信息学奥林匹克竞赛

Olympiad in Informatics

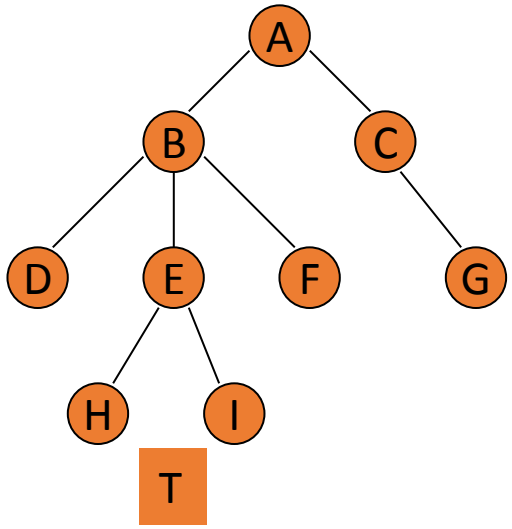
树形动态规划

陈 真

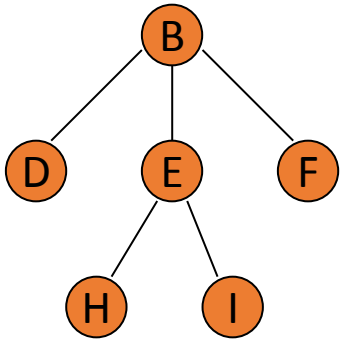
太原市第五中学校

树及其相关概念

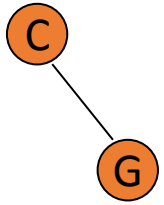
树（tree）是**树形结构**的简称。它是一种重要的非线性数据结构。树——或者是一棵空树，即不含结点的树，或者是一棵非空树，即至少含有一个结点的树。在一棵非空树中，它有且仅有一个称作根（root）的结点，其余的结点可分为m棵（m≥0）互不相交的子树（即称作根的子树），每棵子树（subtree）又同样是一棵树。显然，树的定义是递归的，树是一种递归的数据结构。树的递归定义，将为以后实现树的各种运算提供方便。



T

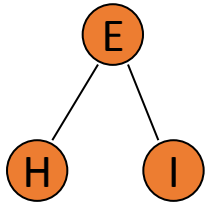


T1



T2

T11



T12

T13

树及其相关概念

1.结点的度和树的度

每个结点具有的子树数或者说后继结点数被定义为该**结点的度（degree）**。所有结点的度的最大值被定义为该树的度。

2.分支结点和叶子结点

度大于0的结点称作分支结点或非终端结点，度等于0的结点称作叶子结点或终端结点。在分支结点中，又把度为1的结点叫做**单分支结点**，度为2的结点叫做**双分支结点**，其余以此类推。

3.孩子结点、双亲结点和兄弟结点

每个结点的子树的根，或者说每个结点的后继，被习惯地称作该结点的孩子（child）或儿子，相应地，该结点被称作**孩子结点的双亲（parent）或父亲**。具有同一双亲的孩子互称**兄弟**（brothers）。每个结点的所有子树中的结点被称作该结点的子孙。每个结点的祖先则被定义为从树根结点到达该结点的路径上经过的所有结点。

树及其相关概念

4.结点的层数和树的深度

树既是一种递归结构，也是一种层次结构，树中的每个结点都处在一定的层数上。结点的层数（1evel）从树根开始定义，根结点为第一层，它的孩子结点为第二层，以此类推。树中结点的最大层数称为**树的深度（depth）或高度（height）**。

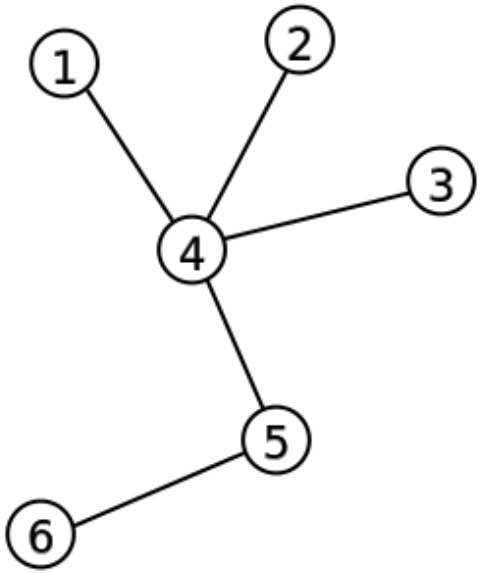
5.有序树和无序树

若树中各结点的子树是按照一定的次序从左向右安排的，则称之为有序树，否则称之为无序树。如对于一棵反映父子关系的家族树，兄弟结点之间是按照排行大小有序的，所以它是一棵有序树。以后若不特别指明，均认为树是有序的。

6.森林

森林是m（m≥0）棵互不相交的树的集合。例如，对于树中每个分支结点来说，其子树的集合就是森林。在图1的树T中，由A结点的子树所构成的森林为{T1，T2}，由B结点的子树所构成的森林为{T11，T12，T13}，等等。

树及其相关概念



树是连通且无环的无向图

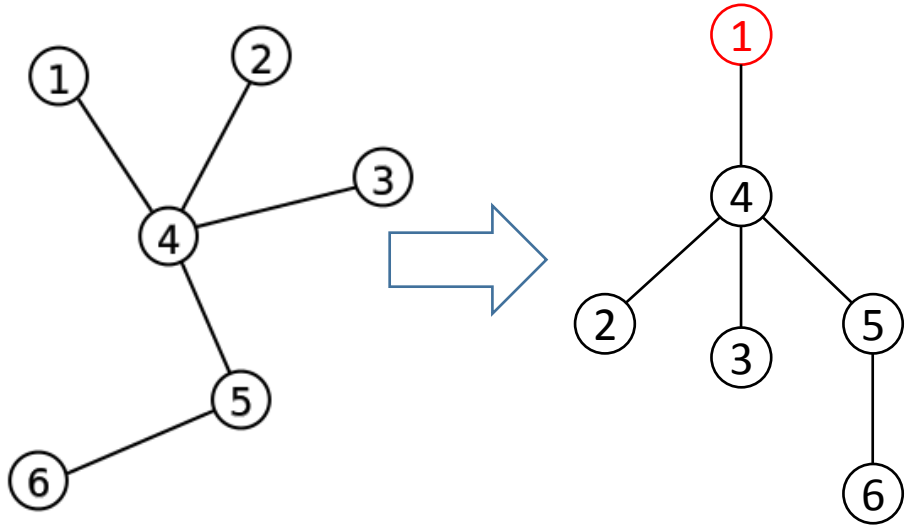
等价条件：

连通，且含有n个点、n-1条边，任意两点间恰有一条路径

无根树：满足上述定义、没有其他限制，每个节点的地位是相同的

当做普通的无向图来处理

有根树：在定义的基础上、指定一个节点为“根”。相比无根树，有根树更多地利用树的性质，组织结构更加清晰，树上的问题一般先转化成有根树再解决。



树及其相关概念

有根树的结构

描述有根树的结构：

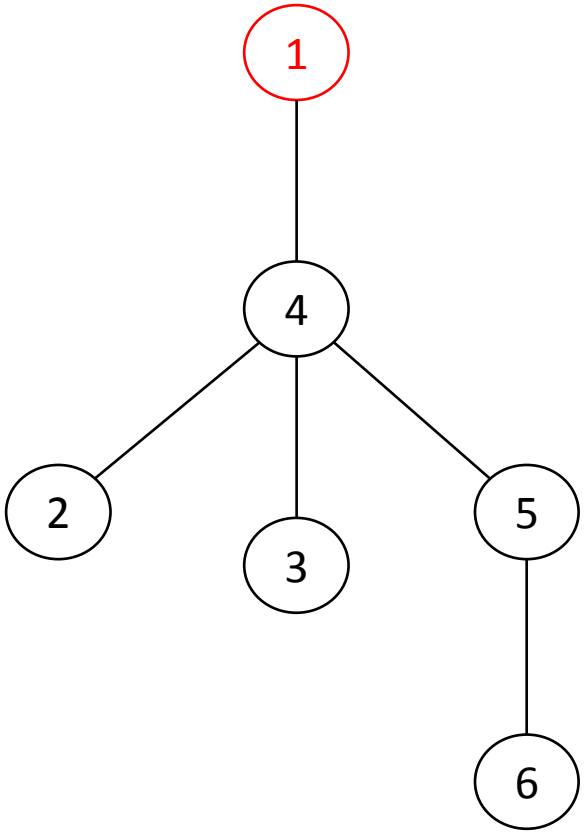
若a在b到根的路径上(如a=1,b=3; a=4,b=6)，

则称a是b的祖先、b是a的子孙

特殊地，若a是b的祖先、且a和b相邻(如a=1,b=4; a=5,b=6)，则称a是b的父亲(父节点)，b是a的儿子(子节点)

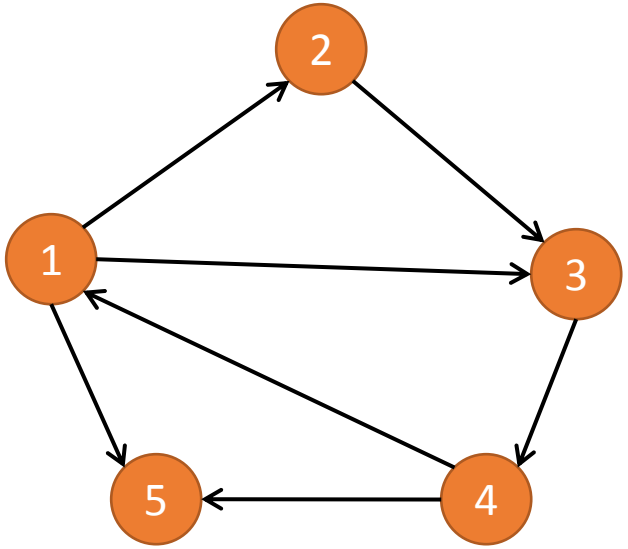
一个节点及其所有子孙节点组成一棵子树

深度：根据需要定义根的深度为0或1；每走过一条向下的边深度+1



树的链式前向星存储

前向星



前向星是一种特殊的边集数组，把数组中每一条边按起点从小到大排序，如果起点相同，按终点从小到大排序。

记录下以某个点为起点的所有边在数组中的起始位置和存储长度，前向星就构造好了。

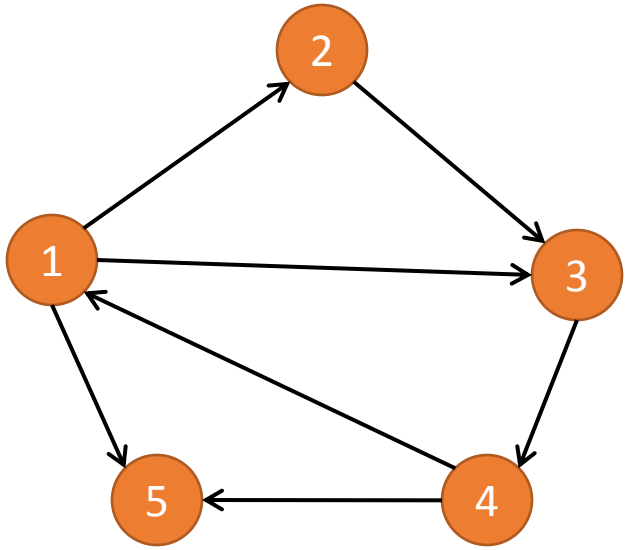
head[i] 记录以i为边集数组中的第一个存储位置。

len[i] 来记录所有以i为起点的边在数组中的存储长度

编号	1	2	3	4	5	6	7
起点u	1	1	1	2	3	4	4
终点v	2	3	5	3	4	1	5
权值w	1	1	1	1	1	1	1

树的链式前向星存储

前向星



head[i] 记录以i为编辑数组中的第一个存储位置

len[i] 来记录所有以i为起点的边在数组中的存储长度

head[1]=1
len[1]=3

head[2]=4
len[2]=1

head[3]=5
len[3]=1

head[4]=6
len[4]=2

排序一般用快排

O(nlog(n))

