



浙江财经大学

Zhejiang University Of Finance & Economics



# 树形动态规划

信智学院 陈琰宏

# 什么是树型动态规划

---

顾名思义，树形动态规划就是在“树”的数据结构上的动态规划。树形动态规划是建立在树上的，所以也相应的有二个方向：

根→叶：不过这种动态规划在实际的问题中运用的不多，也没有比较明显的例题，所以不在今天讨论的范围之内。

叶→根：既根的子节点传递有用的信息给根，完后根得出最优解的过程。这类的习题比较的多，下面就介绍一些这类题目和它们的一般解法。

在树的后序遍历dfs的过程中，先把子节点的最优解解出，然后来更新全局最优解。然后用父节点继续更新全局最优解，所以是动态规划思想，再加上是运用在树结构中，所以是树形DP。

---

# 共性总结

---

- 1) 顺序：一般按照后序遍历的顺序，即处理完儿子再处理当前节点，这符合树的子结构的性质。
  - 2) 实现方式：树形DP是通过记忆化搜索实现的，因此采用的是递归方式。
  - 3) 时间复杂度：树形DP复杂度基本上是  $O(n)$  ；若有附加维 $m$ ，则是 $O(n*m)$ 。
-

# 1 [2511] 树的高度（理解树）

---

给定一颗树，树中包含  $n$  个结点（编号  $1 \sim n$ ）和  $n-1$  条无向边，根节点的编号为1，求这颗树的高度。

输入第一行包含整数  $n$ ，表示树的结点数。

接下来  $n-1$  行，每行包含两个整数  $a$  和  $b$ ，表示点  $a$  和点  $b$  之间存在一条边。其中 $a$ 为父节点， $b$ 为孩子节点。

输出一个整数  $h$ ，表示这个颗树的高度。

数据范围 $1 \leq n \leq 1000$

---

# 图的方式建树

```
7  const int N = 1005; //数据范围是10的5次方
8  const int M = 2 * N;
9  int h[N]; //邻接表存储树, 有n个节点
10 int e[M]; //存储元素
11 int ne[M]; //存储列表的next值
12 int idx; //单链表指针
13 int n; //题目所给的输入, n个节点
14
15
16 //a所对应的单链表中插入b a作为父节点
17 void add(int a, int b) {
18     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
19 }
```

```
11 vector<int>g[N];
12 int ans = 0;
13 int dep[N];
14
15 void dfs(int x,int fa){
23
24 void solve(){
25     cin>>n;
26     for(int i=1;i<=n;i++){
27         dep[i] = 1;
28     }
29     for(int i=1;i<n;i++){
30         int x,y;cin>>x>>y;
31         g[x].push_back(y);
32         g[y].push_back(x);
33     }
```

# 搜索过程

```
21 int dfs(int u) {
22     int ans=0;
23     for (int i = h[u]; i != -1; i = ne[i]) {
24         int j = e[i]; //遍历链表
25         int gao=dfs(e[i]); //求前点的高度
26         ans=max(ans,gao);
27     }
28     // cout<<u<<"的高度为"<<ans+1<<"\n";
29     return ans+1;
30 }
```

```
32 int main() {
33     memset(h, -1, sizeof h); //初始化h数组 -1表示尾节点
34     cin >> n; //表示树的结点数
35     for (int i = 0; i < n - 1; i++) {
36         int a, b; cin >> a >> b;
37         add(a, b);
38     }
39     cout << dfs(1) << endl;
40     return 0;
41 }
```

## 2 [6375] 树的重心（理解树）

---

给定一颗树，树中包含  $n$  个结点（编号  $1 \sim n$ ）和  $n-1$  条无向边。

请你找到树的重心，并输出将重心删除后，剩余各个连通块中点数的最大值。

**重心定义：**重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。

第一行包含整数  $n$ ，表示树的结点数。接下来  $n-1$  行，每行包含两个整数  $a$  和  $b$ ，表示点  $a$  和点  $b$  之间存在一条边。

输出一个整数  $m$ ，表示将重心删除后，剩余各个连通块中点数的最大值。

---

# [6375] 树的重心

输入

9

1 2

1 7

1 4

2 8

2 5

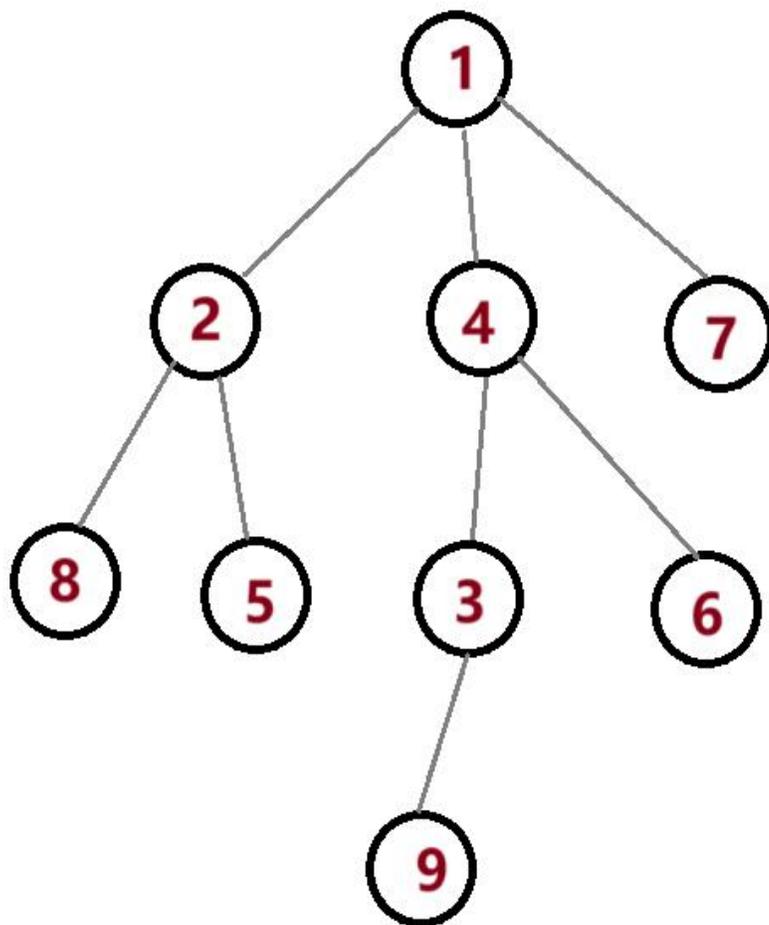
4 3

3 9

4 6

输出

4



# 分析

---

**重心定义：**重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个节点被称为树的重心。

枚举每一个点，求解以该点为根（重心）的所有子树的节点个数即可。  
本题的本质是树的dfs，每次dfs可以确定以u为重心的最大连通块的节点数，并且更新一下ans。

