网络流

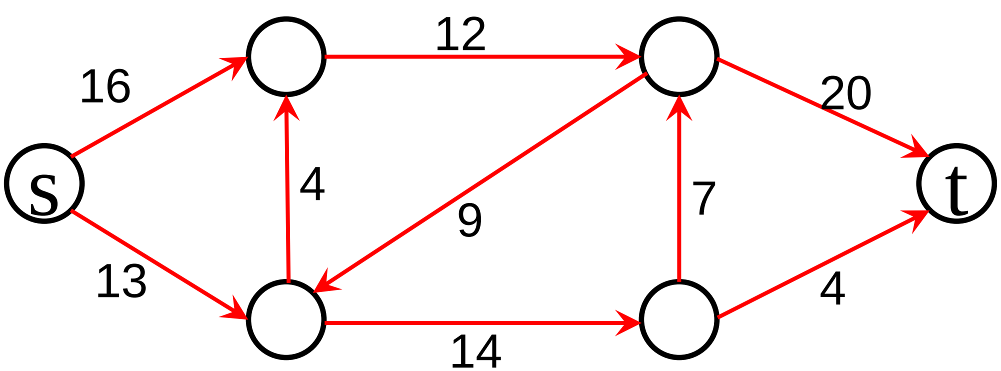
东南大学计算机学院 方效林

本章内容

- 网络流

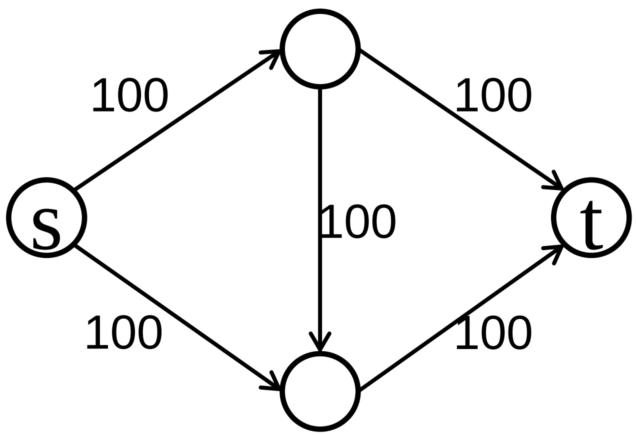
本章内容

- 给定有向图$G=(V,E)$，边上权值表示容量，给定源点s和汇点t，求s到t可流过的最大流量是多少？



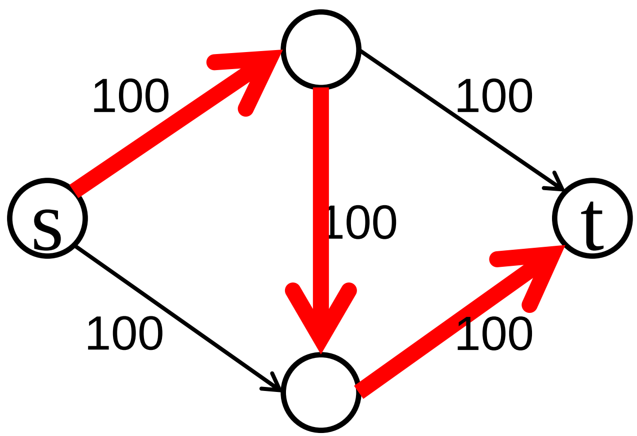
网络流

- 给定有向图$G=(V,E)$，边上权值表示容量，给定源点s和汇点t，求s到t可流过的最大流量是多少？

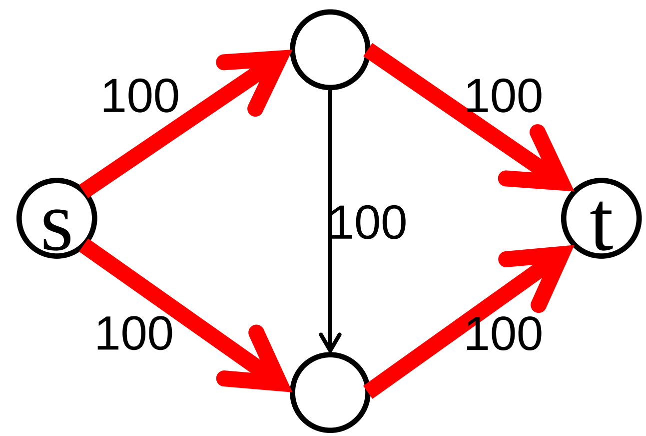


