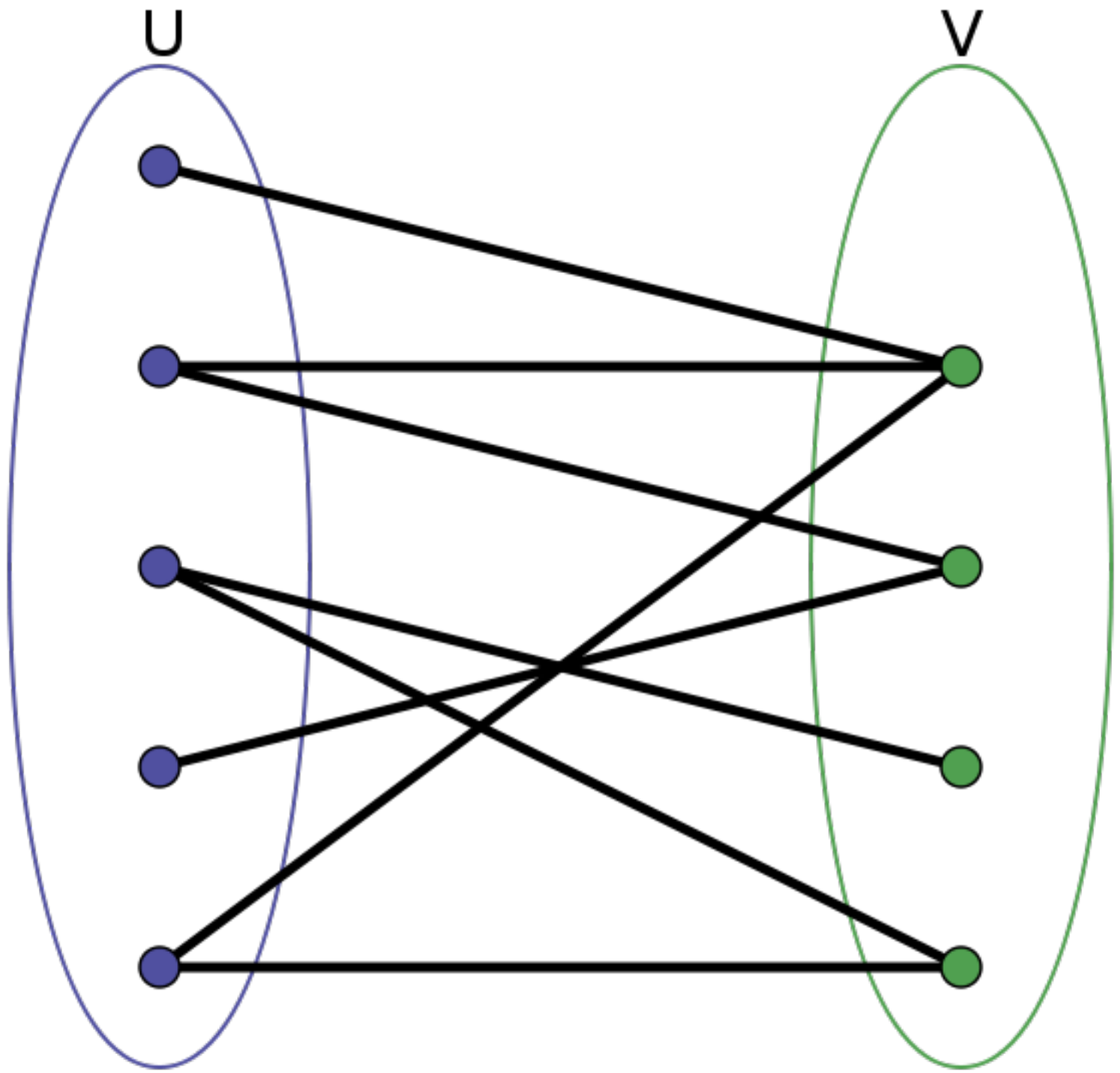
# 二分图匹配


U
V

# 二分图基础知识

- 二分图可以分成左右两个点集，所有的边都连接着U到V
- 二分图中没有奇环
- 匹配：是一些边的集合，任意两条边没有公共顶点
- 二分图的最大匹配：二分图的所有匹配中边数最多的匹配
- 完美匹配：所有的点都在匹配中
- 匹配边，匹配点：在一个匹配中的边与点
- 匹配点又叫做盖点，非匹配点叫做未盖点
- 二分图最大匹配算法：匈牙利算法

