算法设计与分析

第八章 网络流与图匹配算法

丁小欧
dingxiaoou@hit.edu.cn

本章内容

8.1 网络流算法

　　概念及定义

　　算法实例

8.2 图匹配算法

　　概念及定义

　　算法实例

本章内容

8.1 网络流算法

1.概念及定义

2.最大流算法

3.最小费用最大流算法

1.网络流算法：应用背景

- 很多实际问题可以建模为流网络
  - 电网
  - 水管网
  - 交通运输网
  - 通信网
  - 生产管理网

实际中很多问题利用网络流算法来解决！

1.网络流算法：应用背景

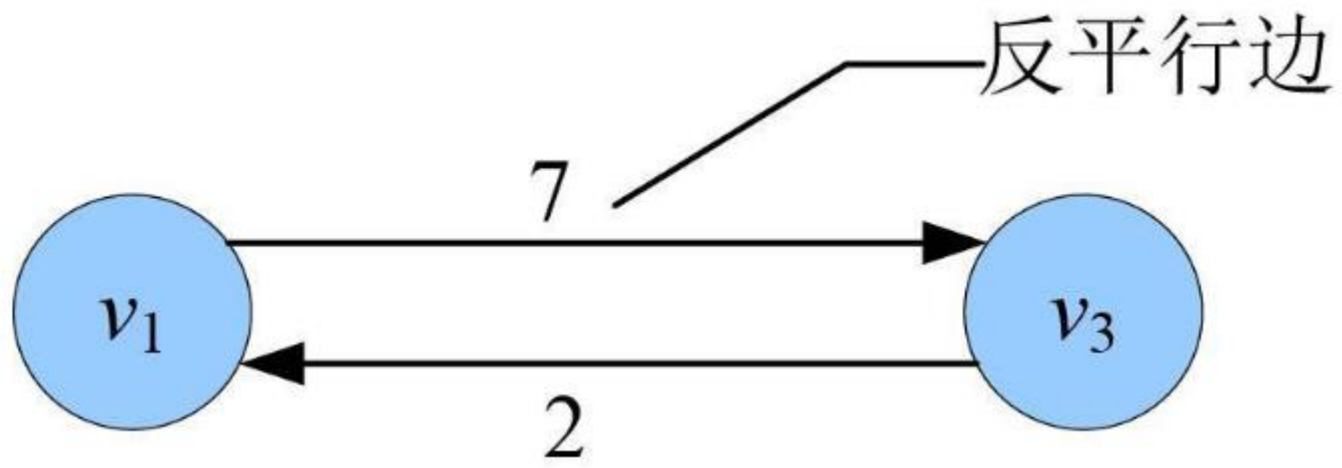
- 很多实际问题可以建模为流网络
  - 装配线上物件的流动
  - 电网中电流的流动
  - 通信网络中信息的流动
  - 道路交通网络中的运输活动
- “流动”：从源结点运送“物料”到终结点
  - 一个源点s、一个汇点t，由源点流向汇点

1.网络流算法：基本概念

- 流网络是一个无自环的有向图$G=(V, E)$，$V=\{s,v1,v2,\ldots,t\}$
  - 有两个特殊结点$s, t\in V$，s称为源结点，t称为汇点: s没有入边，t没有出边
  - 每条边$(u, v)\in E$ 表示允许的流向，边权值表示该边允许通过的最大可能容量 $c(u, v)\geq 0$
  - 若有$(u, v)\in E$ ，则必然不存在反方向边(v,u)
  - 这样的有向带权图称之为网络

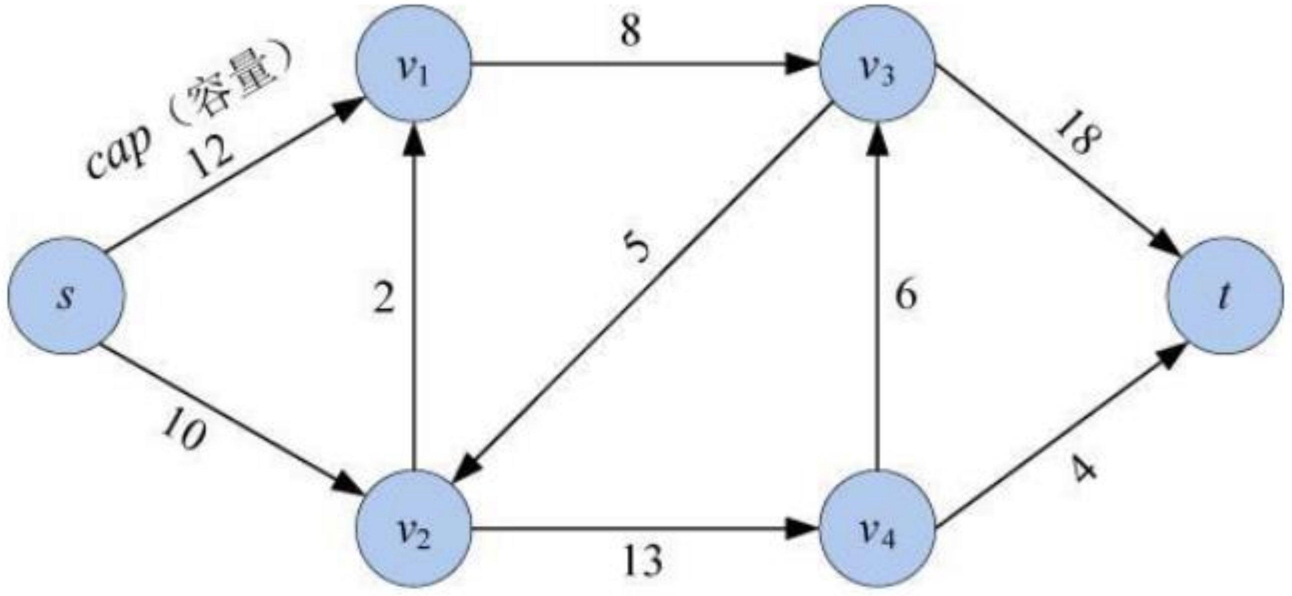
1.网络流算法：基本概念

- 流网络是一个无自环的有向图$G=(V, E)$，$V=\{s,v1,v2,\ldots,t\}$
  - 网络是一个有向带权图，包含一个源结点和一个汇结点
  - 没有反平行边
    - 如果$v1$和$v3$之间有边,则$(v1, v3)$和$(v3, v1)$这两条边不会同时存在



1.网络流算法：基本概念

- 例子:某公司从哈尔滨s到北京t送货物，雇佣一家货运公司，其安排多个运输线路，边权代表两个城市之间每天最多运送的产品数量。不管货运代理是怎么运输的，只需要知道每天从工厂最多发出去多少货，在北京仓库就要收到多少货物，否则由货运代理照价赔偿



1.网络流算法：基本概念

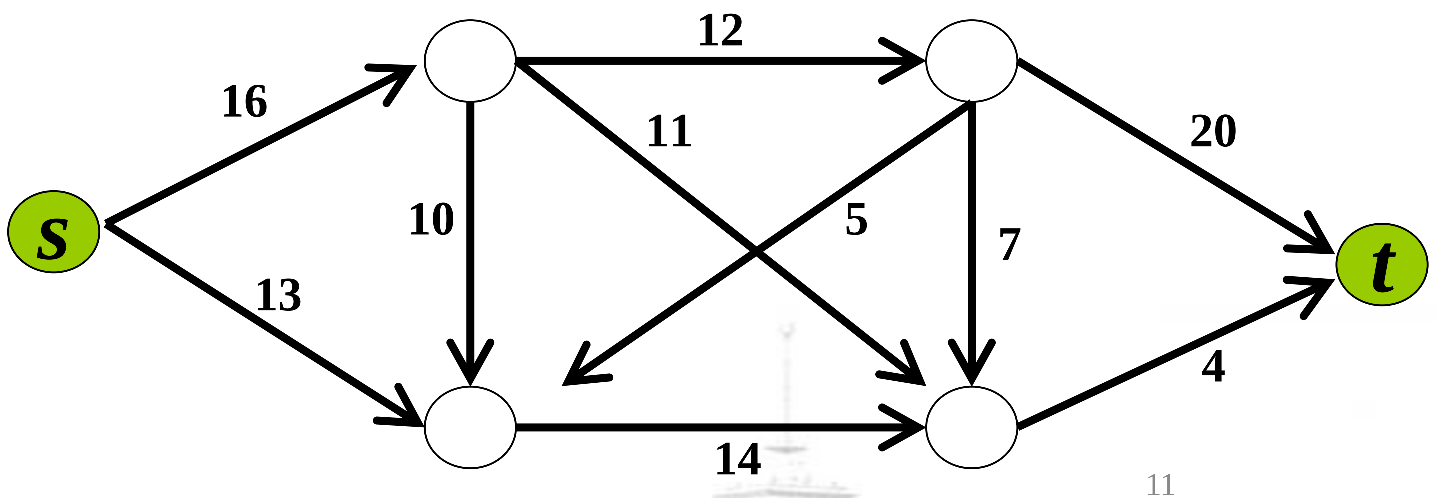
- “网络流”：虽然看不到水的流动，但知道进水口流进多少水，出水口就要流出多少水

1.网络流算法：基本概念

- 流网络是一个无自环的有向图$G=(V, E)$，$V=\{s,v1,v2,\ldots,t\}$
  - 有两个特殊结点$s, t\in V$，s称为源结点，t称为汇点
  - 每条边$(u, v)\in E$ 表示允许的流向，边权值表示该边允许通过的最大可能容量 $c(u, v)\geq 0$
  - 网络流：定义在E上的一个非负函数$flow=\{flow(u,v)\}$,$flow(u,v)$是边上的流量
  - $\forall v\in V$，存在一个s到t经过v的路径$s\Rightarrow v\Rightarrow t$.

1.网络流算法：基本概念

- 流网络是一个无自环的有向图$G=(V, E)$，$V=\{s,v1,v2,\ldots,t\}$



流网络是连通的

除源结点外，每个结点都至少有一条进入的边，所以$|E| \geq |V|-1$

1.网络流算法：基本概念

- 流网络是一个无自环的有向图*G*=(*V*, *E*)，*V*={*s*,*v*1,*v*2,…,*t*}
  - 网络流：定义在E上的一个非负函数*flow*={*flow*(*u*,*v*)},*flow*(*u*,*v*)是边上的流量
  - 可行流:满足以下两个性质的网络流称为可行流

(1)容量约束

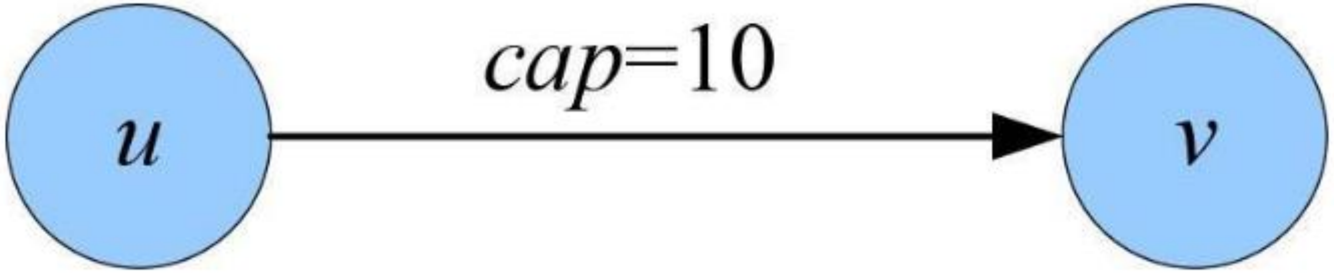
(2)流量守恒

1.网络流算法：基本概念

- 流网络是一个无自环的有向图$G=(V, E)$，$V=\{s,v1,v2,\ldots,t\}$
  - 网络流：定义在E上的一个非负函数$flow=\{flow(u,v)\}$,$flow(u,v)$是边上的流量
  - 可行流:满足以下两个性质的网络流称为可行流

(1)容量约束:每个管道的实际流量$flow$ (缩写为f)不能超过该管道的最大容量cap (缩写为c)

$$\forall u,v\in V,\ 0 \le f(u, v) \le c(u, v)$$

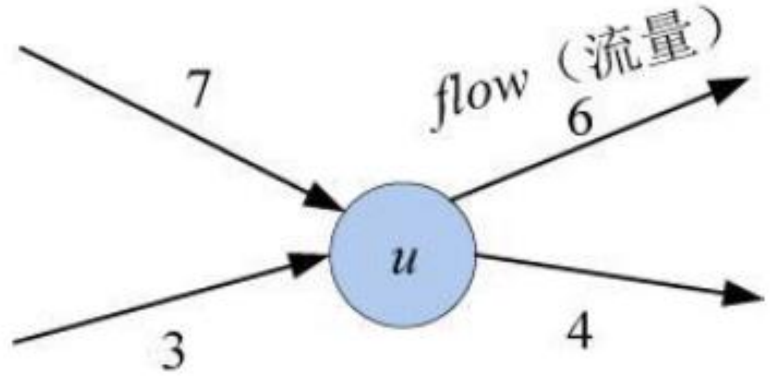


1.网络流算法：基本概念

- 流网络是一个无自环的有向图*G*=(*V*, *E*)，*V*={*s*,*v*1,*v*2,…,*t*}
  - 网络流：定义在*E*上的一个非负函数*flow*={*flow*(*u*,*v*)},*flow*(*u*,*v*)是边上的流量
  - 可行流:满足以下两个性质的网络流称为可行流

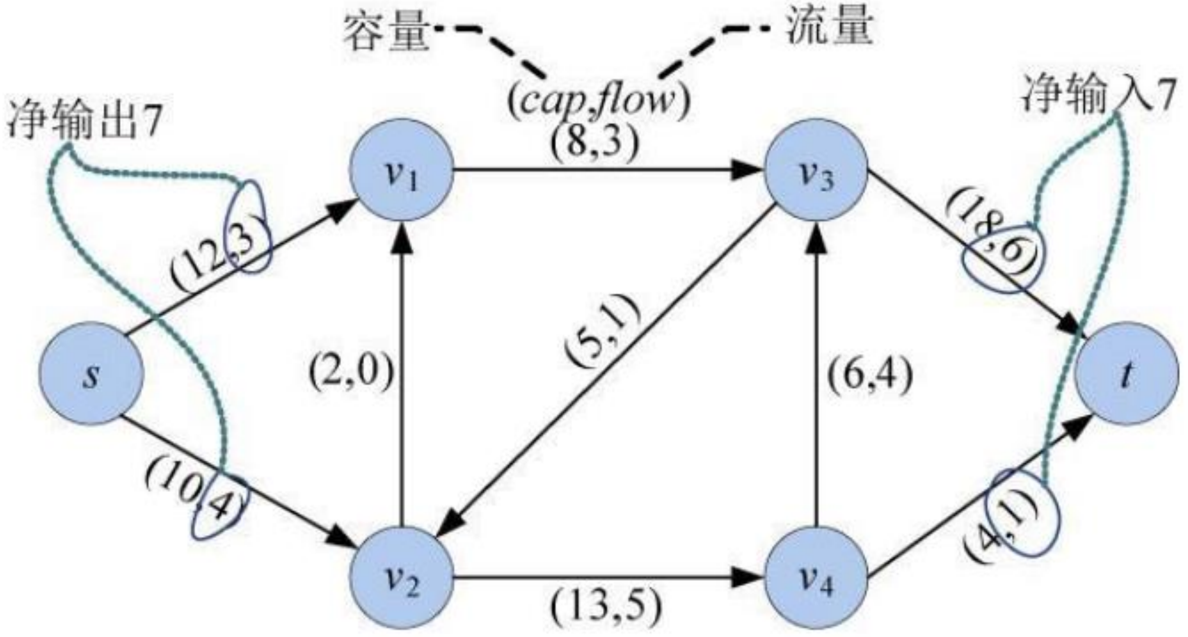
(2)流量守恒:除了源点s和汇点t之外，所有内部结点流入量等于流出量

$\forall u \in V\text{-}\{s,t\}, \sum_{w \in V} f(w,u) = \sum_{v \in V} f(u,v)$



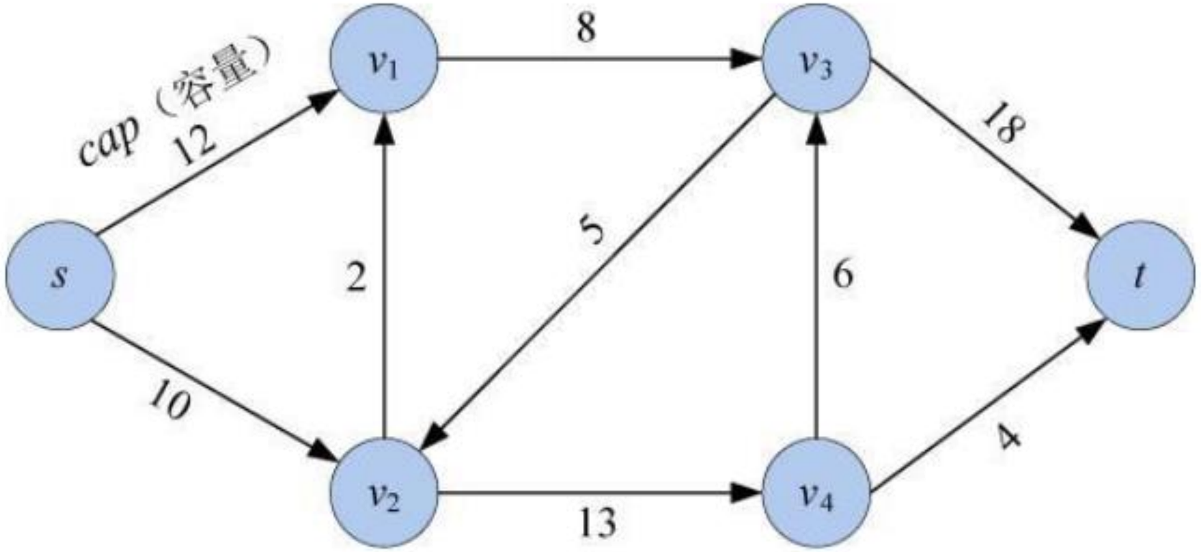
1.网络流算法：基本概念

- 流网络是一个无自环的有向图*G*=(*V*, *E*)，*V*={*s*,*v1*,*v2*,…,*t*}
    - 网络流：定义在*E*上的一个非负函数***flow*={*flow*(*u*,*v*)},*flow*(*u*,*v*)**是边上的流量
    - 对于一个网络可行流，净输入=净输出
    - 一个流*f*的值|*f*|定义为：$\sum_{v \in V} f(s,v)$



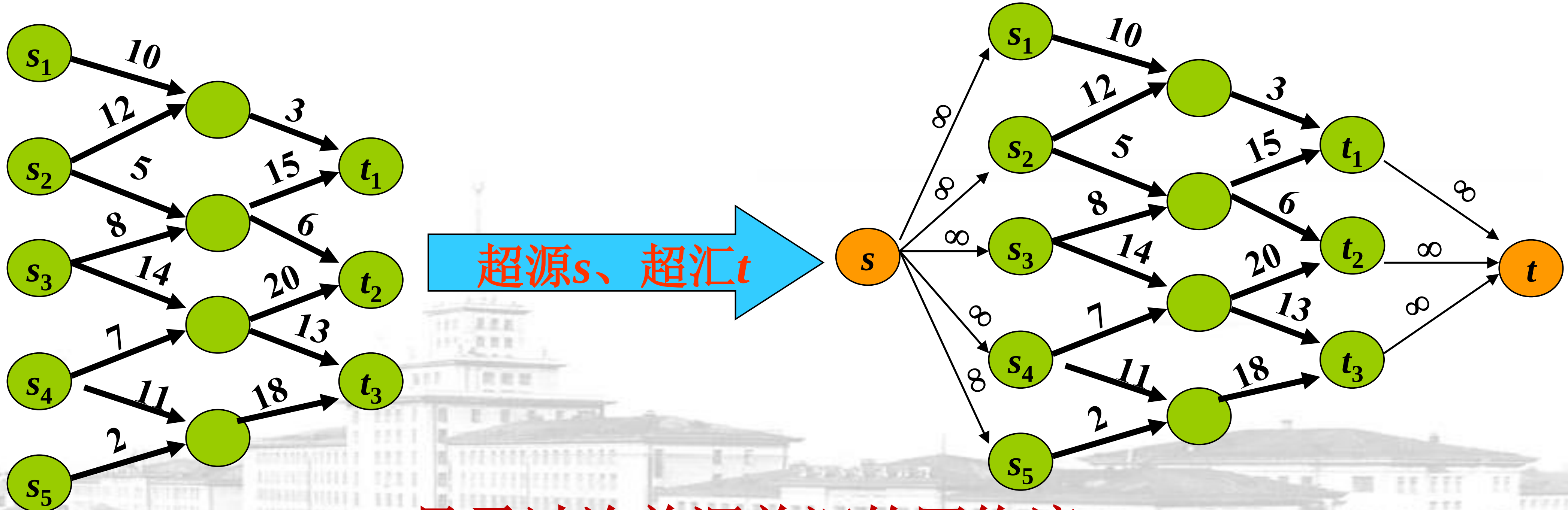
1.网络流算法：基本概念

- 多源多汇的网络
  - 某公司从哈尔滨s到北京t送货物，可能有多个分公司地址和目标分公司地址
  - 多源多汇的最大流问题并不比单源单汇最大流问题难！



1.网络流算法：基本概念

- 多源多汇的网络
  - 加入一个超级源结点s，从s到原来的每个源结点连一条容量为无限的有向边，加入超级汇结点t，从原来每个汇结点增加一条容量为无限的有向边

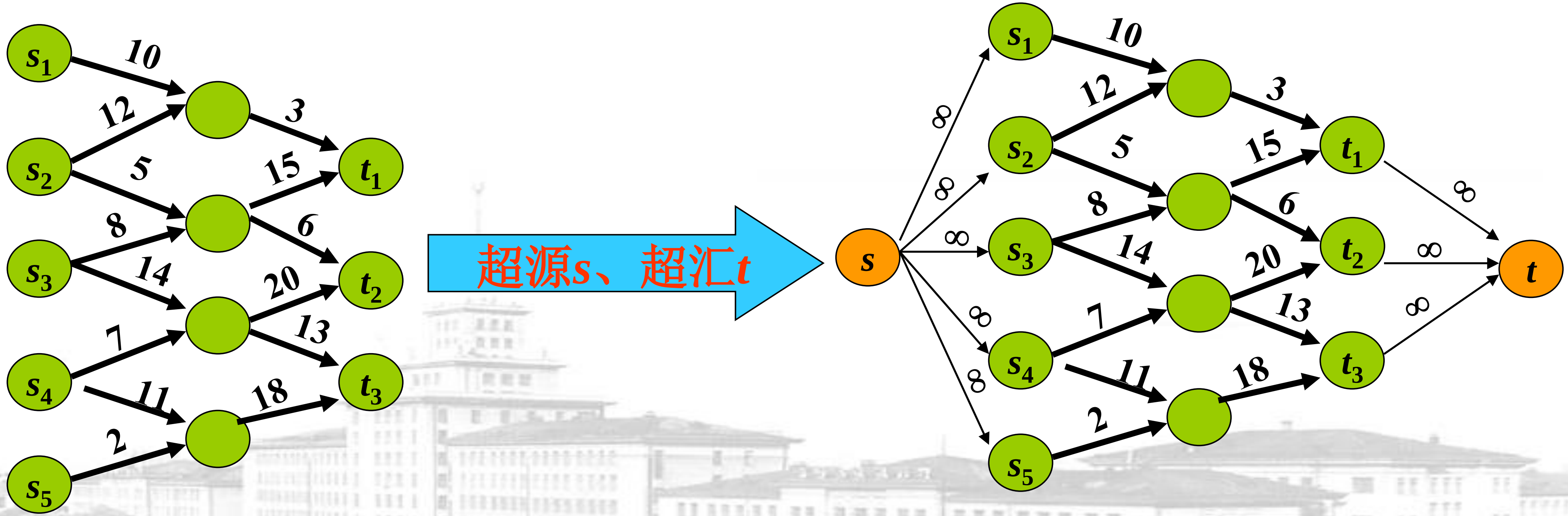


只需讨论单源单汇的网络流

1.网络流算法：基本概念

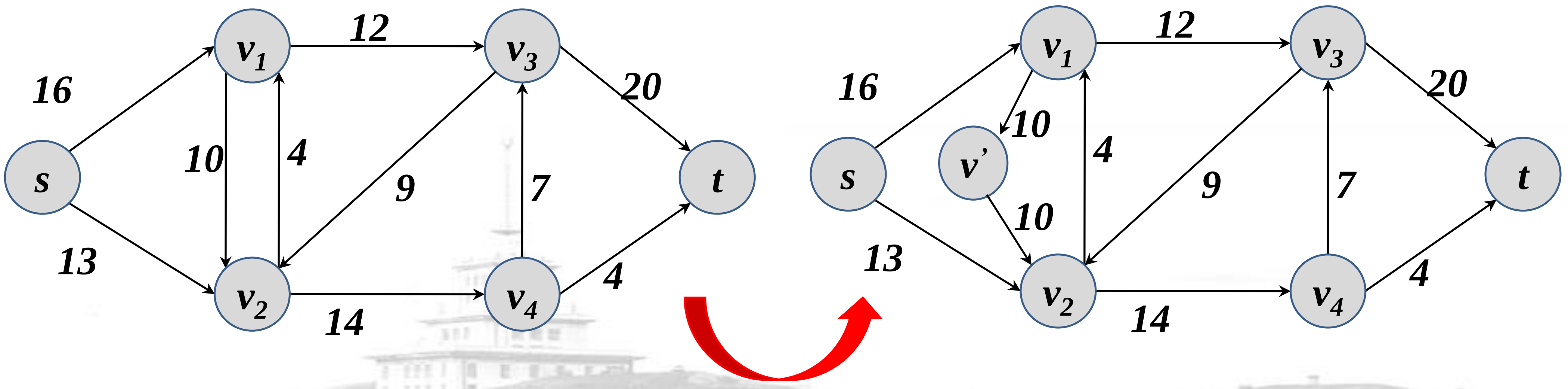
- 多源多汇的网络
  - 在这两个网络上计算最大流问题是等价的

只需讨论单源单汇的网络流



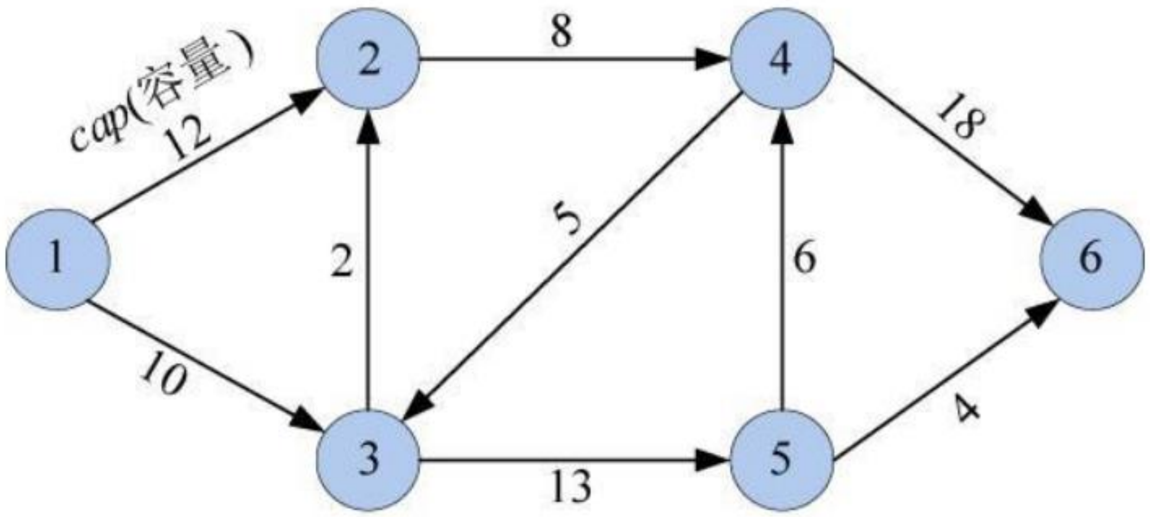
1.网络流算法：一些假设

- 单源点、单汇点流网络
- 假设：流网络中无反向边
  - 给定有向图$G=(V, E)$，如果边$(u, v)\in E$，则边$(v, u)\notin E$



1.网络流算法：最大(网络)流问题

- 问题定义
  - 输入: 流网络$G=(V, E)$
  - 输出: (在满足容量约束和流量守恒情况下)，具有最大流值的流f



1.网络流算法：直观想法

- 循环递进
  - 初始：网络上的流为0
  - 找出一条从s到t的路径p和正数a，使得p上的每一条边(u,v)的流量增加a之后仍能够满足容量约束：$f(u,v)+a\le c(u,v)$

    //将p上的每条边的流量增加a，得到一个更大的流
  - 重复执行第二步，直到找不到满足约束条件的路径.

关键在于：

1. 如何找路径p，以便得到更大的流？
2. 如何判断循环结束的条件？

即：当循环结束时，所得到的流一定是最大流么？

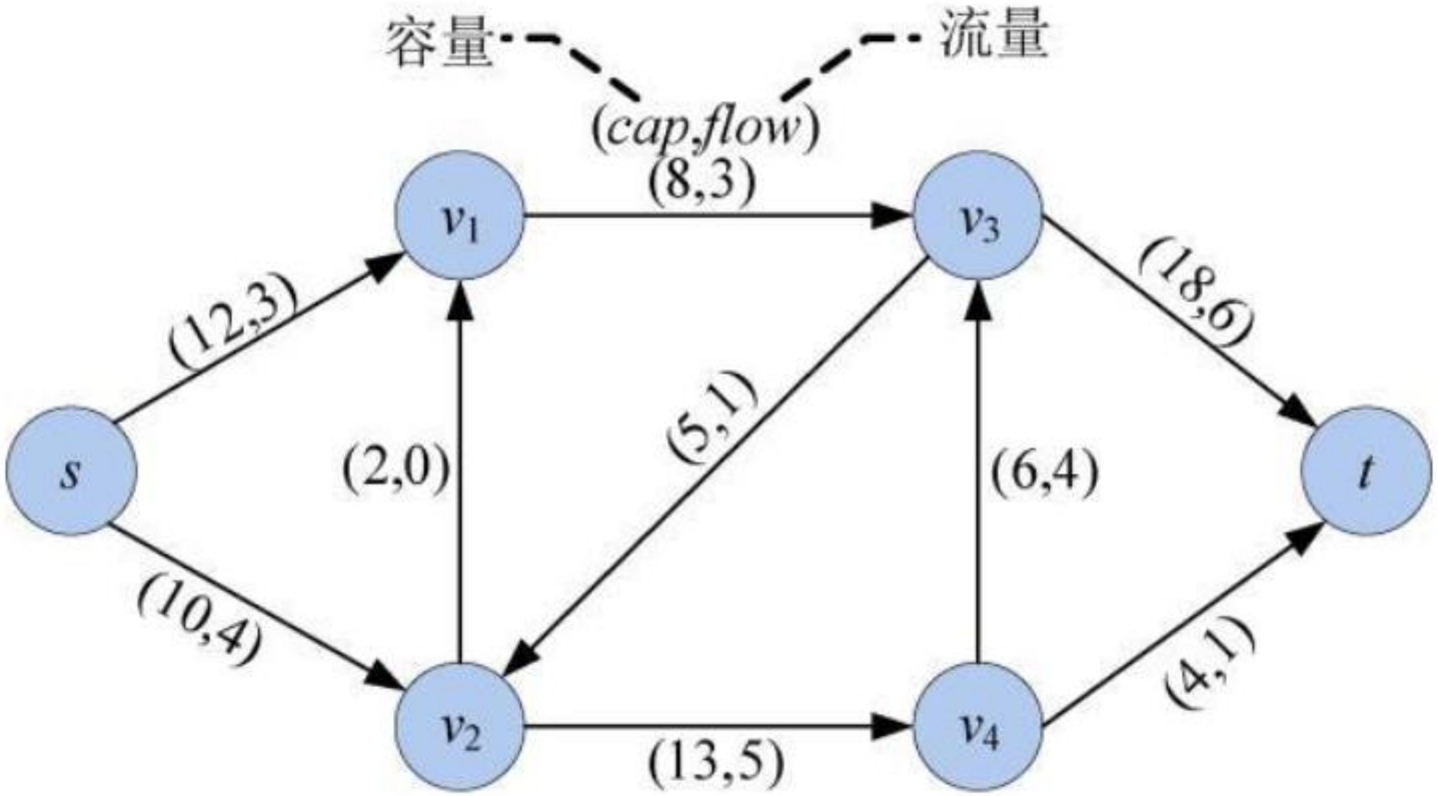
1.网络流算法：直观想法

- Ford-Fulkerson方法
  - 如何找路径p，以便得到更大的流？
  - 如何判断循环结束的条件？

1.网络流算法：直观想法

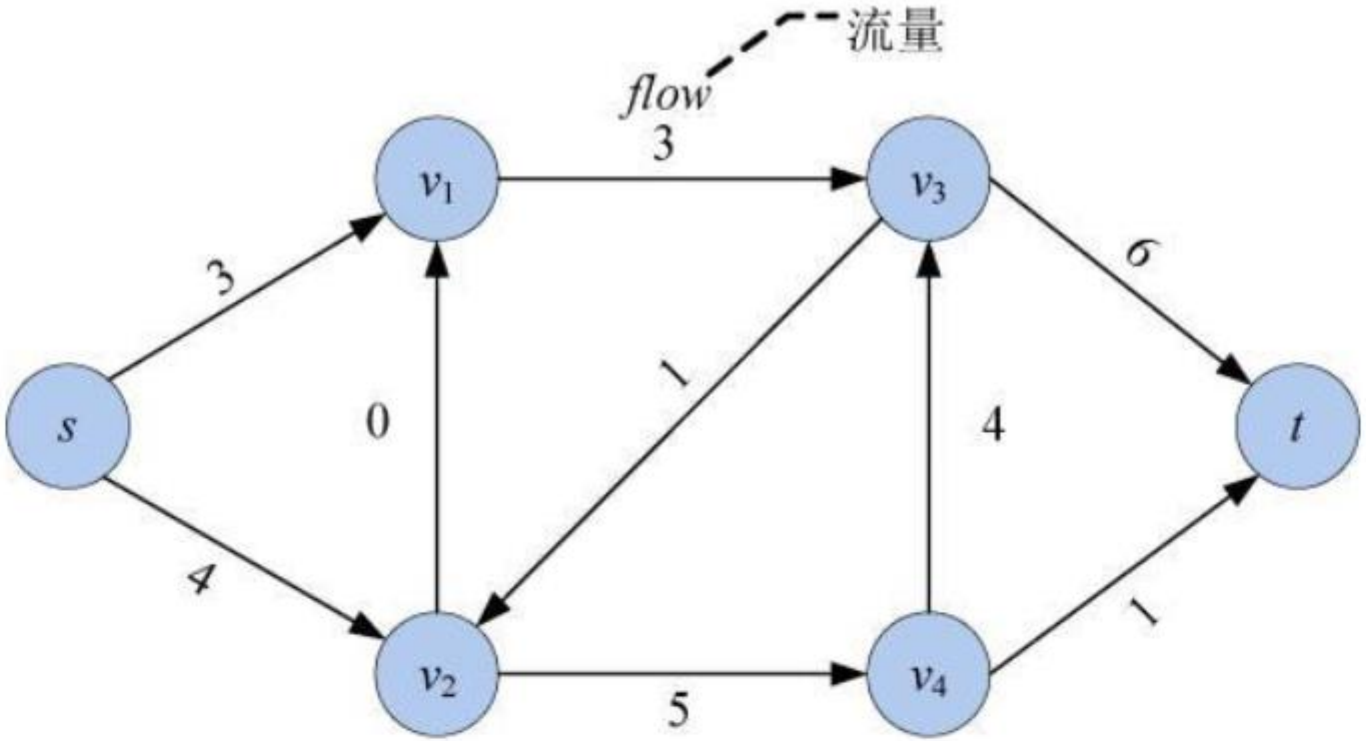
- **Ford-Fulkerson方法**
  - 在一个关联的剩余网络(余图)中寻找一条增广路径
  - 在实流网络中沿可增广路增流，直到不存在可增广路为止

