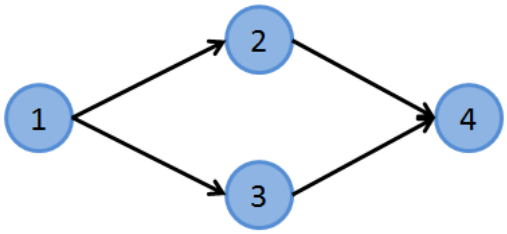
拓扑排序

对于一个有向无回路图（DAG），如：



进行拓扑排序的结果为该图所有节点的一个线性序列，满足如果DAG包含有向边（u,v），则在这个序列中u出现在v之前。如在序列（1，2，3，4）中：
节点1在节点2之前，满足边（1，2）的方向性；
节点1在节点3之前，满足边（1，3）的方向性；
节点2在节点4之前，满足边（2，4）的方向性；
节点3在节点4之前，满足边（3，4）的方向性；
因此，该线性序列可称之为图的一个拓扑排序。但是， 一个有向无回路图拓扑排序的结果并不是唯一的，
如（1，3，2，4）这个线性序列也是该图的拓扑排序结果。需要注意的是，若图含回路则是不可能存在这样的线性序列的。
拓扑排序可以看成是图的所有节点沿水平线排成的一个序列，使得所有的有向边均从左指向右，即将图映射为一个线性序列。 完成拓扑排序后，每个节点都具有无后效性，终点的状态可由起点通过递推等方式转移过来。

[练习]
1．所有连通的有向图都可以实现拓扑排序。（　　）
2．对同一个图而言，拓扑排序的结构是唯一的。（　　）
3．拓扑排序中入度为0的结点总会排在入度大于0的结点的前面。（　　）
4．拓扑排序结果序列中的第一个结点一定是入度大于0的点。（　　）

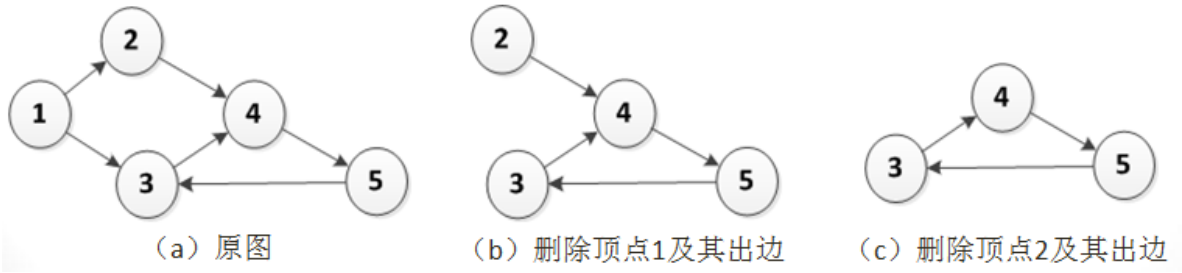
[答案]

1.错，存在环的图无法实现拓扑排序。
2.错，拓扑排序的结果不是唯一的，对于图中的一个有向边（u，v）只需保证在序列中u在v之前即可。
3.错，需要考虑图为非连通图的情况。
4.错，拓扑排序的结果会包含图中的所有节点，如第一个节点入度大于0 ，则它绝非序列中的第一个。

拓扑排序的实现

一、删边法

对于一个有向图，由于每一条拓扑子路径的首节点的入度为0，因此可以采取如下办法：
（1）从图中选择一个入度为0的节点且输出之。
（2）从图中删除该节点及其所有出边（即与之相邻的所有节点的入度减1）。
反复执行这两个步骤，直至所有节点都输出，整个拓扑排序完成。或者直至剩图中再没有入度为0的节点，这就说明此图中有环，拓扑排序不能再进行下去了。



（a）原图　（b）删除顶点1及其出边　（c）删除顶点2及其出边

[代码]

```
void toposort(){
    for(int i=1;i<=n;i++){
        if(d[i]==0)q.push(i);//入度为0的节点入队
    }
    while(!q.empty()){
        int u=q.front();
        a[++len]=u;//len用于统计整个过程中遍历的节点总数
        q.pop();
        for(int i=0;i<G[u].size();i++){
            int v=G[u][i];
            d[v]--;//v入度减1，即删去u节点及其所有出边
            if(d[v]==0){
                q.push(v);
            }
        }
    }
    if(len!=n){
        cout<<"the graph has cycle."<<endl;
    }else{
        for(int i=1;i<=len;i++)cout<<a[i]<<" ";
    }
}