Fig.1

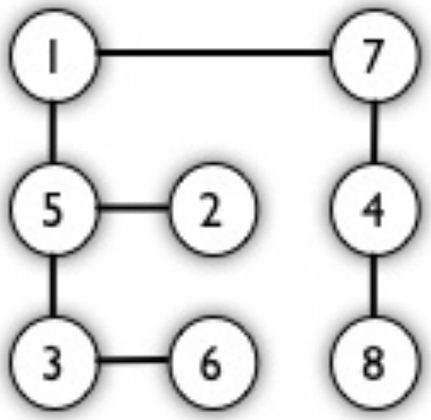



Fig.2

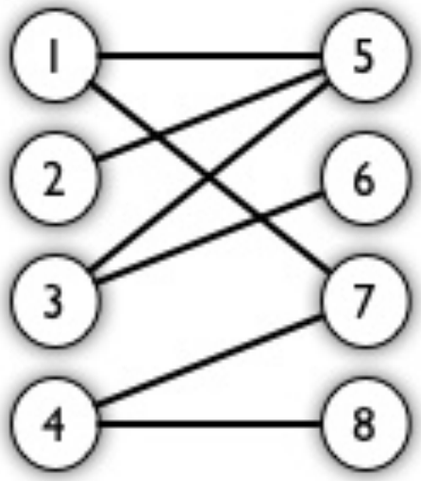



Fig.3