---

# 代码1:

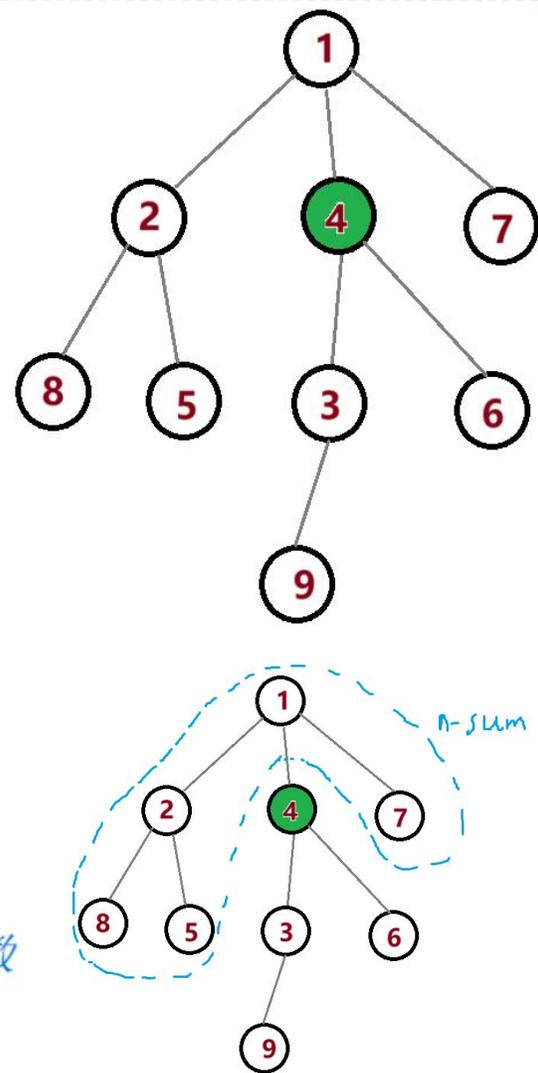
```
48 int main() {
49     memset(h, -1, sizeof h); //初始化h数组 -1表示尾节点
50     cin >> n; //表示树的结点数
51
52     // 树中是不存在环的, 对于有n个节点的树, 必定是n-1条边
53     for (int i = 0; i < n - 1; i++) {
54         int a, b;
55         cin >> a >> b;
56         add(a, b), add(b, a); //无向图
57     }
58
59     dfs(1); //可以任意选定一个节点开始 u<=n
60
61     cout << ans << endl;
62     return 0;
63 }
```

## 代码:2:

```
7  const int N = 1e5 + 10; //数据范围是10的5次方
8  const int M = 2 * N; //以有向图的格式存储无向图, 所以每个节点至多对应2n-2条边
9
10 int h[N]; //邻接表存储树, 有n个节点, 所以需要n个队列头节点
11 int e[M]; //存储元素
12 int ne[M]; //存储列表的next值
13 int idx; //单链表指针
14 int n; //题目所给的输入, n个节点
15 int ans = N; //表示重心的所有的子树中, 最大的子树的结点数目
16
17 bool st[N]; //记录节点是否被访问过, 访问过则标记为true
18
19 //a所对应的单链表中插入b a作为根
20 void add(int a, int b) {
21     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
22 }
```

# 代码3:

```
25 //返回以u为根的子树中节点的个数, 包括u节点
26 int dfs(int u) {
27     int res = 0; //存储 删掉某个节点之后, 最大的连通子图节点数
28     st[u] = true; //标记访问过u节点
29     int sum = 1; //存储 以u为根的树 的节点数, 包括u, 如图中的4号节点
30
31     //访问u的每个子节点
32     for (int i = h[u]; i != -1; i = ne[i]) {
33         int j = e[i];
34         //因为每个节点的编号都是不一样的, 所以 用编号为下标 来标记是否被访问
35         if (!st[j]) {
36             int s = dfs(j); // u节点的单棵子树节点数 如图中的size值
37             res = max(res, s); // 记录最大联通子图的节点数
38             sum += s; //以j为根的树 的节点数
39         }
40     }
41     //n-sum 如图中的n-size值, 不包括根节点4;
42     res = max(res, n - sum); // 选择u节点为重心, 最大的 连通子图节点数
43     ans = min(res, ans); //遍历过的假设重心中, 最小的最大联通子图的 节点数
44     return sum;
45 }
```



### 3 [7279] 树的最长路径(直径)

---

给定一棵树，树中包含  $n$  个结点（编号  $1-n$  和  $n-1$  条无向边，每条边都有一个权值。现在请你找到树中的一条最长路径。

换句话说，要找到一条路径，使得使得路径两端的点的距离最远。注意：路径中可以只包含一个点。

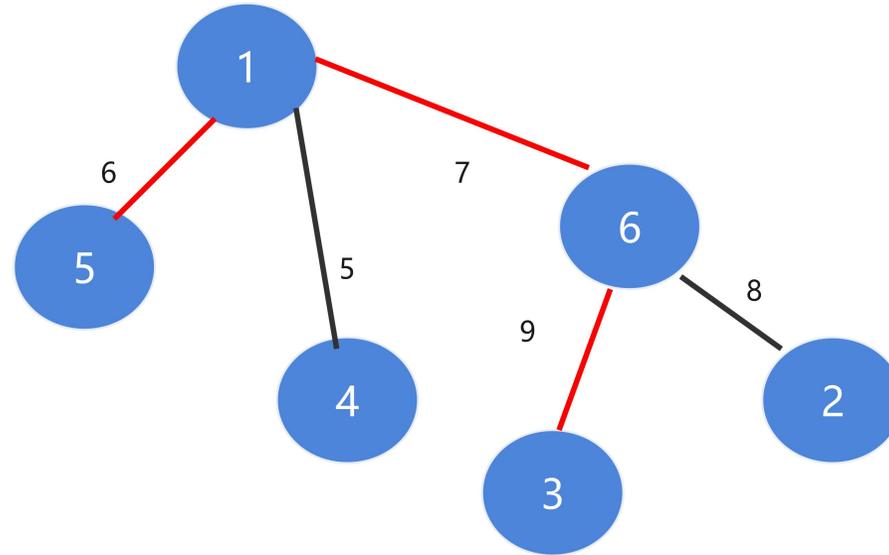
第一行包含整数  $n$ 。接下来  $n-1$  行，每行包含三个整数  $a_i, b_i, c_i$ ，表示点  $a_i$  和  $b_i$  之间存在一条权值为  $c_i$  的边。