网络流

- 简单想法
  - 从源到汇找一条可行路径，塞满流量



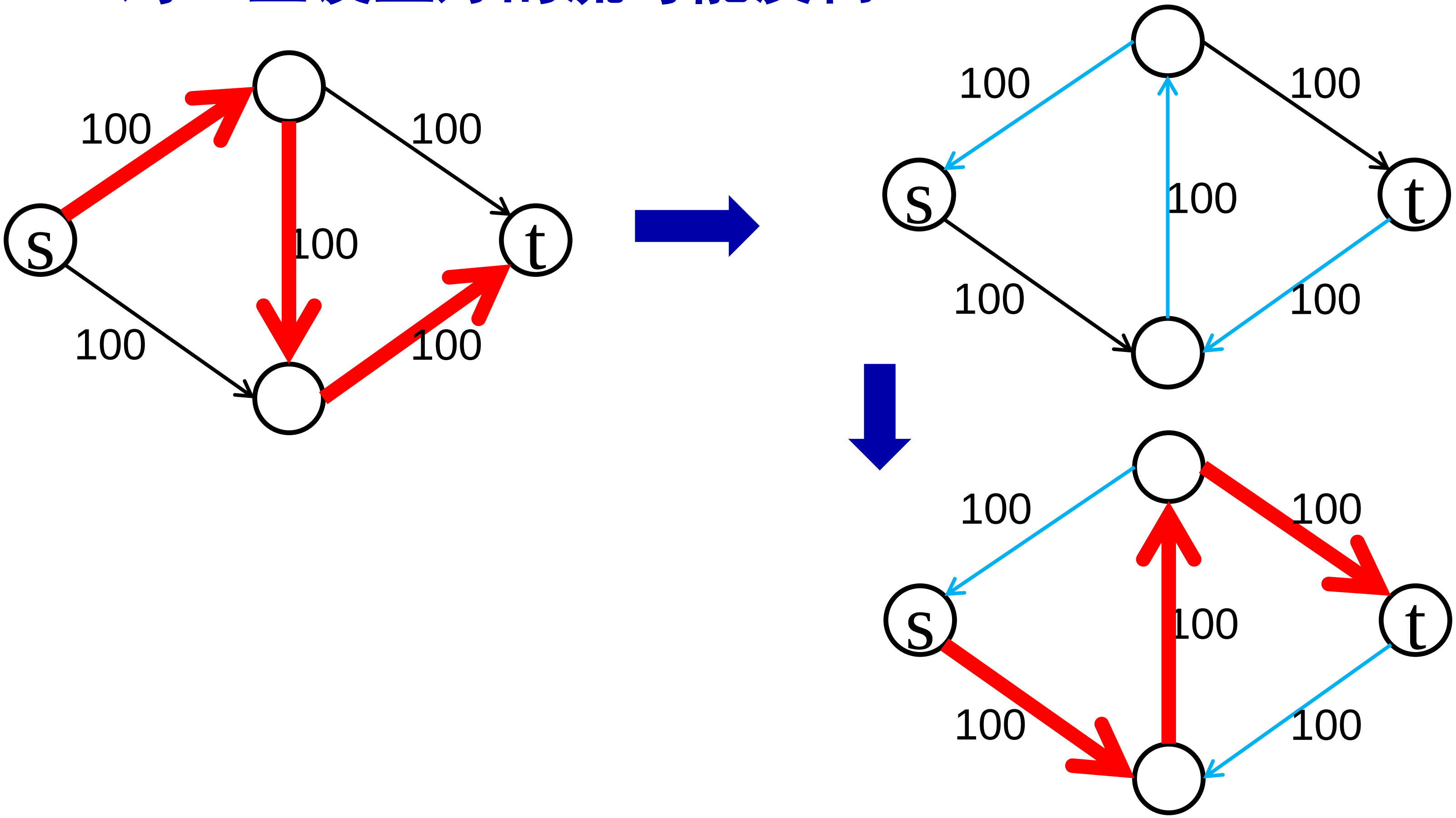
简单想法



最优解

网络流

- 取消部分设置好的流
  - 对一些设置好的流可能反悔

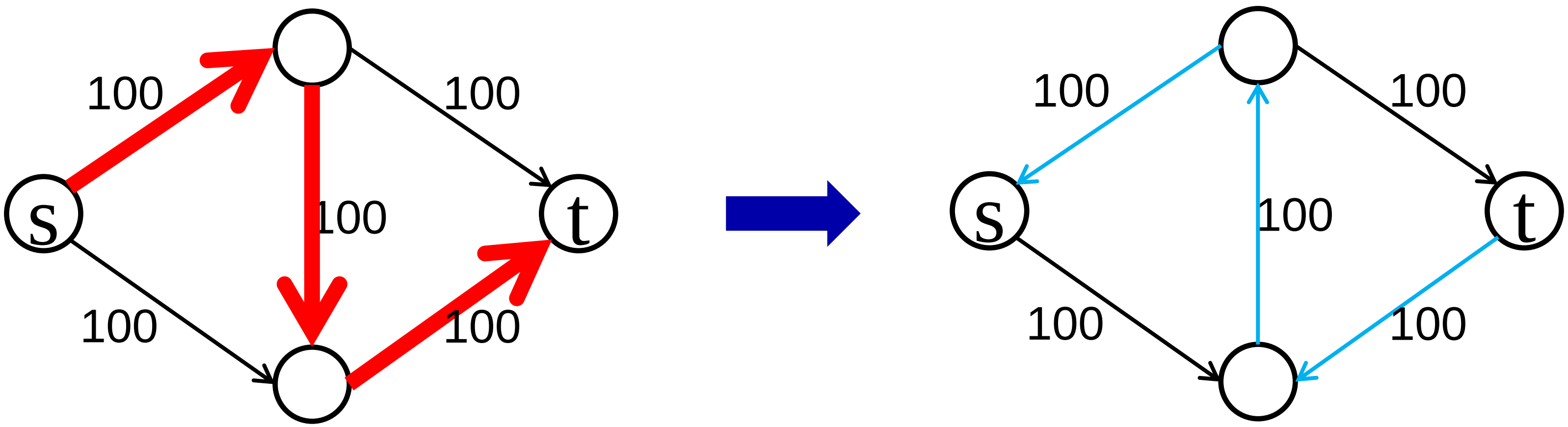


网络流

- **残留网络**

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v), 若(u,v) \in E \\ f(v,u), \quad 若(v,u) \in E \end{cases}$$