- **1.网络流算法：** Ford-Fulkerson方法
  - 在实流网络中沿可增广路增流，直到不存在可增广路为止



• 1.网络流算法：Ford-Fulkerson方法

- 在实流网络:只显示每条边实际流量，不显示容量



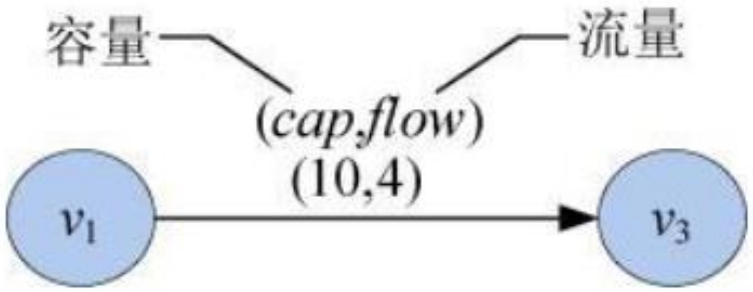
1.网络流算法：Ford-Fulkerson方法

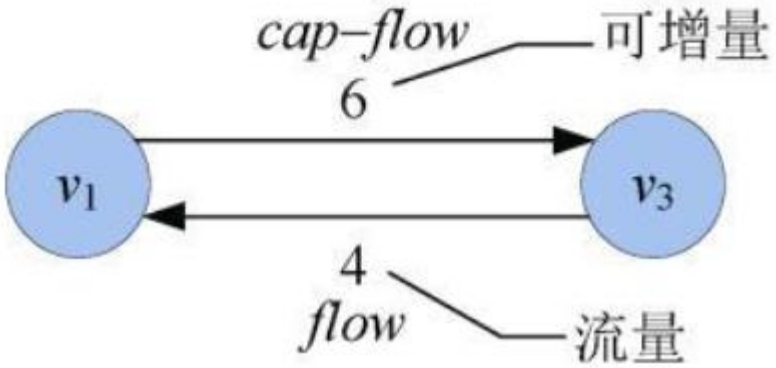
- 残余网络

给定流网络$G(V,E)$和一个流f，则由f诱导的G的剩余网络为$G_f=(V, E_f)$，其中E_f为：

对于G中每条边 (u, v)，若$c(u,v)-f(u,v)>0$，则$(u, v) \in E_f$，且$c_f(u,v)=c(u,v)-f(u,v)$ (称$c_f(u,v)$为剩余容量)

对于G中每条边 (u, v)，在G_f中构造边(v, u)，且$c_f(v, u)=f(u,v)$





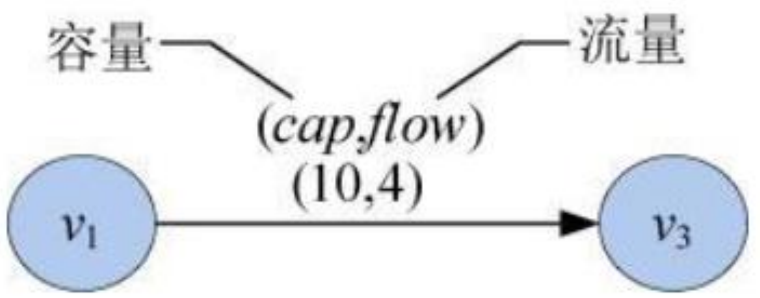
• 1.网络流算法：Ford-Fulkerson方法

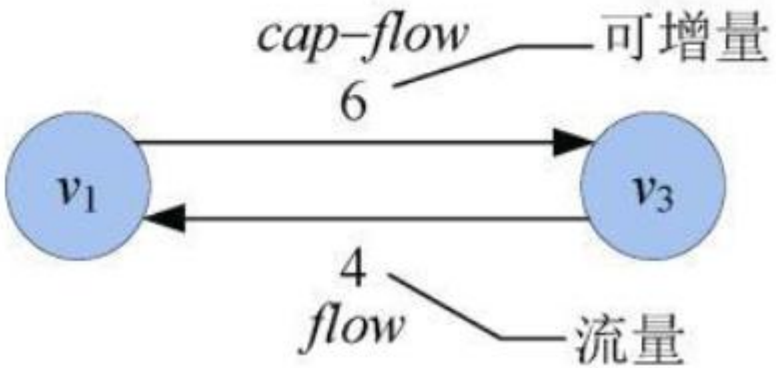
• 剩余网络

给定流网络$G(V,E)$和一个流f，则由f诱导的G的剩余网络为$G_f=(V, E_f)$，其中E_f为：

对于G中每条边 (u, v)，若$c(u,v)-f(u,v)>0$，则$(u, v) \in E_f$，且$c_f(u,v)=c(u,v)-f(u,v)$ (称$c_f(u,v)$为剩余容量)

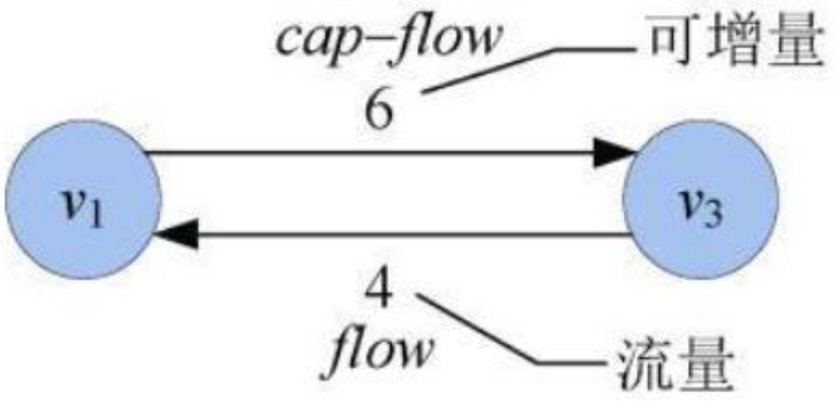
对于G中每条边 (u, v)，在G_f中构造边(v, u)，且$c_f(v, u)=f(u,v)$





在剩余网络中，与网络边对应的同向边是可增量（即还可以增加多少流量），反向边是实际流量！

• 1.网络流算法：Ford-Fulkersc

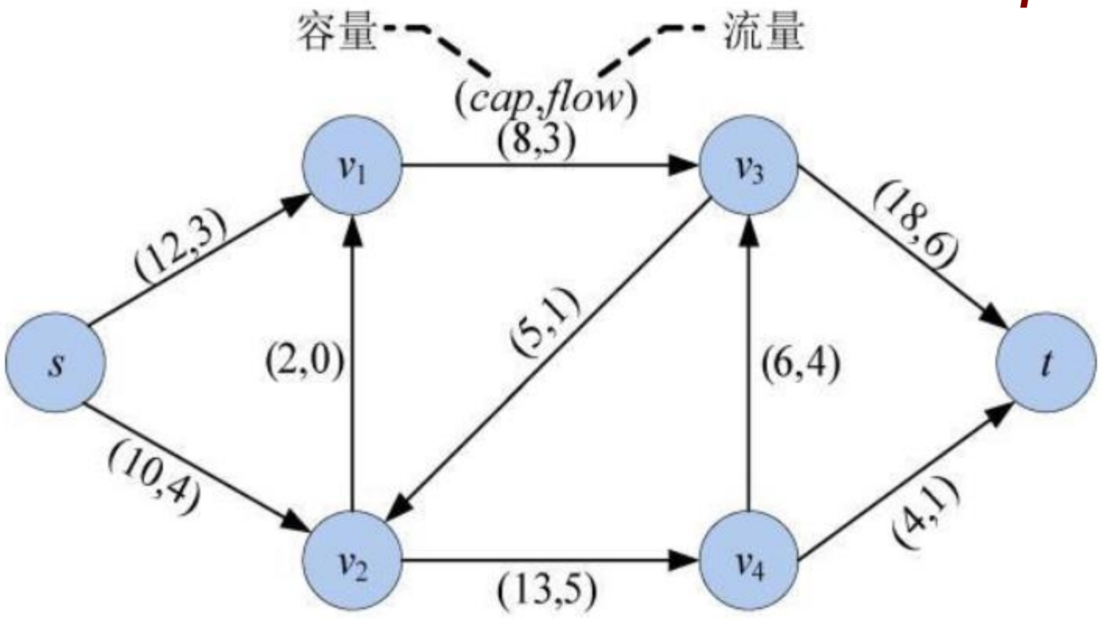


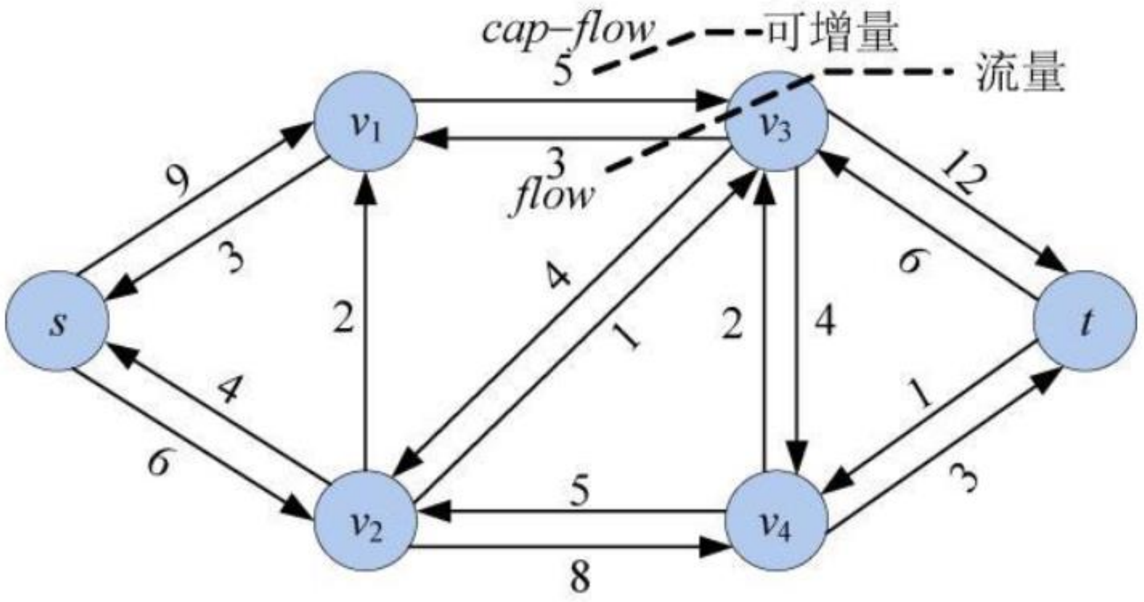
- 剩余网络(右图)

给定流网络$G(V,E)$和一个流f，则由f诱导的G
其中E_f为：

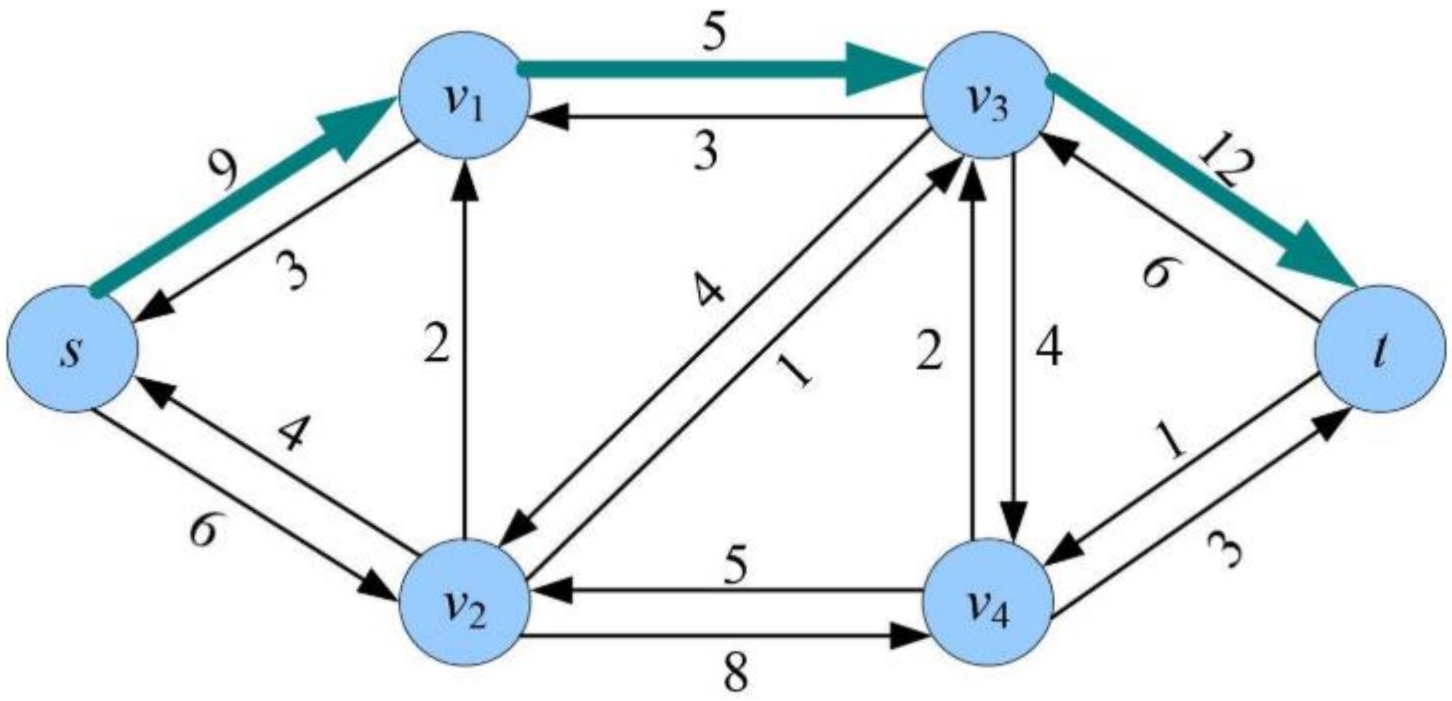
对于G中每条边 (u, v)，若$c(u,v)-f(u,v)>0$，则$(u, v) \in E_f$，且$c_f(u,v)=c(u,v)-f(u,v)$　(称$c_f(u,v)$为剩余容量)

对于G中每条边 (u, v)，在G_f中构造边(v, u)，且$c_f(v, u)=f(u,v)$





- **1.网络流算法：Ford-Fulkerson方法**
  - 增广路
    - 剩余网络中的一条由源结点*s*到汇点*t*的一条路径*p*
      - *s—v1—v3—t*就是一条可增广路

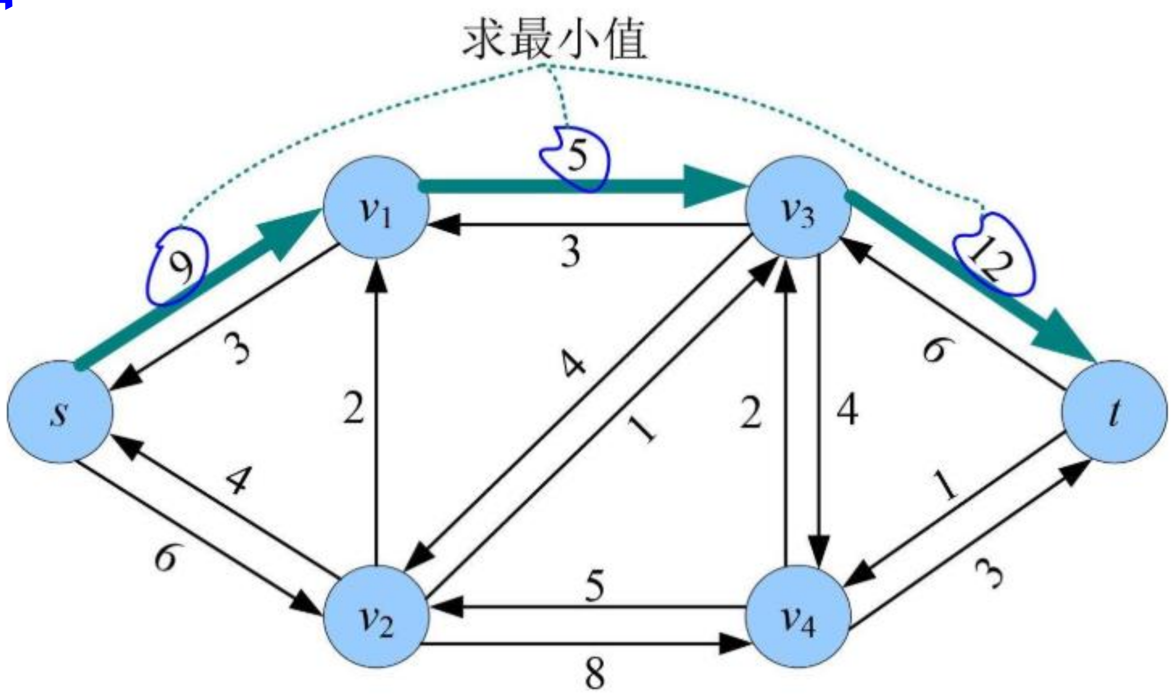


• 1.网络流算法：Ford-Fulkerson方法

- 增广路径*p*的剩余容量
  - $c_f(p)=\min\{c_f(u, v) : (u, v)$属于路径$p\}$
  - 表示了该路径能够增加的流的最大值
  - *s—v1—v3—t*就是一条可增广路

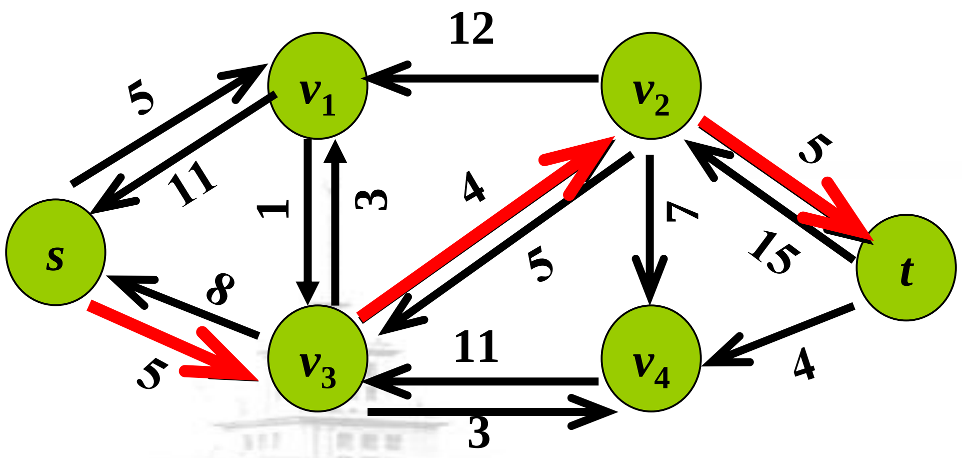
  它的剩余容量是*5*

可增广量*d*等于可增广路*p*上每条边值的最小值



1.网络流算法：相关概念

- 增广路径
  - 剩余网络中的一条由源结点s到汇点t的一条路径p
- 增广路径p的剩余容量
  - $c_f(p)=\min\{c_f(u, v) : (u, v)$属于路径$p\}$
  - 表示了该路径能够增加的流的最大值



图中红色标注的路径为一条增广路径，其剩余容量为4

1.网络流算法：算法思想

- 求网络G的最大流
  - 首先在残余网络中找可增广路，然后在实流网络G'中沿可增广路增流，直到不存在可增广路为止。这时实流网络G'就是最大流网络

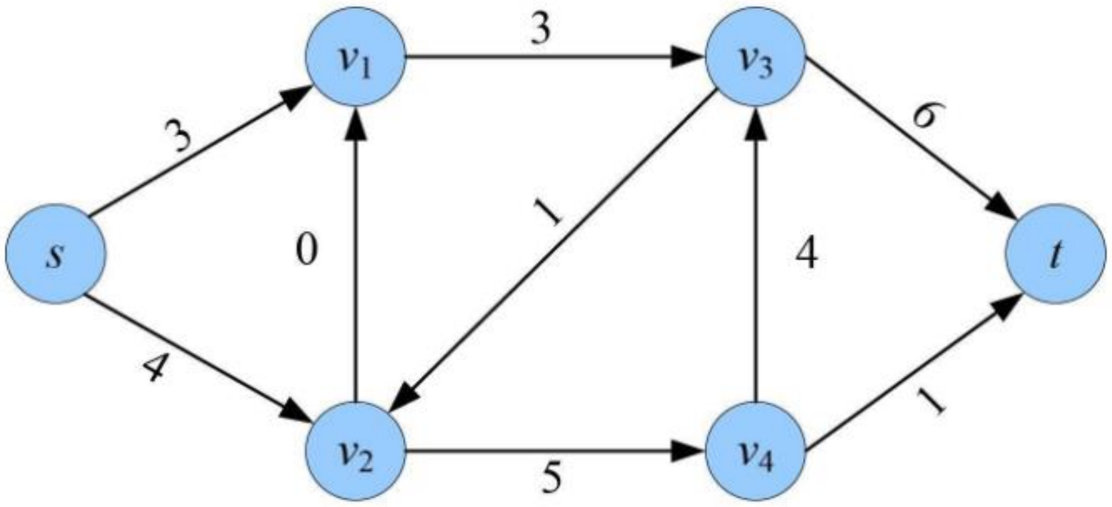
1.网络流算法：增广路增流

- 两个过程：
  - 在实流网络中增流，在残余网络中减流
  - 残余网络中可增广路上的边值表示可增量，在实流网络中流量增加了，对应着可增量要减小
- 实流网络增流

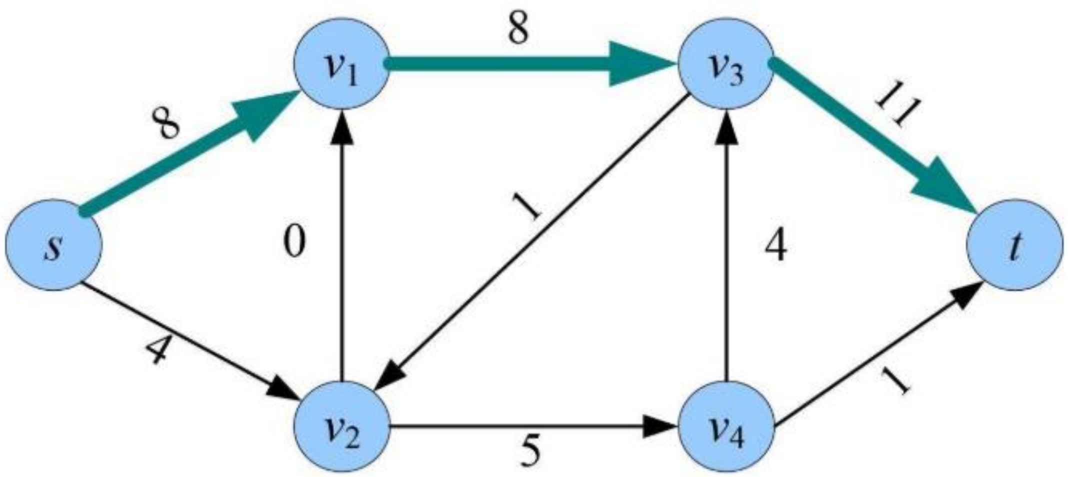
1.网络流算法：增广路增流

实流网络增流

- 找到一条可增广路*s*—*v1*—*v3*—*t*，可增广量*d*=5
- 先在实流网络中沿着可增广路增流：可增广路上同向边增加流量*d*，反向边减少流量*d*
- 本例中都是和可增广路同向的边，因此每条边上增加流量 5



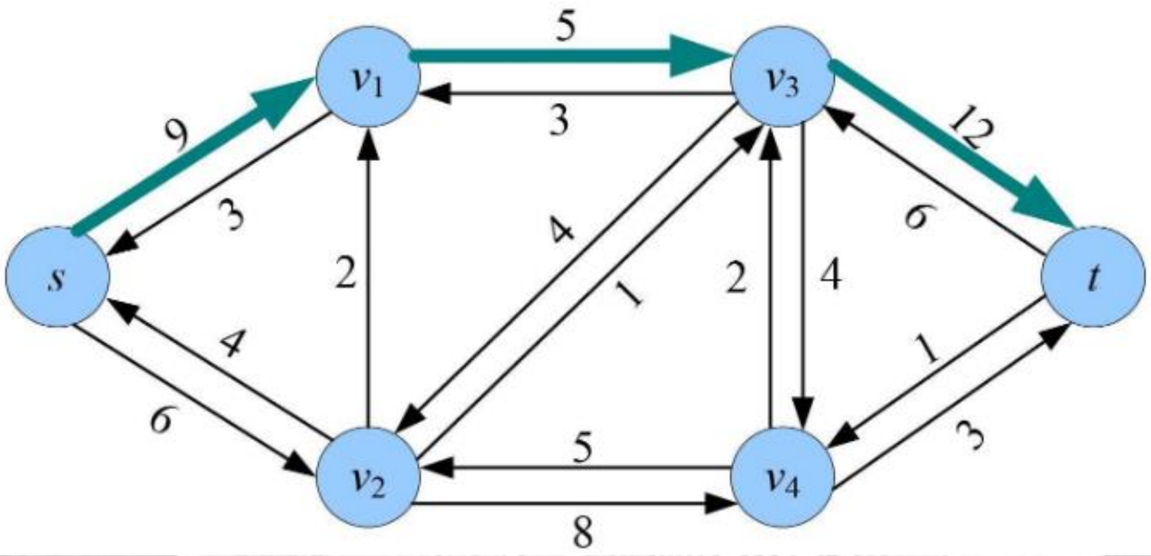
实流网络(增流前)



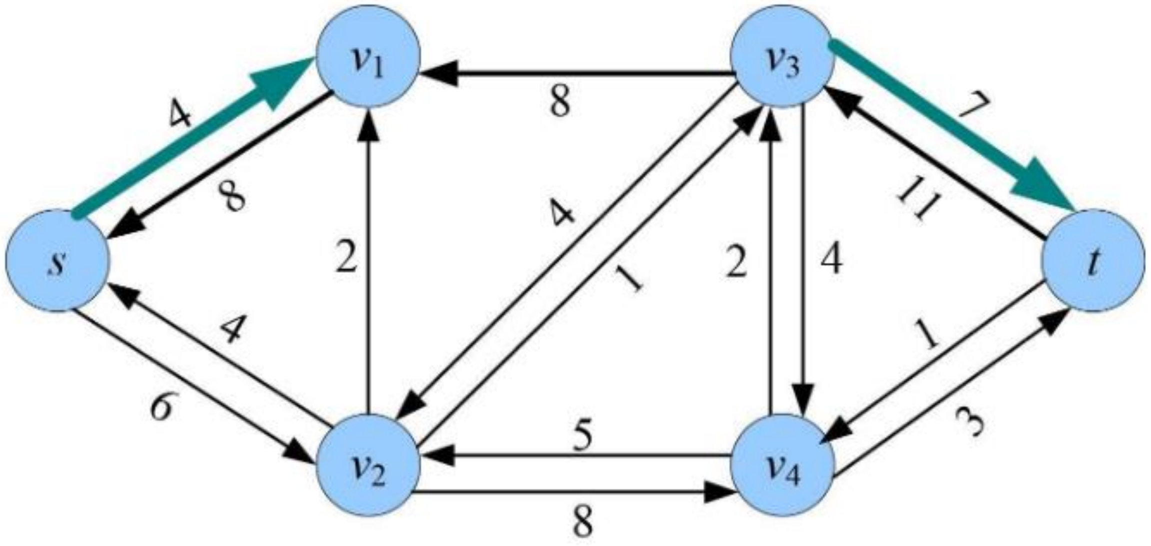
实流网络(增流后)

1.网络流算法：增广路增流

- 残余网络减流
  - 沿着可增广路减流：可增广路上的同向边减少流量*d*，反向边增加流量*d*。沿着可增广路*s—v1—v3—t*，同向边（可增量）减少流量 5，反向边增加流量 5。如果一条边流量为 0，则删除这条边
  - 减流后*v1—v3*流量为 0，删除这条边



残余网络(减流前)



残余网络(减流后)

1.网络流算法：增广路算法

- 增广路定理
  - 设*flow*是网络*G*的一个可行流，如果不存在从源点*s*到汇点*t*关于*flow*的可增广路*p*，则*flow*是*G*的一个最大流
- 增广路算法思想：
  - 在残余网络中找到可增广路，然后在实流网络中沿可增广路增流，在残余网络中沿可增广路减流；继续在残余网络中找可增广路，直到不存在可增广路为止
  - 实流网络中的可行流就是所求的最大流
  - 增广路算法是一种方法:**Ford-Fullkerson** 并没有说明如何找可增广路，而找增广路的算法不同，算法的时间复杂度相差很大

1.网络流算法：Ford-Fulkerson算法思想

算法Ford-Fulkerson(G,s,t)

Input 流网络G，源s，汇t

Output G中从s到t的最大流

1. 初始化所有边的流量为0
2. **while** 存在增广路径p **do**
3. 沿着路径p增加流量得到更大的流f
4. **return** f

1.网络流算法：Ford-Fulkerson算法

算法Ford-Fulkerson(G,s,t)

Input 流网络G，源s，汇t

Output G中从s到t的最大流

For $\forall(u,v)\in E[G]$ **do**
    $f(u,v)\leftarrow 0$
    $f(v,u)\leftarrow c(u,v)$
While G_f存在增广路径p **do**
    $c_f(p)=\min\{c_f(u,v)\mid (u,v)$是$p$上的边$\}$
    For p上的每条边(u,v) **do**
        If (u,v)是流网络中的边 **Then**
            $f(u,v)\leftarrow f(u,v)+c_f(p)$
        Else
            $f(v,u)\leftarrow f(v,u)-c_f(p)$

1.网络流算法：Ford-Fulkerson算法

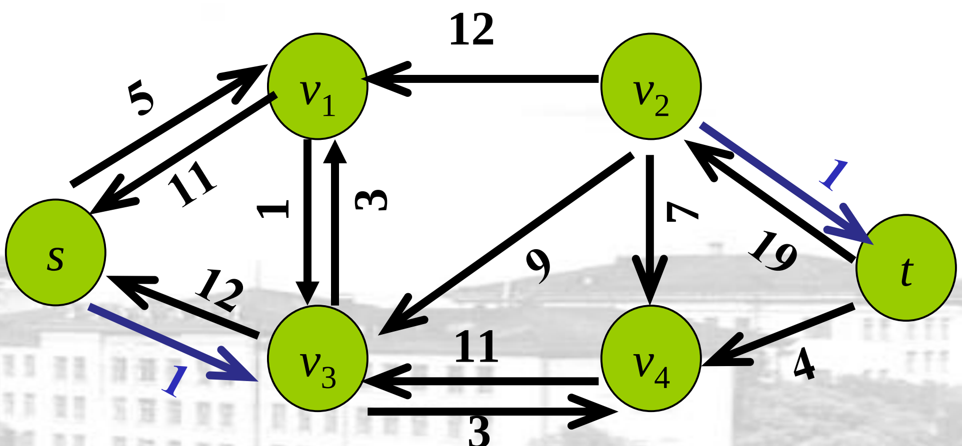
FF算法可以求得最大流*

设FF算法在流 f上停止：

- 从s出发的DFS不包含t
- 这意味着在剩余网络中DFS经过的结点和其余结点之间割的容量是0
- 因此这个割中的每条边都被增广流逆转过，这意味着

$$c(\text{DFS},\text{DFS}') = |f|$$

- 因此可知得到的$|f|$最大



1.网络流算法：Ford-Fulkerson算法分析

算法Ford-Fulkerson(G,s,t)

Input 流网络G，源s，汇t

Output G中从s到t的最大流

1. For $\forall(u,v)\in E[G]$ do
2. $f(u,v)\leftarrow 0$
3. $f(v,u)\leftarrow c(u,v)$
4. While G_f存在增广路径p do
5. $c_f(p)=\min\{c_f(u,v) \mid (u,v)$是$p$上的边$\}$
6. For p上的每条边(u,v) do
7. If (u,v)是流网络中的边 Then
8. $f(u,v)\leftarrow f(u,v)+c_f(p)$
9. Else
10. $f(v,u)\leftarrow f(v,u)-c_f(p)$

1-3步：$O(E)$

4-8步：循环次数最多为$|f^*|$

f^*是增广路的次数

第4步在G_f中找路径(深度或广度优先)代价$O(E)$

总的复杂性$O(|f^*||E|)$

1.网络流算法：最短增广路径算法

- 如何找一条增广路径p？
  - 可设置最大容量优先
  - 可以是最短路径（广度优先）优先
    - **Edmonds-Karp** 算法就是以广度优先的增广路算法，又称为最短增广路算法（**Shortest Augument Path**，**SAP**）

1.网络流算法：最短增广路径算法

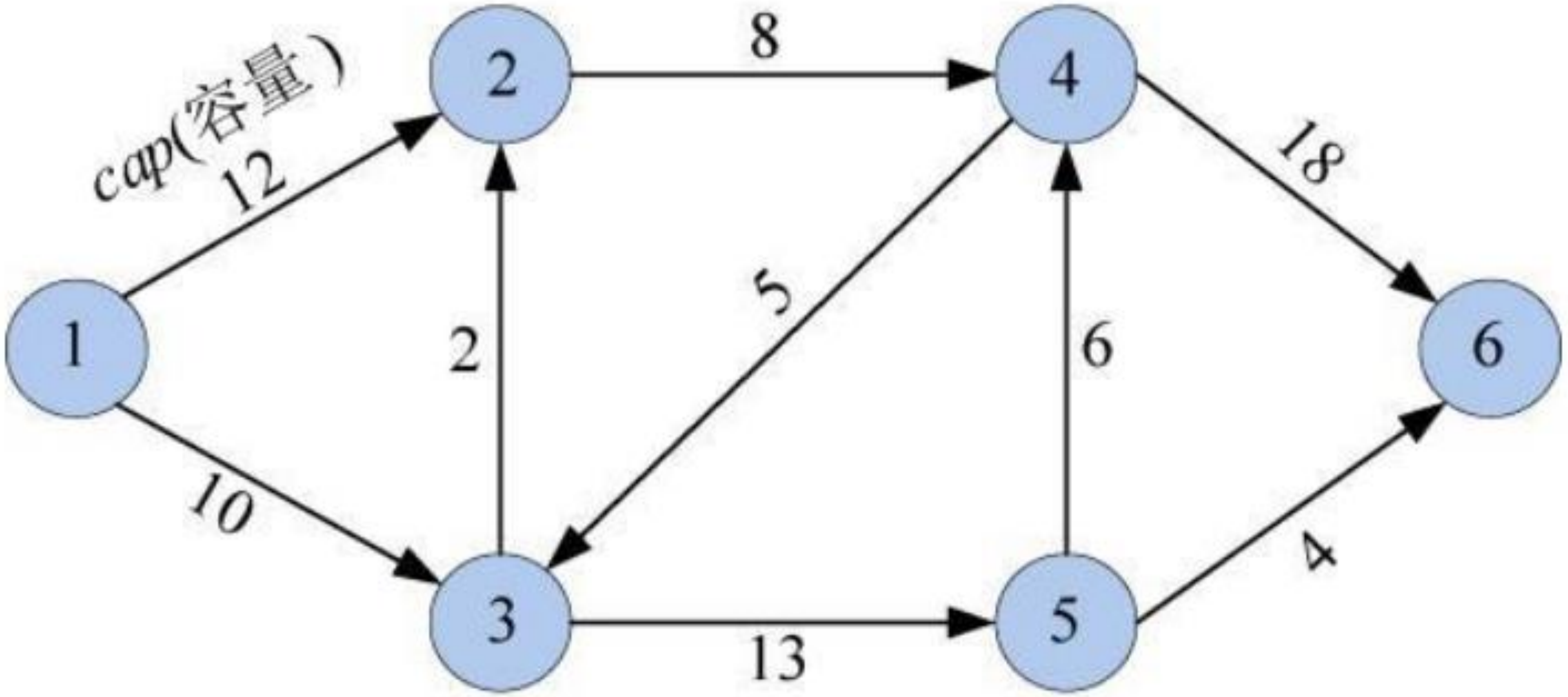
- 算法步骤:
- 采用队列*q*来存放已访问未检查的结点。布尔数组*vis[]*标识结点是否被访问过，*pre[]*数组记录可增广路上结点的前驱。*pre[v]=u*表示可增广路上*v*结点的前驱是*u*，最大流值*maxflow=0*
  - 1.初始化可行流*flow*为零流，即实流网络中全是零流边，残余网络中全是最大容量边（可增量）。初始化*vis[]*数组为 *false*，*pre[]*数组为 *−1*
  - 2.令*vis[s]=true*，*s*加入队列*q*
  - 3.如果队列不空，继续下一步，否则算法结束，找不到可增广路。当前的实流网络就是最大流网络，返回最大流值*maxflow*

