

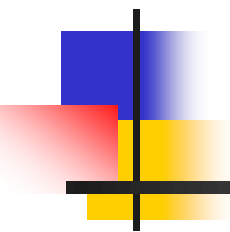


# 算法设计与分析

Computer Algorithm Design & Analysis

文明

[mwenaa@hust.edu.cn](mailto:mwenaa@hust.edu.cn)



## 第十三讲

---

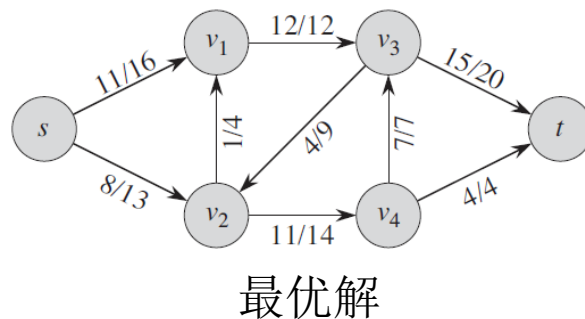
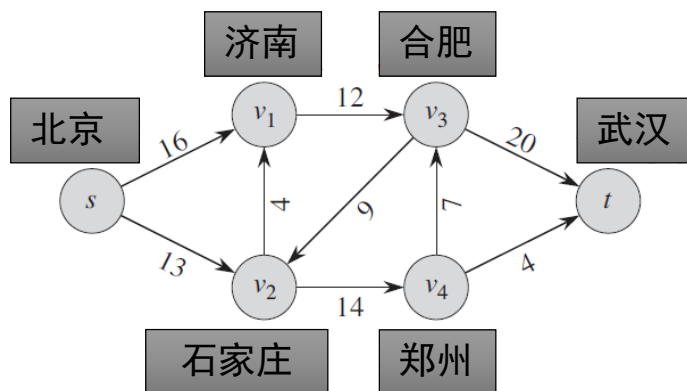
# Maximum Flow

## 最大流

# 最大流问题由 T. E. Harris and F. S. Ross 于1954年提出



## 问题的引出：实例**物流网络**



在物流网络中，从一个城市（称为**源结点**）发送一批货物到另一个城市（称为**汇点**）。假设源结点可以源源不断地提供货物，汇点可以来者不拒地接收货物；路径连接在任意两个城市之间，但路径上有**运输容量有限制**。货物从源结点到汇点可以选择不同的运输路径。

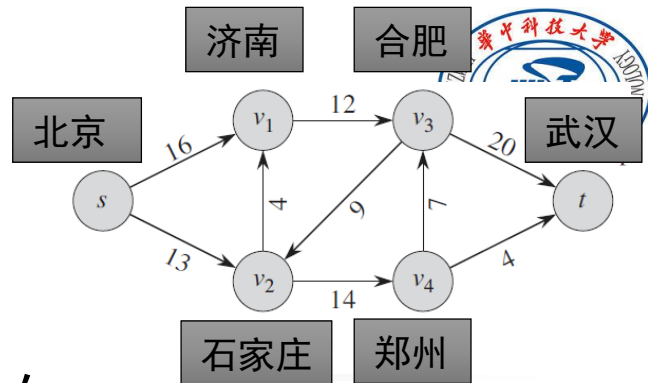
问：在不违反任何路径容量限制的条件下，从源结点到汇点运送货物的最大速率是多少——这一问题的抽象称为**最大流问题**。

## 用带权有向图来表示:

□ **结点**表示城市

□ 结点间的**有向边**表示运输路径和物流的方向

□ 边上的**权重**表示运量限制



—— 这种用来表示 “**流**” 的图称为 “**流网络**” 。

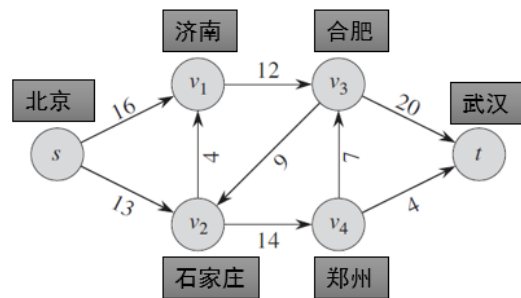
### 重要的约定:

- (1) **流量守恒**: 除源结点和汇点外, 其它结点上物料只是“流过”, 即物料进入的速率等于离开的速率, 不积累和聚集;
- (2) 物料的生成速率和接收速率恒定且足够快、足够多, 满足需要;
- (3) 每条边上的容量是物料通过该边的最大速率, 不能突破。

## 26.1 流网络

**流网络**是一个有向图  $G = (V, E)$ ，边上定义有容量函数： $c: E \rightarrow R^+ \cup \{0\}$

- (1) 有一个**源节点** $s$ 和**汇点** $t$ ;
- (2) 有向边表示流向;
- (3) 每条边  $(u, v) \in E$  上有一个非负的**容量值**  $c(u, v) \geq 0$ ;  
如果  $(u, v) \notin E$ ，为方便起见，定义  $c(u, v) = 0$ ;
- (4) 如果边集合  $E$  中包含边  $(u, v)$ ，则图中不包含其反向边  $(v, u)$ ;
- (5) 图中不允许有自循环;
- (6) **流网络是连通图**，每个结点都在从  $s$  到  $t$  的某条路径上;
- (7) 除源结点外，每个结点至少有一条流入的边;
- (8) 除汇点外，每个结点至少有一条流出的边;
- (9)  $|E| \geq |V| - 1$



设  $G = (V, E)$  是一个流网络，其容量函数为  $c: E \rightarrow R^+ \cup \{0\}$ 。

设  $s$  为源结点， $t$  为汇点。

**流** 是定义在  $G$  上的一个实值函数，记为  $f: V \times V \rightarrow R$ ，并满足以下两条性质：

(1) **容量限制**：对于所有的结点  $u, v \in V$ ，有

$$0 \leq f(u, v) \leq c(u, v)$$

(2) **流量守恒**：对于所有结点  $u \in V - \{s, t\}$ ，有

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

这里， $f(u, v)$  称为从结点  $u$  到结点  $v$  的流。

若  $(u, v) \notin E$ ，记  $f(u, v) = 0$ ，表示从结点  $u$  到结点  $v$  没有流。

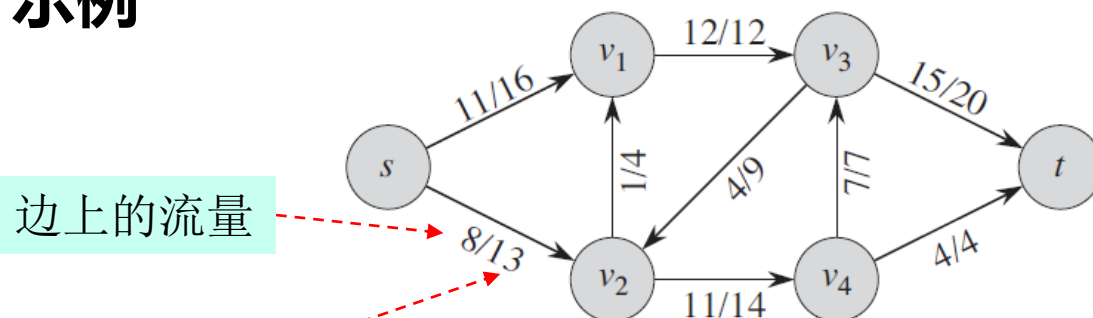
# 流的值:

一个流  $f$  的值定义为流出源结点  $s$  的总流量减去流入源结点  $s$  的总流量, 用  $|f|$  表示:

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$

但通常流网络中没有流入源结点的边, 因此  $\sum_{v \in V} f(v, s) = 0$

## 示例



边上的流量

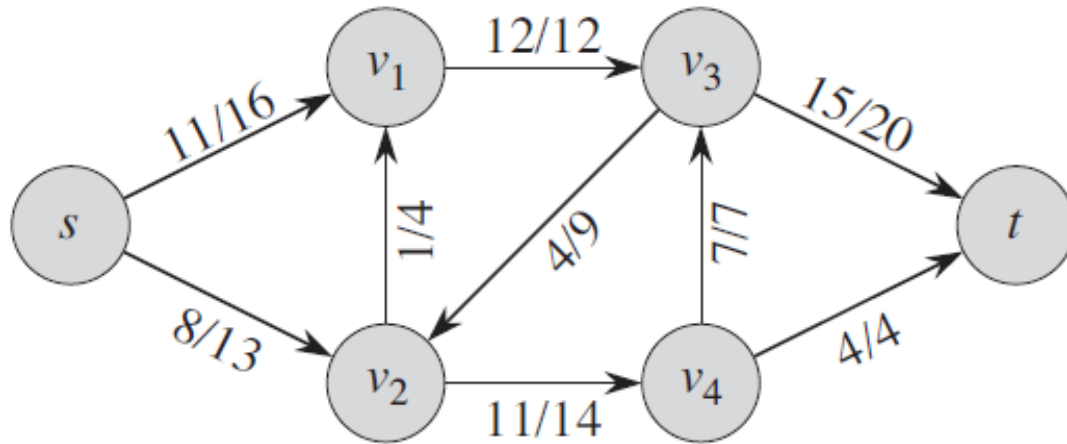
边的容量

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) = 19$$

$f(s, v_1) = 11$   
 $f(s, v_2) = 8$   
 $f(v_1, v_3) = 12$   
 $f(v_2, v_1) = 1$   
 $f(v_2, v_4) = 11$   
 $f(v_3, v_2) = 4$   
 $f(v_4, v_3) = 7$   
 $f(v_3, t) = 15$   
 $f(v_4, t) = 4$

**最大流问题：**就是在给定的流网络G中选找一个**流值最大的流**的问题

**最大流示例**



$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) = 19$$

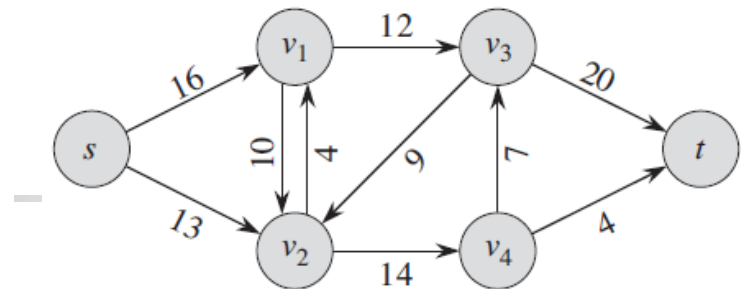


上述定义的流网络有两个标准特性：

(1) **无反向边**，或者称为**反平行边**

即，如果  $(u, v) \in E$ ，则  $(v, u) \notin E$ 。

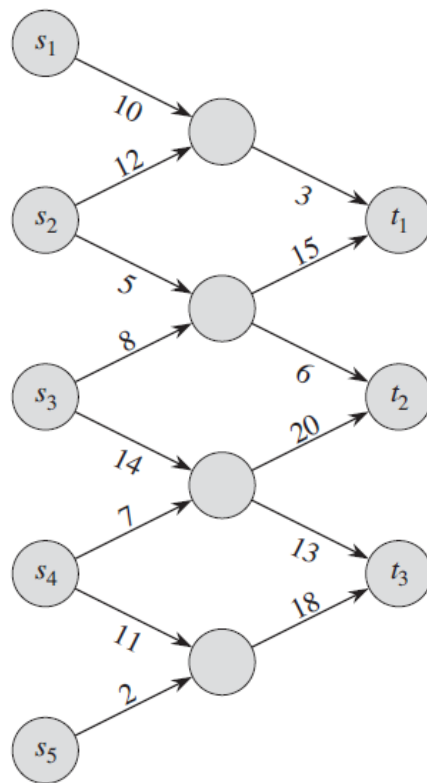
这里， $(v, u)$ 、 $(u, v)$  互为反平行边。



反平行边代表逆向的输入/输出

(2) **只有单一的源结点和汇点**

不满足上述要求的流网络这里视为非标准的**一般流网络**。对于一般流网络需转化为标准流网络进行处理。



一般流网络

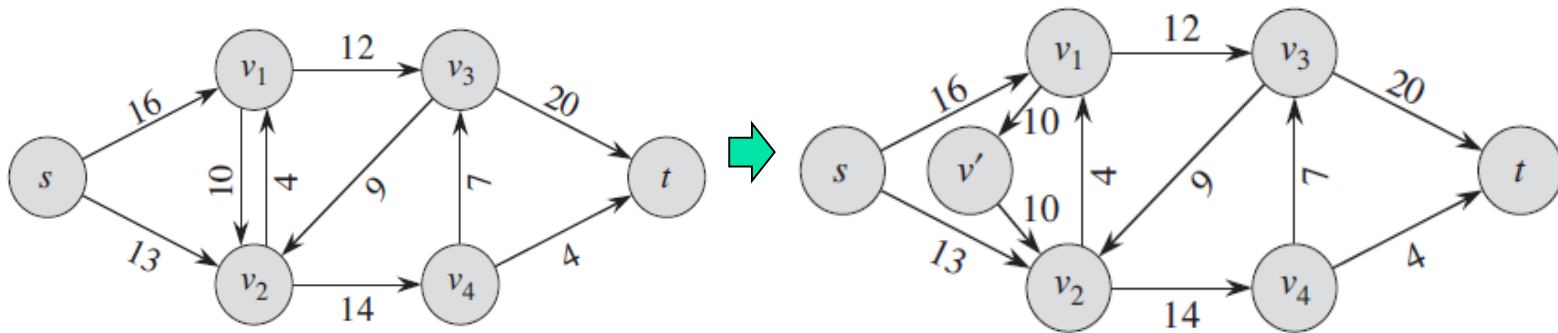
# (1) 具有反平行边的流网络



转化方法：对每一组反平行边  $(u,v)$  和  $(v,u)$ ，选择其中的一条，比如  $(u,v)$ ，然后加入一个新的结点  $v'$ ，将其分为两条边  $(u,v')$  和  $(v',v)$ ，并将两条新加入的边的容量设为被替代掉的边  $(u,v)$  的容量，即

$$c(u, v') = c(v', u) = c(u, v)$$

如：



具有反平行边  $(v_1, v_2)$   $(v_2, v_1)$

替换  $(v_1, v_2)$

可以证明，**转换后的网络与原网络等价**。留作练习 (26.1-1)

## (2) 具有多个源结点和多个汇点的网络

◆如果流网络中有多个源结点  $\{s_1, s_2, \dots, s_m\}$

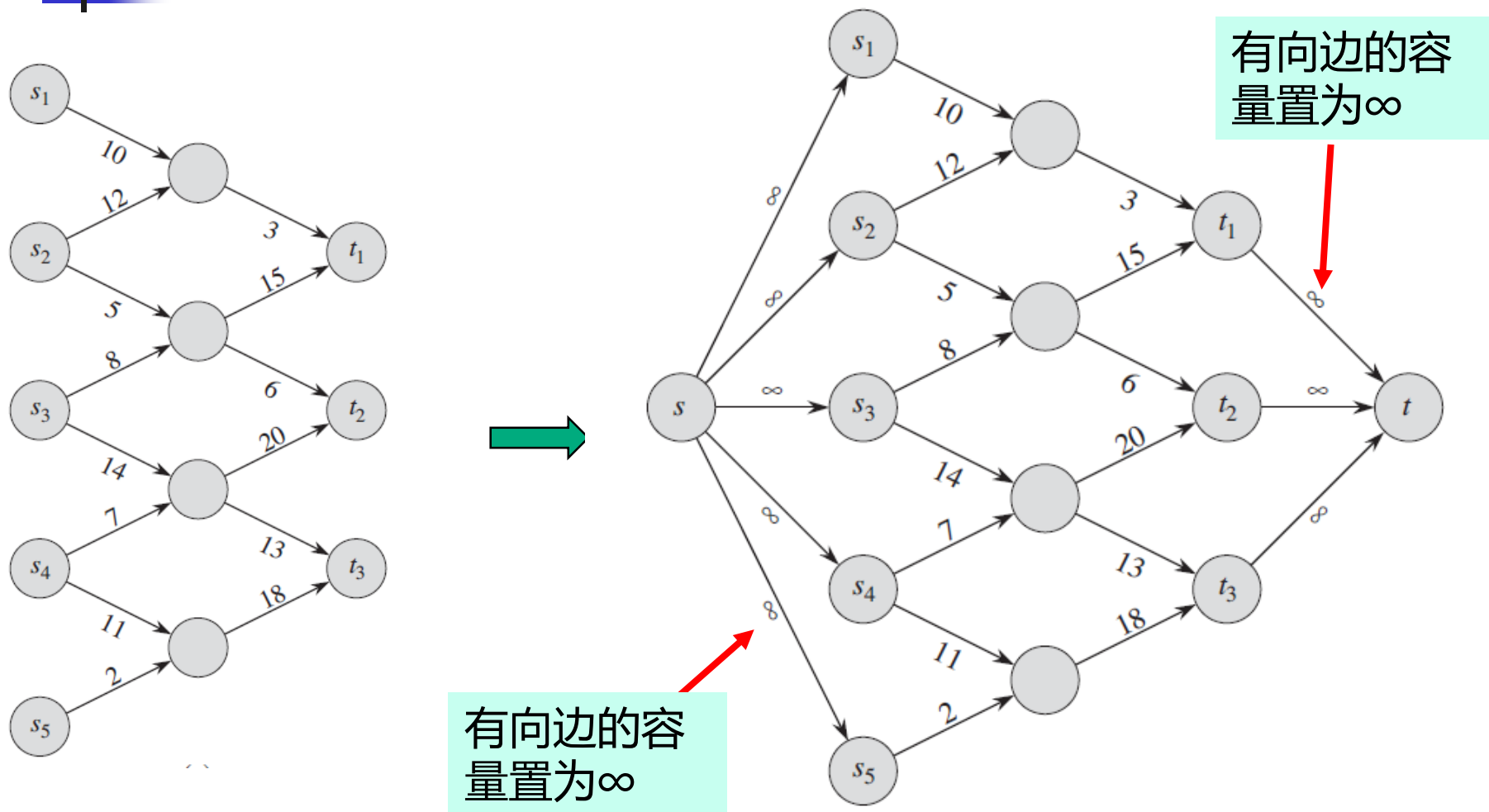
**转化方法:** 加入一个**超级源结点s**, 并加入有向边  $(s, s_i)$ ,  
然后令  $c(s, s_i) = \infty, 1 \leq i \leq m$ ;

