 树。一个节点中所有的子树都能被替换得到需要满足如下至少一个条件：

1. 该节点对应的好树在 \mathcal{T} 中存在。
2. 该节点的四个儿子都存在且四个节点对应的子树都能被替换得到。

时间复杂度 $O(\sum n)$

代码

```
#include<cstdio>
#include<cstring>
#define min(x,y) ((x)<(y)?(x):(y))
const int N=2000005;
int T,n,m,son[N][2],root,size[N],cnt,s[N][4];
bool flag[N];
bool check(int u){
    size[u]=1;
    for(int i=0;i<2;i++){
        if(son[u][i]){
            if(!check(son[u][i]))return 0;
            size[u]+=size[son[u][i]];
        }
    }
    return min(size[son[u][0]],size[son[u][1]])<=1;
}
void ins(int&x,int u){
    if(!x)x=++cnt;
    if(!son[u][0]&&!son[u][1])flag[x]=1;
    if(!son[u][0]&&son[u][1])ins(s[x][0],son[u][1]);
    if(!son[u][1]&&son[u][0])ins(s[x][1],son[u][0]);
    if(size[son[u][0]]==1&&son[u][1])ins(s[x][2],son[u][1]);
    if(size[son[u][1]]==1&&son[u][0])ins(s[x][3],son[u][0]);
}
bool solve(int u){
    if(!u)return 0;
    return flag[u]||(solve(s[u][0])&&solve(s[u][1])&&solve(s[u][2])&&solve(s[u][3]));
}
int main(){
```

```
for (scanf ("%d",&T);T--;){
    memset(flag,0,sizeof(flag)),memset(s,0,sizeof(s)),root=cnt=0;
    scanf ("%d",&m);
    for (;m--;){
        scanf ("%d",&n);
        for (int i=1;i<=n;i++)scanf ("%d%d",son[i],son[i]+1);
        if (check(1))ins(root,1);
    }
    puts(solve(root)?"Almost Complete":"No");
}
return 0;
}
```