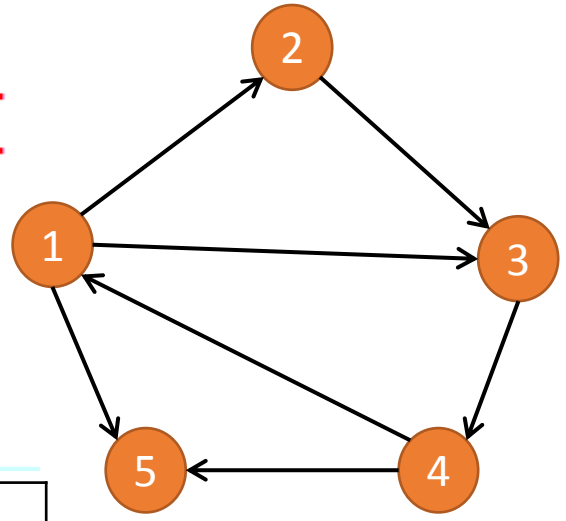
前向星已经构建完毕

编号	1	2	3	4	5	6	7
起点u	1	1	1	2	3	4	4
终点v	2	3	5	3	4	1	5
权值w	1	1	1	1	1	1	1

树的链式前向星存储

链式前向星

```
struct edge{
    int next;
    int to ;
    int w;
};
```



edge[i].to 表示 第i条边的终点

edge[i].next 表示 与第i条边同期点的下条边的存储位置

edge[i].w 表示 边的权值

编号	u	v
1	1	2
2	2	3
3	3	4
4	1	3
5	4	1
6	1	5
7	4	5

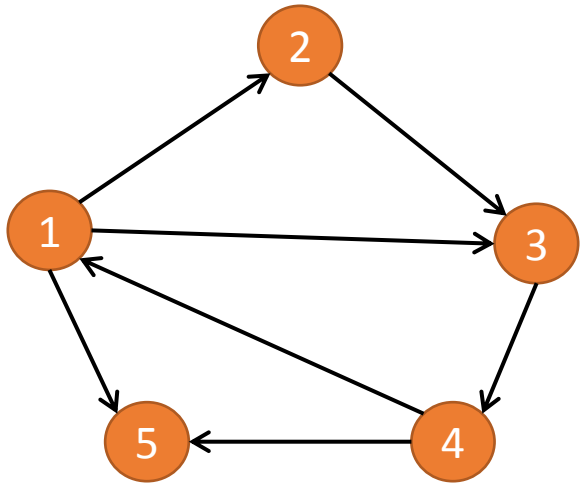
head[1]	6
head[2]	2
head[3]	3
head[4]	7
head[5]	0

边号	edge[i].to	edge[i].next	head[i]
1	2	0	head[1]=1
2	3	0	head[2]=2
3	4	0	head[3]=3
4	3	1	head[1]=4
5	1	0	head[4]=5
6	5	4	head[1]=6
7	5	5	head[4]=7

树的链式前向星存储

链式前向星

```
struct edge{
    int next;
    int to ;
    int w;
};
```




```
void add(int u,v,w){
    edge[cnt].w=w;
    edge[cnt].to=v;
    edge[cnt].next=head[u];
    head[u]=cnt++;
}
```

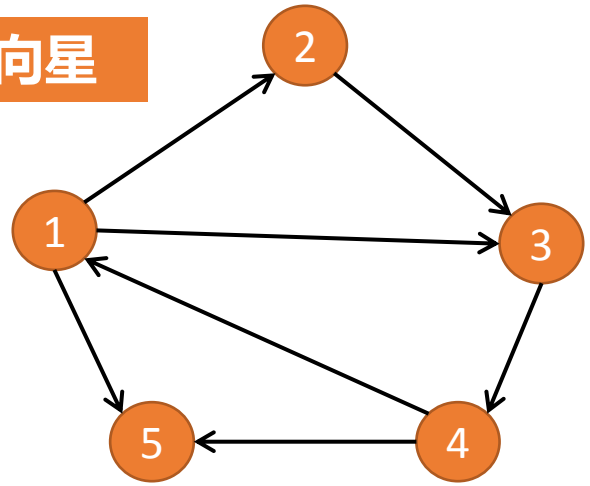
编号	u	v
1	1	2
2	2	3
3	3	4
4	1	3
5	4	1
6	1	5
7	4	5

head[1]	6
head[2]	2
head[3]	3
head[4]	7
head[5]	0

边号	edge[i].to	edge[i].next	head[i]
1	2	0	head[1]=1
2	3	0	head[2]=2
3	4	0	head[3]=3
4	3	1	head[1]=4
5	1	0	head[4]=5
6	5	4	head[1]=6
7	5	5	head[4]=7

树的链式前向星存储

链式前向星




```
for (int i = head[x]; i; i = edge[i].nxt) {
//查询该顶点的邻接点
    int y = edge[i].to;
    if (v[y]) continue;
    … …
}
```

编号	u	v
1	1	2
2	2	3
3	3	4
4	1	3
5	4	1
6	1	5
7	4	5

head[1]	**6**
head[2]	2
head[3]	3
head[4]	7
head[5]	0

边号	edge[i].to	edge[i].next	head[i]
1	2	0	head[1]=1
2	3	0	head[2]=2
3	4	0	head[3]=3
4	3	1	head[1]=4
5	1	0	head[4]=5
6	5	4	head[1]=6
7	5	5	head[4]=7

树的遍历

【试一试】借助链式前向星完成树的深度优先遍历和广度优先遍历

```
void dfs(int t,int x) {
    v[x] = 1;       //标记该结点已遍历
    pre[t]=x;  //记录路径
    if(x==0) { print(t);  return;  }
    for (int i = head[x]; i; i = edge[i].nxt) {
    //查询该顶点的邻接点
        int y = edge[i].to;
        if (v[y]) continue;
        dfs(t+1,y);
        v[y]=0;  //回溯
    }
}
```

```
void bfs(int x) {
    int h=0, t=0;
    memset(q,0,sizeof(q)); //队列初始化
    t++;
    q[t]=x;
    v[x]=1;
    while(h!=t){
        h++;
        for (int i=head[q[h]];i;i=edge[i].nxt) {
            int y=edge[i].to;
            if(!v[y]){
                t++;
                q[t]=y;
                v[y]=1;
            }
        }
    }
}
```

基础树型动态规划

树型dp 一定要确定好**树的结构**，一定是**有根树**(无根树随便选择一个节点当做有根树)。

一般的*dp 的状态设计为：**设f[i] 表示i 节点的子树的答案，根据实际情况会附加额外的维度来维护更多的情况。***

转移一般考虑清楚如何将两棵子树的答案合并成一个。

只要想清楚了两个的合并，那么每次新增一个子树合并就可以了。

【关键】在树型dp 中，**如何将不同的子树的结果合并到当前的根节点上来就是思考的关键。**

而由于要消除后效性，所以一定是按照dfs 序来逆序进行处理。一般实现时，通过dfs，在回朔时更新根节点答案即可。

基础树型动态规划

【例题01】树上统计（题目来源：洛谷U142375）

【问题描述】给定一棵有n个结点的有根树，所有的结点用1-n编号，保证除根结点外，**每个结点都有一个唯一的父亲结点**。一棵子树定义为从某个结点出发能到达的所有结点的集合，显然，这棵树一共有n棵子树。请问，这棵树上有多少棵子树，它所包含的**结点的编号是一个连续的整数区间**。

【输入格式】第一行有一个整数 n，表示这棵树的总结点数。（1<=n<=100000）

接下来n-1行，每行两个整数 u，v，表示存在一条从 u结点到v结点的有向边。

【输出格式】输出一个整数。

样例输入	样例输出
5 2 3 2 1 2 4 3 5	4

基础树型动态规划

【例题01】树上统计（题目来源：洛谷U142375）

一棵子树的结点编号要是一个连续的整数区间，则该子树内结点编号的最大值-结点编号最小值+1=子树结点个数。因此，对每棵子树记录这三个值即可。

阶段划分 树形DP中，**阶段划分以子树为单位进行**（**子树从小到大、方向从叶子到根**），设阶段变量u代表以u为根的子树。

设计状态 设Max[u]代表以u为根的子树中结点编号的最大值，Min[u]代表以u为根的子树中结点编号的最小值

f[u]代表以u为根的子树的结点个数。

确定决策 树形DP中，状态转移常常考虑孩子可以为父亲提供何种信息。

转移方程 假设u的儿子为vi（1<=i<=x），则**Max[u]=max(u,Max[vi])，Min[u]=min(u,Min[vi]),f[u]=1+∑f[vi]。**

确定边界 初始时，**Max[u]=u，Min[u]=u，f[u]=1**，即以u为根的子树中结点编号的最大值为u，最小值为u，结点个数为1。

确定目标 对于每棵子树，若以u为根的子树中最大结点编号减去最小结点编号加1等于结点个数，即Max[u]-Min[u]+1=f[u]，则以u为根的子树便是一棵合法的子树，ans自增，最终ans即为目标解。

