图算法篇：最大流

北京航空航天大学
计算机学院

提纲

问题背景

算法思想

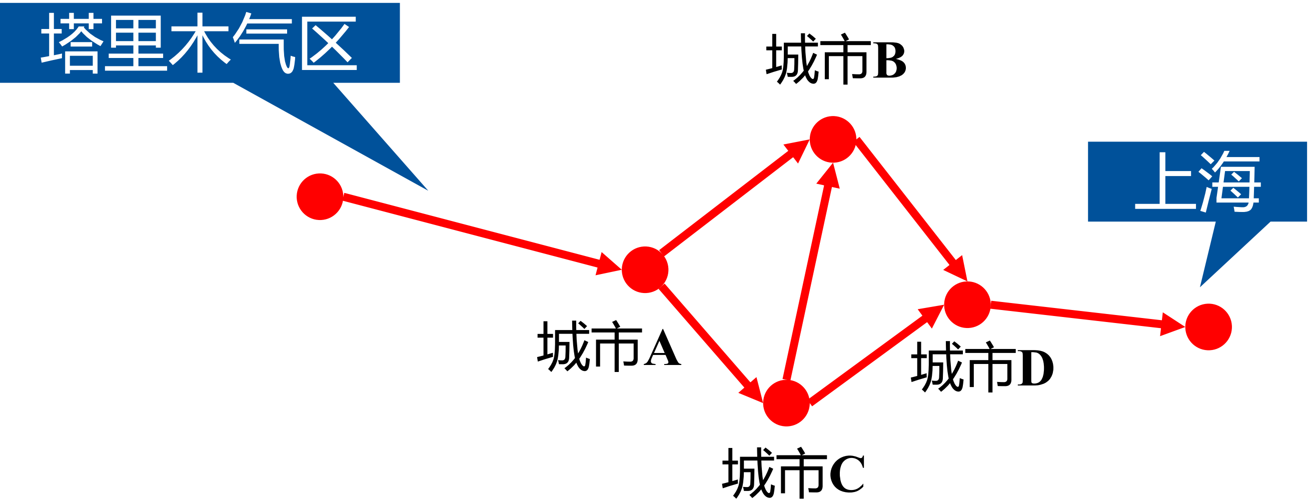
算法实例

算法分析

算法性质

问题背景：西气东输

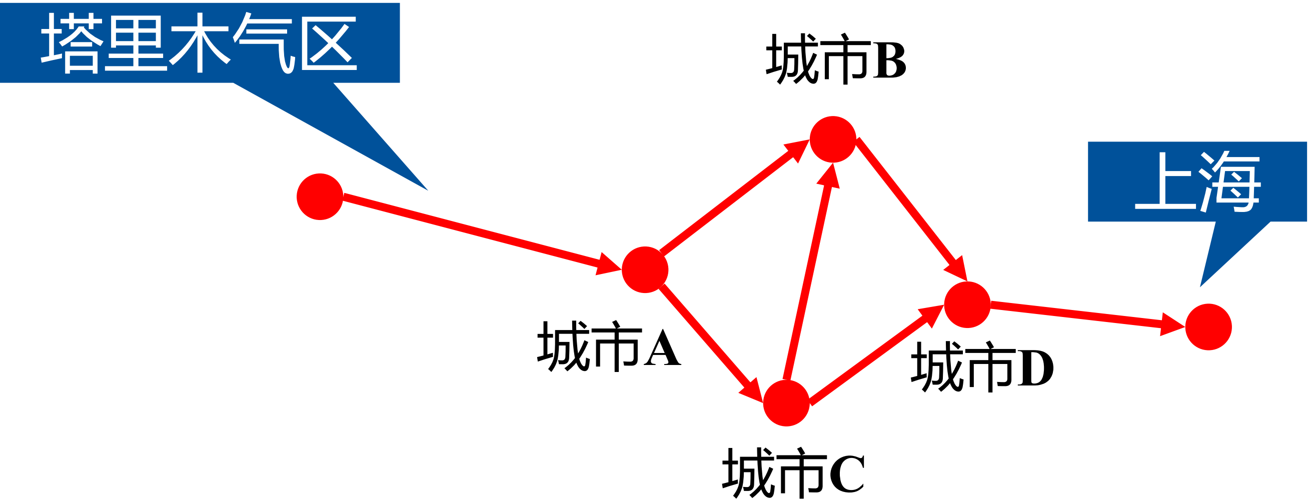
- 从塔里木气区到上海输送天然气，如何计算最大输送量？



问题背景：西气东输

- 从塔里木气区到上海输送天然气，如何计算最大输送量？

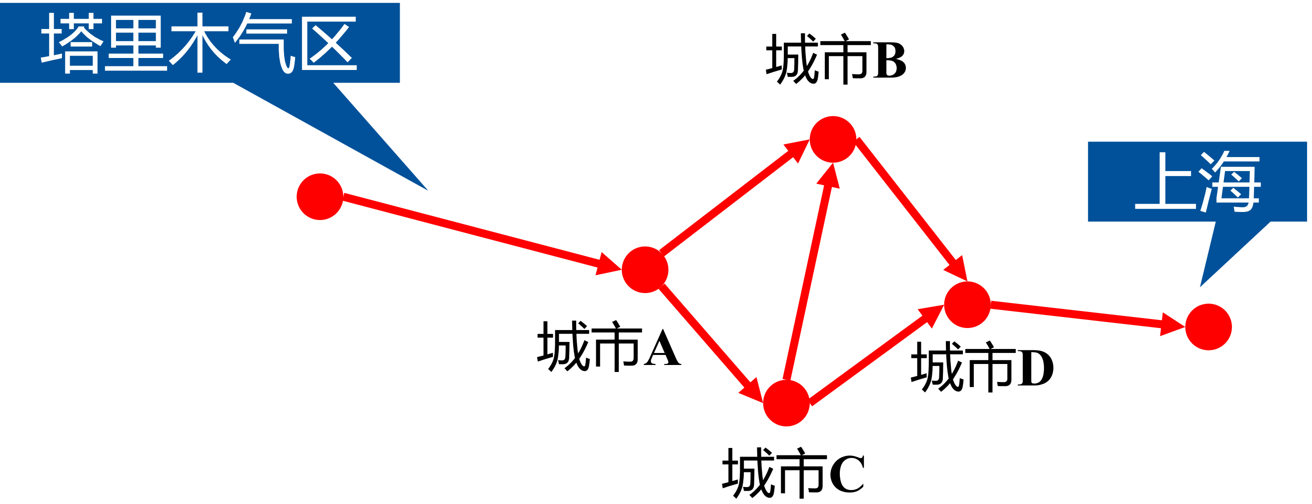
问题：如何描述输送线路？



问题背景：西气东输

- 从塔里木气区到上海输送天然气，如何计算最大输送量？

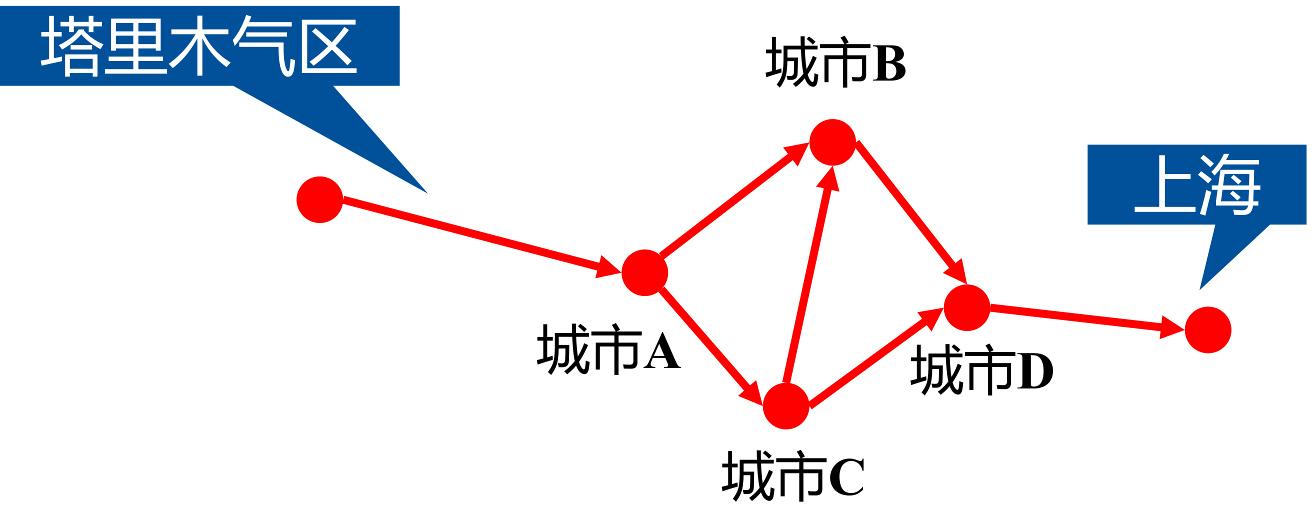
将输送线路抽象为有向图

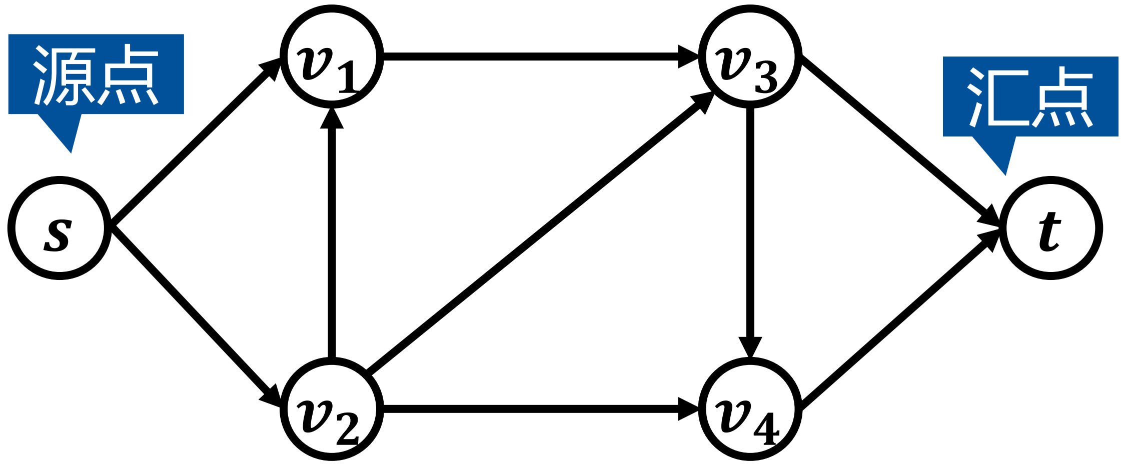


问题背景：西气东输

- 从塔里木气区到上海输送天然气，如何计算最大输送量？

城市视为顶点，管道视为边



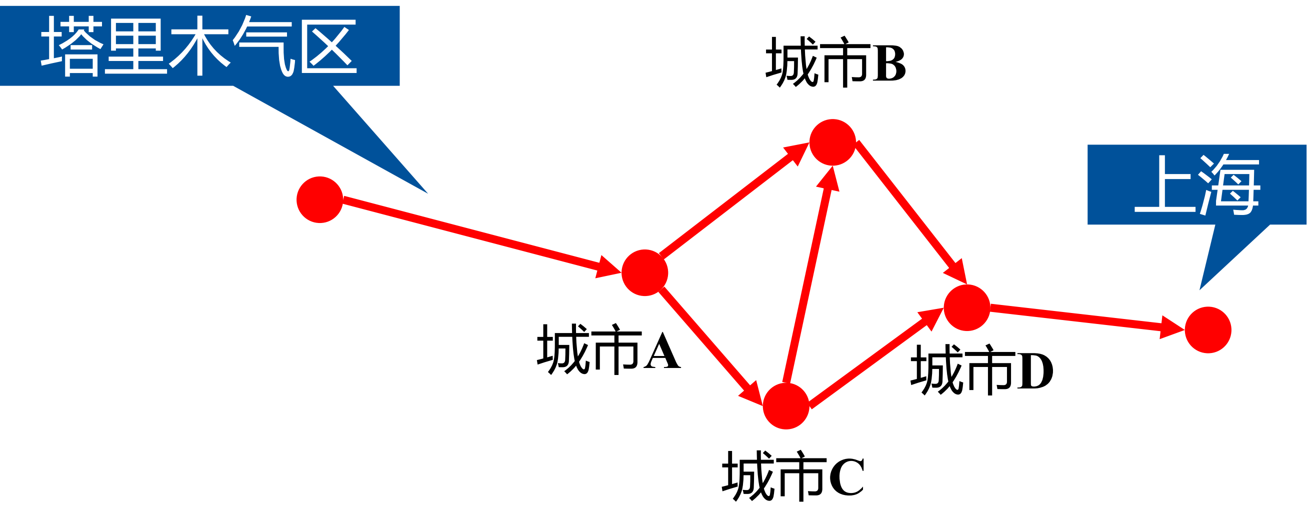


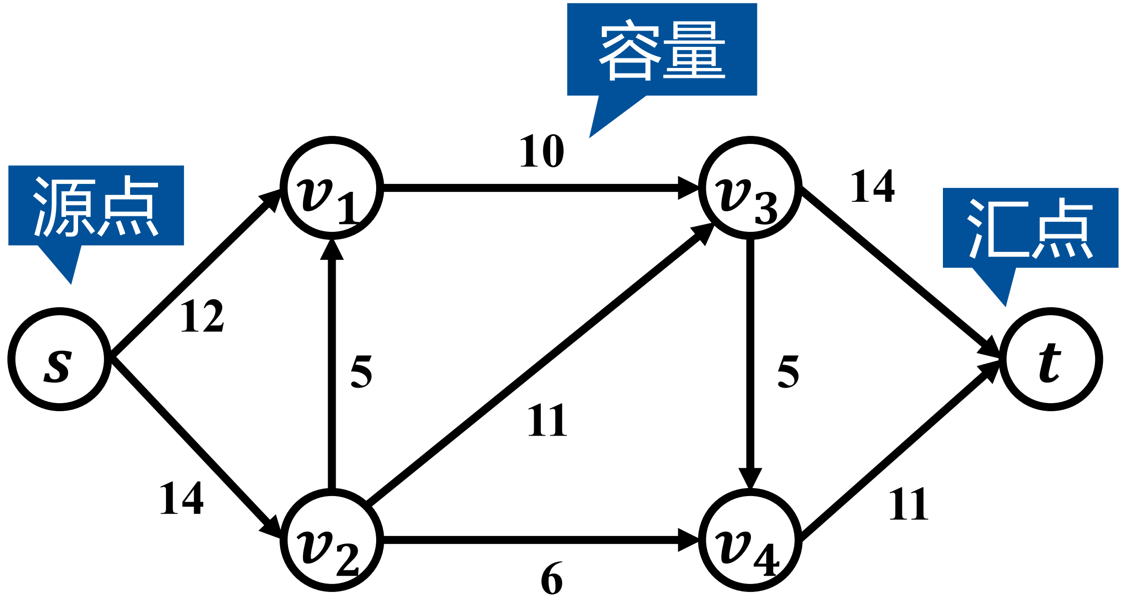
流网络$G = < V, E >$

问题背景：西气东输

- 从塔里木气区到上海输送天然气，如何计算最大输送量？

用边的权值表示管道的容量

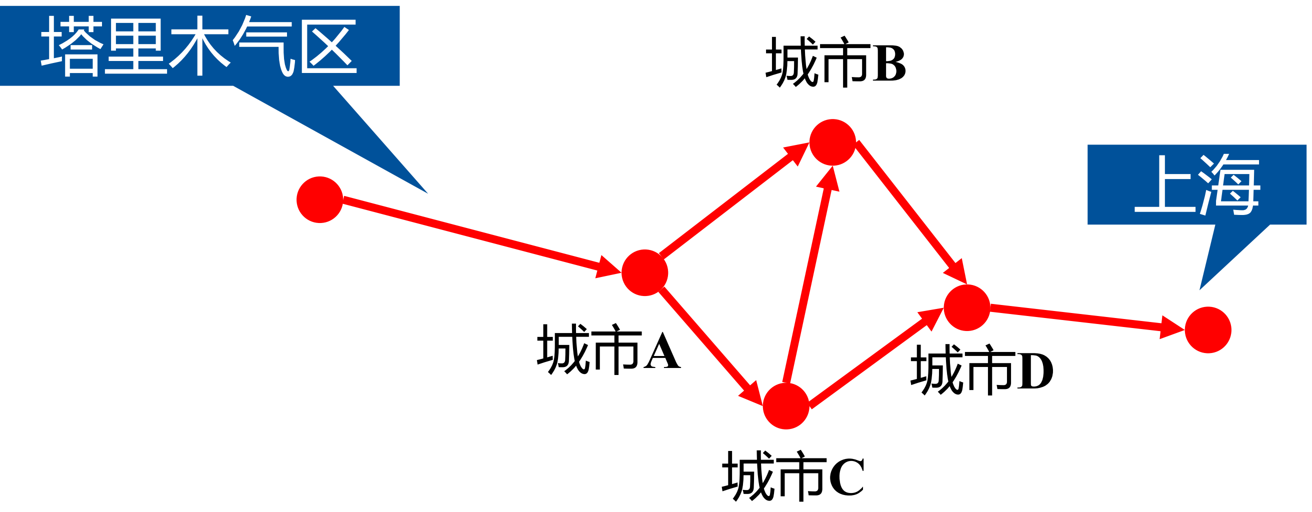




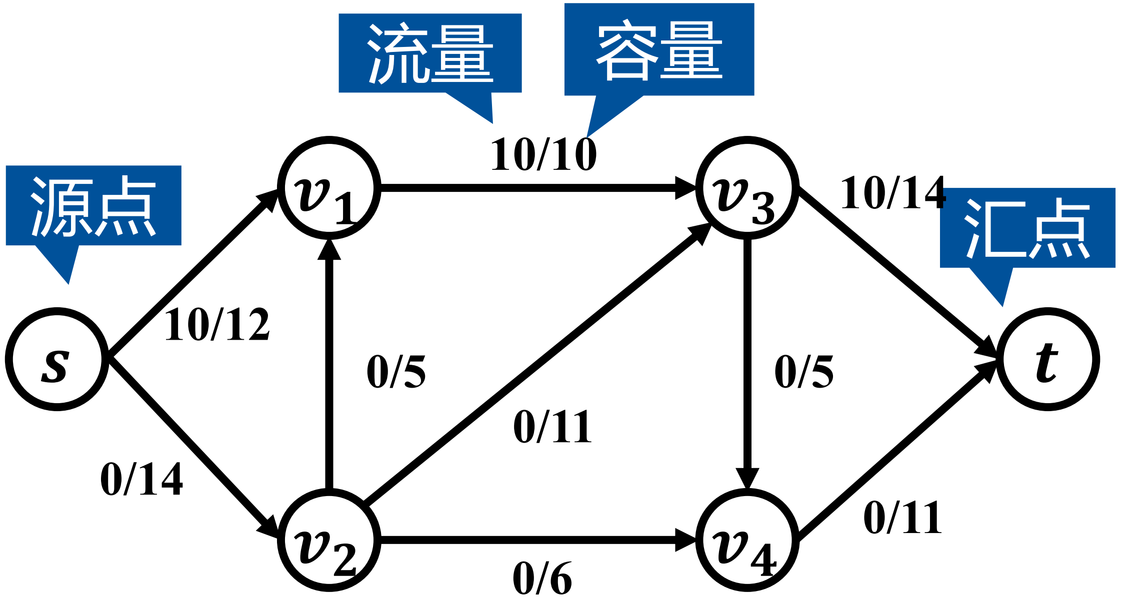
流网络$G = < V, E >$

问题背景：西气东输

- 从塔里木气区到上海输送天然气，如何计算最大输送量？



用边的权值表示管道的容量

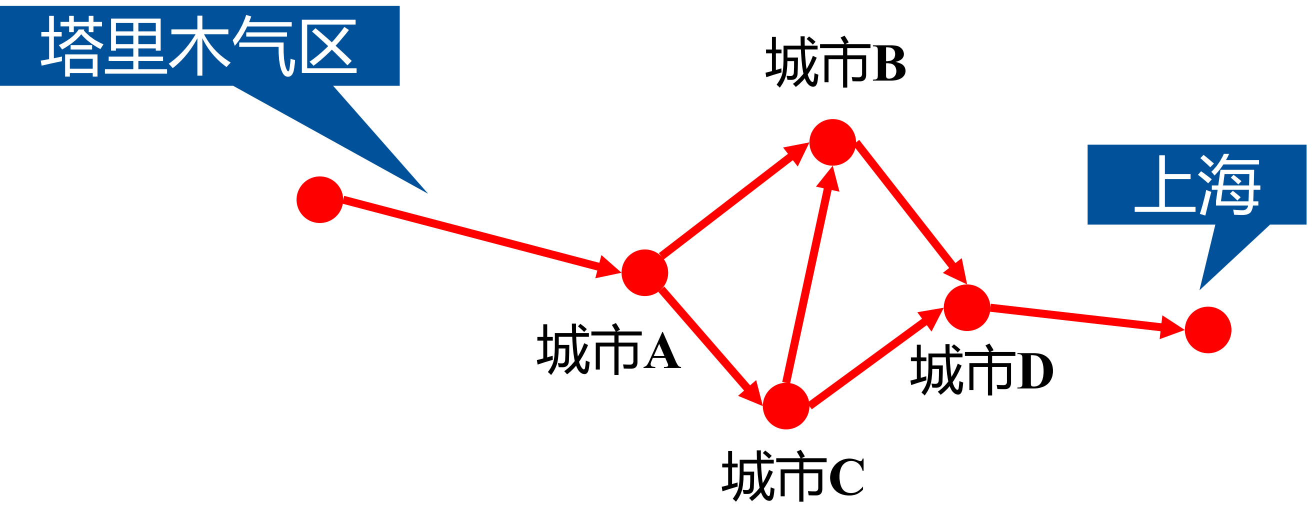


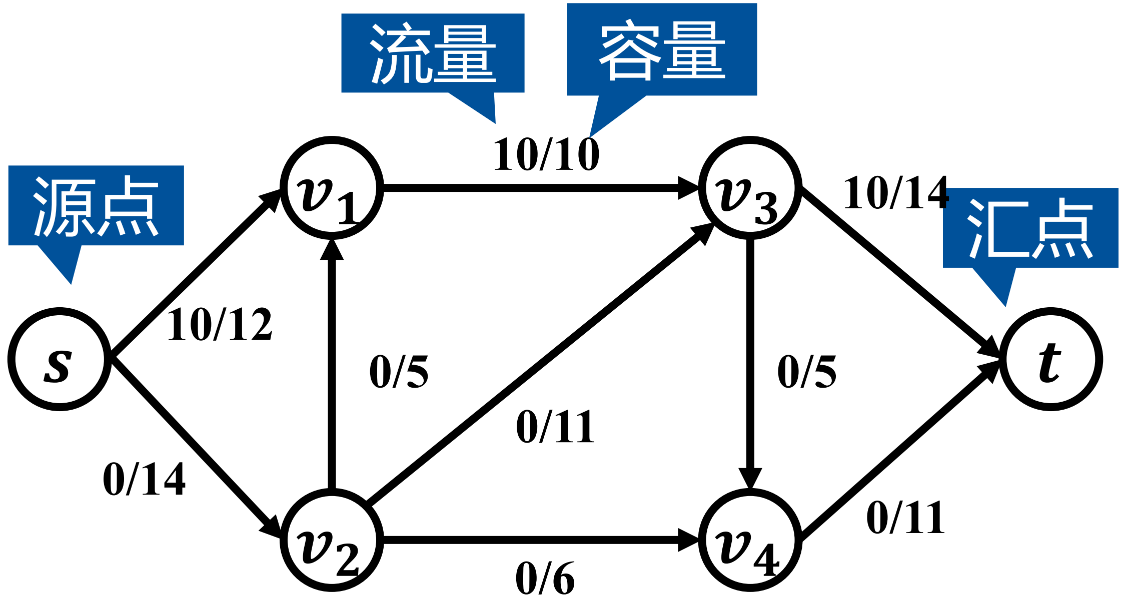
流网络$G =< V, E >$

路径上的流量不应超过边的容量

问题背景：西气东输

- 从塔里木气区到上海输送天然气，如何计算最大输送量？



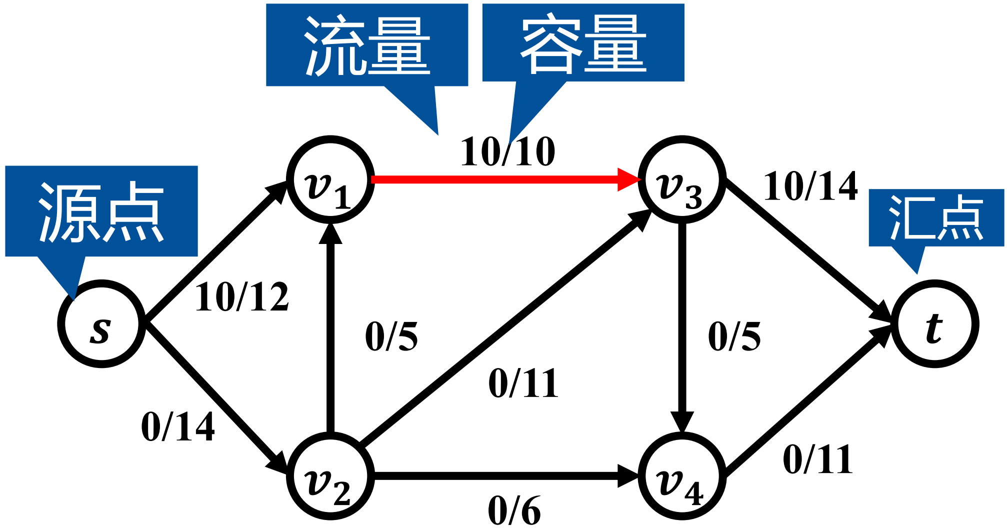


流网络$G = < V, E >$

计算最大天然气输送量（最大化总流量）

流网络

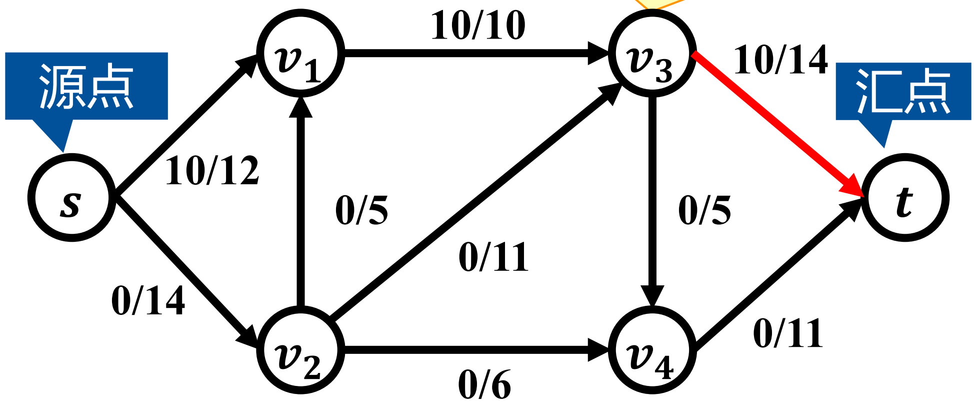
- 给定有向图 $\boldsymbol{G} = < \boldsymbol{V}, \boldsymbol{E}, \boldsymbol{C} >$，其被称为流网络：
  - 容量：对于每条边 $\boldsymbol{e} \in \boldsymbol{E}$，$\boldsymbol{c}(\boldsymbol{e}) \geq \boldsymbol{0}$
  - 流量：对于每条边 $\boldsymbol{e} \in \boldsymbol{E}$，$\boldsymbol{c}(\boldsymbol{e}) \geq \boldsymbol{f}(\boldsymbol{e}) \geq \boldsymbol{0}$



流网络

- 给定有向图 $G=<V,E,C>$，其被称为流网络：
  - 容量：对于每条边 $e \in E$，$c(e) \geq 0$
  - 流量：对于每条边 $e \in E$，$c(e) \geq f(e) \geq 0$
  - 剩余容量：对于每条边，剩余容量为 $c(e) - f(e)$（剩余容量=容量-流量）

剩余容量=14-10=4



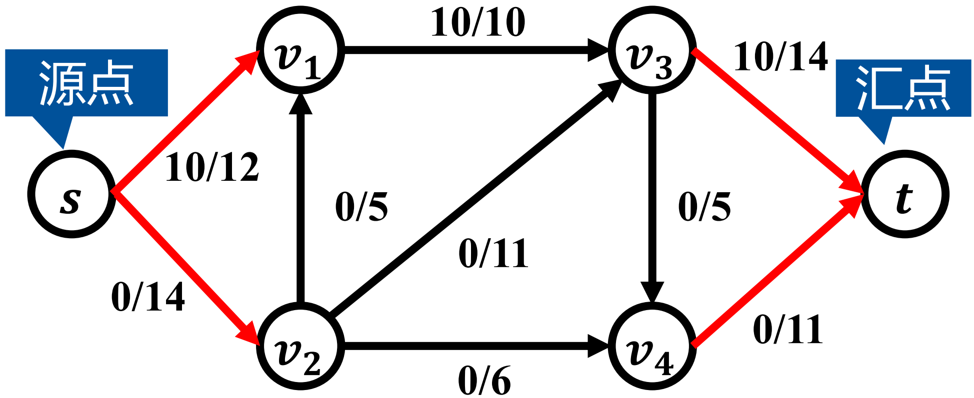
总流量：10

流网络

- 给定有向图 $G = < V, E, C >$，其被称为流网络：
  - 容量：对于每条边 $e \in E$，$c(e) \geq 0$
  - 流量：对于每条边 $e \in E$， $c(e) \geq f(e) \geq 0$
  - 剩余容量：对于每条边，剩余容量为 $c(e) - f(e)$（剩余容量=容量-流量）
  - 总流量：

$$|f| = \sum_{e\ out\ of\ s} f(e) = \sum_{e\ in\ to\ t} f(e)$$

（总流量 = 源点流出量 = 汇点流入量）



总流量：10

流网络

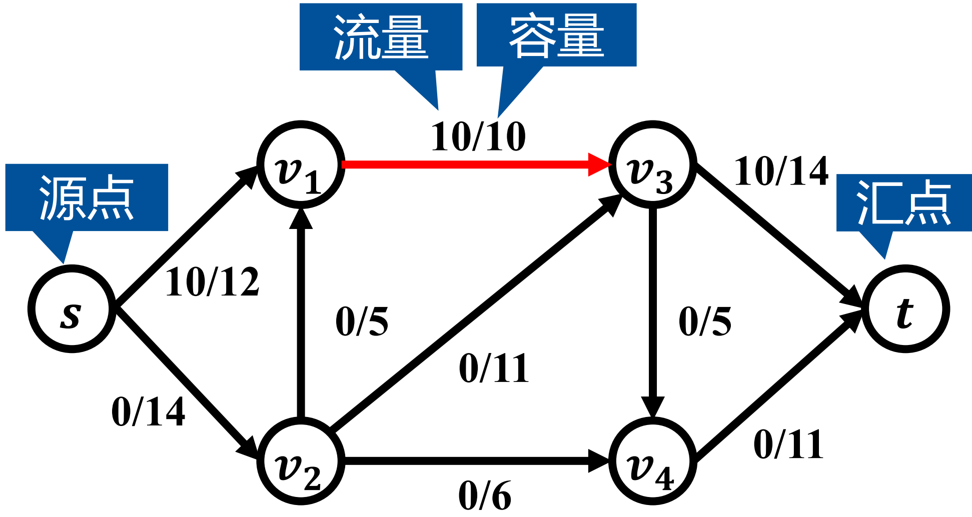
- 给定有向图 $G=<V,E,C>$，其被称为流网络：
  - 容量：对于每条边 $e\in E$，$c(e)\geq 0$
  - 流量：对于每条边 $e\in E$，$c(e)\geq f(e)\geq 0$
  - 剩余容量：对于每条边，剩余容量为 $c(e)-f(e)$（剩余容量=容量-流量）
  - 总流量：

$$|f|=\sum_{e\ out\ of\ s} f(e)=\sum_{e\ in\ to\ t} f(e)$$

（总流量 = 源点流出量 = 汇点流入量）

- 流量的两条性质
  - 容量限制：对边 $e\in E$，有 $0\leq f(e)\leq c(e)$

（边上的流量不应超过边的容量）

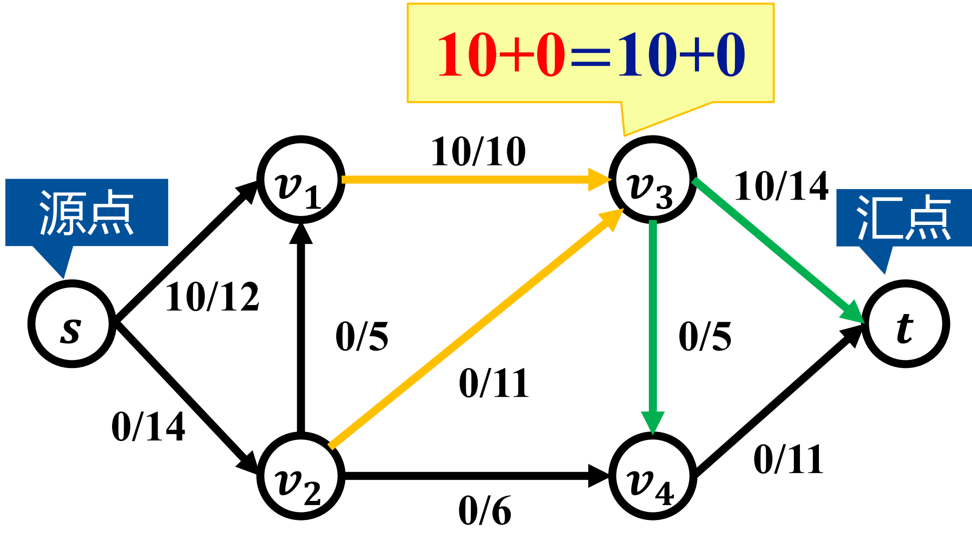


流网络

- 给定有向图 $G=<V,E,C>$，其被称为流网络：
  - 容量：对于每条边 $e\in E$，$c(e)\geq 0$
  - 流量：对于每条边 $e\in E$， $c(e)\geq f(e)\geq 0$
  - 剩余容量：对于每条边，剩余容量为 $c(e)-f(e)$（剩余容量=容量-流量）
  - 总流量：

$$|f|=\sum_{e\ out\ of\ s}f(e)=\sum_{e\ in\ to\ t}f(e)$$

（总流量 = 源点流出量 = 汇点流入量）



- 流量的两条性质
  - 容量限制：对边 $e\in E$，有 $0\leq f(e)\leq c(e)$

    （边上的流量不应超过边的容量）
  - 流量守恒：对顶点$v\in V-\{s,t\}$

$$\sum_{e\ in\ to\ v}f(e)=\sum_{e\ out\ of\ v}f(e)$$

（进入某顶点 v 流量和 = 流出此顶点流量和）

问题定义

最大流问题

Maximum Flow Problem

输入

- 有向图$G=<V,E,C>$，其中$c(e)\in C$表示边 e 的容量
- 源点s，汇点t

输出

- 总流量 $|f|$

优化目标

$$max\ |f| = max \sum_{e\ out\ of\ s} f(e)$$

约束条件

$$s.t.\ \ 0 \le f(e) \le c(e), \sum_{e\ in\ to\ v} f(e) = \sum_{e\ out\ of\ v} f(e)$$