◆如有多个汇点  $\{t_1, t_2, \dots, t_n\}$

**转化方法:** 加入一个**超级汇点t**, 并加入有向边  $(t_i, t)$ ,  
然后令  $c(t_i, t) = \infty, 1 \leq i \leq n$ 。

超级源结点s能够给原来的多个源结点 $s_i$ 提供所需的流量, 超级汇点t可以消费原来所有汇点 $t_i$ 所消费的流量。**可以证明转化后的两个网络是等价的, 具有相同的流。** 证明留作练习26.1-2。

例：将一个具有5个源结点、3个汇点的一般流网络转换成为一个等价的单源结点单汇点的流网络



# 如何求流网络的最大流？

基本思路：从最小流值开始，一点一点地增加，直到最大值。

从而引出一下问题：

(1) 最小流值是多少？——→ 不小于0即可，可以从0开始；

(2) 怎么增加流值？

(3) 何时能得到最大流？

算法不可能无休止地进行下去，当达到最大流时，怎么知道并结束计算呢？

- 由于有容量限制和结点流量守恒，不是所有的边都能以等于容量的最大流值传输流，否则接收结点接收不了（t除外），发送结点也没有足够的流量发送（s除外），甚至有些边上没有流。**整个网络的流要在所有的边中均衡分布，以达到整体最大的效果。**
- 对网络整体流值的增加，不意味着对所有边的流都增加，其中有些不变，甚至不用；而剩下的则是**有的增加，有的可能要减少，以便把流分布到其它路径上以获得更大的整体流量。**
- **那么，对哪些边增加、哪些边减少呢？**

## 26.2 Ford-Fulkerson方法

1955年由Lester R. Ford, Jr. 和 Delbert R. Fulkerson提出

### 1. **Ford-Fulkerson方法**的基本思想

**通过不断增加可行流值的方式找到最大流：**

- (1) 从流值为0的初始流开始；
- (2) 通过某种方法，对流值进行增加；
- (3) 确认无法再增加流值时，即得到最大流；

# Ford-Fulkerson方法的要点:

- (1) 对流网络 $G$ ，在其“**残存网络 $G_f$** ” (residual network) 中寻找一条“**增广路径 $p$** ” (augmenting path) 。
- (2) 如果存在增广路径，则对路径上边的流量进行修改，以增加流网络的流量。
- (3) 重复这一过程，直到不再存在增广路径为止。

判断是否得到最大流的理论基础是**最大流最小切割定理**，该定理将说明在**算法终止时将获得一个最大流**。

## Ford-Fulkerson方法的过程描述

FORD-FULKERSON-METHOD( $G, s, t$ )

- 1 initialize flow  $f$  to 0
- 2 **while** there exists an augmenting path  $p$  in the residual network  $G_f$
- 3     augment flow  $f$  along  $p$
- 4 **return**  $f$

寻找增广路径

沿增广路径增加流值



## 2. 残存网络 (Residual Network)

对给定流网络 $G$ 和流量 $f$ ,  $G$ 的**残存网络**  $G_f$  由 $G$ 中的结点和以下的边组成:

对于 $G$ 中任意边 $(u,v)$ ,

(1) 若  $f(u,v) < c(u,v)$ , 则将**边 $(u,v)$** 和它的**反向边 $(v,u)$** 都加入 $G_f$ , 并设其 “**残存容量**” 为:

$$c_f(u, v) = c(u, v) - f(u, v)$$

$$c_f(v, u) = f(u, v)$$

■ 残存容量 $c_f(u, v)$ 反映了边上可以增加流量的空间。

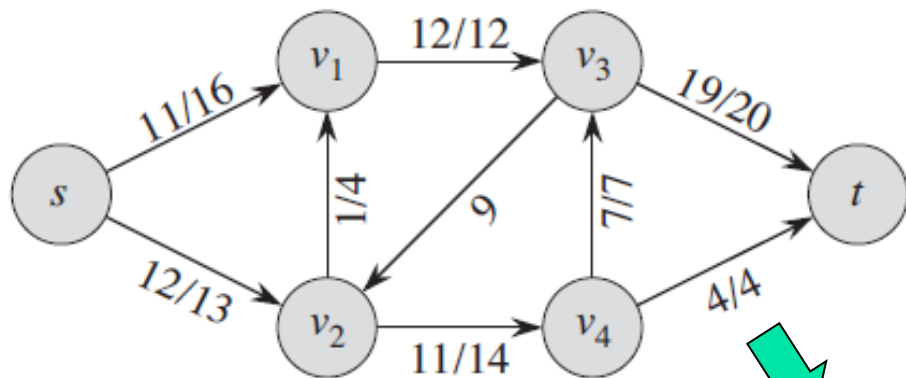
(2) 如果边 $(u,v)$ 的流量等于其容量, 则 $c_f(u,v) = 0$ , 此时 $(u,v)$

不加入 $G_f$ , 即**只有有多余流量的边才加入 $G_f$** 。

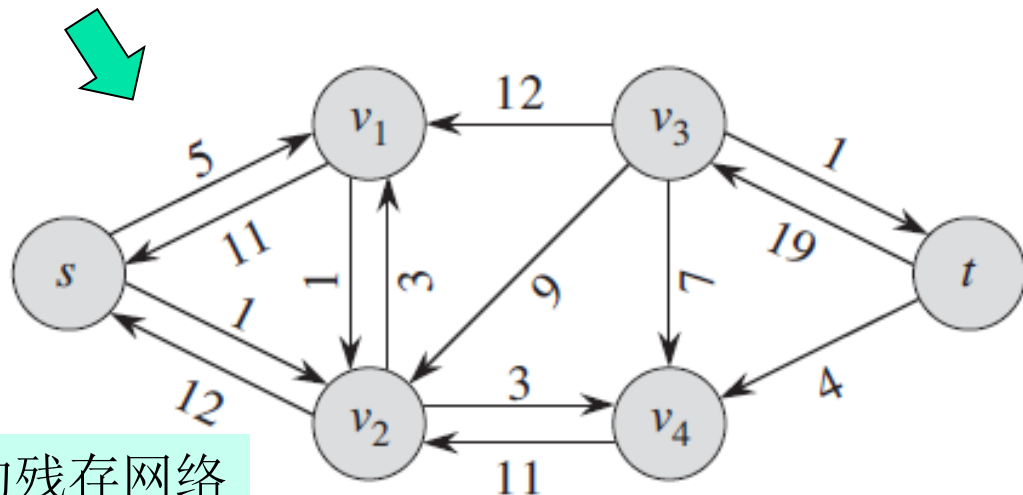
但仍将**反向边 $(v,u)$** 加入 $G_f$ , 并置  $c_f(v,u) = f(u,v)$

■ 目的也是为对一个正流量  $f(u,v)$  进行缩减, 且最多减少  $f(u,v)$ 。

例:



一个流网络G



G的残存网络

# 残存容量

设流网络  $G = (V, E)$ ,  $f$  是  $G$  中的一个流,  $u, v \in V$

◆ 定义边  $(u, v)$  的**残存容量**  $c_f(u, v)$  为:

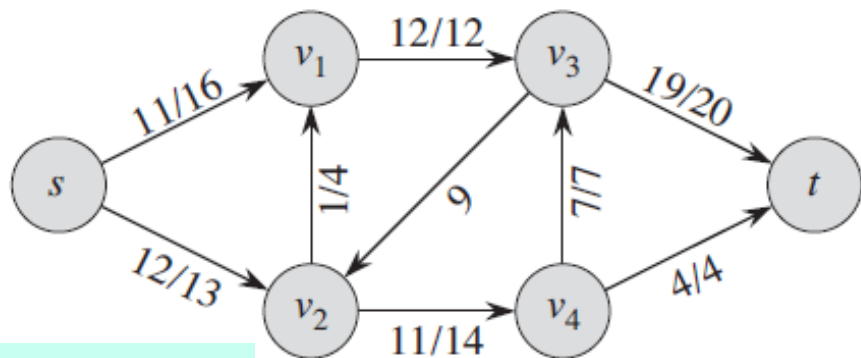
$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E, \\ f(v, u) & \text{if } (v, u) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

**残存网络:** 由  $f$  所诱导的图  $G$  的残存网络记为  $G_f = (V, E_f)$ ,

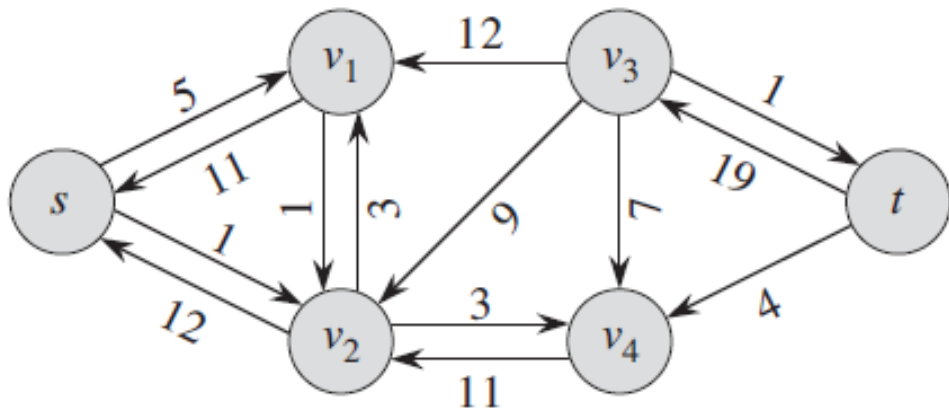
其中  $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$

即  **$G_f$  由残存流量大于 0 的边构成**, 且  $|E_f| \leq 2 |E|$ 。

# 残存网络示例:



流网络G



由f诱导的G的残存网络  $G_f$

$G_f$  中边的残存容量

“正向” 边

反向边

$$c_f(s, v_1) = 5$$

$$c_f(s, v_2) = 1$$

$$c_f(v_1, v_3) = 0$$

$$c_f(v_2, v_4) = 3$$

$$c_f(v_2, v_1) = 3$$

$$c_f(v_3, v_2) = 9$$

$$c_f(v_3, t) = 1$$

$$c_f(v_4, t) = 0$$

$$c_f(v_4, v_3) = 0$$

$$c_f(v_1, s) = 11$$

$$c_f(v_3, v_1) = 12$$

$$c_f(v_2, s) = 12$$

$$c_f(v_1, v_2) = 1$$

$$c_f(v_4, v_2) = 11$$

$$c_f(v_3, v_4) = 7$$

$$c_f(t, v_3) = 19$$

$$c_f(t, v_4) = 4$$

$$c_f(v_2, v_3) = 0$$

■ 残存网络 $G_f$  类似于一个容量为 $c_f$ 的流网络，但 $G_f$ 不满足流网络的定义：因为 $G_f$ 可能包含边 $(u,v)$ 和它的反向边 $(v,u)$ 。

◆ 但除此之外，残存网络和流网络具有同样的性质。

因此，也可以在残存网络中定义一个流：

设 $f$ 是 $G$ 的一个流，记 $f'$ 是残存网络 $G_f$ 中的一个流

- $f'$  针对残存网络 $G_f$ 中的容量 $c_f$ 定义且满足流的性质。
- 容量限制、流量守恒
- $f'$  指出这样一个路线图：如何在原来的流网络中增加流。

定义  $f \uparrow f'$  为流  $f'$  **对流  $f$  的递增** (augmentation) , 是一个从  $V \times V$  到  $R$  的函数:

$$(f \uparrow f')(u, v) = \begin{cases} f(u, v) + f'(u, v) - f'(v, u) & \text{if } (u, v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

其中,  $f'(u, v)$  表示的是边  $(u, v)$  上流量的增加,  
 $f'(v, u)$  表示的是边  $(u, v)$  上流量的减少

在残存网络中将流量发送到反向边上等同于在原来的网络中缩减流量。在残存网络中将流量推送回去称为**抵消操作**(cancellation), 其意义在于调整流网络中总流量的分布。

注:  $f \uparrow f'$  最终成为原网络中的一个流, 是基于  $f'$  对原网络中流  $f$  调整的结果。见下。

**引理26.1** 设 $G=(V,E)$ 为一个流网络，源结点为 $s$ ，汇点为 $t$ 。设 $f$ 为 $G$ 中的一个流。设 $G_f$ 为由流 $f$ 所诱导的 $G$ 的残存网络，设 $f'$ 为 $G_f$ 中的一个流。那么 $f \uparrow f'$  是 $G$ 的一个流，其值为： $|f \uparrow f'| = |f| + |f'|$

**引理26.1说明：**如果在 $G'$  中能够找到一个合法的流 $f'$ ，则可以用 $f'$ 对 $G$ 上原来的流 $f$ 进行调整，调整的结果是：

$$|f \uparrow f'| = |f| + |f'|$$

如果 $|f'| > 0$ ，则可以达到对 $f$ 递增的效果。

## 引理26.1的证明思路:

首先, 证明  $f \uparrow f'$  是图  $G$  的一个流, 即  $f \uparrow f'$  对  $E$  中的每条边满足容量限制, 且对每个结点  $V - \{s, t\}$  满足流量守恒性质。

其次, 证明流量值公式成立。

流必须非负且不能超过给定容量的限制

对于非源结点和汇点的结点, 流入等于流出

**证明过程如下:**



首先, 证明  $f \uparrow f'$  是图G的一个流, 满足容量限制和流量守恒。

## (1) 容量限制

根据定义, 如果边  $(u, v) \in E$ , 则  $c_f(v, u) = f(u, v)$ 。而且

$$f'(v, u) \leq c_f(v, u) = f(u, v)$$

因此,

$$\begin{aligned}(f \uparrow f')(u, v) &= f(u, v) + f'(u, v) - f'(v, u) && \text{(根据定义)} \\ &\geq f(u, v) + f'(u, v) - f(u, v) \\ &= f'(u, v) && (f'(v, u) \leq f(u, v)) \\ &\geq 0.\end{aligned}$$

**即,  $f \uparrow f'$  是非负值。**

此外,

$$(f \uparrow f')(u, v)$$

$$= f(u, v) + f'(u, v) - f'(v, u) \quad (\text{根据定义})$$

$$\leq f(u, v) + f'(u, v) \quad (\text{根据流值非负})$$

$$\leq f(u, v) + \underline{c_f(u, v)} \quad (\text{残存网络容量限制})$$

$$= f(u, v) + \underline{c(u, v) - f(u, v)} \quad (\text{根据残存容量的定义})$$

$$= c(u, v) .$$

即, 递增后, 流  $f \uparrow f'$  也不超过容量的限制。

**容量限制:** 对任一结点对  $u$  和  $v$ , 满足  $0 \leq f(u, v) \leq c(u, v)$

## (2) 流量守恒

因为 $f$ 和 $f'$ 均遵守流量守恒性质，所以对所有的 $u \in V - \{s, t\}$ 结点，有：

$$\begin{aligned}\sum_{v \in V} (f \uparrow f')(u, v) &= \sum_{v \in V} (f(u, v) + f'(u, v) - f'(v, u)) \\&= \sum_{v \in V} f(u, v) + \sum_{v \in V} f'(u, v) - \sum_{v \in V} f'(v, u) \\&= \sum_{v \in V} \underline{f(v, u)} + \sum_{v \in V} \underline{f'(v, u)} - \sum_{v \in V} \underline{f'(u, v)} \\&= \sum_{v \in V} (f(v, u) + f'(v, u) - f'(u, v)) \\&= \sum_{v \in V} (f \uparrow f')(v, u),\end{aligned}$$

