二分图匹配

二分答案，二分查找

此“二分”不同于彼“二分”

田忌赛马

上次赛马输给了田忌，齐威王很不服气，于是他再次约田忌赛马。这次齐威王带来了M匹马，田忌有N匹马。聪明的田忌暗中搜集了齐威王赛马的信息，他事先知道了自己的每一匹马能够赢齐威王的哪些马。每匹马只能参赛一次，问田忌要怎样安排比赛才能让自己赢的场次尽可能多？

输入格式：

第一行 两个整数，N (0 <= N <= 200) 和 M (0 <= M <= 200) 。（田忌的马编号1到N，齐威王的马编号1到M）

接下来N 行，每行对应田忌的一匹马。第一个数字 K表示这匹马能赢齐威王K匹马(0 <= K<= M) 。后面的 K个数表示齐威王这些马的编号。

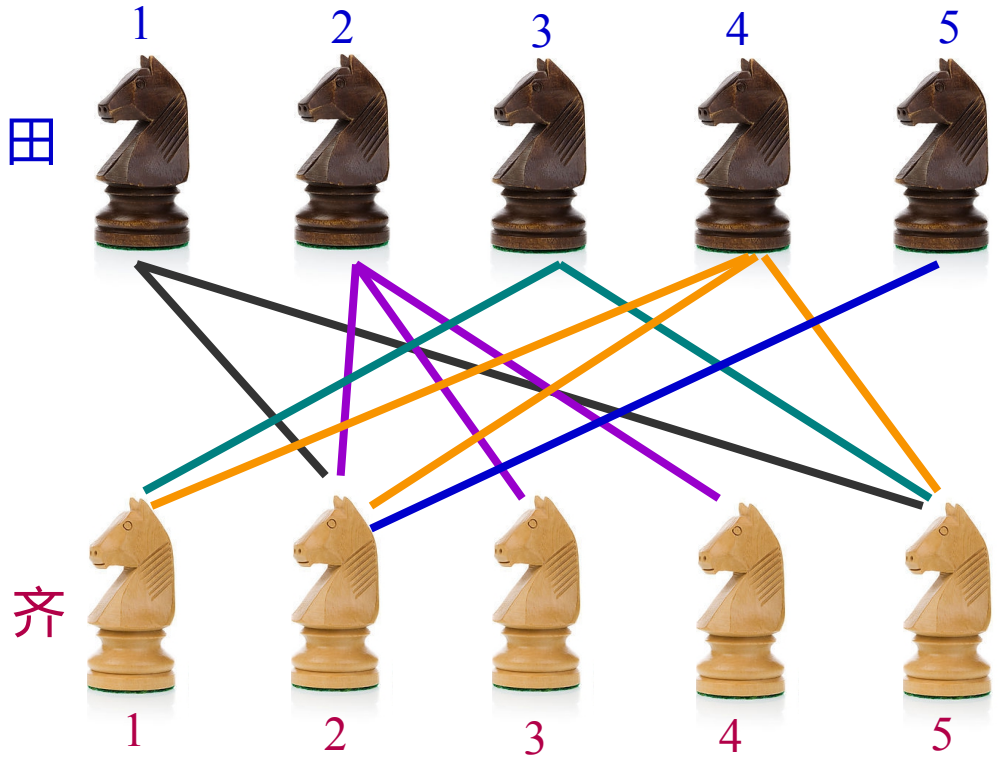
输出格式：一个整数，表示田忌最多能赢的场数。

输入样例：

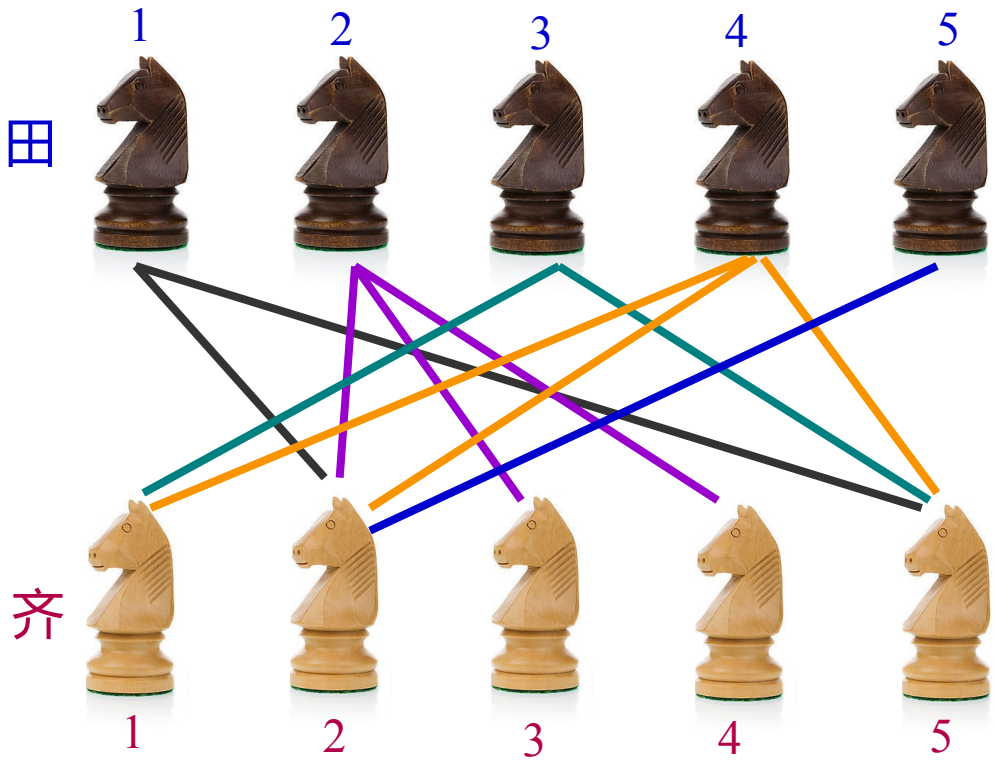
```
5 5
2 2 5
3 2 3 4
2 1 5
3 1 2 5
1 2
```