基础树型动态规划

【例题01】树上统计（题目来源：洛谷U142375）

//Max[u],Min[u],f[u]记录以u为根的子树结点编号最大值、最小值、结点总数

//d[u]记录结点u的入度

```
int head[100005],cnt=0;
void add(int from,int to){
    cnt++;
    e[cnt].next=head[from];
    e[cnt].to=to;
    head[from]=cnt;
}
```

```
18  int ans=0;//记录答案
19  void dfs(int u){
20      Max[u]=Min[u]=u;
21      f[u]=1;
22      for(int i=head[u];i!=0;i=e[i].next){
23          int v=e[i].to;
24          dfs(v);
25          Max[u]=max(Max[u],Max[v]);
26          Min[u]=min(Min[u],Min[v]);
27          f[u]+=f[v];
28      }
29      if(Max[u]-Min[u]+1==f[u])ans++;
30  }
```

基础树型动态规划

树形DP：子树信息的归并

树形DP的一般形式都是对子树的信息进行归并，其本质是dfs/bfs过程中的统计。有些时候，还会涉及组合等方面的知识。

举个栗子，O(n)求一棵树所有节点的siz。

节点的siz为以该节点为根的子树大小（节点数）。

显然，我们可以对树进dfs，然后统计子树。

```
1  int dfs(int x,int father)
2  {
3      size[x]=1;
4      int i,j;
5      for(int i=head[x];i;i=edge[i].next)
6      {
7          j=edge[i].to;
8          if(j==father) continue;
9          size[x]+=dfs(j,x);
10     }
11     return size[x];
12 }
```

基础树型动态规划

树形DP：子树信息的归并

树形dp的性质：递归求解&子树合并。　　树形DP的一般模板。

```
1  int dfs(int x,int father)
2  {
3      int j;
4      for(int i=head[x];i;i=edge[i].next)
5      {
6          j=edge[i].to;
7          if(j==father) continue;
8          dfs(j,x);
9          dp[x]=...;
10     }
11     return dp[x];
12 }
```

基础树型动态规划

树形DP：子树信息的归并

【题目02】树形DP入门例题01

【问题描述】给定一棵n个点的无权树，问树中每个节点的深度和每个子树的大小？（以1号点为根节且深度为0）

【输入格式】第1行：n。

第2~n行：每行两个数x,y，表示x,y之间有一条边。

【输出格式】

n行，每行输出格式为：#节点编号 deep：深度 count：子树节点数（详见样例）

【样例输入输出】

【数据范围】15%: n<=10;40% n<=1000;100% n<=100000;

【样例输入】

```
7
 1 2
2 3
1 4
3 5
1 6
 3 7
```

【样例输出】

```
#1 deep:0 count:7
#2 deep:1 count:4
#3 deep:2 count:3
#4 deep:1 count:1
#5 deep:3 count:1
#6 deep:1 count:1
#7 deep:3 count:1
```

基础树型动态规划

树形DP：子树信息的归并

【题目02】树形DP入门例题01

```
void dfs1(int x) {
    v[x]=1;
    for(int i=head[x];i;i=edge[i].nxt){
        int y=edge[i].to;
        if(v[y])  continue;
        d[y]=d[x]+1;
        dfs1(y);
    }
}
```

```
void dfs(int x) {
    size[x]=1;
    v[x]=1;
    for(int i=head[x];i;i=edge[i].nxt){
        int y=edge[i].to;
        if(v[y]) continue;
        dfs(y);
        size[x]+=size[y];
    }
}
```

基础树型动态规划

树形DP：子树信息的归并

【题目03】树形DP入门例题02

【问题描述】给定一颗n(n<=10^5)个点的点权树，问树中每个子树的点权和，点权最大值。

【输入格式】一行输入一个正整数n（2≤n≤ 10^5）;

第二行输入n个正整数d(1≤d≤2700000000),表示每点的权值。

第三行到第n+1行输入每条边的信息，每行输入两个整数u（1≤u≤n）,v（1≤v≤n）。分别表示边的起点、终点。数据保证输入的是一棵树。

【输出格式】

共两行; 第一行输出n个数,表示当前点所包含最大子树的点权和;

第二行输出n个数,表示当前点所包含最大子树的最大点权;

不过滤行末空格

【样例输入输出】

【输入样例】

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

【输出样例】

```
25 10 14 8 7
8 7 8 8 7
```

基础树型动态规划

树形DP：子树信息的归并

【题目03】树形DP入门例题02

```
//y是x 的父亲节点
void dfs(int x,int y) {
    Count[x]=a[x];
    deep[x]=a[x];
    for (int i=head[x];i;i=edge[i].nxt) {
        if (edge[i].to!=y) {
            dfs(edge[i].to,x);
            Count[x]+=Count[edge[i].to];
            deep[x]=max(deep[x],deep[edge[i].to]);
        }
    }
}
```

基础树型动态规划

树形DP：子树信息的归并

【题目04】最大子树和

【问题描述】小明对数学饱有兴趣，并且是个勤奋好学的学生，总是在课后留在教室向老师请教一些问题。一天他早晨骑车去上课，路上见到一个老伯正在修剪花花草草，顿时想到了一个有关修剪花卉的问题。于是当日课后，小明就向老师提出了这个问题：

一株奇怪的花卉，上面共连有 N朵花，共有N-1条枝干将花儿连在一起，并且未修剪时每朵花都不是孤立的。每朵花都有一个“美丽指数”，该数越大说明这朵花越漂亮，也有“美丽指数”为负数的，说明这朵花看着都让人恶心。所谓“修剪”，意为：去掉其中的一条枝条，这样一株花就成了两株，扔掉其中一株。经过一系列“修剪“之后，还剩下最后一株花（也可能是一朵）。老师的任务就是：通过一系列“修剪”（也可以什么“修剪”都不进行），使剩下的那株（那朵）花卉上所有花朵的“美丽指数”之和最大。

老师想了一会儿，给出了正解。小明见问题被轻易攻破，相当不爽，于是又拿来问你。

基础树型动态规划

树形DP：子树信息的归并

【题目04】最大子树和

【输入格式】第一行一个整数N(1≤N≤16000)。表示原始的那株花卉上共N朵花。

第二行有N个整数，第III个整数表示第III朵花的美丽指数。

接下来N−1行每行两个整数a,b，表示存在一条连接第a 朵花和第b朵花的枝条。

【输出格式】一个数，表示一系列“修剪”之后所能得到的“美丽指数”之和的最大值。保证绝对值不超过2147483647。

【样例输入】

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

【样例输入】

```
3
```

【数据规模与约定】

对于60%的数据，有N≤1000；

对于100%的数据，有N≤16000。

基础树型动态规划

树形DP：子树信息的归并

【题目04】最大子树和

【分析】无根树中选择连通块，任意定一个根对答案是没有影响的。考虑到最优策略中，每个子树都是最优决策，所以具有最优子结构

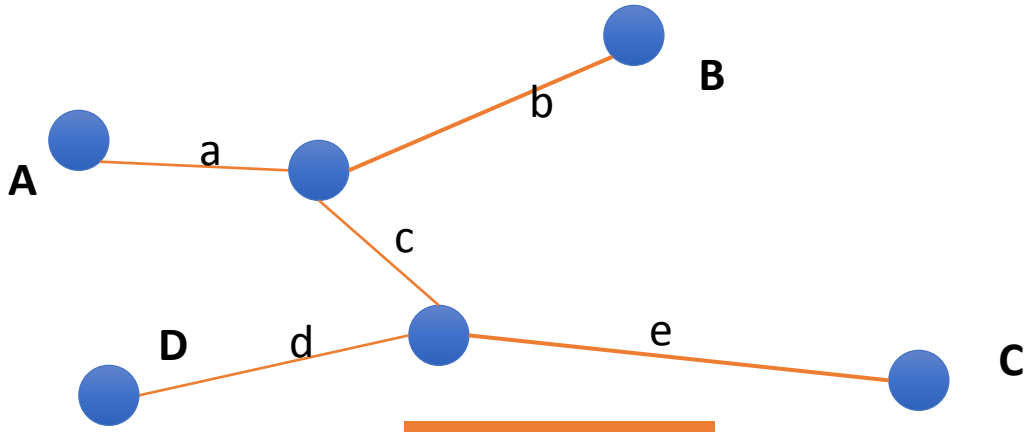
可以使用dp

```
void dfs(int x,int father){
    dp[x]=edge[x].w;
    for(int i=head[x];i;i=edge[i].nxt){
        int y=edge[i].to;
        if(y==father) continue;
        dfs(y,x);
        if(dp[y]>0) dp[x]+=dp[y];
    }
}
```

树形动态规划的应用

树的直径

给定一棵树，树中每条边都有一个权值，树中两点之间的距离定义为连接两点的路径边权之和。树中最远的两个节点之间的距离被称为**树的直径**，连接这两点的路径被称为**树的最长链**。后者通常也可称为直径，即直径是一个数值概念，也可代指一条路径



【备注】树是没有环路的连通图，任意两点都有一条通路，而且也只有一条通路

双向搜索

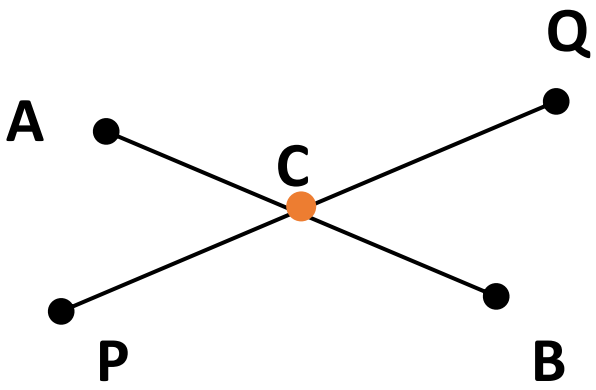
【步骤1】：任选一点x为根，bfs/dfs,更新到x的最远路径，记录到达点p

【步骤2】：以点p为根,bfs/dfs,更新到p的最远路径，记录到达点q,则从p到q即为树的直径

算法依赖于一个**性质**：**对于树中的任一个点，距离它最远的点一定是树上一条直径的一个端点。**

树形动态规划的应用

树的直径



从任意一点P出发，找离它最远的点Q，再从点Q出发，找离它最远的点W，

W到Q的距离就是是的直径

证明：

① **若P已在直径上**，根据树的直径的定义可知Q也在直径上且为直径的一个端点

②**若P不在直径上**，用反证法，假设此时**WQ不是直径**，AB是直径.