输出一个整数，表示树的最长路径的长度。  $1 \leq n \leq 10000$

---

# [7279] 树的最长路径(直径)

6  
5 1 6  
1 4 5  
6 3 9  
2 6 8  
6 1 7



22

## [7279] 树的最长路径(直径)

---

我们知道：树上任意两点的路径是 唯一 确定的，因此我们可以暴力枚举起点 和 终点 找出最长路径

如果这样做的话，我们来思考一下时间复杂度：

枚举 起点 和 终点 —  $O(n^2)$ ，找出两点之间的路径长度 —  $O(\log n)$

但是光是枚举 起点 和 终点，时间复杂度 就直接拉满了，显然这种做法不可取

既然这  $O(n^2)$  条路径不能 一一枚举，那么有什么方式可以把他们 分类枚举呢？

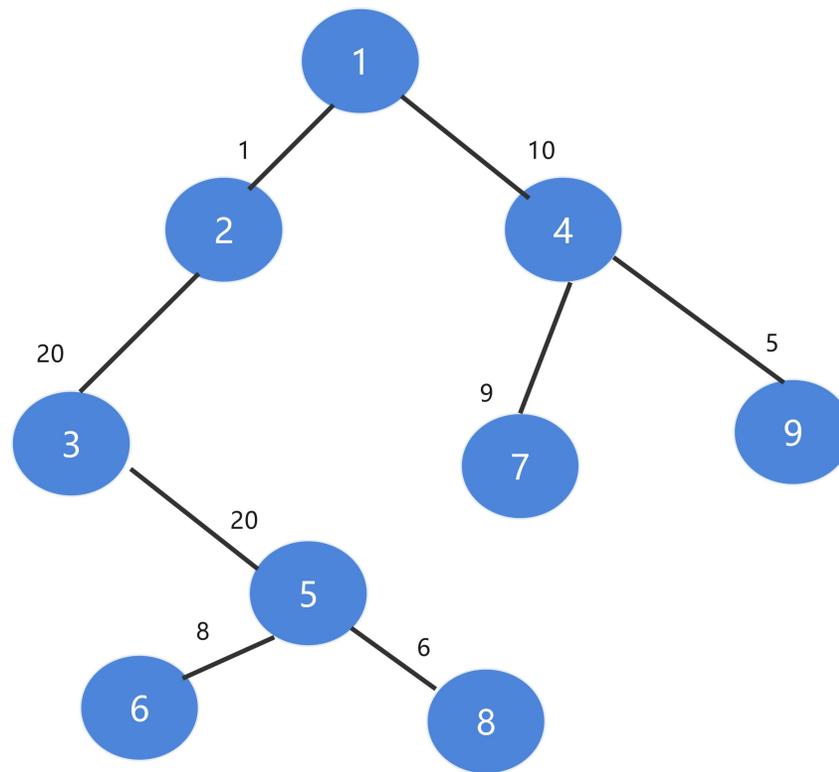
---

# 方法1：两次dfs求解

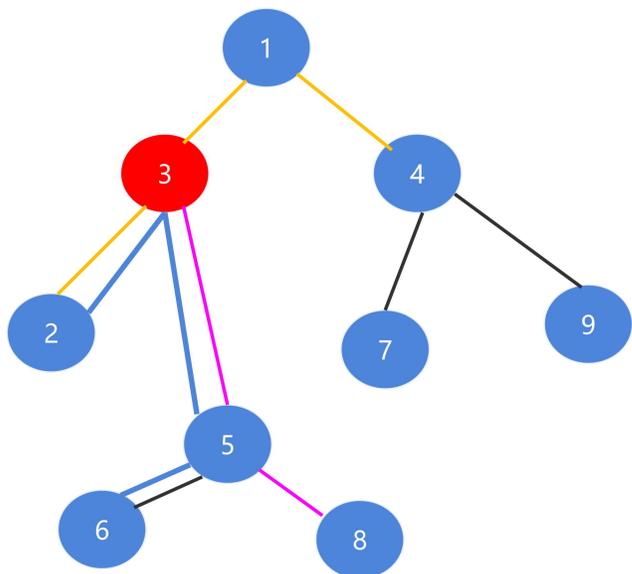
找树的直径。

参考 [1q2013]9223 大臣的旅费

1. 任取一点作为起点，找到距离该点最远的一个点u。
2. 再找到距离u最远的一点v。那么u和v之间的路径就是一条直径。



## 方法2：树形DP求解



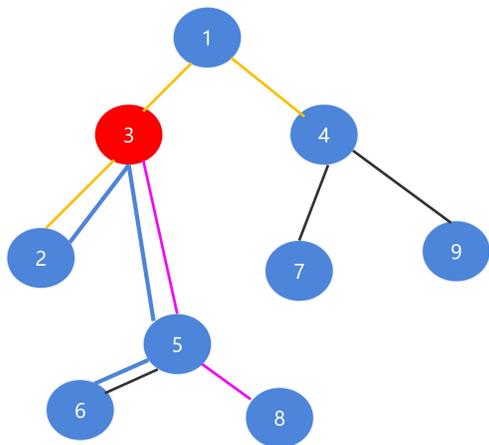
观察标红的节点，那么经过他的路径有：

- 以其子树中的某个节点作为起点，以他作为终点的粉色路径
- 以其子树中的某个节点作为起点，以子树中的某个节点作为终点的蓝色路径
- 以其子树中的某个节点作为起点，以非其子树的节点作为终点的橙色路径

对于第 1 种情况，我们可以直接递归处理其子树，找出到当前子树根节点最长的路径长度即可

对于第 2 种情况，我们在处理第 1 种情况时，顺便找出 1 类路径的次长路径把 最长 和 次长 拼在一起，就是我们要的第 2 种情况

# 分析



而对于第 3 种情况，我们可以把它归类为其 祖先节点 的第 1,2种情况，让其 祖先节点 去处理即可

**状态表示—集合**  $f_{1/2,i}$ : 以节点  $i$  为根的子树中，从子树某个节点到  $i$  的最长路为  $f_{1,i}$ ，次长路为  $f_{2,i}$

**状态表示—属性**  $f_{1/2,i}$ : 路径长度的最大值  $Max$

**状态计算—**  $f_{1/2,i}$

$$\begin{cases} f_{1,i} = \max(f_{1,i_{c1}} + w_{i,i_{c1}}) & c1 \in i_{child} \\ f_{2,i} = \max(f_{1,i_{c2}} + w_{i,i_{c2}}) & c2 \in i_{child} \ \& \ c2 \neq c1 \ \& \ f_{2,i} \leq f_{1,i} \end{cases}$$

# 写法1:

```
5 // 初始不确定树的拓扑结构, 因此要建立双向边
6 const int N = 1e4 + 10, M = N << 1;
7
8 int n;
9 int h[N], e[M], w[M], ne[M], idx;
10 int f1[N], f2[N], res;
11
12 void add(int a, int b, int c)
13 {
14     e[idx] = b, w[idx] = c;
15     ne[idx] = h[a], h[a] = idx ++ ;
16 }
```

```
36 int main()
37 {
38     memset(h, -1, sizeof h);
39     scanf("%d", &n);
40     for (int i = 0; i < n - 1; i ++ )
41     {
42         int a, b, c;
43         scanf("%d%d%d", &a, &b, &c);
44         add(a, b, c), add(b, a, c);
45     }
46     // 我们可以任意选取一个点作为根节点
47     // 这样整棵树的拓扑结构被唯一确定下来了
48     dfs(1, -1);
49     printf("%d\n", res);
50     return 0;
51 }
```

# 写法1:

```
19 //father表示u的父节点,因为该图为无向图,  
20 //并且迭代过程中不能回到父节点,所以要特殊标记.  
21 void dfs(int u, int father)  
22 {  
23     f1[u] = f2[u] = 0;  
24     for (int i = h[u]; ~i; i = ne[i])  
25     {  
26         int j = e[i];  
27         if (j == father) continue;  
28         dfs(j, u);  
29         if (f1[j] + w[i] >= f1[u]) //最长路转移  
30             f2[u] = f1[u], f1[u] = f1[j] + w[i];  
31         else if (f1[j] + w[i] > f2[u]) //次长路转移  
32             f2[u] = f1[j] + w[i];  
33     }  
34     res = max(res, f1[u] + f2[u]);  
35 }
```

## 写法2:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N =1e4+5;
4 struct node{
5     int x,w;
6 };
7 vector<node>g[N];
8
9 int n;
10 int ans=0;
11
12 int dfs(int x,int fa){
25
```

```
26 int main(){
27     cin>>n;
28     for(int i=1;i<n;i++){
29         int x,y,w;cin>>x>>y>>w;
30         g[x].push_back({y,w});
31         g[y].push_back({x,w});
32     }
33
34     dfs(1,-1);
35
36     cout<<ans;
37     return 0;
38 }
```

## 写法2:

```
12 int dfs(int x, int fa){
13     int dis=0;
14     int d1=0, d2=0;
15     for(const auto &[y, w]:g[x]){
16         if(y==fa) continue;
17         int d=dfs(y, x)+w;
18         dis=max(dis, d);
19         if(d>=d1) d2=d1, d1=d;
20         else if(d>d2) d2=d;
21     }
22     ans=max(ans, d1+d2);
23     return dis;
24 }
```

## 写法3:

```
7  const int N=10010;
8
9  int h[N],w[2*N],ne[2*N],e[2*N],idx;
10 int ans;
11
12 void add(int a,int b,int c)
13 {
14     w[idx]=c,e[idx]=b,ne[idx]=h[a],h[a]=idx++;
15 }
16
```

```
38 int main()
39 {
40     memset(h,-1,sizeof(h));
41     int n;scanf("%d",&n);
42     for(int i=1;i<n;i++)//n-1条边
43     {
44         int a,b,w;
45         scanf("%d%d%d",&a,&b,&w);
46         add(a,b,w),add(b,a,w);
47     }
48     dfs(1,-1);
49     printf("%d",ans);
50 }
```

## 写法3:

```
17 //father表示u的父节点,因为该图为无向图,  
18 //并且迭代过程中不能回到父节点,所以要特殊标记.  
19 int dfs(int u,int father)  
20 {  
21     int d1=0,d2=0;//因为题目描述中有:注意:路径中可以只包含一个点  
22     //所以题目中的结果一定不为负数,负的路径由此可以忽略掉  
23     for(int i=h[u];i!=-1;i=ne[i])  
24     {  
25         int j=e[i];  
26         if(j==father) continue;  
27         int d=dfs(j,u)+w[i];//求出路径的长度  
28         //d1,d2求出以该点为顶点的最长路径  
29         if(d>=d1) d2=d1,d1=d;//最长路径和次长路径  
30         else if(d>d2) d2=d;  
31     }  
32     ans=max(ans,d1+d2);  
33     return d1;//返回当前点的f[x];  
34 }
```

## 4 [3707] 二叉苹果树

有一棵苹果树，如果树枝有分叉，一定是分**2叉**（就是说没有只有1个儿子的结点）这棵树共有N个结点（叶子点或者树枝分叉点），编号为1-N，**树根编号一定是1**。我们用一根树枝两端连接的结点的编号来描述一根树枝的位置。

下面是一颗有4个树枝的树现在这颗树**枝条太多了**，需要剪枝。但是一些树枝上长有苹果。**给定需要保留的树枝数量**，求出最多能留住多少苹果。

