树形动态规划

复习：动态规划

动态规划主要用来解决：以时间划分阶段的动态过程的优化问题，一些与时间无关的静态规划（如线性规划、非线性规划），只要人为的引入时间因数，就可以视为多阶段决策过程，也可使用动态规划方便的求解。

适应动态规划具备的条件：

（1）最优子结构；（2）无后效性；（3）子问题重叠

动态规划算法基本思路：

（1）把问题进行分阶段，定义问题的状态

（2）确定策略，定义问题的状态转移方程

（3）设置边界条件，即最简单子问题的解

（4）计算子问题的解并存储记录

（5）确定问题最终解

线性结构经典问题：

数字三角形问题、最长公共子序列、最大子段和、最长公共子串、矩阵连乘及0-1背包问题等多个经典案例

后继问题都是1:1的，线性结构

如果是1：多？甚至多对多呢？

实现方式：记忆化搜索（DFS）、递推公式（转移方程）

树：连通的无环图。

**1.树型DP基本概念**

顾名思义，就是在“树”的数据结构上做动态规划，通过有限次地遍历树，记录相关信息，以求解问题。通常，动态规划都是线性的或者建立在图上的，分为逆推和顺推。

①叶->根，即根的子节点传递有用的消息给根，之后由根得出最优解的过程。这种方式DP的题目应用比较多。

②根->叶，即需要取所有点作为一次根节点进行求值，此时父节点得到了整棵树的信息，只需要去除这个儿子的DP值的影响，然后再转移给这个儿子，这样就能达到根->叶的顺序

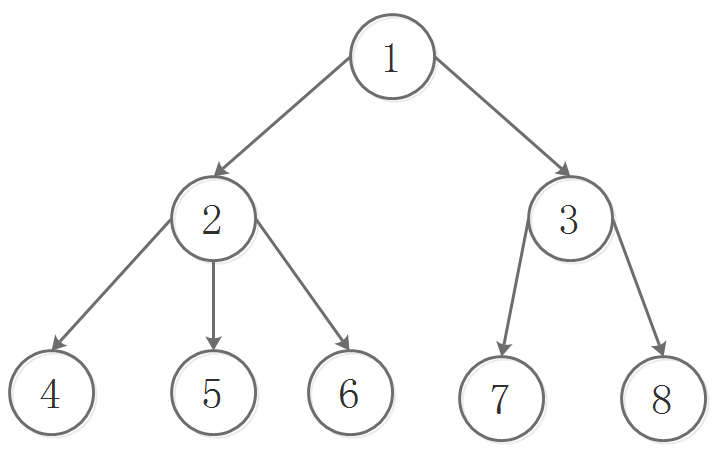
动态规划的顺序：一般按照后序遍历的顺序，几处理完儿子再处理当前结点，才符合树的子结构的性质。

实现方式：树型DP是通过记忆化搜索实现的，因此**采用的是递归方式**

时间复杂度：树型动态规划的时间复杂度基本上是O（n)；若有附加维m，则是O（n \* m)

树形动态规划中，一般先计算子树在进行合并，实现上与后续遍历类似，即先遍历子树，遍历完子树后将子树的值传递给父亲。

例：给定一棵有N个结点的树，根结点为结点1.对于i=1,2,…,N，求以结点i为根的子树大小（即子树上的结点个数，包括根结点）



分析：如果要求以A为根的子树大学，得先求出其子结点B，C的子树大小

因此设状态f[v]代表以v为根的子树大小，即

一个结点子树大小=1（根结点）+每个子树的大小

临界值：叶节点的子树大小为1；

问题解：f[root]

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int const maxN = 324;

int n; //m需要加1，增加了空节点

vector<int> edge[maxN];

int dp[maxN];

void dfs(int v){

dp[v]=1;

for(int u:edge[v]){

dfs(u);

dp[v]+=dp[u];

}

}

int main() {

cin >> n ; //n个顶点

int u, v;

for (int i = 1; i < n; i++) { //n-1条边

cin >> u >> v; //u为树根，v为孩子

edge[u].push\_back(v);

}

dfs(1);//假设1为

cout<<dp[1];

return 0;

