高级数据结构

信智学院　陈琰宏

主要内容

01 拓扑排序概念

02 线性结构（栈、队列）

03 树形结构（二叉树、线段树、红黑树...）

04 图结构(最小生成树、最短路径...)

05 字符串(KMP、哈希...)

1 拓扑排序(Topological Sort)

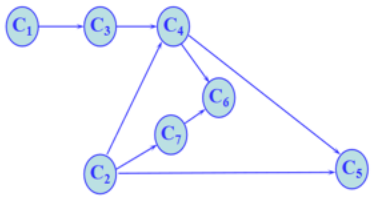
设G=(V，E)是一个具有n个顶点的有向图，V中顶点序列v1，v2，…，vn称为一个拓扑序列。在一个有向图中找一个拓扑序列的过程称为拓扑排序。

拓扑排序就是指有向无环图（Directed Acyclic Graph，DAG）中各顶点构成的有序序列。该序列满足如下条件：如果图中一顶点v_i到另一顶点v_j存在一条路径，那么v_j在此图的拓扑排序序列中位于v_i之后。

例如，计算机专业的学生必须完成一系列规定的基础课和专业课才能毕业，假设这些课程的名称与相应代号有如下关系：

课程代号	课程名称	先修课程
C_1	高等数学	无
C_2	程序设计	无
C_3	离散数学	C_1
C_4	数据结构	C_2，C_3
C_5	编译原理	C_2，C_4
C_6	操作系统	C_4，C_7
C_7	计算机组成原理	C_2

课程之间的先后关系可用有向图表示：



1 拓扑排序

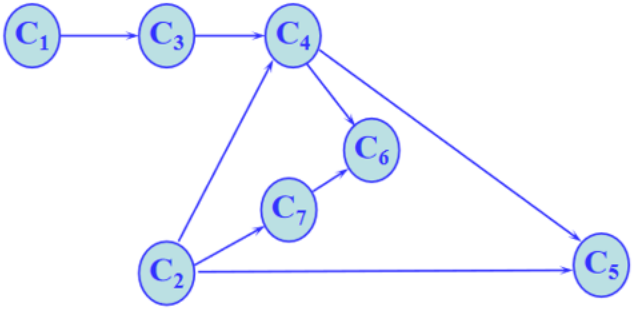
对这个有向图进行拓扑排序可得到一个拓扑序列：

C1-C3-C2-C4-C7-C6-C5。

也可得到另一个拓扑序列：

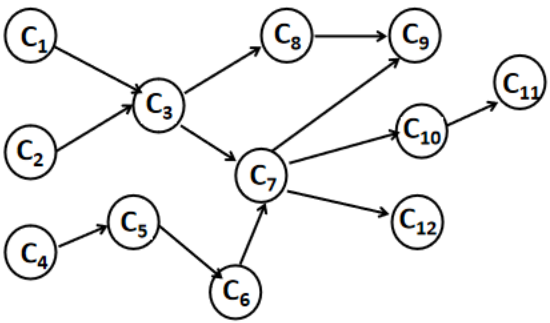
C2-C7-C1-C3-C4-C5-C6，

还可以得到其他的拓扑序列。



1.1 拓扑排序举例

【例】某大学计算机专业部分必修的课程以及修学这些课程的先后顺序关系。



	课程名称	先修课程
C_1	程序设计基础	无
C_2	离散数学	无
C_3	数据结构	C_1,C_2
C_4	高等数学1	无
C_5	高等数学2	C_4
C_6	线性代数	C_5
C_7	汇编语言	C_3,C_6
C_8	数据库	C_3
C_9	操作系统	C_7
C_{10}	计算机组成原理	C_7
C_{11}	编译原理	C_{10}
C_{12}	计算机网络	C_7课程编号

- 下列两种顶点序列都是该DAG的拓扑排序：
  - C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}
  - C_4, C_1, C_2, C_3, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{12}, C_{11}

