第七章 MaxMin方法

张炜
计算机科学与技术学院

HIT
CS&E

教学目的

难点： **Max-Min**关系及其在算法设计和分析中的应用

重点： 基本算法层面： **(1)**最大流-**最小割算法
(2)最大匹配-**最小覆盖算法
(3)基本图论算法的总结和复习

算法设计技术层面： **(1)Max-min**方法
(2)精益求精的算法设计过程

问题特征分析能力层面： **(1)**准确，渐进

参考书和最新文献：

1. 网络流：理论、算法和应用. 机械工业出版社，**2004**
2. **Incremental graph pattern matching, VLDB, 2011**
3. **On the complexity of view update and application in annotation propagation. TKDE, 2012**

7.1 Maximum Flow

7.1.1 流网络与流

7.1.2 Ford-Fulkerson方法

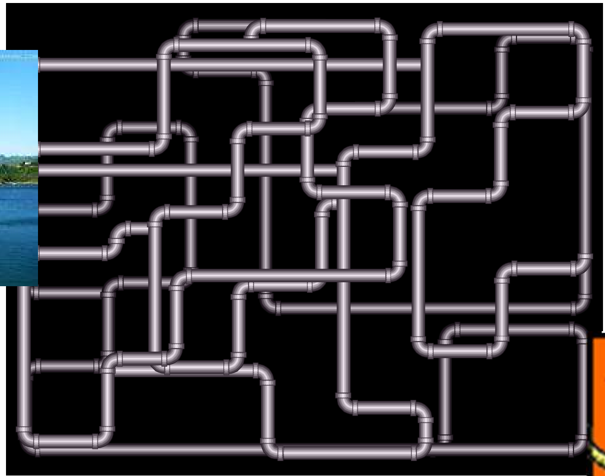
7.1.3 推送复标算法

7.1.4 复标前置算法

构建和使用各种网络时，需要考虑网络的通行能力

网络的通行能力受网络结构的限制

- 道路的宽窄
- 管道的粗细
- ……

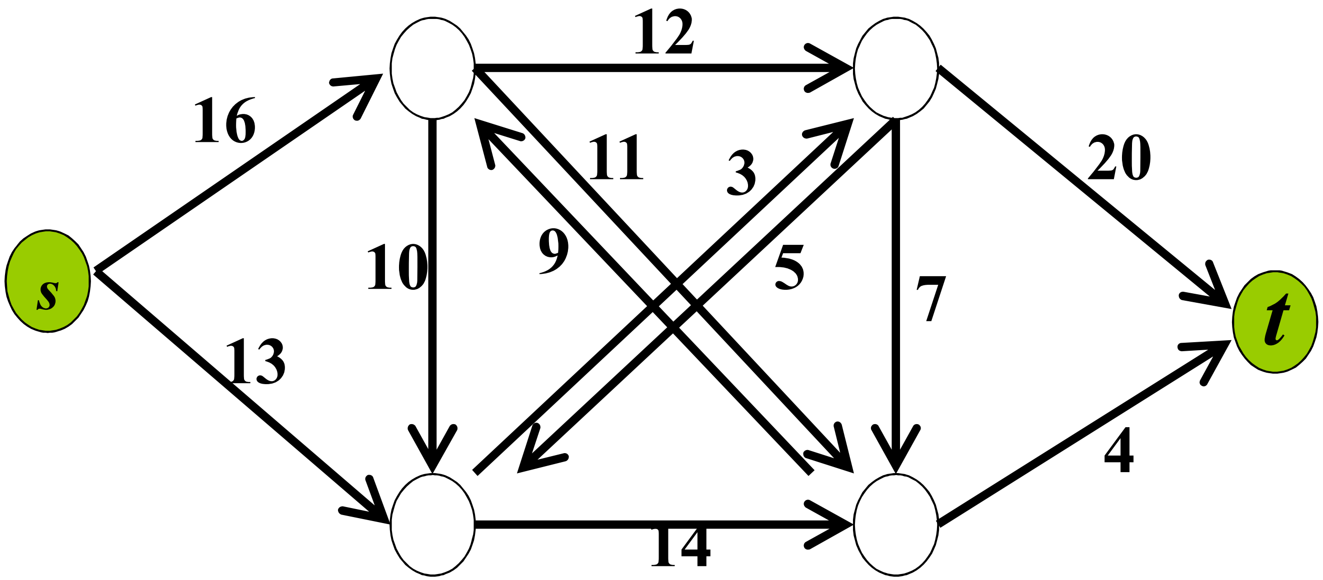

流网络的种类

- 英特网
- 电话网
- 高速路网
- 铁路网
- 电网
- 输气网络
- 排水网络
- 输水网络

能否建立通用模型来研究网络的最大通行能力呢？

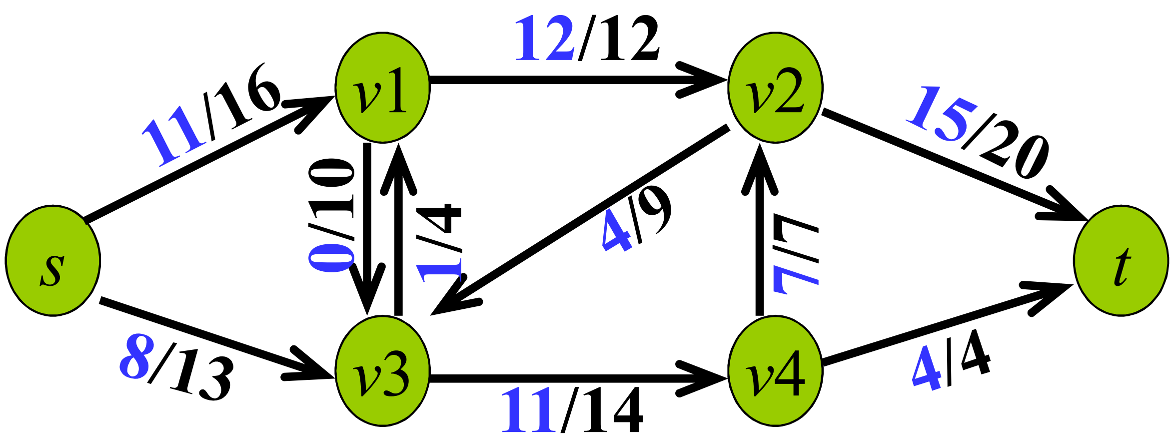
加权有向图G=(V,E)

- 顶点表示网络中的关键节点
- 边 uv表示u和v之间的连接，容量$c(u,v)\geq 0$表示边的通行能力；如果$uv\notin E$，则定义容量$c(u,v)=0$
- 两个特殊顶点s和t，s称为源(source)，t称为汇(sink)
- G中每个顶点均位于某条由s到t的路径上($|E|\geq|V|-1$)

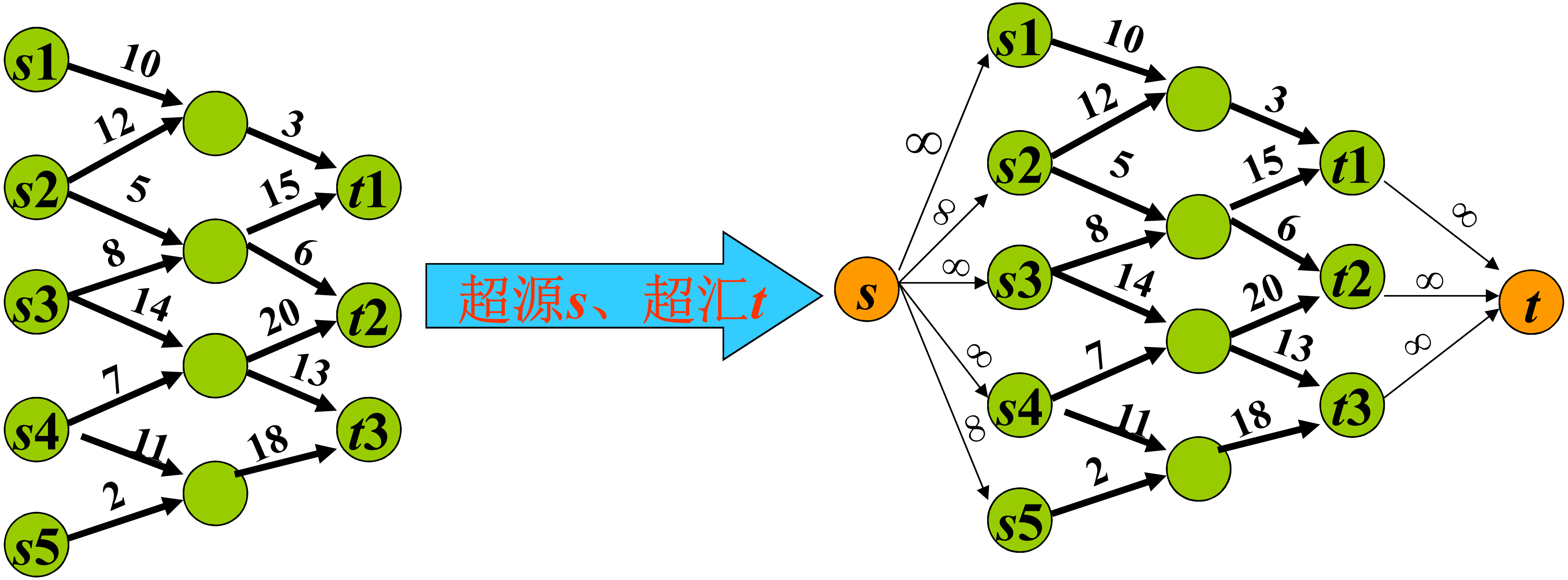


流网络$G=(V,E)$上的一个流

- 是一个实值函数f:$V\times V\rightarrow R$, 满足
  - 容量约束：$f(u,v)\le c(u,v)$ 对$\forall uv\in E$成立
  - 反对称性：$f(u,v) = - f(v,u)$
  - 守恒约束：$\sum_{v\in V} f(u,v)=0$对任意 $u\in V-\{s,t\}$成立
- $f(u,v)$称为顶点u到顶点v的流量,可为正、负、零值
- 流f的值定义为$|f|=\sum_{v\in V} f(s,v)$



多源多汇的网络

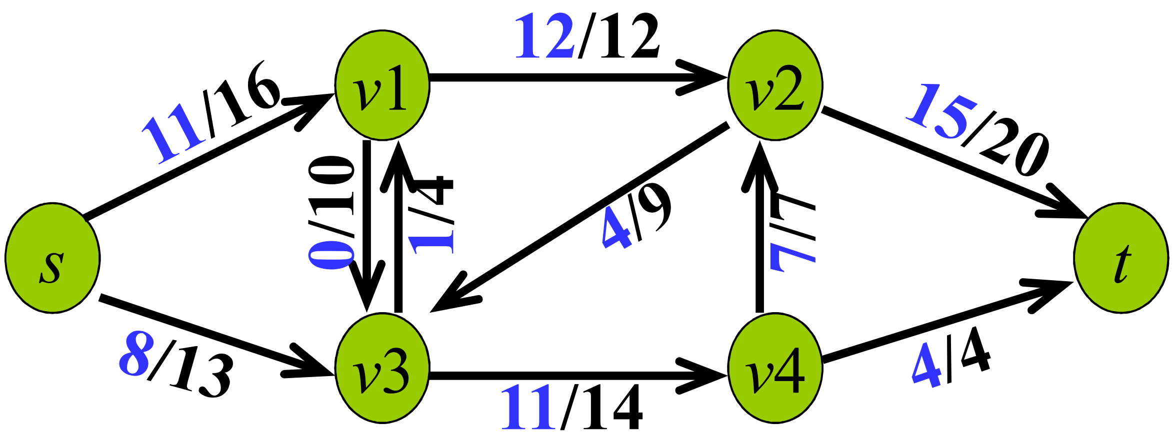


只需讨论单源单汇的网络流

最大流问题

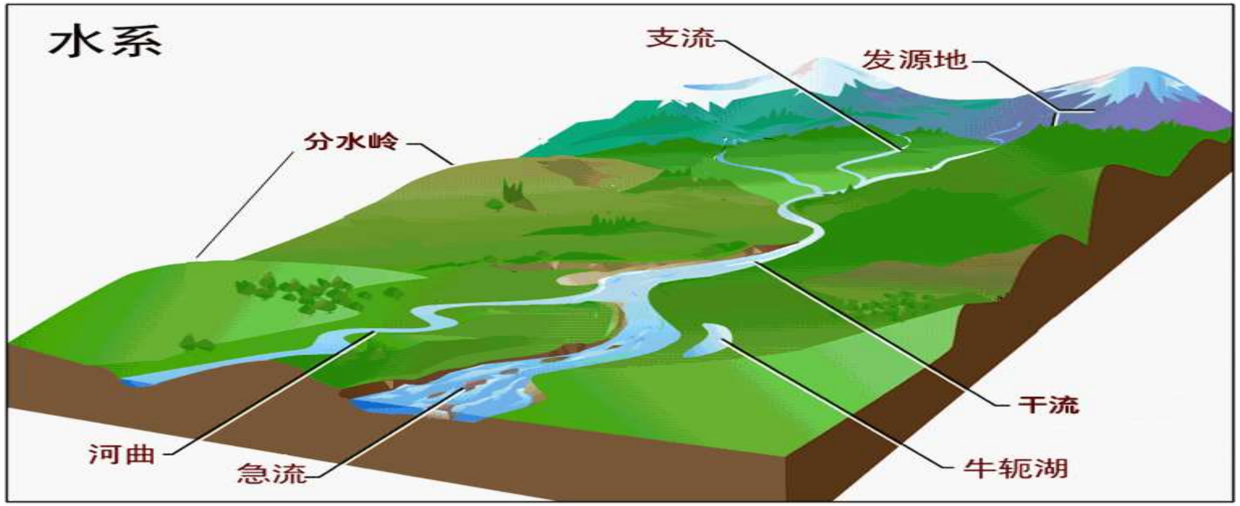
给定流网络 $G=(V,E)$,其源为s，汇为t，找出从s到t的最大流.

- 怎样高效率地找?



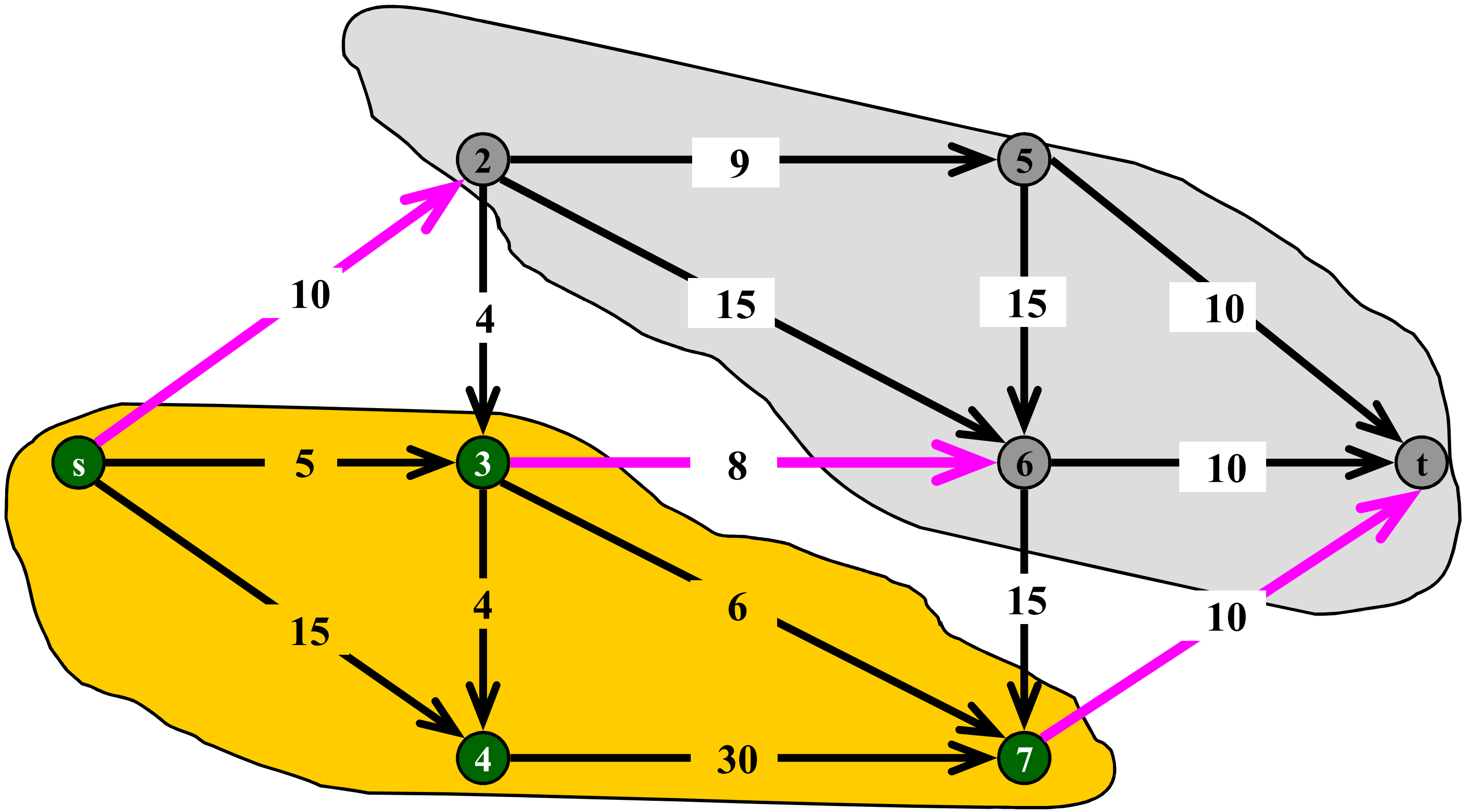
河水的最大流量取决于

干流中河道狭窄处的通行能力



这种观察能否用于研究最大流问题呢？

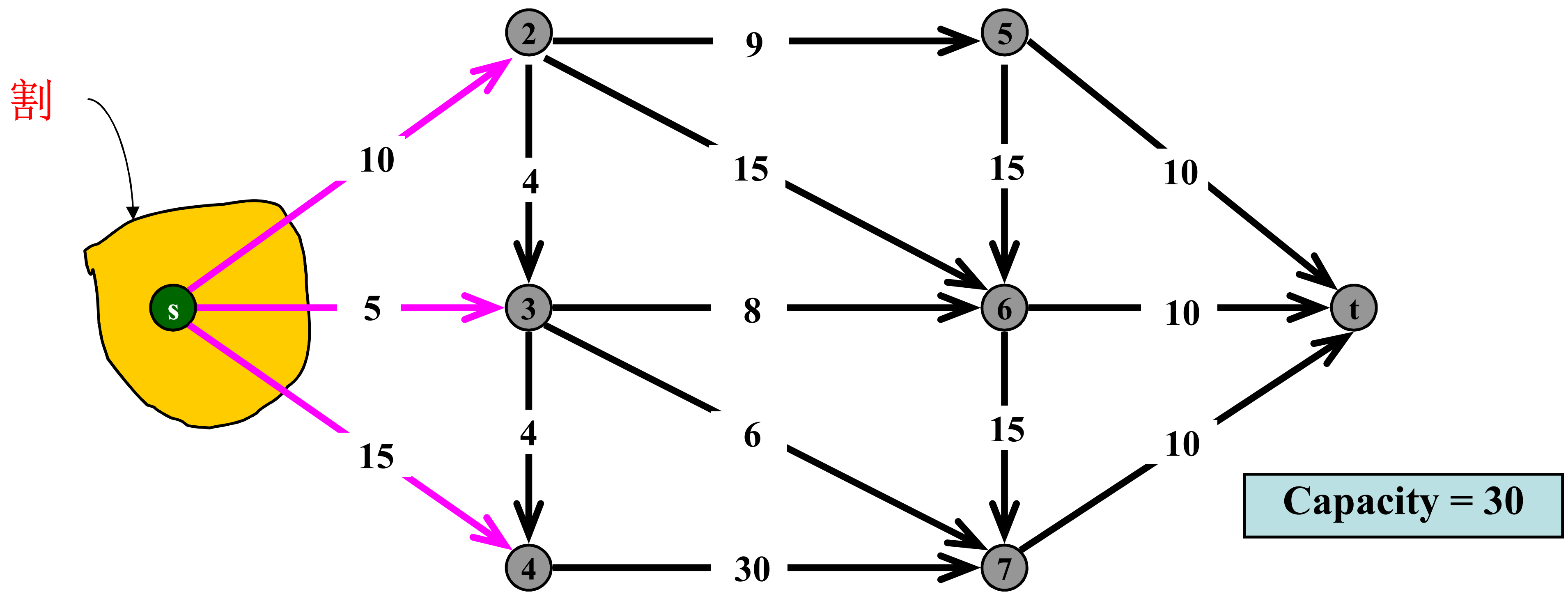
从s流到t的最大流量不会超过10+8+10=28



割

给定流网络 $G=(V,E)$,其源为s，汇为t

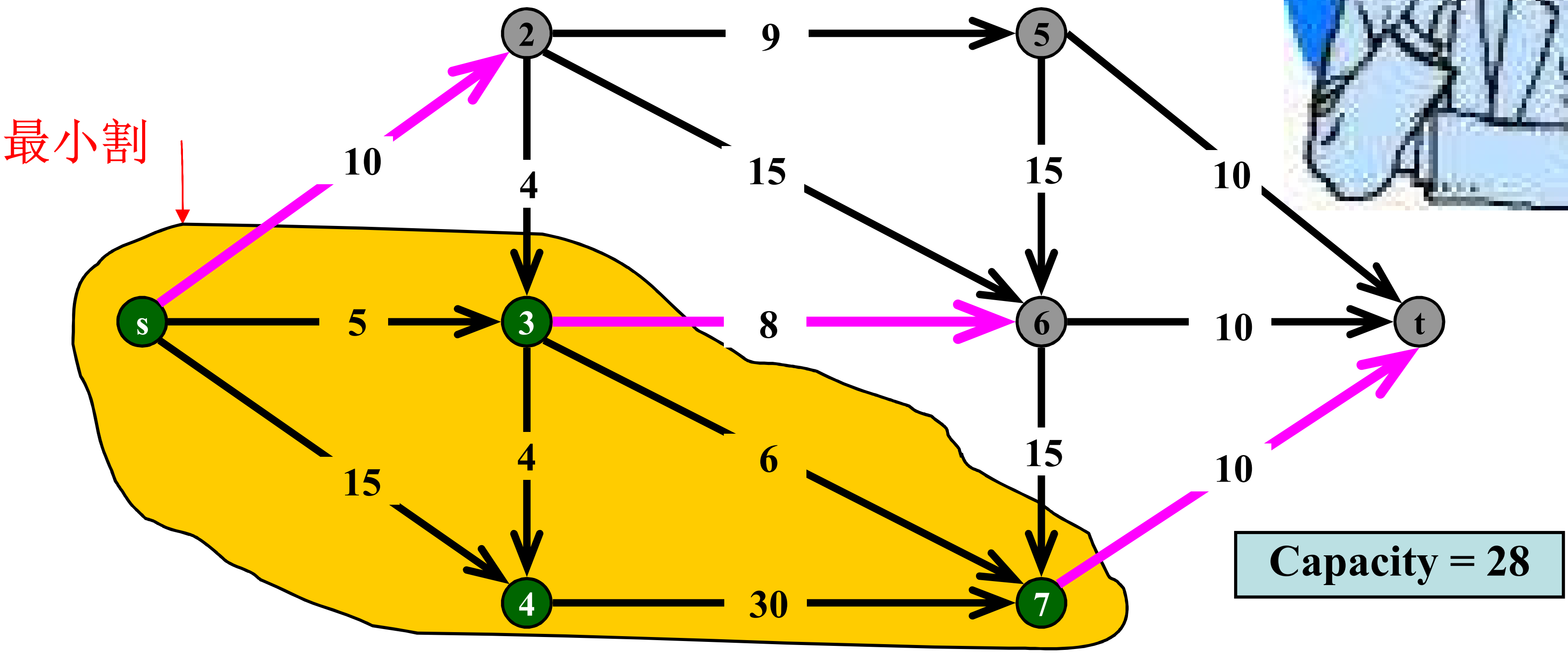
G的一个割(cut)是V的2-集合划分S,T ($T=V-S$)使得$s\in S$, $t\in T$



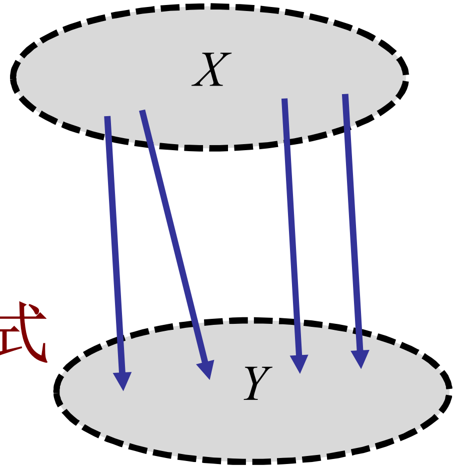
割的容量定义为 $$c(S,T)=\sum_{u\in S,v\in T}c(u,v)$$

给定流网络 $G=(V,E)$,其源为s，汇为t,
找出流网络G中容量最小的割.

- 怎样高效率地找?



- 设$X,Y\subseteq V$　记$f(X,Y)=\sum_{x\in X}\sum_{y\in Y}f(x,y)$
- 称$f(X,Y)$顶点集X和Y之间的流量
- 守恒约束等价于 $f(u,V)=0$对$\forall u\in V-s-t$
- 上述表示经常用来简化网络流涉及的等式



引理1　设f是流网络$G=(V,E)$中的一个流，则

- $f(X,X)=0$　对$\forall X\subseteq V$成立
- $f(X,Y)=-f(Y,X)$ $\forall X,Y\subseteq V$成立
- $\forall X,Y,Z\subseteq V$ 且$X\cap Y=\phi$，有

$$f(X\cup Y,Z)=f(X,Z)+f(Y,Z)$$
$$f(Z,X\cup Y)=f(Z,X)+f(Z,Y)$$

故，$|f|=f(s,V)\quad=f(V,V)-f(V-s,V)\quad=-f(V-s,V)\quad=f(V,V-s)$

$$=f(V,t)+f(V,V-s-t)$$
$$=f(V,t)$$

HIT
CS&E

最大流—最小割间弱对偶关系

引理2 给定流网络 $G=(V,E)$, s是源, t是汇. 设f是G上的一个流，S,T是G的一个割，则$f(S,T)=|f|$.

证明: $f(S,T)=f(S,V)-f(S,S)$ 引理1第3部分

$=f(S,V)$ 引理1第1部分

$=f(s,V)+f(S-s,V)$ 引理1第3部分

$=f(s,V)$ $f(S-s,V)=0$

$=|f|$

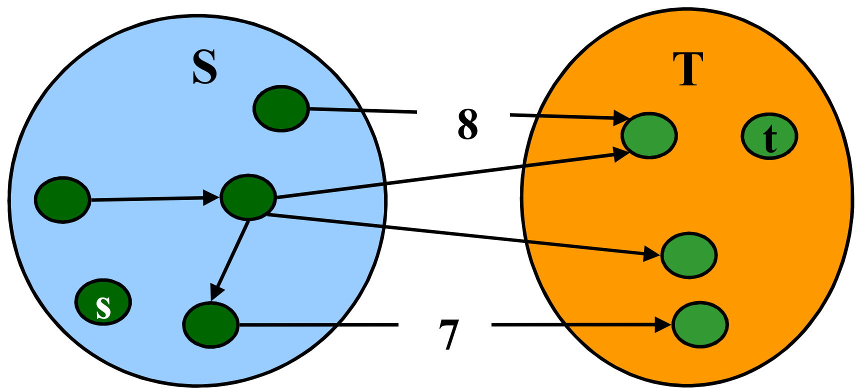
引理3 给定流网络 $G=(V,E)$. 设f是G上的一个流，S,T是G的一个割，则$|f|\leq c(S,T)$

$$|f|=f(S,T)$$

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

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

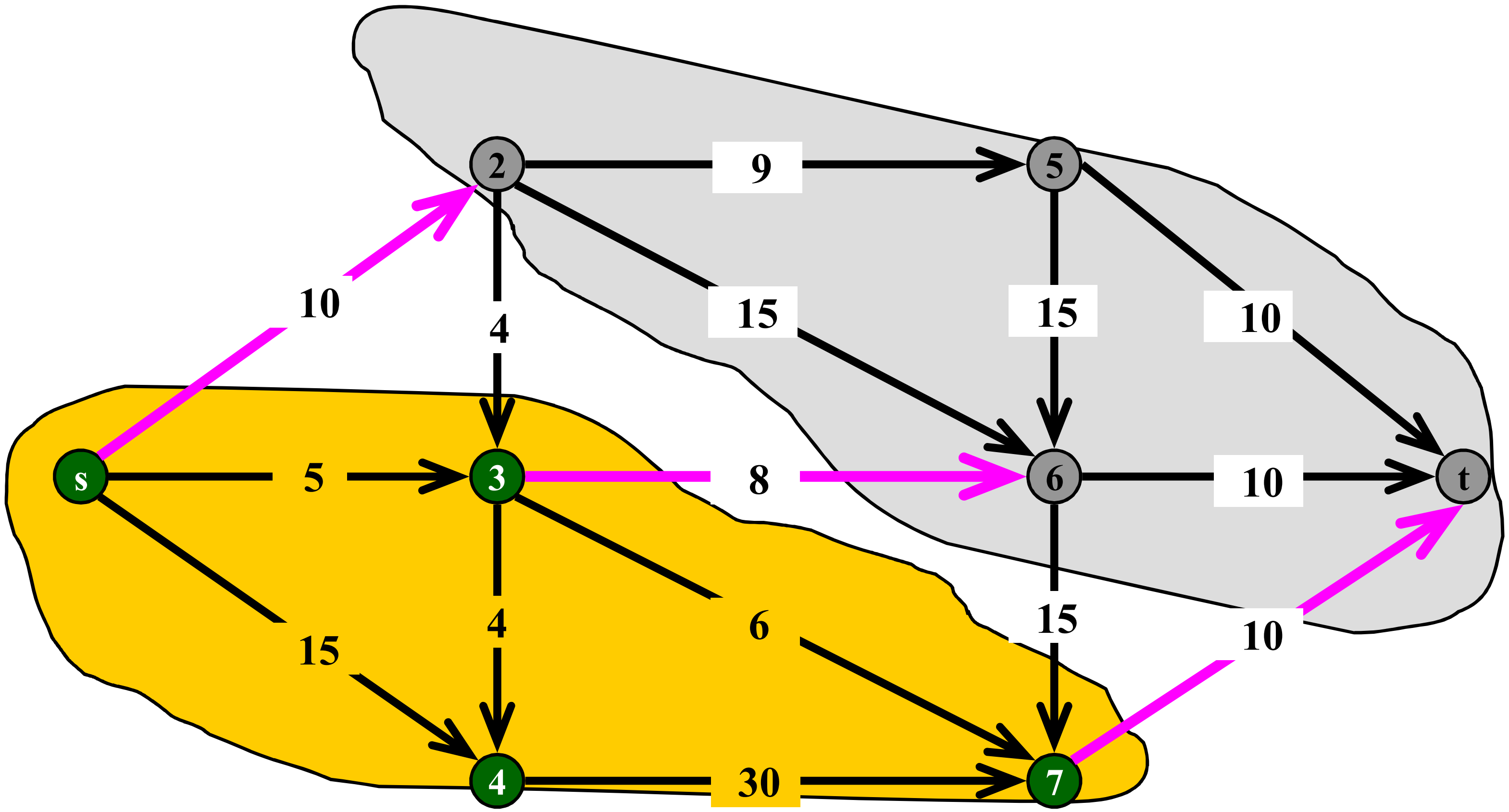
$$=c(S,T)$$



最大流—最小割间的弱对偶关系

HIT CS&E

引理3 给定流网络 $G=(V,E)$. 设f是G上的任意一个流，S,T 是G的任意一个割，则$|f| \le c(S,T)$



怎么用这个**max-min**关系来高效地求出最大流（最小割）呢？

利用max-min关系设计最大流算法

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

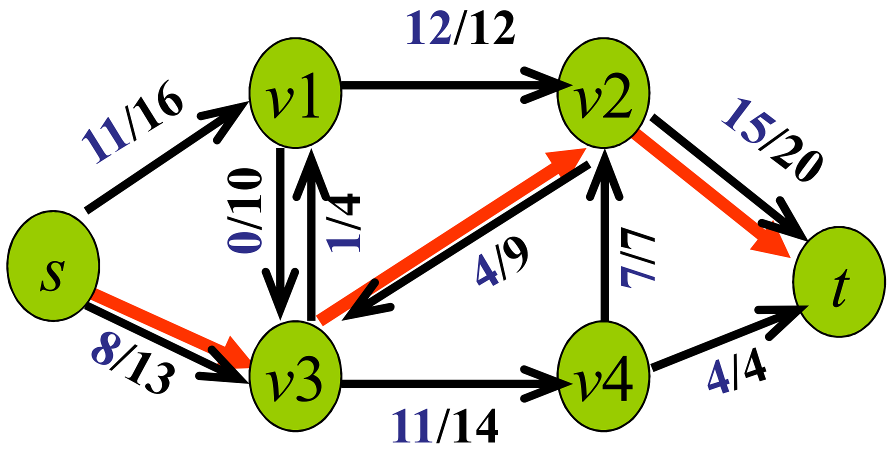
max-min关系提供了高效求解最大流**-**最小割问题的机制！
关键在于第**2**步怎么做？

HIT
CS&E

7.1.2 Ford-fulkerson方法

算法的基本思想

- 对于当前的流,…
- …找出一条从s到t的路径 p (增广路径)和正数$a > 0$, 使得p上的每条边 uv的流量增加a之后仍然满足容量约束，即 $f(u,v) + a \le c(u,v)$
- 则将p上每条边的流量增加a，得到一个更大的流



a=4

怎么增大当前的流？

Ford-fulkerson算法概要

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 沿路径p增大流量得到更大的流f

4 return f

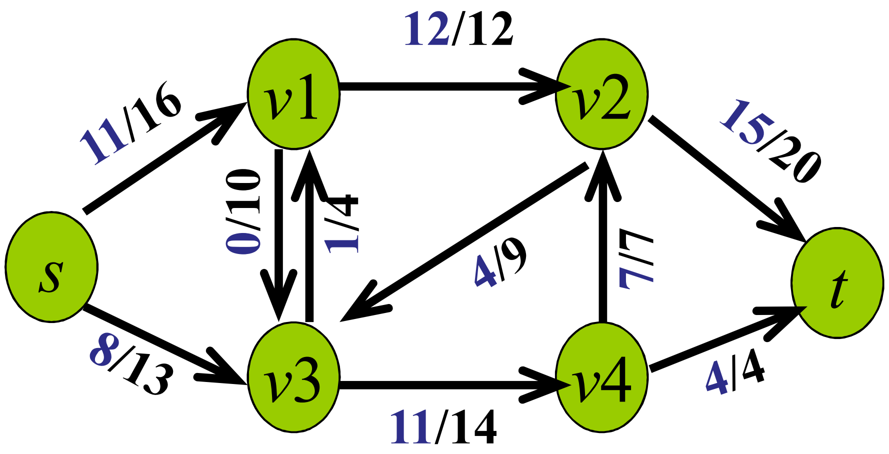
- 增广路径如何找?
- 增广路径上可以增加的流量有多大?
- 该方法总能找到最大流吗?

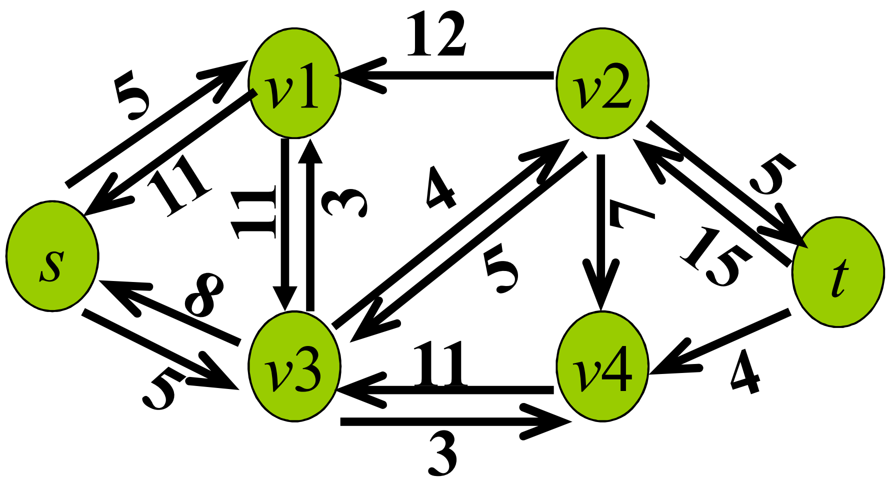
剩余网络(Residual Network)

增广路径如何找?

我们擅长找路径

寻找增广路径能否转变为在某个图中找路径呢?





$E_f=\{uv: c_f(u,v)=c(u,v)-f(u,v)>0\}$

增广路径如何找?

- 增广路径是**Residual network**中从s到t的路径
  - **Residual capacities:** $c_f(u,v) = c(u,v) - f(u,v)$
  - **Residual network:** $G_f=(V,E_f)$, 其中

    $$E_f = \{(u,v) \in V \times V \mid c_f(u,v) > 0\}$$

  - $f(u,v) < c(u,v)$, 则 $c_f(u,v) = c(u,v) - f(u,v) > 0$, $(u,v) \in E_f$

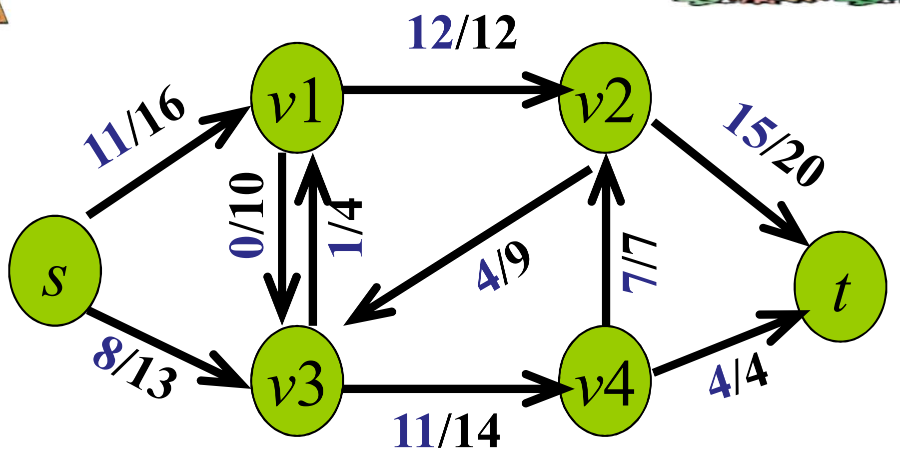
    $f(u,v) > 0$ 则 $c_f(v,u) = c(v,u) - f(v,u) > 0$, $(v,u) \in E_f$

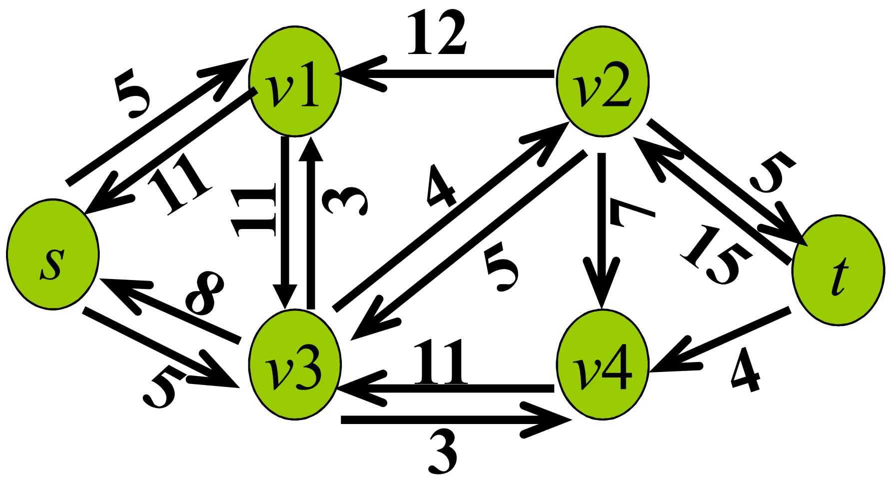
    $c(u,v)=c(v,u)=0$, 则$f(u,v)=f(v,u)=0$, 进而 $c_f(u,v) = c_f(v,u) = 0$

  注意: E_f中的边要么是 E中的边，要么是E中边的反向边: $|E_f| \le 2|E|$
- **Residual Network**本身也可以看成是流网络

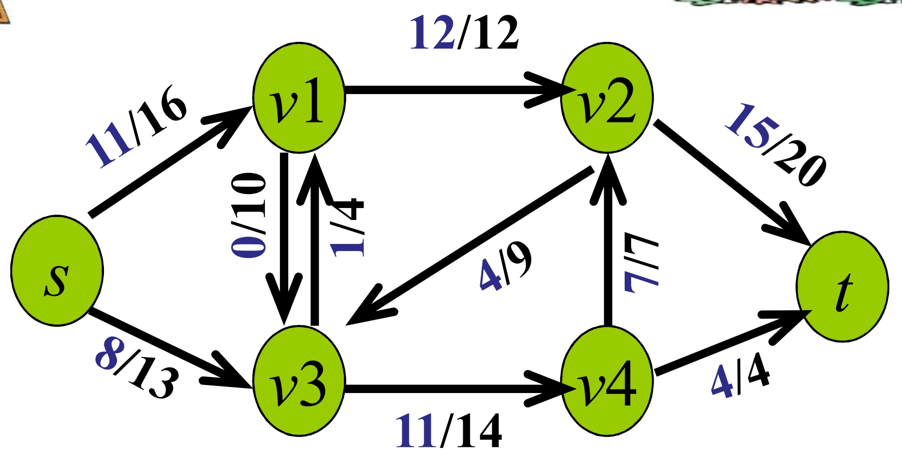
HIT
CS&E

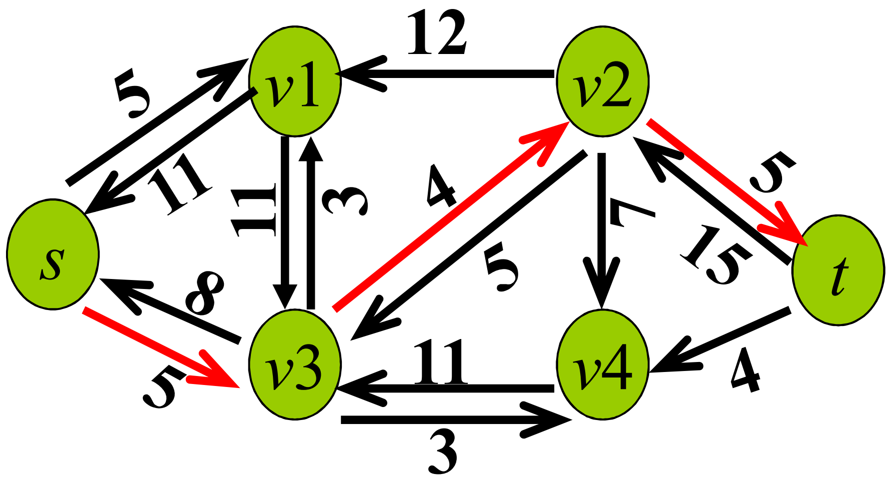
利用剩余网络找增广路径

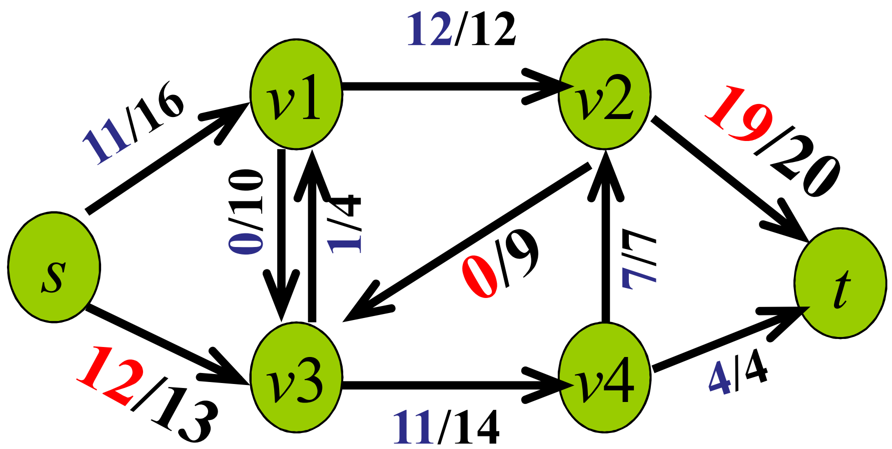


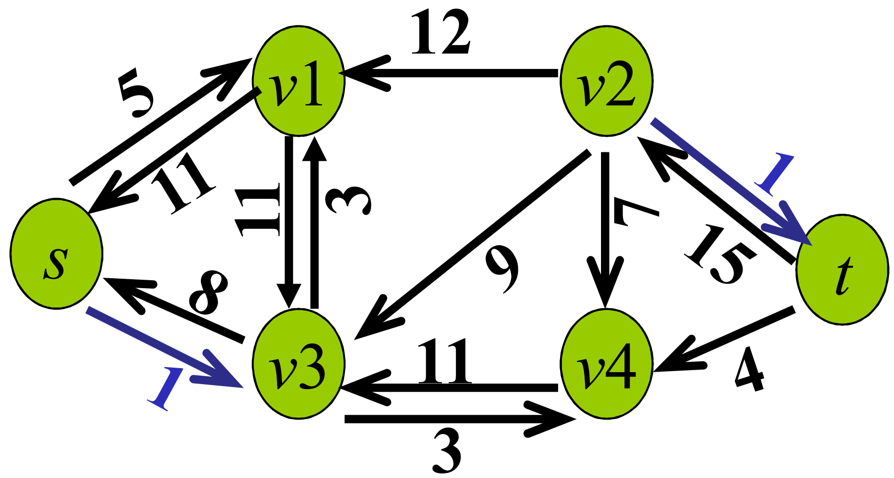


利用剩余网络找增广路径









增广路径上可以增大多少流量?