1.网络流算法：最短增广路径算法

- 算法步骤：
- 采用队列*q*来存放已访问未检查的结点。布尔数组*vis[]*标识结点是否被访问过，*pre[]*数组记录可增广路上结点的前驱。*pre[v]=u*表示可增广路上*v*结点的前驱是*u*，最大流值*maxflow=0*
  - 4.（续）队头元素*new*出队，在残余网络中检查*new*的所有邻接结点*i*。如果未被访问，则访问之，令*vis[i]=true*，*pre[i]=new*；如果*i=t*，说明已到达汇点，找到一条可增广路，转向第5步；否则结点*i*加入队列*q*，转向第3步
  - 5.从汇点开始，通过前驱数组*pre[]*，逆向找可增广路上每条边值的最小值，即可增量*d*。
  - 在实流网络中增流，在残余网络中减流，*maxflow+=d*，转向第2步

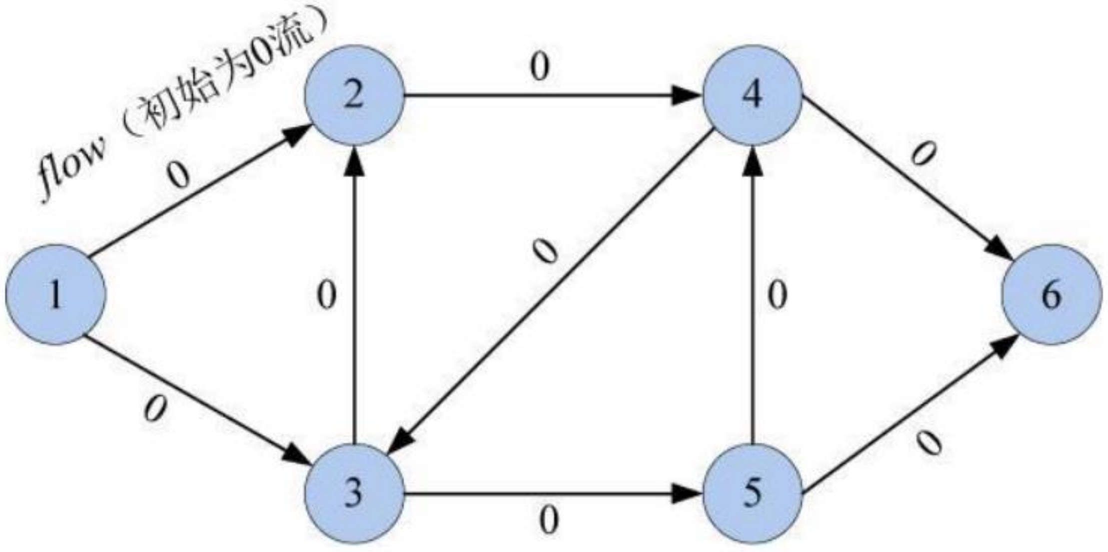
1.网络流算法：最大流算法实例

- 标记每个结点之间的最大容量*cap*，求解最大流
  - 1 号结点为源点，6 号结点为汇点

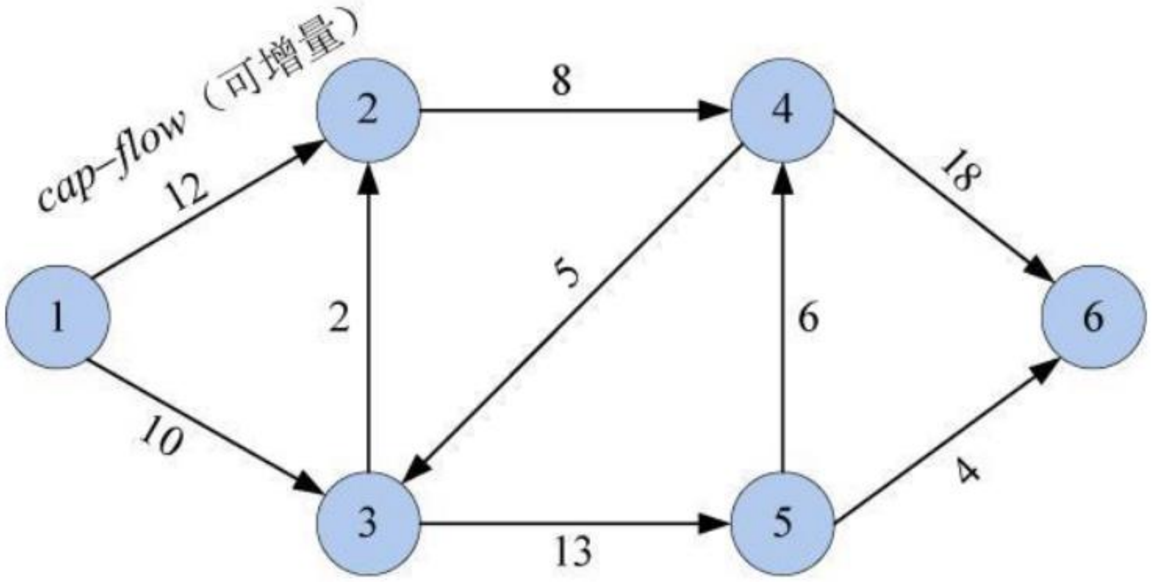


1.网络流算法：最大流算法实例

- 初始化实流网络为 0 流

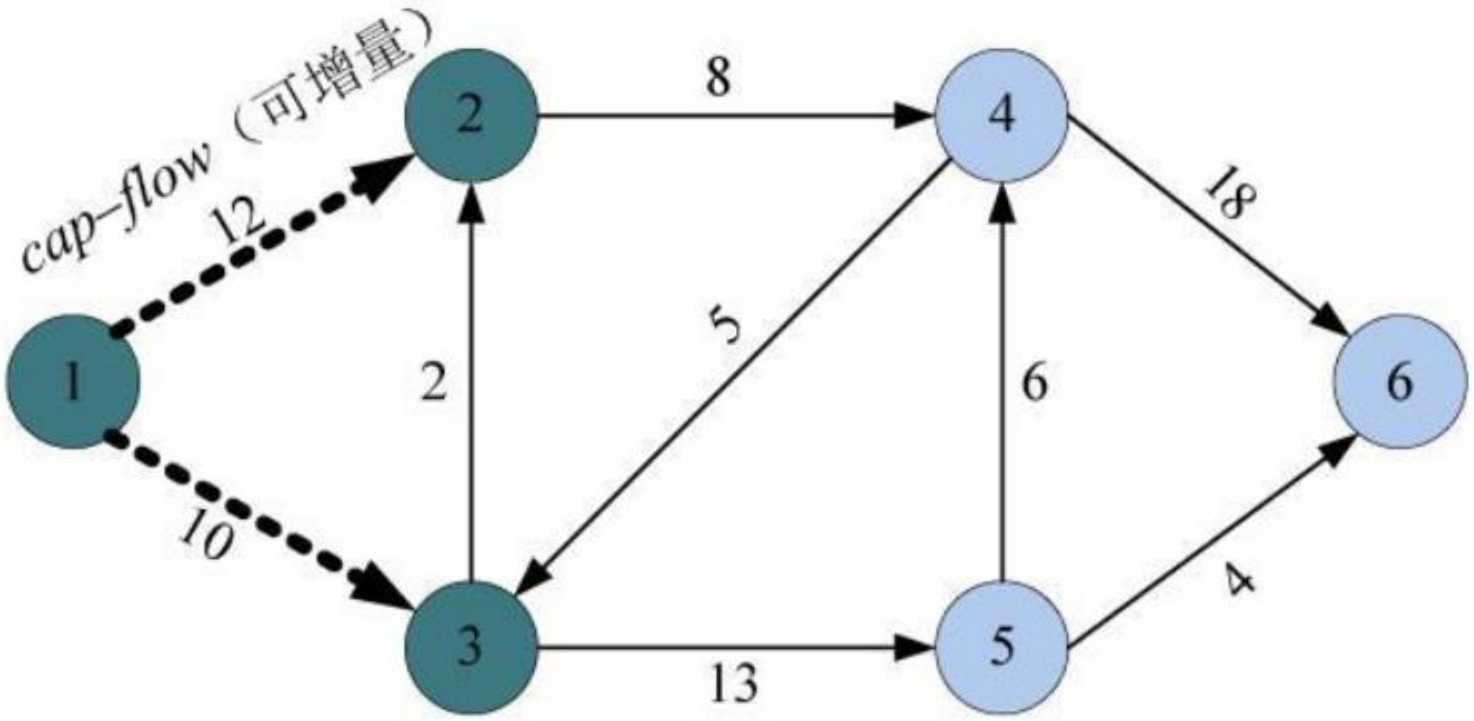


- 得到*G*对应的残余网络*G**



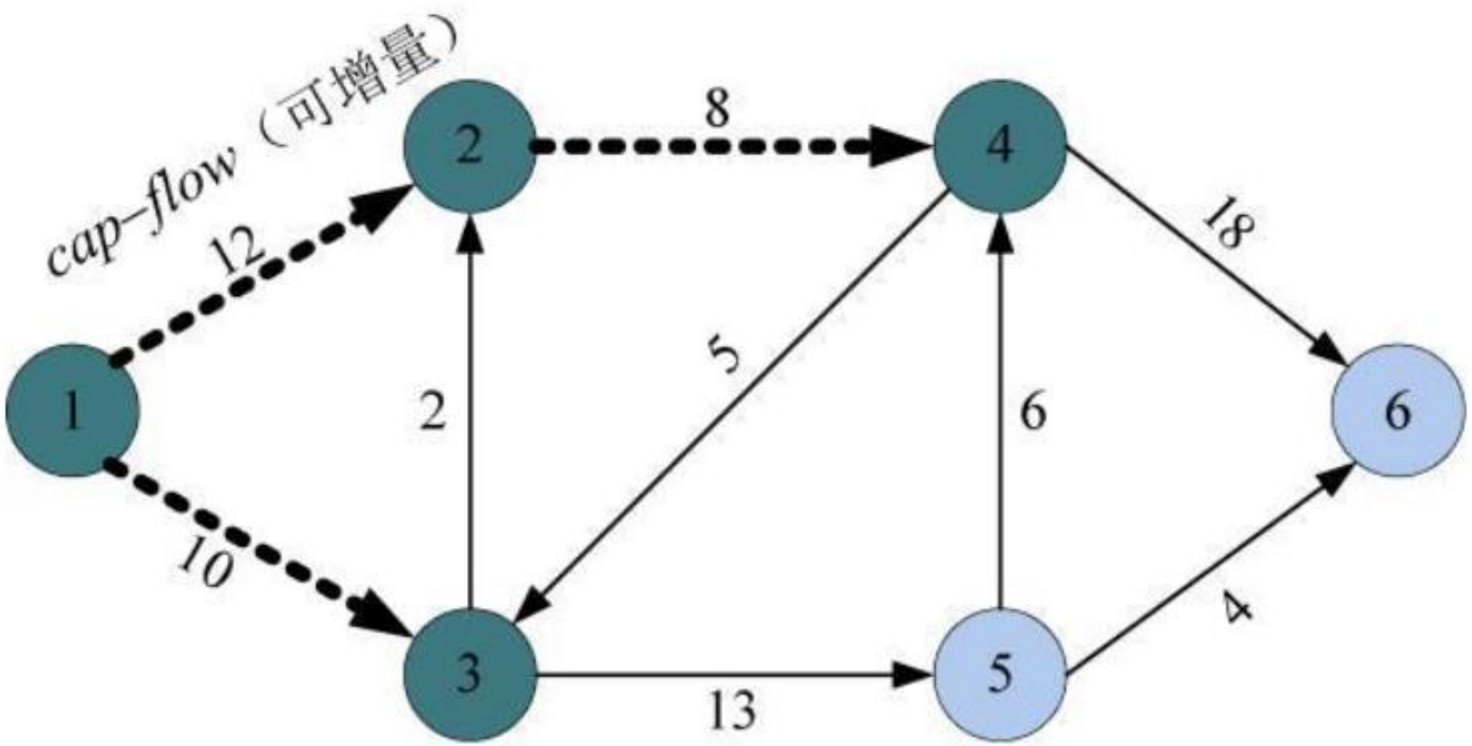
1.网络流算法：最大流算法实例

- 在残余网络*G**中依次检查 1 的所有邻接结点 2 和 3，两个结点都未被访问，令*vis[2] =true*，*pre[2]=1*，结点 2 加入队列*q*
- *vis[3]=true*，*pre[3]=1*，结点 3 加入队列*q*



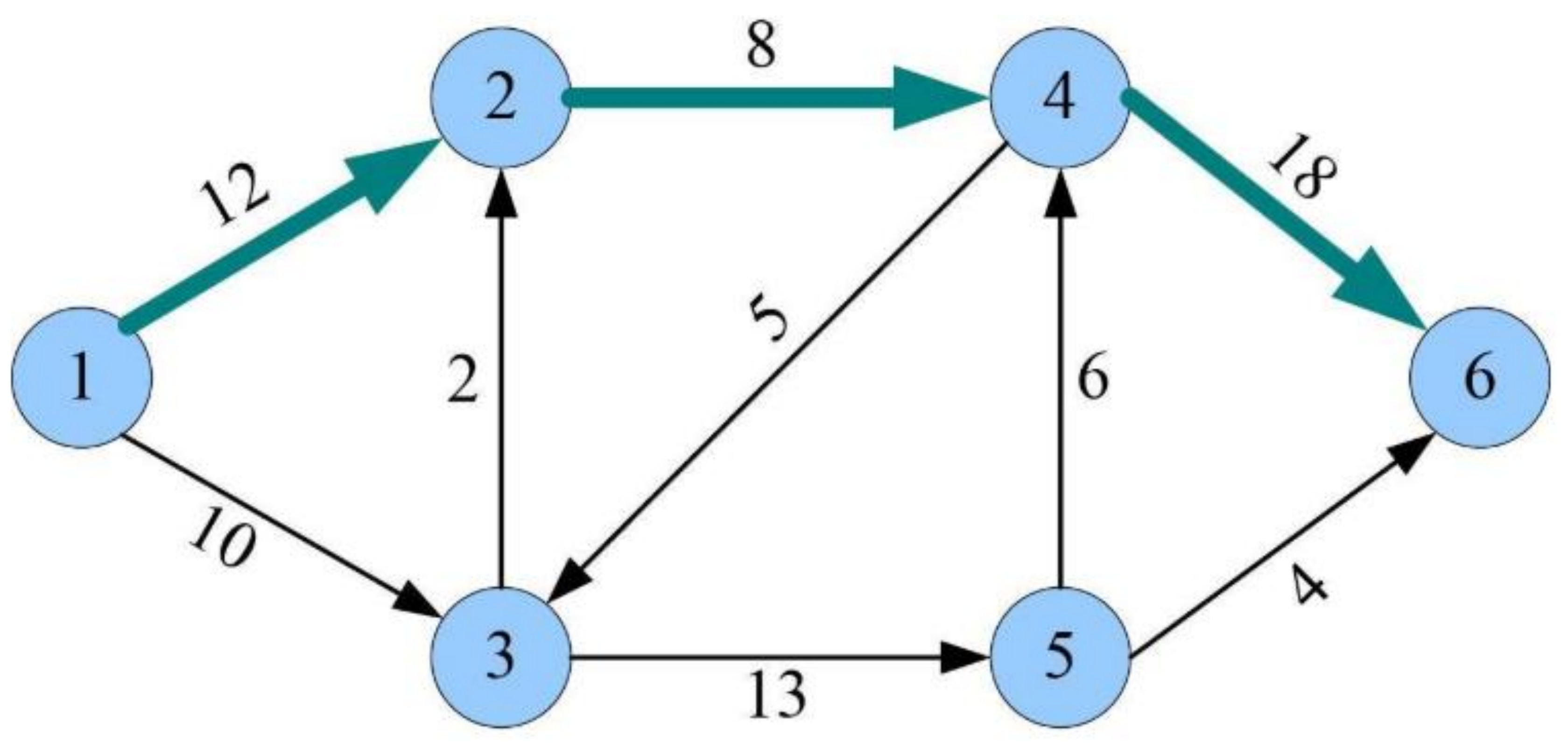
1.网络流算法：最大流算法实例

- 在残余网络中依次检查 2 的所有邻接结点 4，4 未被访问，令 *vis[4]= true*，*pre[4]=2*，结点 4 加入队列*q*



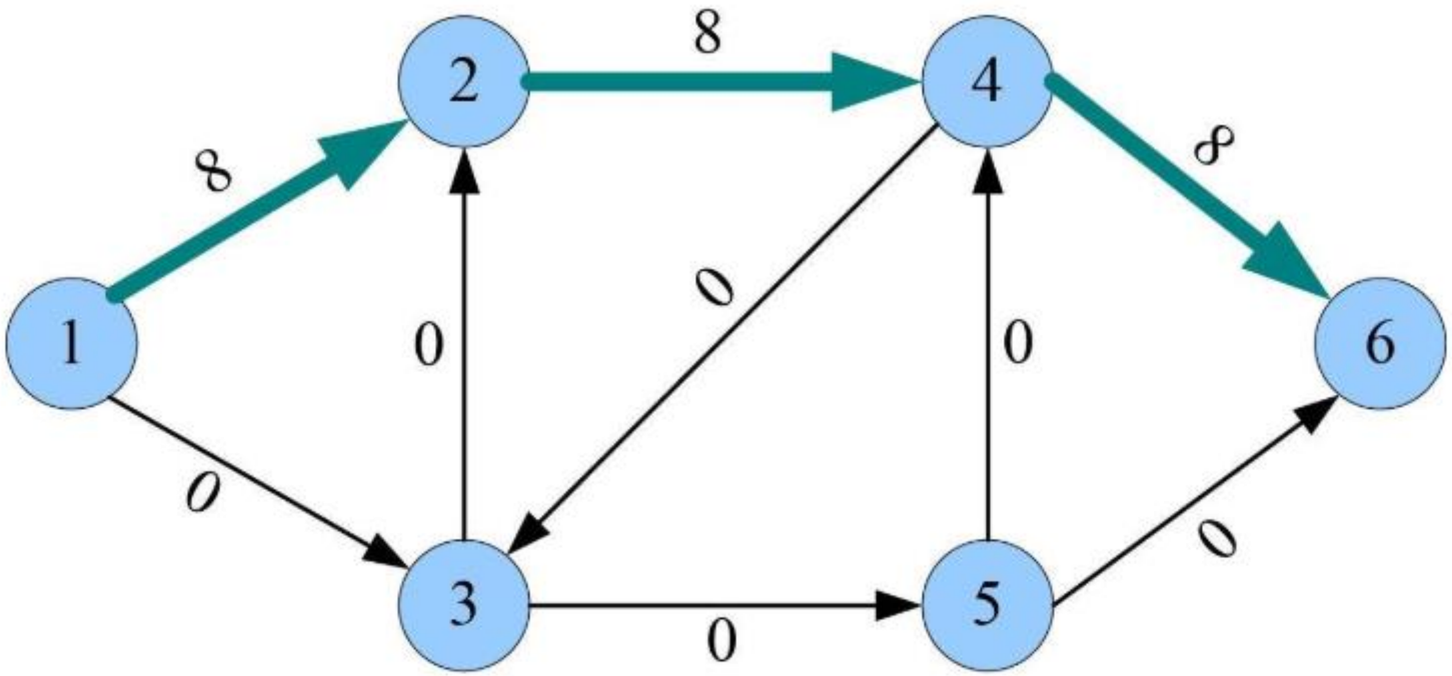
1.网络流算法：最大流算法实例

- 读取前驱数组*pre[6]=4*，*pre[4]=2*，*pre[2]=1*，即：*1—2—4—6* 找到该路径上最小的边值为 8，即可增量*d=8*



1.网络流算法：最大流算法实例

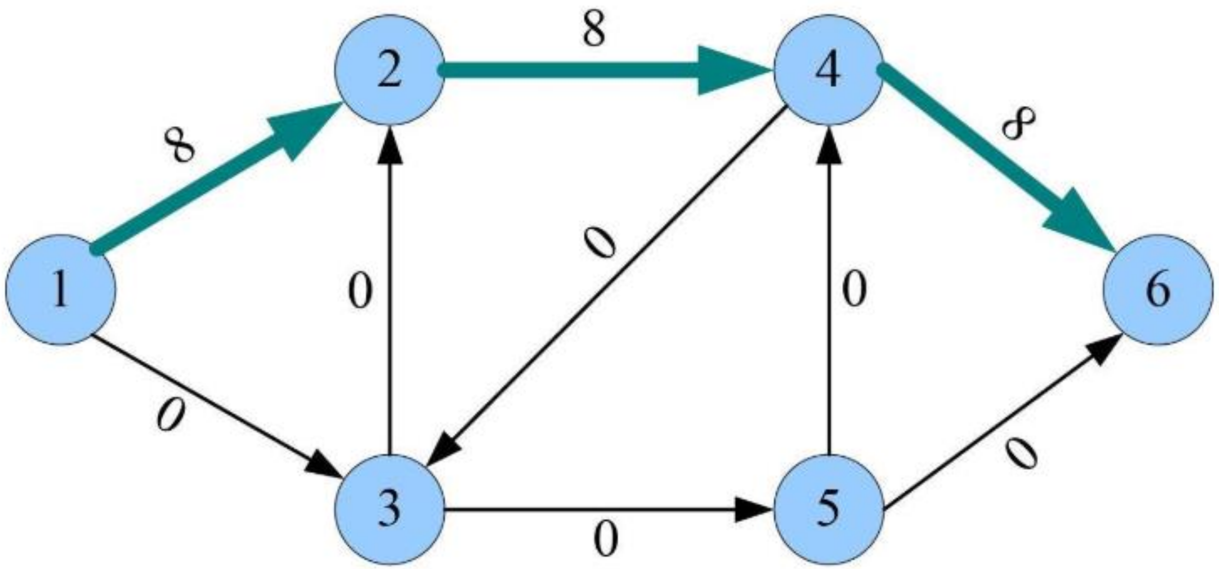
- 实流网络增流
  - 与可增广路同向的边增流d，反向的边减流d



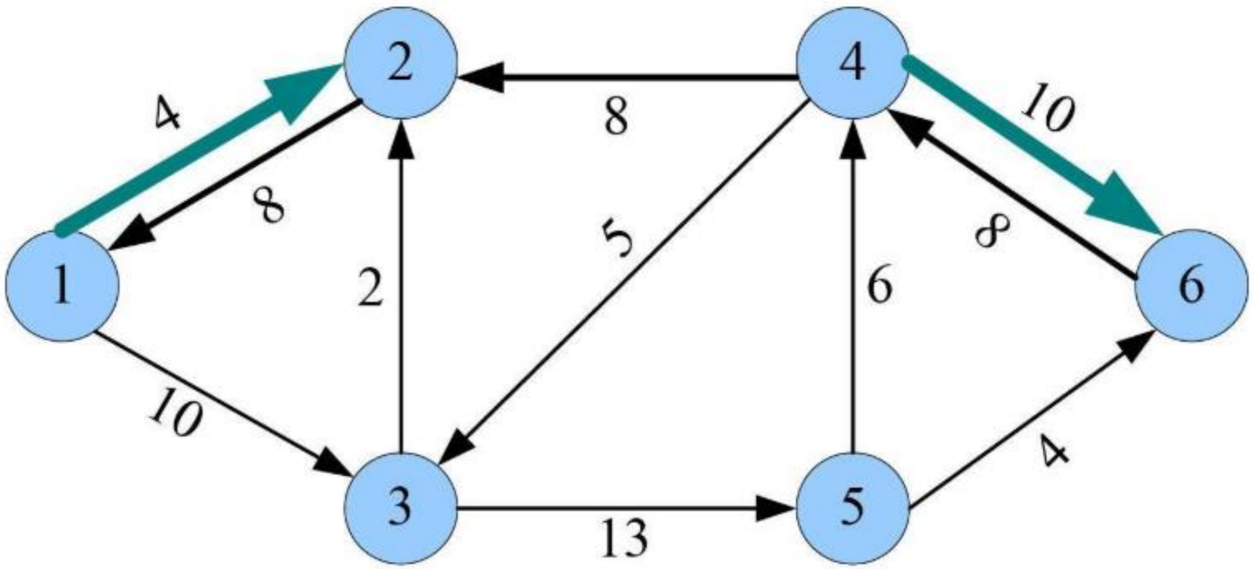
实流网络

1.网络流算法：最大流算法实例

- 残余网络减流
  - 与可增广路同向的边减流*d*，反向的边增流*d*



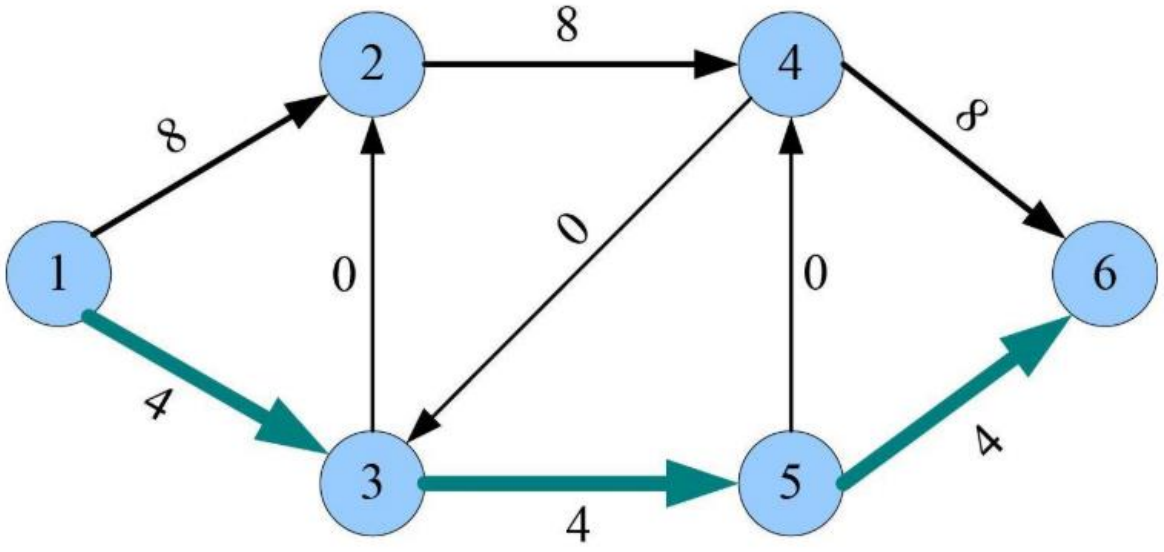
实流网络



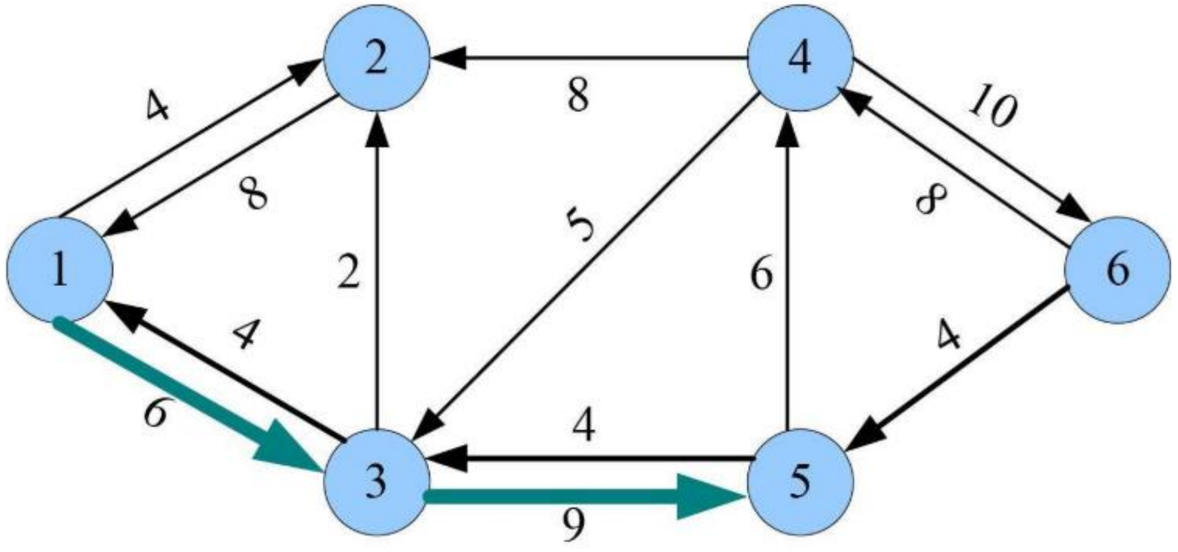
残余网络

1.网络流算法：最大流算法实例

- 继续找到第 2 条可增广路*1—3—5—6*
- 该路径上最小的边值为 4，即可增量*d=4*
- 更新增流后的实流网络和残余网络



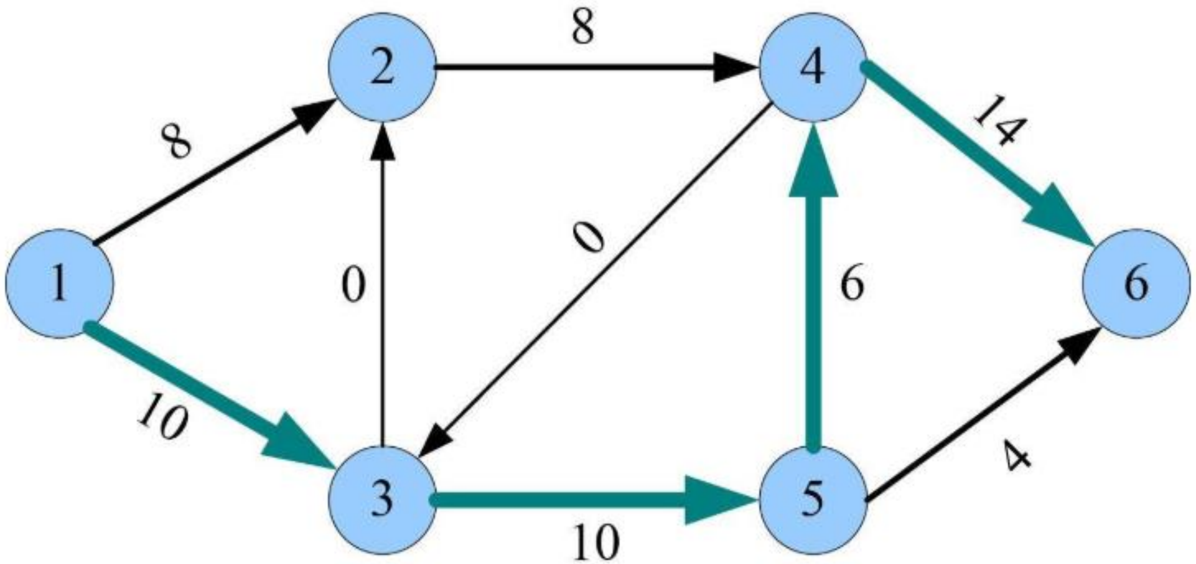
实流网络



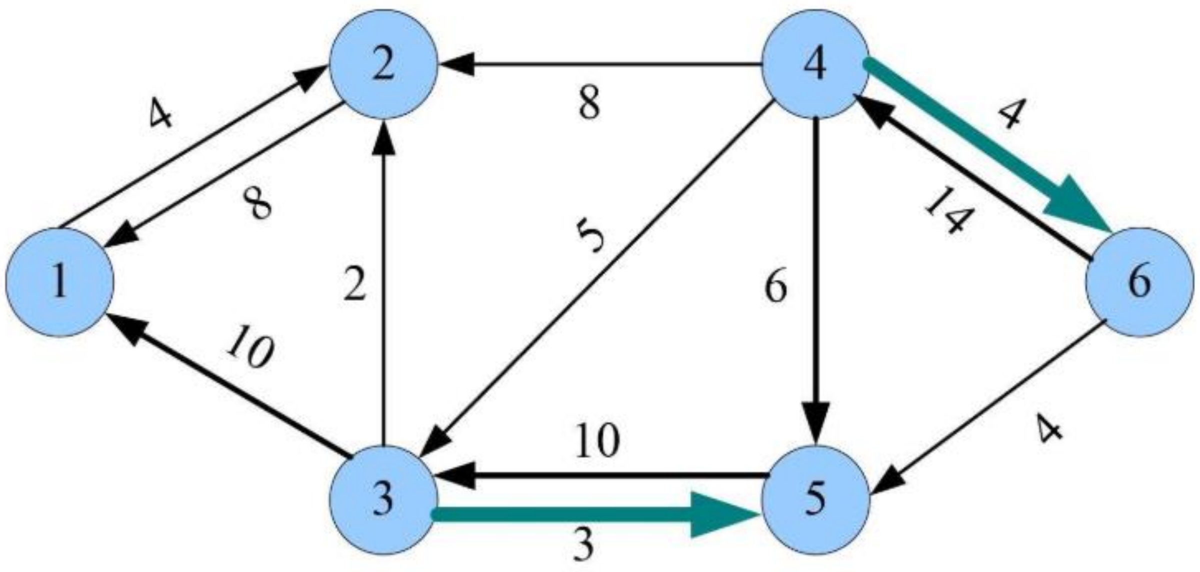
残余网络

1.网络流算法：最大流算法实例

- 继续找到第 3条可增广路*1—3—5—4—6*
- 该路径上最小的边值为 6，即可增量*d*=6
- 更新增流后的实流网络和残余网络



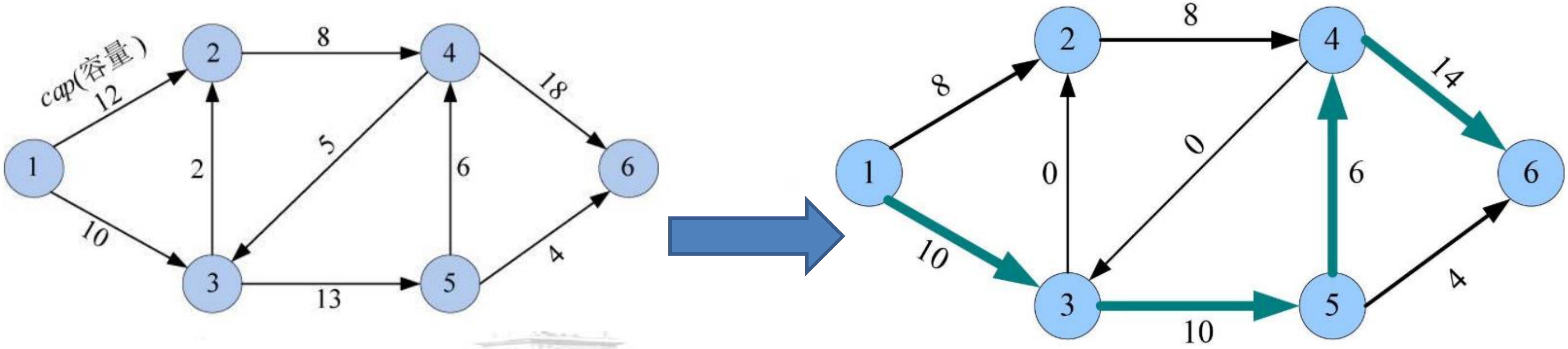
实流网络



残余网络

1.网络流算法：最大流算法实例

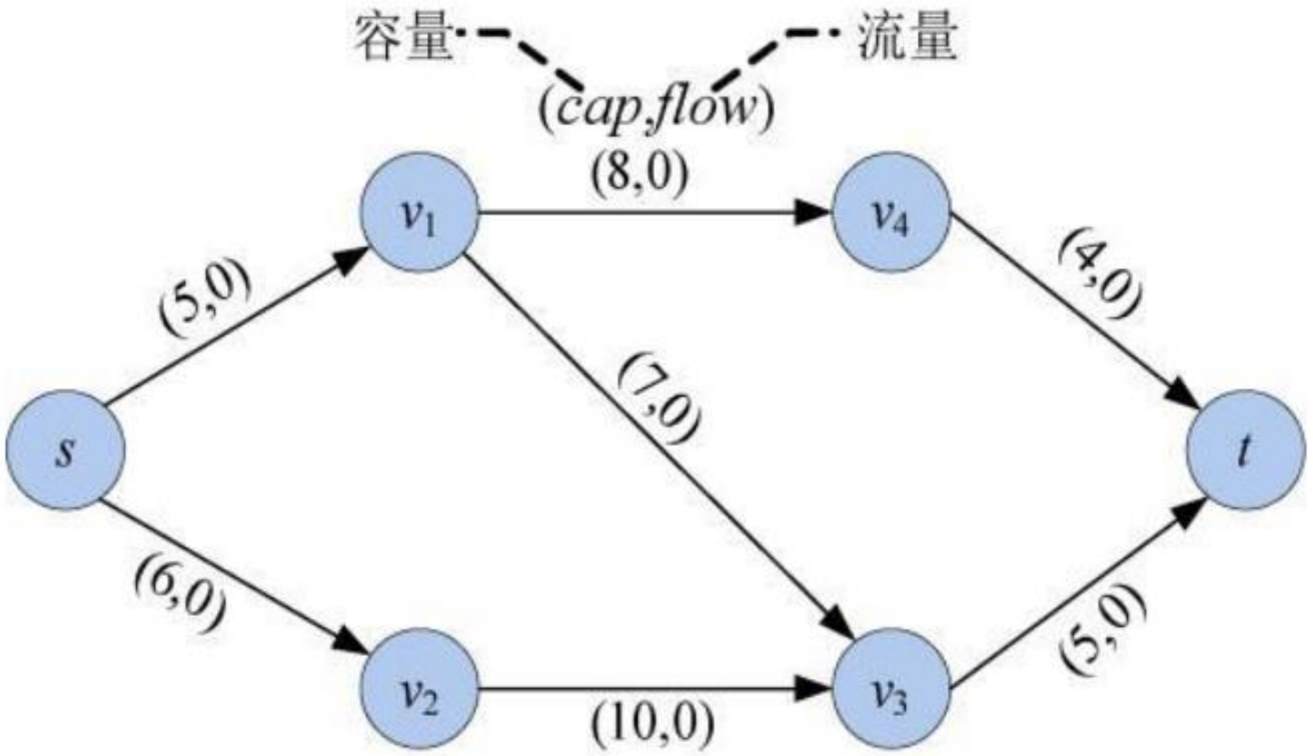
- 重复上述步骤直至无可增广路，算法结束
- 最大流值为所有的增量*d*之和 **18**



