Part #0 树形 dp 是什么

众所周知，树是一个有 n 个节点，$n-1$ 条无向边的图。这种图可以表示一些事物之间的关系，且这种关系是联通的、无环的。当动态规划建立在一种依赖关系（或者其它互相的关系）之上，树形 dp 便是解决这类问题的好帮手。

通常对于树上的每一个节点，我们要求的 dp 的值通过其父亲节点/儿子节点推算过来。对于特殊的节点（根节点、叶子节点），有时需要初始化。

这样泛泛而谈并不能让我们深入了解树形 dp ，让我们来看树形 dp 的架构与习题。

Part #1 树形 dp 的架构

首先，我们要把这个树建好，通常我们把读入的边建立双向的，即如果 u 与 v 相连，建立 $u \to v$ 以及 $v \to u$ 。

来看一下树形 dp 伪代码。

```
void dfs(u,fa,other): // 因为是棵树，所以我们用递归遍历每个节点。fa 是 u 的父节点，other 是其它传参
    if special
        do  sth // 如果这个节点有什么特殊之处特别计算
    for each v (存在 u->v)
        if v=fa continue //我们在建树过程中建的是双向边，会与父亲相连，需要特判
        dfs(v,u) // 一般是先将子树的 dp 值算好
        do dp // 计算 u 的 dp 值（一般是通过子树的值）
```

Part #2 树形 dp 习题

Part #2.1 简单计算

- 例 2.1.1 [Luogu P1352 没有上司的舞会]

做过 dp 的人都知道分类讨论一波吧。

> 设 $\mathrm{dp}_{u,0/1}$ 表示编号为 u 的职员不参加/参加舞会，他所领导的所有人能得到的最大快乐指数。

这句话我们要转化为树形 dp 专用语言：

> 设 $\mathrm{dp}_{u,0/1}$ 表示 u 节点不选/选，该子树内能得到的最大点权和。

这个 dp 值是要计算该节点与其所有子节点的最大点权和，而其所有子节点都是他的儿子们所领导的，所以其 dp 值显然可以累加其儿子的 dp 值！但是要注意这题有约束，根据题意，可以得到以下转移方程：

$\mathrm{dp}_{u,0} = \sum\limits_{v=\mathrm{son}(u)} \max(\mathrm{dp}_{v,0}, \mathrm{dp}_{v,1})$（该节点不选，儿子们不受限制。 $\mathrm{son}(u)$ 表示 u 的儿子。）

$\mathrm{dp}_{u,1} = \sum\limits_{v=\mathrm{son}(u)} \mathrm{dp}_{v,0}$（其儿子不能选）

[Code]

- 例 2.1.2 [Luogu P2796 Facer的程序]

简化题意：统计树的子树数量。

设 dp_u 表示以 u 为根节点的子树数量。

对于 u ，选择每一个儿子 v， 有 dp_v 种形态可以选择，还有一种情况就是不选儿子 v 。根据乘法原理，可以得到这个东西：

$$\text{dp}_u=\prod_{v=\text{son}(u)}(\text{dp}_v+1)$$

注意一下取模和 long long ，以及答案是每一个 dp_u 的和就好了。

Code

- 例 $2.1.3$ POJ 1463 Strategic game

题意：一棵树上，如果选择一个节点，就可以控制其所有儿子节点，问至少选择几个节点可以控制这棵树。

分类讨论。这个节点被控制有两种可能，一个是被自己控制，一个是被儿子控制后该子树的最小选择节点数，设为 $\text{dp}_{u,0/1}$ （ 0 是被儿子控制，1 反之）。

易得转移方程如下：

$$\text{dp}_{u,0}=\sum_{v=\text{son}(u)}\text{dp}_{v,1}$$ （自己不选，儿子必须选择）

$$\text{dp}_{u,1}=1+\sum_{v=\text{son}(u)}\min(\text{dp}_{v,0},\text{dp}_{v,1})$$ （自己选，儿子可选可不选）