输出样例：

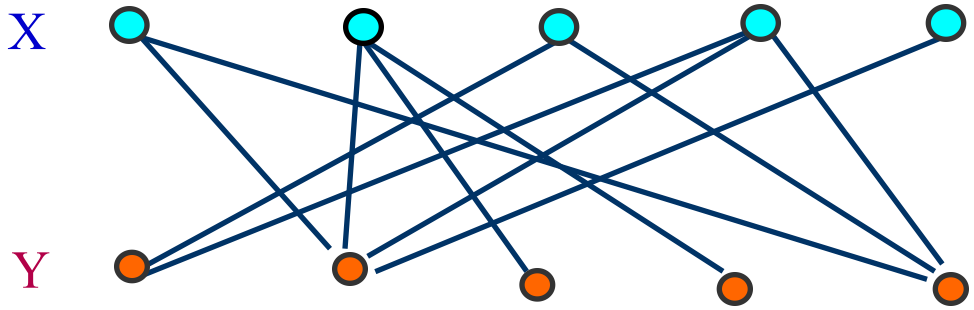
```
4
```



p1074 书本分配



即是找出最多有多少条一端位于"田"，一端位于"齐"的边且该边不于其它边有公共的交点(也就是每匹马最多连一条边)。



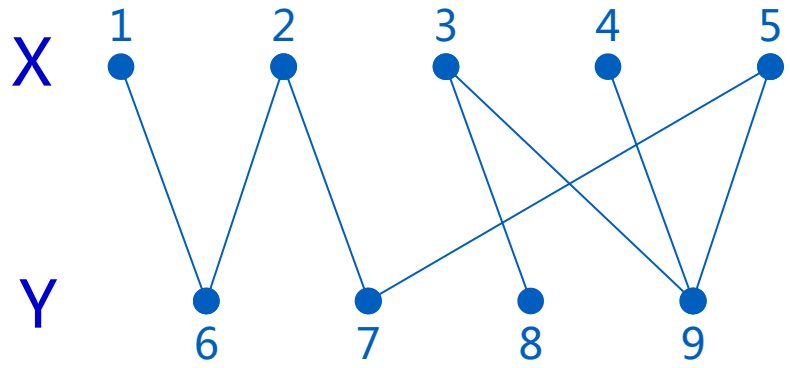
二分图

图中的点可分为X和Y两部分，图中的每条边的**两个端点**一定是**一个位于X，一个位于Y**。

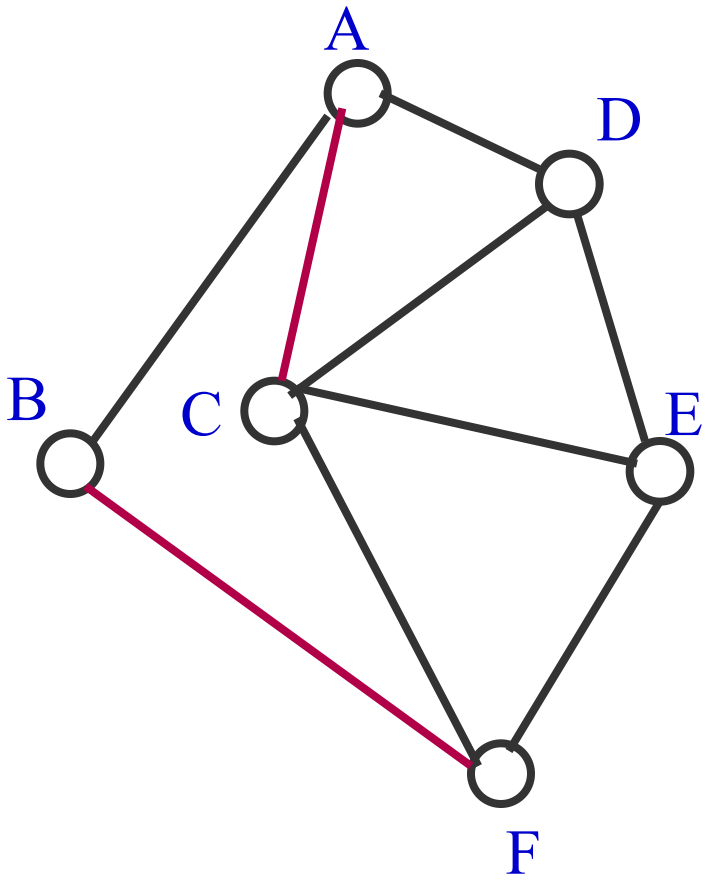
二分图的概念

二分图又称作二部图

若G是一个**无向图**。G的顶点分成X和Y两部分，G中每条**边**的两个顶点**一定是一个属于X另一个属于Y**。图G称为二分图。



什么是**匹配**？



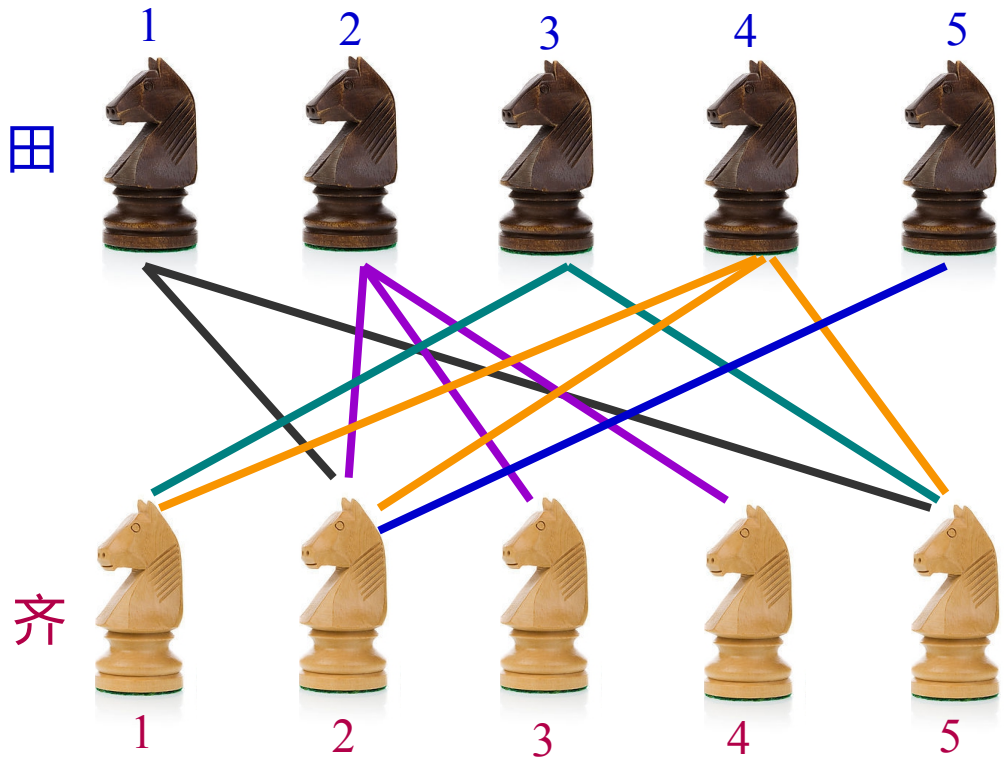
M {(A,C)（B,F)}

M是无向图G的若干条**边**的**集合**，如果M中任意两条边都没有公共端点，则称M是一个**匹配.**

若点Vi不与任何一条属于M的边相关联
则称Vi是一个**未盖点(未匹配点)**。
比如图中D、E就是未盖点。

最大匹配

- 给定一个二分图G，在G的一个子图M中，M的边集中的**任意两条边都没有公共顶点**，则称M是一个**匹配**。
- 这样的边数最多的集合称为图的**最大匹配**(maximal matching)，田忌赛马就是求最大匹配
- 如果一个匹配中，图中的每个顶点都和匹配中某条边相关联，则称此匹配为**完全匹配**，也称作**完备匹配**。



即是找出最多有多少条一端位于"田"，一端位于"齐"的边且该边不于其它边有公共的交点

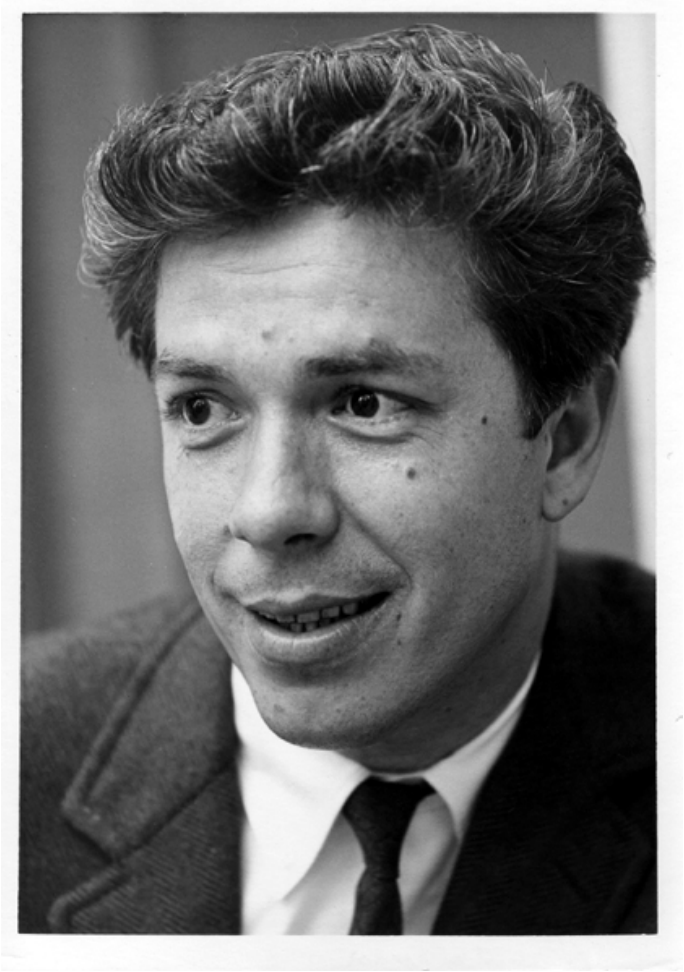
二分图+边无交点

也就是求二分图的最大匹配

求最大匹配

匈牙利算法

Hungarian Method



Harold W. Kuhn

匈牙利算法是用来解决二分图最大匹配问题的经典算法。

美国数学家**Harold W. Kuhn** 于1955年提出。此算法之所以被称作匈牙利算法是因为算法很大一部分是基于以前匈牙利数学家Dénes Kőnig和Jenő Egerváry的工作之上建立起来的.

