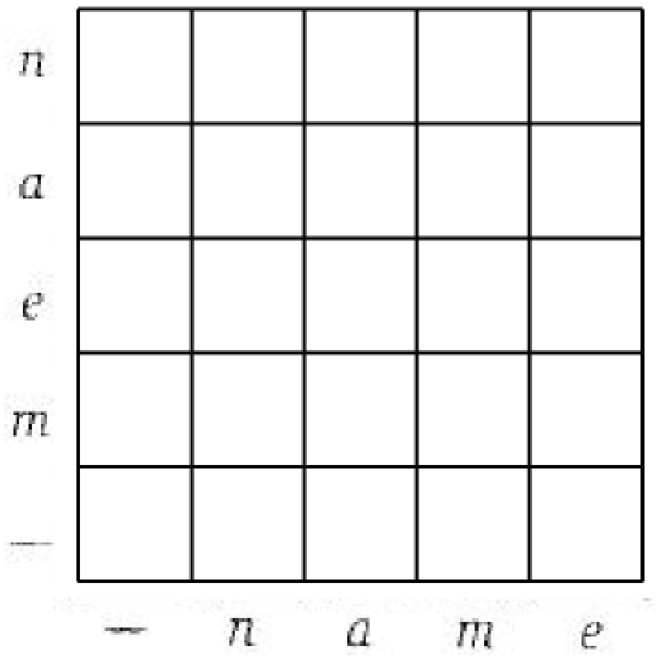
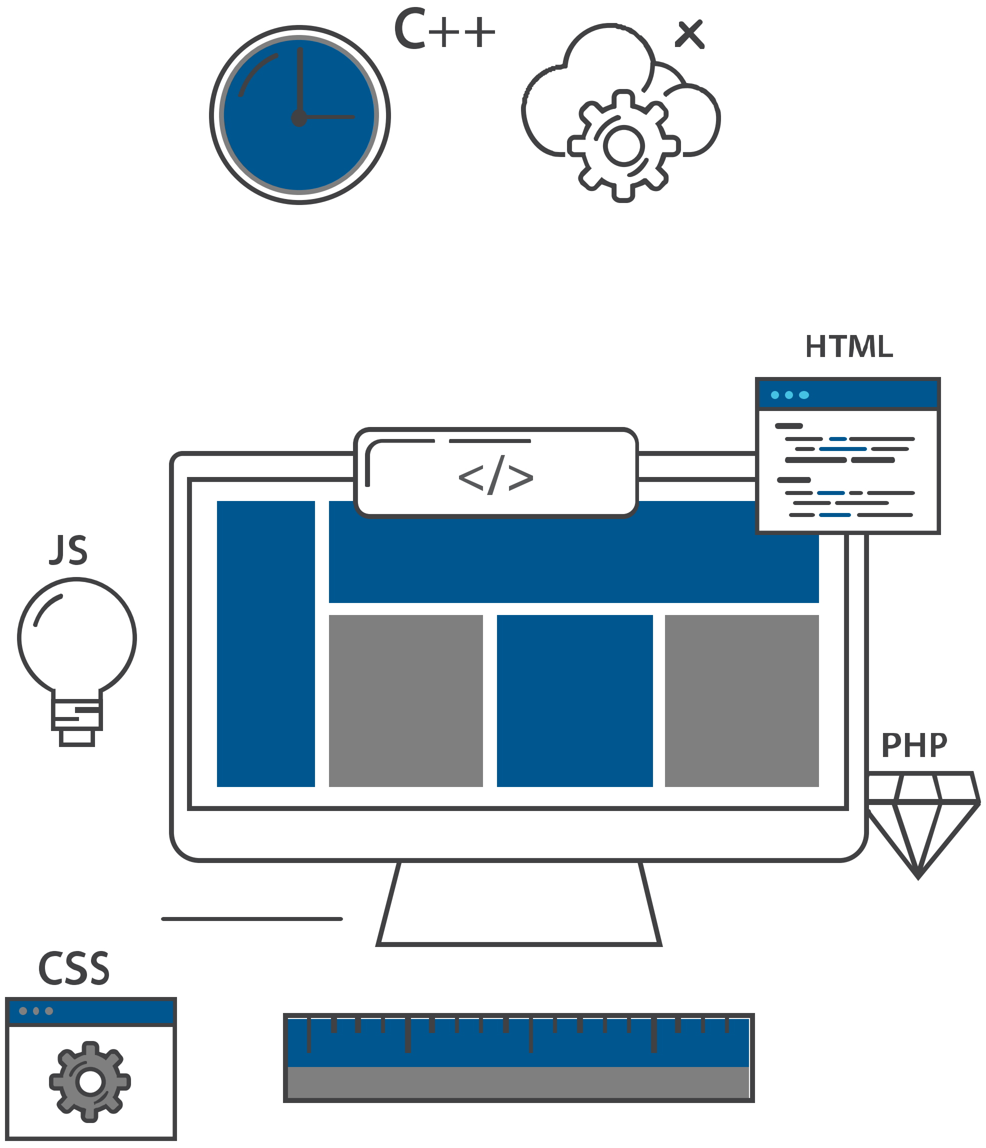
- 建立一个二维方格图G_{XY}，用来表示状态之间的转换。
- 例子：间隙罚分$\delta = 3$，元音与不同元音配对，辅音与不同辅音配对不匹配罚分1；元音与辅音配对罚分2.



第七章 网络流

李 翔

https://implus.github.io/



2

网络流

- 一个二部图G=(V,E)是一个无向图，结点集合可以划分成V=X∪Y，每条边e有一个端点在X中，一个端点在Y中。

- 图G=(V,E)中的一个匹配是边的集合M⊆E，并且每个结点至多出现在M中的一条边上。

网络流

- **组合算法**中最古老的问题：确定二部图中最大匹配的大小

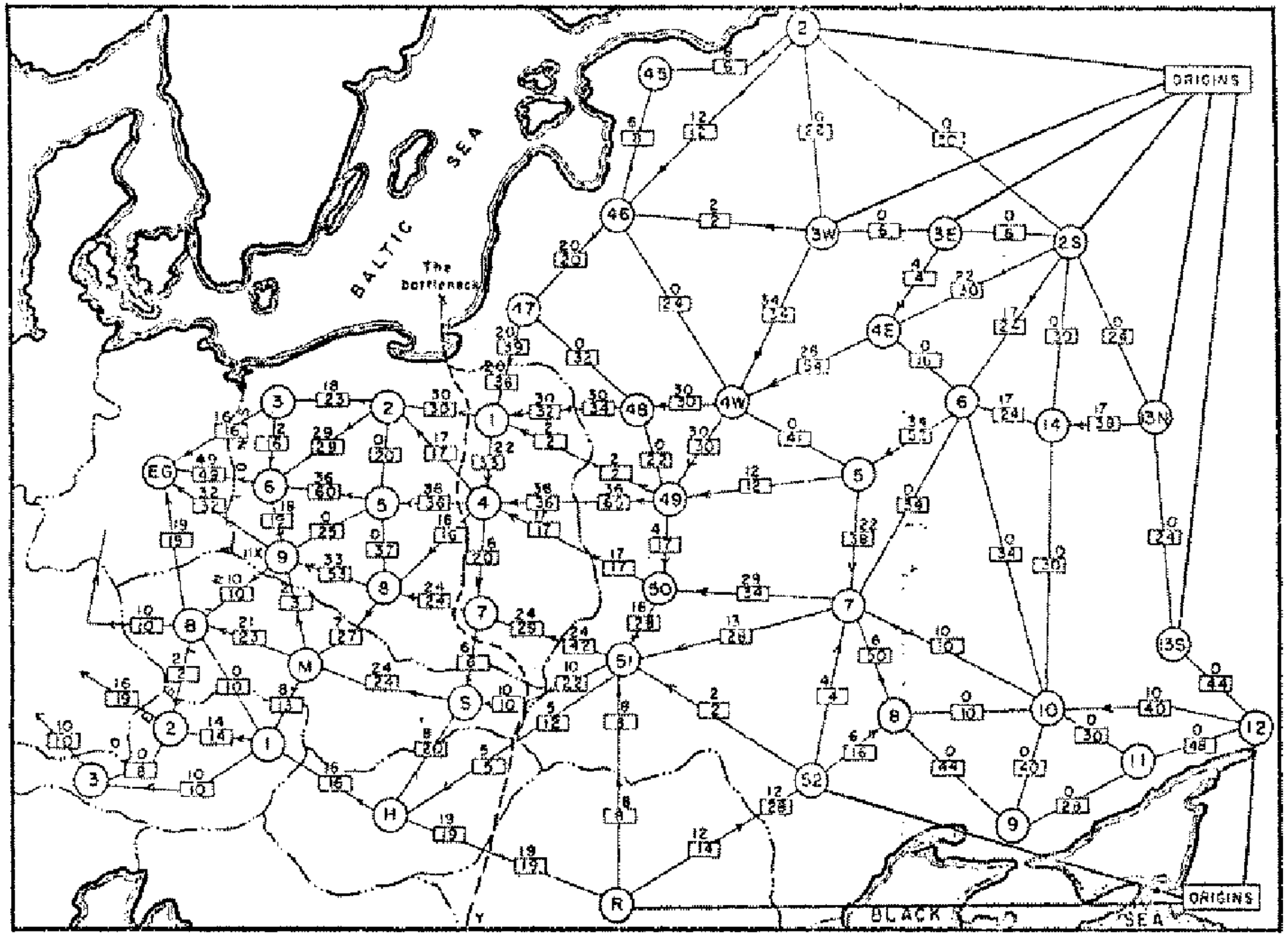
- 需要新的思路：多项式时间的算法

- 本章中介绍一类一般化的问题-网络流问题，对一般性的问题，即最大流问题，开发一个多项式时间的算法，然后说明其一般性应用

网络流

- 研究网络流问题的**原始动力**来自于网络的交通问题
- 当前解密的美国空军报告：揭示了在最小割应用的初始动机中，网络是一张前苏联的铁路线地图，目标是尽可能高效的破坏铁路运输

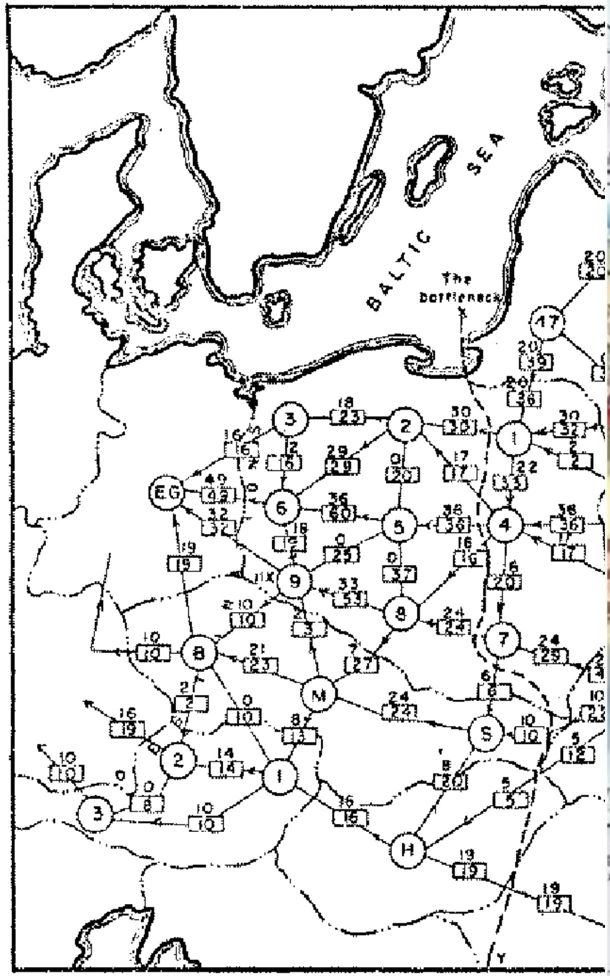
网络流



Soviet Rail Network, 1955

On the history of the transportation and maximum flow problems.

网络流



Soviet Rail Network

网络流

- 应用场景
  - ✓ Airline scheduling.
  - ✓ Bipartite matching.
  - ✓ Baseball elimination.
  - ✓ Image segmentation.
  - ✓ Network connectivity.
  - ✓ Network reliability.
  - ✓ …

7.1 最大流问题

- 用图对交通网络建模
  - ✓ 比如*公路系统*：边是公路，结点交叉路口
  - ✓ 比如*计算机网络*：边是链接线，结点是开关
  - ✓ 比如*管网*：边是输送些体的管道，结点是管道连接点
- 抽象出来的要素：

  边上的容量；

  源点；终点；交通量通过边运送

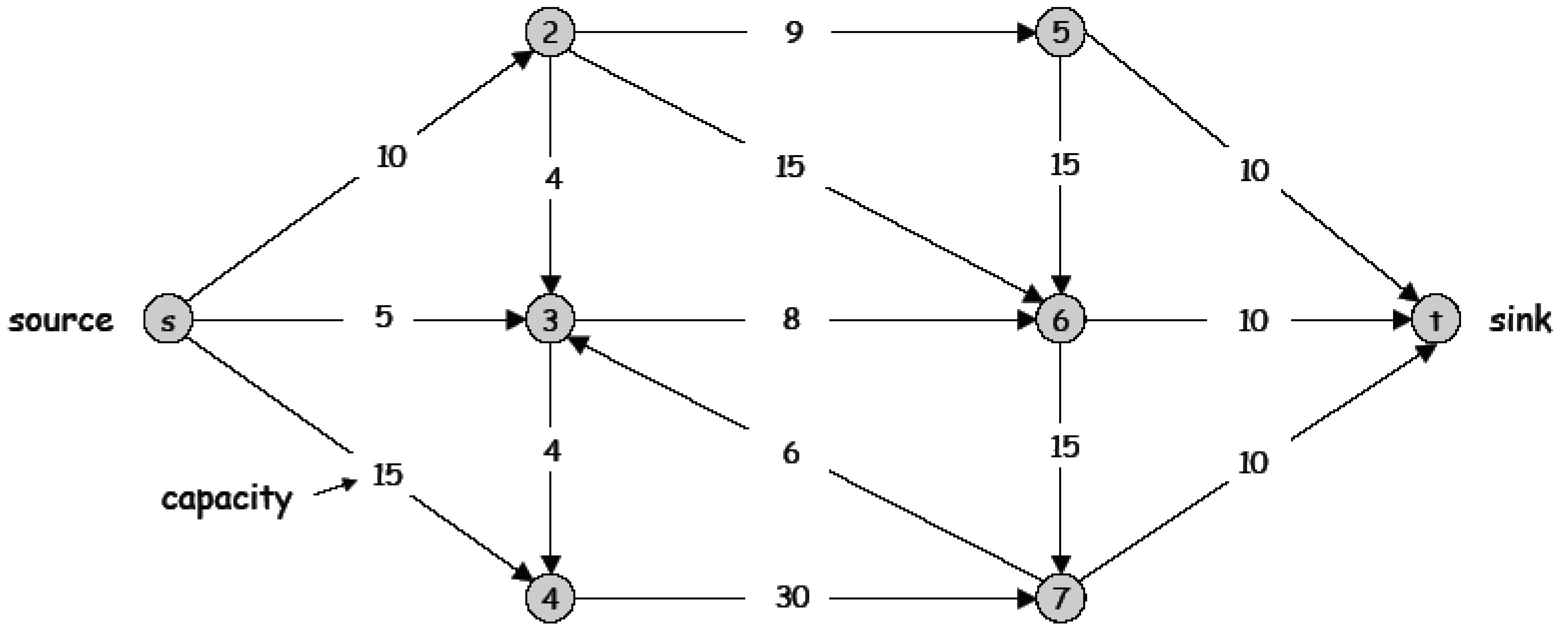
最大流问题

流网络

有向图G=(V,E)

每条边关联一个容量，非负数Ce.