  - 某一条边使用了多少流量，则其反方向设置多少可反悔的流量

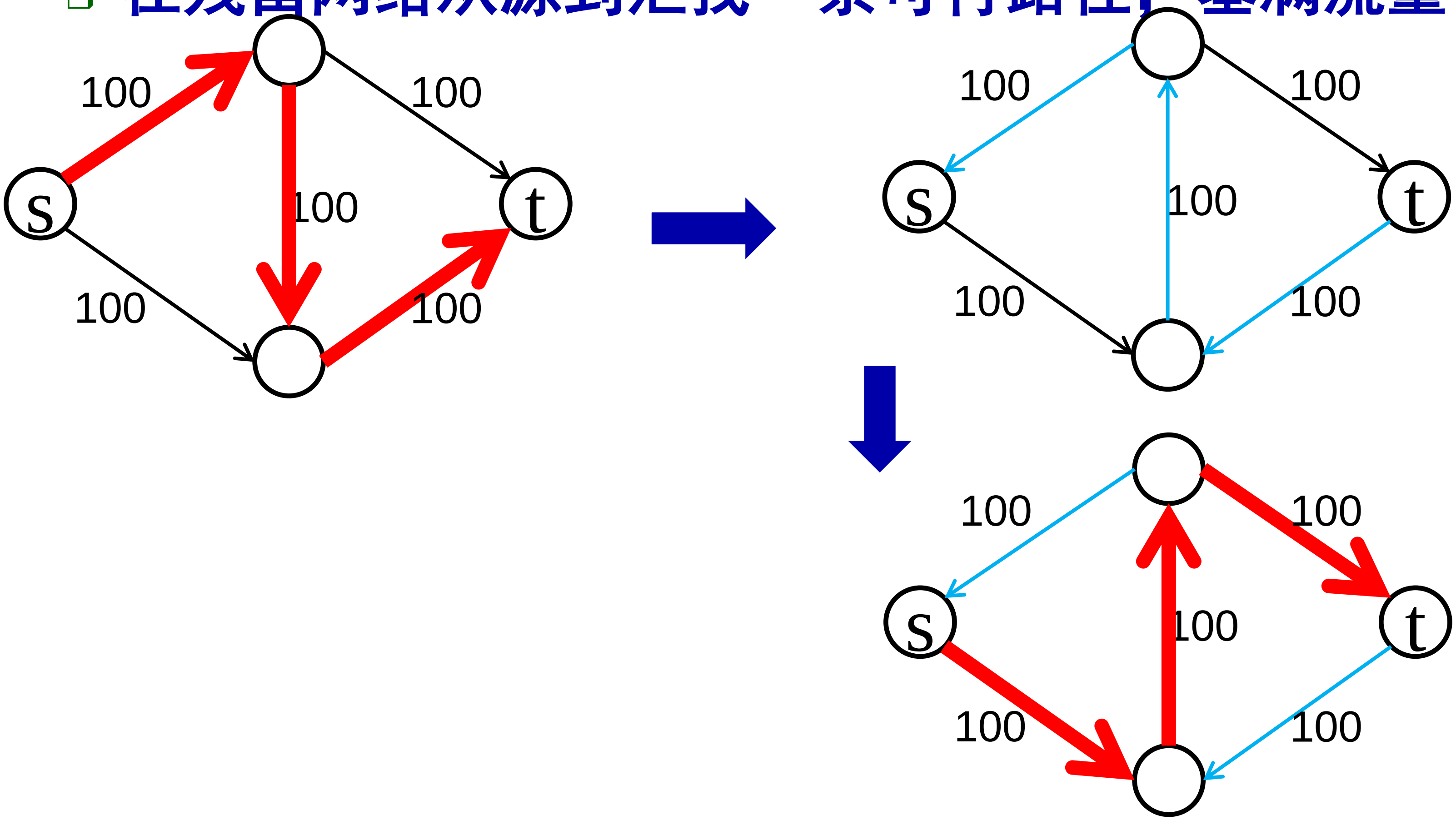


网络流

- **Fold-Fulkerson算法**

$$c_f(u,v)=\begin{cases} c(u,v)-f(u,v), & 若(u,v)\in E \\ f(v,u), & 若(v,u)\in E \end{cases}$$