**流量守恒：** 对于  $u \in V - \{s, t\}$  , 有  $\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$

**其次证明:**  $|f \uparrow f'| = |f| + |f'|$

证明:

定义  $V_1 = \{v : (s, v) \in E\}$

$V_1$ 是所有从源结点有边可达的结点的集合

$V_2 = \{v : (v, s) \in E\}$

$V_2$ 是所有有边通往源结点s的结点的集合

由于 $(s, v)$ 和 $(v, s)$ 不可能同时存在于流网络中, 所以有:

$$V_1 \cap V_2 = \emptyset \text{ 且 } V_1 \cup V_2 \subseteq V。$$

$$\begin{aligned} \text{则有, } |f \uparrow f'| &= \sum_{v \in V} (f \uparrow f')(s, v) - \sum_{v \in V} (f \uparrow f')(v, s) \\ &= \sum_{\underline{v \in V_1}} (f \uparrow f')(s, v) - \sum_{\underline{v \in V_2}} (f \uparrow f')(v, s), \end{aligned}$$

注: 若  $(w, x) \notin E$ , 则  $|f \uparrow f'(w, x)| = 0$ 。

根据  $f \uparrow f'$  的定义，重组上式有，

$$|f \uparrow f'|$$

$$= \sum_{v \in V_1} (f \uparrow f')(s, v) - \sum_{v \in V_2} (f \uparrow f')(v, s)$$

$$= \sum_{v \in V_1} f(s, v) + \sum_{v \in V_1} f'(s, v) - \sum_{v \in V_1} f'(v, s) - \sum_{v \in V_2} f(v, s) - \sum_{v \in V_2} f'(v, s) + \sum_{v \in V_2} f'(s, v)$$

$$= \sum_{v \in V_1} f(s, v) - \sum_{v \in V_2} f(v, s) + \sum_{v \in V_1} f'(s, v) + \sum_{v \in V_2} f'(s, v) - \sum_{v \in V_1} f'(v, s) - \sum_{v \in V_2} f'(v, s)$$

$$= \sum_{\underline{v \in V_1}} f(s, v) - \sum_{\underline{v \in V_2}} f(v, s) + \sum_{\underline{v \in V_1 \cup V_2}} f'(s, v) - \sum_{\underline{v \in V_1 \cup V_2}} f'(v, s)$$

$$= \sum_{\underline{v \in V}} f(s, v) - \sum_{\underline{v \in V}} f(v, s) + \sum_{\underline{v \in V}} f'(s, v) - \sum_{\underline{v \in V}} f'(v, s)$$

注:  $V_1 \cup V_2 \subseteq V$

$$= \underline{|f|} + \underline{|f'|}$$

证毕

★  $|f \uparrow f'| = |f| + |f'|$  给出了对G的流增加的方法，如何找  $f'$  ?

### 3. 增广路径

对给定流网络 $G=(V,E)$ 和流 $f$ ，**增广路径** $p$  (Augmenting Path)

是其残存网络 $G_f$ 中一条从源结点 $s$ 到汇点 $t$ 的简单路径。

➤根据残存网络的定义，对于增广路径上的一条边 $(u,v)$ ，其可增加的流值最大为该边的残存容量  $c_f(u,v)$ 。

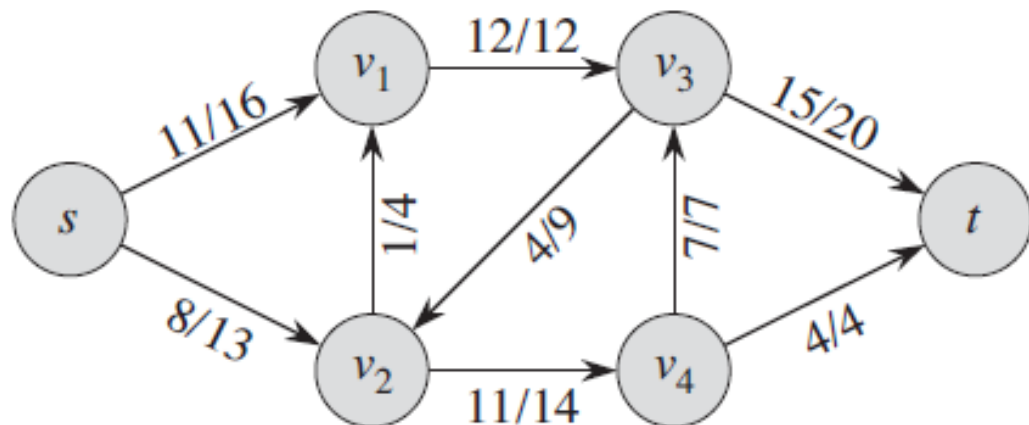
➤对于一条增广路径 $p$ ，能增加的最大流值称为该路径的**残存容量**，等于 $p$ 上所有边残存容量的最小值，即：

$$c_f(p) = \min \{c_f(u,v) : (u,v) \text{ is on } p\}$$

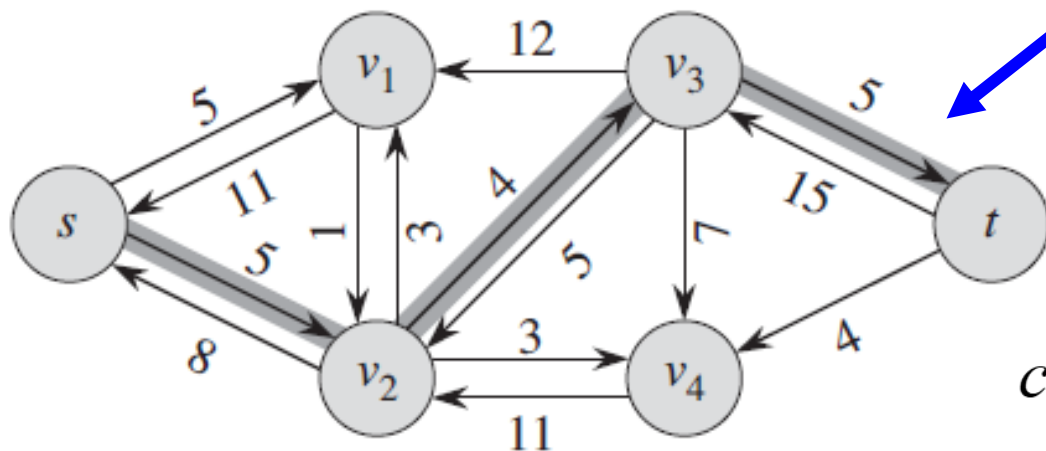
**注：**  $c_f(p) > 0$

**残存容量最小的边是瓶颈，增广路径的流量受其限制。**

示例:



流网络



残存网络

$$p = \{s, v_2, v_3, t\}$$

$$c_f(s, v_2) = 5$$

$$c_f(v_2, v_3) = 4$$

$$c_f(v_3, t) = 5$$

边的残存容量

$$c_f(p) = \min\{c_f(s, v_2), c_f(v_2, v_3), c_f(v_3, t)\} = 4$$

增广路径的  
残存容量

**引理26.2** 设  $G = (V, E)$  为一个流网络,  $f$  是图  $G$  的一个流。记  $p$  为其残存网络  $G_f$  中的一条增广路径。

定义一个函数  $f_p : V \times V \rightarrow R$  如下:

$$f_p(u, v) = \begin{cases} c_f(p) & \text{if } (u, v) \text{ is on } p, \\ 0 & \text{otherwise.} \end{cases}$$

则  $f_p$  是残存网络  $G_f$  中的一个流, 其值为  $|f_p| = c_f(p) > 0$ 。

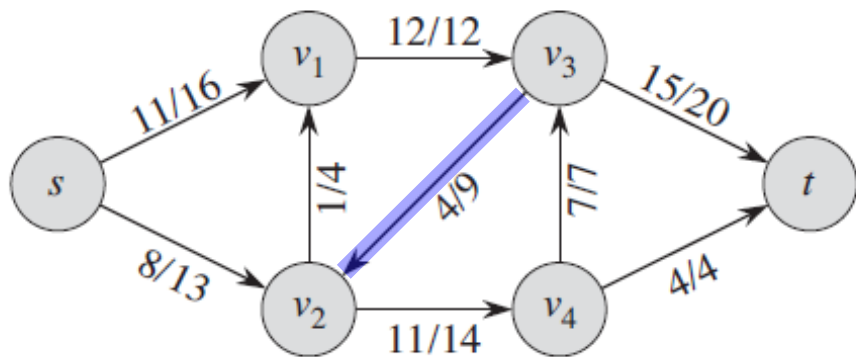
引理26.2告诉我们,  $G$  的残存网络  $G'$  中, 一个合法的流  $f'$  怎么去找:  $f_p$  就是这样的合法的流。

**所以可以用  $f_p$  为  $f$  进行递增计算。**

**证明:** 略 (参见26.2-7)。

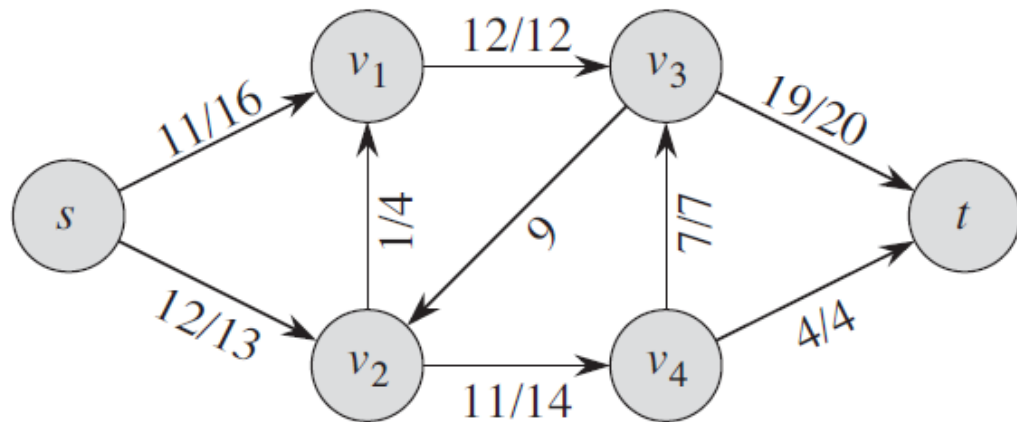


$f_p$  的作用：将流  $f$  增加  $f_p$  的量，得到的仍是  $G$  的一个流，  
且该流的值更加接近最大值。

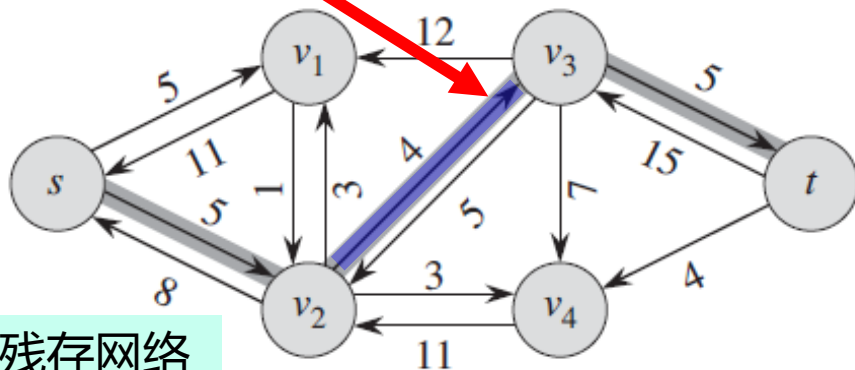


流网络

增广路径



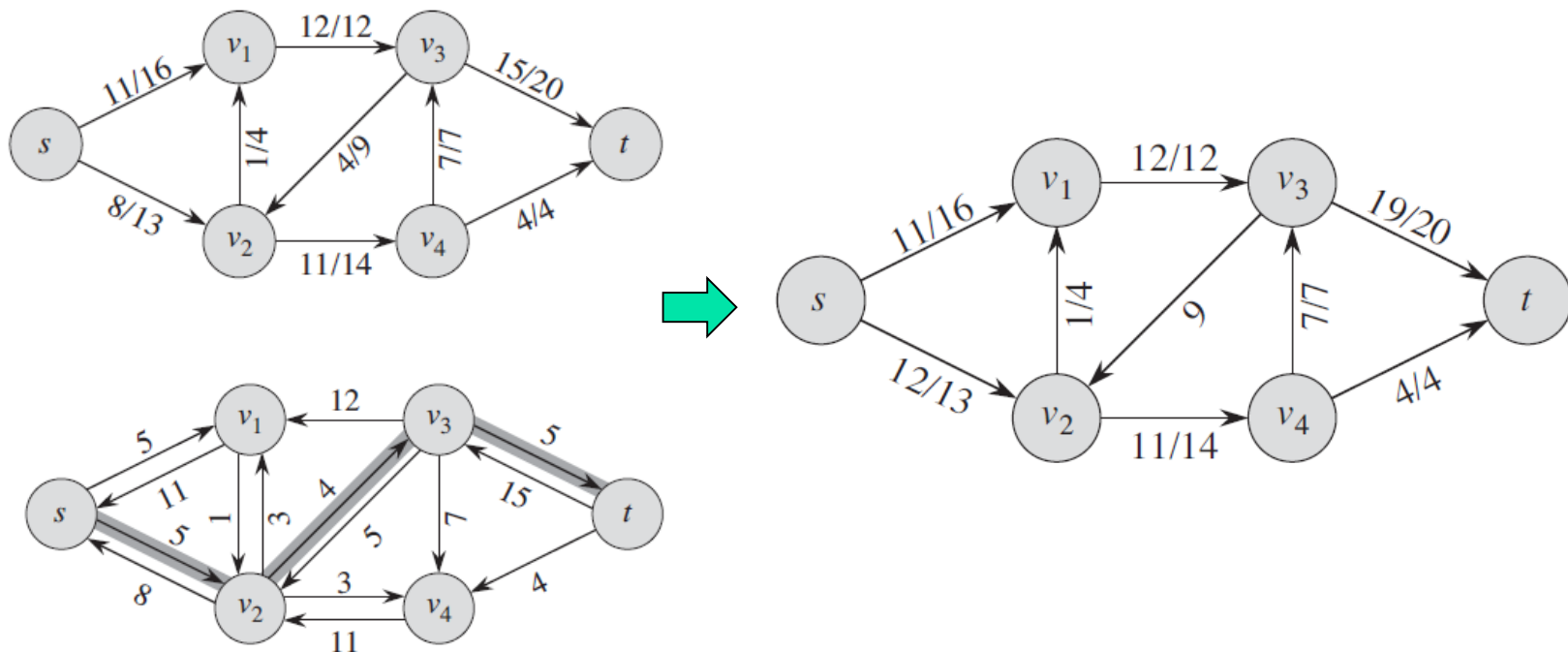
用  $f_p$  增加流量后的流网络



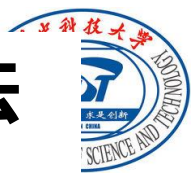
残存网络

推论26.3: 设  $G = (V, E)$  为一个流网络,  $f$  是图  $G$  的一个流,  $p$  为残存网络  $G_f$  中的一条增广路径。设  $f_p$  是上式定义的残存网络的流, 假定将  $f$  增加  $f_p$  的量, 则**函数  $f \uparrow f_p$  是图  $G$  的一个流, 其值为  $|f \uparrow f_p| = |f| + |f_p| > |f|$** 。