★**若AB与PQ有交点C**，由于P到Q最远，那么**PC+CQ>PC+CA**，所以**CQ>CA**，易得**CQ+CB>CA+CB**，即**CQ+CB>AB**，与**AB是直径矛盾**，不成立如图（其中AB，PQ不一定是直线，画成直线是为了方便）

★**若AB与PQ没有交点**，M为AB上任意一点，N为PQ上任意一点。首先还是NP+NQ>NP+MN+MB，同时减掉NP，得NQ>MN+MB，易知NQ+MN>MB，所以NQ+MN+MA>MB+MA，即NQ+MN+MA>AB，与AB是直径矛盾，所以这种情况也不成立

树形动态规划的应用

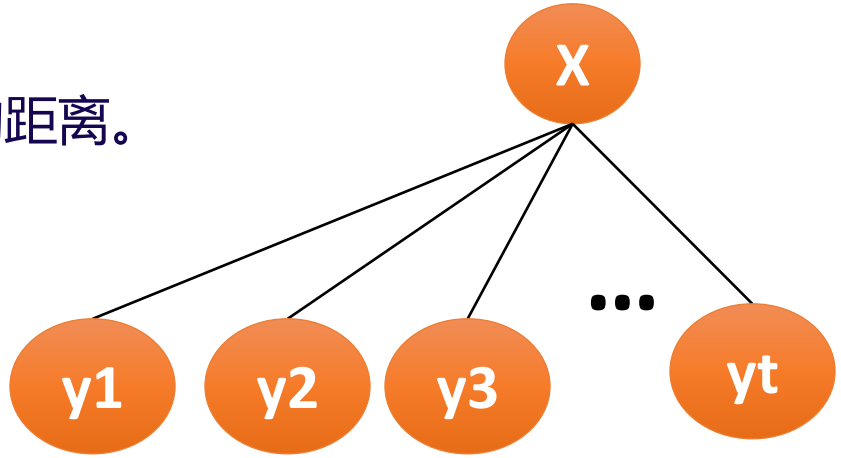
树的直径

设1号节点为根,"N个点N-1条边的无向图"就可以看做“有根树”

设d[x]表示从节点x出发走向以x为根的子树，能够到达的最远节点的距离。

设x的子节点为y1,y2, y3, ..., yt，edge(x, y)表示边权，显然有"

$$d[x] = \max_{1<=i<=t} \{d[yi] + edge(x, yi)\}$$



树型DP

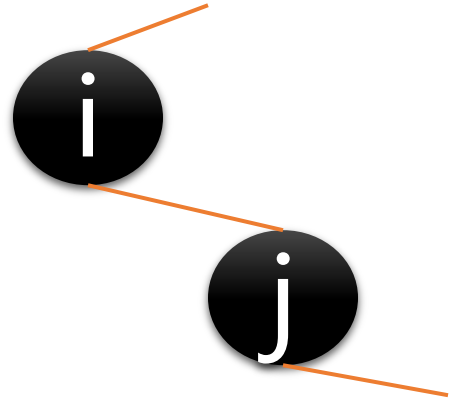
对每个节点x求出“经过节点x的最长链的长度”f[x]，次长链g[x]

整棵树的直径就是 $\max_{1\le x\le n}\{F[x]+G[x]\}$

如何求解F[x]？

树形动态规划的应用

树的直径



【思路】：对于每个节点我们要记录**两个值**：

f1 [i] 表示以 i 为根的子树中，i 到叶子结点距离的**最大值**

f2 [i] 表示以 i 为根的子树中，i 到叶子结点距离的**次大值**

对于一个节点，它到叶子结点距离的最大值和次大致所经过的路径肯定是不一样的

若j是i的儿子，那么（下面的 edge [i][j] 表示 i 到 j 的路径长度）：

若 f1 [i] < f1 [j] + edge[i][j]，则f2 [i] = f1 [i]，f1 [i] = f1 [j] + edge [i][j]；

否则，若 f2 [i] < f1 [j] + edge [i][j]，则f2 [i] = f1 [j] + edge [i][j]；

【说明】：如此，先看能否更新最大值，若能，它的次大值就是原先的最大值，再更新它的最大值；

若不能，就看能不能更新次大值，若能，就更新，不能就不管它

最后的答案： answer = max { f1 [i] + f2 [i] }

树形动态规划的应用

树的直径

```
void dp(int x,int father)
{
    int i,j;
    for(i=head[x];i;i=edge[i].next)
    {
        j=edge[i].to;
        if(j==father)  continue;
        dp(j,x);
        if(f1[x]<f1[j]+edge[i].w)   f2[x]=f1[x],f1[x]=f1[j]+edge[i].w;
        else if(f2[x]<f1[j]+edge[i].w)  f2[x]=f1[j]+edge[i].w;
    }
    ans=max(ans,f1[x]+f2[x]);

}
```

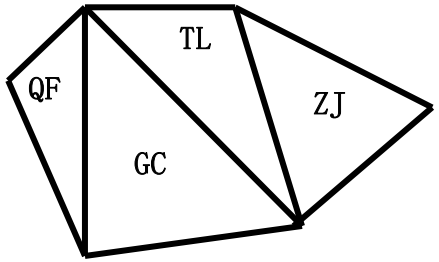
树形动态规划的应用

树的直径

【题目05】旅游（题目来源：ZJOI2012）

【问题描述】到了难得的暑假，为了庆祝小白在数学考试中取得的优异成绩，小蓝决定带小白出去旅游。经过一番抉择，两人决定将T国作为他们的目的地。T国的国土可以用一个凸N边形来表示，N个顶点表示N个入境/出境口。T国包含N-2个城市，每个城市都是顶点均为N边形顶点的三角形(换而言之，城市组成了关于T国的一个三角剖分)。两人的旅游路线可以看做是连接N个顶点中不相邻两点的线段。

为了能够买到最好的纪念品，小白希望旅游路线上经过的城市尽量多。作为小蓝的好友，你能帮帮小蓝吗？



【输入格式】每个输入文件中仅包含一个测试数据。

第一行包含一个正整数N，N的含义如题目所述。

接下来有N-2行，每行包含三个整数 p,q,r，表示该城市三角形的三个顶点的编号(T国的N个顶点按顺时间方向从1至n编号)。

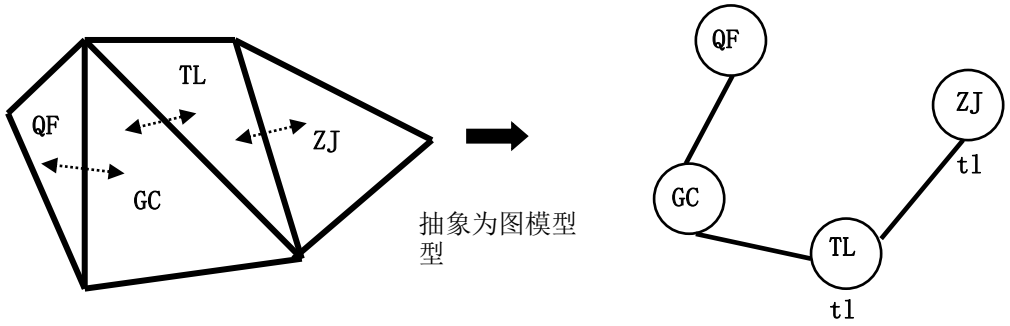
【输出格式】输出文件共包含1行，表示最多经过的城市数目。(一个城市被当做经过当且仅当其与线路有至少两个公共点)

输入样例	输出样例
6 1 2 4 2 3 4 1 4 5 1 5 6	4

树形动态规划的应用

树的直径

【题目05】旅游（题目来源：ZJOI2012）



连边转化图模型。

在建立图模型实际处理时，先将三角形三个顶点按照大小排序，一条边的两个顶点存入结构体内，并且用map记录第一次与该边接触的三角形（城市），当该边再次出现，说明该边是两个三角形（城市）的公共边，将这两个城市之间建边。

进一步思考，一个三角形抽象为一个点（城市），那么显然它最多与三个城市连边,并且在边界上的城市是不会有三个城市与它相连的，且**图中不存在环**，即两个三角形不可能有超过一条边的公共边。**基于这些性质，可确定该模型是一棵树。**

题目要求“旅游路线上经过的城市尽量多”，建立模型的边权是1，该问题转换为“在树上找一条最长的路径”，显然就是求树的直径。

树形动态规划的应用

树的直径:最长链与次长链

【题目06】树形DP入门例题05

【问题描述】大神给定一棵n个点的边权树，由于他太强了，所以他想考考你。让你求树中每个子树的最长链是多长？次长链是多长？（链可以长度为零）

注： 一条链指子树中节点到根节点的链,最长链和次长链必须是彼此没有相同边的两条独立的链

【输入格式】第一行输入一个T((1<<T)<=2^20)，表示有T组数据

对于每组数据，输入总共n行 。 第一行有两个整数n和S，分别表示总点数和树根。

第2 ~ n行每行有3个整数x,y,z，表示x到y有一条边，边权为z(0<=z<=10000)。

【输出格式】输出总共T*2行

对于每组数据，输出两行

第一行输出n个数，第i个数表示以i号点作为根节点的子树的最长链，数之间用2个空格隔开

第二行输出n个数，第i个数表示以i号点作为根节点的子树的次长链，数之间用2个空格隔开

【样例输入输出】

基础树型动态规划

树的直径:最长链与次长链

【题目06】树形DP入门例题05

f[x][0] 以x为根节点最长链，f[x][1]以x为根节点的次长链

【输入样例】

```
1
6 3
1 3 999
2 3 1
5 4 23
5 3 1
3 6 1
```

【输出样例】

```
0  0  999  0  23  0
0  0  24  0  0  0
```

```
void dfs(int x, int father) {
    f[x][0] =0; f[x][1] = 0;
    for (int i=head[x];i;i=edge[i].nxt) {
        int y=edge[i].to;
        if (y==father)continue;
        dfs(y,x);
        if (f[y][0]+edge[i].w>f[x][1])
            f[x][1]=f[y][0]+edge[i].w;
        if (f[x][1]>f[x][0])
            swap(f[x][1],f[x][0]);
    }
    return;
}
```

树形动态规划的应用

树的重心

若有一点，其所有子树中最大子树的节点数最少，则该点就是这棵树的重心。

一般的树只有一个重心，有些有偶数个节点的树，有两个节点。

求树的重心方法：

随意确定一个根节点，先把无根树转化为有根树，dfs求出所有点的子树的节点个数。如果有一点满足该点的子树的节点数的二倍大于等于总结点数(size[u]*2>=n)，并且该点的儿子都满足子树的节点数的二倍小于等于总结点数(size[son_u]*2<=n)，这个点就是树的重心。

树形动态规划的应用

树的重心

代码实现

```
void dfs(int u){
    size[u]=1;
    for(int i=head[u];i;i=edge[i].nxt){
        int v=edge[i].to;
        if(father[u]!=v){
            father[v]=u;
            dfs(v);
            size[u]+=size[v];
        }
    }
    if(size[u]*2>=n&&!ans) ans=u;
}
```