1.2 拓扑排序步骤

❖求出给定DAG的一个拓扑序列，相当于进行一个作业调度。
❖如何来求一个拓扑序列呢？

简单算法：

[step 1] 如果找得到任何一个入度为0的顶点v，则step 2，否则step 4；

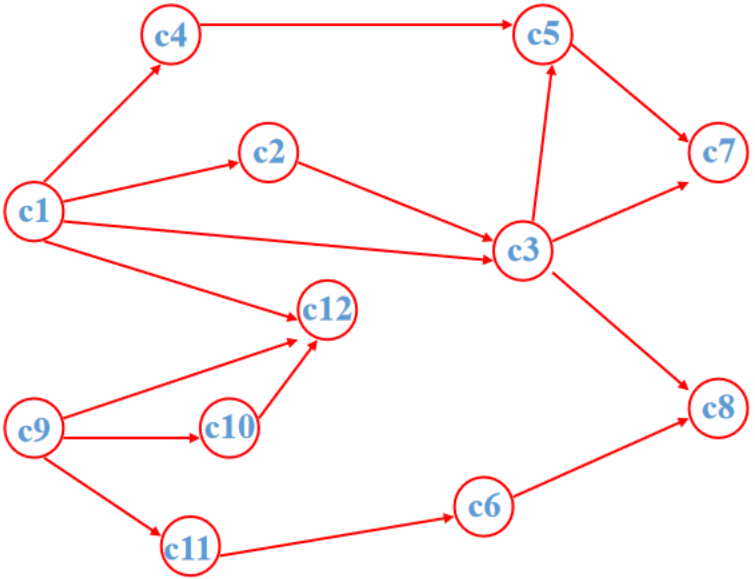
[step 2] 输出顶点v，并从图中删除该顶点以及与其相连的所有边；

[step 3] 对改变后的图重复这一过程，转step 1；

[step 4] 如果已经输出全部顶点，则结束；否则该有向图不是DAG。

➢ 理论依据是基于下面的结论：一个顶点数|V| > 1的有向图，
如果每个顶点的入度都大于0，那么它必定存在回路。

1.3 排序过程模拟



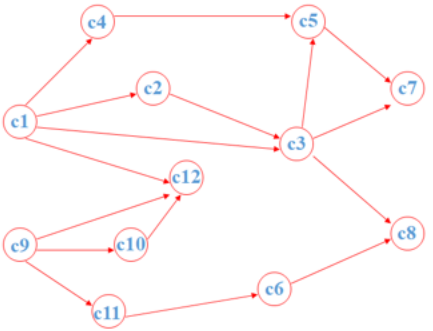
1.3 排序过程模拟

拓扑排序的步骤：

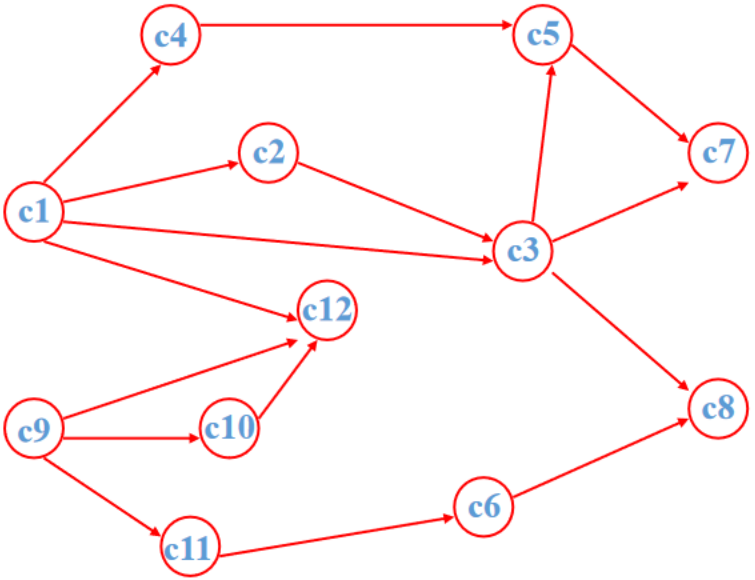
（1）从图中选择一个入度为0的顶点且输出之；

（2）从图中删掉该顶点及其所有以该顶点为弧尾的弧；

反复执行这两个步骤，直到所有的顶点都被输出，输出的序列就是这个无环有向图的拓扑序列。

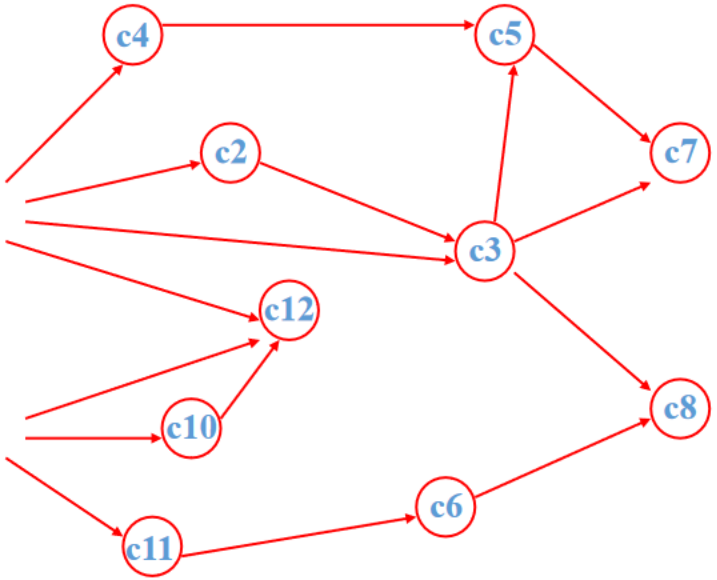


1.3 排序过程模拟



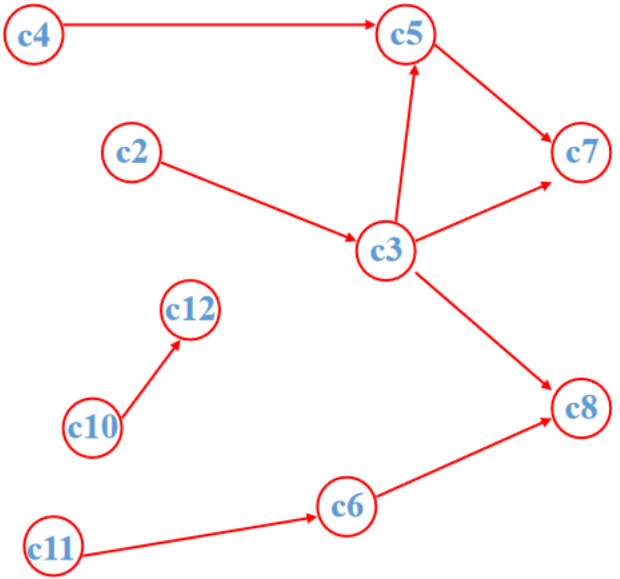
拓扑序列：c1,c91

1.3 排序过程模拟



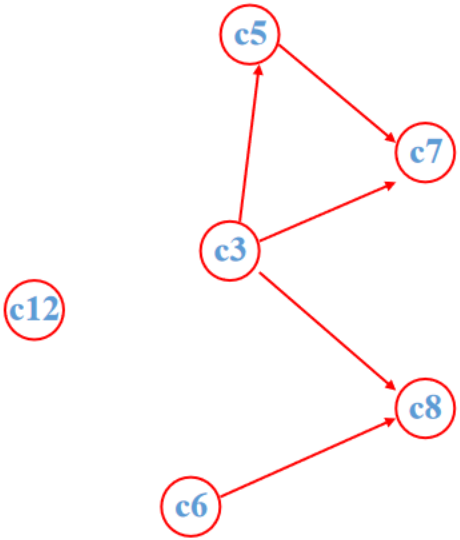
拓扑序列：c1,c9

1.3 排序过程模拟



拓扑序列：c1,c9,c4,c2,c10,c11

1.3 排序过程模拟



拓扑序列：c1,c9,c4,c2,c10,c11

1.3 排序过程模拟

拓扑序列：c1,c9,c4,c2,c10,c11,c12,c3,c6

1.3 排序过程模拟

拓扑序列：c1,c9,c4,c2,c10,c11,c12,c3,c6,c5,c8,c7

2 算法描述

❖如何来求一个拓扑序列呢？

简单算法：

[step 1] 如果找得到任何一个入度为0的顶点v，则step 2，否则step 4；

[step 2] 输出顶点v，并从图中删除该顶点以及与其相连的所有边；

[step 3] 对改变后的图重复这一过程，转step 1；

[step 4] 如果已经输出全部顶点，则结束；否则该有向图不是DAG。

➢ 理论依据是基于下面的结论：一个顶点数|V| > 1的有向图，
如果每个顶点的入度都大于0，那么它必定存在回路。

2 算法描述

```
void TopologicalSort ( Graph G, int TopNum[ ] )
{
    int Counter;     /* 拓扑序号 */
    int v, w;
    int *InDegree = (int *)malloc( G.n * sizeof(int) );
    GetInDegree(G, InDegree);    /* 计算图G中各顶点的入度 */
    for( Counter = 0; Counter < G.n; Counter++ ) {
        v = FindNewVertexOfDegreeZero( );
        /* 查找入度为0的顶点。若找不到入度为0的顶点，返回
NotAVertex */
        if ( v == NotAVertex
            Error( "图存在环
            break;
        }
        TopNum[v] = Counter;
        for( G中关于v 的每个邻接点w )
            Indegree[w]--;  /* 与顶点v相连的各顶点的入度减1 */
    }
}
```

检查整个**InDegree**数组，所需时间是**O(|V|)**，此函数被调用**|V|**次，故本算法的时间复杂性为 **$O(|V|^2)$**

改进：

高效算法：（核心思想是用一个队列动态记录所有入度为0的顶点）

[step 0] 求各顶点入度，并存于InDegree[ ]，把入度为0的顶点入队列q；

[step 1] 如果队列非空，则step 2，否则step 4；

[step 2] 输出队列头顶点v；

[step 3] 删除与其相连的所有边，即v的所有邻接点的入度减1，转step 1；

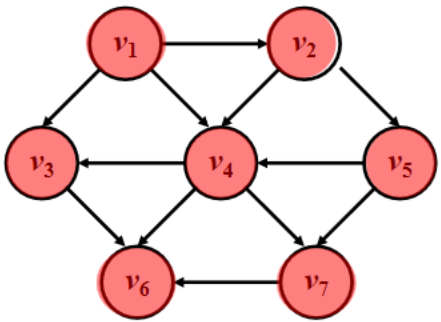
[step 4] 如果已经输出全部顶点，则结束；否则该有向图不是DAG。

改进：

```
void ToplogicalSort ( Graph G, int TopNum[ ] )
{  Queue queue;
   int Counter = 0;    int v, w;
   int *InDegree = (int *)malloc( G.n * si
   GetInDegree(G, InDegree);  /* 计算
   queue = CreateQueue( G.n ); /* 创建空
   for( v = 1; v <= G.n; v++ )
      if( InDegree[v]==0 )     Ad      queue, v );
   while( ! IsEmptyQ(queue) ) {
      v = DeleteQ( queue );
      TopNum[v] = ++Counter;       /* 赋顶点拓扑排序序号 */
      for( G中关于v 的每个邻接点w )
        if( --InDegree[ w ] == 0 )
           AddQ( queue, w );
   }
   if( Counter != G.n )     Error( "图存在环路" );
   DisposeQueue(queue);   /* 释放队列空间*/
   free(InDegree);
}
```