- P是G_f中的一条增广路径
  - 其剩余容量 $c_f(p) = \min\{c_f(u,v): (u,v)$ 是路径 p上的边$\}$
- 增广过程： 对路径p上的每条边uv
  - 要么在边uv的流量上增加 $c_f(p)$,即 $f(u,v)=f(u,v)+c_f(p)$
  - 要么在边vu的流量上减去$c_f(p)$, 即 $f(v,u)=f(v,u)-c_f(p)$
  - 具体属于哪种情况，根据G
- 增广过程完成后，得到值更大的流

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

Ford-fulkerson算法的正确性

引理4 给定流网络 $G=(V,E)$,s是源,t是汇; f是G上的一个流. G_f是流f在G上导出的Residual Network, f'是G_f上的一个流. 则$f+f'$是G上值为$|f|+|f'|$的流。

其中，$(f+f')(u,v)=f(u,v)+f'(u,v)$

证明:

(1)验证反对称性

(2)验证容量约束

(3)验证守恒约束

(4)验证$|f+f'|=|f|+|f'|$

Ford-fulkerson算法的正确性

引理4 (最大流—最小割定理)给定流网络 $G=(V,E)$,s是源,t是汇; f是G上的一个流. 则下列论断等价。

(1)f是最大流;

(2)G_f中不存在增广路径

(3)对G的某个割(S,T), $|f|=c(S,T)$

证明: (1)⇒(2) 反证. 设p是G_f中最大流f对应的增广路径，其剩余容量为f_p。由引理4知，$f+f_p$是一个值比$|f|$大的流. 这与f是最大流矛盾。

(2)⇒(3) 由于G_f中没有从s到t的路径，定义$S=\{v: G_f$中存在从s到v的路径$\}$，$T=V-S$。显然$s\in S$且$t\in T$。$\forall u\in S$且$v\in T$，$f(u,v)=c(u,v)$，否则$uv\in E_f$进而导致$v\in S$。于是S,T是G的一个割。由引理2知道$|f| = f(S,T)=c(S,T)$

(3)⇒(1) 引理3表明$|f|\le c(S,T)$, 故$|f|=c(S,T)$表明f是最大流。

复杂性(1)

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 $\forall uv \in E[G]$ do
    $f(u,v) \leftarrow 0$
    $f(v,u) \leftarrow 0$
While G_f存在增广路径p do $O(|E_f|)=O(|E|)$
    $c_f(p)=\min\{c_f(u,v) \mid uv$是$p$上的边$\}$
    For p上的每条边uv do
        If uv是流网络中的边 Then
            $f(u,v) \leftarrow f(u,v)+c_f(p)$
            $f(v,u) \leftarrow -f(u,v)$
        Else
            $f(v,u) \leftarrow f(v,u)-c_f(p)$
            $f(u,v) \leftarrow -f(v,u)$
        修改剩余中网络Gf相应的边

$O(|p|)=O(|E|)$

复杂性(2)

Ford-Fulkerson算法的时间复杂度$O(|f^*||E|)$

怎么改进**Ford-Fulkerson**算法呢？

1.选用好的路径计算方法

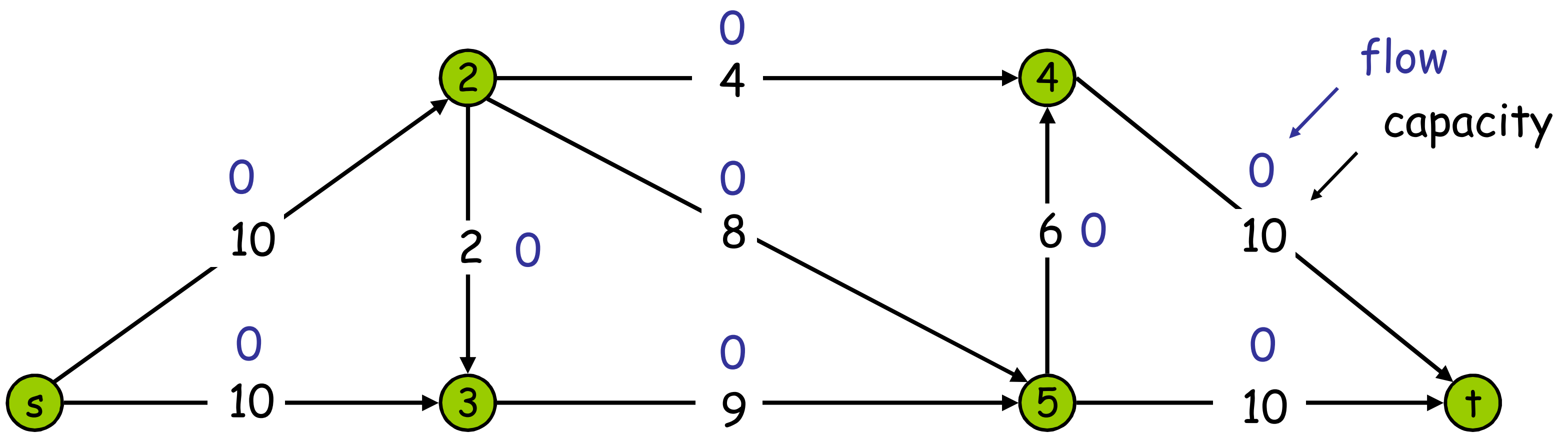
- 明确选用**BFS (Edmonds-Karp**算法**)O(VE²)**

2.计算特殊的增广路径

- 第**5**步：剩余容量 $c_f(p)$达到最大值

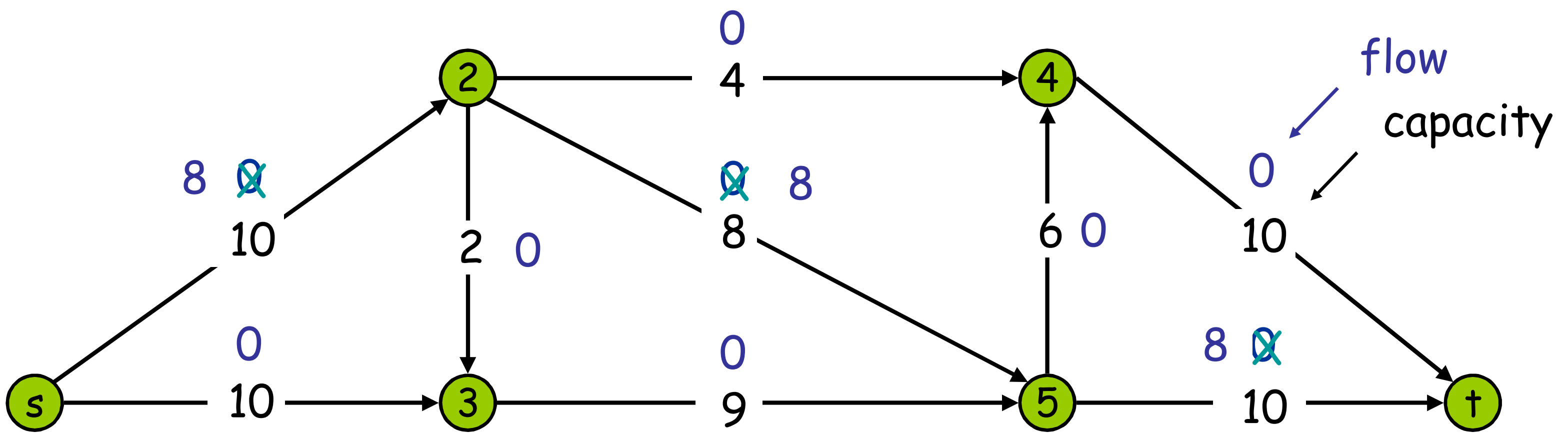
3.利用"最大流等于最小割"这一结论重新设计其他算法

G:



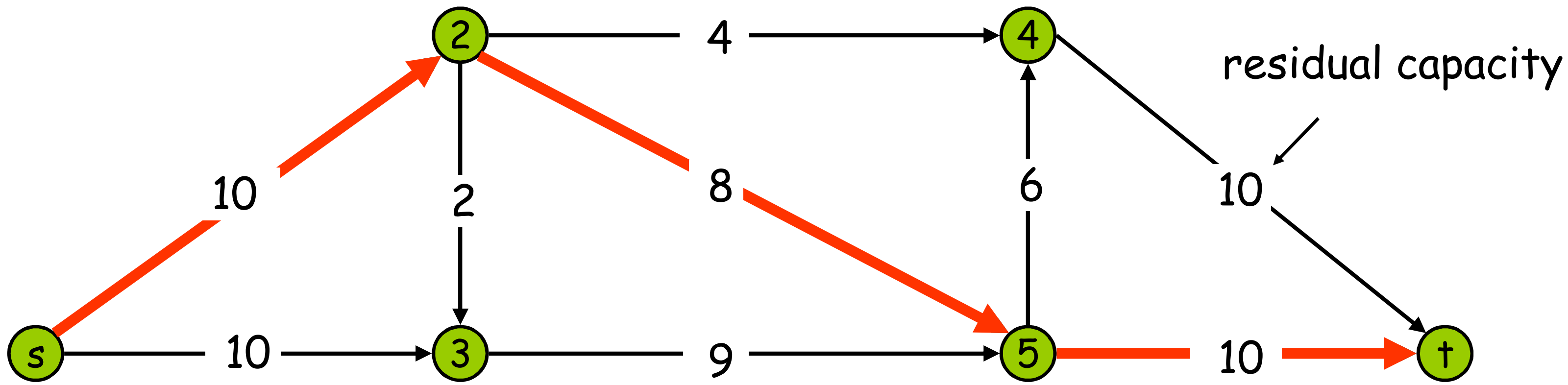
Flow value = 0

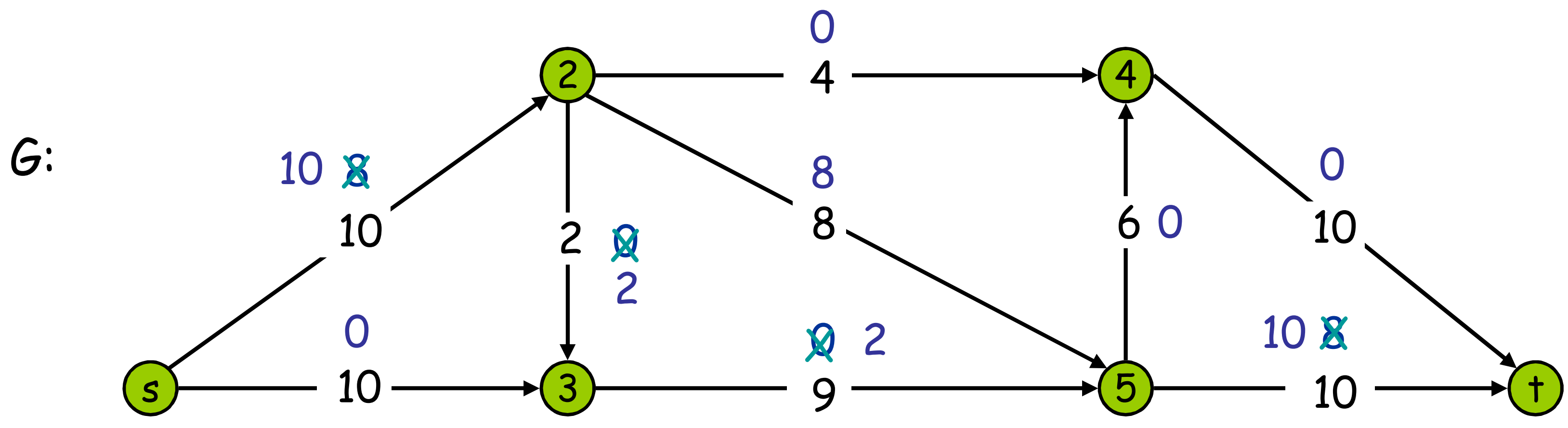
G:



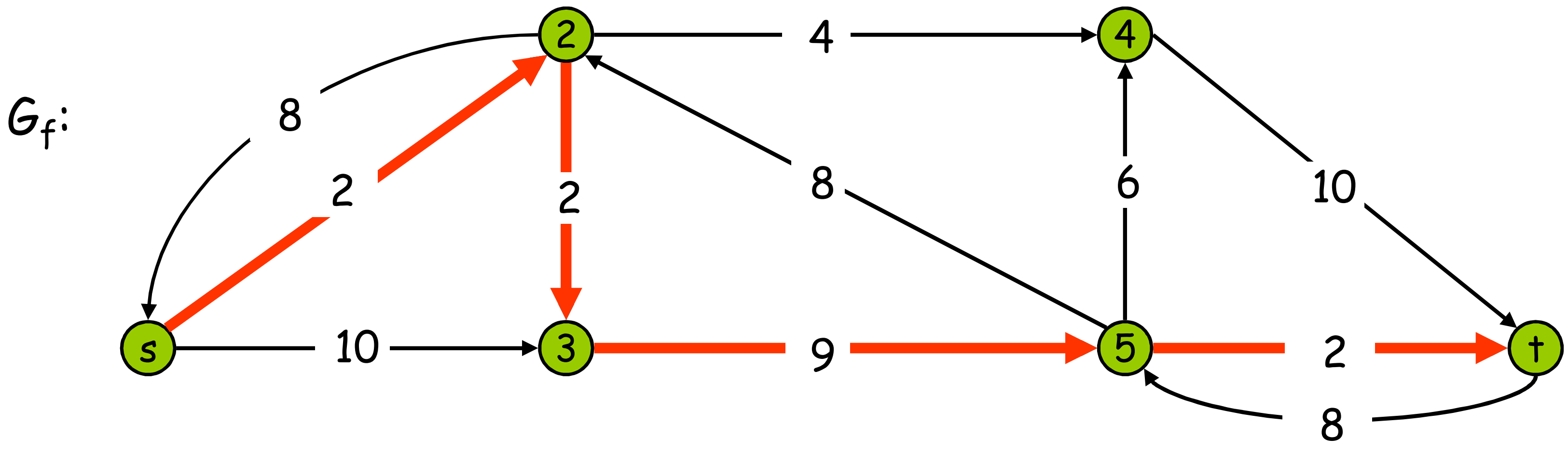
Flow value = 0

G_f:

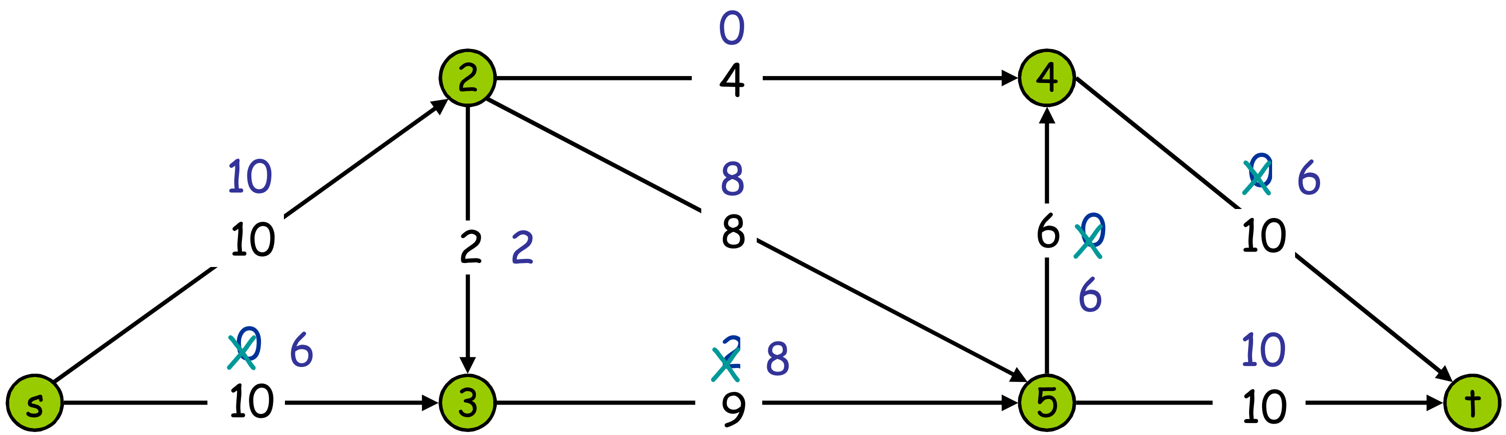



G:
s
2
3
4
5
t
10 8
10
0
10
2 0
2
0
4
8
8
0 2
9
6 0
0
10
10 8
10

Flow value = 8

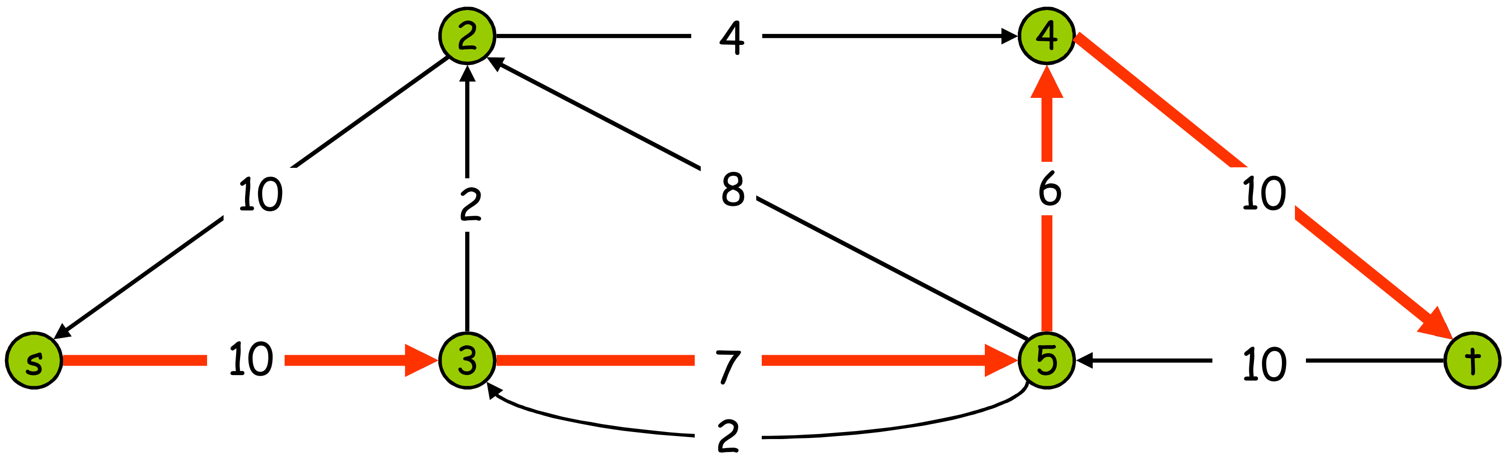

Gf:
s
2
3
4
5
t
8
2
10
2
4
8
9
6
10
2
8

G:

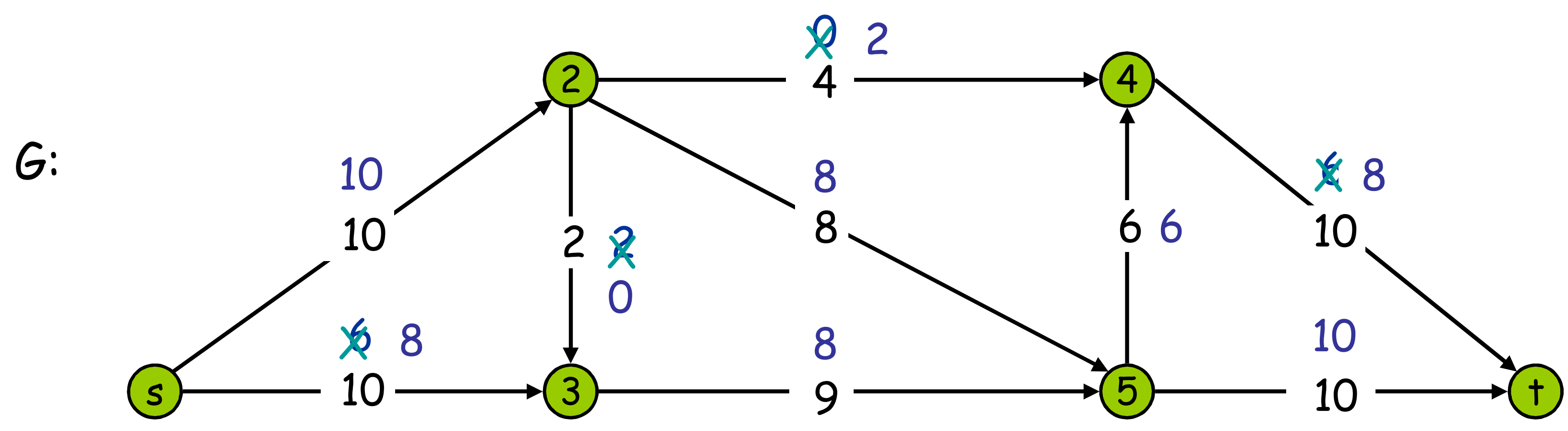


Flow value = 10

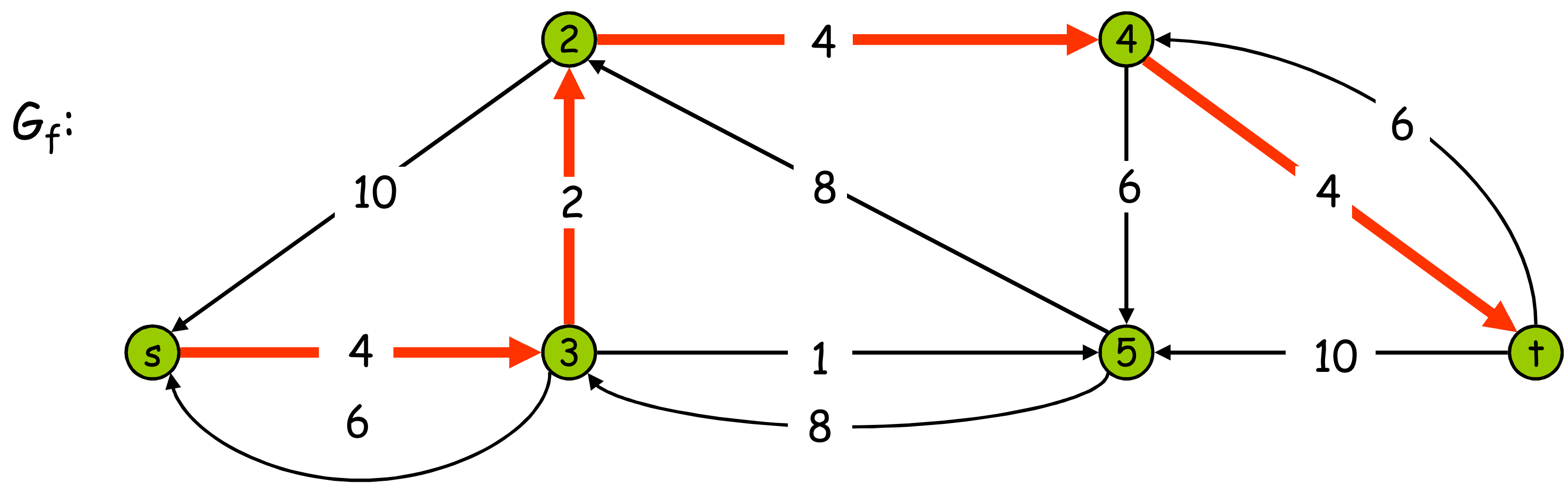
G_f:



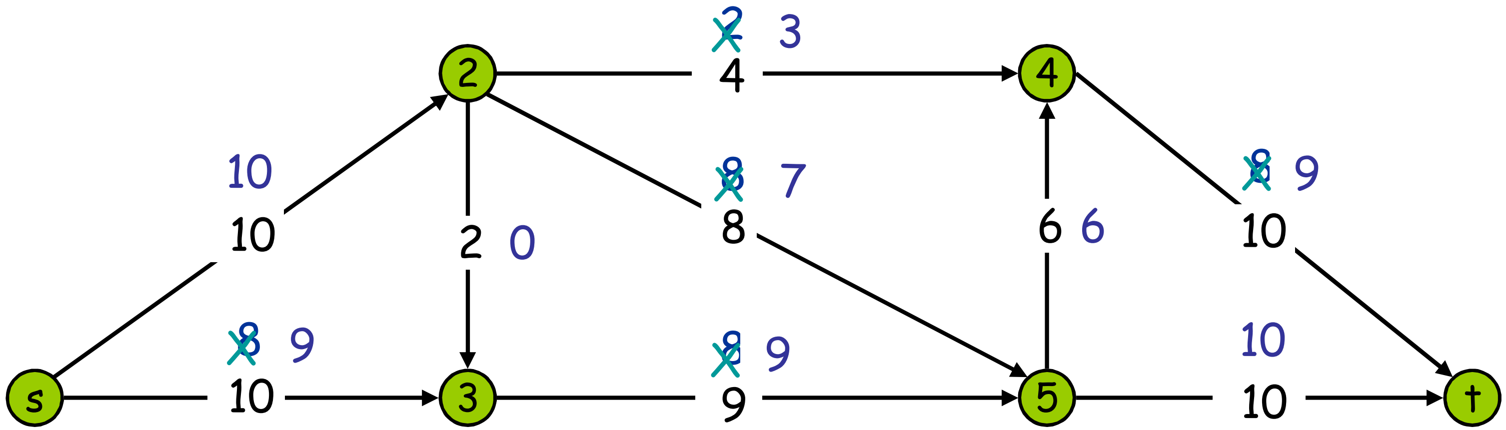
HIT
CS&E



Flow value = 16

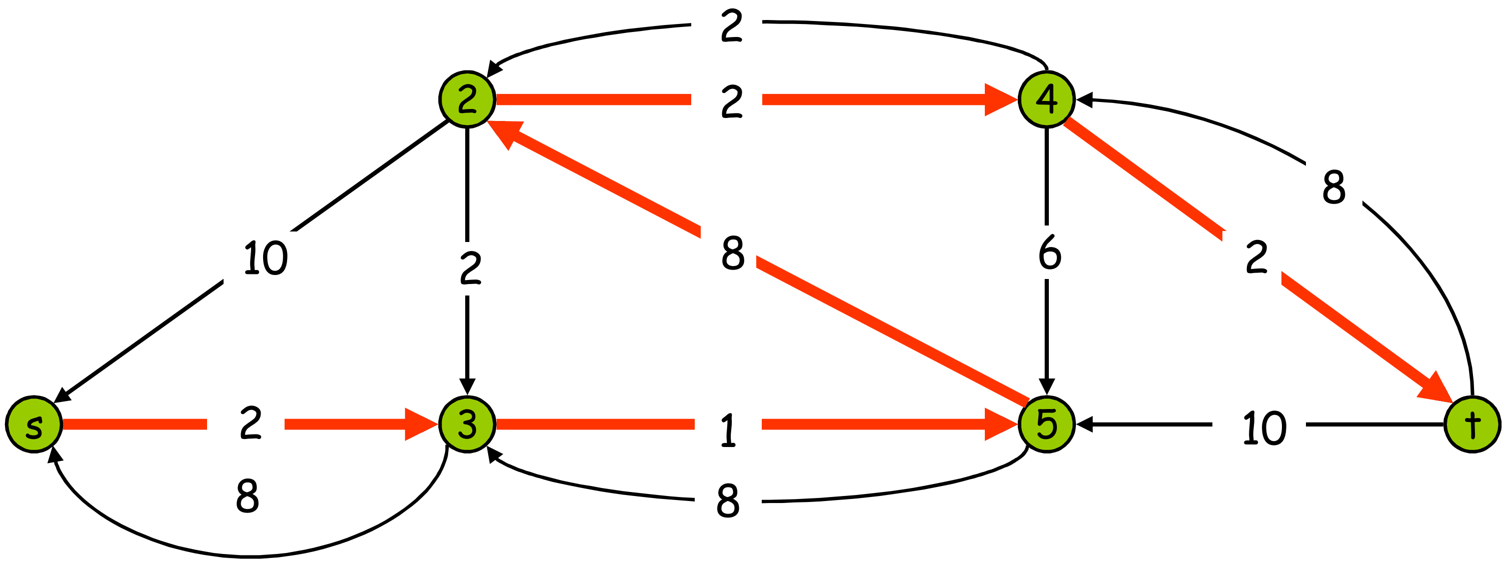


G:

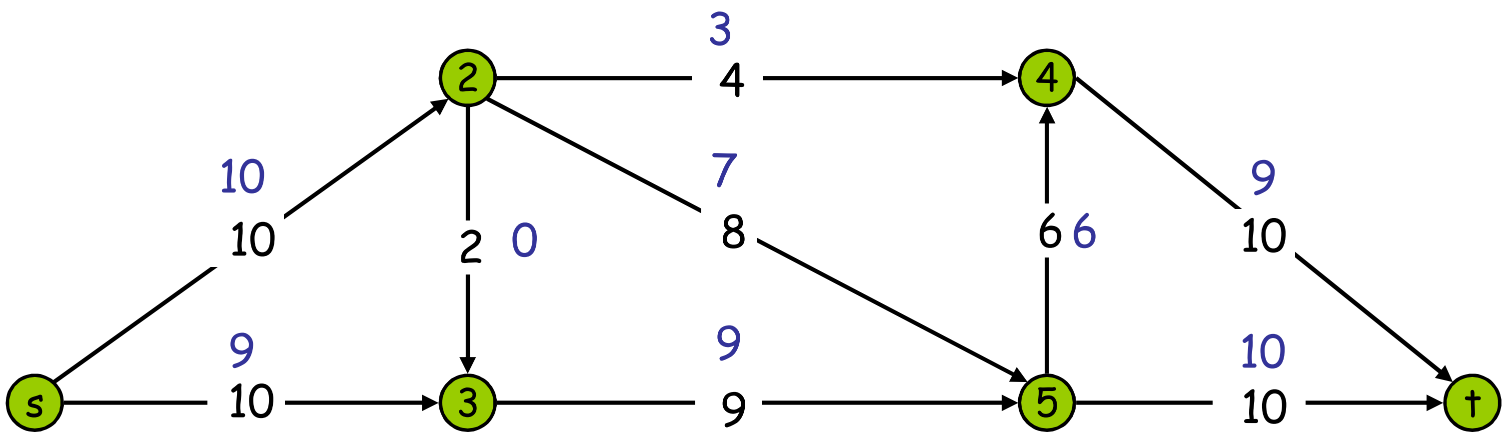


Flow value = 18

G_f:

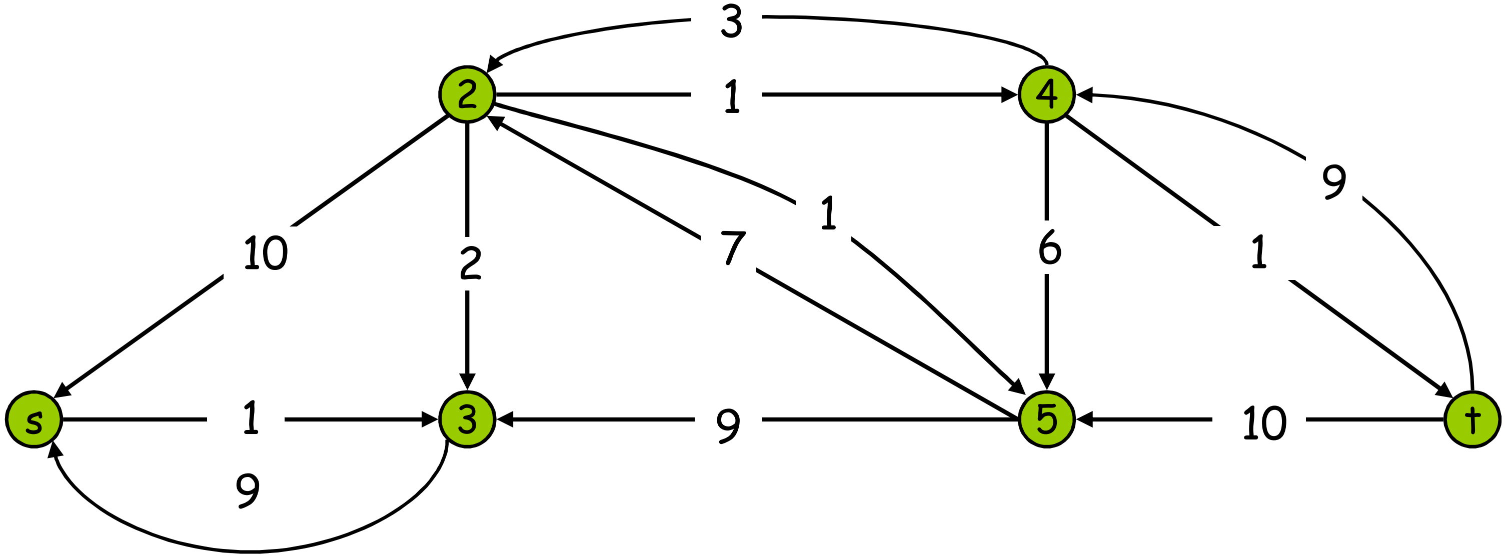


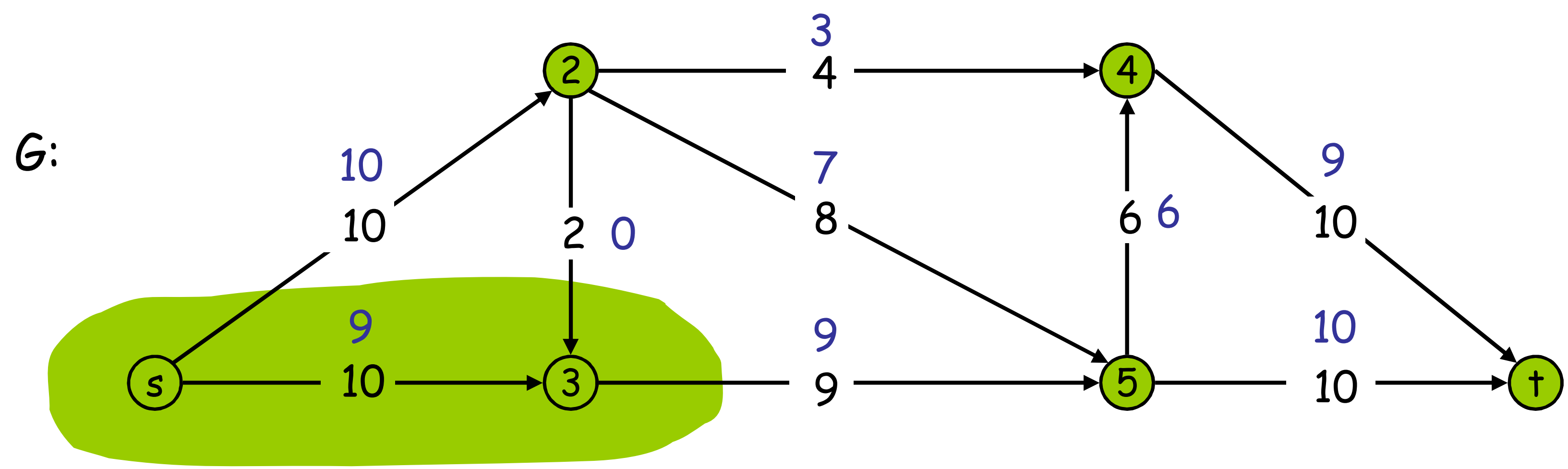
G:



Flow value = 19

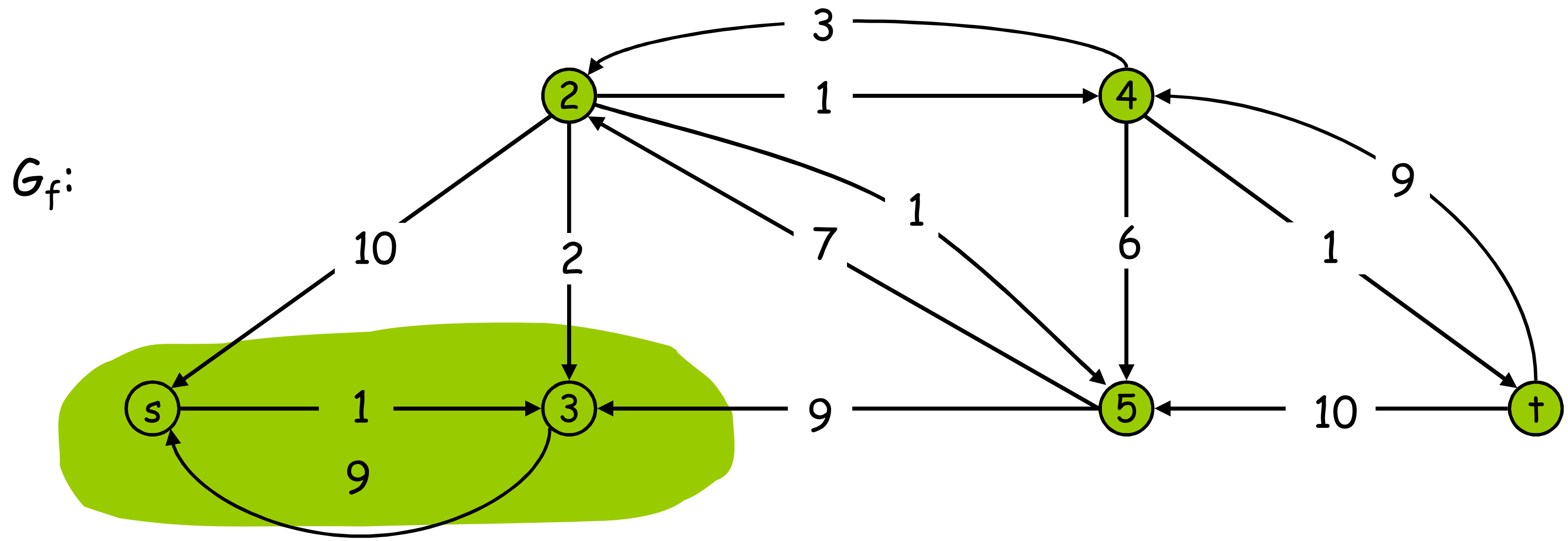
G_f:





Cut capacity = 19

Flow value = 19



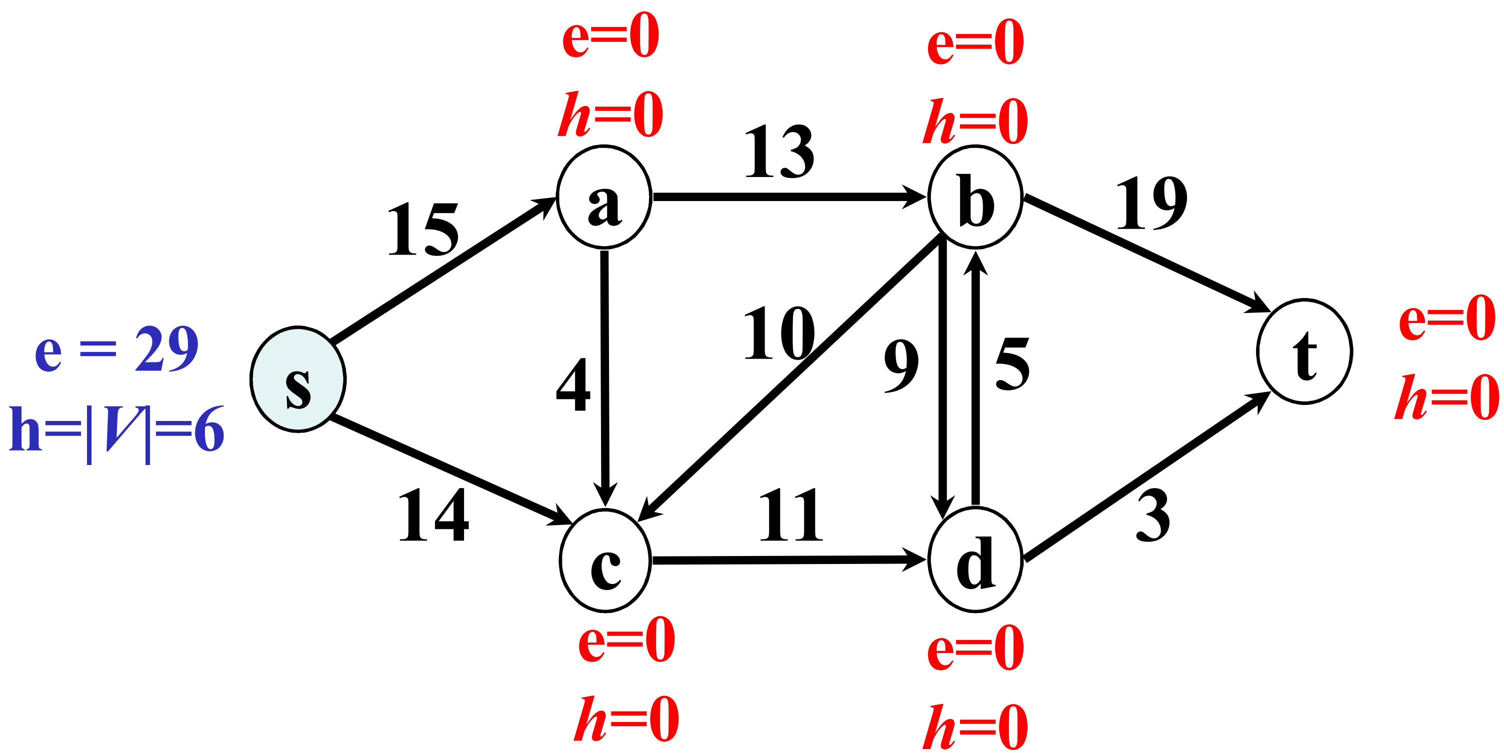
7.1.4 推送复标算法

- **Ford-Fulkson算法**
  - ✓ 构造剩余网络
  - ✓ 选取s-t增广路径来增大流值
  - ✓ 具有全局优化的观点

- 能否只考虑顶点的局部情况
  - ✓ 逐个顶点查看
  - ✓ 仅查看其邻接点
  - ✓ 确定流值的变化

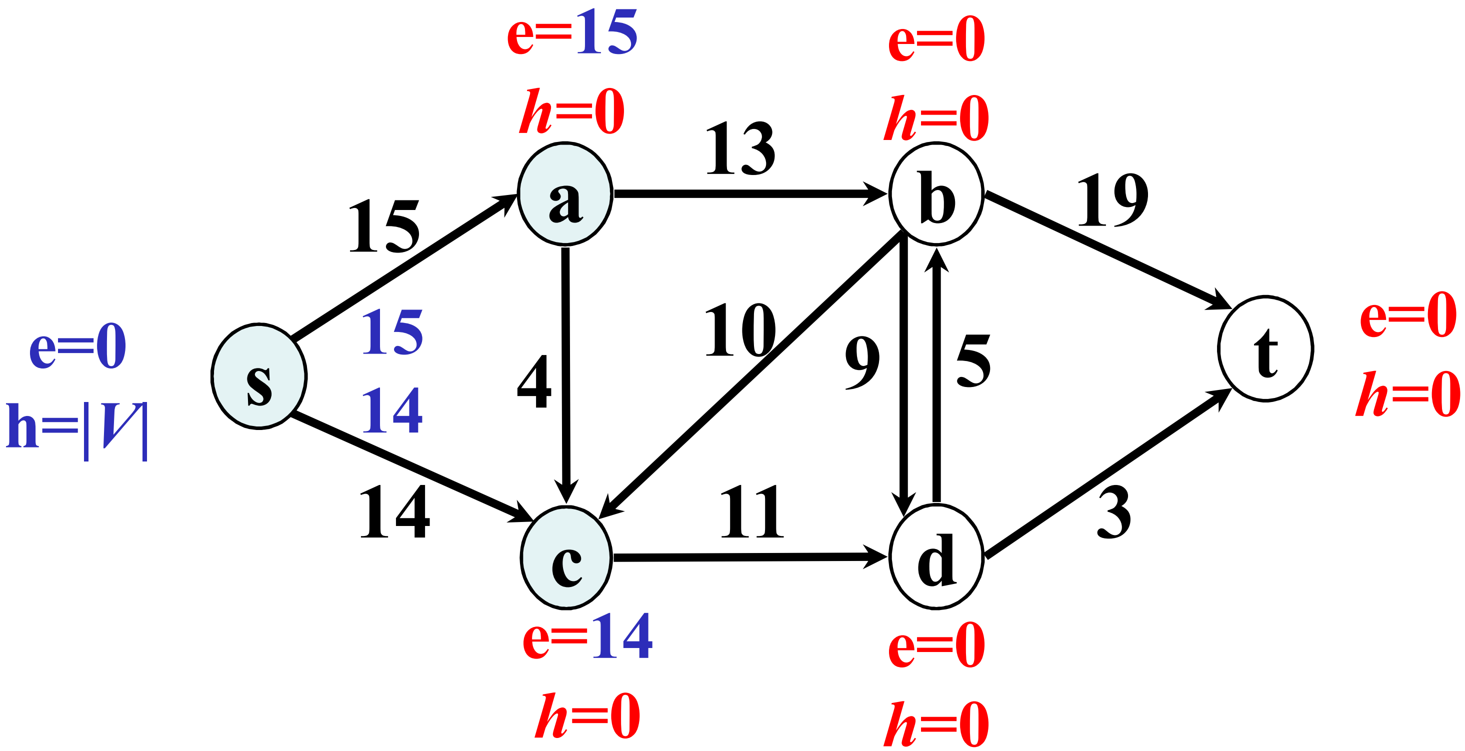
基本思想

- 将流网络看成管道网络
  - ✓ 顶点具有库存能力，$s.e$=-$\sum_{su\in E}c(uv)$， $u.e$=0
  - ✓ 顶点具有不同高度，$s.h$=|V|，$u.h$=0
  - ✓ 液体将怎么流动？

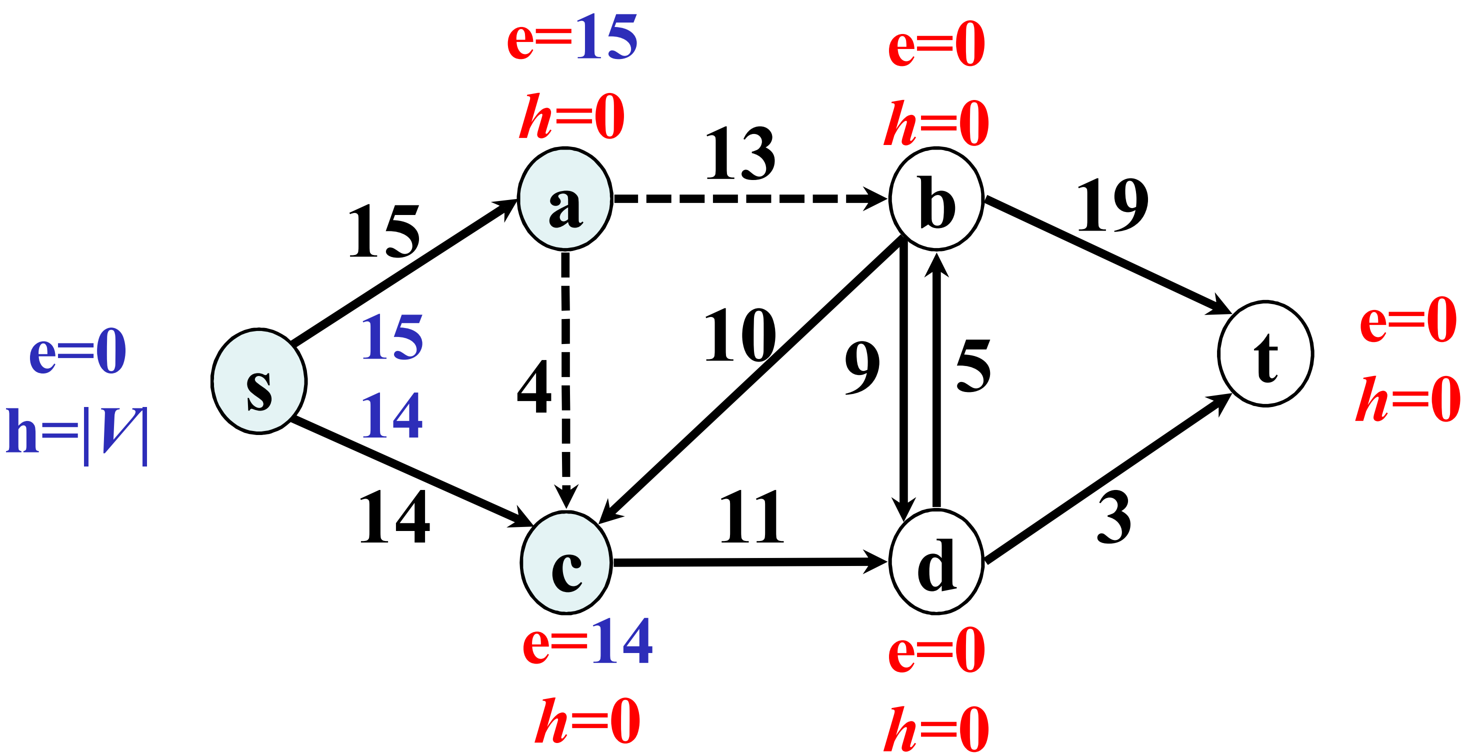


将流网络看成管道网络

- ✓顶点具有库存能力，$s.e$=-$\sum_{su\in E}c(uv)$，　$u.e$=0
- ✓顶点具有不同高度，$s.h$=$|V|$，$u.h$=0
- ✓从高度较高的顶点流向高度较低的顶点



- a有库存，但a的高度可能流动顶点高度一样
  - ✓ 修改高度(复标高度)
  - ✓ 确保流可以继续流动
  - ✓ 如何修改a的高度?

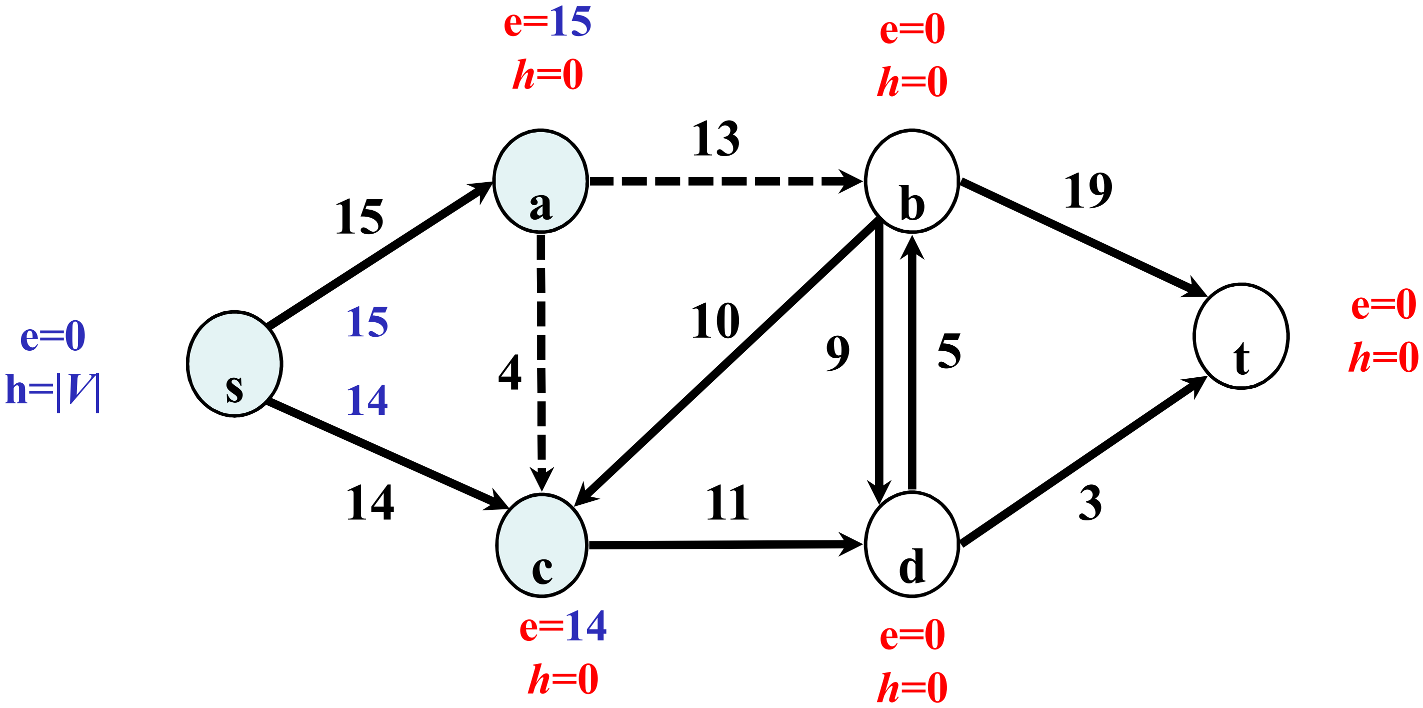


复标操作

- u有库存，但u的高度与可能流动顶点高度一样
  - ✓ $u.e>0$
  - ✓ $uv \in E_f$ $(c(uv)-uv.f>0$,剩余网络边), $u.h \le v.h$

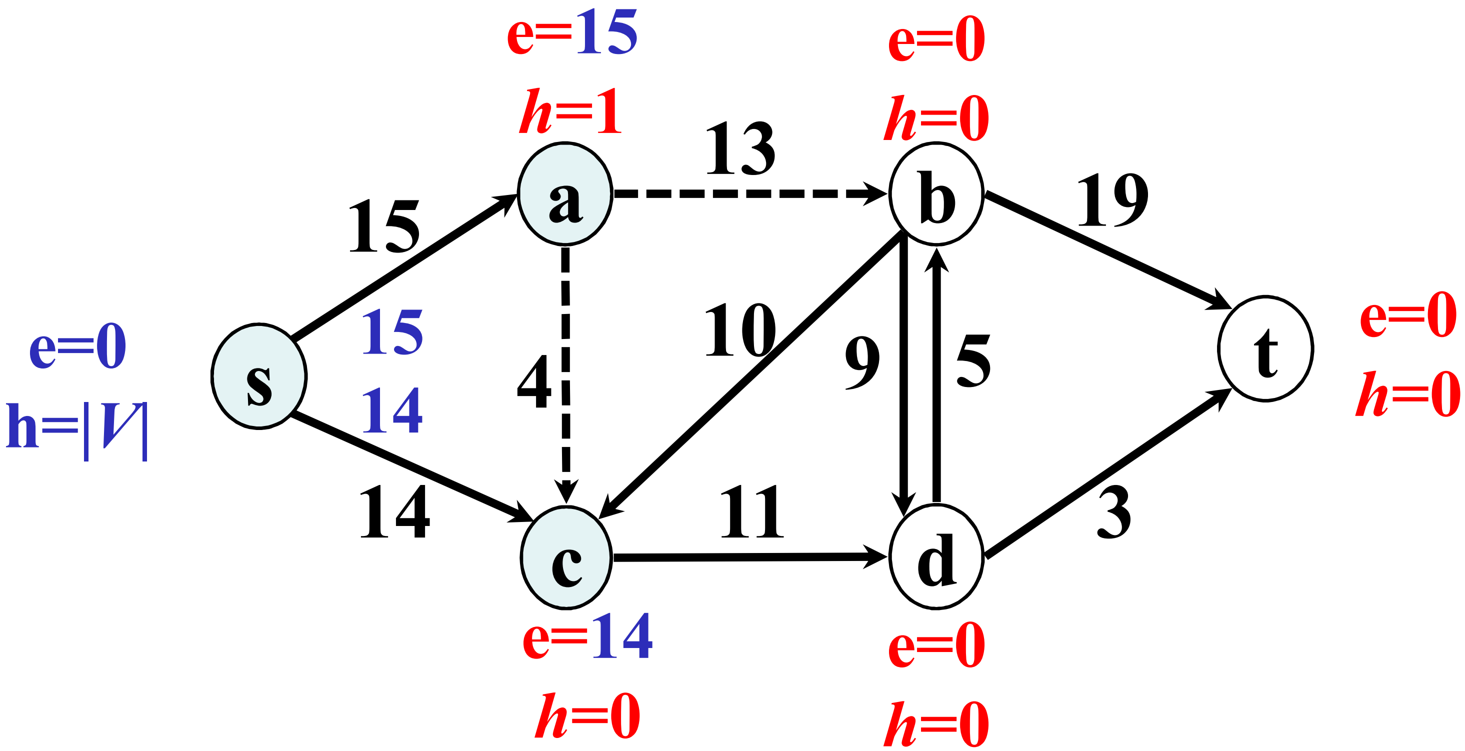
操作： $u.h = 1+\min\{v.h: uv \in E_f\}$

性质：复标操作使得顶点高度单调递增

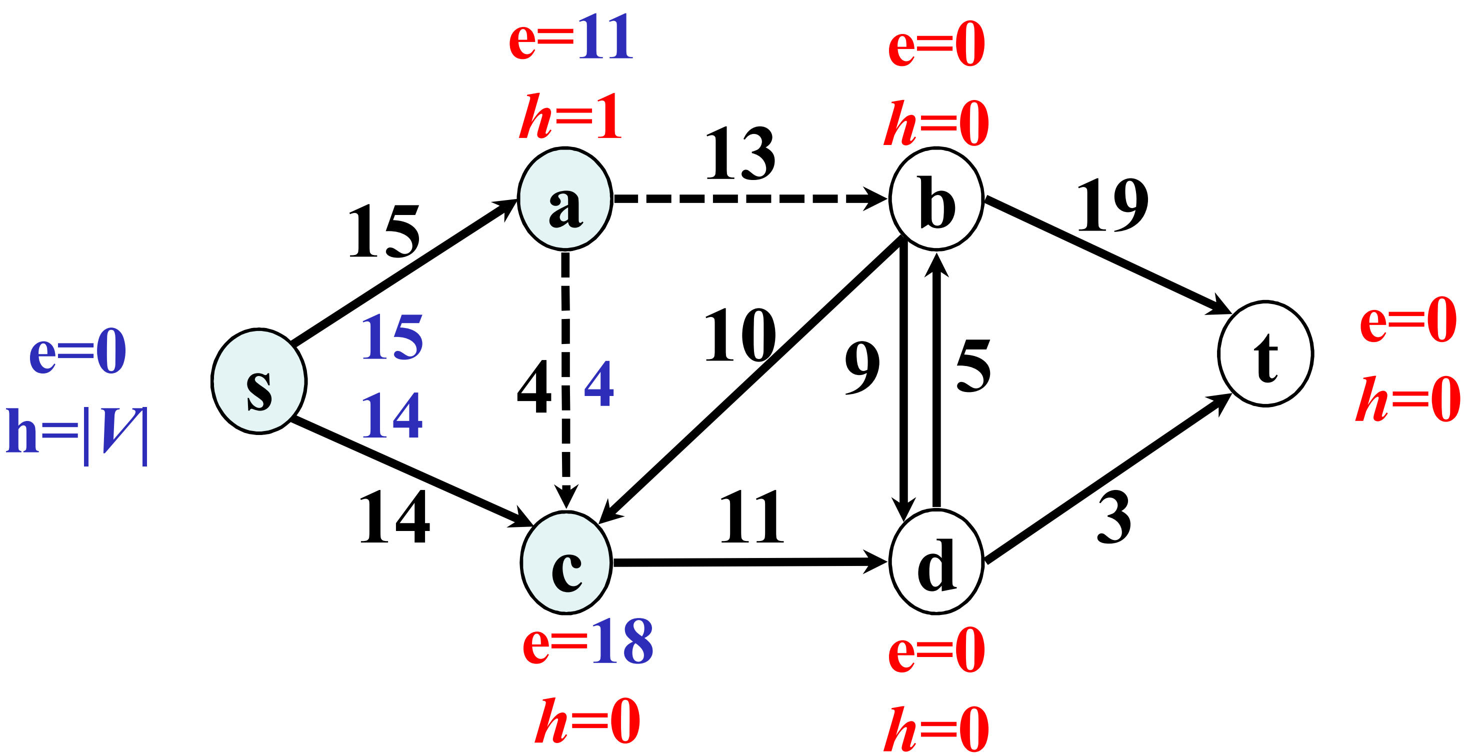


HIT
CS&E

- a有库存，但a的高度与其他顶点高度一样
  - ✓ 修改高度(复标高度)
  - ✓ 确保流可以继续流动
  - ✓ a.h = 1+ min {c.h, b.h} =1



- a有库存可以向c流动4



推送操作

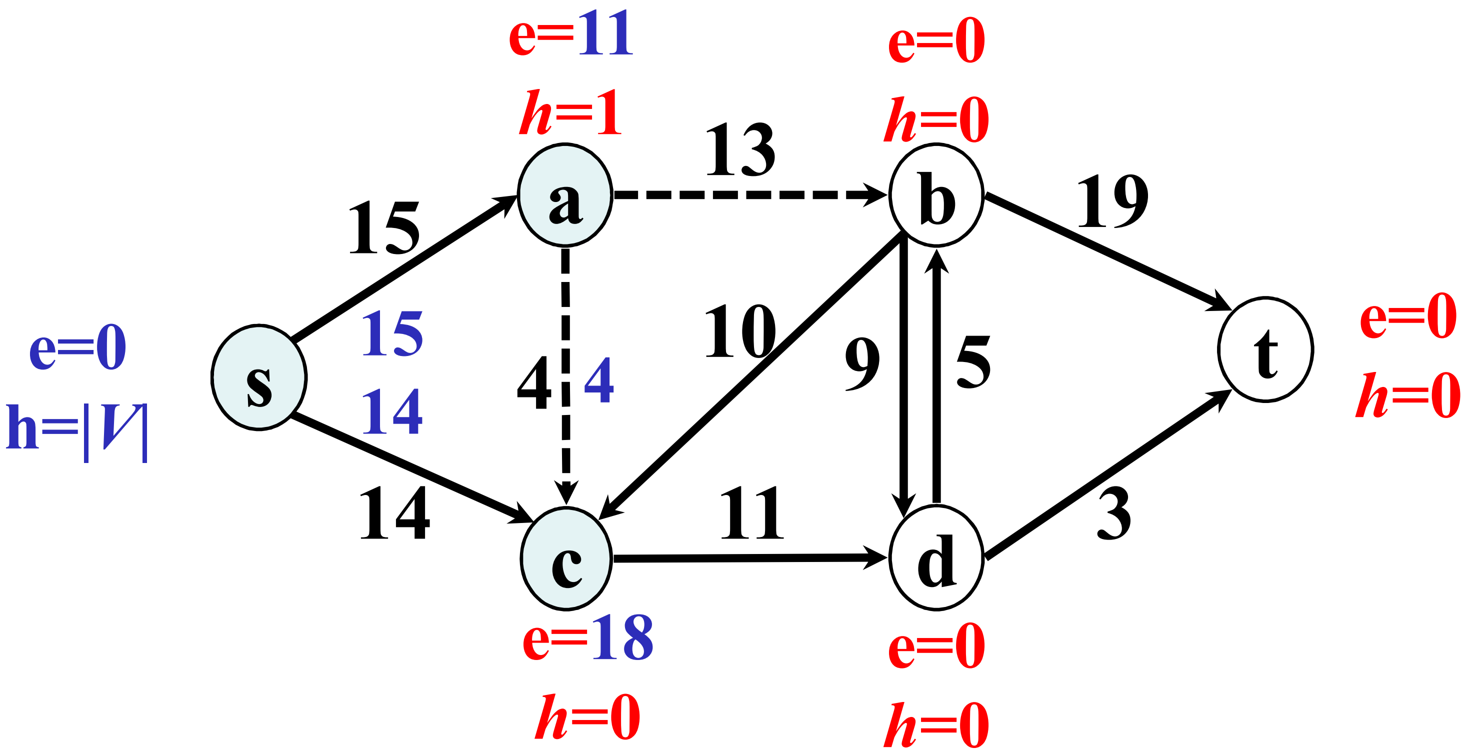
- u的库存可向其相邻顶点流动，则将库存推送至此

$u.e>0$, $u.h = v.h + 1$ 且 $c(uv)-uv.f>0$

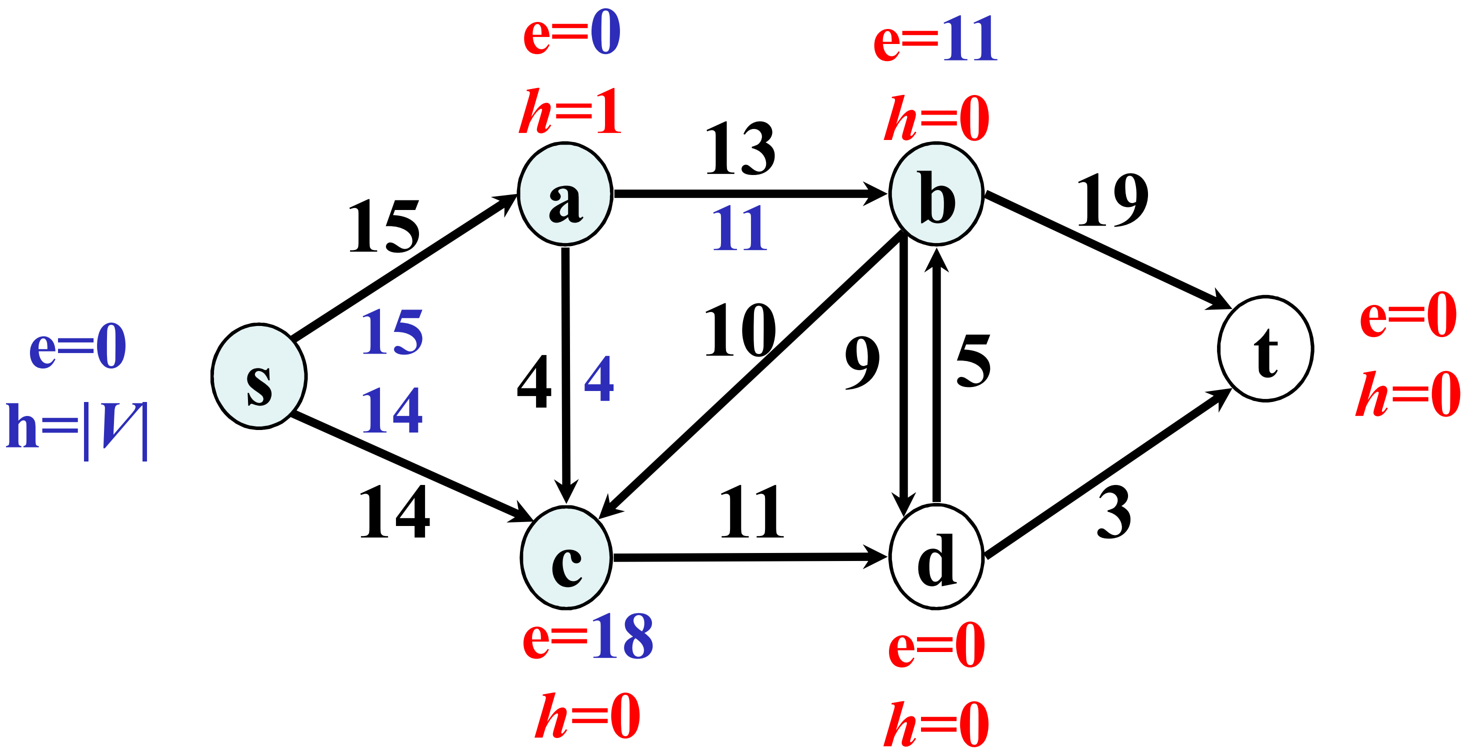
操作：1.If $uv \in E$ Then $uv.f = uv.f + \min\{u.e, c(uv)-uv.f\}$

2.Else $vu.f = vu.f - \min\{u.e, c(uv)-uv.f\}$

3.$u.e = u.e - \Delta_f$, $v.e = v.e + \Delta_f$

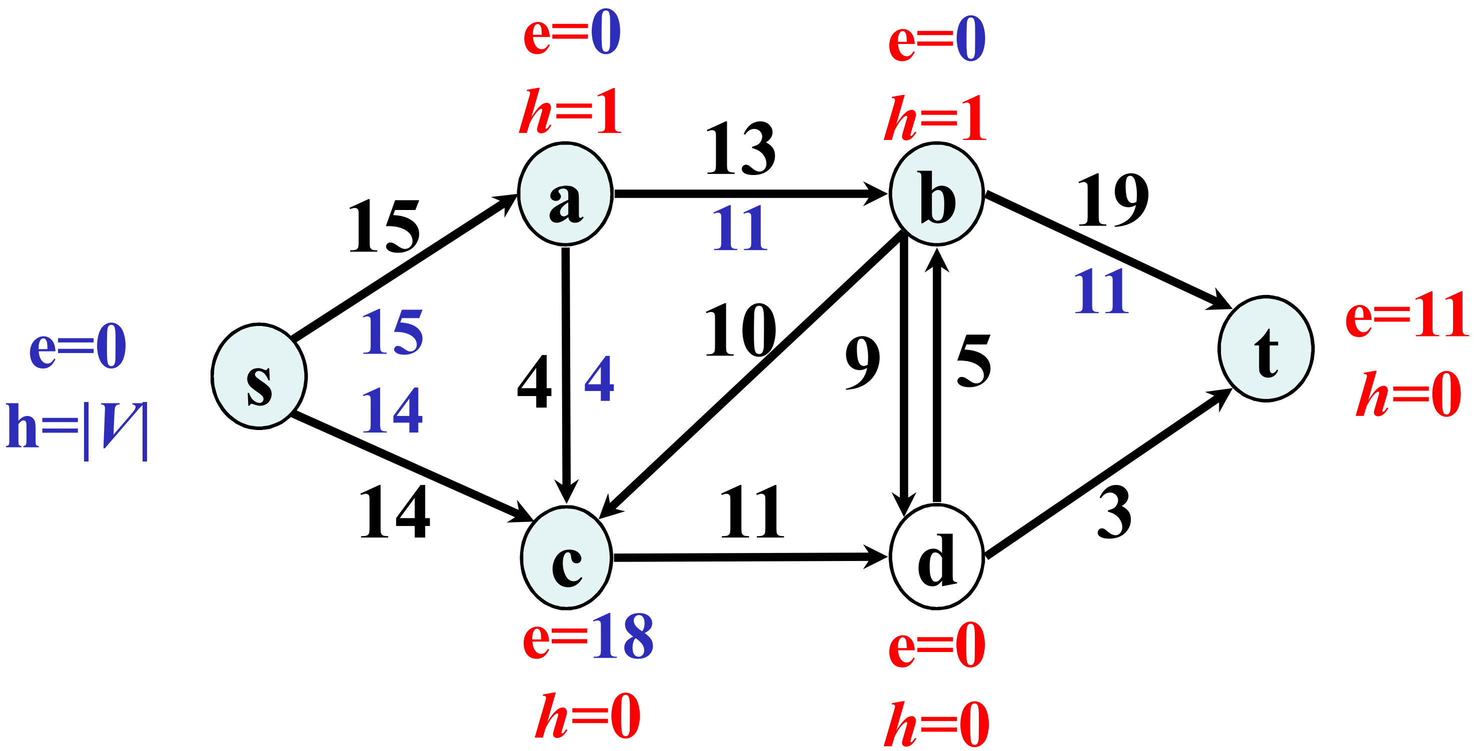


- **Push(a,b)还可以将11推送到b**
- 推送操作完成后，如果**c(uv)-*f*(uv)=0**,则称饱和推送
- 性质：非饱和推送完成后，***u*.*e*=0**



Relabel (b)

Push(b,t)还可以将11推送到t

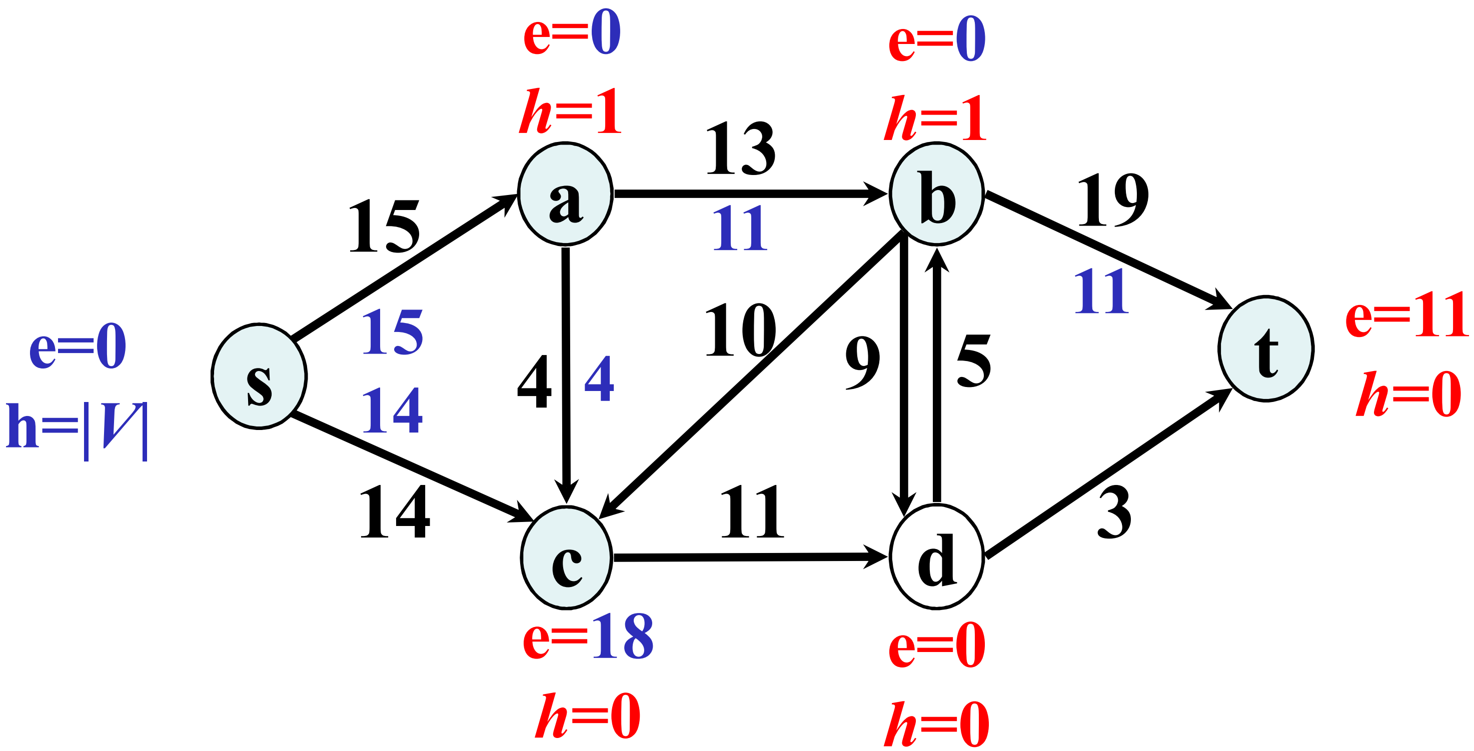


推送复标算法

While (推送或复标操作可行)

执行推送或复标操作

/*算法停止吗，怎么办？ */



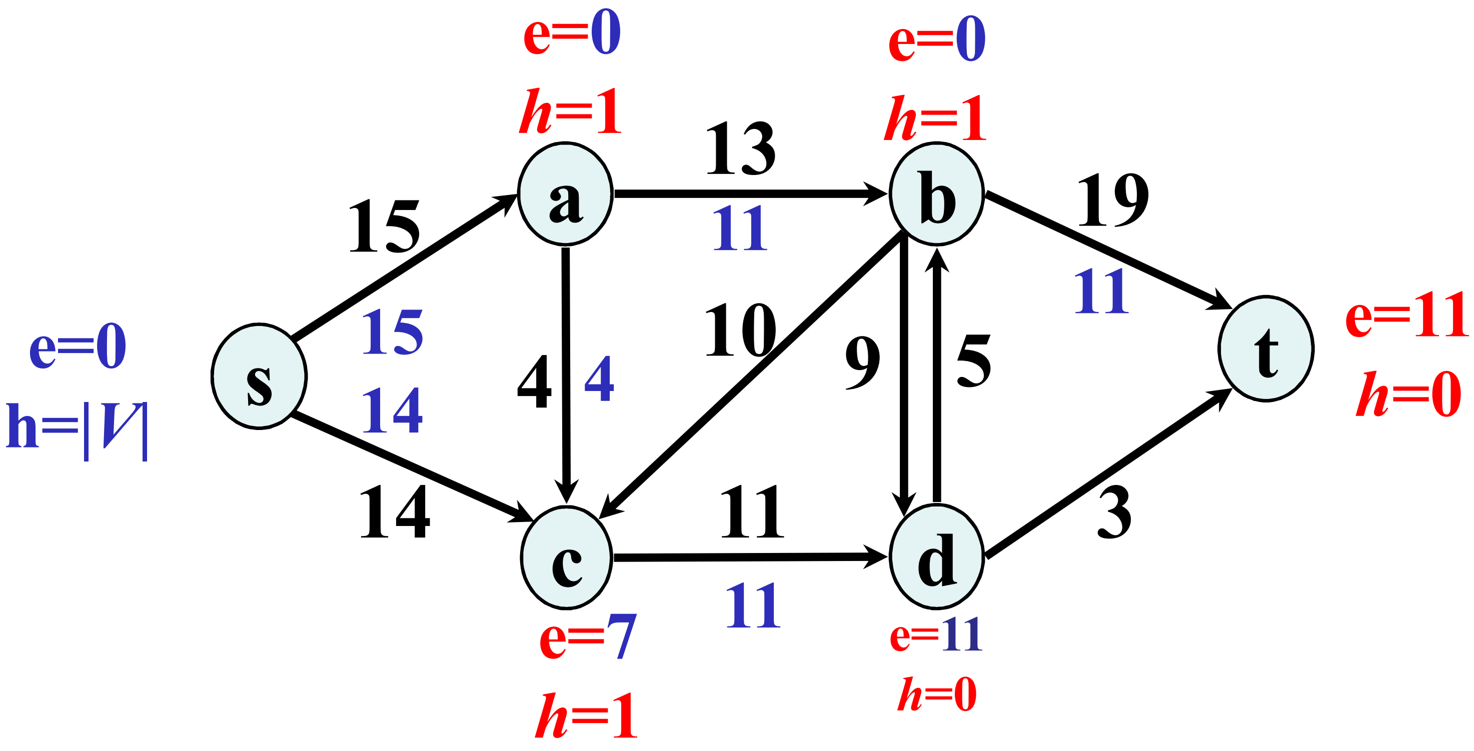
推送复标算法

预流

(1) 除了在顶点可能不满足守恒约束之外

(2) 满足流的其他所有性质

(3) 推送操作将一个预流变成另一个预流



高度

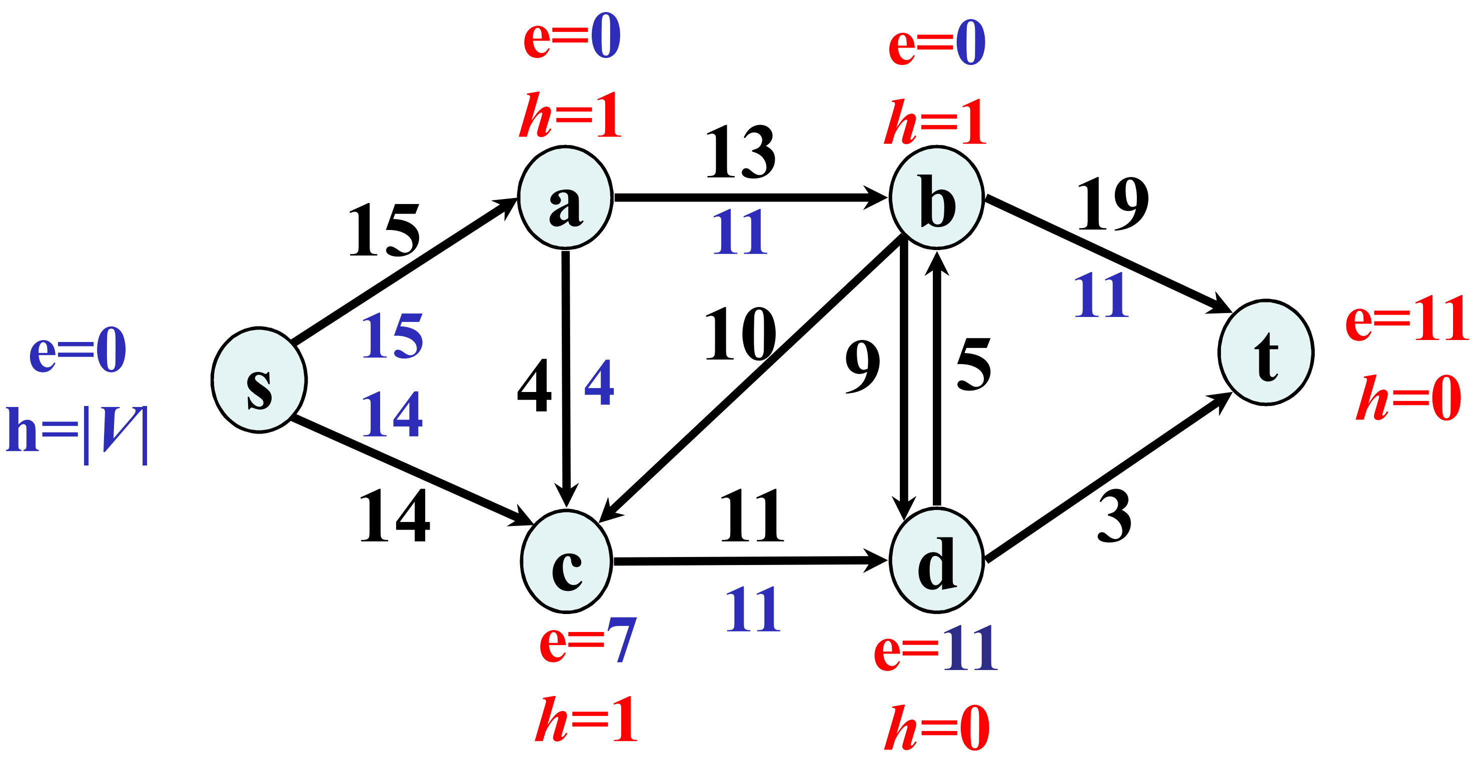
(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$　　$t.h=0$保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