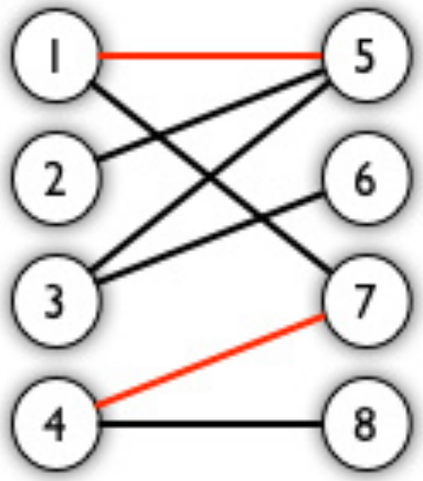
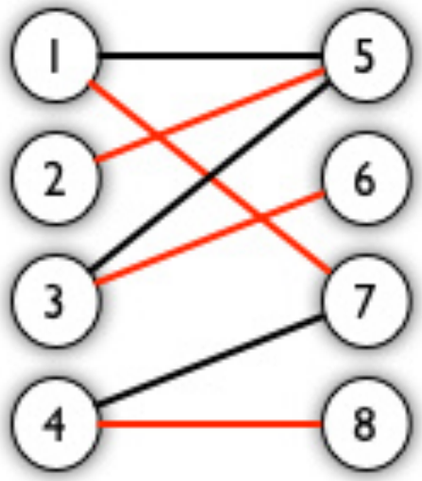
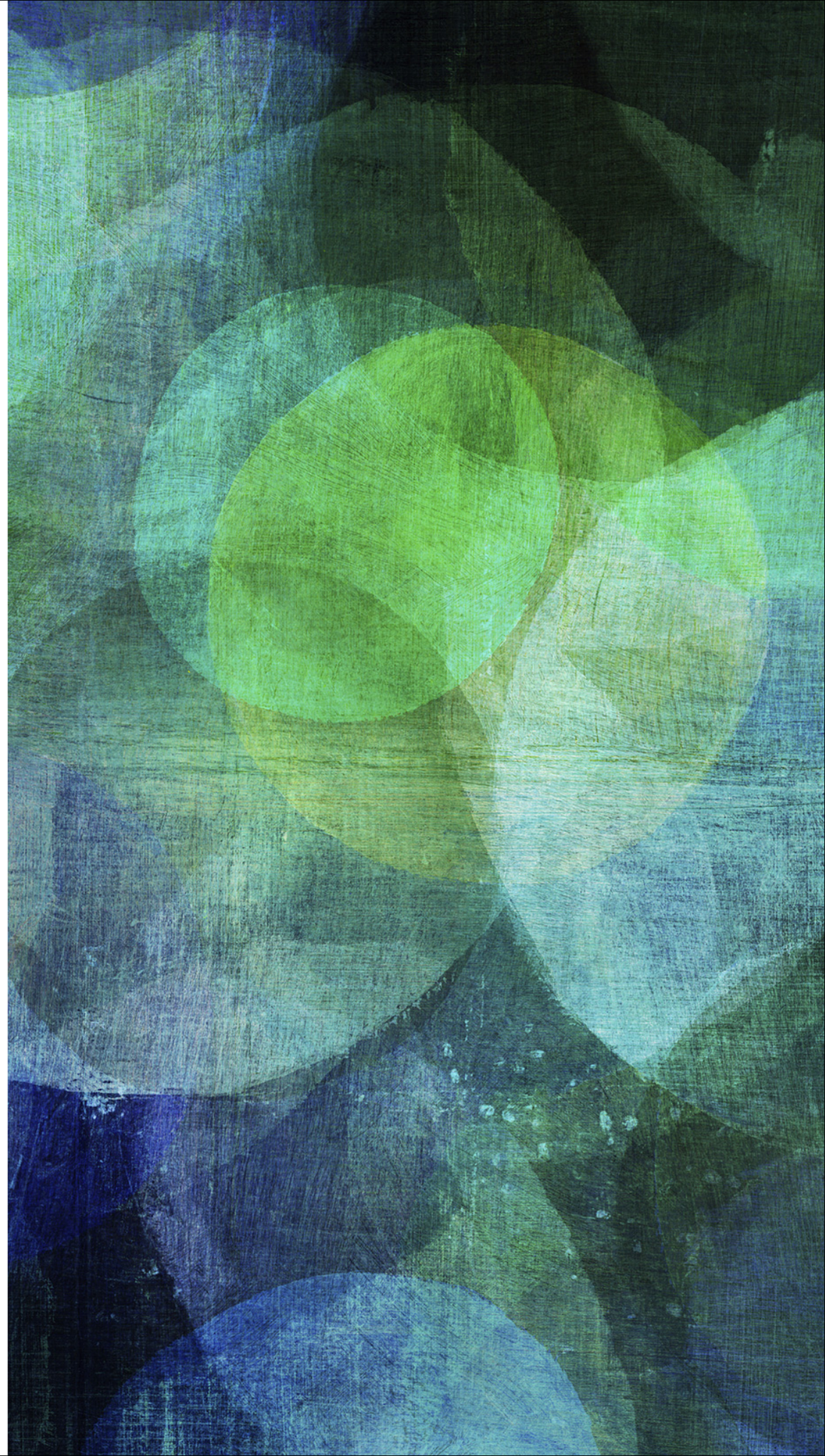