  - 在残留网络从源到汇找一条可行路径，塞满流量

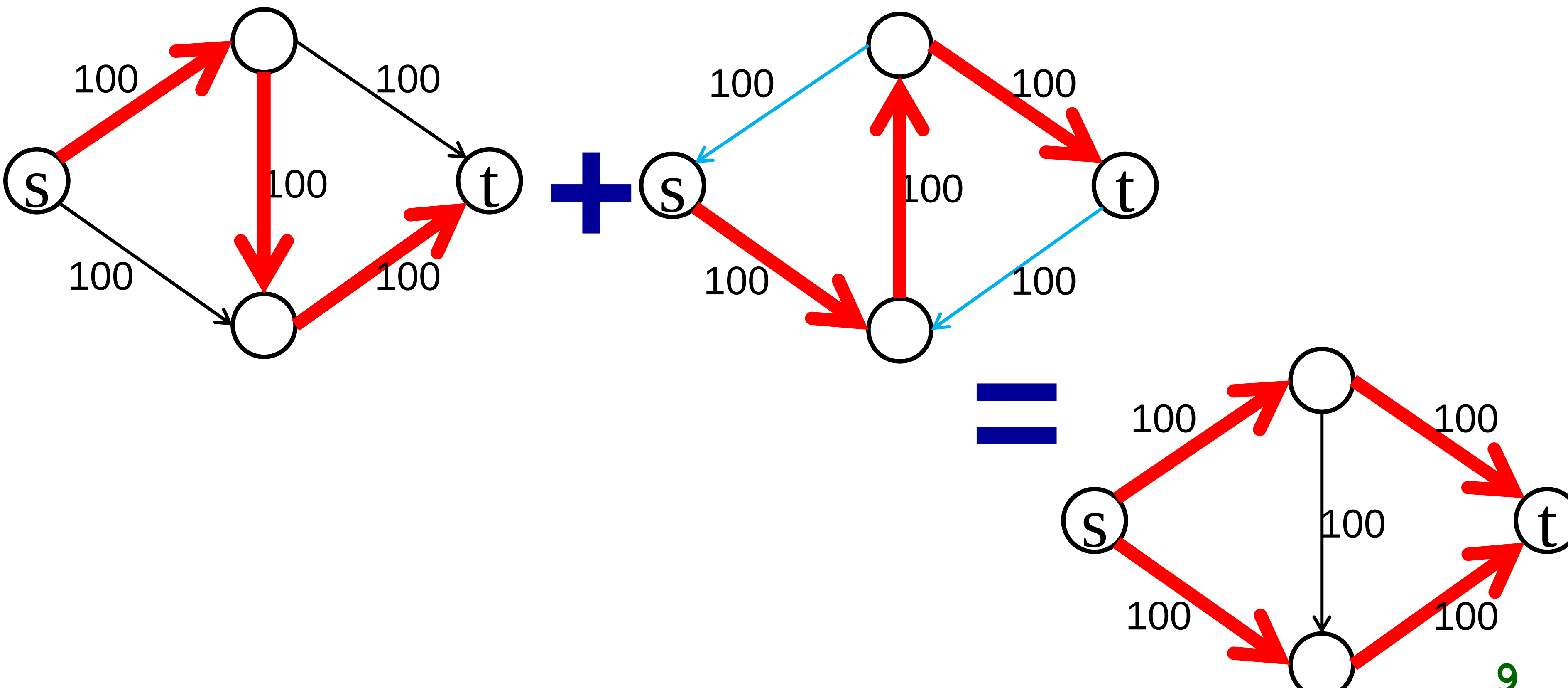


网络流

- **Fold-Fulkerson算法**

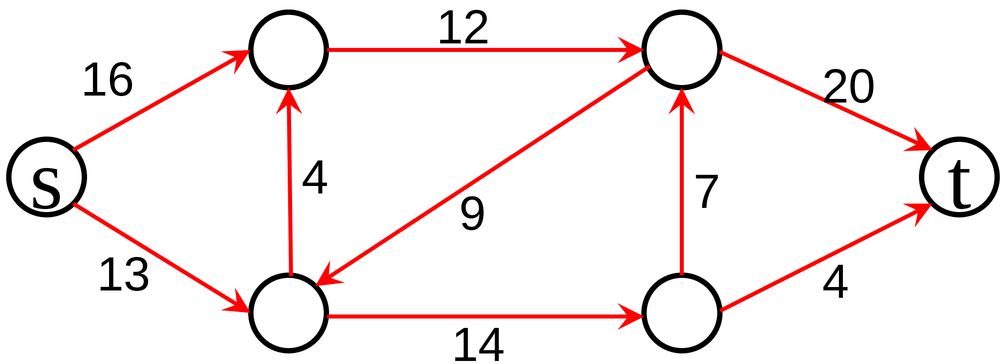
$$c_f(u,v)=\begin{cases} c(u,v)-f(u,v), 若(u,v)\in E \\ f(v,u), \qquad\qquad\quad 若(v,u)\in E \end{cases}$$

  - 在残留网络从源到汇找一条可行路径，塞满流量



网络流

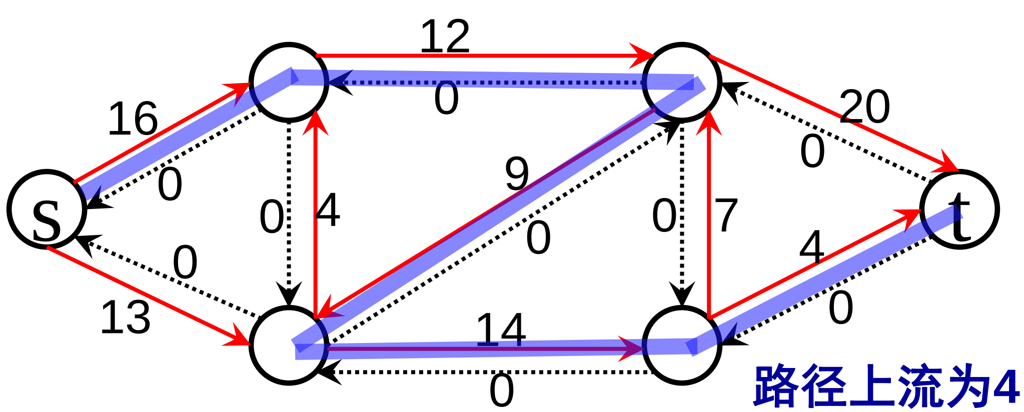
- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases} c(u,v)-f(u,v), & 若(u,v)\in E \\ f(v,u), & 若(v,u)\in E \end{cases}$
  - 在残留网络从源到汇找一条可行路径，塞满流量



初始网络

网络流

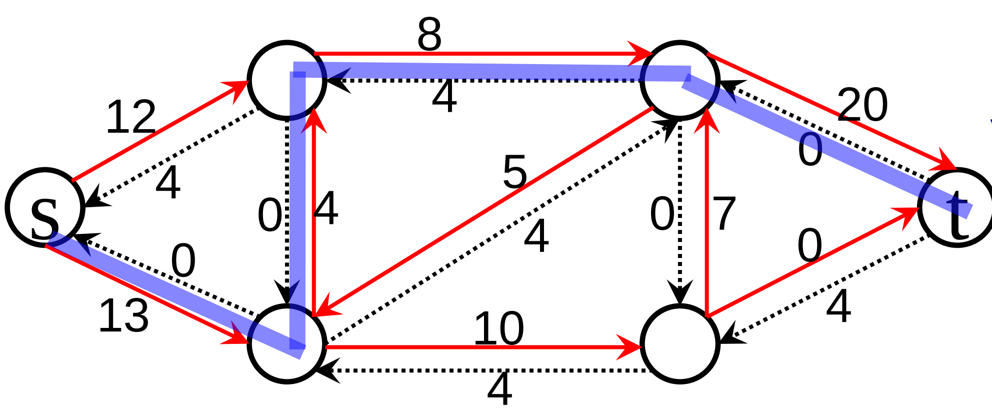
- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases} c(u,v)-f(u,v), & 若(u,v)\in E \\ f(v,u), & 若(v,u)\in E \end{cases}$
  - **在残留网络从源到汇找一条可行路径，塞满流量**



残留网络1

网络流

- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases} c(u,v)-f(u,v), 若(u,v)\in E \\ f(v,u), \quad 若(v,u)\in E \end{cases}$
  - 在残留网络从源到汇找一条可行路径，塞满流量