复标操作：维持高度满足上述三条性质，且高度递增

性质：如果$u.h>v.h+1$,则uv不是剩余边



推送复标算法

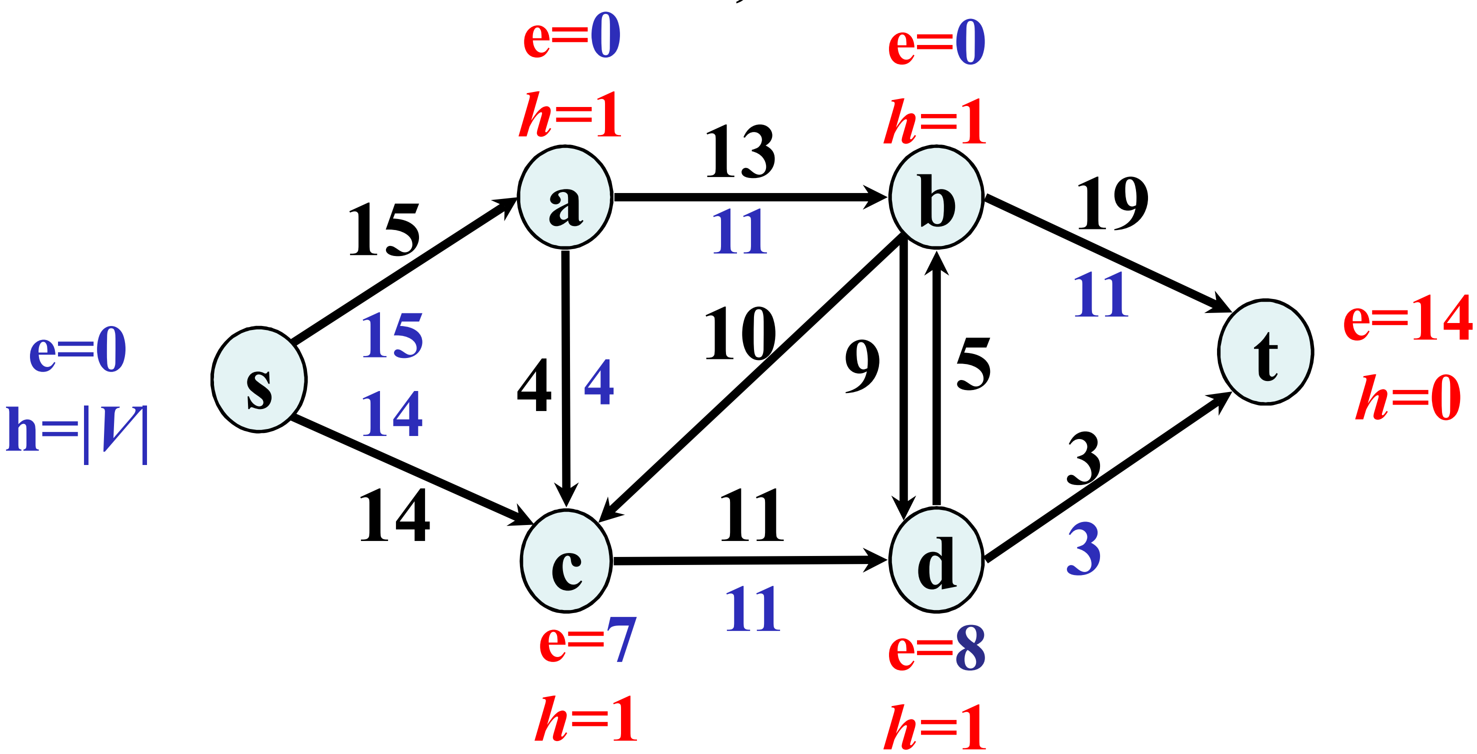
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h=|V|$ $t.h=0$保持不变

(3)若$uv\in E_f$, 则 $u.h\leq v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



HIT
CS&E

推送复标算法

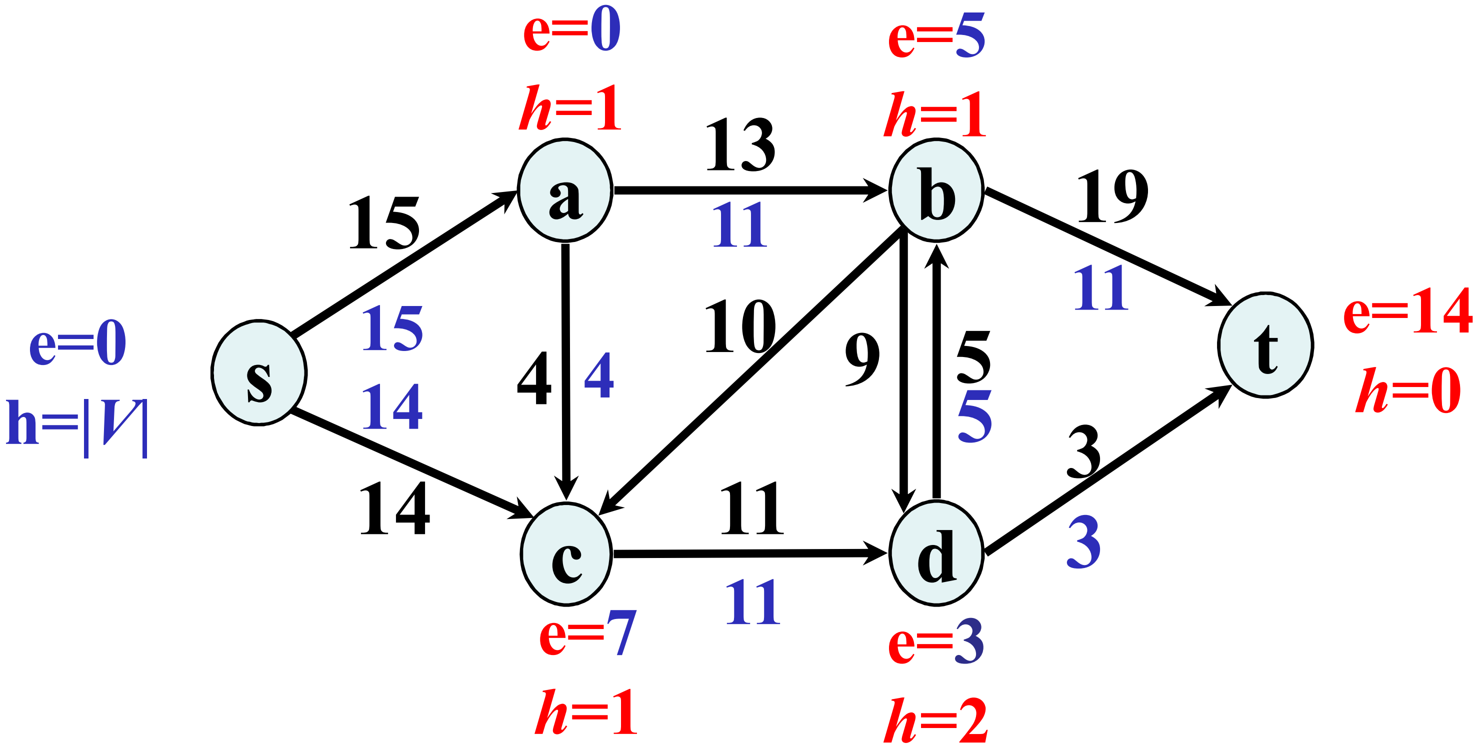
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$　　$t.h=0$保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



HIT
CS&E

推送复标算法

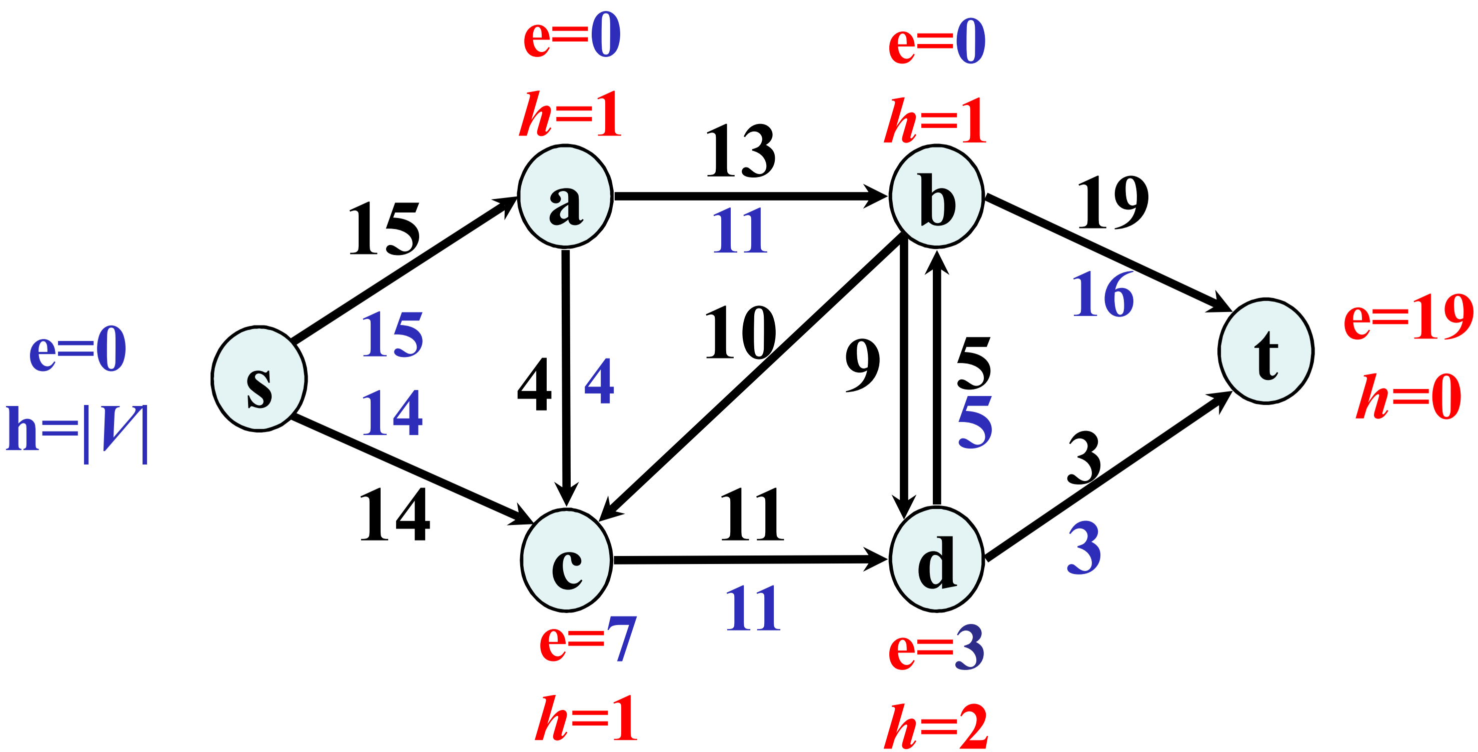
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h=|V|$　　$t.h=0$保持不变

(3)若$uv\in E_f$, 则 $u.h\leq v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



推送复标算法

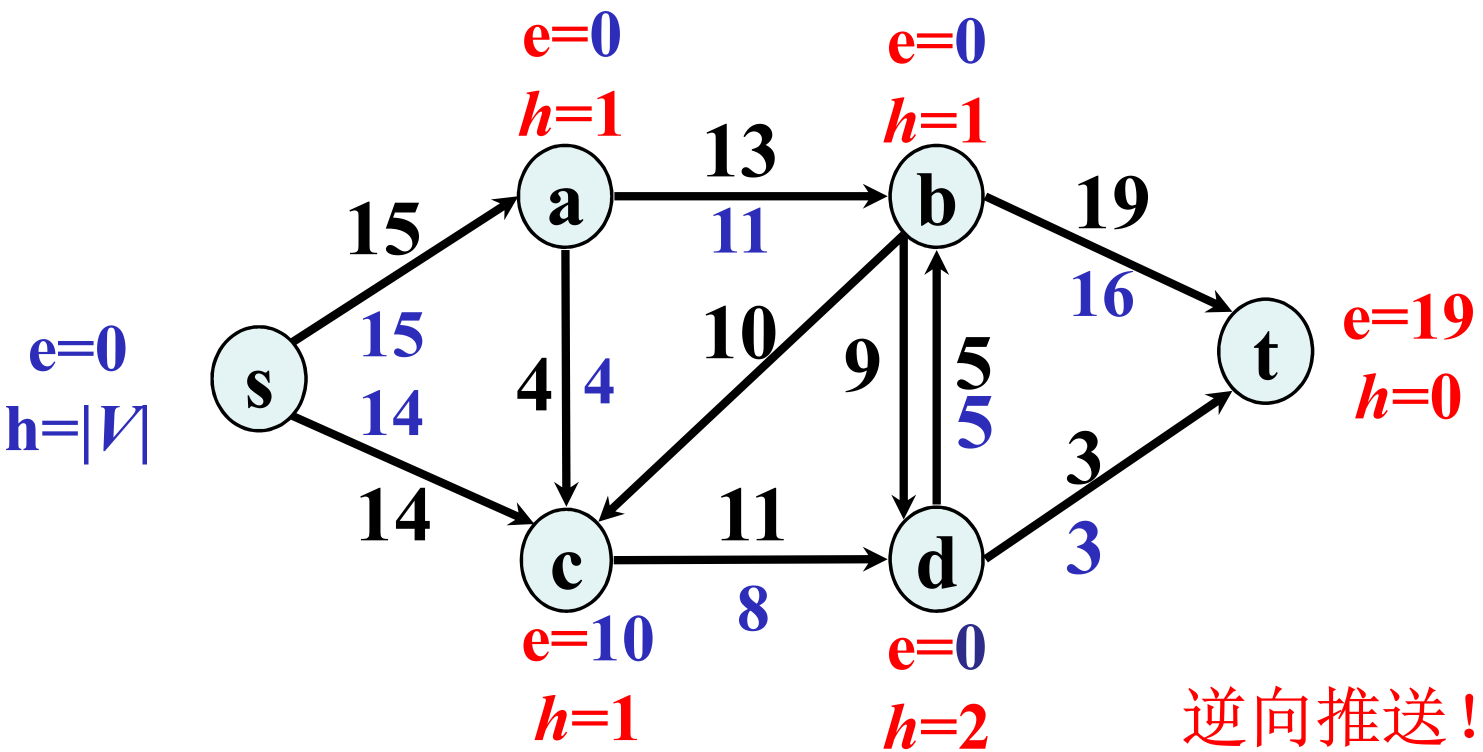
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$　　$t.h=0$保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



推送复标算法

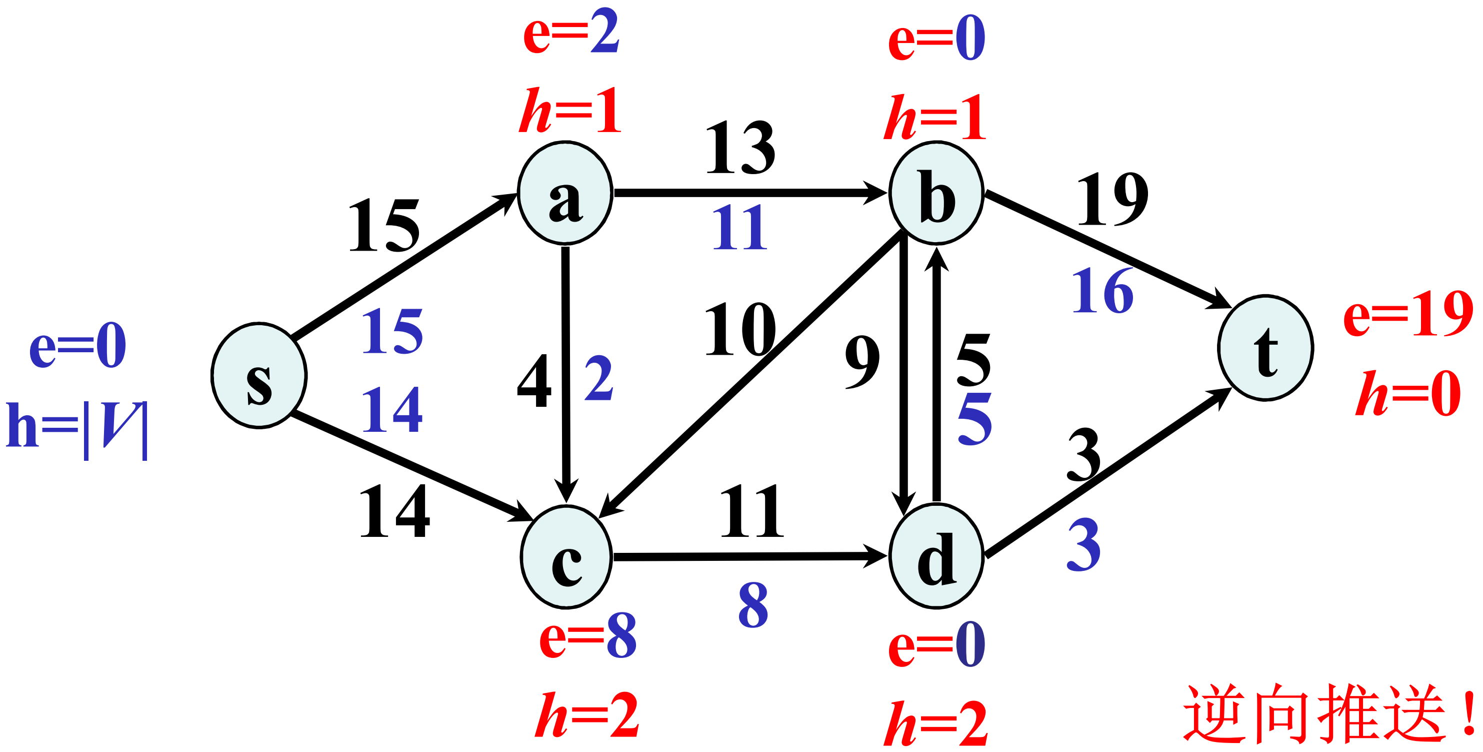
性质：剩余网络中不存在从s到t的路径

事实：(1)若$uv\in E_f$, 则 $u.h\le v.h+1$ (2)$s.h=|V|$, $t.h=0$

不然，$s=v_0,v_1,\ldots,v_k=t$是剩余网络中简单路径 ($k<|V|-1$)

$s.h\le v_1.h+1\le v_2.h+2\le\ldots\le v_k.h+k$

$|V|\le k$ 矛盾



逆向推送！

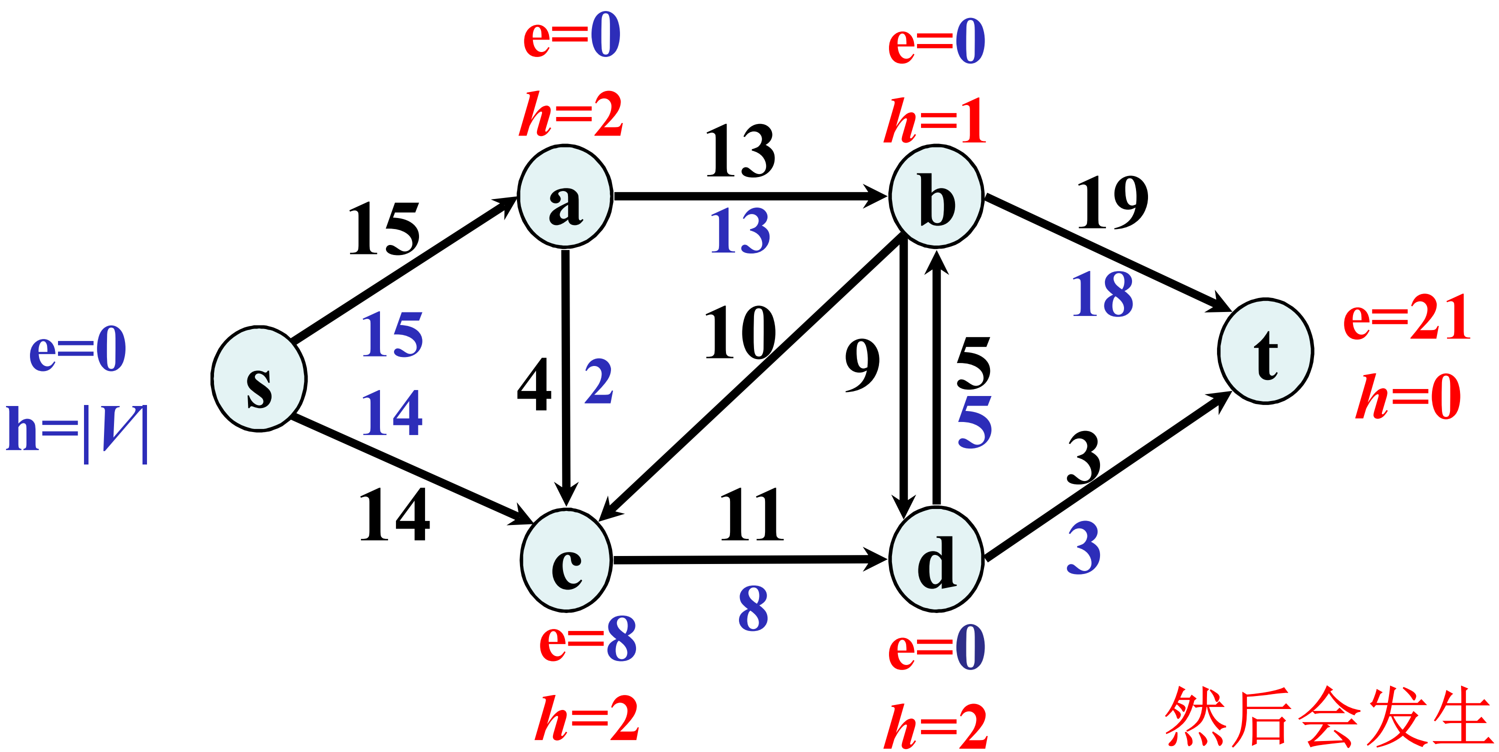
HIT
CS&E

u.e>0

推送操作沿*uv*(某个*v*)可行 **OR** 复标操作在*u*上可行

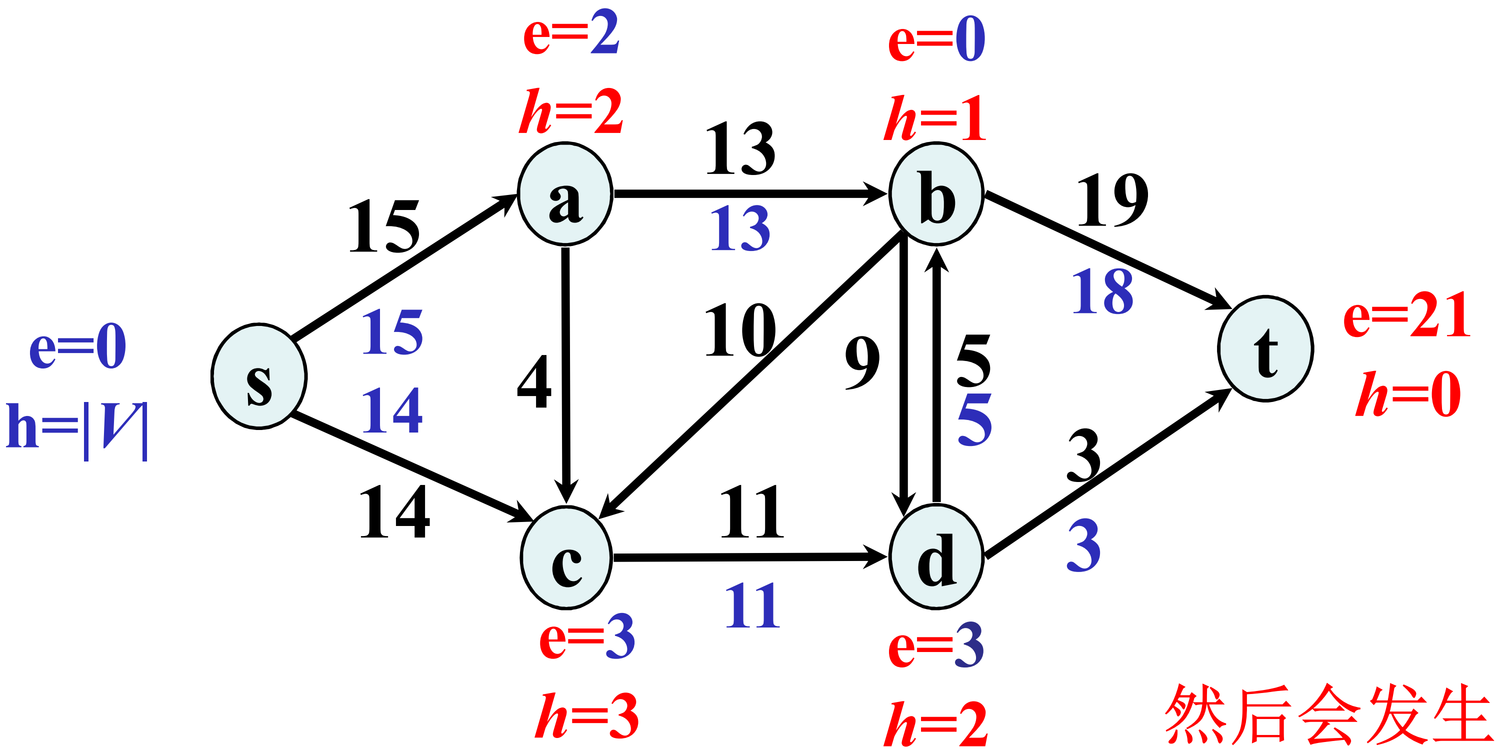
存在$uv\in E_f$使$u.h=v.h+1$　　　　$\forall uv\in E_f$满足$u.h<v.h+1$

（$uv\in E_f \Rightarrow$ u.h≤v.h+1）



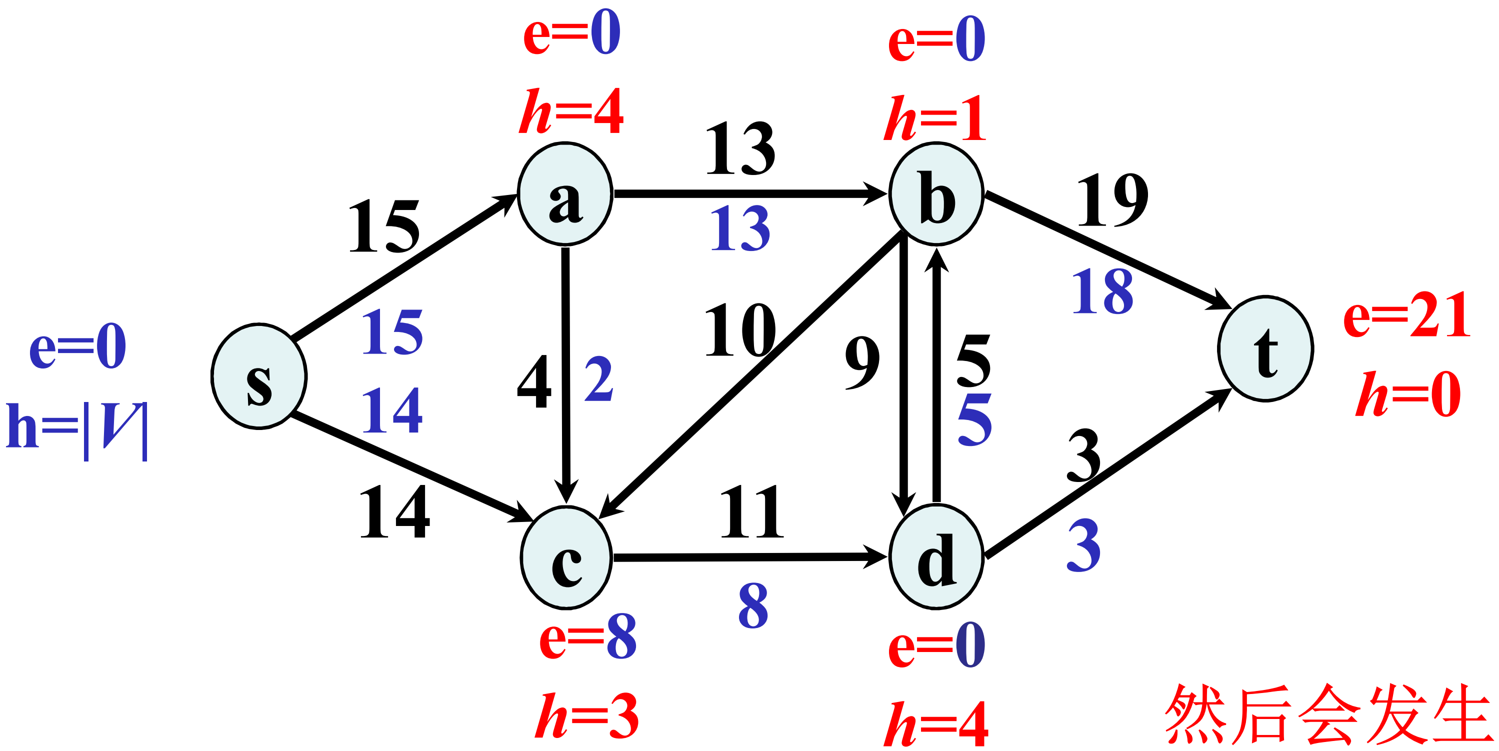
然后会发生什么！

推送复标算法



然后会发生什么！

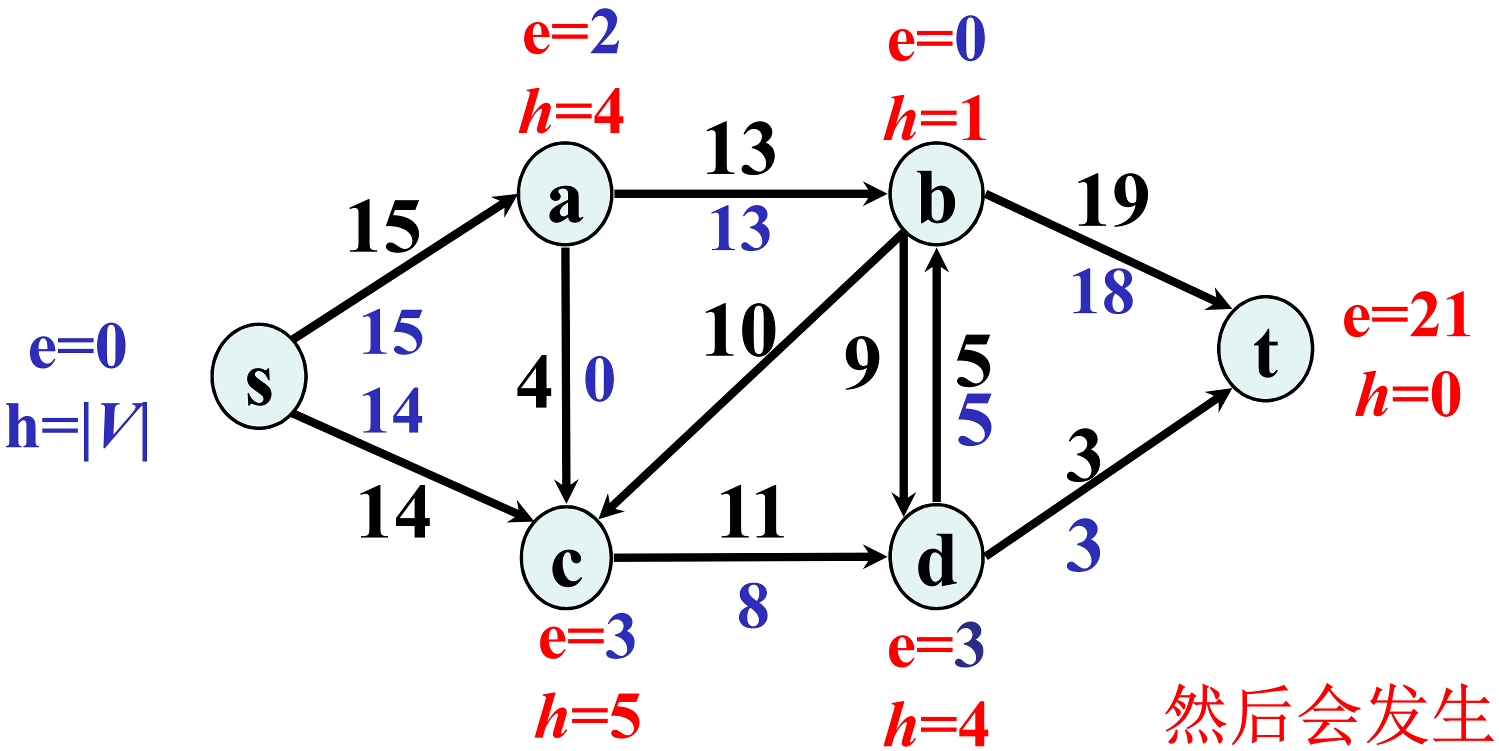
推送复标算法



然后会发生什么！

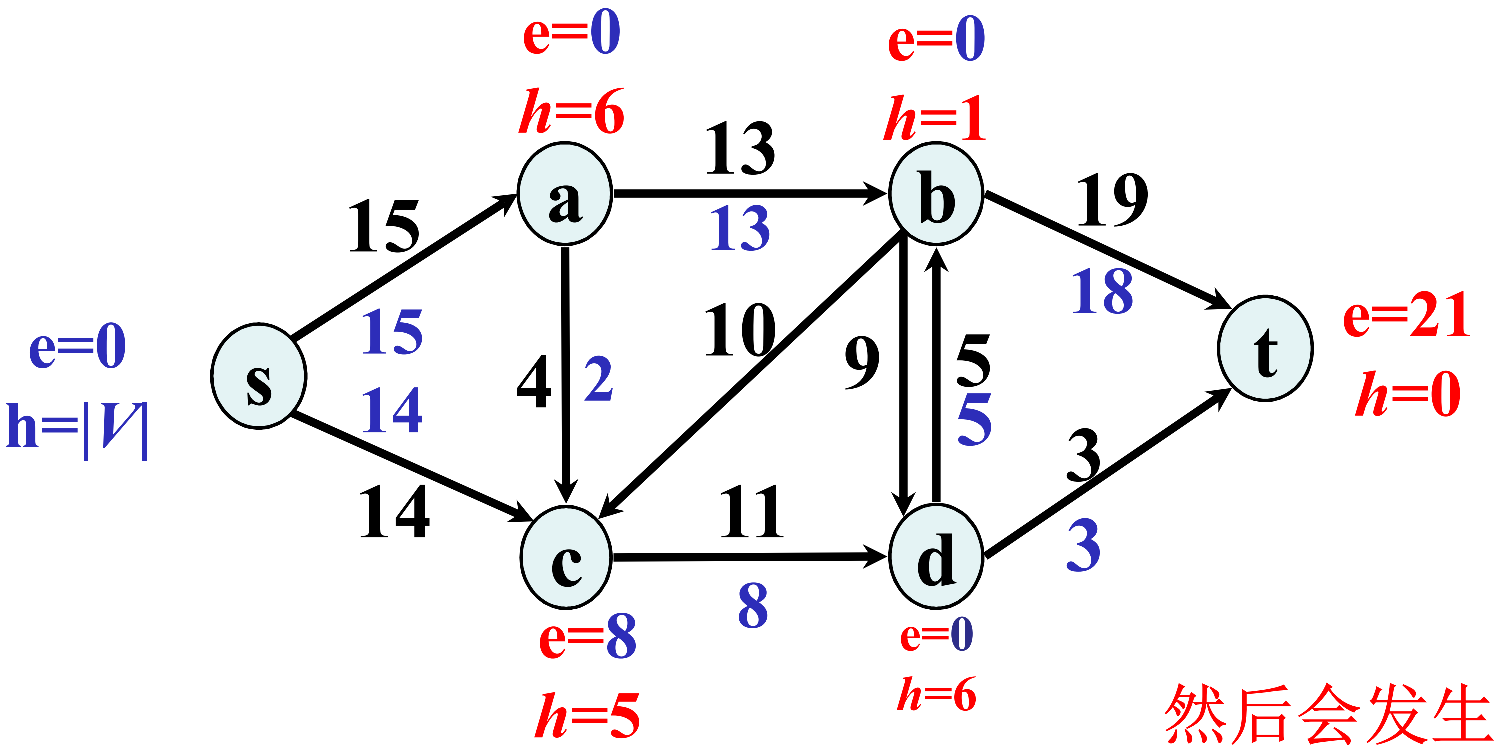
HIT
CS&E

推送复标算法



然后会发生什么！

推送复标算法

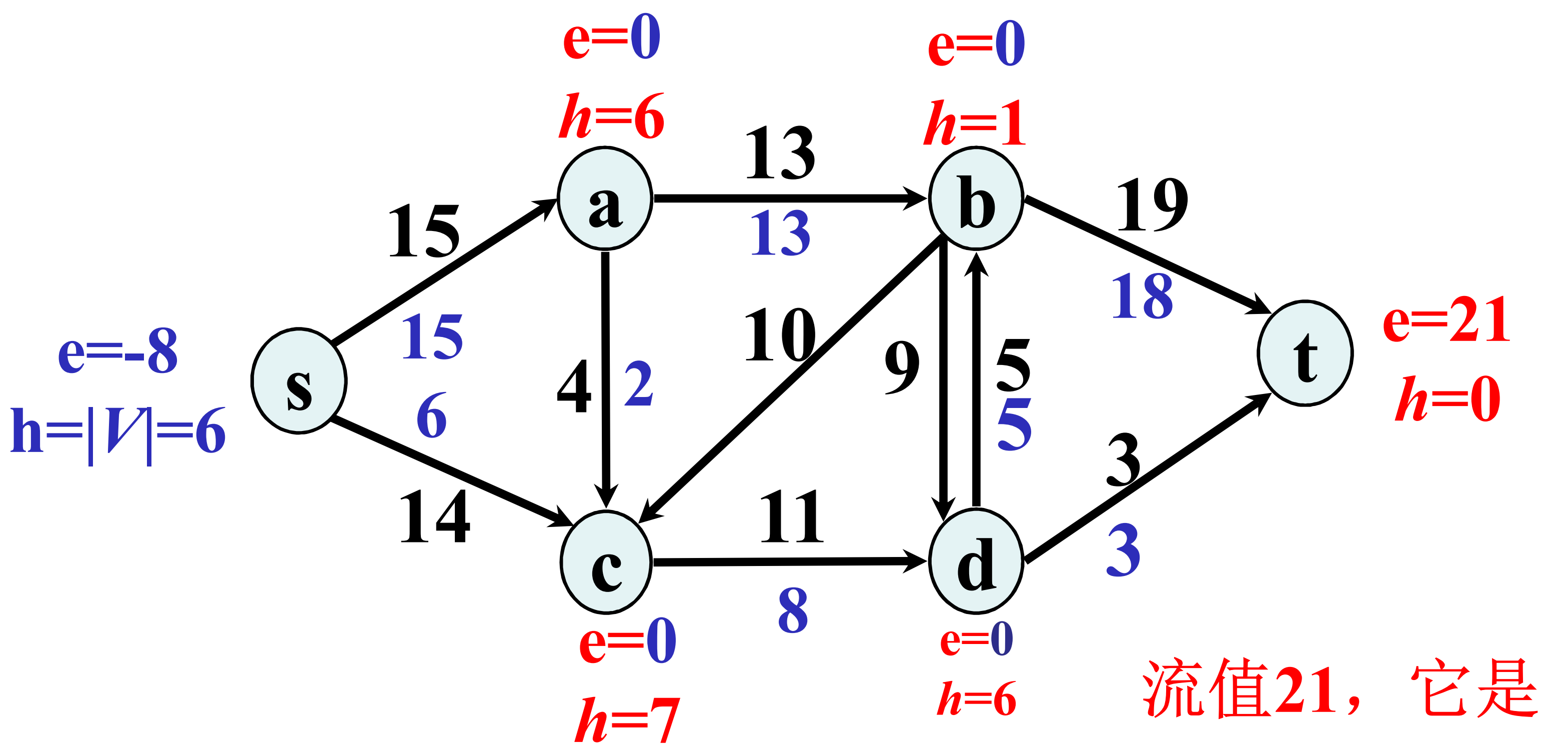


然后会发生什么！

得到了一个流

所有顶点上推送复标操作都不可行

e.u=0　对$\forall u \in V-\{s,t\}$



流值21，它是最大流吗？

推送复标算法

初始化： **u.e=0; u.h=0; uv.f=0** $\forall u \in V, uv \in E$

$s.e = -\sum_{sv \in E} c(sv)$， $s.h = |V|$

$sv.f = c(sv)$, $v.e = c(sv)$ $\forall sv \in E$

While (推送或复标操作可行)

执行推送或复标操作

算法正确性

定理.如果推送算法终止，则最后的预流是最大流。

初始化：算法初始化是预流

循环　：每次推送操作或复标操作后，仍是预流

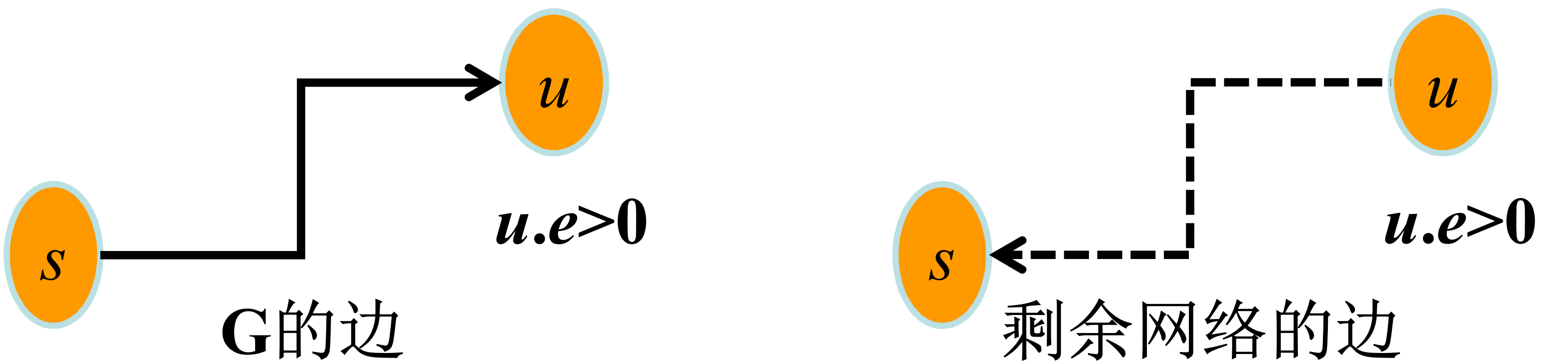
终止　：算法结束后得到一个预流f

此时$u.e$=0对任意u≠s,t成立，　故f是流

剩余网络中不存在s-t路径，　由最大流最小割定理，f是最大流

引理5. 给定s-t流网络$G=(V,E)$, 对于任意预流f及其上的高度函数h。如果$u.e>0$，则在剩余网络中存在**u-s**路径。

顶点**u**的库存是哪儿来的？

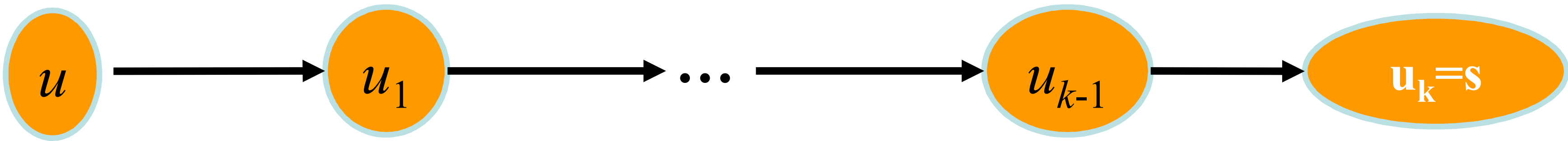


HIT
CS&E

引理6.给定s-t流网络$G=(V,E)$, 则对于任意顶点u，在推送复标算法运行过程的任意时刻，$h(u)\leq 2|V|-1$.