Fig.4



# 匈牙利算法

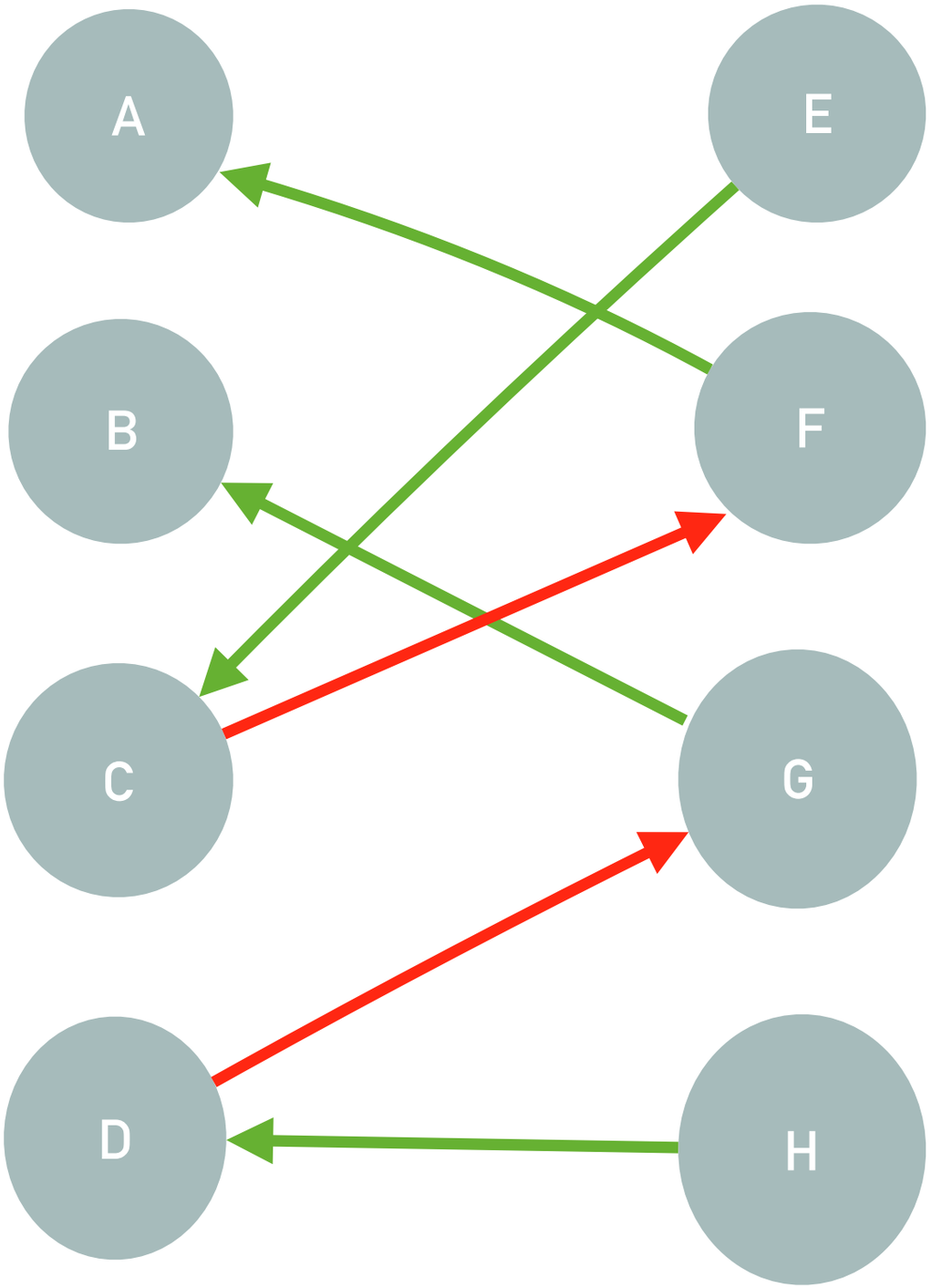


# 核心

- 假设你有了一个匹配P，我们比较这个匹配与最大匹配M
- 如果M里面有匹配边是P里面没有的而且匹配边对应的两个匹配点在当前也是未盖点，那就把这些边直接连上，得到Q
- 现在的Q已经没法再通过单独增加一个与之前毫不相关的匹配边来扩大匹配了
- 也就是说任意一个M里面的匹配边在Q里面都会覆盖Q的某个匹配点
- 假设Q还不是最大匹配，那么我们要将Q变成M的方法就是将Q中的边替换成M中的边
- 这个替换的前提就是边集X（M - M交Q）与 边集Y(Q - M交Q)，我们称之为对称差， 一定会形成x-y-x...-x-y-x这样交错的路径，属于X边集的边会比属于Y边集的边多一条，如果不存在这样的路径，说明M的匹配数不比Q多。
- 因此我们可以将交错路径的匹配边与非匹配边互换

## 另一个角度

- 换个角度看，只要当前的匹配存在增广路，我们就可以通过不断的交换匹配的方式到达最大匹配
- 如果某时刻不存在增广路了，就说明已经到达了最大匹配
- 因为任意一个非最大匹配的匹配一定会存在增广路
- 从一边的未盖点出发，交替着走匹配边，非匹配边，走到另一边的未盖点就是找到了增广路
- 已经匹配的点永远不会退出匹配，只会更换匹配
- 枚举左侧的每个点跑增广路，每增加一个匹配，新增的匹配点是当前点与某个右侧的点，左侧未枚举到的点不会成为匹配点


