

算法设计与分析

Computer Algorithm Design & Analysis

文明

mwenaa@hust.edu.cn



第15讲 All-Pairs Shortest Paths

最短路径例题

想去学校

小明很热爱学习，他打算偷偷跑回学校学习，为了多学习他希望可以找最快的路线回到学校。北京市里有 N 个($2 \leq N \leq 1000$)个地铁站，编号分别为 $1 \dots N$ 。小明家在1号地铁站旁边，清华大学旁边是 N 号地铁站。地铁站之间共有 M ($1 \leq M \leq 2000$)条双向路径。

小明现在在1号地铁站，他希望知道到学校最短要多长时间。可以保证他能到达学校。忽略换乘地铁时需要的等待时间

输入格式

第一行:两个整数: M 和 N

接下来 M 行:每一行有 A B C 三个整数。代表 A 站到 B 站或者 B 站到 A 站需要 C 分钟， C 的范围为1到100。

输出格式

一个整数，表示从1号站回到学校的最少时间。

输入输出样例

输入

```
5 5
1 2 20
2 3 30
3 4 20
4 5 20
1 5 100
```

输出

```
90
```

```

1 #include <iostream>
2 #include <cstdio>
3 #include <algorithm>
4 #include <cmath>
5 #include <string>
6 #include <cstring>
7 #include <map>
8 #include <queue>
9 using namespace std;
10 typedef long long LL;
11 const int N = 1005;
12 int n, m, g[N][N], dist[N], st[N]; //g数组是记录从某点到某点的权值, dist数组表示任意点到点1的最短距离, st数组是标记已求出
13 int read(){
14     int x, f = 1;
15     char ch;
16     while(ch = getchar(), ch < '0' || ch > '9') if(ch == '-') f = -1;
17     x = ch - '0';
18     while(ch = getchar(), ch >= '0' && ch <= '9') x = x * 10 + ch - 48;
19     return x * f;
20 }
21 int dijkstra(){
22     int i, j, t;
23     memset(dist, 0x3f, sizeof(dist)); //把所有点到起点最短距离初始化为正无穷
24     dist[1] = 0; //起点到起点的最短距离为0
25     for(i = 1; i <= n; i++){ //每轮循环得到一个确定最短路径的顶点
26         t = -1; //t是个临时变量, 为了确定某个点到起点的最短距离, 并且将其放入已求出最短路径的顶点的集合里面
27         for(j = 1; j <= n; j++){
28             if(!st[j] && (t == -1 || dist[t] > dist[j])) t = j;
29         }
30         st[t] = 1; //标记t点已经被放入已求出最短路径的顶点的集合里面
31         for(j = 1; j <= n; j++){
32             dist[j] = min(dist[j], dist[t] + g[t][j]); //更新未确定最短路径的顶点 目前已得到的最短路径
33         }
34     }
35     if(dist[n] == 0x3f3f3f3f) return -1; //此点跟起点不是连通的
36     return dist[n]; //返回点n到起点的最短路径
37 }
38 int main(){
39     int a, b, c;
40     m = read();
41     n = read();
42     memset(g, 0x3f, sizeof(g)); //把各个点之间的距离都初始化为正无穷
43     while(m--){
44         a = read();
45         b = read();
46         c = read();
47         g[a][b] = g[b][a] = min(g[a][b], c); //因为这是无向图, 所以从a到b和从b到a的距离相同, 两点的距离可能会被读入不同
48     }
49     printf("%d\n", dijkstra());
50     return 0;
51 }

```

```

1  #include<iostream>
2  #include<algorithm>
3  #include<string>
4  #include<cstdio>
5  #include<cstring>
6  #include<queue>
7  #include<vector>
8
9  using namespace std;
10
11  const int INF=0x3f3f3f3f;
12  const int maxn=1005;
13  int maps[maxn][maxn];//存两点间的距离
14  int dis[maxn];//表示源点到该点的最短路
15  int n;//节点数
16
17  struct node
18  {
19      int num;
20      int dis;
21      friend bool operator <(node a,node b)
22      {
23          if(a.dis==b.dis) return a.num>b.num;
24          return a.dis>b.dis;
25      }
26  };
27  priority_queue<node> que;//用优先队列优化
28
29  void init()
30  {
31      for(int i=1;i<=n;i++)
32          for(int j=1;j<=n;j++)
33              maps[i][j]=INF;
34  }