```

在这个过程中，进行拓扑排序前我们需要先统计所有节点入度的时间复杂性为O（V），删除边花费的时间也是O（E）。因此，该方法的时间复杂度为O（V+E）。

二、采用DFS计算拓扑排序

DFS计算拓扑排序运用了有向无回路图DAG的一个重要特征：一个有向图是无回路的，当且仅当对图进行DFS遍历时没有反向边，即对于图中的任意边（u,v），都有f[v]<f[u]。拓扑序列表的节点按照f[u]递减顺序由左至右排列，即拓扑排序的节点是以与其完成时刻相反的顺序出现的。由DFS的运行时间为O（V+E），每个节点插入拓扑序列表首的时间为O（1），因此执行拓扑排序所需的总时间为O（V+E）。

[代码]

```
void dfs(int u){
    vis[u]=1;//标记节点正在访问中
    for(int i=0;i<G[u].size();i++){
        int v=G[u][i];
        if(vis[v]==1){//访问到正在访问的节点，说明存在环。拓扑排序无法继续开展
            cout<<"the graph has cycle."<<endl;
            exit(0);
        }
        if(vis[v]==0){//未访问过的节点
            dfs(v);
```

```
        }
    }
    vis[u]=-1;//该节点访问结束
    s.push(u);//用栈存储访问结束的节点，通过输出弹出栈顶，还原节点访问顺序
}
```

A.挤牛奶

【题目描述】

小A有N头奶牛，每头奶牛有个挤奶的时间；且某两头奶牛有挤奶顺序，即：xy，则只有在奶牛x挤完奶时，才能挤奶牛y。现在给定N头奶牛的挤奶时间，及M对先后关系，求N头奶牛都挤完奶的最早时间。

【输入】

第一行为两个空格隔开的整数N和M。
以下N行，第i+1行表示第i头奶牛的挤奶时间T_i；
以下M行，每行两个整数x,y，表示奶牛x在奶牛y之前挤奶。保证关系无环，即保证有解。

【输出】

输出共一行一个整数Ret，N头奶牛都挤完奶的最早时间。

样例输入	样例输出
3 1 10 5 6 3 2	11

【提示】

[样例解释]
先挤奶牛1和3，第6秒时，奶牛3挤完了；接着挤奶牛1和2，第10秒时，奶牛1挤完了，第11秒时，奶牛2也挤完了。
[数据规模]
对于30%的数据：$1 \leq N \leq 100$；$1 \leq M \leq 50$；
对于100%的数据：$1 \leq N \leq 10,000$；$1 \leq M \leq 50,000$；$1 \leq T_i \leq 100,000$。

【分析1】

本题要求出全局最大耗时。我们可利用深搜思想，由拓扑排序求以入度为0的节点为起点的一条线性序列的总耗时。图中多条线性序列的耗时最大值为解。

【代码1】

```
#include<iostream>
#include<vector>
using namespace std;
int n,m,u,v,r[10001];
long long maxn,a[10001],f[10001];
bool vis[10001];
vector<int> g[10001];
long long  dfs(int u){
    if(g[u].size()==0) return a[u];//该线性序列末端
    if(f[u]!=0) return f[u];
    long long ans=0;
```

```
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        ans=max(ans,dfs(v));
    }
    return f[u]=ans+a[u];
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u].push_back(v);
        r[v]++;
    }
    for(int i=1;i<=n;i++){
        if(!r[i]){//入度为0，以此为起点拓展延伸
            f[i]=dfs(i);//该序列的总耗时
        }
        maxn=max(maxn,f[i]);
    }
    cout<<maxn;
}
```

【分析2】

本题也可以采用广搜的思想，通过一步步找到入度为0的节点开展拓扑排序，有序在节点间转移状态，求出完成每一头牛挤牛奶所需的耗时，全局最大耗时为解。

【代码2】

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
int n,m,u,v,r[10001];
long long maxn,a[10001],t[10001];
vector<int> g[10001];
queue<int> q;
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u].push_back(v);
        r[v]++;
    }
    for(int i=1;i<=n;i++){
        if(!r[i]){
            q.push(i);
            t[i]=a[i];//入度为0，不需等待，直接完成挤奶工作
        }
    }
    while(!q.empty()){
        int u=q.front();
        maxn=max(maxn,t[u]);
```