基础树型动态规划

树的重心

【题目07】树形DP入门例题03

【问题描述】给定一个n个点的无权树，求树的重心？ 重心定义为，删去该点之后，树中的所有连通块的最大尺寸最小。

【输入格式】第一行一个n。

接下来n-1行，每行分别有两个数a和b，表示节点a到节点b有一条无向边。

【输出格式】

求这棵树的重心（保证唯一）

【样例输入输出】

【输入样例】

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

【输出样例】

```
1
```

基础树型动态规划

【题目07】树形DP入门例题03

```
//找到一个点，使得把树变成以该点为根的有根树时
//最大子树的结点数最小。
void find_rt(int u, int pre){
    size[u]=1;//结点本身
    int maxSub=0; //节点u的最大子树节点个数
    for(int i=head[u];i;i=edge[i].nxt){
        int v=edge[i].to;
        if(v==pre) continue;
        find_rt(v,u);
        size[u]+=size[v];
        maxSub=max(maxSub,size[v]);
    }
    maxSub=max(maxSub,n-size[u]);
    if (Mn>maxSub)
        Mn = maxSub, root = u;
}
```

树形动态规划的应用

树的重心

【题目08】Kay And Snowflake（题目来源：CF685B）

【问题描述】给定一棵有n个结点的有根树，树根为1号结点。有q次询问，对于第i次询问，你需要回答以结点vi为根的这棵子树的重心是哪个结点？

【输入格式】第一行包含两个正整数n和q（1<=n,q<=300000），分别代表树的结点个数和询问次数。

第二行包含n-1个整数$p_2,p_3,\ldots,p_n$，代表结点2~n的父亲结点。数据保证这是一棵以1号结点为树根的有根树。

接下来q行，每行包含1个整数vi，代表询问的结点。

【输出格式】输出共q行，每行一个整数，代表每次询问的答案。

样例输入	样例输出
7 4 1 1 3 3 5 3 1 2 3 5	3 2 3 6

算法一：对应每次询问，可以从询问结点vi开始DFS，按照树的重心求解算法计算以vi为根的子树的重心，时间复杂度O(nq)

树形动态规划的应用

树的重心

【题目08】Kay And Snowflake（题目来源：CF685B）

算法二：设f[u]代表以u为根的子树的结点总数，g[u]代表以u为根的子树的重心。

若u为叶子结点，则g[u]=u。

若u为非叶子结点，根据树的重心性质可知，如果不存在f[vi]*2>=f[u]，那么g[u]=u；否则，假设u的孩子为vi，若这棵子树的重心为g[vi]，g[u]一定是g[vi]到u的路径上的某个结点，枚举结点x∈[g[vi],u)，若f[x] *2>=f[u]，则g[u]=x。这样我们便可一次dfs预处理g[u]，再在O(1)的时间内给出回答，时间复杂度为O(n+q)。

树形动态规划的应用

树的重心

【题目08】Kay And Snowflake（题目来源：CF685B）

//f[u]代表以u为根的子树中的结点总数

//g[u]代表以u为根的子树的重心

//p[u]代表u的父亲

```
17 void dfs(int u){
18     f[u]=1;
19     for(int i=head[u];i!=0;i=e[i].next){
20         int v=e[i].to;
21         dfs(v);
22         f[u]+=f[v];
23     }
24     int tmp=0;
25     for(int i=head[u];i!=0;i=e[i].next){
26         int v=e[i].to;
27         if(f[v]*2>=f[u]&&f[v]>tmp){
28             tmp=f[v];
29             //重心是g[v]-u路径中的某个结点
30             g[u]=g[v];
31             while(g[u]!=u){
32                 if(f[g[u]]*2>=f[u])break;
33                 g[u]=p[g[u]];
34             }
35         }
36     }
37     //重心是u
38     if(tmp==0)g[u]=u;
39 }
```

基础树型动态规划

边的贡献

【题目09】树形DP入门例题04

【问题描述】给定一棵 个节点的树，求其中每个点到其他节点的距离和。

【输入格式】第一行一个整数 n；

接下来 n-1行是三个数 x,y,z，表示 x到 y有一条长度为 z的边；

数据保证给出的是一棵树。

【输出格式】

n行，一行一个数，第 个表示点 到其他点的距离和。

【样例输入输出】

【输入样例】

```
3
1 2 3
1 3 5
```

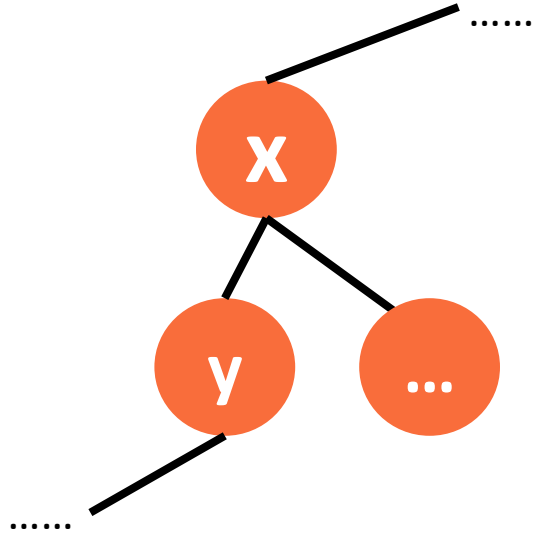
【输出样例】

```
8
11
13
```

基础树型动态规划

边的贡献

【题目09】树形DP入门例题04




```
void dfs1(int x,int father){
    //求出每棵子树的节点数量
    size[x]=1;
    //以x为根的子树中节点的个数
    ans+=deep[x];
    for (int i=head[x];i;i=edge[i].nxt){
        int y=edge[i].to;
        if (y!=father){
            deep[y]=deep[x]+edge[i].w;
            dfs1(y,x);
            size[x]+=size[y];
        }
    }
}
```

```
void dfs2(int x,int father){
    for (int i=head[x];i;i=edge[i].nxt){
        int y=edge[i].to;
        if (y!=father){
            dp[y]=dp[x]+(n-2*size[y])*edge[i].w;
            dfs2(y,x);
        }
    }
}
```

对于x和y的答案 他们的区别只有x到y那条边的累加次数不同，那么只要减去原本这条边的计算次数，再加上现在这条边的计算次数，就是y的答案，减去的其实就是num[y]，加上的就是n-num[y]

基础树型动态规划

边的贡献

【题目10】树上染色（题目来源：HAOI2015）

【问题描述】有一棵点数为 n 的树，树边有边权。给你一个在 0~n之内的正整数 k ，你要在这棵树中选择 k个点，将其染成黑色，并将其他 的 n−k个点染成白色。将所有点染色后，你会获得黑点两两之间的距离加上白点两两之间的距离的和的受益。问受益最大值是多少。

【输入格式】第一行包含两个整数 n,k。

第二到 n行每行三个正整数 fr,to,disf表示该树中存在一条长度为 dis 的边 (fr,to)。输入保证所有点之间是联通的。

输出格式输出一个正整数，表示收益的最大值。

树形背包

“背包”问题中学过一类特殊的背包问题：有依赖的背包。复杂依赖背包问题中，物品有多层依赖关系，若不存在循环依赖关系，物品之间的依赖关系形成了一棵树，此时也称为**树形背包**。

树形背包常常将某个结点u的若干棵子树看做若干个分组，因此以**u为根的子树**能得到的最大收益（最小代价），可当作**分组背包**处理。

树形背包问题求解时，重点关注树形背包问题的**阶段划分**、**状态设计与优化**。

树形背包

【题目11】选课

【问题描述】学校实行学分制。每门的必修课都有固定的学分，同时还必须获得相应的选修课程学分。学校开设了N（N<300）门的选修课程，每个学生可选课程的数量M是给定的。学生选修了这M门课并考核通过就能获得相应的学分。

在选修课程中，有些课程可以直接选修，有些课程需要一定的基础知识，必须在选了其它的一些课程的基础上才能选修。例如《Frontpage》必须在选修了《Windows操作基础》之后才能选修。我们称《Windows操作基础》是《Frontpage》的先修课。每门课的直接先修课最多只有一门。两门课也可能存在相同的先修课。每门课都有一个课号，依次为1，2，3，…。 例

课号	先修课号	学分
1	无	1
2	1	1
3	2	3
4	无	3
5	2	4

上例中1是2的先修课，即如果要选修2，则1必定已被选过。同样，如果要选修3，那么1和2都一定已被选修过。 你的任务是为自己确定一个选课方案，使得你能得到的学分最多，并且必须满足先修课优先的原则。假定课程之间不存在时间上的冲突。

树形背包

【题目11】选课

【输入格式】输入文件的第一行包括两个整数N、M（中间用一个空格隔开）其中1≤N≤300,1≤M≤N。以下N行每行代表一门课。课号依次为1，2，…，N。每行有两个数（用一个空格隔开），

第一个数为这门课先修课的课号（若不存在先修课则该项为0），

第二个数为这门课的学分。学分是不超过10的正整数。

【输出格式】输出文件只有一个数,实际所选课程的学分总数。

【输入格式】

```
7 4
2 2
0 1
0 4
2 1
7 1
7 6
2 2
```

【输出格式】

```
13
```

课号	先修课号	学分
1	无	1
2	1	1
3	2	3
4	无	3
5	2	4

树形背包

【题目11】选课

在树形背包问题中，阶段划分有两个部分：

阶段划分

（1）对整棵树而言，以**子树为单位划分阶段**，设阶段变量u表示以u为根的子树。

（2）对以u为根的子树而言，若这棵树共选修了j门课程，那么选择哪j门课程能使得该子树所获学分最大。

假设u的儿子为vi（1<=i<=x），以这些儿子为根的子树记为Vi，可以将Vi视作一个分组，问题转化为有x个分组，背包总体积为j（选修课程总数），如何分配j给每个分组可以获得最大收益？

可按**已处理分组数量（子树数量）划分**阶段，**设阶段变量i代表u的前i棵子树（前i个分组）**。

设计状态

设f[u][i][j]代表以u为根的子树中，u的前i棵子树（前i个分组）选修j门课程得到的最大学分。

确定决策

对于**状态f[u][i][j]**，决策对象为u的第i棵子树Vi（第i个分组），可以做出的决策是这棵子树选修课程数，设决策变量k代表这棵子树选修k门课程。

转移方程

f[u][i][j]=max(f[u][i-1][j-k]+f[vi][x][k])，0<=k<=j，f[u][i-1][j-k]即u的前i-1棵子树（前i-1个分组）选修j-k门课得到的最大学分，f[vi][x][k]即子树Vi选修k门课程得到的最大学分，x为子树Vi的儿子数量。

修改状态为f[u][j]代表以u为根的子树选修j门课程得到的最大学分，那么：f[u][j]=max(f[u][j-k]+f[vi][k])，0<=k<j

f[vi][k]代表子树Vi选修k门课程得到的最大学分

确定边界

对于以u为根的子树，f[u][0]=0，f[u][1]=a[u]，a[u]代表选修根结点u这门课程可以获得的学分，其余f[u][j]=-∞。

确定目标

max(f[root][j])，f[root][j]即整棵树选修j门课程的最大学分，root为树根。

树形背包

【题目11】选课

//f[u][j]代表以u为根的子树选修j门课程得到的最大学分

```
15 void dfs(int u){
16     f[u][0]=0;
17     f[u][1]=val[u];
18     for(int i=head[u];i!=0;i=e[i].next){
19         int v=e[i].to;
20         dfs(v);
21         for(int j=m+1;j>=1;j--){
22             for(int k=0;k<j;k++){
23                 f[u][j]=max(f[u][j],f[u][j-k]+f[v][k]);
24             }
25         }
26     }
27 }
```

