# 更多图算法

网络最大流

二部图匹配

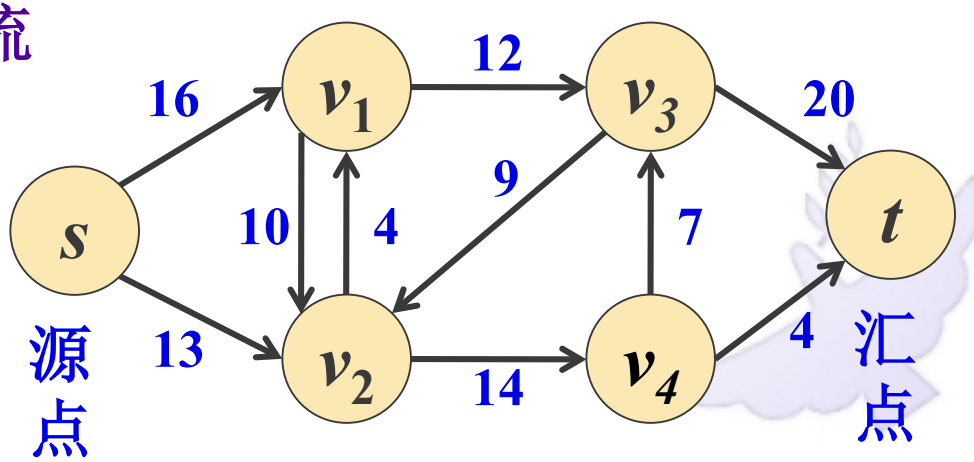
# 网络最大流

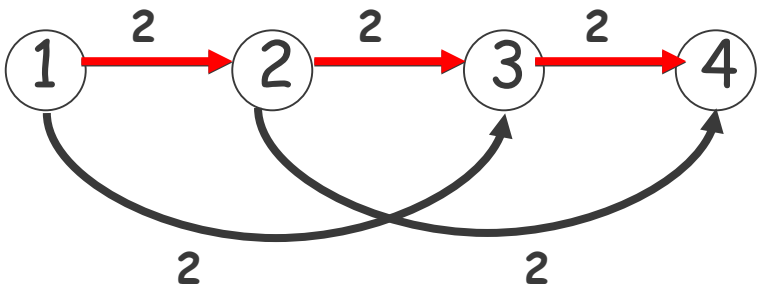
《算法导论》

《计算机算法设计与分析》

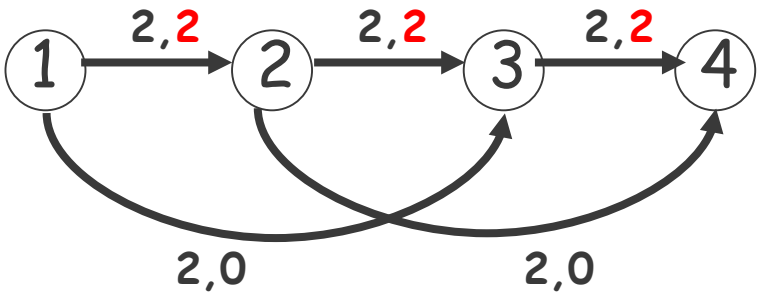
# 术语：流, 流量, 最大流

- 有向网络中，边(u,v)上的权值称为边的容量c(u,v)>=0。
- 假定每个顶点都处于源点到汇点的某条路径上。
- 称 $f$: V×V→R 为流, 若$f$ 满足：
  - (1) 容量限制, $f(u,v) \le c(u,v)$
  - (2) 反对称性, $f(u,v) = -f(v,u)$
  - (3) 流守恒性, 任意$u \in V\setminus\{s,t\}$, $\Sigma_{v\in V} f(u,v) = 0$，即除源点和汇点，对于每个顶点，进入流量和流出流量相同。
  - 称$f(u,v)$为顶点u到v的流
- 流量 $|f| = \Sigma_{v\in V} f(s,v)$.
- 最大流: 给定流网络 G,s,t,c, 求 max{ $|f|$ : $f$是G的流}

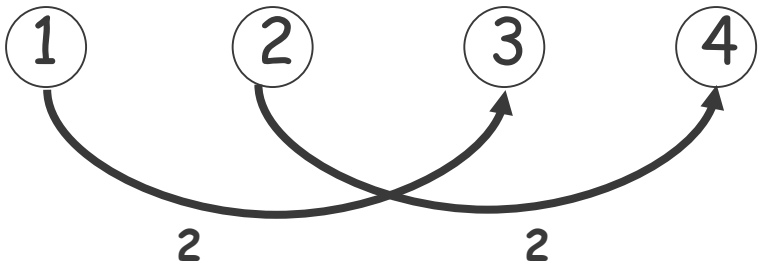




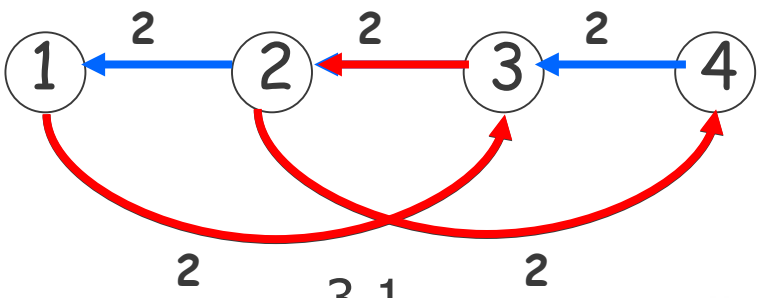

1-1



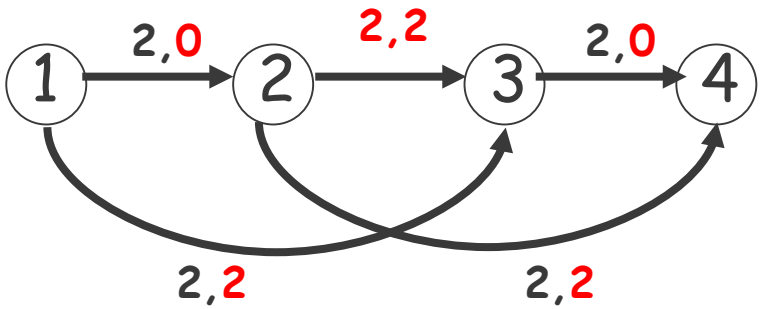

1-2



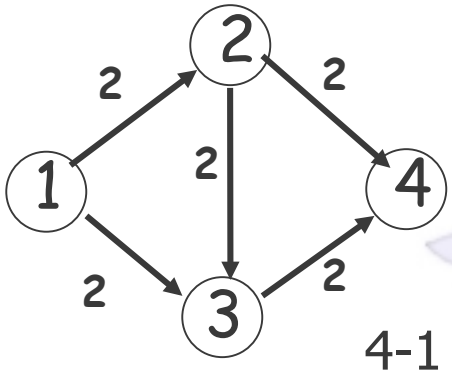

2-1




3-1




3-2

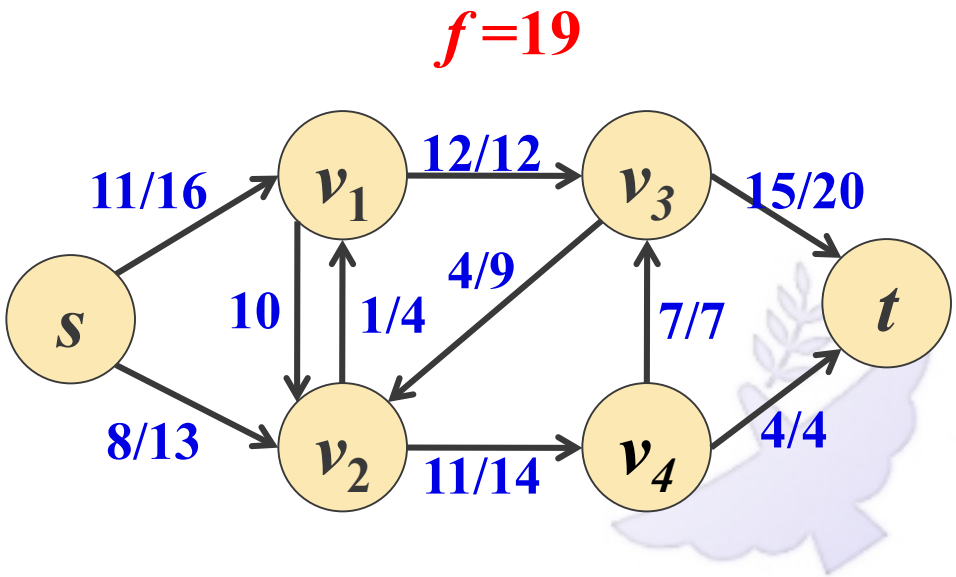



4-1

# Ford-Fulkerson方法

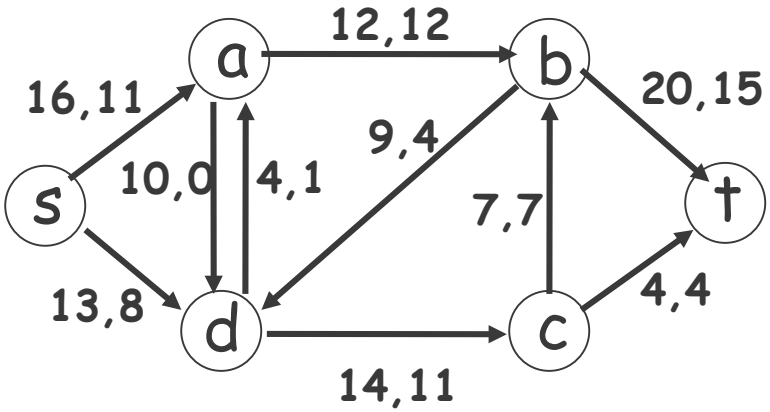
- **Ford-Fulkerson方法：**
  - 基本思想：迭代
  - 初始时，对所有u,v，$f(u,v)=0$;
  - 每次迭代，寻找增广路径增加流值。
  - 直到找到所有的增广路径

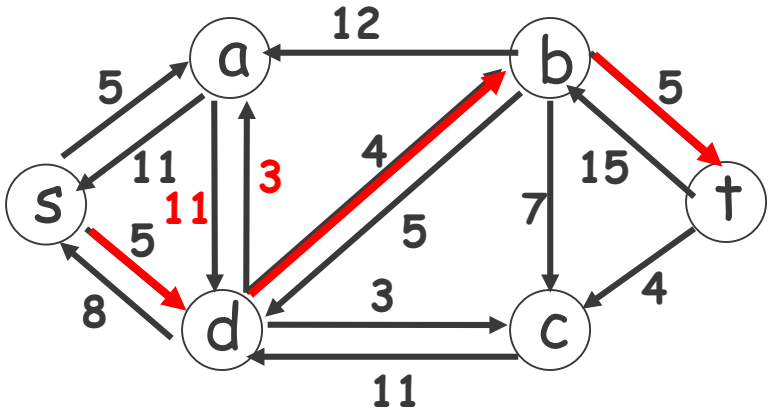
```
FORD-FULKERSON-METHORD(G, s, t)
1  initialize flow f to 0
2  while there exists an augmenting path p
3        do augment flow f along p
4  return f
```

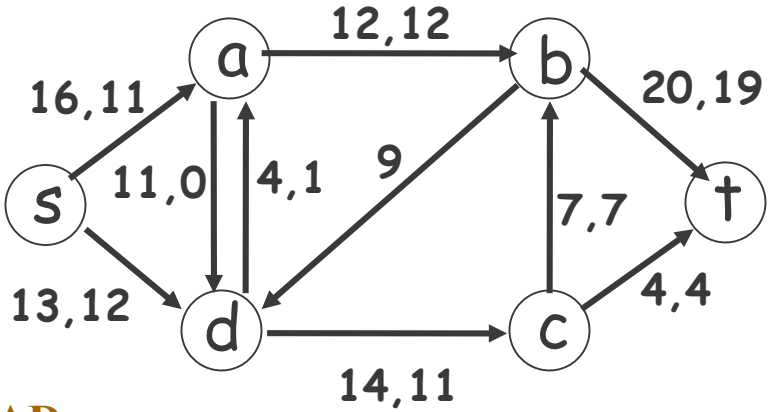


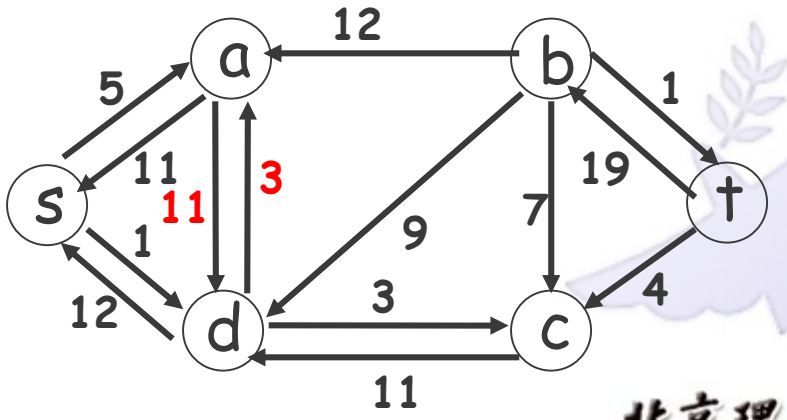
# Ford-Fulkerson方法

- 残留容量：$c_f(u,v) = c(u,v) - f(u,v)$.
- 残留网络：$G_f = \{V, E_f\}$，$E_f = \{(u,v) \in E, c_f(u,v) > 0\}$
- 增广路径：从s到t的路径，可以承载更多流量