```
        q.pop();
        for(int i=0;i<g[u].size();i++){
            int v=g[u][i];
            t[v]=max(t[v],t[u]+a[v]);//v需要等待u完成后才能开始，多个状态中取最大值
            r[v]--;
            if(!r[v]) q.push(v);
        }
    }
    cout<<maxn;
}
```

B.奖金

【题目描述】

由于无敌的凡凡在2005年世界英俊帅气男总决选中胜出，Yali Company总经理Mr.Z心情好，决定给每位员工发奖金。公司决定以每个人本年在公司的贡献为标准来计算他们得到奖金的多少。于是Mr.Z下令召开m方会谈。每位参加会谈的代表提出了自己的意见："我认为员工a的奖金应该比b高！"Mr.Z决定要找出一种奖金方案，满足各位代表的意见，且同时使得总奖金数最少。每位员工奖金最少为100元。

【输入】

第一行两个整数n,m，表示员工总数和代表数；
以下m行，每行2个整数a,b，表示某个代表认为第a号员工奖金应该比第b号员工高。

【输出】

若无法找到合法方案，则输出"-1"；否则输出一个数表示最少总奖金。

样例输入	
2 1 1 2	201

【提示】

数据范围：
80%的数据满足$n <= 1000$，$m <= 2000$；
100%的数据满足$n <= 10000$，$m <= 20000$。

【分析】

首先通过建图，声明每个员工的奖金的高低级关系。如边（u,v）表示v的奖金需高于u，至少高1元，即f[v]=max(f[v],f[u]+1)。可通过删边法实现拓扑排序，递推求出每个员工的可行最小值，所有员工奖金和为解。

【代码】

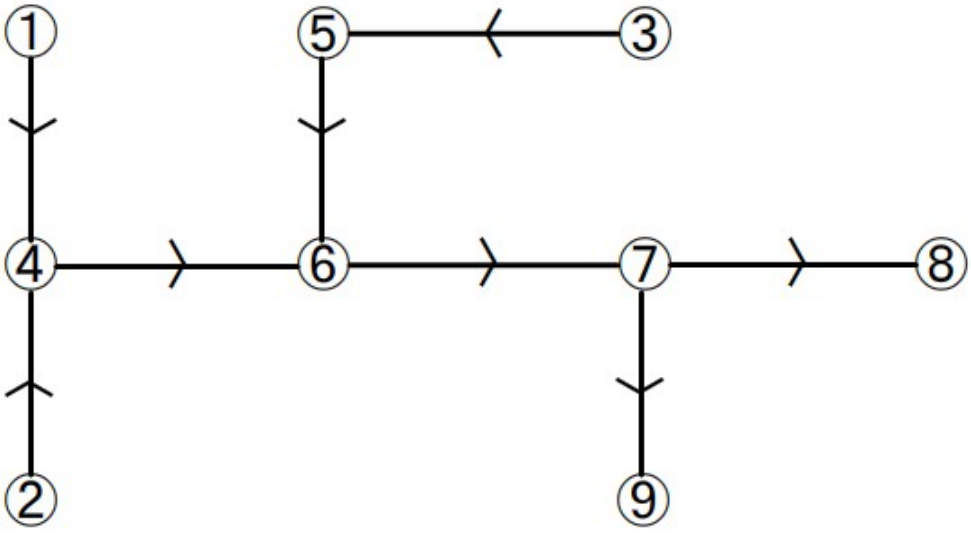
```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;
int n,m,u,v,d[10001],val[10001],sum;
vector<int> g[10001];
queue<int> q;
```