}

输入样例：

8

1 2

2 4

2 5

2 6

1 3

3 7

3 8

输出样例

8

**2.模板题**

树的最大独立集问题

例： P1352 没有上司的舞会

【题目描述】

某大学有n个职员，编号为1…n。

他们之间有从属关系，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。

现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数ri，但是呢，如果某个职员的直接上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。

所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。

输入格式

输入的第一行是一个整数n。

第 2 到第(n+1) 行，每行一个整数，第(i+1) 行的整数表示i号职员的快乐指数ri。

第(n+2) 到第 2n 行，每行输入一对整数l,k，代表k 是 l 的直接上司。

输出格式

输出一行一个整数代表最大的快乐指数。

输入输出样例

输入 #1

7

1

1

1

1

1

1

1

1 3

2 3

6 4

7 4

4 5

3 5

输出 #1

5

说明/提示

数据规模与约定

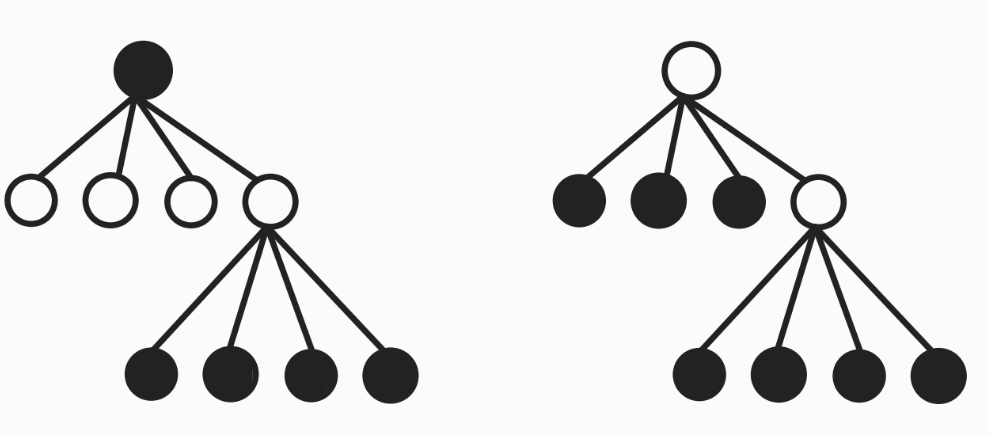
对于100% 的数据，保证 1≤n≤6×103，−128≤ri≤127，1≤l,k≤n，且给出的关系一定是一棵树。

思路分析：

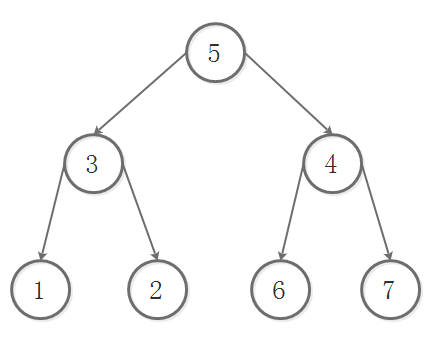
对题意抽象一下：没有职员会和上司一同参会，也就是说在这棵树上不存在任何一条边使得连接的两个点都来参会，换句话说这道题其实要我们求的是 树的最大权值的独立集

**黑白染色**，也就是在树上一层选一层不选这样的贪心。

容易证明简单的黑白染色是不对的，如下图所示，左边是黑白染色的结果，右边是正解的结果。其中黑点表示要来参加的人，白点表示不来参加的人，在这个例子中我们默认每个人的快乐指数都是一样的。



样例1将上司与职员直接关系描述为一棵树，如下图



（1）分阶段，对于每一层看作一个阶段

（2）状态转移

对于第i个职员参不参加与其父亲参不参加有关，同样的如果i是父亲，那么其参不参加也影响其孩子能不能参加。

用f[i][0]表示不选i点去参加舞会时i点及其子树所能选的最大的快乐指数；

f[i][1]表示选点去参加舞会时i点及其子树所能选的最大的快乐指数

如果i参加了，那么它的儿子结点就一定不能参加，则有

如果i未参加，那么它的儿子结点可以参加，也可以不参加，即

其中soni代表i结点的所有儿子结点集合。

（3）无后效性

对于第i个员工的团队，只考虑本团队产生的最大快乐值，其父亲产生最大值时，不会改变之前的情况。

临界值：每个叶节点员工最大快乐值参加活动就是其自身快乐值，不参加就是0

结果是为根结点的所有结点产生的最大快乐值 ans=max(f[root][0],f[root][1])