输入第1行2个数，N和Q ( $1 \leq Q \leq N$ ,  $1 < N \leq 100$ )。

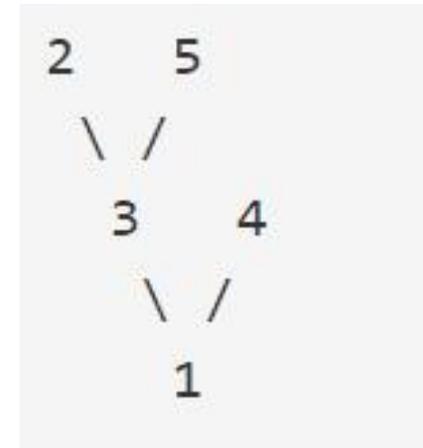
N表示树的结点数，**Q表示要保留的树枝数量**。

接下来N-1行描述树枝的信息。

每行3个整数，前两个是它连接的结点的编号。

第3个数是这根树枝上苹果的数量。

每根树枝上的苹果不超过30000个。



# 4 [3707] 二叉苹果树

输出

一个数，最多能留住的苹果的数量。

样例输入

5 2

1 3 1

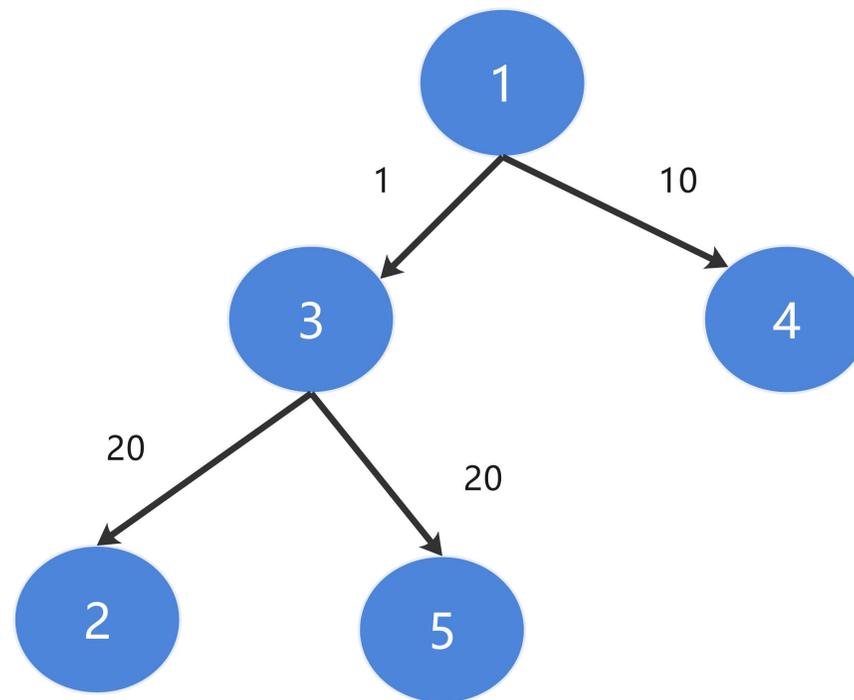
1 4 10

2 3 20

3 5 20

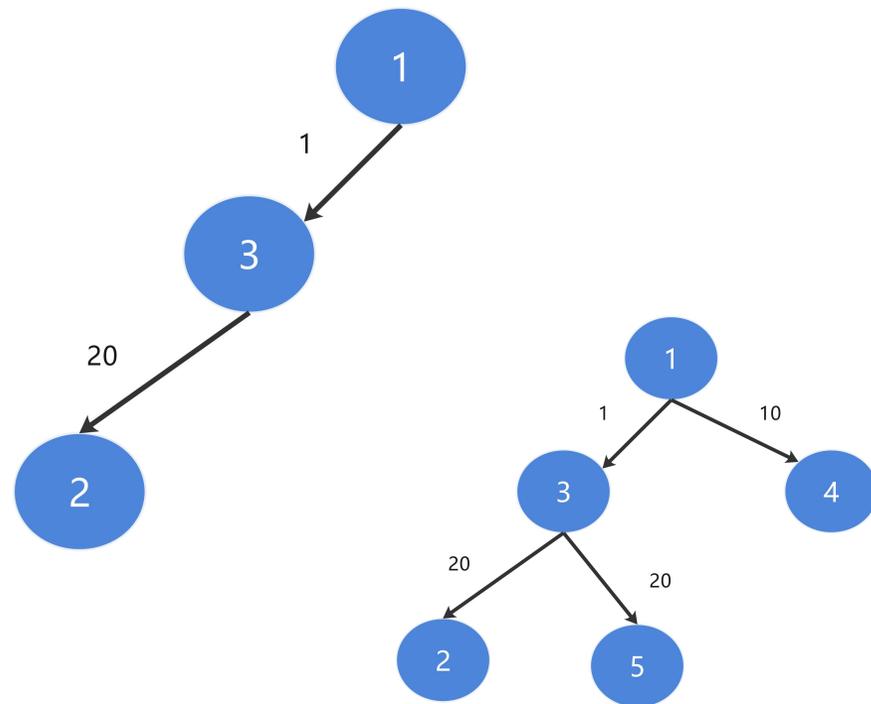
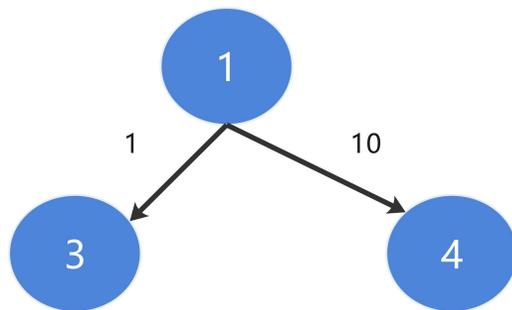
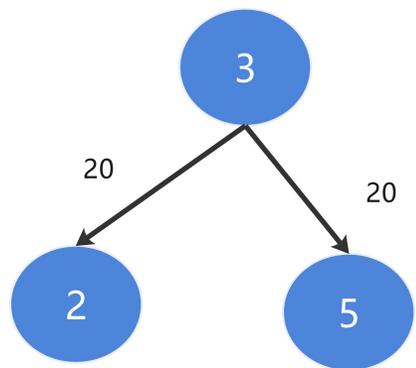
样例输出

21



# 分析

先分析样例数据。根据题意可以把样例数据画成如下二叉树。4条边选两条，很明显， $20+20=40$ 是最大的，但是按照题目要求，我们在选择时**必须从根节点开始**，不然树不成树。所以只能选择：



# 分析: 树形DP、树上背包

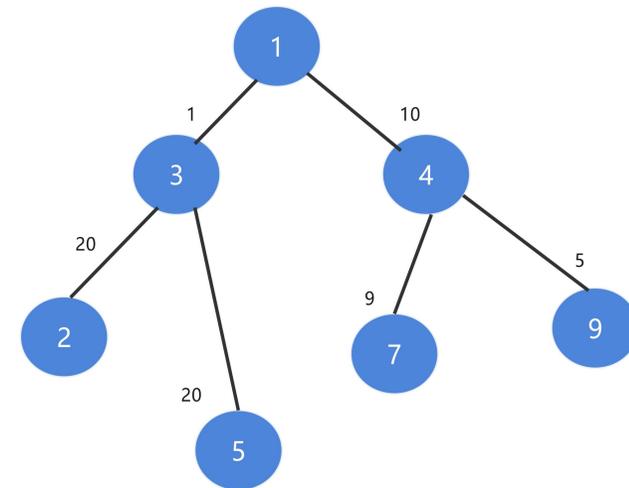
对于每条边，都有选或不选 两种请看看，这就是树上的01背包问题。

设 $f[u][j]$  表示以 $u$ 为根，选择 $j$ 条边 能留住的最大苹果树。

状态转移:

$$f[u][j] = \max(f[u][j], f[u][j-k-1] + f[v][k] + e[i].w)$$

$e[i].w$  为  $uv$ 的边长  $j \in [m, 1], k \in [0, j-1]$

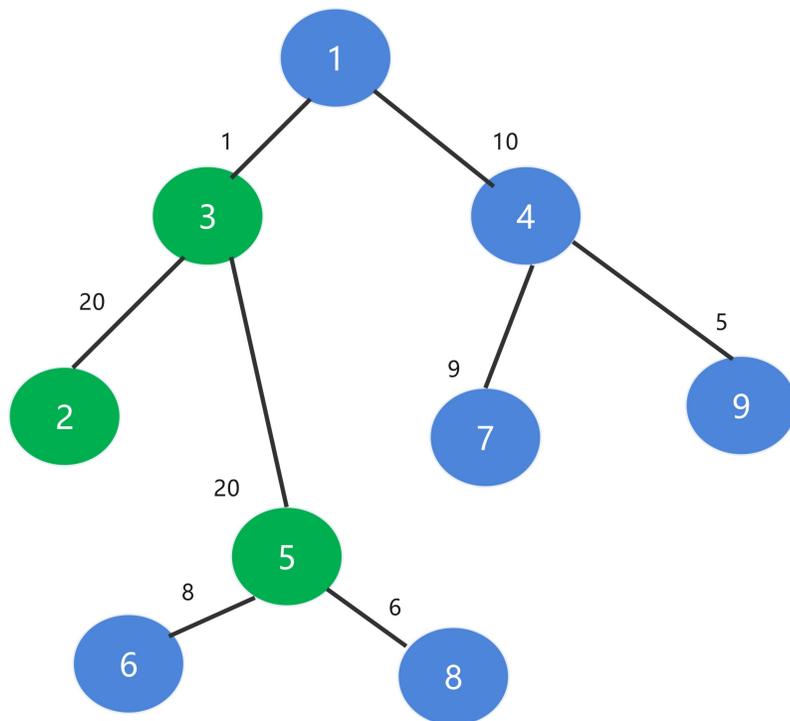


$v$ 子树中选 $k$ 条边，已遍历的 $u$ 子树中选 $j-k-1$ 条边， $(u, v)$ 这条边权是必选的（就是 $e[i]$ ），这就得到当前 $u$ 子树选 $j$ 条边的状态。注意：

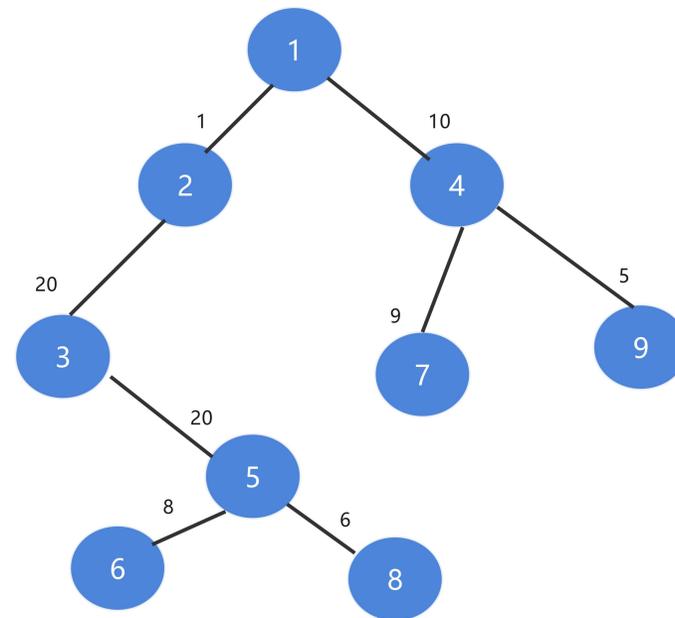
1.  $j$ 要逆序枚举，这保证每条边最多选一次。
2. 为保证时间复杂度 $O(n*m)$ ，用 $siz[u]$ 记录当前 $u$ 子树的边数，枚举 $j, k$ 时收缩一下范围。

# 分析：树形DP、树上背包

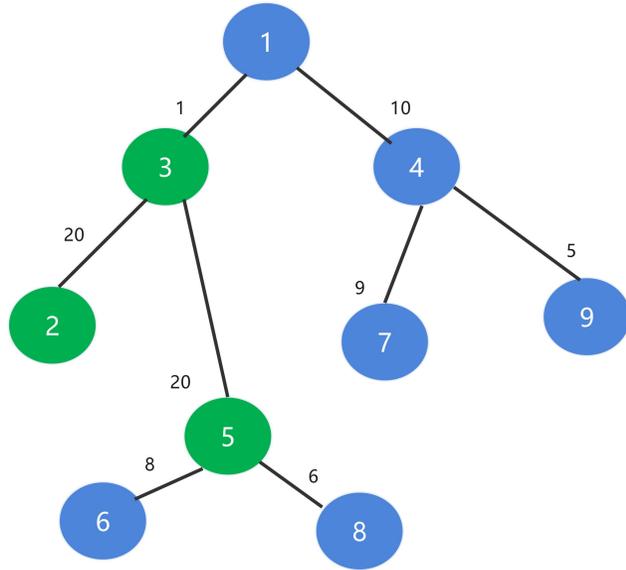
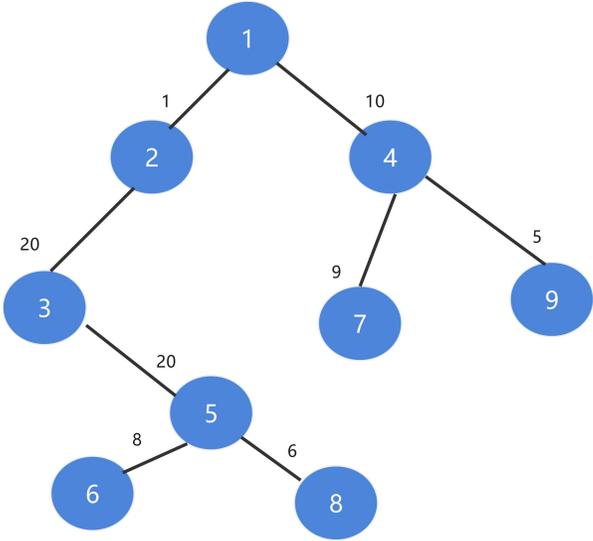
9 4  
1 3 1  
1 4 10  
2 3 20  
3 5 20  
4 7 9  
4 9 5  
5 6 8  
5 8 6



如 $f[5][1]=8$ , 即就选了边56  
 $f[5][2]$ 的最大值为14:  
 $f[3][3]$ 的最大值为48



# 分析



j	1	2	3	4	5	6	7	8	9
i									
1	10	21	41	51	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	20	40	48	54	0	0	0	0	0
4	9	14	14	14	0	0	0	0	0
5	8	14	14	14	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