```cpp
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>v>>u;
        g[u].push_back(v);
        d[v]++;
    }
    for(int i=1;i<=n;i++){
        if(!d[i]){
            val[i]=100;//入度为0 ， 则说明该员工的奖金处于最底层
            q.push(i);
        }
    }
    int ans=0;
    while(!q.empty()){
        int u=q.front();
        sum+=val[u];//入度为0，则该员工的奖金已能确定
        q.pop();
        ans++;
        for(int i=0;i<g[u].size();i++){
            int v=g[u][i];
            val[v]=max(val[v],val[u]+1);
            d[v]--;
            if(!d[v]) q.push(v);
        }
    }
    if(ans==n) cout<<sum;
    else cout<<-1;
}
```

C.巧克力牛奶

【题目描述】

FJ的牛奶生产和运输是一个复杂的过程，他用挤奶器给他的那么多头奶牛挤奶然后流入管道。
每一个管道把一台挤奶器和一个可能连有一台或多台挤奶器的接口连接起来（这样几个管道里的牛奶就汇合了）。然后牛奶流入附加管道（连在各个接口之间的管道）直到流到中央管道，通向储存室。 然后这些牛奶又经历一个逆向的过程通过管道分流到各个牛奶桶，最后被运至市场。
FJ发现对于牛奶来说有一种最多的方式从一个接口流到另一个接口。并且由于FJ是一个高效率的人，他需要确保每一个管道都有牛奶经过，也就是说，没有多余的管道。
如果我们把每个挤奶机、接口和奶罐都看成一个节点，就共有 N 个节点，输入有序的节点对A_i和B_i，代表牛奶从A_i节点流到B_i节点，如果没有相对应的父节点，那就说明这是一个挤奶器，同样的如果没有对应的尾节点，则这是一个奶罐。
FJ把这些节点编号为 1..N，这样表示牛奶只能从编号较小节点流到编号较大节点。也就是说有N-1个管道。
这几天巧克力牛奶的需求量激增，所以FJ想要在某一个接口处安装一个巧克力混合器以得到巧克力牛奶，为了节约，FJ只买了一个巧克力混合器。所以他想把这个东西放到一个所有牛奶都能经过的接口，事实上，有这种接口存在。
编程帮助FJ找到这样的节点（注意：不能把巧克力混合器放在挤奶机里）。
例如：这样的情况:



一看就知道，在6或者7可以装巧克力混合器。

【输入】

第一行：一个整数N；
第2..N行 ：第i+1行用两个整数A_i 和 B_i，表示第i个管道的两端。

【输出】

若干行，升序输出每一个可以安装混合器的节点。

样例输入	样例输出
9 1 4 3 5 2 4 5 6 6 7 7 8 4 6 7 9	6 7

【提示】

数据规模：
【数据规模】
对于40%的数据： $2 \leq N \leq 250$ ；
对于50%的数据： $2 \leq N \leq 6,000$ ；
对于100%的数据：$2 \leq N \leq 100,000$；$1 \leq A_i < B_i \leq N$；

【分析】

题意要找的节点需要满足一下条件：1.非挤奶器，即入度大于0；2.所有挤奶器都通向该节点；3.这一个节点可通向所有奶罐（出度为0的节点）。在遍历过程中，我们需要对节点的出度进行考虑。如已知节点u为可行节点，若其出度为1，即它的出边指向的节点v也存在成为可行节点的可能性。但若出度大于1，则它不同的出边将指向v1、v2、v3等节点，而这些节点将会对进过的牛奶进行分流到不同的奶罐，无法成为满足题意的节点。通过此判断，一可在遍历过程中进行剪枝提高效率，二可保证节点满足条件3。

【代码】

```cpp
#include<iostream>
#include<cstdio>
#include<vector>
#include<queue>
using namespace std;
int n,u,v,f[100001],d[100001],tp[100001],tot;
vector<int> g[100001];
queue<int> q;
int main(){
    scanf("%d",&n);
    for(int i=1;i<n;i++){
        scanf("%d %d",&u,&v);
        g[u].push_back(v);
        d[v]++;
    }
    for(int i=1;i<=n;i++){
        if(!d[i]){
            tot++;
            tp[i]=1;
            f[i]=1;
            q.push(i);
        }
    }
    while(!q.empty()){
        int u=q.front();
        q.pop();
        if(f[u]==tot&&g[u].size()>1) continue;//虽所有挤奶器都能经过，但下来会进行分流，后面的接管不可能为可行性选择
        for(int i=0;i<g[u].size();i++){
            int v=g[u][i];
            f[v]+=f[u];
            d[v]--;
            if(!d[v])q.push(v);
        }
    }
    for(int i=1;i<=n;i++){
        if(f[i]==tot&&tp[i]!=1){
            printf("%d\n",i);
        }
    }
}
```

D.流感会结束吗

【题目描述】

A班一共有n个学生。这n个学生里一共有m对朋友关系。
在流感发作期，每个健康学生都要看望当天他生病的朋友（如果有的话），并在第二天被传染上疾病（除非他在免疫期内）；
每个生病的学生在第二天都会痊愈，并在这一天具有免疫性。从第三天起，看望生病的朋友将再次使他染上流感。
初始时（第一天），只有一个学生患有流感。试问多少天后流感会自动结束。

【输入】

第一行输入两个正整数n和m。
接下来m行每行两个正整数x,y，表示编号为x的学生和编号为y的学生是一对朋友。输入数据保证每一对朋友关系只描述一次。
最后一行输入一个正整数，代表初始时患有流感的学生的编号。

【输出】

如果流感永远不会结束，请输出-1，否则输出多少天后流感会结束。
答案保证不超过2 000 000 000。

样例输入	样例输出
4 4 1 2 2 3 3 4 2 4 1	3

【提示】

[样例说明]
第一天1号学生生病，2号学生访问他；
第二天2号学生生病，其它三个学生访问他，由于1号处于免疫期，未患流感；
第三天3、4号学生生病，2号学生访问他们。
第四天3、4号学生痊愈，流感结束。

[数据范围]
$n, m <= 100000$

【分析】

首先要注意的是本题中对朋友关系进行建图以呈现访问路径，但是朋友关系非单项，一对朋友关系需建立一对双向边。通过关系图，朋友间线性转移状态，记录每个人的生病日期，以日期差距判断是否处于免疫期，若否，则将进一步扩展。在此过程中，需要注意如存在有人曾生病且已超过免疫期被再次传染，说明传染关系中存在死循环 ，无法结束。

【代码】