```

```

35  void Dijkstra(int s)
36  {
37      int vis[maxn];//表示该点是否已经访问过
38      for(int i=1;i<=n;i++)
39          dis[i]=INF;
40      memset(vis,0,sizeof(vis));
41
42      dis[s]=0;
43      node u;
44      u.dis=0;
45      u.num=s;
46      que.push(u);
47      while(!que.empty())
48      {
49          node u=que.top();
50          que.pop();
51          int x=u.num;
52
53          if(vis[x])
54              continue;
55          vis[x]=1;
56
57          for(int i=1;i<=n;i++)
58          {
59              if(!vis[i]&&dis[i]>dis[u.num]+maps[u.num][i])
60              {
61                  dis[i]=dis[u.num]+maps[u.num][i];
62                  node v;
63                  v.num=i;
64                  v.dis=dis[i];
65                  que.push(v);
66              }
67          }
68      }
69      printf("%d\n",dis[n]);
70  }
71

```

命令传递

由于叛徒朱子明的出卖，导致独立团在赵家峪的团部驻军在团长李云龙大婚之日几乎全军覆没，突出重围之后，李云龙决定集合所有驻扎在外的部队，使用重型武器意大利炮攻打平安县城，消息从团部传出之后到达各部驻地之后，驻地长官会派出自己的通讯人员通知其他驻地部队，自团部派人传达命令开始，至少经过多长时间才能使得所有驻扎在外的部队受到命令。（假设通讯员在路上不会遭遇任何意外）

Input

第一行输入一个 n ，表示包括团部在内有多少不同的驻军（团部当然是编号为1了）。

随后 $n-1$ 行，以邻接矩阵的形式给出各驻军（ $1 \sim n$ ）之间派遣通讯员需要的时间。

由于两地驻军派遣通讯员所需时间相等（从一营三连驻地到二营一连驻地和二营一连驻地到一营三连驻地所需时间是一样的），且驻地自己和自己不需要派遣通讯员，所以只给出了矩阵的下三角。

x 表示由于部分驻地之间因特殊原因不能派遣通讯员，比如途中需要经过敌占区

该题所有数据范围 $0 \sim 100$ 。

Output

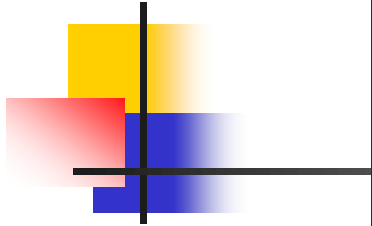
输出一行表示所有驻地都受到来自团部的命令所需要的时间。

Sample Input

```
5
50
30 5
100 20 50
10 x x 10
```

Sample Output

```
35
```



```
1  #include<stdio.h>
2  #include<algorithm>
3  #include<iostream>
4  #include<string.h>
5  #include<ctype.h>
6  #include<queue>
7  #include<set>
8  #include<stack>
9  #include<cmath>
10 const int maxn=99999999;
11 using namespace std;
12 int a[110][110];
13 char s[20];
14 int main()
15 {
16     int n,i,j,k;
17     while(~scanf("%d",&n)&&n)
18     {
19         for(i=1;i<=n;i++)
20             for(j=1;j<=n;j++)
21                 if(i==j) a[i][j]=0;
22                 else a[i][j]=maxn;
23         for(i=2;i<=n;i++)
24             for(j=1;j<=i-1;j++)
25             {
26                 scanf("%s",s);
27                 if(s[0]=='x') continue;
28                 else
29                 {
30                     a[i][j]=atoi(s);
31                     a[j][i]=a[i][j];
32                 }
33             }
34         for(k=1;k<=n;k++)
35             for(i=1;i<=n;i++)
36                 for(j=1;j<=n;j++)
37                     a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
38         int res=a[1][2];
39         for(i=3;i<=n;i++)
40             if(a[1][i]!=maxn)
41                 res=max(res,a[1][i]);
42         printf("%d\n",res);
43     }
44     return 0;
45 }
```