A
E
B
F
C
G
D
H

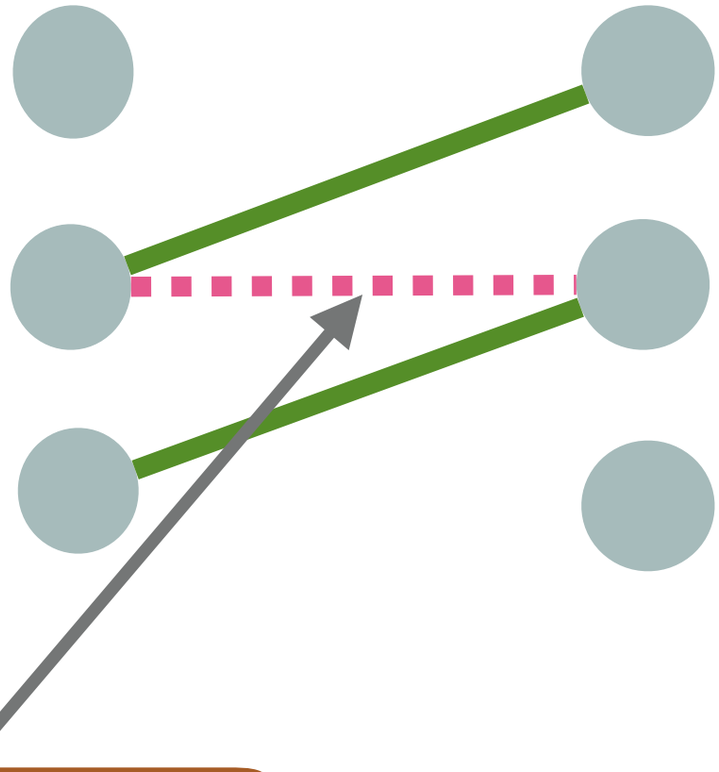
# 代码实现

```cpp
const int N = 1010; //left
const int M = 1010; //right
vector<int> edge[N];
bool visy[M];
int mx[N], my[M];

bool dfs(int s) {
    for (auto t: edge[s]) {
        if(!visy[t]) {
            visy[t] = true;
            if(my[t] == -1 || dfs(my[t])) {
                mx[s] = t;
                my[t] = s;
                return true;
            }
        }
    }
    return false;
}

int max_match(int n, int m) {
    fill(mx, mx + n, -1);
    fill(my, my + m, -1);
    int ret = 0;
    for (int i = 0; i < n; i++) {
        fill(visy, visy + m, false);
        if (dfs(i)) {
            ret++;
        }
    }
    return ret;
}
```

