公式

二分图最小顶点覆盖 = 二分图最大匹配；

DAG 图的最小路径覆盖 = 节点数（n）- 最大匹配数；

二分图最大独立集 = 节点数（n）- 最大匹配数；

定义

二分图

二分图又称作二部图，是图论中的一种特殊模型。 设 G=(V,E)是一个无向图，如果顶点 V 可分割为两个互不相交的子集(A,B)，并且图中的每条边（i，j）所关联的两个顶点 i 和 j 分别属于这两个不同的顶点集(i in A,j in B)，则称图 G 为一个二分图。
通俗的定义：二分图是一组点集可以分为两部分，且每部分内各点互不相连，两部分的点之间可以有边。

最大二分图匹配

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不依附于同一个顶点，则称 M 是一个匹配.
选择这样的边数最大的子集称为图的最大匹配问题，如果一个匹配中，图中的每个顶点都和图中某条边相关联，则称此匹配为完全匹配，也称作完备匹配.

最大二分图匹配即 ”对于二分图的所有边，寻找一个子集，这个子集满足两个条件，
1：任意两条边都不依赖于同一个点。
2：让这个子集里的边在满足条件一的情况下尽量多。

最大独立集

最大独立集是指求一个二分图中最大的一个点集，该点集内的点互不相连。

独立集即一个点集，集合中任两个结点不相邻，则称V为独立集。

或者说是导出的子图是零图（没有边）的点集。

极大独立集(maximal independent set)：本身为独立集，再加入任何点都不是。

最大独立集(maximum independent set)：点最多的独立集。

独立数(independent number)：最大独立集的点。

最小顶点覆盖

最小覆盖要求用最少的点（X 集合或 Y 集合的都行）让每条边都至少和其中一个

点关联。可以证明：最少的点（即覆盖数）＝最大匹配数

点覆盖集即一个点集，使得所有边至少有一个端点在集合里。或者说是“点” 覆盖了所有“边”。
极小点覆盖(minimal vertex covering)：本身为点覆盖，其真子集都不是。
最小点覆盖(minimum vertex covering)：点最少的点覆盖。
点覆盖数(vertex covering number)：最小点覆盖的点数。

最小边覆盖

边覆盖集即一个边集，使得所有点都与集合里的边邻接。或者说是“边” 覆盖了所有“点”。

极小边覆盖(minimal edge covering)：本身是边覆盖，其真子集都不是。

最小边覆盖(minimum edge covering)：边最少的边覆盖。

边覆盖数(edge covering number)：最小边覆盖的边数。

最大边独立集（最大二分图匹配）

边独立集即一个边集，满足边集中的任两边不邻接。

极大边独立集(maximal edge independent set)：本身为边独立集，再加入任何边都不是。

最大边独立集(maximum edge independent set)：边最多的边独立集。

边独立数(edge independent number)：最大边独立集的边数。

边独立集又称匹配(matching)，相应的有极大匹配(maximal matching)，最大匹配(maximum matching)，匹配数(matching number)。

最小支配集

支配集即一个点集，使得所有其他点至少有一个相邻点在集合里。
或者说是一部分的“点”支配了所有“点”。

极小支配集(minimal dominating set)：本身为支配集，其真子集都不是。

最小支配集(minimum dominating set)：点最少的支配集。

支配数(dominating number)：最小支配集的点数。

最小边支配集

边支配集即一个边集，使得所有边至少有一条邻接边在集合里。
或者说是一部分的“边”支配了所有“边”。

极小边支配集(minimal edge dominating set)：本身是边支配集，其真子集都不是。

最小边支配集(minimum edge dominating set)：边最少的边支配集。

边支配数(edge dominating number)：最小边支配集的边数。

团

团即一个点集，集合中任两个结点相邻。或者说是导出的子图是完全图的点集。极大团(maximal clique)：本身为团，再加入任何点都不是。

最大团(maximum clique)：点最多的团。

团数(clique number)：最大团的点数。

最小路径覆盖

最小路径覆盖是指一个不含圈的有向图G 中，G 的一个路径覆盖是一个其结点不相交的 路径集合P，图中的每一个结点仅包含于P 中的某一条路径。路径可以从任意结点开始和结束，且长度也为任意值，包括 0

用尽量少的不相交简单路径覆盖有向无环图G的所有结点。
解决此类问题可以建立一个二分图模型。把所有顶点 i 拆成两个：X结点集中的 i 和 Y 结点集中的 i’,如果有边 i->j，则在二分图中引入边 i->j’，设二分图最大匹配为 m,则结果就是 n-m。