【参考答案】

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

const int N = 1e4 + 5;

vector<int> son[N];

int hap[N],f[N][2];

bool flag[N];

int n;

void dfs(int root){

f[root][1]=hap[root];

for(auto s:son[root]){

dfs(s);

f[root][0]+=max(f[s][0],f[s][1]);

f[root][1]+=f[s][0];

}

}

int main() {

int t, u, root;

cin >> n;

for (int i = 1; i <= n; i++)

cin >> hap[i];

for (int i = 1; i < n; i++) {

cin >> t >> u;

son[u].push\_back(t);

flag[t] = 1;

}

for (int i = 1; i <= n; i++) {

if (flag[i] == 0) {

root = i;

break;

}

}

dfs(root);

cout<<max(f[root][0],f[root][1]);

return 0;

}

【参考答案2】

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

const int N = 1e4 + 5;

struct Edge{

int v,next;

}e[N];

int head[N],n,cnt;

int hap[N],f[N][2];

int is\_h[N],vis[N];

void addedge(int u,int v){

e[++cnt].v=v;

e[cnt].next=head[u];

head[u]=cnt;

}

void dfs(int k){

vis[k]=1;

for(int i=head[k];i;i=e[i].next){

if(vis[e[i].v]) continue;

dfs(e[i].v);

f[k][1]+=f[e[i].v][0];

f[k][0]+=max(f[e[i].v][0],f[e[i].v][1]);

}

}

int main() {

int t, u, root;

cin >> n;

for (int i = 1; i <= n; i++)

cin >> f[i][1];

for (int i = 1; i < n; i++) {

cin >> t >> u;

is\_h[t] = 1;

addedge(u,t);

}

for (int i = 1; i <= n; i++) {

if (is\_h[i] == 0) {

root = i;

break;

}

}

dfs(root);

cout<<max(f[root][0],f[root][1]);

return 0;

}

树形DP板子：

**void dfs(int v)**

**{**

**for (int i=head[v];i;i=edge[i].next)**

**{**

**int u=edge[i].u; dfs(u);**

**//~~枚举子状态，选取最值的过程~~**

**}**

**}**

例 ：树形背包问题

P2014 [CTSC1997] 选课

题目描述

在大学里每个学生，为了达到一定的学分，必须从很多课程里选择一些课程来学习，在课程里有些课程必须在某些课程之前学习，如高等数学总是在其它课程之前学习。现在有N 门功课，每门课有个学分，每门课有一门或没有直接先修课（若课程 a 是课程 b 的先修课即只有学完了课程 a，才能学习课程 b）。一个学生要从这些课程里选择M 门课程学习，问他能获得的最大学分是多少？

输入格式

第一行有两个整数N , M 用空格隔开。(1≤N≤300 , 1≤M≤300 )

接下来的N 行,第i+1 行包含两个整数ki和 si，ki1表示第i门课的直接先修课， si表示第i门课的学分。若ki =0 表示没有直接先修课（1≤ki≤N , 1≤si≤20）。