实流网络

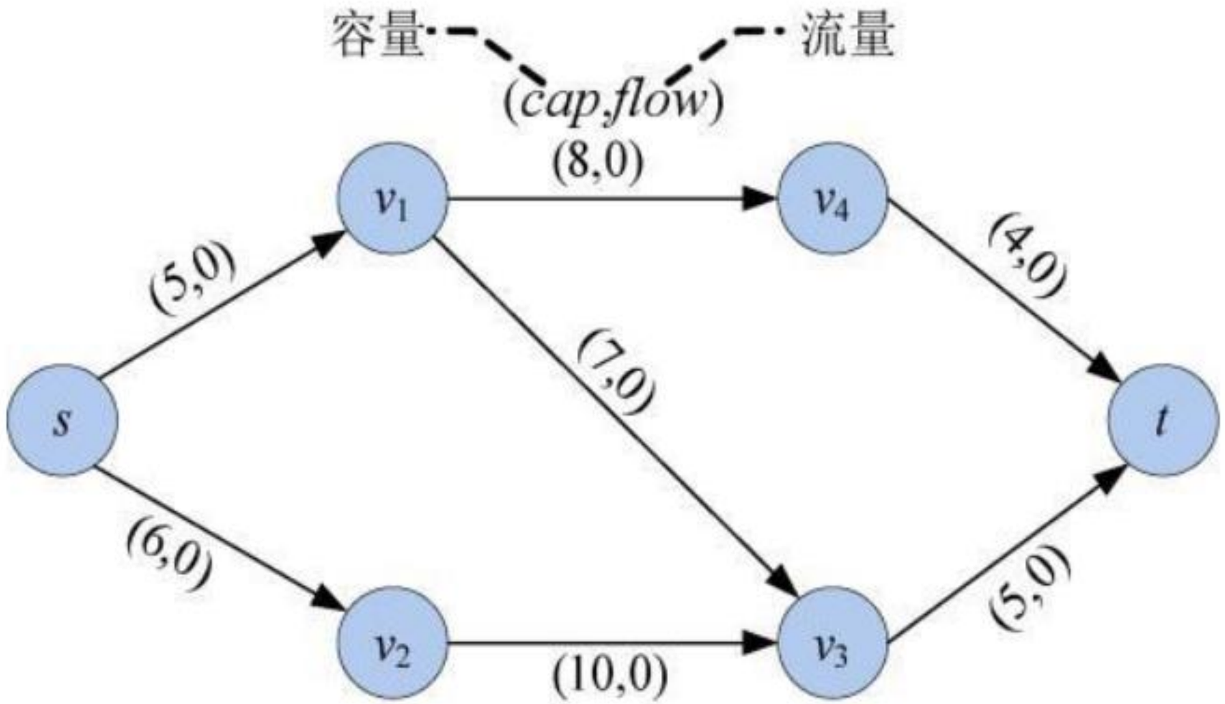
1.网络流算法

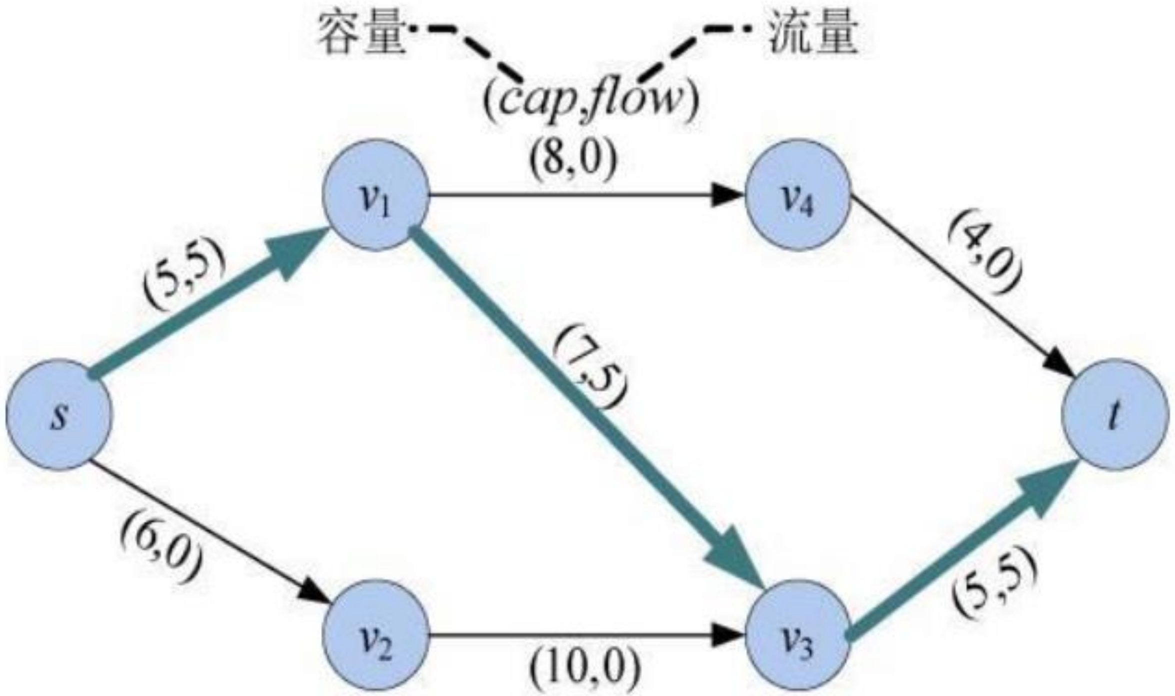
- 思考：为什么要采用残余网络 + 实流网络？
  - 为什么要在残余网络上找可增广路，直接在网络及可行流上面找可增广路可以吗？



1.网络流算法:思考

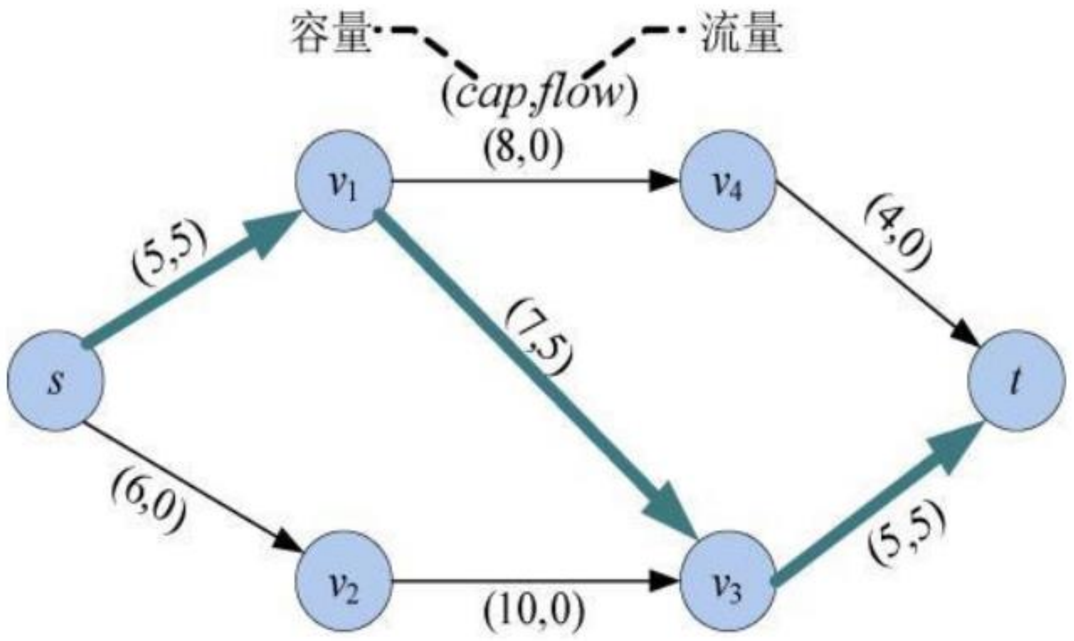
- 首先按照广搜从源点开始，沿着有可增量(cap>flow)的边搜索
- 源点s访问邻接点*v1*、*v2*，*v1*访问邻接点*v3*、*v4*，*v2*没有未被访问的邻接点，*v3*访问邻接点*t*，到达源点
- 找到一条可增广路：*s—v1—v3—t*,沿着可增广路增流，可增量*d*=5





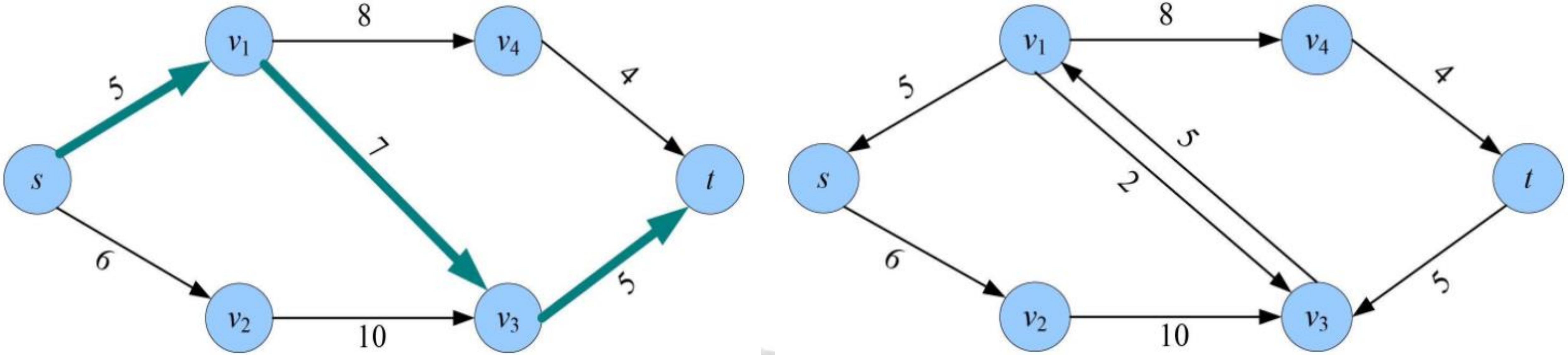
1.网络流算法:思考

- 继续广搜，沿着有可增量的边搜索。源点s访问邻接点*v2*，无法再访问*v1*，因为*s—v1*的边已经没有可增量。*v2*访问邻接点*v3*，*v3*无法再访问*t*，因为*v3—t*的边已经没有可增量。*v3*没有未被访问的邻接点，无法到达汇点，找不到从源点到汇点的可增广路。但是得到的解并不是最大流！
- 在网络G及可行流直接找可增广路，有可能得不到最大流



1.网络流算法:思考

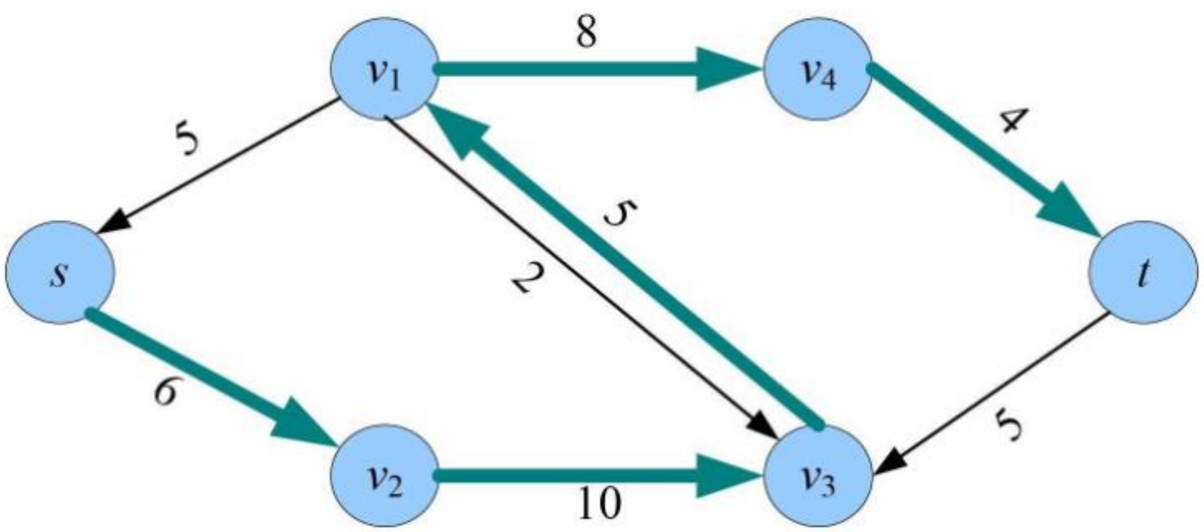
- 残余网络的作用
  - 找到一条可增广路：***s*—*v*1—*v*3—*t***。增加的流量为可增广路上每条边的最小值，可增量***d*=5**，在残余网络中，可增广路上的同向边减少流量***d***，反向边增加流量***d***

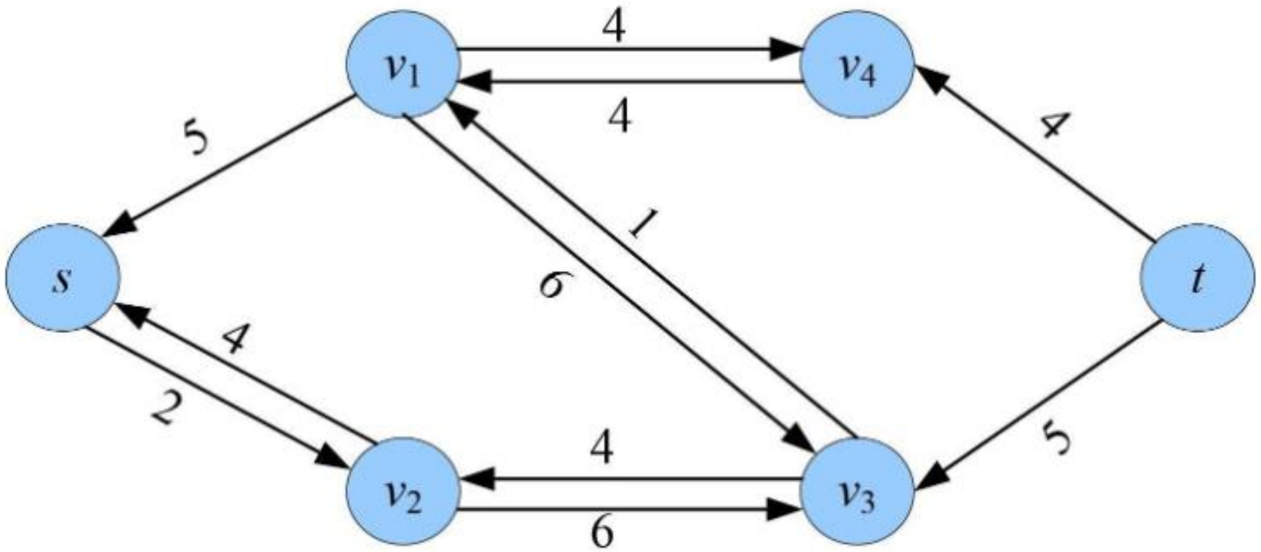


残余网络

1.网络流算法:思考

- 残余网络的作用
  - 源点s访问邻接点v2，无法再访问v1，因为v2—v1没有邻接边。v2访问邻接点v3，v3无法再访问t，因为v3—t没有邻接边。v3访问邻接点v1，v1访问邻接点v4，v4再访问t，回溯到达源点，找到一条可增广路：s—v2—v3—v1—v4—t。增加的流量为可增广路上每条边的最小值，可增量*d*=4

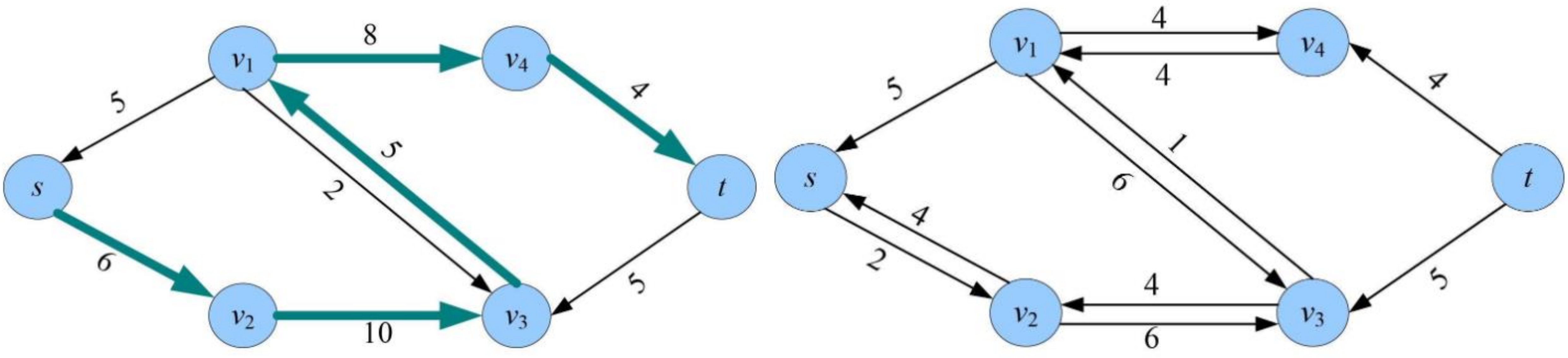




残余网络

1.网络流算法:思考

- 实流网络的作用
  - 继续搜索，找不到从源点到汇点的可增广路。已经得到最大流，最大流值为所有的增量之和，即 5+4=9。 但是，从残余网络图中无法判断哪些是实流边，哪些是可增量边。如果想知道实际的网络流量，就需要借助于实流网络。
  - 采用在残余网络中找可增广路+在实流网络中增流的方式，求解最大流



1.网络流算法：Ford-Fulkerson方法

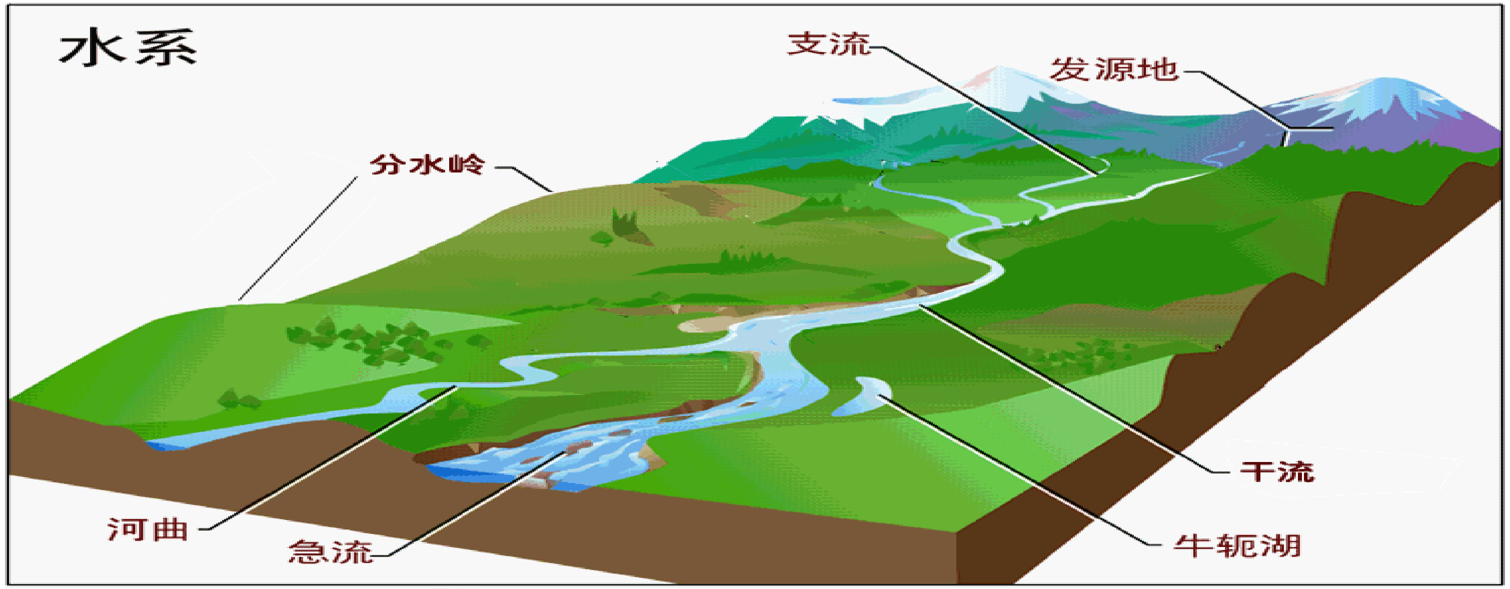
- 在剩余网络中寻找增广路径

问题：

如何判断算法结束时，确实找到了一个最大流？

1.网络流算法：Ford-Fulkerson方法

- 如何判断是否已获得最大流？
- 河水的最大流量取决于干流中河道狭窄处的通行能力

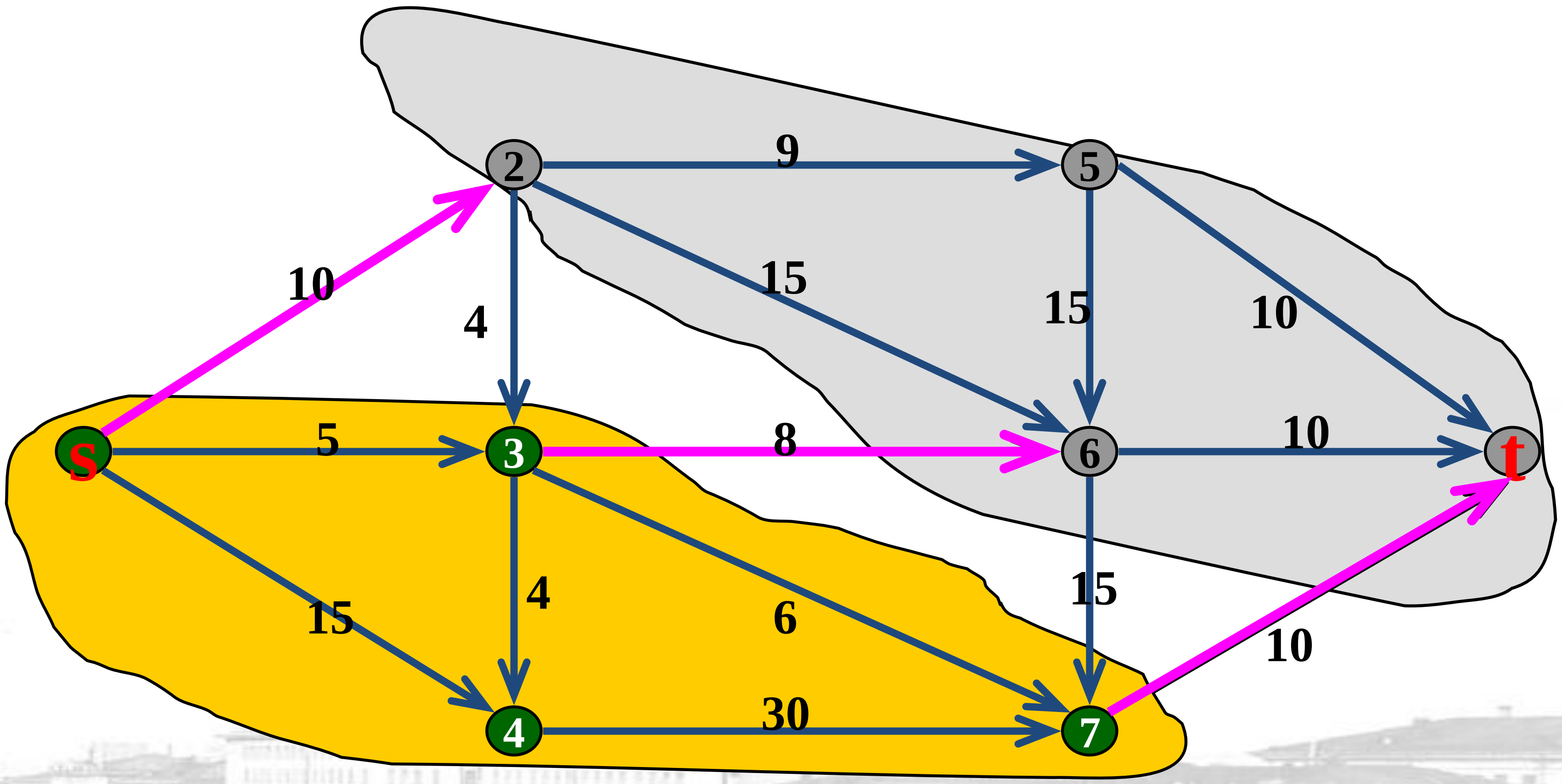


这种观察能否用于最大流问题呢？

1.网络流算法：Ford-Fulkerson方法

- 如何判断是否已获得最大流？

从*s*流到*t*的最大流量不会超过10+8+10=28



1.网络流算法：流网络的割

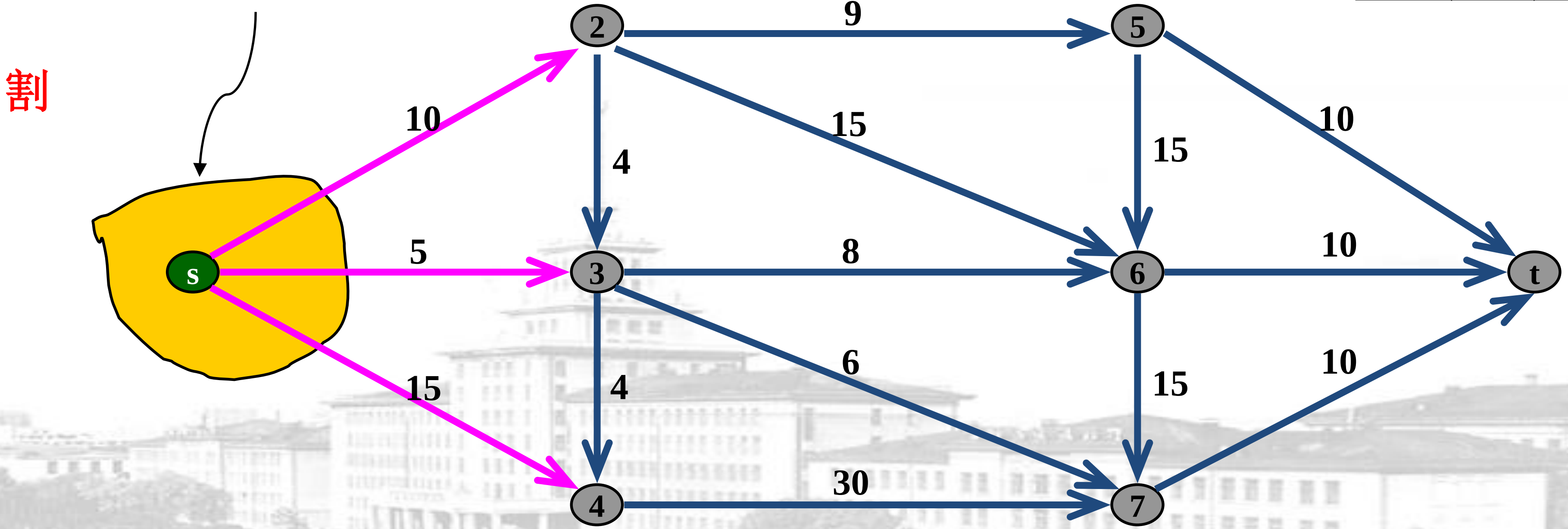
给定流网络 $G=(V,E)$,其源为s，汇为t，

G的一个割(S, T)是结点V的划分, $T=V-S$ 且$s\in S$, $t\in T$。

割的容量定义为

$$c(S,T)=\sum_{u\in S,v\in T} c(u,v)$$

$c(S,T) = 30$

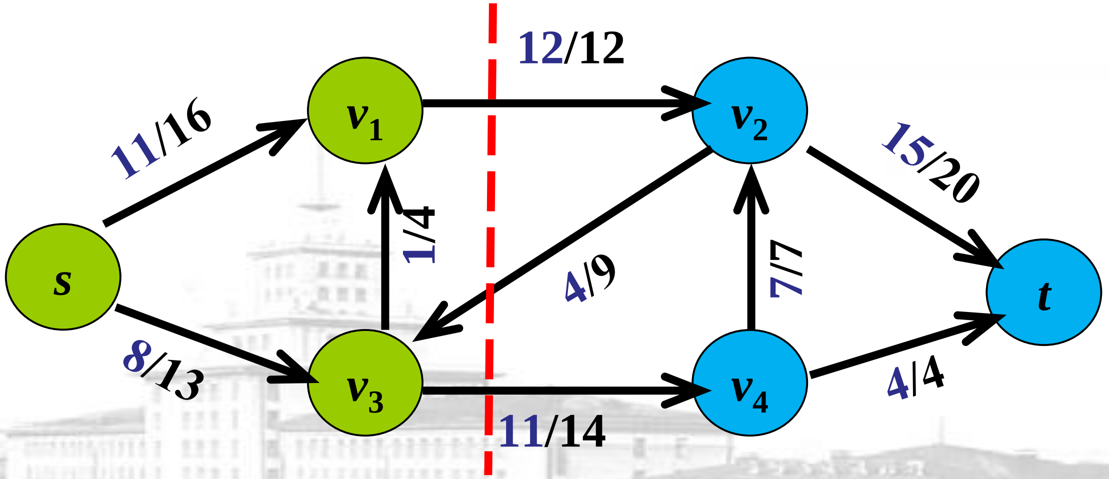


1.网络流算法：流网络的割

引理1. 设f为流网络G的一个流，该流网络的源结点为s，汇点为t，设(S,T)为流网络G的任意一个割，则横跨割(S,T)的净流量为流值$|f|$.

横跨割(S,T)的净流量定义为：

$$f(S,T)=\sum_{u\in S}\sum_{v\in T}f(u,v)-\sum_{u\in S}\sum_{v\in T}f(v,u)$$



$|f|=19$

流网络G及流f

1.网络流算法：流网络的割

推论1. 流网络G中任意流的值不能超过G的任意割的容量.

由引理知：

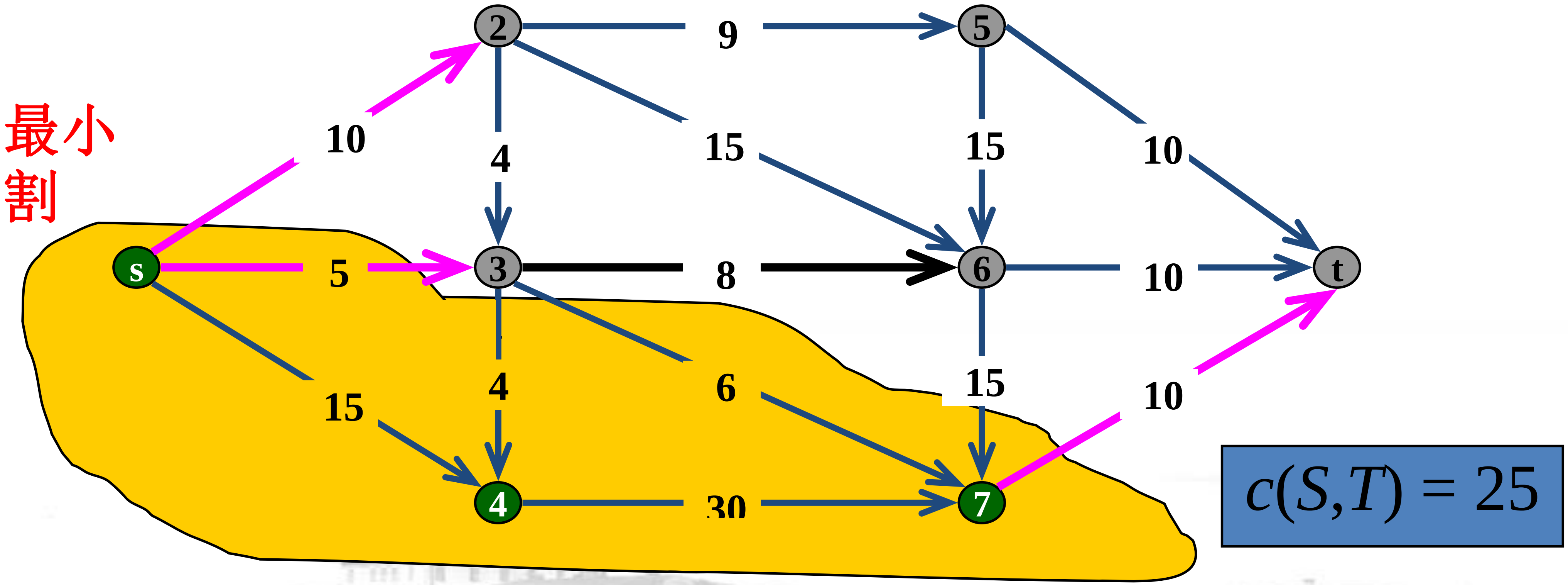
$$|f| = f(S,T) = \sum_{u\in S}\sum_{v\in T} f(u,v) - \sum_{u\in S}\sum_{v\in T} f(v,u)$$

$$\le \sum_{u\in S}\sum_{v\in T} f(u,v)$$

$$\le \sum_{u\in S}\sum_{v\in T} c(u,v) = c(S,T)$$

1.网络流算法：最小割

一个流网络的最小割是指：整个网络中容量最小的割



1.网络流算法：最大流最小割

最大流最小割定理：

设f为流网络$G(V,E)$一个流，该流网络的源结点为s，汇点为t，则下面命题等价：

1. f是G的最大流.
2. 剩余网络G_f不包含增广路径.
3. 对于G的某个划分(S, T), $|f|=c(S, T)$.

