[**树型动态规划**](http://www.cnblogs.com/liyongmou/archive/2010/07/30/1788509.html)

　　树型动态规划就是在“树”的数据结构上的动态规划，平时作的动态规划都是线性的或者是建立在图上的，线性的动态规划有二种方向既向前和向后，相应的线性的动态规划有二种方法既顺推与逆推，而树型动态规划是建立在树上的，所以也相应的有二个方向：

　　1． 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。

　　2． 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。

　　树本身就是一个递归的结构，所以在树上进行动态规划或者递推是在合适不过的事情。

　　必要条件：子树之间不可以相互干扰，如果本来是相互干扰的，那么我们必须添加变量使得他们不相互干扰。

题目：

1. POJ 1463 Strategic game

　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。

　dproot[ i ]表示以i为根的子树，在i上放置一个士兵，看守住整个子树需要多少士兵。

   all[ i ]表示看守住整个以i为根的子树需要多少士兵。

　状态转移方程：

 　　　　叶子节点：dproot[k] =1； all[k] = 0；

　　　　 非叶子节点：      dproot[i] = 1 + ∑all[j](j是i的儿子)；

　　　　　　　　　　　　all[i] = min( dproot[i], ∑dproot[j](j是i的儿子) )，决策是否在i点放士兵

　代码：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

//在poj上这个题用list超时了，用vector就360ms  
#include <cstdio>  
#include <vector>  
using namespace std;  
  
#define MAX 1500  
  
int n;  
int root;  
vector<int> tree[MAX];  
int dproot[MAX]; //dproot[i]表示以i为根的子树，在i上放置一个士兵，看守住整个子树需要多少士兵  
int all[MAX]; //all[i]表示看守住以i为根的子树需要多少士兵  
  
void DFS(int r){  
 vector<int>::iterator it;  
 if(!tree[r].empty()){  
 int sum\_root, sum\_all;  
 sum\_root = sum\_all = 0;  
 for(it=tree[r].begin(); it != tree[r].end(); ++it){  
 DFS(\*it);  
 sum\_root += dproot[\*it];  
 sum\_all += all[\*it];  
 }  
 //状态转移方程  
 dproot[r] = 1 + sum\_all;//r放了一个士兵，其儿子可放可不放  
 all[r] = min(dproot[r], sum\_root); //决策：r放士兵、不放士兵  
 }  
 else{  
 dproot[r] = 1;  
 all[r] = 0;  
 }  
}  
  
bool read\_data(){  
 if(scanf("%d", &n) == EOF)  
 return false;  
 int i, j;  
 for(i=0; i<n; ++i)  
 if(!tree[i].empty())  
 tree[i].clear();  
 int m, u, v;  
 scanf("%d:(%d)", &u, &m);  
 root = u;  
 for(i=0; i<m; ++i){  
 scanf("%d", &v);  
 tree[u].push\_back(v);  
 }  
 for(i=1; i<n; ++i){  
 scanf("%d:(%d)", &u, &m);  
 for(j=0; j<m; ++j){  
 scanf("%d", &v);  
 tree[u].push\_back(v);  
 }  
 }  
 return true;  
}  
int main() {  
// freopen("in", "r", stdin);  
 while(read\_data()){  
 DFS(root);  
 printf("%d\n", all[root]);  
 }  
 return 0;  
}