Code

- 例 $2.1.4$ Luogu P2899 [USACO08JAN]Cell Phone Network G

这题就比较厉害了，可以靠父亲，同样我们分类讨论。

设 $\text{dp}_{u,0/1/2}$ 表示这个节点靠父亲/靠自己/靠儿子，该子树的最小选择节点数，先考虑好考虑的：

$$\text{dp}_{u,0}=\sum_{v=\text{son}(u)}\min(\text{dp}_{v,1},\text{dp}_{v,2})$$ （其子节点不能靠父亲）

$$\text{dp}_{u,1}=1+\sum_{v=\text{son}(u)}\min(\text{dp}_{v,0},\text{dp}_{v,1},\text{dp}_{v,2})$$ （自己选，子节点随便）

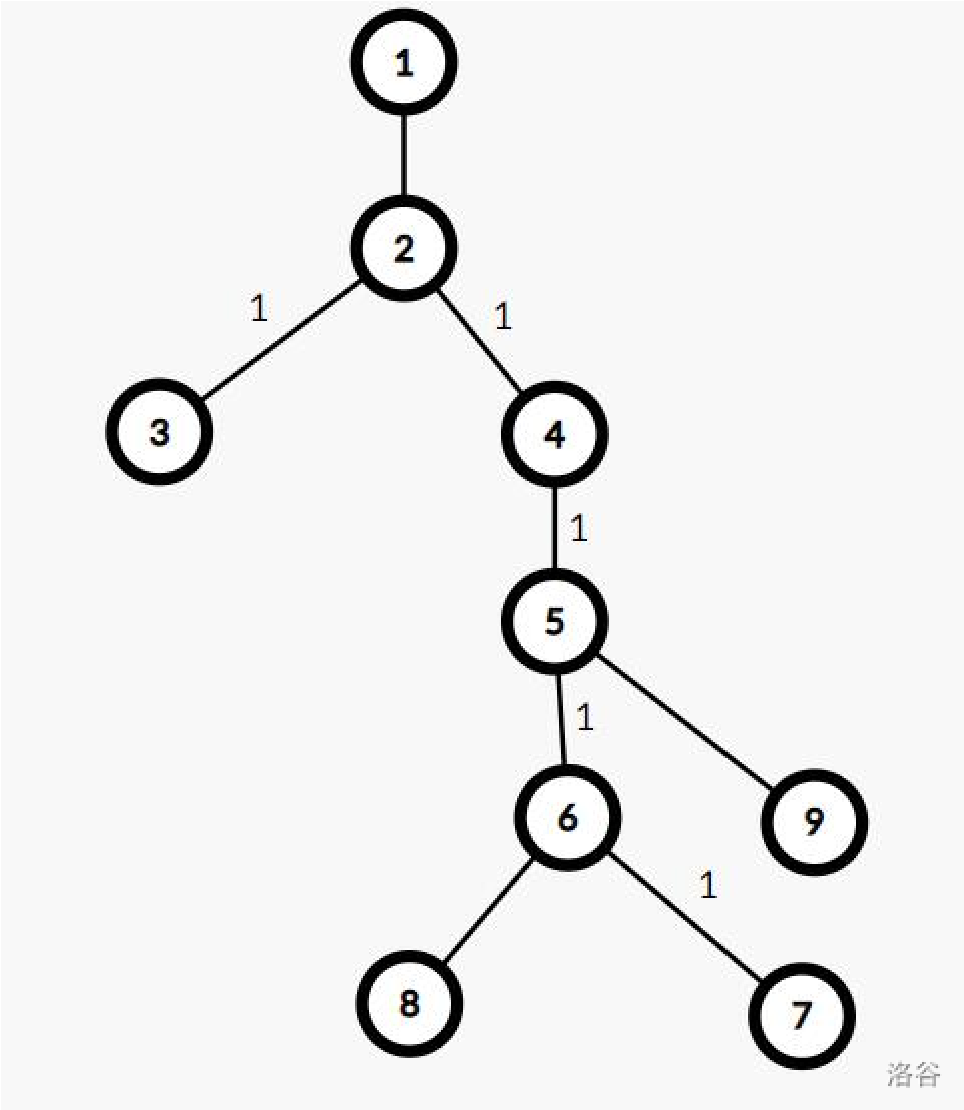
然后就是 $\text{dp}_{u,2}$ 有点意思。每个子节点可以是 $\text{dp}_{v,1/2}$ （不能靠父亲），但是所有子节点里面必须得有一个是 $\text{dp}_{v,1}$ 不然这个节点靠不到儿子了。枚举那个要靠的儿子就可以啦。