输出格式

只有一行，**选 M 门课程的**最大得分。

输入输出样例

输入 #1

7 4

2 2

0 1

0 4

2 1

7 1

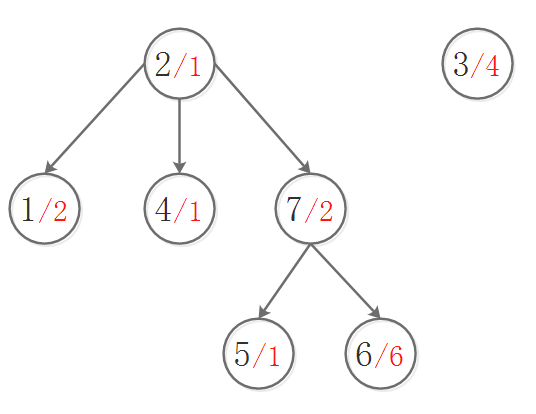
7 6

2 2

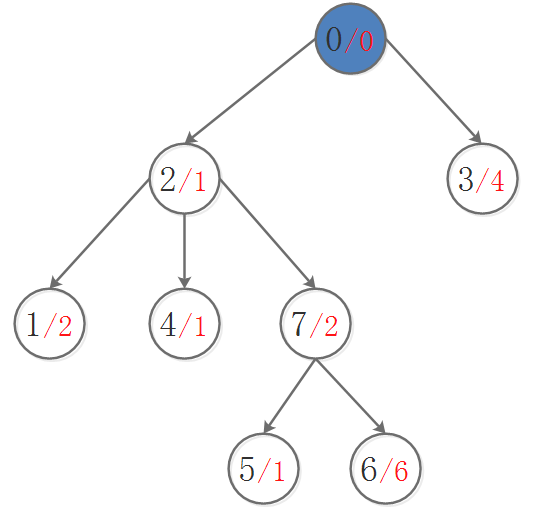
输出 #1

13

分析：每门课最多有一门先修课，也可能没有先修课，很明确所有课程关系会形成一个森林。



为了方便，可以对所有的树构造一个总树根，其编号为0，学分为0，将森林问题转化为树上问题



要求的是在N门课中选M门的最大学分，即以0为树根的树中选M门课的最大学分

状态定义为以i为树根的所有子树中选j门课的最大学分 dp[i][j]

转移方程的实现：如何求出根结点u=0取j=1,2,3,…,m+1个结点的最大值 ？

先求出 孩子结点v取0个，1个，j-1个产生的最大值 对于其孩子结点

设f(u,i,j) 表示以u为根的子树中，已经遍历了u的前i棵子树，选了j门课程的最大学分

记点x的儿子个数为sx，以x为根的子树大小为size\_x

枚举u点的每个子结点v，同时枚举以v为根结点的子树选了几门课程，将子树的结果合并大u上。

则有动态转移方程

**f 的第二维可以很轻松地用滚动数组的方式省略掉，注意这时需要倒序枚举 j 的值。**

临界条件：任何结点为树根选1门课的情况下最大值就是其学分值即dp[i][1]=其本身学分值si（直接输入）

增加了一个虚拟根结点，那么对于从n个结点选m个最大值问题，转化为从n+1中选m+1个最大值问题

输出结果为：dp[0][m+1]，即以0为根的整棵树中选择m+1门课（树根必选）的最大学分值。

#include <cstdio>

#include <cstring>

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

int const maxN = 324;

int n, m; //m需要加1，增加了空节点

int f[maxN][maxN];

struct Node {

int to, next;

Node(int to = 0, int next = 0): to(to), next(next) {

}

} edge[maxN];

int head[maxN], cnt;

void addedge(int u, int v) {

// edge[++cnt].to=v;

// edge[cnt].next=head[u];

edge[++cnt] = (Node) {

v, head[u]

};

head[u] = cnt;

}

void dfs(int u) {

//每个点都是一个组，代表以u为根的子树的最优解

for (int i = head[u]; i; i = edge[i].next) {

int v = edge[i].to;

dfs(v);//先对其每棵子树进行搜索统计找出最大值

for (int j = m + 1; j >= 1; j--) {//01背包问题从大到小计算，优化存储数组，枚举体积V

for (int k = 0; k < j; k++) { //枚举子树中选择的结点个数，大树根中已经至少包含一个1个结点不能k<=j;

f[u][j] = max(f[u][j], f[u][j - k] + f[v][k]); //以u为树根的树中选j个结点构成的最大值，由其在v子树中选j个结点，在

}

}

}

}

int main() {

cin >> n >> m;

memset(f, 0xff, sizeof(f));

for (int i = 1; i <= m + 1; i++)

f[0][i] = 0;

int u;

for (int i = 1; i <= n; i++) {

cin >> u >> f[i][1];

addedge(u, i);