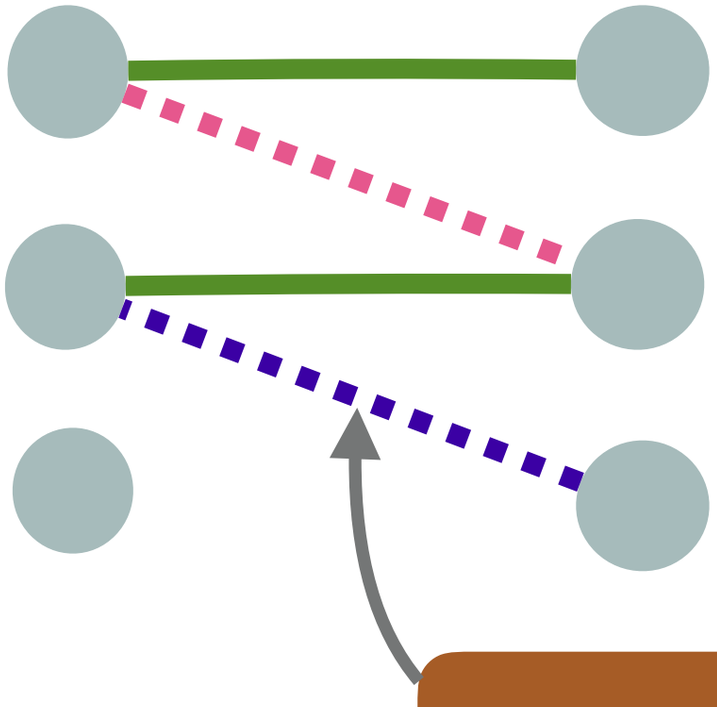
# 二分图最小点覆盖

# 最小点覆盖

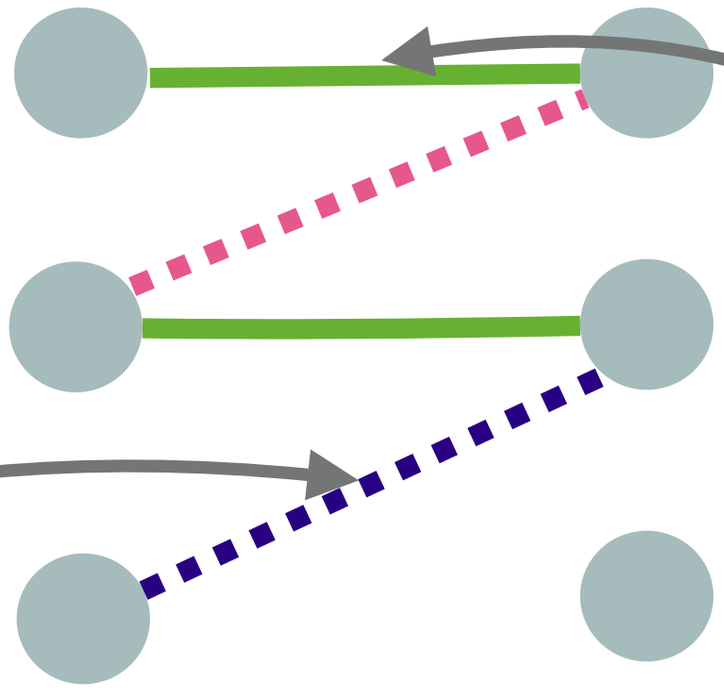
- 给你一个二分图，求在节点上最少安排几个守卫使得所有的边都能被监控到，一个节点可以监控所有经过这个点的边
- 我们不妨先跑一个最大匹配，然后我们可以将边分成四类
- 1：匹配边
- 2：左盖右未盖
- 3：左未盖右盖
- 4：左盖右盖



4号边：左盖右盖



2号边：左盖右未盖



1号边：匹配边

3号边：左未盖右盖

- 虽然此时的二分图已经不会再有增广路了，但是我们还可以充分利用增广路算法
- 3号边：从左边的未盖点出发跑增广路，能搜到的右边的已盖点，就是我们要安排守卫的地方，这些守卫能看守所有的左盖右未盖的边
- 2号边：从右边的未盖点出发跑增广路，能搜到的左边的已盖点，也是我们要安排守卫的地方，这些守卫能看守所有的左未盖右盖的边
- 显然，这两种路线搜索到的已盖点不会出现在同一条匹配边上，不然就会存在增广路，与最大匹配矛盾，
- 于是，第2号边与第3号边已经全部搞定

- 4号边：如果上述两种方法安排的守卫都没有覆盖到一些4号边，那么说明这些4号边无法连通到任何一个未盖点，也说明这些4号边两端的匹配点所在的匹配边的两端都还没有安排守卫
- 对于这种情况，我们只需要在某一侧安排守卫即可，即要么都在左侧，要么都在右侧
- 1号边：剩下的两侧都没有安排守卫的匹配边，也只需要在随便一侧安排即可

- 我们至少需要最大匹配的数量个点去覆盖所有的匹配边
- 现在我们找到了一种方法可以用最大匹配的数量个点去覆盖所有的边
- 因此最小点覆盖数等于最大匹配数
- 这也就是König定理