存在单一源点s, 以及单一汇点t



最大流问题

- 流的定义：
- s-t流是一个函数f,它把每条边e映射到一个非负实数f: $E \to R^+$,值f(e)表示由边e携带的流量，一个流f必须满足下面两个性质：

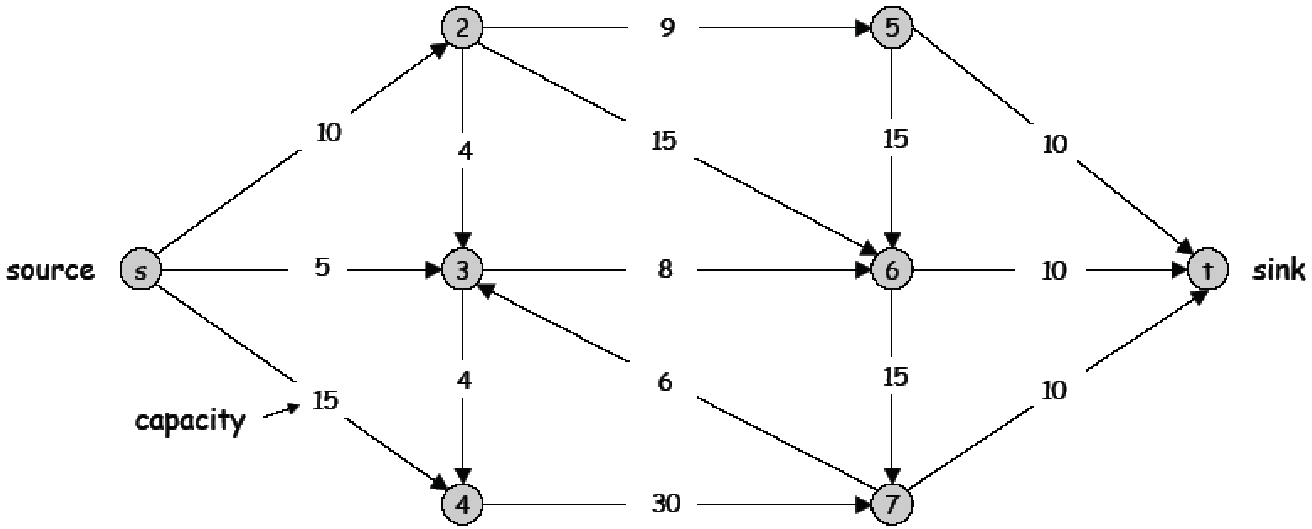
1. (容量条件)0<=f(e)<=C_e
2. (守恒条件)除了s,t外，对每个结点v,满足

$$\sum_{e\_in\_v} f(e) = \sum_{e\_out\_v} f(e)$$

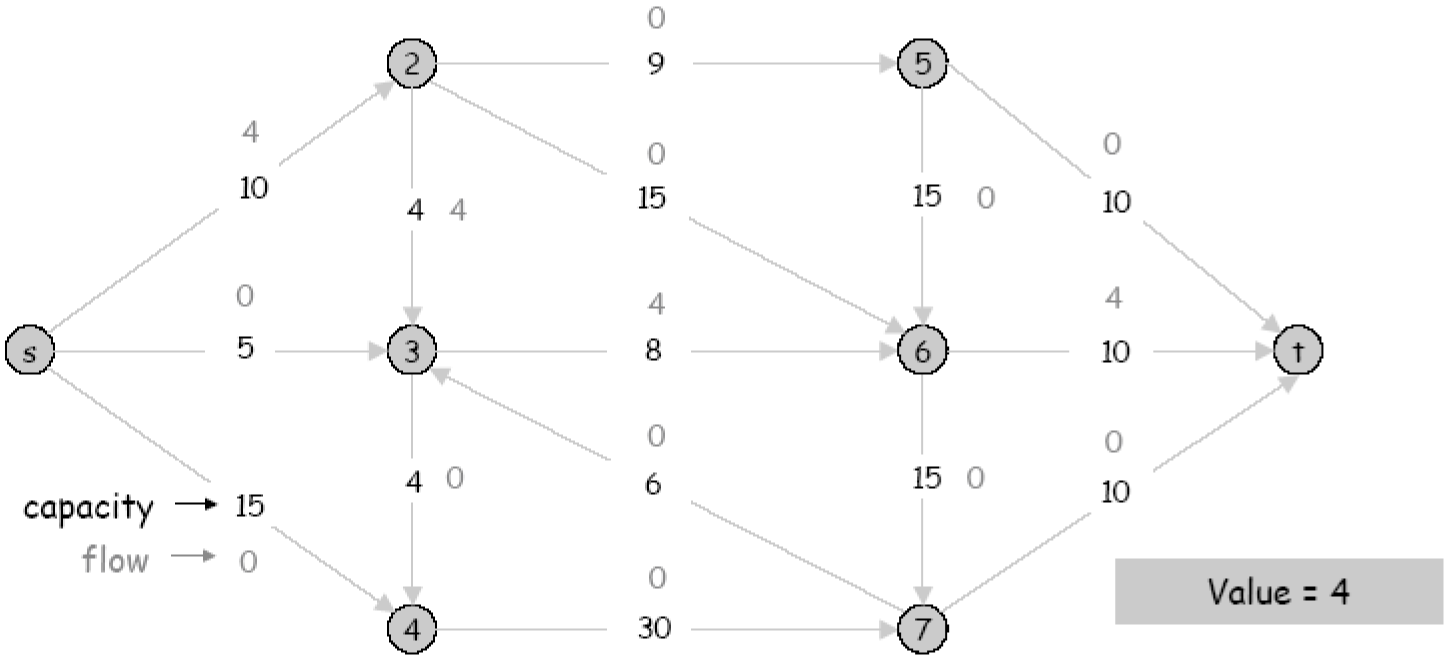
最大流问题

- 定义 $f^{out}(v)=\sum_{e\_out\_v} f(e), f^{in}(v)=\sum_{e\_in\_v} f(e)$
- 我们记 $v(f)=f^{out}(s)$
- 最大流问题：给定一个流网络，自然的目标就是安排交通使得有效容量尽可能得到有效使用：**找出一个具有最大值的流**

最大流问题



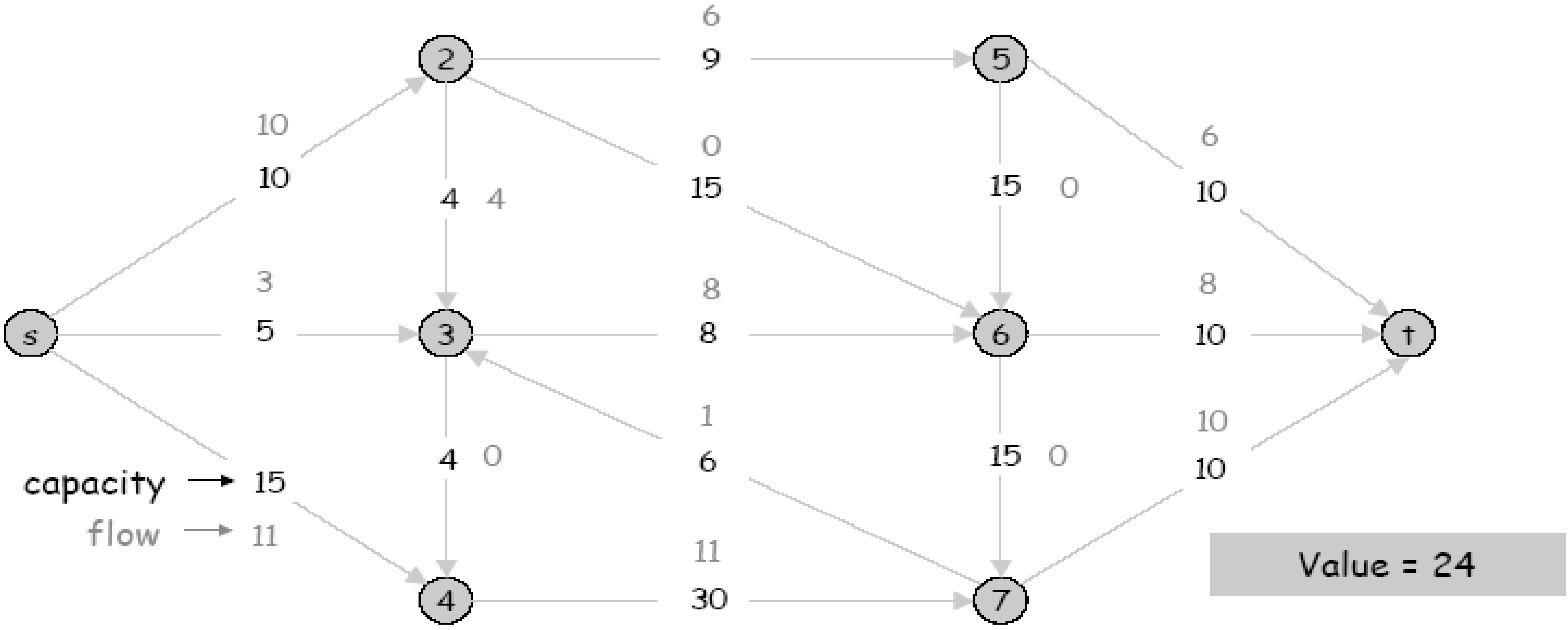
source
sink
capacity
s
2
3
4
5
6
7
t
10
5
15
9
15
4
4
8
6
30
15
15
10
10
10



