树型动态规划

主讲老师：党东

【树型动态规划】

树型动态规划，顾名思议，在树型数据结构上的动态规划，它的状态、阶段、边界、初始值等，都与树有关。

前几讲学习的动态规划都是建立在线性上、坐标上或图结构上，本讲建立在层次分明，有上下关系的树上，它也有顺推和倒推两种方向，对应于树上，则为从根结点到叶结点和从叶结点到根结点。

【树型动态规划】

树形DP的特殊性： 没有环，dfs是不会重复，而且具有明显而又严格的层数关系。 利用这一特性，我们可以很清晰地根据题目写出一个在树（型结构）上的记忆化搜索的程序。

【树型动态规划】

树的特点与性质：

1、 有n个点，n-1条边的无向图，任意两顶点间可达

2、 无向图中任意两个点间有且只有一条路

3、 一个点至多有一个前趋，但可以有多个后继

4、 无向图中没有环；

【树型动态规划】

对于一道树规题，解题步骤如下：

1.判断是否是一道树规题：即判断数据结构是否是一棵树，然后是否符合动态规划的要求。如果是，那么执行以下步骤，如果不是，那么思考其他方法。

2.建树：通过数据量和题目要求，选择合适的树的存储方式。

如果节点数小于5000，那么我们可以用邻接矩阵存储，如果更大可以用邻接表来存储(注意边要开到2*n，因为是双向的。这是血与泪的教训)。如果是二叉树或者是需要多叉转二叉，那么我们可以用两个一维数组brother[]，child[]来存储）。

3.写出树规方程：通过观察孩子和父亲之间的关系建立方程。我们通常认为，树形DP的写法有两种：

a.根到叶子: 不过这种动态规划在实际的问题中运用的不多。本文只有最后一题提到。

b.叶子到根: 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。

【例4】树的最长链 --1532

Description

给你一棵有n个结点的树，结点编号为0到n-1，现要你求出该树最长链的长度。

Input

第一行一个整数n，表示结点的个数。

接下来n-1行每行包括两个整数u, v表示编号为u和v的结点之间存在着一条边。

Output

输出文件仅一个整数为该树的最长链长度。

Sample Input

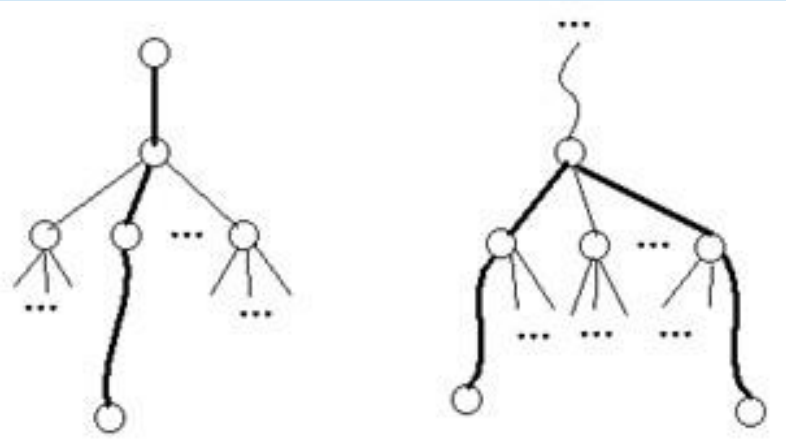
```
10
0 1
0 2
0 4
0 6
0 7
1 3
2 5
4 8
6 9
```

Sample Output

```
4
```

【例4】树的最长链 --1532

【思路点拨】一棵有根树的最长链，可能出现如下图的两种情况：



对于每个结点我们都要记录两个值：

- d1[i]表示以i为根的子树中，i到叶子节点的距离最大值；
- d2[i]表示以i为根的子树中，i到叶子节点的距离次大值；令j是i的儿子。则：
- ①若d1[j]+1>d1[i]，则d2[i]=d1[i]; d1[i]=d1[j]+1;
- ②否则，若d1[j]+1>d2[i]，则d2[i]=d1[j]+1;
- 最后扫描所有的结点，找最大的d1[i]+d2[i]的值。

【例4】树的最长链 --1532

【参考代码】

```
const int MAXN=200005;
int n,h[MAXN]; //表示以i为起点的第一条边的存储位置
int cnt,x,y,ans,d1[MAXN],d2[MAXN],pre[MAXN];
struct Edge {
        int to;    //第i条边的终点
        int next;  //与第i条边同起点的下一条边的存储位置
}a[MAXN*2];
void AddEdge(int x,int y)
{ cnt++;a[cnt].to=y;a[cnt].next=h[x];h[x]=cnt; }
```

【例4】树的最长链 --1532

【参考代码】

```
void Treedp(int x)
{
	for(int i=h[x];i;i=a[i].next)
	{
		if(a[i].to==pre[x]) continue;
		pre[a[i].to]=x;
		Treedp(a[i].to);
		if(d1[a[i].to]+1>d1[x])
		{
			d2[x]=d1[x];
			d1[x]=d1[a[i].to]+1;
		}
		else if(d1[a[i].to]+1>d2[x]) d2[x]=d1[a[i].to]+1;
	}
}
```