```

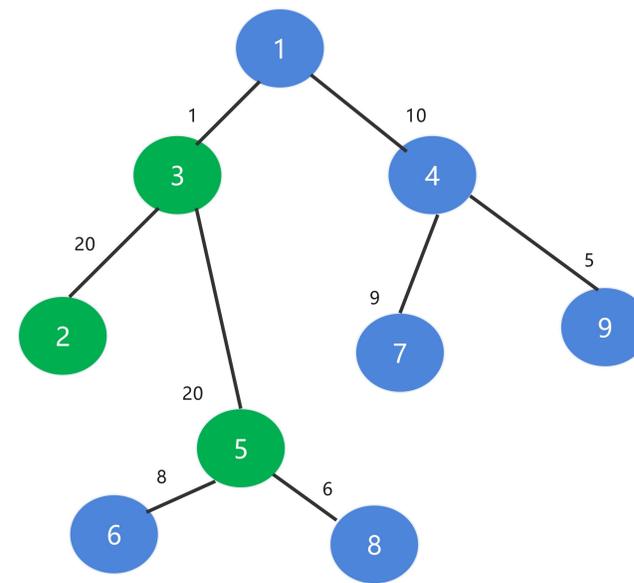
9 4
1 3 1
1 4 10
2 3 20
3 5 20
4 7 9
4 9 5
5 6 8
5 8 6
10 21 41 51 0 0 0 0
0 0 0 0 0 0 0 0
20 40 48 54 0 0 0 0
9 14 14 14 0 0 0 0
8 14 14 14 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

```



# 状态转移方程:

```
for (int j=m; j>=1; j--)  
  for (int k=j-1; k>=0; k--)  
    f[u][j] = max(f[u][j], f[左孩子][k]+f[右孩子][j-k-1]+e[i].w )  
    //u表示当前根节点; j 表示以u为根, 取多少条边  
    e[i].w 为 uv的边长
```



# 如何建树

```
4  const int N = 110, M = N * 2;
```

```
5  
6  int h[N], e[M], ne[M], w[M], idx;
```

```
7  int dp[N][N];
```

```
8  // dp[i][j]:表示考虑到 以第i个节点为根节点的树中
```

```
9  //选取最大枝干 最多选j个树枝 的所有方案, 中的最大苹果数
```

```
10 int n, m;
```

```
11  
12 void add(int a, int b, int c) // 添加一条边a->b, 边权为c
```

```
13 {  
14     e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++ ;  
15 }
```

```
16
```

```
52 int main()  
53 {  
54     memset(h, -1, sizeof h);  
55     cin >> n >> m;  
56     for(int i = 0; i < n - 1; i ++){  
57         int a, b, c;  
58         cin >> a >> b >> c;  
59         add(a, b, c), add(b, a, c);  
60     }  
61     // 题目给了树根节点一定为1, 所以直接dfs(1, -1) 就行  
62     dfs(1, -1);  
63     cout << dp[1][m] << endl;  
64     return 0;
```

# dp过程