有轨电车

萨格勒布的电车网络由许多交叉路口和铁路连接而成。在每个交叉路口都有一个开关，指向一个从交叉路口出来的铁轨。有轨电车进入交叉路口时，只能沿开关指向的方向离开。如果驾驶员要走其他路线，则必须手动更改开关。

当驾驶员确实从交叉路口A到交叉路口B行驶时，他/她将尝试选择路线，该路线将最大限度地减少他/她将不得不手动更改开关的次数。

编写一个程序，计算从路口A到路口B所需的最少开关变化次数。

Input

输入的第一行包含整数N，A和B，以单个空白字符分隔， $2 \leq N \leq 100$ ， $1 \leq A, B \leq N$ ，N是网络中的交点数，并且交叉点从1到N编号。

接下来的N行中的每行均包含由单个空白字符分隔的整数序列。第i行中的第一个数字 K_i ($0 \leq K_i \leq N-1$) 表示从第i个交叉点出来的轨道数。下一个 K_i 数表示直接连接到第i个交点的交点。第i个交点的开关最初指向列出的第一个交点的方向。

Output

输出的第一行和唯一一行应包含目标最小值。如果没有从A到B的路线，则该行应包含整数“-1”。

Sample Input

```
3 2 1
2 2 3
2 3 1
2 1 2
```

Sample Output

```
1
```


POJ 184

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<algorithm>
4  #include<iostream>
5  #include<string.h>
6  #include<ctype.h>
7  #include<queue>
8  #include<set>
9  #include<stack>
10 #include<cmath>
11 const int inf=999999999;
12 using namespace std;
13 int dis[101][101];
14 int main()
15 {
16     ios::sync_with_stdio(false);
17     cin.tie(0);cout.tie(0);
18     int k,i,j,n,m,a,b,c;
19     cin>>n>>a>>b;
20     for(i=1;i<=n;i++)
21         for(j=1;j<=n;j++)
22             if(i==j) dis[i][j]=0;
23             else dis[i][j]=inf;
24     for(i=1;i<=n;i++)
25     {
26         cin>>m>>c;
27         dis[i][c]=0;
28         for(j=2;j<=m;j++)
29         {
30             cin>>c;
31             dis[i][c]=1;
32         }
33     }
34     for(k=1;k<=n;k++)
35         for(i=1;i<=n;i++)
36             for(j=1;j<=n;j++)
37                 dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
38     if(dis[a][b]==inf)
39         cout<<-1<<endl;
40     else
41         cout<<dis[a][b]<<endl;
42     return 0;
43 }
```

2021/11/17

```

// C Program for Floyd Warshall Algorithm
#include<stdio.h>

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
value. This value will be used
for vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall (int graph[][V])
{
    /* dist[][] will be the output matrix
    that will finally have the shortest
    distances between every pair of vertices */
    int dist[V][V], i, j, k;

    /* Initialize the solution matrix
    same as input graph matrix. Or
    we can say the initial values of
    shortest distances are based
    on shortest paths considering no
    intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    /* Add all vertices one by one to
    the set of intermediate vertices.
    ---> Before start of an iteration, we
    have shortest distances between all
    pairs of vertices such that the shortest
    distances consider only the
    vertices in set {0, 1, 2, .. k-1} as
    intermediate vertices.
    ----> After the end of an iteration,
    vertex no. k is added to the set of
    intermediate vertices and the set
    becomes {0, 1, 2, .. k} */

```

```

        for (k = 0; k < V; k++)
        {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++)
            {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++)
                {
                    // If vertex k is on the shortest path from
                    // i to j, then update the value of dist[i][j]
                    if (dist[i][k] + dist[k][j] < dist[i][j])
                        dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }

        // Print the shortest distance matrix
        printSolution(dist);
    }

    /* A utility function to print solution */
    void printSolution(int dist[][V])
    {
        printf ("The following matrix shows the shortest distances"
                " between every pair of vertices \n");
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                if (dist[i][j] == INF)
                    printf ("%7s", "INF");
                else
                    printf ("%7d", dist[i][j]);
            }
            printf ("\n");
        }

        // driver program to test above function
        int main()
        {
            /* Let us create the following weighted graph
            10
            (0)----->(3)
            |             /\
            5 |           | 1
            |             |
            \|/          |
            (1)----->(2)
                3
            */
            int graph[V][V] = { {0, 5, INF, 10},
                                {INF, 0, 3, INF},
                                {INF, INF, 0, 1},
                                {INF, INF, INF, 0}
                                };

            // Print the solution
            floydWarshall(graph);
            return 0;
        }
    }

```

```

#include <limits.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 9

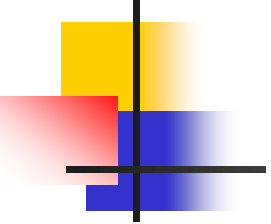
// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
int printSolution(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d tt %d\n", i, dist[i]);
}

```



```

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array. dist[i] will hold the shortest
    // distance from src to i

    bool sptSet[V]; // sptSet[i] will be true if vertex i is included in shortest
    // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum distance vertex from the set of vertices not
        // yet processed. u is always equal to src in the first iteration.
        int u = minDistance(dist, sptSet);

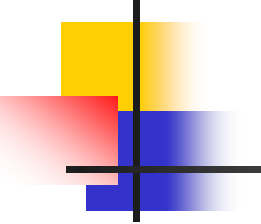
        // Mark the picked vertex as processed
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet, there is an edge from
            // u to v, and total weight of path from src to v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist, V);
}

```



```
// driver program to test above function
int main()
{
    /* Let us create the example graph discussed above */
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

奶牛Party

来自 N ($1 \leq N \leq 1000$) 个农场的奶牛的编号分别为 $1, 2, \dots, N$ 。现在在农场 X ($1 \leq X \leq N$) 举行聚会。总共有 M ($1 \leq M \leq 100,000$) 条单向通道。路 i 需要时间 T_i 才能通过。每头牛都需要参加聚会并返回，而且它们均选择花费时间最短的路线。

问：在所有的奶牛中，所花费的最长时间为多少？

Input

第一行包含三个整数 N , M , X 。

在接下来的 M 行中，每行都包括三个整数 A , B 和 T ，表示从农场 A 到 B 需要花费时间 T 。

Output

输出奶牛所花的最大时间。

Sample Input

```
4 8 2
1 2 4
1 3 2
1 4 7
2 1 1
2 3 5
3 1 2
3 4 4
4 2 3
```

Sample Output

```
10
```

```

1 #include <iostream>
2 #include <cstdio>
3 using namespace std;
4
5 const int INF = 10000000;
6 int dis1[1005], dis2[1005];           //返回最短路径, 过去的最短路径
7 int cost[1005][1005];
8 bool vis[1005];                       //标记是否访问过
9 int N, X, M;
10
11 int min(int x, int y)
12 {
13     return x > y ? y : x;
14 }
15
16 void dijkstra(int s, int dis[])         //dijkstra算法求解单源最短路径
17 {
18     for(int i = 0; i <= N; i++)
19     {
20         vis[i] = false;
21         dis[i] = INF;
22     }
23     dis[s] = 0;
24     while(true)
25     {
26         int v = -1;
27         for(int u = 1; u <= N; u++)     //从没有选过的顶点中
28         {
29             if(!vis[u] && (v < 0 || dis[v] > dis[u]))
30                 v = u;
31         }
32         if(v == -1)
33             break;
34         vis[v] = true;
35         for(int j = 1; j <= N; j++)
36         {
37             dis[j] = min(dis[j], dis[v] + cost[v][j]);
38         }
39     }
40 }
41
42 int main()
43 {
44     freopen("data.txt", "r", stdin);
45     int i, j, a, b, c;
46     while(scanf("%d%d%d", &N, &M, &X) != EOF)
47     {
48         for(i = 1; i <= N; i++)         //初始化各边权值的数组,
49         for(j = 1; j <= N; j++)
50         {
51             if(i == j)
52                 cost[i][j] = 0;
53             else
54                 cost[i][j] = INF;
55         }
56         for(i = 0; i < M; i++)
57         {
58             scanf("%d%d%d", &a, &b, &c);   //输入各边权值
59             cost[a][b] = c;
60         }
61         dijkstra(X, dis1);               //算返回最短路径
62         for(i = 1; i <= N; i++)         //调转各边, 也就是将矩阵转置
63         {
64             for(j = 1; j < i; j++)
65             {
66                 int temp = cost[i][j];
67                 cost[i][j] = cost[j][i];
68                 cost[j][i] = temp;
69             }
70         }
71         dijkstra(X, dis2);               //算出发最短路径
72         int max = -100000;
73         for(i = 1; i <= N; i++)         //枚举求最大的最短路径和
74         {
75             if(i != X)
76             {
77                 int temp = dis1[i] + dis2[i];
78                 if(max < temp)
79                     max = temp;
80             }
81         }
82         printf("%d\n", max);