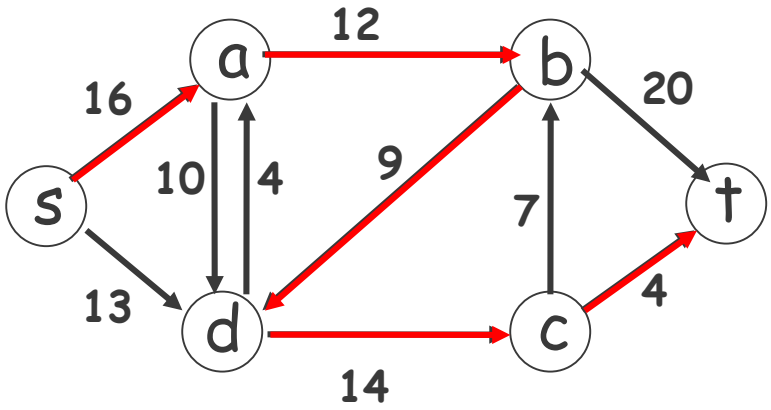



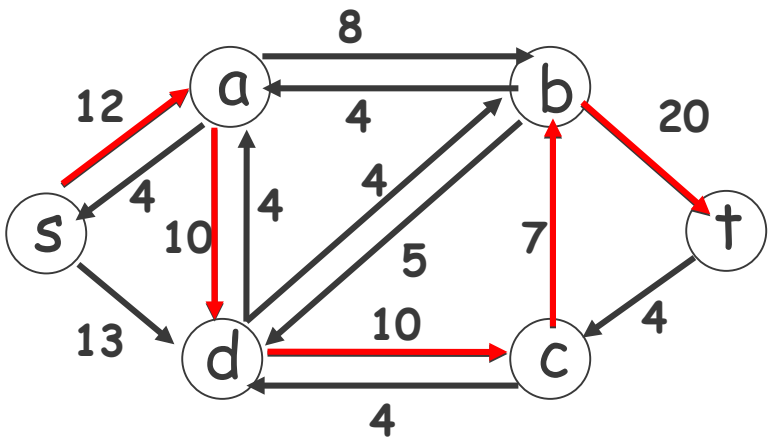



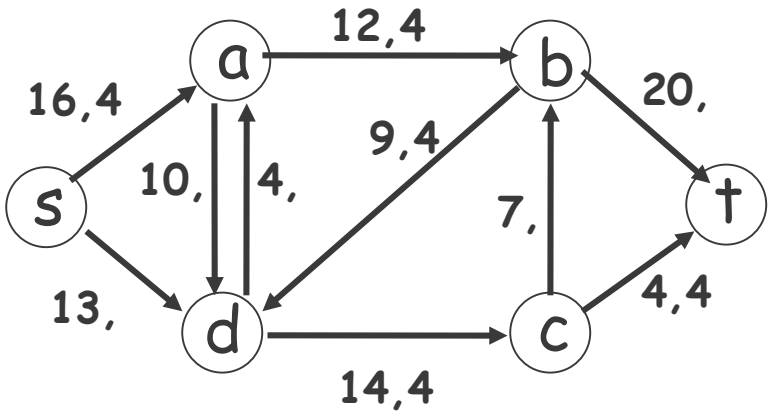
# Ford-Fulkerson方法

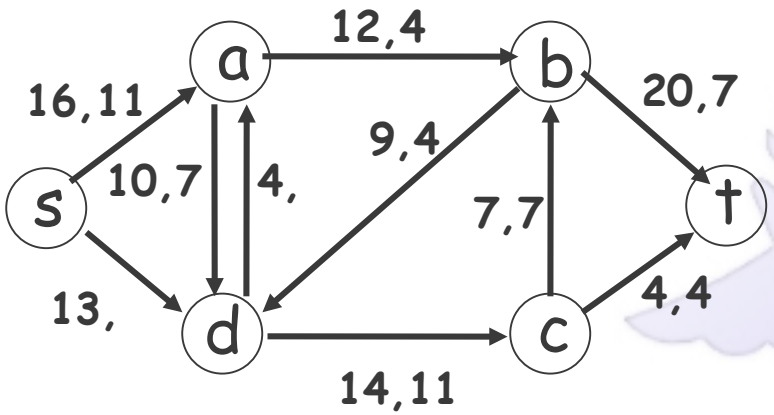
## 残留网络



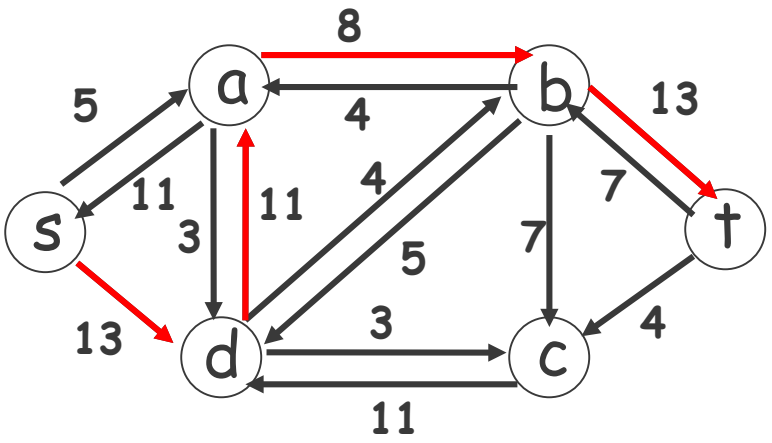

## 增广之后的新流



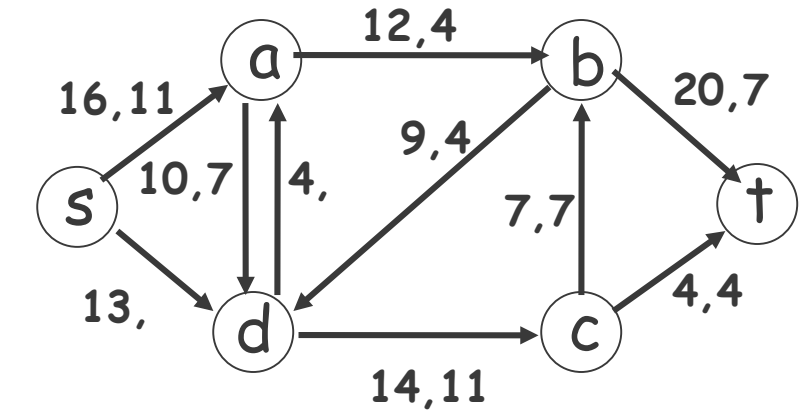



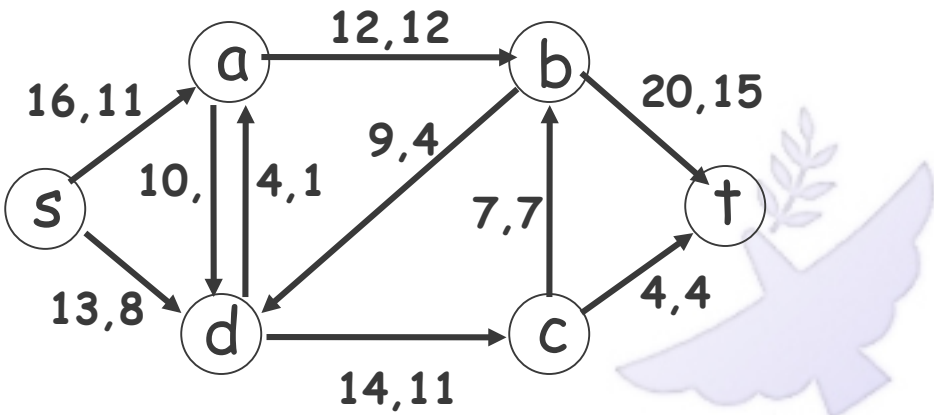
# Ford-Fulkerson方法

## 残留网络

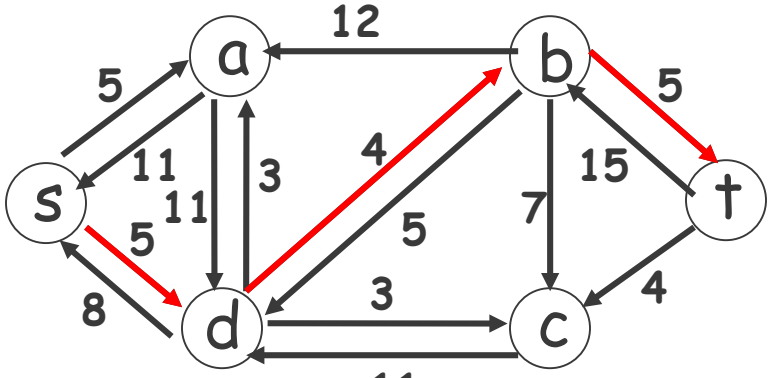


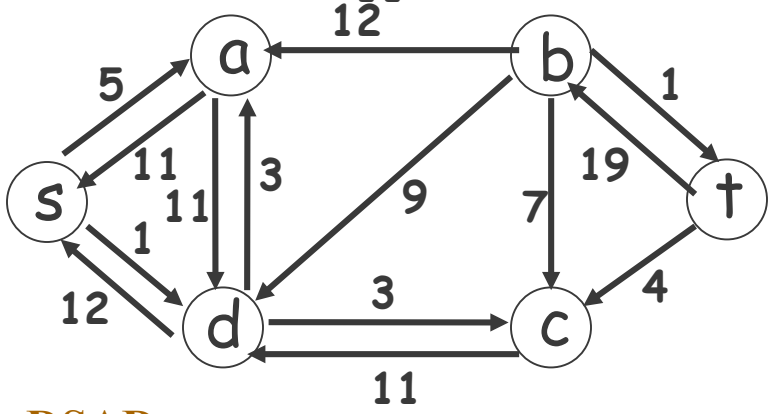

## 增广之后的新流



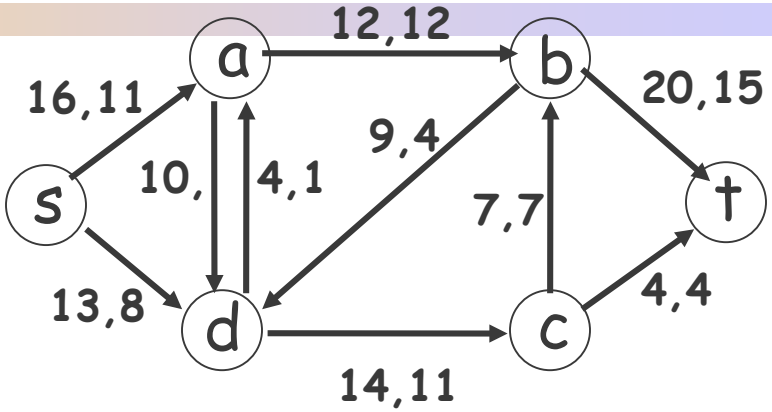



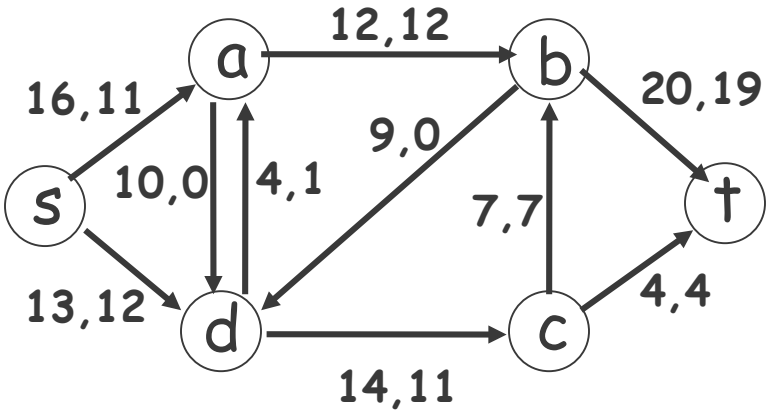
# 残留网络



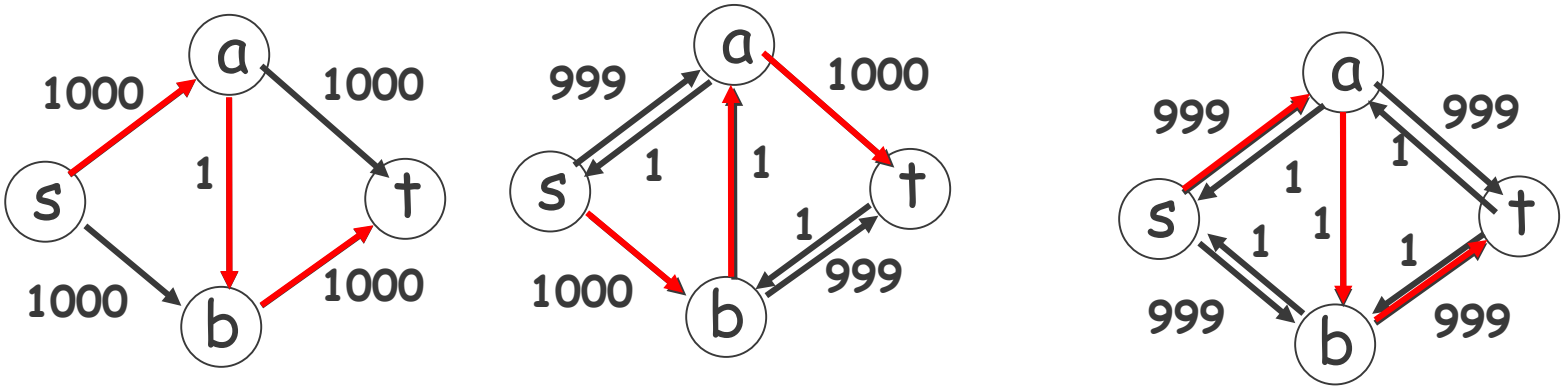

# 增广之后的新流



# Ford-Fulkerson方法

迭代中应该选择哪一条增广路径？




问题：增广路径不同的选择顺序使得算法效率产生很多的差异

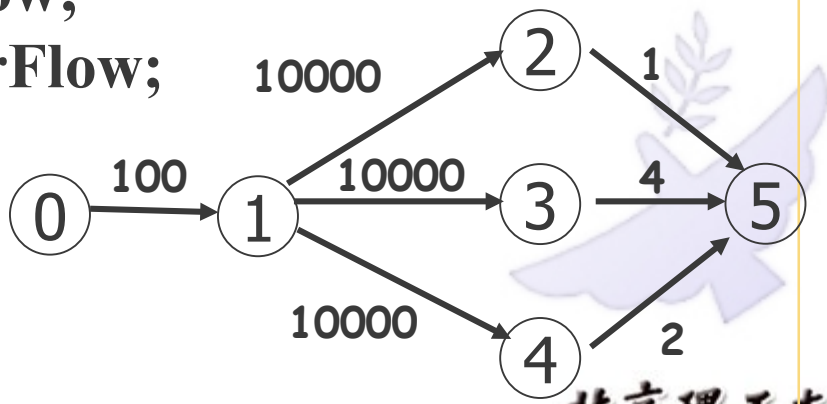
解决方法：Edmonds-Karp算法

在剩余图中找最短增广路径

最短路径：BFS

# EK算法核心代码

```
bool BFS(int src, int dst);//计算残留网络的最短路径
int EK(int src,int dst) {
    int i, totalFlow = 0, curFlow;    if (src == dst) return INF;
    while (BFS(src, dst)) {//计算最短路径
        curFlow = INF;
        for (i = dst; i != src; i = pre[i]) //计算流量
              curFlow= min( curFlow, edge[path[i]].flow);
        totalFlow += curFlow ;
        for (i = dst; i != src; i = pre[i]) {
            edge[path[i]].flow -= curFlow;