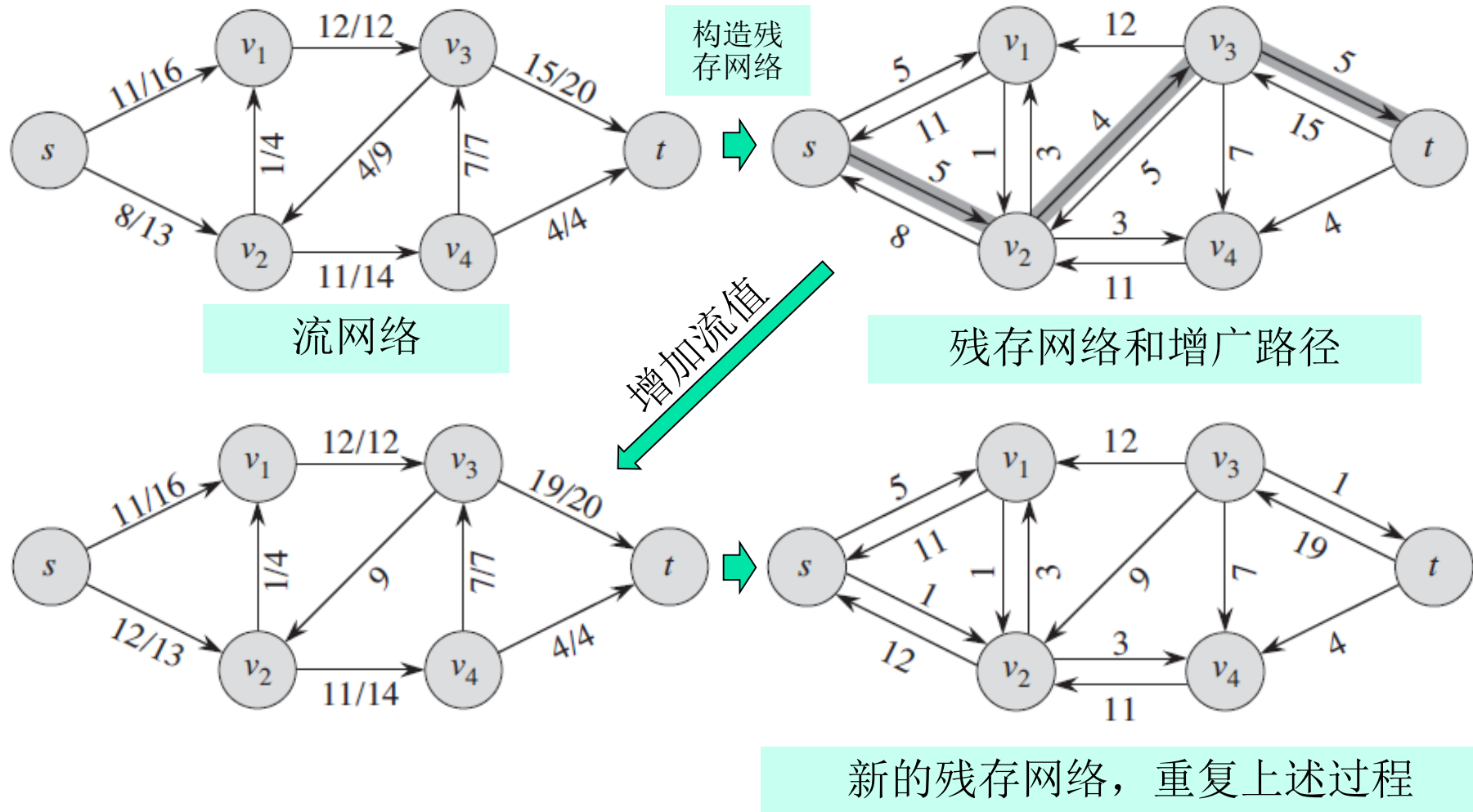
**证明:** 根据引理26.1和引理26.2可得证。



# 利用残存网络和增广路径实现Ford-Fulkerson方法



示例:



# 残存网络和增广路径总结

■ **已经解决的问题**：如何增加流值——利用增广路径。

■ **未解决的问题**：如何判断算法终止时，确实找到了最大流呢？

——利用**最大流最小切割定理**进行判定

## □ **最大流最小切割定理**

◆ **建立最大流和切割容量之间的关系**，从而建立最大流和残存网络增广路径上的残存容量之间的关系。

□ **一个流是最大流当且仅当其残存网络中不包含任何增广路径。**

## 4. 流网络的切割

给定流网络  $G=(V,E)$ ，源结点为  $s$ ，汇点为  $t$ 。

定义一个**切割** $(S,T)$ ，将结点集合  $V$  分成两部分  $S$  和  $T=V-S$ ，使得  $s \in S, t \in T$ 。

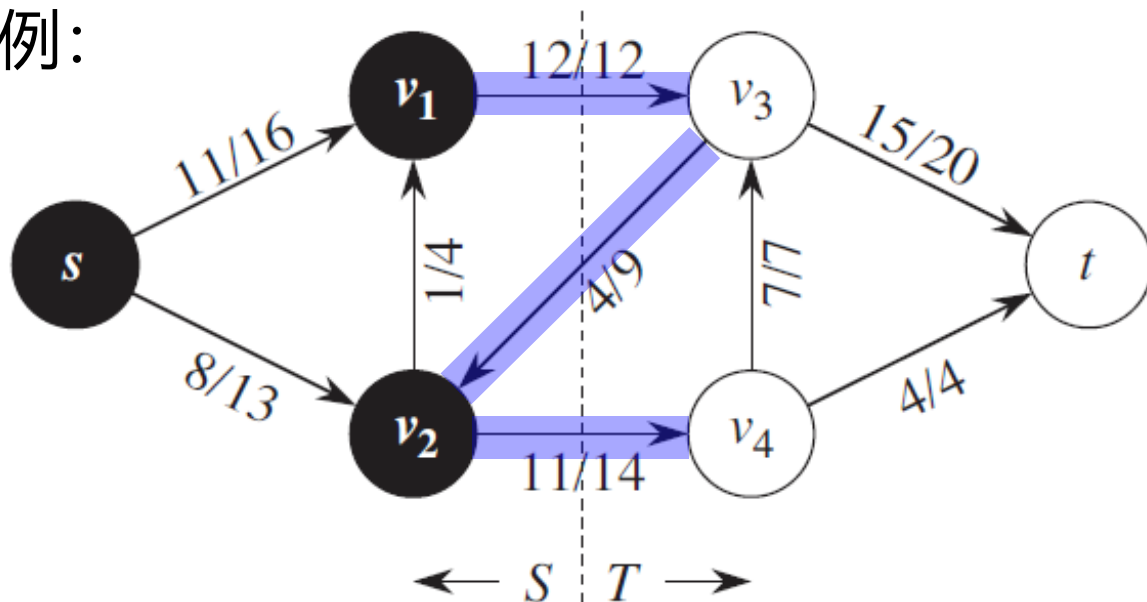
若  $f$  是  $G$  上的一个流，定义**横跨切割** $(S,T)$ 的**净流量** $f(S,T)$ 为：

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

定义**切割** $(S,T)$ 的**容量**为： $c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$

**最小切割**：一个网络的最小切割是**网络中容量最小的切割**。

切割的示例：



**切割：**  $(S, T) : S = \{s, v_1, v_2\}, T = \{v_3, v_4, t\}$

**横跨切割的净流量：**  $f(S, T) = f(v_1, v_3) + f(v_2, v_4) - f(v_3, v_2) = 12 + 11 - 4 = 19$

净流量的计算是从S到T的流量减去从T到S的反方向的流量

**该切割的容量：**  $c(S, T) = c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$

容量计算只考虑从集合S发出进入集合T的边的容量

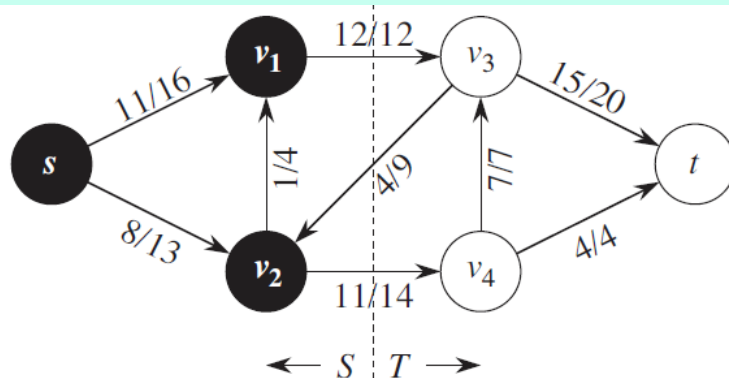
## 流、切割净流量和切割容量之间的关系

引理26.4 设 $f$ 为流网络 $G$ 的一个流, 该流网络的源结点为 $s$ , 汇点为 $t$ , 设 $(S,T)$ 为流网络 $G$ 的任意切割, 则**横跨切割 $(S,T)$**

**的净流量为**  $f(S,T) = |f|$ 。

引理26.4说明: 对流网络的任意切割 $(S,T)$ , 切割截面上的净流量就等于流网络的流量。

而且, 横跨任何切割的净流量都相同, 都等于流的值 $|f|$ 。



◆ 引理26.4的证明如下:

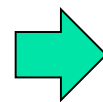
**证明:** 对于任意  $u \in V - \{s, t\}$ , 根据结点的流量守恒性质有:

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

所以有:  $\sum_{u \in S - \{s\}} \left( \sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) \right) = 0$

又根据流的定义:  $|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$

将两式相加





$$\begin{aligned}
 |f| &= \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) + \sum_{u \in S - \{s\}} \left( \sum_{v \in V} f(u, v) - \sum_{v \in V} f(v, u) \right) \\
 &= \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s) + \sum_{u \in S - \{s\}} \sum_{v \in V} f(u, v) - \sum_{u \in S - \{s\}} \sum_{v \in V} f(v, u) \\
 &= \sum_{v \in V} \left( f(s, v) + \sum_{u \in S - \{s\}} f(u, v) \right) - \sum_{v \in V} \left( f(v, s) + \sum_{u \in S - \{s\}} f(v, u) \right) \\
 &= \sum_{v \in V} \sum_{u \in S} f(u, v) - \sum_{v \in V} \sum_{u \in S} f(v, u)
 \end{aligned}$$

**注意到**  $V = S \cup T$  并且  $S \cap T = \emptyset$ , 将上式针对S和T进行分解, 得:

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

## 继续化简，有

$$\begin{aligned}|f| &= \sum_{v \in S} \sum_{u \in S} f(u, v) + \sum_{v \in T} \sum_{u \in S} f(u, v) - \sum_{v \in S} \sum_{u \in S} f(v, u) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\&= \sum_{v \in T} \sum_{u \in S} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) + \left( \sum_{v \in S} \sum_{u \in S} f(u, v) - \sum_{v \in S} \sum_{u \in S} f(v, u) \right) \\&= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \\&= f(S, T)\end{aligned}$$

证毕，即横跨（任何）切割的  
净流量都等于流的值  $|f|$ 。

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

# 净流量和切割的容量之间的关系

**推论26.5：** 流网络G中任意流f的值不能超过G的任意切割的容量。

**证明：** 设(S,T)为流网络G的任意切割，设f为G中的任意流。

根据因引理26.4和容量限制，可以得到

$$\begin{aligned}
 |f| &= f(S, T) \\
 &= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{u \in S} \sum_{v \in T} f(v, u) \\
 &\leq \sum_{u \in S} \sum_{v \in T} f(u, v) \\
 &\leq \sum_{u \in S} \sum_{v \in T} c(u, v) \\
 &= c(S, T) .
 \end{aligned}$$

**推论26.5说明，任何流，包括最大流都不能超过最小切割的容量的限制。**

容量限制：  $0 \leq f(u, v) \leq c(u, v)$

**问题：**最大流不可能超过最小切割的容量，**最大流值是否等于最小切割的容量？**

**答案是肯定的，两者是相等的。**

□ 由**最大流最小切割定理**进行了描述

## 最大流最小切割定理

定理 26.6：设 $f$ 为流网络 $G=(V,E)$ 中的一个流，该流网络的源结点为 $s$ ，汇点为 $t$ ，则下面的条件是等价的：

- (1)  $f$ 是 $G$ 的一个最大流
- (2) 残存网络  $G_f$  不包括任何增广路径
- (3)  $|f| = c(S,T)$  , 其中 $(S,T)$ 是流网络 $G$ 的某个切割

# 最大流最小切割定理

定理 26.6 : 设 $f$ 为流网络 $G=(V,E)$ 中的一个流, 该流网络的源结点为 $s$ , 汇点为 $t$ , 则下面的条件是等价的:

- (1)  $f$ 是 $G$ 的一个最大流
- (2) 残存网络  $G_f$  不包括任何增广路径
- (3)  $|f| = c(S,T)$ , 其中 $(S,T)$ 是流网络 $G$ 的某个切割

**最大流最小切割定理说明**, 流网络 $G$ 的最大流 $f$  在流的值等于任意切割的容量时达到, 而此时在对应的**残存网络 $G_f$** 中**不再有增广路径**。所以可以用有无增广路径判断当前流 $f$  是否是最大流, 如果是, 则算法也可以终止了。

## 无增广路径的含义：

对当前流  $f$ ，由  $f$  诱导的残存网络  $G_f$  中不再有从  $s$  到  $t$  的、路径残余容量大于 0 的简单路径  $p$ 。即不再有路径  $p$  使得  $|f_p| = c_f(p) > 0$ 。

而  $G_f$  中没有残余容量等于 0 的边。所以**无增广路径** 事实上与  $G_f$  中没有从  $s$  到  $t$  的路径等价。

# 最大流最小切割定理的证明

定理 26.6: 设 $f$ 为流网络 $G=(V,E)$ 中的一个流, 该流网络的源结点为 $s$ , 汇点为 $t$ , 则下面的条件是等价的:

- (1)  $f$ 是 $G$ 的一个最大流
- (2) 残存网络  $G_f$  不包括任何增广路径
- (3)  $|f| = c(S,T)$ , 其中 $(S,T)$ 是流网络 $G$ 的某个切割

(1) $\rightarrow$ (2)证明: 使用反证法

假定 $f$ 是 $G$ 的一个最大流, 但残存网络  $G_f$  同时**包含一条增广路径** $p$ 。设  $f_p$  是残存网络  $G_f$  中由增广路径 $p$ 定义的流。

根据推论1, 对 $f$ 增加流  $f_p$  所形成的流是 $G$ 中的一个流, 且流值严格大于 $|f|$ :  $|f \uparrow f_p| = |f| + |f_p| > |f|$ , **与 $f$ 是最大流冲突。**



(2) 残存网络 不包括任何增广路径

(3)  $|f| = c(S, T)$ , 其中 $(S, T)$ 是流网络 $G$ 的某个切割

**(2)→(3) 证明:** 假定  $G_f$  中不包含任何增广路径, 即**残存网络  $G_f$  中不存在从源结点 $s$ 到汇点 $t$ 的路径。**

定义**切割 $(S, T)$** 如下

$S = \{v \in V : \text{在 } G_f \text{ 中存在一条从 } s \text{ 到 } v \text{ 的路径}\}, T = V - S$

**为证明定理, 只需证明  $|f| = c(S, T)$ 。**

**根据引理3,  $f(S, T) = |f|$ 。所以只需要证明  $f(S, T) = c(S, T)$**

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



考虑任意一对结点  $u \in S, v \in T$  , 有三种情况:

**情况1:**  $(u, v) \in E$  , 则有  $f(u, v) = c(u, v)$ 。

否则  $c_f(u, v) = c(u, v) - f(u, v) > 0$  , 从而得到  $(u, v) \in E_f$  ,  
并推出  $v \in S$  , 与  $v$  的定义矛盾。

**情况2:**  $(v, u) \in E$  , 则有  $f(v, u) = 0$  。

否则  $c_f(u, v) = f(v, u) > 0$  , 从而得到  $(u, v) \in E_f$  ,  
并推出  $v \in S$  , 与  $v$  的定义矛盾。

**情况3:**  $(v, u) \notin E$  且  $(u, v) \notin E$  , 则  $f(u, v) = f(v, u) = 0$

**综上**, 可以得到

$$\begin{aligned} f(S, T) &= \sum_{u \in S} \sum_{v \in T} f(u, v) - \sum_{v \in T} \sum_{u \in S} f(v, u) \\ &= \sum_{u \in S} \sum_{v \in T} c(u, v) - \sum_{v \in T} \sum_{u \in S} 0 \\ &= c(S, T) . \end{aligned}$$

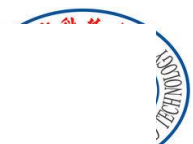
证毕

- (1)  $f$  是  $G$  的一个最大流
- (2) 残存网络  $G_f$  不包括任何增广路径
- (3)  $|f| = c(S, T)$ , 其中  $(S, T)$  是流网络  $G$  的某个切割

**(3)→(1)证明:** 根据推论26.5, 对于所有切割  $(S, T)$ ,  $|f| \leq c(S, T)$ 。

因此,  $|f| = c(S, T)$  意味着  $f$  是一个最大流。

**定理证毕**



# 如何实施Ford-Fulkerson方法求解最大流：？

FORD-FULKERSON-METHOD( $G, s, t$ )

- 1 initialize flow  $f$  to 0
- 2 **while** there exists an augmenting path  $p$  in the residual network  $G_f$
- 3     augment flow  $f$  along  $p$
- 4 **return**  $f$