While循环|V|次，最多每一条边处理一次，故本算法的时间复杂性为 O(|V|+|E|)

拓扑排序示例



队列

入度：

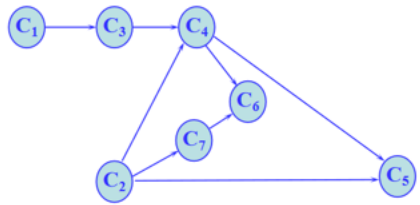
v_1	0
v_2	0
v_3	0
v_4	0
v_5	0
v_6	0
v_7	0

$V_1 \rightarrow \quad V_2 \rightarrow \; V_5 \rightarrow \quad V_4 \rightarrow \quad V_3 \rightarrow \; V_7 \rightarrow \quad V_6$

核心代码1

```
7   vector<int> G[55];//邻接表
8   int Degree[55];//存储入度
9   int GE[55][55];//输入原始数据
10  queue<int> p;//用队列存储度为0点节点
```

```
12  void Input(int N)
13  {
14
15      for(int i=0;i<N;i++)
16          for(int j=0;j<N;j++)
17              cin>> GE[i][j];//输入邻接矩阵
18      for(int i=0;i<N;i++)
19          {
20              int count=0,j;
21              for(int j=0;j<N;j++)
22              {
23                  if(GE[i][j]==1)
24                      G[i].push_back(j);//存储出边信息
25                  if(GE[j][i]==1)
26                      count++;//计数顶点入度
27              }
28              Degree[i]=count;//存储每个顶点的入度
29          }
30  }
```



核心代码2

```
33  bool TopologicalSort()
34  {
35      int num=0;
36      //记录加入拓扑排序的顶点数
37      queue<int> p;
38      for(int i=0;i<n;i++)
39      {//找到入度为0的点入队
40          if(Degree[i]==0)
41          {
42              p.push(i);
43              Degree[i]=-1;
44              //记录以被处理
45          }
46      }
47      while(!p.empty())
48      {
66
67      if(num==n) return true;
68      else return false;
69  }
```

```
46      while(!p.empty())
47      {
48          int u=p.front();
49          node.push(u);//node保存输出的拓扑序列
50          p.pop();//处理完出队
51          for(int i=0;i<G[u].size();i++)
52          {//遍历u点达到的节点，入度减1
53              int v=G[u][i];
54              Degree[v]--;
55              if(Degree[v]==0)
56              {//入度为0的点，入队
57                  p.push(v);
58                  Degree[v]=-1;
59              }
60          }
61
62          G[u].clear();
63          num++;
64      }
```

3.1 [2893]家谱树

有个人的家族很大，辈分关系很混乱，请你帮整理一下这种关系。给出每个人的孩子的信息。输出一个序列，使得每个人的后辈都比那个人后列出。

输入描述：

第1行一个整数N（1≤N≤100），表示家族的人数；
接下来N行，第I行描述第I个人的儿子；
每行最后是0表示描述完毕。（碰到0，表示当前节点的孩子结束，如此就不用表示多少个孩子的n了）

输出描述：

输出一个序列，使得每个人的后辈都比那个人后列出；
如果有多解输出任意一解。

样例输入：

```
5
0
4 5 1 0
1 0
5 3 0
3 0
```

样例输出：

```
2 4 5 3 1
```

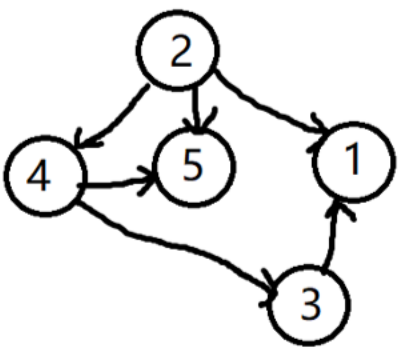
分析

我们把每个人抽象成结点，每个孩子都可以连一条有向边。我们可以用数组模拟邻接表，记录每个结点的出度，入度。并且用一个二维数组表示第i个点发出j条边到达的点。

每次输出出度为0的结点，然后再将有关的结点的出度减一，再找初度为0的结点输出，直到结束。

样例输入：

```
5
0
4 5 1 0
1 0
5 3 0
3 0
```



[2893]家谱树

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   const int maxn =135;
4   vector<int> g[maxn];//邻接表保存边信息
5   int in[maxn],n;//存顶点入度信息
6   void topsort(){
24  int main(){
25      int i,j,x;
26      cin>>n;
27      for(i=1;i<=n;i++){
28          while(cin>>x&&x){
29              in[x]++;//入度+1
30              g[i].push_back(x);//建立邻接表
31
32          }
33      }
34      topsort();
35      return 0;
36  }
```

```
6  void topsort(){
7      queue<int> q;
8      for(int i=1;i<=n;i++){
9          if(!in[i])
10         q.push(i);/*将入度为0的入队*/
11     }
12     while(!q.empty()){
13         int t=q.front();/*取出入度为0的*/
14         q.pop();
15         printf("%d ",t);/*出队输出*/
16         for(int i=0;i<g[t].size();i++){
17             int x=g[t][i];
18             in[x]--;/*对t出发能到的点都减一*/
19             if(!in[x]) /*更新，入度为0进队*/
20             q.push(x);
21         }
22     }
23 }
```

3.2 [7295]奖金

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。

【输入格式】

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。

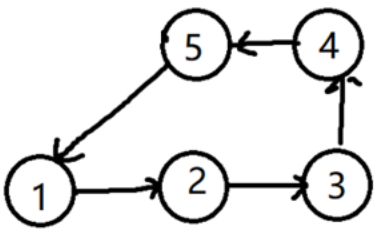
【输出格式】

若无法找到合理方案，则输出“Poor Xed”；否则输出一个数表示最少总奖金

[7295]奖金

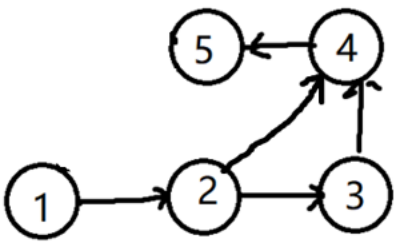
样例输入
2　1//表示员工总数和代表数
1　2 //整数a,b，第a号员工奖金应该比第b号员工高
样例输出
201
解释：1号奖金>2号，每位最少100，则奖金=100+101=201

6 5
1 2
2 3
3 4
4 5
5 1



Poor Xed

6 5
1 2
2 3
3 4
4 5
2 4



610

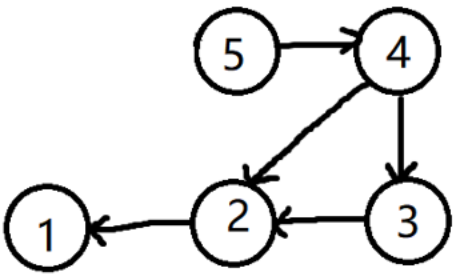
分析

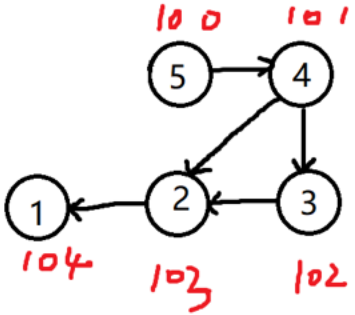
这道题应该使用拓扑排序，首先建立一个有向图，利用各个点的之间的大小关系建立最小等式。

即a>b　　-->　a=b+1;

这样使得奖金总数最少，满足题目要求。

6 5
1 2
2 3
3 4
4 5
2 4





代码

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=10005;
const int N=20005;
int n,m,sum;
int in[maxn],f[maxn];//  in入度  f每个人的奖金
vector<int> g[maxn];//建立邻接表
int topsort(){ //拓扑排序
int main(){
    int i,j,k,a,b;
    cin>>n>>m; //n个员工，m个代表
    for(i=1;i<=m;i++){
        cin>>a>>b;
        g[b].push_back(a);  //表示b指向a
        in[a]++; //a入度加1
    }
    for(i=1;i<=n;i++){
        f[i]=100;//刚开始每个员工的奖金为最低的100
    }
    if(topsort()){ //若排序成功
        for(i=1;i<=n;i++){
            sum+=f[i]; //奖金总和
        }
        printf("%d",sum);
    }
    else printf("Poor Xed");//排序失败
    return 0;
}
```