容量限制

流量守恒

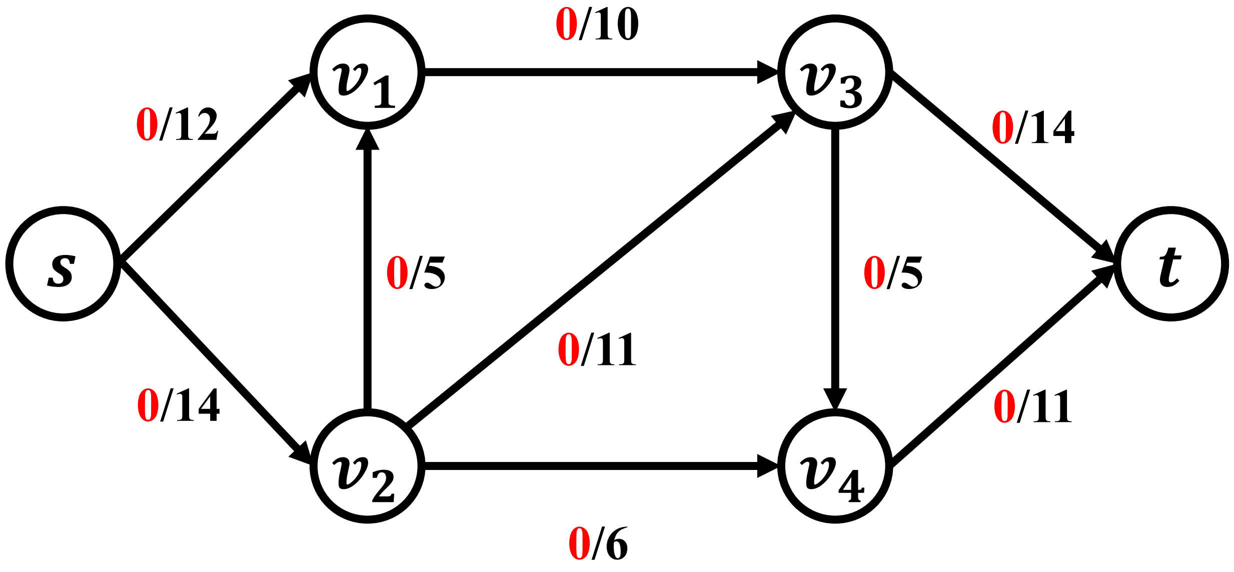
提纲

- 问题背景
- 算法思想
- 算法实例
- 算法分析
- 算法性质

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$

流网络G

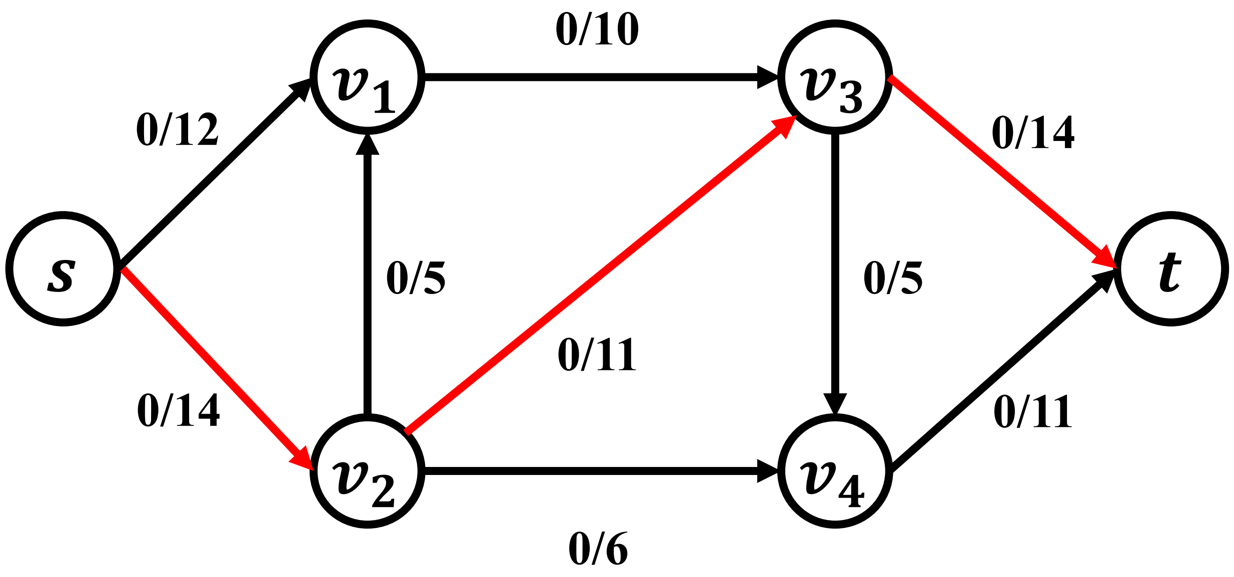


总流量：0

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$

流网络G

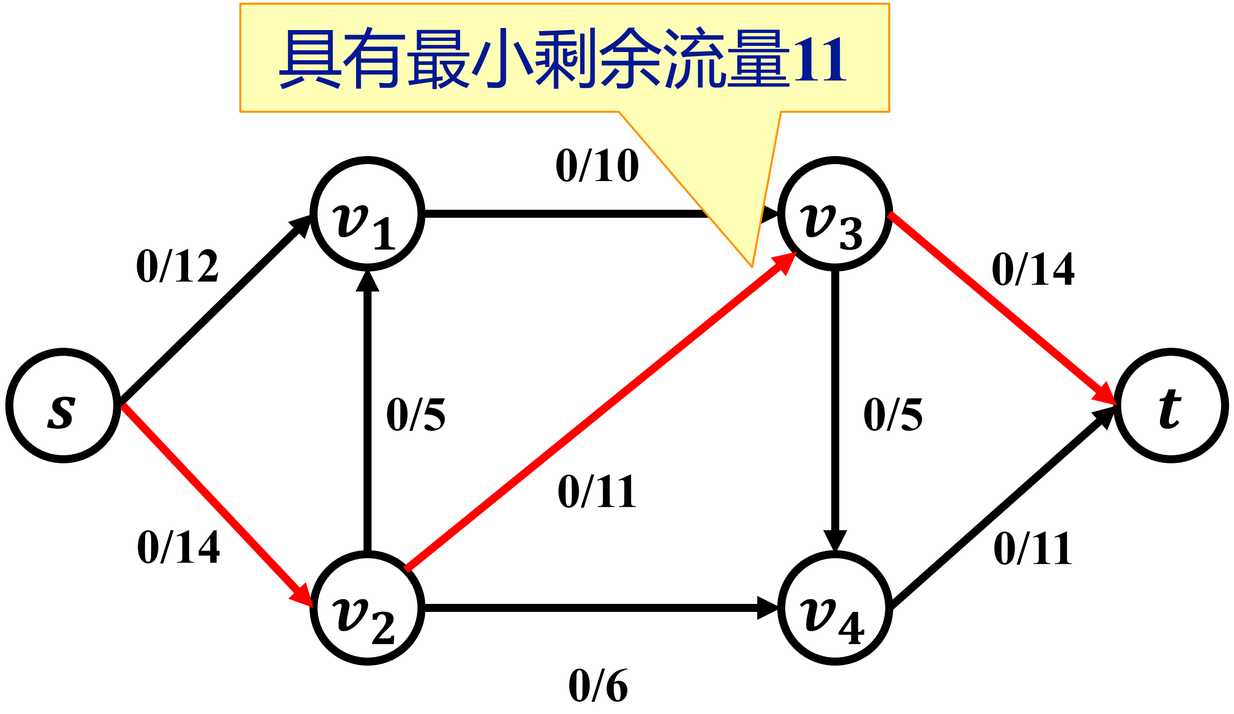


总流量：0

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量

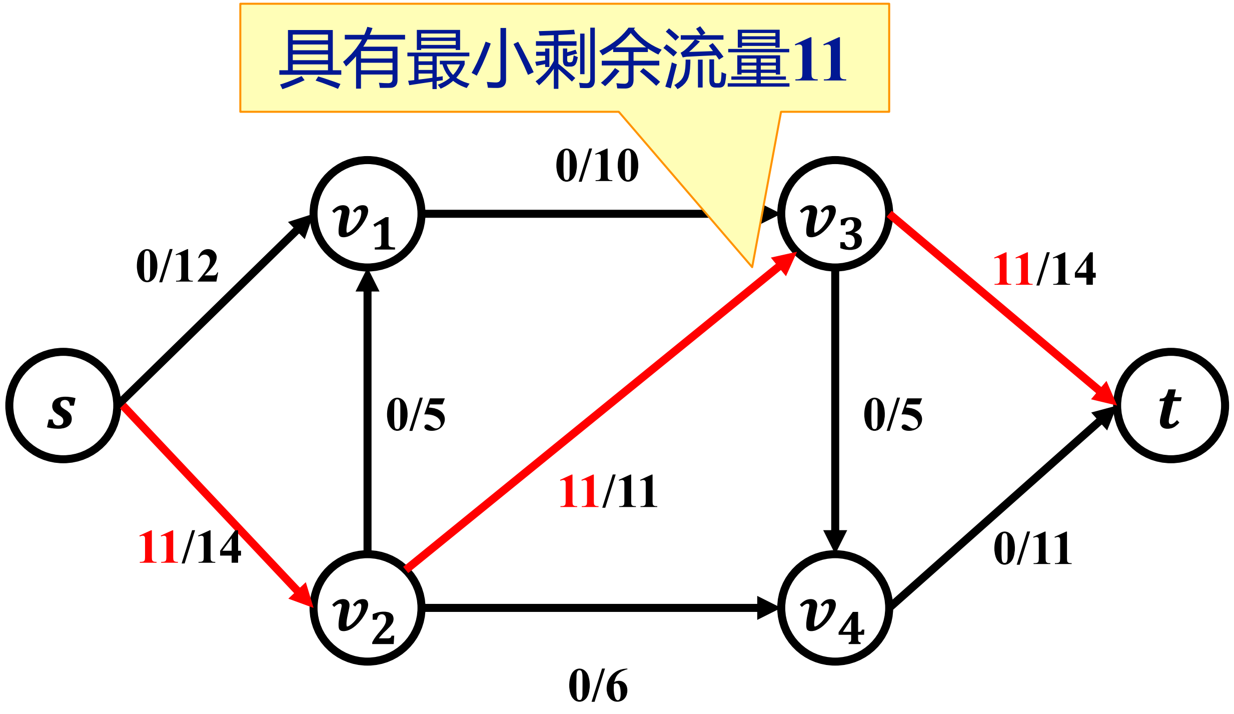
流网络G



总流量：0

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量

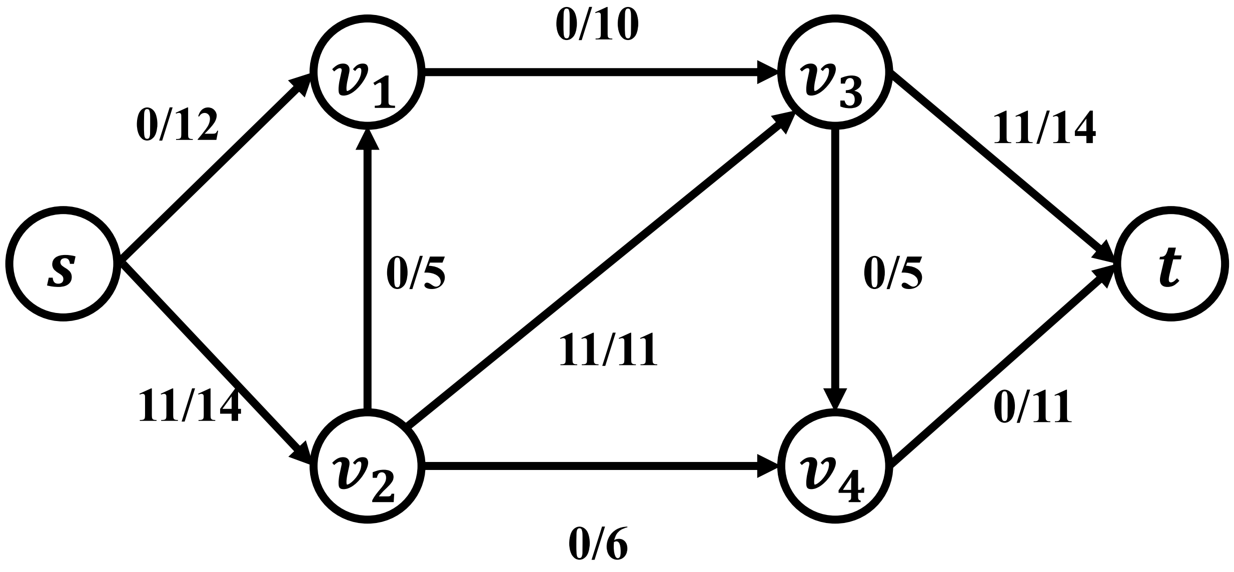


总流量：11

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

流网络G

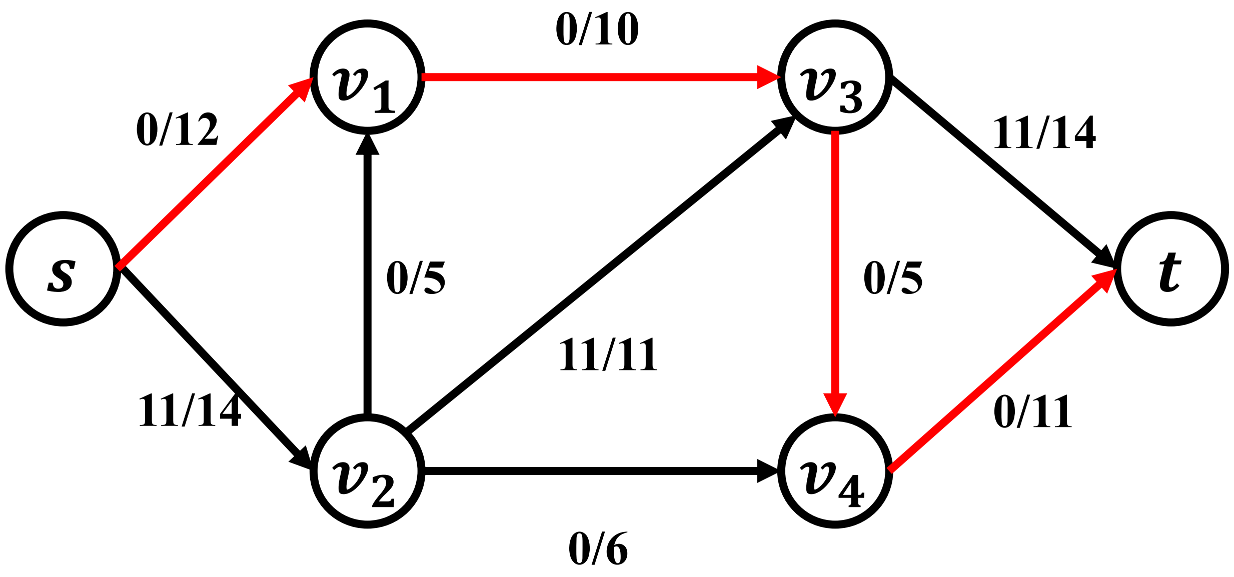


总流量：11

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

流网络G

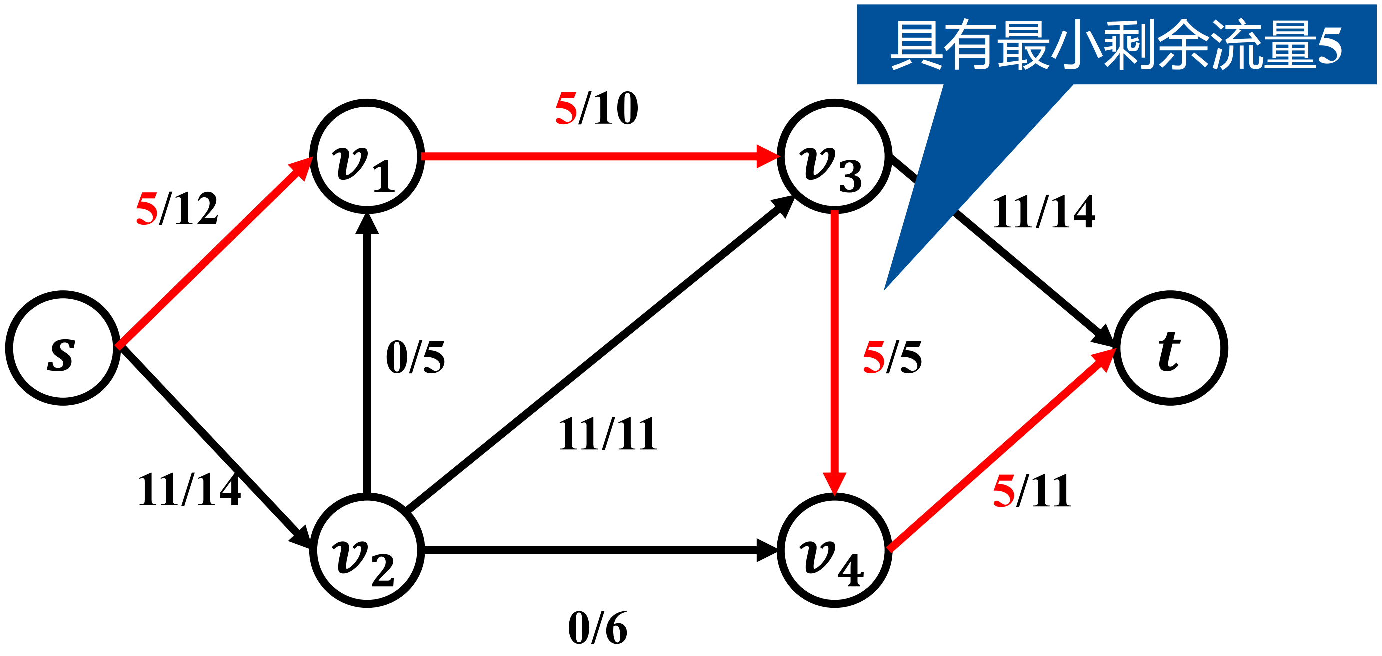


总流量：11

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

流网络G

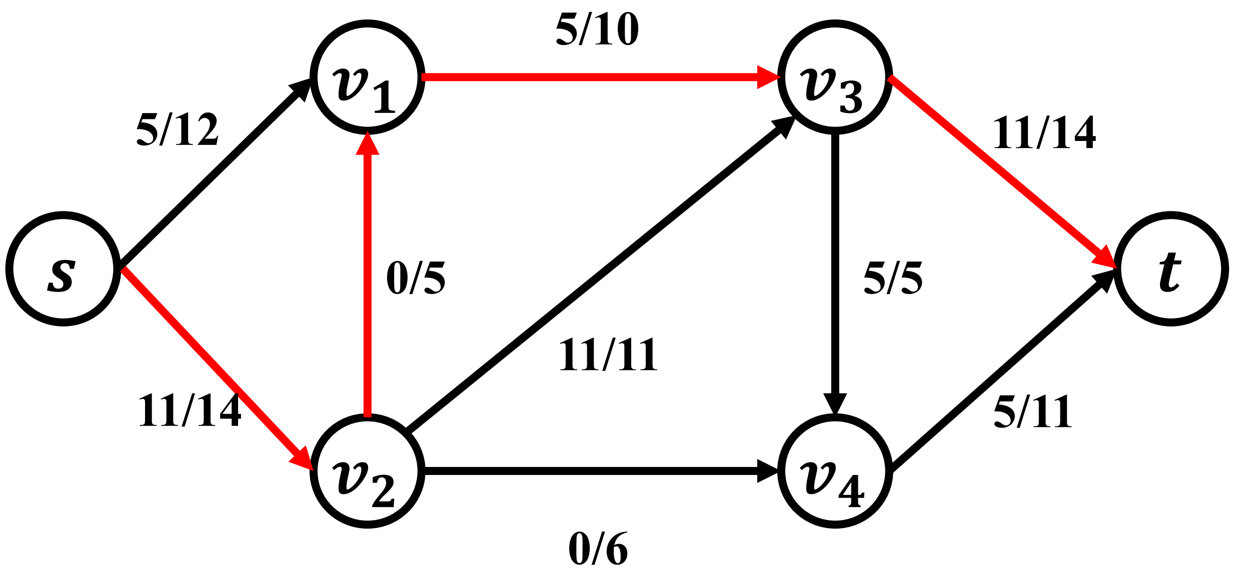


总流量：16

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

流网络G

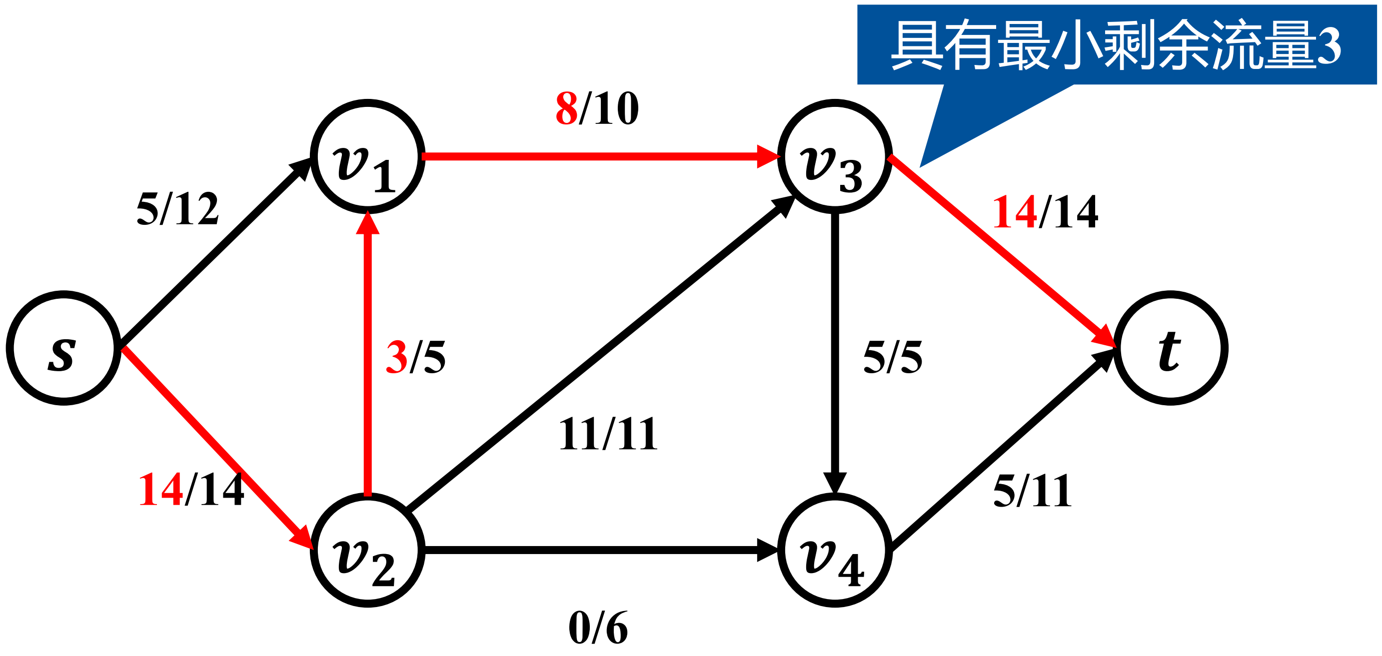


总流量：16

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

流网络G

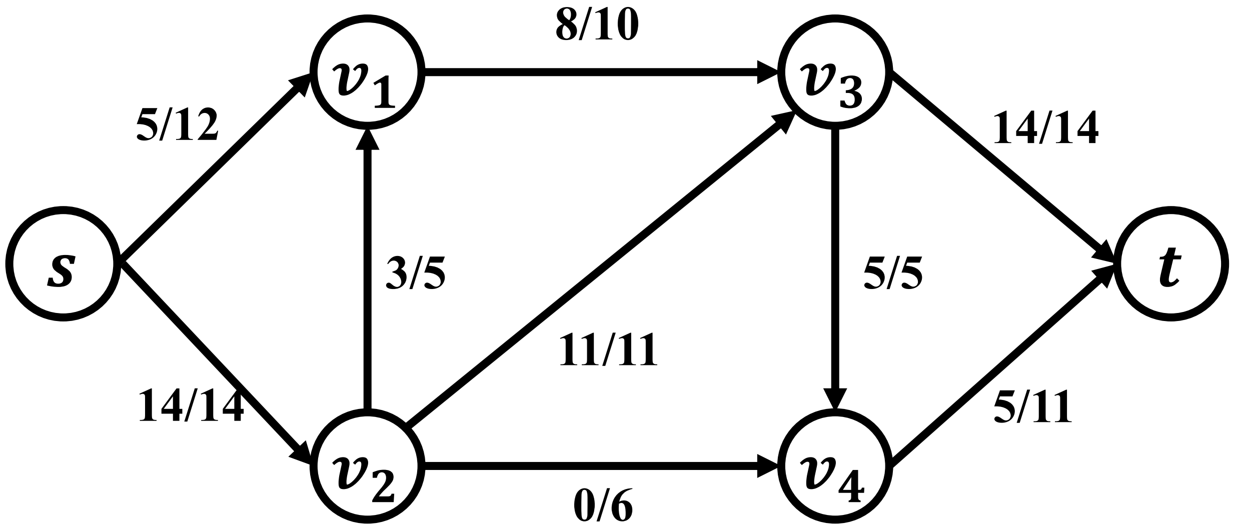


总流量：19

直观策略

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

流网络G

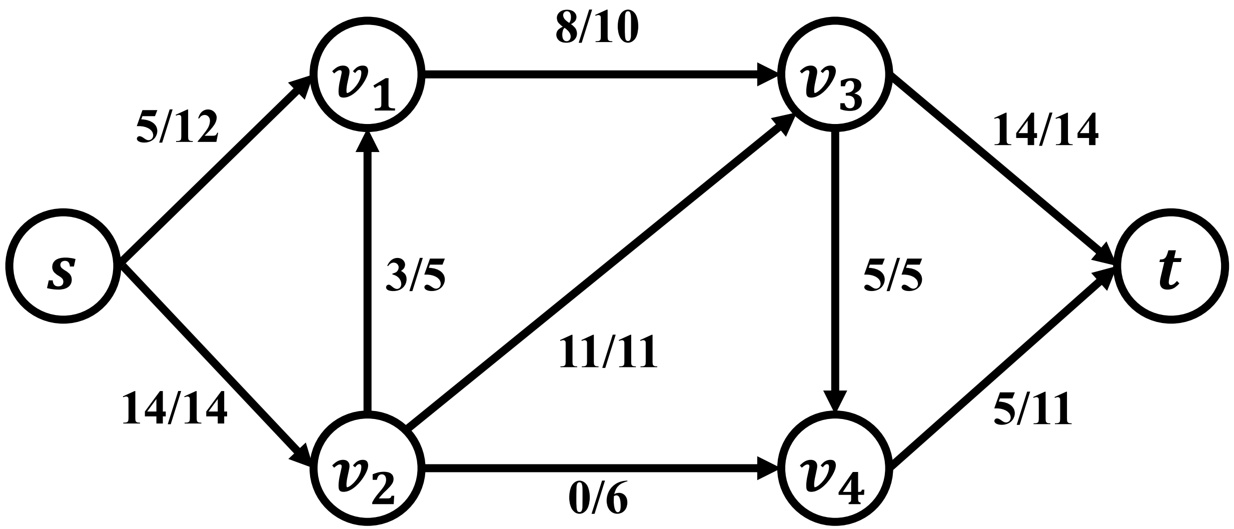


总流量：19

无法寻找到可增加流量的路径

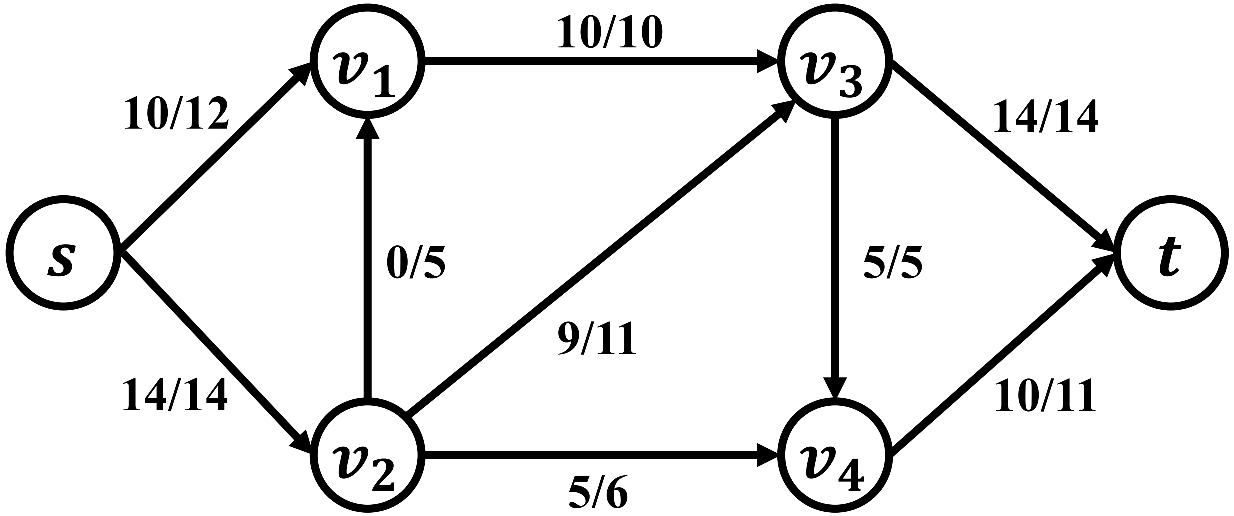
直观策略：比较观察

- 直观策略



总流量：19

- 最优方案



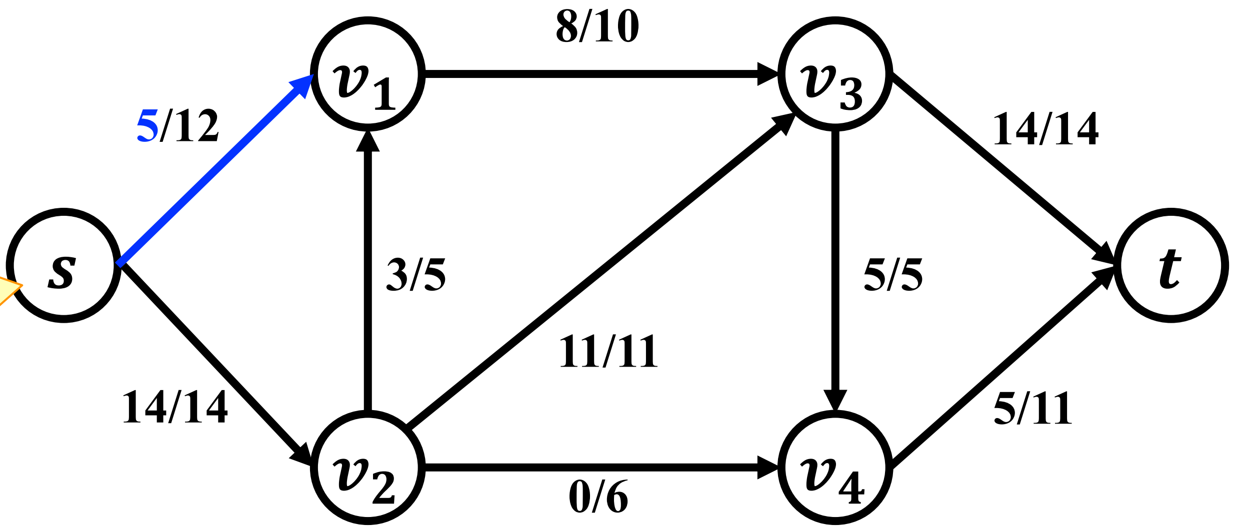
最大总流量：24

直观策略可能达不到最优方案，观察解的区别

直观策略：比较观察

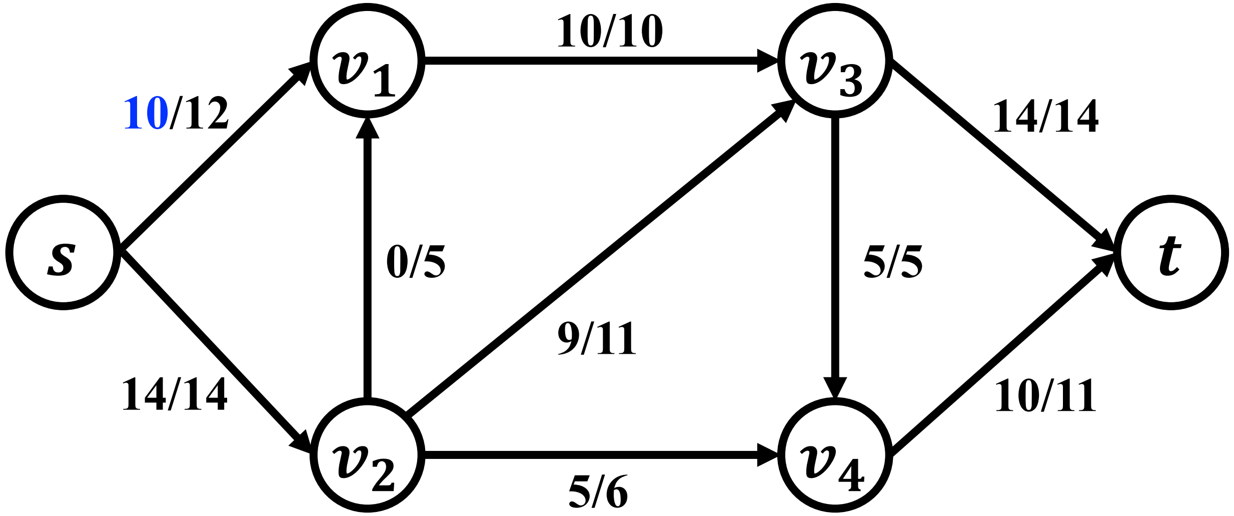
- 直观策略

$$|f| = \sum_{e\ out\ of\ s} f(e)$$



总流量：19

- 最优方案

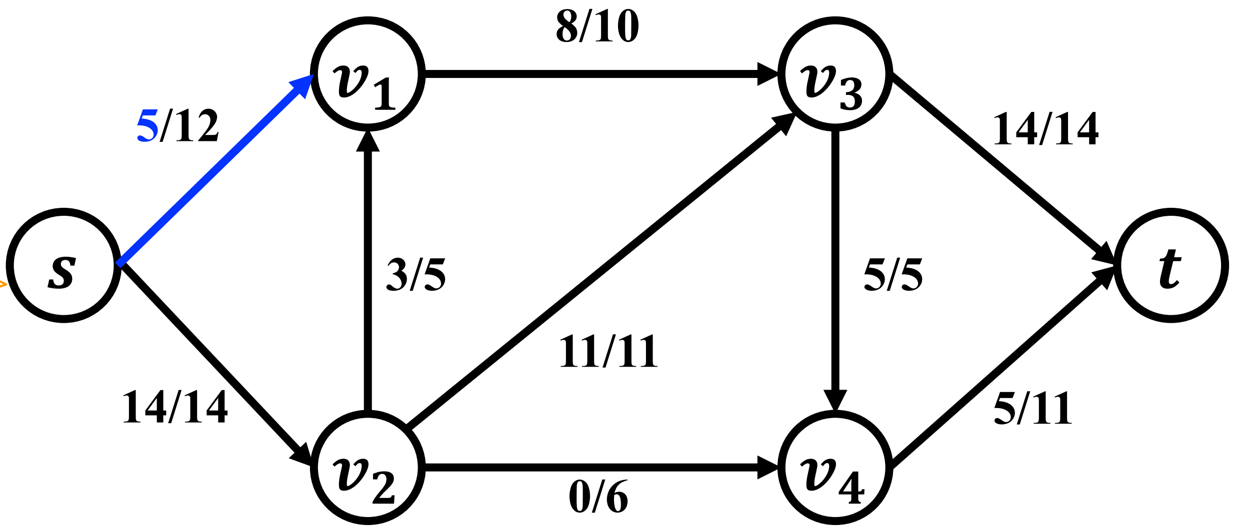


最大总流量：24

直观策略：比较观察

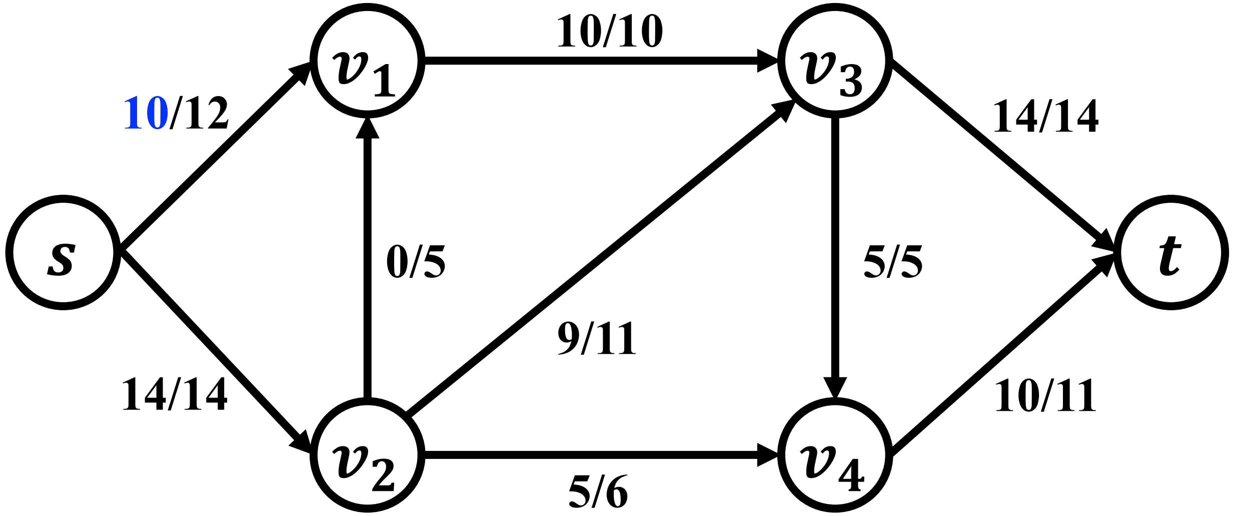
- 直观策略

容量未充分利用



总流量：19

- 最优方案

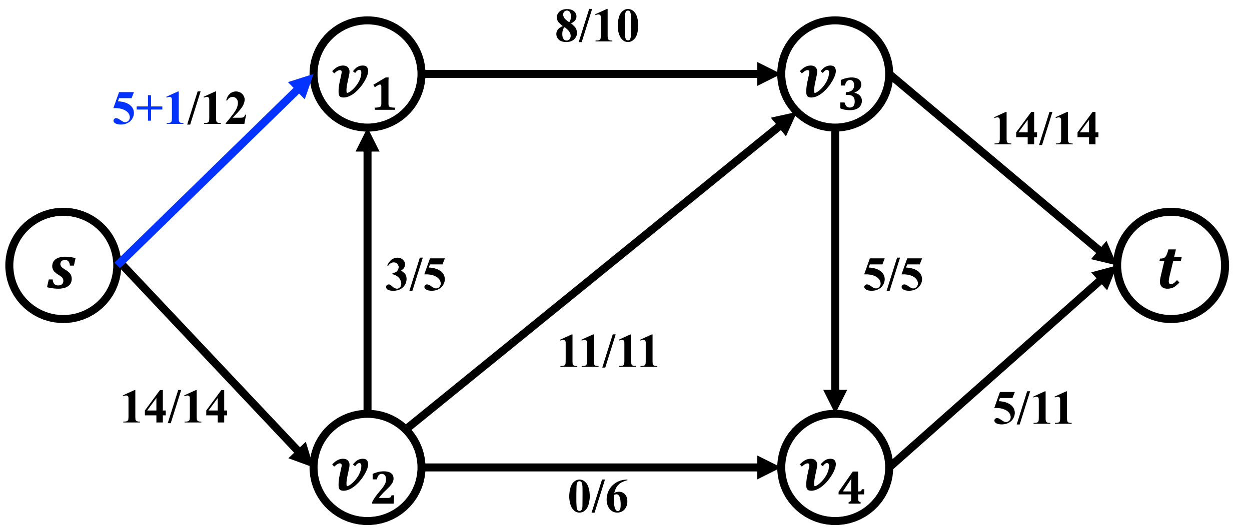


最大总流量：24

直观策略：比较观察

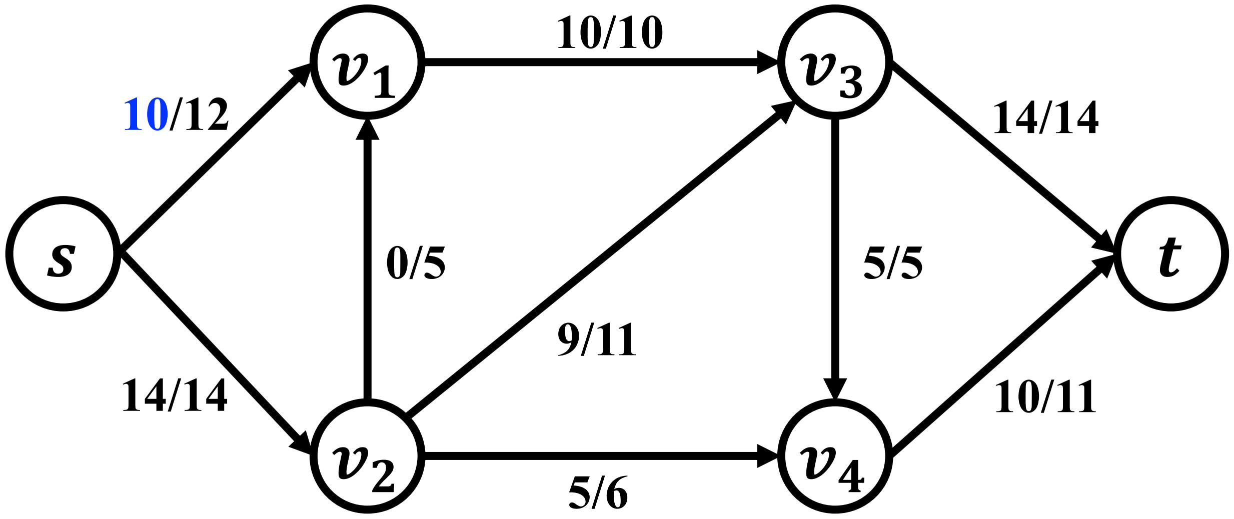
- 直观策略

尝试扩充
一个单位流量



总流量：19

- 最优方案

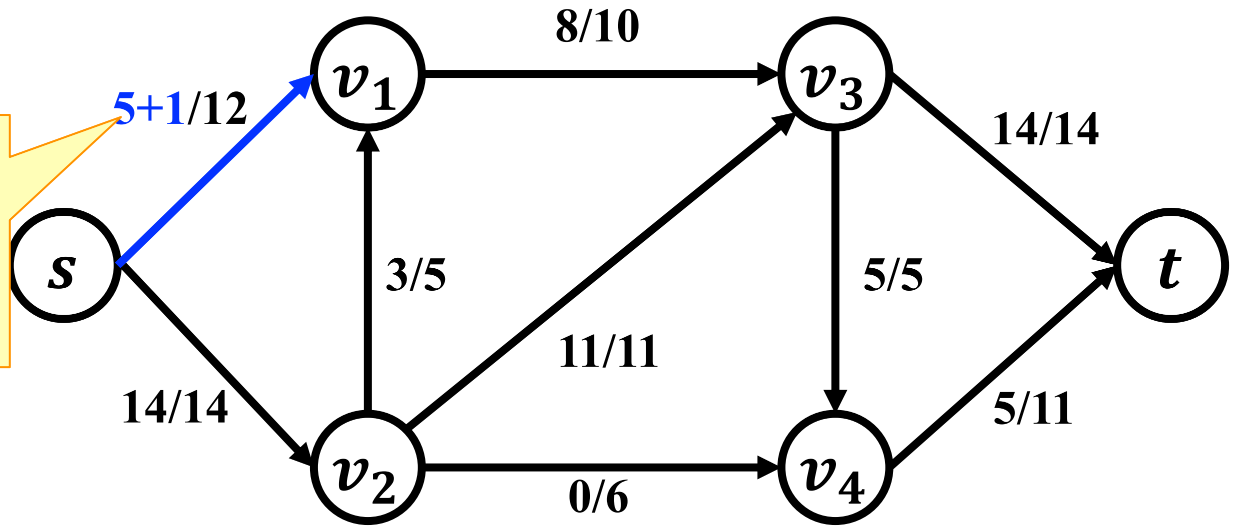


最大总流量：24

直观策略：比较观察

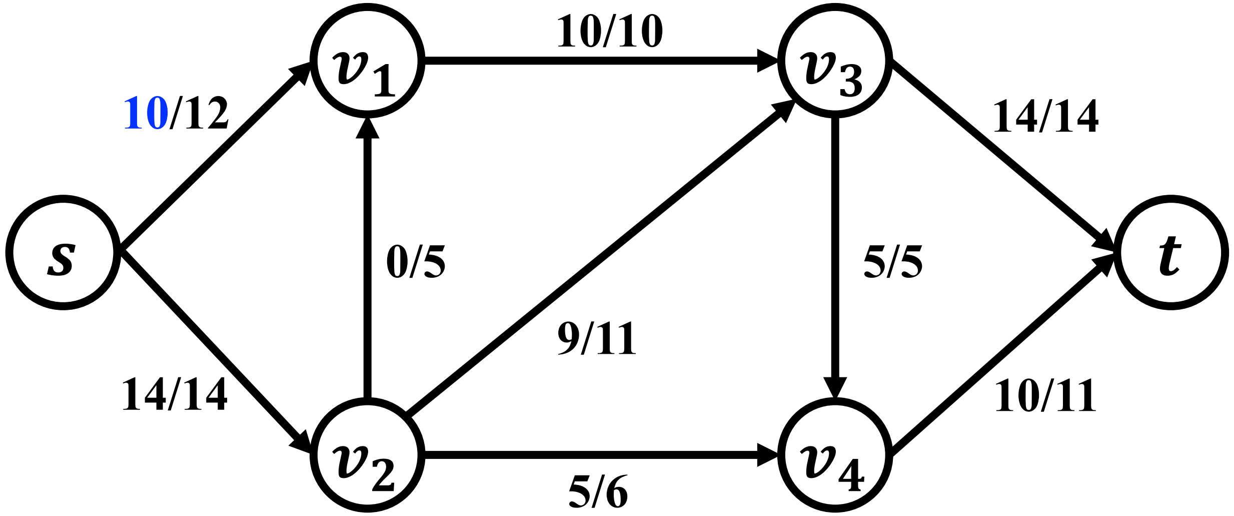
- 直观策略

为满足 v_1 流量守恒，扩充从其它边流出的流量