```
17 void dfs(int u,int father){
18     for(int i = h[u];i != -1;i = ne[i]){
19         int son = e[i];
20         if(son == father) continue; //防止走回去父节点,形成环超时
21         dfs(son,u); //用子节点更新全局最优解
22
23         for(int j = m;j >= 0;j --){
24             for(int k = 0;k <= j - 1;k ++){
25                 dp[u][j] = max(dp[u][j], dp[u][j - k - 1] + dp[son][k] + w[i]);
26             }
27         }
28     }
29 }
```

## 5 [3714] 没有上司的晚会

Ural周立大学的校长正在筹备学校的80周年纪念聚会。由于学校的职员有不同的职务级别，可以构成一棵以校长为根的人事关系树。每个职员都有一个唯一的整数编号（范围在1到N之间），并且对应一个参加聚会所获得的欢乐度。为了使每个参加聚会者都感到欢乐，校长想设法使**每个职员和他（她）的直接上司不会同时参加聚会**。

你的任务是**设计一份参加聚会者的名单，使总的欢乐度最高**。

输入的第一行是一个整数N， $1 \leq N \leq 6000$  以下的N行是对应的N个职员的欢乐度（欢乐度是一个从-128到127之间的整数）。接着是学校的人事关系树，树的每一行格式如下：

$\langle L \rangle \langle K \rangle$

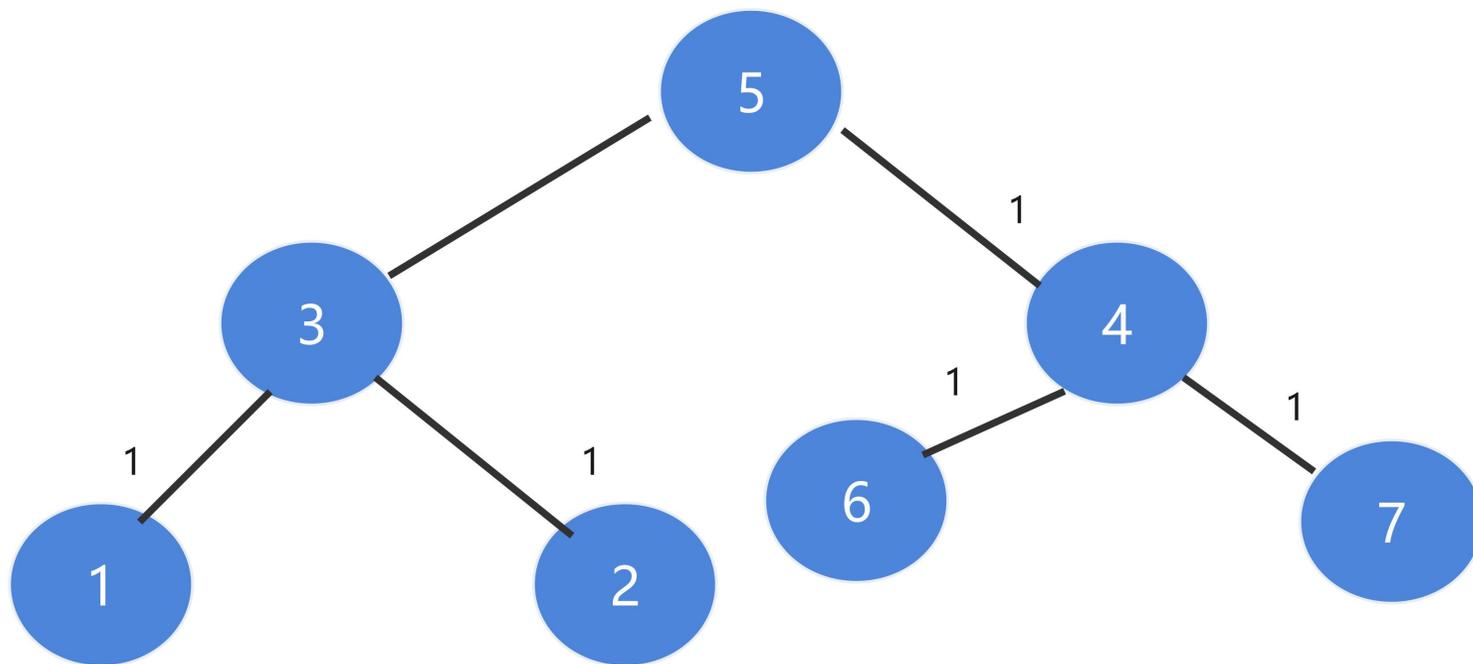
表示第K个职员是第L个职员的直接上司。

输入以0 0表示结束

输出参加聚会者获得的最大总欢乐度

# [3714] 没有上司的晚会

7  
1  
1  
1  
1  
1  
1  
1  
1  
1 3  
2 3  
6 4  
7 4  
4 5  
3 5  
0 0



5

# 分析

较为明显的dp题目，满足在某种情况下，求最优值的问题。对于每个点而言，有“选”和“不选”两种状态，

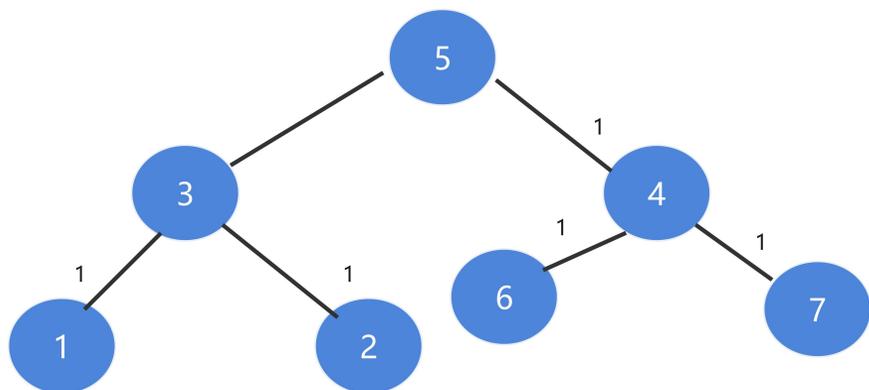
则设

$f[u][0]$ : 为第u个人不来，他的下属所能获得的最大快乐值；（下属可来可不来）

$$\bullet f[u][0] = \sum \max(f[s_i, 0], f[s_i, 1])$$

$f[u][1]$ : 为第u个人来，他的下属所能获得的最大快乐值。（下属不能来）

$$\bullet f[u][1] = \sum (f[s_i, 0])$$



# 主函数：

```
24 int main(){
25     scanf("%d",&n);
26     for(int i = 1;i <= n;i ++)
27         scanf("%d",&happy[i]); //输入每个人的高兴度
28     memset(h,-1,sizeof h); //把h都赋值为-1
29     for(int i = 1;i < n;i ++){
30         int a,b; //对应题目中的L,K,表示b是a的上司
31         scanf("%d%d",&a,&b); //输入~
32         has_father[a] = true; //说明a他有上司
33         add(b,a); //把a加入到b的后面
34     }
35     int root = 1; //用来找根节点
36     while(has_father[root]) root ++; //找根节点
37     dfs(root); //从根节点开始搜索
38     printf("%d\n",max(f[root][0],f[root][1]));
39     //输出不选根节点与选根节点的最大值
```

# 建树：

---

```
5  const int N = 6010;
6  int n;
7  int happy[N]; //每个职工的高兴度
8  int f[N][2];
9  int e[N],ne[N],h[N],idx; //链表,用来模拟建一个树
10 bool has_father[N]; //判断当前节点是否有父节点
11 void add(int a,int b){ //把a插入树中
12     e[idx] = b,ne[idx] = h[a],h[a] = idx ++;
13 }
```

---

# dp过程:

```
15 void dfs(int u){ //开始求解题目
16     f[u][1] = happy[u];
17     //如果选当前节点u, 就可以把f[u,1]先怼上他的高兴度
18     for(int i = h[u]; ~i; i = ne[i]){ //遍历树
19         int j = e[i];
20         dfs(j); //回溯
21         //状态转移部分
22         f[u][0] += max(f[j][1], f[j][0]);
23         f[u][1] += f[j][0];
24     }
25 }
```

# [3709] 数字转换

---

如果一个数 $x$ 的约数和（不包括它本身，下同）比它本身小，那么 $x$ 可以变成它的约数和；如果对于某个 $y > x$ 且 $y$ 的约数和为 $x$ ，那么 $x$ 也可以变成 $y$ 。例如，4可以变为3，1可以变为7。限定所有的数字变换在不超过 $n$ 的正整数范围内进行，求不断进行数字变换且没有重复数字出现的最多变换步数。

输入一个正整数 $n$ 。输出不断进行数字变换且没有重复数字出现的最多变换步数。

---

# 分析：

---

由于任意正整数  $x$ ，的 **约数之和** 是 **唯一的**，且本题要求只有约数之和 **小于** 自身才能转换

故对于所有的  $x$  来说，他向 **小于** 自己的数转换的边 **至多** 只有一条，那就是  $x$  的 **约数之和**  $x' (x' < x)$

这样一个图论模型我们就很熟悉了，我们把每一个 **数** 看作一个 **点**，把上述这个 **转换** 看作该点的 **入边**

每一个 **点**，**至多只有一条** 入边，这就是 **森林** 呀

建好图后，本题就 **等价于** 找出一个森林中，**直径最长** 的树，并求出该树的 **直径**

---

# 分析：

---

求解N以内的所有因子的和的时间复杂度为 $N * \sqrt{N}$ ，  
当 n 很大时，可以考虑用 “筛选” 的方式：