基础应用：树形背包

【题目12】[HAOI2010]软件安装

【问题描述】现在我们的手头有N个软件，对于一个软件i，它要占用Wi的磁盘空间，它的价值为Vi。我们希望从中选择一些软件安装到一台磁盘容量为M的计算机上，使得这些软件的价值尽可能大（即Vi的和最大）。

但是现在有个问题：软件之间存在依赖关系，即软件i只有在安装了软件j（包括软件j的直接或间接依赖）的情况下才能正确工作（软件吗i依赖软件j）。幸运的是，一个软件最多依赖另外一个软件。如果一个软件不能正常工作，那么他能够发挥的作用为0。

我们现在知道了软件之间的依赖关系：软件i依赖Di。现在请你设计出一种方案，安装价值尽量大的软件。一个软件只能被安装一次，如果一个软件没有依赖则Di=0，这是只要这个软件安装了，它就能正常工作。

【输入格式】第1行：N,M（0<=N<=100,0<=M<=500）

第2行：W1,W2, … Wi, … ,Wn 第3行：V1,V2, … Vi, … ,Vn 第4行：D1,D2, … Di, … ,Dn

【输出格式】一个整数，代表最大价值。

基础应用：树形背包

【题目12】[HAOI2010]软件安装

分析：这是一道典型的树上背包问题

定义 f[i][j] 表示在以i号节点为根的子树中，选择体积和为j的物品，得到的最大价值

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最小点覆盖

最小点覆盖定义：对于树T(V,E)，最小点覆盖是指从V中选出尽量少的点组成一个集合，使得E中所有的边都与取出来的点相连。

求解最小点覆盖算法思想：将树中结点分为两个集合V1（点覆盖集）和V2（非点覆盖集），设f[u][0]代表u在集合V1中，以u为根的子树边被覆盖时，子树u中点覆盖集的结点个数；f[u][1]代表u不在V1中，以u为根的子树边被覆盖时，子树u中点覆盖集的结点个数。设v为u的儿子，则有：

```
1  int f[N][2];//f[u][0/1]含义如上所述
2  void dfs(int u,int fa){
3      f[u][0]=1;
4      f[u][1]=0;
5      for(int i=head[u];i!=0;i=e[i].next){
6          int v=e[i].to;
7          if(v==fa)continue;
8          dfs(v,u);
9          f[u][0]+=min(f[v][0],f[v][1]);
10         f[u][1]+=f[v][0];
11     }
12 }
```

①对于f[u][0]，u在V1中，则u连接的所有边均能被覆盖，v在不在V1中均可，因此f[u][0]+=min(f[v][0],f[v][1])。

②对于f[u][1]，u不在V1中，则u连接的所有边只能被其儿子覆盖，v必须在V1中，因此f[u][1]+=f[v][0]。

若根结点为rt，则最小点覆盖集结点个数为min(f[rt][0],f[rt][1])。

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最小点覆盖

【题目13】战略游戏（题目来源：洛谷P2016）

【问题描述】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方法。现在他有个问题。现在他有座古城堡，古城堡的路形成一棵树。他要在这棵树的结点上放置最少数目的士兵，使得这些士兵能够瞭望到所有的路。

注意：某个士兵在一个结点上时，与该结点相连的所有边都将能被瞭望到。请你编一个程序，给定一棵树，帮 Bob 计算出他最少要放置的士兵数。

【输入格式】第一行一个数 N ，表示树中结点的数目。

第二到第 N+1 行，每行描述每个结点信息，依次为该结点编号 i，数值 k，k 表示后面有 k 条边与结点 i 相连，接下来 k 个数，分别是每条边的所连结点编号 r1,r2,...,rk 。

对于一个有 N 个结点的树，结点标号在 0 到 N−1 之间，且在输入文件中每条边仅出现一次。保证输入是一棵树。

【输出格式】输出仅包含一个数，为所求的最少士兵数。

本问题为最小点覆盖模板

样例输入	样例输出
4 0 1 1 1 2 2 3 2 0 3 0	1

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最大独立集

对于树T(V,E)，最大独立集是指从V中选出尽量多的点组成一个集合，使得这些点之间没有边连接。

求解最大独立集算法思想：将树中结点分为两个集合V1（独立集）和V2（非独立集），设f[u][0]代表u在集合V1中，以u为根的子树中独立集结点个数；f[u][1]代表u不在V1中，以u为根的子树独立集结点个数。设v为u的儿子，则有：

```
1  int f[N][2];//f[u][0/1]含义如上所述
2  void dfs(int u,int fa){
3      f[u][0]=1;
4      f[u][1]=0;
5      for(int i=head[u];i!=0;i=edge[i].next){
6          int v=edge[i].to;
7          if(v==fa)continue;
8          dfs(v,u);
9          f[u][0]+=f[v][1];
10         f[u][1]+=max(f[v][1],f[v][0]);
11     }
12 }
```

①对于f[u][0]，u在V1中，则v不能在V1中，因此f[u][0]=∑f[v][1]。

②对于f[u][1]，u不在V1中，则v在不在V1中均可，因此f[u][0]=∑max(f[v][0],f[v][1])。

若根结点为rt，则最大独立集结点个数为max(f[rt][0],f[rt][1])。

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最大独立集

【题目14】没有上司的舞会

【问题描述】Ural大学有N个职员，编号为1~N。他们有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。每个职员有一个快乐指数。现在有个周年庆宴会，要求与会职员的快乐指数最大。但是，没有职员愿和直接上司一起与会。

【输入格式】第一行一个整数N。(1<=N<=6000)

接下来N行，第i+1行表示i号职员的快乐指数Ri。(-128<=Ri<=127)

接下来N-1行，每行输入一对整数L,K。表示K是L的直接上司。

最后一行输入0,0。

【输出格式】输出最大的快乐指数。

【输入输出样例】

【输入格式】

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

【输出样例】

```
5
```

基础应用：覆盖类问题

树的最小点覆盖、最大独立集、最小支配集

最大独立集

【题目14】没有上司的舞会

【分析】

树形DP入门题。

设dp[i][0/1]为以i号节点为根的子树中的最大收益，0表示i参加，1表示不参加。

```
dp[i][0]=sum{max(dp[son(i)][0], dp[son(i)][1])};
dp[i][1]=sum{dp[son(i)][0]}+val[i];
```

基础应用：覆盖类问题

树的最小点覆盖、最大独立集、最小支配集　最大独立集

【题目14】没有上司的舞会

```
void dfs(int u)
{
    for(int i=head[u];i;i=edge[i].nxt)
    {
        int v=edge[i].to;
        dfs(v);
        dp[u][0]+=max(dp[v][1],dp[v][0]);
        dp[u][1]+=dp[v][0];
    }
    dp[u][1]+=r[u];
}
```

基础应用：覆盖类问题

【题目15】[ZJOI2008]骑士)

【问题描述】 Z国的骑士团是一个很有势力的组织，帮会中汇聚了来自各地的精英。

他们劫富济贫，惩恶扬善，受到社会各界的赞扬。最近发生了一件可怕的事情，邪恶的Y国发动了一场针对Z国的侵略战争。战火绵延五百里，在和平环境中安逸了数百年的Z国又怎能抵挡的住Y国的军队。于是人们把所有的希望都寄托在了骑士团的身上，就像期待有一个真龙天子的降生，带领正义打败邪恶。

骑士团是肯定具有打败邪恶势力的能力的，但是骑士们互相之间往往有一些矛盾。**每个骑士都有且仅有一个自己最厌恶的骑士（当然不是他自己）**，他是绝对不会与自己最厌恶的人一同出征的。战火绵延，人民生灵涂炭，组织起一个骑士军团加入战斗刻不容缓！

国王交给了你一个艰巨的任务，**从所有的骑士中选出一个骑士军团，使得军团内没有矛盾的两人（不存在一个骑士与他最痛恨的人一同被选入骑士军团的情况）**，并且，使得这支骑士军团最具有战斗力。为了描述战斗力，我们将**骑士按照1至N编号，给每名骑士一个战斗力的估计，一个军团的战斗力为所有骑士的战斗力总和。**

基础应用：覆盖类问题

树的最小点覆盖、最大独立集、最小支配集

最大独立集

【题目15】[ZJOI2008]骑士)

【输入格式】第一行包含一个正整数N，描述骑士团的人数。

接下来N行，每行两个正整数，按顺序描述每一名骑士的战斗力和他最痛恨的骑士。

【输出格式】应包含一行，包含一个整数，表示你所选出的骑士军团的战斗力。

【输入输出样例】

【输入格式】

```
3
10 2
20 3
30 1
```

【输入格式】

```
30
```

基础应用：覆盖类问题

【题目15】[ZJOI2008]骑士)

$$f[i][0]=\sum\{f[son(i)][0],f[son(i)][1]\}$$

$$f[i][1]=\sum\{f[son(i)][0]\}$$

现在有一条环边，dfs找到环边

从该边的两个端点出发选择

（1）强制不选U,V任意，环的贡献为以U做的f [U][0]

（2）强制不选V，U任意，环的贡献为以V做的f[V][0]

基础应用：覆盖类问题

树的最小点覆盖、最大独立集、最小支配集

最大独立集

【题目15】[ZJOI2008]骑士)

```
//遍历森林
void dfs (int x,int father){
    vis[x]=true;
    for(int i=head[x];i!=-1;i=edg[i].nxt){
        int v=edg[i].to;
        if(v==father)continue;
        if(vis[v]){//在一个子树上找到环边
            U=rt,V=v;
            continue;
        }//遍历完整个子树
        dfs(v,rt);
    }
}
```

```
void tree_dp(int rt){
    dp[rt][1]=val[rt];
    dp[rt][0]=0;
    visdp[rt]=true;
    for(int i=head[rt];i!=-1;i=edg[i].nxt){
        int v=edg[i].to;
        if(visdp[v]) continue;
        tree_dp(v);
        dp[rt][0]+=max(dp[v][0],dp[v][1]);
        dp[rt][1]+=dp[v][0];
    }
}
```