最小路径覆盖(path covering)：是“路径” 覆盖“点”，即用尽量少的不相交简单路径覆盖有向无环图 G 的所有顶点，即每个顶点严格属于一条路径。

路径的长度可能为0(单个点)。

最小路径覆盖数＝G 的点数－最小路径覆盖中的边数。

应该使得最小路径覆盖中的边数尽量多，但是又不能让两条边在同一个顶点相交。

拆点：将每一个顶点 i 拆成两个顶点 Xi 和 Yi。

然后根据原图中边的信息，从 X 部往 Y 部引边。

所有边的方向都是由 X 部到 Y 部。

因此，所转化出的二分图的最大匹配数则是原图 G 中最小路径覆盖上的边数。

因此由最小路径覆盖数＝原图 G 的顶点数－二分图的最大匹配数便可以得解。

增广路（增广轨）

若 P 是图 G 中一条连通两个未匹配顶点的路径，并且属于 M 的边和不属于 M 的边(即已匹配和待匹配的边)在 P 上交替出现，则称 P 为相对于 M 的一条增广路径（举例来说，有 A、B 集合，增广路由 A 中一个点通向 B 中一个点，再由 B 中这个点通向 A 中一个点……交替进行）。

增广路径的性质：

1 有奇数条边。

2 起点在二分图的左半边，终点在右半边。

3 路径上的点一定是一个在左半边，一个在右半边，交替出现。（其实二分图的性质就决定了这一点，因为二分图同一边的点之间没有边相连，不要忘记哦。）

4 整条路径上没有重复的点。

5 起点和终点都是目前还没有配对的点，而其它所有点都是已经配好对的。

6 路径上的所有第奇数条边都不在原匹配中，所有第偶数条边都出现在原匹配中。

7 最后，也是最重要的一条，把增广路径上的所有第奇数条边加入到原匹配中去，并把增广路径中的所有第偶数条边从原匹配中删除（这个操作称为增广路径的取反），则新的匹配数就比原匹配数增加了 1 个。

了解了增广路的定义以及性质之后，我们仔细理解第7条性质。因为增广路径的长度为奇数，我们不妨设为2*K+1，又因为第一条是非匹配边，且匹配边与非匹配边交替出现，所以非匹配边有 K+1条，匹配边有 K 条。此时，我们做取反操作，则匹配边的个数会在原来的基础上+1。求最大匹配的“匈牙利算法”即是此思想：无论从哪个匹配开始，每一次操作都让匹配数+1，不断扩充，直到找不到增广路径，此时便得到最大匹配。

算法

匈牙利算法

算法的核心就是根据一个初始匹配不停的找增广路，直到没有增广路为止。

匈牙利算法的本质实际上和基于增广路特性的最大流算法还是相似的，只需要注意两点：

（一）每个 X 节点都最多做一次增广路的起点；

（二）如果一个 Y 节点已经匹配了，那么增广路到这儿的时候唯一的路径是走到 Y 节点的匹配点（可以回忆最大流算法中的后向边，这个时候后向边是可以增流的）。

找增广路的时候既可以采用 dfs 也可以采用 bfs，两者都可以保证 O(nm)的复杂度，因为每找一条增广路的复杂度是 O(m)，而最多增广 n 次，dfs 在实际实现中更加简短。