Code

Part #2.2 树的直径

- 例 $2.2.1$ SP1437 PT07Z - Longest path in a tree

模板题，我们先来理解一下树的直径。树的直径就是一棵树上的两点之间的最远距离。我们把这棵树画出来，设 x 是直径的最顶端。看张图：



边上标了 1 的路径是这棵树的直径，显然 $x=2$ 。

我们可以发现：

1. 直径在以 x 为根的子树内。
2. 直径是从 x 向下延伸的两条**不重合**的路径。如果有第三条，显然不是一条路线了。

那直径不就是 x 向下延伸的最长路径+次长路径吗（不会有更长的了）？

我们可以 dp 求出每个节点向下延伸的最长路径和次长路径，这个不结合代码不好讲了。以下代码设 $\mathrm{f1}_x$ 和 $\mathrm{f2}_x$ 为最长路径和次长路径。

```
void treedp(int x,int fa){
    for(int i=head[x];i;i=edge[i].nxt){
        int tmp=edge[i].to;
        if(tmp==fa) continue;
        // 枚举子节点
        treedp(tmp,x);
        // 如果从这个节点下去，路径长度最大是 f1[tmp]+1
        if(f1[tmp]+1>f1[x]) f2[x]=f1[x],f1[x]=f1[tmp]+1;
        // 如果比当前的最大值还要大，现在的次大值变成原来的最大值，现在的最大值更新
        else if(f1[tmp]+1>f2[x]) f2[x]=f1[tmp]+1;
        // 如果仅仅比次大值要大，更新次大值
        // 可以发现，这样两条路径一定不重合，因为选择了不同的方向
    }
    if(f1[x]+f2[x]>answer) answer=f1[x]+f2[x];
}
```

因为已经给出重要代码，完整代码不贴了。

- 例 2.2.2 POJ 1383 Labyrinth

也是树的直径，不过这个东西......可能有环？但是实际上是没环的（被坑了被坑了）。

最难的就是建树了吧，执行以下伪代码：

```
BuildTree(x,y)
    for each x1,y1 (两点相邻)
        if (x1,y1) 是 '.' and (not vis[x1][y1])
            vis[x1][y1]=true //标记
            addedge((x,y),(x1,y1)) // 两点连边
            BuildTree(x1,y1)
main
    for each x,y
        if (x,y) 是 '.' and (not vis[x][y])
        // 这是一棵树的根
            vis[x][y]=true
            BuildTree(x,y) // 以它为根建树
```

树建好了就没问题了。为了方便，我们定义以下函数：

```
int code(int x,int y){
    return (x-1)*m+y;
} // m 是一列中的个数
```

给每一个坐标编码，这样就方便很多了。具体可以看一下我~~丑陋~~的代码，由于~~过于乱~~已经讲了很多了就不写注释了。

Code

- 例 2.2.3 POJ 1849 Two

先考虑一个人从一个节点往下走，在这个节点下面的儿子中，有一个是最后一个去的。除了这个最后去的儿子与该节点的连边只用走一遍，其它的儿子的子树中的所有边，为了回到该节点（不然无法走该节点的其它子树），这些边都会被走两次。我们把这个只走一次的路径称为“主路”。这条主路从该节点一直延伸到树的低端。为了让这条路径最长，我们就是求**根节点到叶子节点的最长路径**。

然后两个人，另一个人与这个人的主路重合就不划算了，所以要找另外一条长的路径，就是次长路径。

这样两条主路不就是树的直径吗？我们把所有边长度和的 2 倍减去树的直径的长度就是答案。

代码不贴了。

- 例 2.2.4 HDU 2196 Computer

前方高能！很难，很经典。

首先暴力 dp 是可以过的（恭喜你可以去水了）。

对于每一个点的最远点，有两种情况，一是在子树内，二是在子树外，先来看子树内。

设 $\mathrm{dp}_{u,0/1}$ 表示从 u 向下延伸的最长/次长路径，子树内的最长路径显然是 $\mathrm{dp}_{u,0}$ 。

然后是子树外。如果是子树外，必定是先从 u 向上走，再向下走，哪个划算走哪个。为了降低复杂度，我们直接走向父亲节点，用父亲节点的 dp 值，父亲节点已经考虑过向不向上走了。向上走完向下走。这里有一个难点：向下走不能和向上走的路径重合。如果当前节点 u 要向 fa 走，从 fa 要向下走，fa 向下的最长路径如果包含 u，向下走只能是次长路径。我们记录 prev_u 表示向下的最长路径中，u 的下一