总流量：19

- 最优方案

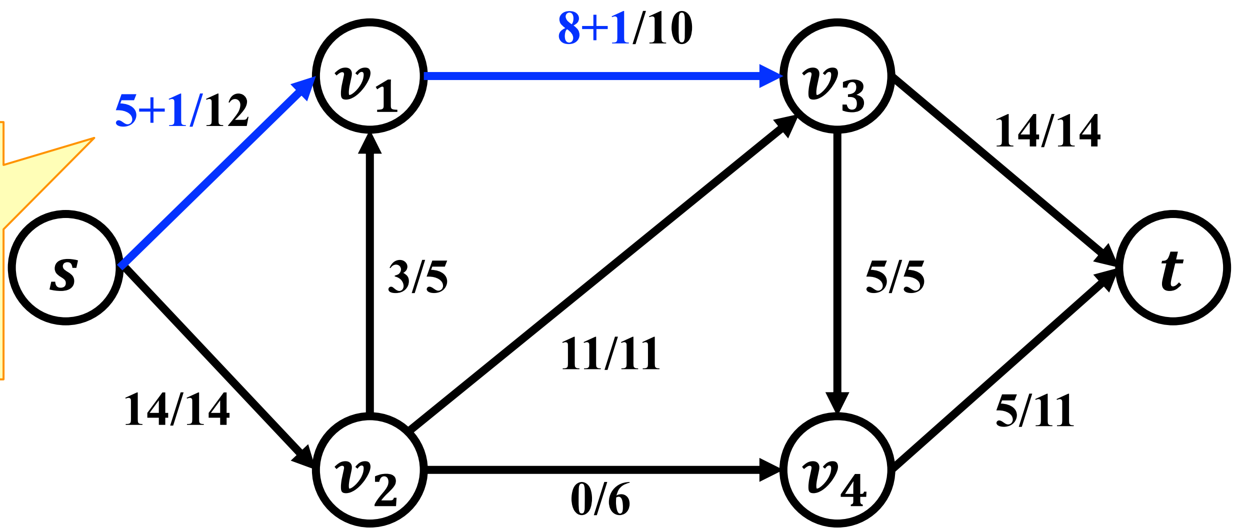


最大总流量：24

直观策略：比较观察

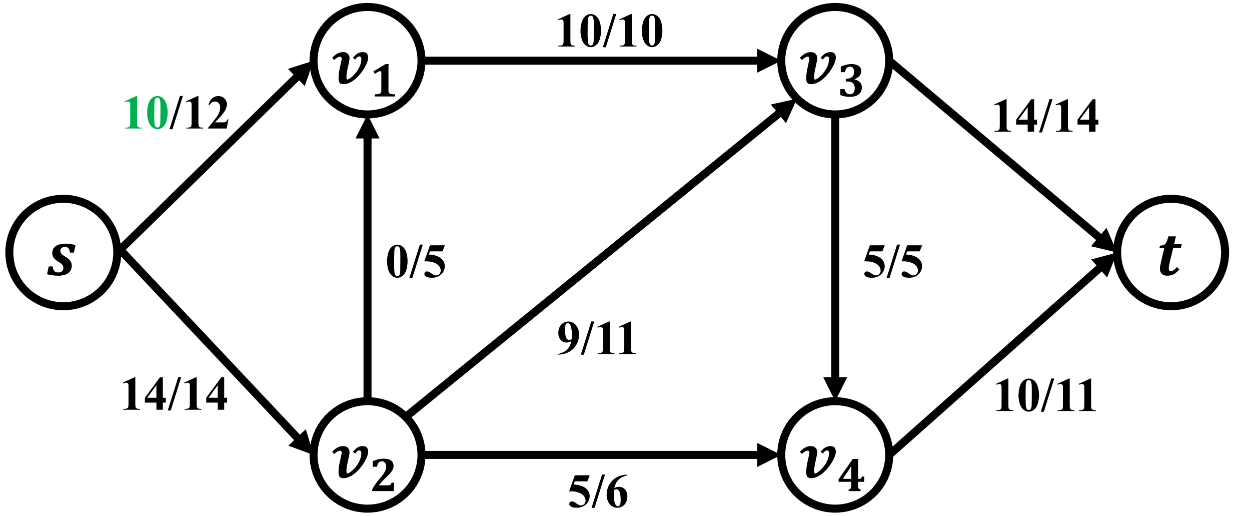
- 直观策略

重新分配经过v_1的流量



总流量：19

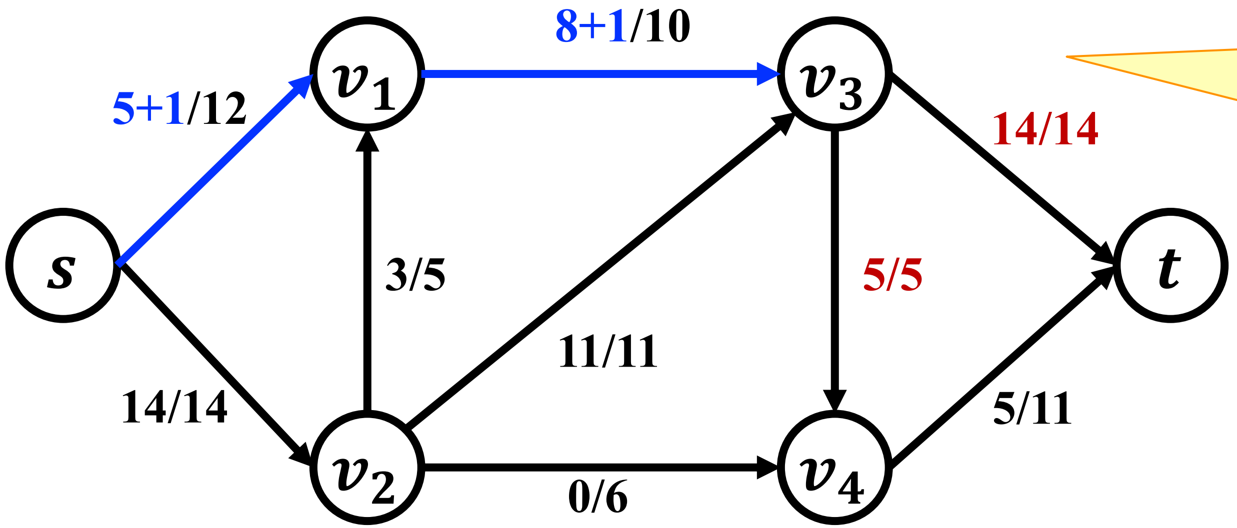
- 最优方案



最大总流量：24

直观策略：比较观察

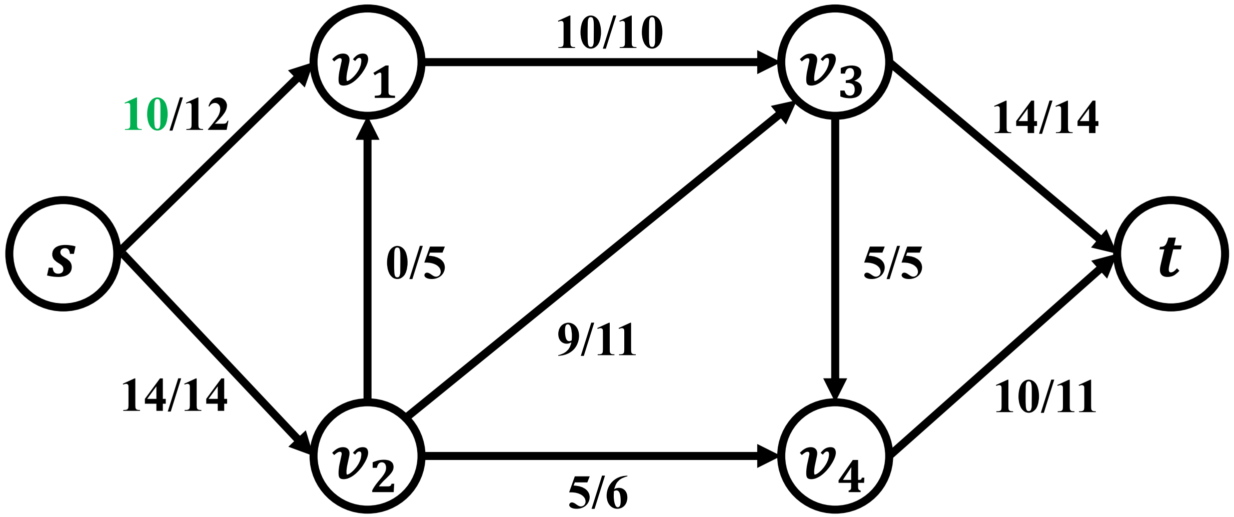
- 直观策略



无法扩充 v_3 从其它边流出的流量

总流量：19

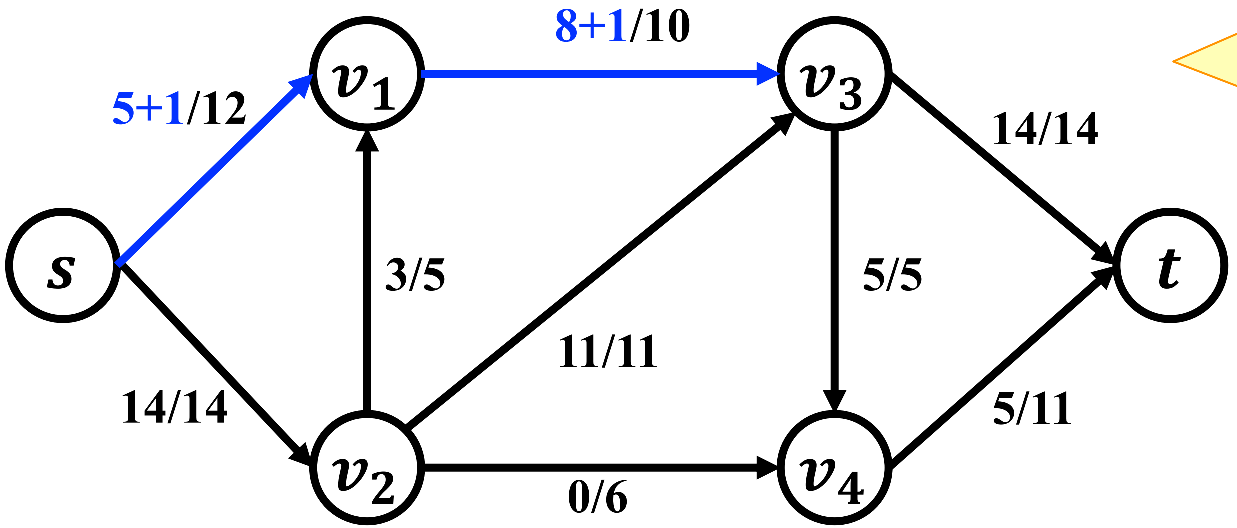
- 最优方案



最大总流量：24

直观策略：比较观察

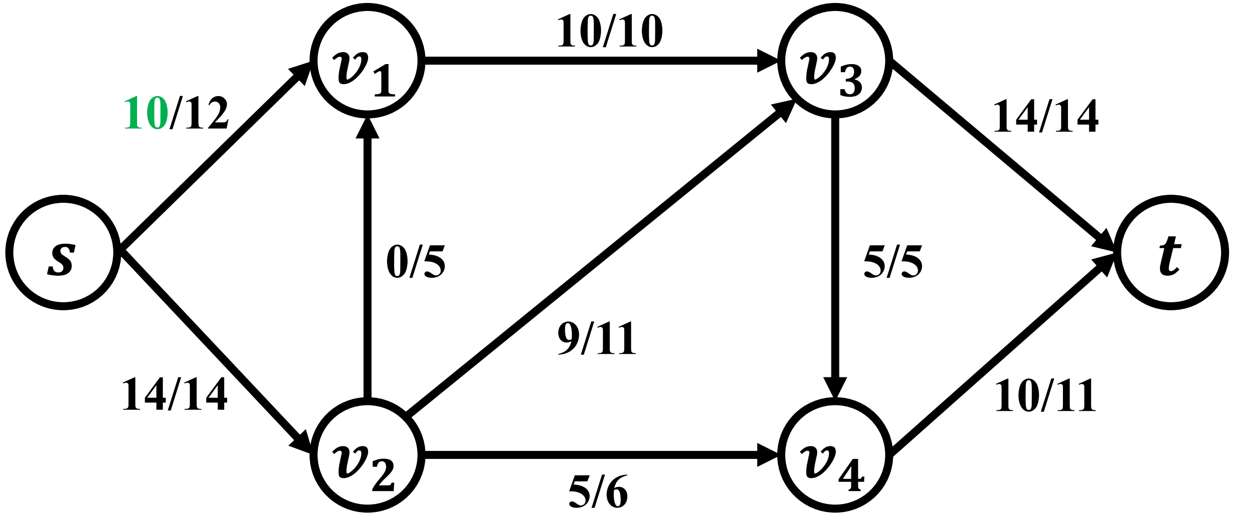
- 直观策略



为满足v_3流量守恒，缩减其它边的进入流量

总流量：19

- 最优方案

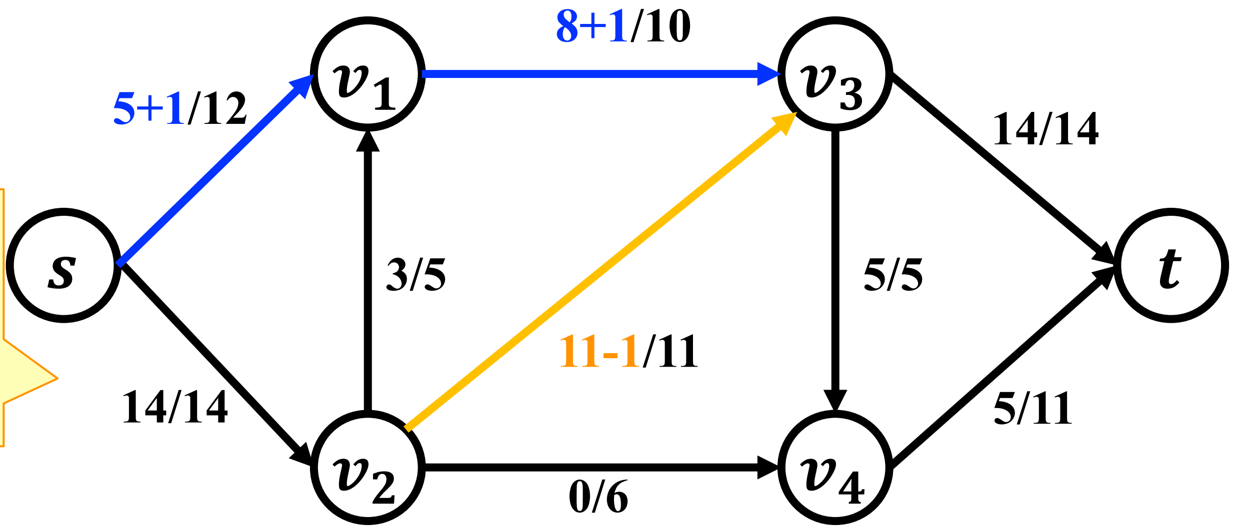


最大总流量：24

直观策略：比较观察

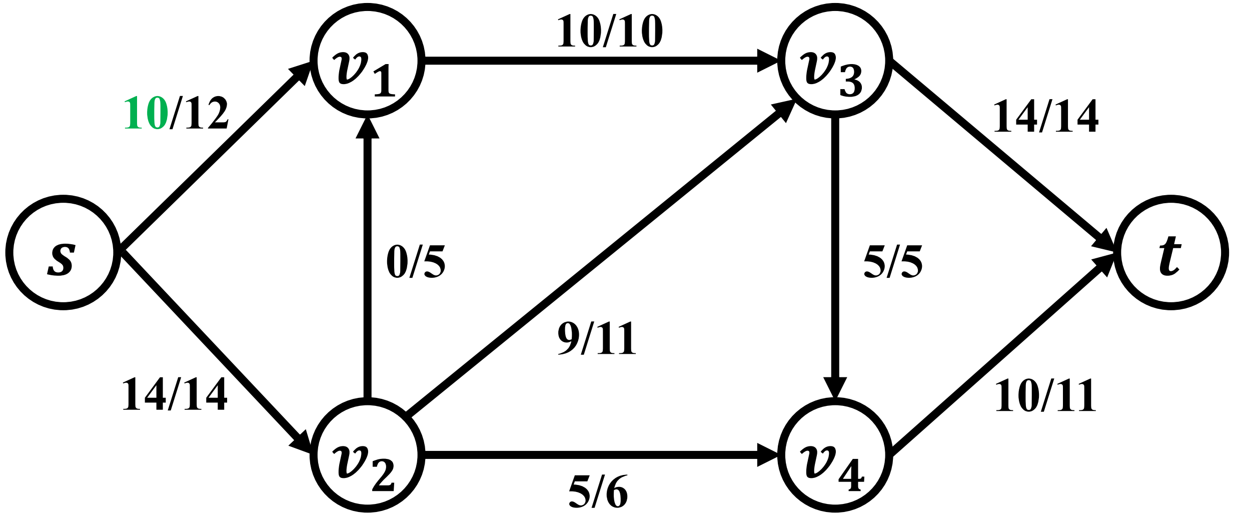
- 直观策略

重新分配经过 v_3的流量



总流量：19

- 最优方案

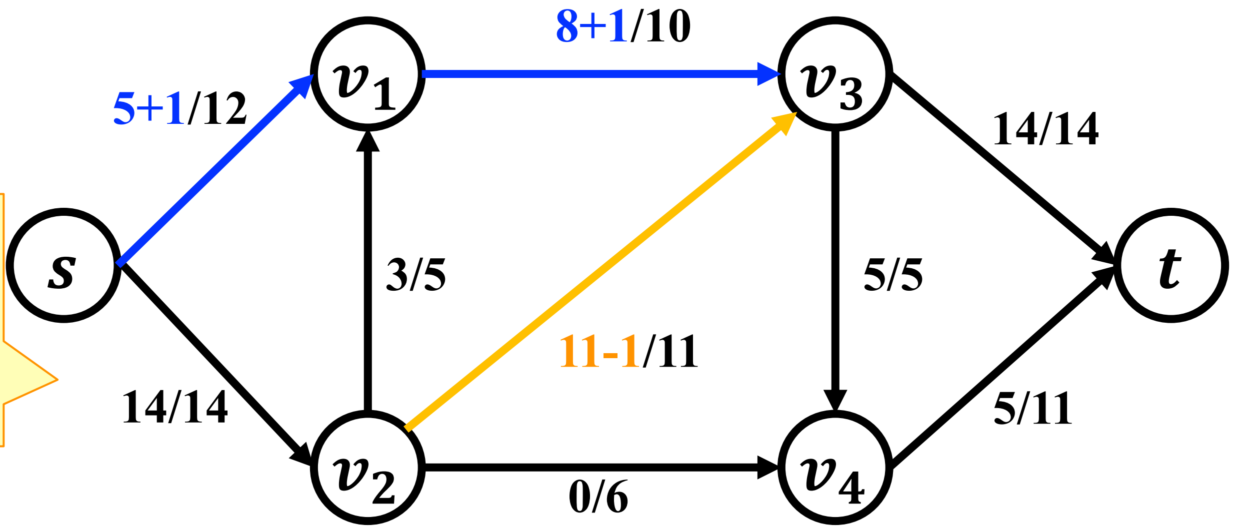


最大总流量：24

直观策略：比较观察

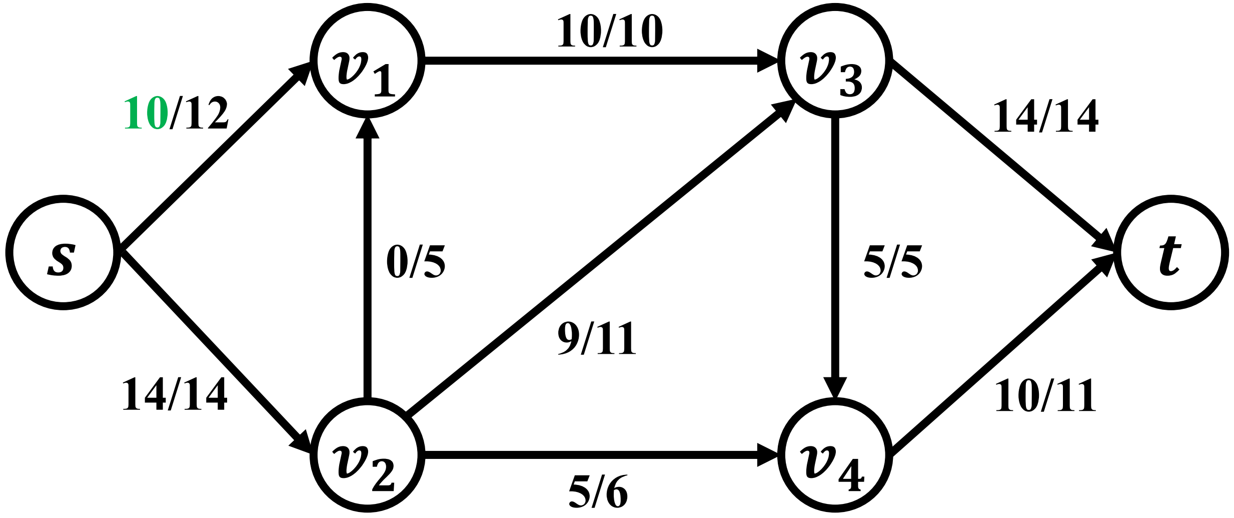
- 直观策略

为满足 v_2 流量守恒，扩充从其它边流出的流量



总流量：19

- 最优方案

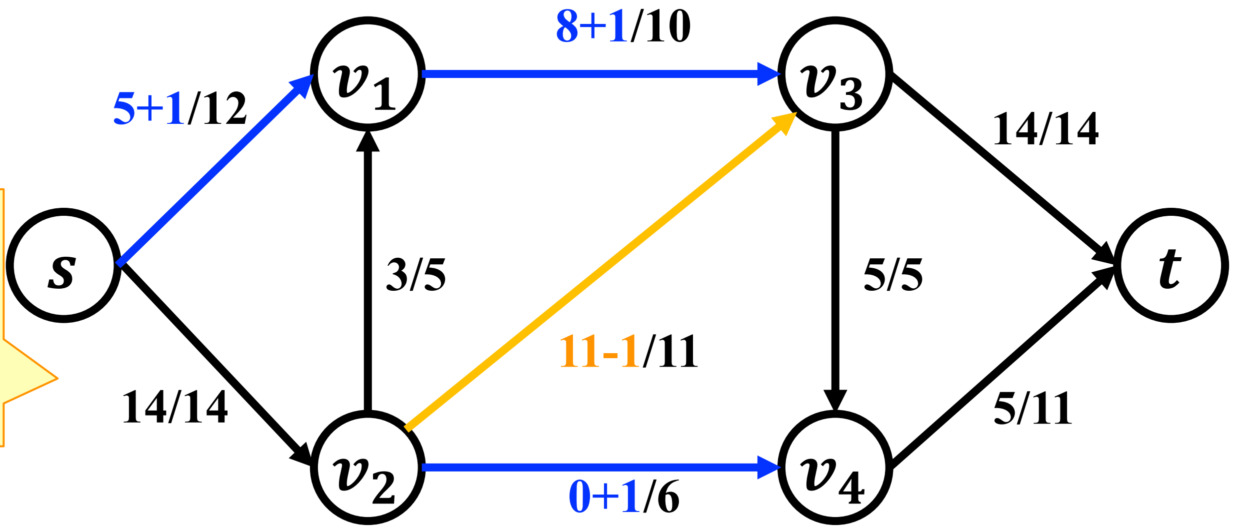


最大总流量：24

直观策略：比较观察

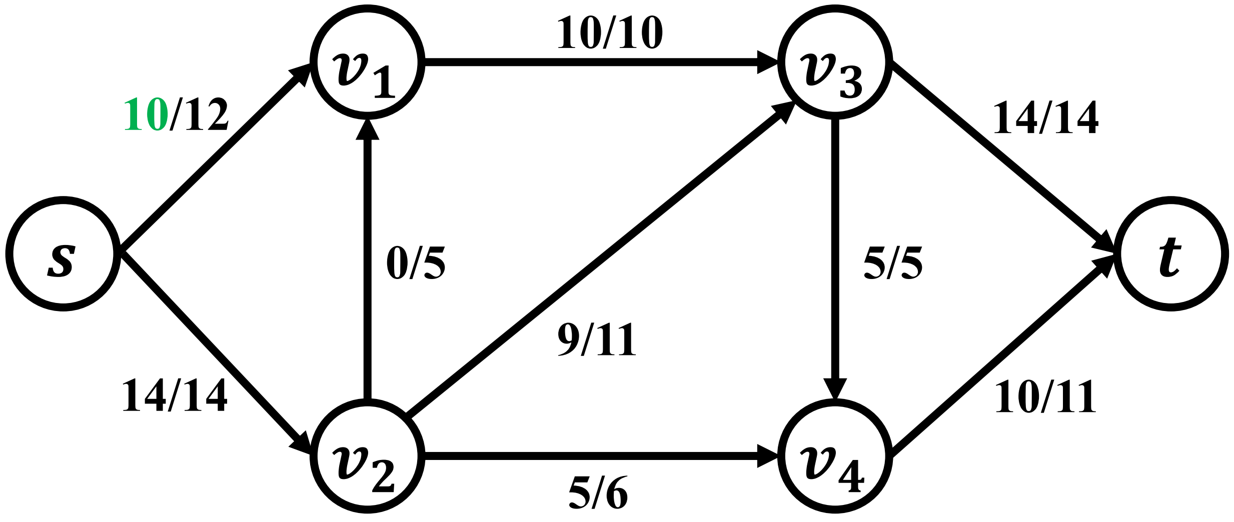
- 直观策略

重新分配经过v_2的流量



总流量：19

- 最优方案

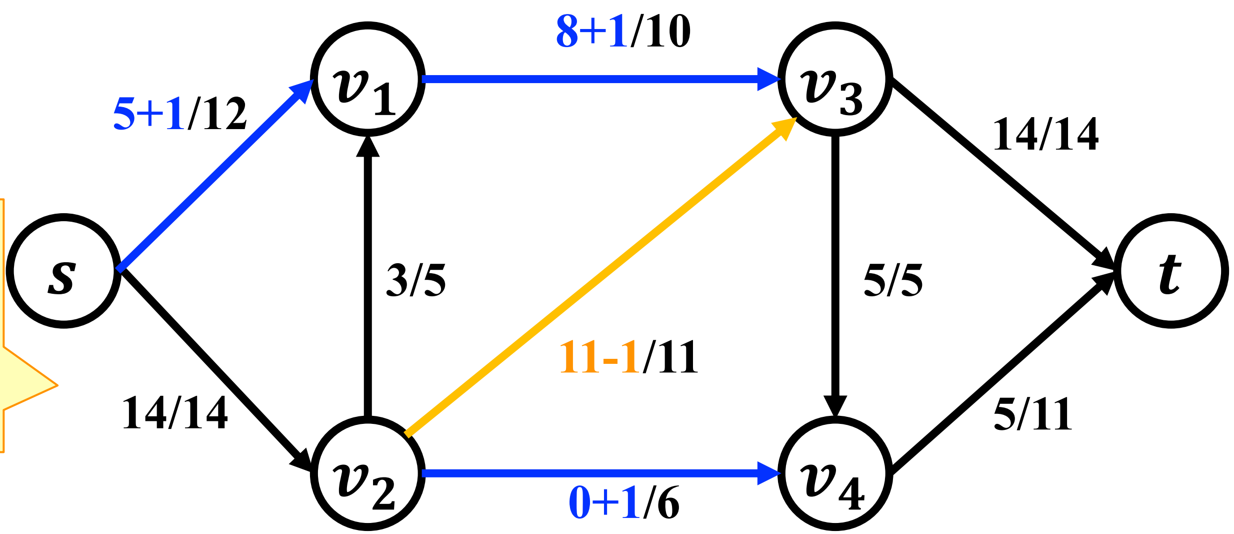


最大总流量：24

直观策略：比较观察

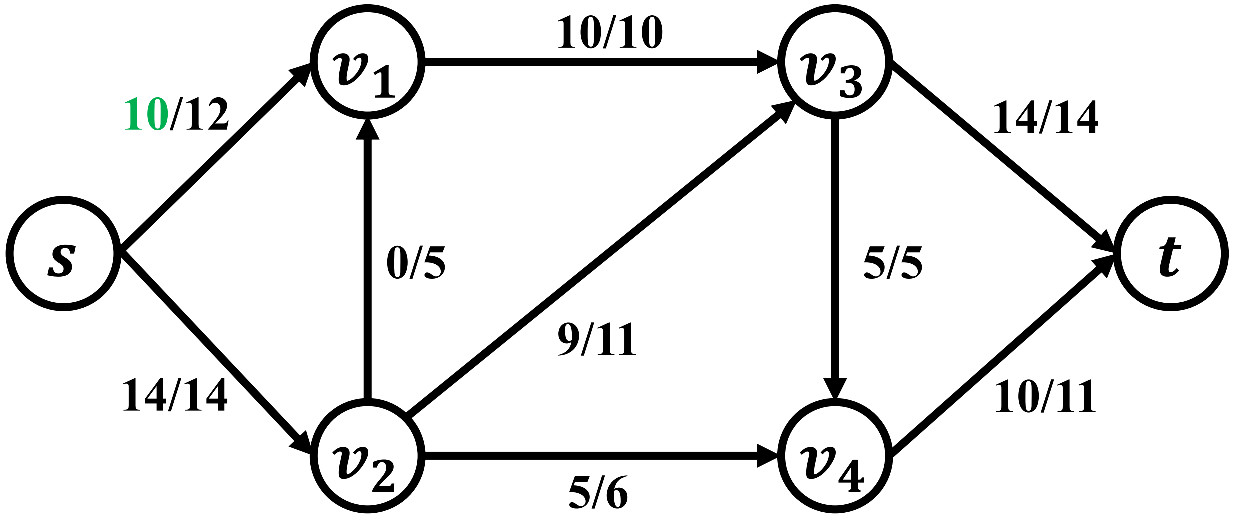
- 直观策略

为满足 v_4 流量守恒，扩充从其它边流出的流量



总流量：19

- 最优方案

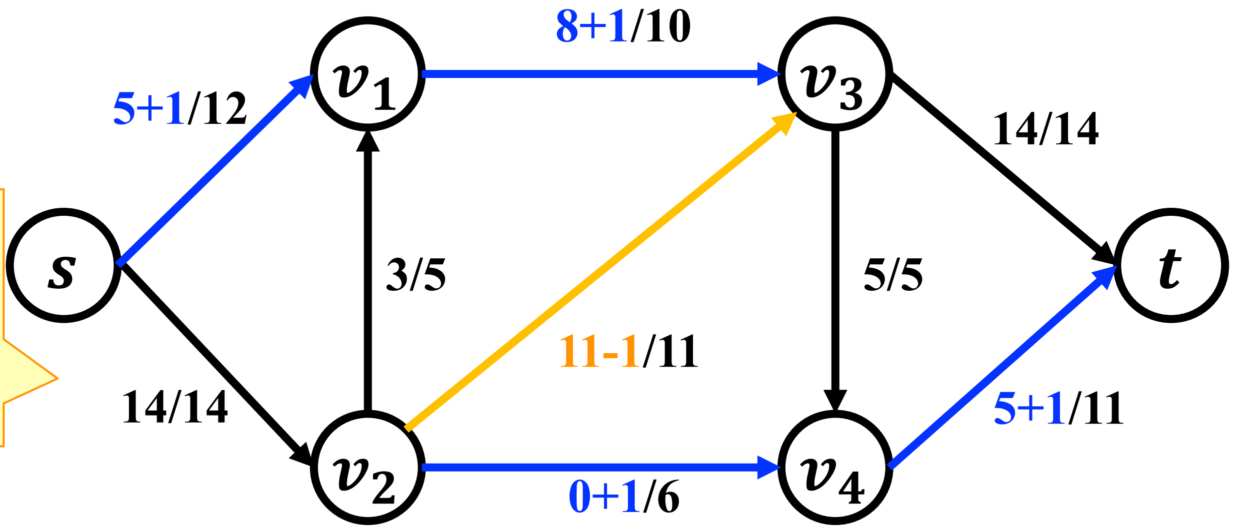


最大总流量：24

直观策略：比较观察

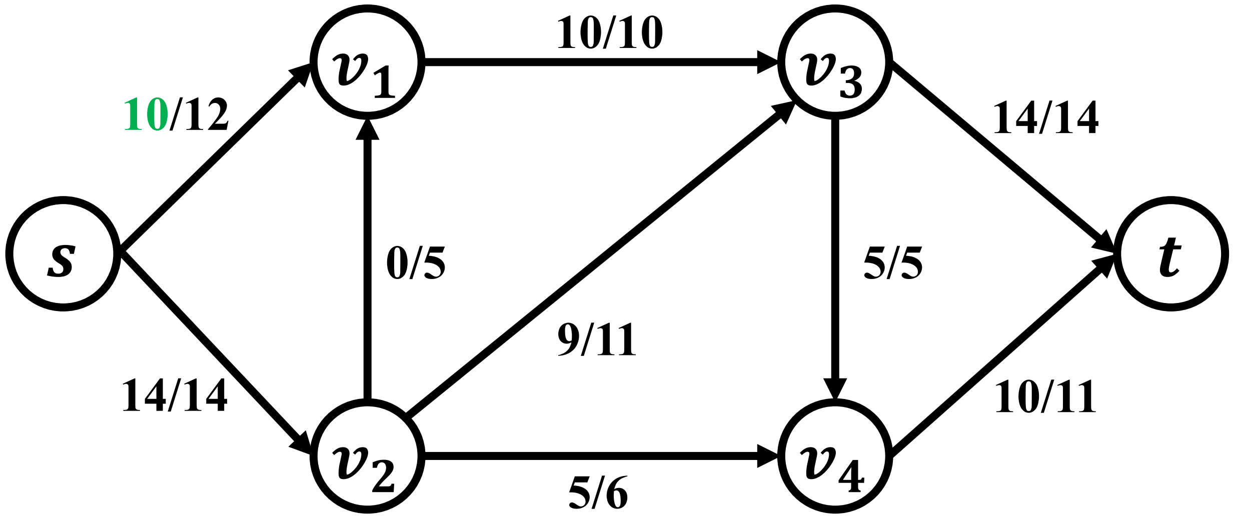
- 直观策略

重新分配经过 v_4的流量



总流量：19+1=20

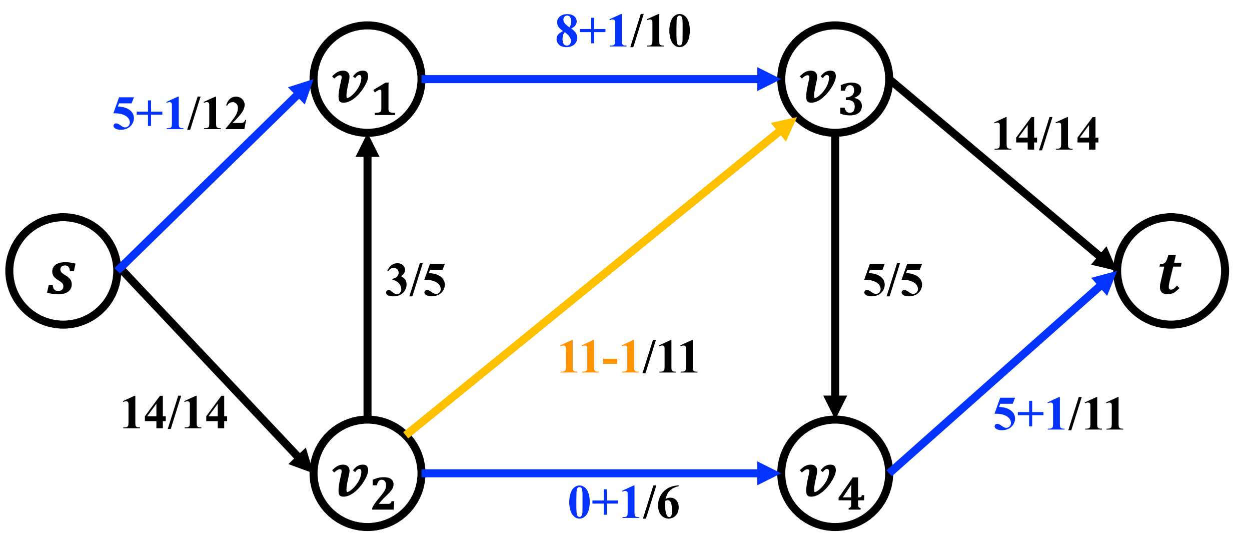
- 最优方案



最大总流量：24

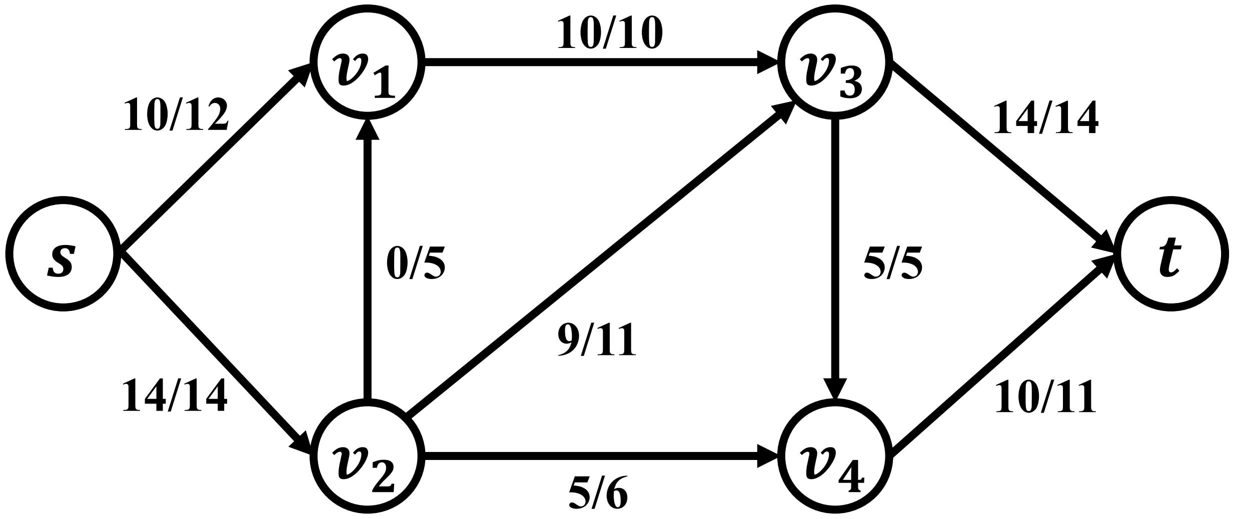
直观策略：比较观察

- 直观策略



总流量：20

- 最优方案

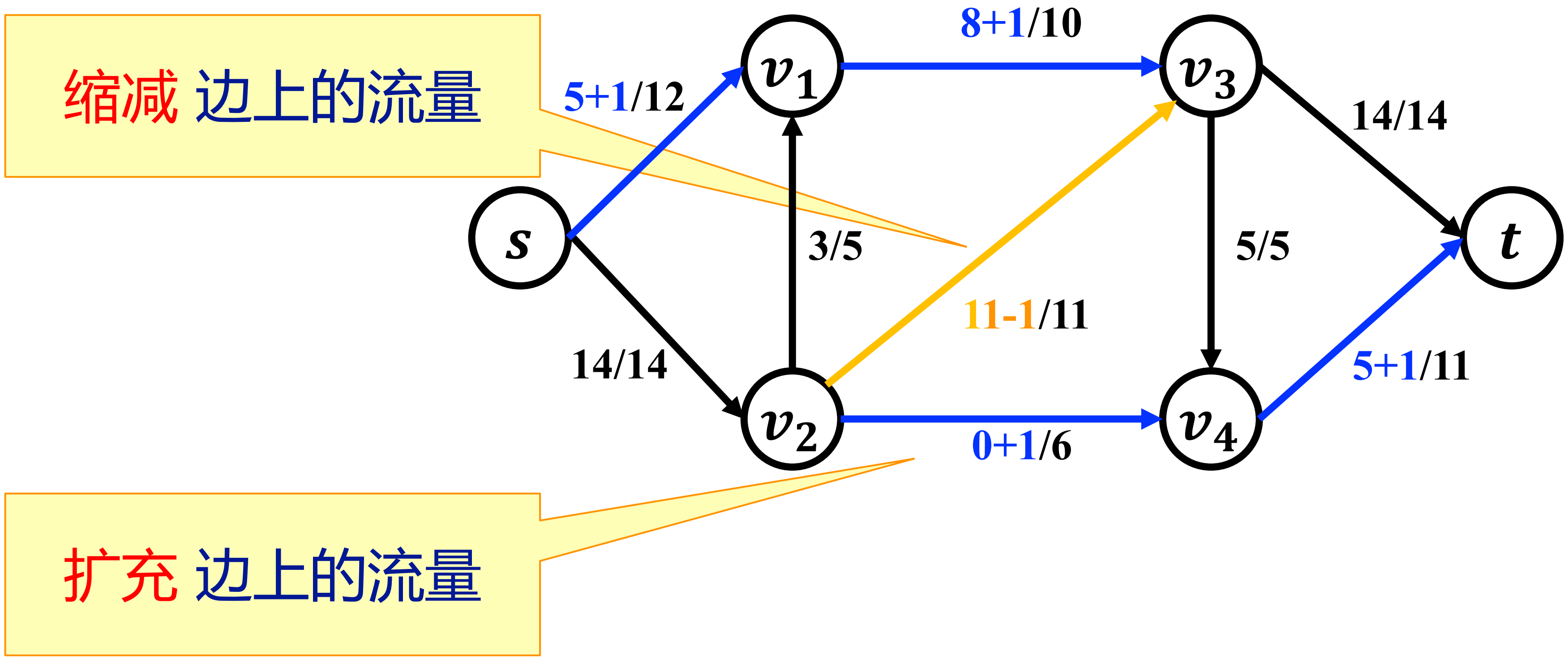


最大总流量：24

如果允许缩减边上的容量，可进一步增大总流量

直观策略：比较观察

- 调整流量的两种方式

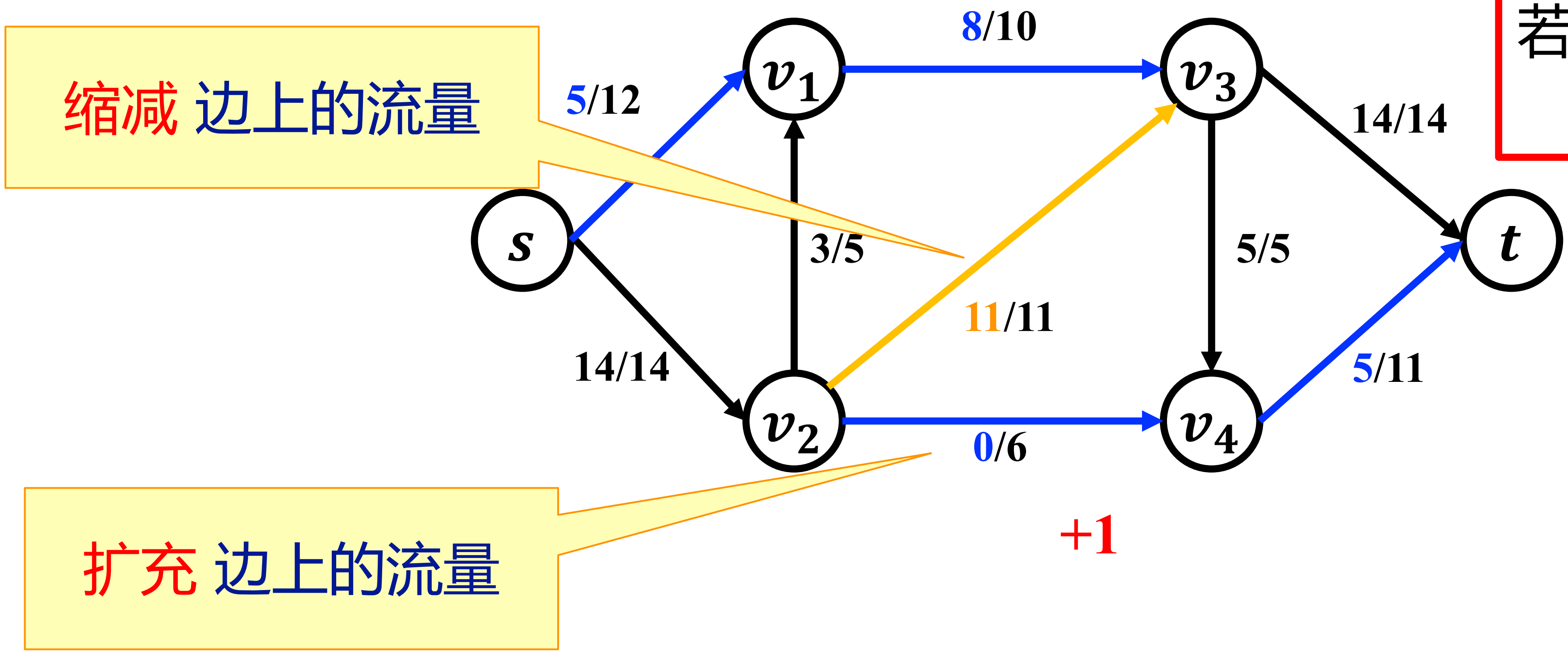


总流量：20

问题：如何改进直观策略？

直观策略：调整

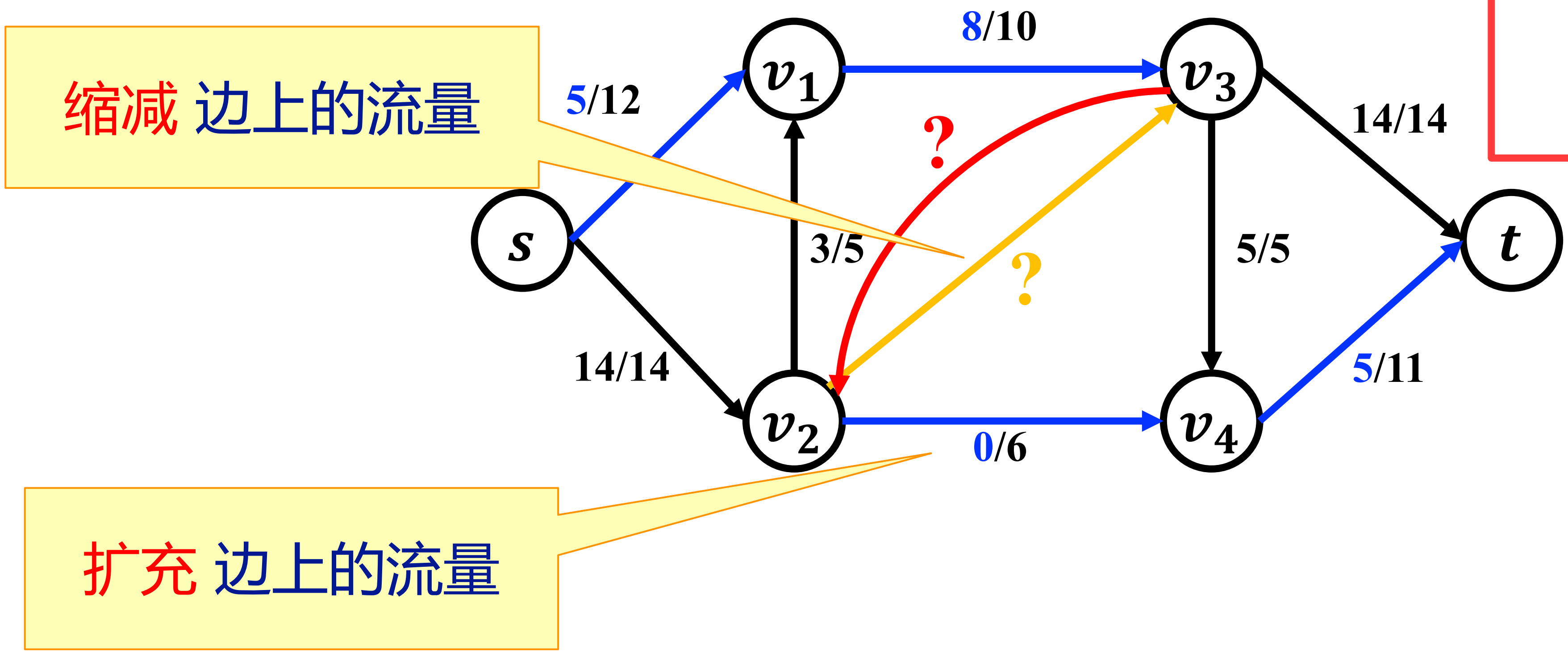
- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量



若寻找路径时允许逆向搜索
可以增大总流量

直观策略：调整

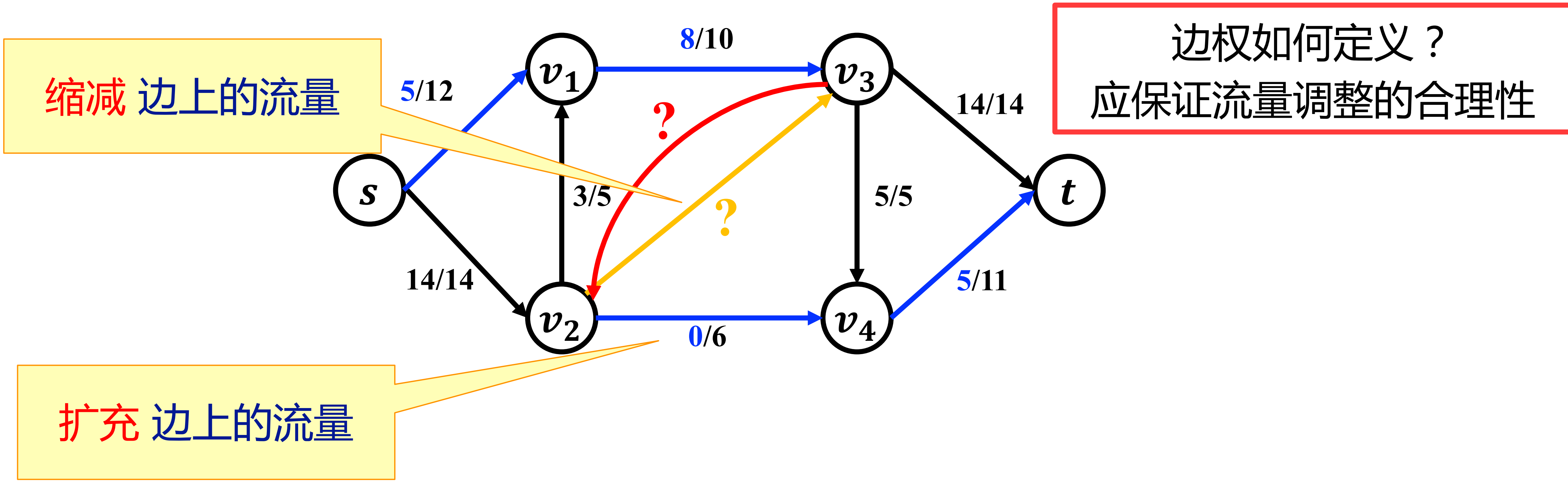
- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量



边权如何定义？

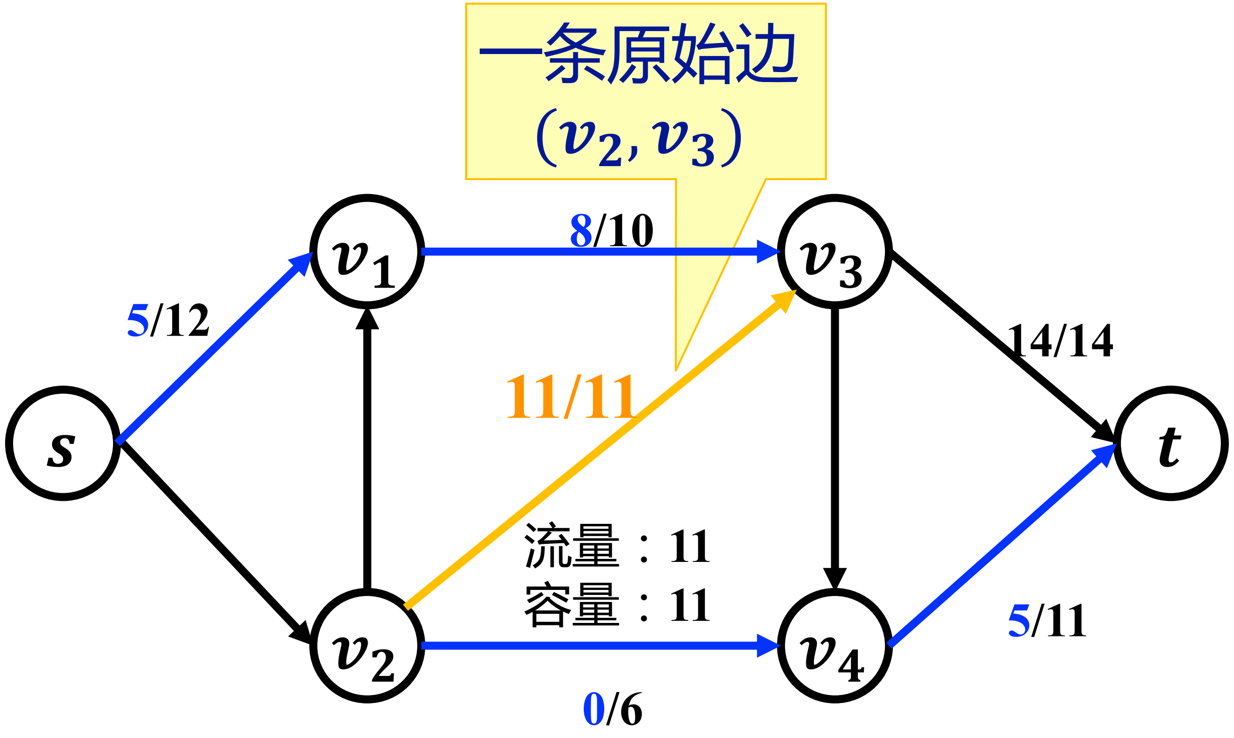
直观策略：调整

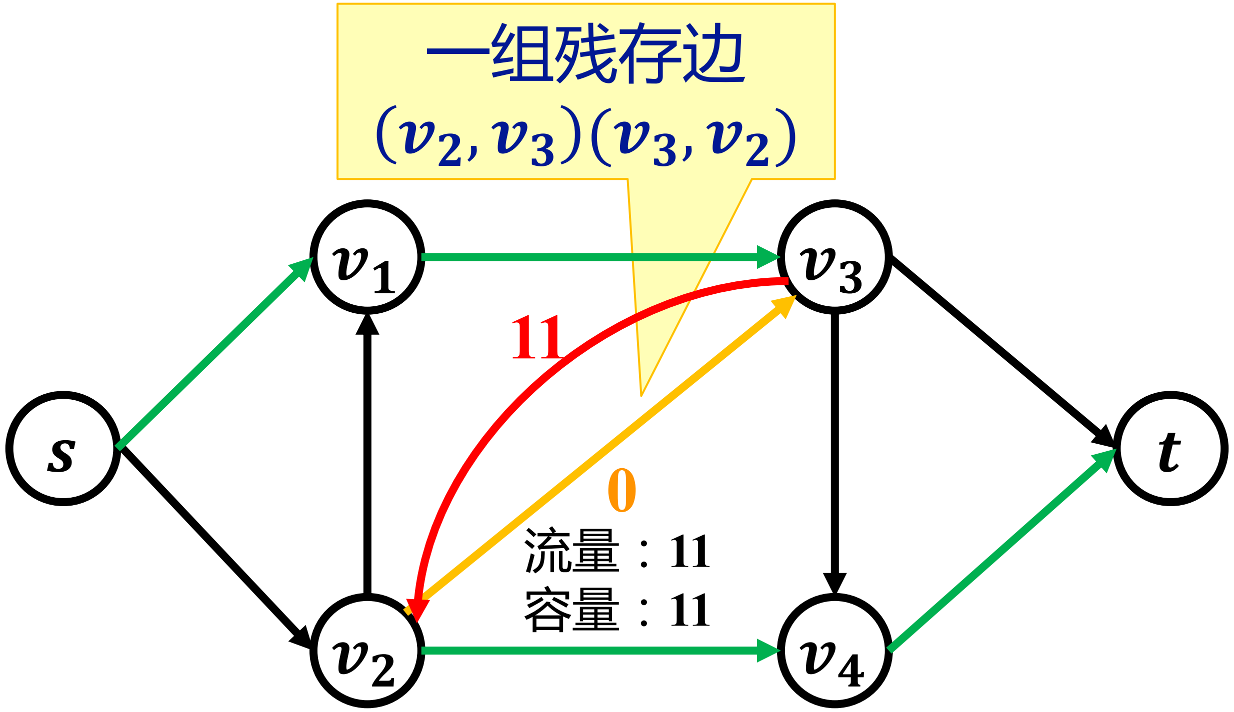
- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量