模板

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
const int N=1001;
int n1,n2,k;
//n1,n2 为二分图的顶点集,其中 x∈n1,y∈n2
int map[N][N],vis[N],link[N];
//link 记录 n2 中的点 y 在 n1 中所匹配的 x 点的编号
int find(int x)
{
    int i;
    for(i=1;i<=n2;i++)
    {
        if(map[x][i]&&!vis[i])//x->i 有边,且节点 i 未被搜索
        {
            vis[i]=1;//标记节点已被搜索
            //如果 i 不属于前一个匹配 M 或被 i 匹配到的节点可以寻找到增广路
            if(link[i]==0||find(link[i]))
            {
                link[i]=x;//更新
                return 1;//匹配成功
            }
        }
    }
    return 0;
}
int main()
{
    int i,x,y,s=0;
    scanf("%d%d%d",&n1,&n2,&k);
```

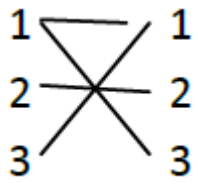
```
    for(i=0;i<k;i++)
    {
        scanf("%d%d",&x,&y);
        map[x][y]=1;
    }
    for(i=1;i<=n1;i++)
    {
        memset(vis,0,sizeof(vis));
        if(find(i))
            s++;
    }
    printf("%d\n",s);
    return 0;
}
```

建图模型

行列匹配法

1 0 1
0 1 0
1 0 0

上图是一个3 * 3 的矩阵，方格内的1表示这个地方有敌人，0表示没有敌人，现在我们有很多箭，每根箭可以杀死一行或者一列的敌人，问题是，我们要杀死所有的敌人至要用到几根箭？
我们要杀死某个敌人，只要让他所在的位置有箭经过就行。也就是所有的位置都被箭覆盖就行，对就是覆盖，就是顶点的最小覆盖，既然是顶点的最小覆盖，而且我们要杀的是敌人，那么我们的点就应该是敌人的位子，即（行列）对于上面那个图我么可以建立下面这个模型



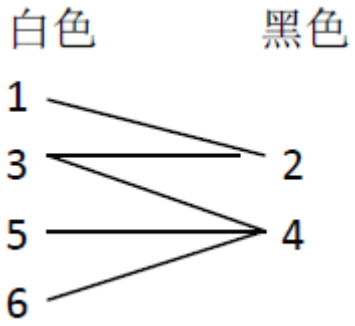
黑白染色法

1 0 1
1 1 1
0 0 1

要求是把方格里的所有的 1 改为零，一次最多只能修改相邻的两个，为最少需要修改几次？

每个点能和他四周的四个点匹配，那么我们怎么把所有的点分成来那个部分呢？

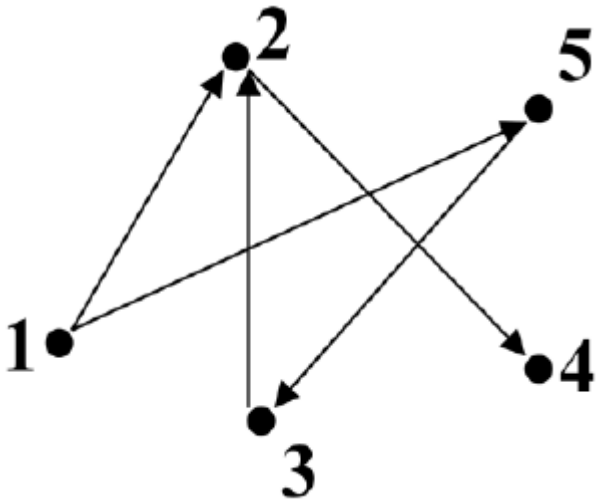
白1　黑　白5
黑**2**　白3　黑**4**
白　黑　白6



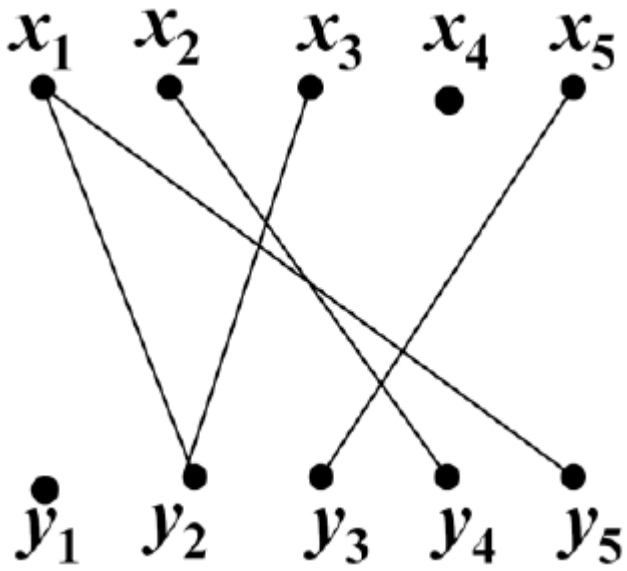
反建法

拆点法

拆点法是用于解决最小路径覆盖问题的，给出一个图



要找到几条路径，可以把所有的点经过，并且路径之间不可以交叉。我们的做法是把点拆成两部分（点 1 拆为 x1,y1. 点 2 拆为 x2,y2……）



如果我们对这个图求二分图的最大匹配，你会发现每个匹配对应着一个路径覆盖，因此，此二分图的最大匹配即：原图中的最小路径覆盖上的边的个数（路径是由 0 条，1 条或多条边组成的）。那么原图的最小路径覆盖数 = 原图顶点数 – 最小路径上的边数 也就是 原图的最小路径覆盖数 = 原图顶点数 – 二分图最大匹配数。

一行变多行，一列变多列

		###	
###			
		###	###
	###		

上面是一个 4*4 的方格，方格内的###表示墙，我们要在表格内没有墙的地方建立碉堡，而且要保证任何两个碉堡之间互相不能攻击，问最多能建多少个碉堡？

1,1	1,2	###	1,4
###	2,2	2,3	2,4
3,1	3,2	###	###
4,1	###	4,3	4,4

原图

1,1	1,2	###	2,4
###	3,2	3,3	3,4
4,5	4,2	###	###
5,5	###	6,6	6,7

（因为有了墙所以第一行变为两行）

（因为上面有了二行固只能从第三行开始）

（因为第一列有了墙，固列数增加为 5）

（第 3,4 列有了墙，固列数增加到了 6 和 7）

一行变多行，一列变多列后的图

练习

POJ 3041 Asteroids

POJ 1274 The Perfect Stall

POJ 1469 COURSES