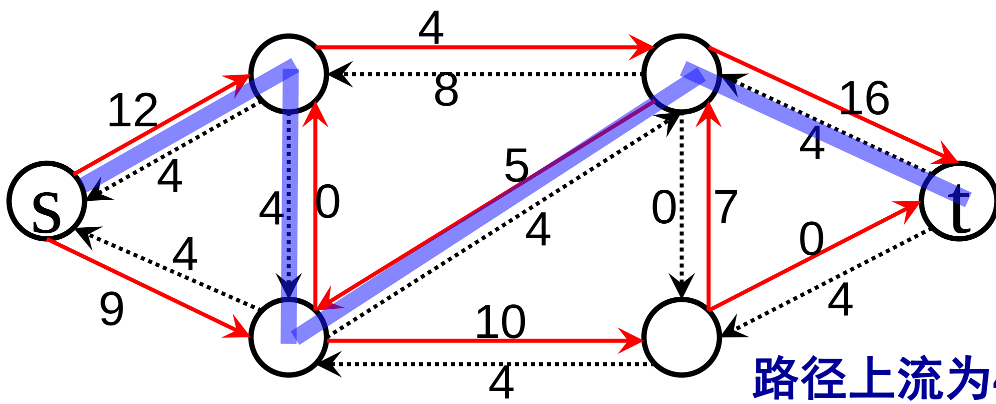


路径上流为4

残留网络2

网络流

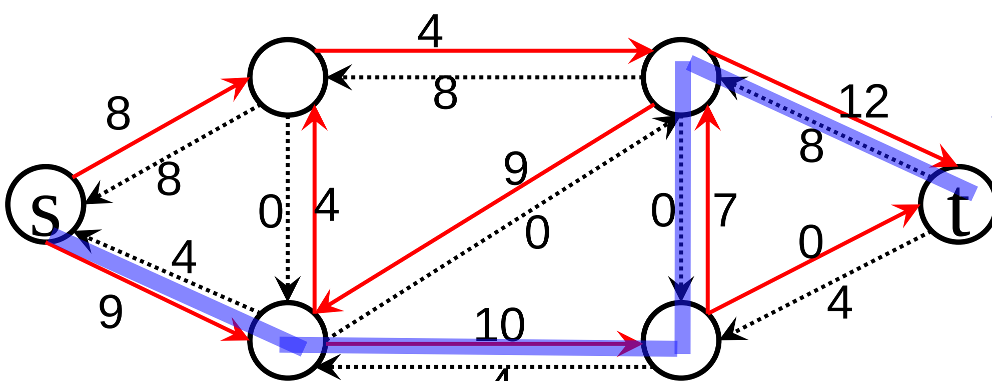
- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases}c(u,v)-f(u,v), & 若(u,v)\in E\\ f(v,u), & 若(v,u)\in E\end{cases}$
  - 在残留网络从源到汇找一条可行路径，塞满流量



残留网络3

网络流

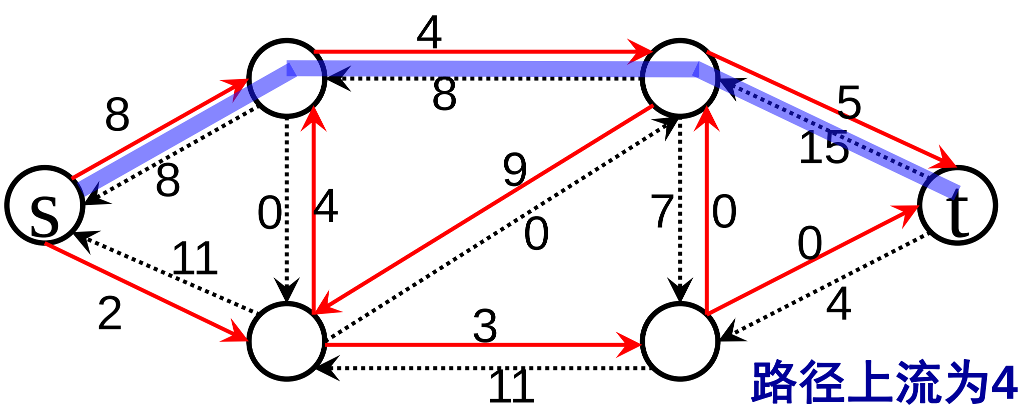
- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases} c(u,v)-f(u,v), 若(u,v)\in E \\ f(v,u), \quad\quad\quad\quad 若(v,u)\in E \end{cases}$
  - 在残留网络从源到汇找一条可行路径，塞满流量



残留网络4

网络流

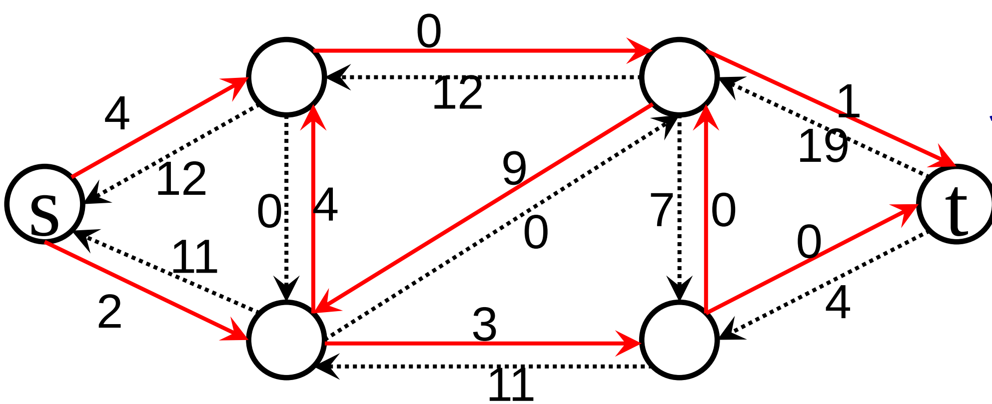
- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases} c(u,v)-f(u,v), & \text{若}(u,v)\in E \\ f(v,u), & \text{若}(v,u)\in E \end{cases}$
  - **在残留网络从源到汇找一条可行路径，塞满流量**



残留网络5

网络流

- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases}c(u,v)-f(u,v), 若(u,v)\in E\\ f(v,u), \quad\quad\quad\quad 若(v,u)\in E\end{cases}$
  - **在残留网络从源到汇找一条可行路径，塞满流量**

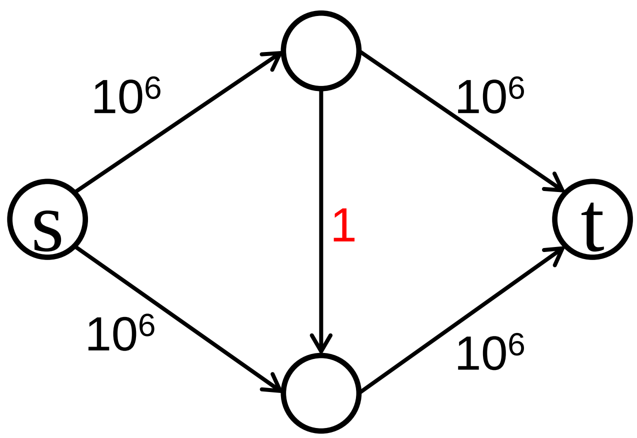


路径上流为0

残留网络6

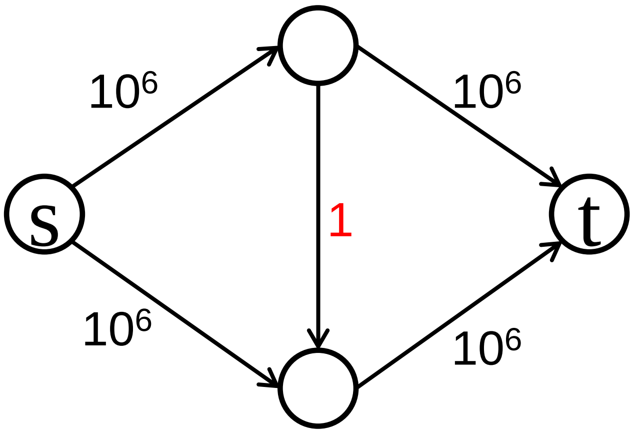
网络流

- **Fold-Fulkerson算法** $c_f(u,v)=\begin{cases}c(u,v)-f(u,v), 若(u,v)\in E\\ f(v,u), \quad 若(v,u)\in E\end{cases}$
  - 在残留网络从源到汇找一条可行路径，塞满流量
  - 遇到下面这个图，再遇到搜索可行路径时每次都经过权值为1那条边，时间复杂度很高