```

虫洞

当走访自己的农场时，农夫 John（下称 FJ）发现了许多神奇的虫洞。虫洞是一个很奇怪的东西，因为它是一个能够将你送到一个时间位于你进入虫洞之前的目的地的单向路径。FJ 的每个农场包含 N 块田（ $1 \leq N \leq 500$ ），编号从 1 到 N，M 条路（ $1 \leq M \leq 2500$ ），W 个虫洞（ $1 \leq W \leq 200$ ）。

FJ 是一个狂热的时间旅行爱好者，他希望做这样的事：从某块田开始，经过一些路径和虫洞，最后回到起始位置且到达时的时间在他出发之前。他有可能与他自己碰面？。

为了帮助 FJ 求出这是否可能成功，他会告诉你他的农场的 F 个地图（ $1 \leq F \leq 5$ ）。没有路径花费超过 10 000 秒才能走完，也没有虫洞能够将 FJ 带回早于进入虫洞的时刻的 10 000 秒之前。

虫洞

Input

第一行：一个整数 F ，描述如上。

接下来对每个农场，第一行三个整数，空格分隔： N 、 M 、 W 。

接下来 M 行，每行三个整数，空格分隔： S 、 E 、 T ，分别描述：存在点 S 到点 E 的双向路径，单程花费 T 秒。两块田之前可以被超过一条路连接。

接下来 W 行，每行三个整数，空格分隔： S 、 E 、 T ，分别描述：存在点 S 到点 E 的单向路径，能够令进入者回到 T 秒之前。

Output

1 到 F 行：对每个农场，如果 FJ 可以达成他的目标，输出

```
1 | YES
```

否则输出

```
1 | NO
```

Sample Input

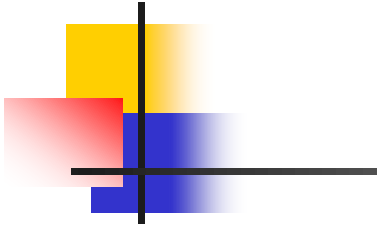
```
1 | 2
2 | 3 3 1
3 | 1 2 2
4 | 1 3 4
5 | 2 3 1
6 | 3 1 3
7 | 3 2 1
8 | 1 2 3
9 | 2 3 4
10 | 3 1 8
11 |
```