**基本思想：**通过不断增加**可行流值**的方式找到最大流

**技术路线：**

- (1) 从流值为0的初始流开始
- (2) 通过某种方法，对流值进行增加：**在残存网络中寻找增广路径，利用增广路径对流值进行增加。**
- (3) 确认无法增加流值，即得到最大流：**当残存网络中不再有增广路径为止，此时获得最大流。**

# Ford-Fulkerson算法的细化

FORD-FULKERSON( $G, s, t$ )

```
1  for each edge  $(u, v) \in G.E$ 
2       $(u, v).f = 0$ 
3  while there exists a path  $p$  from  $s$  to  $t$  in the residual network  $G_f$ 
4       $c_f(p) = \min \{c_f(u, v) : (u, v) \text{ is in } p\}$ 
5      for each edge  $(u, v)$  in  $p$ 
6          if  $(u, v) \in E$ 
7               $(u, v).f = (u, v).f + c_f(p)$ 
8          else  $(v, u).f = (v, u).f - c_f(p)$ 
```

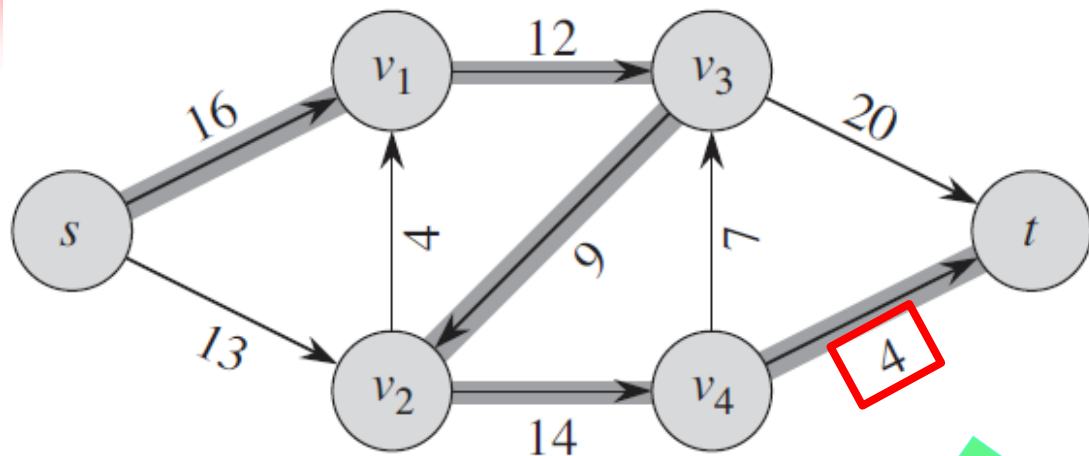
修正与增广路径对应的  
每条边的流值

如何寻找增广路径?