}

dfs(0);

cout << f[0][m + 1];

return 0;

}

代码2：

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

int f[305][305], s[305], n, m;

vector<int> e[305];

int dfs(int u) {

int p = 1;

f[u][1] = s[u];

for (auto v : e[u]) {

int siz = dfs(v);

// 注意下面两重循环的上界和下界

// 只考虑已经合并过的子树，以及选的课程数超过 m+1 的状态没有意义

for (int i = min(p, m + 1); i; i--)

for (int j = 1; j <= siz && i + j <= m + 1; j++)

f[u][i + j] = max(f[u][i + j], f[u][i] + f[v][j]); // 转移方程

p += siz;

}

return p;

}

int main() {

cin.tie(nullptr)->sync\_with\_stdio(false);

cin >> n >> m;

for (int i = 1; i <= n; i++) {

int k;

cin >> k >> s[i];

e[k].push\_back(i);

}

dfs(0);

cout << f[0][m + 1];

return 0;

}

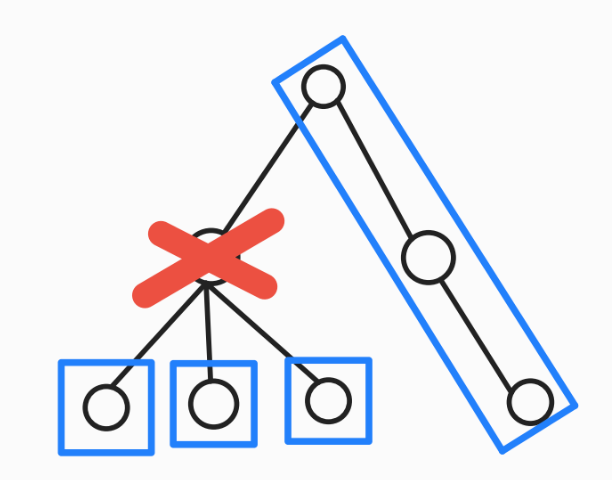
其他经典问题

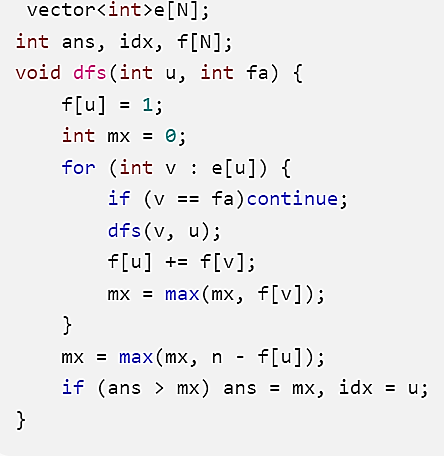
**1.树的重心问题（树的平衡点问题）**

对于一颗n个节点的无根树，找到一个点，使得把树变成以该点为根的有根树时，最大子树的节点最小，换句话说，**删除这个点后最大连通块的节点数最小**，那么这个点就是树的重心。

现在的问题还很简单，一般就是题目问什么我们就设什么，所以我们先设f[i]为删除i号点后最大连通块的块大小。然后我们沿用上一题的设f[i]为以i为根的子树大小，那么很简单就有**f[i]=max(n-f[i],max(f[k]) ，其中k**是i的子节点。这是因为，删除i号点后，我们剩下的连通块要么是i的子树，要么是i的父亲所在的连通块。

**通俗的来讲，即删除某个节点后，它的儿子就成了独立的连通块，那么最大连通块就是 max(x 的所有儿子连通块最大的 size，n - f[x])**





**2.树的****最小支配集、最小点覆盖和最大独立集**

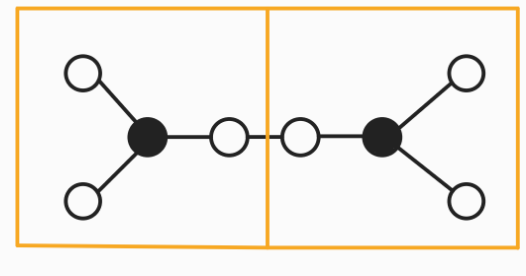
定义1：对于图G=(V,E)来说，最小支配集指从V中取尽量少的点组成一个集合，使得对于V中剩余的点与取出来点有边相连。 也就是说，设V’是图G的一个支配集，则对于图中的任意一个顶点u，要么属于集合V’，要么与V’中的顶点相邻。在V’中出去任何元素后V’不再是支配集，则支配集是极小支配集。称G的所有支配集中顶点个数最少的支配集为最小支配集，最小支配集中顶点的个数称为支配数。