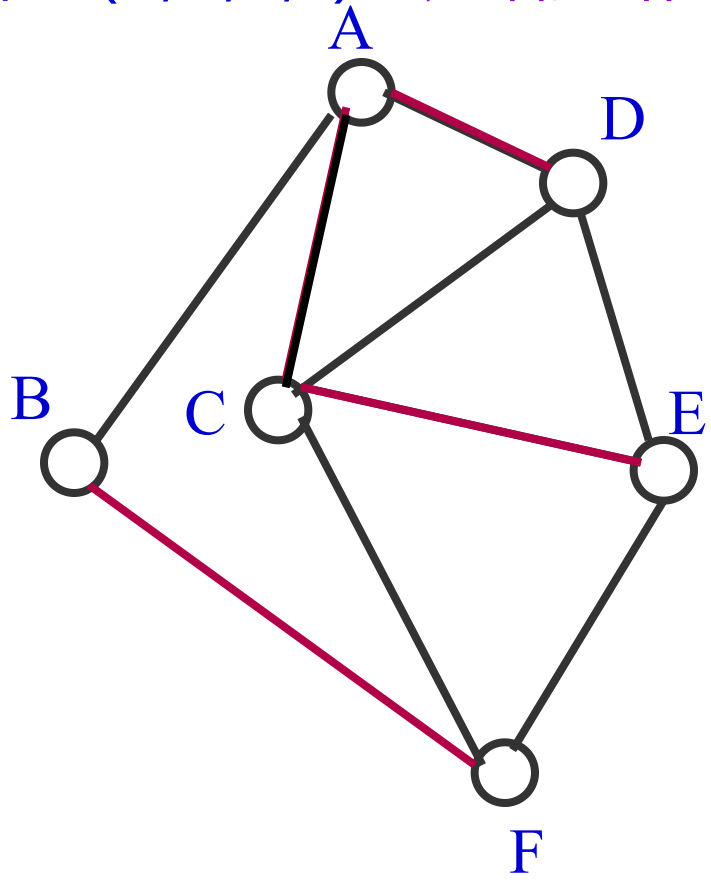
匈牙利算法是基于Hall定理中充分性证明的思想。

匈牙利算法

- 求最大匹配的一种显而易见的算法是：先找出全部匹配，然后保留匹配数最多的。但是这个算法的复杂度为边数的指数级函数。因此，需要寻求一种更加高效的算法。

- **增广路**的定义(也称增广轨或交错轨)：
  若P是图G中一条**连通两个未匹配顶点**的路径，并且属**匹配边集M的边和不属M的边**(即已匹配和未匹配的边)**在P上交替出现**，则称P为相对于M的一条增广路径。

路径P(D,A,C,E) P为增广路　有何作用？　增大匹配！



M{(A,C) (B,F)}

M是无向图G的若干条边的集合，如果M中任意两条边都没有公共端点，则称M是一个匹配

M’ {(A,D) (C,E) (B,F)}

增广路的起点和终点都是未盖点（未匹配点），
并且属于M的边和不属于M的边**交替**出现

把P中原来属于M边从M中删除，把P中原来不属于M边加入到M中，变化后得到的新的匹配M’恰好比原匹配多一条边。

匈牙利算法

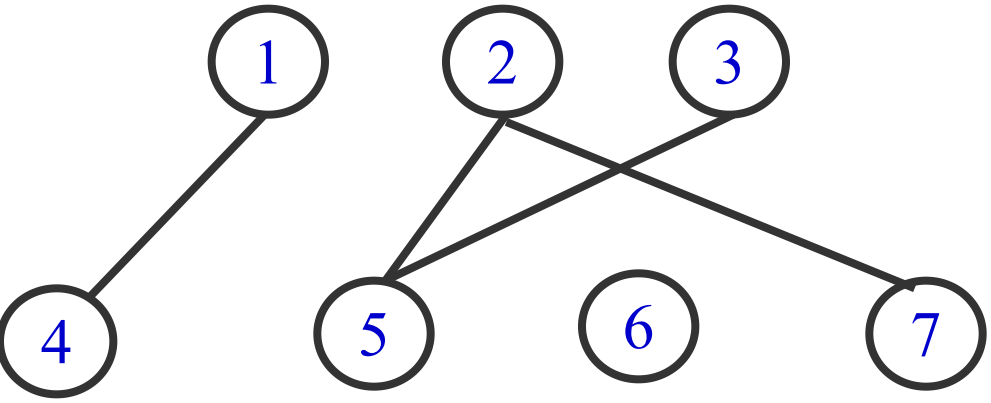
- 由增广路的定义可以推出下述四个结论：
- 1、P的路径长度必定为**奇数**，第一条边和最后一条边都不属于M。
- 2、P经过**取反**操作可以得到一个更大的匹配M’ 。
- 3、M为图G的最大匹配当且仅当**不存在**相对于M的增广路径。
- 4、如果两个未盖点之间仅含一条边，那么这条边是增广路

匈牙利算法

- 匈牙利算法就是用增广路求最大匹配
- 算法轮廓：
- (1)置M为空
- (2)找出一条增广路径P，通过取反操作获得更大的匹配M’代替M
- (3)重复(2)操作直到找不出增广路径为止

```
    bool map[maxn][maxm];                    //map[x][y]=true表示点x和点y有边相连
    bool road[maxm];                         //road[i]记录点i是否已在当前增广路中，防止死循环
    int link[maxm];       //link[i]记录增广路上与i相连的前一个节点的编号，即记录已求出的匹配
bool Find(int v)          //简单的说link[i]用于记录匹配集合中的边
{                                            //find查找从v点出发是否有可增广路
    int i;
    for(i=1;i<=m;i++)//可改用邻接表          //枚举在下半部分图中与v点相关联的点
    if(map[v][i] && (!road[i]))              //如果该点不在增广路上
    {
        road[i]=true;                        //把i标记为已讨论，防止死循环
        if (link[i]==0 || Find(link[i]) )    //i是未匹配点（未盖点）或者 从i的匹配点出发有可增广路
        {                                    //修改与i匹配的点为v
            link[i]=v;                       //则从v出发可找到增广路,返回true;
            return true;
        }
    }
    return false;                            //如果从v出发没有增广路，返回false
}
int main(){
//read the graph into array map[ ]           //首先读入图结构到map数组中
    for(i=1;i<=n;i++)
  {                                          //依次从上半部图的点出发，寻找增广路
      for(j=1;j<=m;j++)road[j]=false;
      if(Find(i))tot++;                      //每找到一条增广路，匹配数加1(最多n个匹配)
  }
  cout<<tot<<endl; return 0;                 //输出最大匹配数
}
```

时间复杂度为 O(nm)




```
for(i=1;i<=n;i++)
{
    for(j=1;j<=m;j++)road[j]=false;
    if(Find(i))       Total++;
}
```

```
bool Find(int v)
{
    int i;
    for(i=1;i<=m;i++)
    if(map[v][i] && (! road[i]) )
    {
        road[i]=true;
        if (link[i]==0 || find(link[i]) )
        {
            link[i]=v;
            return true;
        }
    }
    return false;
}
```

得到匹配 (4,1)

得到匹配 (5,2)(4,1)

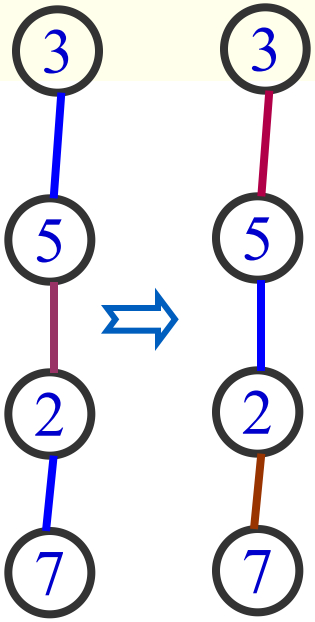
得到匹配 (5,3)(4,1)(2,7)