一个最大流的值实际上等于一个最小割的容量

剩余网络G_f不再包含增广路径即可

1.网络流算法：最大流最小割

- 利用Max-Min关系求解最大流问题

  1.初始化一个可行流f

  – 0-流：所有边的流量均等于0的流

  2.不断将f增大，直到f不能继续增大为止

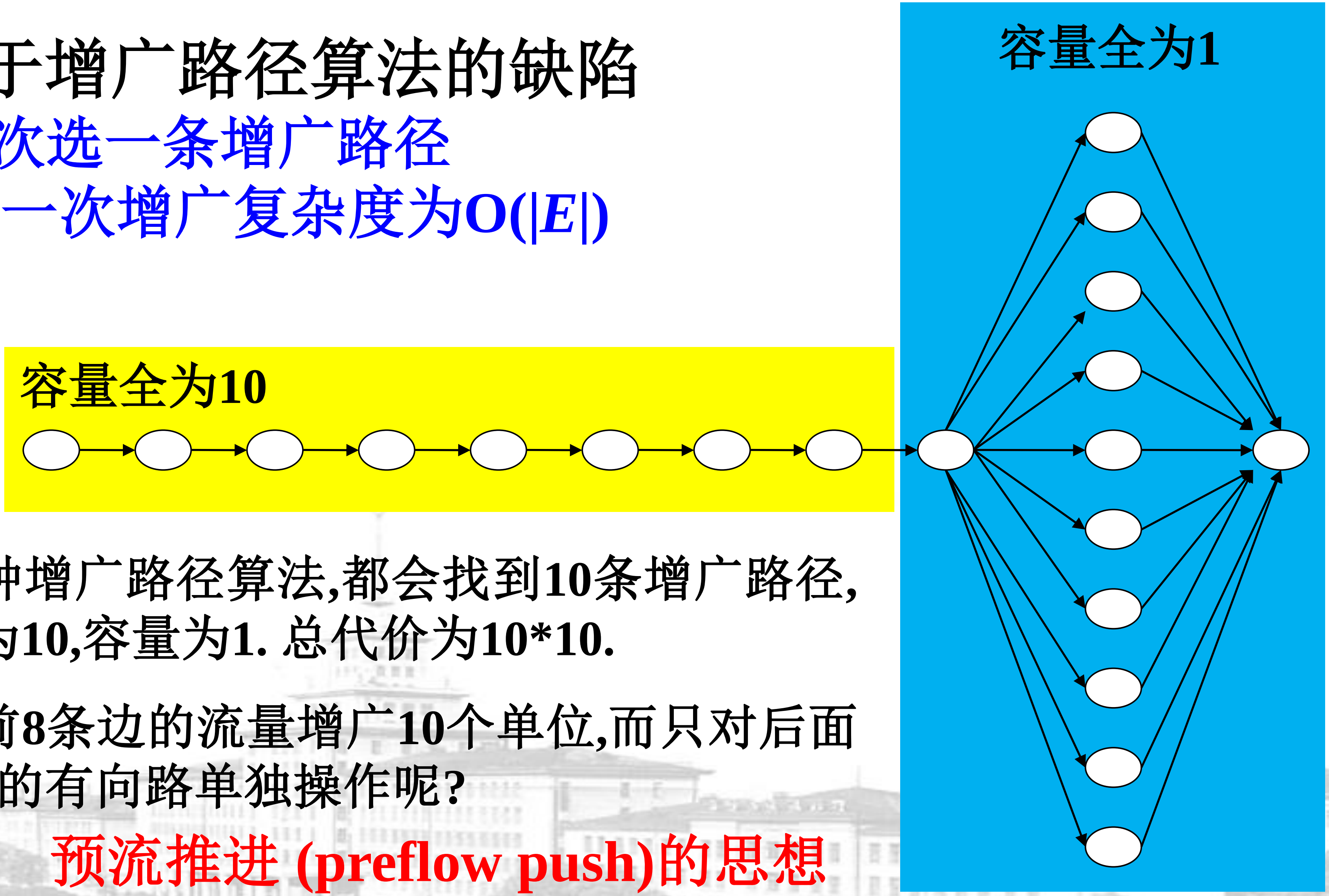
  3.找出一个割(S,T)使得$|f|=c(S,T)$

  – 由此断言f是最大流，而(S,T)是最小割

Max-Min关系提供了高效求解最大流-最小割问题的机制！

1.网络流算法：基于增广路径算法的问题

- 基于增广路径算法的缺陷
  每次选一条增广路径
  每一次增广复杂度为O(|*E*|)



无论采用何种增广路径算法,都会找到10条增广路径,每条路径长为10,容量为1. 总代价为10*10.

能否直接将前8条边的流量增广10个单位,而只对后面长为2的不同的有向路单独操作呢?

预流推进 (preflow push)的思想

1.网络流算法：基于增广路径算法的问题

- 预流推进 (preflow push)与增广路径算法的区别
  - 增广路径算法
    - 从全局考虑，沿着一条从*s*到*t*的路径推送流量
    - 要求结点流量守恒：流入结点*i*的流量＝流出结点*i*的流量
  - 预流推进算法
    - 仅考虑结点局部信息，每次只在一条边上最大可能地推送流量
    - 没有结点流量守恒约束

流入节点 i 的流量 $\geq$ 流出节点 i 的流量$(i \neq s)$.

1.网络流算法：预流(preflow)

- 在每个中间阶段，允许到达结点的流量比离开结点的流量多
- 预流：是一个函数*f*：$V \times V \to R$，满足
  - 容量约束性质
  - 弱化的流量守恒性质：$\forall u \in V\text{-}\{s\}$，

$$\sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) \geq 0$$

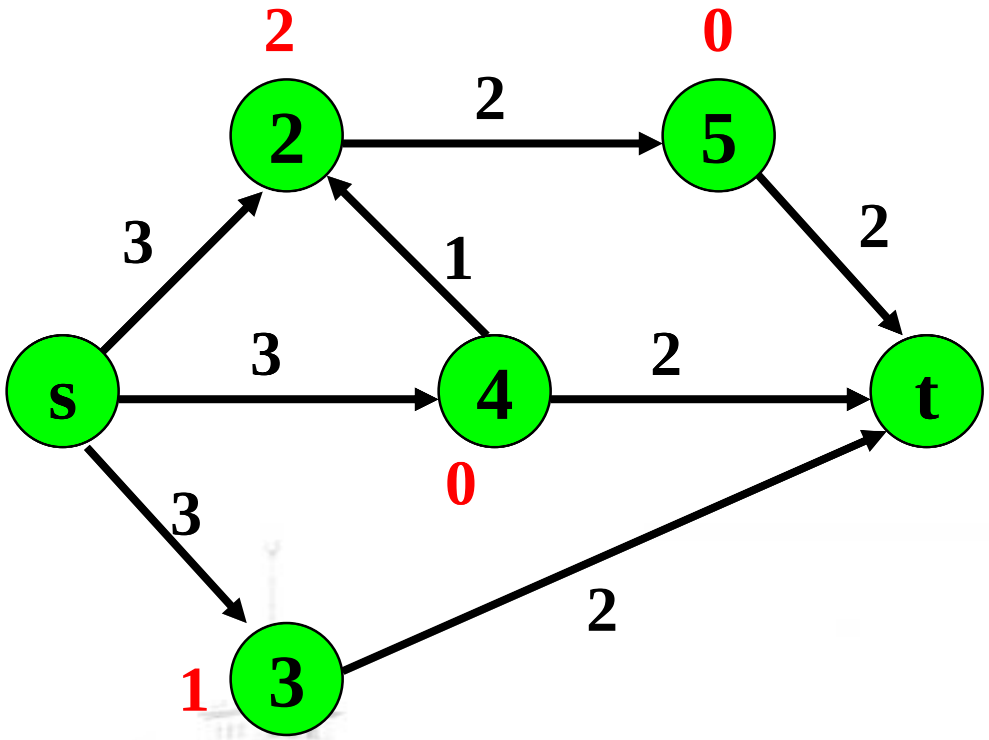
    - 即：进入一个结点的流可以超过流出该结点的流

$e(u) = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v)$ 为进入结点**u**的超额流

如果对于$\forall u \in V\text{-}\{s\}$，$e(u)>0$，则称结点**u**溢出

1.网络流算法：预流(preflow)

- 一个可行的预流

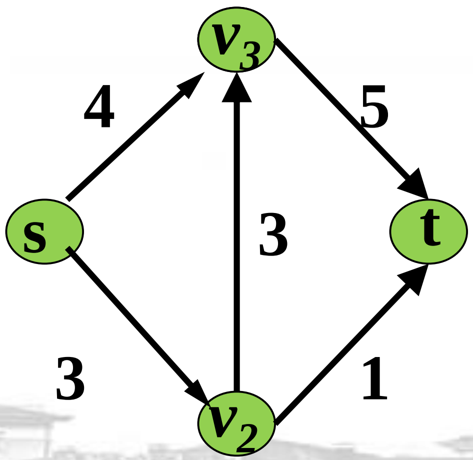


每个节点$u(j\neq s)$的超额流 $e(u)$=流入u的流量一流出u的流量

总超额流=流出s与流入t的流量之差。

1.网络流算法：预流(preflow)

- 将流网络看成管道网络
  - 结点具有库存能力
  - 液体将怎么流动？
  - 引导机制：结点设置不同高度



1.网络流算法：预流(preflow)

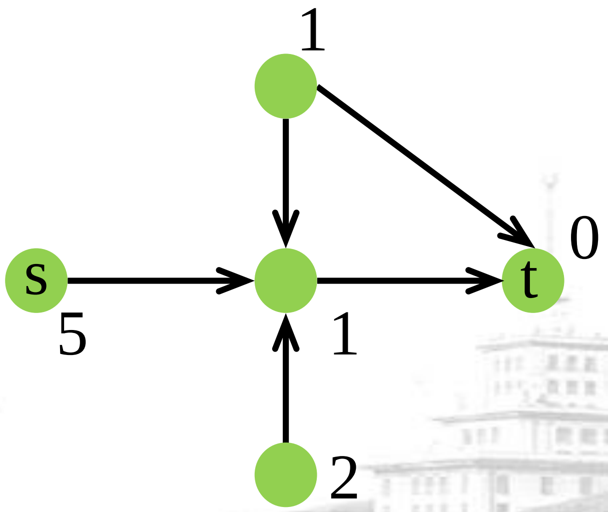
- 结点的高度
  - 设$G(V,E)$是一个源结点为s，汇点为t的流网络，f为G的一个预流。如果函数h：$V \rightarrow N$，满足下面条件，则h是一个高度函数。
    - $h(s)=|V|$, $h(t)=0$，并且
    - 对于所有的边$(u, v) \in E_f$，有$h(u) \leq h(v) + 1$

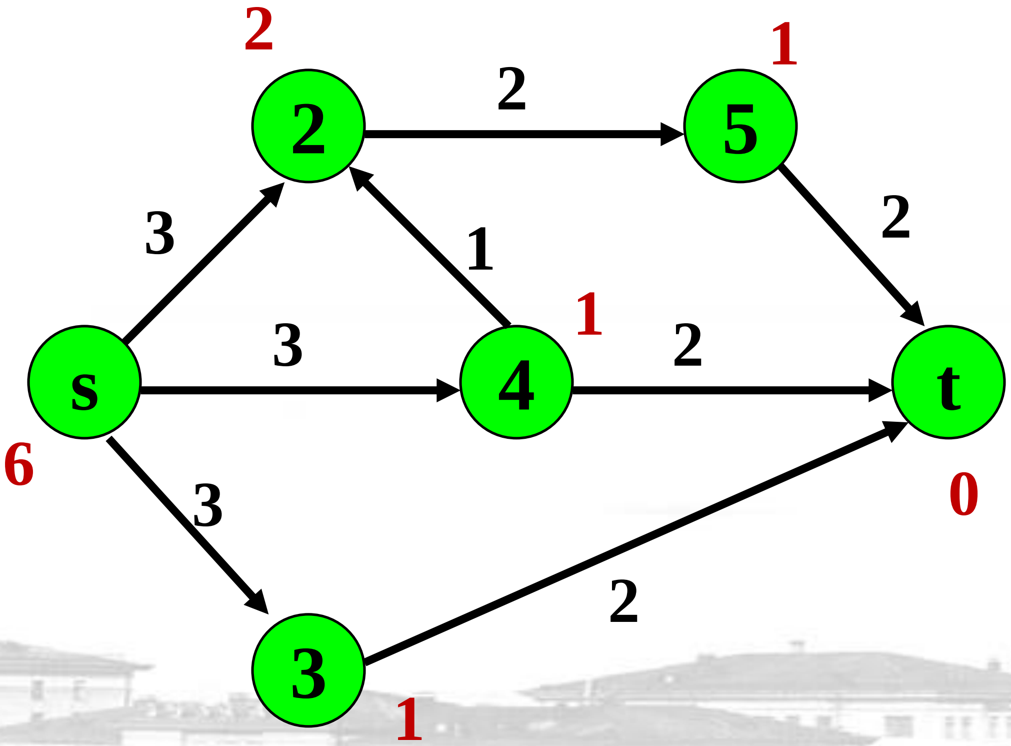
      结点u至多比v高一个高度
- 如果一个结点溢出了，那么它的超额流只能流向高度标号比自己低的结点(水往低处流)
- 对于任意结点u,通常用其在剩余网络中到汇点t的最短距离$d(u)$来表示其高度
  - 满足高度函数h的两个条件

1.网络流算法：预流(preflow)

引理3：设$G(V,E)$为一个流网络，f为一个预流，设h为V上的高度函数。对于任意两个结点$u,v \in V$，如果$h(u)>h(v)+1$，则(u,v)不是剩余网络中的一条边

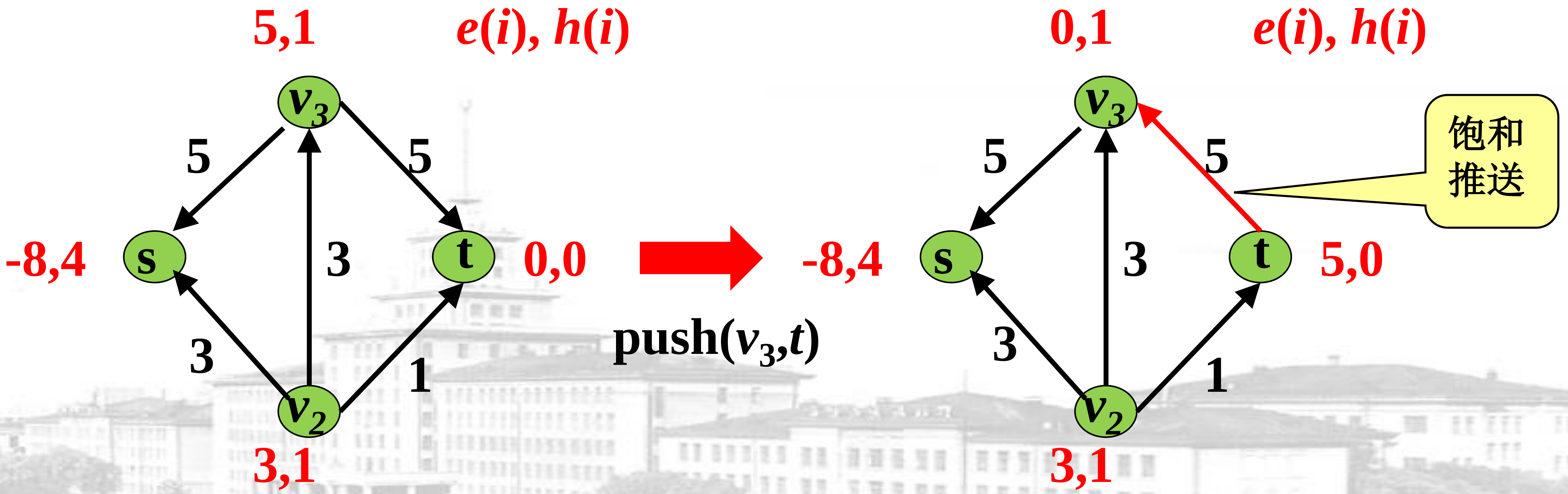
由高度函数的定义可直接得到



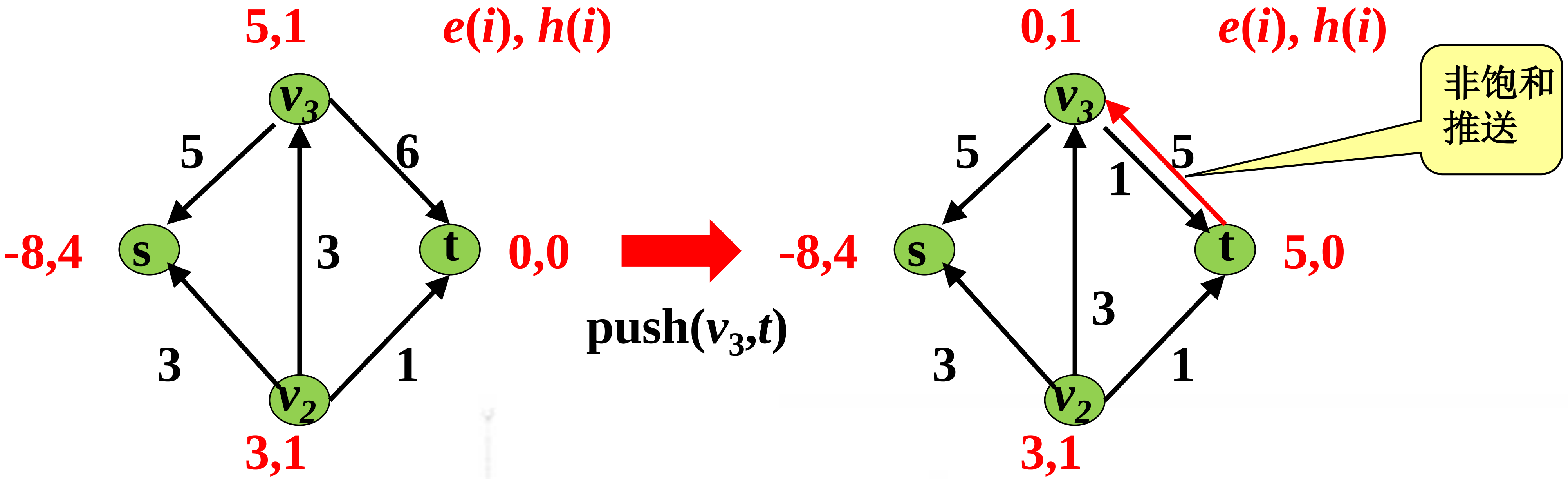


1.网络流算法：两个基本操作

- 推送操作：push(u,v)
  - 饱和推送
    - 如果push(u,v)操作后，$c_f(u,v)=0$. 称边(u,v) 达到饱和状态
    - 如果一条边达到饱和状态，它将从剩余网络中消失
  - 非饱和推送

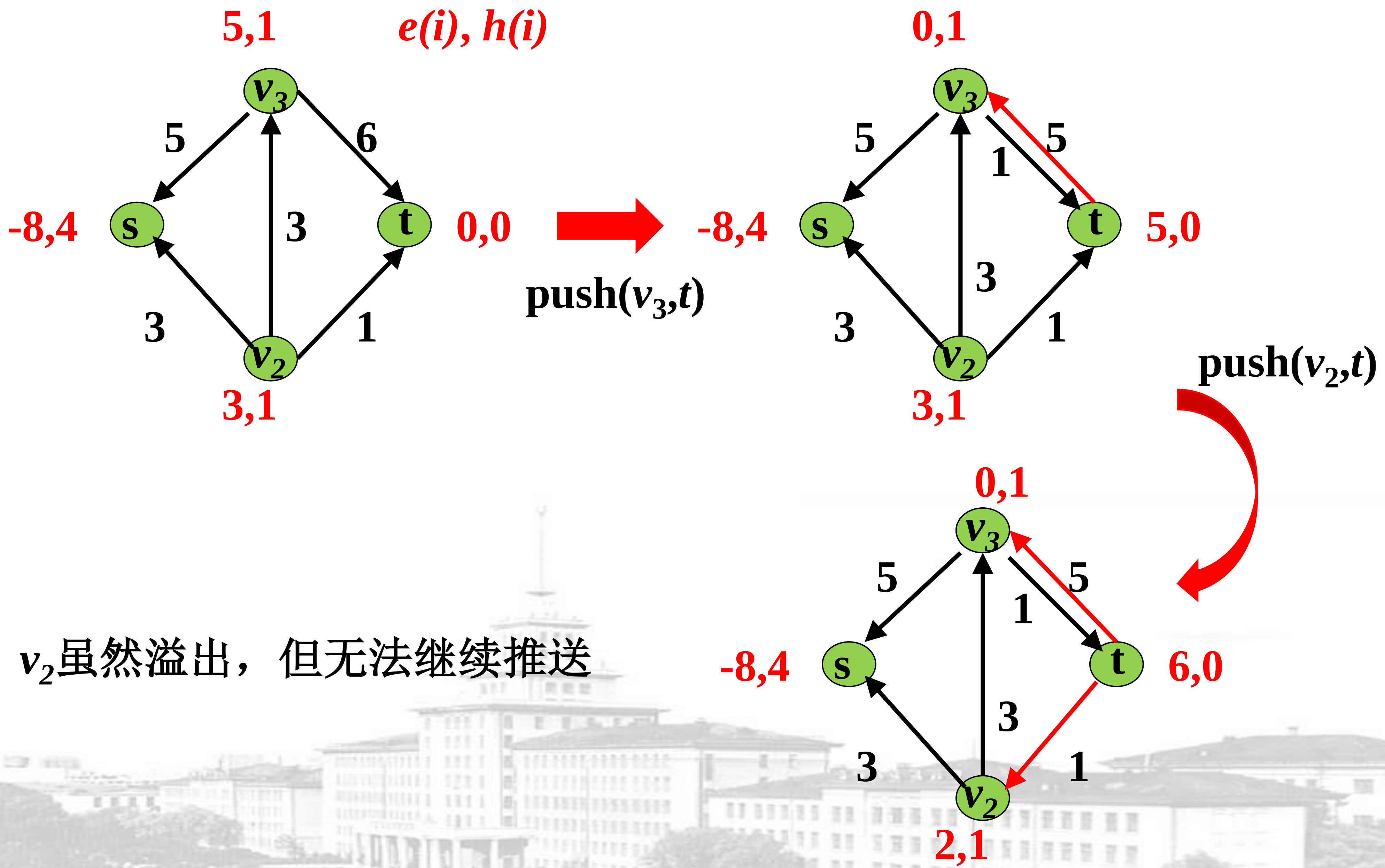


1.网络流算法：两个基本操作



引理4：在从u到v的一个非饱和推送后，结点u将不再溢出

1.网络流算法：两个基本操作

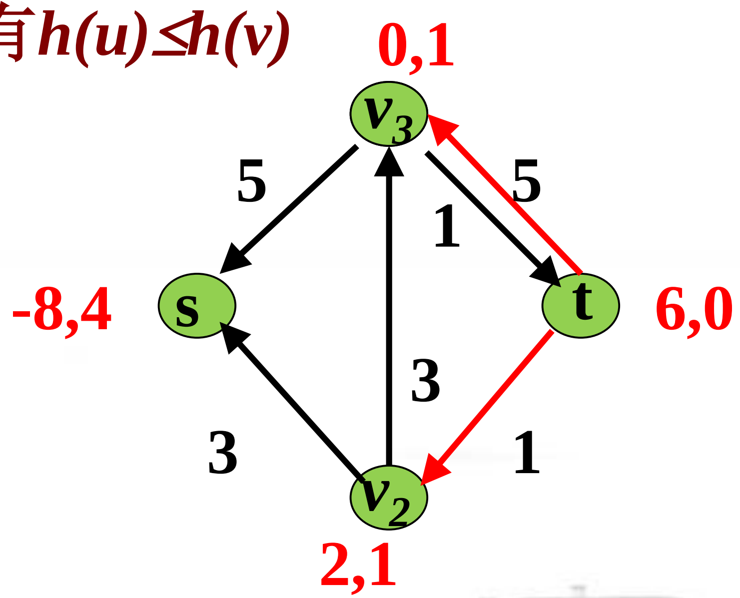


v_2虽然溢出，但无法继续推送

1.网络流算法：两个基本操作

- 重贴标号操作：**Relable(*u*)**
  - 对一个溢出结点进行重贴标号的操作: 提高结点高度
  - 操作执行的前提条件：
    - 结点*u*是一个溢出结点($e(u)>0$)，
    - 对于剩余网络中所有的边$(u,v)\in E_f$，有$h(u)\le h(v)$

Relable (*u*)
//提高结点*u*的高度
1. $h(u)=1+\min\{h(v): (u,v)\in E_f\}$



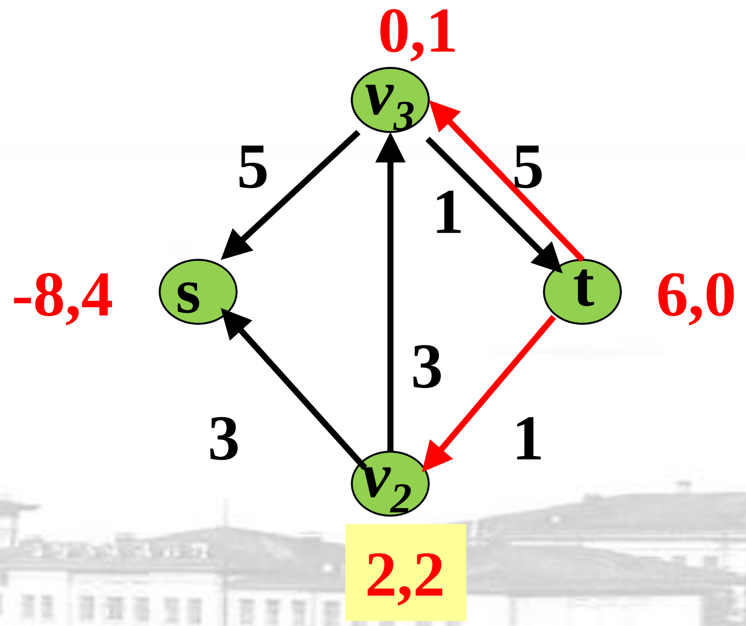
使用了relable操作后,至少存在一个(u,v)满足$h(u)=h(v)+1$

1.网络流算法：两个基本操作

- 重贴标号操作：**Relable(u)**
  - 对一个溢出结点进行重贴标号的操作: 提高结点高度
  - 操作执行的前提条件：
    - 结点u是一个溢出结点($e(u)>0$)，
    - 对于剩余网络中所有的边$(u,v)\in E_f$，有$h(u)\le h(v)$

Relable (u)
//提高结点u的高度
1. $h(u)=1+\min\{h(v): (u,v)\in E_f\}$

例如：对结点v_2重贴标号



1.网络流算法：Push-relable算法

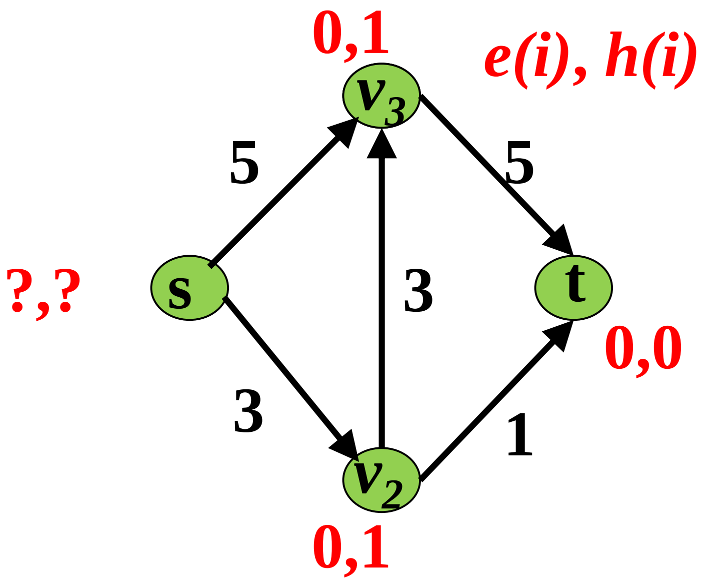
- 初始化预流：
  - $\forall u \in V\text{-}\{s\}$, $h(u)=d(u)$, $e(u)=0$

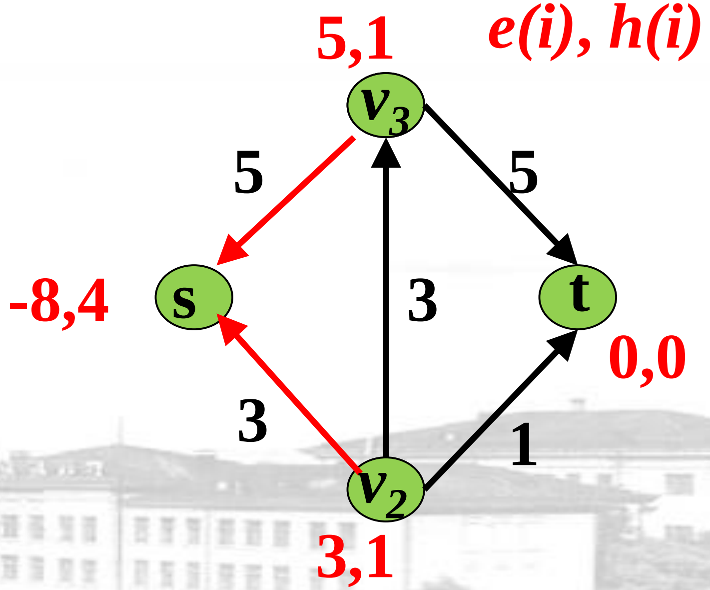
    $\forall (u,v)\in E$, $f(u,v)=0$
  - $h(s)=|V|$

    $\forall (s,v)\in E$,

    $f(s,v) = c(s,v)$, $e(v)= c(s,v)$,

$$e(s) = \sum_{(s,v)\in E} -c(s,v)$$





从s出发的所有边都充满流，且这些边已达到饱和状态而其它边上都没有流

1.网络流算法： Push-relable算法

Generic push-relable(*G*)

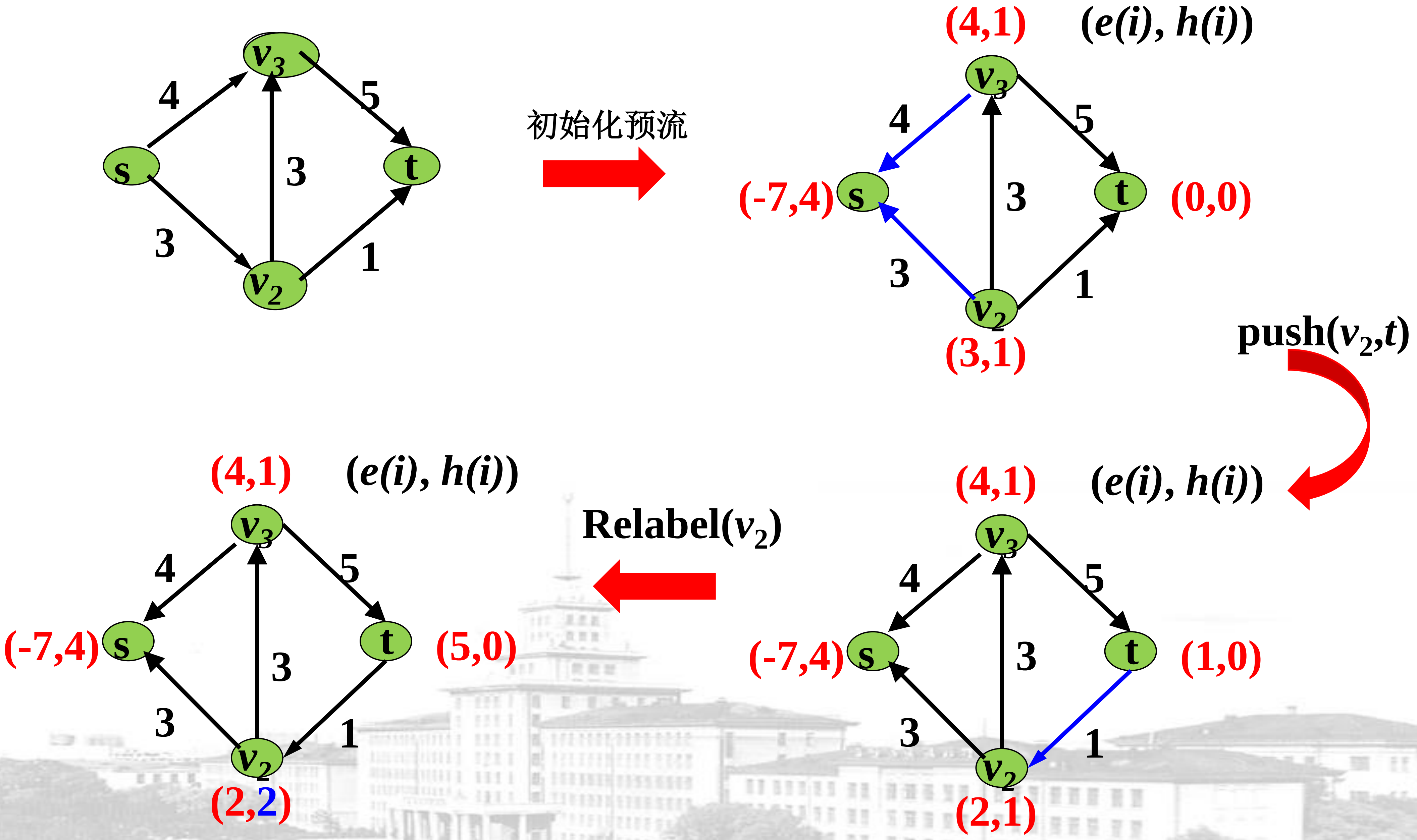
1. 初始化预流；
2. **While** 若存在一个可行的**push**或**relable**操作
3. 选择一个**push**或**relable**操作，执行之