capacity
flow
Value = 4

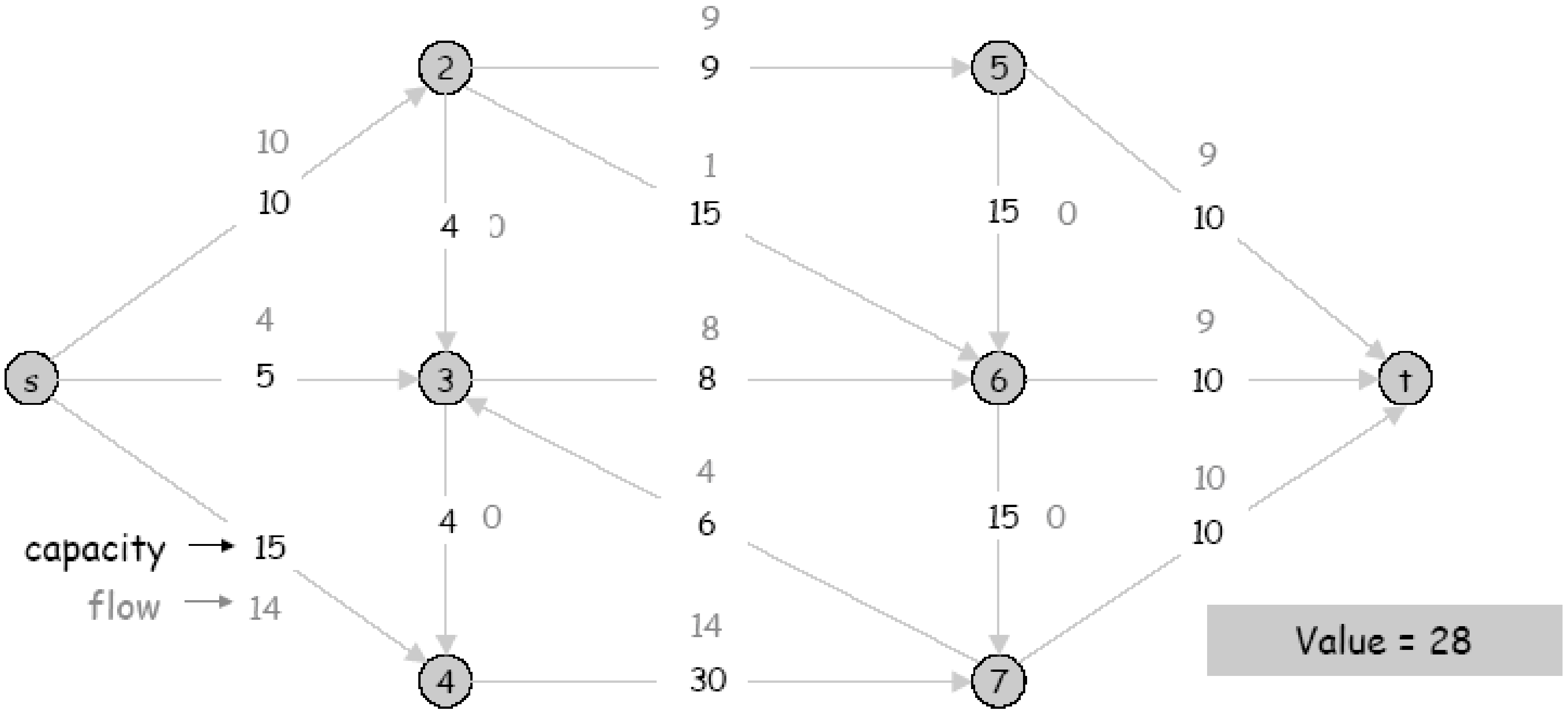
最大流问题

可以找到更大的：



最大流问题

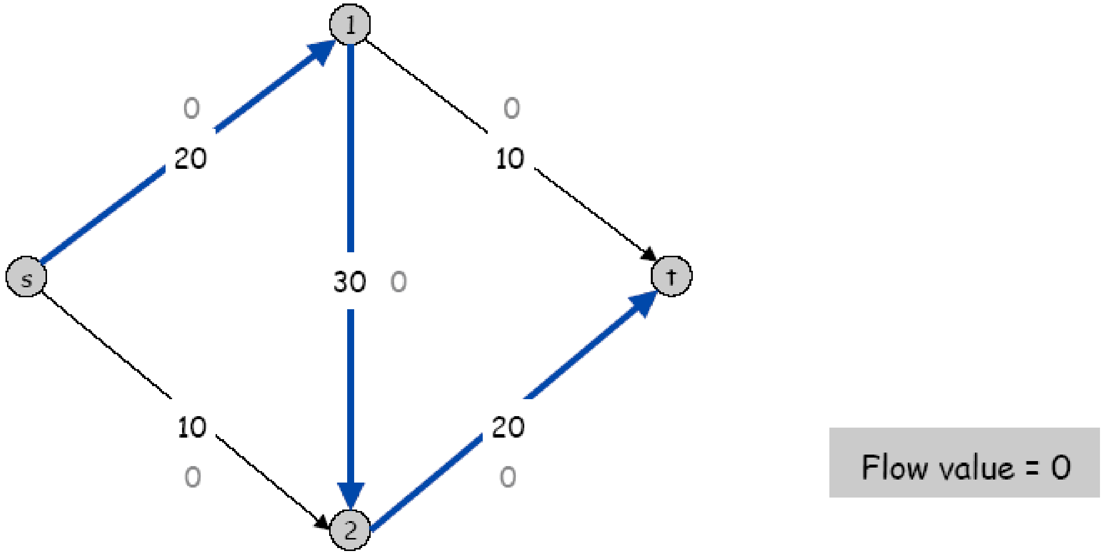
实际上最大的流：



最大流问题

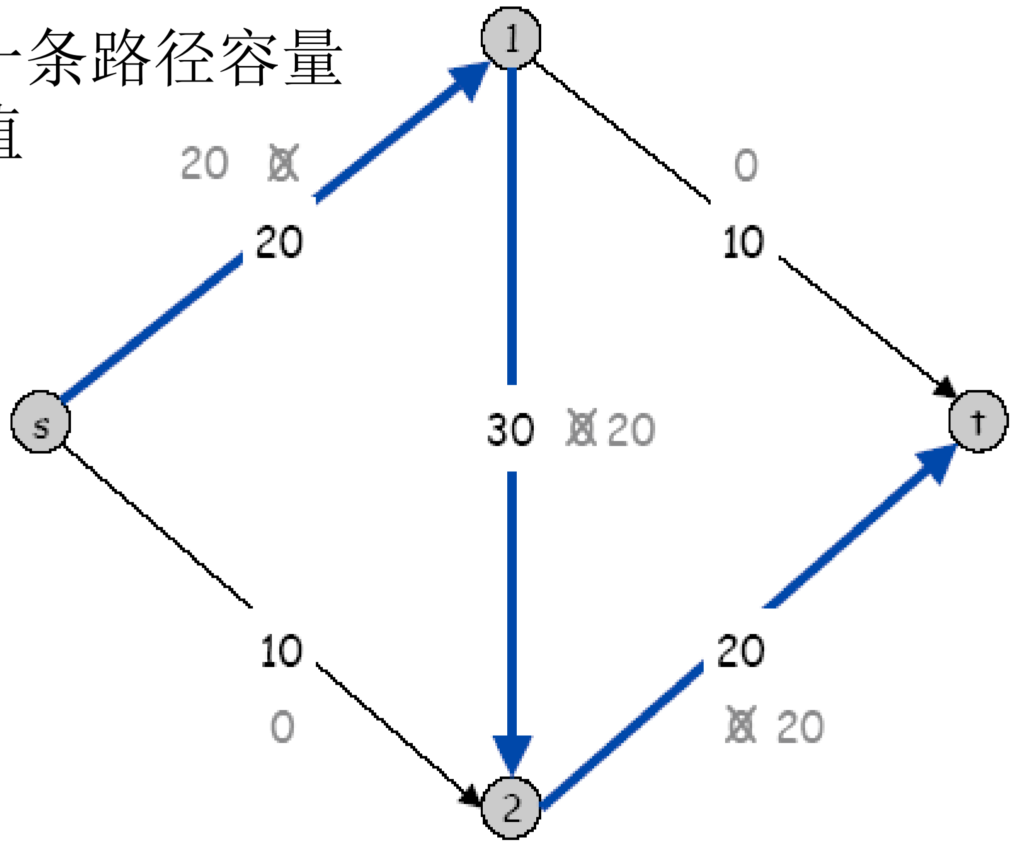
设计算法

- 先从贪心算法开始：对所有的e,f(e)=0.
- 现在，沿着一条从s到t的路径通过“推”这个流来增加f的值，**最大到边容量的限度。**



最大流问题

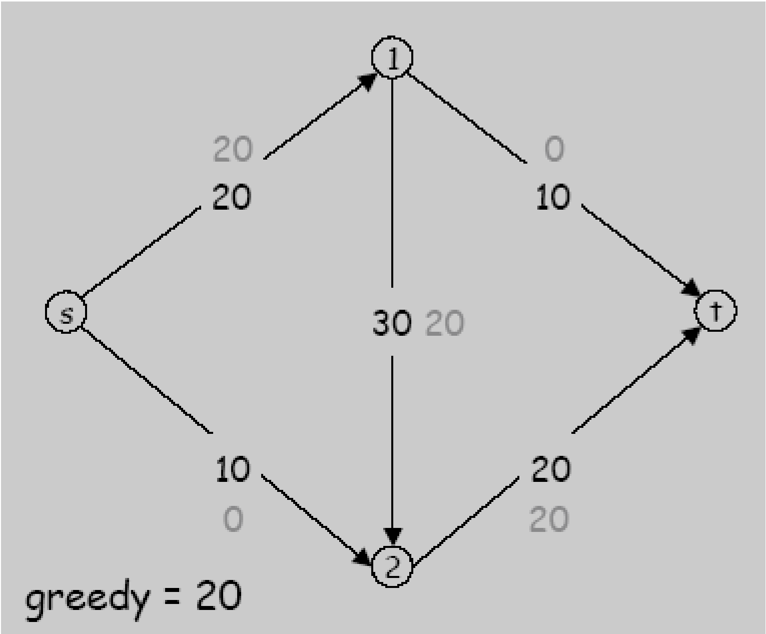
达到一条路径容量最大值

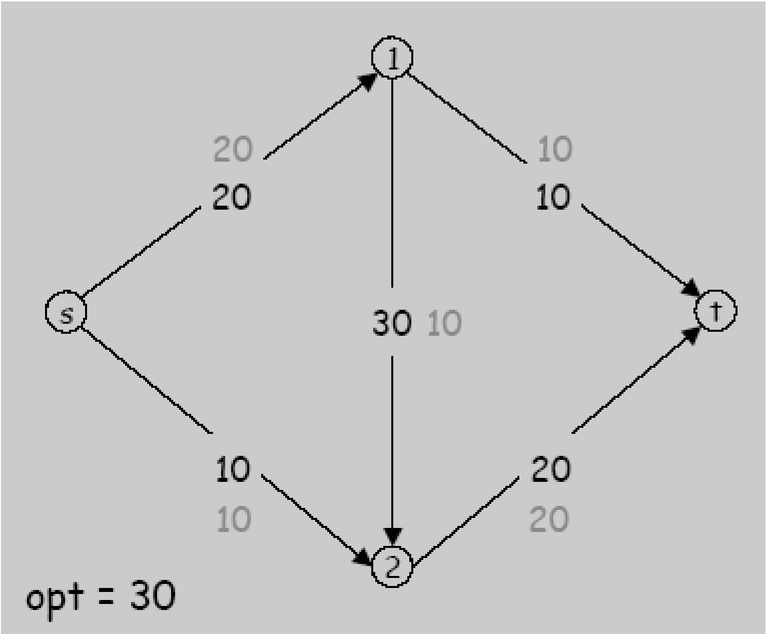


Flow value = 20

最大流问题

- 但是我们很快发现，局部最优不等于全局最优！





最大流问题

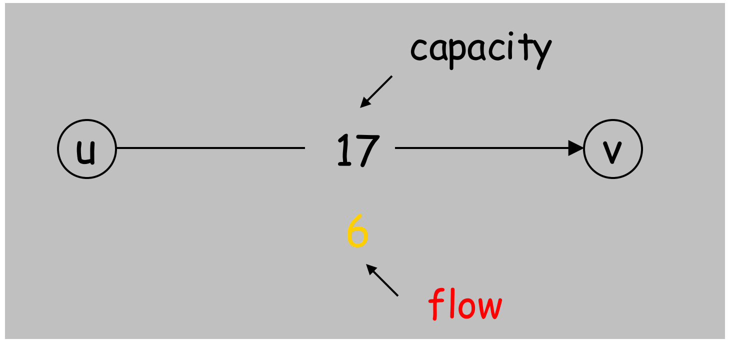
- 解决问题的思路

  --分解
- 流的性质
- 我们可以在边上用剩余的容量向前推，并且我们可以在已经有流的边上向后推，使它转向一个不同的方向。
- 下面将引出**剩余图**的概念

最大流问题

- 原始边

e = (u, v) ∈ E，流 f(e), 容量 c(e).

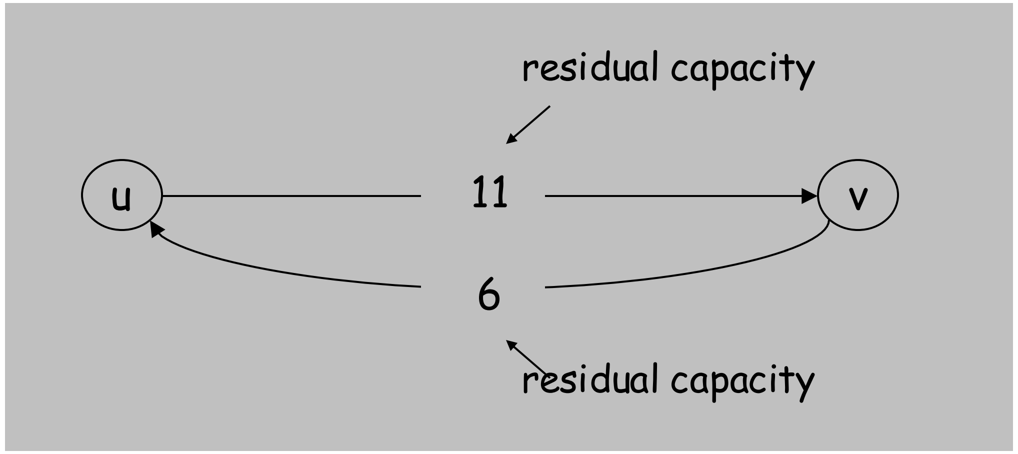


- **剩余边**

  - e = (u, v) and e^R = (v, u).
  - 剩余容量:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{for } e \\ f(e) & \text{for } e^R \end{cases}$$

最大流问题



- 剩余图： $G_f = (V, E_f)$.
  - 具有正的剩余容量的剩余边.
  - $E_f = \{e\} \cup \{e^R\}$.