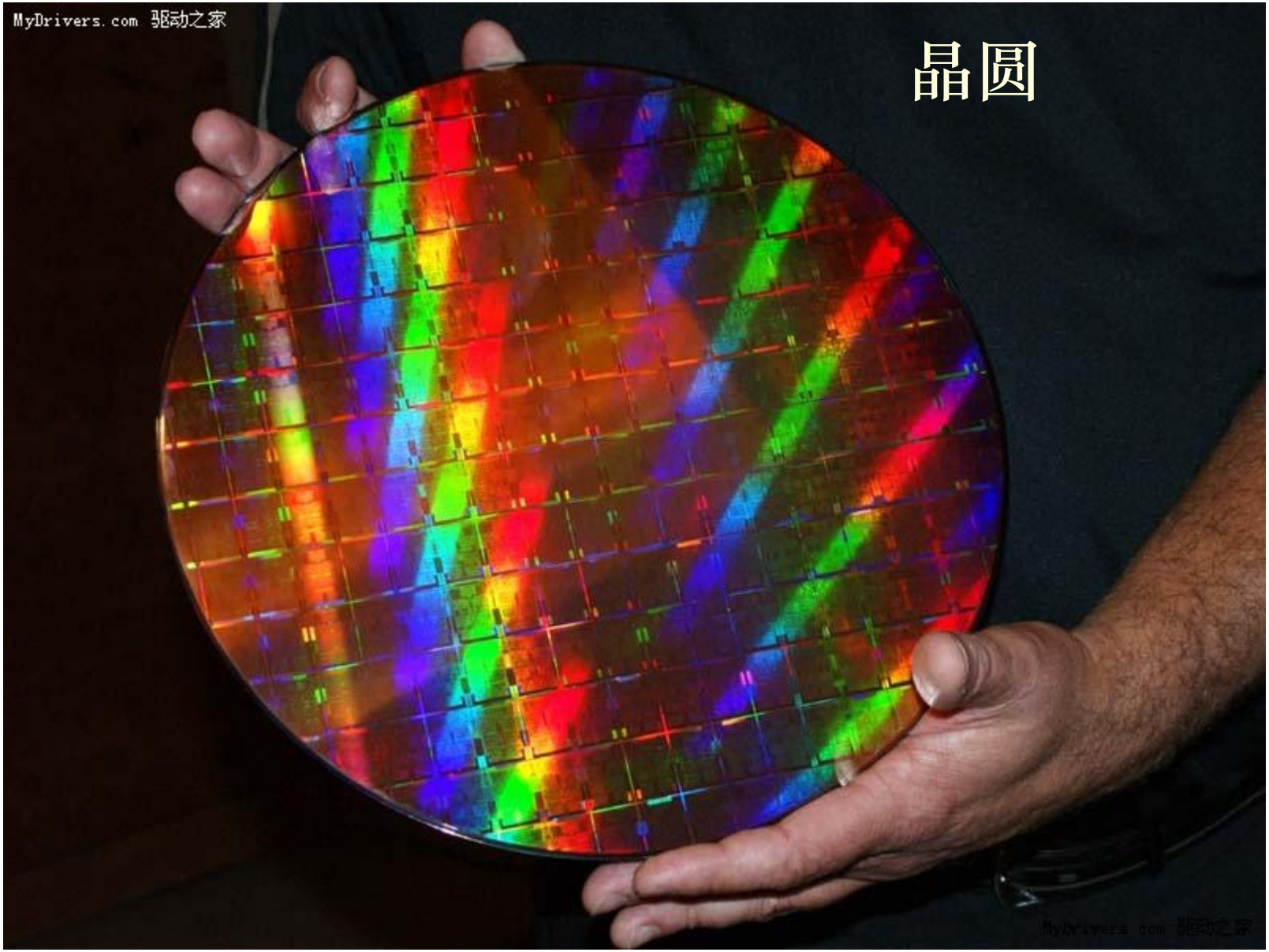
	1	2	3	4	5	6	7	8
link	0	0	0	1	3	0	2	0

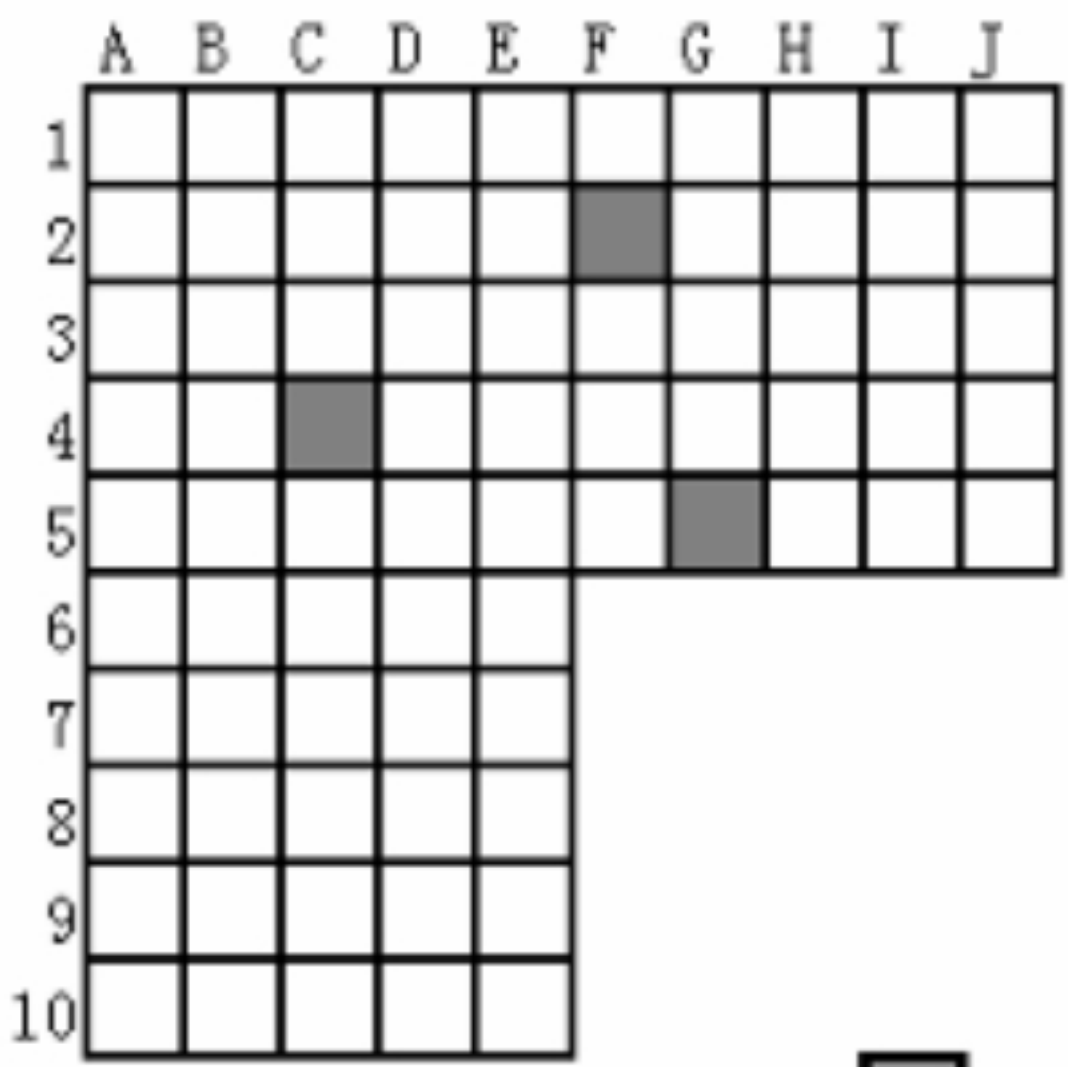
	1	2	3	4	5	6	7	8
road	false	false	false	false	true	false	true	false