只要存在溢出结点，算法就会执行**push**或**relable**操作

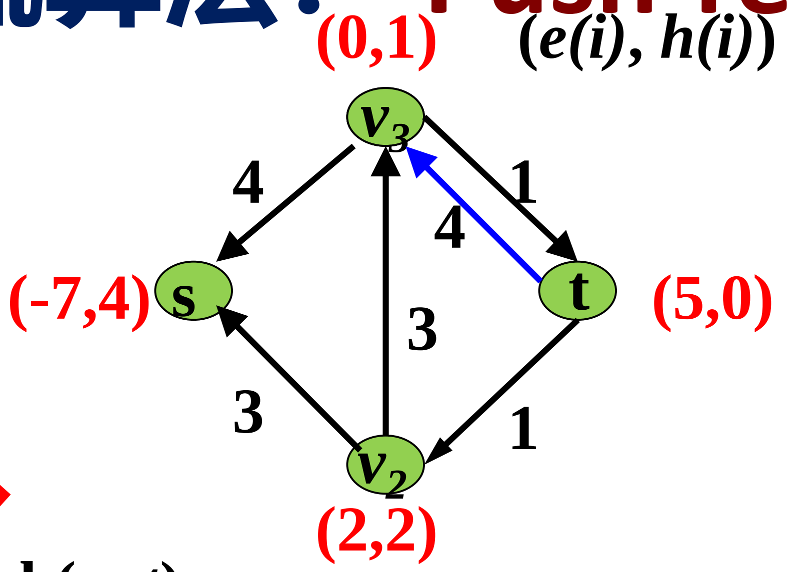
当不存在溢出结点时，算法结束！

得到一个可行流,并且还是最大流！

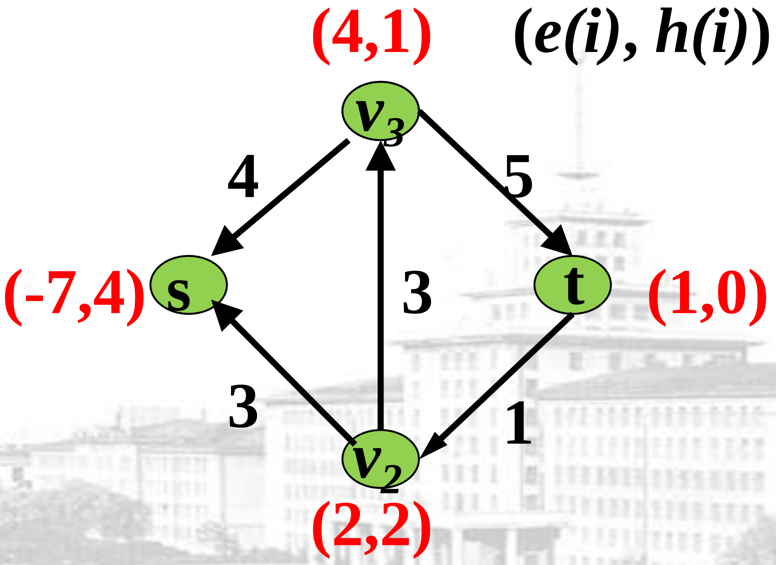
1.网络流算法： Push-relable算法



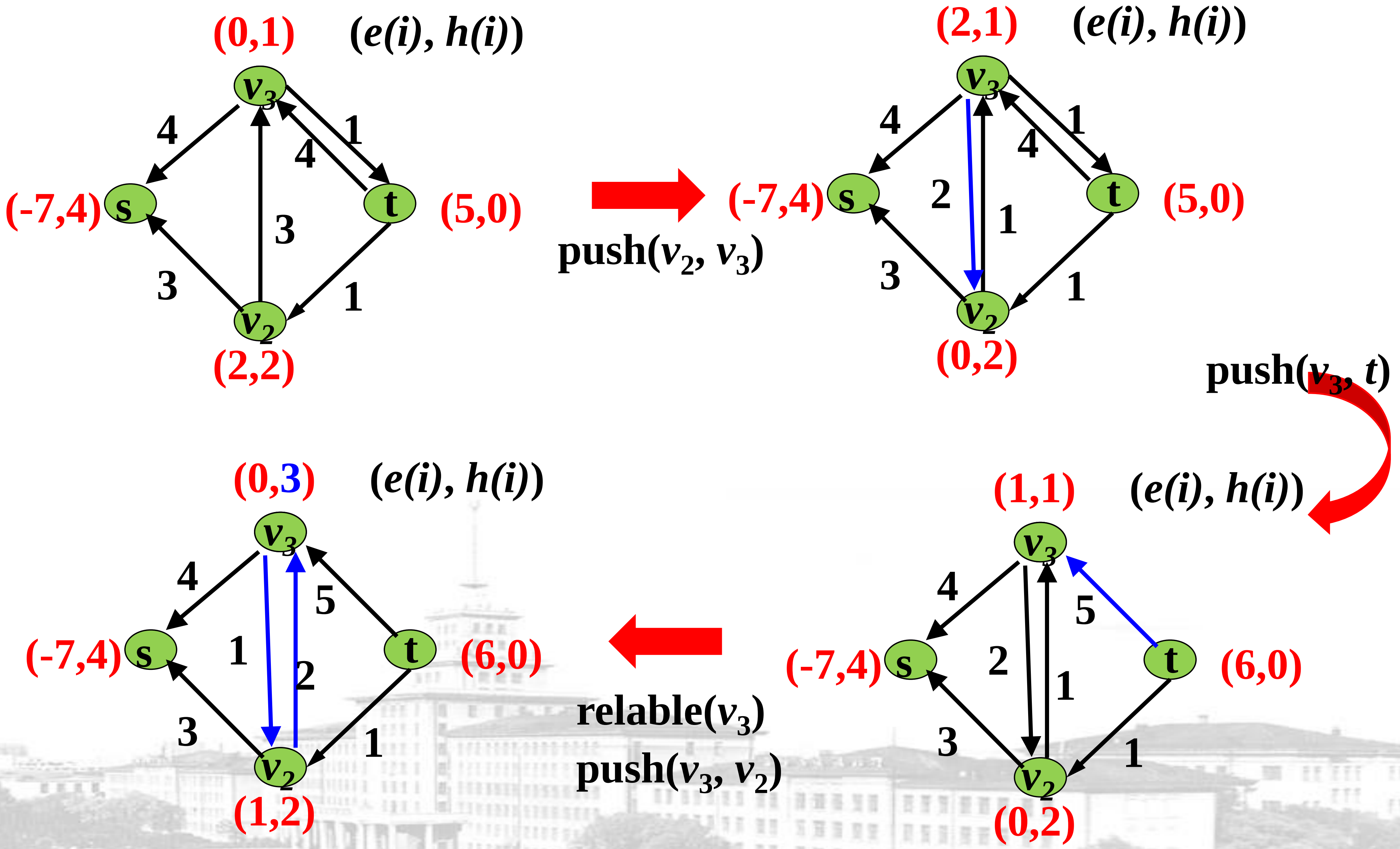
1.网络流算法： Push-relable算法



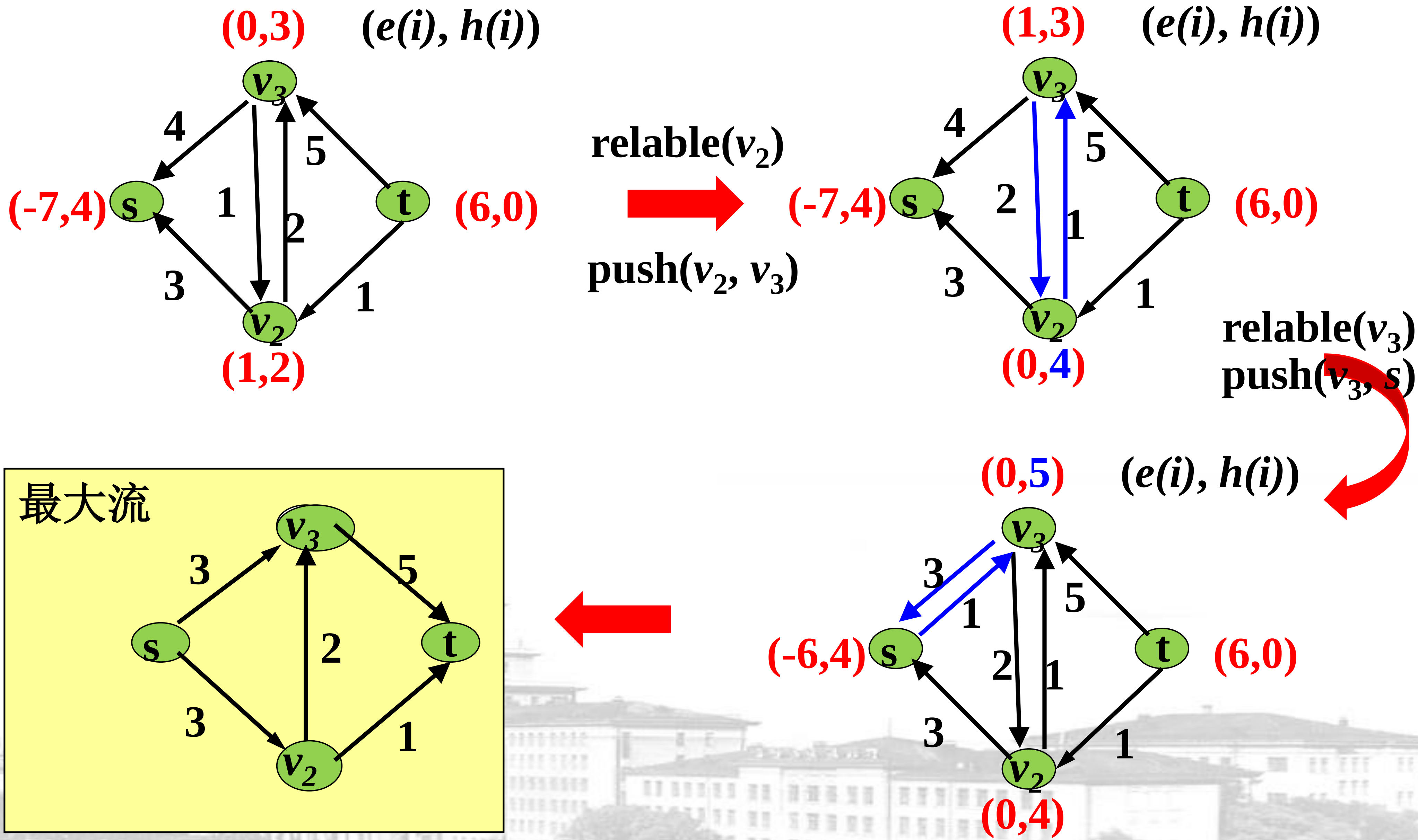
push(v_3,t)



1.网络流算法： Push-relable算法



1.网络流算法： Push-relable算法



本章内容

8.1 网络流算法

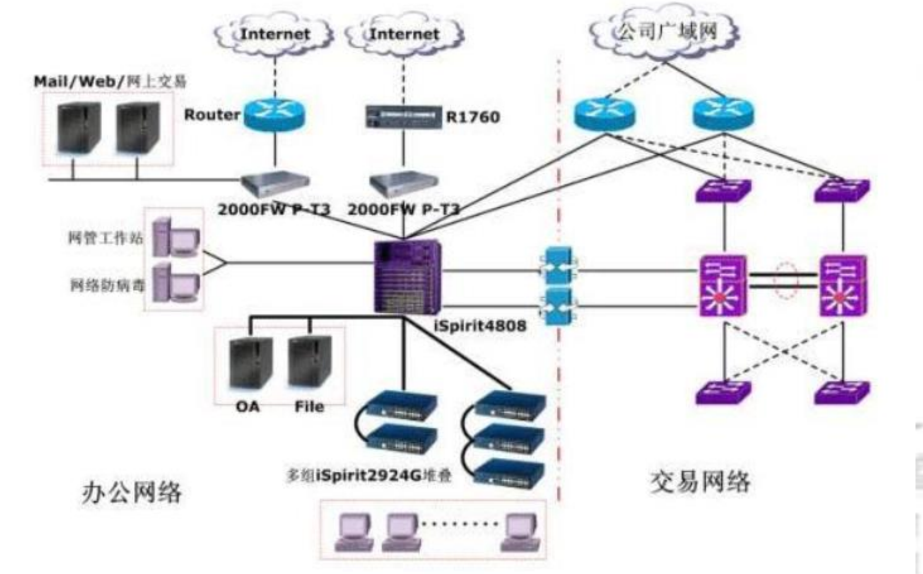
1.概念及定义

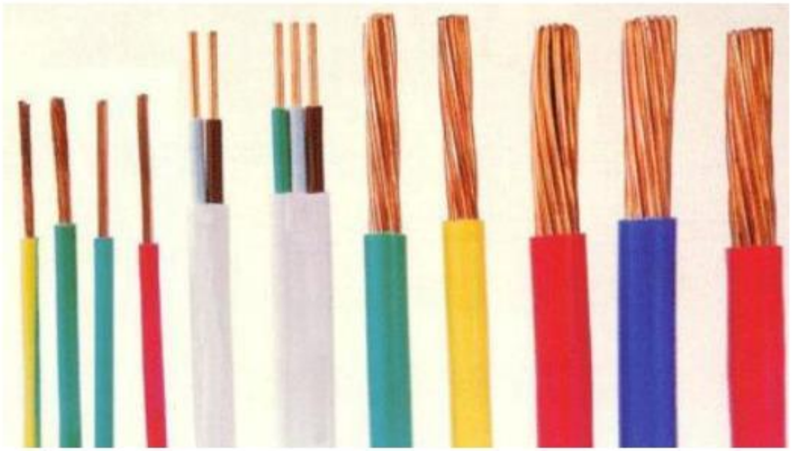
2.最大流算法

3.最小费用最大流算法

3.最小费用最大流算法：背景

- 在实际应用中，不仅要考虑流量，还要考虑费用
- 在网络布线工程中有很多中电缆，电缆的粗细不同，流量和费用也不同。如果全部使用较粗的电缆，则造价太高；如果全部使用较细的电缆，则流量满足不了要求。
- 我们希望建立一个费用最小、流量最大的网络，即最小费用最大流

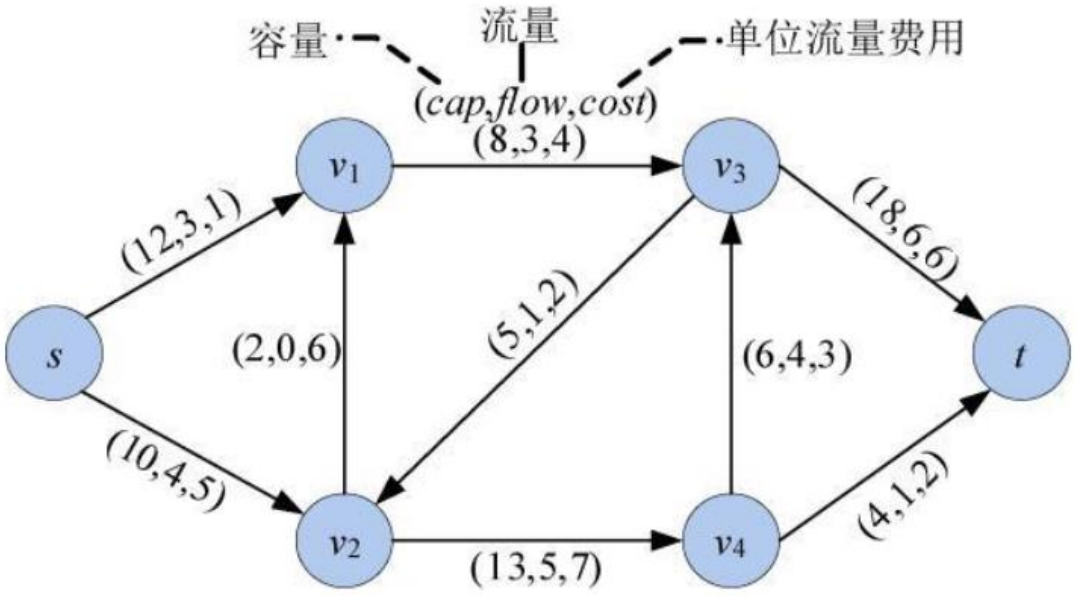




3.最小费用最大流算法：概念和定义

- 实际应用中要同时考虑流量和费用
- 每条边除了给定容量之外，还定义了一个单位流量的费用
- 对于网络上的一个流*flow*，其费用为： 网络流的费用 = 每条边的流量*单位流量费用
- 流的费用 =3×1 + 4×5 + 3×4 + 0×6 + 1×2 + 5×7 + 4×3 + 6×6 + 1×2 = 122
- 我们希望费用最小，流量最大
- 因此需要求解最小费用最大流！

$$cost(flow) = \sum_{(x,y)\in E} cost(x,y) \cdot flow(x,y)$$



3.最小费用最大流算法：概念和定义

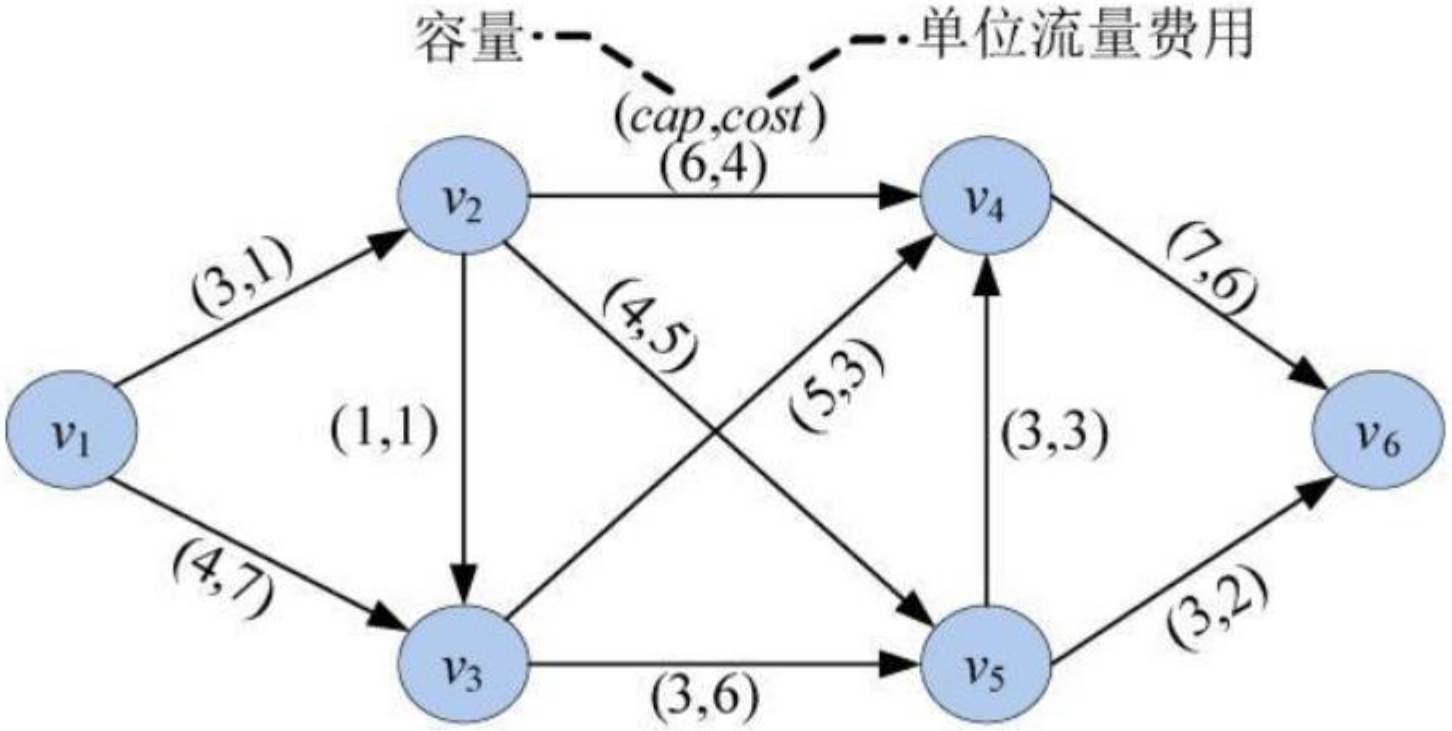
- 求解最小费用最大流有两种思路：
  - 最小费用路算法：先找最小费用路，在该路径上增流，增加到最大流
  - 消圈算法：先找最大流，然后找负费用圈，消减费用，减少到最小费用

  最小费用路算法，是在残余网络上寻找从源点到汇点的最小费用路，即从源点到汇点的以单位费用为权的最短路，然后沿着最小费用路增流，直到找不到最小费用路为止。

  最短增广路算法中求最短增广路是去权值的最短路，而最小费用路是以单位费用为权值的最短路。

3.最小费用最大流算法：实例

- 现给定一个网络及其边上的容量和单位流量费用，求该网络的最小费用最大流

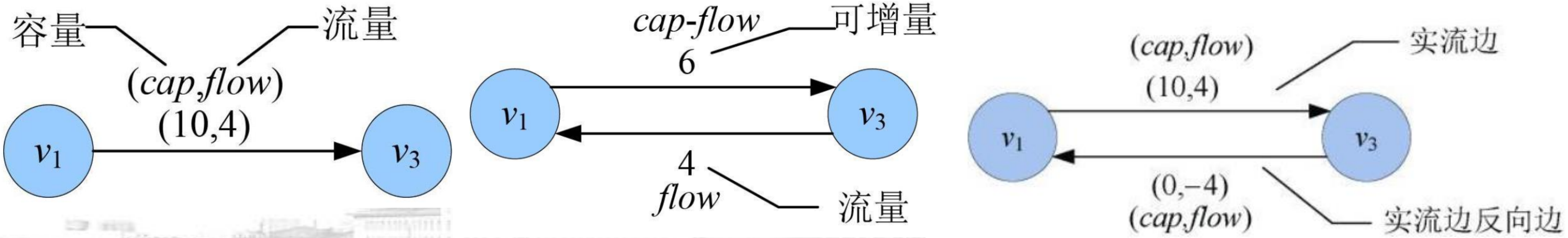


3.最小费用最大流算法：实例

- 创建混合网络:
  - 把残余网络 + 实流网络结合为一体
  - 从每条边的流量可以看出哪些边是实流边（$flow>0$），哪些边是实流边的反向边（$flow<0$）
  - 特殊之处:它的正向边不是显示的可增量$cap-flow$，而是作为两个变量cap、$flow$，增流时cap不变，$flow+=d$；它的反向边不是显示的实际流$flow$，也用两个变量cap，$flow$，不过$cap=0$，$flow=-flow$；增流时cap不变，$flow-=d$

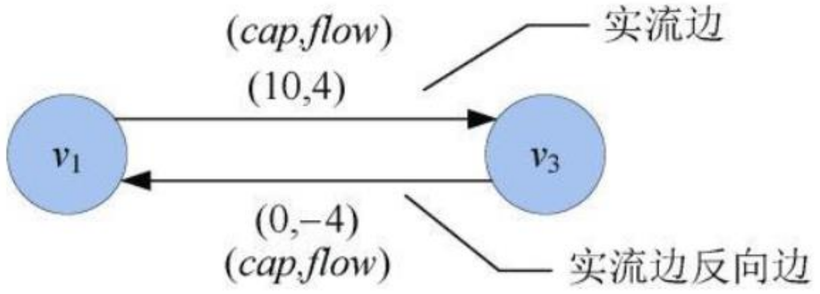
3.最小费用最大流算法：实例

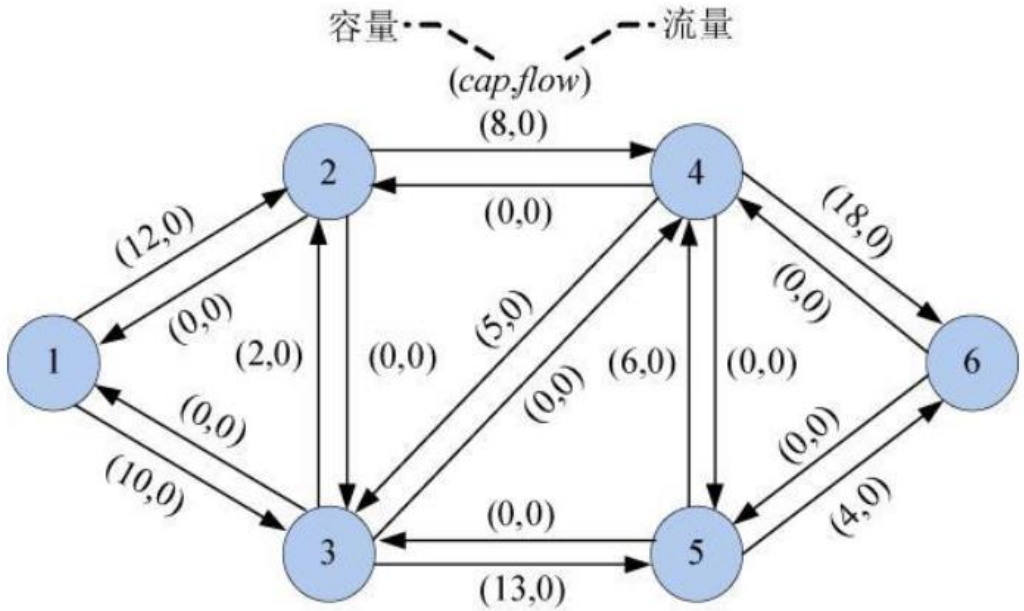
- 创建混合网络:
- 它的正向边不是显示的可增量*cap−flow*，而是作为两个变量*cap*、*flow*，增流时*cap*不变，*flow*+=*d*
- 它的反向边不是显示的实际流*flow*，也用两个变量*cap*，*flow*，不过*cap*=0，*flow*=−*flow*；增流时*cap*不变，*flow*−=*d*



3.最小费用最大流算法：实例

- 创建混合网络：
- 它的正向边不是显示的可增量*cap−flow*，而是作为两个变量*cap*、*flow*，增流时*cap*不变，*flow*+=*d*
- 它的反向边不是显示的实际流*flow*，也用两个变量*cap*，*flow*，不过*cap*=0，*flow*=−*flow*；增流时*cap*不变，*flow*−=*d*

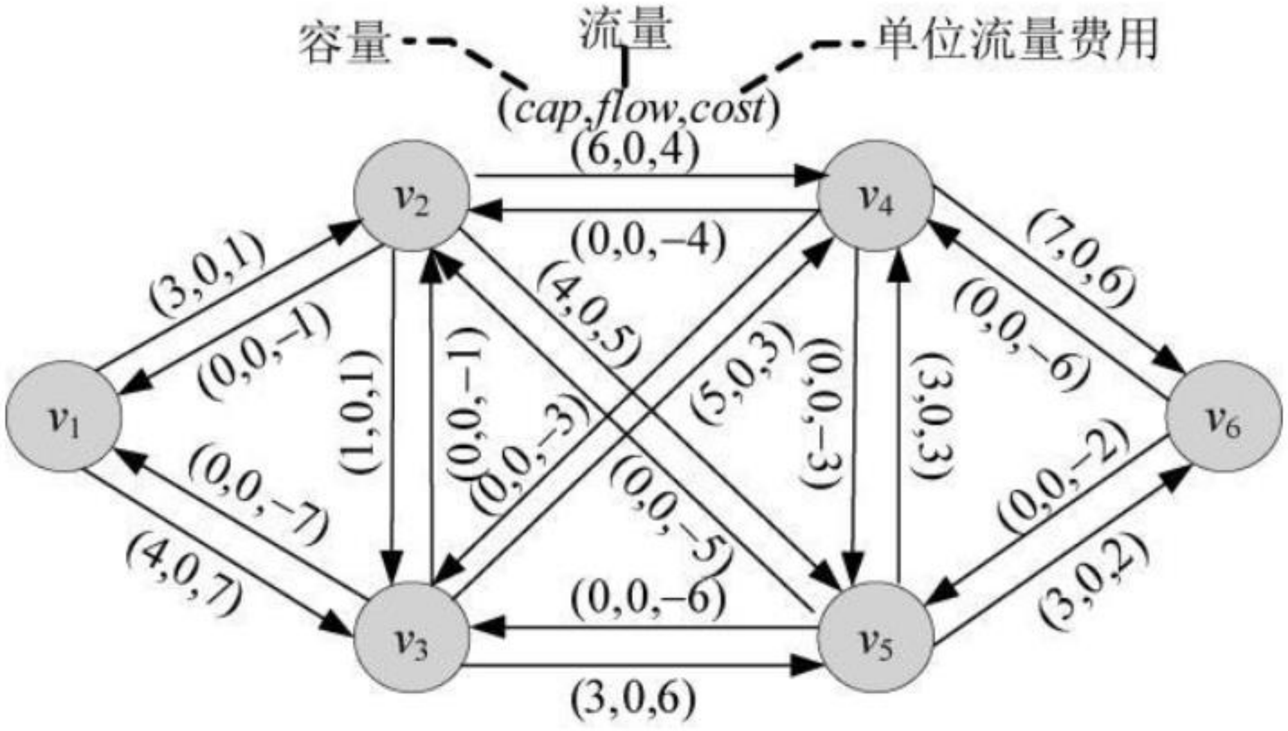




3.最小费用最大流算法：实例

- 创建混合网络：

  先初始化为零流，零流对应的混合网络中，正向边的容量为 cap，流量为 0，费用为$cost$，反向边容量为 0，流量为 0，费用为 $-cost$

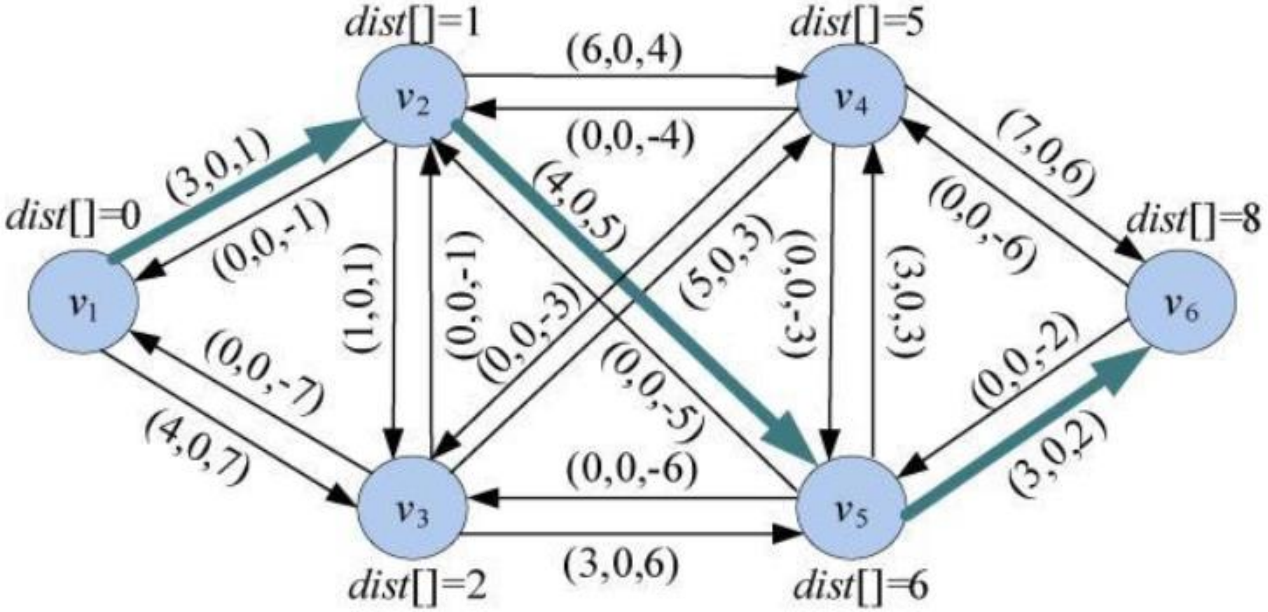


3.最小费用最大流算法：实例

- 找最小费用路:

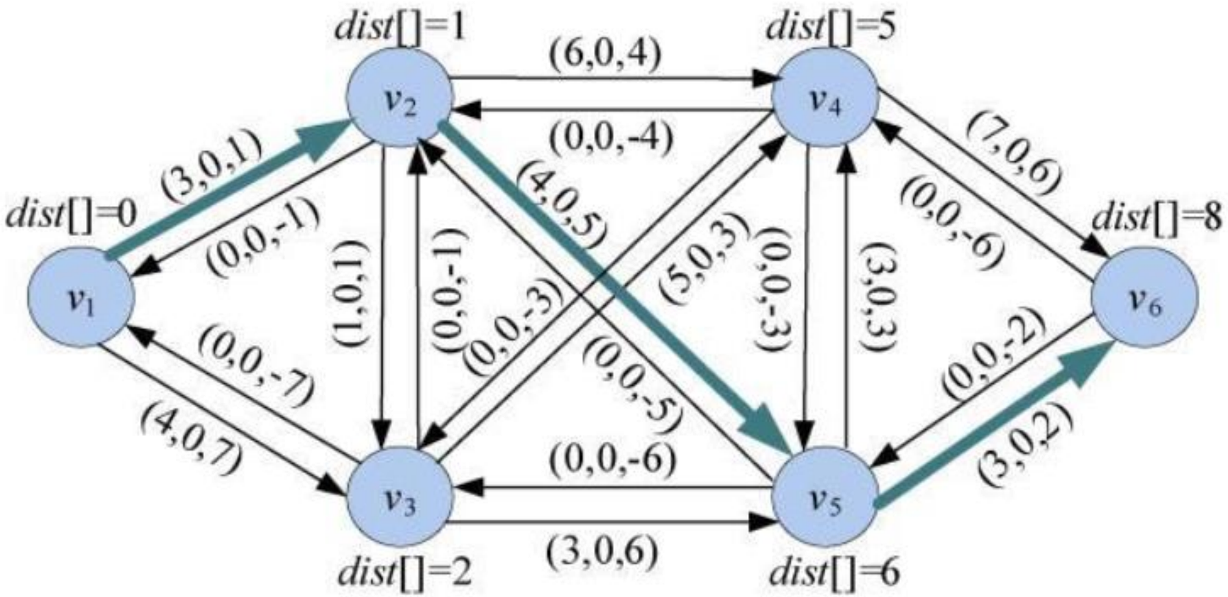
  先初始化每个结点的距离为无穷大，令源距离*dist[v1]=0*。在混合网络中，从源点出发沿可行边*E[i].cap>E[i].flow*广度搜索每个邻接点，如果当前距离*dist[v]>dist[u]+E[i].cost*，则更新为最短距离：*dist[v]=dist[u]+E[i].cost*，并记录前驱