若$u=s$, 则$h(s)=|V|$不变，结论成立
若$u=t$, 则$h(t)=0$不变，结论成立

若$u\neq s,t$,初始时，$h(u)=0$
对u的每次复标操作时(u.e>0)，由引理5，剩余网络中存在u-s路径$u=u_0,u_1,\ldots,u_k=s$



$h(u)\leq h(u_1)+1$　　$h(u_1)\leq h(u_2)+1$　…　$h(u_{k-1})\leq h(u_k)+1$　　$h(u_k)=|V|$

$h(u)\leq h(s)+k\leq 2|V|-1$,　　　　$k\leq |V|-1$

引理7.给定s-t流网络$G=(V,E)$, 则推送复标算法在G上至多执行$(2|V|-1)(|V|-2) \leq 2|V|^2$ 次复标操作。

在s,t上永远不执行复标操作

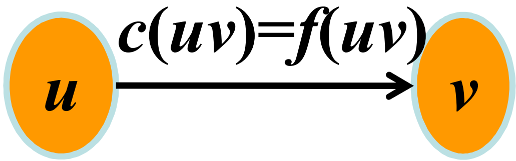
对于其余每个顶点u（共|V|-2个顶点）
$h(u)$的初始值为0
每次复标使得$h(u)$增大，且引理6表明$h(u) \leq 2|V|-1$
故在u上至多执行2|V|-1次复标操作

引理8. 给定*s*-*t*流网络$G=(V,E)$，则推送复标算法在G上至多执行$2|V||E|$ 次饱和推送操作。

$\forall u,v\in V$，考察u和v之间饱和推送的总次数

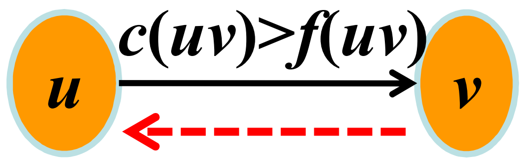
- 从u到v的饱和推送
- 从v到u的饱和推送
- $uv\in E$或$vu\in E$　　　　顶点对的个数$\leq|E|$

饱和推送



$h(u)-1=h(v)$

下一次同向推送前



$h(u)+1=h(v)$

$h(u)$至少增大2

$0\leq h(u)\leq 2|V|-1$

- 从u到v的饱和推送至多$|V|$次
- 同理，从v到u的饱和推送至多$|V|$次

引理9. 给定*s*-*t*流网络$G=(V,E)$, 则推送复标算法在G上至多执行$4|V|^2(|V|+|E|)$次非饱和推送操作。

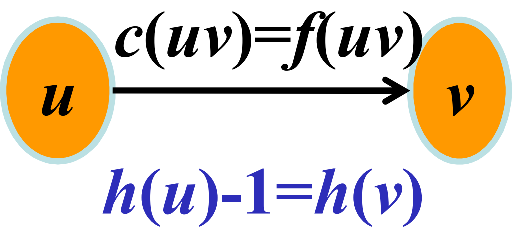
定义$\varphi=\sum_{u.e>0} h(u)$　初始时，$\varphi=0$且$\varphi\geq 0$恒成立

复标顶点u使得其高度增加，导致φ增大

顶点u上的复标操作，导致φ增大总量$\leq 2|V|-1$

所有复标操作导致φ的总增量$\leq 2|V|^2 * 2|V|$

饱和推送



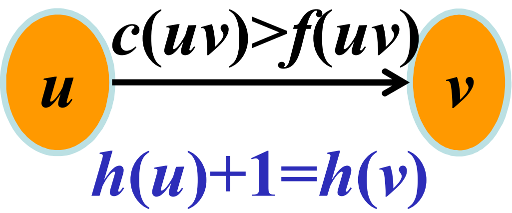
推送前：$u.e>0$,　　$v.e=0$或$v.e>0$

推送后：$u.e=0$或$u.e>0$　$v.e>0$

每次饱和推送导致φ的增量$\leq 2|V|$

所有饱和推送导致φ的增量$\leq 2|V|.2|V||E|$

非饱和推送



推送前：$u.e>0$,　　$v.e=0$或$v.e>0$

推送后：$u.e=0$　　$v.e>0$

每次非饱和推送导致φ至少减小1

由于$\varphi\geq 0$恒成立，总增量≥总减量

定理10. 给定s-t流网络$G=(V,E)$, 则推送复标算法在G上至多执行$O(|V|^2|E|)$ 次基本操作后终止。

由引理7，算法至多执行$2|V|^2$次复标操作

由引理8，算法至多执行$2|V||E|$次饱和推送操作

由引理9，算法至多执行$4|V|^2(|V|+|E|)$次非饱和推送操作

7.1.5前置复标算法

推送复标算法

- 以不确定的顺序选择推送、复标顶点
- 对每个顶点的处理不彻底
  - ✓ 对u的推送或复标操作后，**u.e>0**
  - ✓ 可能转而处理其他顶点
- 时间复杂性$\mathbf{O}(V^2E)$

提高算法性能的着手点

- 如果精细选择推送、复标操作顺序
- 对每个顶点进行彻底处理，处理后**u.e=0**

DisCharge操作

彻底处理顶点u

- $u.e>0$, 则要么推送操作可行，要么复标操作可行
- 重复在u顶点处进行推送或复标操作，直到$u.e=0$
- 仅需考察u的相邻顶点
- 对u维护邻接链表$L(u)$　　$v \in L(u) \Leftrightarrow uv \in E$或$vu \in E$

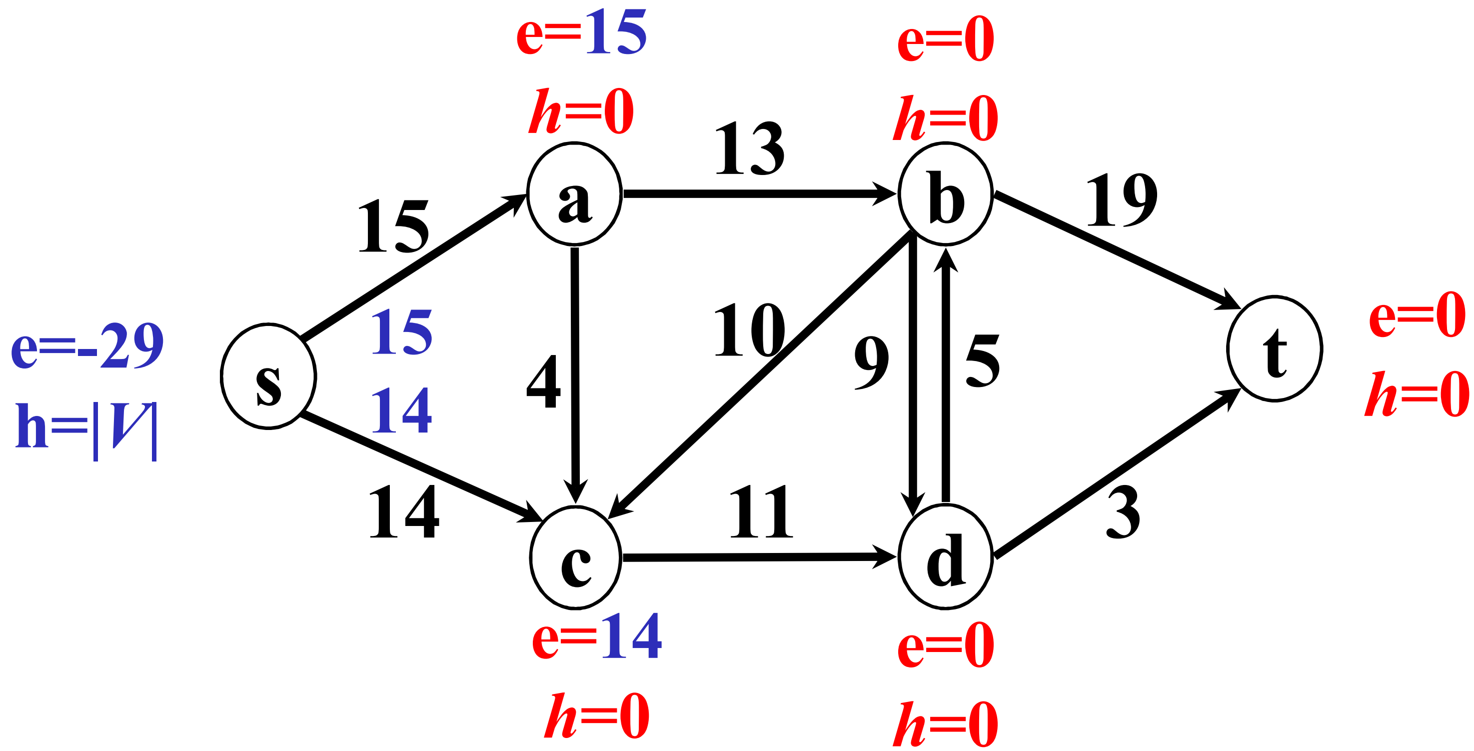
DisCharge(*u*)
　While $u.e>0$
　　v ← L(u).current　　　/*考察当前处理的顶点*/
　　If v = Null Then ReLabel(*u*), L(u).current=L(u).head
　　ElseIf c(uv)-*f*(uv)>0 且 u.h=v.h+1 Then Push(u)
　　Else L(u).current ← L(u).next

考虑初始化操作之后DisCharge(c)的执行过程

L(c):	s	a	b	d	Null

v=s≠Null，　　不执行复标操作

c(cs)-f(cs) = 0-(-14) =14 但c.h≠s.h+1 不执行Push



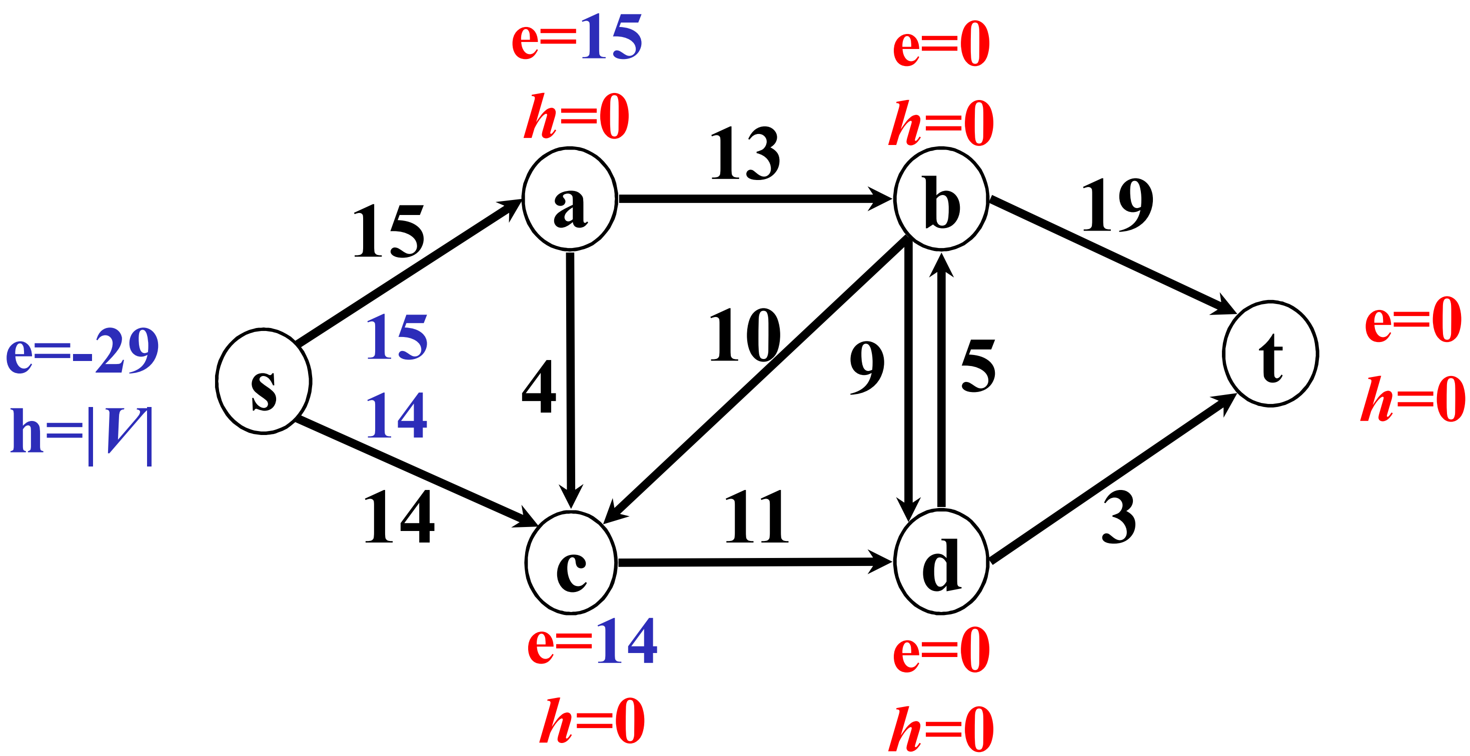
考虑初始化操作之后DisCharge(c)的执行过程

L(c):

s	a	b	d	Null

v=a≠Null， 不执行复标操作

c(ca)-f(ca) = 0-0 =0 不执行推送操作

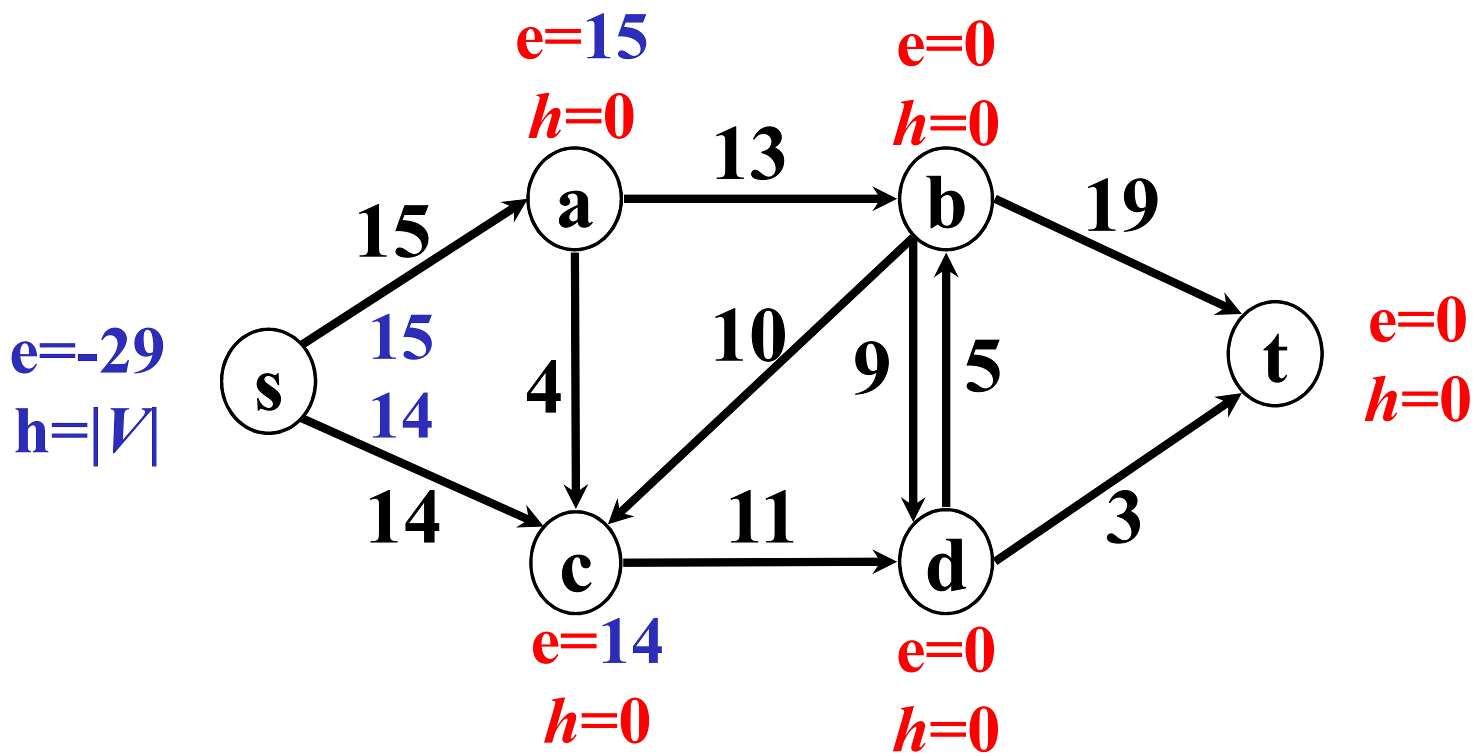


考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):	s	a	b	d	Null

v=b≠Null， 不执行复标操作

c(cb)-f(cb) = 0-0 =0 不执行推送操作

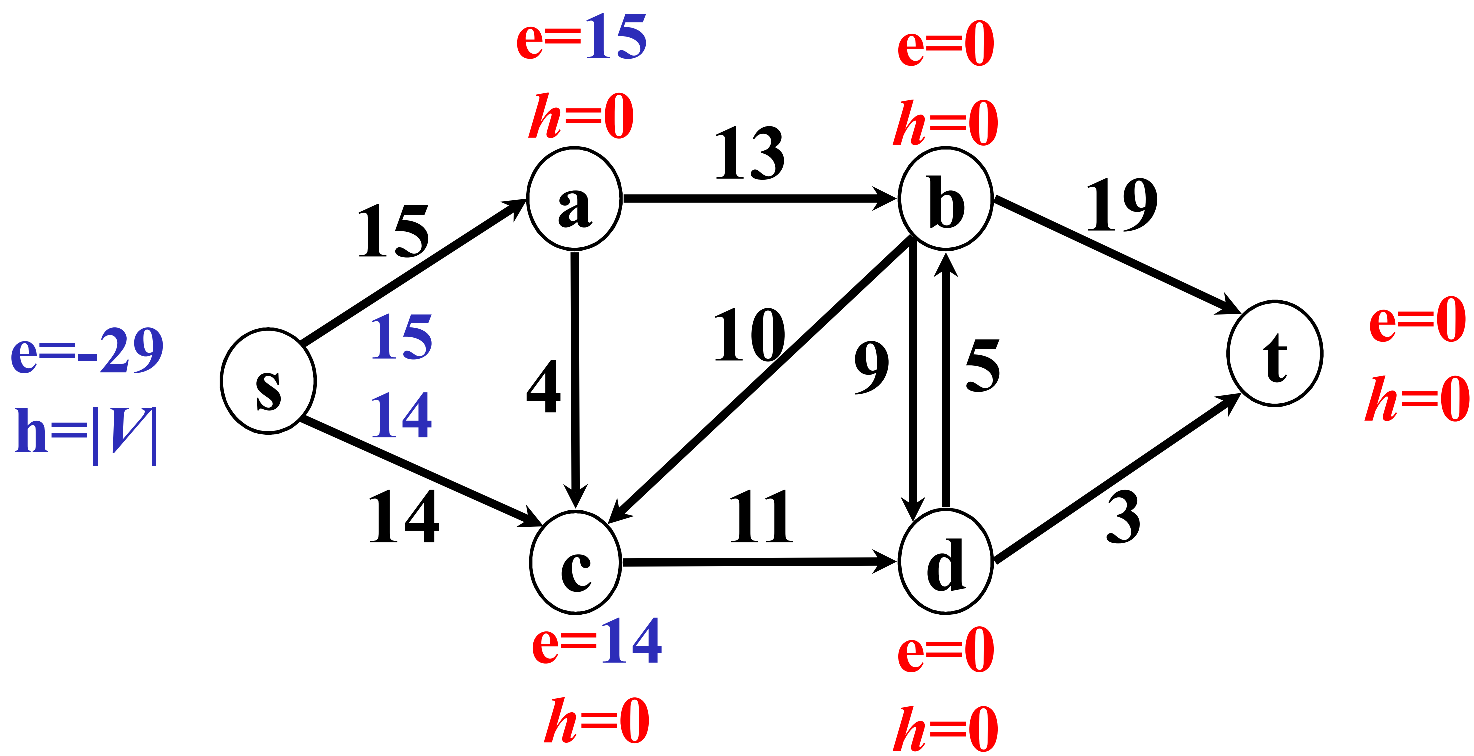


考虑初始化操作之后DisCharge(c)的执行过程

L(c):	s	a	b	d	Null

v=d≠Null， 不执行复标操作

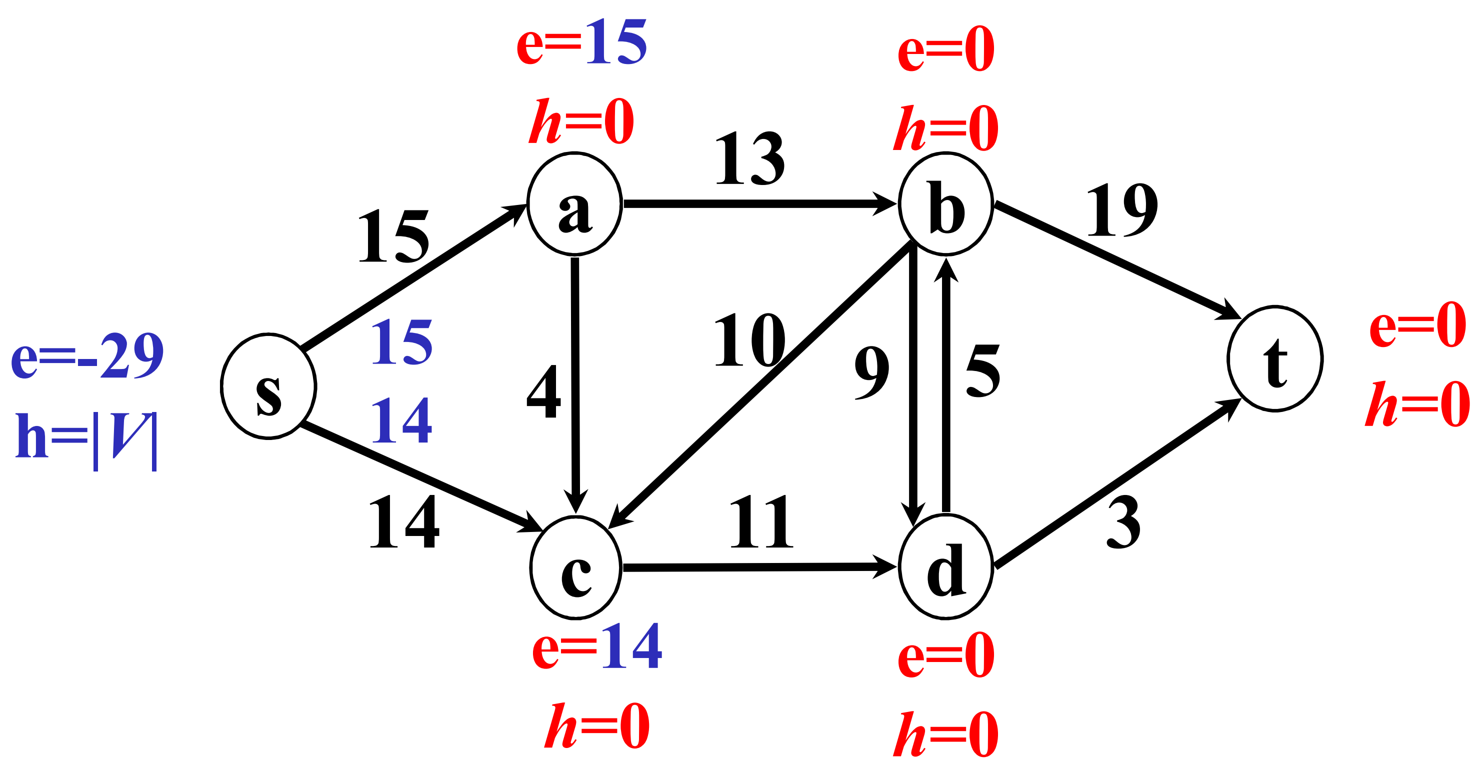
c(cd)-f(cd) =11 但 c.h≠d.h+1 不执行推送操作



考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*): | s | a | b | d | Null |

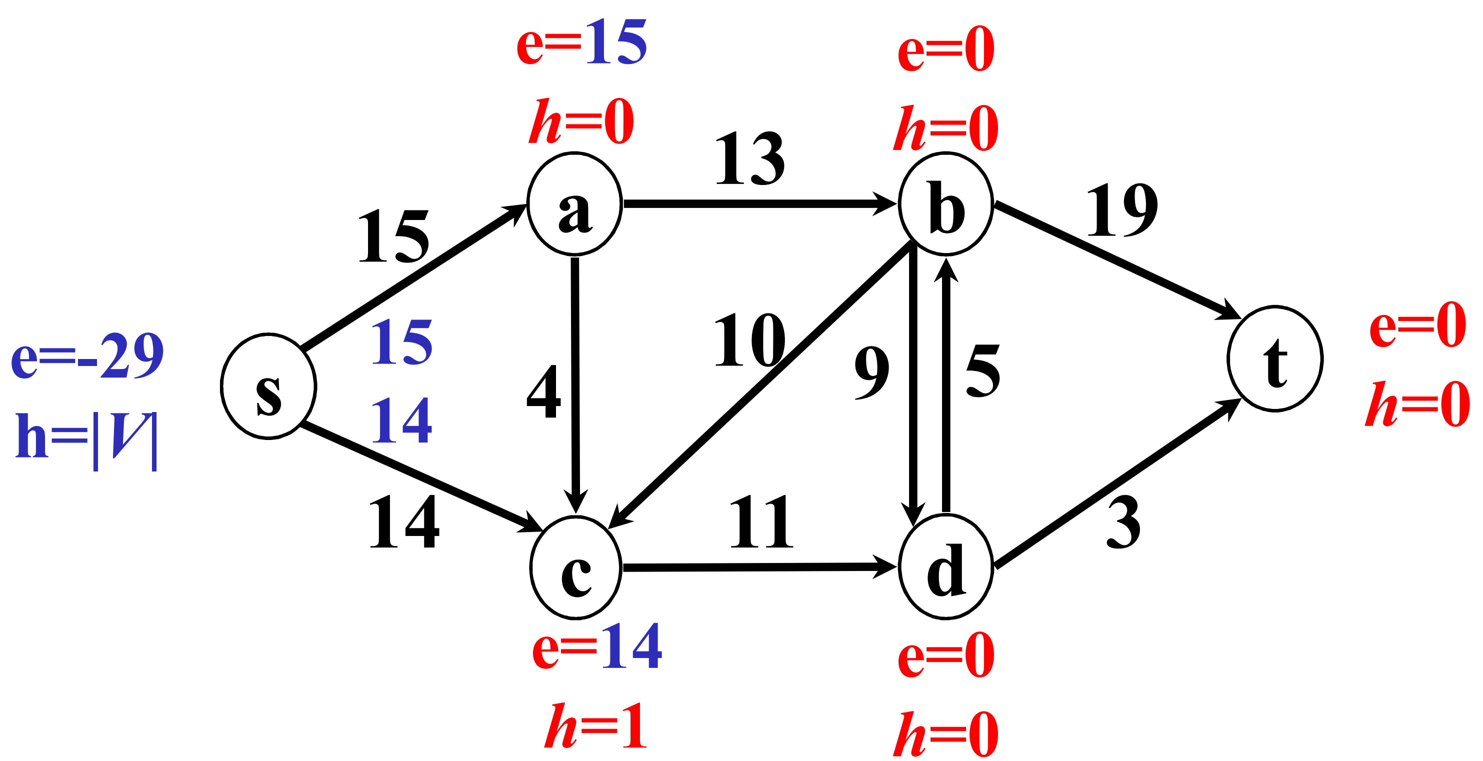
v=Null，   执行复标操作



考虑初始化操作之后DisCharge(c)的执行过程

L(c):				
s	a	b	d	Null

v=Null， 执行复标操作



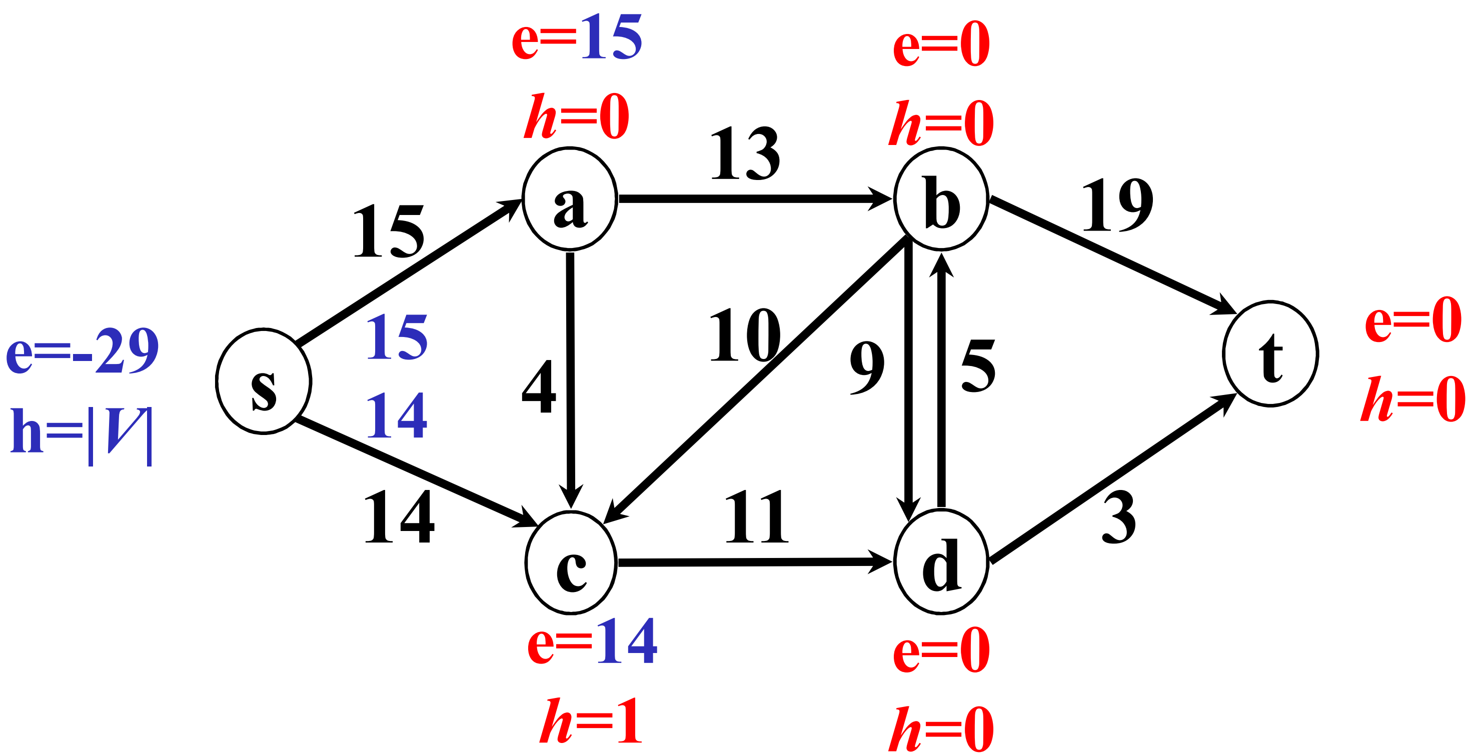
考虑初始化操作之后DisCharge(c)的执行过程

L(c):

s	a	b	d	Null

v=s≠Null， 不执行复标操作

c(cs)-f(cs) = 0-(-14) =14 但c.h≠s.h+1 不执行Push



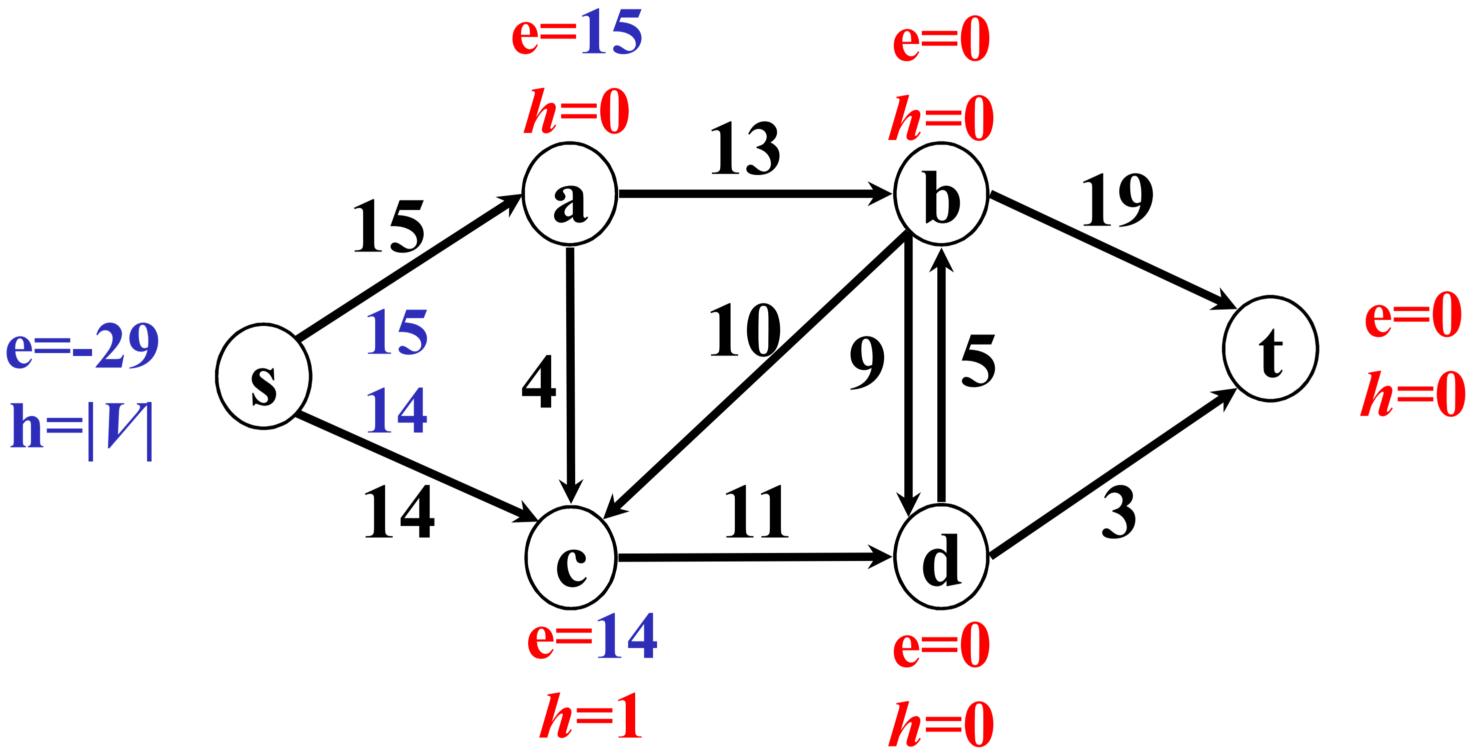
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=a≠Null， 不执行复标操作