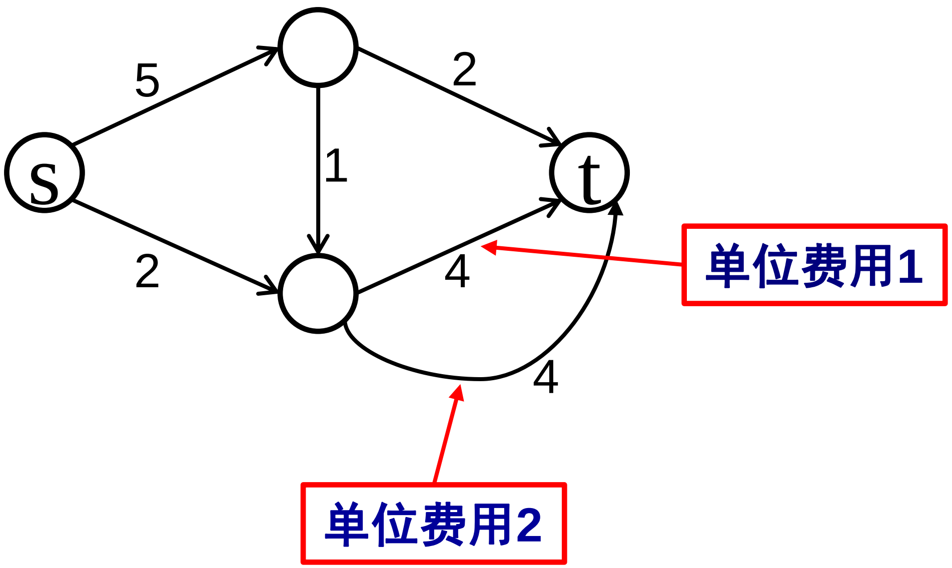
网络流

- Edmonds-Karp算法
  - 使用BFS搜索可行路径，时间复杂度$O(VE^2)$

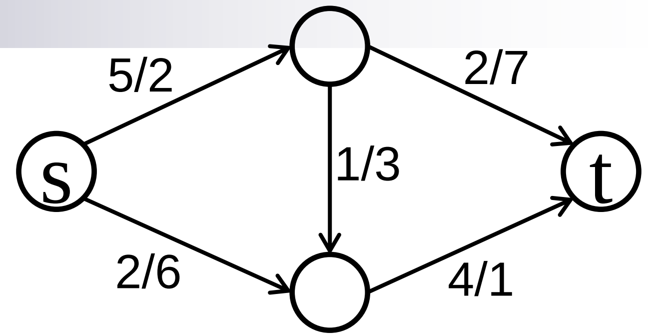


网络流

- **最小费用最大流**

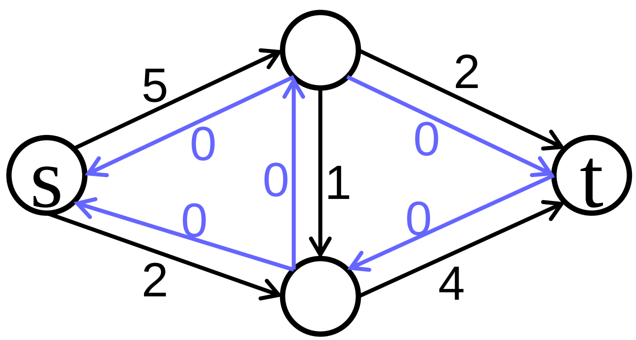


网络流

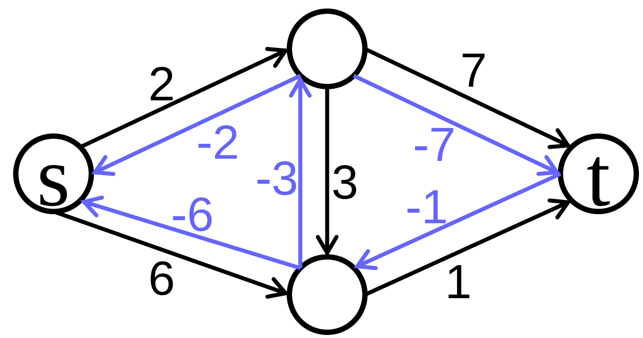


边上的(c/b)表示(容量/单位费用)

- 最小费用最大流



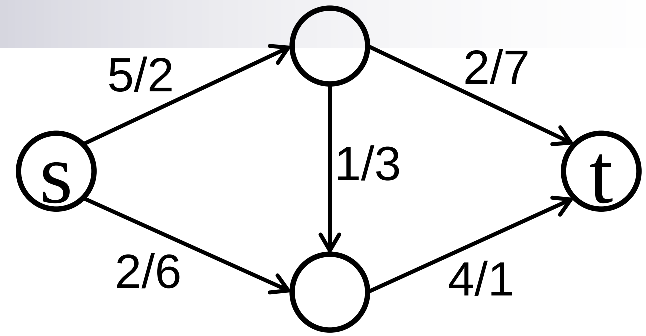
残留网络



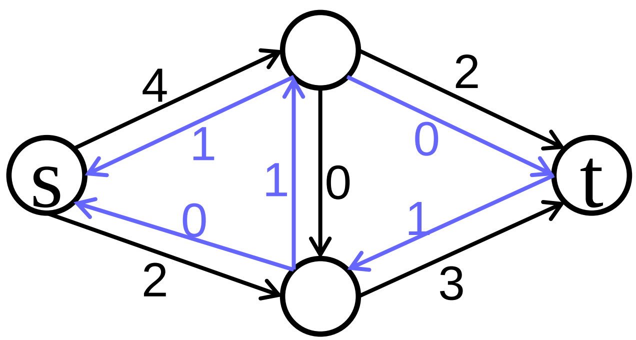
费用网络

网络流

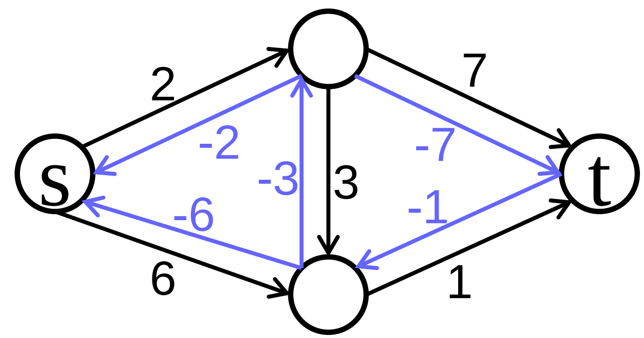
- **最小费用最大流**



边上的(c/b)表示(容量/单位费用)