代码

```
int topsort(){ //拓扑排序
    int cnt=0;   //表示利用拓扑排序，能够排序的数目
    queue<int> q; //建立队列
    for(int i=1;i<=n;i++){
        if(in[i]==0) q.push(i);
        //找出入度为0的点，压入队列（
        //入度为0，表示这个点的奖金最低）
    }
    while(!q.empty()){
        int p=q.front(); //出队
        cnt++;
        q.pop();
        for(int i=0;i<g[p].size();i++){ //遍历这个点指向的所有点
            int x=g[p][i];
            f[x]=max(f[x],f[p]+1);
                //p这个点所指向的点x的奖金比 p的奖金最少要大1
            in[x]--; //x的入度减1
            if(in[x]==0){            //若此时x的入度为0，则入队
                q.push(x);
            }
        }
    }
    if(cnt==n){ //若cnt=n，表示完成了排序，否则排序失败
        return 1;
    }
    else return 0;
}
```

[3405] 确定排序序列

题目描述：

一个由不同的值组成的按升序排序的序列，通常使用小于操作符，把元素从小到大排列。

例如，有序序列A，B，C，D表示A<B，B<C和C<D。

现给你一组形如A<B的关系，请你确定是否已经形成一个排序的序列。

输入描述：

输入包含多组测试数据。每组输入的第一行是两个正整数n和m。

n表示排序对象的个数，2<=n<=26。排序对象是字母表开始的n个大写字符。

m表示形如A<B的关系的个数。

接下来m行，每行输入一个关系，由三个字符构成：第一个大写字母，符号“<”，第二个大写字母。字母不会超过字母表开始的n个字母的范围。

当n=m=0时，输入结束。

[3405] 确定排序序列

输出描述：

对于每组输入，输出一行。该行将是以下三者之一：

Sorted sequence determined after xxx relations: yyy...y. （在xxx个关系后，确定了排序序列：yyy...y）

Sorted sequence cannot be determined. （不能确定排序序列）

Inconsistency found after xxx relations. （在xxx个关系后，发现关系矛盾）

解释说明：

xxx是处理关系时，确定排序序列已经形成或发现关系矛盾时的关系数目，哪种情况先出现，就输出哪种。yyy...y是排序的升序序列。

样例输入

[3405] 确定排序序列

样例输入：

```
4 6
A<B
A<C
B<C
C<D
B<D
A<B
3 2
A<B
B<A
26 1
A<Z
0 0
```

样例输出：

Sorted sequence determined after 4 relations: ABCD.
Inconsistency found after 2 relations.
Sorted sequence cannot be determined.

[3405] 确定排序序列

分析：

根据题意能看出可以构成一个有向无环图，就想到要用拓扑排序来做，选择一个没有前驱的顶点进行输出，删除所有和他相关的边并循环到所有顶点输出，或者图中不存在没有前驱的顶点结束以此来判断这个图是否成环。根据输出描述，可以分为三种情况：1、成环矛盾 2、不能确定排序 3、确定排序。这样就可以很清晰的写出过程了

```cpp
#include <bits/stdc++.h>
using namespace std;
int N, M;
const int MAXN = 27, MAXM = 1000;
int g[MAXN][MAXN], in[MAXN],  ans[MAXN];
int topoSort() {
    int in1[MAXN];
    for(int i = 0; i < N; i++) in1[i] = in[i];
    for(int k = 0; k < N; k++) {
        int i = 0;
        while(in1[i]!=0) {
            i++;
            if(i >= N)
                return 2;   //成环，矛盾
        }
        ans[k] = i;
        in1[i]--;
        for(int j = 0; j < N; j++)
            if(g[i][j])
                in1[j]--;
    }
    for(int i = 0; i < N-1; i++) {
        int ok = 0;
        if(g[ans[i]][ans[i+1]])
            ok = 1;
        if(!ok)
            return 1;   //不能确定排列序列
    }
    return 0;           //确定排序
}
```

```
int main() {

    while(scanf("%d%d", &N, &M)!=EOF && !(!N&&!M)) {
        memset(g, 0, sizeof(g));
        memset(in, 0, sizeof(in));
        char s[10];
        int next = 0;
        for(int i = 0; i < M; i++) {
            scanf("%s", s);
            if(next)
                continue;
            int u = s[0] - 'A', v = s[2] - 'A';
            g[u][v] = 1;
            in[v]++;
            int r = topoSort();
            if(r == 0) {    //确定了排序
                printf("Sorted sequence determined after %d relations: ", i+1);
                for(int j = 0; j < N; j++) printf("%c", ans[j]+'A');
                printf(".\n");
                next = 1;
                continue;
            } else if(r == 1 && i == M-1)  //不能确定排序序列
                printf("Sorted sequence cannot be determined.\n");
            else if(r == 2) {  //关系矛盾，成环了
                printf("Inconsistency found after %d relations.\n", i+1);
                next = 1;
                continue;
            }
        }
    }
    return 0;
}
```

3.3 [2944]病毒

题目描述：

有一天，小y突然发现自己的计算机感染了一种病毒！还好，小y发现这种病毒很弱，只是会把文档中的所有字母替换成其它字母，但并不改变顺序，也不会增加和删除字母。

现在怎么恢复原来的文档呢！小y很聪明，他在其他没有感染病毒的机器上，生成了一个由若干单词构成的字典，字典中的单词是按照字母顺序排列的，他把这个文件拷贝到自己的机器里，故意让它感染上病毒，他想利用这个字典文件原来的有序性，找到病毒替换字母的规律，再用来恢复其它文档。

现在你的任务是：告诉你被病毒感染了的字典，要你恢复一个字母串。

[2944]病毒

输入描述：

第一行为整数K（≤50000），表示字典中的单词个数。

以下K行，是被病毒感染了的字典，每行一个单词。

最后一行是需要你恢复的一串字母。

所有字母均为小写。

输出描述：

输出仅一行，为恢复后的一串字母。当然也有可能出现字典不完整、甚至字典是错的情况，这时请输出一个0。

[2944]病毒

样例输入：

6　　字典中的单词个数

cebdbac
cac
ecd
dca
aba
bac

被病毒感染了的字典

cedab　　需要你恢复的字母

样例输出：

abcde　　恢复后的字母

[2944]病毒

样例字典		还原后的字典
cebdbac	→	abeceda
cac	→	ada
ecd	→	bac
dca	→	cad
aba	→	ded
bac	→	eda

字典序排列

根据字典序的特性，比较相邻两个单词的第一个不相同的字母，可以得出这两个字母的大小关系，而不能确定它们后面的字母的大小关系。

如abeceda
ada
} b<d，而e不小于a

abcd
abc
} 从这样的两个单词中无法得到字母的大小关系

结论：

每两个相邻的字母串至多可以得出一对有大小关系的字母，我们可以根据大小关系建图，拓扑排序得出完整的字母排列顺序，与原字母一一对应，保存这些对应关系，逐一还原输入数据中需要恢复的字母串，最后输出。