直观策略：调整

- 如何保证流量调整合理
  - 反向边权重：可缩减流量的上限，即原始边上的流量$f(e)$
  - 正向边权重：可扩充流量的上限，即原始边上的剩余容量$c(e) - f(e)$

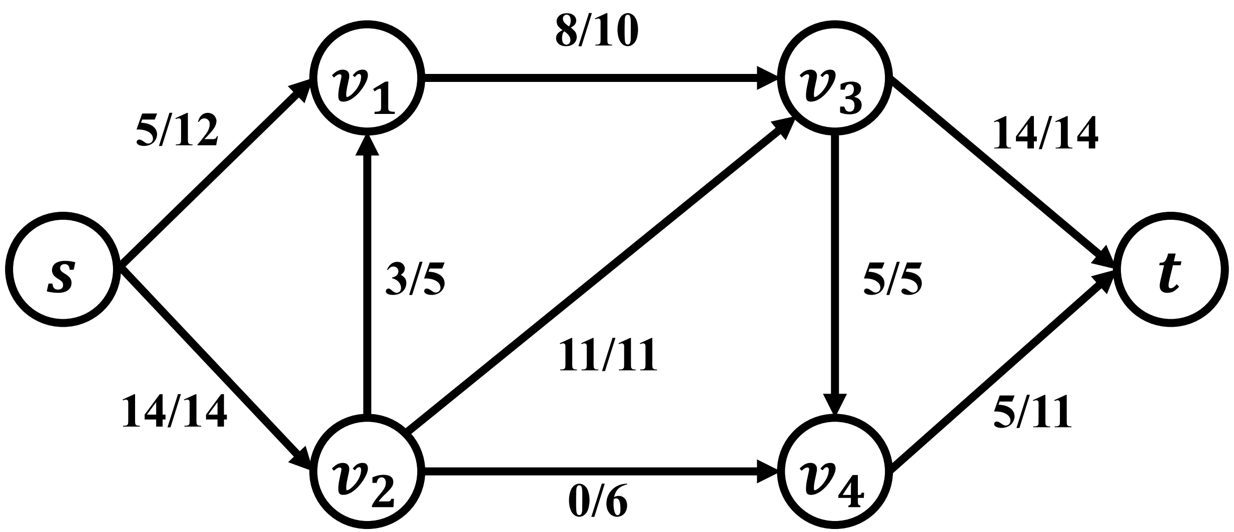




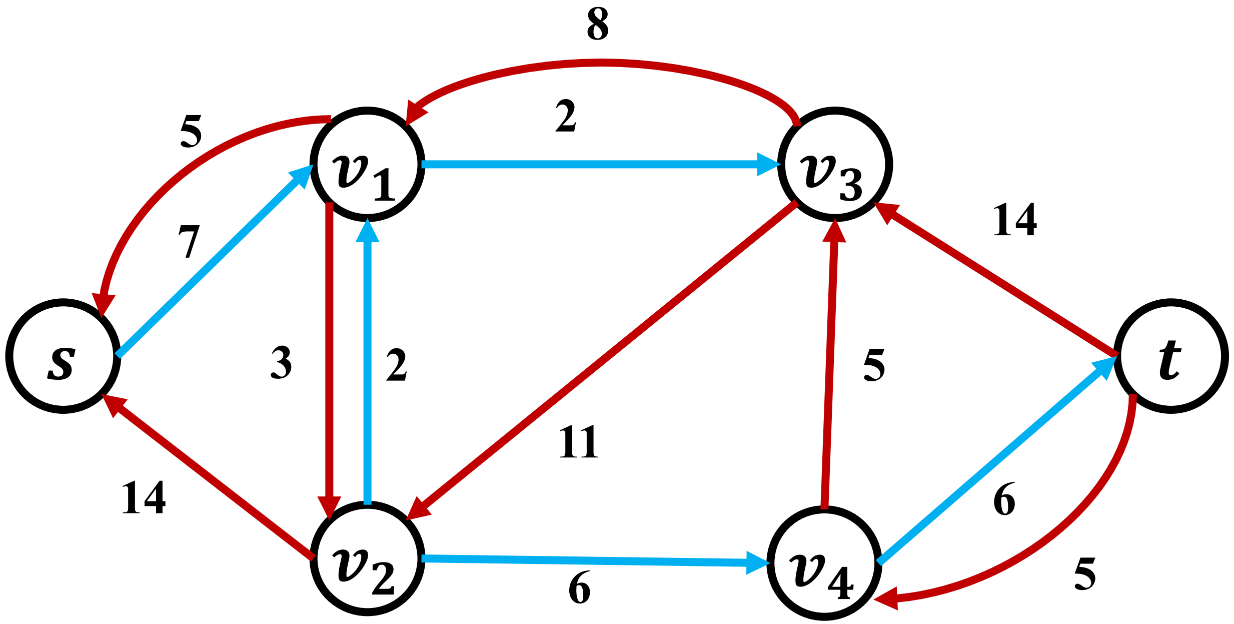
残存网络(Residual Network)

- 给定流网络 $G=<V,E,C>$和流量 f，可得残存网络 $G_f=<V,E_f>$，其中每条边的残存容量：

  - $$c_f(e)=\begin{cases} c(e)-f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$



流网络

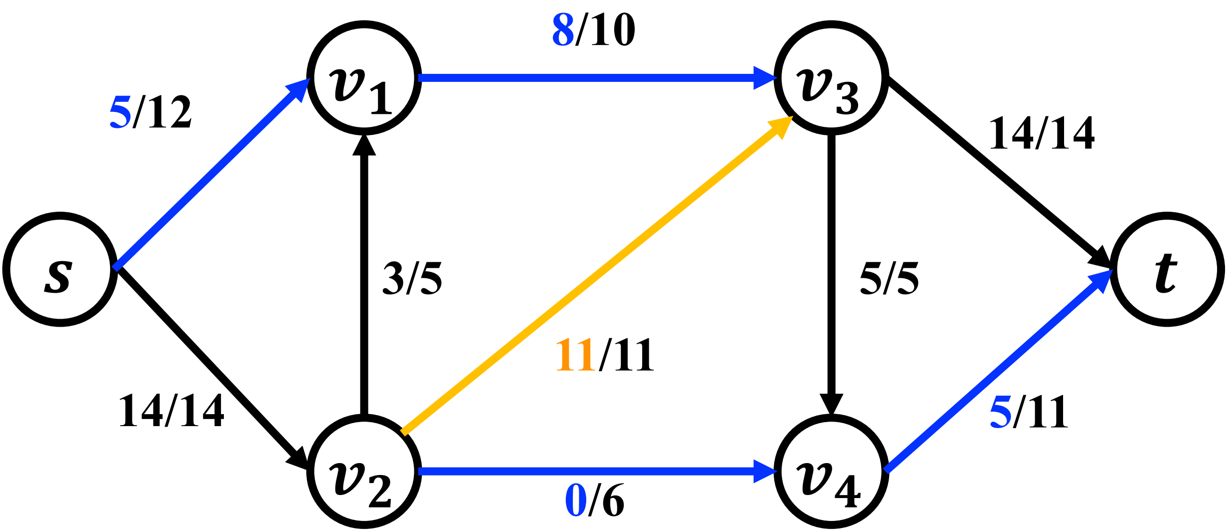


残存网络

残存网络(Residual Network)

- 增广路径(Augmenting Path)
  - 给定流网络$G=<V,E>$和流f，增广路径p是残存网络G_f中一条从源顶点s到汇点t的简单路径（路径上的各顶点均不互相重复）

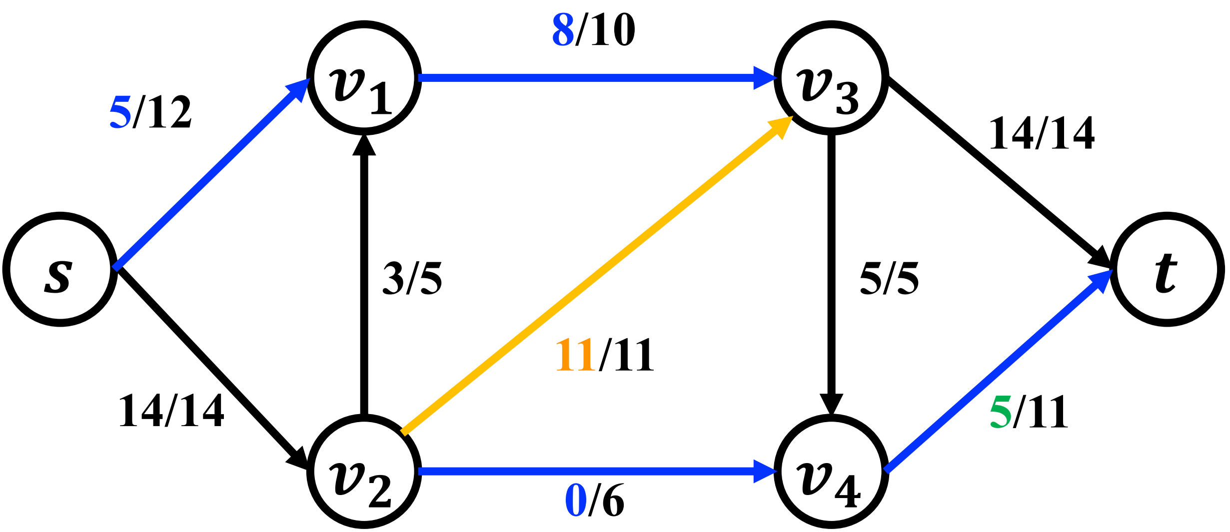
8

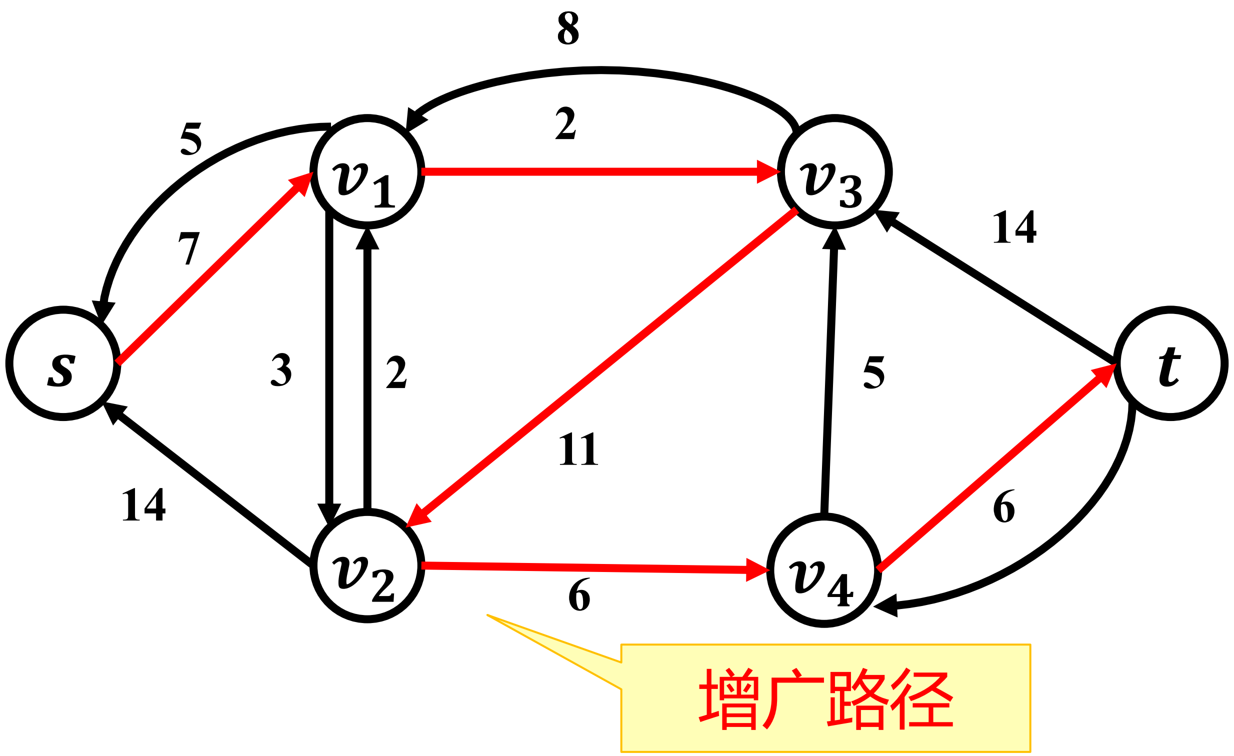


在残存网络上搜索增广路径，整合流量调整方式

残存网络(Residual Network)

- 增广路径(Augmenting Path)
  - 给定流网络$G=<V,E>$和流f，增广路径p是残存网络G_f中一条从源顶点s到汇点t的简单路径（路径上的各顶点均不互相重复）



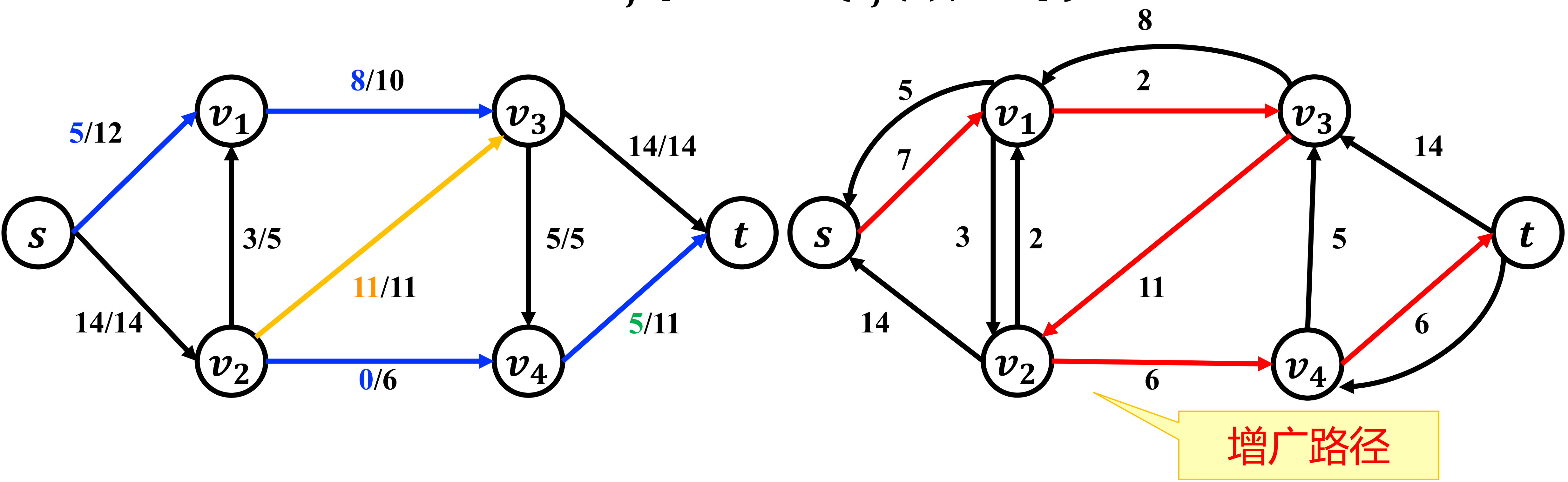


在残存网络上搜索增广路径，整合流量调整方式

残存网络(Residual Network)

- 增广路径(Augmenting Path)
  - 一条增广路径p上各边残存容量的最小值

$$c_f(p) = min\{c_f(e) \mid e \in p\}$$

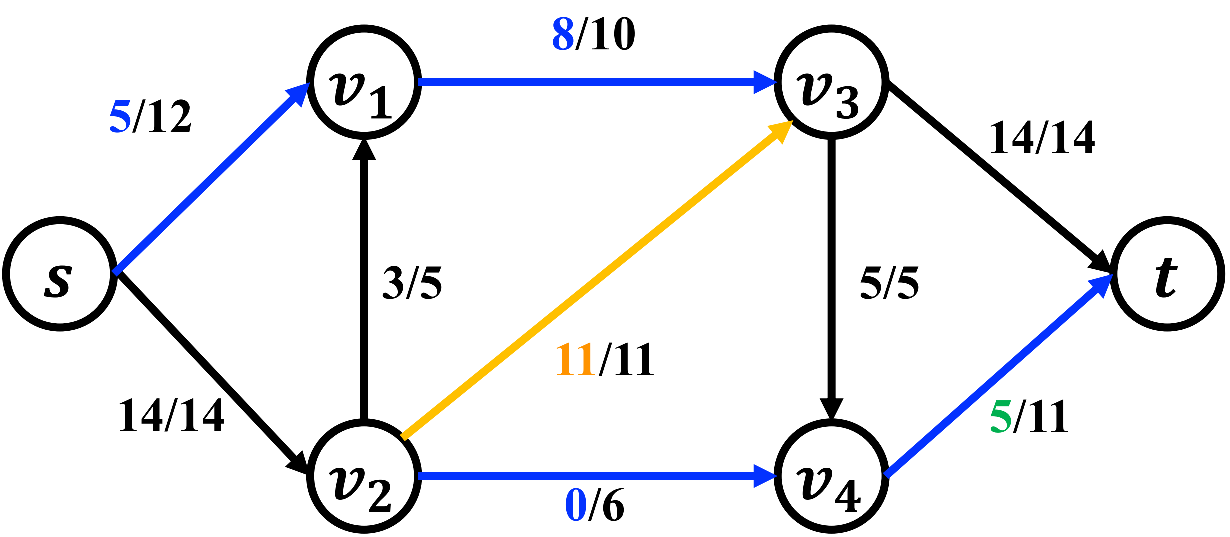


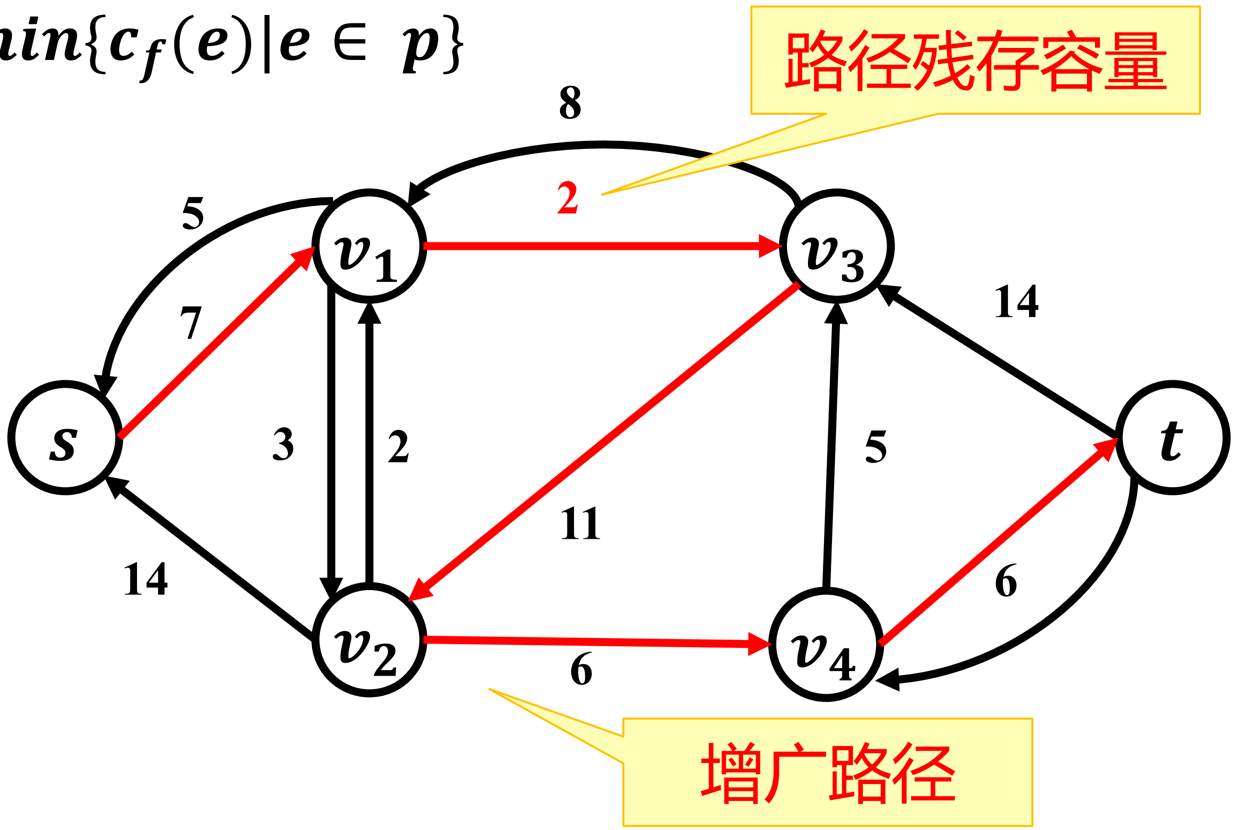
在残存网络上搜索增广路径，整合流量调整方式

残存网络(Residual Network)

- 增广路径(Augmenting Path)
  - 一条增广路径p上各边残存容量的最小值

$$c_f(p) = min\{c_f(e) | e \in p\}$$





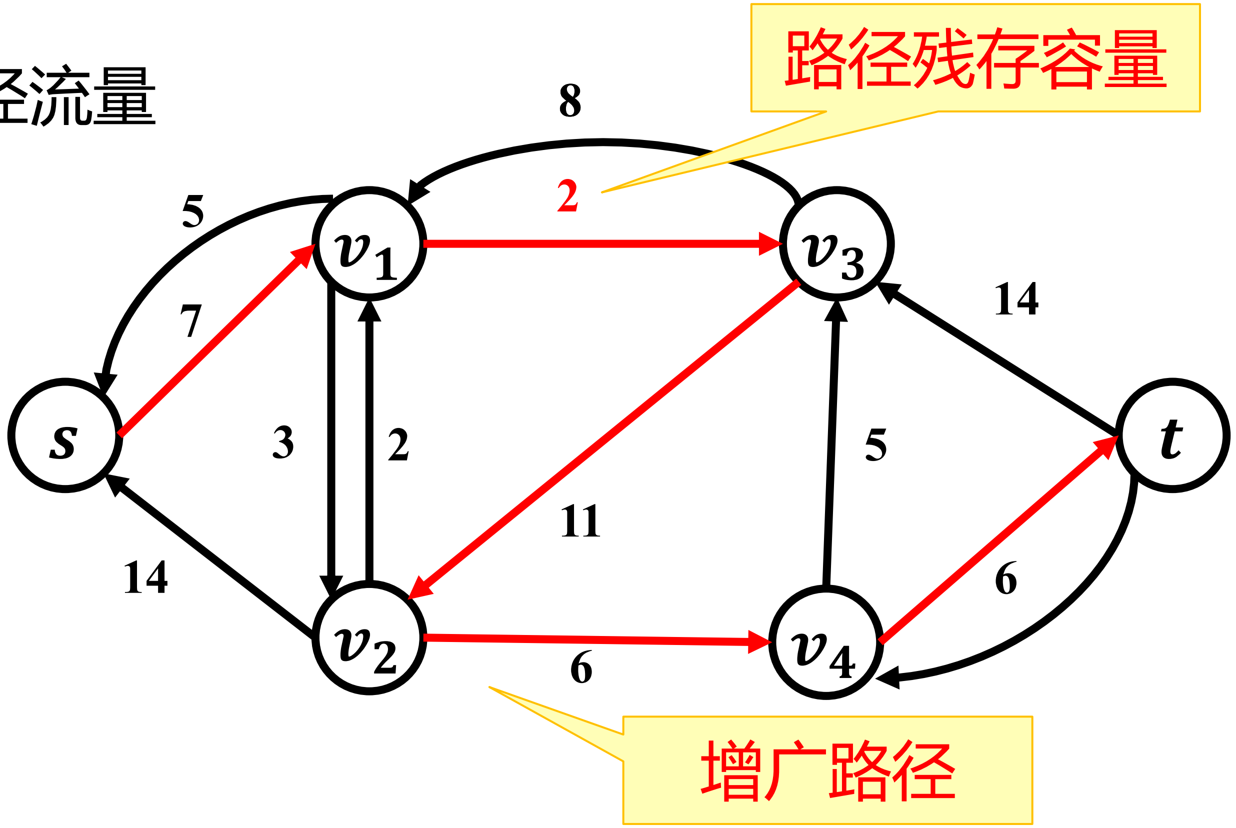
在残存网络上搜索增广路径，整合流量调整方式

从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 寻找一条 s 到 t 的路径 P，此路径上的每条边 e 均满足 $f(e) < c(e)$
  - 按路径 P 上最小剩余容量增加路径流量
  - 迭代寻找路径 P 直至无法增加路径流量

从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 构造残存网络G_f，寻找 s 到 t 的增广路径 P
  - 按路径P 的残存容量增加流量
  - 迭代寻找路径 P 直至无法增加路径流量



从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 构造残存网络G_f，寻找 s 到 t 的增广路径 P
  - 按路径P 的残存容量增加流量
  - 迭代寻找路径 P 直至无法增加路径流量
- 如何按路径 P 的残存容量增加流量 ？
  - 路径 P 包括两种边：正向边、反向边

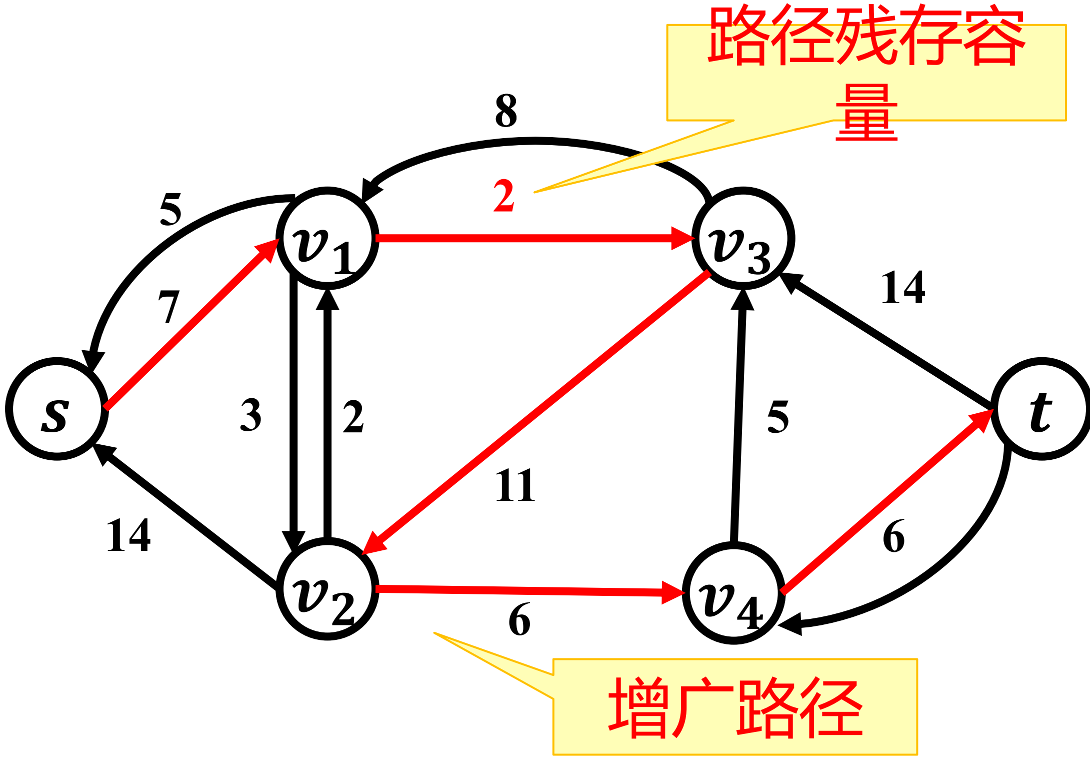
$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 构造残存网络G_f，寻找 s 到 t 的增广路径 P
  - 按路径P 的残存容量增加流量
  - 迭代寻找路径 P 直至无法增加路径流量

$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

- 如何按路径 P 的残存容量增加流量？
  - 路径 P 包括两种边：正向边、反向边
    - 若 $e \in P$ 是正向边，则 $e \in E$，网络流 G 中



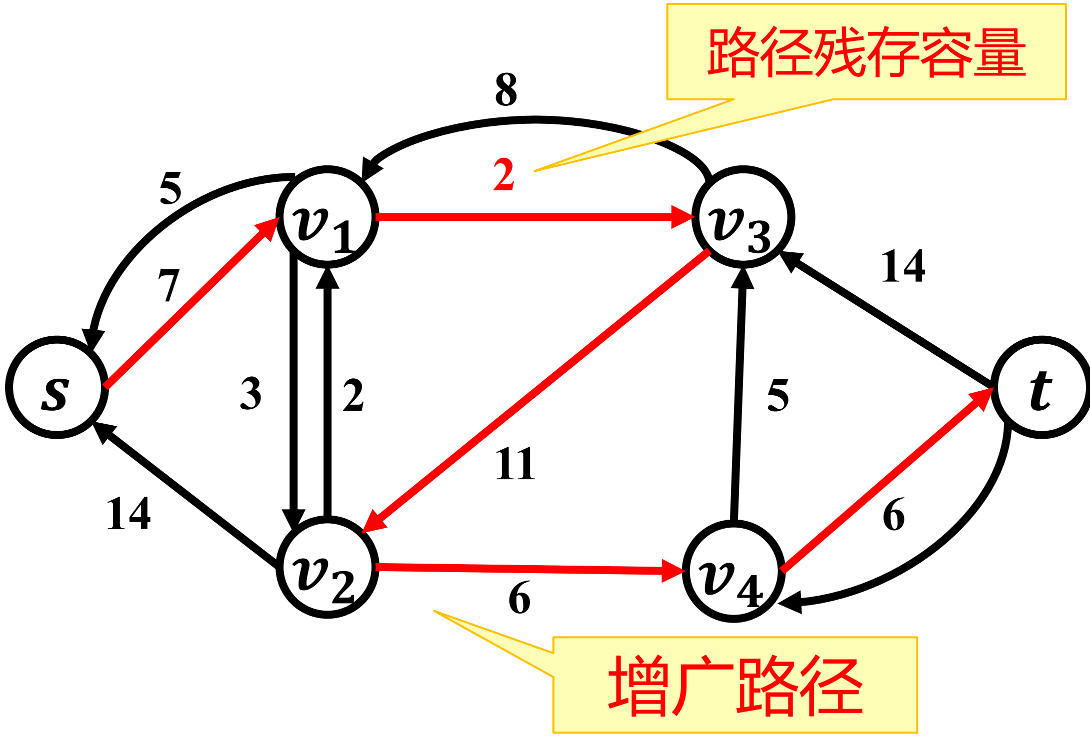
从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 构造残存网络G_f，寻找 s 到 t 的增广路径 P
  - 按路径P 的残存容量增加流量
  - 迭代寻找路径 P 直至无法增加路径流量

$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

- 如何按路径 P 的残存容量增加流量 ？
  - 路径 P 包括两种边：正向边、反向边
    - 若 $e \in P$ 是正向边，则 $e \in E$，网络流 G 中

$$f(e) \leftarrow f(e) + c_f(P)$$



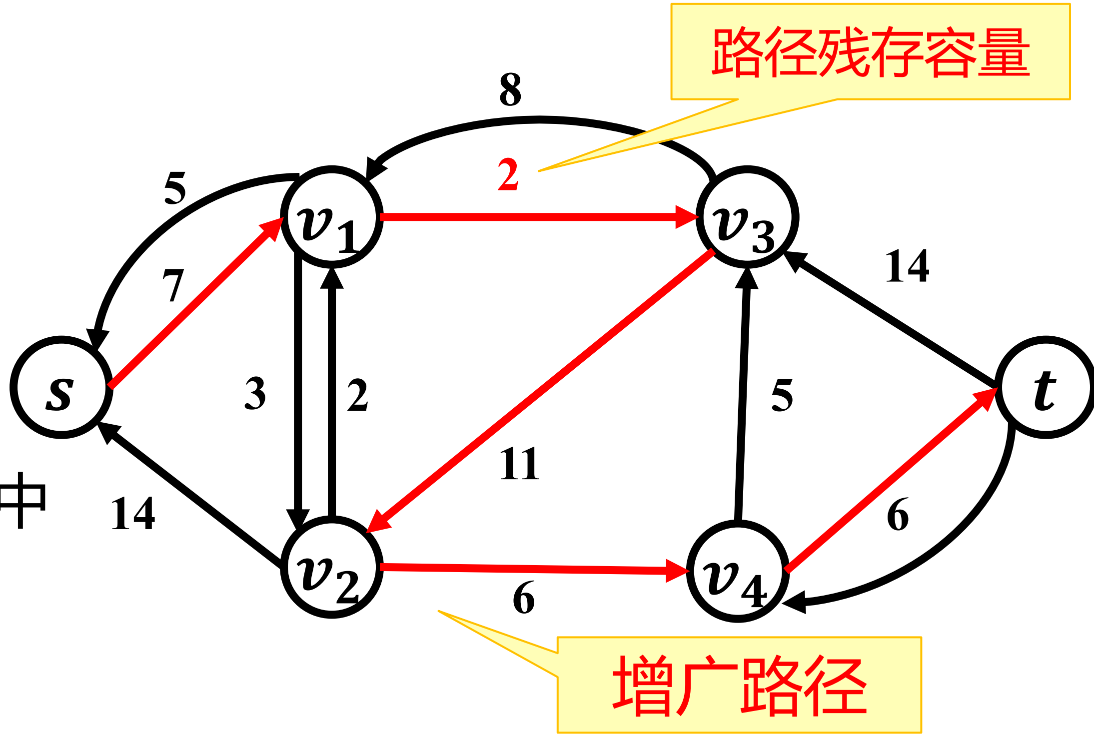
从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e)=0$
  - 构造残存网络G_f，寻找 s 到 t 的增广路径 P
  - 按路径P 的残存容量增加流量
  - 迭代寻找路径 P 直至无法增加路径流量

$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

- 如何按路径 P 的残存容量增加流量？
  - 路径 P 包括两种边：正向边、反向边
    - 若 $e \in P$ 是正向边，则 $e \in E$，网络流 G 中

      $$f(e) \leftarrow f(e) + c_f(P)$$
    - 若 $e \in P$ 是反向边，则 $e^R \in E$，网络流 G 中

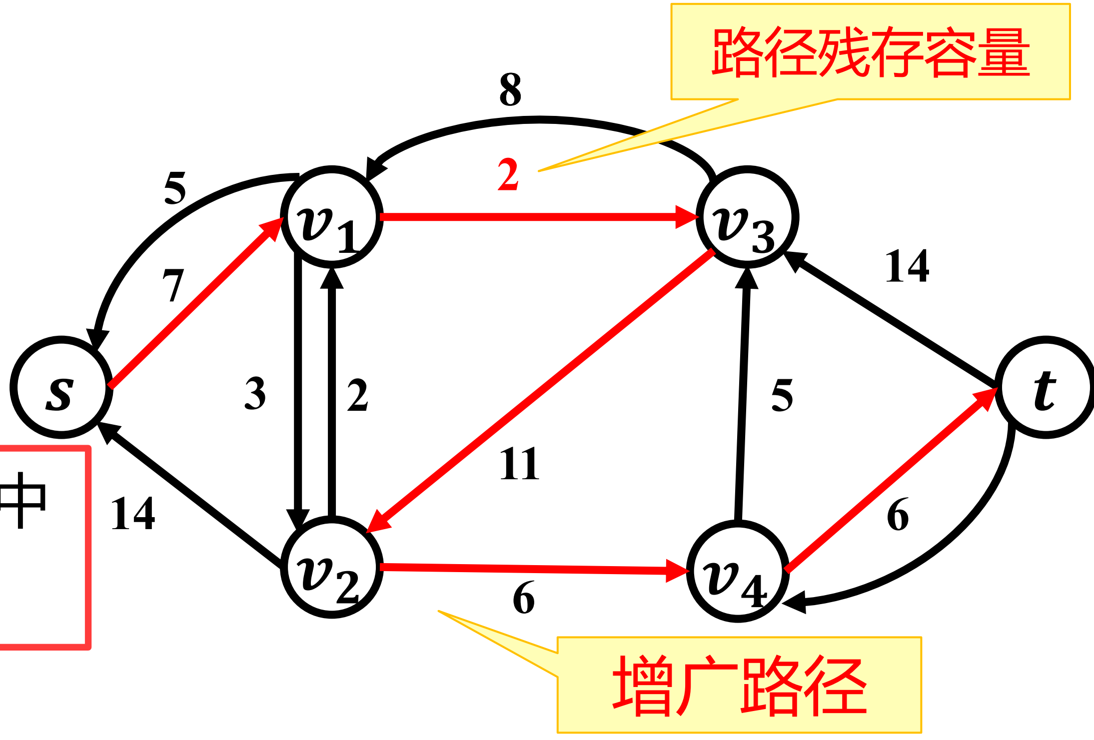


从直观策略到Ford-Fulkerson算法

- 算法思想
  - 对所有边 $e \in E$，初始化流量为零 $f(e) = 0$
  - 构造残存网络G_f，寻找 s 到 t 的增广路径 P
  - 按路径P 的残存容量增加流量
  - 迭代寻找路径 P 直至无法增加路径流量

$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

- 如何按路径 P 的残存容量增加流量 ？
  - 路径 P 包括两种边：正向边、反向边
    - 若 $e \in P$ 是正向边，则 $e \in E$，网络流 G 中
      $f(e) \leftarrow f(e) + c_f(P)$
    - 若 $e \in P$ 是反向边，则 $e^R \in E$，网络流 G 中
      $f(e^R) \leftarrow f(e^R) - c_f(P)$



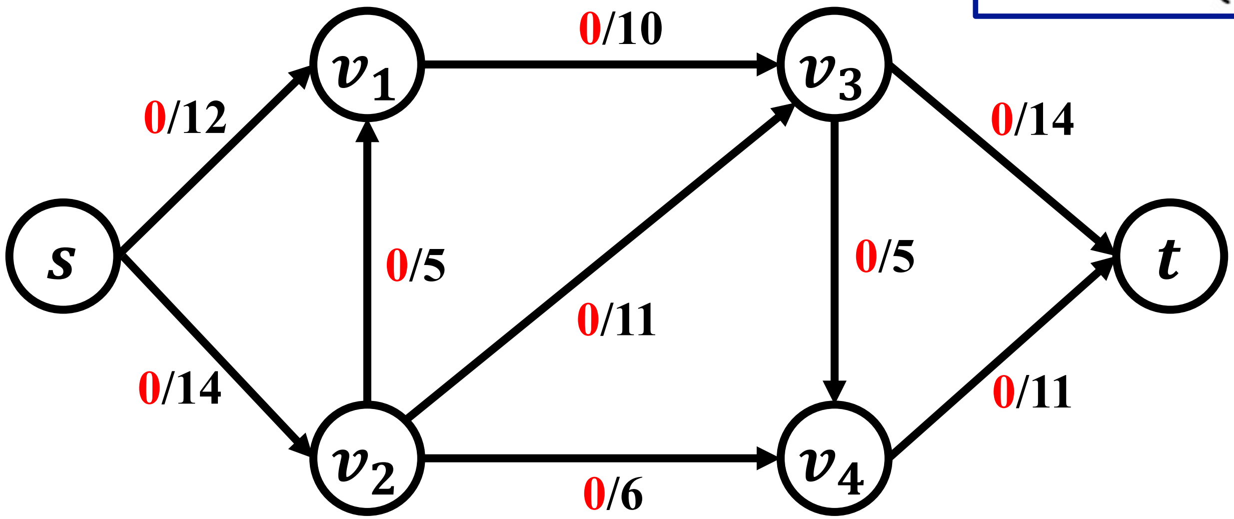
提纲

- 问题背景
- 算法思路
- 算法实例
- 算法分析
- 算法性质

算法实例

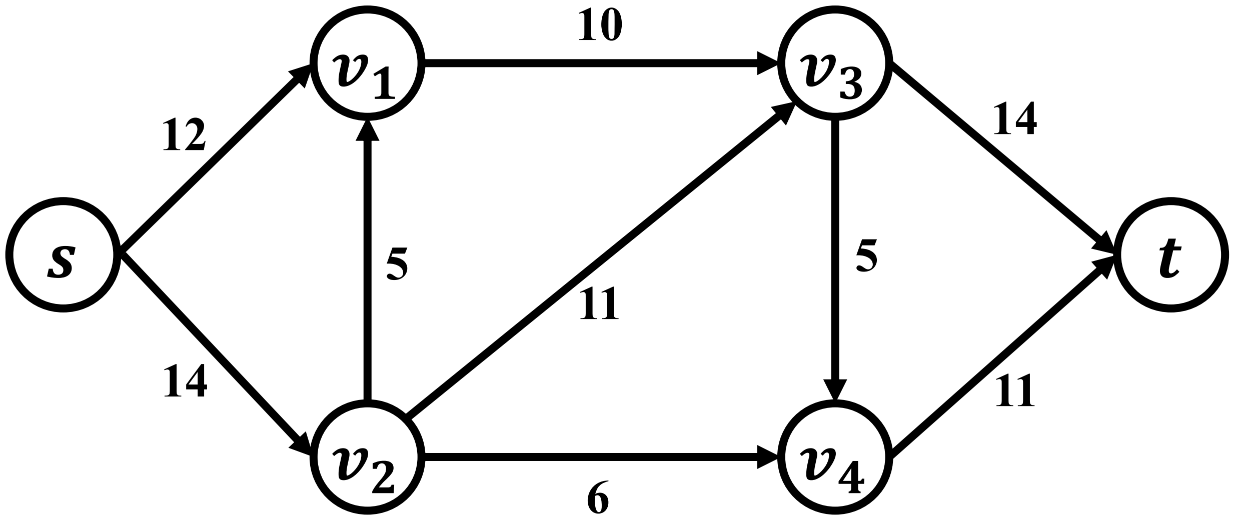
$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