c(ca)-f(ca) = 0-0 =0 不执行推送操作

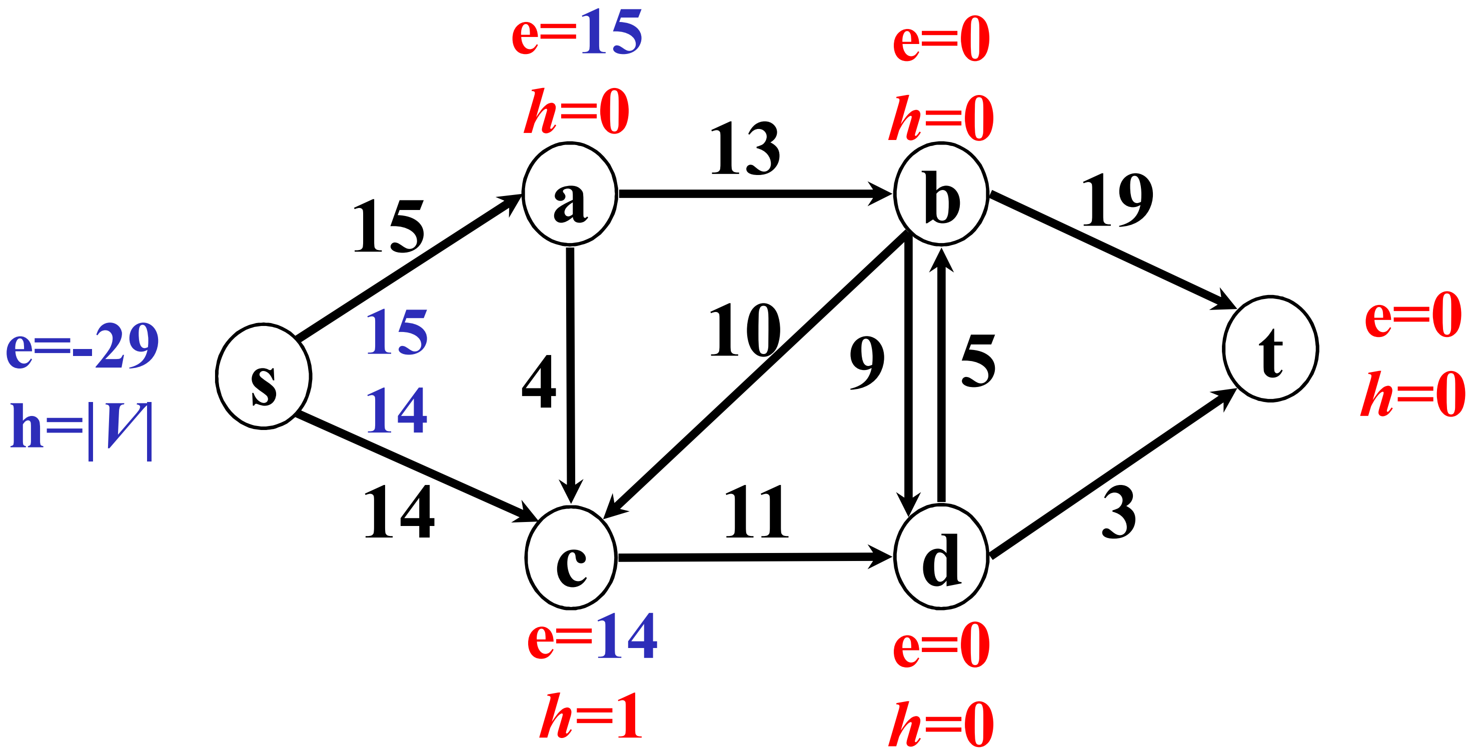


考虑初始化操作之后DisCharge(c)的执行过程

L(c):	s	a	b	d	Null

v=b≠Null，不执行复标操作

c(cb)-f(cb) = 0-0 =0 不执行推送操作

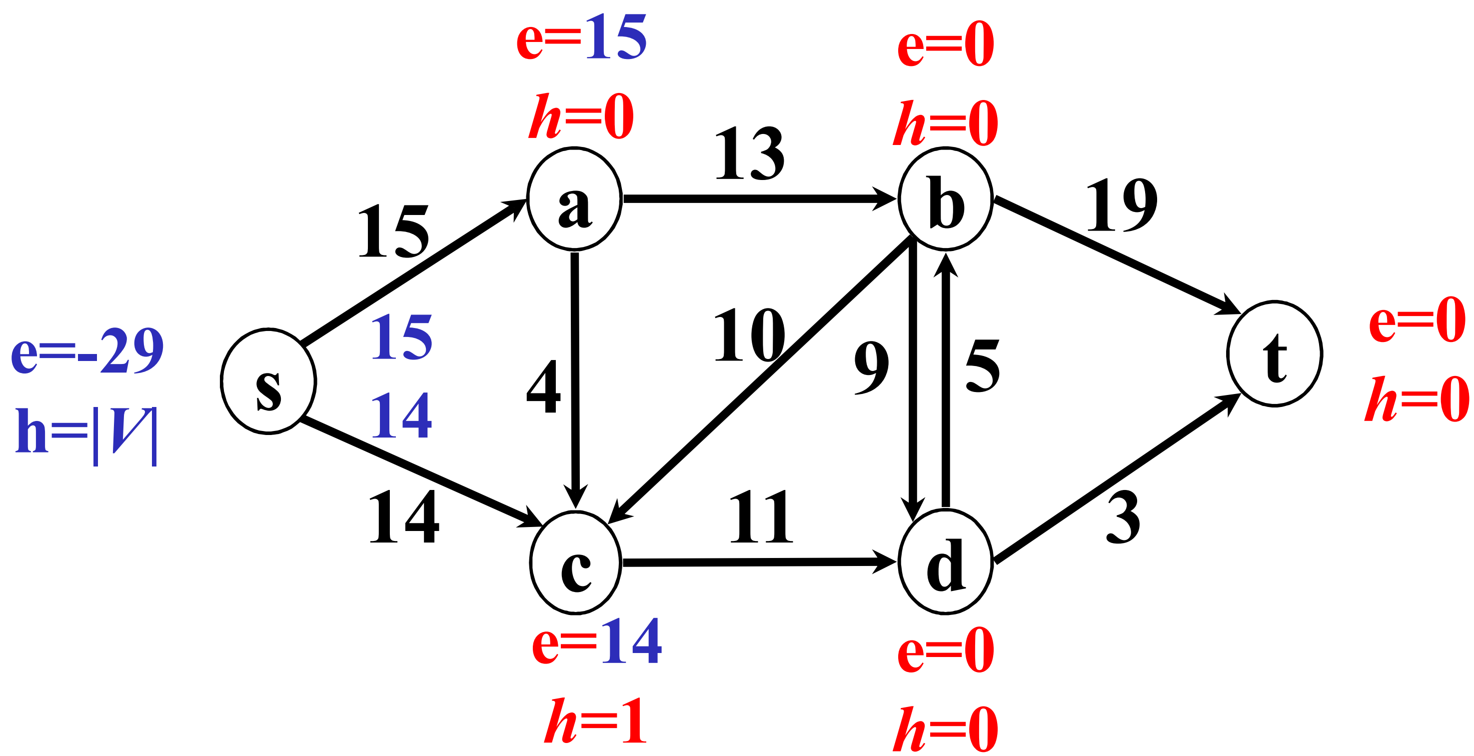


考虑初始化操作之后DisCharge(c)的执行过程

L(c): | s | a | b | d | Null |

v=d≠**Null**，　　　　　不执行复标操作

c(cd)-f(cd) =11 且 **c.h=d.h+1** 执行**Push**操作



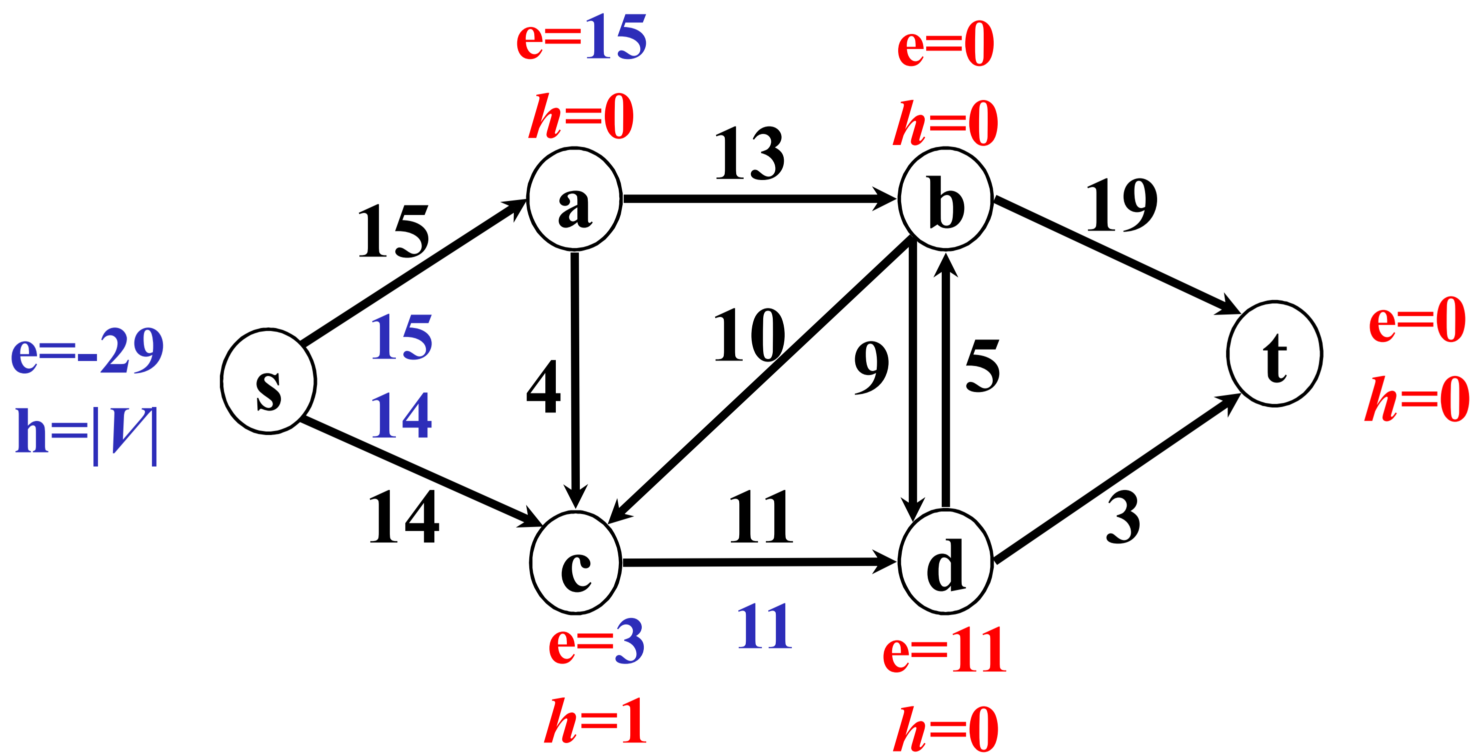
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=d≠Null， 不执行复标操作

c(cd)-f(cd) =11 且 c.h=d.h+1 执行Push操作

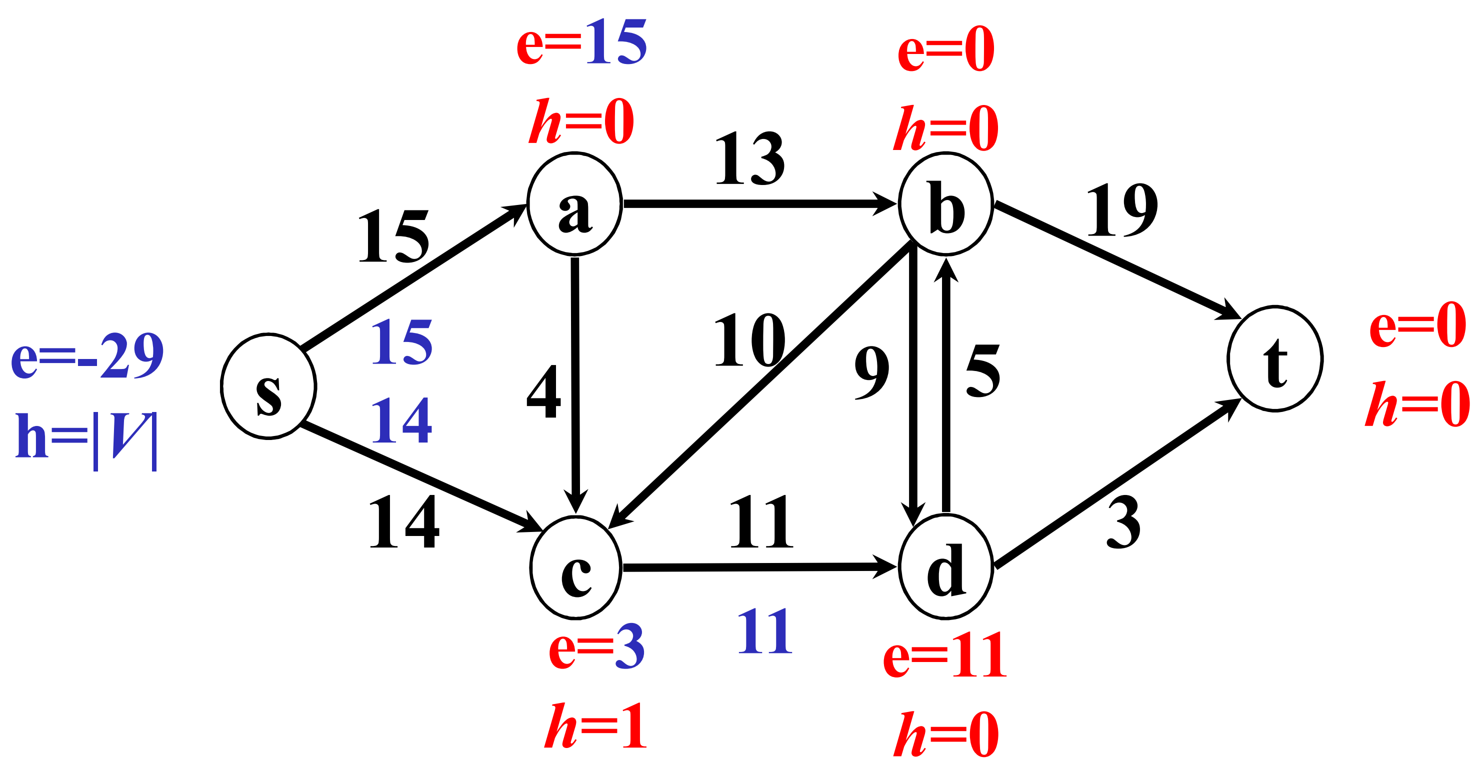


考虑初始化操作之后DisCharge(c)的执行过程

L(c):	s	a	b	d	Null

v=Null， 执行复标操作

$$c.h = 1 + \min\{v.h \mid c(cv)-f(cv)>0\} = |V|+1$$

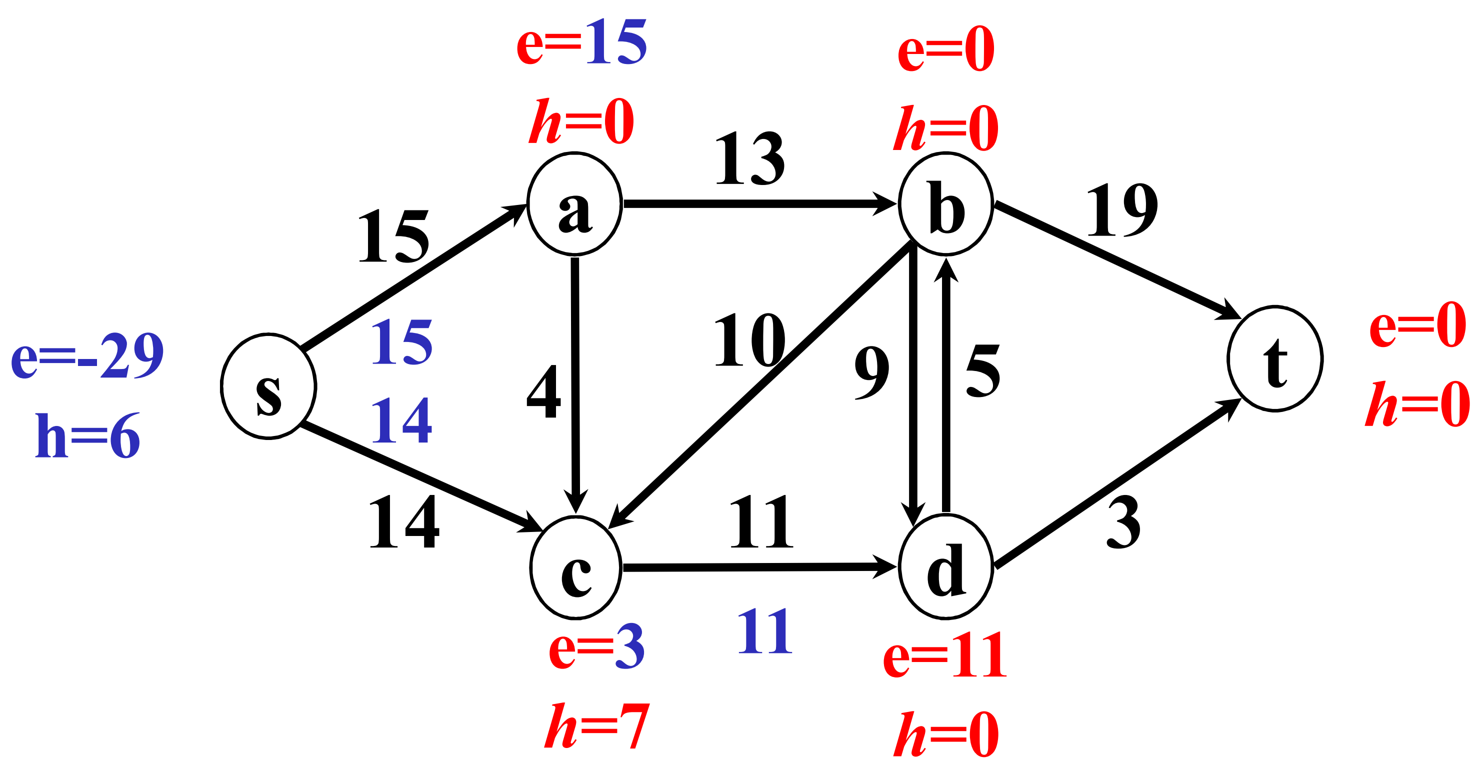


考虑初始化操作之后DisCharge(c)的执行过程

L(c): | s | a | b | d | Null |

v=Null， 执行复标操作

c.h = 1+ min {v.h | c(cv)-f(cv)>0 } = |V|+1

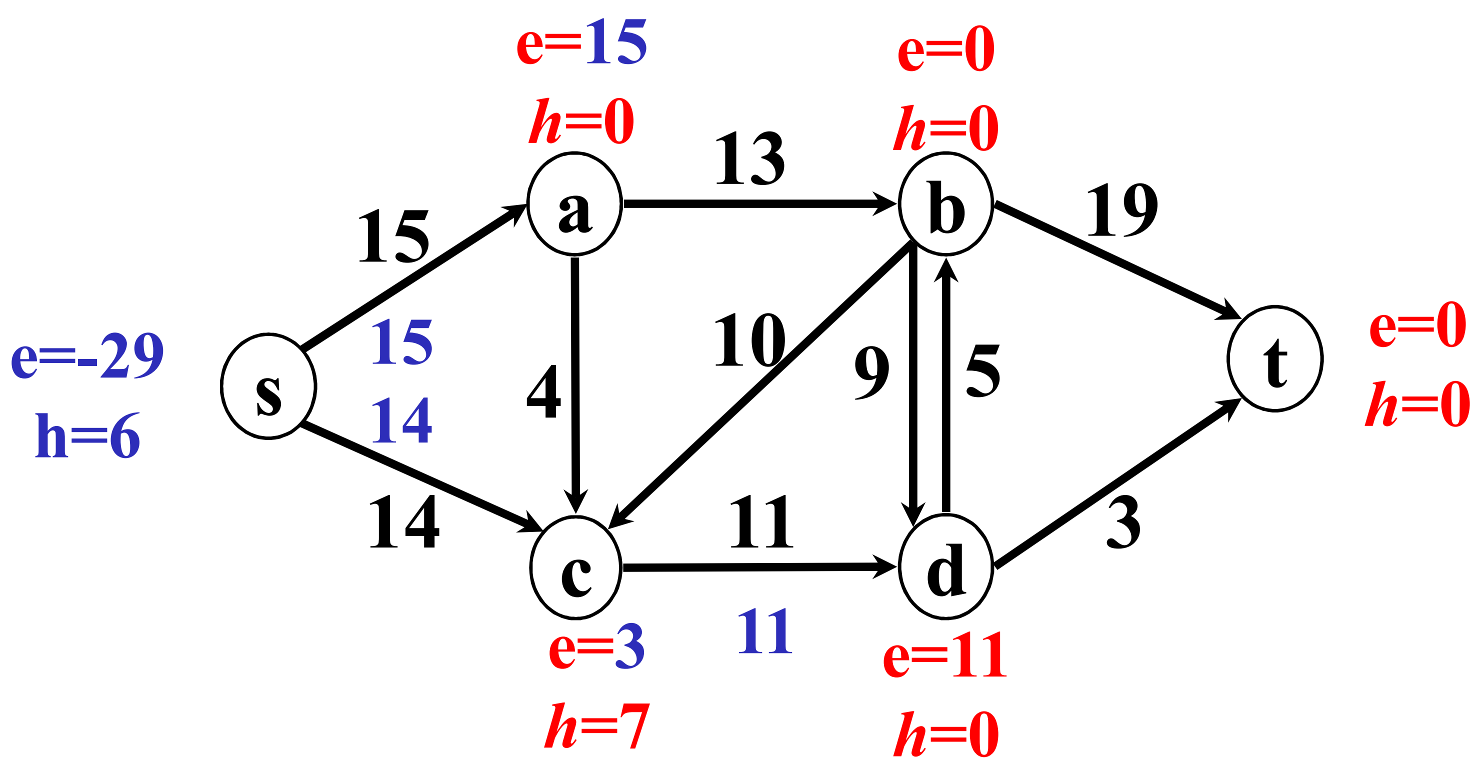


考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*): | s | a | b | d | Null |

v=s≠Null， 不执行复标操作

c(cs)-f(cs)=14且 c.h = c.s+ 1 执行Push

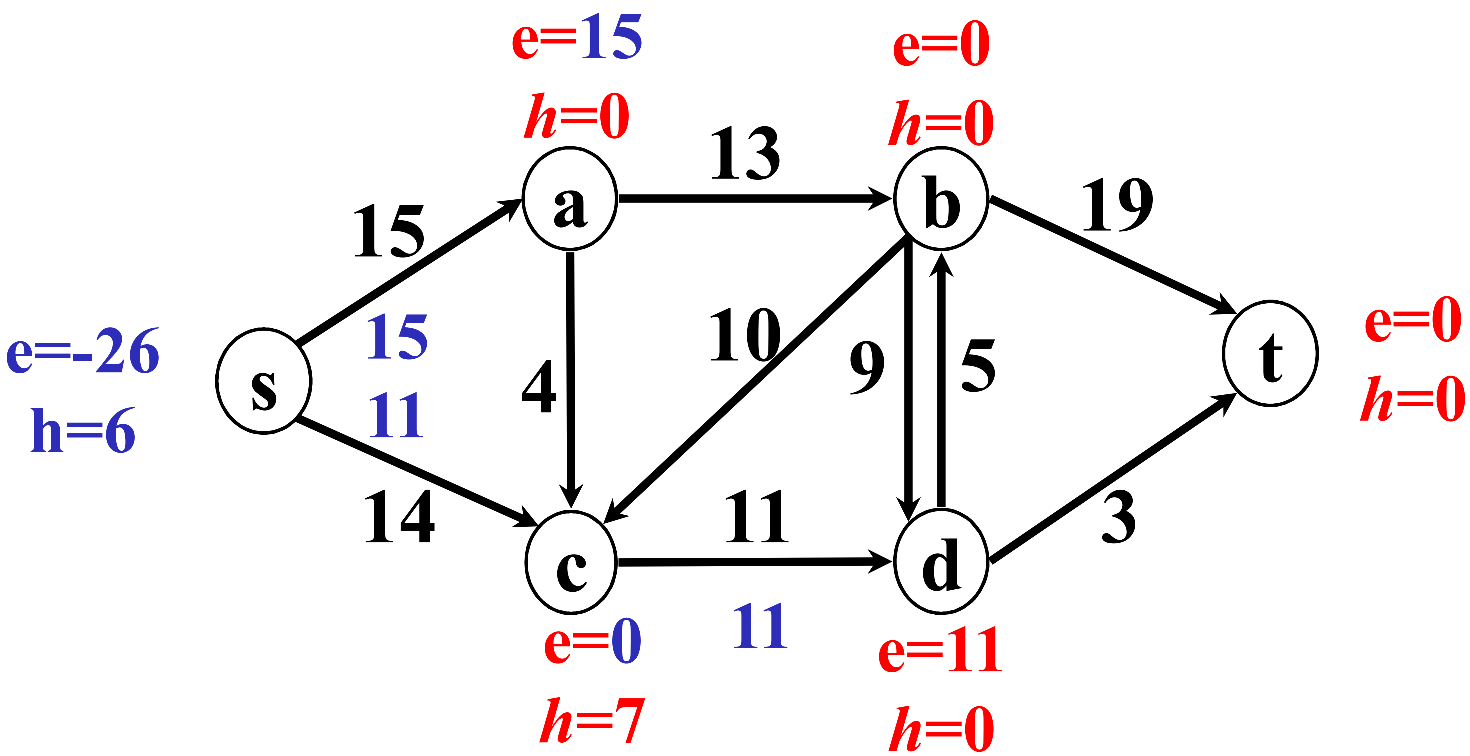


考虑初始化操作之后DisCharge(c)的执行过程

L(c):	s	a	b	d	Null

v=s≠Null， 不执行复标操作

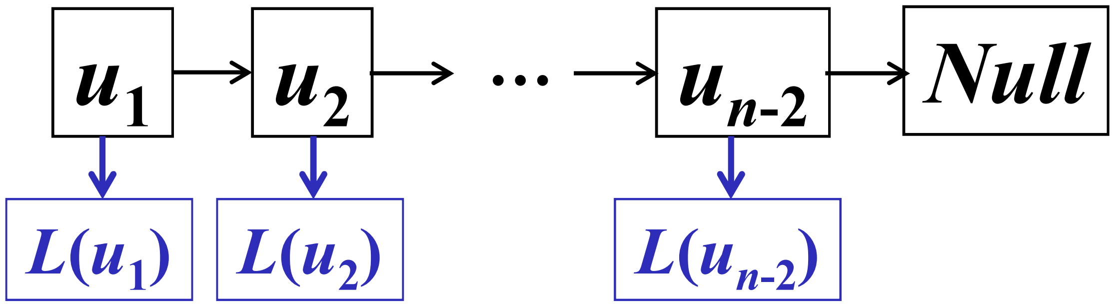
c(cs)-f(cs)=14且 c.h = c.s+ 1 执行Push



前置复标算法

顶点链表B

邻接链表L



依次处理B中的每个顶点u

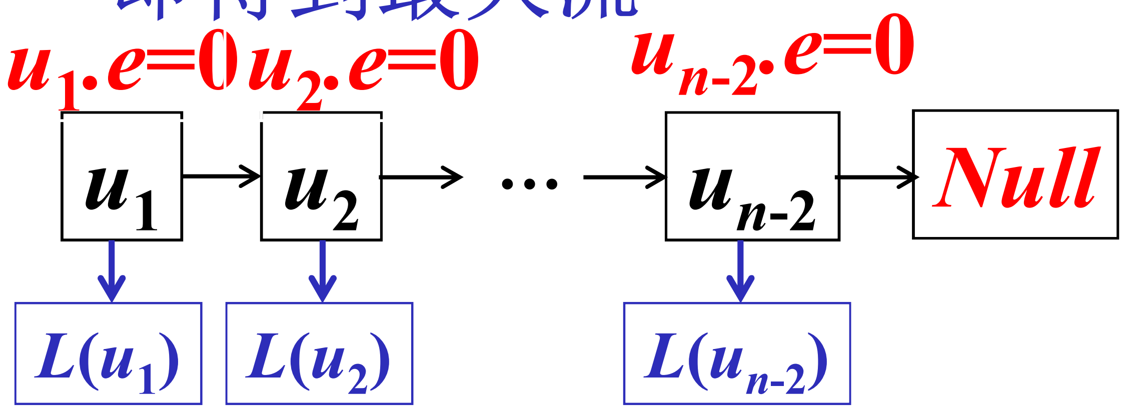
•处理过程即调用**DisCharge**操作使得$u.e$=0

•处理后将u前置于B的开始位置

•然后顺序处理下一个顶点

到达**B**的结束位置，则**u.e=0**对所有顶点成立

即得到最大流



HIT
CS&E

前置复标算法

初始化： u.e=0; u.h=0; uv.f=0 $\forall u\in V, uv \in E$

$s.e= -\sum_{sv\in E} c(sv)$， $s.h=|V|$

$sv.f=c(sv)$, v.e = c(sv) $\forall sv\in E$

创建链表B管理V-{s,t}的所有顶点

L(v)←v的相邻顶点链表

L(v).current ← L(v).head

1. $u \leftarrow B$.head
2. While $u\neq$NULL
3. oldHeight ←u.h
4. DisCharge(u);
5. If u.h>oldHeight then 将u前置到B的前端
6. u ←u.next

前置复标算法分析

引理1.前置复标算法终止后，则最后的预流是最大流。

初始化：算法初始化是预流

循环　：每次推送操作或复标操作后，仍是预流

终止　：算法结束后得到一个预流f

算法结束后$u.e=0$对任意$u \neq s,t$成立，　故f是流

类似与推送复标算法，可以证明剩余网络中不存在s-t路径，　由最大流最小割定理，f是最大流

前置复标算法分析

引理2. 前置复标算法在**O(V^3)**个基本操作之后必然终止。

前置复标算法是推送复标操作的特例

- 至多执行$2V^2$个复标操作
- 至多执行$2VE$个饱和推送操作
- 只需限定非饱和推送的个数

非饱和推送操作 个数 ≤ **DisCharge**执行次数

每次非饱和推送执行后，**u.e=0, DisCharge**操作结束

考察执行复标操作的两个**DisCharge**操作之间

- 算法不改变任意顶点的高度,故算法顺序扫描**B**中顶点
- 算法顺序处理链表**B**中的一个连续区段
- 该区段的长度不超过**B**的总长度**V-2**

非饱和推送至多执行$2V^2\cdot(V-2) < 2V^3$个

7.2 匹配算法

7.2.1　匹配与覆盖

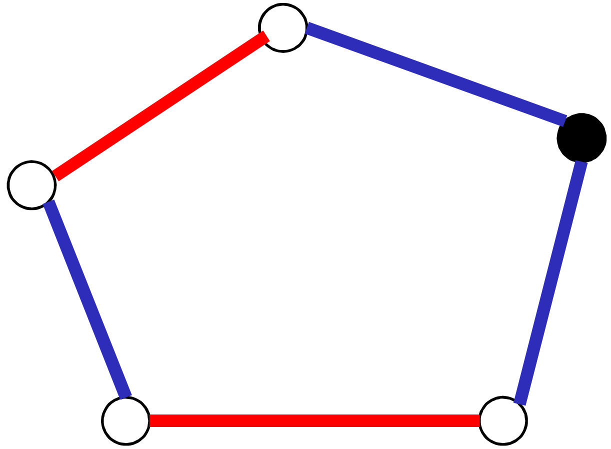
7.2.2　最大二分匹配算法

7.2.3　最大权值二分匹配

7.2.1 覆盖与匹配

匹配——图$G=(V,E)$中没有公共端点的一组边M

- 匹配边——M中的边
- 自由边——E/M中的边
- 被浸润的顶点——M中边的端点
- 未被浸润的顶点——其他顶点



完美匹配——浸润G的每个顶点的匹配

最大匹配——边的条数达到最大值的匹配

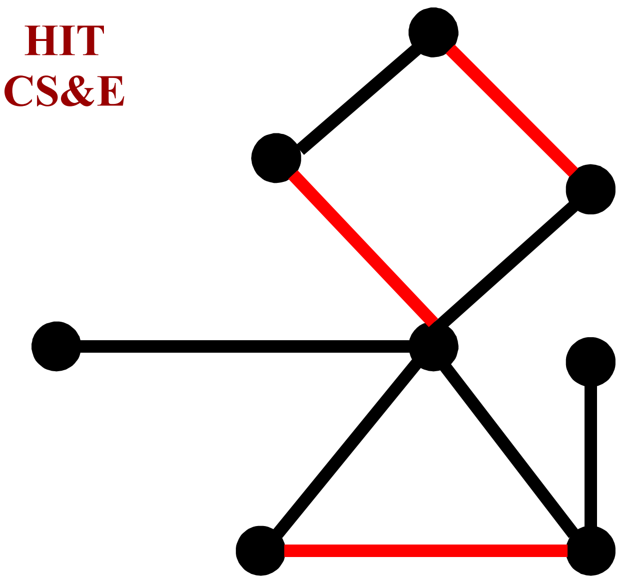
性质：完美匹配是最大匹配，反之不然

最大匹配问题

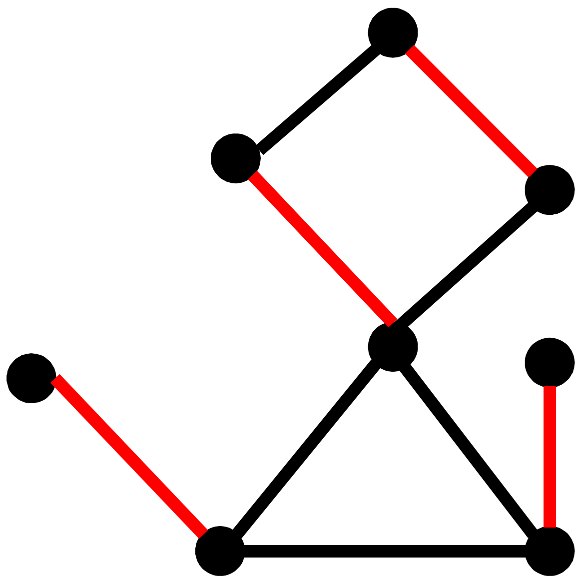
输入：图$G=(V,E)$

输出：G的最大匹配M

HIT
CS&E



最大匹配
非完美匹配

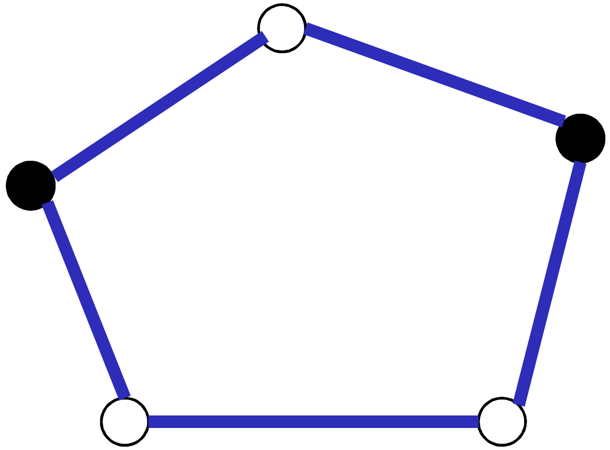


最大匹配
完美匹配

HIT
CS&E

顶点覆盖——图$G=(V,E)$中的一个顶点子集C

E中每条边都至少有一个端点在C中



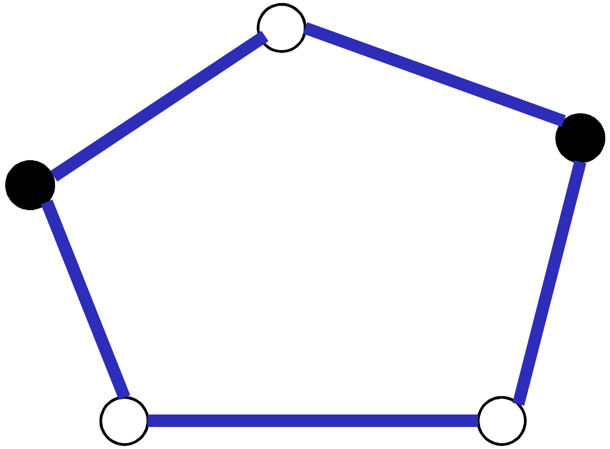
最小顶点覆盖——G的顶点个数最少的覆盖

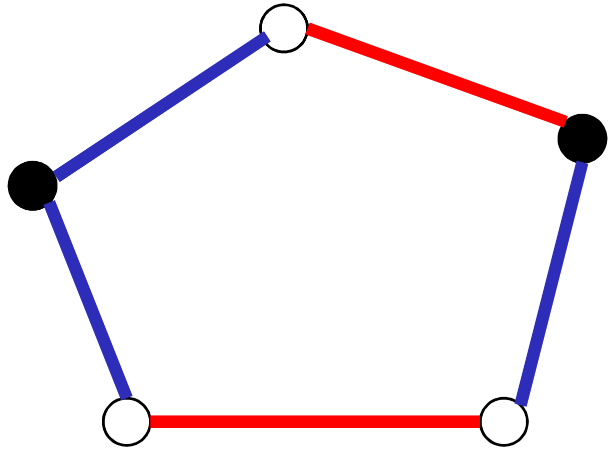
最小顶点覆盖问题

输入：图$G=(V,E)$

输出：G的最小顶点覆盖

弱对偶性

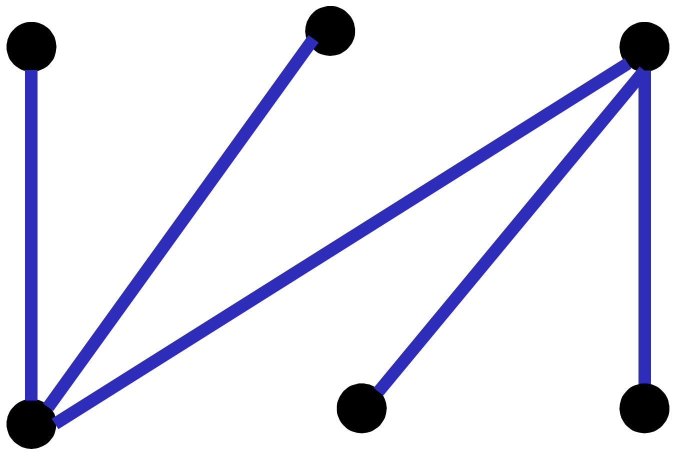


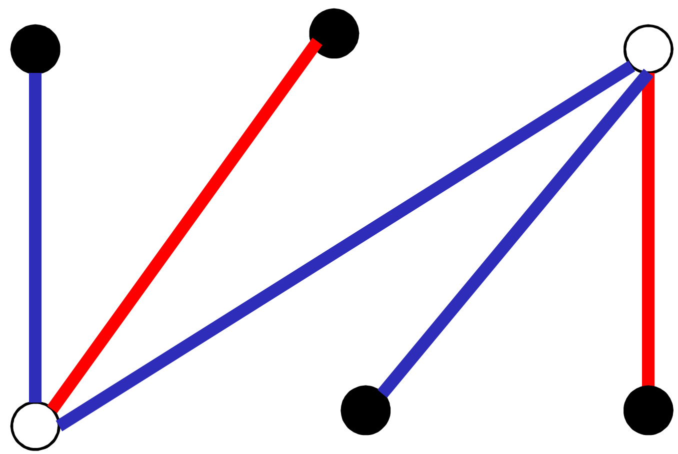


若C是图G的任意顶点覆盖，M是图G的任意匹配

- ◆M中每条边都至少有一个端点在C中
- ◆M中任意两条边不存在公共端点

故$|M|\le|C|$





若C是图G的任意顶点覆盖，M是图G的任意匹配

- ◆M中每条边都至少有一个端点在C中
- ◆M中任意两条边不存在公共端点

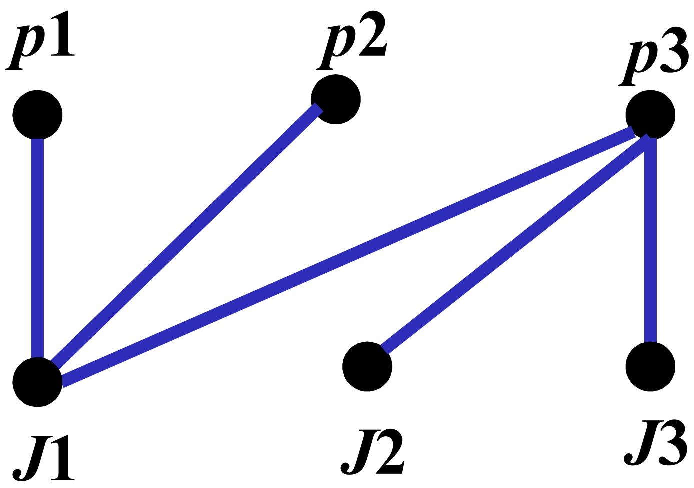
故$|M|\leq|C|$

定理: 图G的最小顶点覆盖C和最大匹配M满足$|M| \leq|C|$
在二分图G中,$|M|=|C|$

研究匹配算法的意义

工作分配

输入：*n*个人*p*1,…,*pn*，*n*项工作*J*1,…,*Jn*，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



计算节点　　实习单位

计算任务　　实习人员

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！

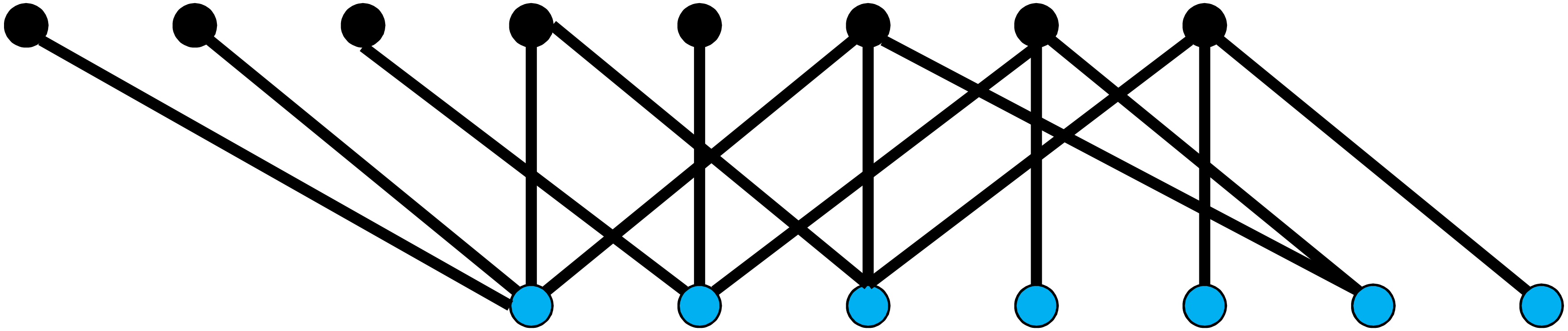
7.2.2 最大二分匹配算法

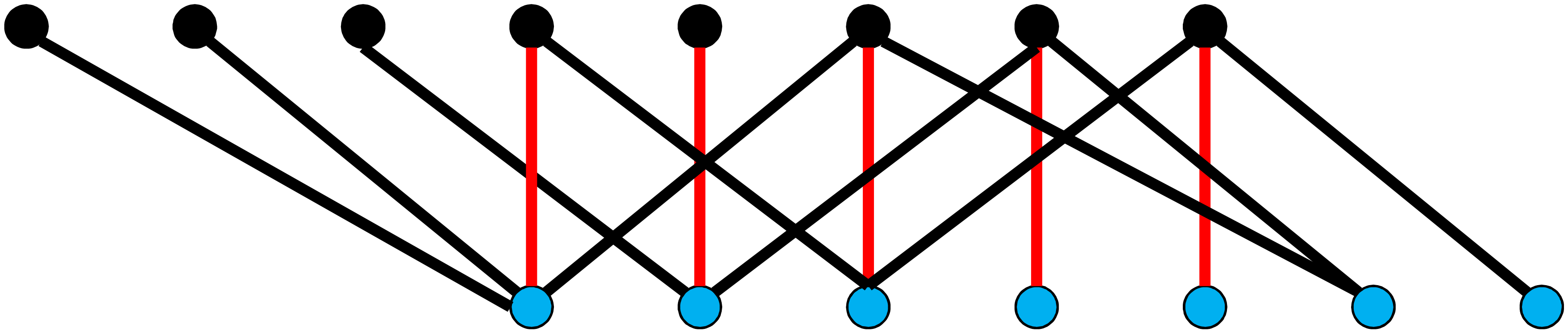
定理:图G的最小顶点覆盖C和最大匹配M满足$|M| \leq |C|$
在二分图G中,$|M|=|C|$