[2944]病毒

两种输出0的情况：

1.拓扑排序时若剩下的所有字母入度都不为零，即代表有环，字典错误，输出0。

2.若需要恢复的字母串中出现了大小关系未知的字母，则字典不完整，输出0。

[2944]病毒

```
#include<bits/stdc++.h>
using namespace std;
#define MAXN 50010
vector<int>v[27];
queue<int>q;
int in[27],vis[27];
//in:入度，vis:标记有效字母（字典中提取的一对有大小关系的字母）
string a[MAXN],target;//a:病毒感染了的字典，target:需要恢复的目标字母串
char ans[27];//ans:保存每个有效字母对应的原字母
int main(){
    int i,j,n,cnt=0;
    cin>>n;//字典中的单词个数n
    for(i=0;i<n;i++)cin>>a[i];
    for(i=0;i<n-1;i++){//寻找大小关系，建图
        for(j=0;j<a[i].size()&&j<a[i+1].size();j++){
            int x=a[i][j]-'a',y=a[i+1][j]-'a';//字母转成数字
            if(x==y)continue;
            v[x].push_back(y);
            in[y]++;//入度++
            if(vis[x]==0)vis[x]=1,cnt++;//标记每个有效字母，统计有效字母的个数
            if(vis[y]==0)vis[y]=1,cnt++;
            break;//找到一对关系就可以退出了，寻找下一个字母串
        }
    }
```

[2944]病毒

```
26      cin>>target;//需要恢复的目标字母串
27      for(i=0;i<target.size();i++)
28          if(vis[target[i]-'a']==0){
29              cout<<'0';
30              return 0;
31          }//目标字母串中出现了大小关系未知的字母
32      for(i=0;i<26;i++)
33          if(vis[i]&&in[i]==0)q.push(i);//找入度为0的点，入队
34      char t='a';//排在最前的字母对应的是 a
35      while(!q.empty()){//拓扑排序
36          int x=q.front();
37          q.pop();
38          ans[x]=t++;//第一个字母对应完之后 t++，第二个字母对应 b，以此类推
39          cnt--;//每弹出一个字母，cnt就减一
40          for(i=0;i<v[x].size();i++){
41              int to=v[x][i];
42              if(--in[to]==0)q.push(to);//新产生的入度为0的点入队
43          }
44      }
45      if(cnt){ //如果cnt！=0，说明还有字母没有入队过，出现了环，字典错误
46          cout<<'0';
47          return 0;
48      }
49      for(i=0;i<target.size();i++)//输出每个目标字母对应的原字母
50          cout<<ans[target[i]-'a'];
51      return 0;
52  }
```

3.4 [2943] 烦人的幻灯片

李教授将于今天下午作一次非常重要的演讲。不幸的事他不是一个非常爱整洁的人，他把自己演讲要用的幻灯片随便堆在了一起。因此，演讲之前他不得不去整理这些幻灯片。作为一个讲求效率的学者，他希望尽可能简单地完成它。教授这次演讲一共要用n张幻灯片（n≤26），这n张幻灯片按照演讲要使用的顺序已经用数字1~n编了号。因为幻灯片是透明的，所以我们不能一下子看清每一个数字所对应的幻灯片。

现在我们用大写字母A,B,C……再次把幻灯片依次编号。你的任务是编写一个程序，把幻灯片的数字编号和字母编号对应起来，显然这种对应应该是唯一的；若出现多种对应的情况或是某些数字编号和字母编号对应不起来，我们称对应是无法实现的。

3.4 [2943] 烦人的幻灯片

输入描述： 第一行只有一个整数n，表示有n张幻灯片，接下来的n行每行包括4个整数xmin,xmax,ymin,ymax（整数之间用空格分开）为幻灯片的坐标，这n张幻灯片按其在文件中出现的顺序从前到后依次编号为A,B,C……，再接下来的n行依次为n个数字编号的坐标x,y，显然在幻灯片之外是不会有数字的。

输出描述： 若是对应可以实现，输出文件应该包括n行，每一行为一个字母和一个数字，中间以一个空格隔开，并且每行以字母的升序排列，注意输出的字母要大写并且定格；反之，若是对应无法实现，在文件的第一行顶格输出None即可。首行末无多余的空格。

样例输入

```
4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11
```

样例输出

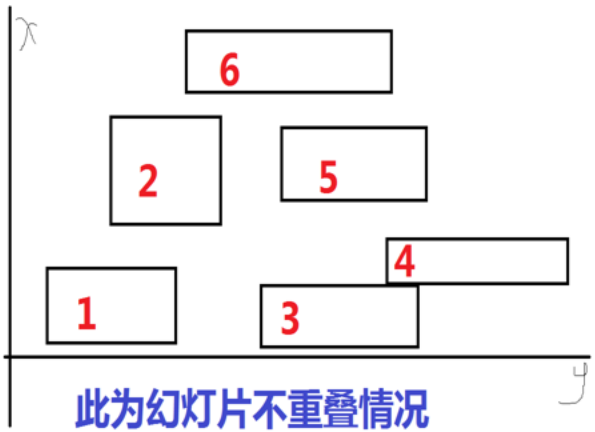
```
A 4
B 1
C 2
D 3
```

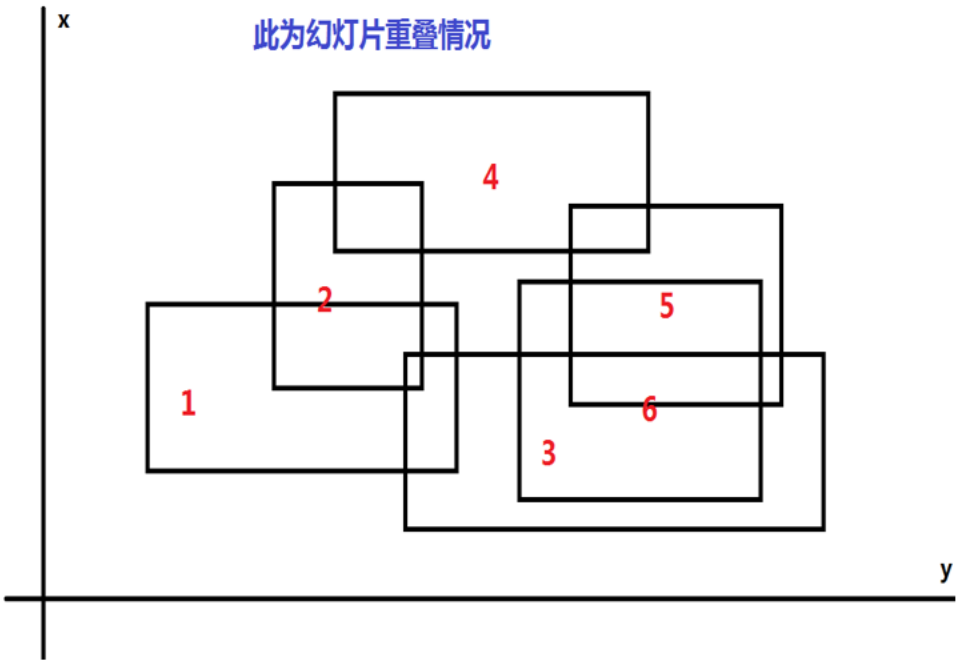
分析

样例输入
4　　// n张幻灯片
A: 6 22 10 20　//幻灯片的坐标, 4个整数xmin,xmax,ymin,ymax
B: 4 18 6 16
C: 8 20 2 18
D: 10 24 4 8
1：9 15　　// n行依次为n个数字编号的坐标 x,y
2：19 17
3：11 7
4：21 11
样例输出
A 4
B 1
C 2
D 3