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最小支配集

最小支配集，对于树T(V,E)，最小支配集是指从V中选出尽量少的点组成一个集合，使得V中剩余结点都与取出来的点有边连接。

求解最小支配集算法思想：将树中结点分为两个集合V1（支配集）和V2（非支配集），设f[u][0]代表u在V1中，以u为根的子树被覆盖时支配集的结点个数；f[u][1]代表u不在V1中，u的儿子至少1个在V1中，以u为根的子树被覆盖时支配集的结点个数；设f[u][2]代表u不在V1中，u的儿子子树被覆盖时支配集的结点个数。设v为u的儿子，则有：

```
1  int f[N][3];//f[u][0/1/2]含义如上所述
2  void dfs(int u,int fa){
3      f[u][0]=w[u];
4      f[u][2]=0;
5      int sum=0;
6      for(int i=head[u];i!=0;i=e[i].next){
7          int v=e[i].to;
8          if(v==fa)continue;
9          dfs(v,u);
10         f[u][0]+=min(f[v][0],min(f[v][1],f[v][2]));
11         sum+=min(f[v][0],f[v][1]);
12         f[u][2]+=min(f[v][0],f[v][1]);
13     }
14     for(int i=head[u];i!=0;i=e[i].next){
15         int v=e[i].to;
16         if(v==fa)continue;
17         if(f[v][0]<f[v][1])f[u][1]=min(f[u][1],sum);
18         else f[u][1]=min(f[u][1],sum+f[v][0]-f[v][1]);
19     }
20 }
```

①对于f[u][0]，u在V1中，u覆盖了v，则确保v的子树被覆盖即可，因此**f[u][0]=∑min(f[v][0],f[v][1],f[v][2])+1。**

②对于f[u][1]，u不在V1中，但u至少一个儿子在V1中，假设该儿子为v，u的其余儿子记为vj，v会覆盖u，则确保vj及其子树被覆盖即可。因此**f[u][1]=min(f[v][0]+∑min(f[vj][0],f[vj][1]))**，若u为叶子结点，则f[u][1]=+∞。

③对于f[u][2]，u不在V1中，要确保u的儿子子树被覆盖，则**f[u][2]=∑min(f[v][0],f[v][1])。**

若根结点为rt，则最小支配集结点个数为min(f[rt][0],f[rt][1])。

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最小支配集

【题目16】消防局的设立（题目来源：HNOI2003）

【问题描述】2020 年，人类在火星上建立了一个庞大的基地群，总共有 n 个基地。起初为了节约材料，人类只修建了 n-1条道路来连接这些基地，并且每两个基地都能够通过道路到达，所以所有的基地形成了一个巨大的树状结构。如果基地 A 到基地 B 至少要经过 d 条道路的话，我们称基地A到基地B的距离为 d。

由于火星上非常干燥，经常引发火灾，人类决定在火星上修建若干个消防局。消防局只能修建在基地里，每个消防局有能力扑灭与它距离不超过 2 的基地的火灾。你的任务是计算至少要修建多少个消防局才能够确保火星上所有的基地在发生火灾时，消防队有能力及时扑灭火灾。

【输入格式】输入文件的第一行为 n（1≤n≤1000），表示火星上基地的数目。接下来的 n-1行每行有一个正整数，其中文件第 i 行的正整数为ai，表示从编号为 i 的基地到编号为 ai的基地之间有一条道路，为了更加简洁的描述树状结构的基地群，有 ai<i。

【输出格式】仅有一个正整数，表示至少要设立多少个消防局才有能力及时扑灭任何基地发生的火灾。

样例输入	样例输出
6 1 2 3 4 5	2

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最小支配集

【题目16】消防局的设立（题目来源： HNOI2003 ）

本问题为最小支配集模型，区别在于每个结点可覆盖距离<=2的结点。将树中结点分为两个集合V1（支配集）和V2（非支配集），定义状态如下：

f[u][0]代表u在V1中，以u为根的子树被覆盖时支配集的结点个数；（被自己覆盖）

f[u][1]代表u不在V1中，u的儿子至少1个在V1中，以u为根的子树被覆盖时支配集的结点个数；（被儿子覆盖）

设f[u][2]代表u不在V1中，u的儿子不在V1中，u的孙子至少1个在V1中，以u为根子树被覆盖时支配集的结点个数。（被孙子覆盖）

f[u][3]代表u不在V1中，u的儿子子树被覆盖时支配集的结点个数。（被爷爷覆盖）

f[u][4]代表u不在V1中，u的孙子子树被覆盖时支配集的结点个数。（被父亲覆盖）

状态f[u][3]/f[u][4]是为在u的父亲、爷爷上选点做准备，例如在u的父亲上选点，该点覆盖u和u的儿子，此时确保u的孙子子树被覆盖即可；在u的爷爷上选点，该点覆盖u，此时确保u的儿子子树被覆盖即可。并且，状态f[u][3]/f[u][4]有包含关系。

假设v为u的儿子，则有：

①对于f[u][0]，u在V1中，u覆盖了v及v的儿子，确保v的子树被覆盖即可，因此f[u][0]=Σmin(f[v][0],f[v][1],f[v][2],f[v][3],f[v][4])+1。

②对于f[u][1]，u不在V1中，但u至少一个儿子在V1中，假设该儿子为v，u的其余儿子记为vj，v会覆盖u和vj，则确保vj的子树被覆盖即可。因此f[u][1]=min(f[v][0]+Σmin(f[vj][0],f[vj][1],f[vj][2],f[vj][3]))，若u为叶子结点，则f[u][1]=+∞。

③对于f[u][2]，u不在V1中，u的儿子不在V1中，但u的孙子至少1个在V1中，假设u的某个儿子为v，其余儿子为vj，选择的孙子是v的儿子，要确保vj的子树被覆盖，则f[u][2]=f[v][1]+Σmin(f[vj][1],f[vj][2])。

④对于f[u][3]，确保u的儿子子树被覆盖，f[u][3]+=min(f[v][0],f[v][1],f[v][2])。

⑤对于f[u][4],要确保u的孙子子树被覆盖，则f[u][4]+=min(f[v][0],f[v][1],f[v][2],f[v][3])。

若u为叶子结点，f[u][0]=1，f[u][1]=f[u][2]=+∞，f[u][3]=f[u][4]=0。若根结点为rt，**答案为min(f[rt][0],f[rt][1],f[rt][2])。**

树形动态规划的应用

树的最小点覆盖、最大独立集、最小支配集

最小支配集

【题目16】消防局的设立（题目来源：洛谷P2279）

```
void dfs(int u,int fa){
    f[u][0]=1;
    f[u][1]=f[u][2]=inf;
    f[u][3]=f[u][4]=0;
    int sum1=0,sum2=0;
    for(int i=head[u];i!=0;i=e[i].next){
        int v=e[i].to;
        if(v==fa)continue;
        dfs(v,u);
        f[u][0]+=min(f[v][0],f[v][1],f[v][2],f[v][3],f[v][4]);
        sum1+=min(f[v][0],f[v][1],f[v][2],f[v][3]);
        sum2+=min(f[v][1],f[v][2]);
        f[u][3]+=min(f[v][0],f[v][1],f[v][2]);
        f[u][4]+=min(f[v][0],f[v][1],f[v][2],f[v][3]);
    }
    for(int i=head[u];i!=0;i=e[i].next){
        int v=e[i].to;
        if(v==fa)continue;
        int tmp1=min(f[v][0],f[v][1],f[v][2],f[v][3]);
        if(f[v][0]==tmp1)f[u][1]=min(f[u][1],sum1);
        else f[u][1]=min(f[u][1],sum1-tmp1+f[v][0]);

        int tmp2=min(f[v][1],f[v][2]);
        if(f[v][1]==tmp2)f[u][2]=min(f[u][2],sum2);
        else f[u][2]=min(f[u][2],sum2-tmp2+f[v][1]);
    }
}
```

树形动态规划常见应用技巧

应用技巧1：变无根为有根树。

许多树形DP问题常常是无根树，一般求解的策略是**任选一个结点作为树根**，将无根树转换为有根树处理。

【例题17】联合权值（题目来源：NOIP2014 ）

【问题描述】无向连通图 G 有 n 个点，n-1条边。点从 1 到 n 依次编号,编号为 i 的点的权值为 Wi，每条边的长度均为 1。图上两点(u,v)的距离定义为 u 点到 v 点的最短距离。对于图 G 上的点对(u,v)，若它们的距离为 2，则它们之间会产生Wv*Wu的联合权值。请问图 G 上所有可产生联合权值的有序点对中，联合权值最大的是多少？所有联合权值之和是多少？

【输入格式】第一行包含 1 个整数 n。

接下来 n-1行,每行包含 2 个用空格隔开的正整数u,v，表示编号为 u 和编号为 v 的点之间有边相连。

最后 1 行,包含 n 个正整数，每两个正整数之间用一个空格隔开，其中第 i 个整数表示图 G 上编号为 i 的点的权值为Wi。

【输出格式】

输出共 1 行,包含 2 个整数，之间用一个空格隔开,依次为图 G 上联合权值的最大值和所有联合权值之和。由于所有联合权值之和可能很大，输出它时要对10007取余。

样例输入	样例输出
5 1 2 2 3 3 4 4 5 1 5 2 3 10	20 74

树形动态规划常见应用技巧

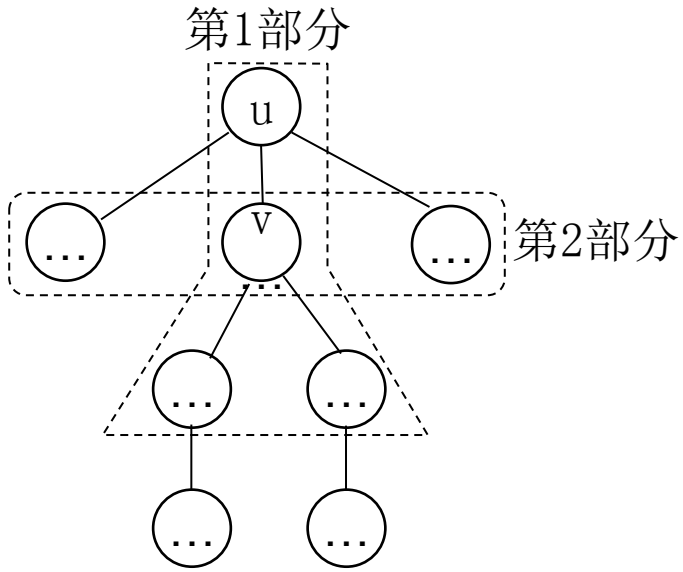
应用技巧1：变无根为有根树。

【例题17】联合权值（题目来源：NOIP2014 ）

任选一个点作为根，将无根树转有根树。对以结点u为根的子树而言，联合权值可以分为如图所示的两个部分，

一是结点u与结点v的孩子之间形成的联合权值（图第1部分）

二是结点u的孩子之间形成的联合权值（图第2部分）。



联合权值之和与联合权值最大值均可分成两个部分处理。

设w[u]代表结点u的权值，f[u]代表结点u的孩子的权值之和，设g[u][0/1]代表结点u的孩子的权值的最大值和次大值。

对于第1部分，联合权值之和2*w[u]*∑f[v]