            edge[path[i]^1].flow += curFlow;
        }
    }
    return totalFlow;
}
```



- 设$f$是一个流，$G_f = \{V, E_f\}$是由f导出的残留网络；设$f'$是$G_f$中的一个流，则

$$|f + f'| = |f| + |f'|$$

- 由此上述算法中网络的流是递增的。

# 流网络的割

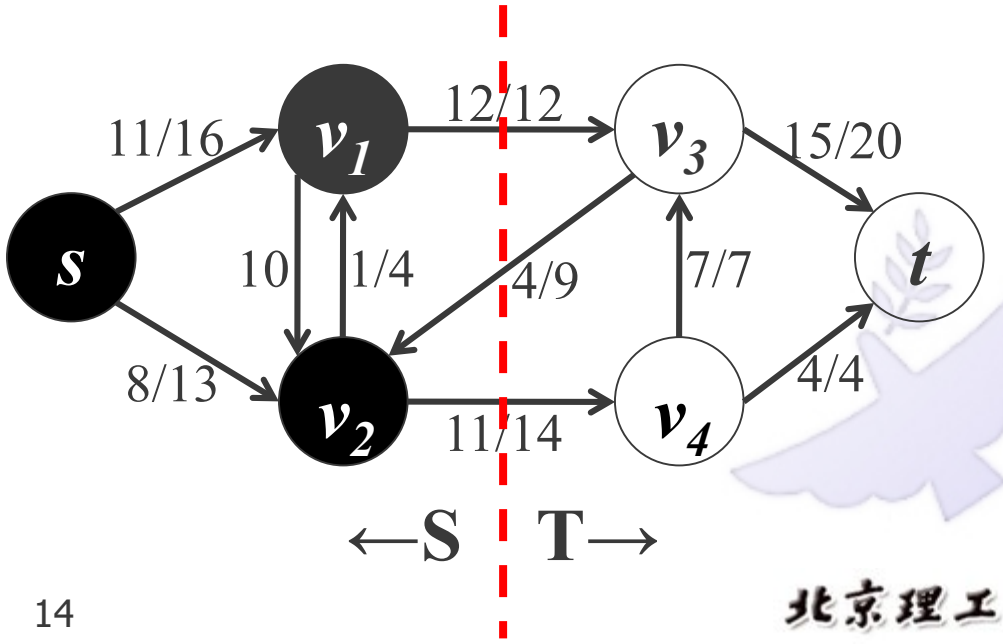
- 割(S,T)： (1) T=V-S (2) s∈S, t∈T.
- 割(S,T)的容量：$c(S,T)=\Sigma_{u\in S,v\in T}c(u,v)$
- $f$穿过割(S,T)的净流：$f(S,T)=\Sigma_{u\in S,v\in T}f(u,v)-\Sigma_{u\in T,v\in S}f(u,v)$

- 流穿过任意割的净流量都相同，都等于流的值。
- $c(S,T)\geq f(S,T)$，任意割的容量大于等于网络流$f$穿过割的净流

- 例.图中的割(S,T),
- $S=\{s,v_1,v_2\}, T=\{v_3,v_4,t\}$.

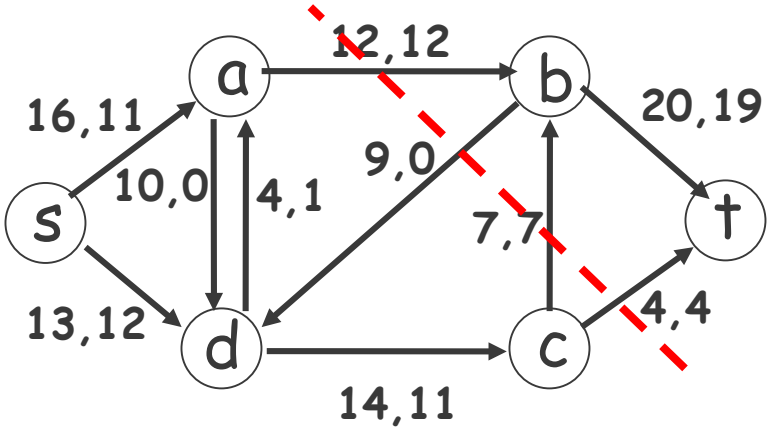
$c(S,T)=12+14=26.$

$f(S,T)=12+11-4=19$



# 最大流最小割定理(算法证明)

- 定理(最大流最小割) 下列条件等价
  - (1) $f$是$G$的一个最大流;
  - (2) $G_f$不包含增广路径;
  - (3) 存在割$(S,T)$使得 $|f| = c(S,T)$.
- 使得$|f| = c(S,T)$的割为最小割。



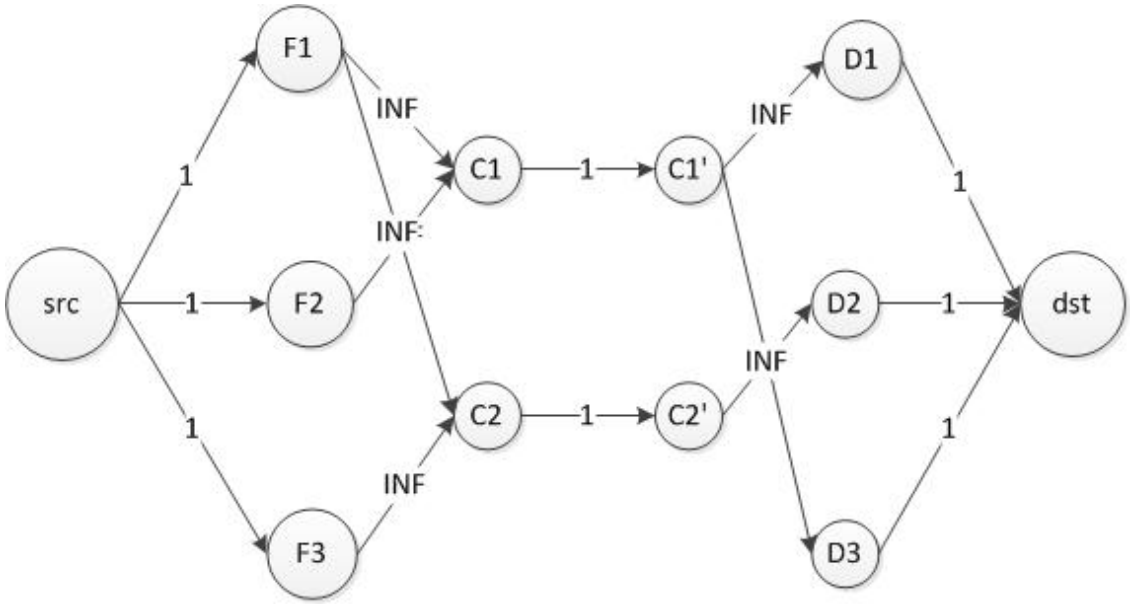

- 最大流算法基本思想:
  找从s到t的增广路径, 增大流, 无则停止, 得最大流

# 最大流题目选讲

- **1. Dinning**
- **Farmer John的牛饿了，到了午餐时间每头牛需要一份食物和一份饮料.**
- **现在已知有n头牛，a种食物和b种饮料，每种食物和饮料只有一份。每头牛都有一些食物和饮料是喜欢的，牛们只会吃喜欢的食物和喜欢的饮料，**
- **现在请问最多有多少头牛能够吃到喜欢的食物和饮料。**

# 1. Dinning

- 增加源点src与汇点dst，src到每种食物连一条容量为1的边，保证每种食物只用一次，同理每种饮料到汇点连一条容量为1的边，保证每种饮料只用一次。
- 将每头牛拆成两个点，中间连一条容量为1的边，保证每头牛只用一次，每种食物到喜欢他的牛左侧的点连容量为INF的边，每头牛右侧的点到每种饮料连容量为INF的边，求最大流即可。