最大流问题

- 对G的每条边e=(u,v),其中f(e) < c(e),那么存在c(e)-f(e)的剩余的容量，我们还可以尝试在这个容量**往前推**，于是G_f 中包含这条边e,容量为c(e)-f(e)，称为前向边。

- 对G的每条边e=(u,v),其中f(e)>0,我们可以通过**向后推**这个流来“撤销”它，于是G_f 中包含边e'=(v,u),容量是f(e),称为后向边。

最大流问题

在剩余图中的**增广路径**

- 令P是G_f中一条简单的s-t路径。定义bottleneck(P,f)是P上任何边关于流f的 **最小剩余容量**。如下算法augment(f,P)在G中产生一个新的流f′. (f→f′)

```
Augment(f, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E)  f(e) ← f(e) + b       前向边
      else        f(eR) ← f(e) - b      后向边
   }
   return f
}
```

最大流问题

- 通常把剩余图中的任何一条s-t路径认为是一条增广路径。
- 命题7.1 f′是G中的一个流。
- 证明：验证容量条件与守恒条件。

对于前向边：$0 \le f(e) \le f'(e) = f(e) + bottleneck(P, f) \le C_e$

对于后向边：$c_e \ge f(e) \ge f'(e) = f(e) - bottleneck(P, f) \ge 0$

守恒条件：分情形讨论

要证明通过剩余图中的s-t增广路径构造的新流f'是图G中的一个有效流，需验证其满足流的两个条件：容量约束和流量守恒。

步骤1：构造新流f'

- 设增广路径P的瓶颈容量为δ，即路径中边的最小剩余容量。
- 对路径P中的每条正向边e，增加流量δ：$f'(e) = f(e) + \delta$。
- 对路径P中的每条反向边e'（对应原边e），减少原边流量δ：$f'(e) = f(e) - \delta$。

步骤2：验证容量约束

- **正向边**：剩余容量为$c(e) - f(e) \geq \delta$，故$f'(e) = f(e) + \delta \leq c(e)$。
- **反向边**：剩余容量为原边流量$f(e) \geq \delta$，故$f'(e) = f(e) - \delta \geq 0$。

因此，所有边流量均满足$0 \leq f'(e) \leq c(e)$，满足容量约束。

步骤3：验证流量守恒

- **中间节点**：对于增广路径中的中间节点v，进入v的边和离开v的边各调整一次流量，调整量均为δ，故流入增量等于流出增量，保持守恒。
- **源点s**：仅出边流量增加δ，总流出量增加δ。
- **汇点t**：仅入边流量增加δ，总流入量增加δ。

因此，除s和t外，所有节点流量守恒。

综上，f'满足流的定义，故命题7.1得证。

算法设计

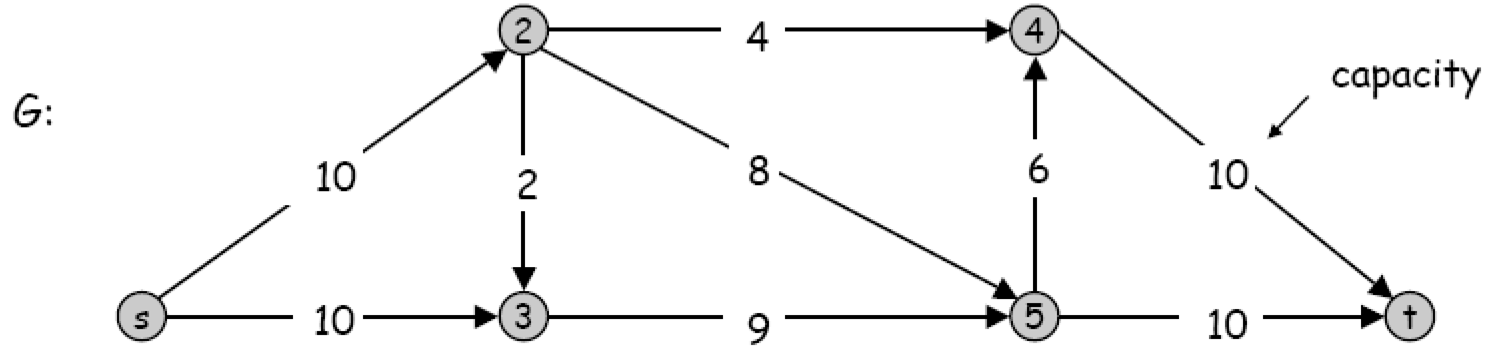
- 增广操作保持了向前和向后流的守恒性
- 直觉上告诉我们，可以不断调整G_f来获取更大的流量。