流网络G



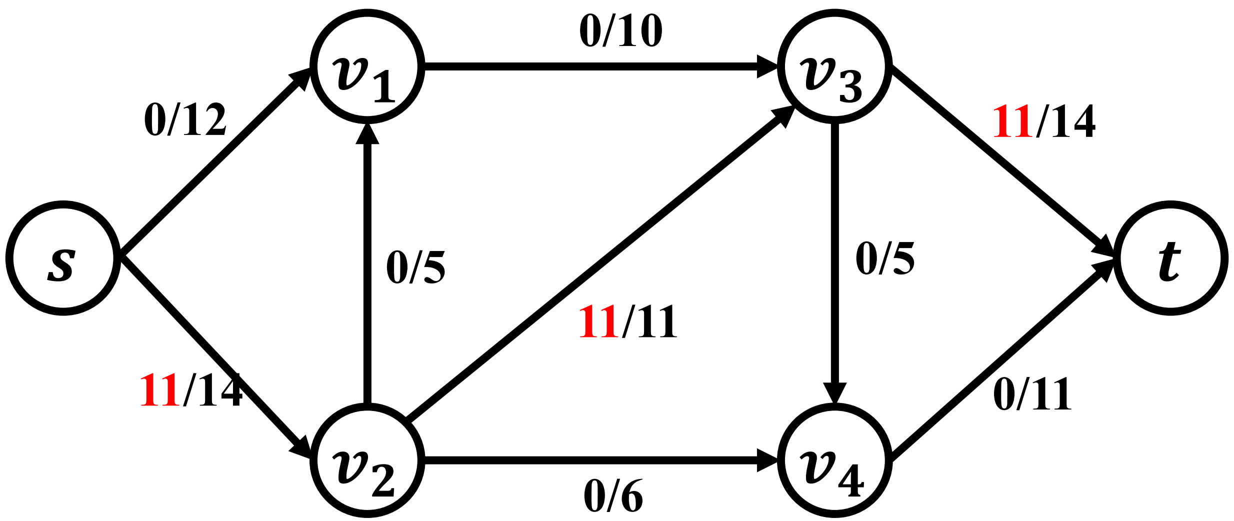
总流量：0

残存网络G_f



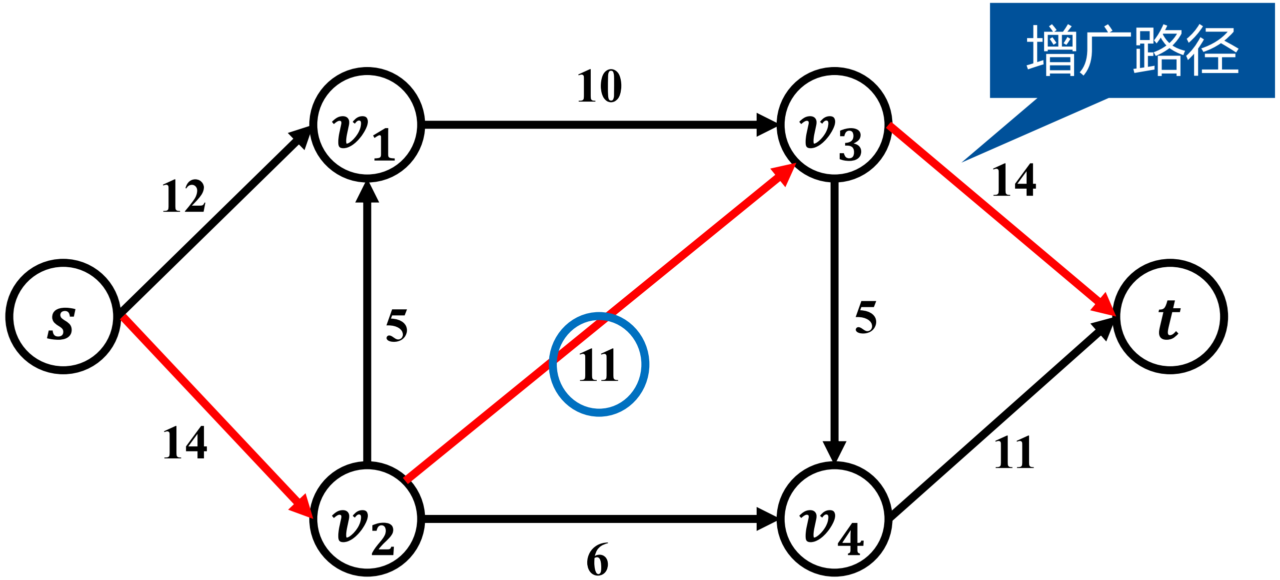
算法实例

流网络G



总流量： $0+11=11$

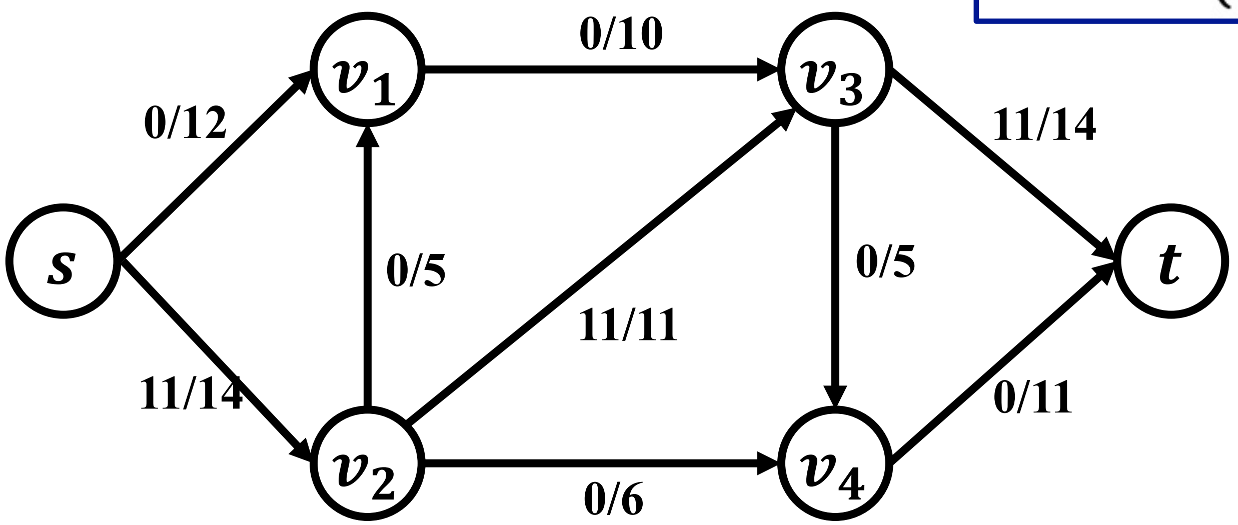
残存网络G_f



算法实例

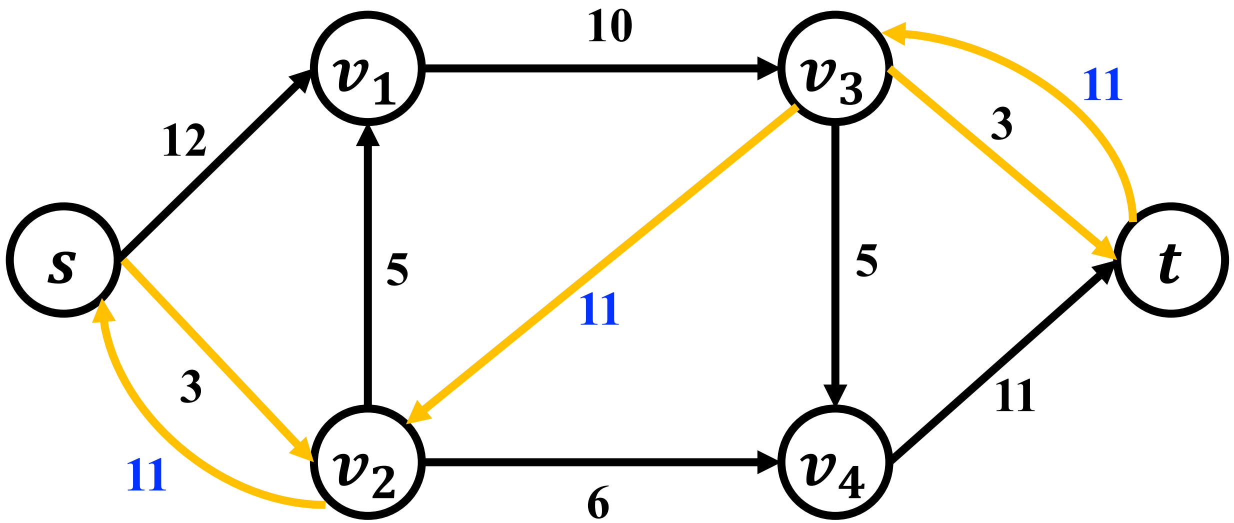
$$c_f(e) = \begin{cases} c(e) - f(e), & e\text{为正向边} \\ f(e), & e\text{为反向边} \end{cases}$$

流网络G



总流量：11

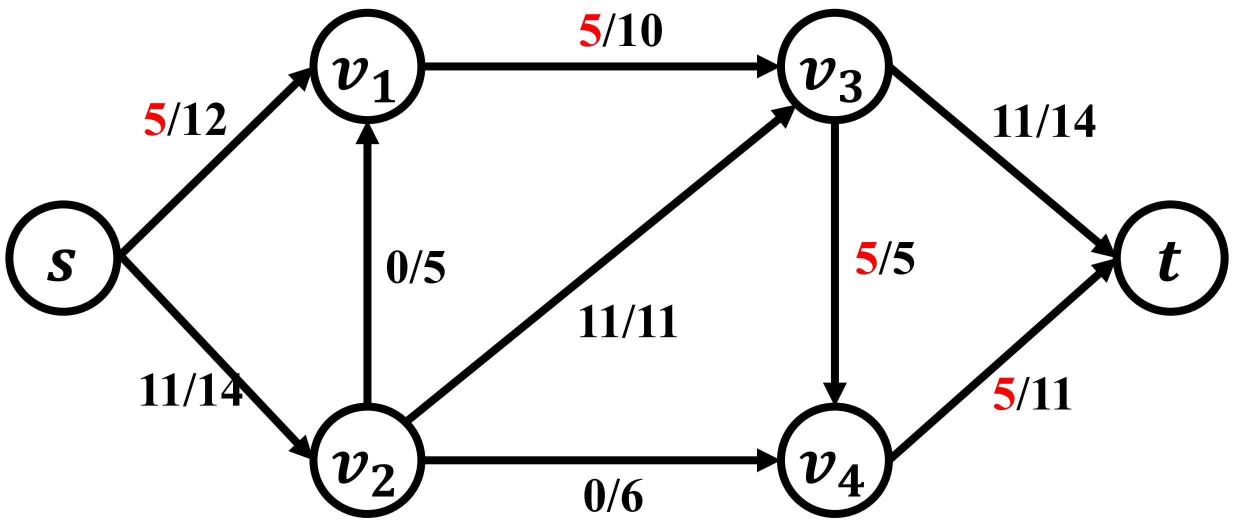
残存网络G_f



更新残存网络

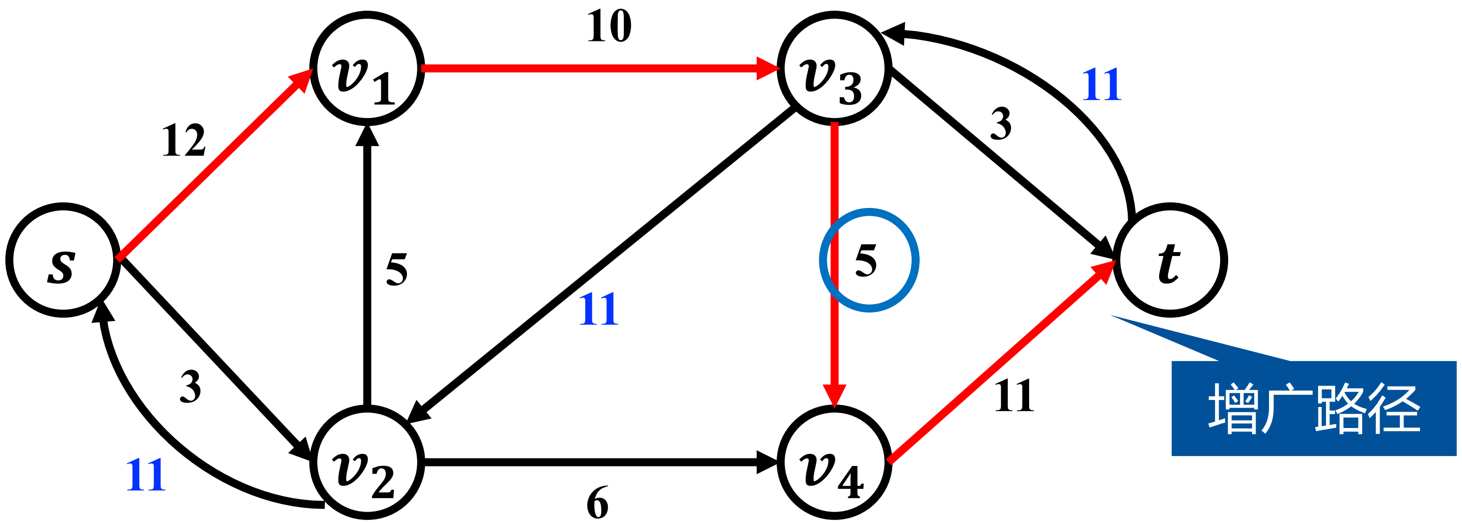
算法实例

流网络G



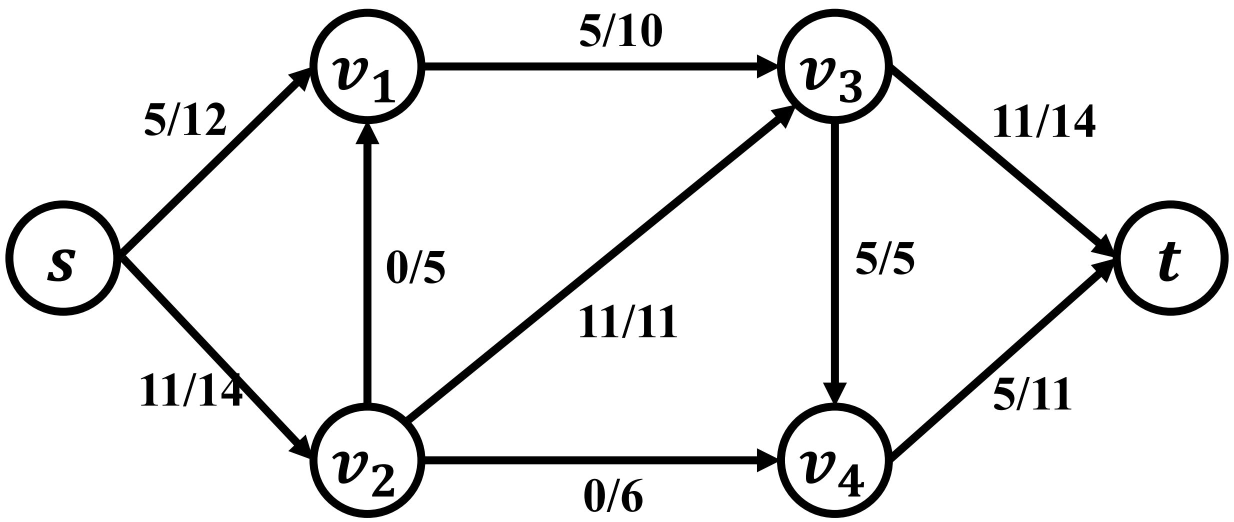
总流量：11+5=16

残存网络G_f



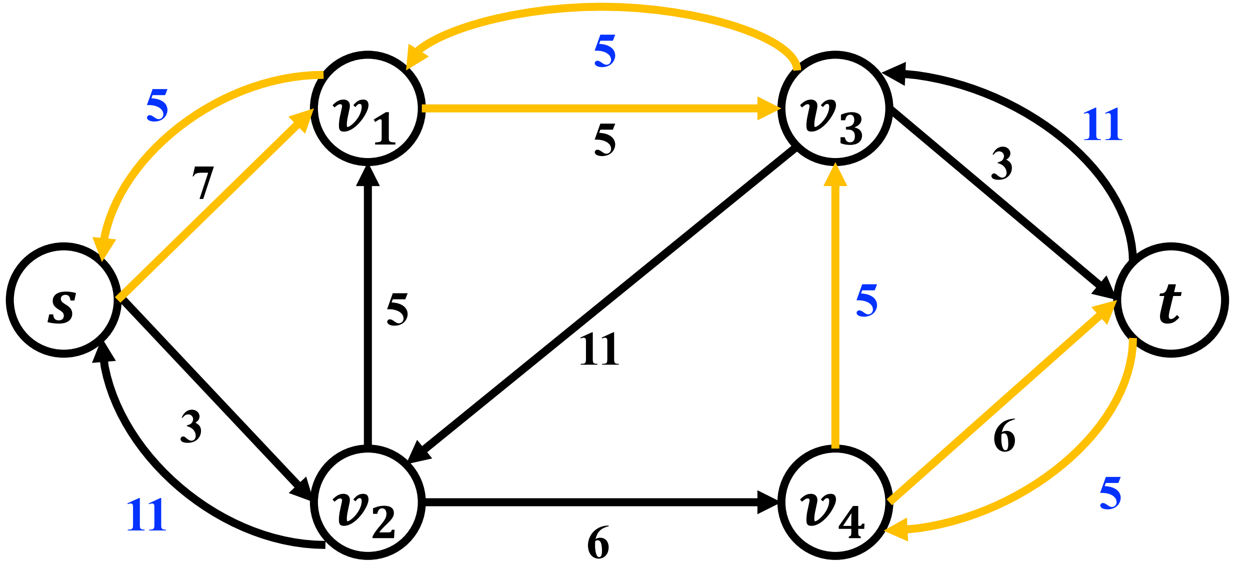
算法实例

流网络G



总流量：16

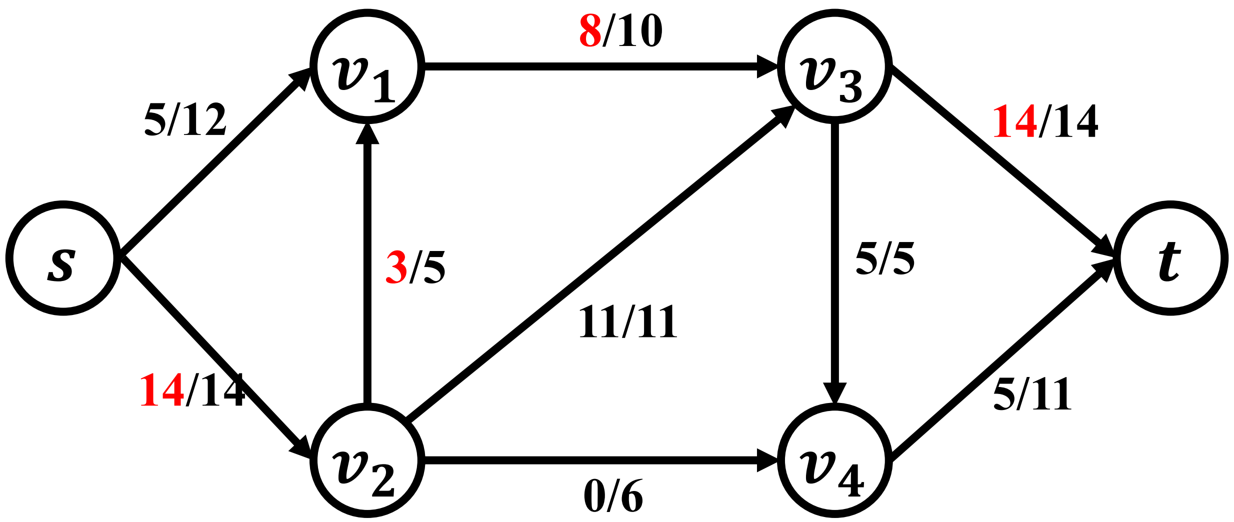
残存网络G_f



更新残存网络

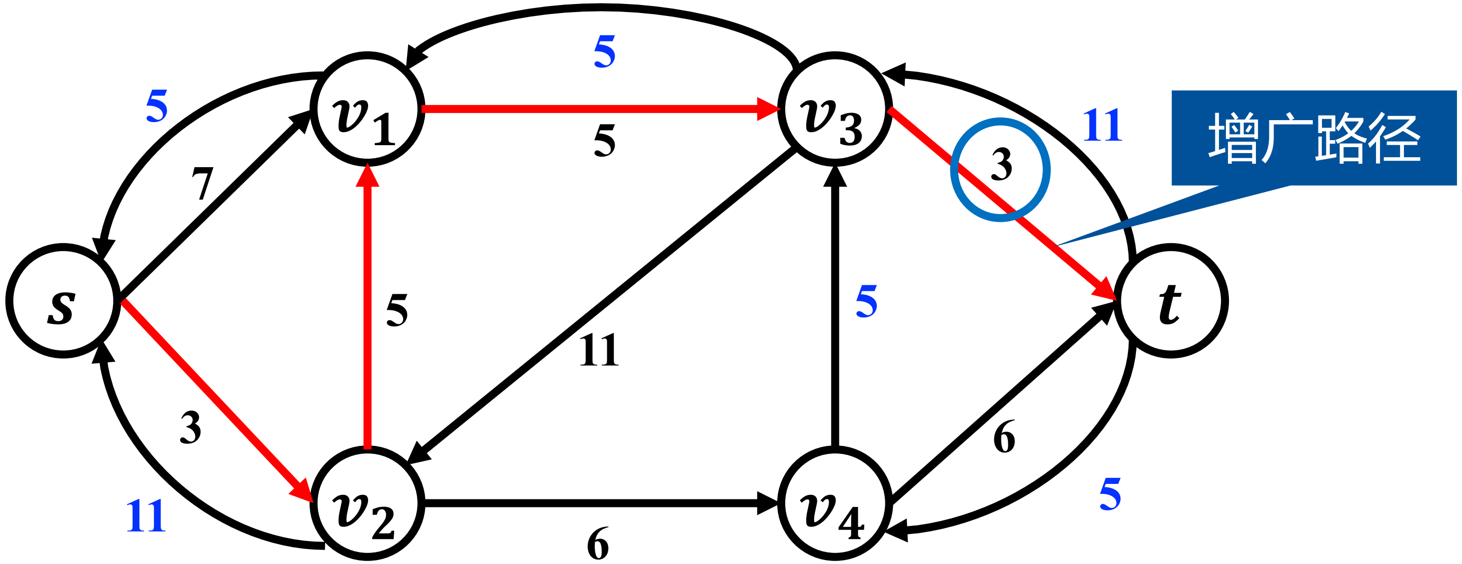
算法实例

流网络G



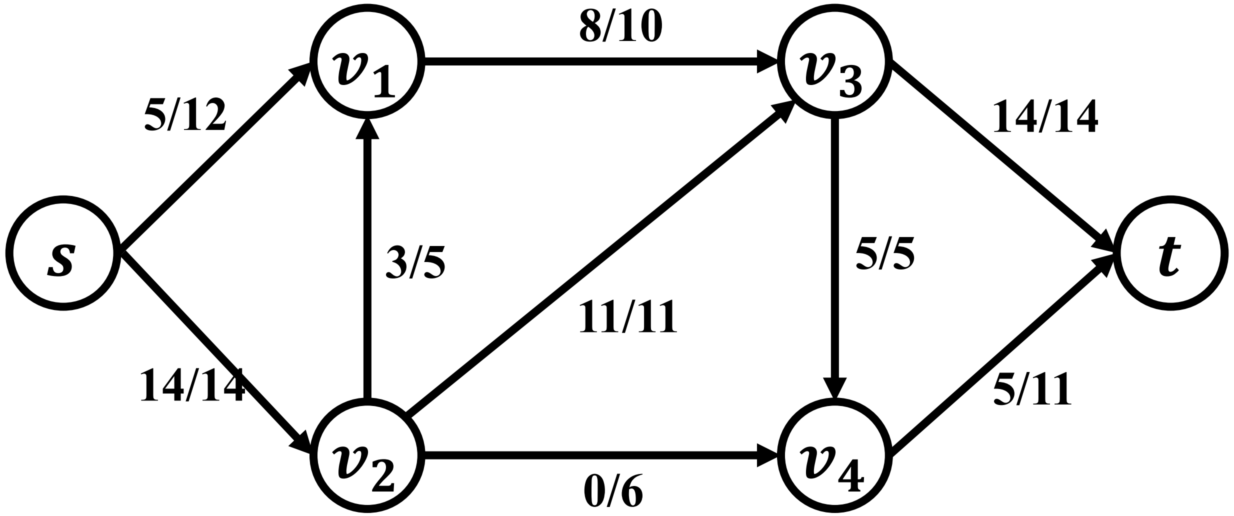
总流量：16+3=19

残存网络G_f



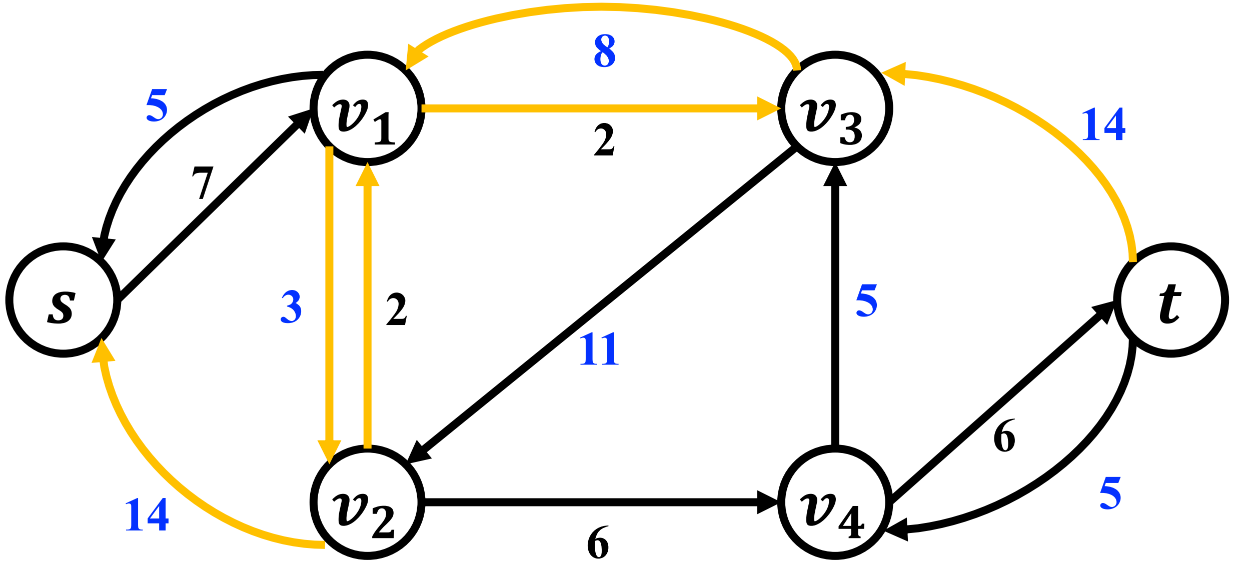
算法实例

流网络G



总流量：19

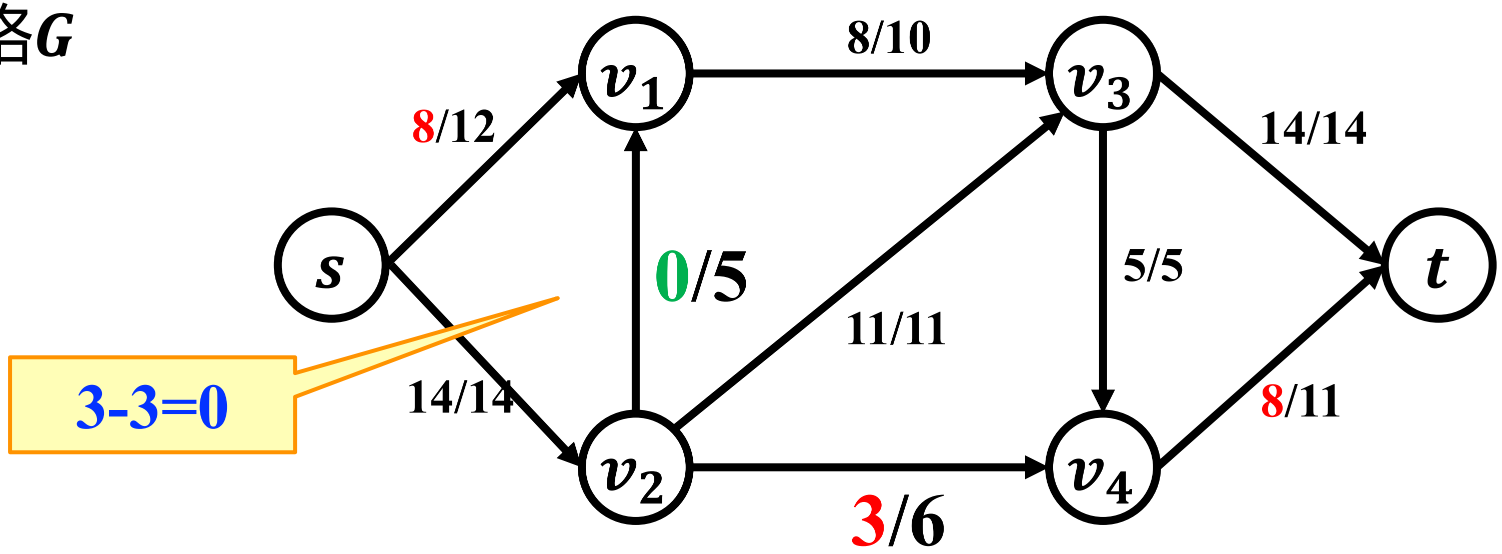
残存网络G_f



更新残存网络

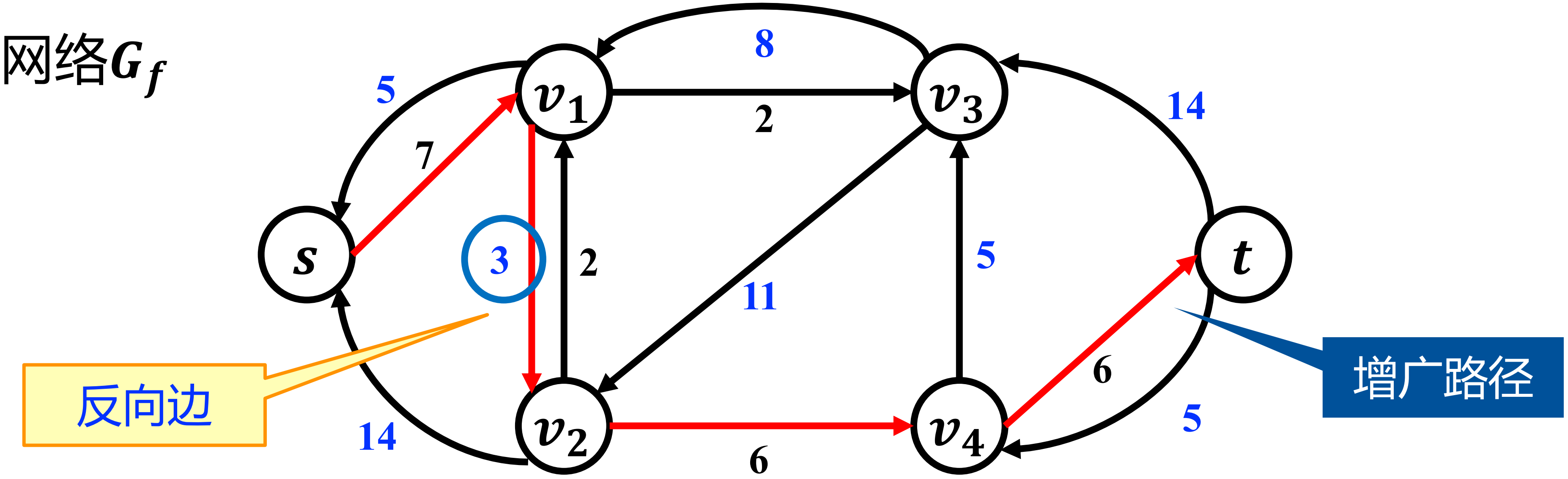
算法实例

流网络G



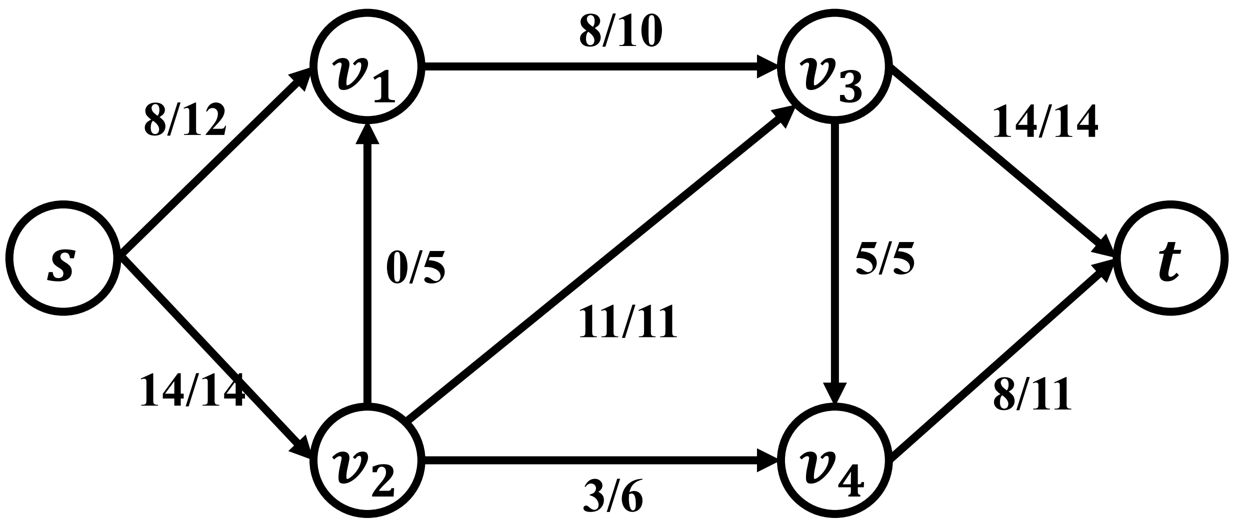
总流量：19+3=22

残存网络G_f



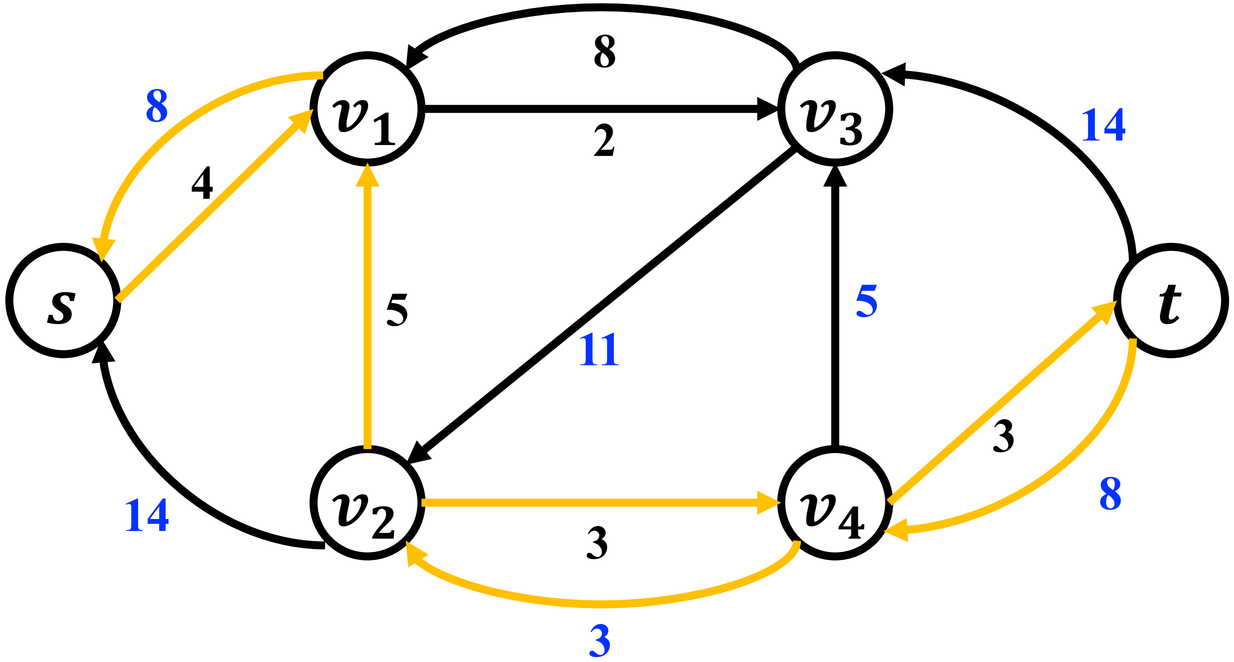
算法实例

流网络G



总流量：22

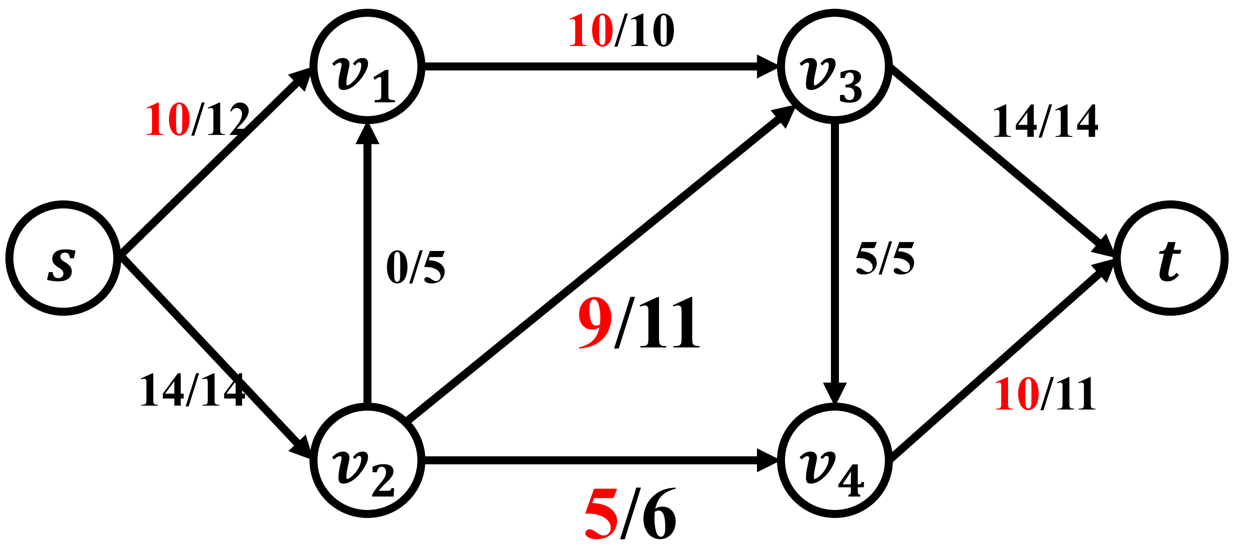
残存网络G_f



更新残存网络

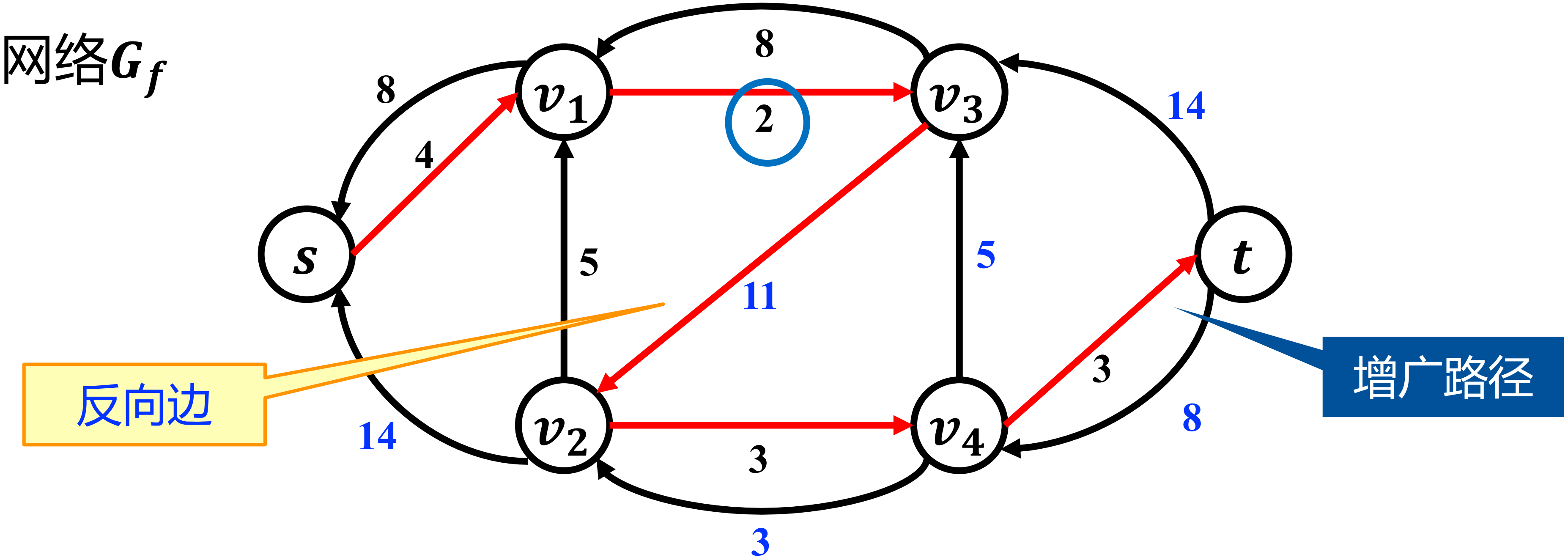
算法实例

流网络G



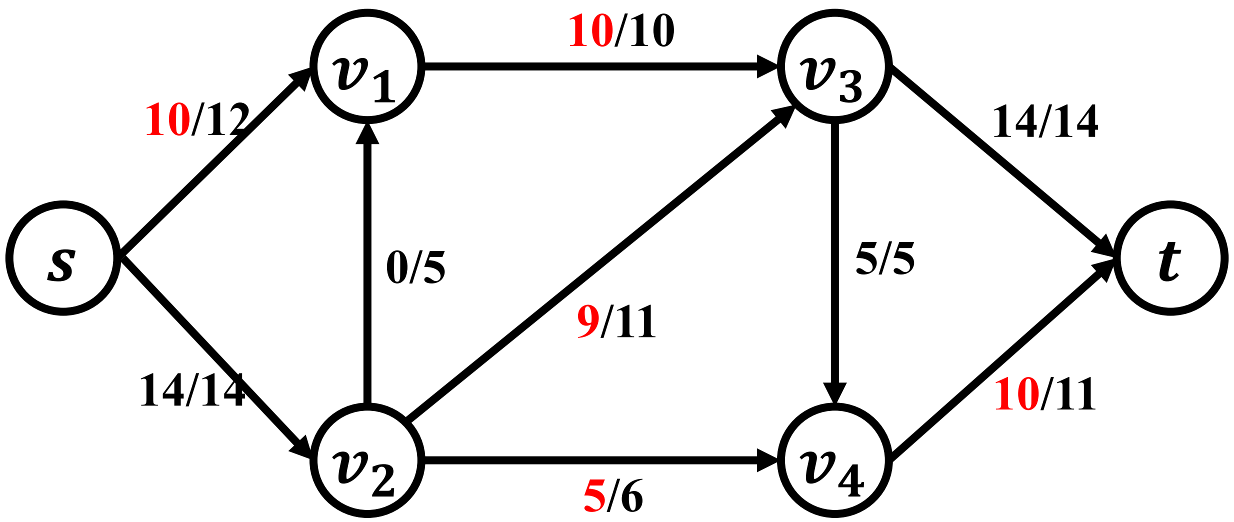
总流量：22+2=24

残存网络G_f



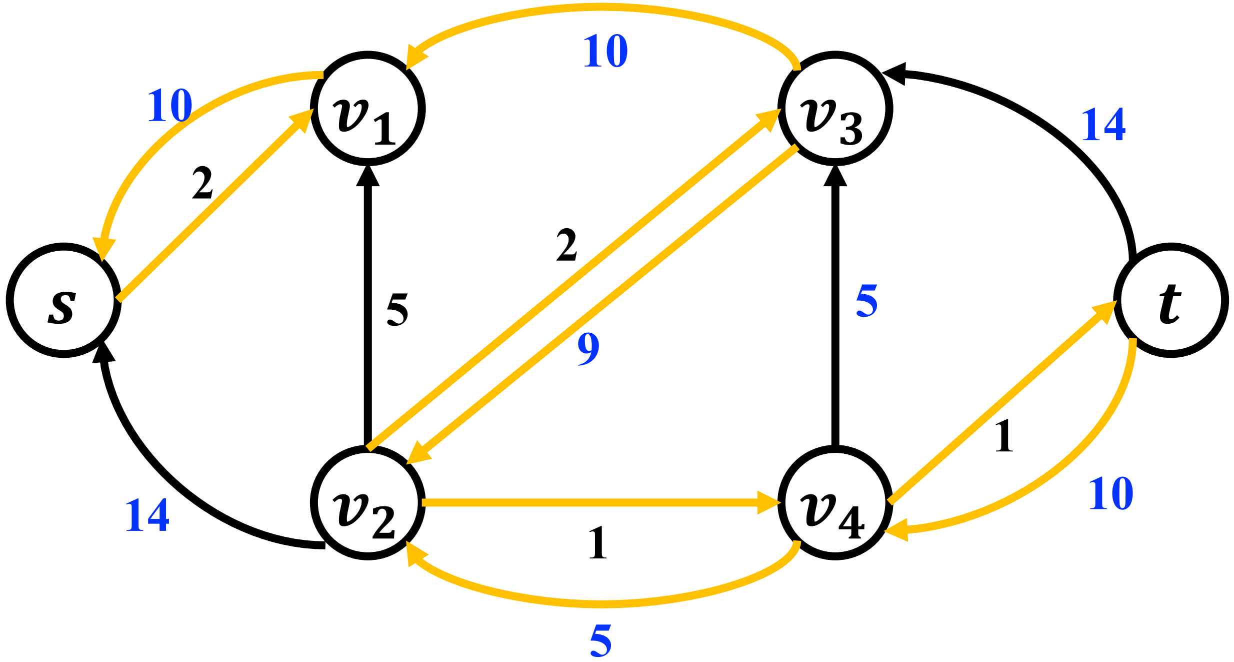
算法实例

流网络G



总流量：24

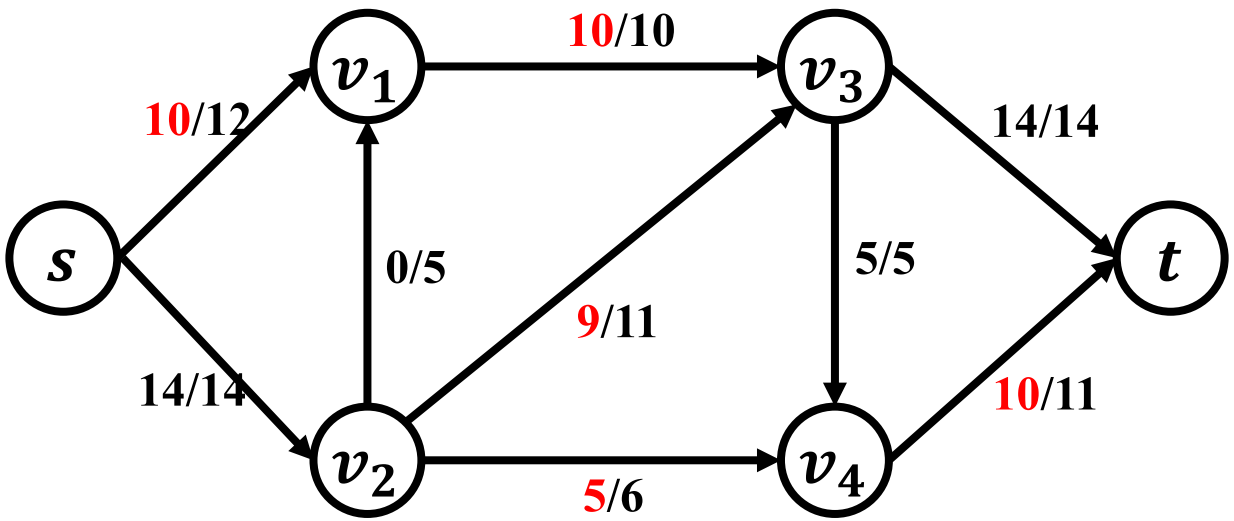
残存网络G_f



更新残存网络

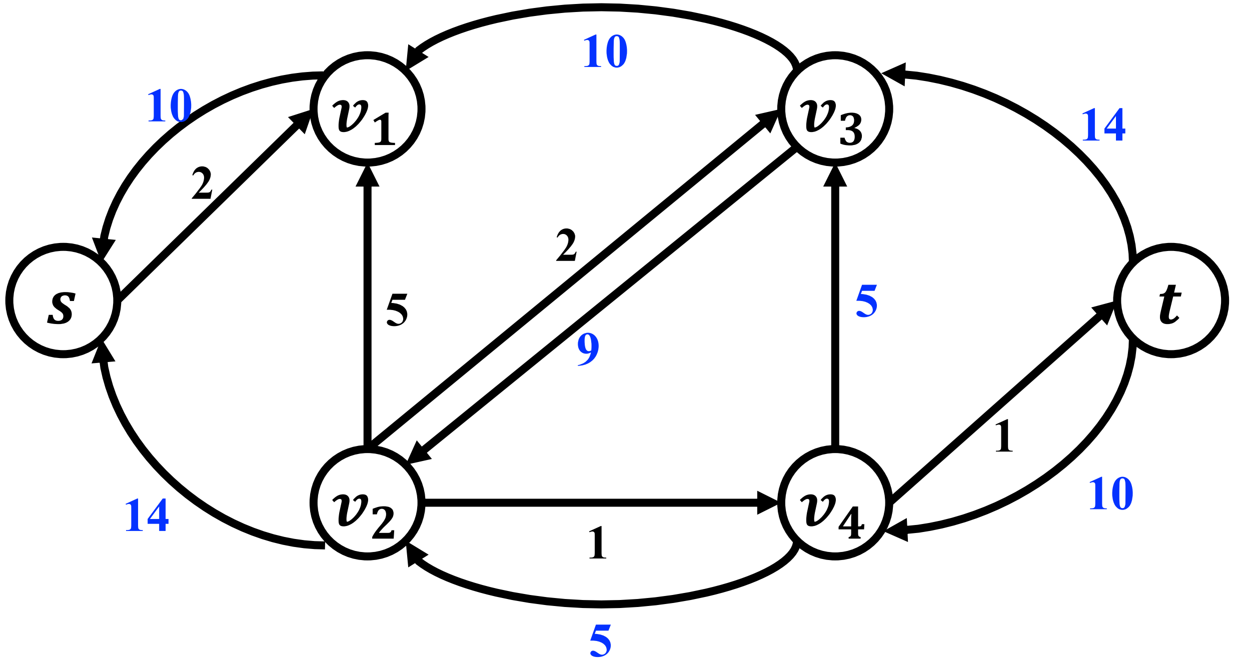
算法实例

流网络G



总流量：24

残存网络G_f



无增广路径
算法可终止

提纲

- 问题背景
- 算法思想
- 算法实例
- **算法分析**
- 算法性质

伪代码

- **Ford-Fulkerson(G, s, t)**

```
输入: 图G =< V, E, C >，源点s，汇点t
输出: 最大流f*
//初始化边的流量
for each edge e∈G.E do
 |  e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
 |  c_f(p) ← min{c_f(e) : e is in p}
 |  //更新流f
 |  for each edge e in p do
 |   |  if e ∈ E then
 |   |   |  e.f ← e.f + c_f(p)
 |   |  end
 |   |  else
 |   |   |  e.f ← e.f − c_f(p)
 |   |  end
 |  end
 |  update G_f
end
f* ← f
return f*
```