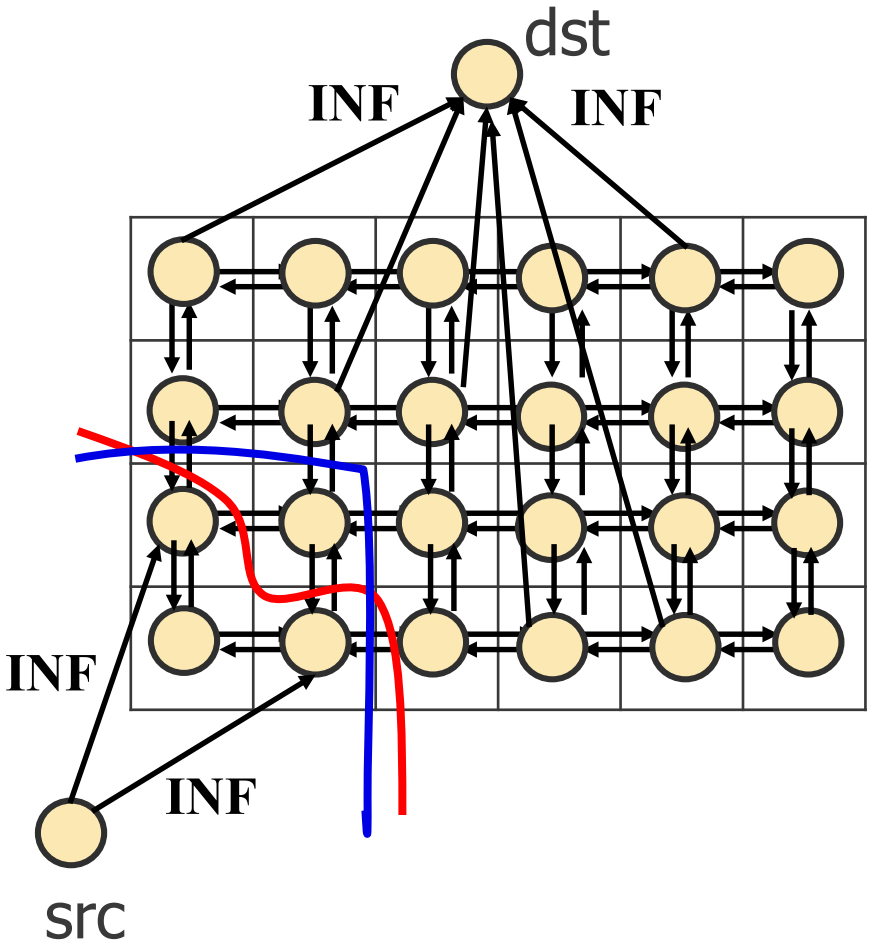


# 最大流题目选讲

- 2. 喜羊羊与灰太狼
- 在一个M*N的方格形地图上，有一些格子中有羊，有一些格子中有狼，还有一些格子是空地。
- 每个格子内最多只能有一只狼或羊。
- 要沿着格子边沿修建围墙，请问最少修建多长的围墙能够将狼和羊隔开。

# 最大流题目选讲

- 实际上是一个多源汇最小割问题
- 把地图中每个方格作为网络中的一个节点，任意相邻两个方格建立双向容量为1的边。
- 假设有狼的格子为源点，有羊的格子为汇点，显然只要有从任意源点到达任意汇点的一个流，则说明狼可以通过这条路径吃到羊，所以只要割断这个图就可以隔开狼和羊，而每割断一条边正好对应修建一段墙，原问题也就成功转化为多源汇最小割问题。
- 为解决问题，我们添加超级源点src和超级汇点dst，src到每个狼的格子有容量为INF的边，每个羊的格子到dst有容量为INF的边，在此网络中由最大流最小割定理，求出最大流即为答案


dst
INF
INF
INF
INF
src

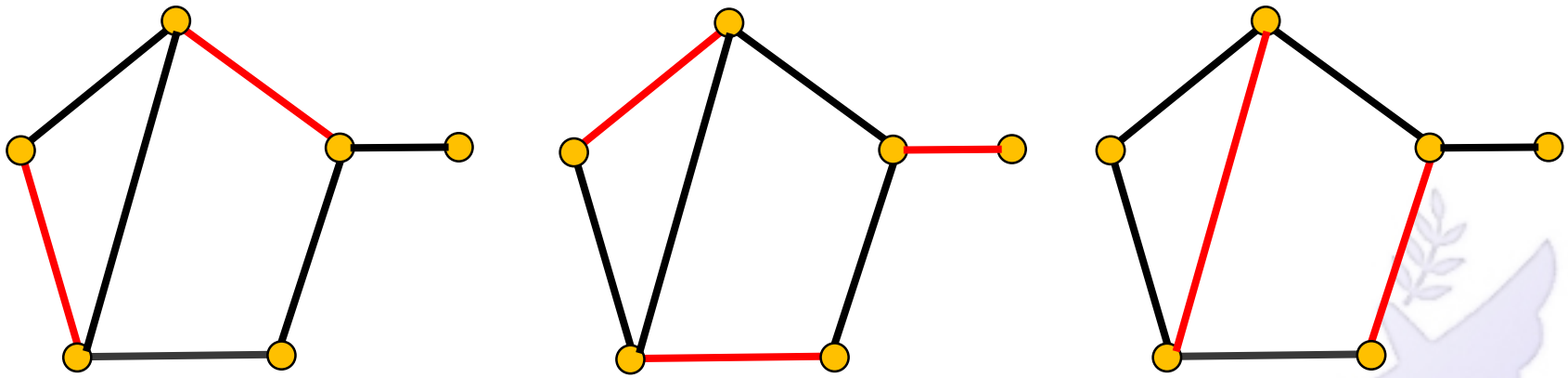

Answer = 4

# 二部图中的匹配

匈牙利算法

# 匹配与匹配数

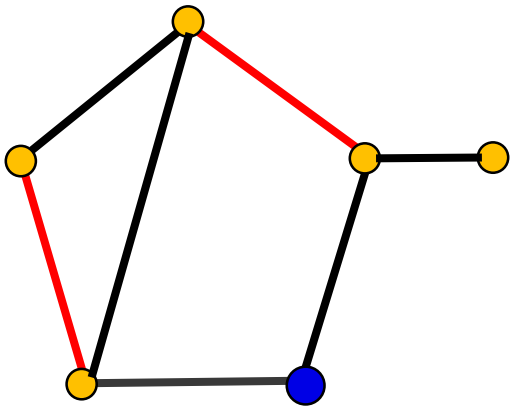
- 定义18.5 设无向简单图 $G=<V,E>$, $E^* \subseteq E$,
- (1) 匹配(边独立集)$E^*$——$E^*$中任意两条边均不相邻.
- (2) 极大匹配$E^*$——$E^*$中再加任意一条边后都不是匹配.
- (3) 最大匹配——边数最多的匹配.
- (4) 匹配数(边独立数)$\beta_1$——最大匹配中的边数或者匹配数.



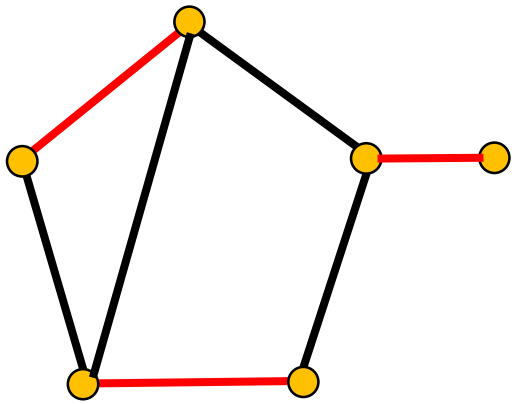
匹配数$\beta_1=3$

# 关于匹配中的其它概念

- 定义18.6 设$M$为$G$中一个匹配.
- (1) 匹配边——$M$中的边.
- (2) 非匹配边——不在$M$中的边.
- (3) 饱和点——与$M$中边与相关联的顶点.
- (4) 非饱和点——不与$M$中边与相关联的顶点.
- (5) 完美匹配——$G$中每个顶点都是饱和点.



非饱和点

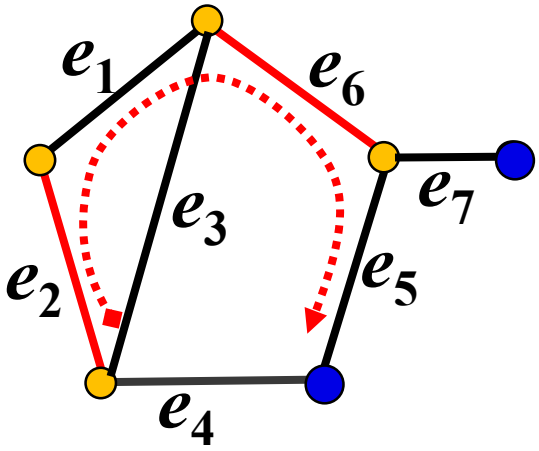


完美匹配

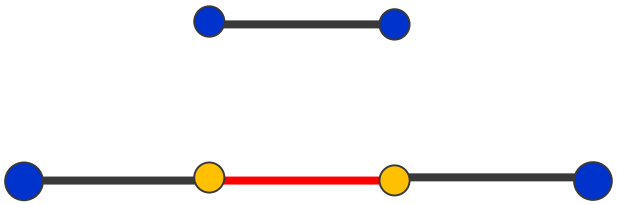
最小边覆盖

# 关于匹配中的其它概念

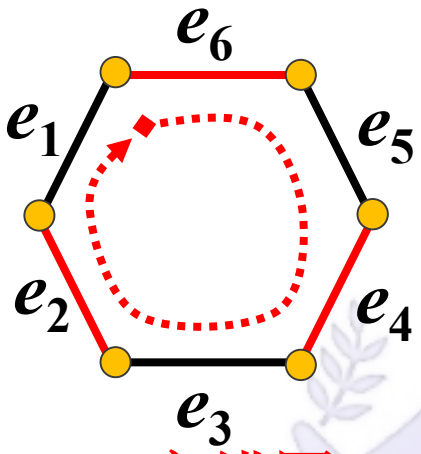
- 定义18.6 设$M$为$G$中一个匹配.
- (6) 交错路径——$G$中由匹配边和非匹配边交替构成的路径.
- (7) 可增广交错路径——起点和终点都是非饱和点的交错路径.
- (8) 交错圈——$G$中由匹配边和非匹配边交替构成的圈.




交错路径



可增广交错路径
增广路径

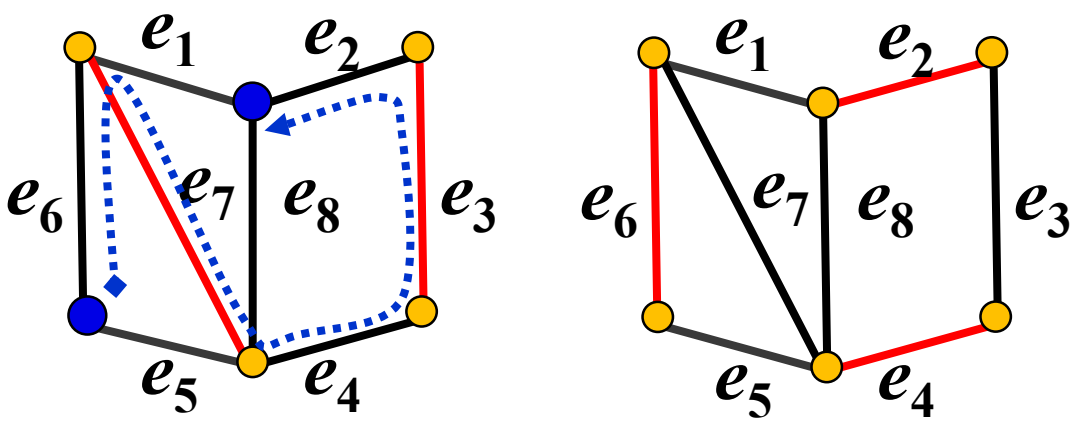


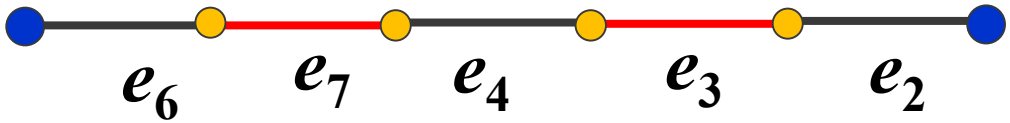

交错圈

说明(1)可增广交错路径中，不在$M$中的边比在$M$中的边多一条.
(2)交错圈一定为偶圈; 可增广交错路径长度为奇数

# 最大匹配判别定理

- **定理18.4** *M*为*G*中最大匹配当且仅当*G*中不含*M*的可增广交错路径.