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

 while (there exists augmenting path P)
 {
      f ← Augment(f, c, P)
      update Gf
 }
   return f
}
```

算法分析

- Demo演示(增广路径、瓶颈容量、剩余网络、f(x,y)最大流)
- 初始图G



https://blog.csdn.net/qq_38662930/article/details/105242945

算法分析

- 1956， 由Ford, Fulkerson开发
- 正确性—确实最大(最大流与最小割)
- 算法复杂度--定量分析while循环在何时终止
- 命题7.2 在Ford-Fulkerson算法的每个中间步，流值{f(e)}和G_f中的剩余容量是整数。

算法分析

- 命题7.3 令f是G中的流，且令P是G_f中的一条简单的s-t路径，那么v(f')=v(f)+bottleneck(P,f);并且由于bottleneck(P,f)>0,我们有v(f')>v(f).

- 证明：P的第一条边e是从剩余图G_f中从s出来的边，边e一定是向前边。我们通过bottleneck(P,f)增加了这条边上的流，且不改变其他的流。

算法分析

- **最大可能的流值：** $v(f) \le C = \sum_{e\_out\_of\_s} c_e$
- 因为有一个上界，我们知道Ford-Fulkerson算法一定会终止
- 定理7.4 如上所述，假设在流网络G中的所有容量都是整数。那么Ford-Fulkerson算法在至多C次While循环的迭代后终止。

算法分析

下面考虑Ford-Fulkerson算法的运行时间。

- n表示G中的结点数，m表示G中的边数，所有的结点至少有一条关联边，于是m>=n/2；
- 算法复杂度应该是？
- 定理7.5 假设在流网络G中的所有容量都是整数，那么Ford-Fulkerson算法可以在O(mC)时间内实现

算法分析

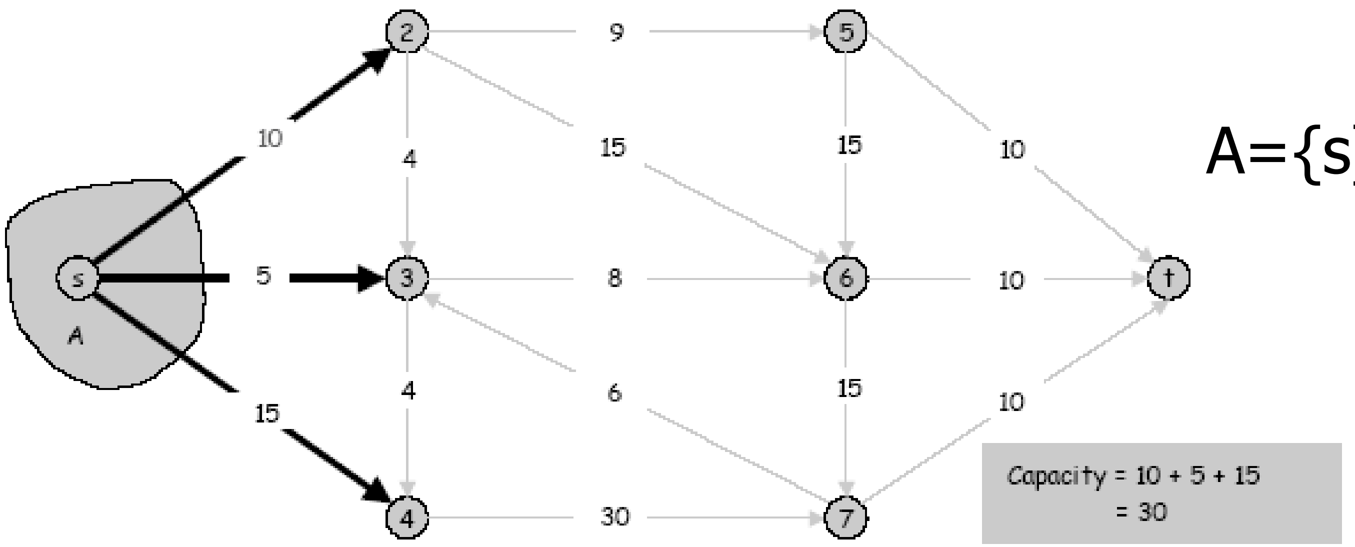
- 证明：

我们知道While循环至多在C次迭代后终止。于是考虑**流调整一次时需要的复杂度**：

- 剩余图至多有2m条边，为找到G_f中一条s-t路径，可以考虑宽度优先或者广度优先搜索，代价为O(m+n)=O(m);
- 因为路径P至多有n-1条边，建立新的剩余图需要O(m)时间。

7.2 网络中的最大流与最小割

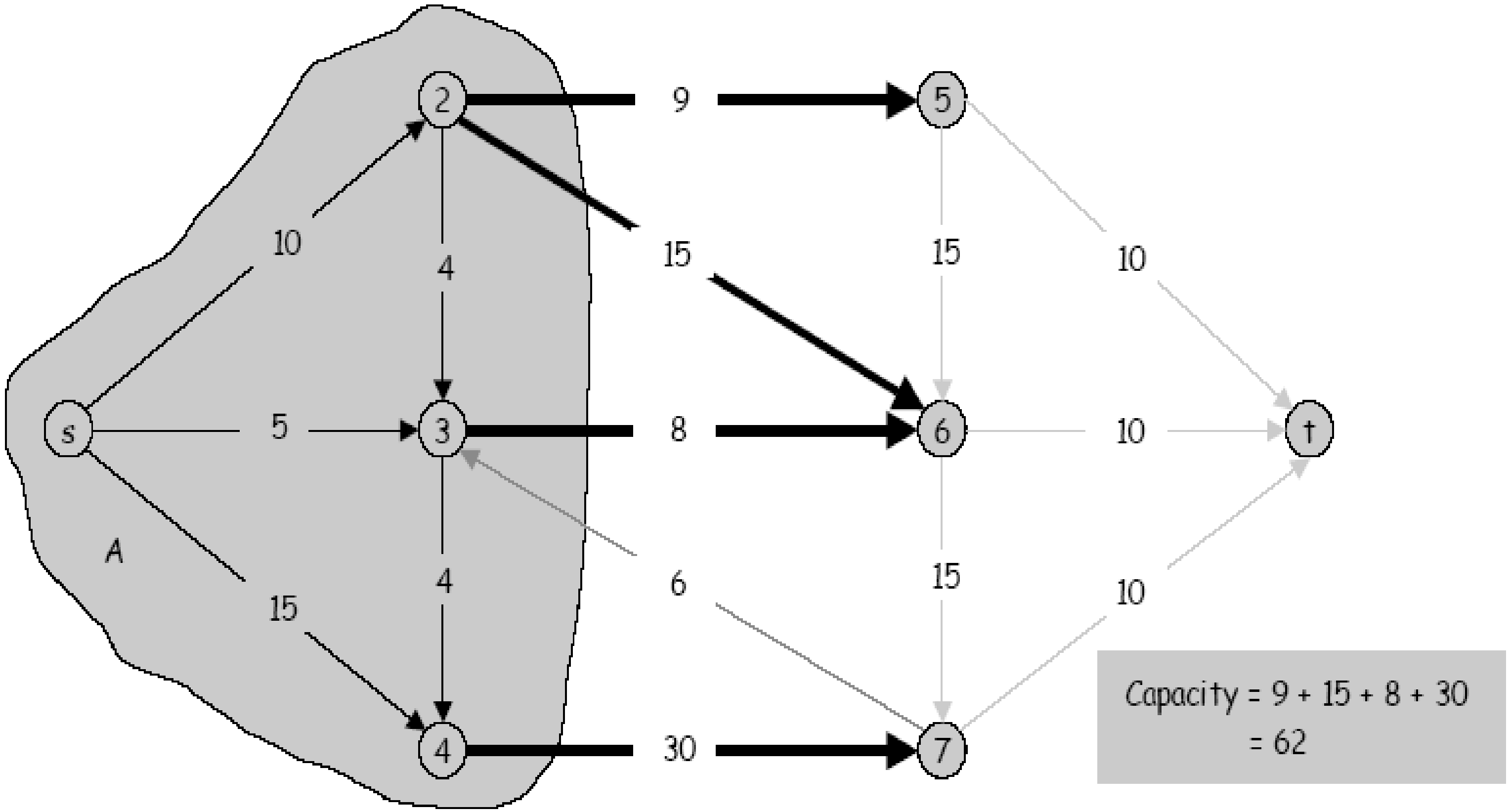
- 我们说一个s-t割是结点集合V的一个划分(A,B)，使得 $s \in A, t \in B$.一个割(A,B)的容量记为c(A,B). 也就是从A**出来的所有边的容量之和。**



A={s}

最大流与最小割

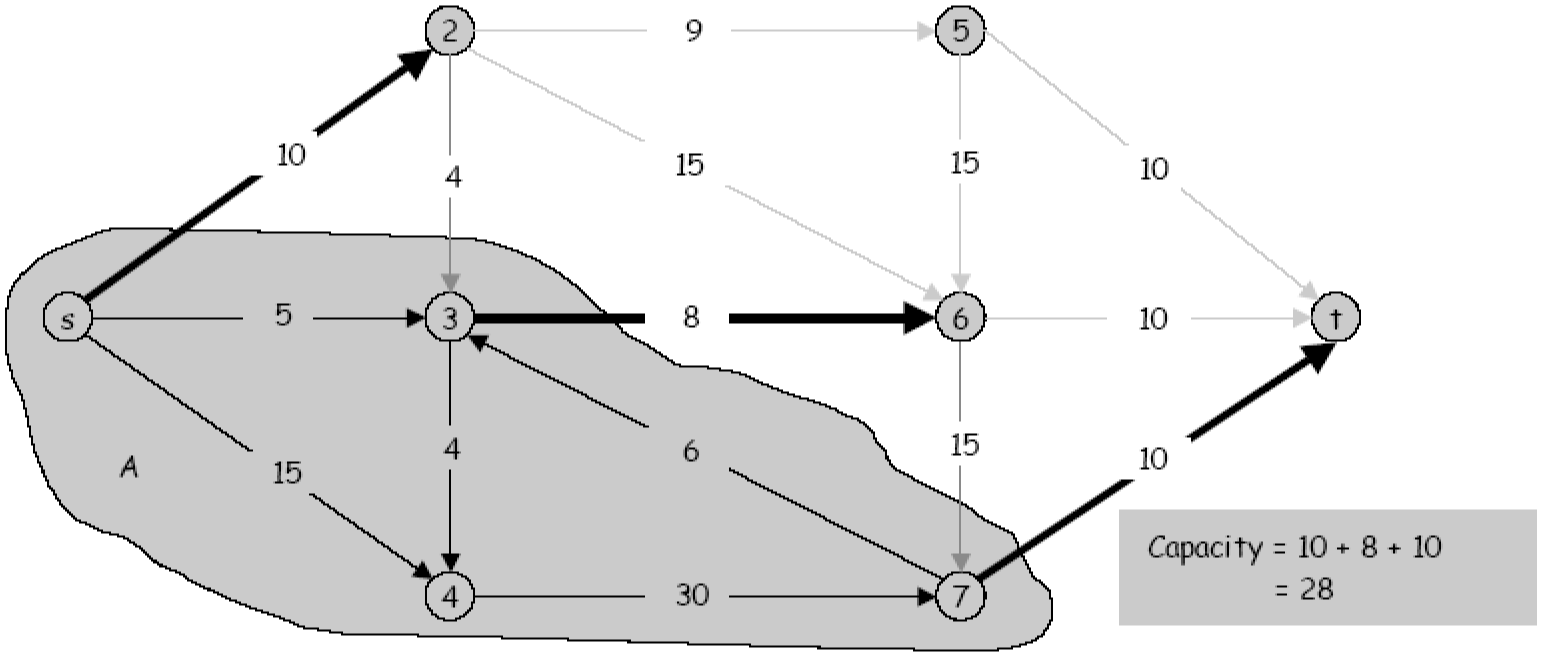
另外一种割的划分：A={s,2,3,4}



最大流与最小割

最小s-t割问题:

- 寻找一个最小容量的 s-t 割.



最大流与最小割

- 割原来提供了流值上的非常自然的上界
- 定理7.6 令f是任何s-t流，且(A,B)是任意s-t割，那么

  $v(f)=f^{out}(A)-f^{in}(A).$

- 证明：因为**源点s**没有边进入，所以 $v(f)=f^{out}(s)-f^{in}(s)$

  此外其他v是**内点**，所以 $v(f)=\sum_{v\in A}(f^{out}(v)-f^{in}(v))$

  注意到 $\sum_{v\in A}(f^{out}(v)-f^{in}(v))=\sum_{e\_out\_A}f(e)-\sum_{e\_in\_A}f(e)=f^{out}(A)-f^{in}(A)$

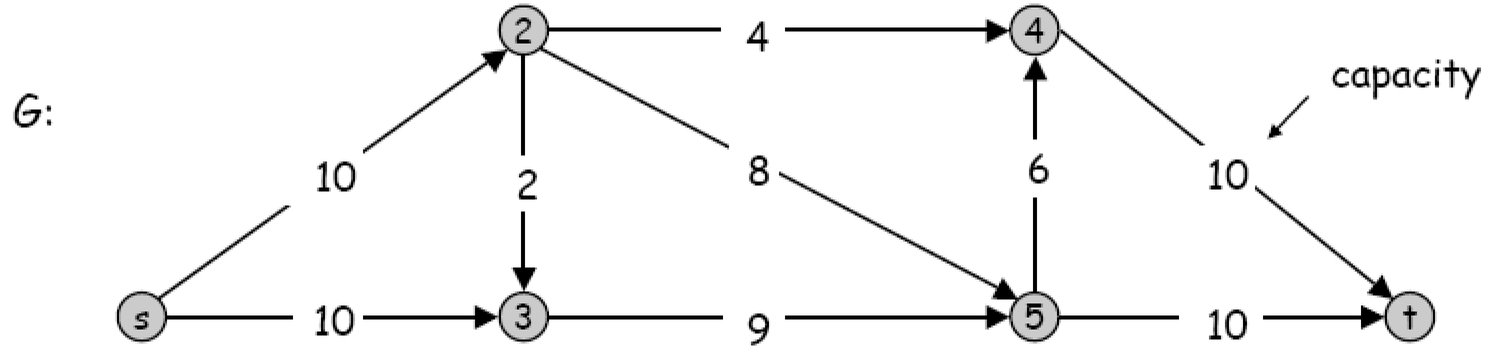
最大流与最小割

- 命题7.7 令f是任意s-t流，且(A,B)是任意s-t割，那么

$$v(f) = f^{in}(B) - f^{out}(B)$$

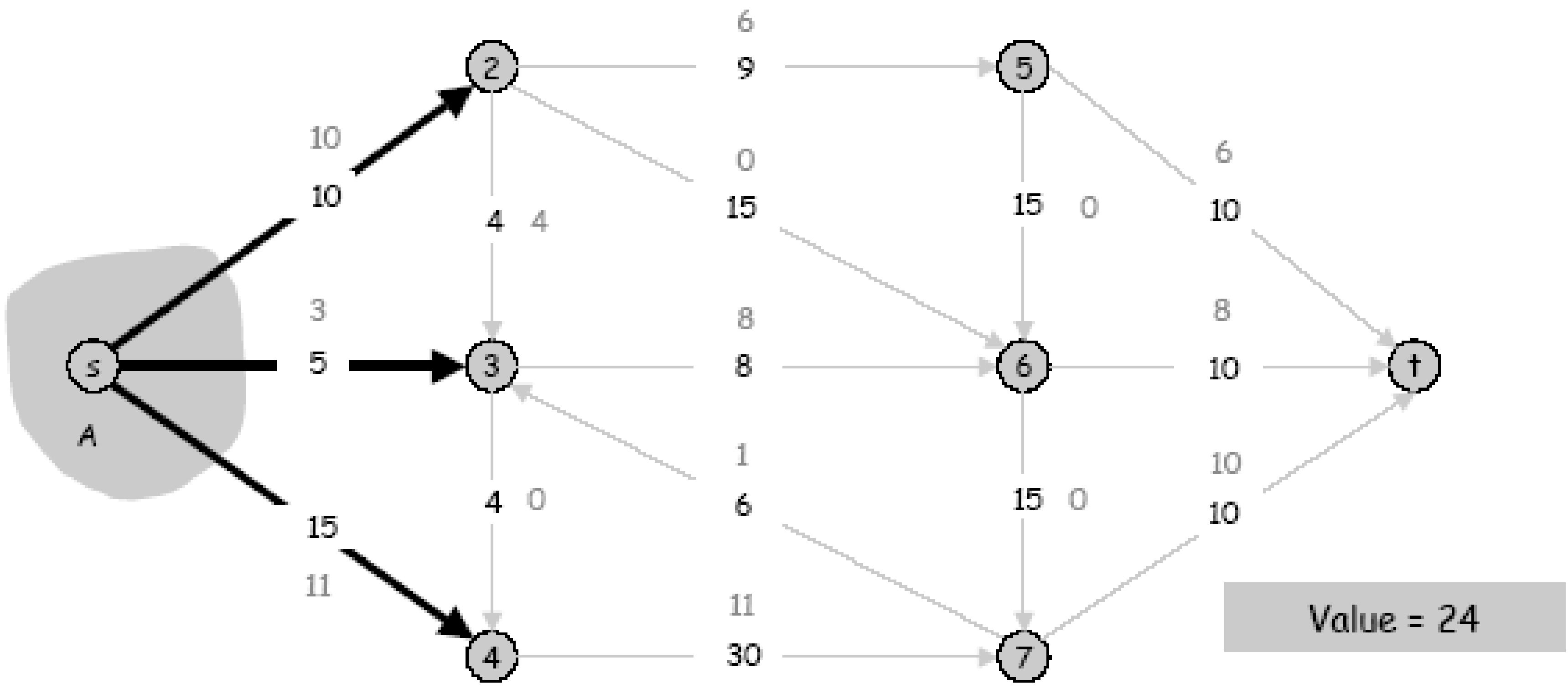
算法分析

- Demo演示(增广路径、瓶颈容量、剩余网络、f(x,y)最大流)
- 初始图G



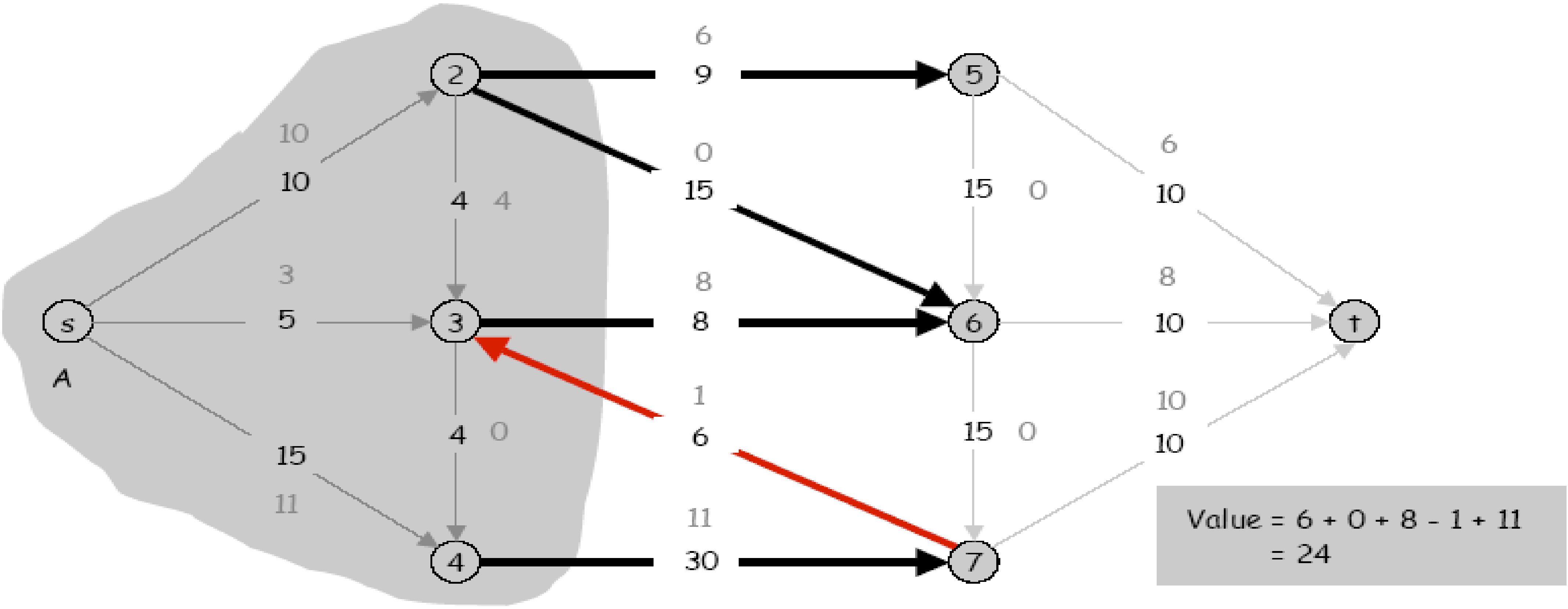
https://blog.csdn.net/qq_38662930/article/details/105242945

最大流与最小割



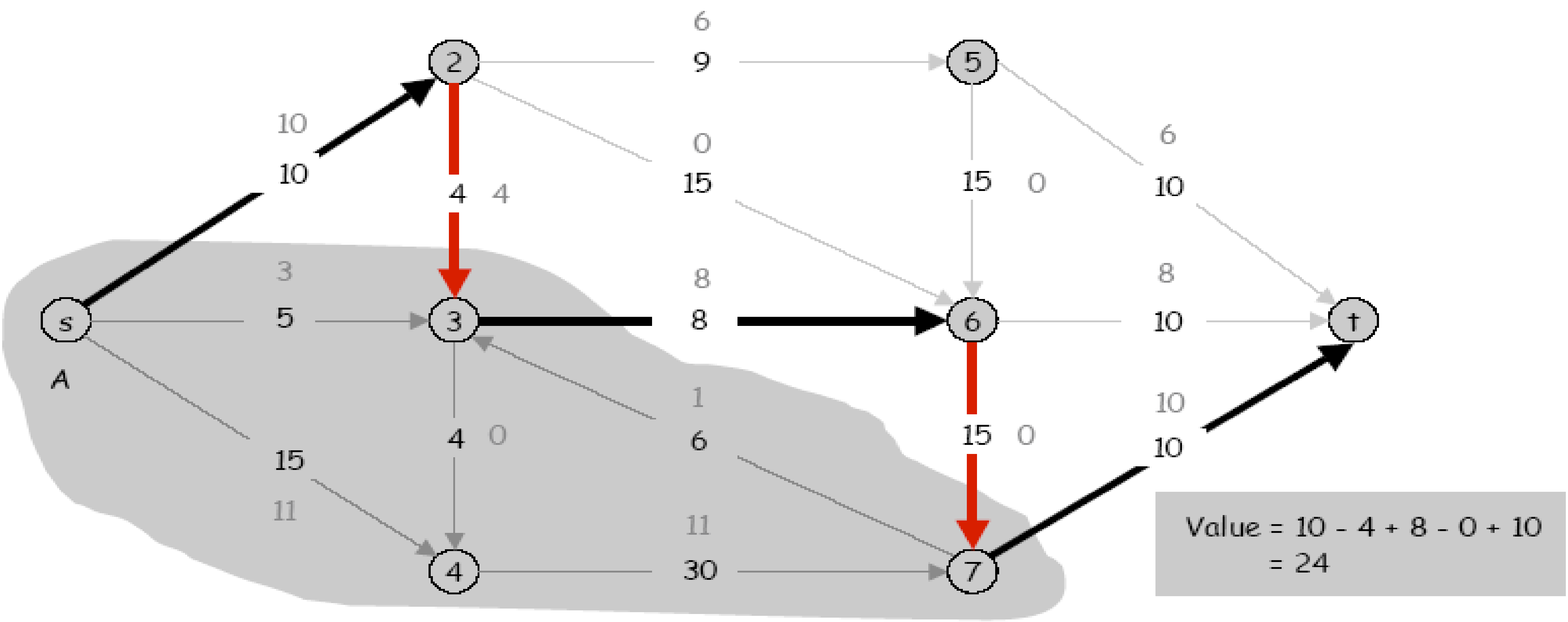
s
2
3
4
5
6
7
t
A
10
10
3
5
15
11
4 4
4 0
6
9
0
15
8
8
1
6
11
30
15 0
15 0
6
10
8
10
10
10
Value = 24

最大流与最小割



s
A
2
3
4