定义2: 对于图G=(V,E)来说，最小点覆盖指的是从V中取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。也就是说，设V’是图G的一个顶点覆盖，则对于图中的任意一条边(u,v)，要么u属于集合V’，要么v属于集合V’。在V’中除去任何元素后V’不在是顶点覆盖，则V’是极小顶点覆盖。称G的所有顶点覆盖中顶点个数最少的覆盖为最小点覆盖。

定义3: 对于图G=(V,E)来说，最大独立集指的是从V中取尽量多的点组成一个集合，使得这些点之间没有边相连。也就是说，设V’是图G的一个独立集，则对于图中任意一条边(u,v)，u和v不能同时属于集合V'，甚至可以u和v都不属于集合V’。在V’中添加任何不属于V’元素后V’不再是独立集，则V’是极大独立集。称G的所有顶点独立集中顶点个数最多的独立集为最大独立集。

树的最小支配集

给你一个有n 个点的树，每两个点之间至多只有一条边。如果在第i个点部署信号塔，就可以让它和所有与它相连的点都收到信号。求最少部署多少个信号塔能让所有点都能收到信号。



树的最小点覆盖

一城堡的所有的道路形成一个n个节点的树，如果在一个节点上放上一个士兵，那么和这个节点相连的边就会被看守住，问把所有边看守住最少需要放多少士兵

**3.树的最长路径（最远点对、树的直径）**

给定一颗n个结点的边带权树找到一条最长的路径，换句话说，要找到两个点，使得他们的距离最远，他们之间的路径就是树的最长路径

解法：用d1[i],d2[i]记录以i为根的子树中到叶节点的最大值和次大值，j是i的儿子

①d1[j] + dis[i][j] > d1[i] 则d2[i] = d1[i],d1[i] = d1[j] + dis[i][j]

②d1[j] + dis[i][j] > d2[i] 则d2[i] = d1[j] + dis[i][j]

4.树的中心问题

给出一颗带权的树，求树中的点，使得此点到树中的其他节点的最远距离最近

分析：从任意一个点i出发的最长路径的可能形态有两种。

1. i点出发向上，即终点不在以i为根的子树中的最长路径长度为u[i]

②从i点出发向下，即终点在以i为根的子树中的最长路径长度为d1[i]

注意：第一种里面不要重复经过i点

分别用c1[i]和c2[i]记录d1[i]和d2[i]是从哪个子树更新来的。

（1）

①d1[j] + dis[i][j] > d1[i] 则d2[i] = d1[i],d1[i] = d1[j] + dis[i][j]

②d1[j] + dis[i][j] > d2[i] 则d2[i] = d1[j] + dis[i][j]

（2）设prt[i]=x （i的父节点）

如果c1[x] ！= i，u[i] = max(d1[x],u[x]) + dist[x][i]

否则u[i] = max(d2[x],u[x]）+ dist[x][i]

（3）

在最后n个节点中找到最大值t[i] = max(u[i],d1[i])

（4）树的中心 ans= min(t[i])

树形DP还有一个重要拓展是与各类树形数据结构集合，例如Trie上的DP，AC自动机的DP，后缀自动机的DP等。

参考文献

<https://www.cnblogs.com/RioTian/p/15163878.html>

<https://blog.csdn.net/qq_41383801/article/details/81637289>

练习：

**Farthest Nodes in a Tree：**<https://codeforces.com/problemset/problem/1153/D>

D. Bear and Tree Jumps：

<https://codeforces.com/contest/791/problem/>D

D. Serval and Rooted Tree：

<https://codeforces.com/problemset/problem/1153/D>

洛谷P4084 P2986