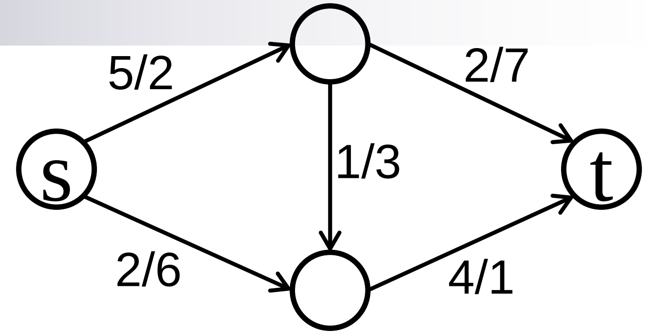


残留网络



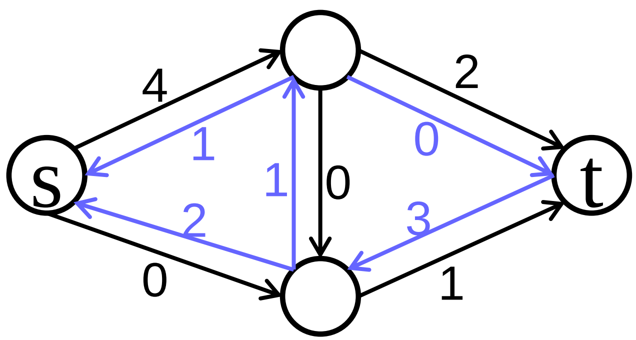
费用网络

网络流

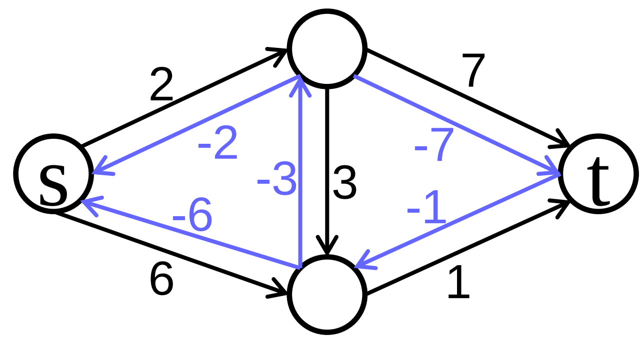


边上的(c/b)表示(容量/单位费用)

- 最小费用最大流



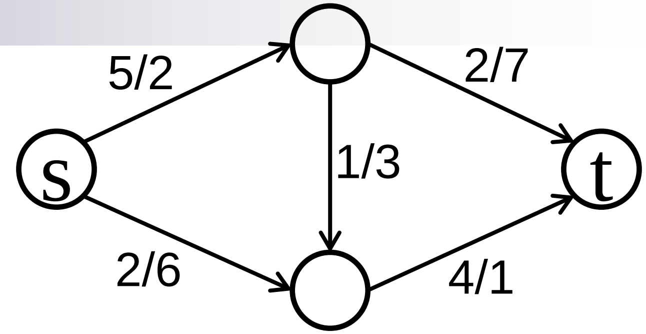
残留网络



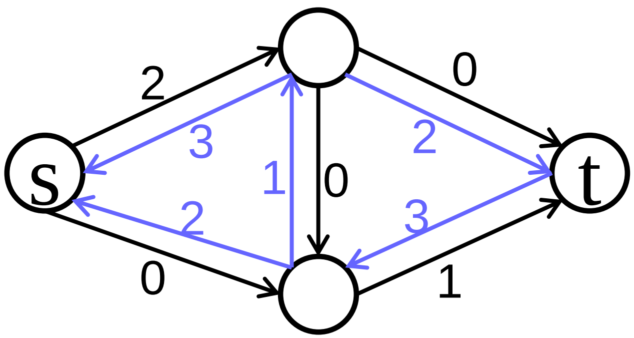
费用网络

网络流

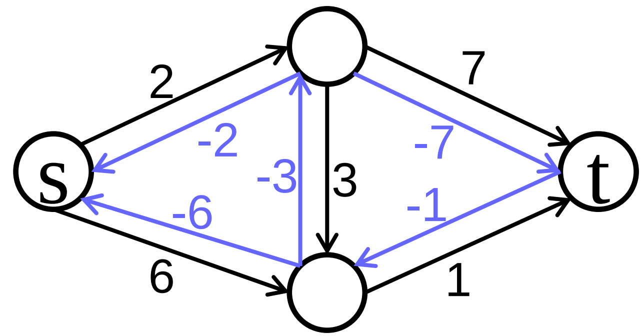
- **最小费用最大流**



边上的(c/b)表示(容量/单位费用)