```cpp
#include<iostream>
#include<queue>
#include<vector>
#include<cmath>
#include<cstring>
using namespace std;
int n,m,day[100001],ans,u,v,x;
bool s[100001];
vector<int> g[100001];
queue<int> q;
int main(){
    cin>>n>>m;
    memset(day,-1,sizeof(day));
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);//朋友关系为双向关系
    }
```

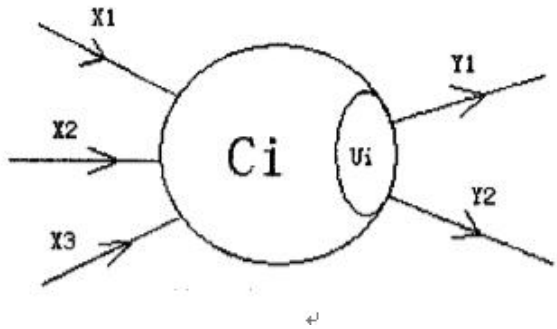
```
    cin>>x;
    s[x]=true;
    day[x]=1;
    q.push(x);//生病者入队
    ans=1;
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=0;i<g[u].size();i++){
            int v=g[u][i];
            if(abs(day[v]-day[u])<=1) continue;//处于免疫期
            q.push(v);
            day[v]=day[u]+1;
            ans=max(ans,day[v]);
            if(s[v]){//访问到曾生病且已超过免疫期者
                cout<<-1;
                return 0;
            }
            s[v]=true;
        }
    }
    cout<<ans;
}
```

E.神经网络

【题目描述】

人工神经网络（Artificial Neural Network）是一种新兴的具有自我学习能力的计算系统，在模式识别、函数逼近及贷款风险评估等诸多领域有广泛的应用。对神经网络的研究一直是当今的热门方向，兰兰同学在自学了一本神经网络的入门书籍后，提出了一个简化模型，他希望你能帮助他用程序检验这个神经网络模型的实用性。

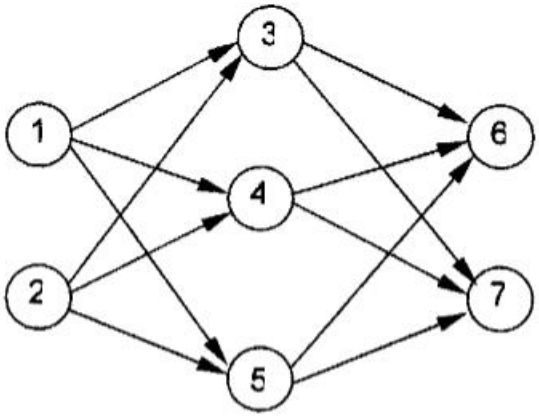
在兰兰的模型中，神经网络就是一张有向图，图中的节点称为神经元，而且两个神经元之间至多有一条边相连，下图是一个神经元的例子：



神经元〔编号为1）

图中，X1—X3是信息输入渠道，Y1 - Y2是信息输出渠道，C1表示神经元目前的状态，Ui是阈值，可视为神经元的一个内在参数。

神经元按一定的顺序排列，构成整个神经网络。在兰兰的模型之中，神经网络中的神经无分为几层；称为输入层、输出层，和若干个中间层。每层神经元只向下一层的神经元输出信息，只从上一层神经元接受信息。下图是一个简单的三层神经网络的例子。



兰兰规定，Ci服从公式：（其中n是网络中所有神经元的数目）

$$C_i = \sum_{(j,i)\in E} W_{ji} C_j - U_i$$

公式中的Wji（可能为负值）表示连接j号神经元和 i号神经元的边的权值。当 Ci大于0时，该神经元处于兴奋状态，否则就处于平静状态。当神经元处于兴奋状态时，下一秒它会向其他神经元传送信号，信号的强度为Ci。
如此．在输入层神经元被激发之后，整个网络系统就在信息传输的推动下进行运作。
现在，给定一个神经网络，及当前输入层神经元的状态（Ci），要求你的程序运算出最后网络输出层的状态。

【输入】

输入文件第一行是两个整数n（$1 \leq n \leq 100$）和p（$1 \leq p \leq 100$）。接下来n行，每行两个整数，第i＋1行是神经元i最初状态和其阈值（Ui），非输入层的神经元开始时状态必然为0。再下面P行，每行由两个整数i，j及一个整数Wij，表示连接神经元i、j的边权值为Wij。

【输出】

输出文件包含若干行，每行有两个整数，分别对应一个神经元的编号，及其最后的状态，两个整数间以空格分隔。仅输出最后状态非零的输出层神经元状态，并且按照编号由小到大顺序输出！
若输出层的神经元最后状态均为 0，则输出 NULL。

样例输入	样例输出
5 6 1 0 1 0 0 1 0 1 0 1 1 3 1 1 4 1 1 5 1 2 3 1 2 4 1 2 5 1	3 1 4 1 5 1

【分析】

本题中各神经元的激发（遍历顺序）满足拓扑排序特点。我们可以借助队列，将处于兴奋状态的神经元入队，层层推进进行状态转移。最后查找出出度为0的节点，即为输出层神经元，根据要求输出即可。

【代码】