5
6
7
t
10
10
3
5
15
11
4 4
4 0
6
9
0
15
8
8
1
6
11
30
15 0
15 0
6
10
8
10
10
10
Value = 6 + 0 + 8 - 1 + 11
= 24

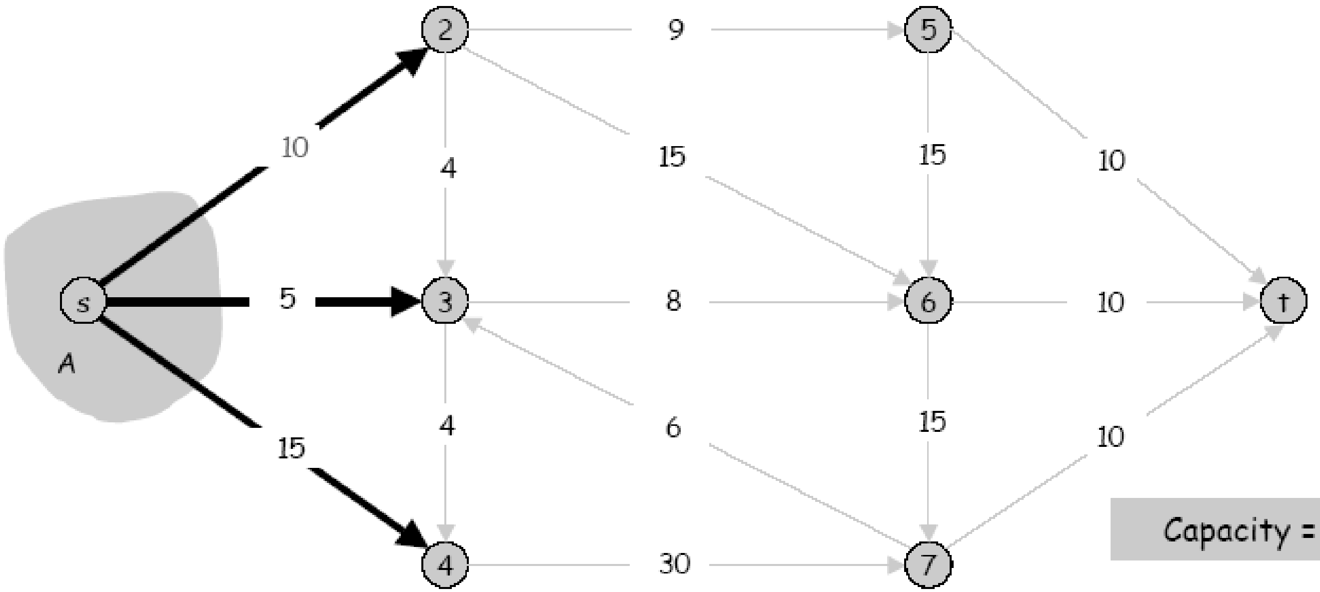
最大流与最小割


6
9
2
5
10
10
4 4
0
15
15 0
6
10
3
5
s
3
8
8
6
8
10
t
A
1
6
4 0
15 0
10
10
15
11
11
4
30
7
Value = 10 - 4 + 8 - 0 + 10
= 24

最大流与最小割

- 定理7.8令f是任意s-t流，且(A,B)是任意s-t割，那么 $v(f) \le c(A,B)$

Cut capacity = 30 ⇒ Flow value ≤ 30

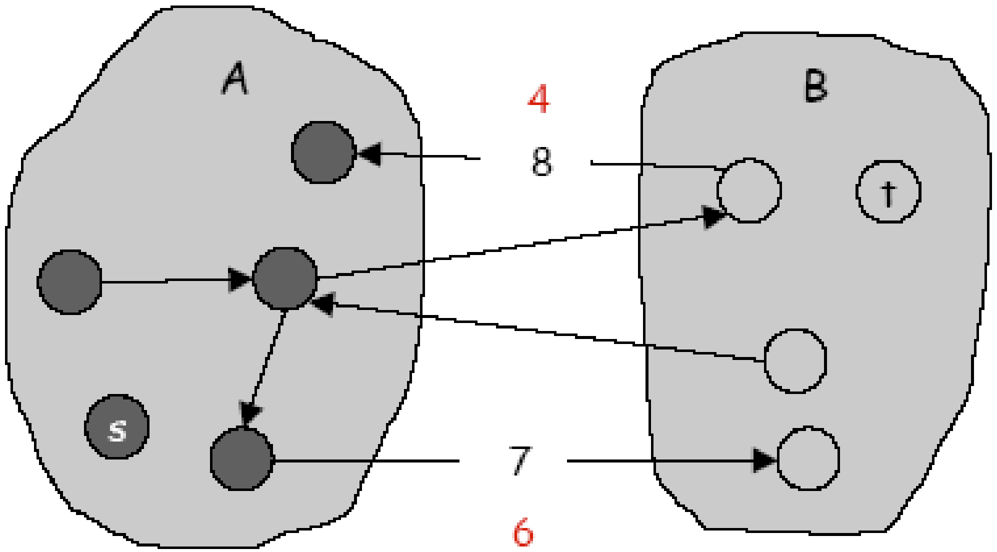


Capacity = 30

最大流与最小割

- 证明：

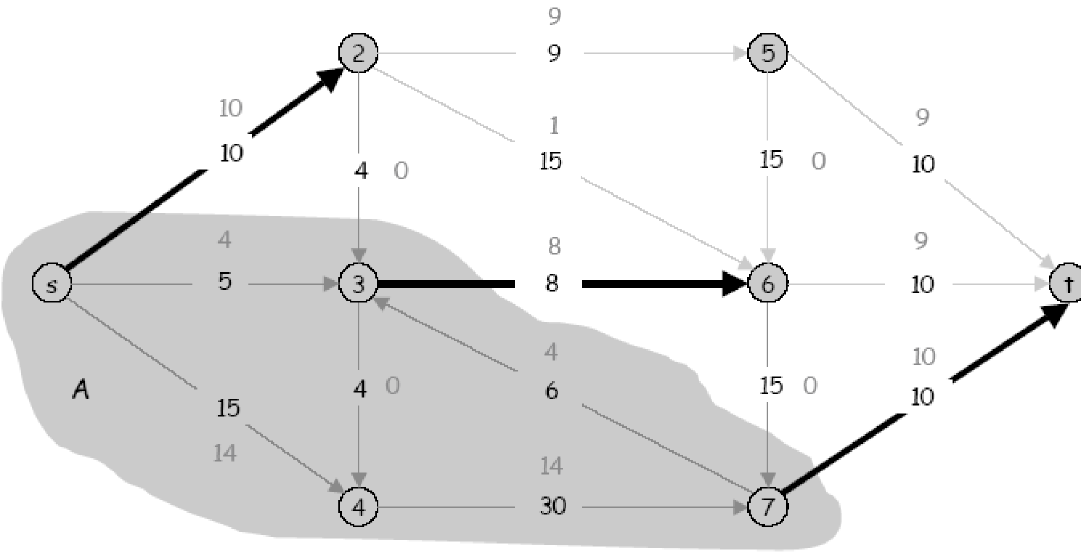
$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&\le \sum_{e \text{ out of } A} f(e) \\
&\le \sum_{e \text{ out of } A} c(e) \\
&= \operatorname{cap}(A, B) \qquad \blacksquare
\end{aligned}
$$



最大流与最小割

- 推论. 设f是任意的流， 并设 (A, B) 是任意的割. 如果v(f) = cap(A, B), 那么 f 是最大流， 并且(A, B) 是最小割.

Value of flow = 28
Cut capacity = 28 ⇒ Flow value ≤ 28



最大流与最小割

- 令 $\overline{f}$ 表示由Ford-Fulkerson返回的流
- 下面我们将给出一个s-t割(A*,B*)使得

$$v(\overline{f}) = c(A^*, B^*)$$

这直接说明 $\overline{f}$ 有任何流的最大值，并且(A*,B*)有任何s-t割最小的容量

Ford-Fulkerson终止时的流有什么性质？

最大流与最小割

- 定理7.9 如果f是使得剩余图G_f中没有s-t路径的一个s-t流，那么在G中存在一个 s-t割(A*,B*)使得v(f)=(A*,B*).因此，f有G中任何流的最大值，且(A*,B*)有G中任何s-t割的最小容量。
- 最大流最小割定理. [Ford-Fulkerson 1956] 最大流的值等于最小割

最大流与最小割

- 证明思路:

  (i) 存在 cut (A, B)使得 v(f) = cap(A, B).

  (ii) 流 f 是一个最大流.

  (iii) f中没有增广路径.

  (i) $\Rightarrow$ (ii) $\Rightarrow$ (iii) $\Rightarrow$ (i)

最大流与最小割

- (i) $\Rightarrow$ (ii)显而易见
- (ii) $\Rightarrow$ (iii)运用反证法. 设f是一个流，如果还存在一条增广路经，那么我们还可以继续改进f,矛盾。
- (iii) $\Rightarrow$ (i)

实际上这是算法停止运行的条件

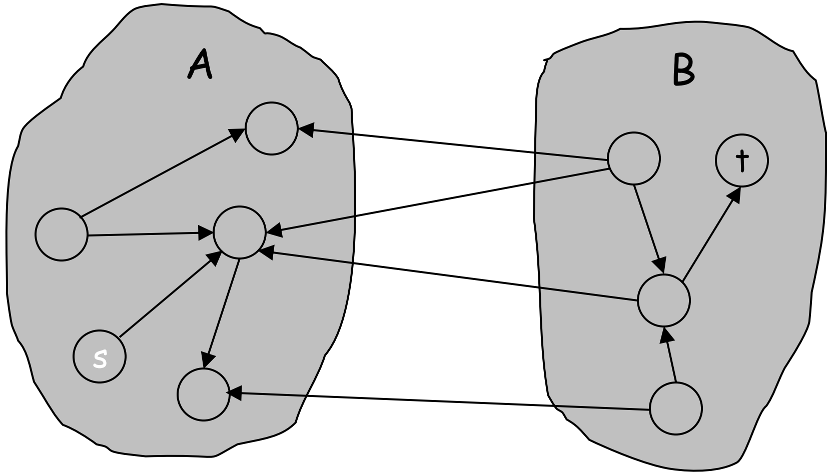
最大流与最小割

- 设流f没有增广路径.
- 定义集合A 是**剩余图Gf**中从源点s可达顶点集合.
- 根据定义，那么s ∈ A；终点 t ∉ A, ∈ B.

  如果e=(u,v), u∈A,v∈B,那么f(e)=c(e);

  如果e'=(u',v'), u'∈B,v'∈A,那么f(e')=0.

$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) \\
&= \sum_{e \text{ out of } A} c(e) \\
&= cap(A,B)
\end{aligned}
$$



剩余图

最大流与最小割

- 定理7.10 由Ford-Fulkerson算法返回的流 $\overline{f}$ 是最大流。
- 给定$\overline{f}$，计算最小s-t割(A,B)的时间？

最大流与最小割

- 定理7.11 给定一个最大值的流 $\overline{f}$，我们可以在**O(m)**时间内计算一个最小容量的s-t割。
- 命题7.12 在每个流网络中，存在一个流f和一个割(A,B),使得v(f)=c(A,B).
- 定理7.14 如果在流网络中所有的容量都是整数，那么存在一个最大流f,它的每个流值f(e)都是整数。

最大流与最小割

- 如果边的权值(容量)是实数，最大流最小割定理依然成立。