Total= 3

起点3是一个未盖点，
终点7是一个未盖点，
表明找到一条增广路



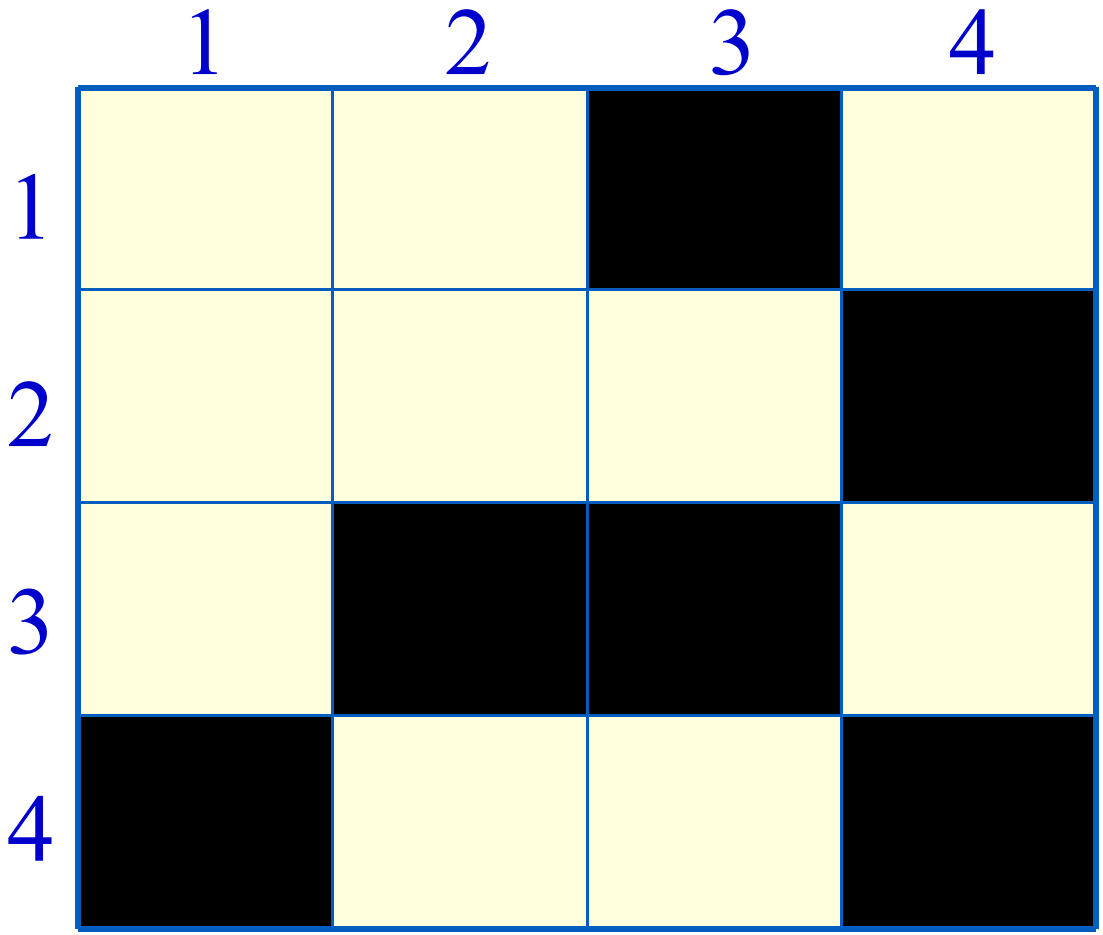

MyDrivers.com 驱动之家
晶圆

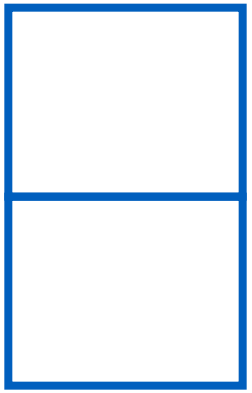


CPU内核蚀刻在硅片上，这种硅片由$3n^2$个内核拼成的L型。当n=5的时候，硅片的形状如图所示。现在要把CPU内核从硅片上切割下来，封装起来出售。

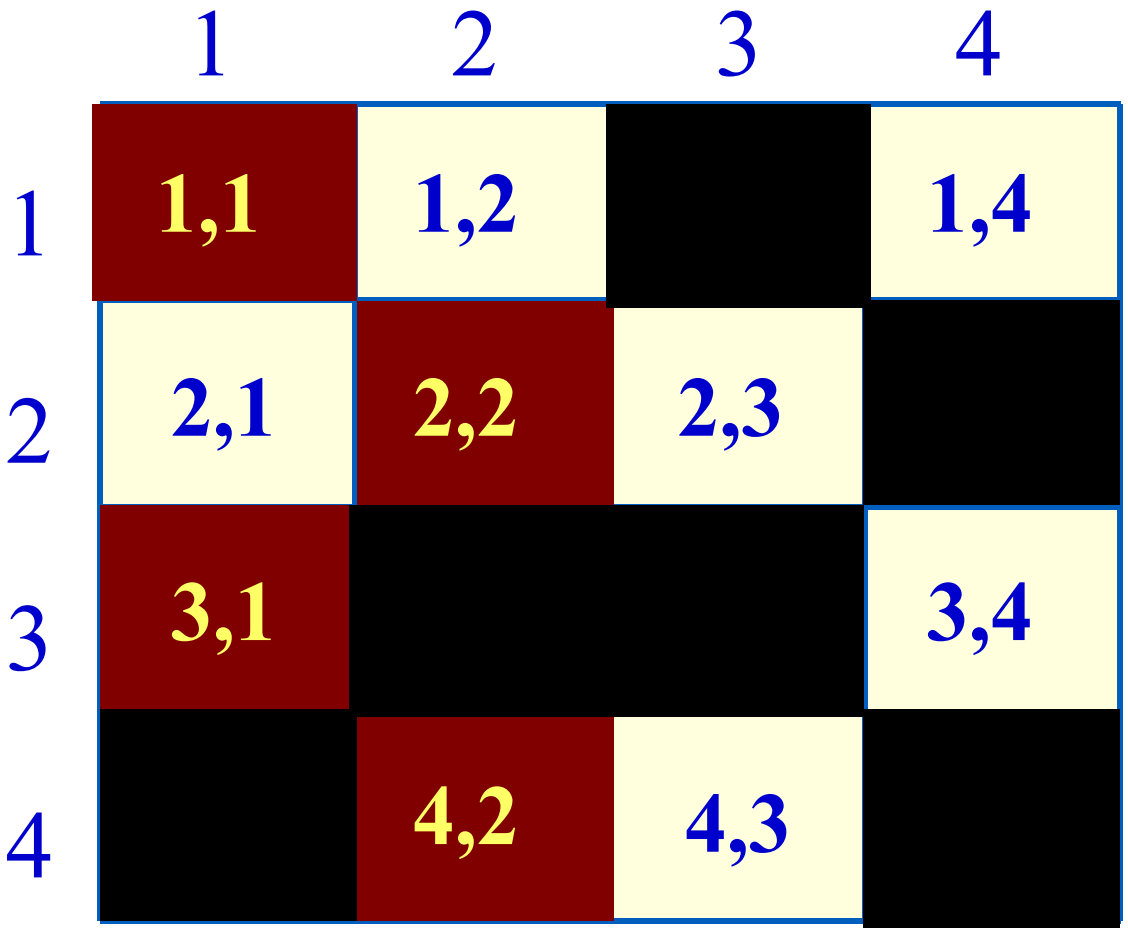
在图中，有3个不良内核，不良内核的坐标分别是C4，F2，G5。如果硅片上有m个不良内核，则有$3n^2-m$个内核是可以出售的，如果每个CPU内只装1个内核，此时就可以做出$3n^2-m$个CPU。

现在要做的是双核处理器，如果在L型硅片上任意两个相邻的CPU内核都是完好的，就可以把它们一起切下来做成一个双核CPU。所谓相邻，指的是CPU内核有一条公共边界。算算最多能切出多少个双核的CPU。