# 化简做法

- 上述过程有些繁琐，整合一下做法就是如下两类点
- 从任意一侧X的未盖点出发找增广路，选择增广路过程中搜到的对侧的所有的已盖点
- X侧所有的没搜到的点

# 二分图最小边覆盖

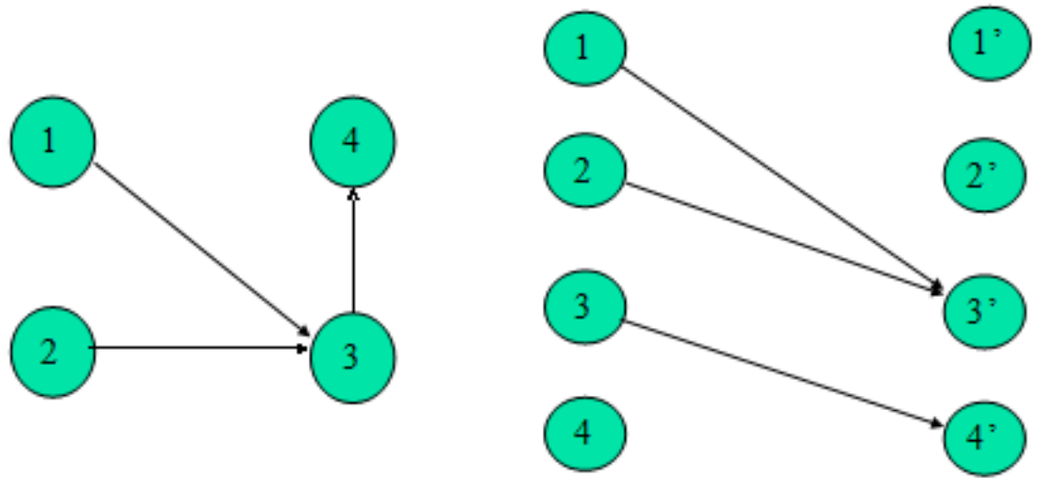
# 最小边覆盖

- 用最少的边覆盖所有的点
- 除了最大匹配所覆盖的点之外，其他点都需要一条边去覆盖，因此最小边覆盖等于 n - 最大匹配数

# 一般图最小路径覆盖

# 最小路径覆盖

- 给你一副有向图，选择最少的不相交的路径覆盖所有点
- 每条路径除了第一个点，与最后一个点，都有一个出点，一个入点，而且每个点只有一个入点，一个出点，这提示我们可以利用匹配的思想
- 将每个点拆成两个出点和 入点两个点，左边是出点，右边是入点，那么现在原图中的一条边u->v就会连接左部u的出点的到右部v的出点
- 一个匹配就对应了原图的一条边，每增加一匹配可以减少一条路径
- 因此最小路径覆盖数＝n-最大匹配数


1
4
2
3
1
2
3
4
1’
2’
3’
4’

# 二分图最大独立集

# 二分图最大独立集

- 独立集：一个点集，点集内的点两两不相邻
- 假设一开始选择了所有的点，那么每条边上都要去掉一个点才能达到目的
- 答案就是总点数减去最大匹配数
- 求方案可以不停的找一定在最大匹配上的点删除，最后剩下的点就是最大独立集

# 字典序问题

# POJ 3715

- 红方与蓝方要在一起玩游戏，但是已知红方于蓝方之间存在一些好朋友，游戏不允许好朋友同时存在两方，请问最少需要需要去掉几个人，输出字典序最小的方案。
- 经典的二分图最大独立集问题
- 字典序问题可以从小到大枚举匹配点删除，能删就删，如果一个点删除之后，最大匹配减少了，那么说明这个点可以删除
- 不必每次重新建图跑匈牙利
- 可以直接从匹配点对应的对侧的点出发跑增广路
- 此题暴力跑匈牙利也可以过，跑不进100ms不算ac

# 例題

# UVA-11419

- 有一个网格，里面有一些障碍，你可以在一些行和列放置大炮，一个大炮可以轰掉一行或者一列的所有障碍
- 问你最少需要多少大炮，输出任意一种方案
- 左边是行号，右边是列号，一个位于(x,y)的障碍可以抽象成一条边连接第x行和第y列，问题转换成最小点覆盖模型

# SRM 594 DIV1 T2

- 给你一个50*50的网格，网格里面有三种东西，"O X 。"，你可以在空地放X，如果一个O的周围的空地都被X放了，这个O就会变成空地，问你最多能得到多少空地?
- O与空地不可兼得
- 最大独立集！