- 但是由于增广路径选择的不合理，具有实数容量的Ford-Fulkerson算法可能永远运行下去

- 解决问题思路：选择好的增广路径

7.3 选择好的增广路径

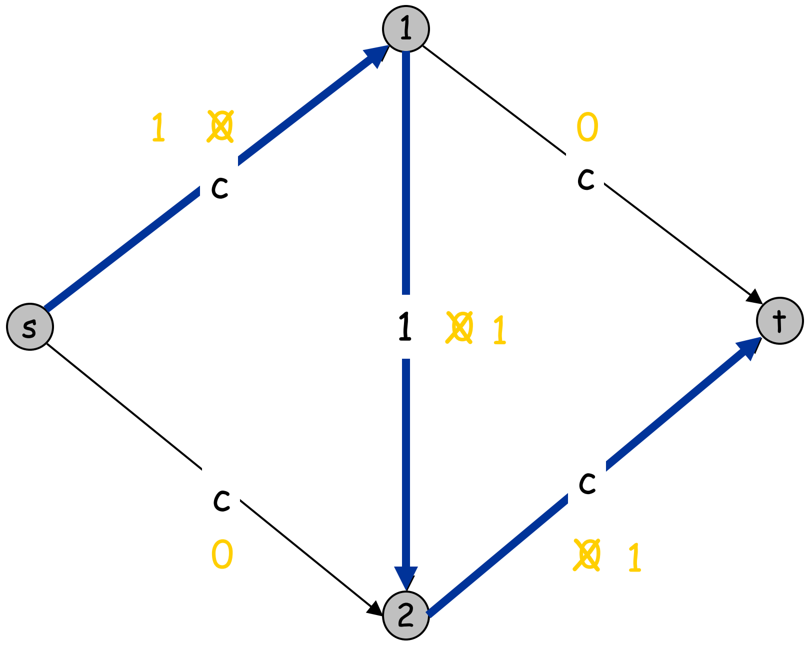
- 一般的Ford-Fulkerson算法复杂度是不是**输入规模**的多项式时间？(输入数据：m,n,logC)

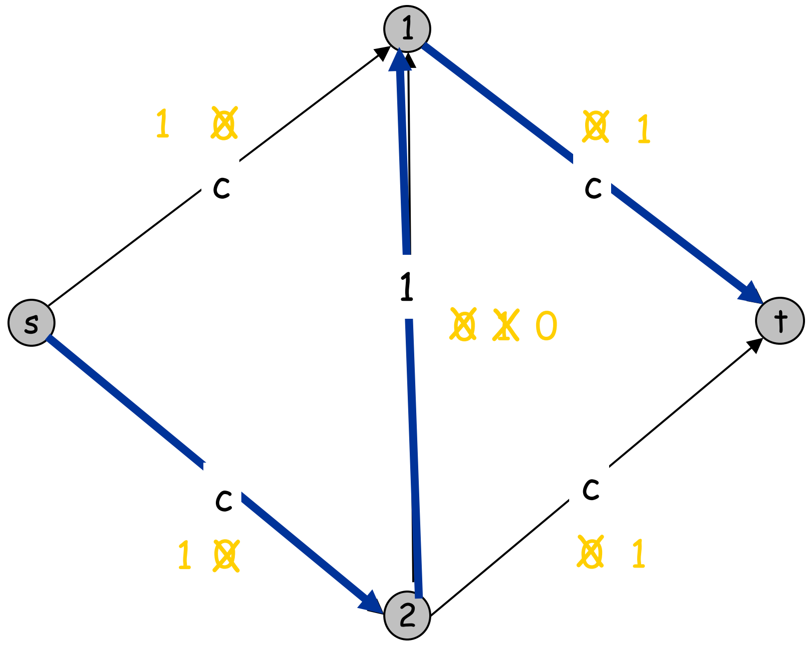
- 不是，定理7.5 告诉我们，Ford-Fulkerson算法的运行时间在O(mC)，*伪多项式时间*

- 有时算法执行会非常没有效率

选择好的增广路径

- 如果最大的流容量是C, 算法可能要循环C次





选择好的增广路径

- 之所以出现这样的问题，在于我们刚才选择了一条瓶颈容量很小的增广路径，导致**收敛**很慢

- 所以我们的思路是：

  因为增广路径通过选择路径的瓶颈容量来增加最大流的值，我们选择**具有大的瓶颈容量的路径**，那么算法进展会更大些

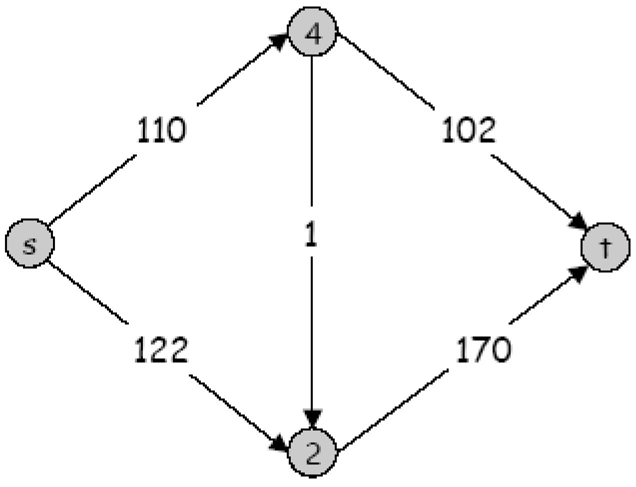
选择好的增广路径

- 选择好的增广路径:
  Sufficiently large bottleneck capacity

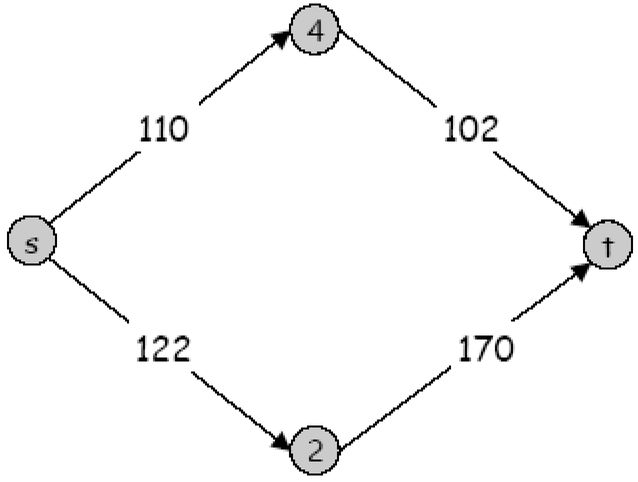
- 这里为了便于控制，我们维护一个称之为缩放参数的 Δ ，算法中将寻找瓶颈容量至少是Δ的路径。

选择好的增广路径

令$G_f(\Delta)$是仅由剩余容量至少为Δ的边组成的剩余图的子集.



G_f



$G_f(100)$

算法

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

算法分析

- 流在算法中始终保持整数值，因此所有的剩余容量也是整数值。

- 定理7.15 如果容量是整数值，那么在缩放最大流算法中流和剩余容量也始终保持整数值，这就推出当$\Delta = 1$，$G_f(\Delta) = G_f$，算法终止时，流f是最大值的流。

算法分析

- 现在我们开始关注算法的循环部分，估计各部分循环的次数

- 最外层循环While的次数？

- 命题7.16 外层While循环的迭代次数至多是 $1 + \lceil \log_2 C \rceil$

  证明： 最开始 $C \le \Delta < 2C$， Δ每次缩小一半

算法分析

- 下面需要界定内层循环在每个缩放阶段所做的增广次数
- 在Δ 缩放阶段，我们只用到剩余容量至少是Δ的边，根据算法每次增加瓶颈容量的性质，就有
- 命题**7.17** 在Δ 缩放阶段，每次增广增加的流值至少是Δ.

算法分析

- 另外关键一点是Δ 缩放阶段结束时，流f不可能距最大值太远

- 命题7.18 令f是Δ 缩放阶结束时的流.G中存在一个s-t割(A,B)使得c(A,B)<=v(f)+m Δ, 其中m是图G中的边数。因此在网络中的最大流值至多是v(f)+m Δ.

算法分析

- 证明：(与最大流最小割定理证明思路一样)

  在 Δ-缩放阶段, 存在割 (A, B) 使得cap(A, B) $\leq$ v(f) + m Δ, 采用构造法。

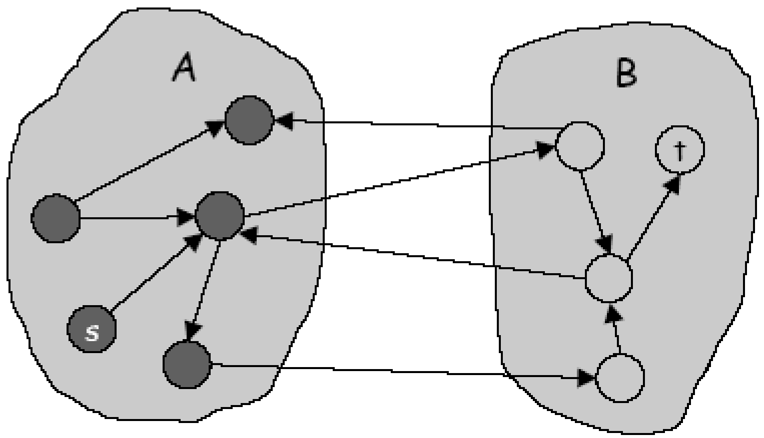
  - 选择A是$G_f(\Delta)$ 中从s出发可达的顶点集合
  - 根据定义 s $\in$ A; t $\notin$ A， $\in$ B.

    如果e=(u,v), u$\in$A,v$\in$B,那么c(e)<f(e)+ Δ;

    如果e′=(u′,v′), u′$\in$B,v′$\in$A,那么f(e′)< Δ.

算法分析

$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
&\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\
&= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
&\geq cap(A, B) - m\Delta \ .
\end{aligned}
$$



original network

算法分析

- 命题7.19 在一个缩放阶段增广次数至多是2m.

- 证明：考虑缩放阶段Δ,令f_p是前一缩放阶段结束时的流。那时采用Δ'=2 Δ作为参数，于是最大流f*至多是$v(f_p)+2m$ Δ,因此至多可能存在2m次增广。

算法分析

- 一次增广用O(m)时间(包括建立图，找到合适路径)
- 缩放次数：至多 $1+\lceil \log_2 C \rceil$
- 缩放阶段增广次数：至多2m
- 定理7.20 在具有m条边和整数容量的图中，缩放最大流算法找最大流至多用 $2m(1+\lceil \log_2 C \rceil)$次增广，于是可在$O(m^2 \log C)$ 时间内运行。

算法分析

- 一般的Ford-Fulkerson算法需要与容量的数量级成正比的时间；