个节点（好吧我承认写代码时打错了英文单词）。定义 $\mathrm{dp}_{u,2/3}$ 表示从 u 节点向上再向下的最长距离，其中 2 表示当前情况下可以向下走最长路径，3 表示只能走次长的。

初始化：$\mathrm{dp}_{u,2}=\mathrm{dp}_{u,0}$（该节点认怂向下去了），$\mathrm{dp}_{u,3}=\mathrm{dp}_{u,1}$（该节点认怂被迫绕道走了）

转移方程：只能上代码了 qwq

```
void dfs2(int u,int fa,int val){
    // val 记录 u->fa 的长度
    dp[u][2]=dp[u][0];
    dp[u][3]=dp[u][1];
    if(x!=1){ // 根节点没父亲
        if(prev[fa]==u){
            // 如果受控不能走最长
            dp[u][2]=maxx(dp[u][2],dp[fa][3]+val);
            // 不受控的 x 此时向上走也被控制了
            dp[u][3]=maxx(dp[u][3],dp[fa][3]+val);
            // 受控的继续受控
        }else{
            dp[u][2]=maxx(dp[u][2],dp[fa][2]+val);
            // 受控的解控了
            dp[u][3]=maxx(dp[u][3],dp[fa][2]+val);
            // 不受控的继续不受控
        }
        // 向上走会获得 val 的长度
    }
    for(int i=head[u];i;i=edge[i].nxt){
        int v=edge[i].to,val1=edge[i].w;
        if(v==fa) continue;
        dfs2(v,u,val1); // 向下 dp
    }
}
```

写完代码之后，发现 $\mathrm{dp}_{u,2/3}$ 可以合并，发现自己懒得合并了......可以参考网上的做法。

Code

Part #2.3.0 铺垫：分配子树，引入背包

- 例 2.3.0.1 Luogu P2015 二叉苹果树

先处理一下：把每个点的左儿子和右儿子记录，把边上的苹果变成点上的苹果（这是一个技巧，父亲与儿子之间的苹果给儿子），把 Q 加 1（因为边变成点之后多了个根节点）。

直接推，发现根本推不出来，我们可以加一维，记录一个子树留下的节点数量。设 $\mathrm{dp}_{u,k}$ 表示以 u 为子树留 k 个点最多留下多少苹果。分两种情况：

1. 只有左儿子/右儿子：给自己留一个，剩下全给儿子。设 v 是单独的儿子，$\mathrm{dp}_{u,k}=\mathrm{cost}_u+\mathrm{dp}_{v,k-1}$，$\mathrm{cost}_u$ 是点权。
2. 有两个儿子：自己要一个，枚举给左儿子多少个，剩下的自然给右儿子。这个的状态转移方程（左儿子 v_1，右儿子 v_2）：
   $\mathrm{dp}_{u,k}=\mathrm{cost}_u+\max\{\mathrm{dp}_{v_1,i}+\mathrm{dp}_{v_2,k-i-1}\}\ \ (0\le i<k)$

Code 不给了，转移方程已经给得很详细了，可以参考别人的代码。

- 例 2.3.0.2 Luogu P1270 “访问”美术馆

首先我们把每条边长翻倍（因为要来回走），然后这题就和上一题很相似了。同样是分配时间，对于叶子节点，计算在这些时间里最多能拿的画；对于非叶子节点，分配时间给两个儿子，取和的最大值。建树很麻烦，相信大家可以搞定。

Code 不贴了，可以参考别人的。

Part #2.3.1 树上的背包 dp

- 例 2.3.1.1 [Luogu P2014 [CTSC1997]选课]