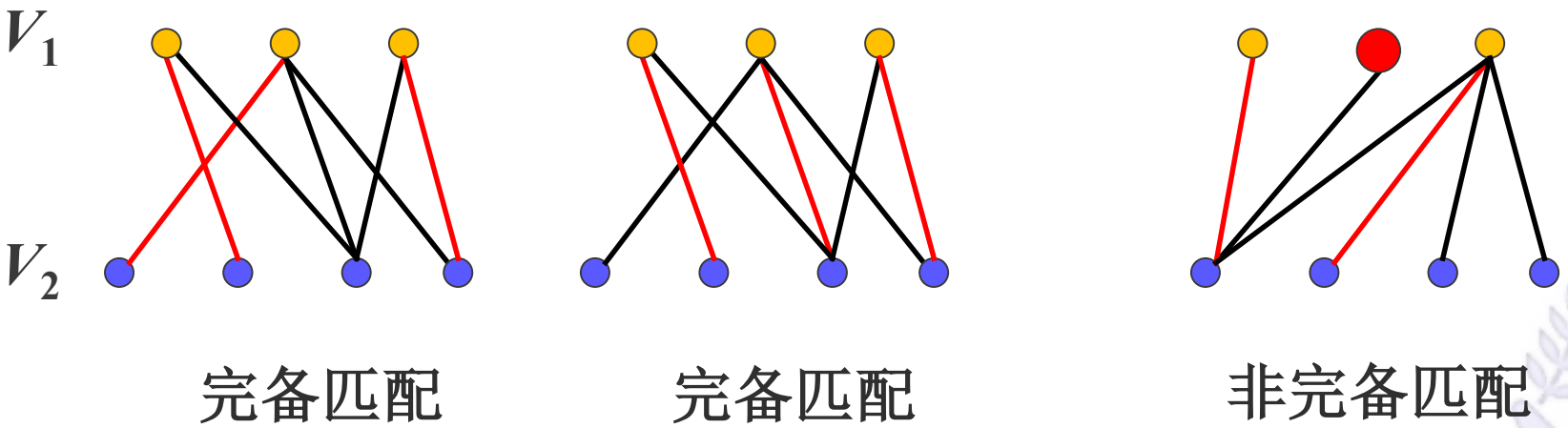


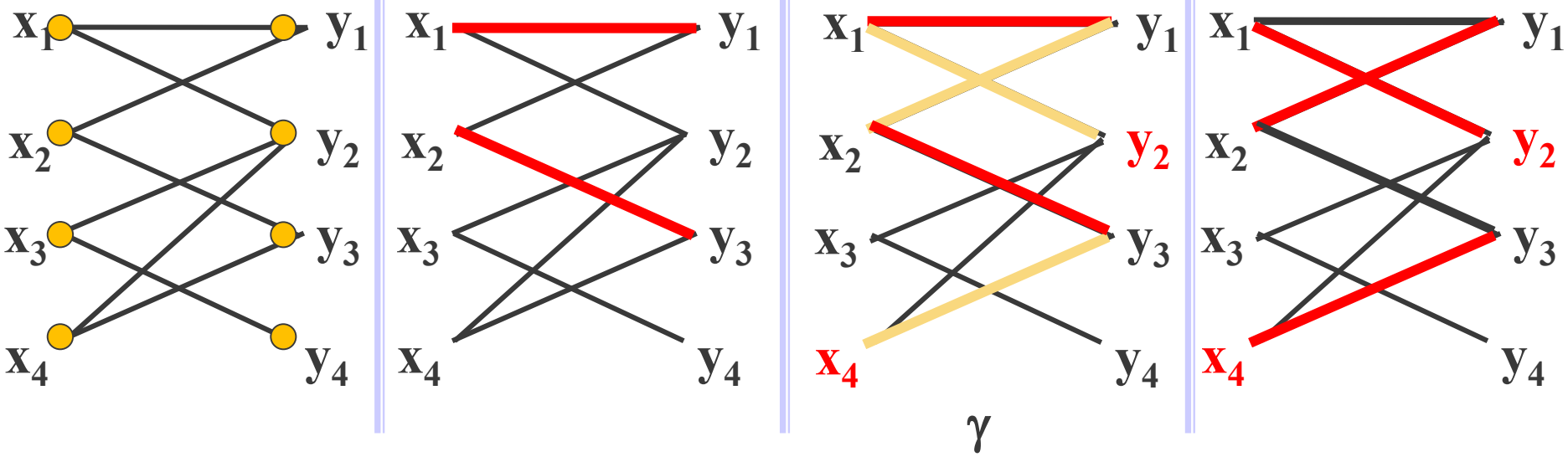

可增广交错路径

# 二部图中的匹配

- 定义18.7 设$G=<V_1,V_2,E>$为二部图，$|V_1|\le|V_2|$，$M$是$G$中最大匹配，若$V_1$中顶点全是$M$饱和点，则称$M$为$G$中完备匹配.
- $|V_1|=|V_2|$ 时完备匹配变成完美匹配.



# 二部图中的增广路径




匹配M={{$x_1$,$y_1$}, {$x_2$,$y_3$}}有一条增广路径γ

由M增广路径可构造比M大的匹配

存在M增广路径⇒M非最大匹配

用(M-γ)∪(γ -M)代替M

- **定理18.4** *M*为*G*中最大匹配当且仅当*G*中不含*M*的可增广交错路径.

# 增广路径的性质

- 由增广路径的定义可以推出下述三个结论：

1、γ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、γ经过取对称差操作可以得到一个更大的匹配M′。

3、M为G的最大匹配当且仅当不存在关于M的增广路径。

# 二部图中的匹配

- 用增广路径求最大匹配(称作匈牙利算法)
- 算法：
  - (1)置匹配M为空
  - (2)找出一条增广路径γ，通过取对称差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

1955年，库恩(W.W.Kuhn)利用匈牙利数学家康尼格(D.Kőnig)的一个定理构造了这个解法，故称为匈牙利法。

# 匈牙利算法核心代码

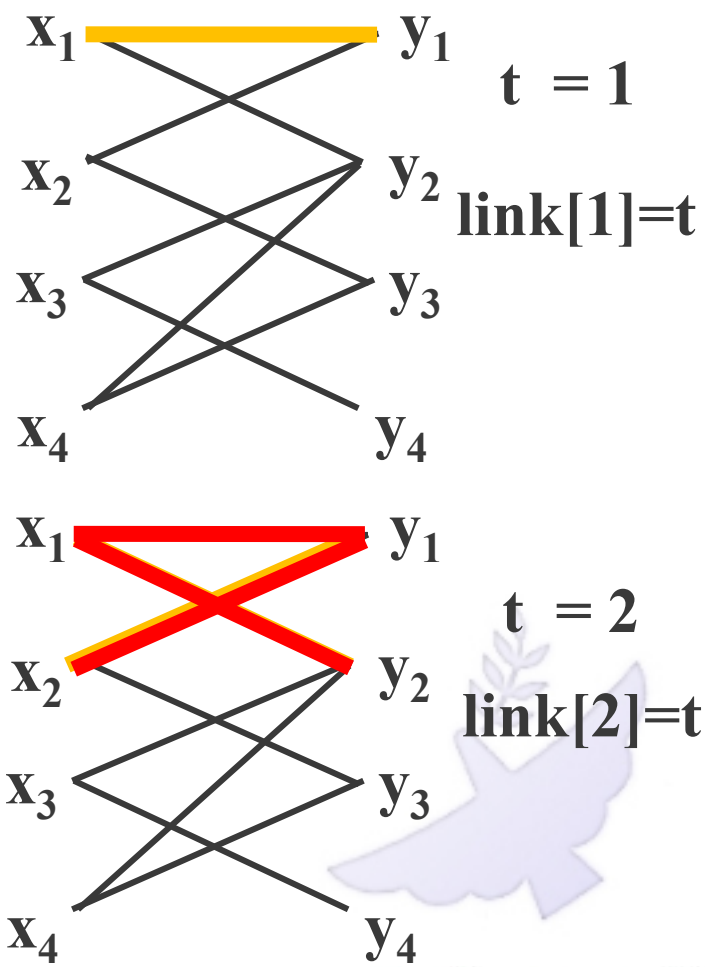
```
#define N 202
int useif[N];   //记录y中结点是否使用
int link[N];   //记录当前与y结点相连的x的结点
int mat[N][N]; //记录x和y的边,若i和j之间有边则为1,否则为0
int gn,gm;    //二部图中x和y中点的数目
```

```
int MaxMatch(){
    int i,num;     num=0;
    memset(link,0xff,sizeof(link)); //link全置-1
    for(i=1;i<=gn;i++) { //扫描X中的节点,是否存在匹配
       memset(useif,0,sizeof(useif));//useif全置0
       if(can(i)) num++; //如果有增广路径，则匹配数加一
    }
    return num;
}
```

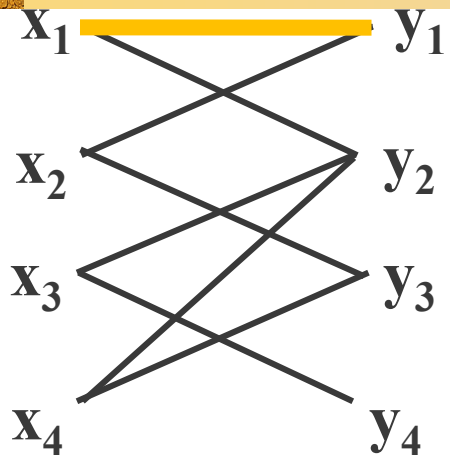
# 匈牙利算法核心代码

```
int useif[N];   //记录y中结点是否使用
int link[N];   //记录当前与y结点相连的x的结点
int mat[N][N]; //记录匹配中的边
int gn,gm;    //二部图中x和y中点的数目
```

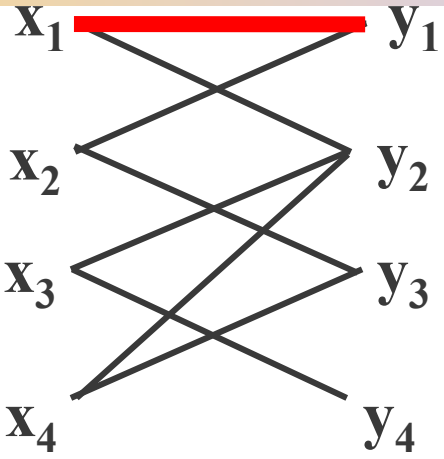
```
int can(int t){ //是否有增广路径
    for(i=1;i<=gm;i++) {//检查Y的点
       if(useif[i]==0 && mat[t][i]){
          useif[i]=1;
          if(link[i]==-1 || can(link[i]))  {
               link[i]=t;  return 1; }
       }
    }
    return 0;
}
```