初始化边的流量

伪代码

- **Ford-Fulkerson(G, s, t)**

```
输入: 图G =< V, E, C >，源点s，汇点t
输出: 最大流f*
//初始化边的流量
for each edge e∈G.E do
 |  e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
 |  c_f(p) ← min{c_f(e) : e is in p}
 |  //更新流f
 |  for each edge e in p do
 |   |  if e ∈ E then
 |   |   |  e.f ← e.f + c_f(p)
 |   |  end
 |   |  else
 |   |   |  e.f ← e.f − c_f(p)
 |   |  end
 |  end
 |  update G_f
end
f* ← f
return f*
```

构造残存网络

伪代码

- **Ford-Fulkerson(G, s, t)**

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for *each edge* $e \in G.E$ **do**
| $e.f \leftarrow 0$
end
$G_f \leftarrow$ *build residual network*
//在G_f中寻找增广路径
while *there exists a path p from s to t in the residual network* G_f **do**
| $c_f(p) \leftarrow min\{c_f(e) : e \text{ is in } p\}$
| //更新流f
| **for** *each edge e in p* **do**
| | **if** $e \in E$ **then**
| | | $e.f \leftarrow e.f + c_f(p)$
| | **end**
| | **else**
| | | $e.f \leftarrow e.f - c_f(p)$
| | **end**
| **end**
| *update* G_f
end
$f^* \leftarrow f$
return f^*

寻找增广路径

伪代码

- **Ford-Fulkerson(G, s, t)**

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*

```
//初始化边的流量
for each edge e∈G.E do
|  e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
|  c_f(p) ← min{c_f(e) : e is in p}
|  //更新流f
|  for each edge e in p do
|  |  if e ∈ E then
|  |  |  e.f ← e.f + c_f(p)
|  |  end
|  |  else
|  |  |  e.f ← e.f − c_f(p)
|  |  end
|  end
|  update G_f
end
f* ← f
return f*
```

确定增广路径残存容量

伪代码

- **Ford-Fulkerson(G, s, t)**

```
输入: 图G =< V, E, C >，源点s，汇点t
输出: 最大流f*
//初始化边的流量
for each edge e∈G.E do
|   e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
|   c_f(p) ← min{c_f(e) : e is in p}
|   //更新流f
|   for each edge e in p do
|   |   if e ∈ E then
|   |   |   e.f ← e.f + c_f(p)
|   |   end
|   |   else
|   |   |   e.f ← e.f − c_f(p)
|   |   end
|   end
|   update G_f
end
f* ← f
return f*
```

更新流

伪代码

- **Ford-Fulkerson**(G, s, t)

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for *each edge* $e \in G.E$ **do**
  $e.f \leftarrow 0$
end
$G_f \leftarrow$ *build residual network*
//在G_f中寻找增广路径
while *there exists a path p from s to t in the residual network* G_f **do**
  $c_f(p) \leftarrow min\{c_f(e) : e \text{ is in } p\}$
  //更新流f
  for *each edge e in p* **do**
    if $e \in E$ **then**
      $e.f \leftarrow e.f + c_f(p)$
    end
    else
      $e.f \leftarrow e.f - c_f(p)$
    end
  end
  update G_f
end
$f^* \leftarrow f$
return f^*

更新残存网络

复杂度分析

- **Ford-Fulkerson(G, s, t)**

```
输入: 图G =< V, E, C >， 源点s， 汇点t
输出: 最大流f*
//初始化边的流量
for each edge e∈G.E do
 |  e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
 |  c_f(p) ← min{c_f(e) : e is in p}
 |  //更新流f
 |  for each edge e in p do
 |   |  if e ∈ E then
 |   |   |  e.f ← e.f + c_f(p)
 |   |  end
 |   |  else
 |   |   |  e.f ← e.f − c_f(p)
 |   |  end
 |  end
 |  update G_f
end
f* ← f
return f*
```

$O(E)$

复杂度分析

- **Ford-Fulkerson(G, s, t)**

输入: 图$G=<V,E,C>$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for *each edge* $e \in G.E$ **do**
  $e.f \leftarrow 0$
end
$G_f \leftarrow$ *build residual network*
//在G_f中寻找增广路径
while *there exists a path p from s to t in the residual network* G_f **do**
  $c_f(p) \leftarrow min\{c_f(e) : e \text{ is in } p\}$
  //更新流f
  for *each edge e in p* **do**
    if $e \in E$ **then**
      $e.f \leftarrow e.f + c_f(p)$
    end
    else
      $e.f \leftarrow e.f - c_f(p)$
    end
  end
  update G_f
end
$f^* \leftarrow f$
return f^*

$O(E)$

残存网络最多有 $2|E|$ 条边
且$O(|V|) \le O(|E|)$

复杂度分析

- **Ford-Fulkerson(G, s, t)**

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*

```
//初始化边的流量
for each edge e∈G.E do
 |  e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
 |  c_f(p) ← min{c_f(e) : e is in p}
 |  //更新流f
 |  for each edge e in p do
 |   |  if e ∈ E then
 |   |   |  e.f ← e.f + c_f(p)
 |   |  end
 |   |  else
 |   |   |  e.f ← e.f − c_f(p)
 |   |  end
 |  end
 |  update G_f
end
f* ← f
return f*
```

$\}$ $O(E)$

使用DFS，可在$O(V+2E) = O(E)$时间内找到一条增广路

复杂度分析

- **Ford-Fulkerson(G, s, t)**

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for $each\ edge\ e \in G.E$ **do**
　$e.f \leftarrow 0$
end
$G_f \leftarrow build\ residual\ network$
//在G_f中寻找增广路径
while $there\ exists\ a\ path\ p\ from\ s\ to\ t\ in\ the\ residual\ network\ G_f$ **do**
　$c_f(p) \leftarrow min\{c_f(e) : e\ is\ in\ p\}$
　//更新流f
　for $each\ edge\ e\ in\ p$ **do**
　　if $e \in E$ **then**
　　　$e.f \leftarrow e.f + c_f(p)$
　　end
　　else
　　　$e.f \leftarrow e.f - c_f(p)$
　　end
　end
　$update\ G_f$
end
$f^* \leftarrow f$
return f^*

$O(E)$

$O(E)$

$O(E)$

$O(E)$

复杂度分析

- **Ford-Fulkerson(G, s, t)**

输入: 图$G = < V, E, C >$，源点s，汇点t
输出: 最大流f^*

```
//初始化边的流量
for each edge e∈G.E do
    e.f ← 0
end
G_f ← build residual network
//在G_f中寻找增广路径
while there exists a path p from s to t in the residual network G_f do
    c_f(p) ← min{c_f(e) : e is in p}
    //更新流f
    for each edge e in p do
        if e ∈ E then
            e.f ← e.f + c_f(p)
        end
        else
            e.f ← e.f − c_f(p)
        end
    end
    update G_f
end
f* ← f
return f*
```

每次循环中，
流的值至少提升一个单位

复杂度分析

- **Ford-Fulkerson(G, s, t)**

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for *each edge* $e \in G.E$ **do**
  $e.f \leftarrow 0$
end
$G_f \leftarrow$ *build residual network*
//在G_f中寻找增广路径
while *there exists a path p from s to t in the residual network* G_f **do**
  $c_f(p) \leftarrow min\{c_f(e) : e \text{ is in } p\}$
  //更新流f
  for *each edge e in p* **do**
    if $e \in E$ **then**
      $e.f \leftarrow e.f + c_f(p)$
    end
    else
      $e.f \leftarrow e.f - c_f(p)$
    end
  end
  update G_f
end
$f^* \leftarrow f$
return f^*

每次循环后，
流的值至少增加1

最多循环$|f^*|$次

复杂度分析

- **Ford-Fulkerson(G, s, t)**

输入: 图$G =< V, E, C >$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for $each\ edge\ e{\in}G.E$ **do**
　$e.f \leftarrow 0$
end
$G_f \leftarrow build\ residual\ network$
//在G_f中寻找增广路径
while $there\ exists\ a\ path\ p\ from\ s\ to\ t\ in\ the\ residual\ network\ G_f$ **do**
　$c_f(p) \leftarrow min\{c_f(e) : e\ is\ in\ p\}$
　//更新流f
　for $each\ edge\ e\ in\ p$ **do**
　　if $e \in E$ **then**
　　　$e.f \leftarrow e.f + c_f(p)$
　　end
　　else
　　　$e.f \leftarrow e.f - c_f(p)$
　　end
　end
　$update\ G_f$
end
$f^* \leftarrow f$
return f^*

$O(E)$

$O(E \cdot |f^*|)$

复杂度分析

- **Ford-Fulkerson(G, s, t)**

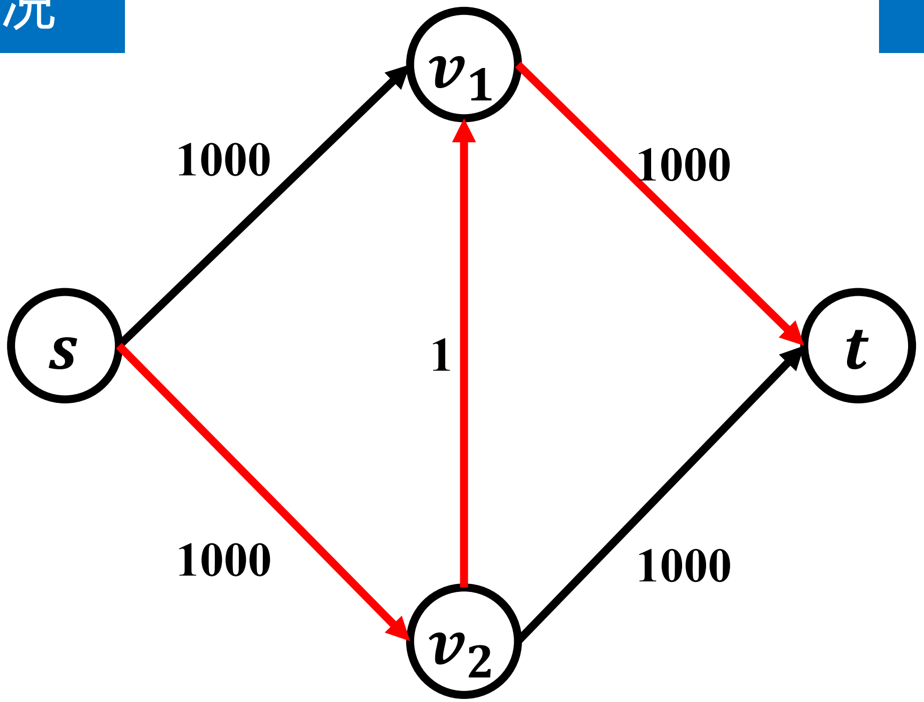
输入: 图$G = < V, E, C >$，源点s，汇点t
输出: 最大流f^*
//初始化边的流量
for *each edge* $e \in G.E$ **do**
  $e.f \leftarrow 0$
end
$G_f \leftarrow$ *build residual network*
//在G_f中寻找增广路径
while *there exists a path p from s to t in the residual network* G_f **do**
  $c_f(p) \leftarrow min\{c_f(e) : e \text{ is in } p\}$
  //更新流f
  for *each edge e in p* **do**
    if $e \in E$ **then**
      $e.f \leftarrow e.f + c_f(p)$
    end
    else
      $e.f \leftarrow e.f - c_f(p)$
    end
  end
  update G_f
end
$f^* \leftarrow f$
return f^*

时间复杂度$O(E \cdot |f^*|)$

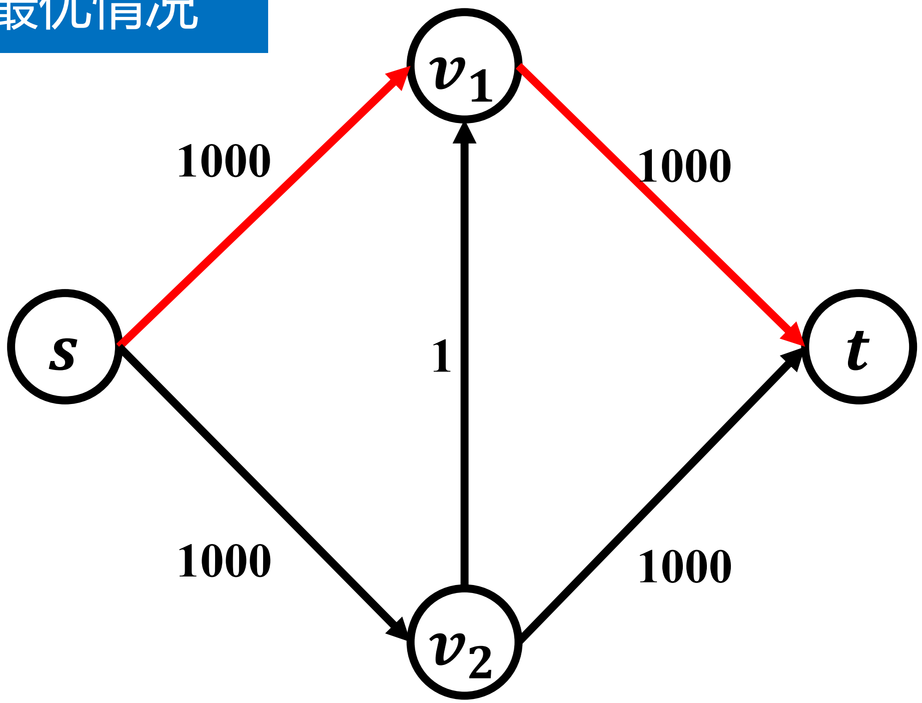
复杂度分析

- 运行时间依赖于增广过程

最坏情况

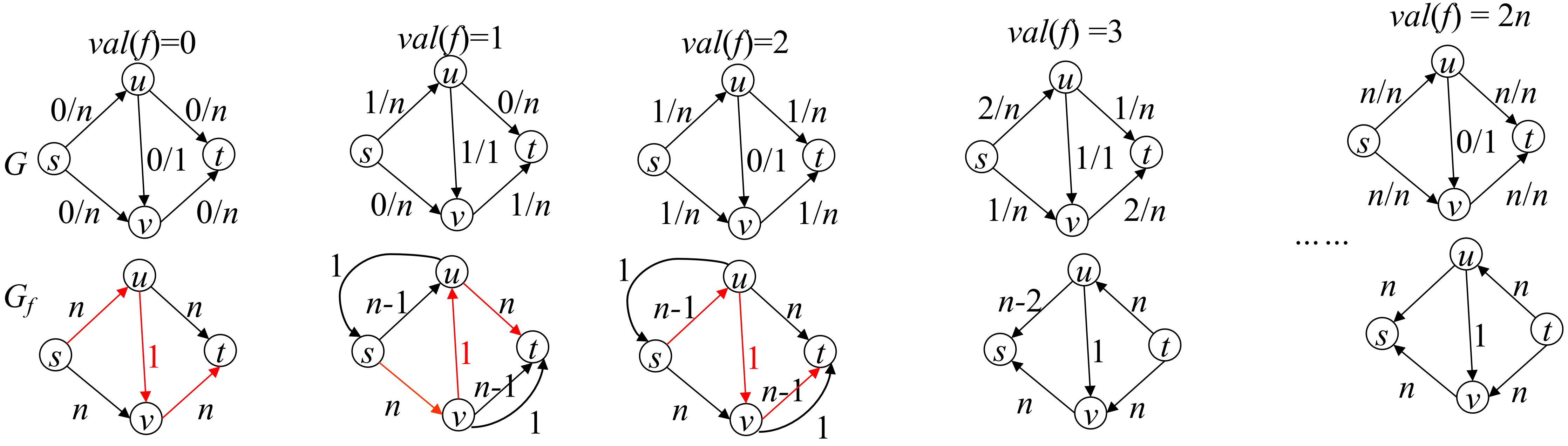


最优情况



复杂度分析

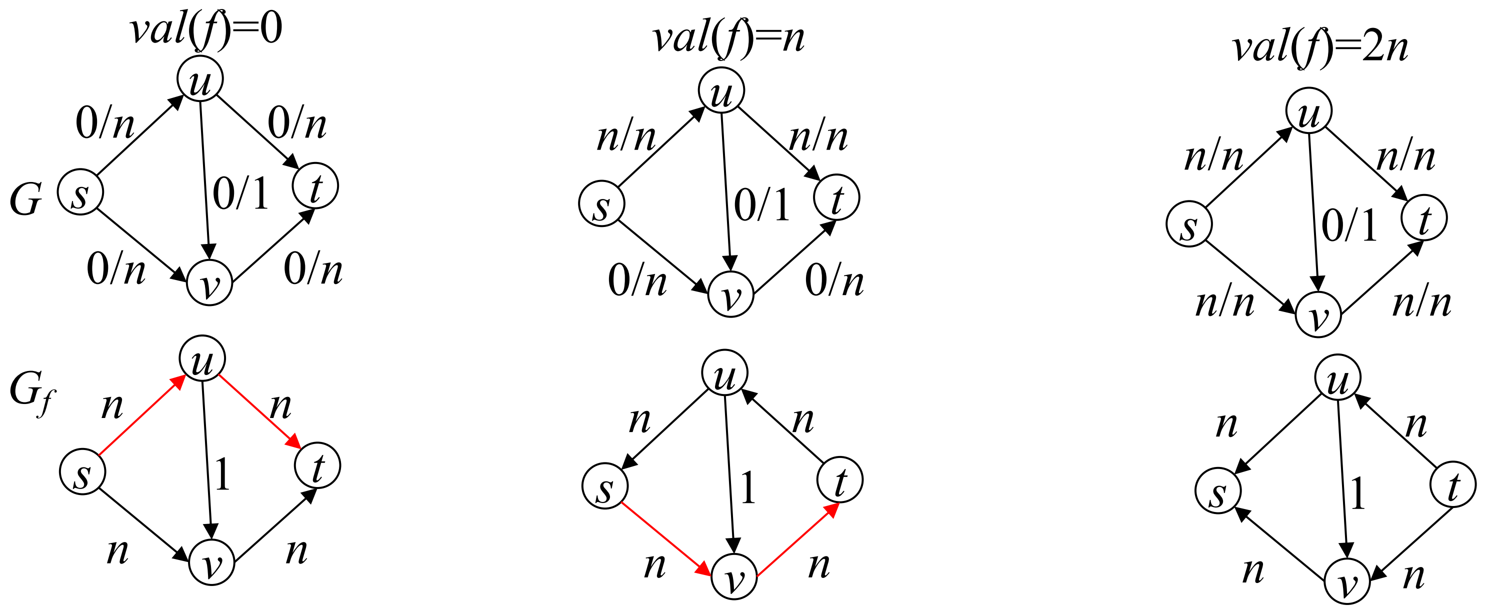
- 运行时间依赖于增广过程



$2n$ 次增广

复杂度分析

- 运行时间依赖于增广过程



2 次增广

提纲

问题背景

算法思想

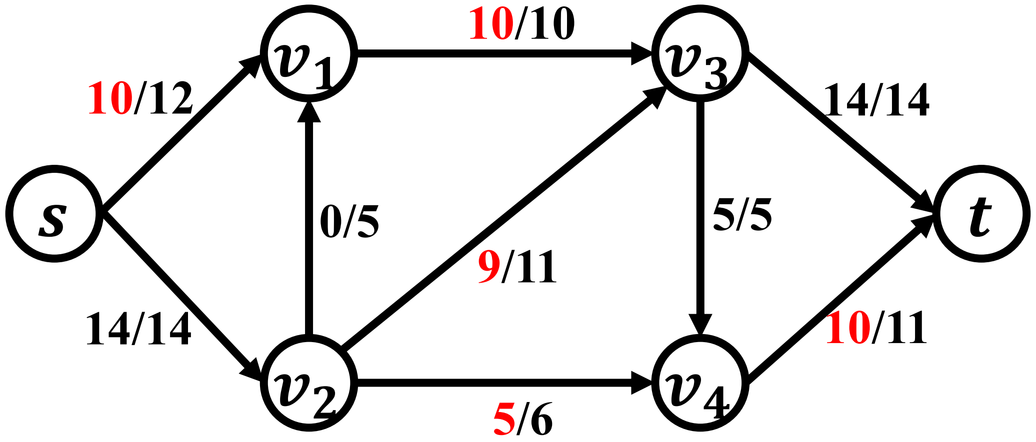
算法实例

算法分析

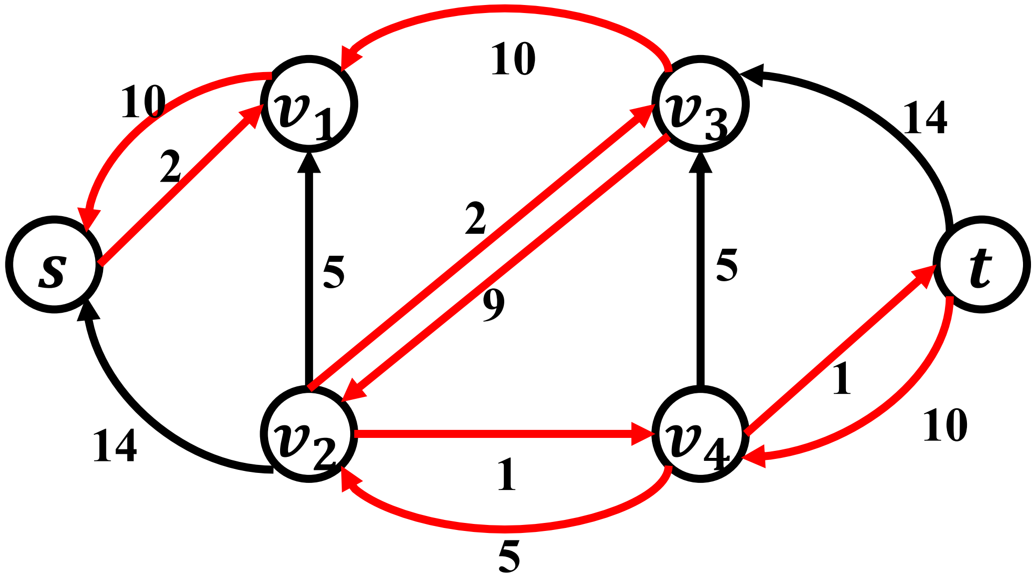
算法性质

正确性证明

流网络G



流网络G_f



问题：如何证明Ford-Fulkerson算法可获得最优解？

正确性证明

- $\boldsymbol{f}$是最大流 $\Leftrightarrow$ 残存网络$\boldsymbol{G_f}$中无增广路径

证明：（充分性：$\boldsymbol{f}$是最大流 $\Longrightarrow$ 残存网络 $\boldsymbol{G_f}$ 中无增广路径）

反证：如果流 $\boldsymbol{f}$ 有增广路 径$\boldsymbol{p}$ ，则总流量 $|\boldsymbol{f}|$ 可被增加，$\boldsymbol{f}$ 不是最大流，矛盾。

正确性证明

- f是最大流 $\Leftrightarrow$ 残存网络G_f中无增广路径

证明：（充分性：f是最大流 $\Rightarrow$ 残存网络 G_f 中无增广路径）

反证：如果流 f 有增广路 径p ，则总流量 $|f|$ 可被增加，f 不是最大流，矛盾。

（必要性：f是最大流 $\Leftarrow$ 残存网络 G_f 中无增广路径）

引入割的概念，构造证明桥梁

正确性证明

- f是最大流$\Leftrightarrow$残存网络G_f中无增广路径

  - 必要性：f是最大流$\overset{?}{\Leftarrow}$ $\overset{?}{\Leftarrow}$残存网络G_f中无增广路径

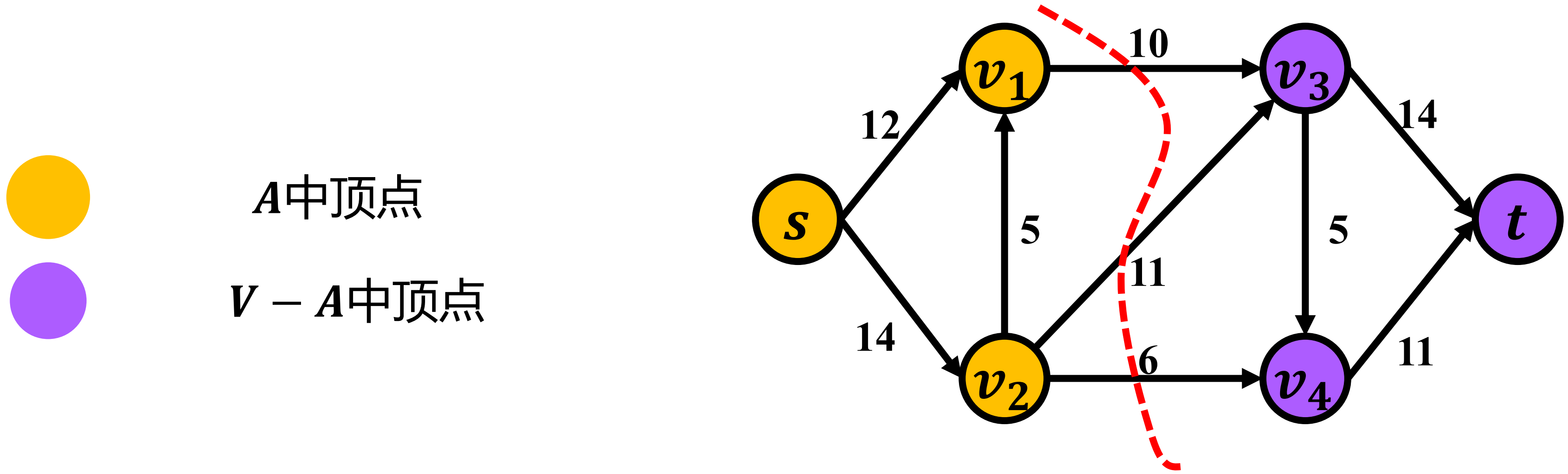
寻找最大流问题的上界

建立容量/流量和图结构的关系

引入割的概念，构造证明桥梁

割(Cut)

- 流网络的割
  - 图$G =< V, E, C >$是一个有向图，割$(A, V - A)$将图G的顶点集V划分为两部分
  - 其中源点$s \in A$，汇点 $t \in V - A$



割的容量

- 横跨(Cross)
  - 给定割$(A, V - A)$和边 (u, v)，$u \in A, v \in V - A$，称边 (u, v) 横跨割$(A, V - A)$

- 割的容量

$$cap(A, V - A) = \sum_{e \text{ out of } A} c(e)$$

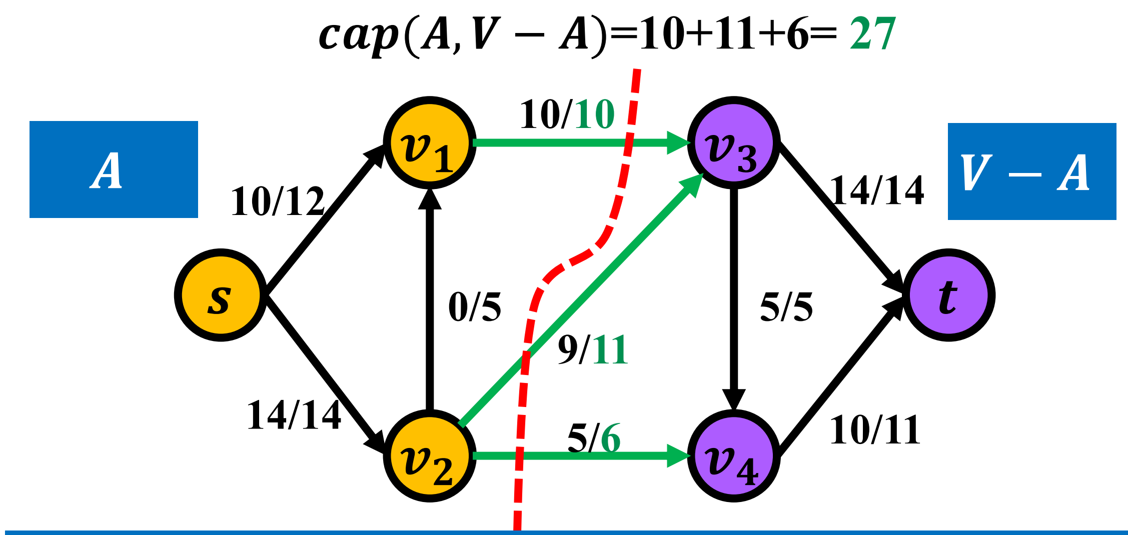
不同的割可能具有不同的容量

割的容量

- 横跨(Cross)
  - 给定割$(A, V-A)$和边 (u,v) , $u \in A, v \in V-A$, 称边 (u,v) 横跨割$(A, V-A)$

- 割的容量

$$cap(A, V-A) = \sum_{e \text{ out of } A} c(e)$$



不同的割可能具有不同的容量

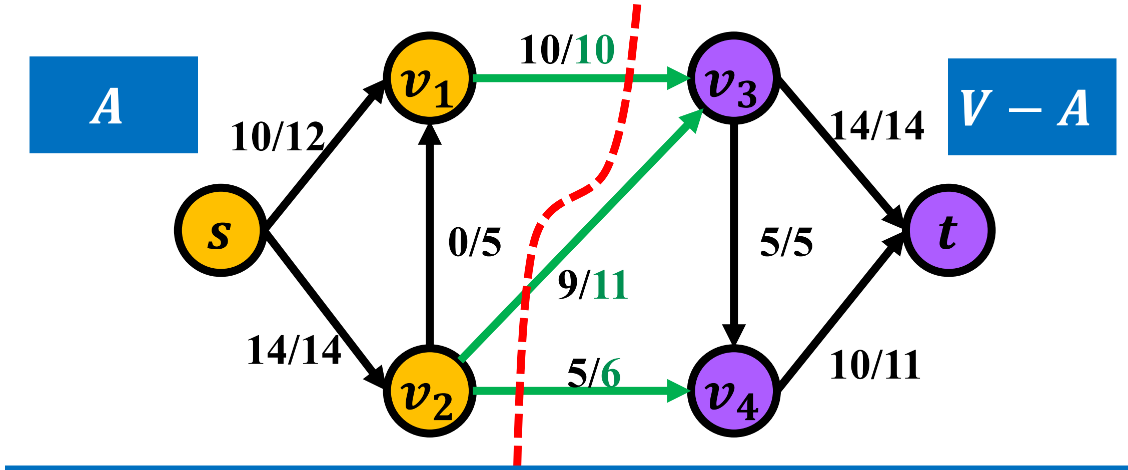
割的容量

- 横跨(Cross)
  - 给定割$(A, V-A)$和边 (u,v) , $u \in A, v \in V-A$, 称边 (u,v) 横跨割$(A, V-A)$

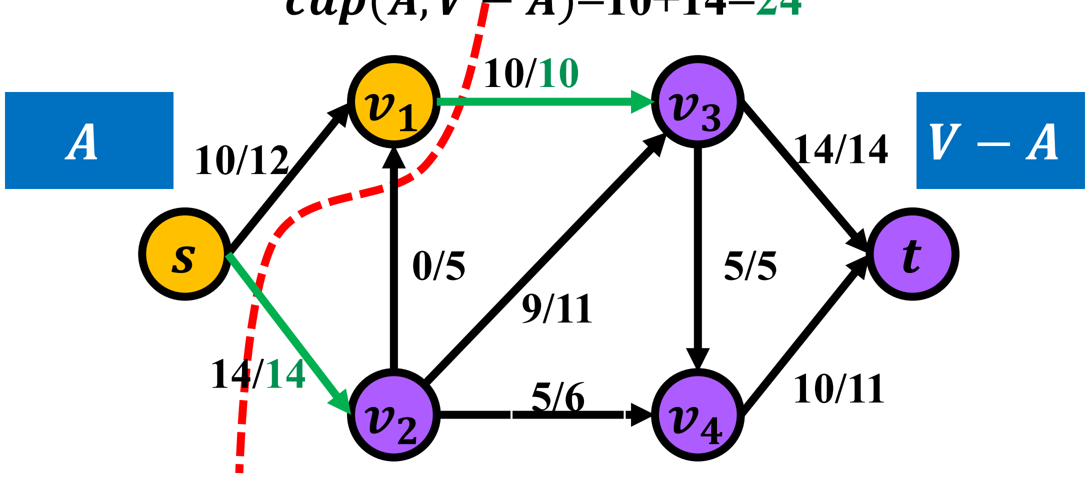
- 割的容量

$$cap(A, V-A) = \sum_{e\ out\ of\ A} c(e)$$

$cap(A, V-A)$=10+11+6= 27



$cap(A, V-A)$=10+14=24

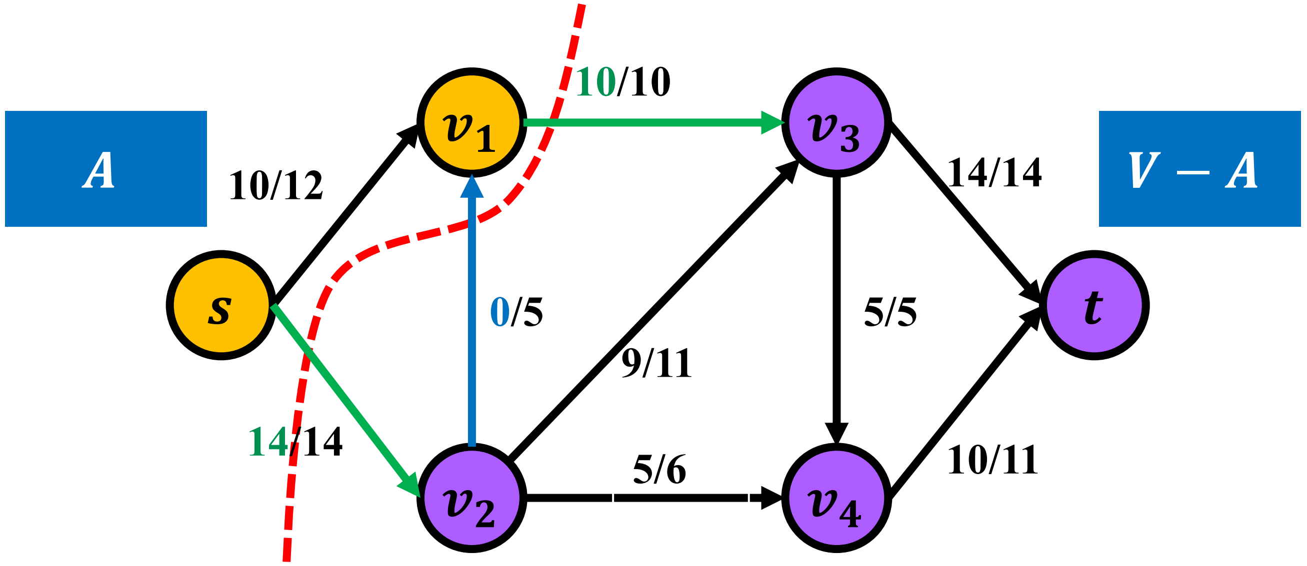


不同的割可能具有不同的容量

流值定理

- 给定割$(A, V-A)$和流 f ，那么

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

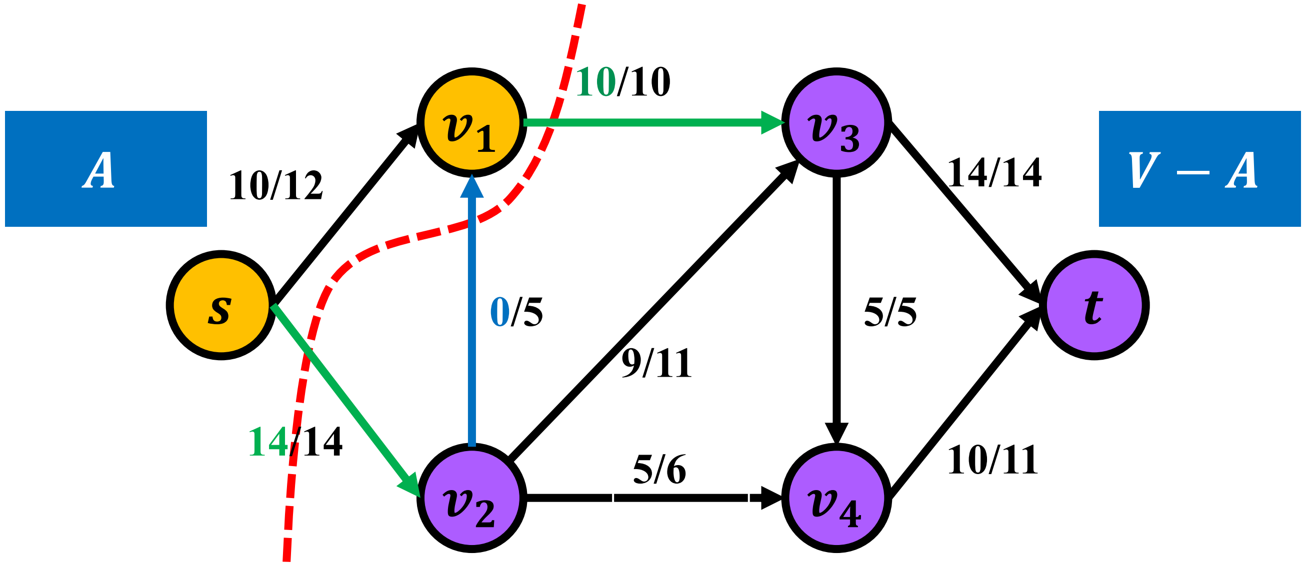


流值定理

- 给定割$(\boldsymbol{A}, \boldsymbol{V}-\boldsymbol{A})$和流 $\boldsymbol{f}$ ，那么

$$\boldsymbol{val(f)} = \sum_{e\ out\ of\ A} f(e) - \sum_{e\ in\ to\ A} f(e)$$

流过割的流量=10+14-0

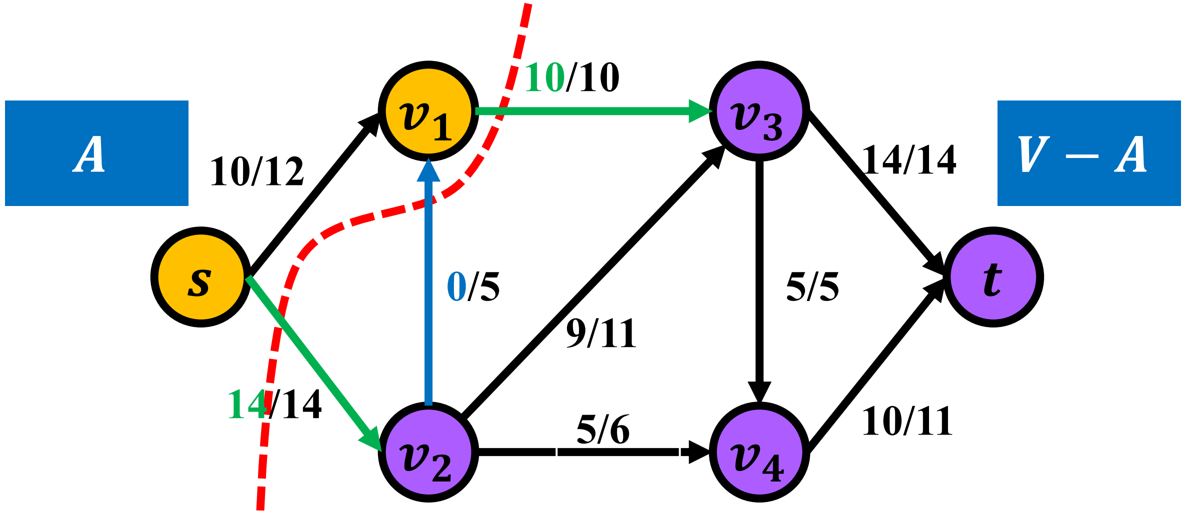


流值定理

- 给定割$(A, V-A)$和流 f，那么

$$val(f) = \sum_{e\ out\ of\ A} f(e) - \sum_{e\ in\ to\ A} f(e)$$

- 证明：



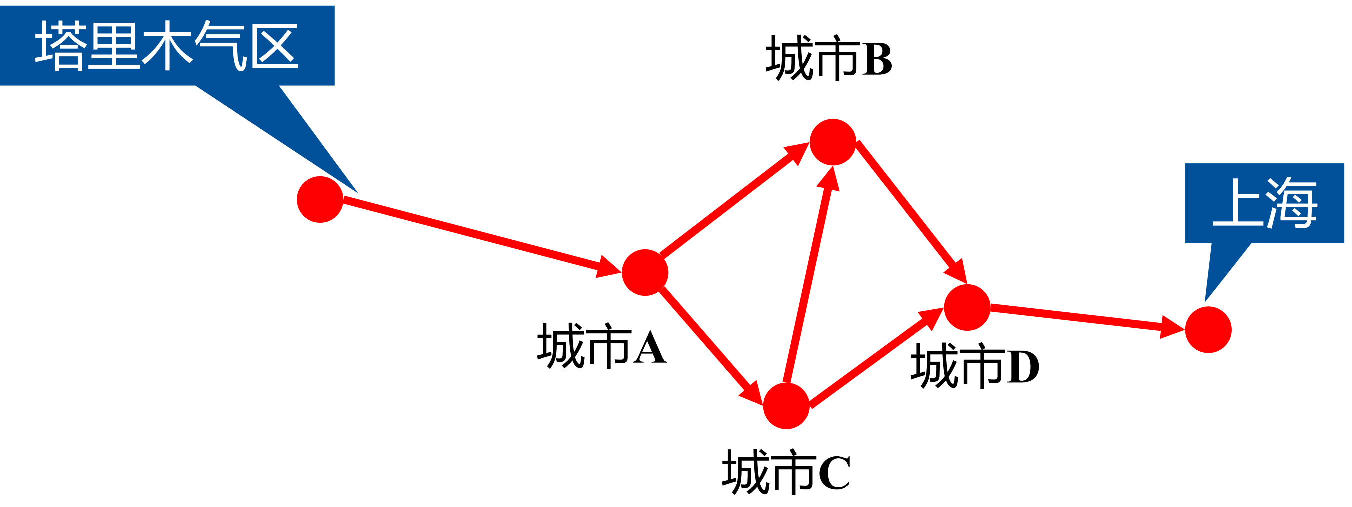
$$val(f) = \sum_{e\ out\ of\ s} f(e)$$

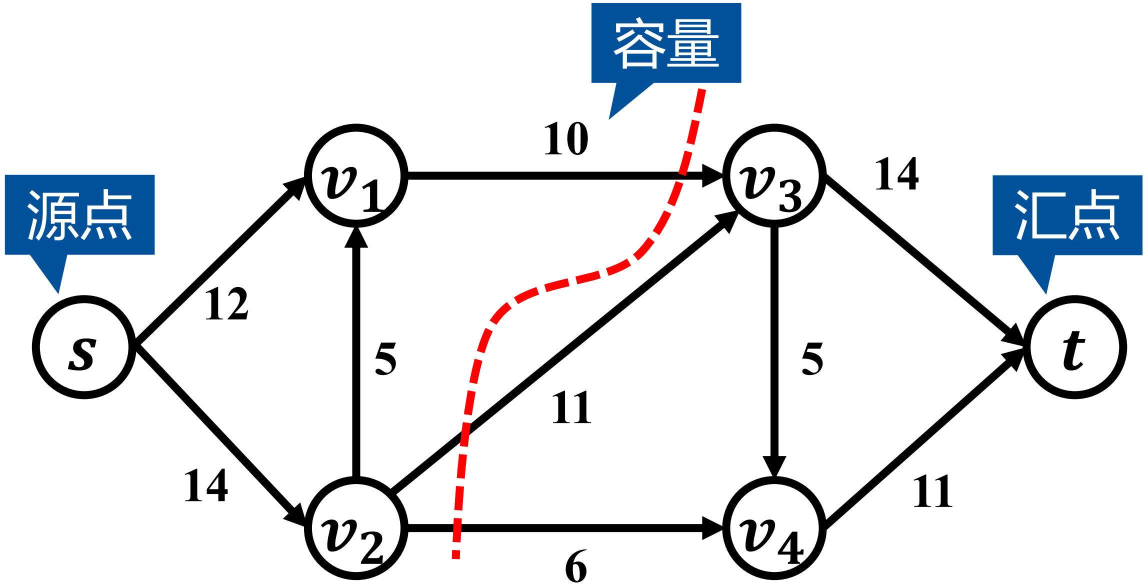
$$= \sum_{v \in A} \left(\sum_{e\ out\ of\ v} f(e) - \sum_{e\ in\ to\ v} f(e) \right)$$

$$= \sum_{e\ out\ of\ A} f(e) - \sum_{e\ in\ to\ A} f(e)$$

最小割

- 最大流：计算最大的天然气输送量
- 最小割：寻找天然气输送管道的瓶颈总容量





流网络$G=<V,E,C>$

最大流的流值 ≤ 最小割的容量（上界）

正确性证明

- $\boldsymbol{f}$ 是最大流 $\Leftrightarrow$ 残存网络 $\boldsymbol{G_f}$ 中无增广路径
  - 必要性：$\boldsymbol{f}$是最大流$\overset{?}{\Longleftarrow}$存在割是最大流的上界$\overset{?}{\Longleftarrow}$残存网络$\boldsymbol{G_f}$中无增广路径