联合权值最大值为max(w[u] *g[v][0])。

对于第2部分，联合权值之和为∑w[v] * (f[u]-w[v])

联合权值最大值为g[u][0]*g[u][1]。

```
22  void dfs(int u,int fa){
23      for(int i=head[u];i!=0;i=e[i].next){
24          int v=e[i].to;
25          if(v==fa)continue;
26          dfs(v,u);
27          f[u]+=w[v];
28          ans+=w[u]*f[v]*2%mod;
29          if(w[v]>g[u][0])g[u][1]=g[u][0],g[u][0]=w[v];
30          else if(w[v]>g[u][1])g[u][1]=w[v];
31          ans1=max(ans1,w[u]*g[v][0]);
32      }
33      for(int i=head[u];i!=0;i=e[i].next){
34          int v=e[i].to;
35          if(v==fa)continue;
36          ans+=w[v]*(f[u]-w[v])%mod;
37      }
38      ans1=max(ans1,g[u][0]*g[u][1]);
39  }
```

树形动态规划常见应用技巧

应用技巧2：二次扫描与换根法

一类树形DP问题中，状态的转移不仅与子树内结点有关，还与子树外结点有关；或者一类树形DP问题中，需要讨论以每个结点作为树根时的解，此时可采用二次扫描与换根法。具体操作是通过实现一次自底向上的深度优先搜索（DFS）和一次自顶向下的DFS来计算“换根”后的解。

第1次扫描：任选一个结点X为根，在“有根树”上执行DFS，回溯时进行自底向上的状态转移,用子结点的状态更新父结点的状态，即在以结点X为根的树上执行一次“树形DP”。

第2次扫描：从结点X为根出发，对整棵树执行一次DFS，在每次递归前进行自顶向下的转移，用父结点的状态更新子结点的状态，进而计算出“换根”之后的解。

树形动态规划常见应用技巧

应用技巧2：二次扫描与换根法

【例题18】Great Cow Gathering（题目来源： USACO10MAR）

【问题描述】Bessie 正在计划一年一度的奶牛大集会，来自全国各地的奶牛将来参加这一次集会。当然，她会选择最方便的地点来举办这次集会。

每个奶牛居住在 N 个农场中的一个，这些农场由 N-1条道路连接，并且从任意一个农场都能够到达另外一个农场。道路 i 连接农场Ai和Bi，长度为Li。集会可以在 N 个农场中的任意一个举行。另外，每个牛棚中居住着Ci只奶牛。

在选择集会的地点的时候，Bessie 希望最大化方便的程度（也就是最小化不方便程度）。比如选择第 X 个农场作为集会地点，它的不方便程度是其它牛棚中每只奶牛去参加集会所走的路程之和（比如，农场 i 到达农场 X 的距离是 20，那么总路程就是Ci×20）。

请帮助 Bessie 找出最方便的地点来举行大集会。

【输入格式】第一行一个整数 N；

第二到 N+1行：第 i+1 行有一个整数Ci；

第 N+2行到 2N 行：第 i+N+1行为 3 个整数：Ai，Bi和Li。

【输出格式】一行一个整数，表示最小的不方便值。

样例输入	样例输出
5 1 1 0 0 2 1 3 1 2 3 2 3 4 3 4 5 3	15

树形动态规划常见应用技巧

应用技巧2：二次扫描与换根法

【例题18】Great Cow Gathering（题目来源： USACO10MAR）

算法一：枚举每个结点作为集会地点（树根）时所有奶牛到达它的距离总和取最小值。

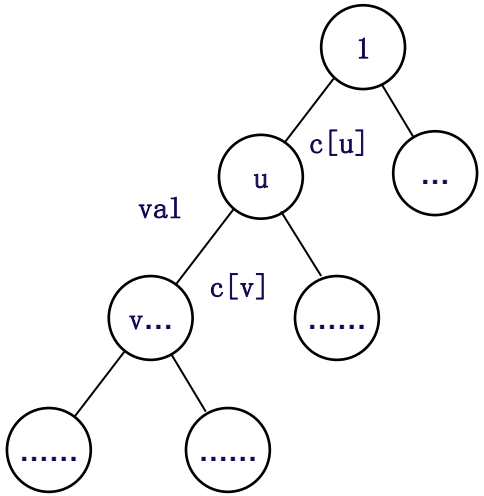
假设结点1为树根，现在需求解所有奶牛到达结点1的距离总和。假设树的结构，c[u]代表结点u的奶牛只数，val代表结点u和v之间的边权。

设f[u]代表以u为根的子树中所有奶牛到达u的距离总和，假设u的儿子为v，那么f[v]记录了以v为根的子树中所有奶牛到达v的距离总和，思考f[u]和f[v]有何关系？

设sum[v]代表以v为根的子树中的奶牛总数，这些奶牛要到达u，都需要再走val距离，所以只需要在f[v]的基础上加上sum[v]*val即可。

综上，**f[v]=∑(f[v]+sum[v]*val)，sum[u]=c[u]+∑sum[v]**。f[1]即是以1为根结点时所有奶牛到达它的距离总和（不方便程度）。

枚举每个结点作为树根，用类似的方法，可以求出以每个结点作为树根时所有奶牛到达该结点的距离总和（不方便程度），时间复杂度O（n^2）

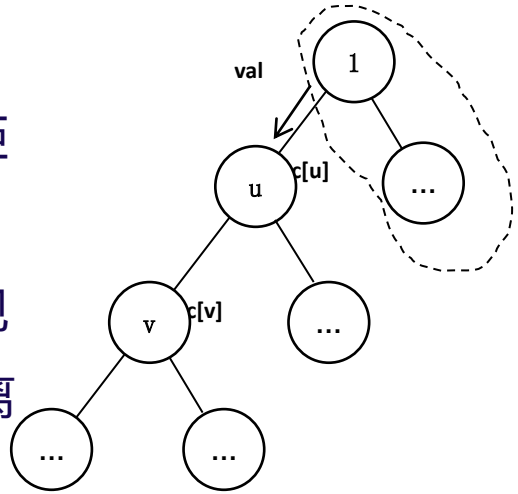


树形动态规划常见应用技巧

应用技巧2：二次扫描与换根法

【例题18】Great Cow Gathering（题目来源： USACO10MAR）

算法二：二次扫描换根法。二次扫描与换根法是指进行两次dfs：第一次dfs时，任选一个结点作为树根，将无根树转为有根树进行处理，维护以该结点作为树根的相关信息；第二次dfs时，动态更换树根，维护以每个结点作为树根的相关信息。



假设选择结点1为树根，对以u为根的子树，在第一次dfs时仍然维护**f[u]、sum[u]**等信息。

第二次dfs的目的是希望在dfs过程中能求出以每个结点作为树根时所有奶牛到达该结点的距离总和。

经过第一次dfs，f[1]、sum[1]中存储的是所有奶牛到达结点1的距离总和与奶牛总只数，现假设从结点1搜索到结点u，当前f[u]中存储的是以u为根的子树中所有奶牛到达结点u的距离总和，sum[u]中存储的是以u为根的子树中的奶牛总只数。

首先，f[1]-f[u]-sum[u]*val代表了虚线框内的所有奶牛到根结点1的距离总和；

其次，虚线框内的所有奶牛只数为sum[1]-sum[v]；

虚线框内的所有奶牛到结点u的距离总和为f[1]-f[u]-sum[u]*val+(sum[1]-sum[u])*val。

树形动态规划常见应用技巧

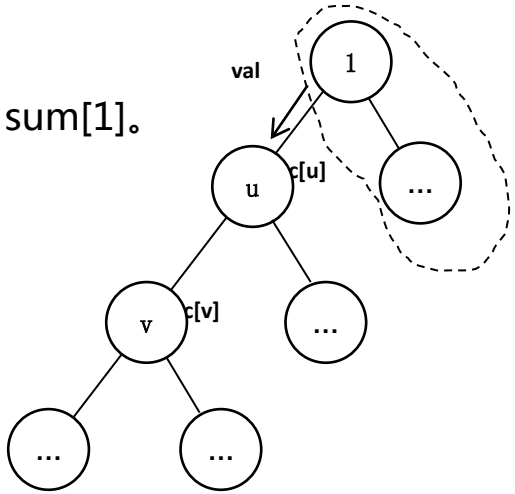
应用技巧2：二次扫描与换根法

【例题18】Great Cow Gathering（题目来源： USACO10MAR）

算法二：二次扫描换根法。二次扫描与换根法是指进行两次dfs：第一次dfs时，任选一个结点作为树根，将无根树转为有根树进行处理，维护以该结点作为树根的相关信息；第二次dfs时，动态更换树根，维护以每个结点作为树根的相关信息。

若以u作为树根，那么所有奶牛到达u的距离总和:

`f[u]=f[u]+f[1]-f[u]-sum[u]*val+(sum[1]-sum[u])*val`，以u作为树根的奶牛总只数sum[u]=sum[1]。



当u搜索到v时，若想求解出以v作为树根时所有奶牛到v的距离总和，由于f[u]，sum[u]已经存储的是u作为树根时的真实信息，那么可以将u看作原来的结点1，v看作原来的结点u，f[v]，sum[v]的求解与f[u]，sum[u]的求解过程是相似的。

这样，可在第二次dfs过程中求出以每个结点作为树根时所有奶牛到该结点的距离总和了，算法时间复杂度为O(n)。

二次扫描与换根法的核心在于将一棵树分为两个部分：子树内和子树外。在第一次dfs时，维护每个结点子树内的信息；在第二次dfs时根据每个结点子树外的信息，求得以该结点作为树根时的真实信息。

树形动态规划常见应用技巧

应用技巧2：二次扫描与换根法

【例题18】Great Cow Gathering（题目来源： USACO10MAR）

//sum[u]代表以u为根的子树中奶牛的总只数

//f[u]代表以u为根的子树中所有奶牛到达u的距离之和

```
3  void dfs(int u,int fa){
4      sum[u]=c[u];
5      for(int i=head[u];i!=0;i=e[i].next){
6          int v=e[i].to,val=e[i].val;
7          if(v==fa)continue;
8              dfs(v,u);
9          sum[u]+=sum[v];
10         f[u]+=f[v]+sum[v]*val;
11     }
12 }
```

```
13 void dfs1(int u,int fa){
14     for(int i=head[u];i!=0;i=e[i].next){
15         int v=e[i].to,val=e[i].val;
16         if(v==fa)continue;
17         f[v]+=(f[u]-f[v]-sum[v]*val)+(sum[u]-sum[v])*val;
18         sum[v]=sum[u];
19         ans=min(ans,f[v]);
20         dfs1(v,u);
21     }
22 }
```