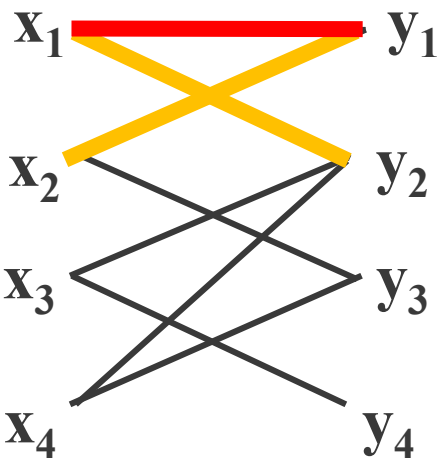


# 匈牙利算法演示

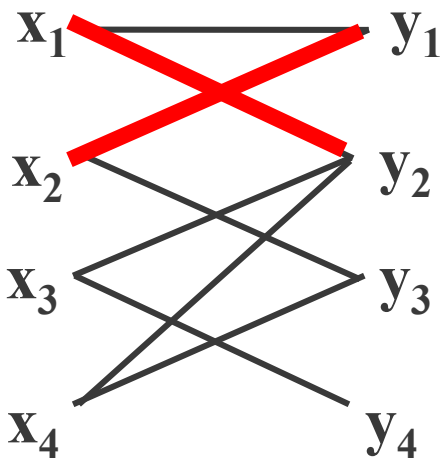



t = 1



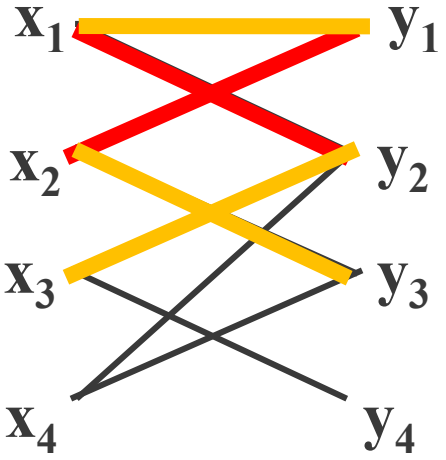




t = 2

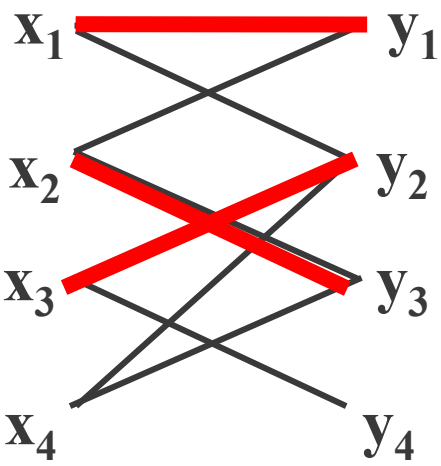


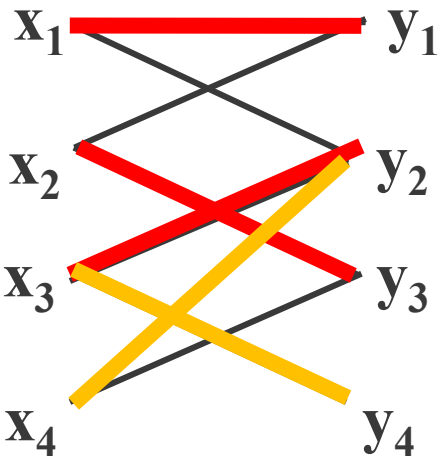

t = 2

# 匈牙利算法演示

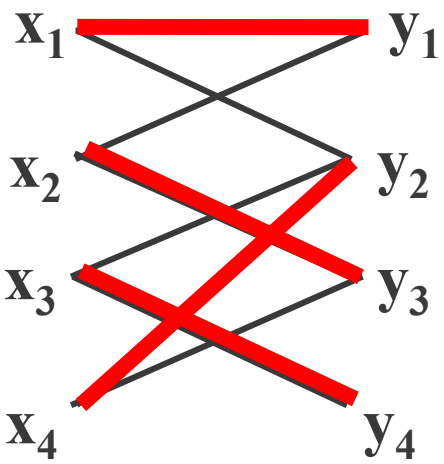



t = 3

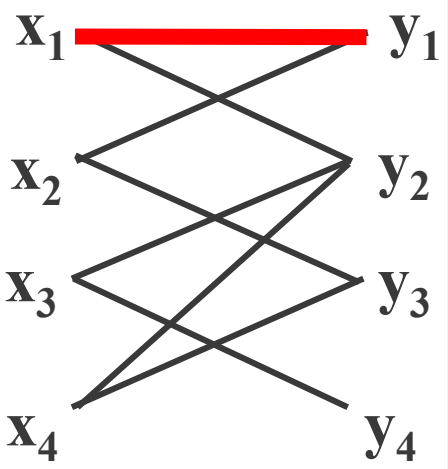


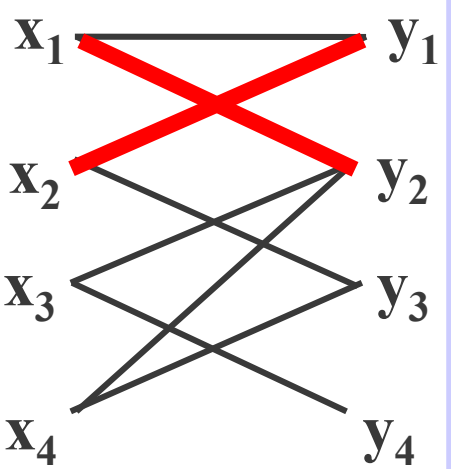

t = 4



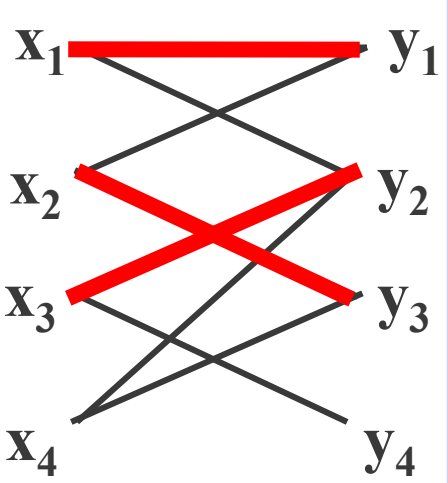
# 匈牙利算法演示



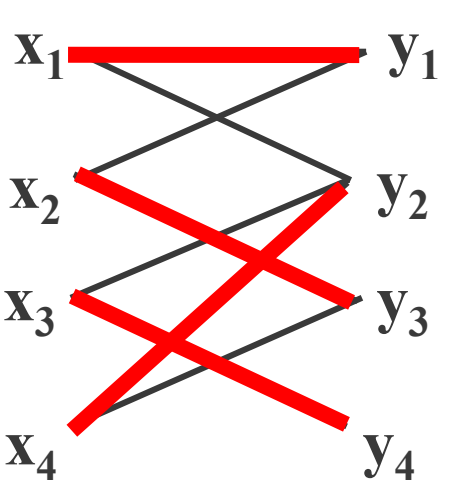

t = 1




t = 2




t = 3



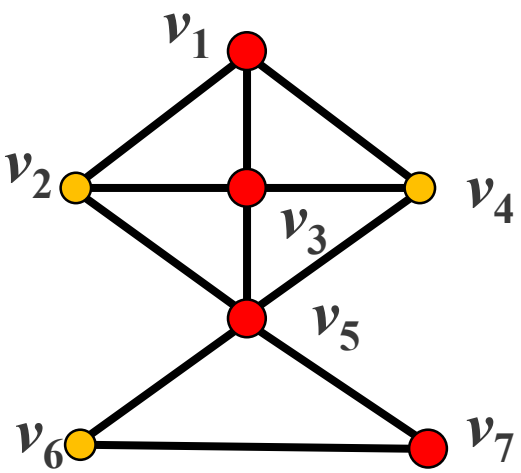

t = 4

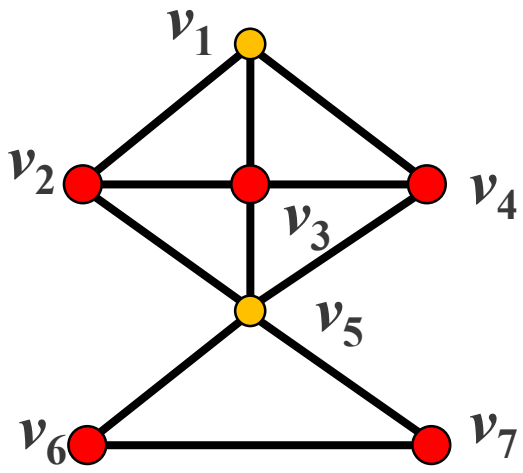
# 点覆盖集与点覆盖数

点到边的关系

- 设 无向简单图 $G=<V,E>$, $V^* \subseteq V$.
- (1) $V^*$是点覆盖集——$\forall e \in E$，$\exists v \in V^*$，使$e$与$v$关联
- (2) $V^*$是极小点覆盖集——$V^*$的任何真子集都不是点覆盖集
- (3) 最小点覆盖集(或最小点覆盖)——顶点数最少的点覆盖集
- (4) 点覆盖数$\alpha_0(G)$——最小点覆盖的元素个数

点集$V^*$能关联(覆盖)到图中所有其它边



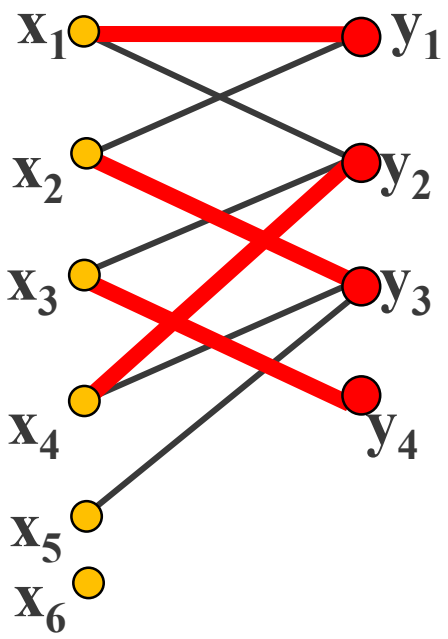

$\alpha_0(G)=4$

# 二部图的最小点覆盖

**定理: $G$是二部图, 则$\alpha_0 = \beta_1$.**

**即二部图的最小点覆盖数 = 其最大匹配数**

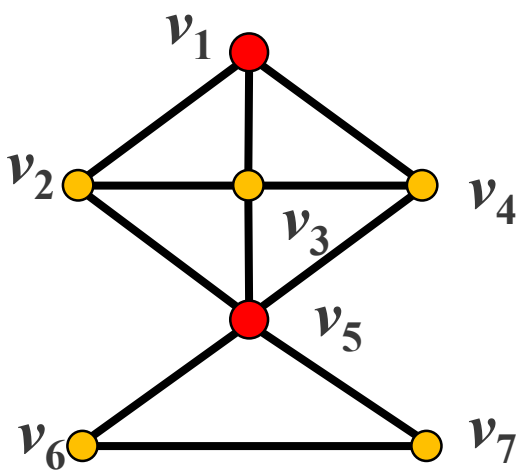
注: 定理对一般图不成立.

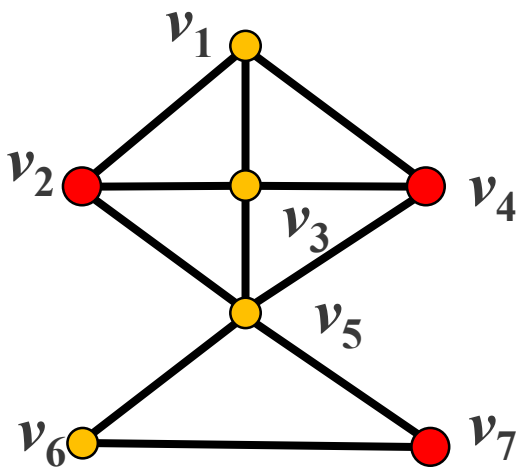


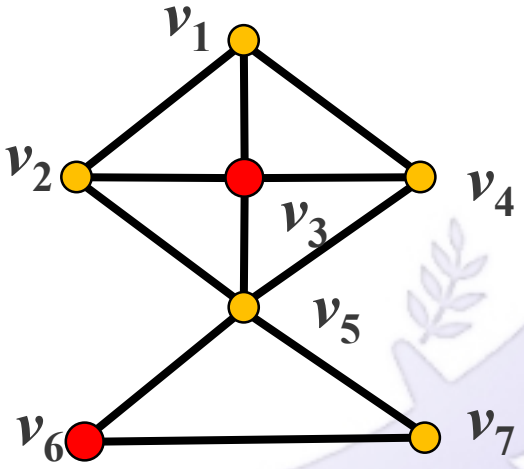
# 点独立集与点独立数



- 定义18.2 设无向简单图 $G=<V,E>$，$V^* \subseteq V$.
- (1) 点独立集$V^*$——$V^*$中顶点彼此不相邻
- (2) $V^*$为极大点独立集——$V^*$中加入任何顶点就不是点独立集
- (3) 最大点独立集——元素最多的点独立集
- (4) 点独立数$\beta_0(G)$——最大点独立集中的元素个数.



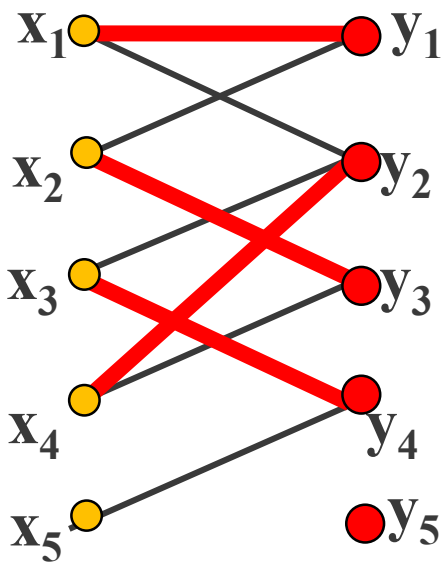

$\beta_0(G)=3$

# 二部图的最大独立集

- 定理：二部图中：

**最大独立集数 = 顶点总数 – 最小点覆盖数**

**也 = 顶点总数 – 最大匹配数**



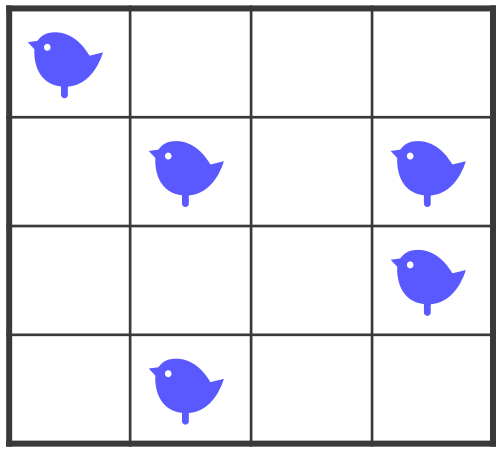
# 二部图匹配的实例：打猎

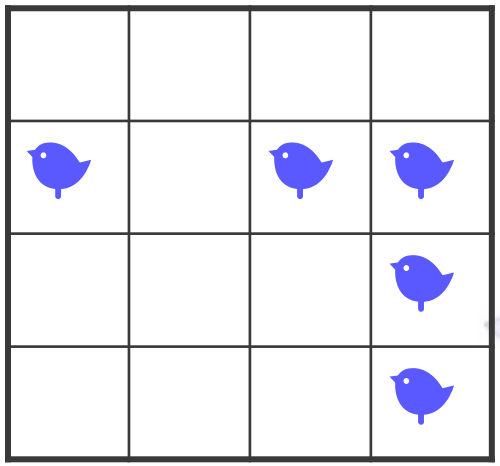
- 猎人要在n*n的格子里打鸟,他可以在某一行中打一枪,这样此行中的所有鸟都被打掉，也可以在某一列中打,这样此列中的所有鸟都打掉.问至少打几枪,才能打光所有的鸟？