残留网络

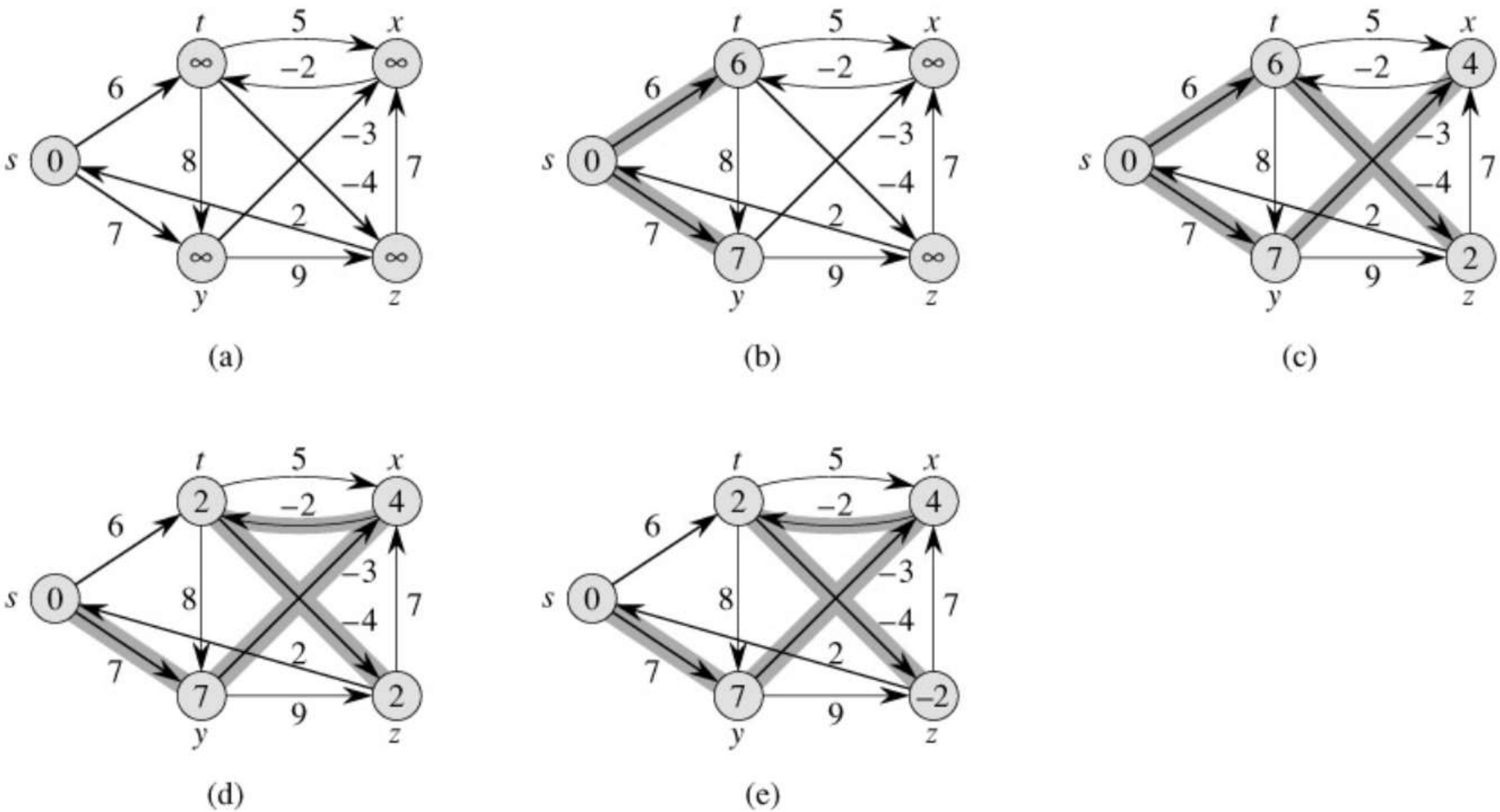


费用网络

最短路径Bellman-ford算法

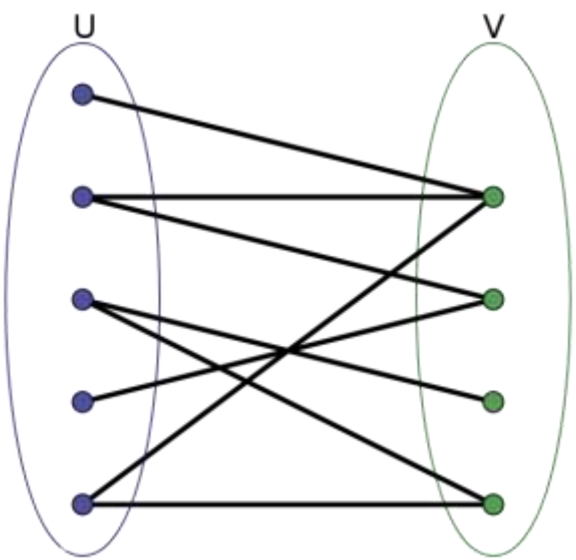
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| - 1
        do for each edge (u, v) ∈ E[G]
                do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
        do if d[v] > d[u] + w(u, v)
                then return FALSE
return TRUE
```

最短路径Bellman-ford算法



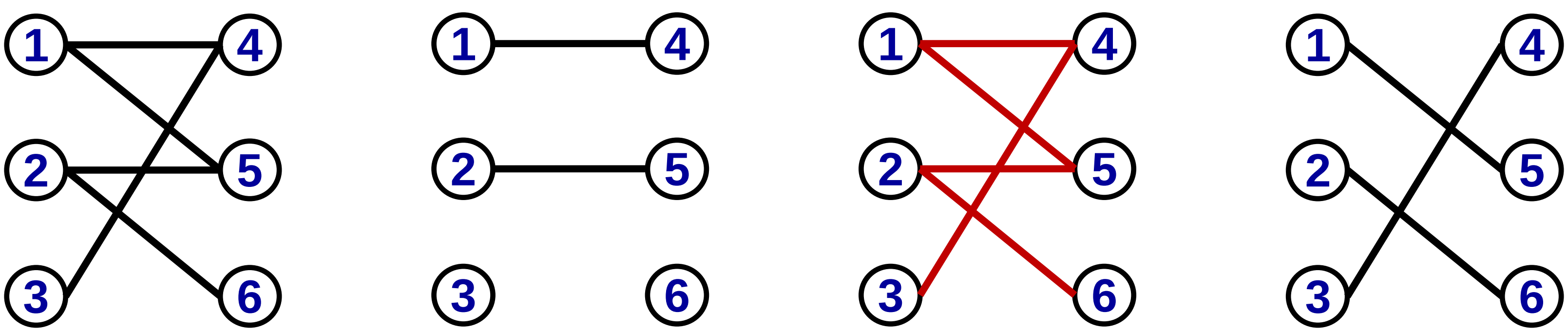
二分图匹配匈牙利算法

- 匈牙利算法是由匈牙利数学家Edmonds于1965年提出，因而得名



二分图匹配匈牙利算法

- 匈牙利算法是由匈牙利数学家Edmonds于1965年提出，因而得名
  - 介绍匈牙利算法前必须了解增广路径，匹配边，未匹配边，匹配点，未匹配点的概念
    - 增广路径就是在二分图中从未匹配点开始，按照未匹配边，匹配边交替的模式找到一个未匹配点结束。
    - 将增广路径上已匹配边断开
    - 不断寻找增广路径



POJ1274