➤ 可通过深度优先搜索或者广度优先搜索得到

# 基本的Ford-Fulkerson算法

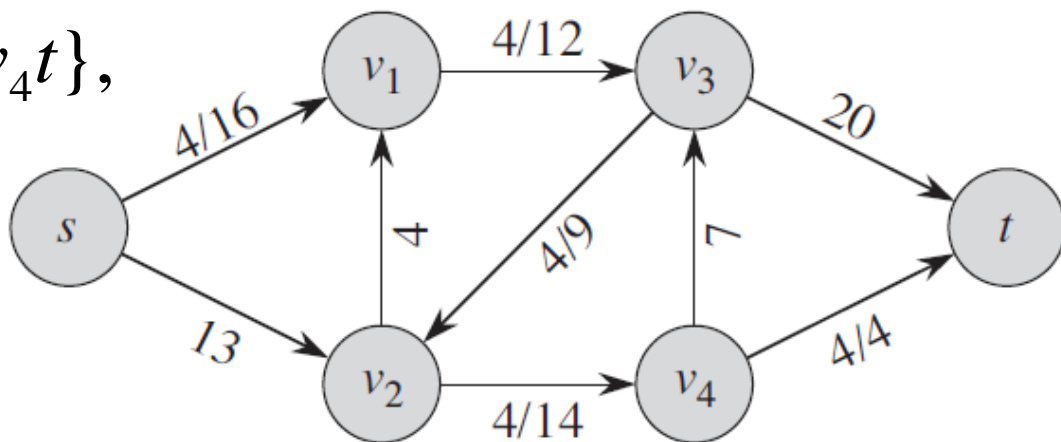
## 算法运行示例：Iteration 1



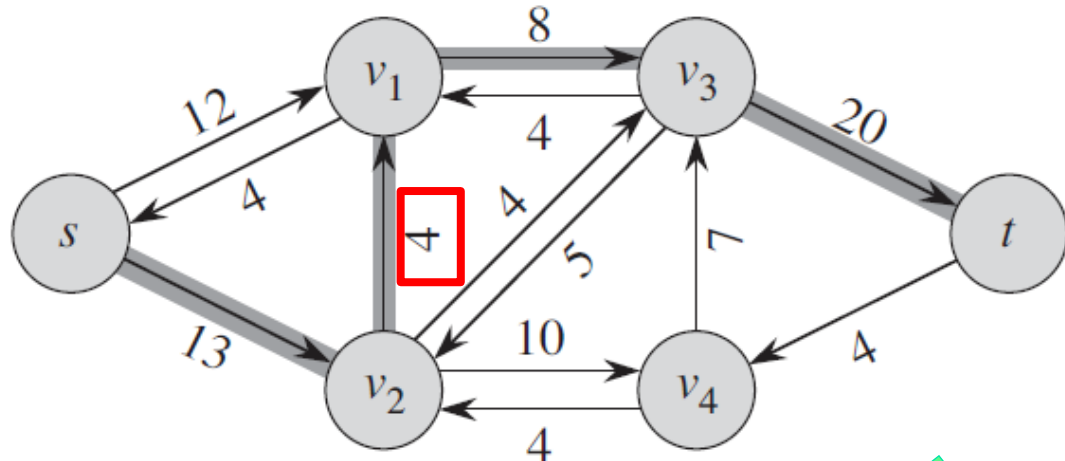
流网络和增广路径

$$p = \{sv_1, v_1v_3, v_3v_2, v_2v_4, v_4t\},$$

$$c_f(p) = 4$$



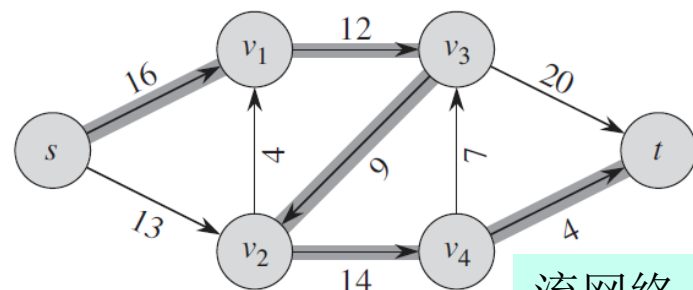
# Iteration 2



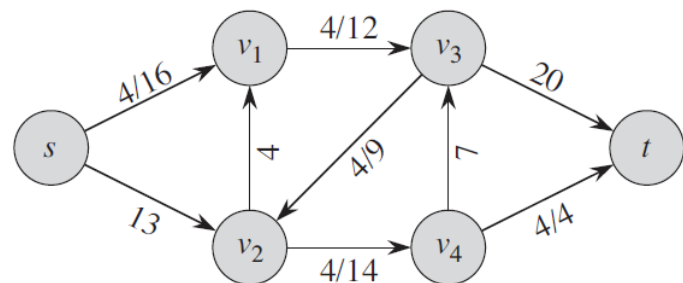
残余网络和增广路径

$$p = \{sv_2, v_2v_1, v_1v_3, v_3t\},$$

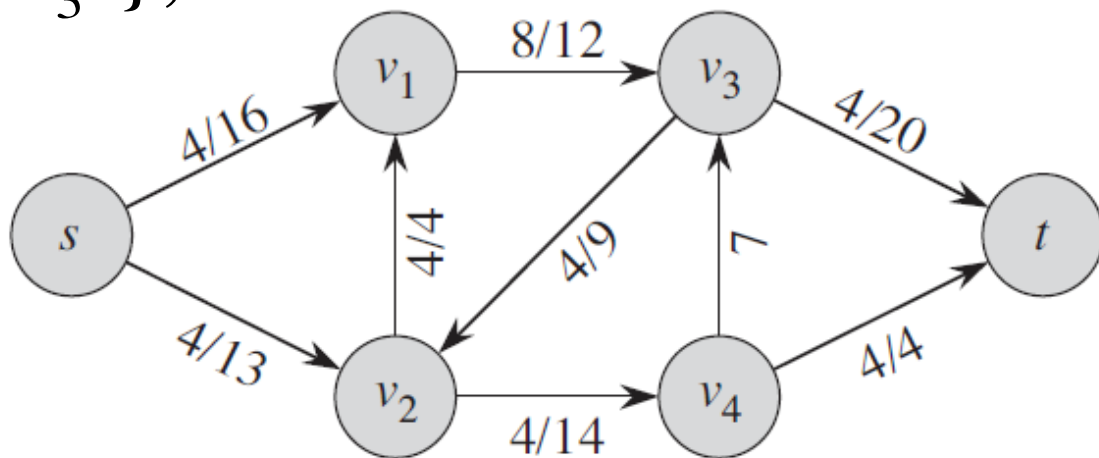
$$c_f(p) = 4$$



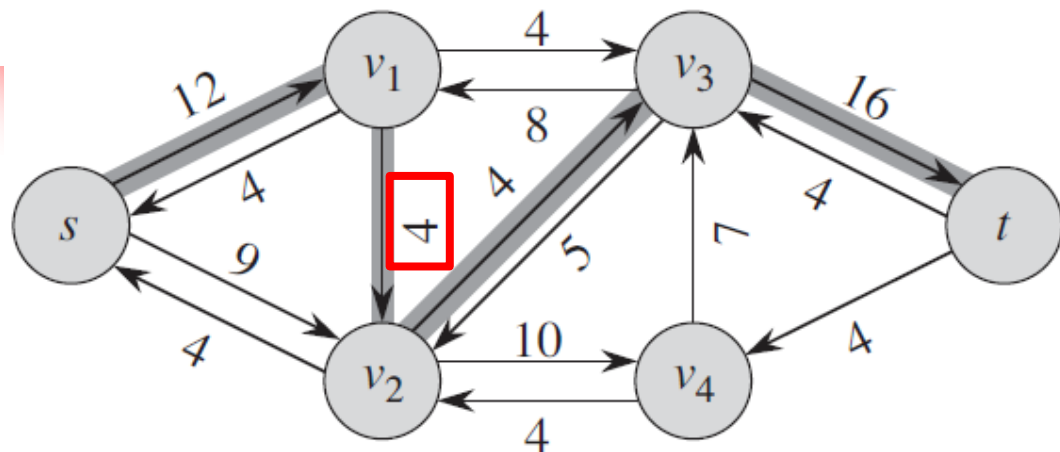
流网络



Iteration 1 流网络



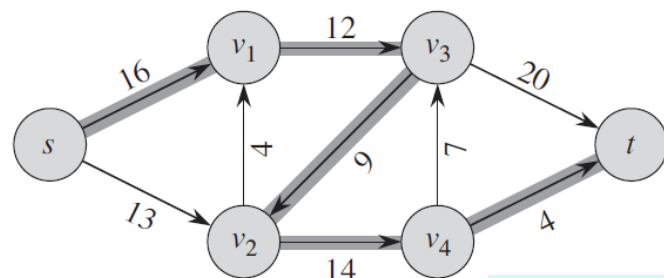
# Iteration 3



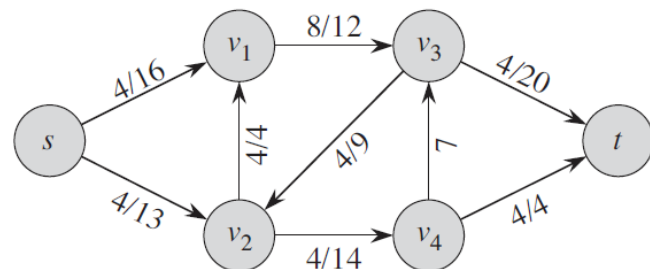
残余网络和增广路径

$$p = \{sv_1, v_1v_2, v_2v_3, v_3t\},$$

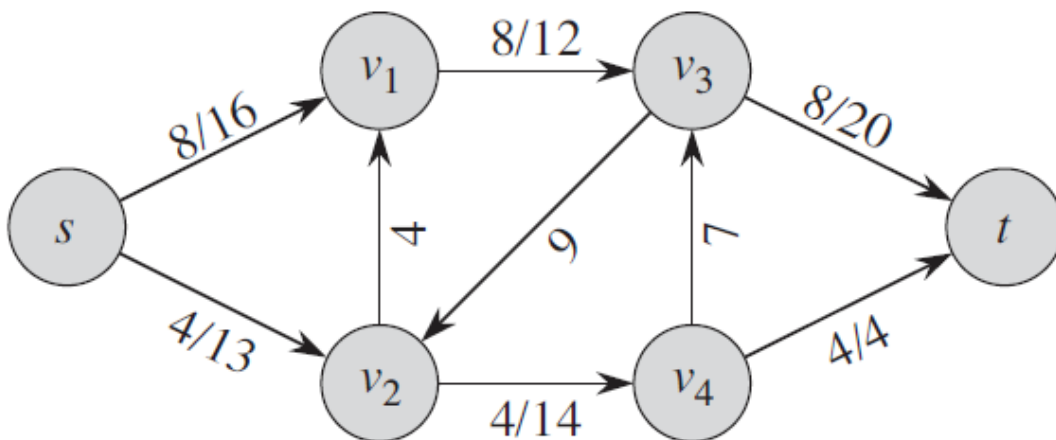
$$c_f(p) = 4$$



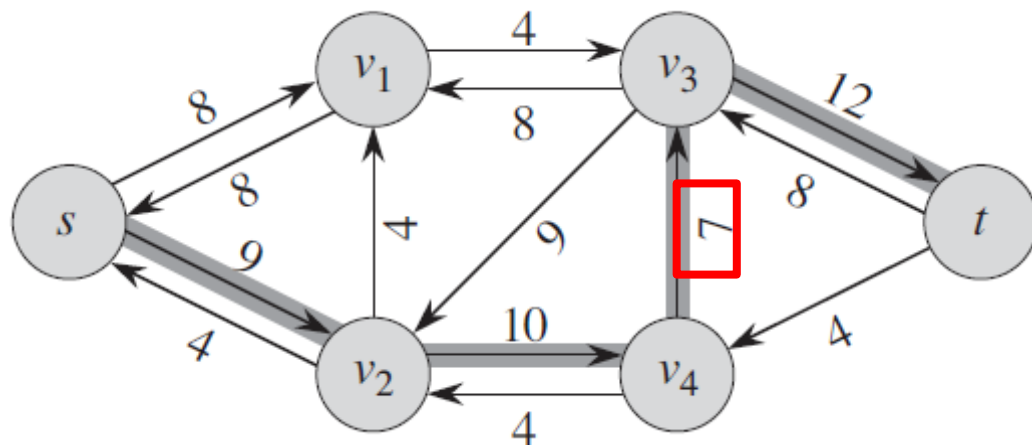
流网络



Iteration 2流网络



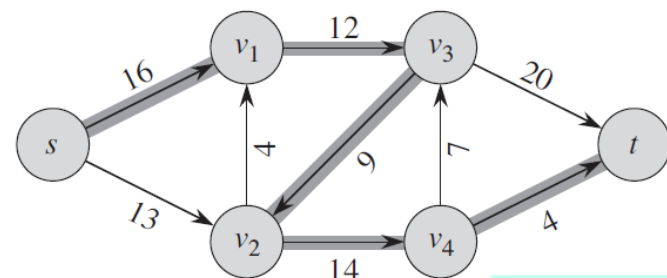
# Iteration 4



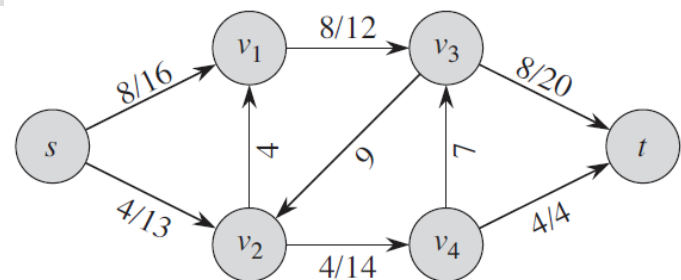
残余网络和增广路径

$$p = \{sv_2, v_2v_4, v_4v_3, v_3t\},$$

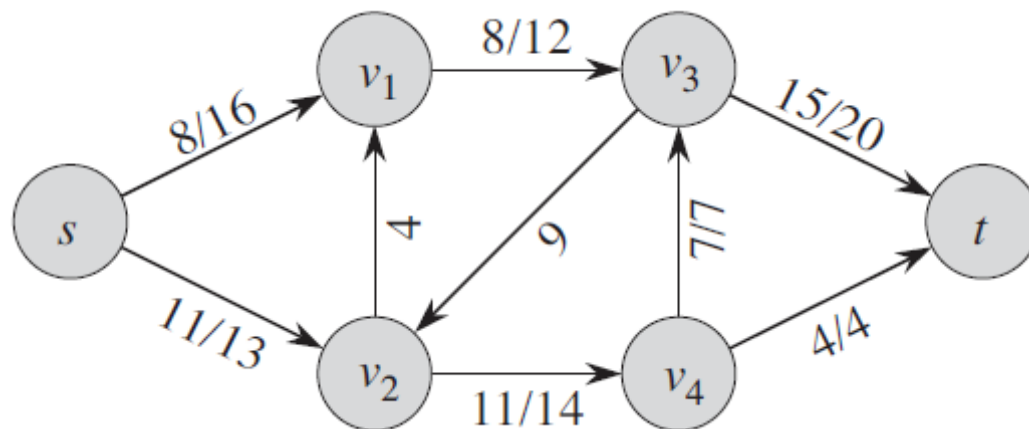
$$c_f(p) = 7$$



流网络

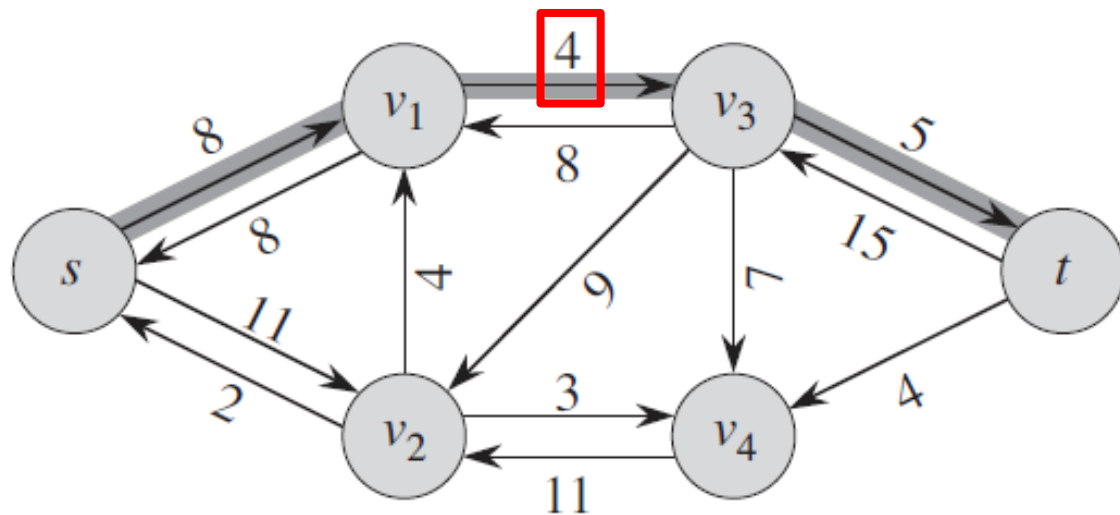


Iteration 3流网络





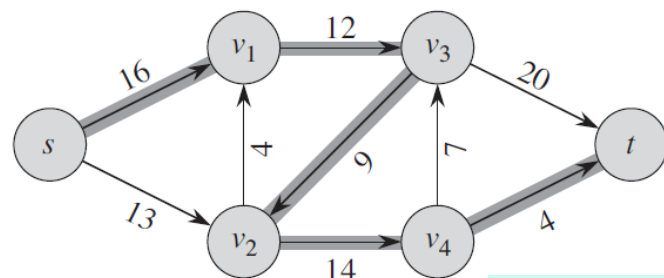
# Iteration 5



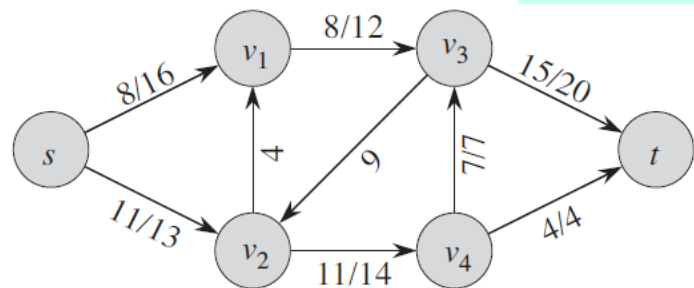
残余网络和增广路径

$$p = \{sv_1, v_1v_3, v_3t\},$$

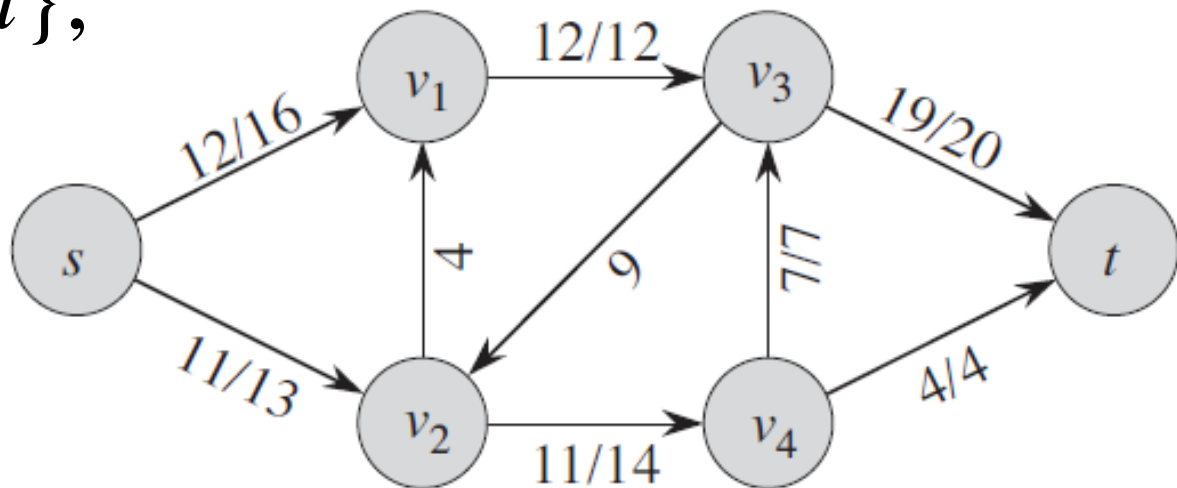
$$c_f(p) = 4$$

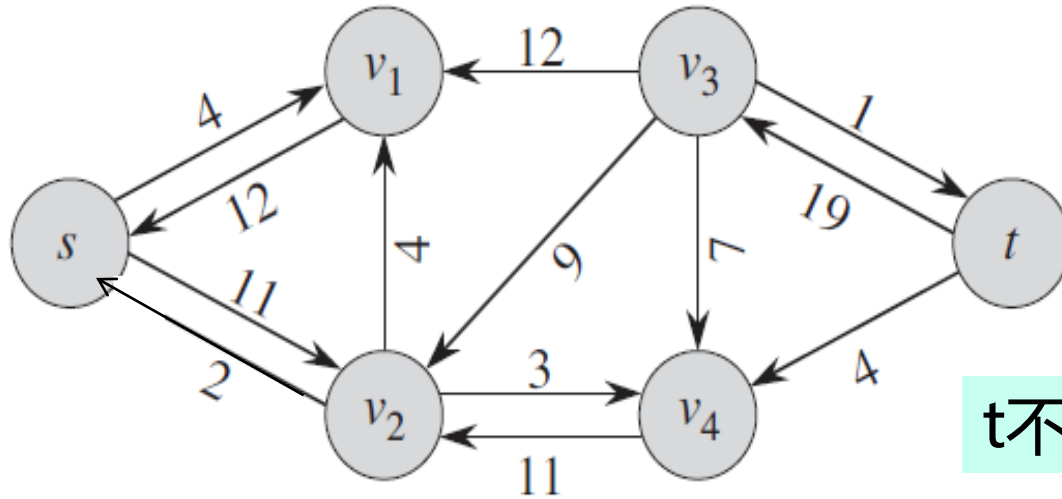


流网络



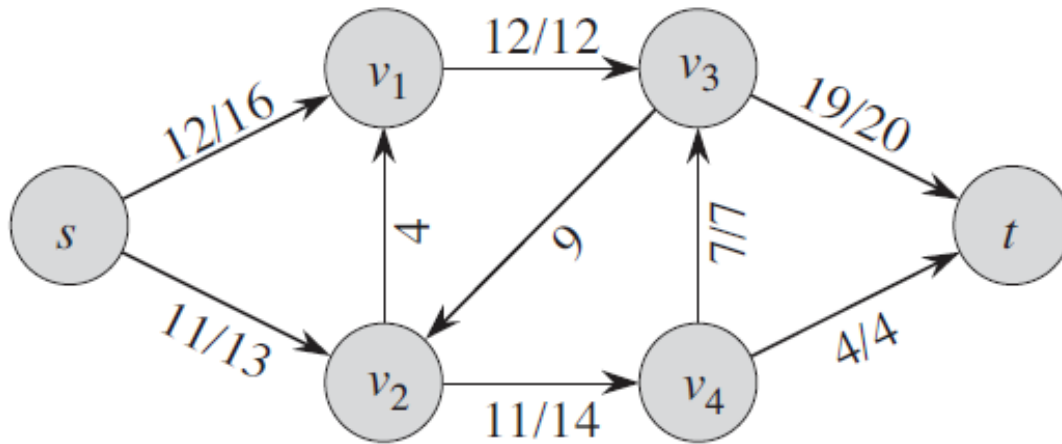
Iteration 4流网络





t不可达 = 无增广路径

无增广路径，得到最大流  $|f| = 23$



# Ford-Fulkerson算法复杂性分析

**假定：**所有的容量均为整数。

**时间复杂度：** $O(|E| \times |f^*|)$  ,

其中 $E$ 是流网络的边集,  $f^*$ 是最大流值

注：如边的容量是无理数时, Ford-Fulkerson方法可能不能终止, 也就是不会得到最大流。

**运行时间分析：**包括两部分

(1) while循环的次数：因为每一次循环, 流值至少增加1,

所以最多有 $O(|f^*|)$ 循环

## (2) 每次循环所用时间

每次循环做三个主要操作，**计算残存网络、寻找增广路径和更新每条边的流值。**

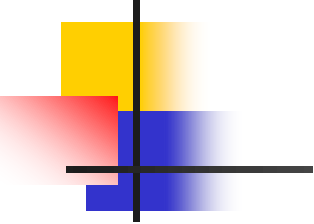
◆计算残存网络需要计算每条边的残存容量，运行时间为  $O(|E|)$

◆利用深度优先搜索或者广度优先搜索，计算增广路径，运行时间为  $O(|V| + |E|) \in O(|E|)$

◆更新每条边的流值的时间是：  $O(|E|)$

综上，**总的计算时间是**  $O(|E| \times |f^*|)$

**算法存在的问题：**时间复杂性与最大流值有关，当最大流值非常大时，效率较低。

A decorative graphic on the left side of the slide, consisting of overlapping yellow, red, and blue squares with a black crosshair.

```
// C++ program for implementation of Ford Fulkerson algorithm
#include <iostream>
#include <limits.h>
#include <string.h>
#include <queue>
using namespace std;

// Number of vertices in given graph
#define V 6

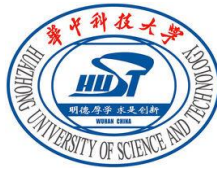
/* Returns true if there is a path from source 's' to sink 't' in
residual graph. Also fills parent[] to store the path */
bool bfs(int rGraph[V][V], int s, int t, int parent[])
{
    // Create a visited array and mark all vertices as not visited
    bool visited[V];
    memset(visited, 0, sizeof(visited));

    // Create a queue, enqueue source vertex and mark source vertex
    // as visited
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    // Standard BFS Loop
    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v=0; v<V; v++)
        {
            if (visited[v]==false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    // If we reached sink in BFS starting from source, then return
    // true, else false
    return (visited[t] == true);
}
```



```
// Returns the maximum flow from s to t in the given graph
```

```
int fordFulkerson(int graph[V][V], int s, int t)
```

```
{
```

```
    int u, v;
```

```
    // Create a residual graph and fill the residual graph with
```

```
    // given capacities in the original graph as residual capacities
```

```
    // in residual graph
```

```
    int rGraph[V][V]; // Residual graph where rGraph[i][j] indicates
```

```
        // residual capacity of edge from i to j (if there
```

```
        // is an edge. If rGraph[i][j] is 0, then there is not)
```

```
    for (u = 0; u < V; u++)
```

```
        for (v = 0; v < V; v++)
```

```
            rGraph[u][v] = graph[u][v];
```

```
    int parent[V]; // This array is filled by BFS and to store path
```

```
    int max_flow = 0; // There is no flow initially
```

```
    // Augment the flow while there is path from source to sink
```

```
    while (bfs(rGraph, s, t, parent))
```

```
    {
```

```
        // Find minimum residual capacity of the edges along the
```

```
        // path filled by BFS. Or we can say find the maximum flow
```

```
        // through the path found.
```

```
        int path_flow = INT_MAX;
```

```
        for (v=t; v!=s; v=parent[v])
```

```
        {
```

```
            u = parent[v];
```

```
            path_flow = min(path_flow, rGraph[u][v]);
```

```
        }
```

```
        // update residual capacities of the edges and reverse edges
```

```
        // along the path
```

```
        for (v=t; v != s; v=parent[v])
```

```
        {
```

```
            u = parent[v];
```

```
            rGraph[u][v] -= path_flow;
```

```
            rGraph[v][u] += path_flow;
```

```
        }
```

```
        // Add path flow to overall flow
```

```
        max_flow += path_flow;
```

```
    }
```

```
    // Return the overall flow
```

```
    return max_flow;
```

```
}
```

```
// Driver program to test above functions
```

```
int main()
```

```
{
```

```
    // Let us create a graph shown in the above example
```

```
    int graph[V][V] = { {0, 16, 13, 0, 0, 0},
```

```
                        {0, 0, 10, 12, 0, 0},
```

```
                        {0, 4, 0, 0, 14, 0},
```

```
                        {0, 0, 9, 0, 0, 20},
```

```
                        {0, 0, 0, 7, 0, 4},
```

```
                        {0, 0, 0, 0, 0, 0}}
```

```
};
```

```
cout << "The maximum possible flow is " << fordFulkerson(graph, 0, 5);
```

```
return 0;
```

```
}
```

# Edmonds-Karp算法

**Edmonds-Karp算法**仍然是基于Ford-Fulkerson方法，不同的是**使用广度优先搜索寻找源结点到汇点的最短路径作为增广路径**，从而得到不依赖于最大流值的运行时间上界

FORD-FULKERSON( $G, s, t$ )

```

1  for each edge  $(u, v) \in G.E$ 
2       $(u, v).f = 0$ 
3  while there exists a path  $p$  from  $s$  to  $t$  in the residual network  $G_f$ 
4       $c_f(p) = \min \{c_f(u, v) : (u, v) \text{ is in } p\}$ 
5      for each edge  $(u, v)$  in  $p$ 
6          if  $(u, v) \in E$ 
7               $(u, v).f = (u, v).f + c_f(p)$ 
8          else  $(v, u).f = (v, u).f - c_f(p)$ 
    
```

**Edmonds-Karp算法**使用  
广度优先搜索寻找最短路  
径作为增广路径

运行时间:  $O(VE^2)$

# Edmonds-Karp算法运行时间分析

(1) 在残存网络中，采用广度优先搜索找一条从s到t的最短路径的时间是 $O(E)$ 。

(2) 可以证明，随着算法的进行，流的值不断增加，同时增广路径的长度也逐步是递增的。这使得在算法的整个执行过程中，总共最多会处理 $O(VE)$ 条关键边。

所以Edmonds-Karp算法运行时间为： $O(VE^2)$ 。

**证明过程：**

令  $\delta_f(u, v)$  表示残存网络中  $G_f$  中从结点u到结点v的最短路径距离，这里每条边的权重为单位距离，路径长度等于路径上的边数。



**引理4:** 如果Edmonds-Karp算法运行的流网络  $G = (V, E)$  上, 该网络的源结点是  $s$  汇点为  $t$ , 则对于所有结点  $v \in V - \{s, t\}$ , 残存网络  $G_f$  中的最短路径距离  $\delta_f(s, v)$  随着每次流量的递增而单调递增