```cpp
#include<iostream>
#include<vector>
#include<cstring>
#include<queue>
using namespace std;
int n,p,c[101],U[101],x,y,w,d[101];
struct node{
    int m,w;
};
bool flag;
vector<node> g[1001];
queue<int> q;
int main(){
    cin>>n>>p;
    for(int i=1;i<=n;i++){
        cin>>c[i]>>U[i];
        if(c[i]>0){
            q.push(i);//处于兴奋状态的神经元，能向下一层输出，入队准备扩展
        }else{
            c[i]-=U[i];//根据Ci的计算公式进行预处理
        }
    }
    for(int i=1;i<=p;i++){
        cin>>x>>y>>w;
        g[x].push_back((node){y,w});
        d[y]++;
    }
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=0;i<g[u].size();i++){
            node v=g[u][i];
            c[v.m]+=v.w*c[u];
            d[v.m]--;
            if(!d[v.m]&&c[v.m]>0){//注意：满足Ci值大于0才可继续向下一层传递
                q.push(v.m);
            }
        }
    }
    for(int i=1;i<=n;i++){
        if(g[i].size()==0&&c[i]>0){//出度为0的节点才属于输出层
            cout<<i<<" "<<c[i]<<endl;
            flag=true;
        }
    }
    if(!flag) cout<<"NULL";
}
```

F.车站分级

【题目描述】

一条单向的铁路线上，依次有编号为 1, 2, …, n 的 n 个火车站。每个火车站都有一个级别，最低为 1 级。现有若干趟车次在这条线路上行驶，每一趟都满足如下要求：如果这趟车次停靠了火车站 x，则始发站、终点站之间所有级别大于等于火车站 x 的都必须停靠。（注意：起始站和终点站自然也算作事先已知需要停靠的站点）
例如，下表是 5 趟车次的运行情况。其中，前 4 趟车次均满足要求，而第 5 趟车次由于停靠了 3 号火车站（ 2 级）却未停靠途经的 6 号火车站（亦为 2 级）而不满足要求。

车站编号	1		2		3		4		5		6		7		8		9
车站级别 / 车次	3		1		2		1		3		2		1		1		3
1	始	→	→	→	停	→	→	→	停	→	终						
2					始	→	→	→	停	→	终						
3	始	→	→	→	→	→	→	→	停	→	→	→	→	→	→	→	终
4							始	→	停	→	停	→	停	→	停	→	终
5					始	→	→	→	停	→	→	→	→	→	→	→	终

现有 m 趟车次的运行情况（全部满足要求），试推算这 n 个火车站至少分为几个不同的级别。

【输入】

第一行包含 2 个正整数 n, m，用一个空格隔开。
第 i + 1 行（ $1 \leq i \leq m$ ）中，首先是一个正整数 si（ $2 \leq s_i \leq n$ ），表示第 i 趟车次有 si 个停靠站；接下来有 s_i 个正整数，表示所有停靠站的编号，从小到大排列。每两个数之间用一个空格隔开。输入保证所有的车次都满足要求。

【输出】

输出只有一行，包含一个正整数，即 n 个火车站最少划分的级别数。

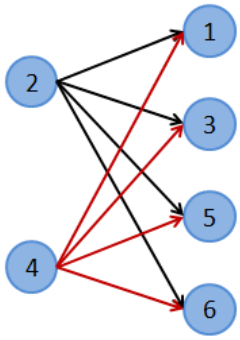
样例输出	样例输入