这意味着二分图上的最大匹配可以这样求解

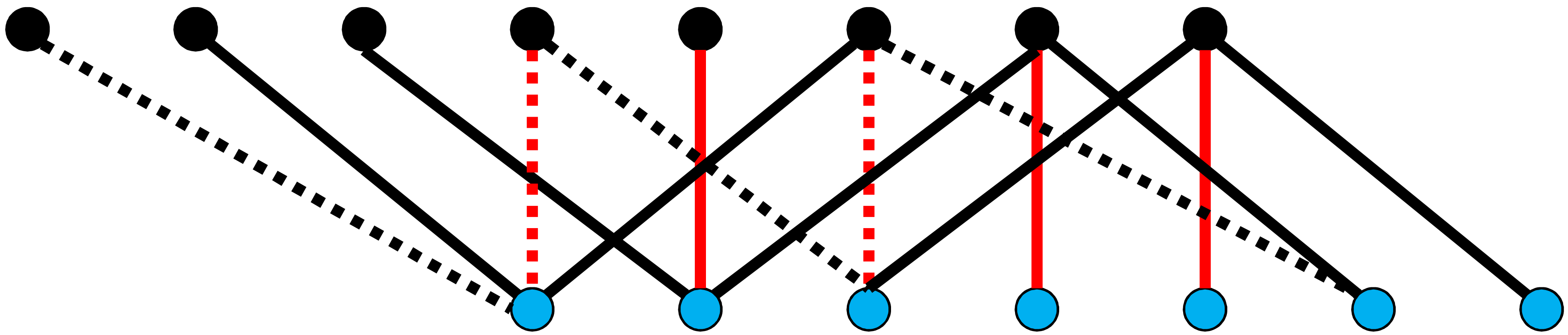
- ◆初始化一个匹配M
- ◆不断地增大M
- ◆M无法增大时，找出一个顶点覆盖C使得$|M|=|C|$
- ◆M是最大匹配，C是最小覆盖

