匹配：在图G中两两没有公共端点的边集合M

二分图最大匹配：给出一个**二分图**，找出一个边数最大的匹配

匈牙利算法：利用增广路径求二分图最大匹配

设M是G的一个匹配

M-交错路，p是G的一条路径，如果p中的边为属于和不属于M的边交替出现，则称p是一条M-交错路。

M-可增广路，p是一条M-交错路，如果p的起点和终点都不在M的边中，则p为M-可增广路。

算法：

1. 置M为空
2. 找出一条增广路径p，通过异或操作获得更大的匹配M‘
3. 重复2直到找不出增广路径为止

#include<iostream>

#include<cstring>

using namespace std;

int map[105][105];

int visit[105],flag[105];

int n,m;

bool dfs(int a)

{

    for(int i=1; i<=n; i++)

    {

        if(map[a][i]&&!visit[i])

        {

            visit[i]=1;

            if(flag[i]==0||dfs(flag[i]))

            {

                flag[i]=a;

                return true;

            }

        }

    }

    return false;

}

int main()

{

    int n1,n2;

    while(cin>>n1 >>n2 >>m)

    {

        n=n1;

        memset(map,0,sizeof(map));

        for(int i=1; i<=m; i++)

        {

            int x,y;

            cin>>x>>y;

            map[x][y]=1;

        }

        memset(flag,0,sizeof(flag));

        int result=0;

        for(int i=1; i<=n1; i++)

        {

            memset(visit,0,sizeof(visit));

            if(dfs(i))result++;

        }

        cout<<result<<endl;

    }

    return 0;

}

完全二分图：第一个集合的所有顶点和第二个集合的所有顶点相连，必存在完备匹配

完美匹配：包含了所有顶点的匹配

完备匹配：匹配数为min(|X|,|Y|)

算法：

X集和Y集的每个顶点各有一个标杆值

1. 初始情况下，Y集中的顶点标杆值均为0，X集中的顶点标杆值为顶点相连的边集的最大权重。
2. 利用匈牙利算法找相等子图的完备匹配
3. 若未找到增广路径则修改可行顶标的值
4. 重复2，3直到找到相等子图的完备匹配

修改顶标的原则：对正在增广路径上属于集合X的所有点减去常数Delta，属于集合Y的所有点加上常数Delta。

Delta = min(x + y - weight( x, y)) x是属于增广路径的顶点，y是不属于增广路的顶点。

#include <cstring>

#include <cstdio>

const int maxn = 305;

const int INF = 0x3f3f3f3f;

int match[maxn],lx[maxn],ly[maxn],slack[maxn];

int G[maxn][maxn];

bool visx[maxn],visy[maxn];

int n,nx,ny,ans;

bool findpath(int x)

{

int tempDelta;

visx[x] = true;

for(int y = 0 ; y < ny ; ++y){

if(visy[y]) continue;

tempDelta = lx[x] + ly[y] - G[x][y];

if(tempDelta == 0){//(x,y)在相等子图中

visy[y] = true;

if(match[y] == -1 || findpath(match[y])){

match[y] = x;

return true;

}

}

else if(slack[y] > tempDelta)

slack[y] = tempDelta;//(x,y)不在相等子图中且y不在交错树中

}

return false;

}

void KM()

{

for(int x = 0 ; x < nx ; ++x){

for(int j = 0 ; j < ny ; ++j) slack[j] = INF;//这里不要忘了，每次换新的x结点都要初始化slack

while(true){

memset(visx,false,sizeof(visx));

memset(visy,false,sizeof(visy));//这两个初始化必须放在这里,因此每次findpath()都要更新

if(findpath(x)) break;

else{

int delta = INF;

for(int j = 0 ; j < ny ; ++j)//因为dfs(x)失败了所以x一定在交错树中，y不在交错树中，第二类边

if(!visy[j] && delta > slack[j])

delta = slack[j];

for(int i = 0 ; i < nx ; ++i)

if(visx[i]) lx[i] -= delta;

for(int j = 0 ; j < ny ; ++j){

if(visy[j])

ly[j] += delta;

else

slack[j] -= delta;

//修改顶标后，要把所有的slack值都减去delta

//这是因为lx[i] 减小了delta

//slack[j] = min(lx[i] + ly[j] -w[i][j]) --j不属于交错树--也需要减少delta，第二类边

}

}

}

}

}

void solve()

{

memset(match,-1,sizeof(match));

memset(ly,0,sizeof(ly));

for(int i = 0 ; i < nx ; ++i){

lx[i] = -INF;

for(int j = 0 ; j < ny ; ++j)

if(lx[i] < G[i][j])

lx[i] = G[i][j];

}

KM();

}

int main()

{

while(scanf("%d",&n) != EOF){

nx = ny = n;

for(int i = 0 ; i < nx ; ++i)

for(int j = 0 ; j < ny ; ++j)

scanf("%d",&G[i][j]);

solve();

}

return 0;

}