**证明:** 利用反证法证明。假设对于某个结点  $v \in V - \{s, t\}$ , 存在一个流量递增操作, 导致从源结点  $s$  到  $v$  的路径距离减小。

设  $f$  是第一个导致某条最短路径距离减少的流量递增操作之前的流量,  $f'$  是递增操作之后的流量。

设  $v$  是在流递增操作中 shortest path 被减小的结点中  $\delta_{f'}(s, v)$  最小的结点, 根据假设, 应有  $\delta_{f'}(s, v) < \delta_f(s, v)$ 。

设  $p = s \cdots \rightarrow u \rightarrow v$  为残存网络  $G_{f'}$  中从源结点  $s$  到结点  $v$  的一条最短路径。

可以得到  $(u, v) \in E_{f'}$ , 并且  $\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1$

根据  $v$  的选取, 可以得到  $\delta_{f'}(s, u) \geq \delta_f(s, u)$

我们断言  $(u, v) \notin E_f$ 。否则就有

$$\begin{aligned}\delta_f(s, v) &\leq \delta_f(s, u) + 1 \\ &\leq \delta_{f'}(s, u) + 1 \\ &= \delta_{f'}(s, v)\end{aligned}$$

就与假设  $\delta_{f'}(s, v) < \delta_f(s, v)$  矛盾。

$v$  是最短路径变小的结点中最短路径最小的结点, 所以  $s$  到  $u$  的最短路径不能变小, 否则应选  $u$  而不是  $v$  了。

由上可得:  $(u, v) \in E_{f'}$  时,  $(u, v) \notin E_f$ 。

(1)  $(u, v) \notin E_f$ : 意味着相对流  $f$ ,  $f(u, v) = c(u, v)$ ,  $c_f(u, v) = 0$ ,

所以  $(u, v)$  不在残余网络  $G_f$  中;

(2)  $(u, v) \in E_{f'}$ : 意味着相对流  $f'$ ,  $f'(u, v) < c(u, v)$ ,  $c_{f'}(u, v) > 0$ 。

也因此有: **由残存网络  $G_f$  中计算得到的增广路径上边  $(v, u)$  上有流量, 导致增量计算以后, 边  $(u, v)$  上的流量因边  $(v, u)$  上流量的抵消而减少, 从而有  $c_{f'}(u, v) = c(u, v) - f'(u, v) > 0$ 。**



这也意味着,  $(v,u)$ 是相对于流 $f$ 的残存网络 $G_f$ 的增广路径上的一条边, 而Edmonds-Karp算法中, 增广路径是最短路径, 所以这也意味着,  $(v,u)$ 存在于残存网络 $G_f$ 中 $s$ 到 $u$ 的最短路径上, 而且是从源结点 $s$ 到结点 $u$ 的最短路径上的最后一条边。因此有

$$\begin{aligned}\delta_f(s, v) &= \delta_f(s, u) - 1 \\ &\leq \delta_{f'}(s, u) - 1 && \delta_{f'}(s, u) \geq \delta_f(s, u) \\ &= \delta_{f'}(s, v) - 2 && \delta_{f'}(s, u) = \delta_{f'}(s, v) - 1\end{aligned}$$

所以 $s$ 到 $v$ 的最短路径并没有减小, 与假设  $\delta_{f'}(s, v) < \delta_f(s, v)$  相矛盾。所以流量递增操作导致从源结点 $s$ 到 $v$ 的路径距离减小不成立。

**定理2：** 如果Edmonds-Karp算法运行在源结点为s汇点为t的流网络 $G=(V,E)$ 上，则算法执行的流量递增操作的次数为 $O(VE)$ 。

**关键边：** 在残存网络 $G_f$ 中，如果一条路径 $p$ 的残存容量是该条路径上边  $(u,v)$  的残存容量，即  $c_f(p) = c_f(u,v)$ ，那么 $(u,v)$ 称为增广路径 $p$ 上的关键边。

- 任何一条增广路径上至少存在一条关键边；
- 增流后，当前增广路径上的关键边将从残存网络中消失。

◆ 由前一引理已知：**Edmonds-Karp算法中，当流值增加的时候，从s到每个结点的距离会增加。**

◆ 下面利用这个性质证明：**每条边  $(u,v)$  成为关键边之后，再下一次成为关键边之时，s到u的最短距离会增加2。**

设 $u, v \in V$ , 且 $(u, v) \in E$ 。

**考虑 $(u, v)$ 第一次成为关键边时:**  $(u, v)$ 处于增广路径上, 而

**增广路径是最短路径**, 所以有:  $\delta_f(s, v) = \delta_f(s, u) + 1$

**而一旦对流进行增加后**,  $(u, v)$ 就从下一面的残存网络中消失, 直到某一步时从 $u$ 到 $v$ 的流量减小了。此时,  **$(v, u)$ 是增广路径上的边, 并且有正流量**。记此时的流为 $f'$ , 则有

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$$

根据引理26.7,  $\delta_f(s, v) \leq \delta_{f'}(s, v)$ , 所以有

$$\begin{aligned}\delta_{f'}(s, u) &= \delta_{f'}(s, v) + 1 \\ &\geq \delta_f(s, v) + 1 \\ &= \delta_f(s, u) + 2.\end{aligned}$$

**通过上述分析得到：**当 $(u,v)$ 从成为关键边到再次成为关键边，从 $s$ 到 $u$ 的距离至少增加2个单位。

$s$ 到 $u$ 的距离最初至少为0，而 $(u,v)$ 成为增广路径上的边时，这条路径上的中间结点不可能包含 $t$ （ $(u,v)$ 能成为增广路径上的边，意味着 $u \neq t$ ，路径上的中间结点只可能是除 $u$ 和 $t$ 以外的其他结点）。因此，一直到 $u$ 成为不可到达结点之前（最后成为不可达结点的时候意味着流网络中不再有增广路径）， $s$ 到 $u$ 的距离最大是 $|V|-2$

因此，从 $(u,v)$ 第一次成为关键边后算起， $(u,v)$ 最多还能成为关键边的次数至多是  $(|V| - 2)/2 = |V|/2 - 1$

即， $(u,v)$ 能成为关键变的总次数最多为  $|V|/2$

注意到每次流值增加，都会至少有一条关键边，因此流值递增的操作次数至多为  $O(|V||E|)$



最后，由于一共有 $O(E)$ 对结点可以在残存网络中有边彼此相连，**每条边都可能成为关键边。**

根据上面的分析，在Edmonds-Karp算法执行的整个过程中，每条边最多有 $|V|/2$ 次机会成为关键边，所以在算法执行的整个过程中关键边的总数为 $O(VE)$ 。

而每条增广路径至少有一条关键边，每条增广路径意味着要对流网络进行一次流量递增计算，所以Edmonds-Karp算法执行的流量递增操作的次数为 $O(VE)$ 。**证毕。**

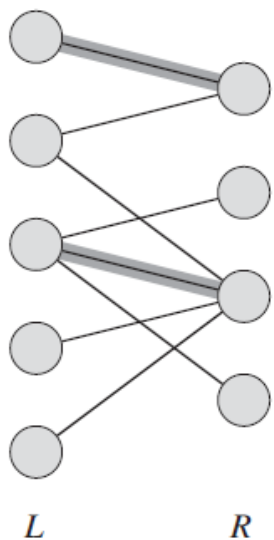


# 最大流算法应用：寻找最大二分匹配

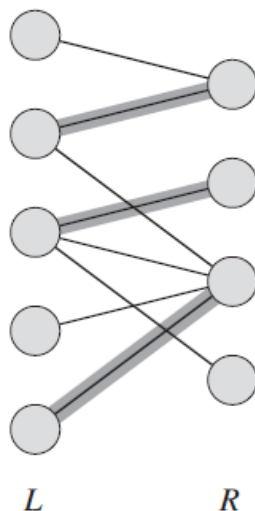
**匹配**：对无向图 $G=(V,E)$ 的一个**匹配是边的一个子集**  $M \subseteq E$ ，使得对于所有的结点 $v$ ，子集 $M$ 中最多有一条边与结点 $v$ 相连。

$M$ 中边的数量称为 $M$ 的**基数**，记为 $|M|$ 。

**最大匹配**：基数最大的匹配。即，如果 $M$ 是一个最大匹配，则对于任意匹配 $M'$ ，有  $|M| \geq |M'|$



一个匹配



最大匹配

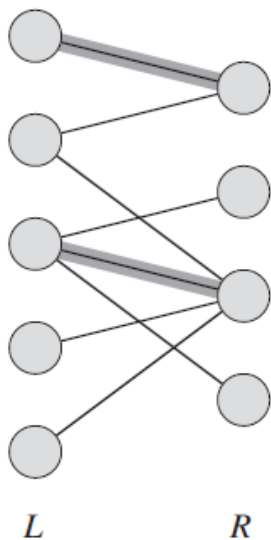
如果子集 $M$ 中的某条边与结点 $v$ 相连，则称结点 $v$ 由 $M$ 所**匹配**；否则，结点 $v$ 就是**没有匹配**的。

## 二分图:

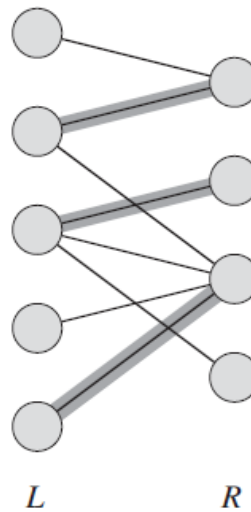
结点集合 $V$ 可以划分为两部分 $L$ 和 $R$ ,  $L \cup R = V, L \cap R = \emptyset$ ,

边集 $E$ 中所有边横跨 $L$ 和 $R$ , 即对于任意的  $(u, v) \in E$ , 有

$$u \in L, v \in R \text{ 或 } v \in L, u \in R$$



二分匹配



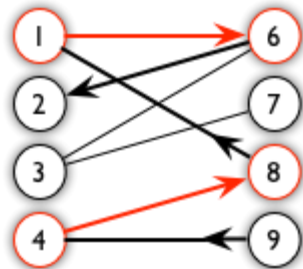
最大二分匹配

**问题: 设计算法在二分图中寻找最大匹配**

## 增广路的定义：

若P是图G中一条连通两个未匹配顶点的路径，并且属于M的边和不属于M的边（即已匹配和待匹配的边）在P上交替出现，则称P为相对于M的一条增广路径。

Example:

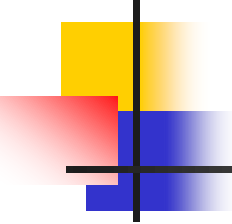


左图的一条增广路径：



推论：

1. P的路径长度必定为奇数，第一条边和最后一条边都不属于M。
2. 将M和P进行异或操作可以得到一个更大的匹配M'。
3. M为G的最大匹配当且仅当不存在M的增广路径。



匈牙利算法框架：

1. 置M为空。
2. 找出一条增广路径P，通过异或操作获得更大的匹配 代替M。
3. 重复（2）操作直到找不出增广路径为止。

复杂度： $O(VE)$

# 求解最大匹配问题的一个算法——匈牙利算法

```
1 #include <iostream>
2 #include <string.h>
3 using namespace std;
4
5 #define MAXN 10 //MAXN表示X集合和Y集合顶点个数的最大值
6 int nx,ny; //x和y集合中顶点的个数
7 int g[MAXN][MAXN]; //邻接矩阵, g[i][j]为1表示有连接
8 int cx[MAXN],cy[MAXN]; //cx[i],表示最终求得的最大匹配中,与x集合中元素xi匹
9 //cy[i],表示最终求得的
10
11 //DFS算法中记录顶点访问状态的数据mk[i]=0表示未访问过,为1表示访问过
12 int mk[MAXN];
13
14 //从集合x中的定顶点u出发,用深度有限的策略寻找增广路
15 //这种增广路只能是当前的匹配数增加1
16 int path(int u){
17     for(int v=0;v<ny;++v){ //考虑所有Yi顶点v
18         if(g[u][v] && !mk[v]){ //Y中顶点v与u邻接,且没有访问过
19             mk[v]=1; //访问v
20
21             //如果v没有匹配,则直接将v匹配给u,如果v已经匹配了,但是从cy[v],也就是
22             //如果第一个条件成立,则不会递归调用
23             if(cy[v]==-1 || path(cy[v])){
24                 cx[u]=v; //把Y中v匹配给x中u
25                 cy[v]=u; //把x中u匹配给Y中v
26                 return 1;
27             }
28         }
29     }
30     return 0; //如果不存在从u出发的增广路,则返回0
31 }
32
33 int maxMatch(){ //求二分图最大匹配的匈牙利算法
34     int res=0;
35     memset(cx,-1,sizeof(cx)); //从0匹配开始增广,将cx和
36     memset(cy,-1,sizeof(cy));
37     for(int i=0;i<nx;++i){
38         if(cx[i]==-1){
39             memset(mk,0,sizeof(mk));
40             res+=path(i);
41         }
42     }
43     return res;
44 }
45
46 int main() {
47     nx=3;
48     ny=4;
49     g[0][0]=0; g[0][1]=1; g[0][2]=1; g[0][3]=0;
50     g[1][0]=0; g[1][1]=1; g[1][2]=0; g[1][3]=0;
51     g[2][0]=1; g[2][1]=0; g[2][2]=1; g[2][3]=1;
52
53     int num= maxMatch();
54     cout<<"num="<<num<<endl;
55     for(int num=0;num<3;++num){
56         cout<<"cx["<<num+1<<"] -> "<<cx[num]+1<<endl;
57     }
58     cout << "!!!Hello World!!!" << endl; // prints !!!Hello
59     return 0;
60 }
61
62 }
```

# 寻找最大二分匹配

**基本的思想**：构建一个流网络，将寻找最大二分匹配问题转化为求流网络的最大流问题，**流对应于匹配**，然后用Ford-Fulkerson方法寻找最大二分匹配

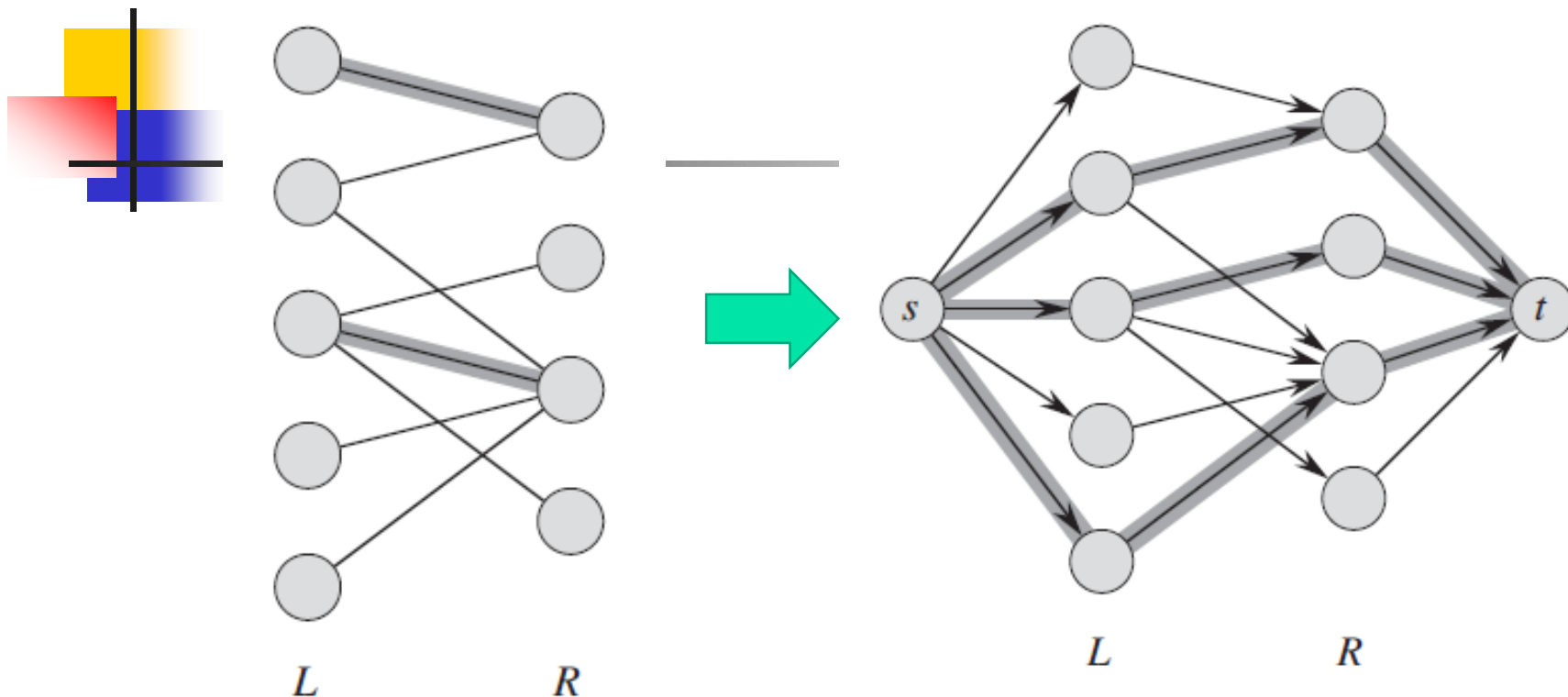
**转化**：将二分图G所对应的流网络  $G' = (V', E')$  定义如下