解：

建图：二部图的X部为每一行，Y部为每一列，如果(i,j)有一只鸟，那么连接X部的i与Y部的j。

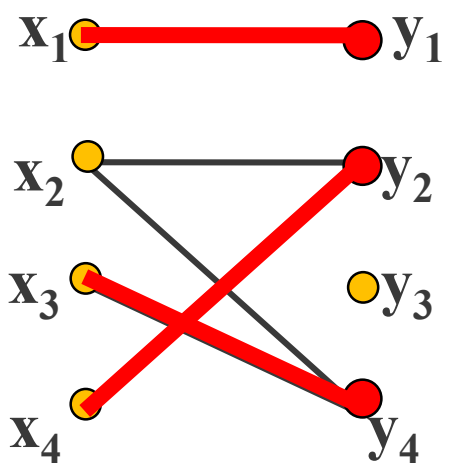
该二部图**最小点覆盖数**即是最少要打的枪数。

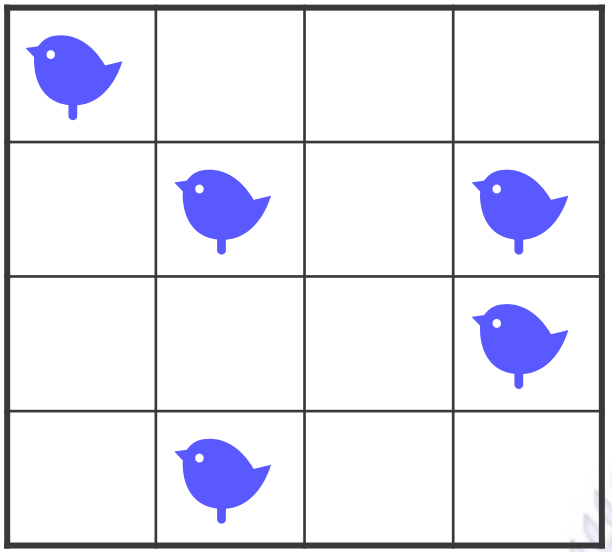




# 二部图匹配的实例：打猎

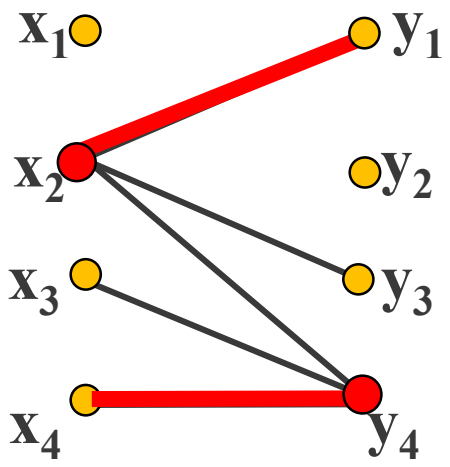
- 猎人要在n*n的格子里打鸟,他可以在某一行中打一枪,这样此行中的所有鸟都被打掉，也可以在某一列中打,这样此列中的所有鸟都打掉.问至少打几枪,才能打光所有的鸟？

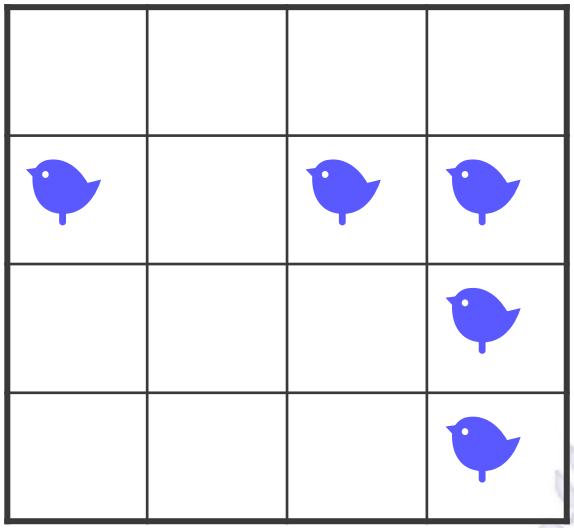


二部图的最大匹配数=最小点覆盖数=3

- 猎人要在n*n的格子里打鸟,他可以在某一行中打一枪,这样此行中的所有鸟都被打掉，也可以在某一列中打,这样此列中的所有鸟都打掉.问至少打几枪,才能打光所有的鸟？



二部图的最大匹配数=最小点覆盖数=2

# Girls and Boys

The second year of the university somebody started a study on the romantic relations between the students. The relation "romantically involved" is defined between one girl and one boy. For the study reasons it is necessary
to find out the maximum set satisfying the condition: there are no two students in the set who have been "romantically involved". The result of the program is the number of students in such a set.
The input contains several data sets in text format. Each data set represents one set of subjects of the study,
with the following description:
the number of students
the description of each student, in the following format

# Girls and Boys

student_identifier:(number_of_romantic_relations)
student_identifier1 student_identifier2 ...

or

student_identifier:(0)

The student_identifier is an integer number between 0 and n-1, for n subjects.

For each given data set, the program should write to standard output a line containing the result.

# Girls and Boys

Sample Input

7

0: (3) 4 5 6

1: (2) 4 6

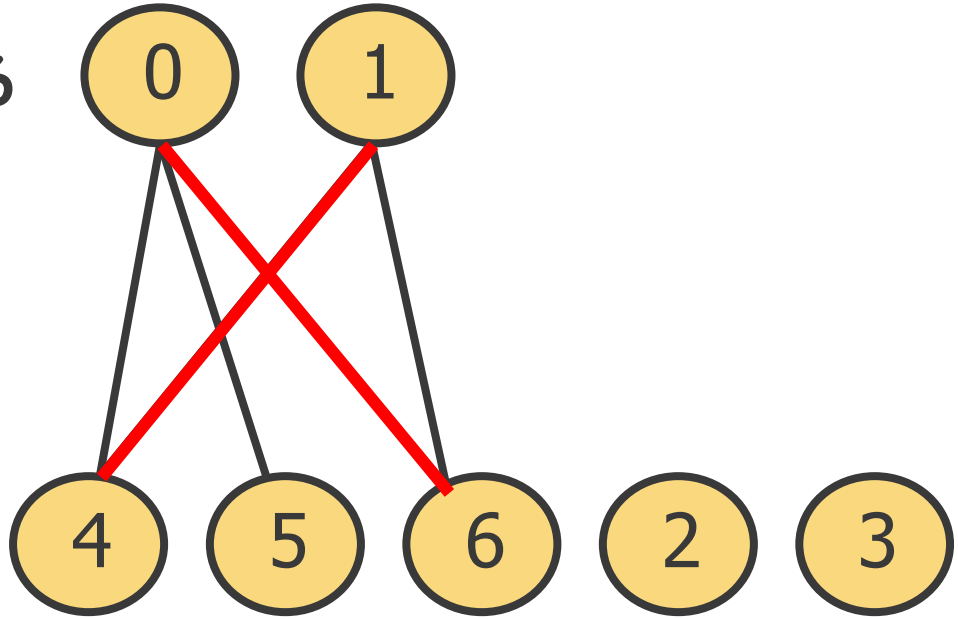
2: (0)

3: (0)

4: (2) 0 1

5: (1) 0

6: (2) 0 1




Output

5

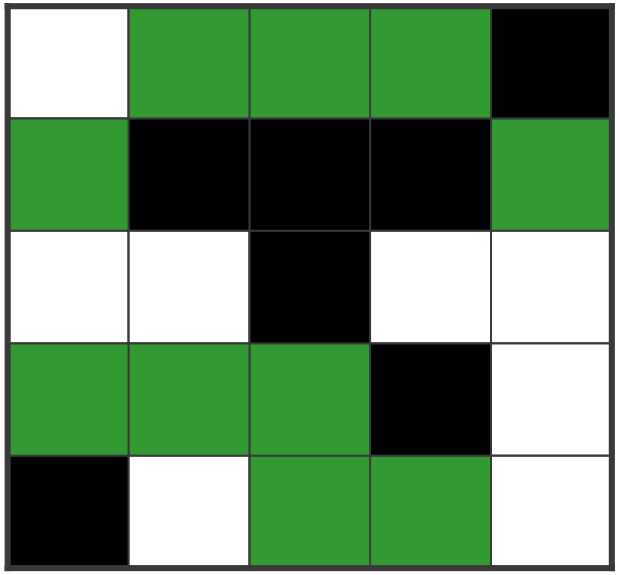
题意 在一群有恋爱关系的男女学生中选出一组学生，组内两两之间没有恋爱关系的人，求最大组数。