正确性证明

- f是最大流$\Leftrightarrow$残存网络G_f中无增广路径
  - 必要性：f是最大流$\Leftarrow$存在割是最大流的上界$\Leftarrow$残存网络G_f中无增广路径

步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A) \Leftarrow$ 残存网络G_f中无增广路径

正确性证明

- $\boldsymbol{f}$是最大流⇔残存网络$\boldsymbol{G_f}$中无增广路径
  - 必要性：$\boldsymbol{f}$是最大流⟸存在割是最大流的上界⟸残存网络$\boldsymbol{G_f}$中无增广路径

步骤1：存在割$(\boldsymbol{A},\boldsymbol{V}-\boldsymbol{A})$使得$\boldsymbol{val}(\boldsymbol{f})=\boldsymbol{cap}(\boldsymbol{A},\boldsymbol{V}-\boldsymbol{A})$ ⟸ 残存网络$\boldsymbol{G_f}$中无增广路径

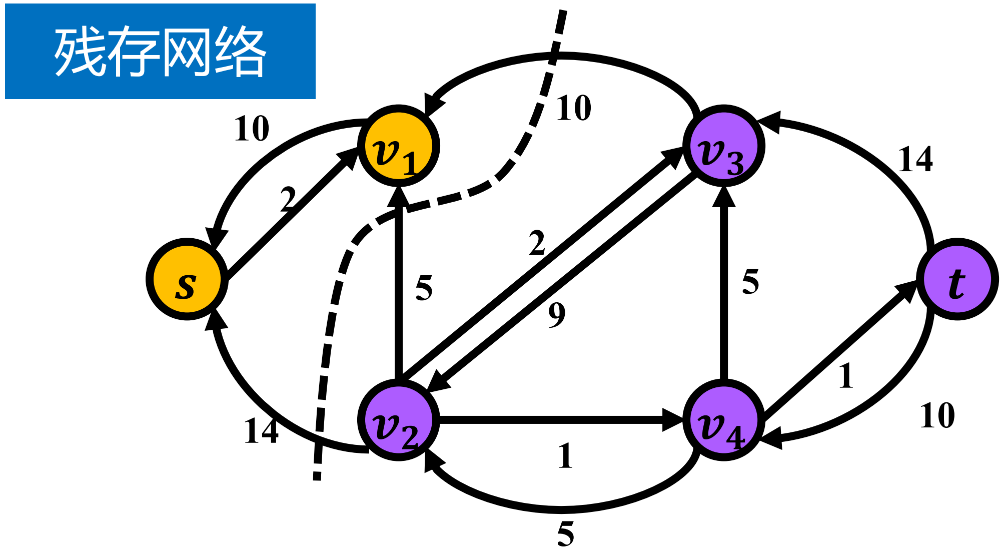
步骤2：$\boldsymbol{f}$ 是最大流 ⟸ 存在割$(\boldsymbol{A},\boldsymbol{V}-\boldsymbol{A})$使得$\boldsymbol{val}(\boldsymbol{f})=\boldsymbol{cap}(\boldsymbol{A},\boldsymbol{V}-\boldsymbol{A})$

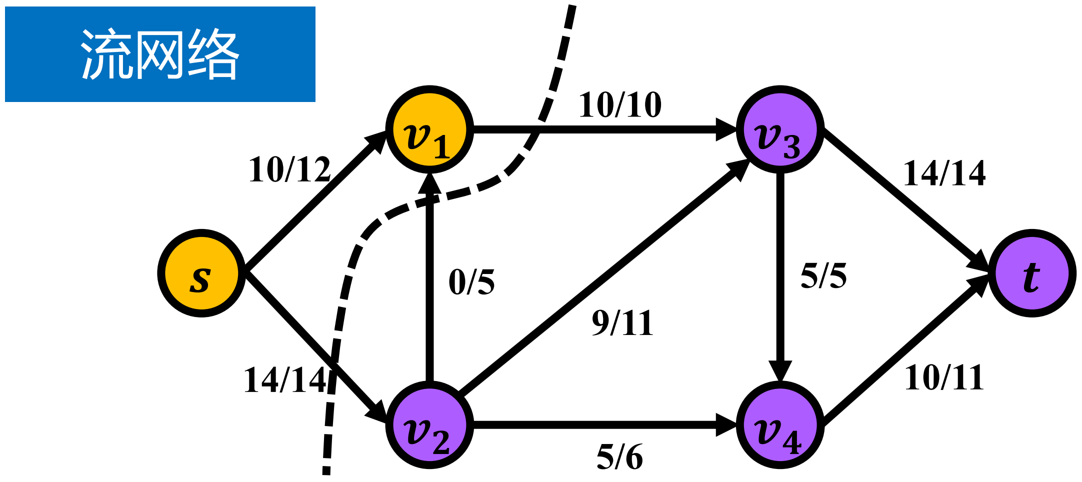
正确性证明

- f是最大流$\Leftrightarrow$残存网络G_f中无增广路径

  步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$ $\Leftarrow$ 残存网络G_f中无增广路径

  - 证明
    - 无增广路径，s只可达部分顶点，依据顶点可达性，划分形成割$(A, V-A)$



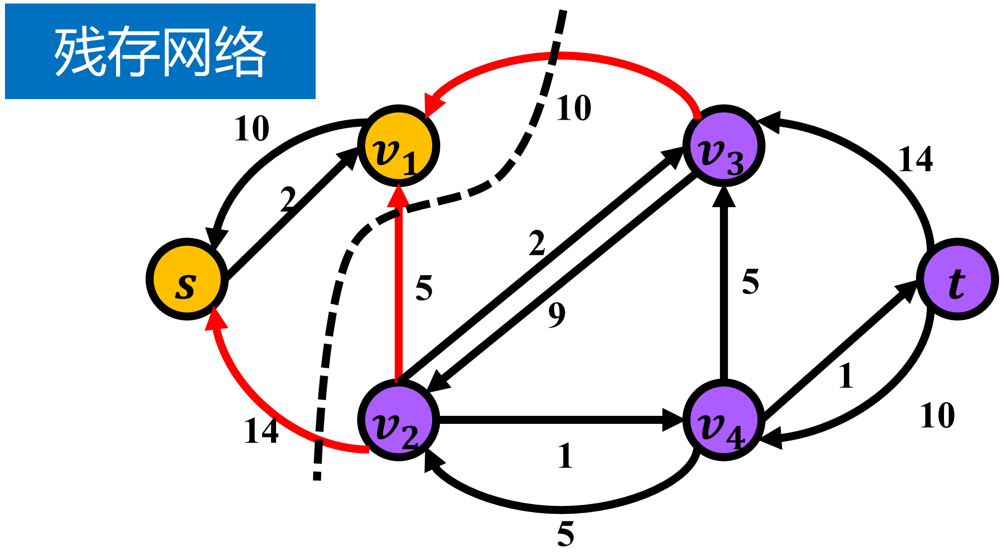


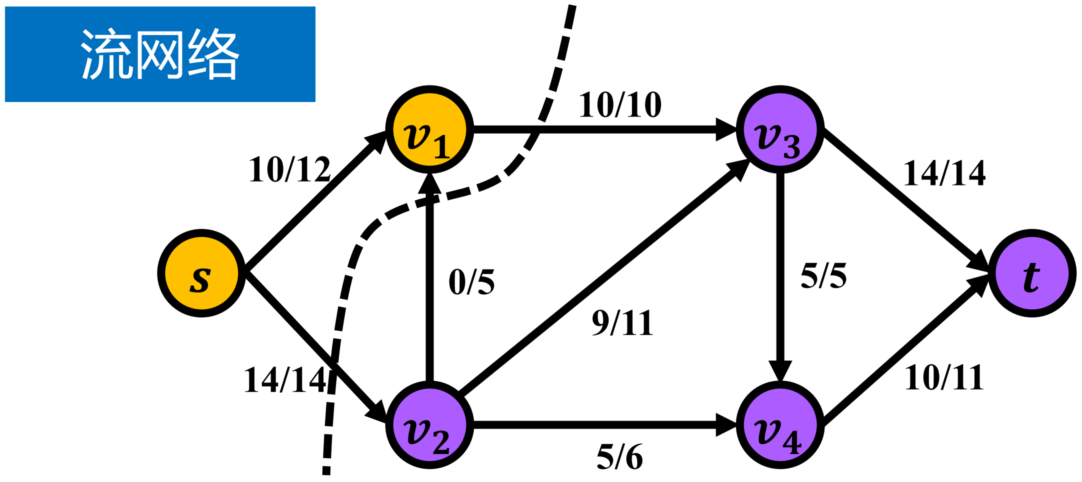
正确性证明

- f是最大流⇔残存网络G_f中无增广路径

  步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$ ⇐ 残存网络G_f中无增广路径

  - 证明
    - 无增广路径，s只可达部分顶点，依据顶点可达性，划分形成割$(A, V-A)$
    - 残存网络中，因为s不可达 $V-A$ 中的点，所以横跨割的残存边一定从$V-A$指向 A



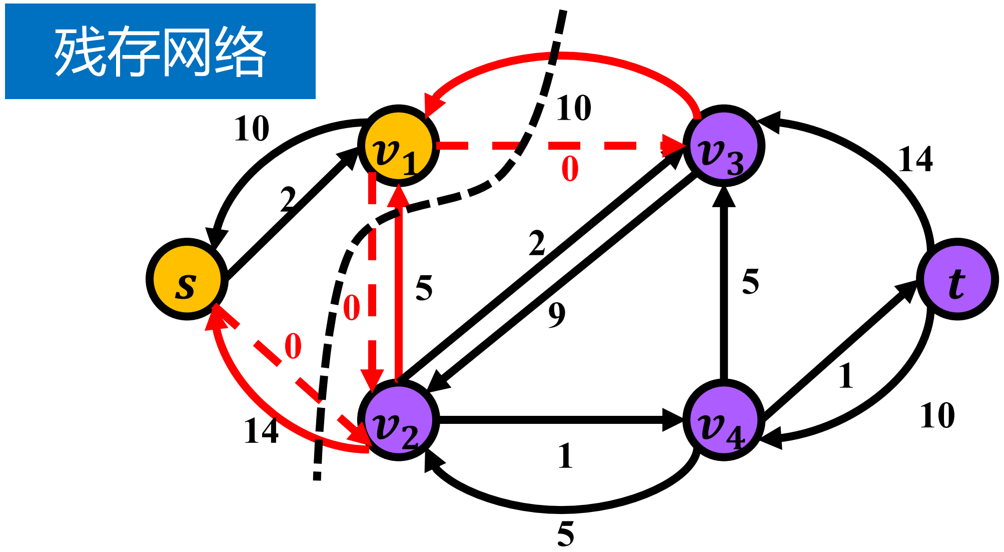


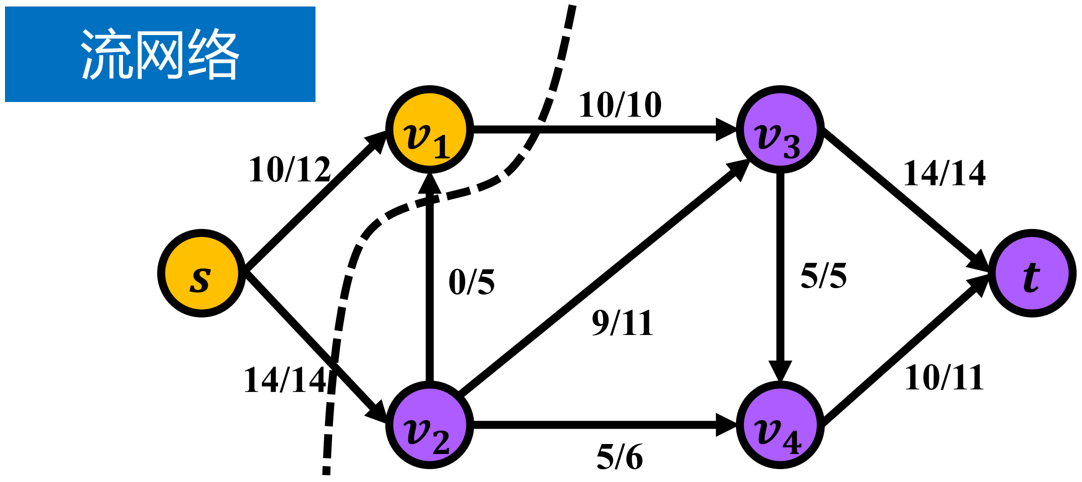
正确性证明

- f是最大流⇔残存网络G_f中无增广路径

  步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$ ⟸ 残存网络G_f中无增广路径

  - 证明
    - 无增广路径，s只可达部分顶点，依据顶点可达性，划分形成割$(A, V-A)$
    - 残存网络中，因为s不可达 $V-A$ 中的点，所以横跨割的残存边一定从$V-A$指向 A
    - 因此，一组横跨割的残存边只剩一条，说明$f(e) = 0$或$f(e) = c(e)$





正确性证明

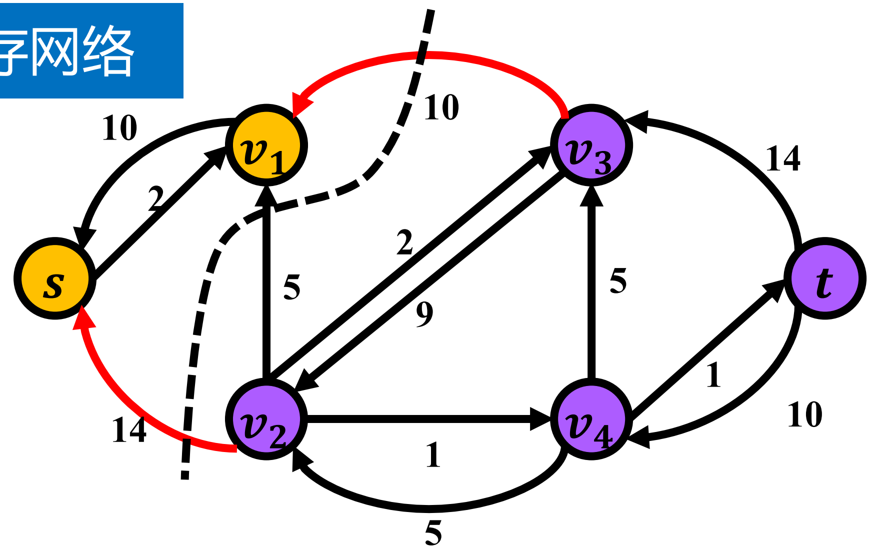
- f是最大流⇔残存网络G_f中无增广路径

  步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$ ⟸ 残存网络G_f中无增广路径

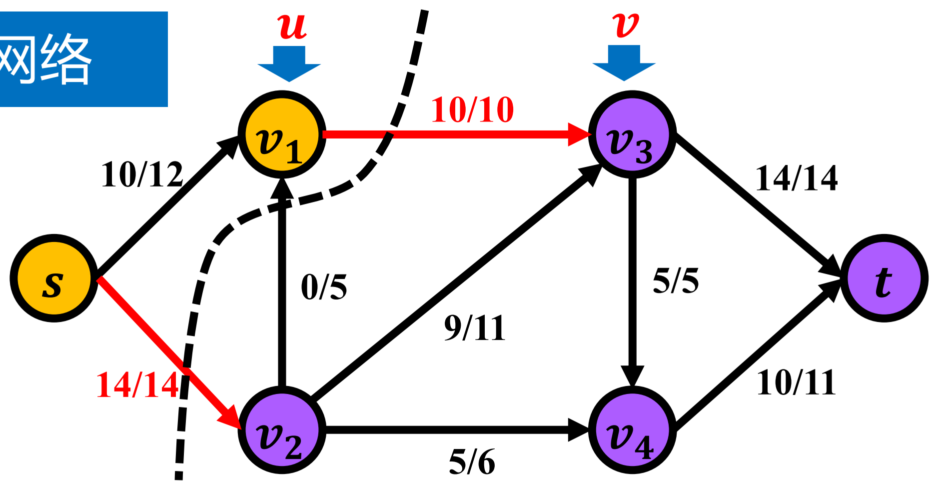
  - 证明
    - 无增广路径，s只可达部分顶点，依据顶点可达性，划分形成割 $(A, V-A)$
    - 残存网络中，因为s不可达 $V-A$ 中的点，所以横跨割的残存边一定从$V-A$指向 A
    - 因此，一组横跨割的残存边只剩一条，说明$f(e) = 0$或$f(e) = c(e)$
    - 流网络中，若$(u, v) \in E$，则必有 $f(u, v) = c(u, v)$，否则s可达顶点v，$v \in V-A$

$u \in A$,
$v \in V-A$

残存网络



流网络



正确性证明

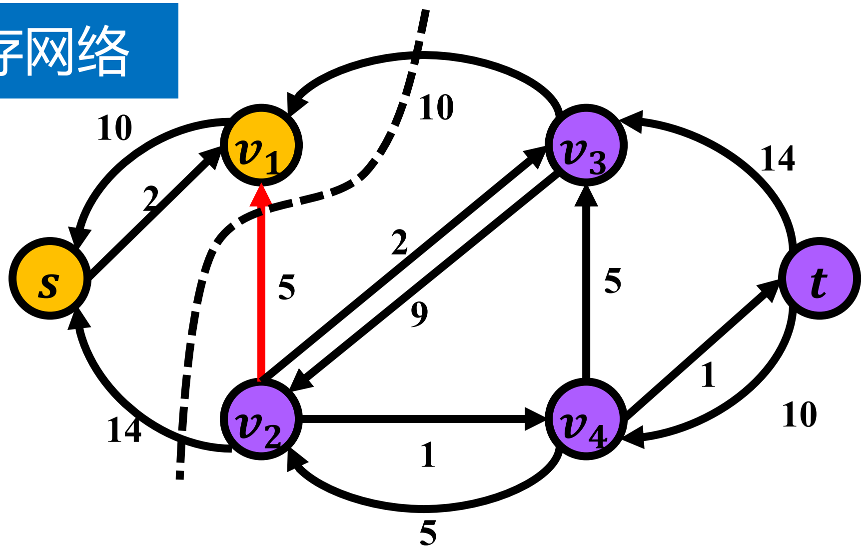
- f是最大流⟺残存网络G_f中无增广路径

  步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A) \Longleftarrow$ 残存网络G_f中无增广路径

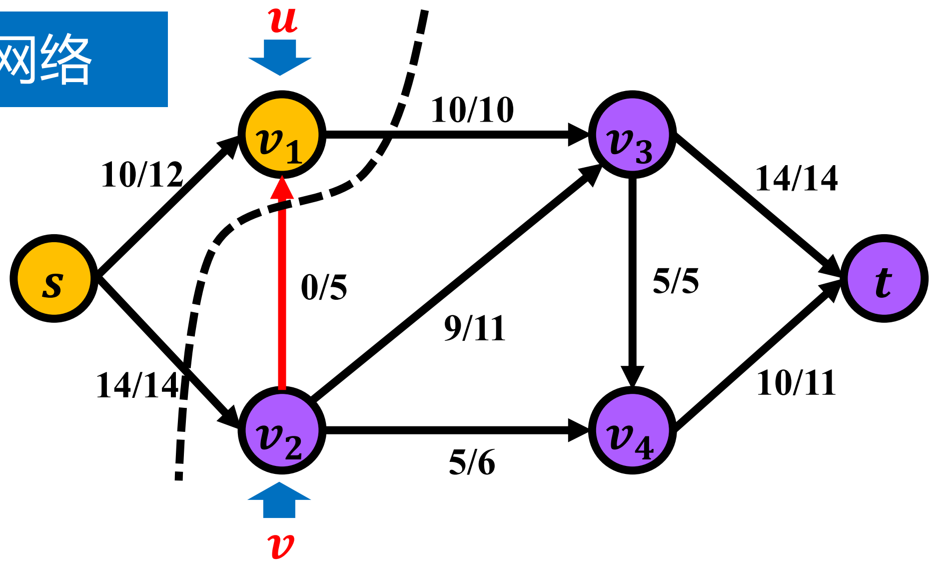
  - 证明
    - 无增广路径，s只可达部分顶点，依据顶点可达性，划分形成割 $(A, V-A)$
    - 残存网络中，因为s不可达 $V-A$ 中的点，所以横跨割的残存边一定从$V-A$指向 A
    - 因此，一组横跨割的残存边只剩一条，说明$f(e)=0$或$f(e)=c(e)$
    - 流网络中，若$(u,v)\in E$，则必有 $f(u,v)=c(u,v)$，否则s可达顶点v，$v\in V-A$
    - 流网络中，若$(v,u)\in E$，则必有$f(v,u)=0$，否则$c_f(u,v)=f(v,u)>0$，顶点$v\in A$

$u \in A$,
$v \in V - A$

残存网络



流网络



正确性证明

- f是最大流⇔残存网络G_f中无增广路径

  步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$ ⟸ 残存网络G_f中无增广路径

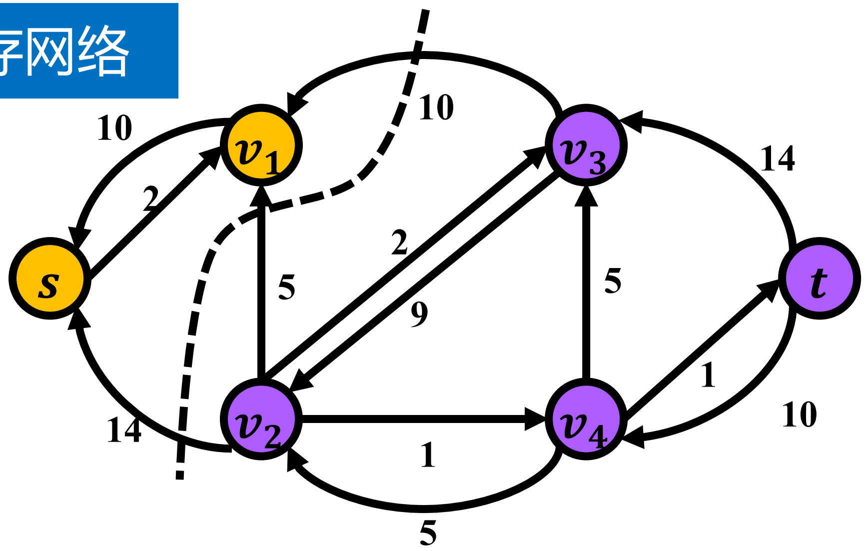
  - 证明
    - 流网络中，若$(u, v) \in E$，则必有 $f(u,v) = c(u,v)$
    - 流网络中，若$(v, u) \in E$，则必有 $f(v,u) = 0$

$u \in A$, $v \in V - A$

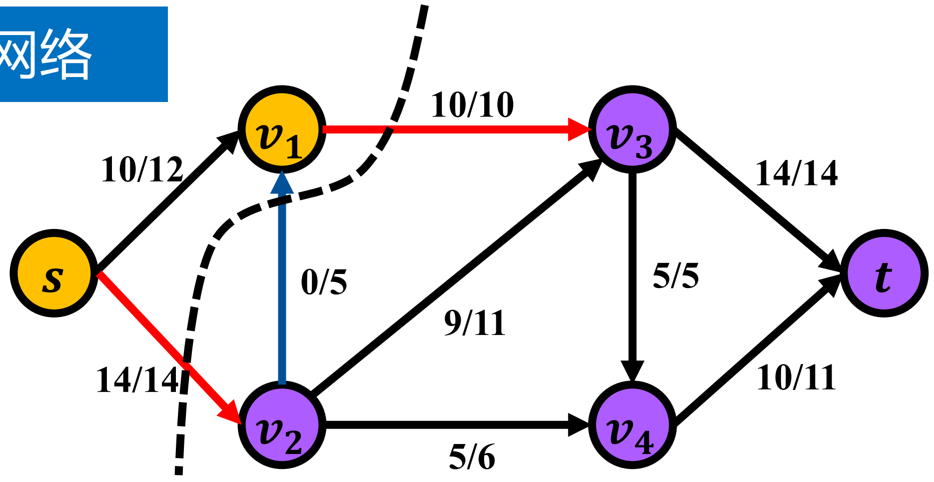
$$val(f) = \sum_{e\ out\ of\ A} f(e) - \sum_{e\ in\ to\ A} f(e)$$

流值定理

残存网络



流网络



正确性证明

- f是最大流⟺残存网络G_f中无增广路径

步骤1：存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$ ⟸ 残存网络G_f中无增广路径

- 证明

$u \in A$,
$v \in V - A$

  - 流网络中，若$(u, v) \in E$，则必有 $f(u, v) = c(u, v)$
  - 流网络中，若$(v, u) \in E$，则必有 $f(v, u) = 0$

$f(e) = 0$

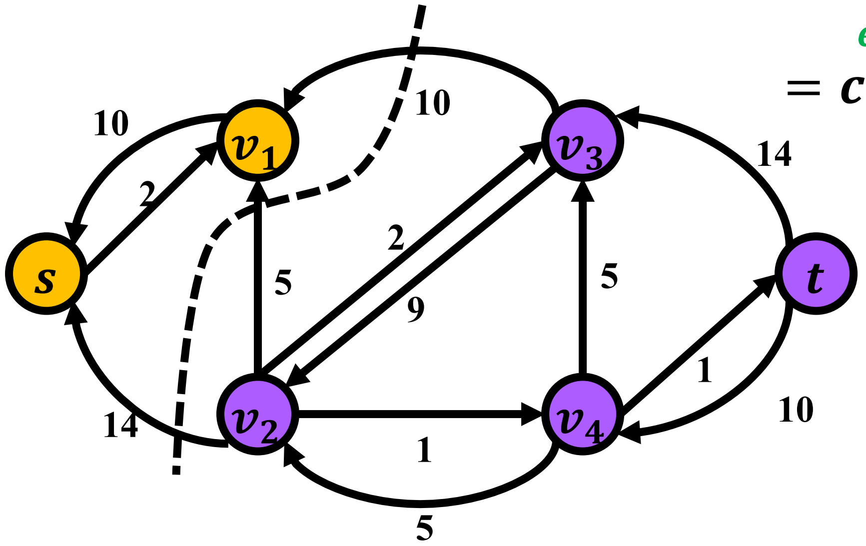
$f(e) = c(e)$

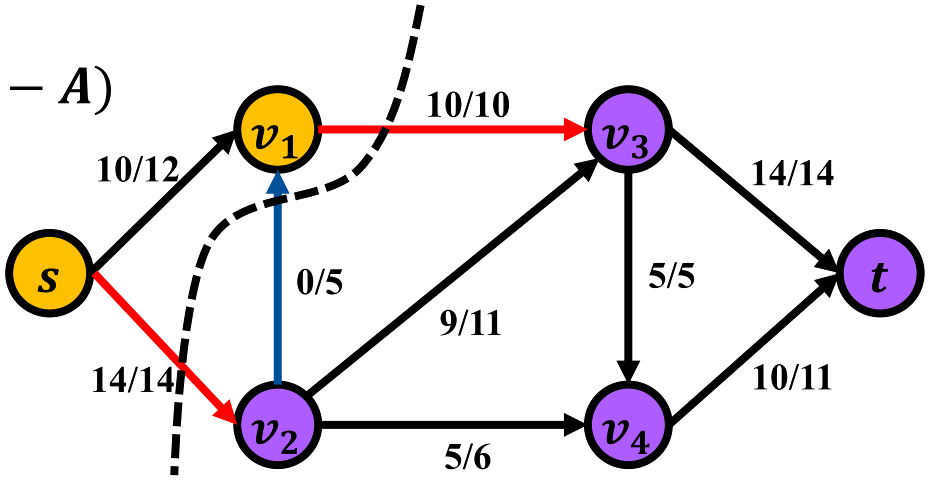
流值定理

$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

$$= \sum_{e \text{ out of } A} c(e)$$

$$= cap(A, V-A)$$





正确性证明

- f是最大流⇔残存网络G_f中无增广路径

  步骤2：f是最大流 $\Leftarrow$ 存在割$(A, V-A)$使得$val(f) = cap(A, V-A)$

  - 弱对偶性：对于任意流f与任意割$(A, V-A)$，$val(f) \leq cap(A, V-A)$
    - 证明：

$$
\begin{aligned}
val(f) &= \sum_{e\ out\ of\ A} f(e) - \sum_{e\ in\ to\ A} f(e) \\
&\leq \sum_{e\ out\ of\ A} c(e) \\
&= cap(A, V-A)
\end{aligned}
$$

流值定理

每条边流量必小于容量

弱对偶性(Weak Duality)：最大流的流值小于等于任意割容量

正确性证明

- $\boldsymbol{f}$是最大流⇔残存网络$\boldsymbol{G_f}$中无增广路径

  步骤2：$\boldsymbol{f}$是最大流 $\Leftarrow$ 存在割$(\boldsymbol{A}, \boldsymbol{V}-\boldsymbol{A})$使得$\boldsymbol{val}(\boldsymbol{f}) = \boldsymbol{cap}(\boldsymbol{A}, \boldsymbol{V}-\boldsymbol{A})$

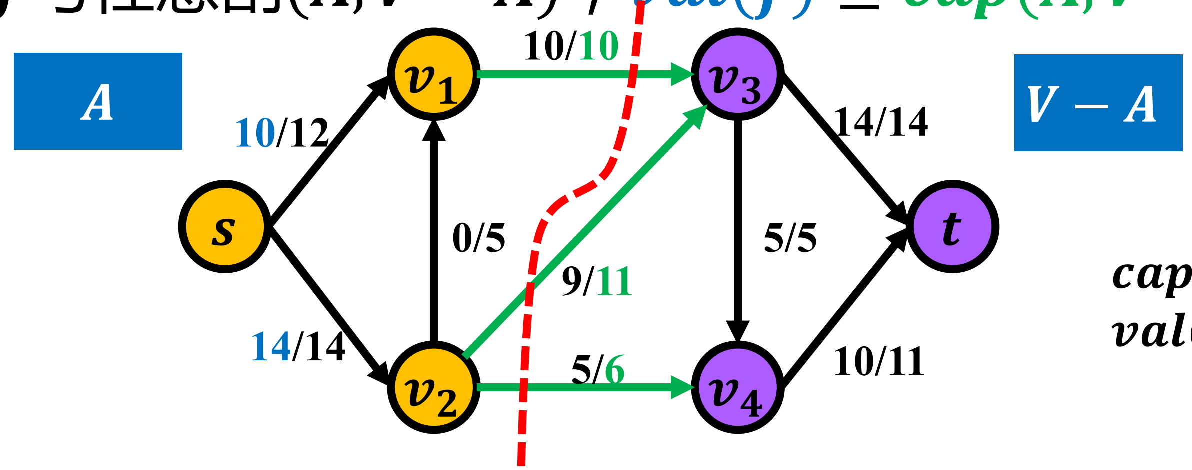
  - 强对偶性：最大流可取到其上界（最小割）
    - 证明：
      - 设割$(\boldsymbol{A}, \boldsymbol{V}-\boldsymbol{A})$使得$\boldsymbol{cap}(\boldsymbol{A}, \boldsymbol{V}-\boldsymbol{A}) = \boldsymbol{val}(\boldsymbol{f})$
      - 根据弱对偶性，对于任意流$\boldsymbol{f}'$，有$\boldsymbol{val}(\boldsymbol{f}) = \boldsymbol{cap}(\boldsymbol{A}, \boldsymbol{V}-\boldsymbol{A}) \geq \boldsymbol{val}(\boldsymbol{f}')$
      - 流$\boldsymbol{f}$是最大流

弱对偶性

强对偶性(**Strong Duality**)：最大流的流值等于最小割的容量

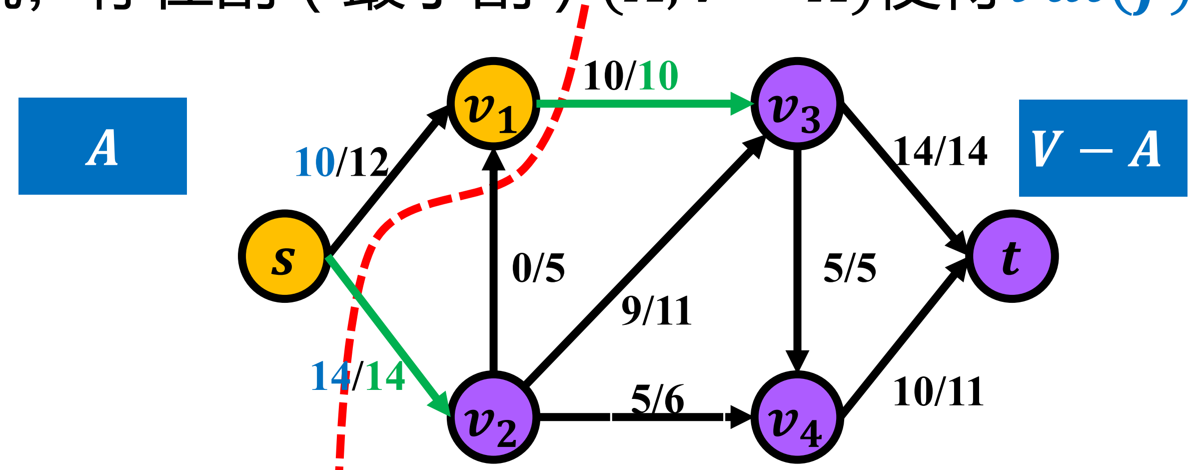
对偶性

- 弱对偶性(Weak Duality)
  - 对于任意流f与任意割$(A, V-A)$，$val(f) \le cap(A, V-A)$



$cap(S, V-S)$=10+11+6=27
$val(f)$=10+14=24

- 强对偶性(Strong Duality)
  - f是最大流，存在割（最小割）$(A, V-A)$使得$val(f) = cap(A, V-A)$



$cap(S, V-S)$=10+14=24
$val(f)$=10+14=24

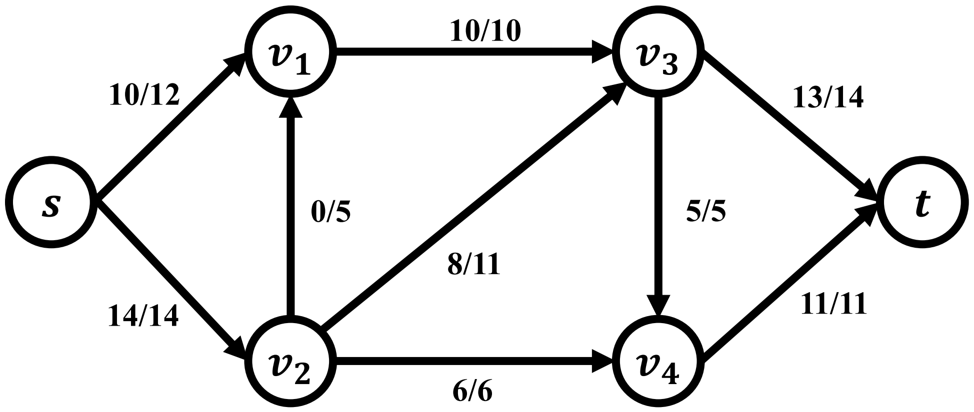
正确性证明

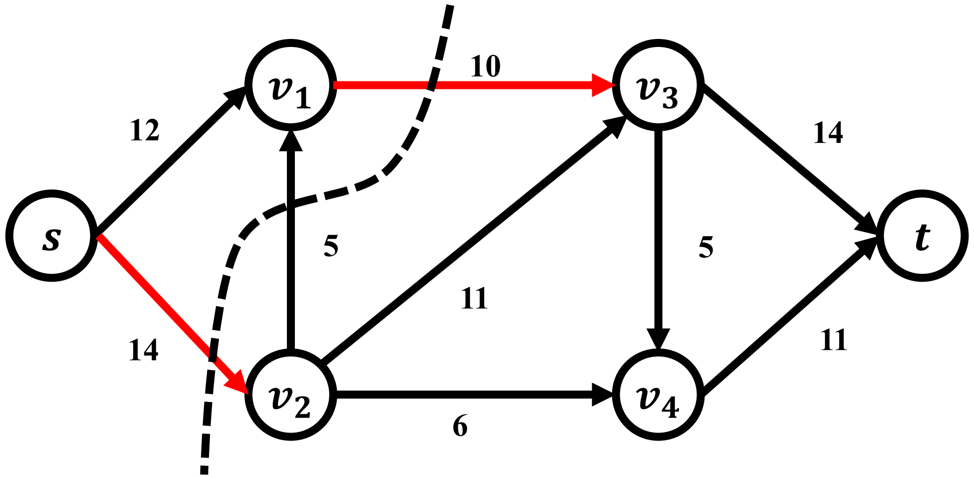
- 最大流
  - f是最大流⟺残存网络G_f中无增广路径
    - 充分性：f是最大流⟹残存网络G_f中无增广路径
    - 必要性：f是最大流⟸存在割是最大流的上界⟸残存网络G_f中无增广路径

回顾以上证明过程，得到最大流最小割定理

最大流最小割定理

- f为流网络$G=<V,E,C>$中一个流，该流网络源点为s，汇点为t，以下三条件相互等价
  - (i) 存在割$(A,V-A)$使得$val(f)=cap(A,V-A)$
  - (ii) 流f是最大流
  - (iii) 流f没有增广路径