分析

本题适合用拓扑排序，每个点与多个幻灯片相交，通过拓扑排序，一步步取出相匹配的点与幻灯片。




x
此为幻灯片重叠情况
4
2
5
1
6
3
y

```cpp
#include<bits/stdc++.h>
using namespace std;
int x[30];//幻灯片与点之间的关系
int y[30];//点与幻灯片之间的关系
int a[30][5];//储存幻灯片位置
int b[30][3];//储存点的位置
int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)//输入幻灯片位置
    {
        cin>>a[i][1]>>a[i][2]>>a[i][3]>>a[i][4];
    }
    for(int i=1;i<=n;i++)//输入点的位置
    {
        cin>>b[i][1]>>b[i][2];
    }
    int t=n;//一共应该有n个点，最多需要n遍
    int s=0;
    while(t!=0)//一共应该有n个点，最多需要n遍
    {
        for(int i=1;i<=n;i++)
        {
            if(y[i]!=0)continue;//如果这个点已经使用就跳过
            int num=0;
            int id;
```

```
            if(y[i]!=0)continue;//如果这个点已经使用就跳过
            int num=0;
            int id;
            for(int j=1;j<=n;j++)
            {
                if(x[j]!=0)continue;//如果这个幻灯片已经使用就跳过
                if(b[i][1]>=a[j][1]&&b[i][1]<=a[j][2]&&b[i][2]>=a[j][3]&&b[i][2]<=a[j][4])
                {//判断这个点是否在这个幻灯片里
                    num++;
                    id=j;//获取A到Z的编号
                }
            }
            if(num==1)//如果这个点刚好只与一个点匹配就取出
            {
                x[id]=i;//幻灯片与点之间的关系
                y[i]=id;//点与幻灯片之间的关系
                s++;//成功匹配的个数
            }
        }
        t--;
    }
    if(s!=n)cout<<"None"<<endl;//如果幻灯片与点数不一一匹配
    else
    for(int i=1;i<=n;i++)printf("%c %d\n",64+i,x[i]);//输出
    return 0;
    }

```

3268 [2013_p4]车站分级庄

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。

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。

3268 [2013_p4]车站分级

题目描述：

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

3268 [2013_p4]车站分级

输入描述：

每组输入数据的第一行包含2个正整数n，m，用一个空格隔开。

第i+1行（1≤i≤m）中，首先是一个正整数si（2≤si≤n），表示第i趟车次有si个停靠站；接下来有si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。

数据规模：

对于20%的数据，1≤n，m≤10；

对于50%的数据，1≤n，m≤100；

对于100%的数据，1≤n，m≤1000。

输出描述：

每组输出只有一行，包含一个正整数，即n个火车站最少划分的级别数。

3268 [2013_p4]车站分级

样例输入：

```
9 2
4 1 3 5 6
3 3 5 6
```

样例输出：

```
2
```

3268 [2013_p4]车站分级

分析：

可以把每个车站作为一个节点，根据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 的都必须停靠得出"没有停留的车站一定比停留的车站级别低"，那么假设没有停留的车站为级别1，这时候可以在没有停留的车站与停留的车站之间建立一条边，表示两者的从属关系。同时统计每个节点的入度，输入完成后入度为0的车站级别一定是最低的。然后从入度为0的车站开始拓扑排序，更新每个车站的级别。

分析：

构建车站等级的拓扑排序的有向图，然后靠逐层断边计算车站等级。

每趟车次从始发站到终点站，**停靠过的车站至少比没有停靠过的车站大一级别。**因此我们把没有停靠过的车站连一条有向边到停靠过的车站。可以想象为没有停靠过的从下面向上指向停靠过的，意为等级比上面的低。 那么**每有一层就代表有一个等级**，不断地断掉边，每断掉一层就ans加一。

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

3268 [2013_p4]车站分级

未命名1.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;
int sta[1010],mapn[1010][1010],in[1010];
bool vis[1010];
int main(){
    int t,n,m;
    queue<int>q;
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        memset(vis,0,sizeof(vis));
        cin>>t;
        for(int j=1;j<=t;j++){
            cin>>sta[j];
            vis[sta[j]]=true;
        }
        for(int j=sta[1];j<sta[t];j++){
            if(!vis[j]){
                vis[j]=true;
                for(int k=1;k<=t;k++){
                    if(mapn[j][sta[k]]==0){//把没有停下的车站连接到停下的
                        mapn[j][sta[k]]=1;
                        in[sta[k]]++;
                    }
                }
            }
        }
    }
    memset(vis,0,sizeof(vis));
    int ans=0;
    while(1){//只要还有数就一直循环
        for(int i=1;i<=n;i++){
            }
        }
    }
    memset(vis,0,sizeof(vis));
    int ans=0;
    while(1){//只要还有数就一直循环
        for(int i=1;i<=n;i++){
            if(in[i]==0&&!vis[i]){
                q.push(i);
                vis[i]=true;
            }
        }
        if(q.empty())break;//一旦都找完了，就跳出
        while(!q.empty()){
            for(int i=1;i<=n;i++){
                if(mapn[q.front()][i]==1){
                    mapn[q.front()][i]=0;
                    in[i]--;
                }
            }
            q.pop();//每找完一个就把找的弹出来
        }
        ans++;
    }
    cout<<ans;
    }
```

问题 J: 烦人的幻灯片

题目描述

李教授将于今天下午作一次非常重要的演讲。不幸的事他不是一个非常爱整洁的人，他把自己演讲要用的幻灯片随便堆在了一起。因此，演讲之前他不得不去整理这些幻灯片。作为一个讲求效率的学者，他希望尽可能简单地完成它。教授这次演讲一共要用n张幻灯片（n≤26），这n张幻灯片按照演讲要使用的顺序已经用数字1~n编了号。因为幻灯片是透明的，所以我们不能一下子看清每一个数字所对应的幻灯片。现在我们用大写字母A,B,C……再次把幻灯片依次编号。你的任务是编写一个程序，把幻灯片的数字编号和字母编号对应起来，显然这种对应应该是唯一的；若出现多种对应的情况或是某些数字编号和字母编号对应不起来，我们称对应是无法实现的。

问题 J: 烦人的幻灯片

输入描述： 第一行只有一个整数n，表示有n张幻灯片，接下来的n行每行包括4个整数xmin,xmax,ymin,ymax（整数之间用空格分开）为幻灯片的坐标，这n张幻灯片按其在文件中出现的顺序从前到后依次编号为A,B,C……，再接下来的n行依次为n个数字编号的坐标x,y，显然在幻灯片之外是不会有数字的。

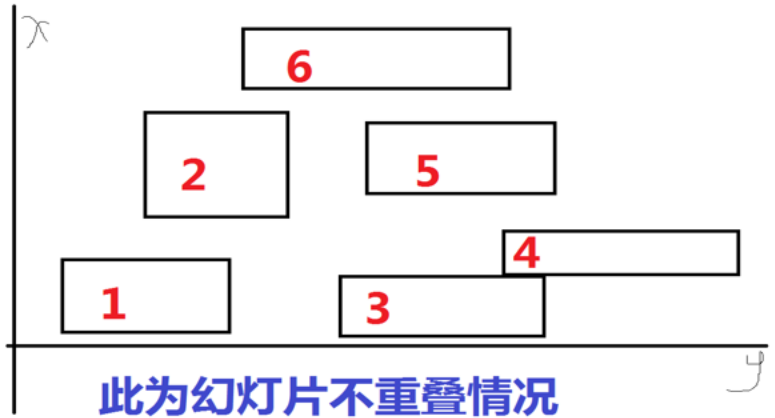
输出描述： 若是对应可以实现，输出文件应该包括n行，每一行为一个字母和一个数字，中间以一个空格隔开，并且每行以字母的升序排列，注意输出的字母要大写并且定格；反之，若是对应无法实现，在文件的第一行顶格输出None即可。首行末无多余的空格。

问题 J: 烦人的幻灯片

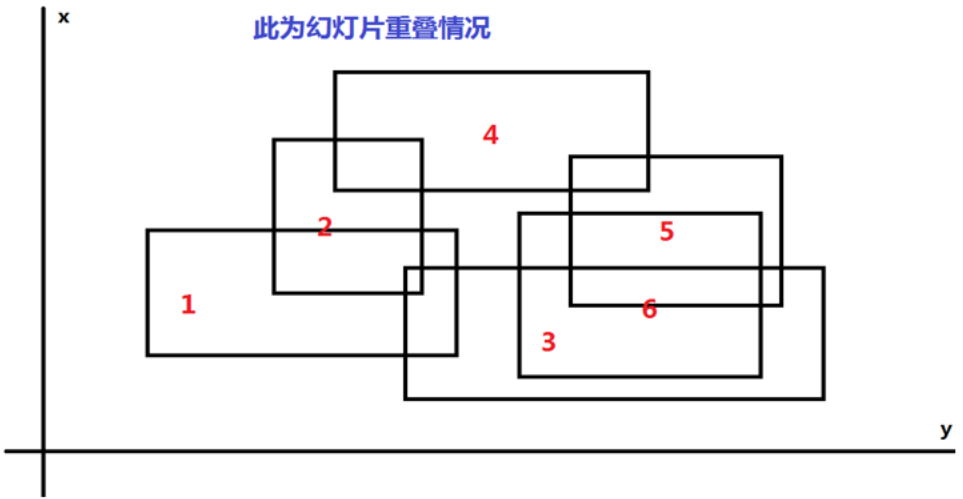
样例输入
4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11
样例输出
A 4
B 1
C 2
D 3

问题 J: 烦人的幻灯片

分析：本题适合用拓扑排序，每个点与多个幻灯片相交，通过拓扑排序，一步步取出相匹配的点与幻灯片。



问题 J: 烦人的幻灯片



问题 J: 烦人的幻灯片