与二叉苹果树的区别，就是不止两个儿子了。这该怎么办？

我们枚举每个儿子分配的课程个数，把每个儿子、每个课程的分配个数的dp 值看作一个物品，总课程数已经知道，啊，这就是个 01 背包！来看看这类问题的架构。

```
void treedp(int u) {
    // 初始化
    int len=vec[u].size();
    for(int i=0; i<len; i++) {
        int tmp=vec[u][i];
        // 枚举儿子
        treedp(tmp);
        // dp 过去
        for(int j=m; j>=1; j--)
        // 01 背包是倒着枚举防止选多次的
        for(int k=0; k<j; k++)
            dp[u][j]=maxx(dp[u][j],dp[u][j-k]+dp[tmp][k]);
            // 给 tmp 分配 k 个，自己保留 j-k 个的 dp 值之和
            // 而自己保留的 dp 值是通过之前的物品计算过来的
            // 这样就成功完成了物品分配
            // 可能还要加入别的计算
    }
}
```

这道题的代码基本已经给出来了，注意初始化，对于任意的 $\mathrm{dp}_{u,j}=\mathrm{score}_i(1\le j\le m)$，score 是学分。

还有，这是一颗森林，我们建立一个超级源点 0 连向所有没有父亲的节点（实际上不用特意搞，连边的时候连的就是 0），课程数量 M 自然要加 1。

[Code]

- 例 2.3.1.2 [Luogu P1273 有线电视网]

和上一题很相似，$\mathrm{dp}_{u,k}$ 表示给 u 为根的子树播给 k 个用户看能赚到的最多的钱。这题要注意，有些情况是不满足条件的，比如一棵子树一共才 3 个叶子节点（用户），给它播 4 个用户显然是不对的，所以我们要初始化为 $-\infty$，计算子树的大小确定分配数量的范围。细节看代码吧。

[Code]

- 例 2.3.1.3 [CodeForces 815C Karen and Supermarket]

这题比上面的难度确实大很多。

首先我们第一反应是给一棵子树分配钱，这种做法被很快叉掉 $(b\le 10^9)$，我们肯定要分配商品了。又因为选不选该节点的优惠券情况不同，所以要分类讨论。

设 $\mathrm{dp}_{u,j,0/1}$ 表示 u 商品的优惠券不选/选，以 u 为子树内选 j 个商品至少要多少钱。得到转移方程如下：

$$\mathrm{dp}_{u,j+k,0} = \min_{v=\mathrm{son}(u),k\le \mathrm{size}(v)} (\mathrm{dp}_{u,j,0} + \mathrm{dp}_{v,k,0})$$

$$\mathrm{dp}_{u,j+k,1} = \min_{v=\mathrm{son}(u),k\le \mathrm{size}(v)} (\mathrm{dp}_{u,j,0} + \mathrm{dp}_{v,k,0/1})$$

哎，我们以前不都倒推做吗？现在咋顺推了？

根据有线电视网的经验，当前背包最大容量是当前访问的子树的节点值和。现在我们改枚举 j ，这个 j 是之前的子树节点数量之和，时间大大减少了。在这道题中，必须顺推才能通过，不然会 Time limit exceeded on test 17 （亲测的）。