最大流

$|f^*|=24$

最小割

$cap(A,V-A)=24$

二分图最大匹配：问题背景

- 共享出行与人们生活息息相关

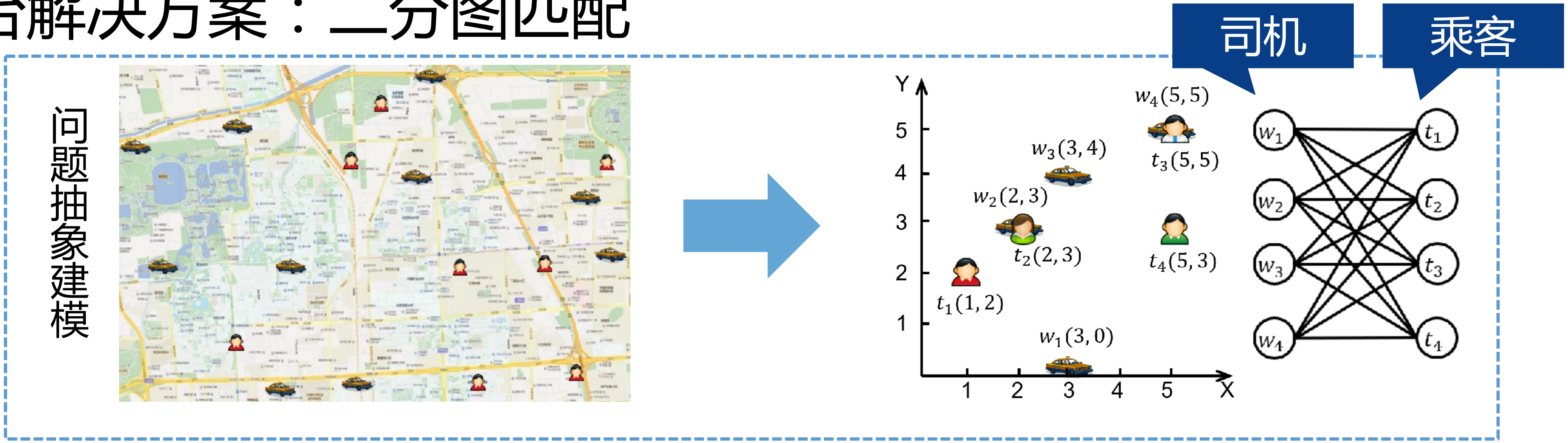
高峰期打车难

社会关注热点

问题：网约车平台如何解决打车难问题？

问题背景

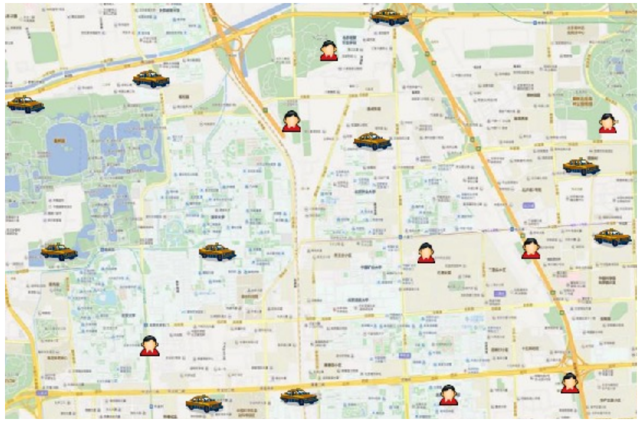
- 平台解决方案：二分图匹配

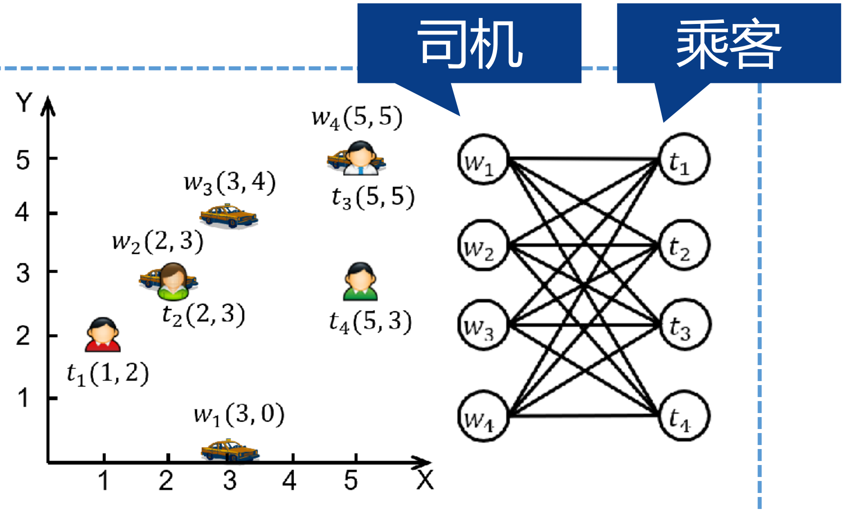


问题背景

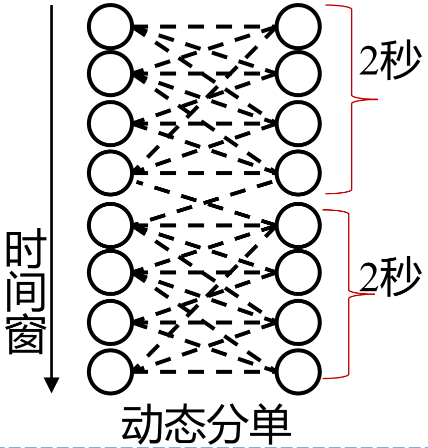
- 平台解决方案：二分图匹配

问题抽象建模





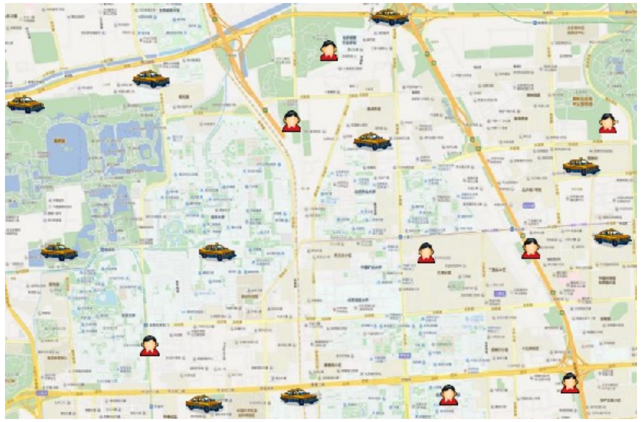
平台解决方案

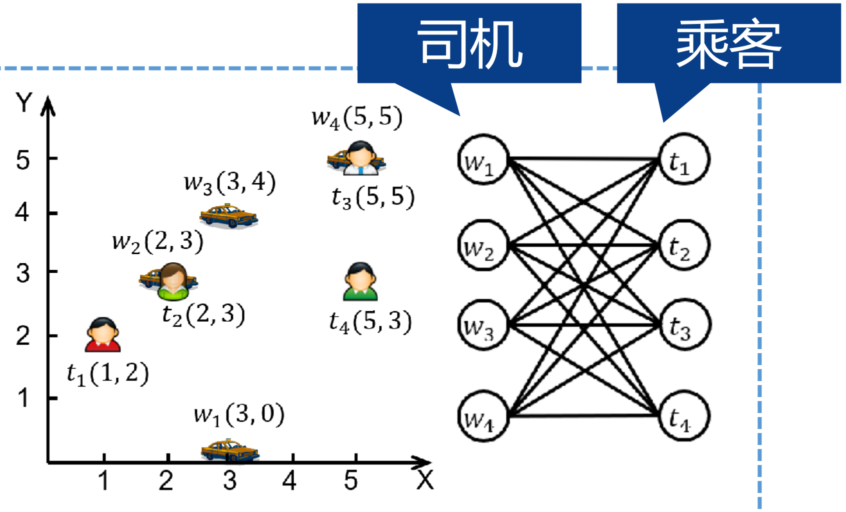


问题背景

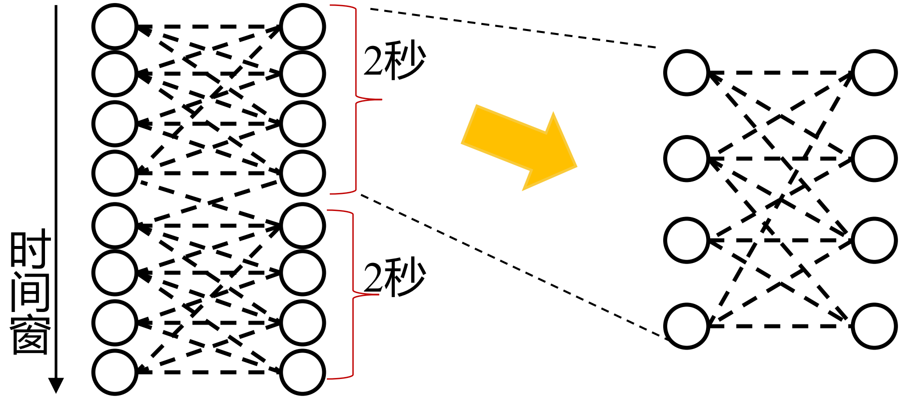
- 平台解决方案：二分图匹配

问题抽象建模





平台解决方案

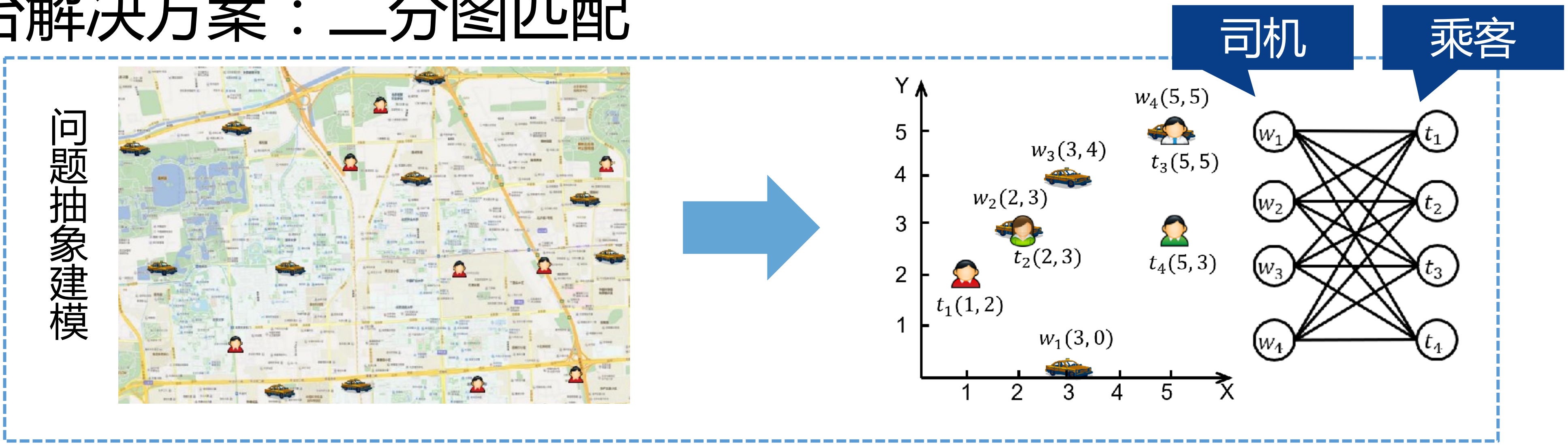


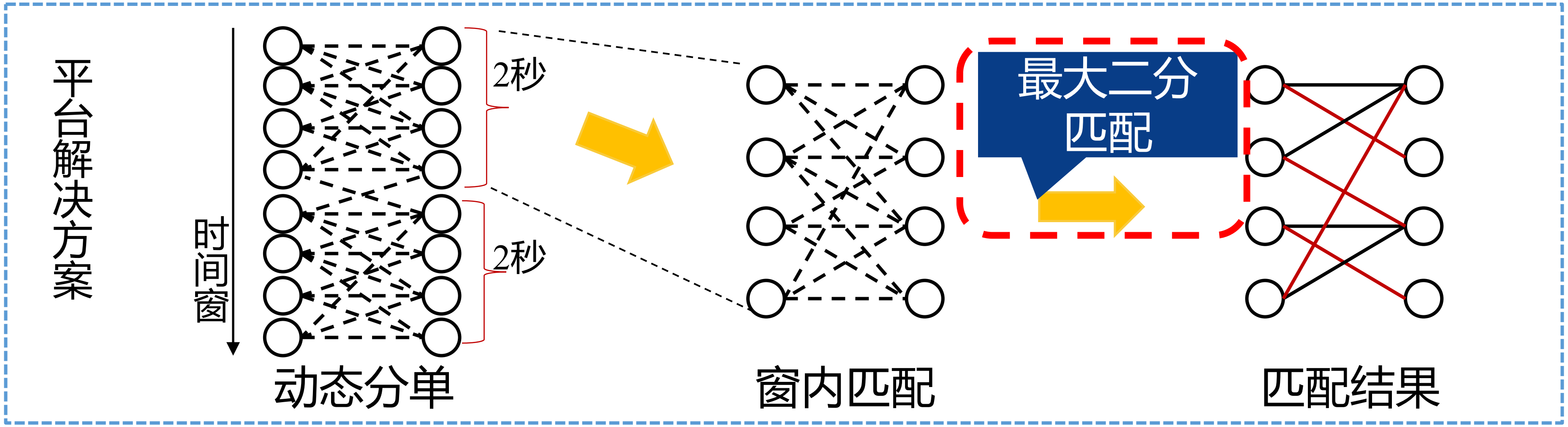
动态分单

窗内匹配

问题背景

- 平台解决方案：二分图匹配

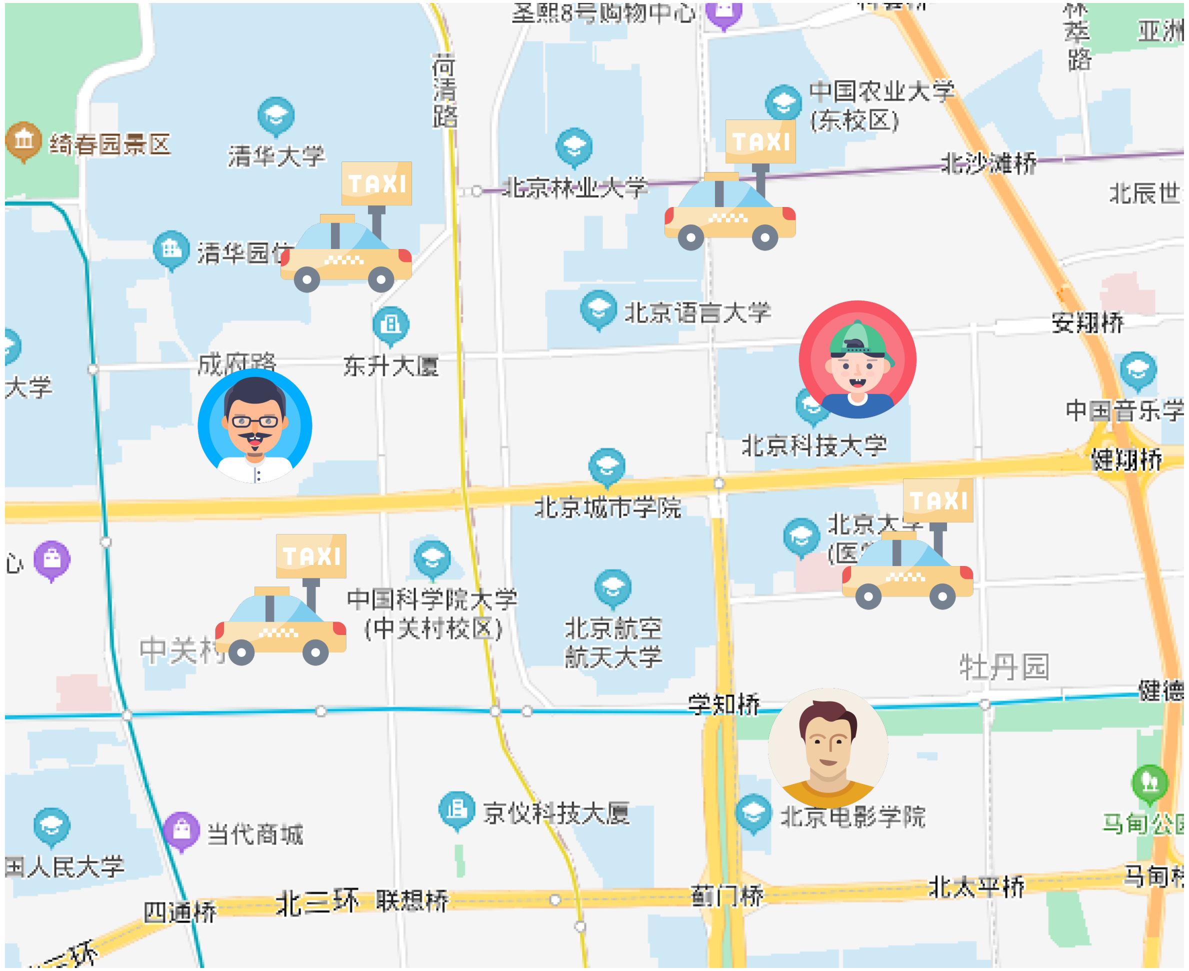


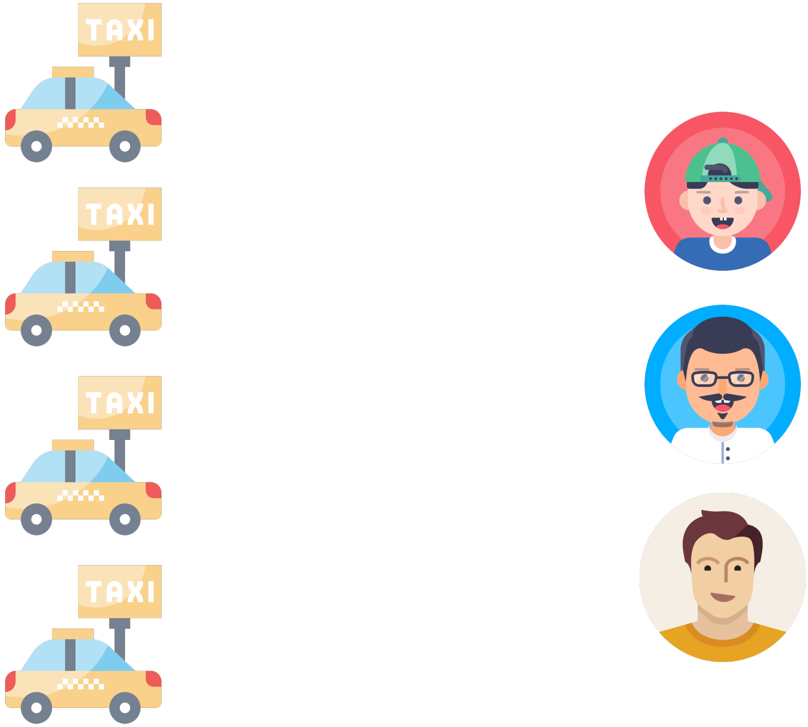


问题：如何最大化匹配数目？

问题背景

- 关键问题：最大化匹配数目

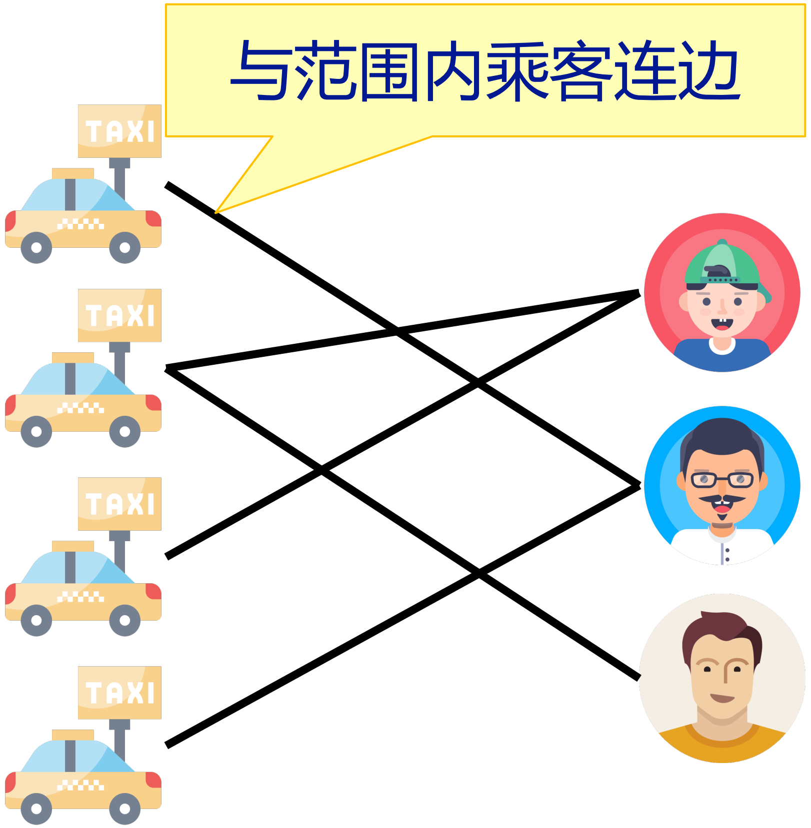




司机乘客视为顶点

问题背景

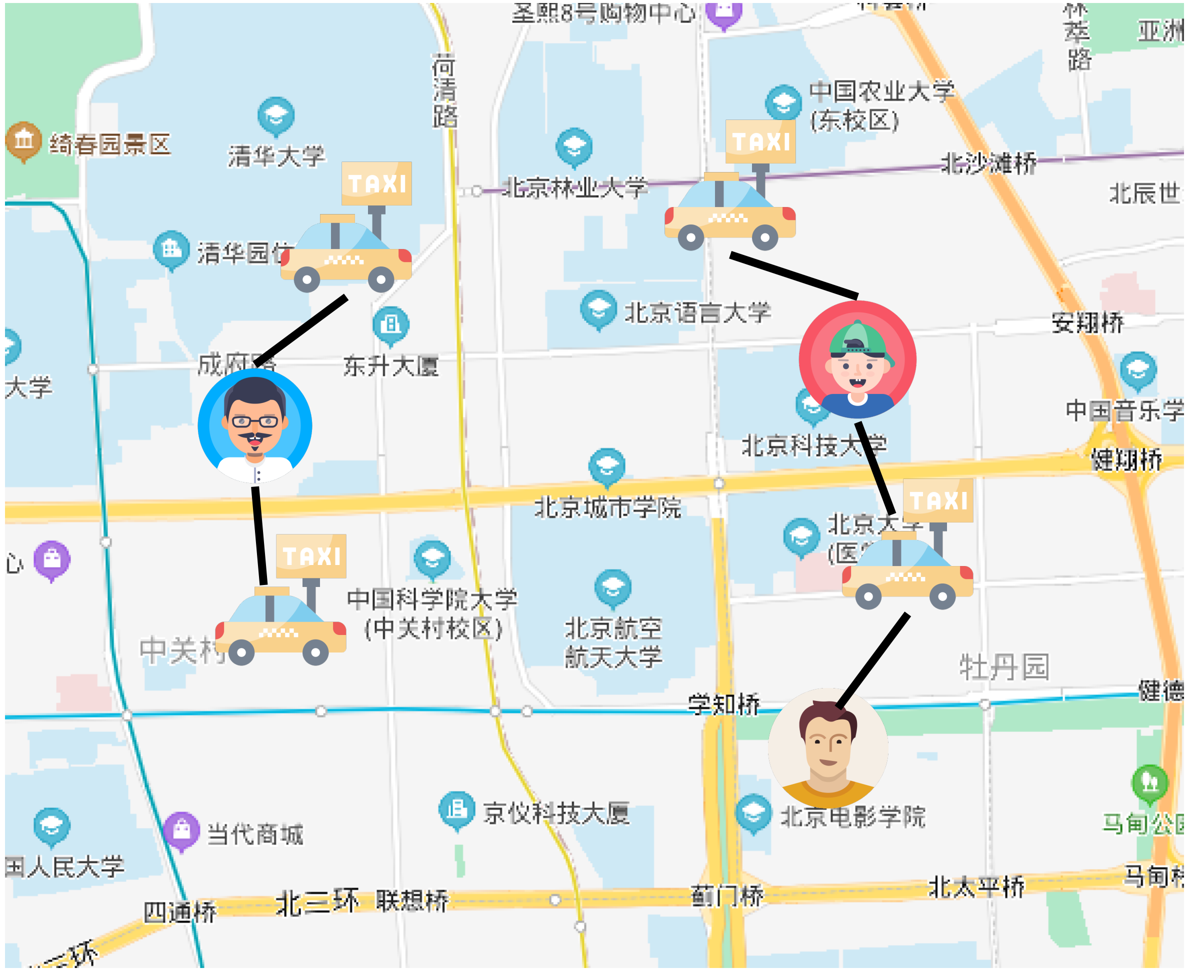
- 关键问题：最大化匹配数目

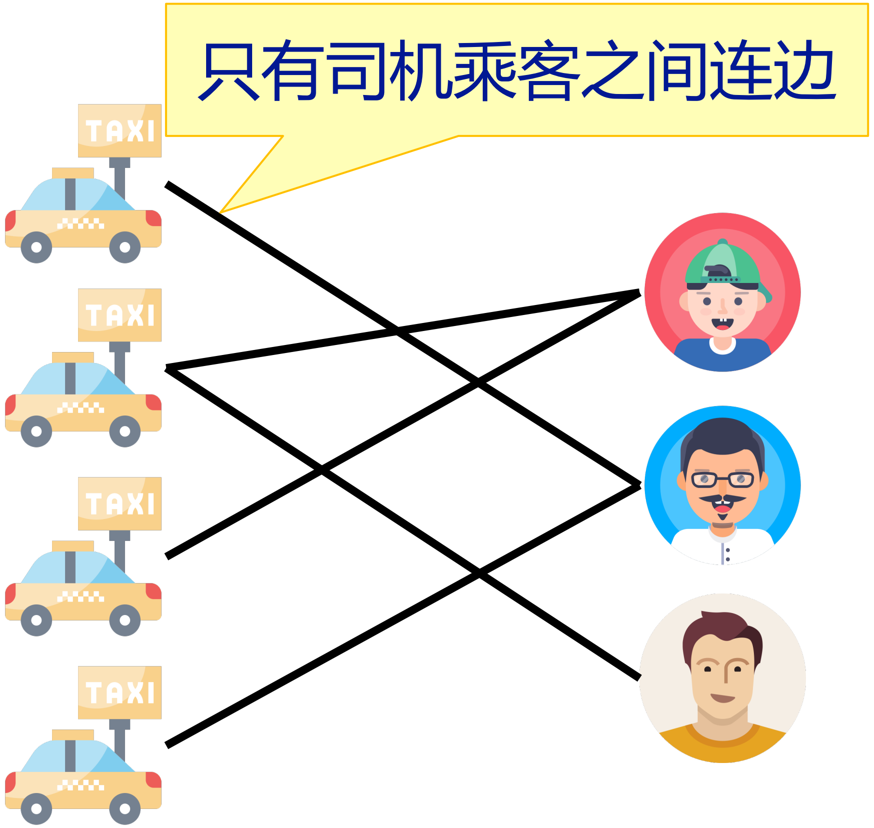


司机服务一定范围内的乘客

问题背景

- 关键问题：最大化匹配数目

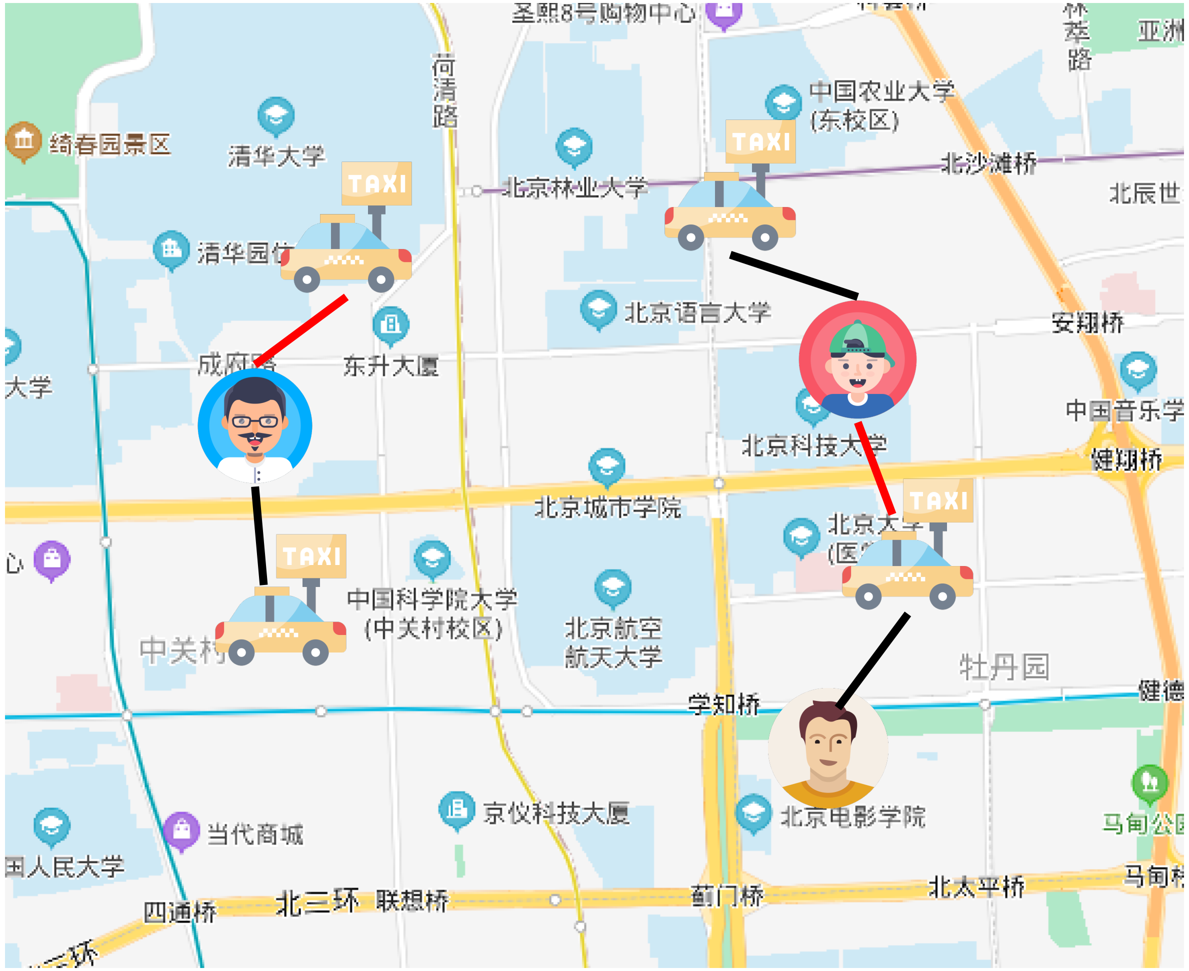




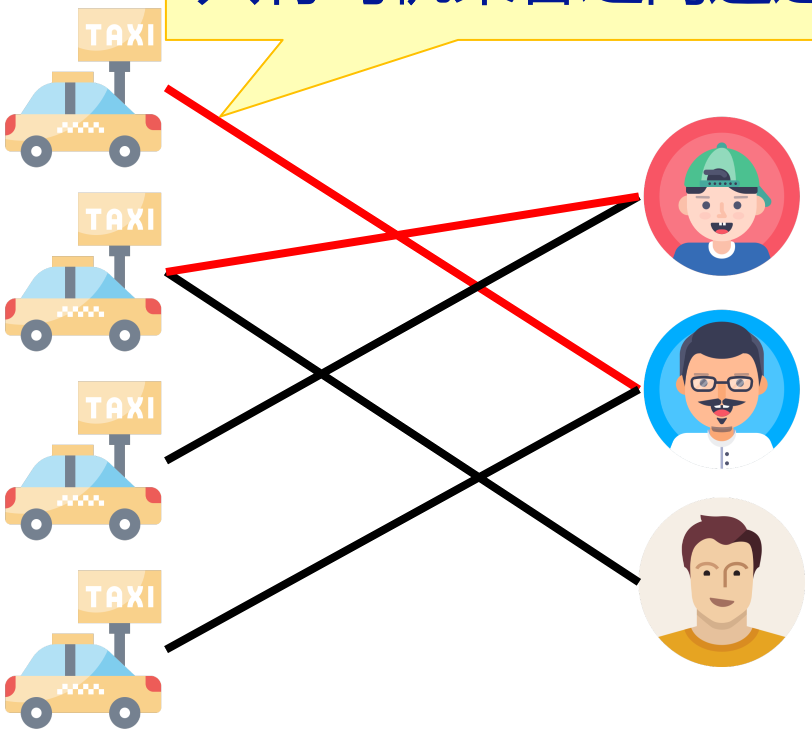
特殊的图：二分图

问题背景

- 关键问题：最大化匹配数目



只有司机乘客之间连边



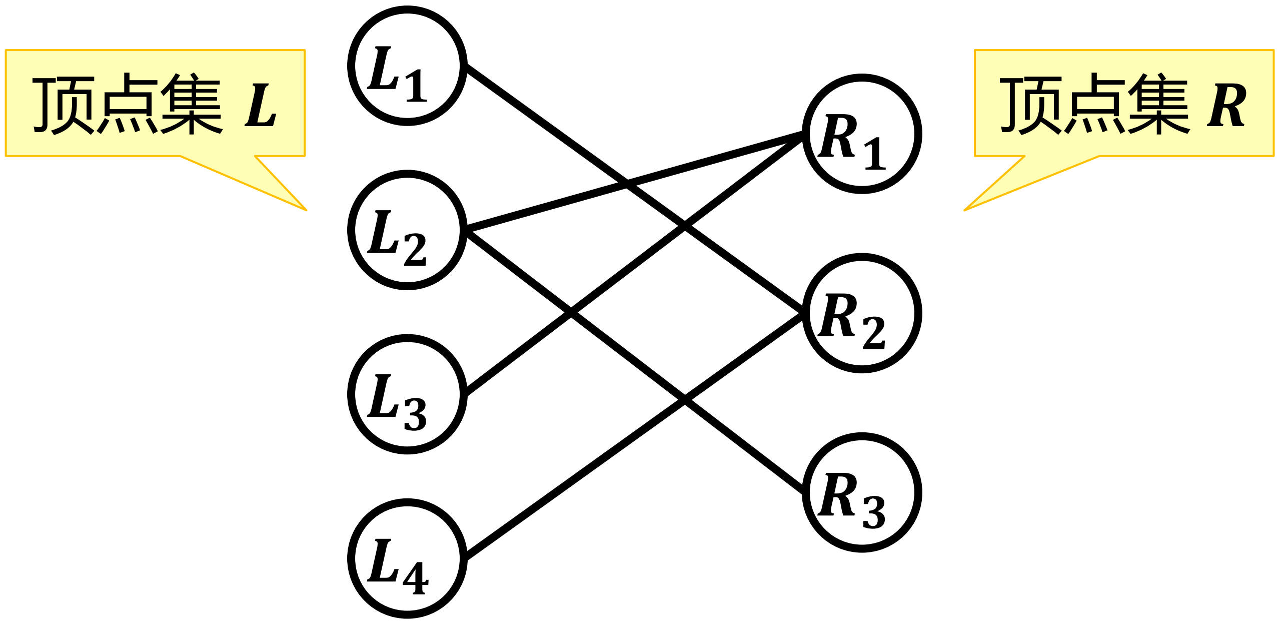
特殊的图：二分图

分配结果：匹配

相关概念

- 二分图(Bipartite Graph)：给定一个无向图 $\boldsymbol{G} =< \boldsymbol{V}, \boldsymbol{E} >$，其中
  - $\boldsymbol{V} = \boldsymbol{L} \cup \boldsymbol{R}, \boldsymbol{L} \cap \mathbf{R} = \emptyset$，并且
  - 每条边 $\boldsymbol{e} \in \boldsymbol{E}$ 有一个端点在 $\boldsymbol{L}$ 中而另一个端点在 $\boldsymbol{R}$ 中，

  其可记为二分图$\boldsymbol{G} =< \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{E} >$

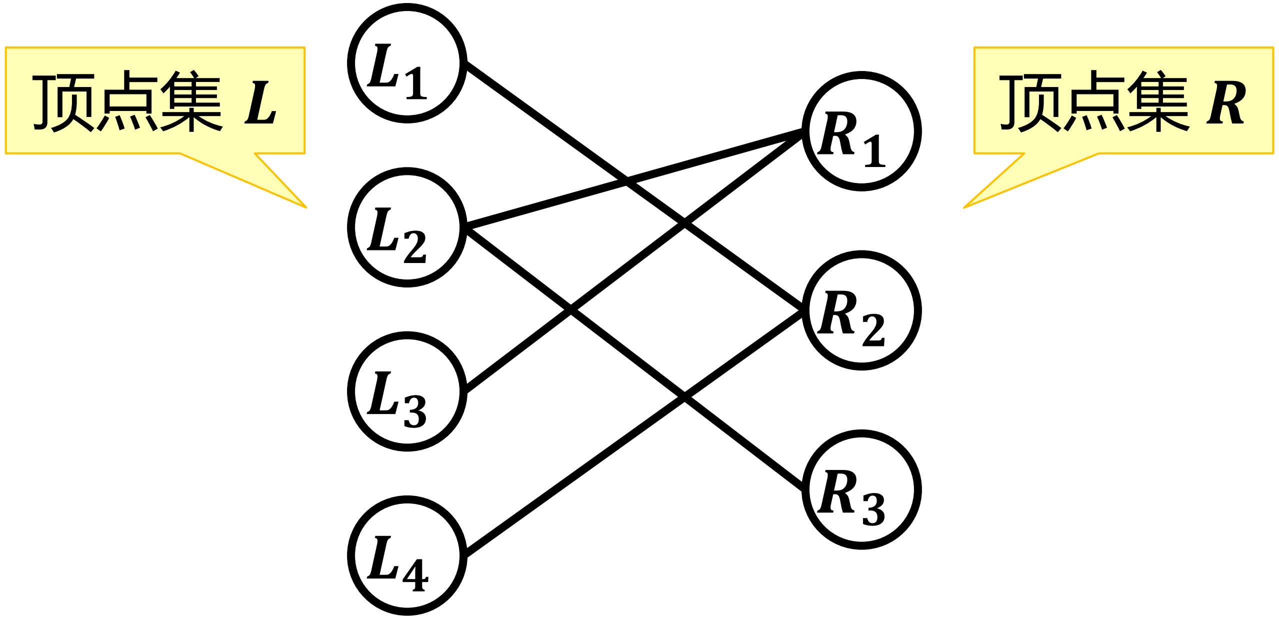


二分图

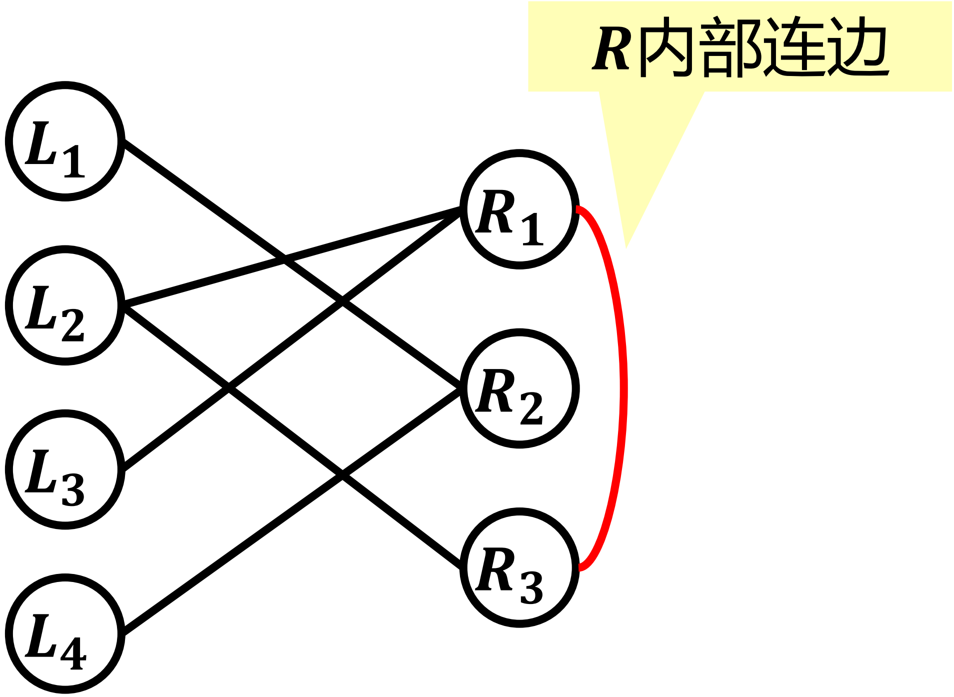
相关概念

- 二分图(Bipartite Graph)： 给定一个无向图 $G=<V,E>$，其中
  - $V=L\cup R, L\cap \mathrm{R}=\emptyset$，并且
  - 每条边 $e\in E$ 有一个端点在 L 中而另一个端点在 R 中，

其可记为二分图$G=<L,R,E>$



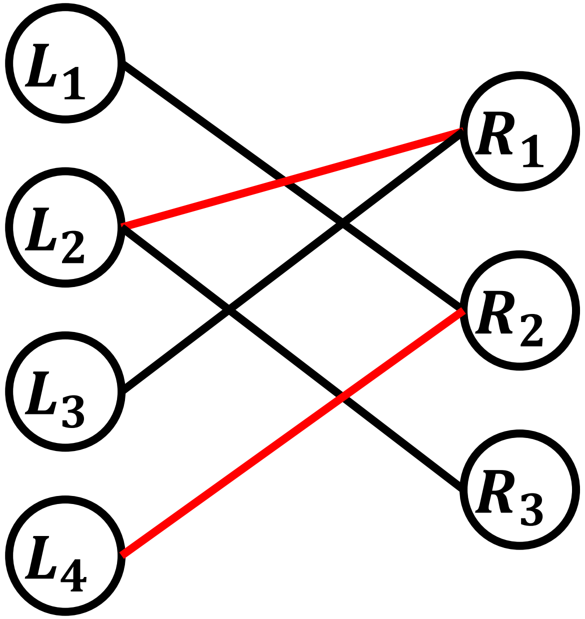
二分图



非二分图

相关概念

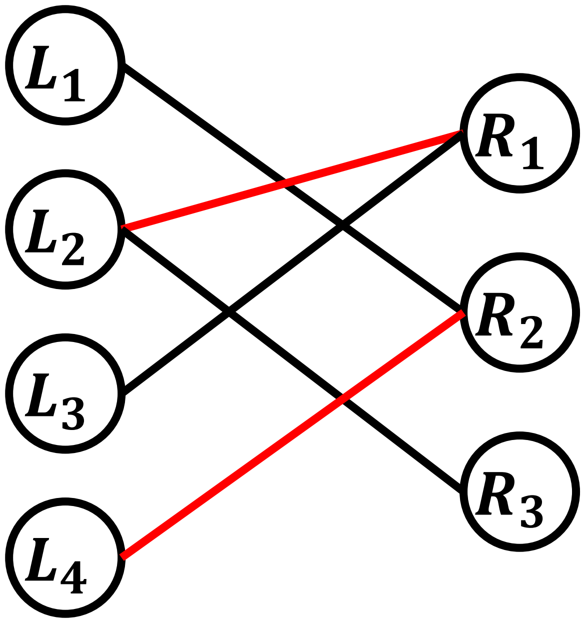
- 匹配(Matching)
  - 给定一个无向图 $G=<V,E>$和边集 E 的子集 S，如果对于 G 中任意一个节点 $v\in V$，S至多有一条边与 v 关联，则称 S 为一个匹配。



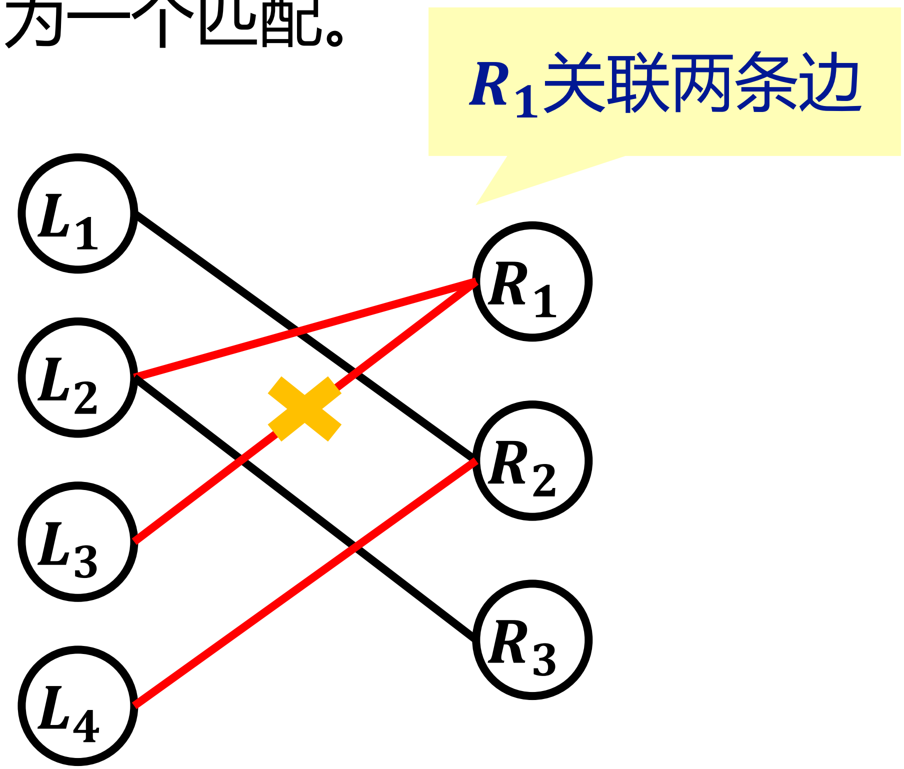
匹配

相关概念

- 匹配(Matching)
  - 给定一个无向图 $G=<V,E>$和边集 E 的子集 S，如果对于 G 中任意一个节点 $v\in V$，S至多有一条边与 v 关联，则称 S 为一个匹配。



匹配



非匹配

问题定义

最大二分匹配问题

Maximum Bipartite Matching Problem

输入

- 二分图$\boldsymbol{G} =< \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{E} >$

问题定义

最大二分匹配问题

Maximum Bipartite Matching Problem

输入

- 二分图$\boldsymbol{G} =< \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{E} >$

输出

- 求出匹配 $\boldsymbol{M} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \dots \boldsymbol{e}_k\}$ ，令

$$\max |\boldsymbol{M}|$$

满足$\forall \boldsymbol{i}, \boldsymbol{j} (\boldsymbol{i} \neq \boldsymbol{j}), \boldsymbol{e}_i = (\boldsymbol{l}_i, \boldsymbol{r}_i), \boldsymbol{e}_j = (\boldsymbol{l}_j, \boldsymbol{r}_j)$, 有$\boldsymbol{l}_i \neq \boldsymbol{l}_j$且$\boldsymbol{r}_i \neq \boldsymbol{r}_j$

问题定义

最大二分匹配问题

Maximum Bipartite Matching Problem

输入

- 二分图$\boldsymbol{G} =< \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{E} >$

输出

- 求出匹配 $\boldsymbol{M} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \dots \boldsymbol{e}_k\}$，令

$$\max |\boldsymbol{M}|$$

满足$\forall \boldsymbol{i}, \boldsymbol{j}(\boldsymbol{i} \neq \boldsymbol{j}), \boldsymbol{e}_i = (\boldsymbol{l}_i, \boldsymbol{r}_i), \boldsymbol{e}_j = (\boldsymbol{l}_j, \boldsymbol{r}_j)$，有$\boldsymbol{l}_i \neq \boldsymbol{l}_j$且$\boldsymbol{r}_i \neq \boldsymbol{r}_j$

优化目标：最大匹配数量

约束条件：每个顶点至多关联一条边

问题定义

最大二分匹配问题

Maximum Bipartite Matching Problem

输入

- 二分图$G =< L, R, E >$

输出

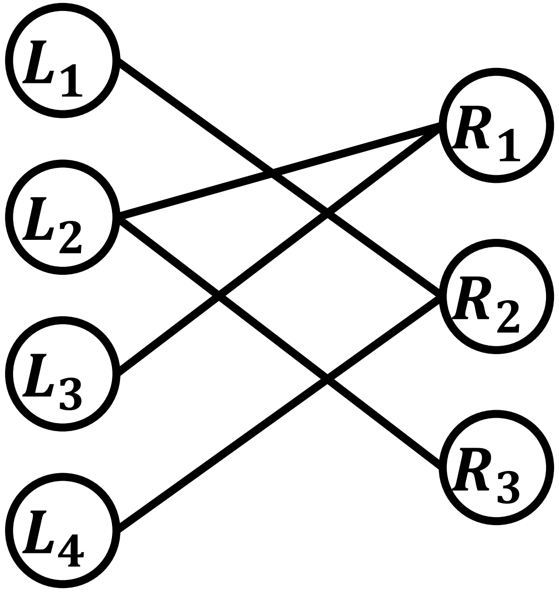
- 求出匹配 $M = \{e_1, e_2, \dots e_k\}$，令

$$\max |M|$$

优化目标：最大匹配数量

满足$\forall i, j (i \neq j), e_i = (l_i, r_i), e_j = (l_j, r_j)$，有$l_i \neq l_j$且$r_i \neq r_j$

约束条件：每个顶点至多关联一条边



问题定义

最大二分匹配问题

Maximum Bipartite Matching Problem

输入

- 二分图$\boldsymbol{G} =< \boldsymbol{L}, \boldsymbol{R}, \boldsymbol{E} >$

输出

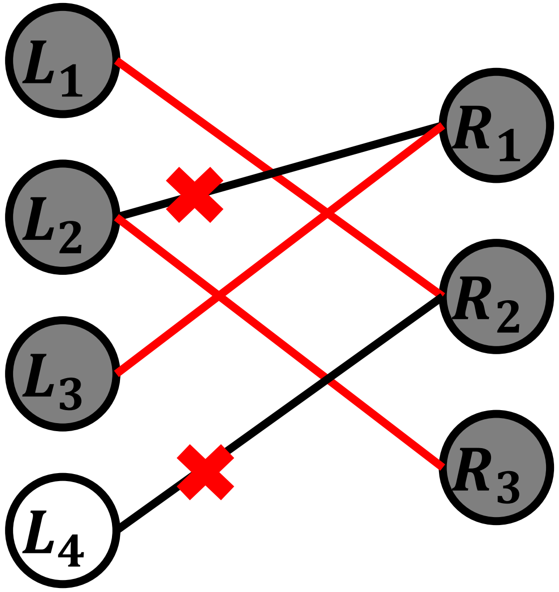
- 求出匹配 $\boldsymbol{M} = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \boldsymbol{e}_k\}$ ，令

$$\max |\boldsymbol{M}|$$

优化目标：最大匹配数量

满足$\forall \boldsymbol{i}, \boldsymbol{j} (\boldsymbol{i} \neq \boldsymbol{j}), \boldsymbol{e}_i = (\boldsymbol{l}_i, \boldsymbol{r}_i), \boldsymbol{e}_j = (\boldsymbol{l}_j, \boldsymbol{r}_j)$,有$\boldsymbol{l}_i \neq \boldsymbol{l}_j$且$\boldsymbol{r}_i \neq \boldsymbol{r}_j$

约束条件：每个顶点至多关联一条边



小结

- **Ford-Fulkerson算法可求解最大二分匹配问题**
  - 求解思想：将二分图转化为流网络