9 2 4 1 3 5 6 3 3 5 6	2
9 3 4 1 3 5 6 3 3 5 6 3 1 5 9	3

【提示】

对于20%的数据，$1 \leq n, m \leq 10$；
对于50%的数据，$1 \leq n, m \leq 100$；
对于100%的数据，$1 \leq n, m \leq 1000$。

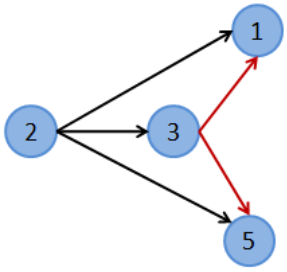
【分析】

通过对题意的分析，我们可将各个车站根据级别的高低顺序建图，这是本题的难点。所建的图中边（u，v）表示u的级别低于v。对于每一趟车，将起点和终点间每个未停靠的车站与停靠的车站建边，声明级别差异。
如样例1可建图如下：



第一趟列车起点为车站1终点为车站6，未停靠车站2，因此车站2的级别低于1、3、5、6，建有向边（2，1）、（2，3）、（2，5）、（2，6），未停靠的车站4同理进行建边。第二趟车起点为车站3终点为车站6，未停靠车站4，所需建的边（4，3）（4，5）（4，6）已存在，无需重复建边，建图完毕。从图中可见，图中所有路径的最长长度为1，因此只需划分为两个级别，即车站2与车站4为低级别，车站1、3、5、6为高级别。

再如样例2，我们对样例做一个精简，以便建图分析。
车站只保留1、2、3、5四个，2趟列车分别停靠（1，3，5）和（1，5），建图如下：



从图中可见，图中所有路径的最长长度为2，因此只需划分为三个级别，即车站2为最低级别，其次为车站3，车站4与5为最高级别。
建图后不难发现入度为0者即级别最低者，入队沿边进行扩展，逐步转移状态得出每个车站的级别。

【代码】

```
#include<iostream>
#include<vector>
#include<cstring>
#include<queue>
using namespace std;
int n,m,k,a[1001],d[1001],lev[1001],maxl=1;
bool f[1001],g[1001][1001];
queue<int> q;
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>k;
        memset(f,0,sizeof(f));
        for(int j=1;j<=k;j++){
            cin>>a[j];
            f[a[j]]=true;
        }
        for(int j=a[1];j<=a[k];j++){//只需考虑a[1]-a[k]这一段的车站
            if(!f[j]){//未停靠的车站j
                for(int l=1;l<=k;l++){
                    if(!g[j][a[l]]){//建边，标记j的级别低于a[l]
                        g[j][a[l]]=true;
```

```
                        d[a[l]]++;
                    }
                }
            }
        }
    }
    for(int i=1;i<=n;i++){
        if(d[i]==0){//入度为0，即为最低阶
            q.push(i);
            lev[i]=1;
        }
    }
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=1;i<=n;i++){
            if(g[u][i]){
                lev[i]=max(lev[i],lev[u]+1);
                maxl=max(maxl,lev[i]);
                d[i]--;
                if(!d[i]) q.push(i);
            }
        }
    }
    cout<<maxl;
}
```