- 这里给出的缩放算法只需要与说明问题输入的容量所需字节数成正比的时间
- 缩放算法运行在**输入规模**(边数及容量的数字表示)的多项式时间

推广：强多项式算法

- [Edmonds-Karp 1972, Dinitz 1970]
- 存在强多项式算法
- 仅仅是边数m,顶点数n的多项式
- 每次迭代选择具有**最少边数**的增广路径
- O(mn)
- 其他的一些改进复杂度 O(mnlogn),O(n^3),…

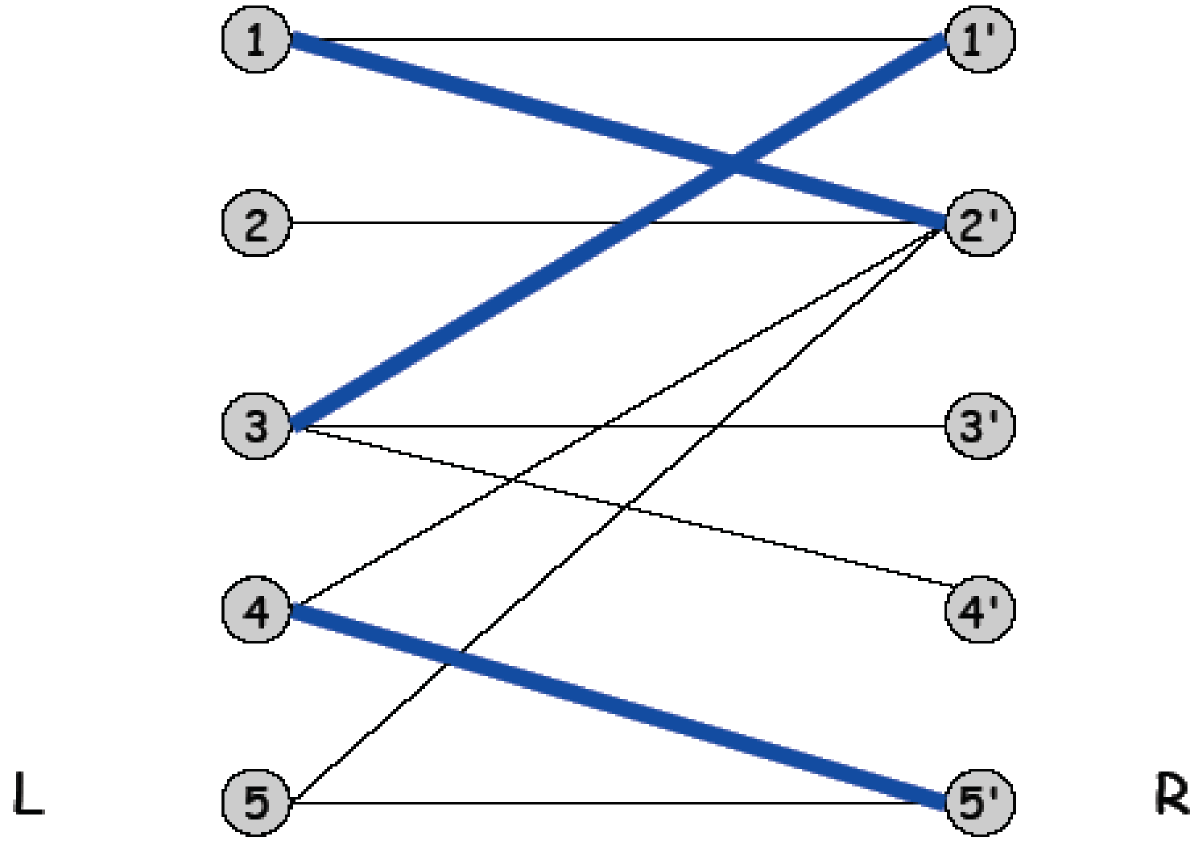
7.5 二分匹配问题

- 输入：无向图 G = (V, E).
- $M \subseteq E$ 是一个匹配，如果每个结点至多出现在M中的一条边中。
- 最大匹配：寻找具有最大数目的匹配

二分匹配问题

- 二部图G=(V,E)是一个无向图，它的结点集合可以被划分成V= L ∪ R,并具有下述性质：每条边e ∈E有一个端点在L中，另一个端点在R中。
- 二分匹配.
  - 输入: 无向，二部图, G = (L ∪ R, E).
  - M ⊆ E 是一个匹配，*如果每个结点至多出现在M中的一条边中。*
  - 最大匹配：**寻找具有最大数目的匹配**

二分匹配问题

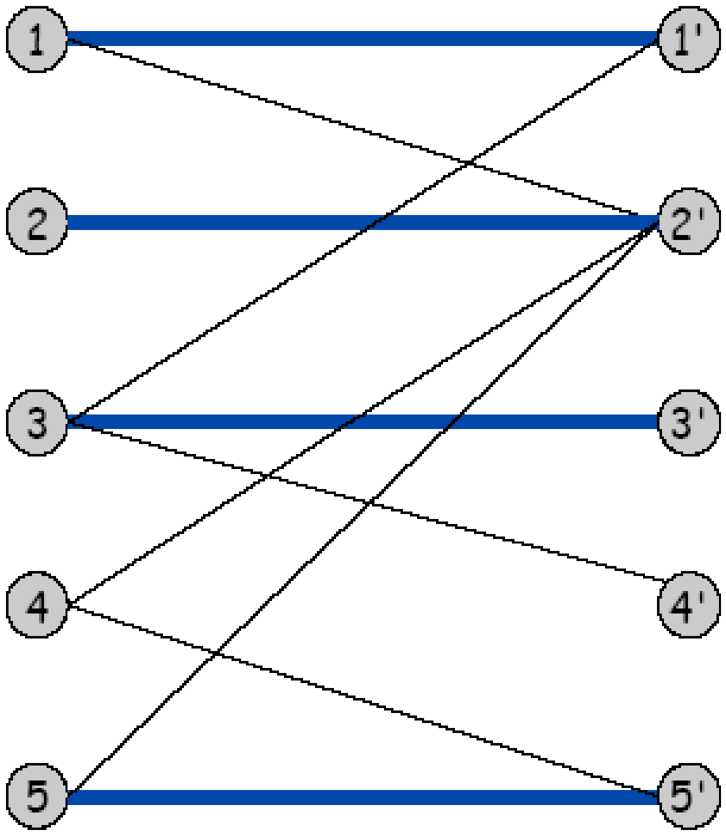


匹配

1-2', 3-1', 4-5'

是最大匹配吗？

二分匹配问题



最大匹配

1-1', 2-2', 3-3' 4-4'

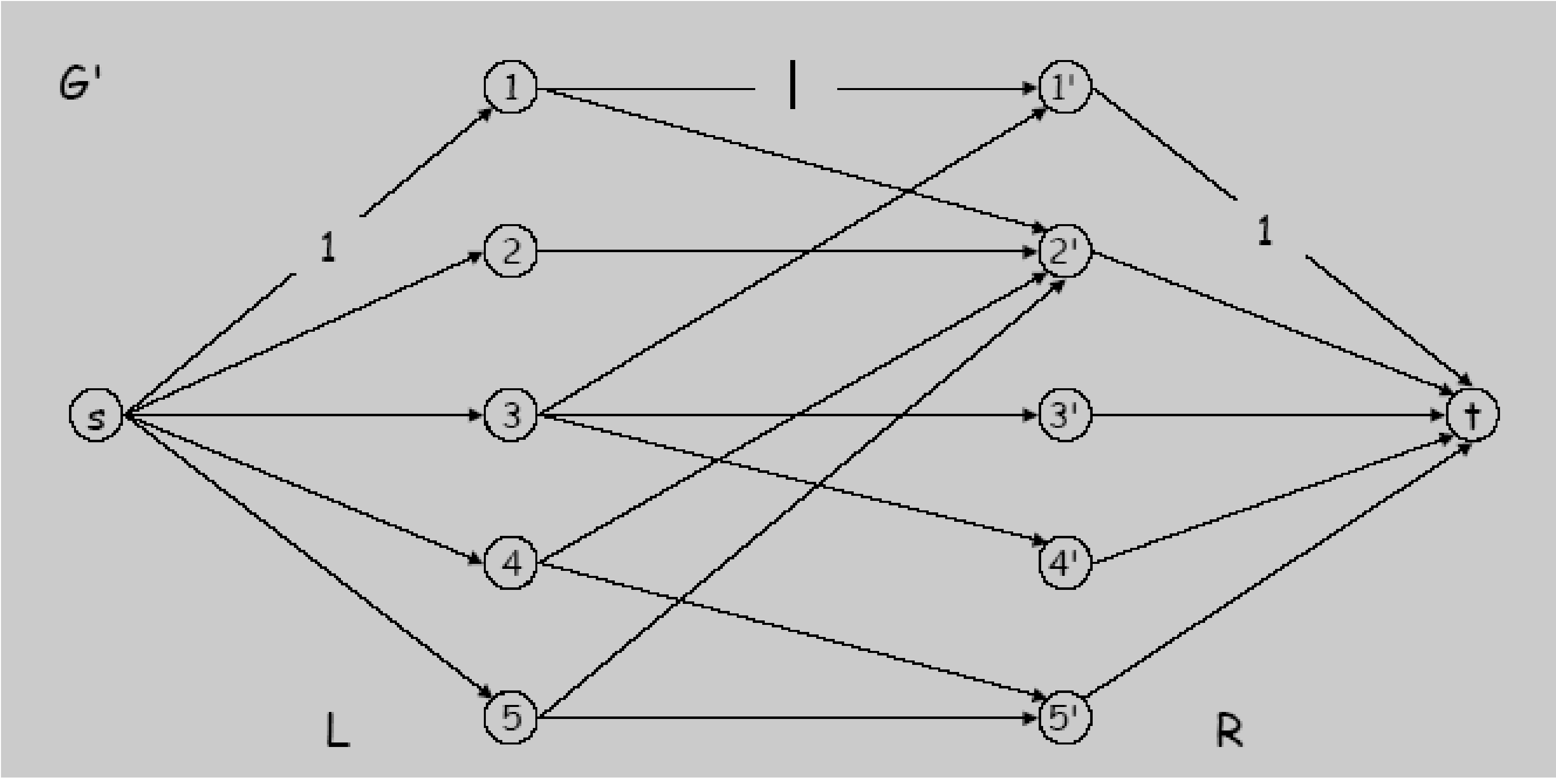
二分匹配问题

- 二分匹配问题看起来与流网络有一定类似的地方
- 这里将应用流网络的相关成型算法
- 首先构造一个流网络，满足需要的容量条件，守恒条件

二分匹配问题

- 最大流的构造.
  - 构造图 $G' = (L \cup R \cup \{s, t\}, E')$.
  - 连接原图L到R的每条边, 每条边赋予单位容量.
  - 增加一个源点s, 从s到L中的每个结点连接一条边，每条边赋予单位容量.
  - 增加一个终点 t, 从R中的每个结点到t连接一条边，每条边赋予单位容量.

二分匹配问题



二分匹配问题

- 现在计算这个网络G’的最大s-t流，我们发现这个流的值等于G中最大匹配的大小。

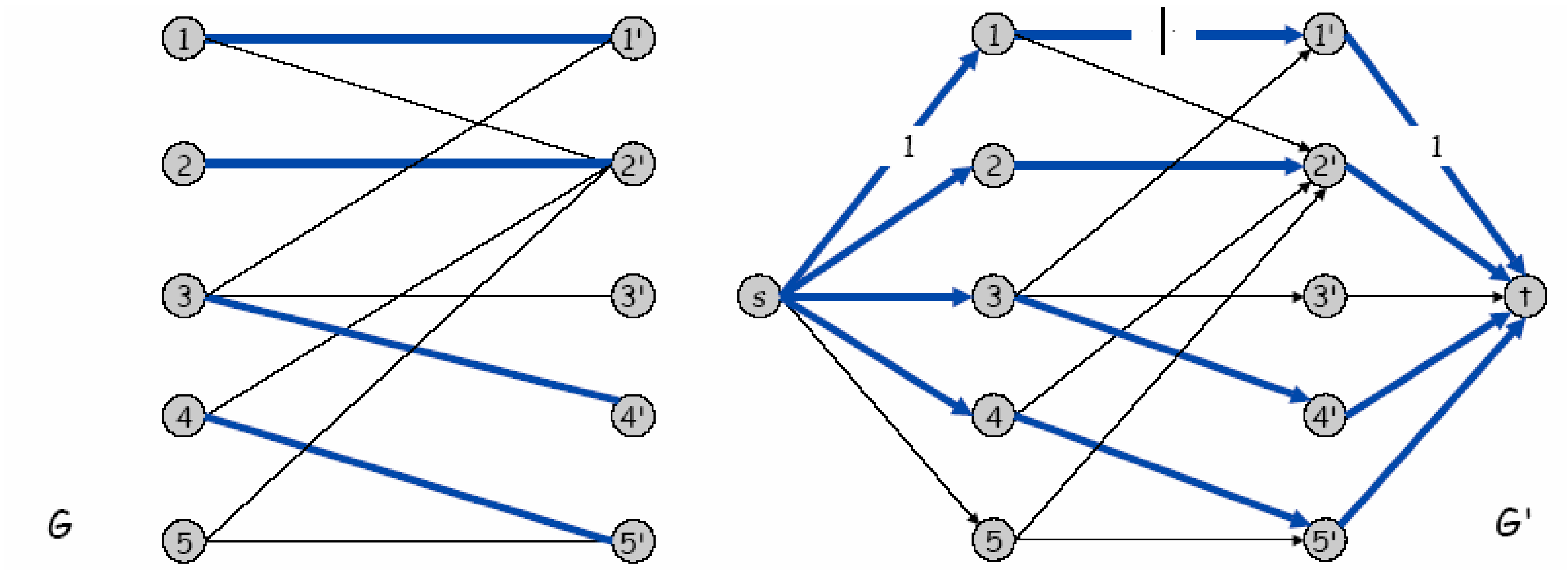
- **定理. G中最大匹配的数目与所定义的G'中最大流值相同.**
- 证明：

  设G中最大匹配集合是M， 其数目是 k.

  - 于是可以构造一个流f, 每一条边从s出发，携带一个单位的容量.
  - f 是一个流，而且流值为k.
  - 所以G'中最大流值>=最大匹配数目;

二分匹配问题

■



二分匹配问题

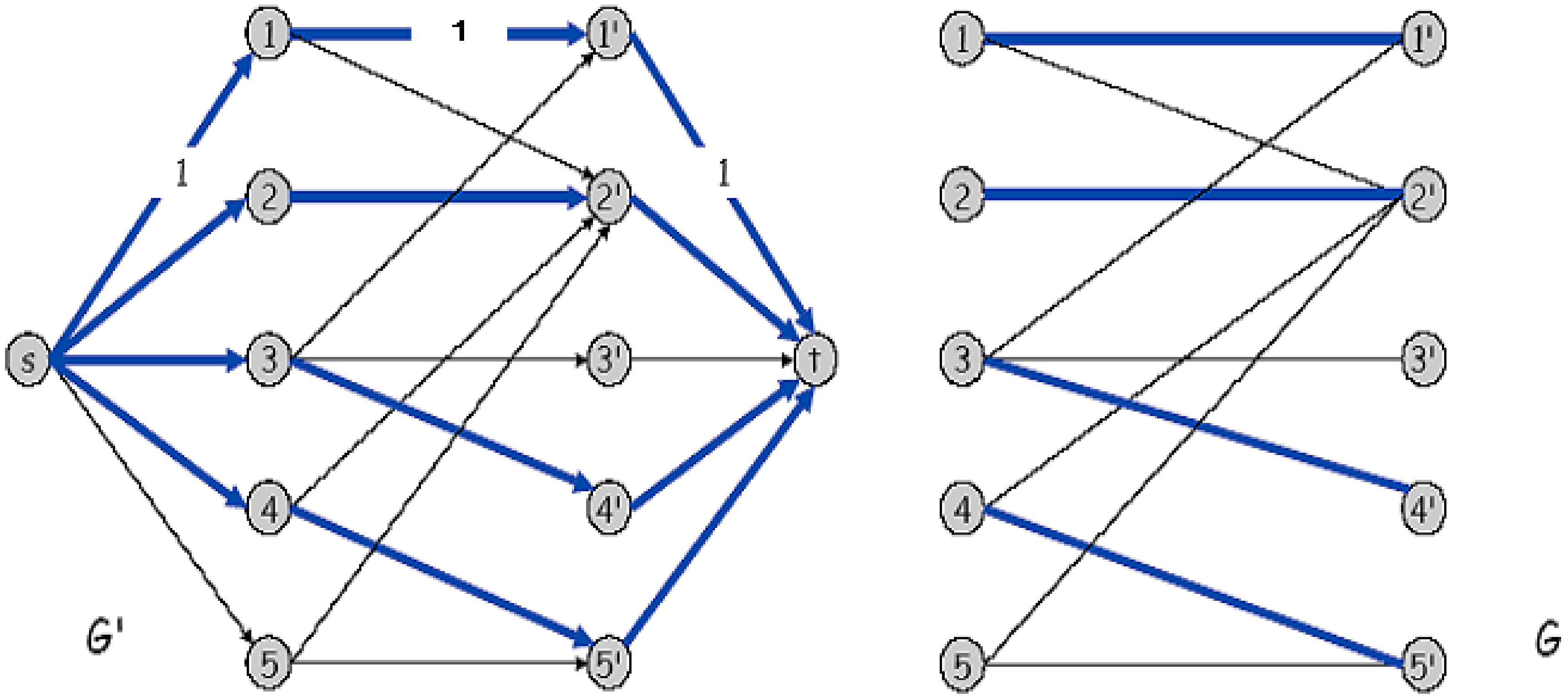
设 f 是 G' 最大流，其值为k.

- 考虑集合M：<u>从L到R权值为1的边的集合</u>，i.e., f(e) = 1.
  - 可以发现每个节点至多在M的一条边中
  - |M| = k：割 (L ∪ s, R ∪ t) 就是一个匹配

所以最大匹配的数目>=最大流值

因此定理成立。

二分匹配问题



要证明图G中的最大匹配数目等于其对应的流网络G'中的最大流值，可以通过以下步骤进行：

构造流网络G'：

1. 若G是二分图，分为两部分X和Y，则：
   - 添加源点s和汇点t。
   - s向X中每个节点连边，容量为1。
   - Y中每个节点向t连边，容量为1。
   - 原图G中的每条边(u, v)（u∈X，v∈Y）转化为从u到v的有向边，容量为1。

证明对应关系：

1. **匹配转换为流：**
   - 对G中的任意匹配M，每条边(u, v) ∈ M对应G'中的路径s→u→v→t，每条路径分配1单位流量。
   - 由于M中每个节点仅出现一次，故s→u和v→t的边容量未被超出，流量守恒成立。此时流值等于|M|，故最大流至少为最大匹配数。
2. **流转换为匹配：**
   - 在G'中，任何整数流f（流量0或1）对应G中的边集合M'。因容量限制，每个节点u∈X和v∈Y至多有一条出边流量为1。
   - 这些流量为1的边(u, v)构成匹配M'，流值v即为|M'|，故最大匹配数至少为最大流值。

结论：
由于最大匹配数不超过最大流值，反之亦然，故两者相等。因此，G中最大匹配数目等于G'中的最大流值。

答案：
通过将二分图G构造为流网络G'，其中每条边的容量为1，并添加源汇点连接两侧节点，可证明G中的每个匹配对应G'中一个等值流，反之亦然。因此，最大匹配数等于最大流值，即二者相等。

二分匹配问题：界定运行时间

令n=|L|=|R|, m是G的边数，一般假定初始问题中每个结点至少存在一条关联边，因此m>=n/2.

- 时间复杂度?
- 注意到C=|L|=n,根据以前O(mC)的界
- 定理7.38　可以用Ford-Fulkerson算法在**O(mn)**时间内找到二部图中的一个最大匹配。