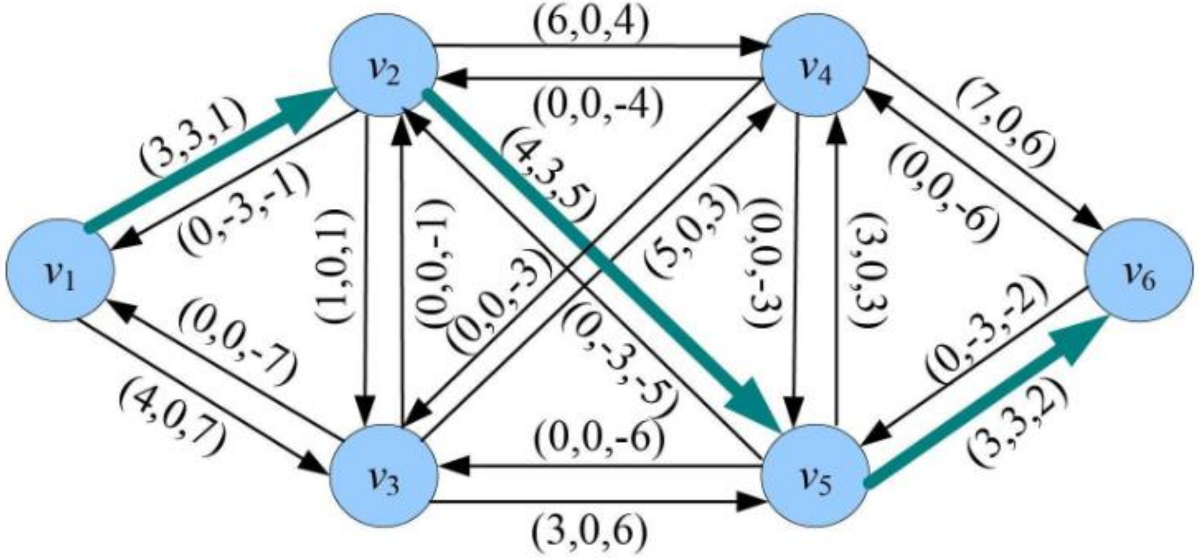
  根据前驱数组，找到一条最短费用路，增广路径：*1—2—5—6*



3.最小费用最大流算法：实例

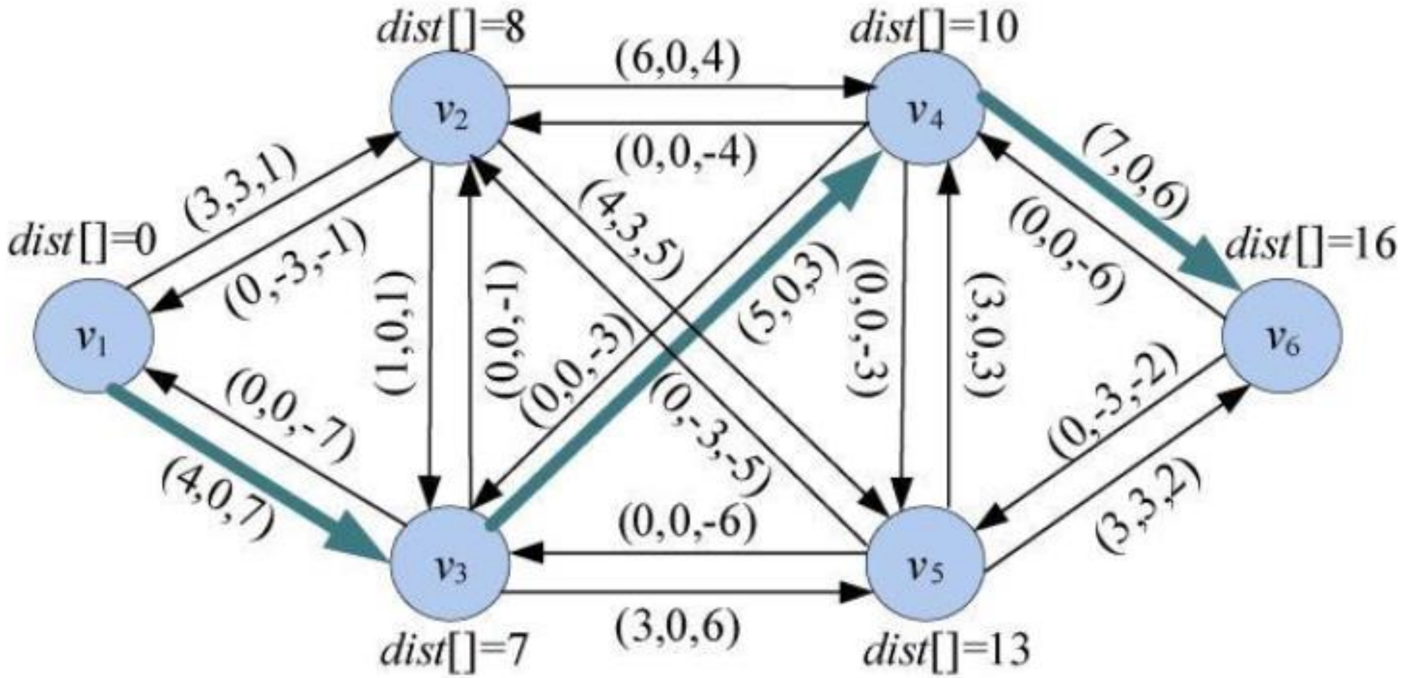
- 沿着增广路径正向增流*d*，反向减流*d*
- 从汇点逆向找最小可增流量$d=\min(d,\ E[i].cap-E[i].flow)$，增流量$d=3$，产生的费用为$mincost+=dist[v6]\times d=(1+5+2)\times 3=8\times 3=24$





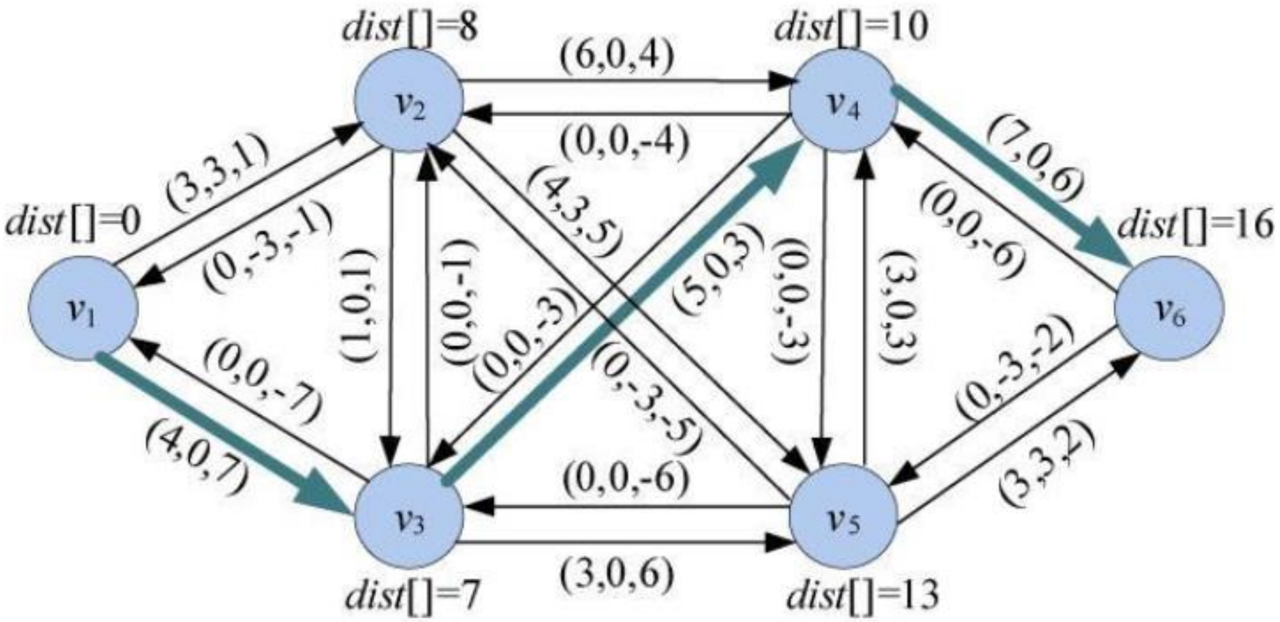
3.最小费用最大流算法：实例

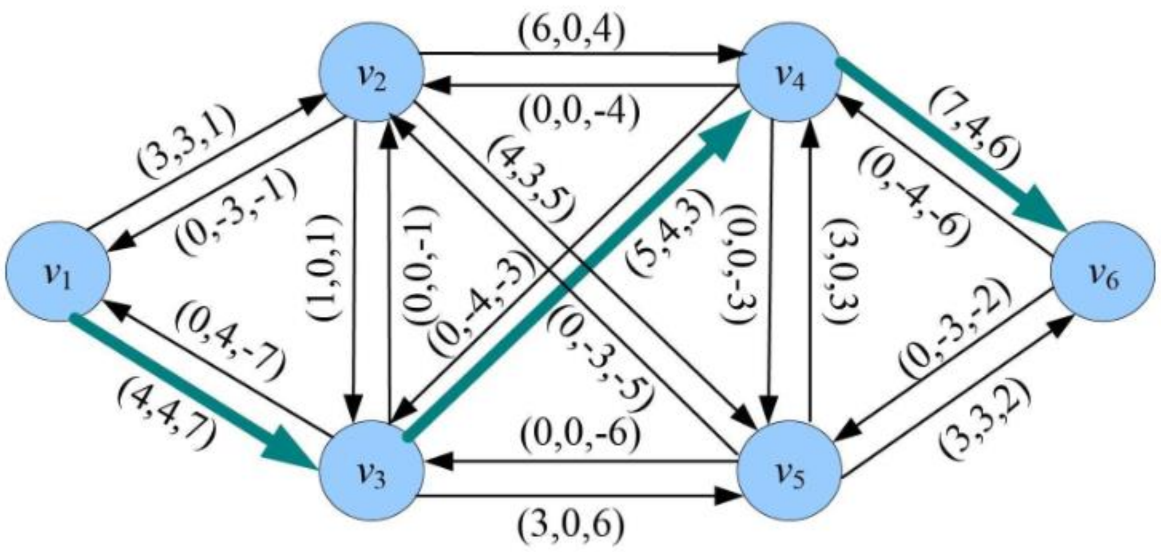
- 找最小费用路:
- 从源点出发，沿可行边*E[i].cap>E[i].flow*广度搜索每个邻接点，如果当前距离*dist[v]>dist[u]+ E[i].cost*，则更新为最短距离：*dist[v]=dist[u]+E[i].cost*，并记录前驱
- 根据前驱数组，找到一条最短费用路，增广路径：*1—3—4—6*



3.最小费用最大流算法：实例

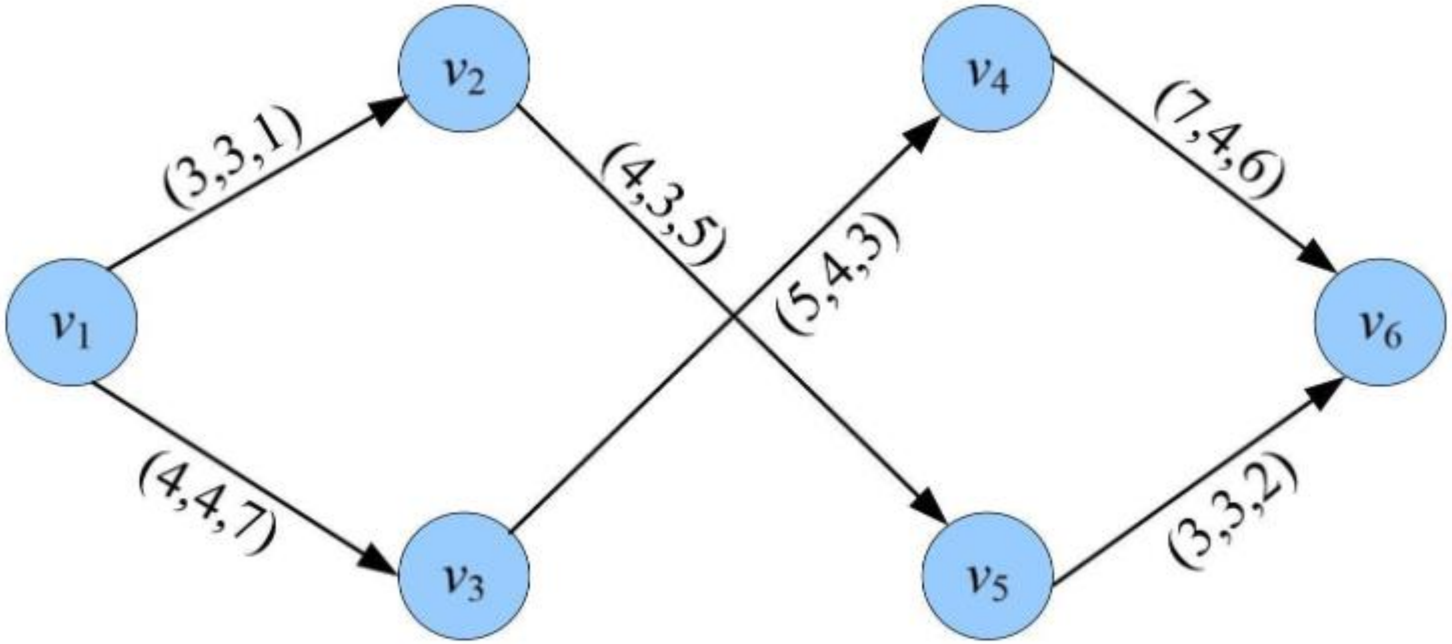
- 沿着增广路径正向增流*d*，反向减流*d*
- 从汇点逆向找最小可增流量*d=min(d, E[i].cap−E[i].flow)*，增流量*d=4*，产生的费用为*mincost=24+dist[v6]*d=24+16*×*4=88*





3.最小费用最大流算法：实例

- 初始化每个结点的距离为无穷大，然后令源点的距离*dist[v1]=0*。在混合网络中，从源点出发，沿可行边*E[i].cap>E[i].flow*广度搜索每个邻接点，发现从源点出发已没有可行边，结束
- 得到的网络流就是最小费用最大流。把混合网络中*flow>0* 的边输出，就是实流网络



3.最小费用最大流算法:算法优化

- 消圈算法的思想：首先找网络中的最大流，然后消除最大流对应的混合网络中所有的负费用圈。

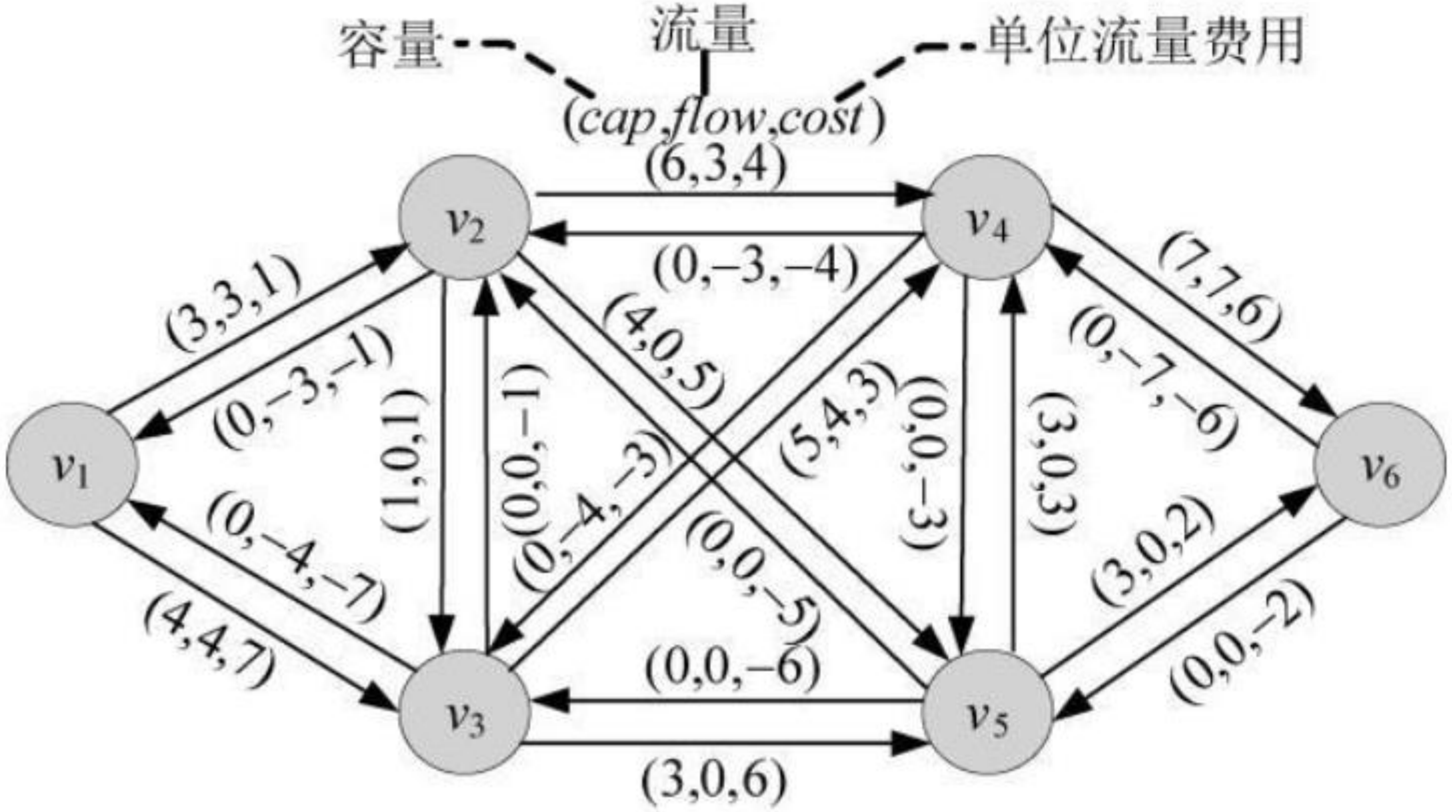
（1）找给定网络的最大流

（2）在最大流对应的混合网络中找负费用圈

（3）消负费用圈：负费用圈同方向的边流量加d，反方向的边流量减d。d为负费用圈的所有边的最小可增量$cap-flow$

3.最小费用最大流算法:算法优化

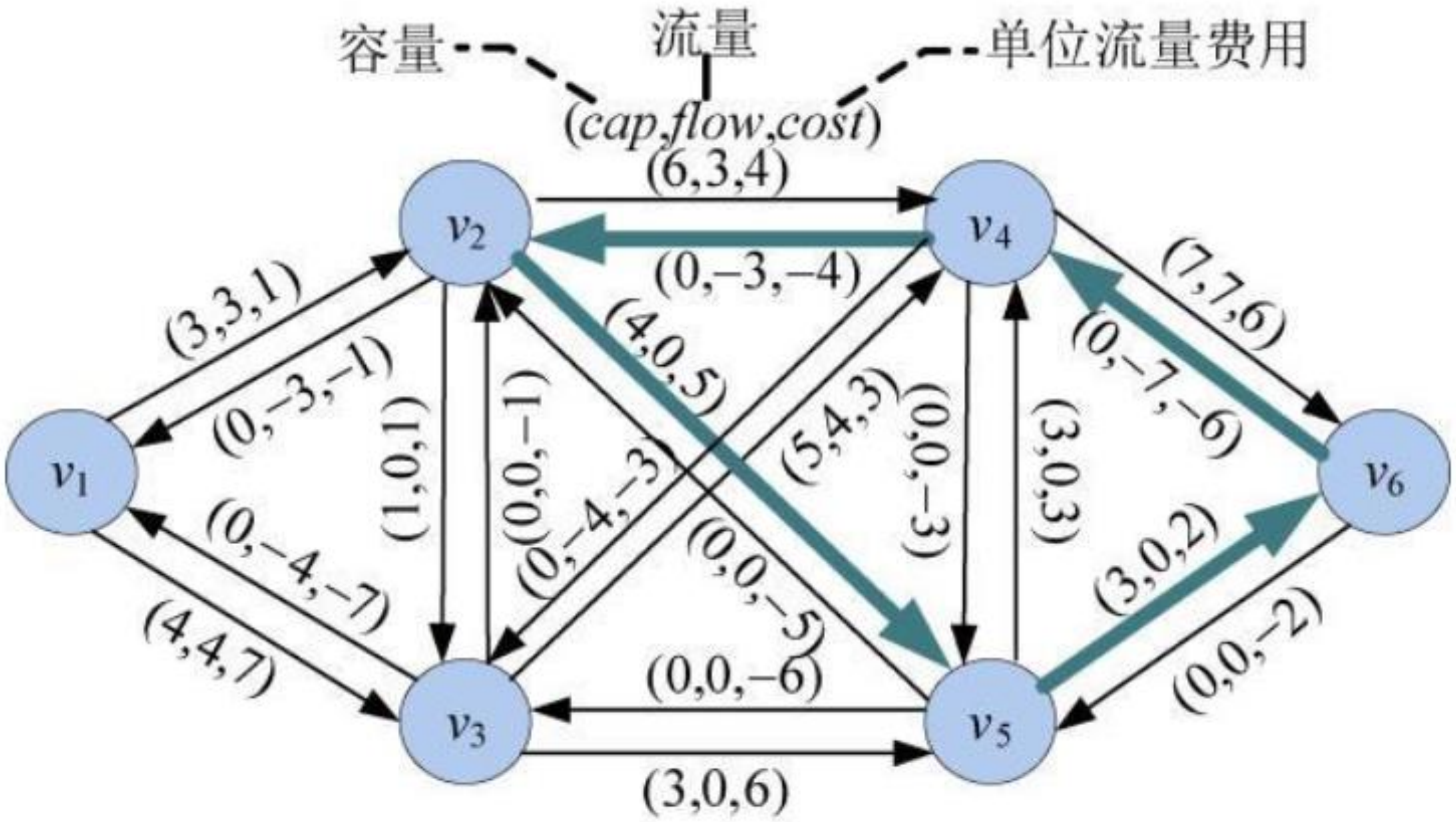
- 运行后得到最大流对应的混合网络



3.最小费用最大流算法:算法优化

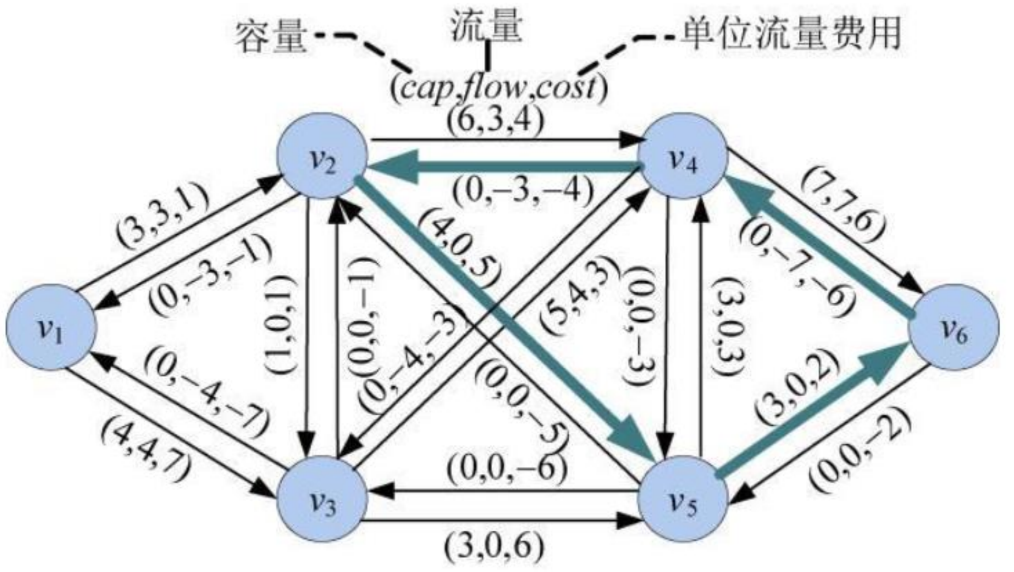
- 在最大流对应的混合网络中找负费用圈 在最大流的混合网络中，沿着*cap>flow*的边找负费用圈，就是各边费用之和为负的圈。首先找到一个负费用圈 ***2—5—6—4—2***，它们的边费用之和为 ***5+2+(−6)+(−4)=−3***

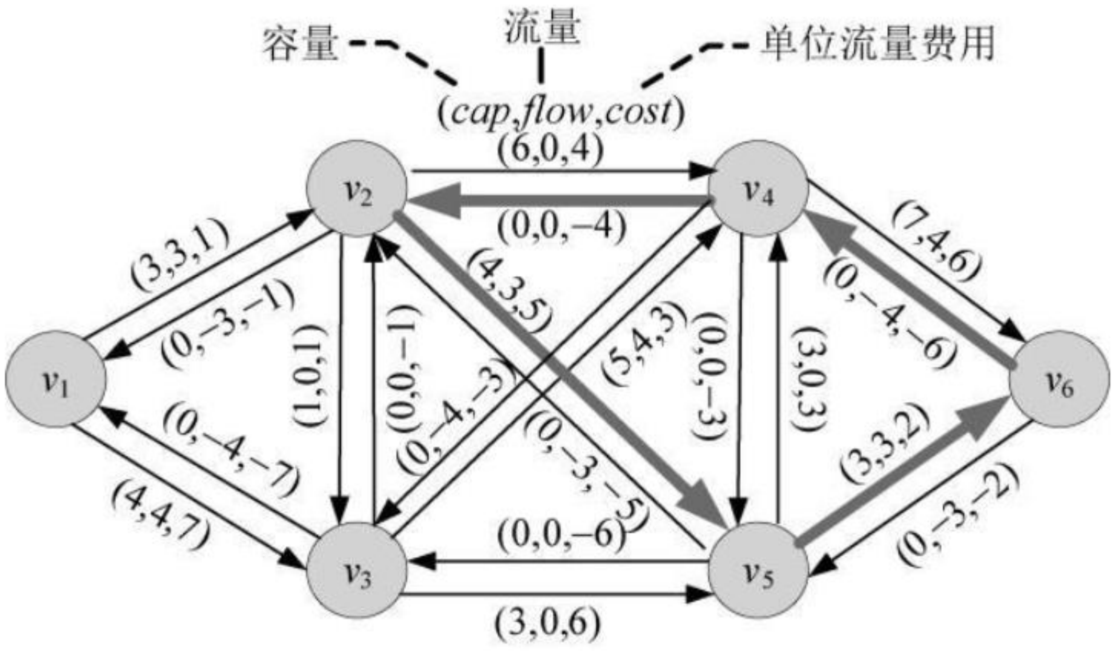
负费用圈同方向的边流量加***d***，反方向的边流量减***d***。



3.最小费用最大流算法:算法优化

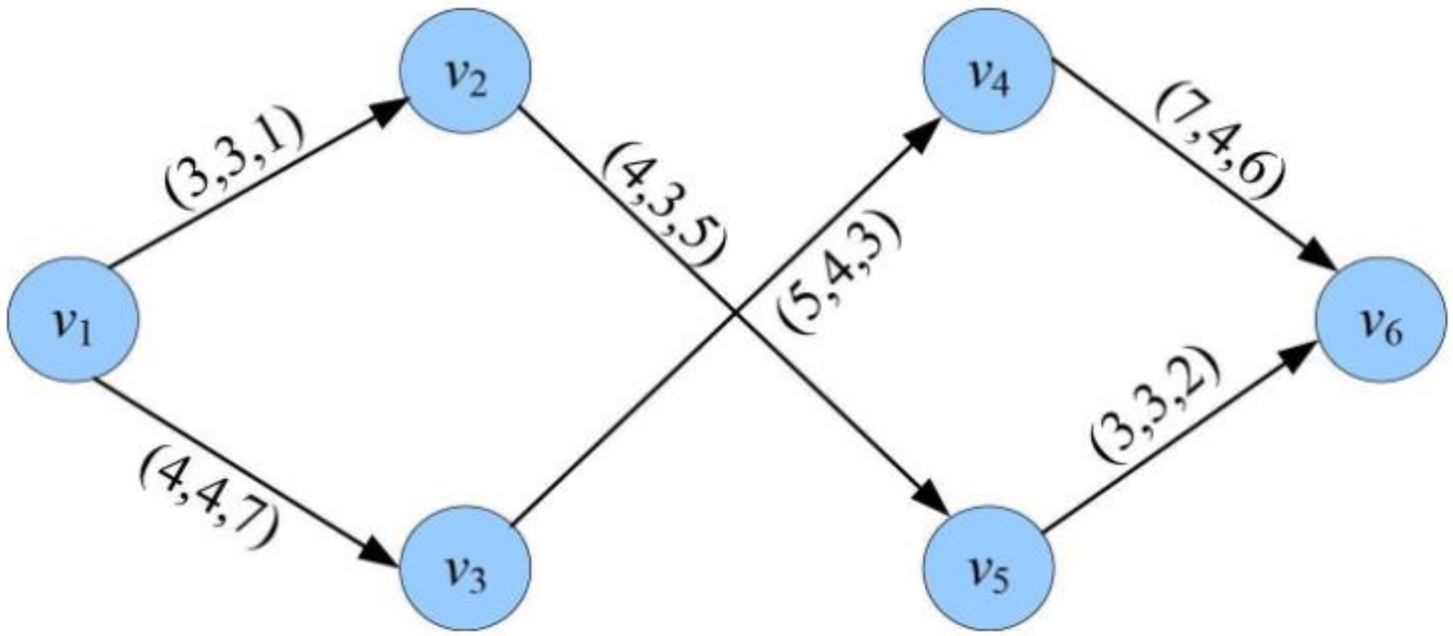
- 负费用圈同方向的边流量加***d***，反方向的边流量减***d***
- 沿找到的负费用圈增流，其增量为组成负费用圈的所有边的最小可增量*cap−flow*
- 负费用圈说明费用较高，可以对费用为负的边减流（因为该残余网络为特殊的残余网络，负费用的边流量也是负值，减流实际上需要加上增流量*d*）





3.最小费用最大流算法:算法优化

- 在混合网络中继续找负费用圈 在混合网络中，沿着*cap>flow*的边找负费用圈，已经找不到负费用圈，算法结束。把混合网络中*flow>0* 的边输出，就是我们要的实流网络



本章内容

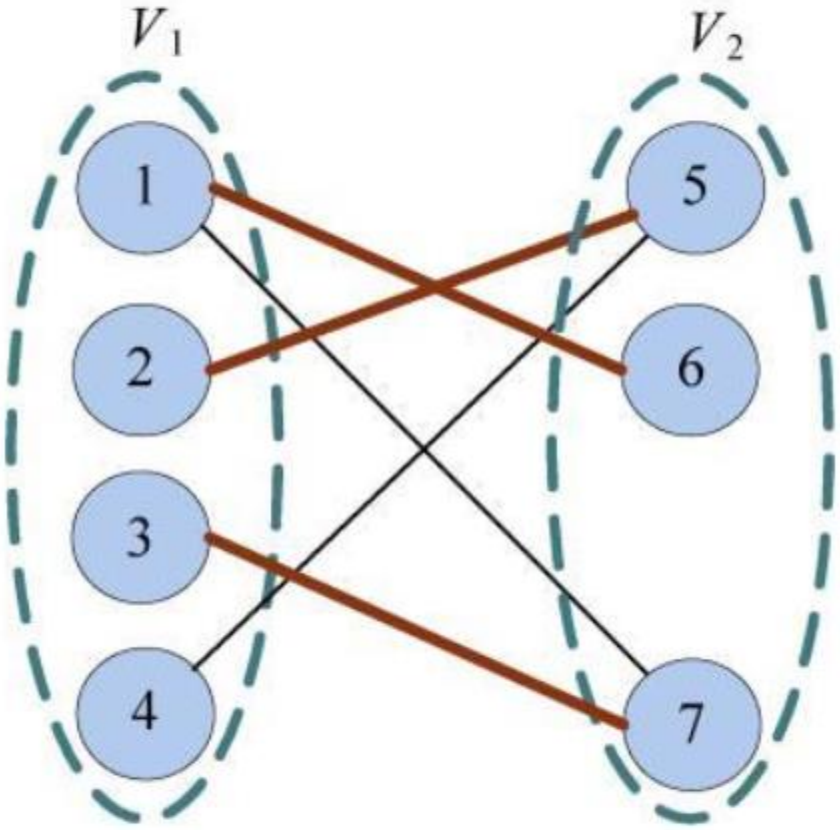
8.2 图匹配算法

1.匹配与覆盖

2.最大二分匹配算法

2. 图匹配：概念和定义

- 匹配(Matching)
  - 给定图*G*(*V*,*E*)，一个匹配*M*是由*G*的一组没有公共端点的不是圈的边构成的集合。
    - 对于所有的结点*v*∈*V*，子集*M*中最多有一条边与结点*v*相连
    - 与匹配*M*中边关联的那些结点是被*M*-浸润的，其余结点是*M*-未浸润的



2. 图匹配：概念和定义

- 极大匹配：不能再通过添加边使其变大的匹配
  - 即：不存在 $e \in E$ 满足 $M \cup \{e\}$ 也是匹配
- 最大匹配：边数最多的匹配：$|M|$最大的匹配
- 完美匹配：浸润了所有结点的匹配
  - 即 $|M|=n/2$
  - 最大匹配一定是极大匹配，而极大匹配不一定是最大匹配。在一个无向图中，可以有多个极大匹配和最大匹配

性质：完美匹配是最大匹配，反之不然

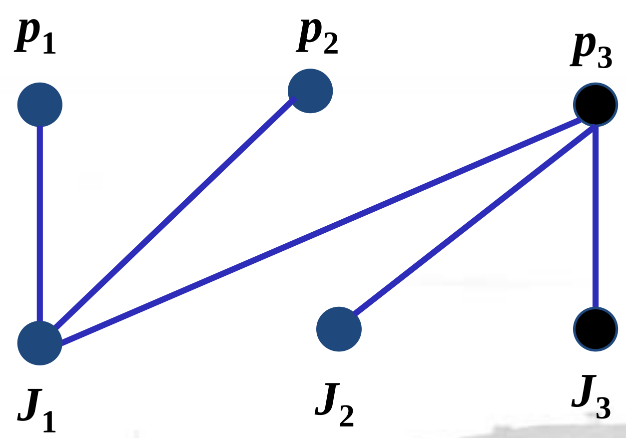
2. 图匹配：概念和定义

- 最大匹配问题
  - 输入：图$G(V,E)$
  - 输出：G的最大匹配M
- 应用
  - 人员指派
  - 教室指派
  - 任务安排
  - 赛程安排

工作分配

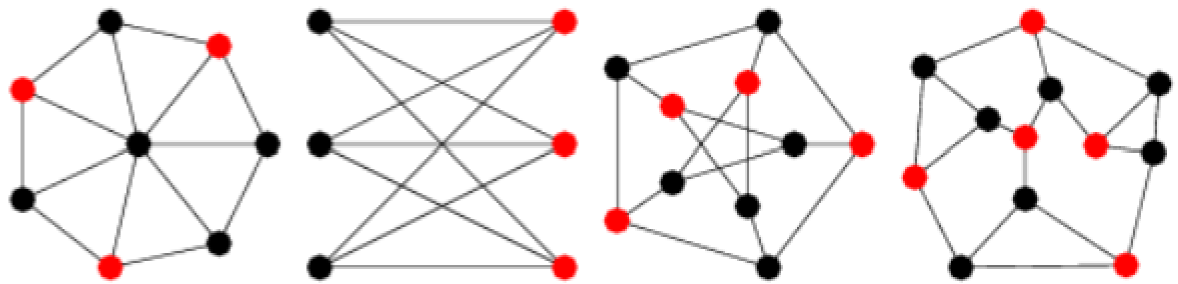
输入：n个人$p_1,\ldots,p_n$，n项工作$J_1,\ldots,J_n$，第i个人胜任其中k项工作

输出：是否存在工作分配方案使得每个人完成1项自己胜任的工作



2. 图匹配：概念和定义

- 顶点覆盖
  - 图$G(V,E)$的一个顶点覆盖是指顶点集合$C \in V$，C包含每条边上的至少一个端点
    - C的所有顶点覆盖边集E
- 最小顶点覆盖
  - $|C|$最小顶点的覆盖
- 最小顶点覆盖问题
  - 输入：图$G(V,E)$
  - 输出：G的最小顶点的覆盖