```cpp
#include<bits/stdc++.h>
using namespace std;
int x[30];//幻灯片与点之间的关系
int y[30];//点与幻灯片之间的关系
int a[30][5];//储存幻灯片位置
int b[30][3];//储存点的位置
int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)//输入幻灯片位置
    {
        cin>>a[i][1]>>a[i][2]>>a[i][3]>>a[i][4];
    }
    for(int i=1;i<=n;i++)//输入点的位置
    {
        cin>>b[i][1]>>b[i][2];
    }
    int t=n;//一共应该有n个点，最多需要n遍
    int s=0;
    while(t!=0)//一共应该有n个点，最多需要n遍
    {
        for(int i=1;i<=n;i++)
        {
            if(y[i]!=0)continue;//如果这个点已经使用就跳过
            int num=0;
            int id;
```

问题 J: 烦人的幻灯片

```
25              if(y[i]!=0)continue;//如果这个点已经使用就跳过
26              int num=0;
27              int id;
28              for(int j=1;j<=n;j++)
29              {
30                  if(x[j]!=0)continue;//如果这个幻灯片已经使用就跳过
31                  if(b[i][1]>=a[j][1]&&b[i][1]<=a[j][2]&&b[i][2]>=a[j][3]&&b[i][2]<=a[j][4])
32                  {//判断这个点是否在这个幻灯片里
33                      num++;
34                      id=j;//获取A到z的编号
35                  }
36              }
37              if(num==1)//如果这个点刚好只与一个点匹配就取出
38              {
39                  x[id]=i;//幻灯片与点之间的关系
40                  y[i]=id;//点与幻灯片之间的关系
41                  s++;//成功匹配的个数
42              }
43          }
44          t--;
45      }
46      if(s!=n)cout<<"None"<<endl;//如果幻灯片与点数不一一匹配
47      else
48      for(int i=1;i<=n;i++)printf("%c %d\n",64+i,x[i]);//输出
49      return 0;
50      }
51
```

问题 z: 鱼塘钓鱼(fishing)

输入描述：

有N个鱼塘排成一排（N<100），每个鱼塘中有一定数量的鱼，例如：N=5时，如下表：
鱼塘编号每1分钟能钓到的鱼的数量（1..1000）每1分钟能钓鱼数的减少量（1..100）
当前鱼塘到下一个相邻鱼塘需要的时间（单位：分钟）110232144532064416545 93鱼塘编号12345每1分钟能钓到的鱼的数量（1..1000）101420169每1分钟能钓鱼数的减少量（1..100）24653当前鱼塘到下一个相邻鱼塘需要的时间（单位：分钟）3544
即：在第1个鱼塘中钓鱼第1分钟内可钓到10条鱼，第2分钟内只能钓到8条鱼，……，第5分钟以后再也钓不到鱼了。从第1个鱼塘到第2个鱼塘需要3分钟，从第2个鱼塘到第3个鱼塘需要5分钟，……
给出一个截止时间T(T<1000)，设计一个钓鱼方案，从第1个鱼塘出发，希望能钓到最多的鱼。

输出描述：

一个整数（不超过231−1231−1），表示你的方案能钓到的最多的鱼。

[问题G]最优贸易搜索实现

样例输入：

5
10 14 20 16 9　钓鱼数
2 4 6 5 3　减少鱼数
3 5 4 4　花的时间
14　时间

样例输出：76

[问题G]最优贸易搜索实现

分析：

一开始的想法是dp，然后看了下是优先队列的题，加上dp的复杂度粗略一算有o（N^3），所以pass，然后想到结构体（这里用了pair函数代替）和优先队列。

pair是将2个数据组合成一个数据，当需要这样的需求时就可以使用pair，如stl中的map就是将key和value放在一起来保存。另一个应用是，当一个函数需要返回2个数据的时候，可以选择pair。 pair的实现是一个结构体，主要的两个成员变量是first second因为是使用struct不是class，所以可以直接使用pair的成员变量。

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,a[105],b[105],c[105];
int t,lefttime,t1=0,sum=0,maxn=0;
int main(){
cin>>n;
for(int i=1;i<=n;i++)cin>>a[i];
for(int i=1;i<=n;i++)cin>>b[i];
for(int i=1;i<n;i++)cin>>c[i];
cin>>t;
for(int i=1;i<=n;i++){
    priority_queue<pair<int,int> >q;
    for(int j=1;j<=i;j++)q.push(make_pair(a[j],j));
    lefttime=t-t1,sum=0;
    while(lefttime>0&&q.top().first>0){
        pair<int,int>u=q.top();
        q.pop();
        sum+=u.first;
        u.first-=b[u.second];
        q.push(u);
        lefttime--;
    }
    maxn=max(maxn,sum);
    t1=t1+c[i];
}
cout<<maxn<<endl;
}
```