黑色表示损坏区域，白色表示可用区域，问最多可放置多少个1×2的方块

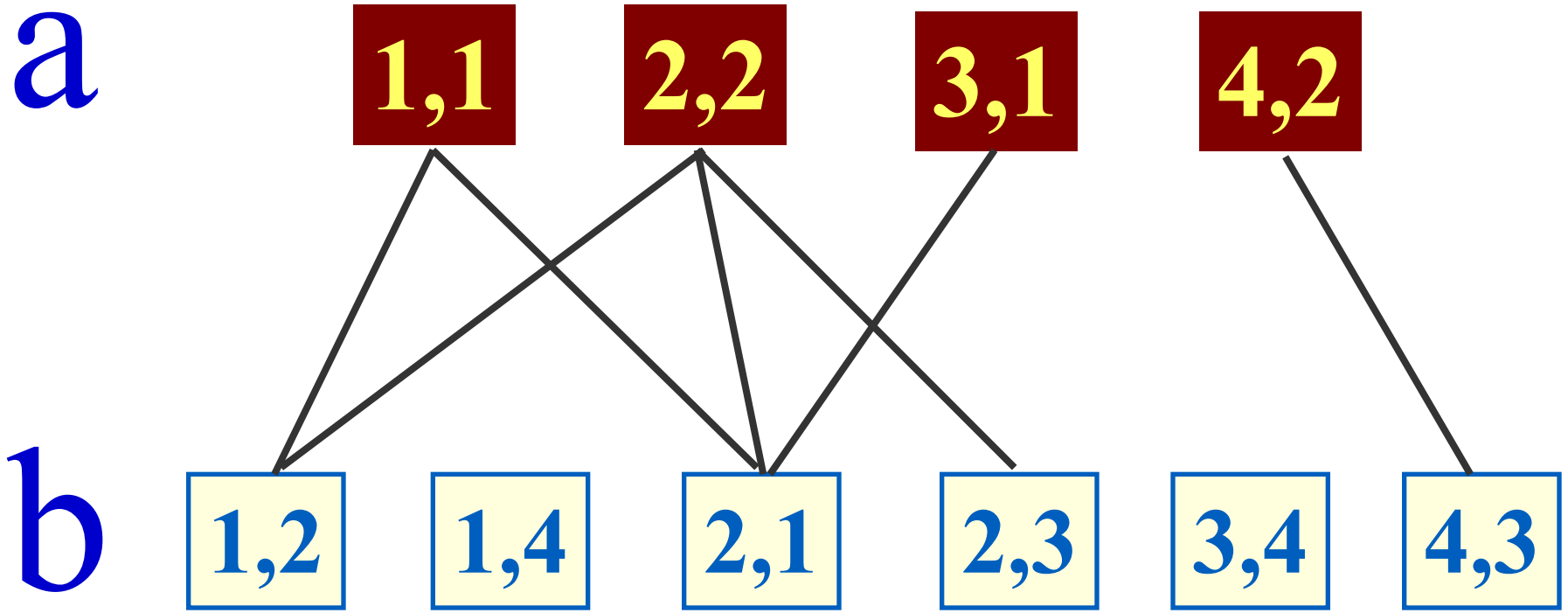


表示可用区域

表示摔坏区域

只要是相邻的两个不同颜色的方块便是可放入1×2方块区域

求最多能放置多少块1×2的方块即是找出最多有多少条一端位于a数组，一端位于b数组的边且该边不于其他边有公共的交点



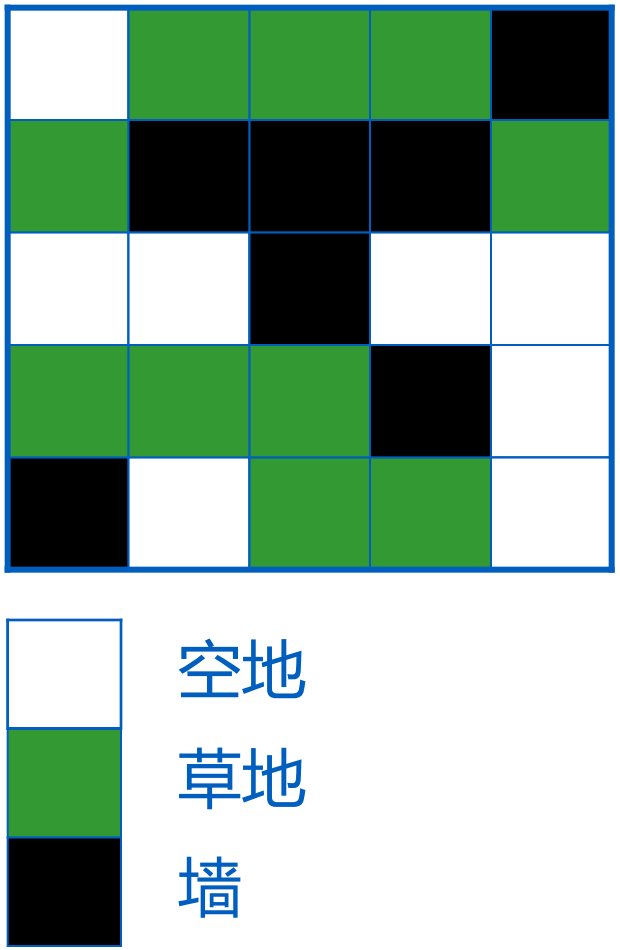
一条边相当于一个1×2的方块，如果一条边与其他边没有公共交点，表示此处可放置一个1×2的方块。

此题就被转换成了就二分图的最大匹配。

例题2 Place the Robots（ZOJ1654）

问题描述

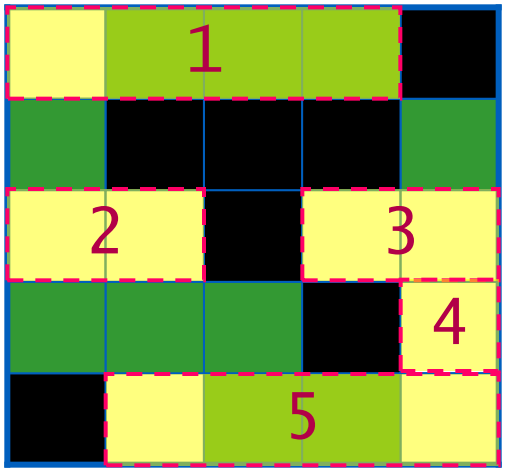
有一个N*M(N,M<=50)的棋盘，棋盘的每一格是三种类型之一：空地、草地、墙。机器人只能放在空地上。在同一行或同一列的两个机器人，若它们之间没有墙，则它们可以互相攻击。问给定的棋盘，最多可以放置多少个机器人，使它们不能互相攻击。