【例4】树的最长链 --1532

【参考代码】

```
int main()
{
	scanf("%d",&n);
	for(int i=1;i<=n-1;i++)
	{
		scanf("%d%d",&x,&y); x++;y++;
		AddEdge(x,y);AddEdge(y,x);
	}
	Treedp(1);
	for(int i=1;i<=n;i++) ans=max(ans,d1[i]+d2[i]);
	printf("%d\n",ans);
	return 0;
}
```

【例5】战略游戏 --1533

Description

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了望到。

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。

Input

输入文件中数据表示一棵树，描述如下：

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r1，r2，...，rk。

对于一个n(0 < n <= 1500)个结点的树，结点标号在0到n-1之间，在输入文件中每条边只出现一次。

Output

输出文件仅包含一个数，为所求的最少的士兵数目。

Sample Input

```
4
0 1 1
1 2 2 3
2 0
3 0
```

Sample Output

```
1
```

【例5】战略游戏 --1533

【思路点拨】

按照要求构建一张关系图，可见这是一棵树。对于这类最值问题，向来是用动态规划求解的。

任何一个点的取舍可以看作一种决策，设f[i][1]表示i点放士兵时，以i为根的子树需要的最少士兵数目；f[i][0]表示i点不放士兵时，以i为根的子树需要的最少士兵数目。

当点i不放时，则它的所有儿子都必须放，即f[i][0]+=f[j][1]；其中j为i的儿子。

当点i放时，则它的所有儿子放与不放无所谓，但应该取两种情况的最大值。即f[i][1]+=max(f[j][1],f[j][0])；其中j为i的儿子。

初始条件：f[i][0]=0;f[i][1]=1

【例5】战略游戏 --1533

【参考代码】

```
struct {int num,child[1505];} node[1505];
int f[1505][2],a[1505],n,root;
void read()
{  int i,j,x,y;
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {  scanf("%d",&x);
      scanf("%d",&node[x].num);
      for(j=1;j<=node[x].num;j++)
         {scanf("%d",&y);node[x].child[j]=y;a[y]=1;}
   }
   root=0;
   while(a[root]) root++;//找根结点编号
}
```

【例5】战略游戏 --1533

【参考代码】

```
void dp(int x)  //计算以x为根的子树的值
{  int i,j;
   f[x][1]=1;  //x上设士兵的初值
   f[x][0]=0;  //x上不设士兵的初值
   if(node[x].num==0) return; //处理叶子节点情况
   for(i=1;i<=node[x].num;i++) //处理x的每个儿子
   {  dp(node[x].child[i]);   //计算第i个儿子的两个值
      f[x][0]+=f[node[x].child[i]][1];
      //x上不设士兵时儿子必设，将其值累加给自己
      f[x][1]+=min(f[node[x].child[i]][0],f[node[x].child[i]][1]);
      //x上设士兵时儿子可设可不设，选最小的累加给自己
   }
}
int main()
{  read();
   dp(root);
   printf("%d",min(f[root][0],f[root][1]));
}
```

【例5】战略游戏 --1533

【方法2】贪心：找出所有度为1的结点，把与它们相连的结点上都放上士兵，然后把这些度为1的结点及已放上士兵的结点都去掉。重复上述过程直至树空为止。

【例6】皇宫看守 --1534

Description

太平王世子事件后，陆小凤成了皇上特聘的御前一品侍卫。皇宫以午门为起点，直到后宫嫔妃们的寝宫，呈一棵树的形状；某些宫殿间可以互相望见。大内保卫森严，三步一岗，五步一哨，每个宫殿都要有人全天候看守，在不同的宫殿安排看守所需的费用不同。

可是陆小凤手上的经费不足，无论如何也没法在每个宫殿都安置留守侍卫。

编程任务：帮助陆小凤布置侍卫，在看守全部宫殿的前提下，使得花费的经费最少。

Input

输入文件中数据表示一棵树，描述如下：

第1行 n，表示树中结点的数目。

第2行至第n+1行，每行描述每个宫殿结点信息，依次为：该宫殿结点标号i(0 < i<=n) 在该宫殿安置侍卫所需的经费k，该边的儿子数m，接下来m个数，分别是这个节点的m个儿子的标号r1，r2，...，rm。