初始化



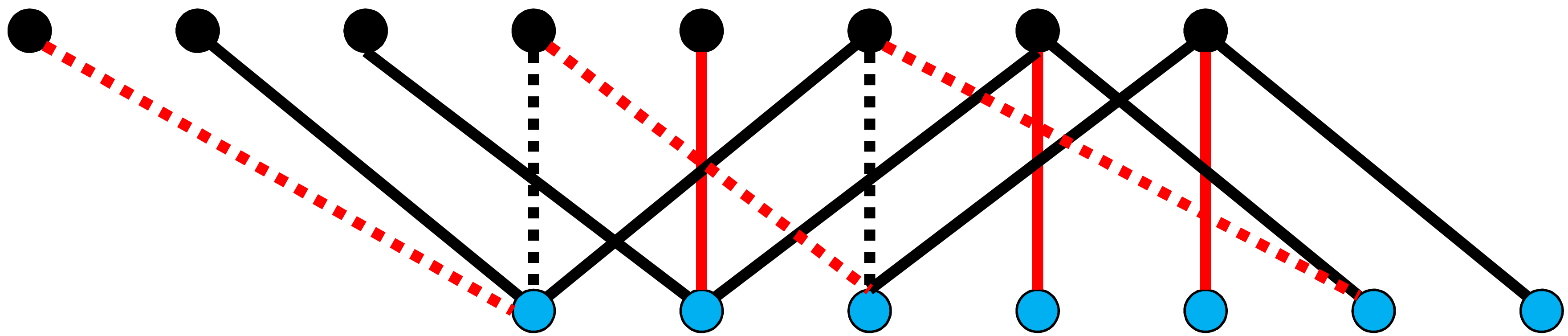


增大



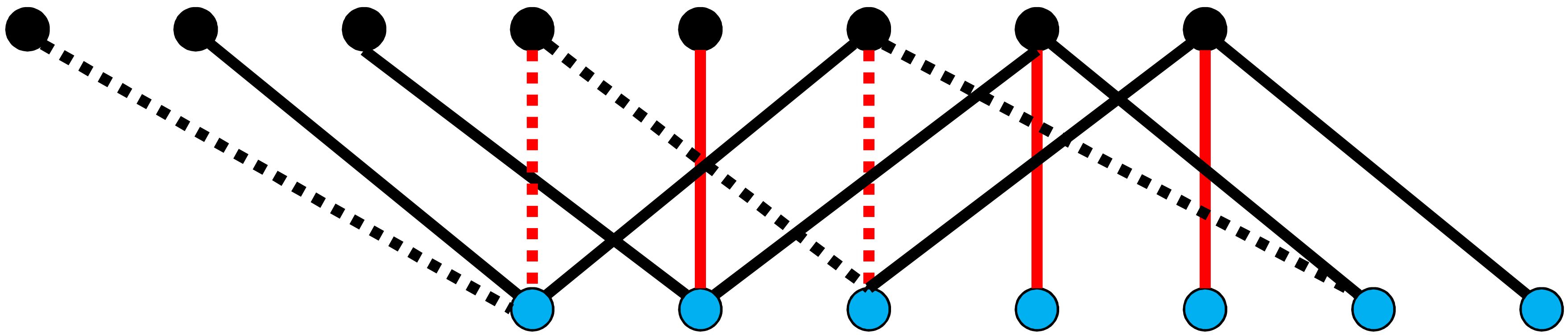
*M*交错路径：边交替出现在*M*和*E-M*的路径

*M*增广路径：端点未被*M*浸润的交错路径



找出增广路径，就能增大*M*！怎么找呢？

通过BFS找增广路径



1.$U \leftarrow X$中未被M浸润的所有顶点

2.For $\forall x \in U$ do

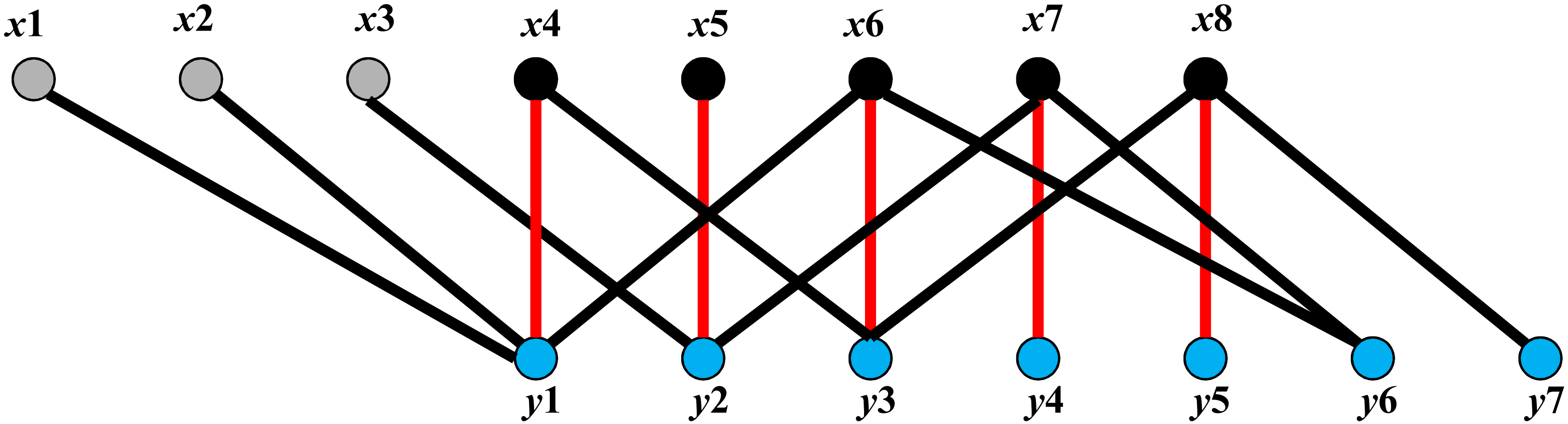
如果存在从x出发的M增广路径，则增大M

Goto 1

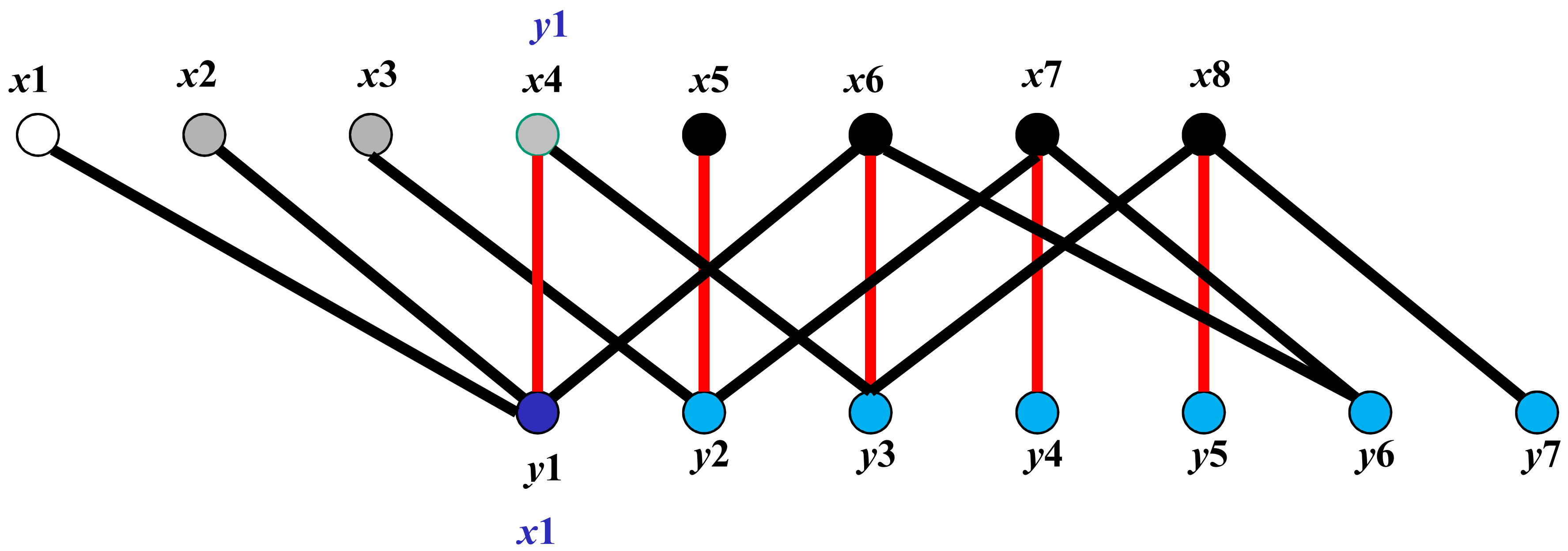
3.M是最大匹配

能够以更高效的方式进行广度优先搜索呢？

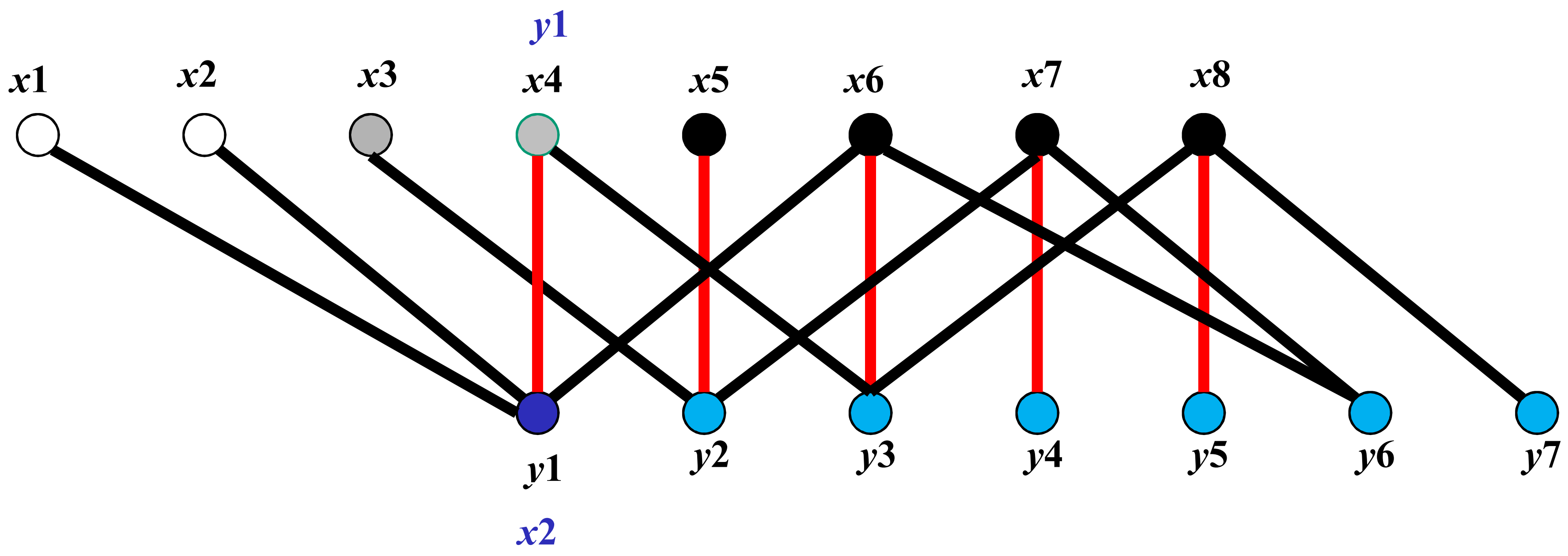
同时进行BFS



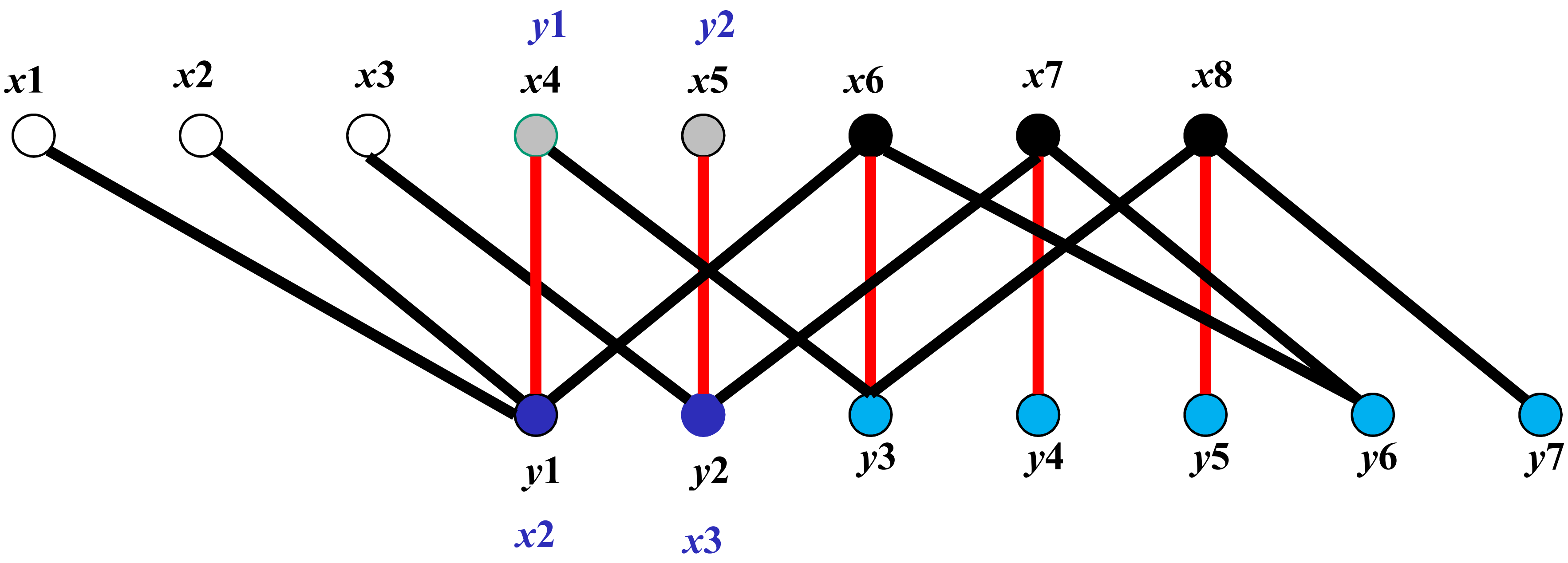
同时进行BFS



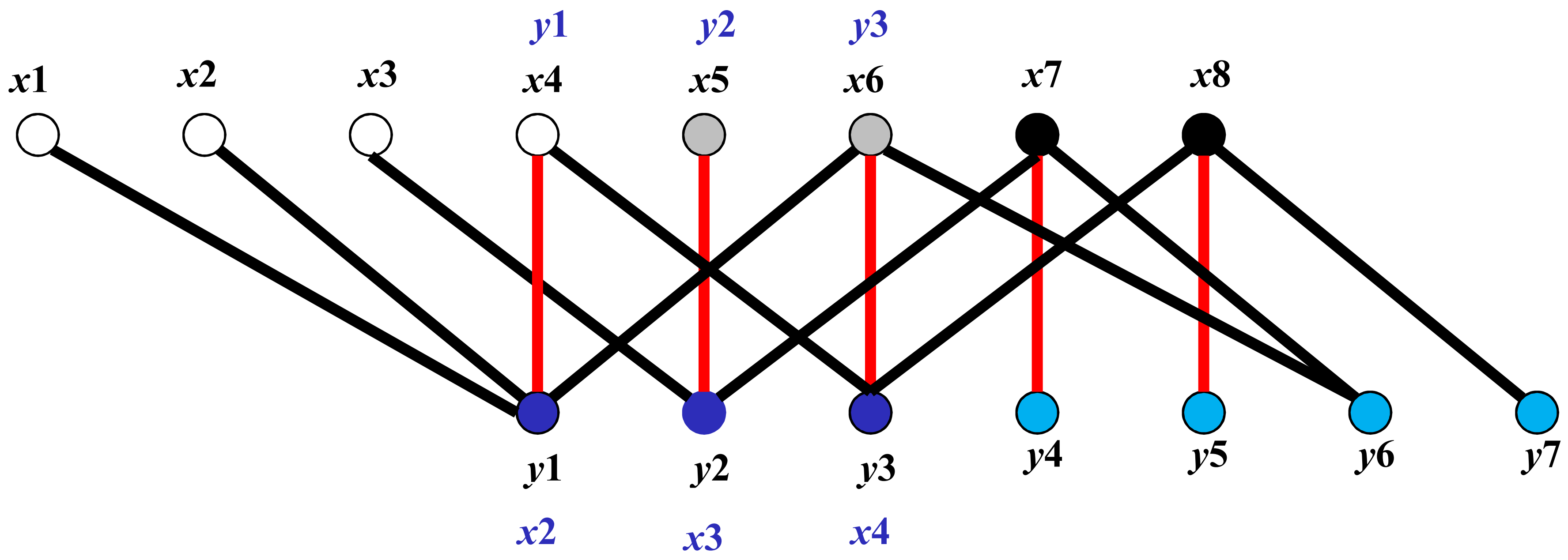
同时进行BFS



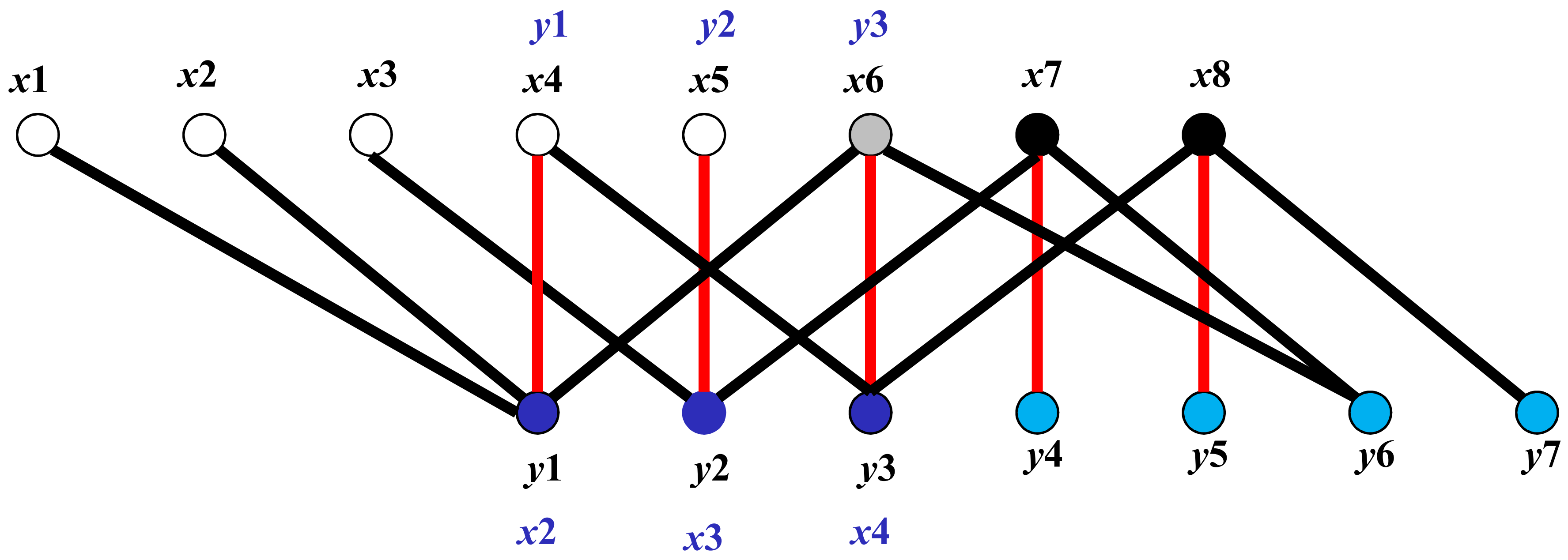
同时进行BFS



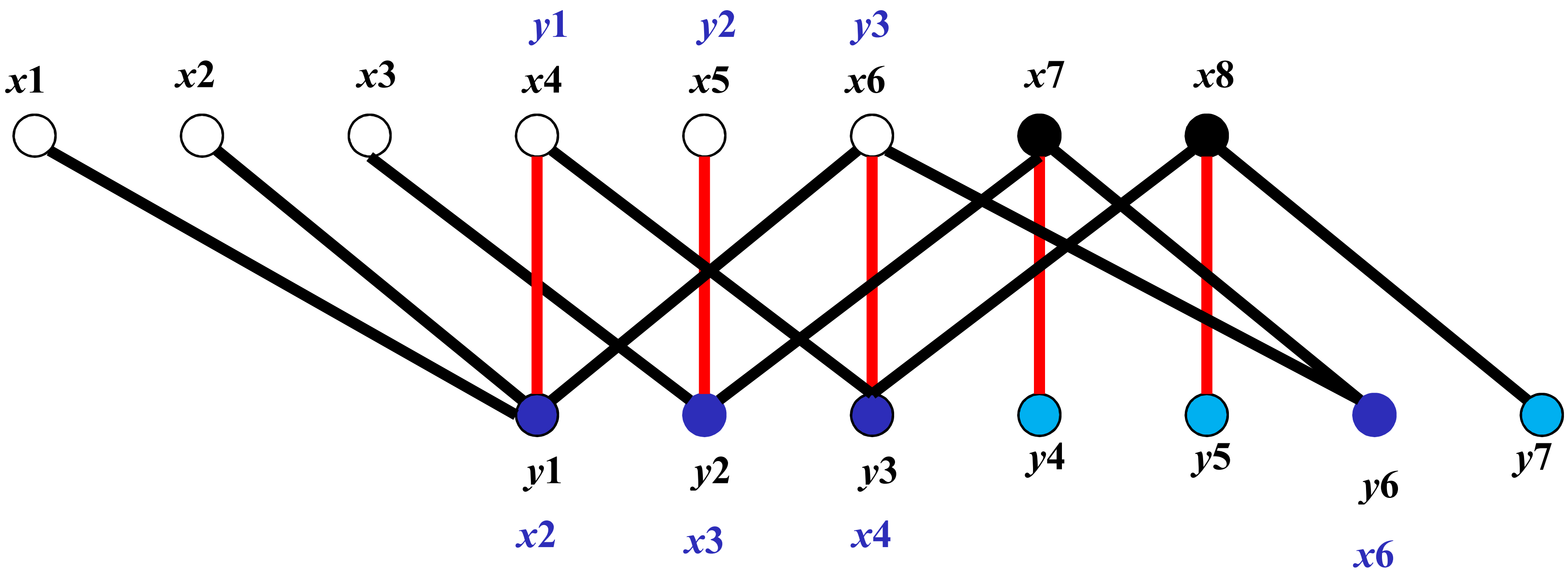
同时进行BFS



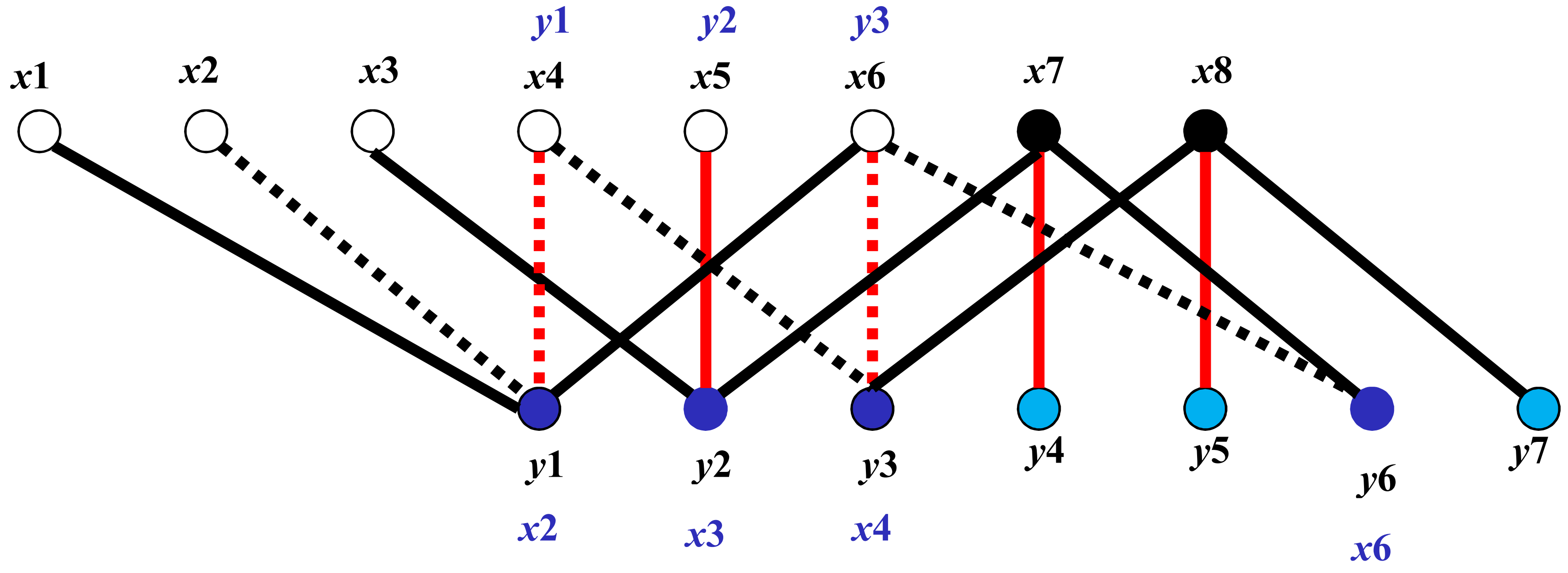
同时进行BFS

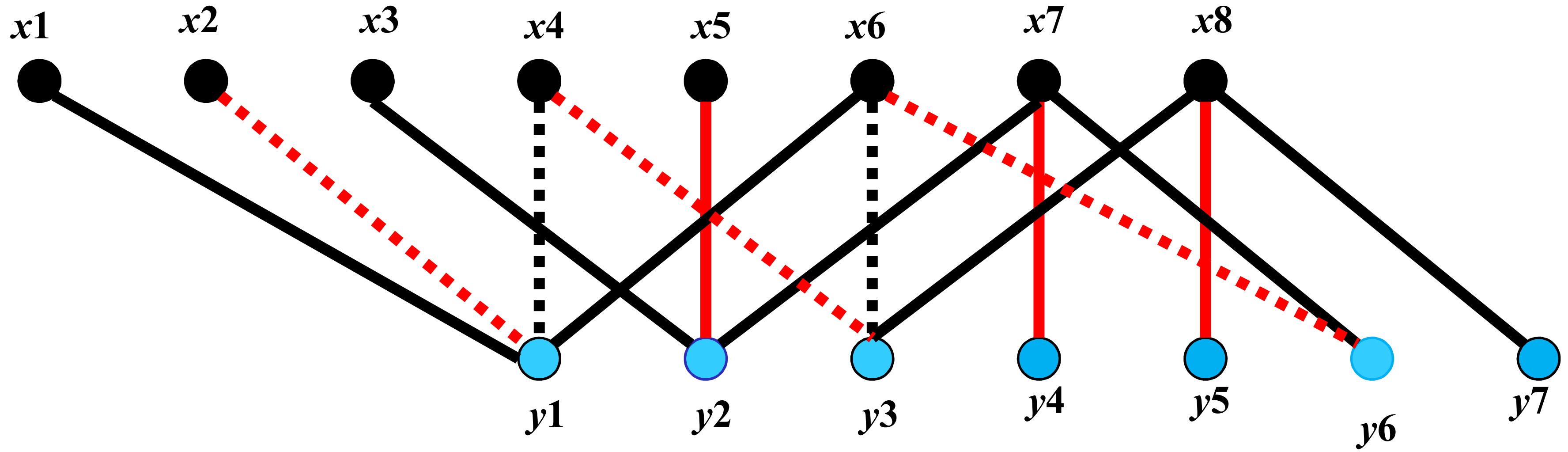


同时进行BFS

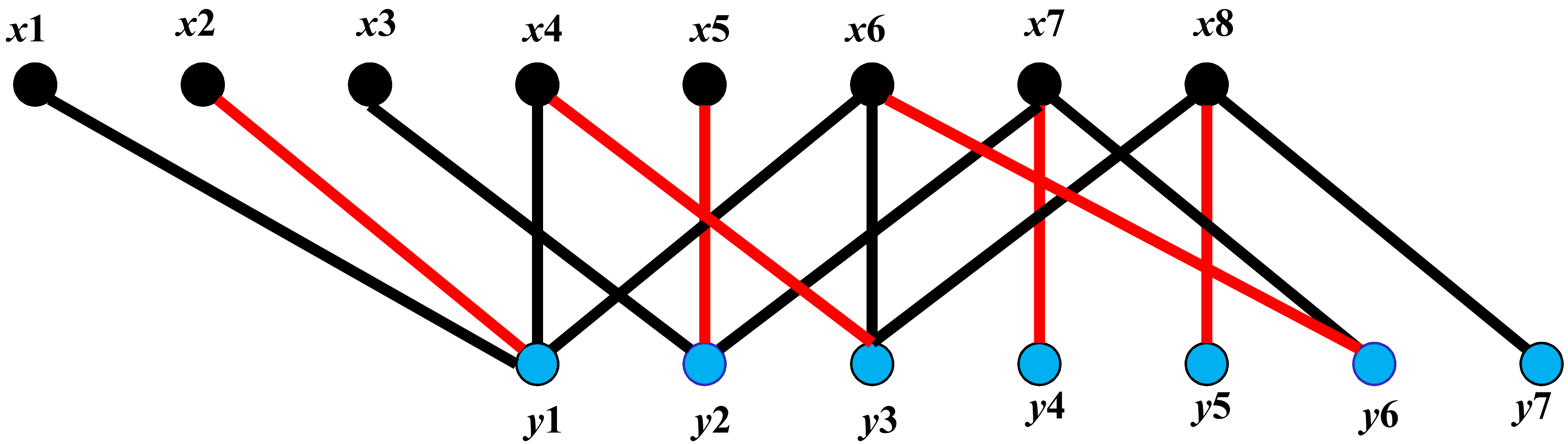


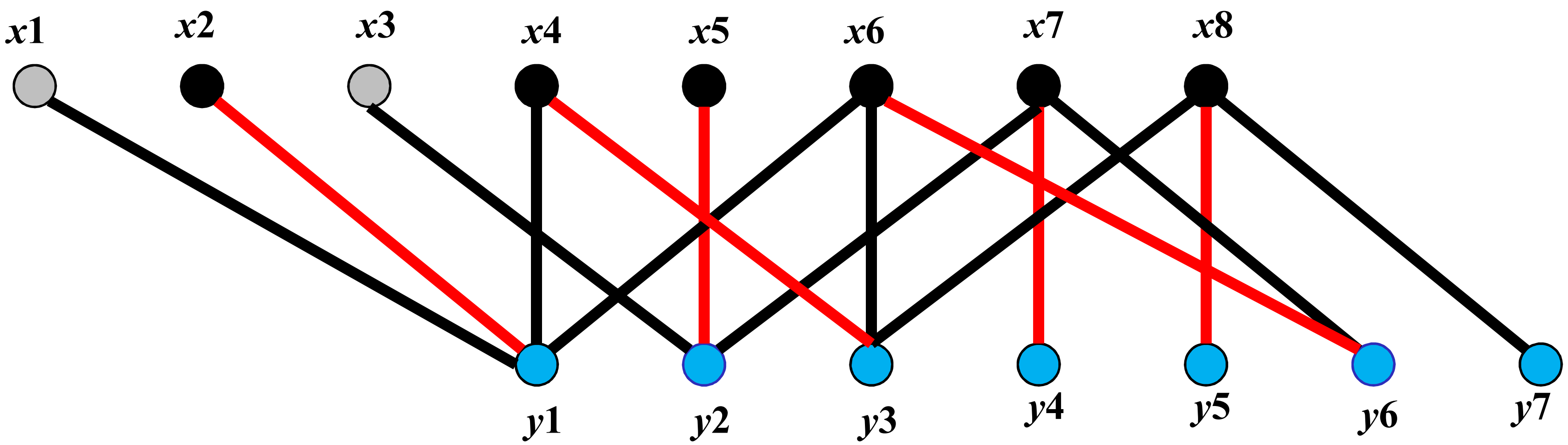
同时进行BFS


y1
y2
y3
x1
x2
x3
x4
x5
x6
x7
x8
y1
y2
y3
y4
y5
y6
y7
x2
x3
x4
x6


x1
x2
x3
x4
x5
x6
x7
x8
y1
y2
y3
y4
y5
y6
y7

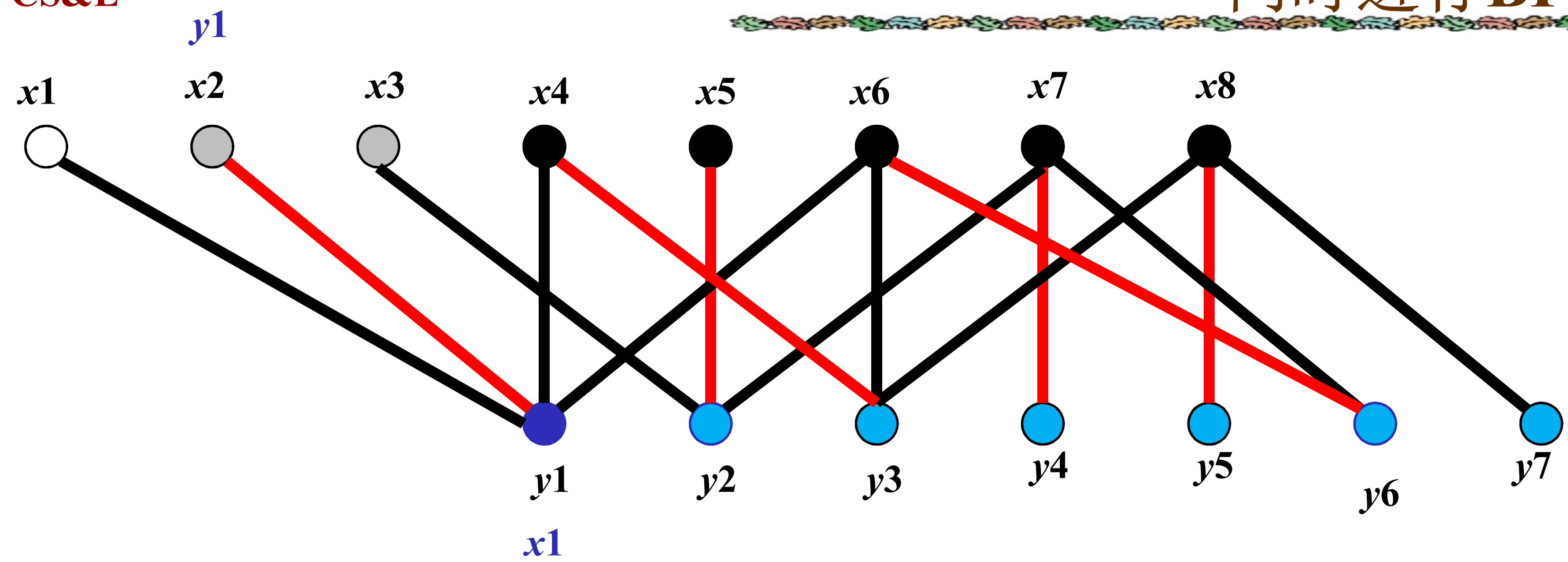
同时进行BFS

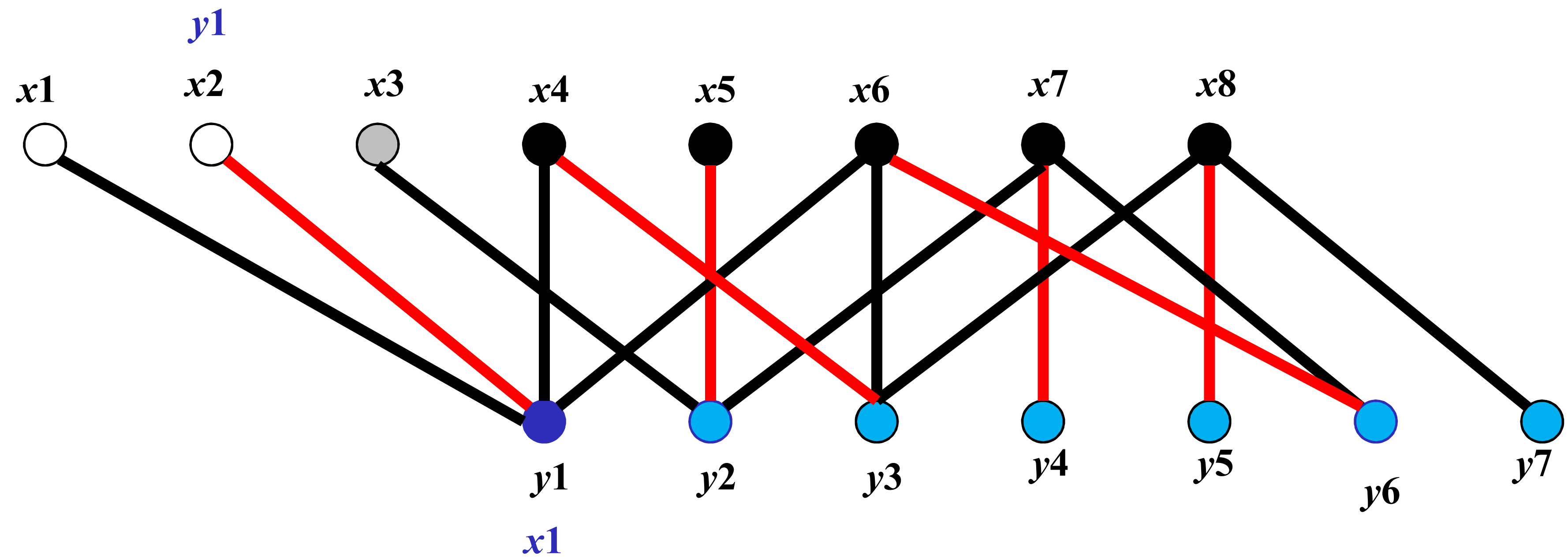

x1
x2
x3
x4
x5
x6
x7
x8
y1
y2
y3
y4
y5
y6
y7


x1
x2
x3
x4
x5
x6
x7
x8
y1
y2
y3
y4
y5
y6
y7

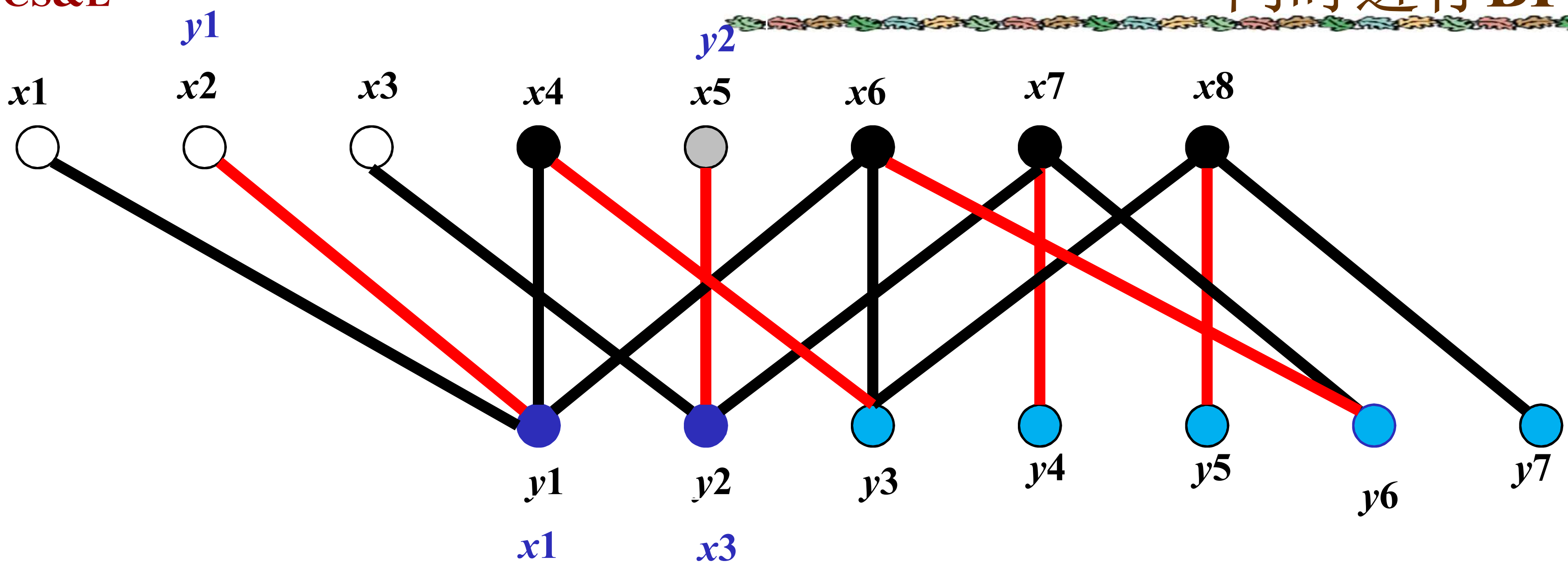
HIT
CS&E

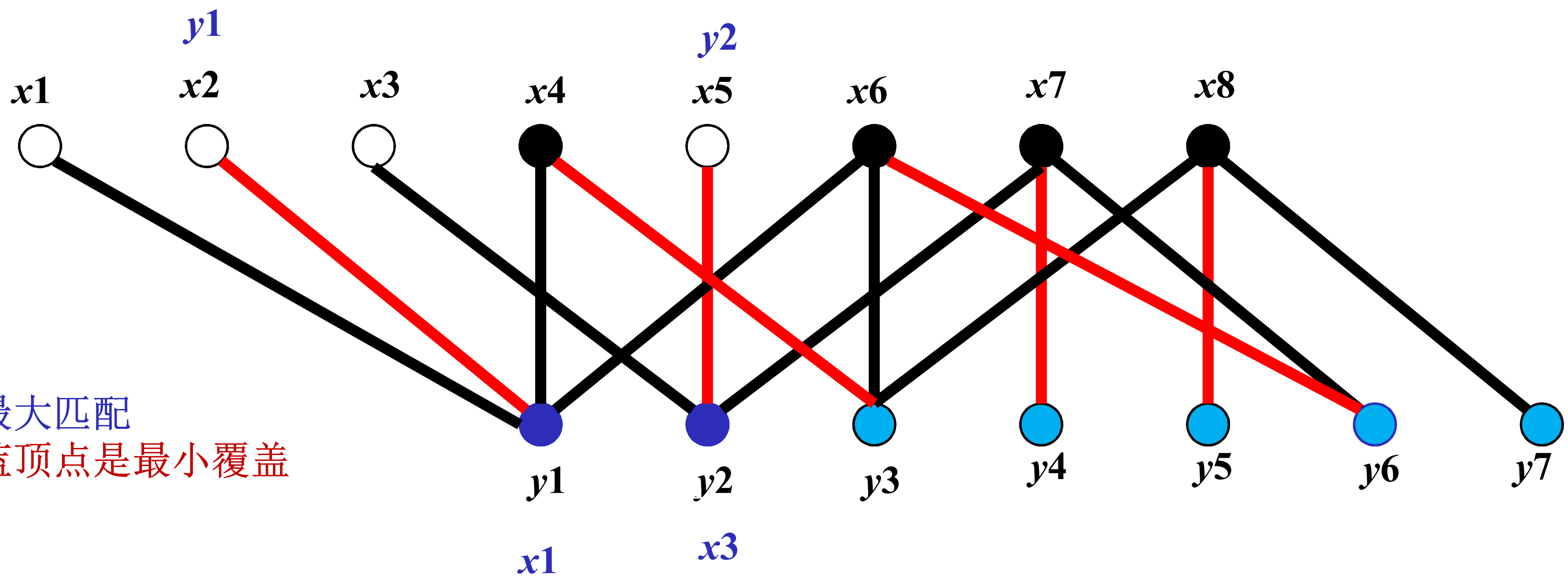
同时进行BFS





同时进行BFS





红边是最大匹配
黑、深蓝顶点是最小覆盖

HIT
CS&E

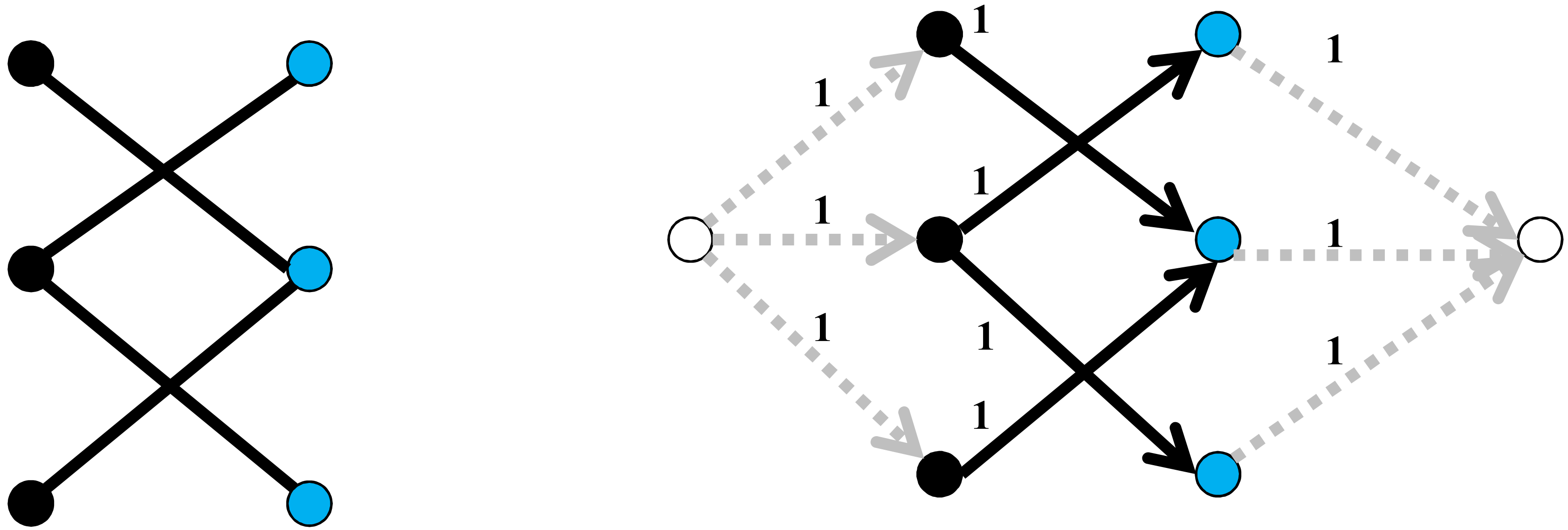
二分匹配的增广路径算法

输入：二分图$G=(X\cup Y,E)$
输出：G的最大匹配和最小覆盖

时间复杂度$O(|V||E|)$

```
1. M←∅;   IsInc←true
2. While  IsInc  do
3.   U←X中未被M浸润的所有顶点;  IsInc←false;
4.   S←U, T←∅ ;             //S记录灰、白顶点，T记录蓝顶点
5.   While  S中存在灰顶点 x   do
6.     For  ∀(x,y) ∈E-M   do
7.         If  ∃(w,y) ∈M  Then
8.            T←T∪{y},记录y由x到达
9.            S←S∪{w}，记录w由y到达,w标记为灰色
10.        else
11.           由y回溯得到一条增广路径，并用它增大M
12.           IsInc ←true; Goto第2步;
13.    标记x为白色;                  //从x的BFS结束
14. 输出M为最大匹配，输出T∪(X-S)为最小覆盖
```

最大二分匹配的另一种算法

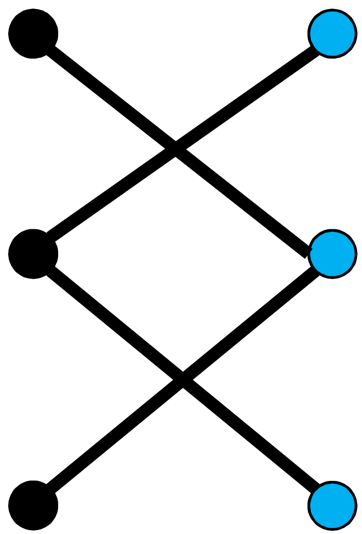


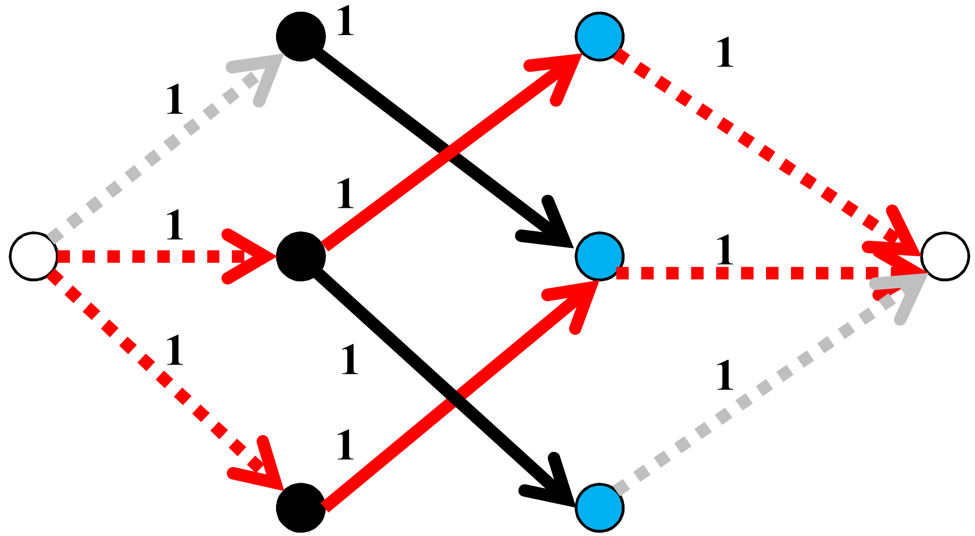
将G=(V,E)中每条边定向为从X到Y并定义容量1

添加虚拟源s,和s到X中各个顶点的有向边，容量为1

添加虚拟汇t,和Y中各顶点到t的有向边，容量为1

基于最大流的最大二分匹配算法



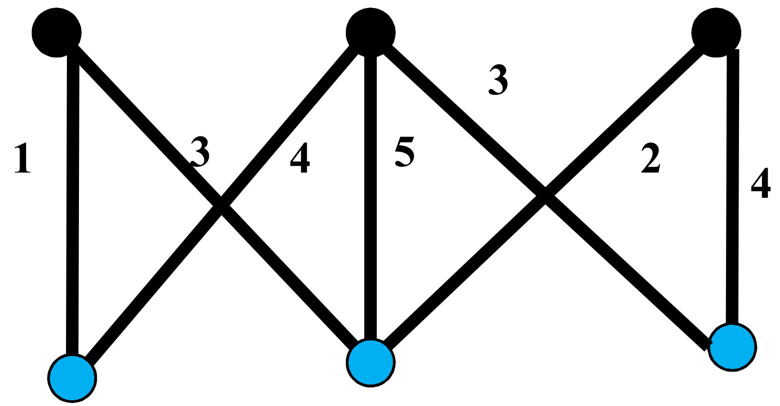


*f*是新图上的最大流

当且仅当

*E*中流量大于**0**的边构成*G*的最大匹配*M*

7.2.3最大权值二分匹配



计算结点

- 边上的权值表示计算效率
- 最大匹配就是计算效率最高的计算任务分配方案

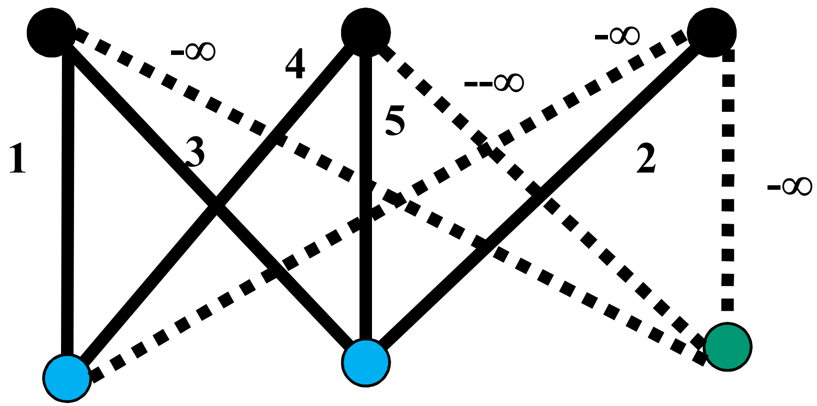
计算任务

匹配M的权值$c(M)$:M中各条边的权值之和

最大权值二分匹配问题

输入：加权二分图$G=(X\cup Y,E)$, $w:E\to R$

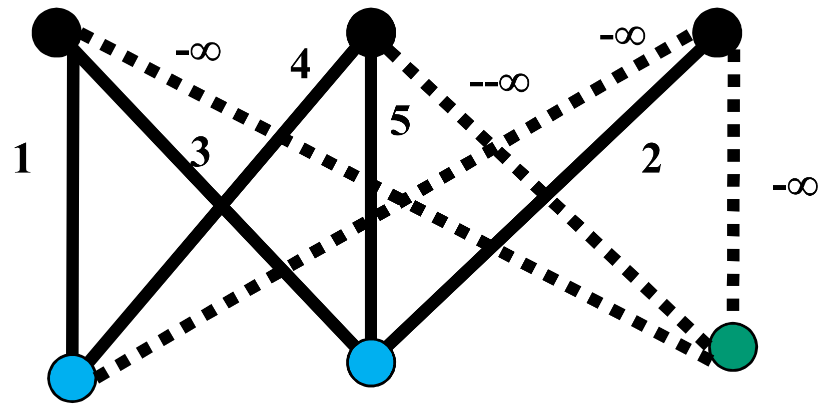
输出：G上的最大权值二分匹配M



最大权值二分匹配问题

输入：加权完全二分图$G=(X \cup Y, E)$，$w: E \rightarrow R$

输出：G上的最大权值二分匹配M

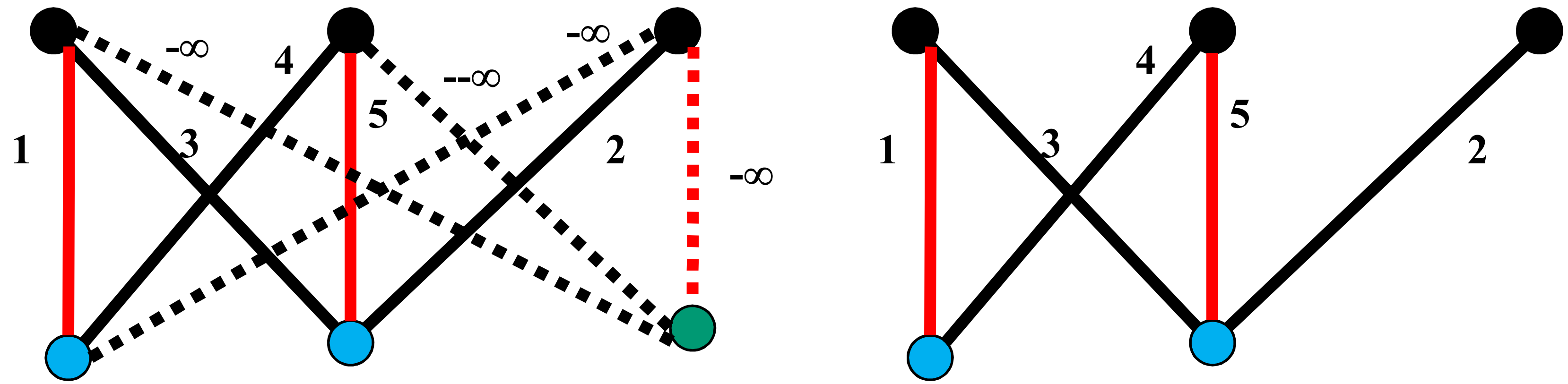


最大权值二分匹配问题

输入：加权完全二分图***Kn,n***，***w***:***E***→***R***

输出：***G***上的最大权值二分匹配***M***

HIT
CS&E

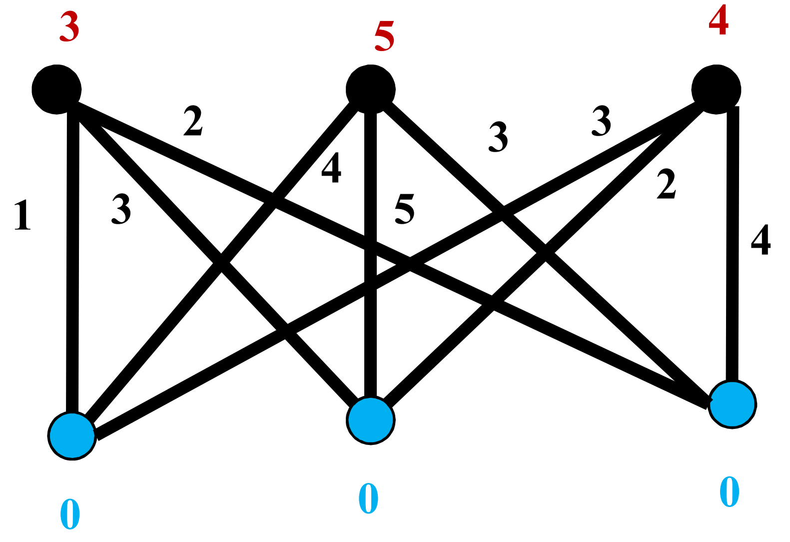


最大权值二分匹配问题

输入：加权完全二分图$K_{n,n}$，$w: E \to R$

输出：G上的最大权值二分匹配M

最小权值顶点覆盖



加权$K_{n,n}$的顶点覆盖

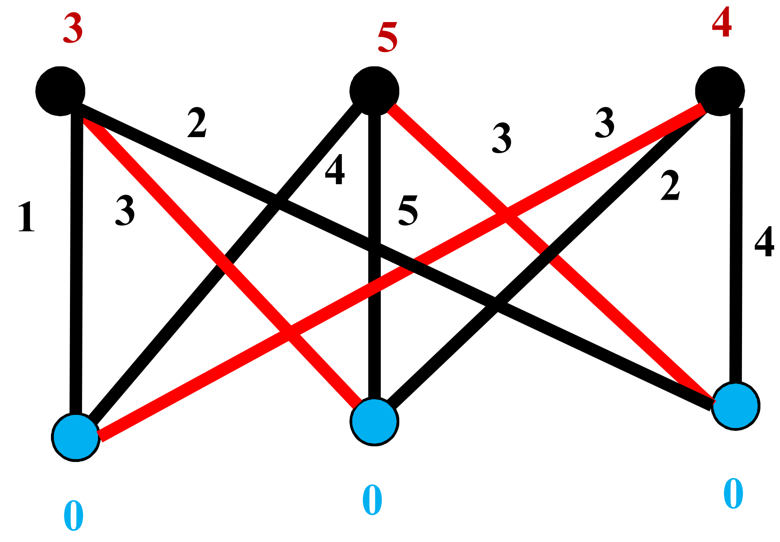
- 两个n维实值向量u,v
- $u_i+v_j \ge w_{ij}$

最小权值顶点覆盖问题

输入：加权二分图$K_{n,n}$，$w:E\to R$

输出：分量之和达到最小值的顶点覆盖

对偶关系



任意完美匹配M，任意覆盖u,v

$$u_i+v_j \geq w_{ij}$$

$$\Sigma u_i + \Sigma v_j \geq \Sigma w_{ij}$$

$$c(u,v) \geq c(M)$$

定理：在加权$K_{n,n}$上，任意完美匹配M和任意顶点覆盖u,v比满足$c(M) \leq c(u,v)$，并且M是最大权值匹配当且仅当$c(M)=c(u,v)$，此时M中的每条边ij均满足$u_i+v_j=w_{ij}$

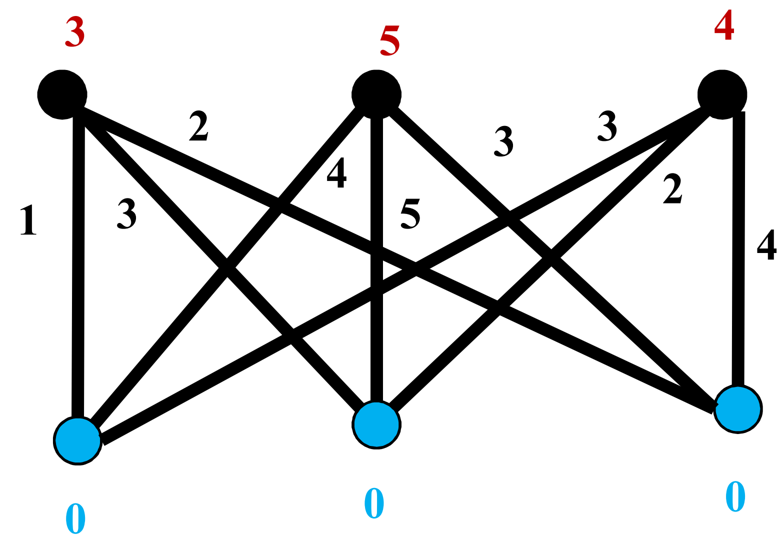
对偶关系给出了问题的求解思路

初始化顶点覆盖u,v

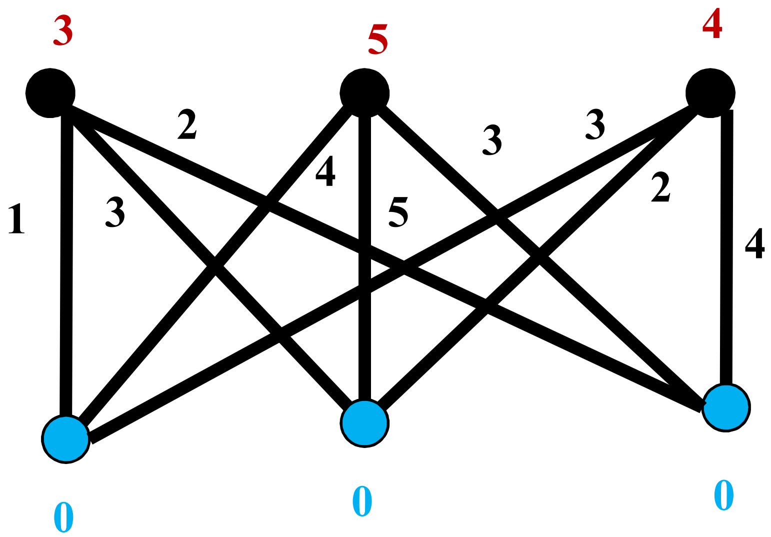
判断$\{ij: u_i+v_j=w_{ij}\}$中能否找出完美匹配M

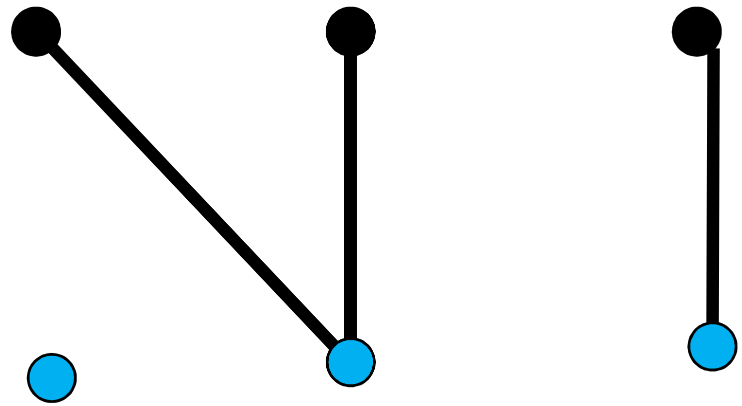
若是，则M是最大匹配，u,v是最小覆盖

否则，调整u,v



初始化





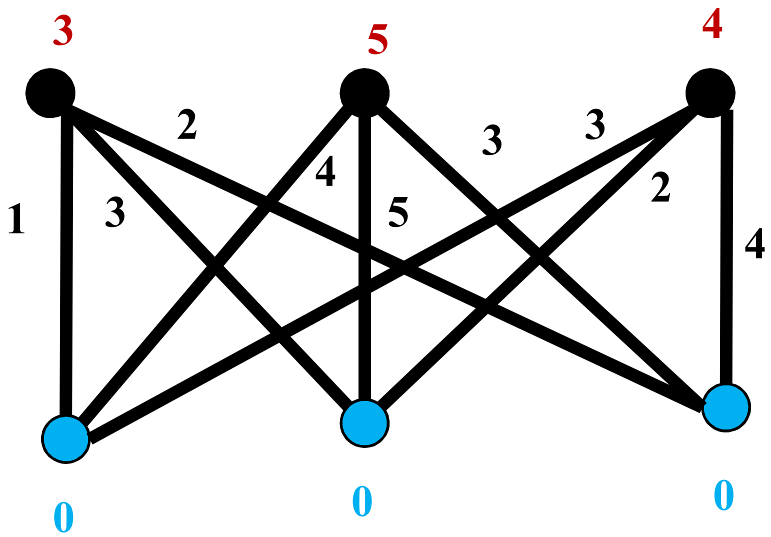
$ui \leftarrow \max\{wij\}$

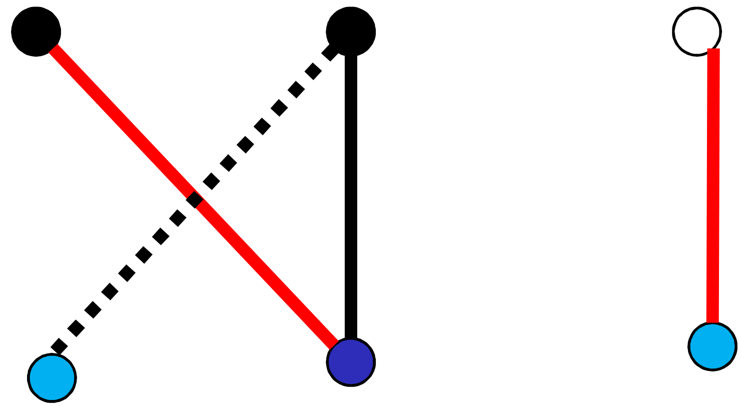
$vj \leftarrow 0$

相等子图

仅含$ui+vj= wij$的所有边

初始化





红边：最大匹配

深蓝、白顶点：最小覆盖

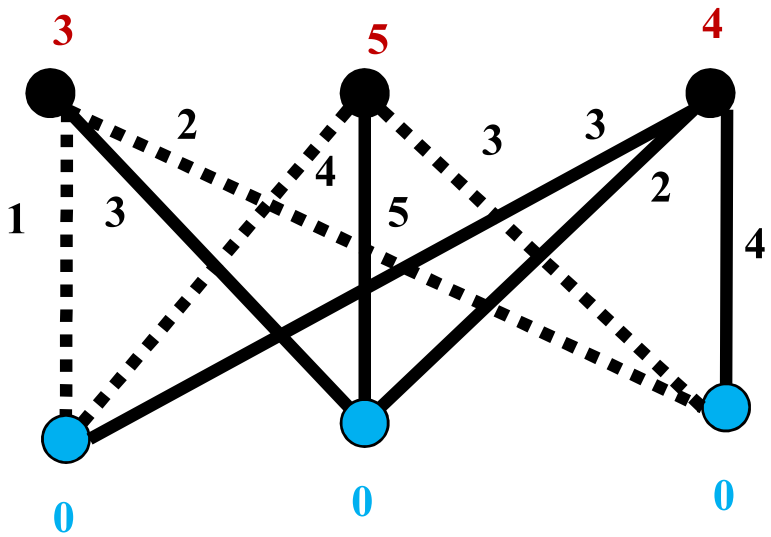
$u_i \leftarrow \max\{w_{ij}\}$

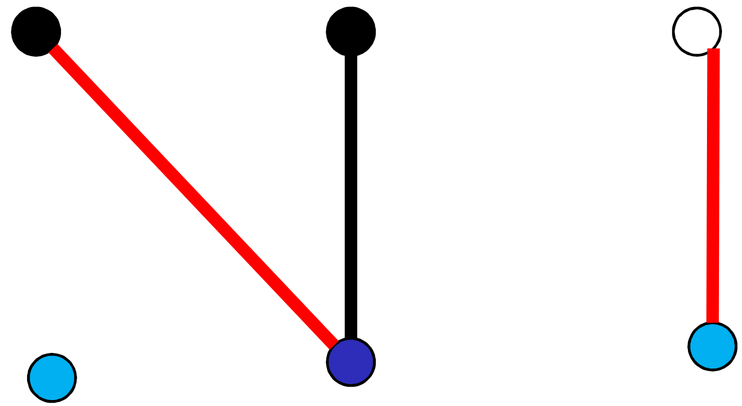
$v_j \leftarrow 0$

引进端点不在最小覆盖中的边，构造新的相等子图

调整覆盖u,v

ε=1



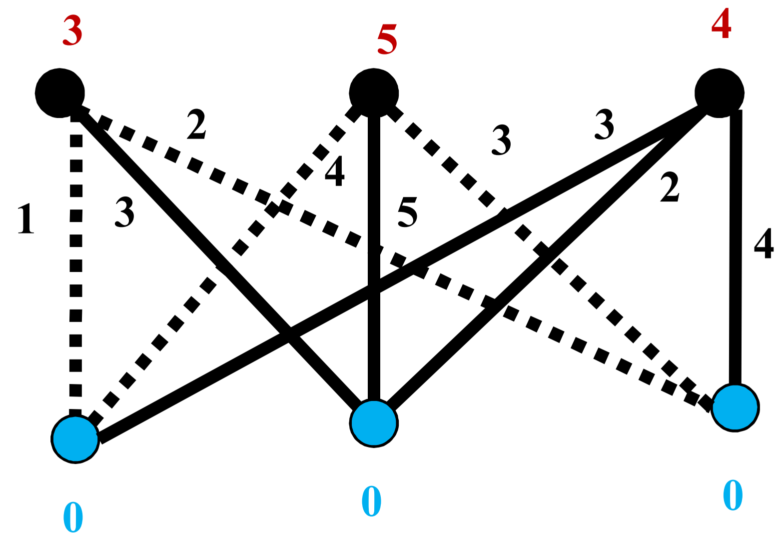


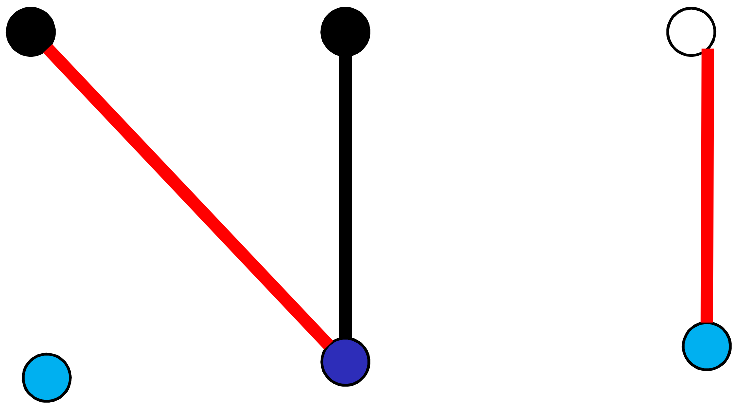
ε←min{*ui*+*vj*-*wij*}　*i,j*不在当前最小覆盖中

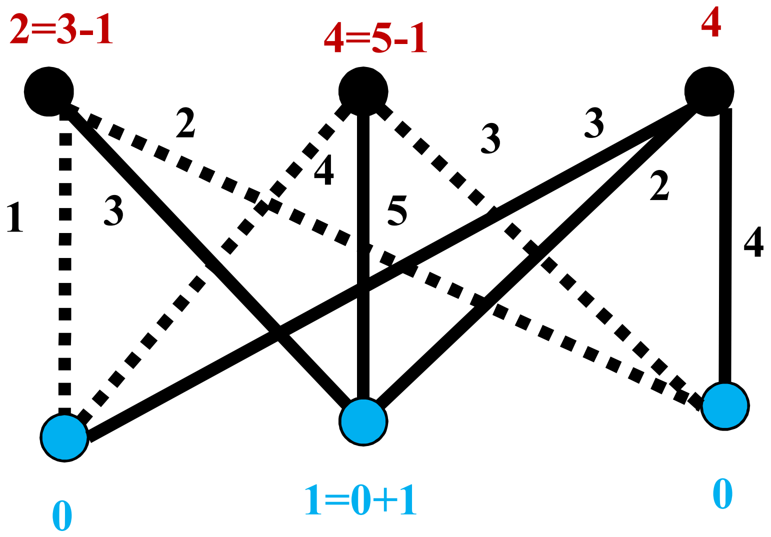
ui←*ui*-ε　*i*不在当前最小覆盖中

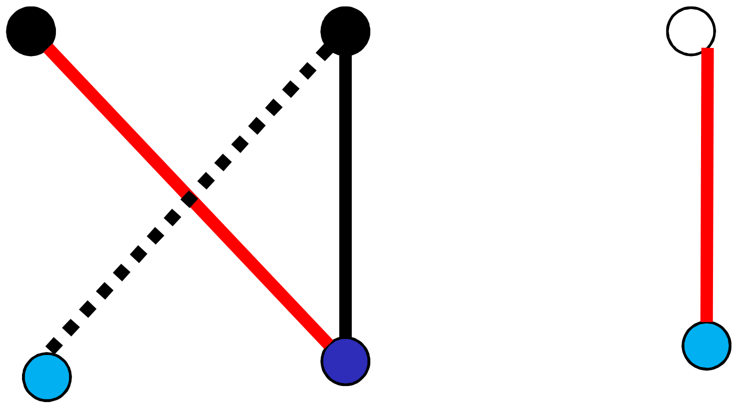
vj←*vj*+ε　*j*在当前最小覆盖中

调整覆盖u,v



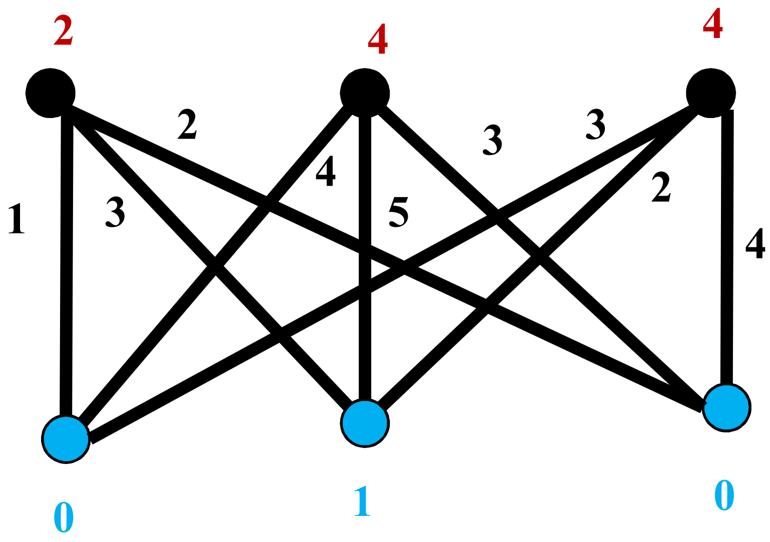


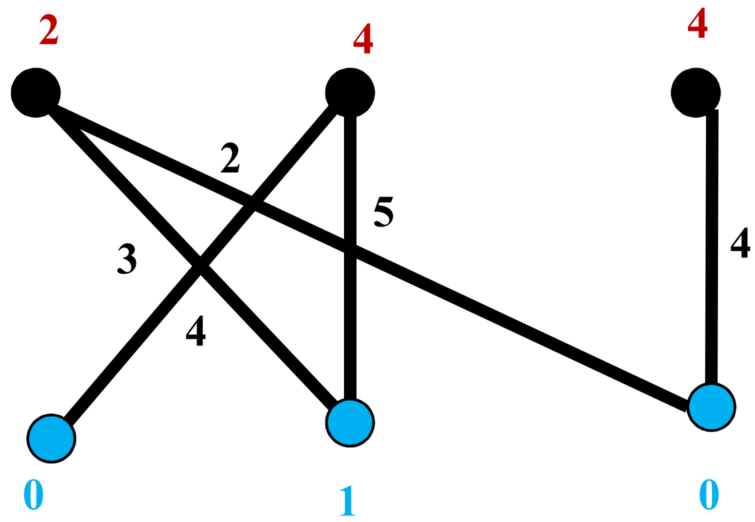




HIT
CS&E

$\varepsilon=1$

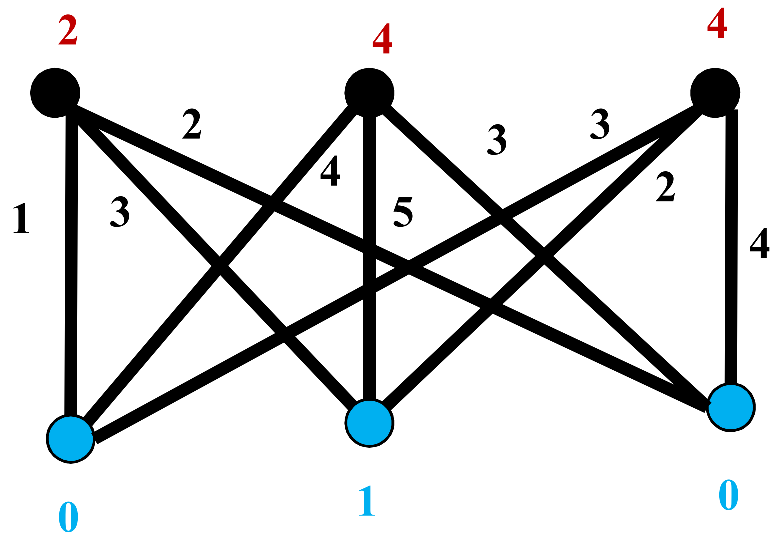


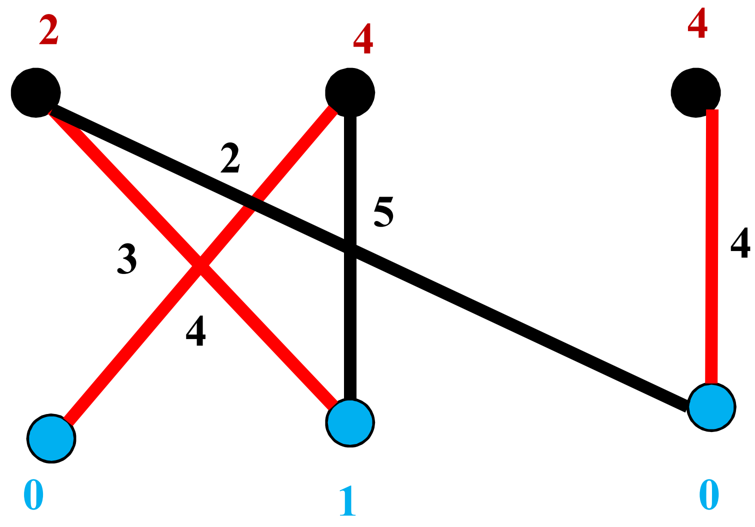


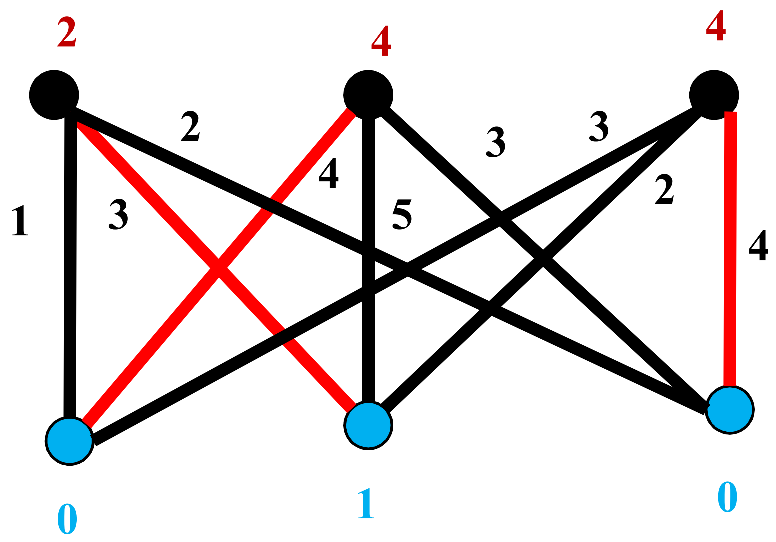
$\varepsilon \leftarrow \min\{u_i+v_j-w_{ij}\}$ i,j不在当前最小覆盖中

$u_i \leftarrow u_i-\varepsilon$ i不在当前最小覆盖中

$v_j \leftarrow v_j+\varepsilon$ j在当前最小覆盖中


2
4
4
2
1
3
4
5
3
3
2
4
0
1
0


2
4
4
2
5
4
3
4
0
1
0


2
4
4
2
1
3
4
5
3
3
2
4
0
1
0

加权最大二分匹配算法

输入：加权完全二分图$K_{n,n}$, 边ij的权值为w_{ij}

输出：$K_{n,n}$加权最大匹配和最小覆盖

时间复杂度$O(n^5)$

1.For $i \leftarrow 1$ To n

2. $u_i \leftarrow \max \{w_{ij}\}$; $v_i \leftarrow 0$;

3.While true do

至多循环n^2次

4. 构建由$\{ij: u_i+v_j=w_{ij}\}$构成的相等子图$G_{u,v}$;

5. 找到$G_{u,v}$的最大匹配M和最小顶点覆盖C;

6. If $|M|=n$ Then 输出M，算法结束

7. Else

8. $\varepsilon \leftarrow \min\{u_i+v_j-w_{ij}:\ x_i,y_j$不属于C$\}$

9. $u_i \leftarrow u_i-\varepsilon$ 任意x_i不属于C

10. $v_j \leftarrow v_j+\varepsilon$ 任意y_j属于C

$O(n^3)$

循环不变量：u,v是加权顶点覆盖

第3-10步每执行n遍，$|M|$至少增大1