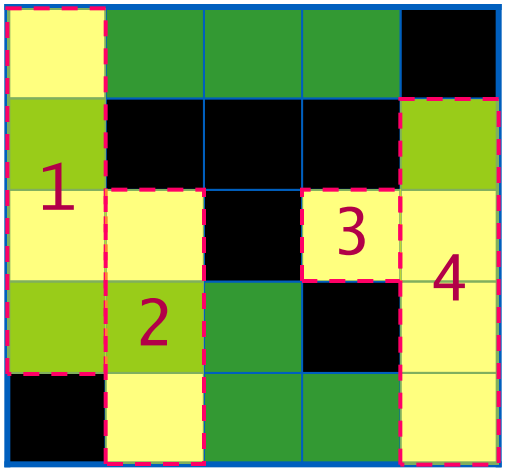
例题2 Place the Robots（ZOJ）

模型

我们将每一行，每一列被墙隔开，且包含空地的连续区域称作“块”。显然，在一个块之中，最多只能放一个机器人。我们把这些块编上号。



同样，把竖直方向的块也编上号。

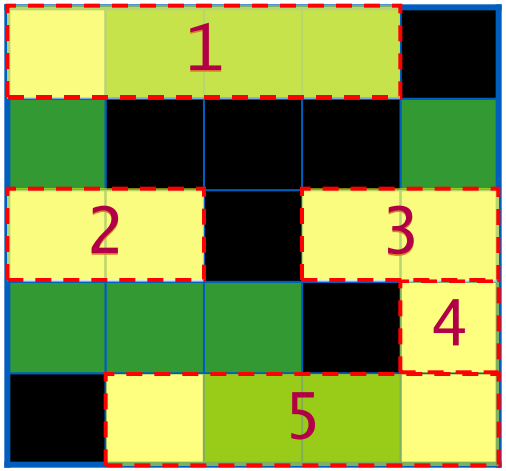


接下来怎么处理？

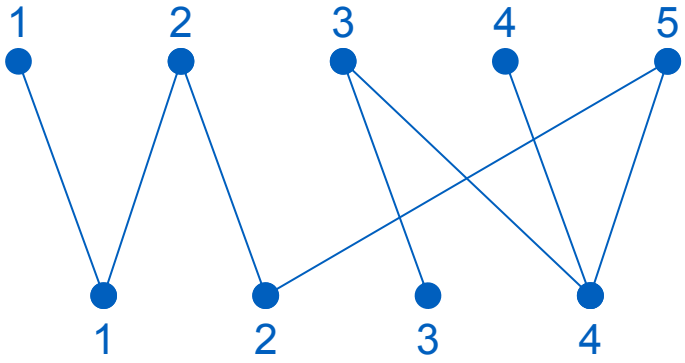
例题2 Place the Robots（ZOJ）

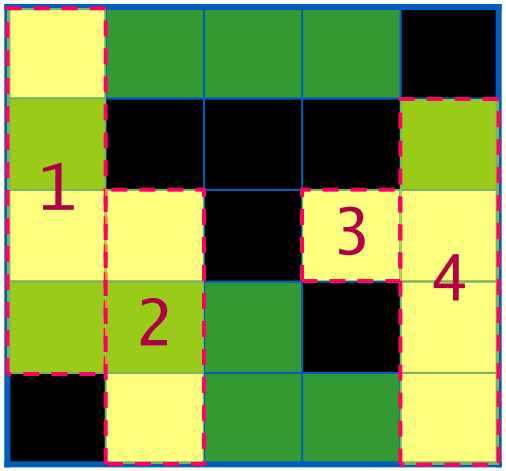
模型

把每个横向块看作X部的点，竖向块看作Y部的点，若两个块有公共的空地，则在它们之间连边。



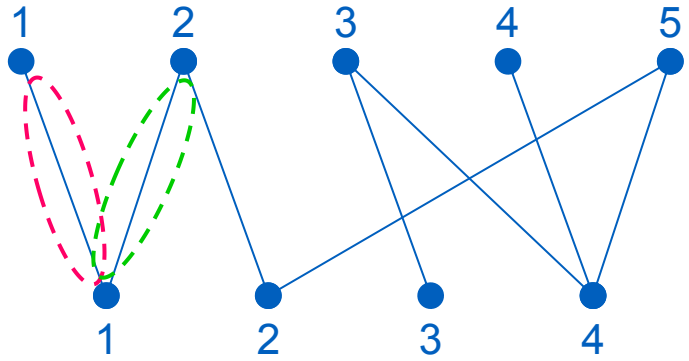
于是，问题转化成这样的一个二部图：

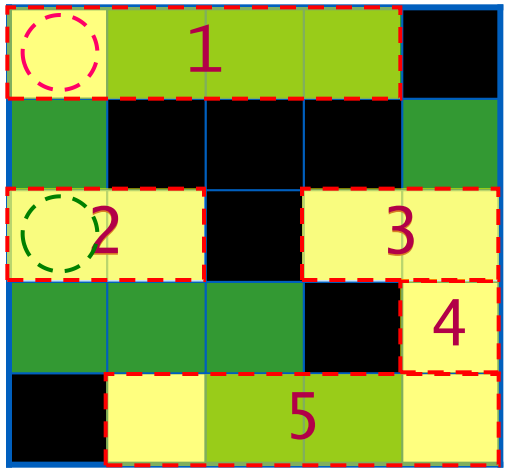




例题2 Place the Robots (ZOJ)

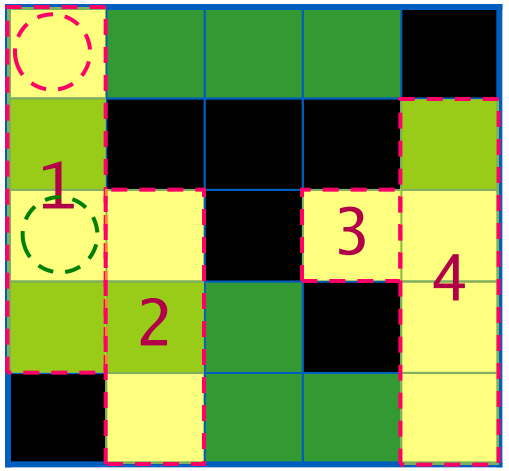
模型





图中每条边表示什么？

由于每条边表示一个空地，有冲突的空地之间必有公共顶点，所以问题转化为二部图的最大匹配问题。



课后习题：

1520
1517
1521
1524
1569

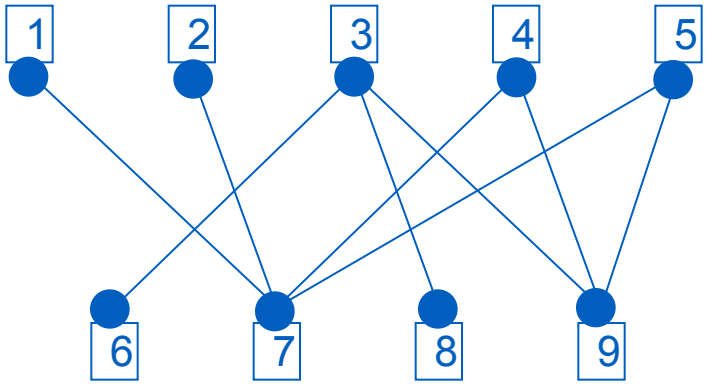
与最大匹配相关的一些定理

König 定理： 一个二分图的最大匹配数等于这个图的最小点覆盖数。

最小点覆盖：

下图是NK中学的校园地图，直线表示道路，圆点表示路口。 现在需要在某些路口上装上摄像头，使得所有道路都处于监视中。注意：一个路口上的摄像头可以监视与它相连的所有道路。

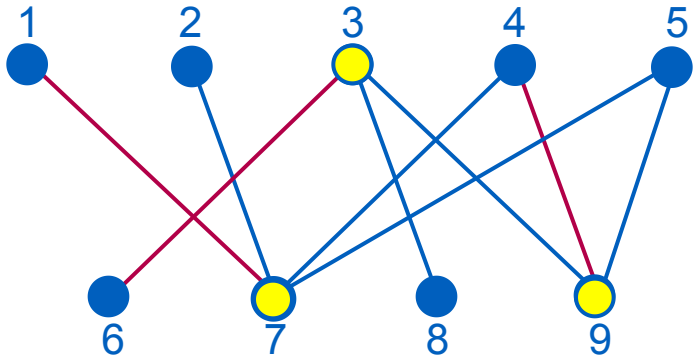
问最少安装几个摄像头就能满足要求，并找出它们的安装位置？



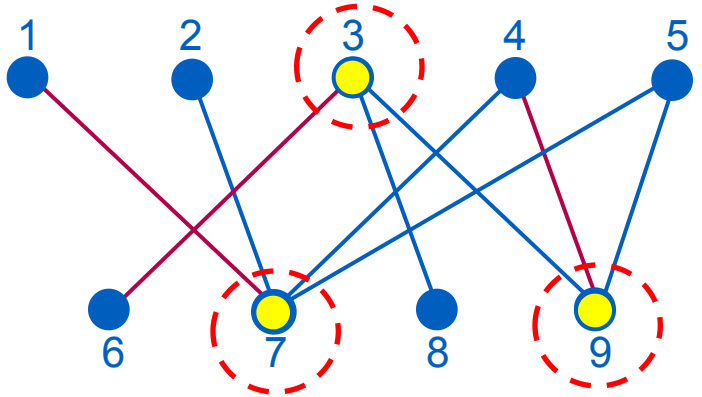
König 定理： 一个二分图的最大匹配数等于这个图的最小点覆盖数。

最小点覆盖：

假如选了一个点就相当于覆盖了和它相关联的所有边。最少需要选择多少个点才能覆盖图中所有的边，这就是**最小点覆盖数。**



在已求出最大匹配后，怎样找出最小覆盖点？



红色线条：最大匹配的边。
黄色的点：最小覆盖的点。

从上半部分的所有未匹配点出发，把按照增广路“交替出现”的要求可以走到的所有点（最后走出的路径是很多条不完整的增广路）都标记为红色。

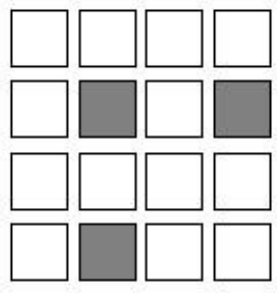
图中有三条路径(2,7,1)，（5,7,1）和（5,9,4,7,1)

上半部分所有**没有**被标记为红色的点，加下半部分所有**被**标记为红色的点。这些点组成了最小覆盖点集(图中红色虚线所示)。

König 定理的应用

柯南开锁

OIBH组织的大门有一个很神奇的锁.锁是由M*N个格子组成, 其中某些格子凸起(灰色的格子). 每一次操作可以把某一行或某一列的格子给按下去.



如果柯南能在组织限定的次数内将所有格子都按下去, 那么他就能够进入总部. 但是OIBH组织不是吃素的, 他们的限定次数恰是最少次数，请您帮助柯南计算出开给定的锁所需的最少次数.

Input

第一行 两个不超过100的正整数N, M表示矩阵的长和宽

以下N行 每行M个数 非0即1 1为凸起方格

Output

一个整数 所需最少次数

Sample Input

4 4

0000

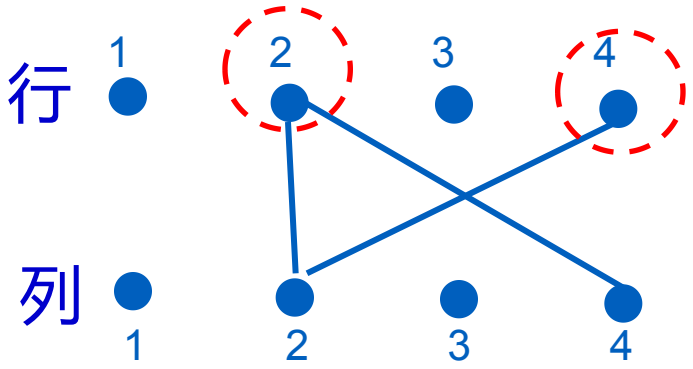
0101

0000

0100

Sample Output

2



二分图中的一条边相当于一个交点（凸起）

最小覆盖点集2和4表示按下第2行和第4行就可以解锁

二分图最大独立集

无向图的最大独立集(数): 从无向图的V个顶点中选出K个顶点，使得这K个顶点互不关联。 那么最大的K就是这个图的最大独立集(数).