对于一个n（ 0 < n <= 1500 ）个结点的树，结点标号在1到n之间，且标号不重复。

Output

输出文件仅包含一个数，为所求的最少的经费。

【例6】皇宫看守 --1534

Sample Input

```
6
1 30 3 2 3 4
2 16 2 5 6
3 5 0
4 4 0
5 11 0
6 5 0
```

Sample Output

```
25
```

【例6】皇宫看守 --1534

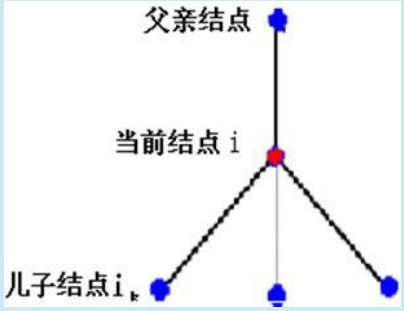
【试题分析】

本题已知模型是一棵树，因此我们试着用树形动态规划来解决。对于本题，每个安全结点i，都有3种状态分别为：

①要么在父亲结点安排警卫，即被父亲看到

②要么在儿子结点安排警卫，即被儿子看到

③要么安排警卫



设:

f(i,0)表示i结点被父亲看时，以i为根的子树需要安排的最少士兵；

f(i,1)表示i被它的儿子看时，以i为根的子树需要安排的最少士兵；

f(i,2)表示在i安排警卫时，以i为根的子树需要安排的最少士兵；

【例6】皇宫看守 --1534

【试题分析】

现在只需针对这三种状态，设计出状态转移方程。

①对于f(i,0)，表示i被父亲看到，这时i没有安排警卫，i的儿子要么安排警卫，要么被它的后代看到，所以有：

$$f[i,0]=\sum_{k=1}^{i\text{的儿子数}}\min\{f[i.k,1],f[i.k,2]\}$$

②对于f(i,1)，表示i被儿子看到，即i的某个儿子安排了警卫，其他儿子需要安排警卫或者被它的后代看到，所以有：

$$f[i,1]=\sum_{k=1}^{i\text{的儿子数}}\min\{f[i.k,1],f[i.k,2]\}+\mathrm{d}$$

其中 $$d=\min\{f[i.k,2]-\min\{f[i.k,1],f[i.k,2]\}\}$$

③对于f(i,2)，表示i安排了警卫，i的儿子可以安排警卫，也可以被i的儿子的儿子看守，还可以被父亲看守，所以有：

$$f[i,2]=\sum_{k=1}^{i\text{的儿子数}}\min\{f[i.k,0],f[i.k,1],f[i.k,2]\}+\mathrm{a[i]}$$

【例6】皇宫看守 --1534

【试题分析】

```
int cost[1505]={0},f[1505][3]={0},n;//f: 0self 1son 2father
bool map[1505][1505]={0},vis[1505]={0};
void dp(int t)
{  int i,d;
   if(vis[t])return;
   vis[t]=true;
   d=0x7fffffff;
   for(i=1;i<=n;i++)
      if(map[t][i]&&!vis[i])  //找t的儿子i，且i未被遍历过
      {  dp(i);
         f[t][0]+=min(f[i][2],f[i][1]);
         f[t][1]+=min(f[i][2],f[i][1]);  //t的儿子i们自己解决警卫问题：i自守或者被i的儿子看
         d=min(d,f[i][2]-min(f[i][2],f[i][1]));//表示在i的所有儿子i.ch警卫方案中，加最少d的费用后，可以让某个儿子i.ch帮忙警卫i
         f[t][2]+=min(f[i][2],min(f[i][1],f[i][0]));  //处理所有儿子的情况累加，因为t可以帮儿子i警卫，所以儿子可以多种情况。
      }
   f[t][1]+=d;   //计算所有儿子i的情况后，才能得到f(t,1)的情况
   f[t][2]+=cost[t];  //单独累加结点t的费用
}
```

【例6】皇宫看守 --1534

【试题分析】

```
int main()
{  int i,j,k,m,t;
   cin>>n;
   for(k=1;k<=n;k++)
   {  scanf("%d",&i);scanf("%d%d",&cost[i],&m);
      for(j=1;j<=m;j++){scanf("%d",&t);map[i][t]=map[t][i]=true;}
   }
   dp(1);
   cout<<min(f[1][1],f[1][2]);
}
```

【例7】消息传递 --1535

Description

巴蜀国的社会等级森严，除了国王之外，每个人均有且只有一个直接上级，当然国王没有上级。如果A是B的上级，B是C的上级，那么A就是C的上级。绝对不会出现这样的关系：A是B的上级，B也是A的上级。

最开始的时刻是0，你要做的就是用1单位的时间把一个消息告诉某一个人，让他们自行散布消息。在任意一个时间单位中，任何一个已经接到消息的人，都可以把消息告诉他的一个直接上级或者直接下属。

现在，你想知道：

1.到底需要多长时间，消息才能传遍整个巴蜀国的所有人？

2.要使消息在传递过程中消耗的时间最短，可供选择的人有那些？

Input

输入文件的第一行为一个整数N（N≤3000），表示巴蜀国人的总数，假如人按照1到n编上了号码，国王的编号是1。第2行到第N行（共N-1行），每一行一个整数，第i行的整数表示编号为i的人直接上级的编号。

Output

文件输出共计两行：

第一行为一个整数，表示最后一个人接到消息的最早时间。

第二行有若干个数，表示可供选择人的编号，按照编号从小到大的顺序输出，中间用空格分开。

【例7】消息传递 --1535

Sample Input

```
8
1
1
3
4
4
4
3
```

Sample Output

```
5
3 4 5 6 7
```