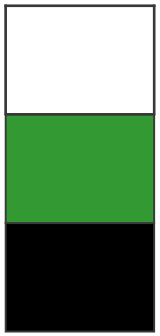
方法：二部图求最大独立集，最大独立集=点数-最大匹配。

# 例题4 Place the Robots

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。





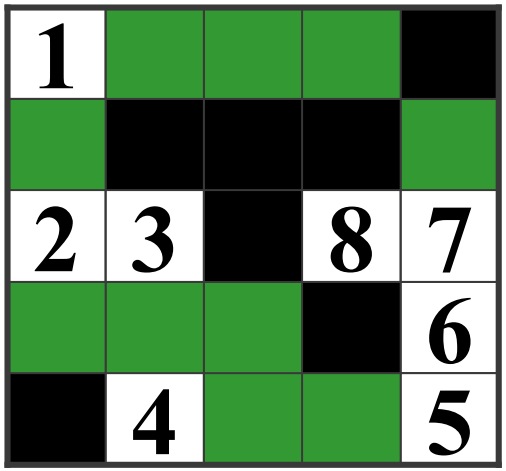
Empty

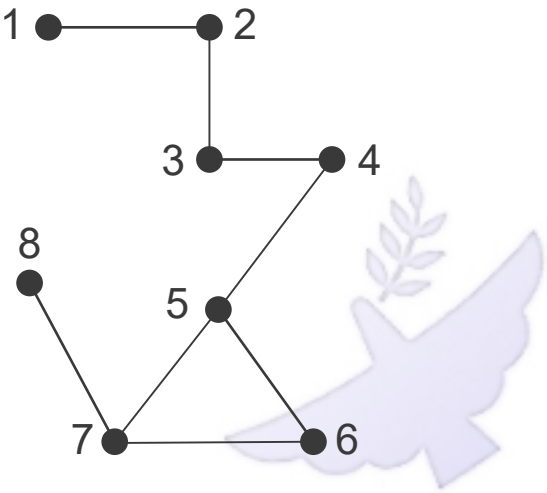
Grass

Wall

# 例题4 Place the Robots

- 模型一
- 以空地为顶点，有冲突的空地之间连边，可以得到右边的这个图.
- 于是，问题转化为求图的最大独立集问题。

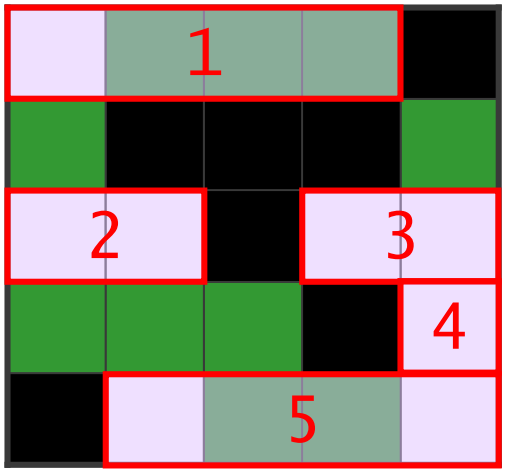


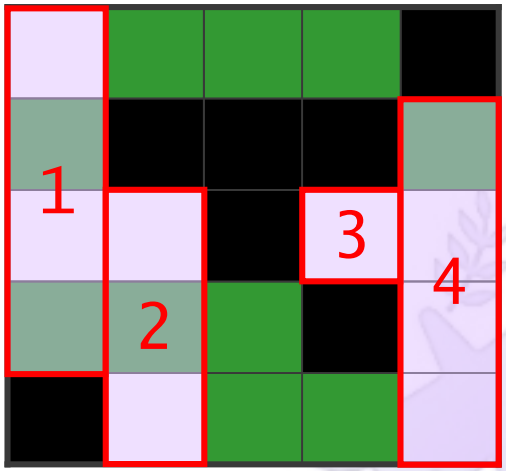

不是二部图，
这是NP问题！

# 例题4 Place the Robots

- 模型二
- 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。
- 把这些块编上号。



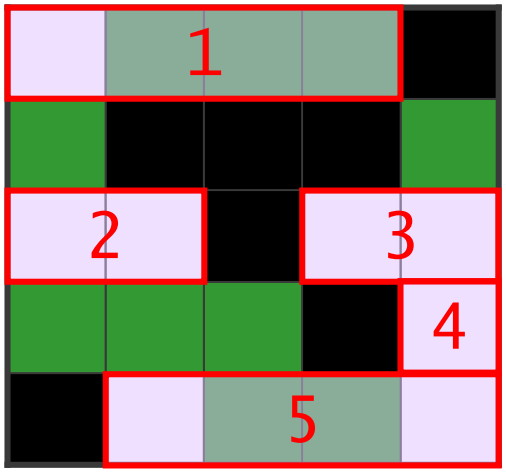

同样，把竖直方向的块也编上号。



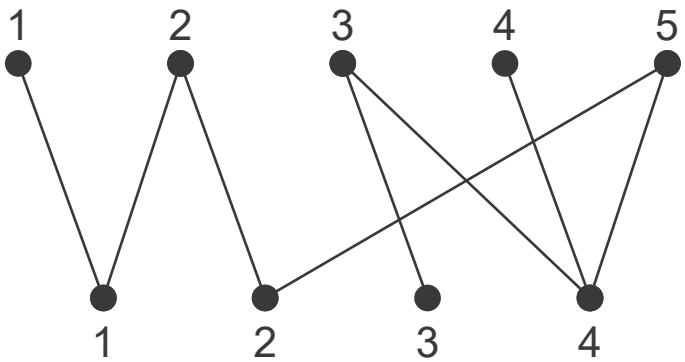
# 例题4 Place the Robots

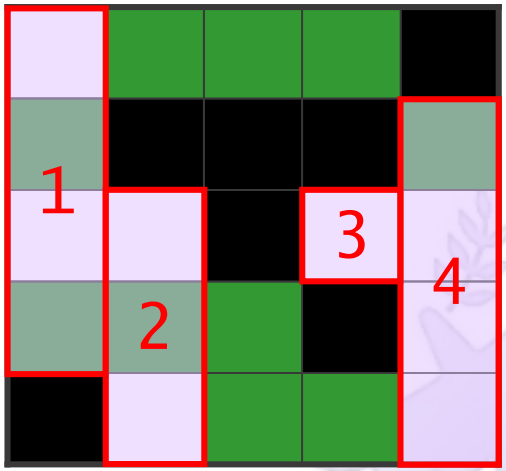
## 模型二

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。



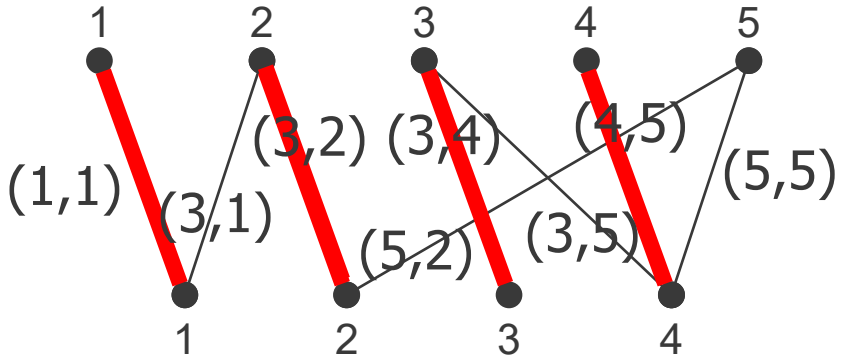

于是，问题转化成这样的一个二部图：

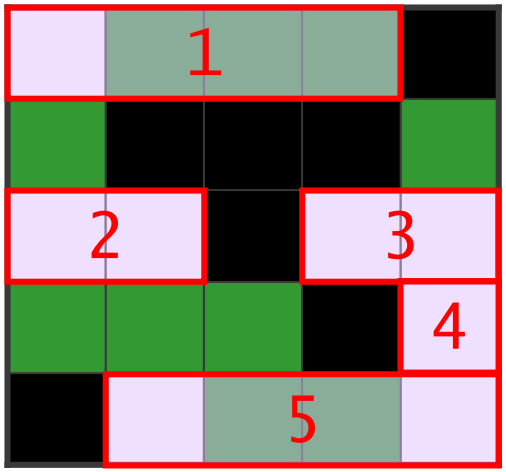


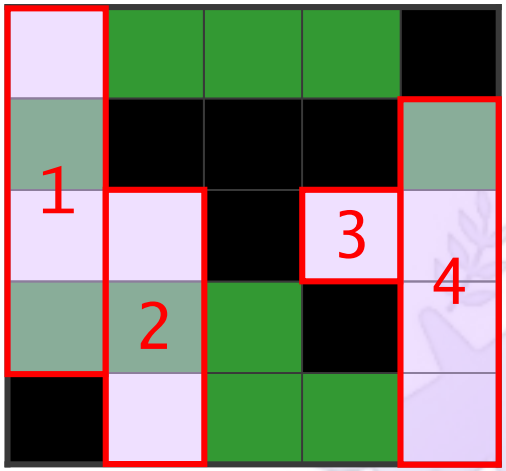



# 例题4 Place the Robots

## 模型二



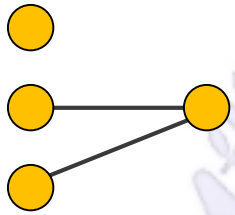

由于每条边表示一个空地（有冲突的空地之间必有公共顶点），边相交代表点之间有冲突

所以问题转化为二部图的最大匹配问题。

# Guardian of Decency

- 3. 一保守教师想带学生郊游, 却怕他们途中谈恋爱,他认为满足下面条件之一的两人谈恋爱几率很小:
- (1) 身高差>40　　(2) 性别相同
- (3) 爱好不同类型的音乐　(4) 爱好同类型的运动
- 输入是学生的数据, 求最多能带多少学生. 例:

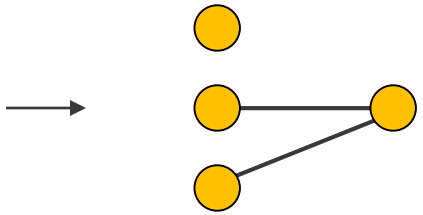
```
35 M classicism programming
0 M baroque skiing
43 M baroque chess
30 F baroque soccer
```



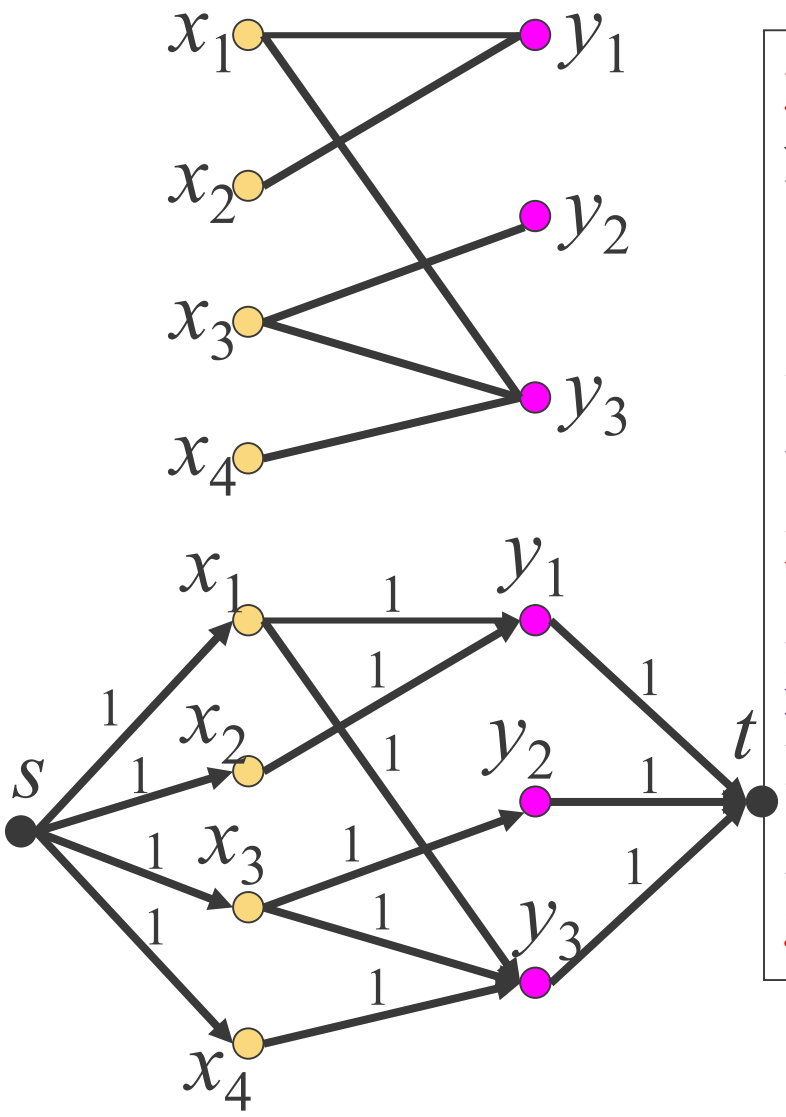
# 保守教师解法

- 将男女生分为左右顶点集,
  若有男女生满足
  身高差≤40, 音乐爱好相同, 运动爱好不同,
  则添边.
- 保守教师带的人是此二部图的最大独立集.

35 M classicism programming
0 M baroque skiing
43 M baroque chess
30 F baroque soccer



# 将二部图匹配转化为最大流




解: 构造流网络. $\rho(G_1)$等于$G_2$的最大流量

1)顶点构造:

男生1:k对应的顶点x[1:k]

女生1:n对应的顶点y[1:n]

添加源点s, 和汇点t.

2)边的构造:

从s各连1条边到k个顶点x[1:k], 容量1

若身高差≤40, 音乐爱好相同, 运动爱好不同, 则添边(x[j],y[i]), 容量1

从n个顶点y[1:n]各连1条边到t, 容量1

3)使用最大流算法, 得到相应解.

# 练习：试题库问题

**问题描述:** 假设一个试题库中有n道试题. 每道试题都标明了所属类别. 同一道题可能有多个类别属性. 现要从题库中抽取m道题组成试卷. 并要求试卷包含指定类型的试题. 试设计一个满足要求的组卷算法.

**算法设计:** 对于给定的组卷要求, 计算满足要求的组卷方案.

**数据输入:** 由文件input.txt提供输入数据. 文件第1行有2个正整数k和n($2 \le k \le 20$, $k \le n \le 1000$), k表示题库中试题类型总数, n表示题库中试题总数. 第2行有k个正整数, 第i个正整数表示要选出的类型i的题数. 这k个数相加就是要选出的总题数. 接下来n行给出了题库中每个试题的类型信息. 每行的第1个正整数p标明该题可以属于p个类, 接着的p个数是该题所属的类型号.

**结果输出:** 将组卷方案输出到文件output.txt. 文件第i行输出“i:”后接类型i的题号. 如果有多个满足要求的方案, 只要输出1个方案. 如果问题无解, 则输出“No Solution!”.

# 练习

输入文件示例

```
3 15
3 3 4
2 1 2
1 3
1 3
1 3
1 3
3 1 2 3
2 2 3
2 1 3
1 2
1 2
2 1 2
2 1 3
1 1
3 1 2 3
```

输出示例

```
1: 1 6 8
2: 7 9 10
3: 2 3 4 5
```

解: 构造流网络.

顶点构造:

…

边的构造:

…

使用最大流算法, 得到相应解.

# 练习答案

输入示例
```
3 15
3 3 4
2 1 2
1 3
1 3
1 3
1 3
3 1 2 3
2 2 3
2 1 3
1 2
1 2
2 1 2
2 1 3
1 1
3 1 2 3
```

输出示例
```
1: 1 6 8
2: 7 9 10
3: 2 3 4 5
```

- 解: 构造流网络.
- 顶点构造:
  - 构造题型号1:k对应的顶点x[1:k]
  - 构造试题号1:n对应的顶点y[1:n]
  - 添加源点s, 和汇点t.
- 边的构造:
  - 从s各连1条边到k个题型顶点(s,x[i]), 容量为题型k需要的题数
  - 若试题i属于题型j, 则添边(x[j],y[i]), 容量1
  - 每个试题顶点i到t添加1条边(y[i],t), 容量1
- 使用最大流算法, 得到相应解.

# END