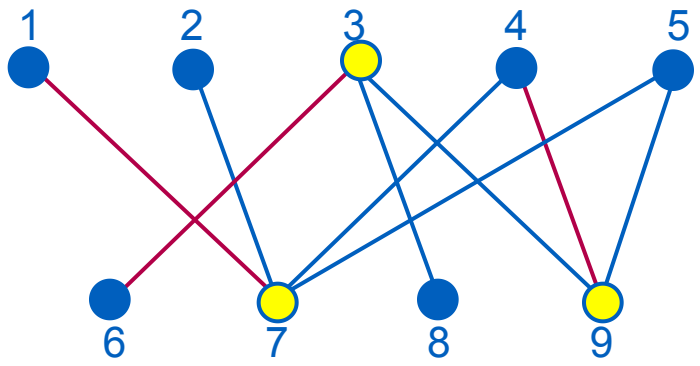
求无向图的最大独立集是NP问题

在N个点的二分图G中选出K个点，使这K个点两两之间没有边．K的最大值为**二分图的最大独立集**

定理：**二分图**最大独立集=**顶点总数V — 最大匹配数**

黄色点代表**最小点覆盖**（最小点覆盖恰好等于**最大匹配数**）

最小点覆盖关联了所有边，把最小点覆盖的这些点删除，意味着所有边都被删掉了，剩下的点自然不会有边相连。



典型例题：1606

二分图最小路径覆盖

路径覆盖: 在图中找一些路径，这些路径覆盖图中所有的顶点，且路径间无交点。

最小路径覆盖：在所有的路径覆盖中，**路径个数最小**的就是最小路径覆盖了。

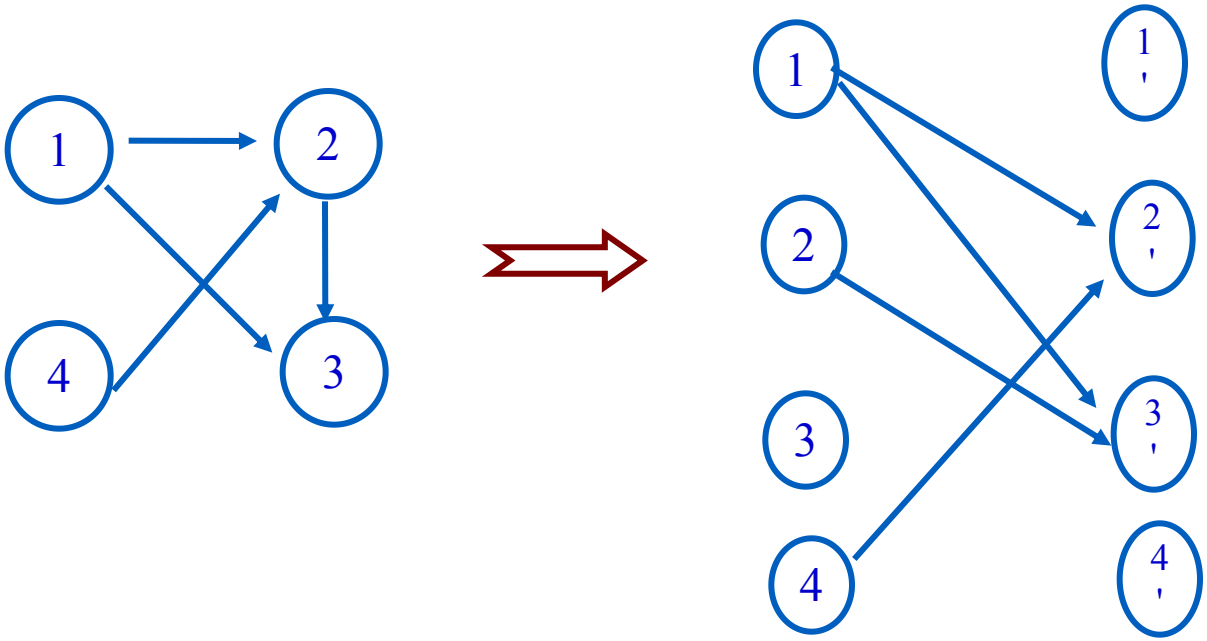
定理：二分图最小路径覆盖=顶点总数V — 最大匹配数

DAG的最小路径覆盖

DAG(有向无环图的缩写)的最小路径覆盖是指，在该图中选出最少的路径条数，使得图中所有点都被覆盖，且路径间公共无交点。

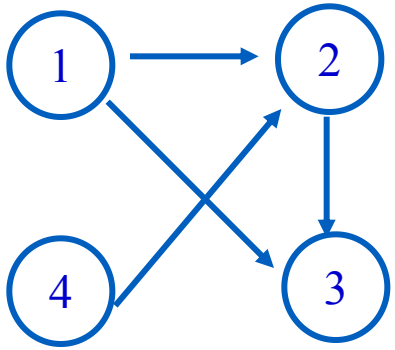
典型问题，给定一个n个点的有向无换图，可以延边画线，问最少画几笔，才能将所有点全部覆盖？要求每个点最多被画一次。

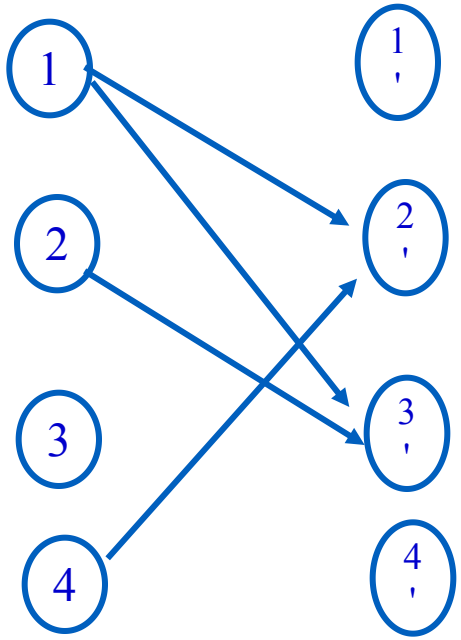
解法：拆点。将原图的n个点拆成2n个点，x号点拆成x和x'两个点。若原图中从x出发有边指向y,那么在新图中，从x出发，连接一条边到y'。



定理：DAG的最小路径覆盖=顶点数n - 对应二分图最大匹配数

DAG的最小路径覆盖





理解：二分图中的一个最大匹配为1->3'， 4->2',对应了原图中的边1->3和4->2。匹配中的边都没有公共交点，恰好满足路径覆盖的要求。

1->3'表明1->3这条边可成为最小路径覆盖中的一条边，且1号点一定不会是这条路径的终点，因为它指向了3号点(3号点是它的后继节点)。同理，4号点也不会是一条路径的终点，它指向了2号点。

排除了1和4，那么剩下的点就没有后继节点了，那他们一定是一条路径的终点，且一条路径只有一个终点。也就是剩下的每个节点都对应了一条路径的终点。

于是有：DAG的最小路径覆盖=顶点数n - 对应二分图最大匹配数

二分图相关定理小结：

二分图的**最大独立集** = 顶点数 - 二分图的最大匹配数

二分图的**最小顶点覆盖**= 二分图的最大匹配数

二分图的**最小路径覆盖**= 顶点数 - 二分图的最大匹配数

附录：Hall定理

Hall定理：

此定理使用于组合问题中；二部图G中的两部分顶点组成的集合分别为X, Y,

X={X1, X2, X3,X4, ……,Xm}

, Y={y1, y2, y3, y4 , ……,yn},

G中有一组无公共点的边，一端恰好为组成X的点的充分必要条件是：

X中的任意k个点至少与Y中的k个点相邻。（1≤k≤m)

本论还有一个重要推论：

二部图G中的两部分顶点组成的集合分别为X,Y, 若|X|=|Y|,且G中有一组无公共端点的边，一端恰好组成X中的点，一端恰好组成Y中的点，

则称二部图G中存在完美匹配。若图G的每个点度数为t，则称二部图G为t---正则的二部图存在完美匹配。

本定理是二分图匹配问题中匈牙利算法的基础。