在1-j号鱼塘钓鱼，减去路途所花时间。每分钟选择最大的池塘钓鱼就行了。

问题 J: 关押罪犯

题目描述

S 城现有两座监狱，一共关押着N 名罪犯，编号分别为1~N。他们之间的关系自然也极不和谐。很多罪犯之间甚至积怨已久，如果客观条件具备则随时可能爆发冲突。我们用“怨气值”（一个正整 数值）来表示某两名罪犯之间的仇恨程度，怨气值越大，则这两名罪犯之间的积怨越多。如果两名怨气值为c 的罪犯被关押在同一监狱，他们俩之间会发生摩擦，并造成影响力为c 的冲突事件。

每年年末，警察局会将本年内监狱中的所有冲突事件按影响力从大到小排成一个列表，然后上报到S 城Z 市长那里。公务繁忙的Z 市长只会去看列表中的第一个事件的影响力，如果影响很坏，他就会考虑撤换警察局长。

在 详细考察了N 名罪犯间的矛盾关系后，警察局长觉得压力巨大。他准备将罪犯们在两座监狱内重新分配，以求产生的冲突事件影响力都较小，从而保住自己的乌纱帽。假设只要处 于同一监狱内的某两个罪犯间有仇恨，那么他们一定会在每年的某个时候发生摩擦。那么，应如何分配罪犯，才能使Z 市长看到的那个冲突事件的影响力最小？这个最小值是多少？

问题 J: 关押罪犯

输入描述： 每行中两个数之间用一个空格隔开。
第一行为两个正整数N 和M，分别表示罪犯的数目以及存在仇恨的罪犯对数。
接下来的M 行每行为三个正整数aj，bj，cj，表示aj 号和bj 号罪犯之间存在仇恨，其怨
气值为cj。数据保证1<aj=<=bj<=N ，0 < cj≤ 1,000,000,000，且每对罪犯组合只出现一次。

输出描述：

共1 行，为Z 市长看到的那个冲突事件的影响力。如果本年内监狱中未发生任何冲突事件，请输出0。

问题 J: 关押罪犯

样例输入

```
4 6
1 4 2534
2 3 3512
1 2 28351
1 3 6618
2 4 1805
3 4 12884
```

样例输出

```
3512
```

问题 J: 关押罪犯

分析：本题适合用并查集来解决，因为一共只有两个监狱，我们之需要把所有数据排序然后从大到小把犯人放入两个监狱，因为只有尽可能把影响力大的两个罪犯拆开，我们才能得到最小值。为了便于规定两人是否处与同一监狱，我们不妨把节点数扩大一倍，也就相当于人为加了一倍的犯人，i与i+n是处于两个监狱的人，这是我们规定的，接下来就运用常规的并查集算法就可以解决。

问题 J: 关押罪犯

```cpp
#include<bits/stdc++.h>
using namespace std;
struct node{
    int x;
    int y;
    int s;
}a[101000];
int f[50000];
int find(int x)
{
    if(f[x]!=x)
    f[x]=find(f[x]);
    return f[x];
}
int cmp(node a,node b)
{
    return a.s>b.s;
}
int main()
{
    int m,n;
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        cin>>a[i].x>>a[i].y>>a[i].s;
    }
    for(int i=1;i<=50000;i++)
    {
        f[i]=i;
    }
    sort(a+1,a+1+m,cmp);
    int flag=0;
    for(int i=1;i<=m;i++)
    {
        int f1=find(a[i].x);
        int f2=find(a[i].y);
        if(f1==f2)
        {
            printf("%d",a[i].s);
            return 0;
        }
        f[f1]=find(a[i].y+n);
        f[f2]=find(a[i].x+n);
    }
```

问题 J: 关押罪犯

```
25          cin>>a[i].x>>a[i].y>>a[i].s;
26      }
27      for(int i=1;i<=50000;i++)
28      {
29          f[i]=i;
30      }//初始化并查集
31      sort(a+1,a+1+m,cmp);//排序
32      int flag=0;
33      for(int i=1;i<=m;i++)
34      {
35          int f1=find(a[i].x);
36          int f2=find(a[i].y);
37          if(f1==f2)//如果迫不得已两个人要在同一个监狱，那么这就是最小的答案
38          {
39              printf("%d",a[i].s);
40              return 0;
41          }
42          f[f1]=find(a[i].y+n);// a[i].x与a[i].y是处与两个监狱 a[i].x与a[i].x+n处与两个监狱
43          f[f2]=find(a[i].x+n);//  一共只有两个监狱，所以a[i].y与a[i].x+n处与一个监狱
44      }
45      printf("0");
46      return 0;
47  }
```

[问题G]最优贸易

输入描述：

每组输入数据的第一行包含2个正整数n和m，中间用一个空格隔开，分别表示城市的数目和道路的数目。
第二行n个正整数，每两个正整数之间用一个空格隔开，按标号顺序分别表示这n个城市的商品价格。
接下来m行，每行有3个正整数，x，y，z，每两个整数之间用一个空格隔开。如果z=1，表示这条道路是城市x到城市y之间的单向道路；如果z=2，表示这条道路为城市x和城市y之间的双向道路。

输出描述：

每组输出共1行，包含1个整数，表示最多能赚取的旅费。如果没有进行贸易，则输出0。

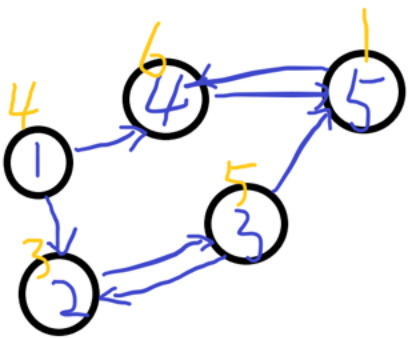
[问题G]最优贸易搜索实现

样例输入：

5 5
4 3 5 6 1 商品价格
1 2 1
1 4 1
2 3 2
3 5 1
4 5 2

样例输出：5

蓝字序号，黄字价格



[问题G]最优贸易搜索实现

分析：

在程序中可以使用一个队列存储表示邻接表。用搜索暴力找出最大价格

本题中的图可以是双向图，搜索途中要判断是否已经用过该城市

[问题G]最优贸易搜索实现

```
#include<bits/stdc++.h>
#include<iostream>
#define M 100005
#define inf 1e7+9
using namespace std;

int n,k,x,y,z;
int w[M],m[M],c[M];//w是各城市的价格，m记录从起点到当前点最小价格，c记录从起点到当前点的最大利润
vector<int> g[M];
void dfs(int x,int minn,int pre)//x当前点，minn当前最小价格，pre前一点
{
        int flag=1;//判断该点有没用过，不然可能会死循环
        if(minn>w[x])
                minn=w[x];
        if(m[x]>minn){
                m[x]=minn;
                flag=0;
        }
        int maxn=max(c[pre],w[x]-minn);
        if(maxn>c[x]){
                c[x]=maxn;
                flag=0;
        }
        if(flag==1)return;
        for(int i=0;i<g[x].size();i++)//去下一个城市
        dfs(g[x][i],minn,x);
};

int main(){
cin>>n>>k;
for(int i=1;i<=n;i++)
cin>>w[i];
for(int i=0;i<=M;i++)//设置初始价格最大
m[i]=inf;
for(int i=1;i<=k;i++)//建表
{
        cin>>x>>y>>z;
        g[x].push_back(y);
        if(z==2)
        g[y].push_back(x);
}
dfs(1,inf,0);
cout<<c[n];
```

搜索的局限性

- 尽管搜索理解简单和空间复杂度较低。但容易超时，要注意剪枝。