- **新增源结点s和汇点t，令  $V' = V \cup \{s, t\}$**
- **新增s到L中所有结点的边和R中所有结点到t的边，令**

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

- **定义每条边上的容量为单位容量，即对任意  $(u, v) \in E'$ ，  
 $c(u, v) = 1$**

# 一个二分图转化为流网络的例子：



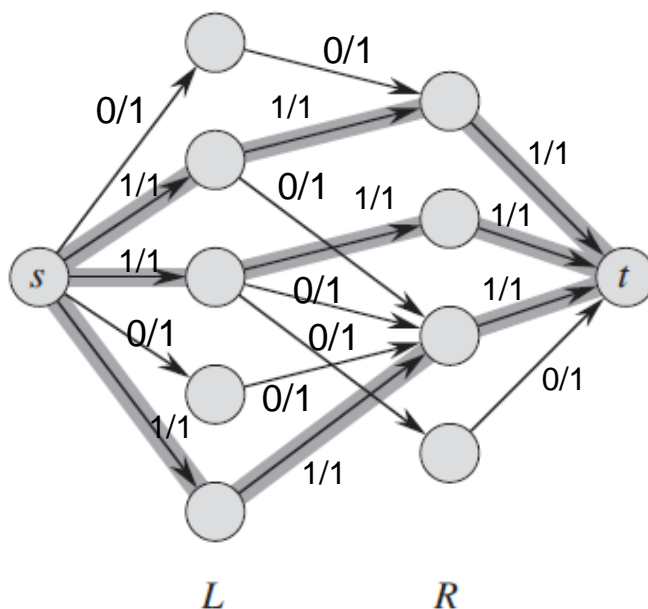
- 不失一般性，假定结点集  $V$  中的每个结点至少有一条相连的边，  
 $|E| \geq |V|/2$ ，则有  $|E| \leq |E'| = |E| + |V| \leq 3|E|$   
 所以  $|E'| \in \Theta(|E|)$

# 寻找最大二分匹配算法

上述转化中，经过赋值， $G'$  中所有边的容量为整数值1，所以之后计算所得的流值也将是整数。

## 算法思路：

利用Ford-Fulkerson算法求得 $G'$  中的最大流。流值大于0且在原图中的边将构成最大匹配，而最大匹配的边数就是最大流的流值。



最大匹配=3



## 转化的正确性证明分三步：

- (1) 证明原图和转化后的流网络中匹配和流一一对应，并且匹配的边数对应于流值。
- (2) 证明在容量是整数前提限制下，Ford-Fulkerson方法产生的流是整值流，从而保证算法计算的流可以还原到原图的匹配。
- (3) 证明最大流的流值等于最大匹配的基数。

# (1) 证明原图和转化后的流网络中，匹配和流一一对应，并且匹配的边数对应于流值

引理26.9：如果 $M$ 是 $G$ 中的一个匹配，则流网络 $G'$ 中存在一个整数值的流 $f$ ，使得 $|f|=|M|$ 。反之，如果 $f$ 是 $G'$ 中的一个整数流，则 $G$ 中存在一个匹配 $M$ ，使得 $|M|=|f|$ 。

证明：假定 $M$ 是 $G$ 中匹配，定义 $G'$ 中对应的流 $f$ ：

如果 $(u, v) \in M$ ， $f(s, u) = f(u, v) = f(v, t) = 1$ ；  
所有其它属于 $E'$ 的边 $(u, v)$ ， $f(u, v) = 0$ 。

- (1) 可以验证 $f$ 满足容量限制和流量守恒性质（自行验证）。
- (2) 能否说明 $|M|=|f|$ ？

**$|M| = |f|$  ?**

**直观上理解：**  $(L \cup \{s\}, R \cup \{t\})$  是一个切割， $M$  中的边恰好是横跨该切割的边，其上是一个单位的流量。所以该切割的净流值就是  $f$  的流值，而切割的净流值等于匹配的边数。

所以得到  $|f| = |M|$ 。

**如何证明呢？**

假定 $f$ 是 $G'$  中如上所定义的一个整数值流, 并设

$$M = \{(u, v) : u \in L, v \in R, \text{ and } f(u, v) > 0\}$$

根据 $G'$  的构造, 每个结点 $u \in L$  只有一条进入的边, 即 $(s, u)$ , 其容量为1:  $u$ 有一个单位的流入, 根据能量守恒性质, 就必有一个点位的流出。由于 $f$ 是整数值的流, 所以 $u$ 不仅最多只能从一个单边流入1单位流量, 而且也只能最多从一条边流出, 即:

**1单位的流进入结点 $u$ 当且仅当恰好存在一个结点 $v \in R$ , 使得 $f(u, v) = 1$ , 并且在离开 $u$ 的边中最多有一条出边带有正值的流。**

同样的讨论可应用于 $v \in R$ ,  **$v$ 最多有一条带有正值流的入边。**

**所以 $M$ 是一个匹配** (即对于所有的结点 $v$ ,  $M$ 中最多有一条边与之相连)。

从而可有 $|M|=|f|$ 。

**可证明如下：**

根据 $f$ 的定义，对每个匹配的结点 $u \in L$ ，有 $f(s, u)=1$ ，而对于每条边 $(u, v) \in E-M$ ，有 $f(u, v)=0$ 。

因此，横跨切割 $(L \cup \{s\}, R \cup \{t\})$ 的净流量 $f(L \cup \{s\}, R \cup \{t\})$ 等于 $|M|$ 。根据引理26.4，流的值 $|f|=|M|$ 。证毕

## (2) 证明在容量限制是整数的前提下, Ford-Fulkerson方法产生的流是整数值的流, 从而保证算法计算的流可以还原到原图的匹配。

- 注: 如果最大流算法返回流网络 $G'$  中的一个非整数流 $f(u,v)$  (即使流的值 $|f|$ 本身是整数), 上述讨论将存在问题。但**定理26.10**说明这种情况不会发生。

**定理26.10 (完整性定理)** 如果容量函数 $c$ 只能取整数值, 则Ford-Fulkerson方法所生成的最大流 $f$ 满足 $|f|$ 是整数值的性质。而且, 对于所有的结点 $u$ 和 $v$ ,  $f(u,v)$  的值都是整数。

**证明:** 可以通过对迭代次数的归纳进行证明, 留作练习。

### (3) 证明最大流的流值等于最大匹配的基数


**推论26.11** 二分图 $G$ 的一个最大匹配 $M$ 的边数等于其对应的流网络 $G'$ 中某一最大流 $f$ 的值。

**证明：**用反证法证明

假定 $M$ 是图 $G$ 中的一个最大匹配，但其相应的流网络 $G'$ 中的流 $f$ 不是最大流。那么 $G'$ 中存在一个最大流 $f'$ ，满足 $|f'| > |f|$ 。

由于 $G'$ 的容量都是整数值，根据定理26.10， $f'$ 的值也是整数值。同时， $f'$ 有一个对应的匹配 $M'$ ，使得 $|M'| = |f'| > |f| = |M|$ ，这与 $M$ 是最大匹配相矛盾。

同理可证，如果 $f$ 是 $G'$ 中的一个最大流，则其对应的匹配是 $G$ 的一个最大匹配。

A decorative graphic consisting of overlapping yellow, red, and blue squares is positioned in the top left corner.

至此，对给定的一个二分无向图 $G$ ，可以通过创建对应的流网络 $G'$ ，并在其上运行Ford-Fulkerson方法来找到 $G$ 的一个最大匹配。

最大匹配 $M$ 可以直接从找到的整数最大流 $f$ 获得，这一过程的时间复杂度是 $O(VE)$ 。



# 抢劫银行

Nieuw Knollendam是一个非常现代的小镇。当查看其地图布局时，这已经很清楚，它只是街道和大道的矩形网格。作为重要的贸易中心，Nieuw Knollendam也拥有许多银行。几乎在每个交叉口上都发现了一个银行（尽管在同一交叉口上从来没有两个银行）。不幸的是，这吸引了很多罪犯。持仓非常普遍，通常一天有几家银行被抢劫。这不仅对银行，而且对罪犯也已成为一个问题。抢劫银行后，劫匪试图尽快离开该镇，大多数时候都被警察高速追赶。有时，两个正在奔跑的罪犯经过同一路口，造成多种风险：例如碰撞。

为了防止这些不愉快的情况，劫匪同意共同协商。他们每个星期六晚上见面，并为下一周制定时间表：谁将在哪一天抢劫哪家银行。他们每天试图计划逃生路线，以至于没有两条路线使用相同的过境点。尽管他们认为应该存在这样的规划，但他们并未成功地根据此条件规划路线。给定 $(s \times a)$ 的网格以及要抢劫的银行所在的交叉口，找出是否有可能计划从每个被抢劫的银行到城市边界的逃生路线，而无需使用多次使用某一个路口。

## 输入

输入的第一行包含要解决的问题 $p$ 的数量。•每个问题的第一行包含街道数量 $s$ （ $1 \leq s \leq 50$ ），然后是大道数量 $a$ （ $1 \leq a \leq 50$ ），然后是要抢劫的银行数量 $b$ （ $b \geq 1$ ）。•然后出现 $b$ 行，每行包含两个数字 $x$ （街道编号）和 $y$ （大街编号）形式的银行位置。显然， $1 \leq x \leq s$ 和 $1 \leq y \leq a$ 。

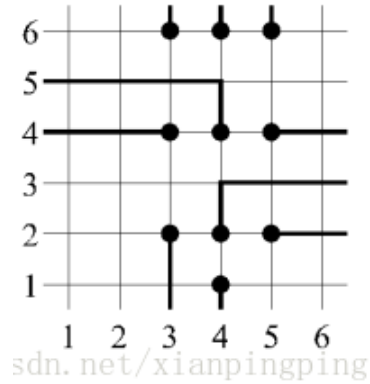
## 输出

输出文件由 $p$ 行组成。每行包含“possible”或“impossible”的文字。如果有可能计划非穿越式的出行路线，则此行应包含“possible”一词。如果不可能，则该行应包含“impossible”字样。

# 抢劫银行

## Sample Input

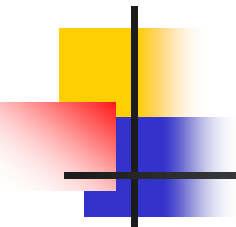
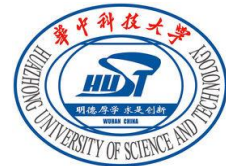
2  
6 6 10  
4 1  
3 2  
4 2  
5 2  
3 4  
4 4  
5 4  
3 6  
4 6  
5 6  
5 5 5  
3 2  
2 3  
3 3  
4 3  
3 4



## Sample Output

possible  
not possible

# 抢劫银行

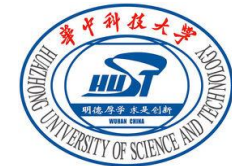


我们发现唯一限制条件为**路线不能相交**，即每个点只能用一次。而且注意到边长很小。所以可以自然想到网络流。至于每个点只能用一次，可以**限制点的容量为1**，拆点即可。把一个点拆成2个，一个进，一个出。在两个点之间建一条容量为1的边。

## 建图：

建一个超级源点和超级终点。建边顺序为从超级源点建立一条到银行的边，容量为1；把一个点拆成2个，之间连容量为1的边；每个点向四周所有点建边，容量为1；如果该点在边界上，那么就向超级终点建边。然后跑一遍最大流即可。如果最大流为N，就说明满足要求。

```
89 int main()
90 {
91     int T;
92     T=read();
93     while(T-->0)
94     {
95         int r,c,n;
96         cnt=1;
97         memset(e,0,sizeof(e));
98         memset(head,0,sizeof(head)); //每次初始化
99         r=read();c=read();n=read();
100        s=r*c*2+1; //超级源点
101        t=r*c*2+2; //超级终点
102        for(int i=1;i<=n;i++)
103        {
104            int x,y;
105            x=read();y=read();
106            add(s,(x-1)*c+y,1); //从超级源点到 银行
107        }
108        for(int i=1;i<=r;i++)
109            for(int j=1;j<=c;j++)
110            {
111                int k=(i-1)*c+j;
112                add(k,k+r*c,1); //拆点，限制点的容量
113                for(int d=0;d<4;d++)
114                {
115                    int tx=i+dx[d];
116                    int ty=j+dy[d];
117                    if(tx<1 || tx>r || ty<1 || ty>c)
118                        add(k+r*c,t,1); //如果该点在边界上，连到超级终点
119                    else
120                        add(k+r*c,(tx-1)*c+ty,1); //向四周建边
121                }
122            }
123        if (flow()==n) //最大流等于监狱数，即满足要求
124            cout<<"possible"<<endl;
125        else cout<<"not possible"<<endl;
126    }
127    return 0;
128 }
```



lxhgww最近迷上了一款游戏，在游戏里，他拥有很多的装备，每种装备都有2个属性，这些属性的值用[1,10000]之间的数表示。当他使用某种装备时，他只能使用该装备的某一个属性。并且每种装备最多只能使用一次。游戏进行到最后，lxhgww遇到了终极boss，这个终极boss很奇怪，攻击他的装备所使用的属性值必须从1开始连续递增地攻击，才能对boss产生伤害。也就是说一开始的时候，lxhgww只能使用某个属性值为1的装备攻击boss，然后只能使用某个属性值为2的装备攻击boss，然后只能使用某个属性值为3的装备攻击boss.....以此类推。现在lxhgww想知道他最多能连续攻击boss多少次？

## 输入格式

输入的第一行是一个整数N，表示lxhgww拥有N种装备接下来N行，是对这N种装备的描述，每行2个数字，表示第i种装备的2个属性值

## 输出格式

输出一行，包括1个数字，表示lxhgww最多能连续攻击的次数。

## 输入输出样例

输入 #1

复制

```
3
1 2
3 2
4 5
```

输出 #1

复制


```
2
```

## 说明/提示

Limitation

对于30%的数据，保证 $N \leq 1000$

对于100%的数据，保证 $N \leq 1000000$

A decorative graphic consisting of overlapping yellow, red, and blue squares is positioned in the top left corner.

考虑左边点表示每一个颜色,  
右边点表示每一件装备。  
如果第 $i$ 件装备的伤害是 $a_i, b_i$ ,  
就把左边的 $a_i$  和 $b_i$ 两点向右边  
的 $i$ 点连边。然后二分图匹  
配,找到第一个无法匹配的点  
即可。

因为二分图匹配匹配一个点  
时不会使之前有匹配的点变  
成没有匹配的,所以正确性显  
然有保证。  
直接上匈牙利即可。

```
#include<bits/stdc++.h>
using namespace std;
int n, a, b, mx, mt[1000010], vis[1000010], tg;
vector<int>e[10010];
int Find(int x) { //匈牙利算法
    vis[x]=tg;
    for(int i=0; i<e[x].size(); i++) {
        if(!mt[e[x][i]]) {
            mt[e[x][i]]=x;
            return 1;
        }
        if(vis[mt[e[x][i]]]!=tg&&Find(mt[e[x][i]])) {
            mt[e[x][i]]=x;
            return 1;
        }
    }
    return 0;
}
int main() {
    scanf("%d", &n);
    for(int i=1; i<=n; i++) {
        scanf("%d%d", &a, &b);
        mx=max(mx, max(a, b));
        e[a].push_back(i);
        e[b].push_back(i);
    }
    for(int i=1, tg=1; i<=mx; tg++, i++) {
        if(!Find(i)) {
            printf("%d", i-1);
            return 0;
        }
    }
    printf("%d", mx); //记得考虑所有点都匹配的情况
    return 0;
}
```

# 最大流总结

1. 基本概念：**流网络**，**流**，**最大流**等；
2. 解决最大流问题的**Ford-Fulkerson方法**，相关概念有**残存网络**、**增广路径**以及理论基础**最大流最小切割定理**；
3. **Ford-Fulkerson算法**和**Edmonds-Karp算法**，相关性质和证明；
4. 最大流算法的应用：解决**最大二分匹配问题**。

作业：26.1-1, 26.2-3, 26.3-1