这道题的好处：

1. 让我们练习了树上背包的分类讨论，以及如何分配东西；
2. 让我们知道了一种树上背包的优化。

还是一道好题！

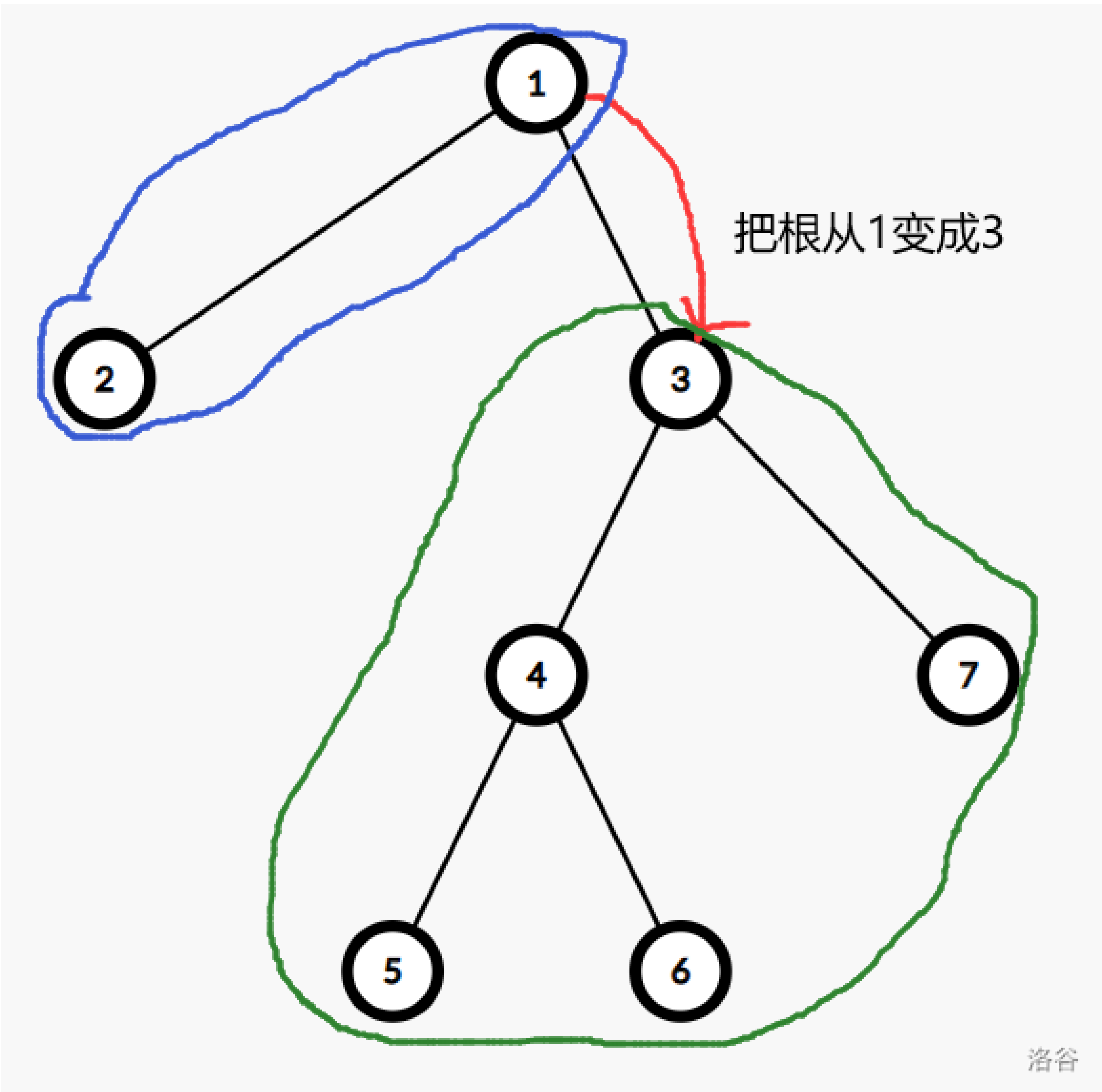
Code

Part #2.4 换根 dp 基础

- 例 2.4.1 Luogu P3478 [POI2008]STA-Station

暴力求复杂度是 $O(n^2)$ 的，不可行。

首先我们可以计算出以 1 为根的深度之和。然后看这张图：



我们把根从 1 变成 3 ，在 3 的子树内（绿色笔圈起来的）到根节点的距离减少了 1，子树之外（蓝色笔圈起来的）到根节点的距离减少了 1 。统计每颗子树的数量，就可以把其它点为根的深度和 dp 出来了。转移方程如下：

$$\mathrm{dp}_v=\mathrm{dp}_u+(n-\mathrm{size}_v)-\mathrm{size}_v\ \ (v=\mathrm{son}(u))$$

其中 size_v 表示以 v 为根的子树大小。

Code

- 例 $2.4.2$ Luogu P2986 [USACO10MAR]Great Cow Gathering G

一个有边权一个没有，没啥好说的，可以用来练练手。

Part #2.5 搭配二分

- 例 $2.5.1$ HDU 3586 Information Disturbing

看到这道题，发现要求选择边权的最大值最小是多少，显然是二分答案。单调性也很好证明，当最大边权变大了，总选择边权相比之前的不可能再变大，因为之前选择的现在更加可以选择了。

在已知最大值的情况之下就比较好做了。设 dp_u 表示以 u 为根的子树脱离整棵树的最小总成本（脱离不了就是 $+\infty$），当 $u\to\mathrm{fa}$ 的边权在范围之内，可以直接选择这条边摧毁，整棵子树都搞定了。要么就是让儿子们分别脱离。如果有一个儿子没法自己脱离，这种方法就行不通。转移方程可以自行推导了。

Code

完结散花！❀❀❀