```
for (int i = 1; i <= n; i ++ )  
    for (int j = 2; j <= n / i; j ++ )  
        sum[i * j] += i;
```

sum[k] 存的是k的所有因子和，时间复杂度为  $n \log n$

---

# 代码1:

```
7  const int N = 5e4 + 10;
8  int h[N], e[N], ne[N], idx;
9  int st[N], d[N], res;
10 int n, sum[N];
11
12 void add(int a, int b)
13 {
14     e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
15 }
16
17 void build()
18 {
19     for (int i = 1; i <= n; i ++ )
20         for (int j = 2; j <= n / i; j ++ )
21             sum[i * j] += i; //求解因子和
22
23     memset(h, -1, sizeof h); //把所有小于n的值建图
24     for (int i = 2; i <= n; i ++ )
25         if (i > sum[i]) add(sum[i], i);
26 }
```

```
42 int main()
43 {
44     scanf("%d", &n);
45     build(); //建图
46     //可能存在多棵树, 所以要枚举每一个点
47     for (int i = 1; i <= n; i ++ )
48         if (!st[i]) dfs(i);
49     printf("%d\n", res);
50
51     return 0;
52 }
```

# 代码2:

```
28 void dfs(int x)
29 { //求解最长直径
30     st[x] = 1;
31     for (int i = h[x]; ~i; i = ne[i])
32     {
33         int y = e[i];
34         if (st[y]) continue ;
35         dfs(y);
36         res = max(res, d[x] + d[y] + 1);
37         d[x] = max(d[x], d[y] + 1);
38     }
39     res = max(res, d[x]);
40 }
```

# [3710] 战略游戏

Bob喜欢玩电脑游戏，特别是战略游戏。但是他经常无法找到快速玩过游戏的办法。现在他有个问题。

他要建立一个古城堡，城堡中的路形成一棵树。他要在这棵树的结点上放置最少数目的士兵，使得这些士兵能了望到所有的路。

注意，某个士兵在一个结点上时，与该结点相连的所有边将都可以被瞭望到。

请你编一程序，给定一树，帮Bob计算出他需要放置最少的士兵。

输入第一行  $N$ ，表示树中结点的数目。第二行至第 $N+1$ 行，每行描述每个结点信息，依次为：该结点标号 $i$ ， $k$ （后面有 $k$ 条边与结点 $i$ 相连）。接下来 $k$ 个数，分别是每条边的另一个结点标号 $r_1, r_2, \dots, r_k$ 。对于一个 $n$  ( $0 < n \leq 1500$ ) 个结点的树，结点标号在 $0$ 到 $n-1$ 之间，在输入数据中每条边只出现一次。输出文件仅包含一个数，为所求的最少的士兵数目

# 题意：

4

0 1 1

1 2 2 3

2 0

3 0

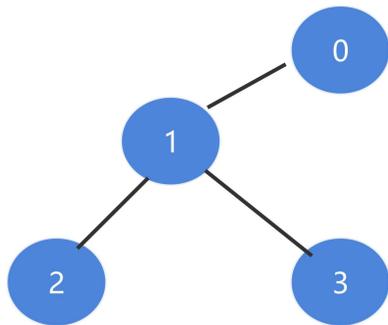
//该结点标号 $i$ ,  
 $k$ (后面有 $k$ 条边  
与结点 $i$ 相连)

给定一棵包含  $n$  个结点的 **树**，以及 **树** 上的  $n - 1$  条边

我们需要在这  $n$  个结点中，选择一定数量的结点放上 **哨兵**

最终要求，树中任意  $n - 1$  条边的左右两端，至少有一个结点上放置了 **哨兵**

求解一个 **方案**，该方案满足上述要求，且放置的哨兵数量 **最少**，输出该 **方案** 放置的 **哨兵数量**



# 分析：



显然我们不能暴力枚举树中所有结点选(1)与不选(0)的状态(时间复杂度为  $O(2^n)$ )。这题和没有上司的舞会本质相同。因此，会想到采用DP思想：考虑以结点  $u$  为根节点的子树，该子树所有的方案，可以由当前结点放或不放哨兵进行划分阶段。

**状态表示** —  $f_{i,1/0}$  **集合**：以结点  $i$  为根节点的子树，在  $i$  上放置哨兵(1)和不放哨兵(0)的方案

**状态表示** —  $f_{i,1/0}$  **属性**：方案中，放置的哨兵数量最少

# 分析:



**状态表示** —  $f_{i,1/0}$  **集合**: 以结点  $i$  为 **根节点** 的子树, 在  $i$  上**放置**哨兵(1) 和**不**放哨兵(0) 的**方案**

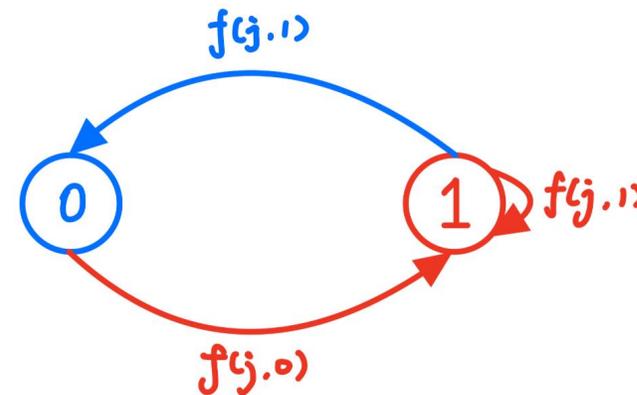
**状态表示** —  $f_{i,1/0}$  **属性**: 方案中, 放置的哨兵数量 **最少**

在  $i$  上**放置**哨兵(1):

$$f_{i,1} = \sum_{i_{ch}} \min(f_{i_{ch},0}, f_{i_{ch},1}) \quad (i_{ch} \in \{ver | ver \text{ 是 } i \text{ 的子节点}\})$$

在  $i$  上**不放**哨兵(0):

$$f_{i,0} = \sum_{i_{ch}} f_{i_{ch},1} \quad (i_{ch} \in \{ver | ver \text{ 是 } i \text{ 的子节点}\})$$



```
7  const int N = 1510;
8
9  int n;
10 int h[N], e[N], ne[N], idx;
11 int f[N][2];
12 bool not_root[N];
13
14 void add(int a, int b)
15 {
16     e[idx] = b;
17     ne[idx] = h[a];
18     h[a] = idx ++ ;
19 }
```

```
29 int main()
30 {
31     cin>>n;
32     memset(h, -1, sizeof h); idx = 0;
33     memset(not_root, 0, sizeof not_root);
34     for (int i = 0; i < n; i ++ )
35     {
36         int a, b, siz;
37         cin>>a>>siz;
38         while (siz -- )
39         {
40             cin>>b;
41             add(a, b);
42             not_root[b] = true;
43         }
44     }
45     int root = 0;
46     while (not_root[root]) root ++ ;
47     dfs(root);
48     printf("%d\n", min(f[root][0], f[root][1]));
49     return 0;
50 }
```

```
21 void dfs(int u)
22 {
23     f[u][0] = 0, f[u][1] = 1; //initialize
24     for (int i = h[u]; ~i; i = ne[i])
25     {
26         int j = e[i];
27         dfs(j);
28         f[u][0] += f[j][1];
29         f[u][1] += min(f[j][0], f[j][1]);
30     }
31 }
```