黑色的点集合都是顶点覆盖集合，图的每一条边都至少有一个顶点在点集合中

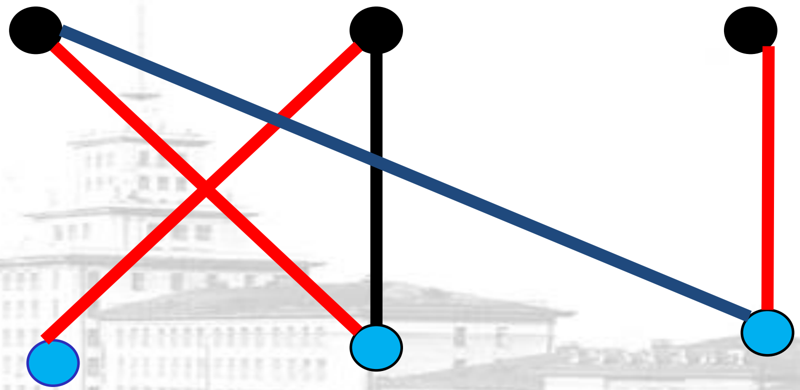
目前，大规模图数据管理已经非常盛行
很多高效算法都以匹配算法或覆盖算法为基础！

2. 图匹配：概念和定义

- 加权最大/小匹配问题
  - 每条边有一个代价，寻找具有最大/小代价的匹配
- 加权最小覆盖问题
  - 每个顶点有一个代价，寻找具有最小代价的覆盖

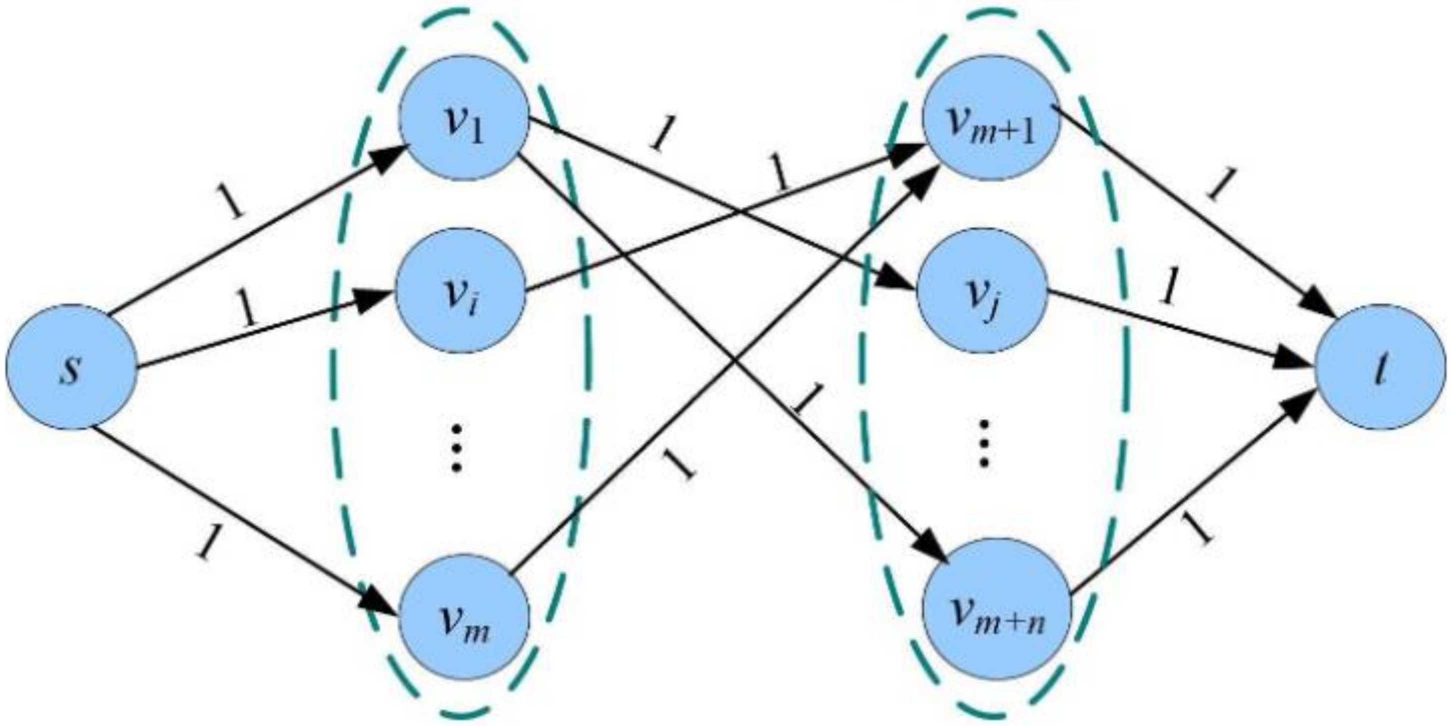
2. 图匹配：最大二分匹配问题

- 二分图(Bipartite Graph，又称二部图)
  - 图$G(V,E)$称为二分图，如果V=$L\cup R$，$L\cap R=\Phi$，E中所有边一定是有一个顶点属于集合L，另一个顶点属于集合R
- 二分图最大匹配问题
  - 输入：二分图$G(V,E)$
  - 输出：G的最大匹配



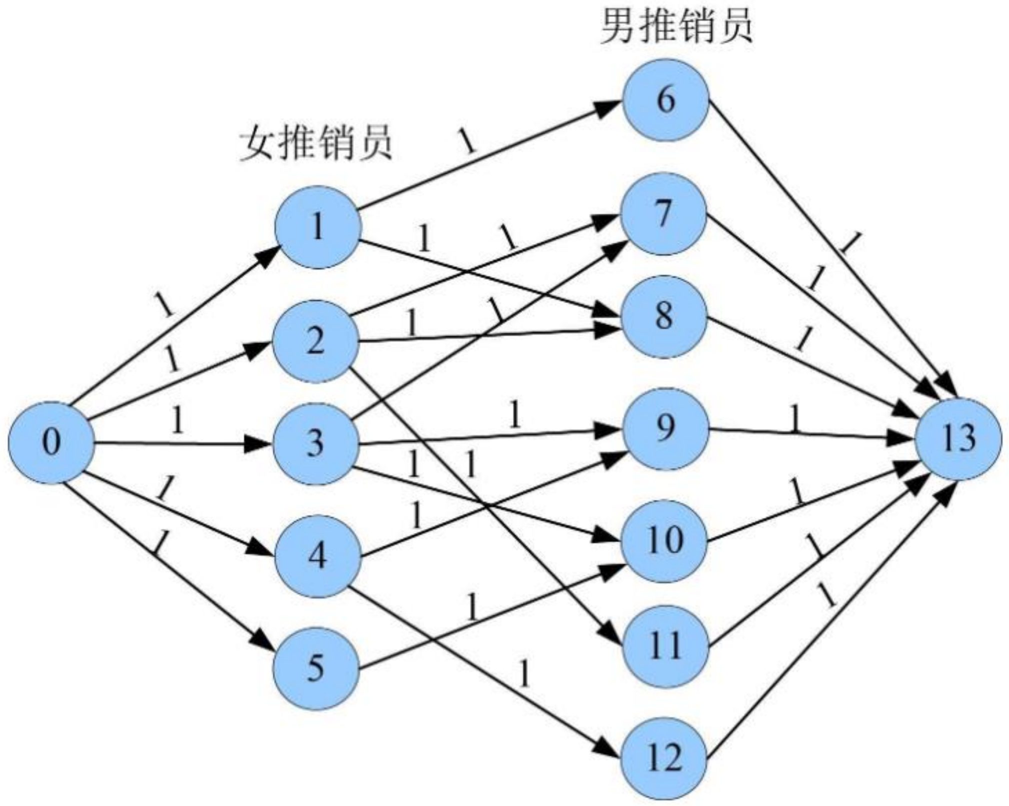
2. 图匹配：最大二分匹配问题

- 如何得到二分图的最大匹配？
  - 借助**最大流算法**
  - 将二分图左边添加一个源点，右边添加一个汇点，将左边的点全部与源点相连，右边的点和汇点相连，所有边的容量均为 1



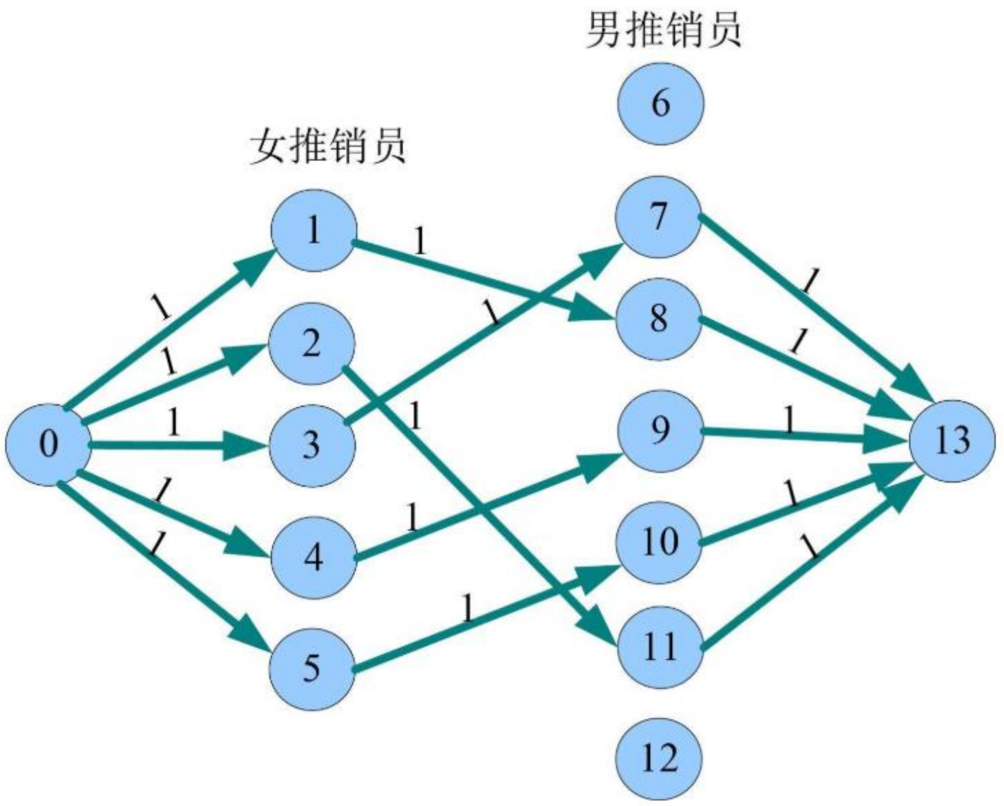
2. 图匹配：最大二分匹配问题

- 女推销员数为 5，编号 1～5；男推销员数为 7，编号 6～12。以下两个编号的推销员可以配合：1—6，1—8，2—7，2—8，2—11，3—7，3—9，3—10，4—12，4—9，5—10



2. 图匹配：最大二分匹配问题

- 求网络最大流，输出最大流值就是最多的配对数
- 最大配对数：5
- 配对方案：1—8，2—11，3—7，4—9，5—10



2. 图匹配：求解最大二分匹配问题

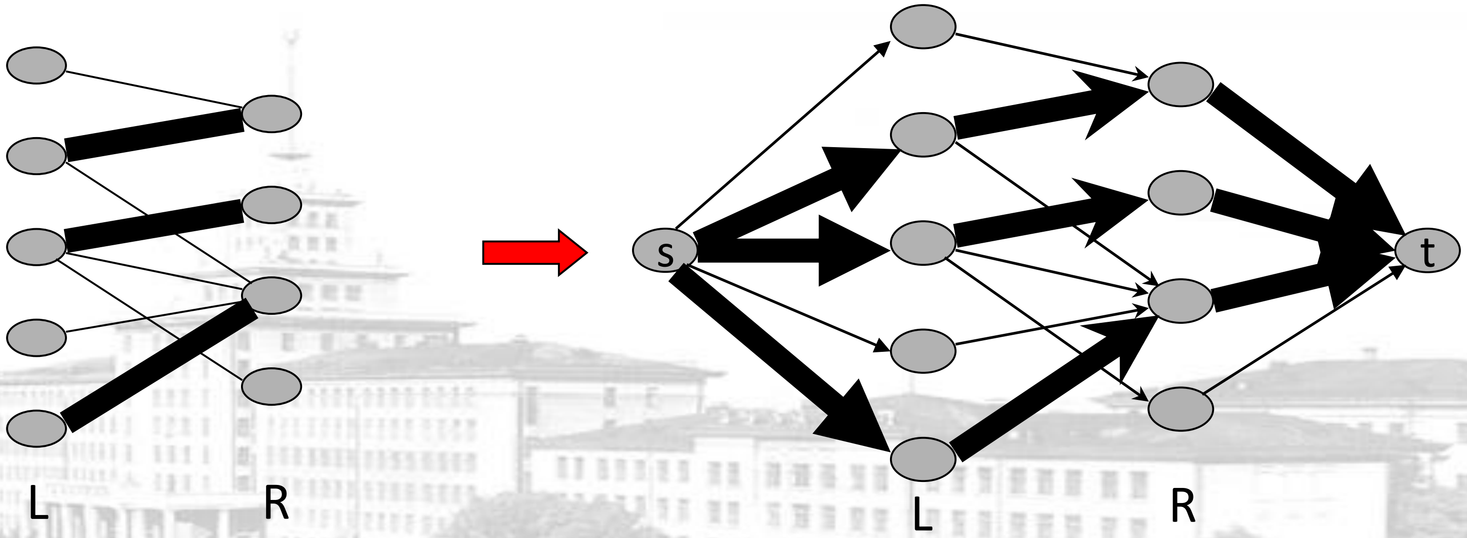
- 规约为最大流问题，利用最大流求解算法
- 二部图的对应流网络

给定二部图$G=(V, E)$, $V=L\cup R$, G对应流网络为$G'=(V', E')$

$V'=V\cup\{s, t\}$,

$E'=\{(s, u) \mid u\in L\}\cup\{(u, v) \mid u\in L, v\in R\}\cup\{(v, t) \mid v\in R\}$

每条边的容量为一个单位



2. 图匹配：求解最大二分匹配问题

引理1. 设$G'=(V', E')$是$G=(V, E)$的对应流网络, 则$|E'|=\Theta(|E|)$.

证明.

由于V中每个节点v至少有一条连接v的边, $|E|\geq|V|/2$.

于是, $|E|\leq|E'|=|E|+|V|\leq3|E|$, 即$|E'|=\Theta(|E|)$.

2. 图匹配：求解最大二分匹配问题

- 算法
  - 1. 由*G*构造对应的*G'*;
  - 2. FORD-FULKERSON(*G'*, *s*, *t*); //或任意其它最大流算法
  - 3. 由*G'*的最大流构造*G*的最大匹配
  - $M=\{\ (u, v) \mid u \in L, v \in R, f(u, v)>0\}$.
- 算法复杂性
  - 第1步: $\Theta(|E|)$;
  - 第2步: $O(|V||E|)$; //或其它
  - 第3步: $O(|E|)$;
  - 总时间: $O(|V||E|)$.

2. 图匹配：求解最大二分匹配问题

– 算法正确性证明

引理2. 设$G=(V, E)$是一个二部图, $V=L\cup R$, G'是G对应的流网络. M是G的一个匹配 iff G'中存在一个整数值流f, $|f|=|M|$.

证明. $\Rightarrow$ 设M是G的匹配. 如下定义G'中的流f：若$(u, v)\in M$, $f(s, u)=f(u, v)=f(v, t)=1$, $f(u, s)=f(v, u)=f(t, v)=-1$. 对于其他$(u, v)\in E'$, $f(u, v)=0$.

容易证明: f满足容量约束性、斜对称性、流守恒性.

往证$|f|=|M|$.

对于$\forall(u, v)\in M$, (u, v)对应一个1单位流$s\to u\to v\to t$. 由M的边导致的路径除s和t外节点不相交.

于是, 跨越划分$(L\cup\{s\}, R\cup\{t\})$的net flow等于$|M|$. 由前节的引理4, $|f|=|M|$.

2. 图匹配：求解最大二分匹配问题

– 算法正确性证明

引理2. 设$G=(V, E)$是一个二部图, $V=L\cup R$, G'是G对应的流网络. M是G的一个匹配 iff G'中存在一个整数值流f, $|f|=|M|$.

证明. $\Leftarrow$ 设f是G'的整数值流. 如下定义G中匹配M: $M=\{(u, v) \mid u\in L, v\in R, f(u, v)>0\}$.

往证M是G中匹配.

对于$\forall u\in L$, u仅有一个容量为1的入边(s, u). 于是, 至多有一个单位正值流进入u, 且由流守恒性, 若有单位流进入u, 必有单位流离开u.

由于f的值是整数, 单位流只能经一条边进入或离开u.于是, 单位流进入u iff 存在一个节点$v\in R$, 使$f(u, v)=1$.

同样的结论对$\forall v\in R$也成立.

于是, M是G的一个匹配.

2. 图匹配：求解最大二分匹配问题

– 算法正确性证明

引理2. 设$G=(V, E)$是一个二部图, $V=L\cup R$, G'是G对应的流网络. M是G的一个匹配 iff G'中存在一个整数值流f, $|f|=|M|$.

证明. $\Leftarrow$ 往证$|f|=|M|$.

对于每个匹配节点$u\in L$, $f(s, u)=1$; 对于$\forall(u, v)\in E-M$, $f(u, v)=0$. $|M|=f(L, R)=f(L, V')-f(L, L)-f(L, s)-f(L, t)$.

由流守恒性, $f(L, V')=0$; 由斜对称性$-f(L, t)=f(s, L)$; 由于无L到t的边, $f(L, t)=0$; $f(L, L)=0$. 于是

$$|M|=f(s, L)=f(s, V')=|f|.$$

2. 图匹配：求解最大二分匹配问题

定理1. 若容量函数c仅取整数值, 则由Ford-Fulkerson方法产生的最大流f具有下列性质:

(1). | f |是整数;

(2). 对于所有节点u和v, f(u, v)是整数.

证明. 对于循环数做数学归纳证明.

推论1. 设M是G中最大匹配, f是G对应的G’的最大流, 则|M|=| f |.

证明. 设M是G的最大匹配, f不是G’的最大流.

必存在G’的最大流f’, | f’|>| f |.

由于G’的容量值是整数, 由定理1, 可以设f’是整数值流. 于是, f’对应于G的一个最大流M’, |M’|=| f’|>| f |=|M|, 与M是最大流矛盾.

类似可证, 若f是G’的最大流, 它对应的匹配M必为G的最大匹配.

2. 图匹配：最大二分匹配问题

- 定理(**Konig-Egervary**)：如果*G*是一个二分图，则*G*中最大匹配的大小等于*G*的最小顶点覆盖的大小
- 这意味着二分图上的最大匹配可以这样求解
  - 初始化一个匹配*M*
  - 不断地增大*M*
  - *M*无法增大时，找出一个顶点覆盖*C*使得|*M*|=|*C*|
  - *M*是最大匹配，*C*是最小覆盖

2. 图匹配：求解最大二分匹配问题

匈牙利算法（1965年匈牙利数学家 Edmonds提出）

1. $U \leftarrow V$中未被M浸润的所有顶点

2. **While** 存在从$x \in U$出发的M增广路径

   则增大M

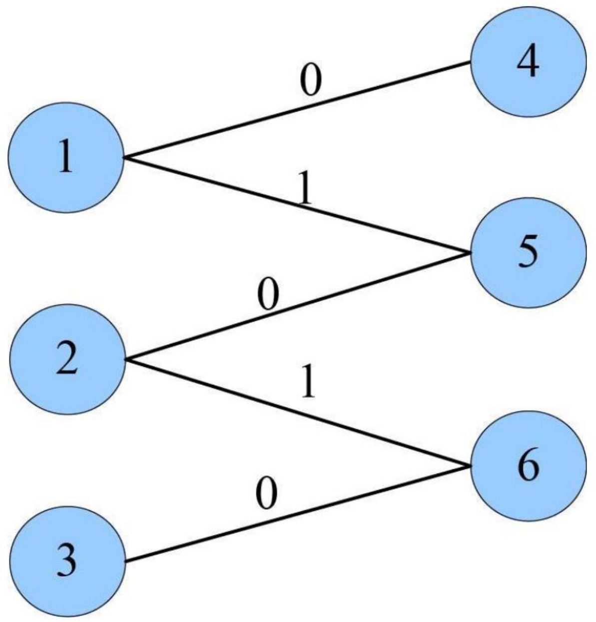
   $U \leftarrow V$中未被M浸润的所有顶点

3. M是最大匹配

复杂度：**O(|*V*||*E*|)**

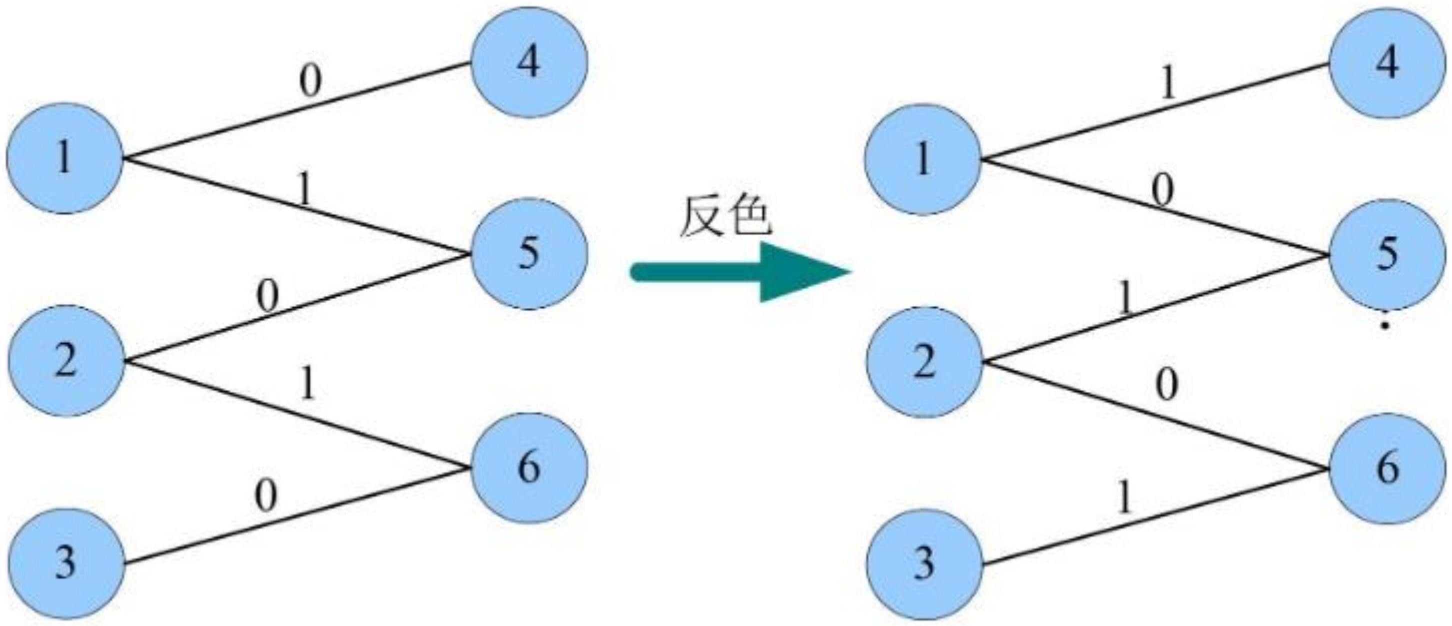
2. 图匹配：求解最大二分匹配问题

- 匈牙利算法
  - 若 *P* 是图 *G* 中一条连通两个未匹配结点的路径，待匹配的边（边值为 0）和已匹配边（边值为 1）在 *P* 上交替出现，则称 ***P* 为一条增广路径**
  - **有一条增广路径 4—1—5—2—6—3**



2. 图匹配：求解最大二分匹配问题

- 匈牙利算法
  - 将第一条边改为已匹配（边值为 1），第二条边改为未匹配（边值为 0），以此类推。也就是将所有的边进行「反色」
  - 注意：修改以后，匹配仍然是合法的，但是匹配数加增加一对



2. 图匹配：求解最大二分匹配问题

- 匈牙利算法
  - 匹配数增多，且仍满足匹配要求（任意两条边都没有公共结点）
  - 这里:增广路径是一条可以使匹配数变多的路径!
  - 和最大流的增广路径含义不同，最大流中的增广路径是指可以增加流量的路径

2. 图匹配：求解最大二分匹配问题

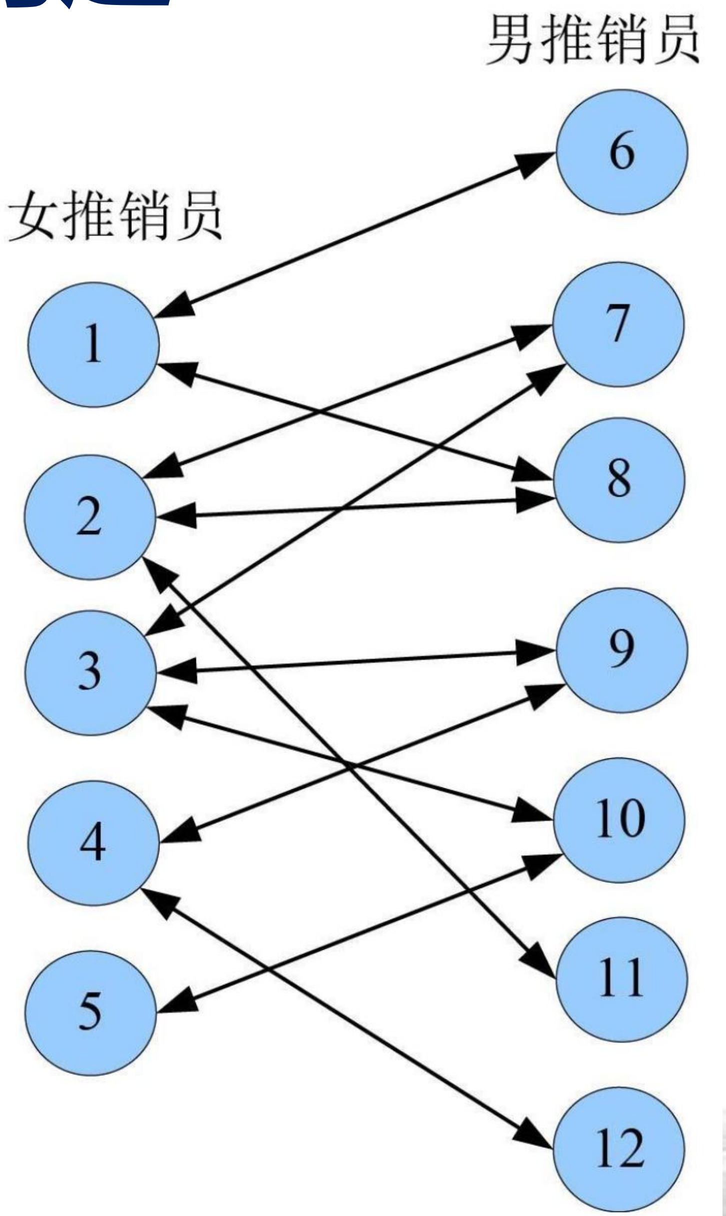
- 匈牙利算法
  - 在匹配问题中，增广路径的表现形式是一条交错路径，也就是说，这条由边组成的路径，它的第一条边还没有参与匹配，第二条边已参与匹配，第三条边没有参与匹配，最后一条边没有参与匹配，并且始点和终点还没有匹配。
  - 另外，单独的一条连接两个未匹配点的边显然也是交错路径
  - 算法思路:不停地找增广路径，并增加匹配的个数
  - 可以证明:当不能再找到增广路径时，就得到了一个最大匹配

2. 图匹配：求解最大二分匹配问题

- 匈牙利算法
  - （1）根据输入的数据，创建邻接表。
  - （2）初始化所有结点为未访问，检查第一个集合中的每一个结点u。
  - （3）依次检查u的邻接点v，如果v未被访问，则标记已访问，然后判断如果v未匹配，则令u、v匹配，即$match[u]=v$，$match[v]=u$，返回 ***true***；如果v已匹配，则从v的邻接点出发，查找是否有增广路径，如果有则沿增广路径反色，然后令u、v匹配，即$match[u]=v$，$match[v]=u$，返回 ***true***。否则，返回 ***false***，转向第（2）步。
  - （4）当找不到增广路径时，即得到一个最大匹配。

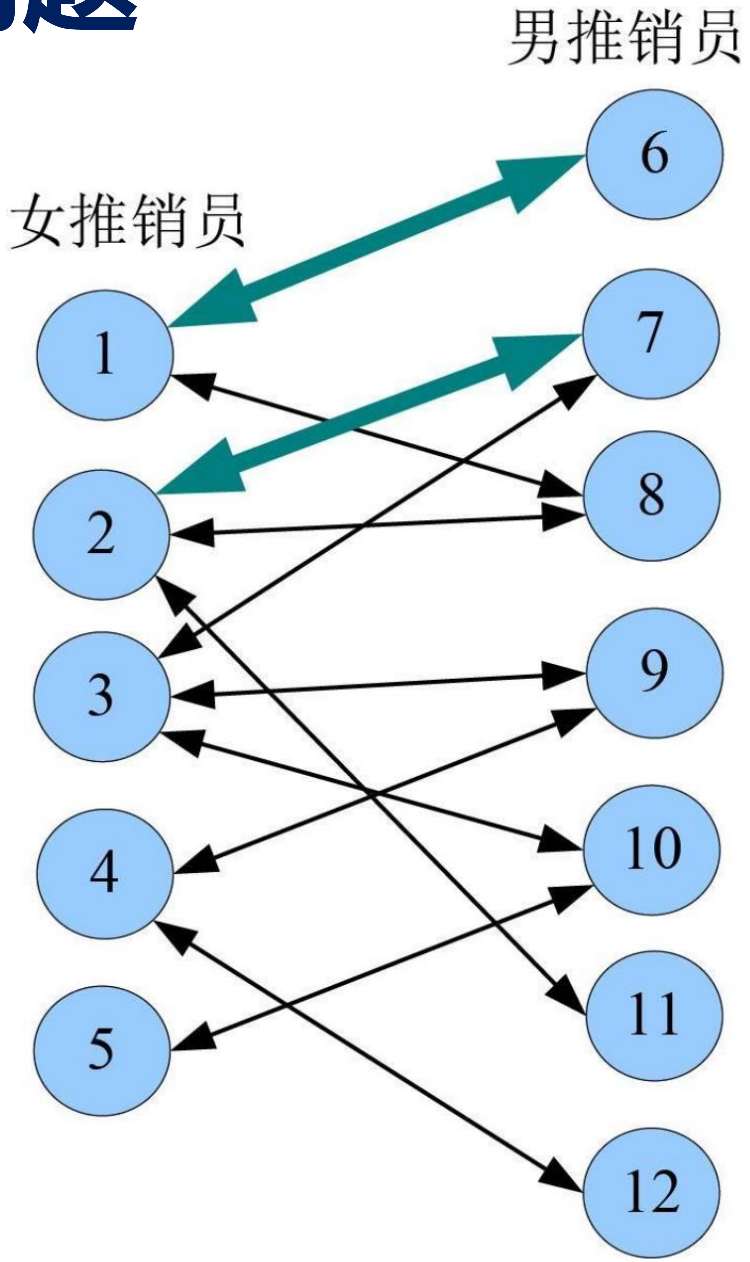
2. 图匹配：求解最大二分匹配问题

- 初始化访问数组*vis[i]=0*，*i=1*，…，*12*；检查 1 的第一个邻接点 6，6 未被访问，标记*vis[6]=1*。6 未匹配，则令 1 和 6 匹配，即*match[1]=6*，*match[6]=1*，返回 *true*



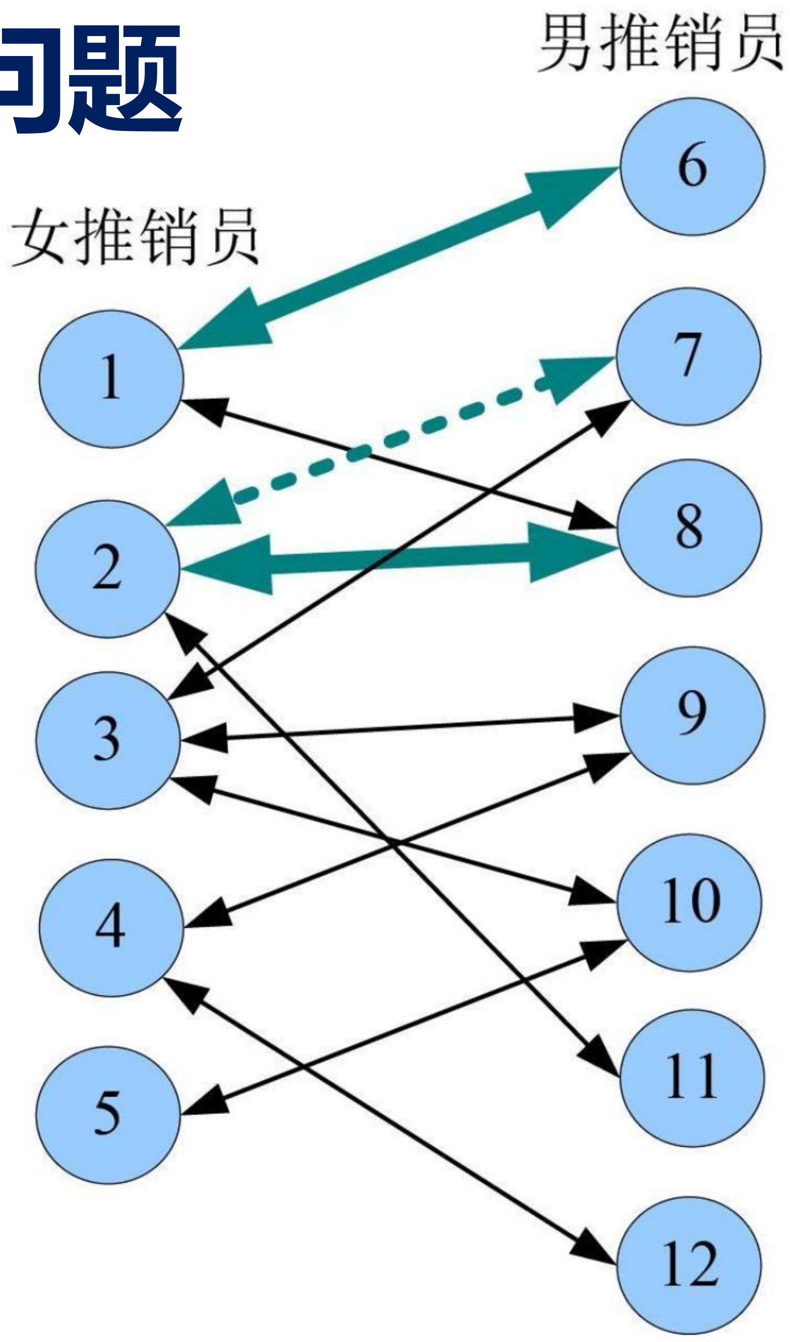
2. 图匹配：求解最大二分匹配问题

- 初始化访问数组*vis[i]=0*；检查 2 的第一个邻接点 7，7 未被访问，标记 *vis[7]=1*。7 未匹配，则令 2 和 7 匹配，即 *match[2]=7*，*match[7]=2*，返回 *true*



2. 图匹配：求解最大二分匹配问题

- 检查 3 的第一个邻接点 7，7 未被访问，标记*vis[7]=1*。7 已匹配，*match[7]=2*，从 7 的匹配点 2出发寻找增广路径，实际上就是为 2 号结点再找一个其他匹配点，如果找到了，就「舍己为人」把原来的匹配点 7 让给 3 号，
- 从 2 出发，检查 2 的第一个邻接点 7，7 已访问，检查第二个邻接点 8，8 未被访问，标记 *vis[8]=1*。8 未匹配，则令*match[2]=8*，*match[8]=2*，返回 true



2. 图匹配：求解最大二分匹配问题

- 检查 4 的第一个邻接点 9，9 未被访问，标记*vis[9]=1*。9 未匹配，则令*match[4]=9*，*match[9]=4*，返回 *true*。
- 检查 5 的第一个邻接点 10，10 未被访问，标记 *vis[10]=1*，10 未匹配，则令 *match[5]=10*，*match[10]=5*，返回 *true*