Sample Output

```
1 | NO
2 | YES
3 |
```

2021/11/17

虫洞



```
1  #include<cstdio>
2  #include<algorithm>
3  #pragma warning(disable:4996)
4  using namespace std;
5  struct edge { int u, v, t; };
6  int F, M, N, W, d[501], p, t; edge e[5201];
7  inline bool HasNegLoop() {
8      int T = 1;
9      for (;;) {
10         for (int i = 1; i <= t; ++i) {
11             if (d[e[i].v] > d[e[i].u] + e[i].t)d[e[i].v] = d[e[i].u] + e[i].t;
12         }
13         ++T; if (T == N)break;
14     }
15     for (int i = 1; i <= t; ++i)if (d[e[i].v] > d[e[i].u] + e[i].t)return true;
16     return false;
17 }
18 int main() {
19     scanf("%d", &F); ++F;
20     while (--F) {
21         scanf("%d%d%d", &N, &M, &W); fill(d + 2, d + N + 1, 200000000); d[1] = 0;
22         t = 2 * M;
23         for (p = 1; p <= t; ++p) {
24             scanf("%d%d%d", &e[p].u, &e[p].v, &e[p].t);
25             ++p; e[p].u = e[p - 1].v, e[p].v = e[p - 1].u, e[p].t = e[p - 1].t;
26         }
27         t += W;
28         for (; p <= t; ++p) {
29             scanf("%d%d%d", &e[p].u, &e[p].v, &e[p].t); e[p].t = -e[p].t;
30         }
31         switch (HasNegLoop()) {
32             case false:puts("NO"); continue;
33             default:puts("YES");
34         }
35     }
36     return 0;
37 }
```

奶牛比赛

FJ的 N ($1 \leq N \leq 100$)头奶牛们最近参加了场程序设计竞赛:)。在赛场上，奶牛们按 $1 \dots N$ 依次编号。每头奶牛的编程能力不尽相同，并且没有哪两头奶牛的水平不相上下，也就是说，奶牛们的编程能力有明确的排名。整个比赛被分成了若干轮，每一轮是两头指定编号的奶牛的对决。如果编号为 A 的奶牛的编程能力强于编号为 B 的奶牛 ($1 \leq A \leq N; 1 \leq B \leq N; A \neq B$)，那么她们的对决中，编号为 A 的奶牛总是能胜出。FJ想知道奶牛们编程能力的具体排名，于是他找来了奶牛们所有 M ($1 \leq M \leq 4,500$)轮比赛的结果，希望你能根据这些信息，推断出尽可能多的奶牛的编程能力排名。比赛结果保证不会自相矛盾。

Input

第1行: 2个用空格隔开的整数: N 和 M
第2... $M+1$ 行: 每行为2个用空格隔开的整数 A 、 B ，描述了参加某一轮比赛的奶牛的编号，以及结果（编号为 A ，即为每行的第一个数的奶牛为胜者）

Output

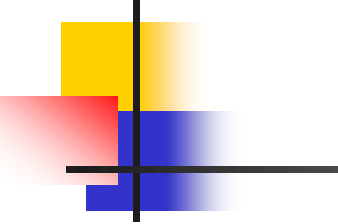
第1行: 输出1个整数，表示排名可以确定的奶牛的数目

Sample Input

```
5 5
4 3
4 2
3 2
1 2
2 5
```

Sample Output

```
2
```



```

1  #include<iostream>
2  #include<cstring>
3  #include<algorithm>
4  #include<cmath>
5  #include<cstdio>
6  #include<queue>
7  #include<vector>
8  typedef long long ll;
9  #define pii pair<int,int>
10 using namespace std;
11 const int N=10010,inf=0x3f3f3f3f;
12 int n,m,d[N][N];
13 int sum[N],suf[N];//sum[i]比i大的数量, suf[i]比i小的数量
14 int main()
15 {
16     cin>>n>>m;
17     while(m-->0)
18     {
19         int a,b;
20         cin>>a>>b;
21         d[a][b]=1;
22     }
23     for(int k=1;k<=n;k++)
24         for(int i=1;i<=n;i++)
25             for(int j=1;j<=n;j++)
26                 if(d[i][k]&& d[k][j])
27                     d[i][j]=1;
28     int ans=0;
29     for(int i=1;i<=n;i++)
30     {
31         for(int j=1;j<=n;j++)
32         {
33             sum[i]+=d[j][i];
34             suf[i]+=d[i][j];
35         }
36         if(sum[i]+suf[i]==n-1) ans++;
37     }
38     cout<<ans;
39     return 0;
40 }

```