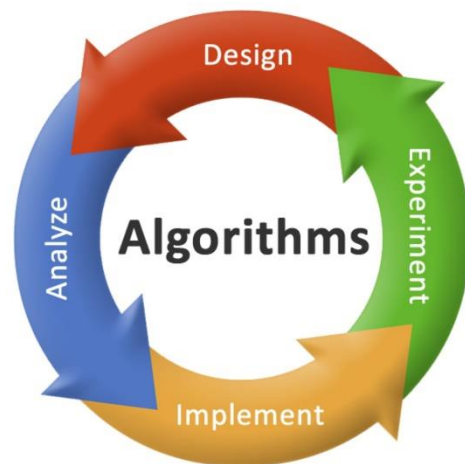


华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY



Algorithm Design and Analysis

算法设计与分析

Chapter 6: Network Flow

张乾坤

课程提要

- 最大流和最小割问题
- Ford- Fulkerson 算法
- 最大流最小割定理
- 容量缩放算法
- 最短增广路经

课程提要

- 最大流和最小割问题
- Ford- Fulkerson 算法
- 最大流最小割定理
- 容量缩放算法
- 最短增广路经

流网络

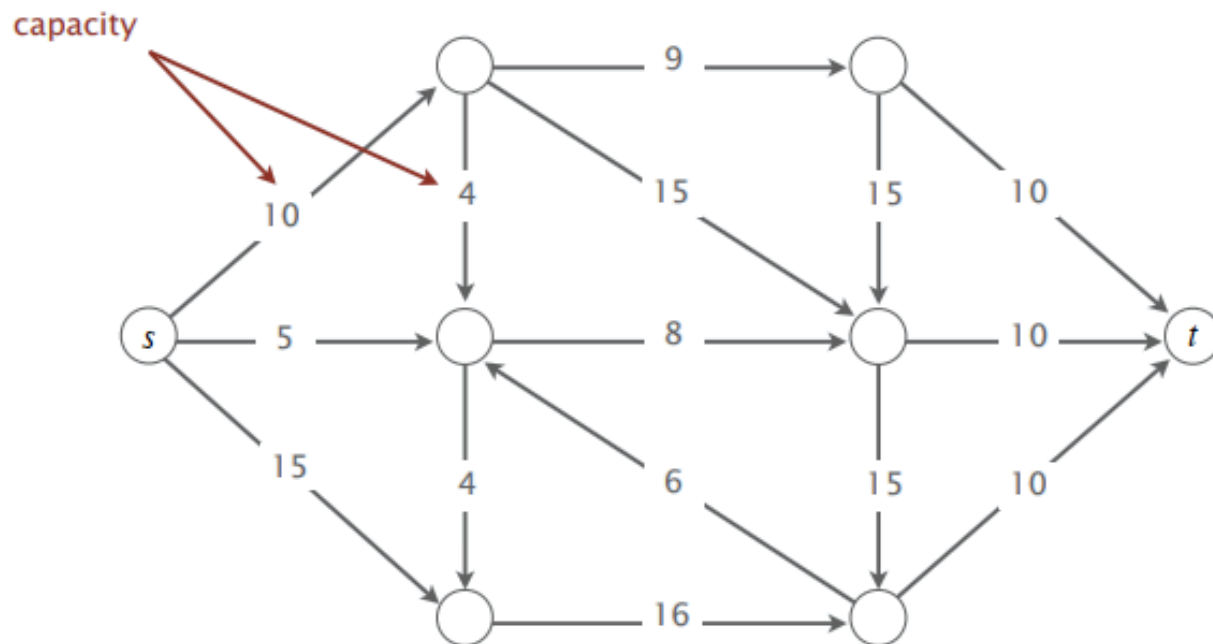
流网络 (flow network) 是一个五元组 $G = (V, E, s, t, c)$

- 有向图 (V, E) 中, 源节点 $s \in V$, 汇节点 $t \in V$

- 容量 $c(e) \geq 0$, 对任意 $e \in E$ 成立

假设所有节点都可从 s 到达

应用: 材料流经运输网络, 来自于源节点, 并被送往汇节点

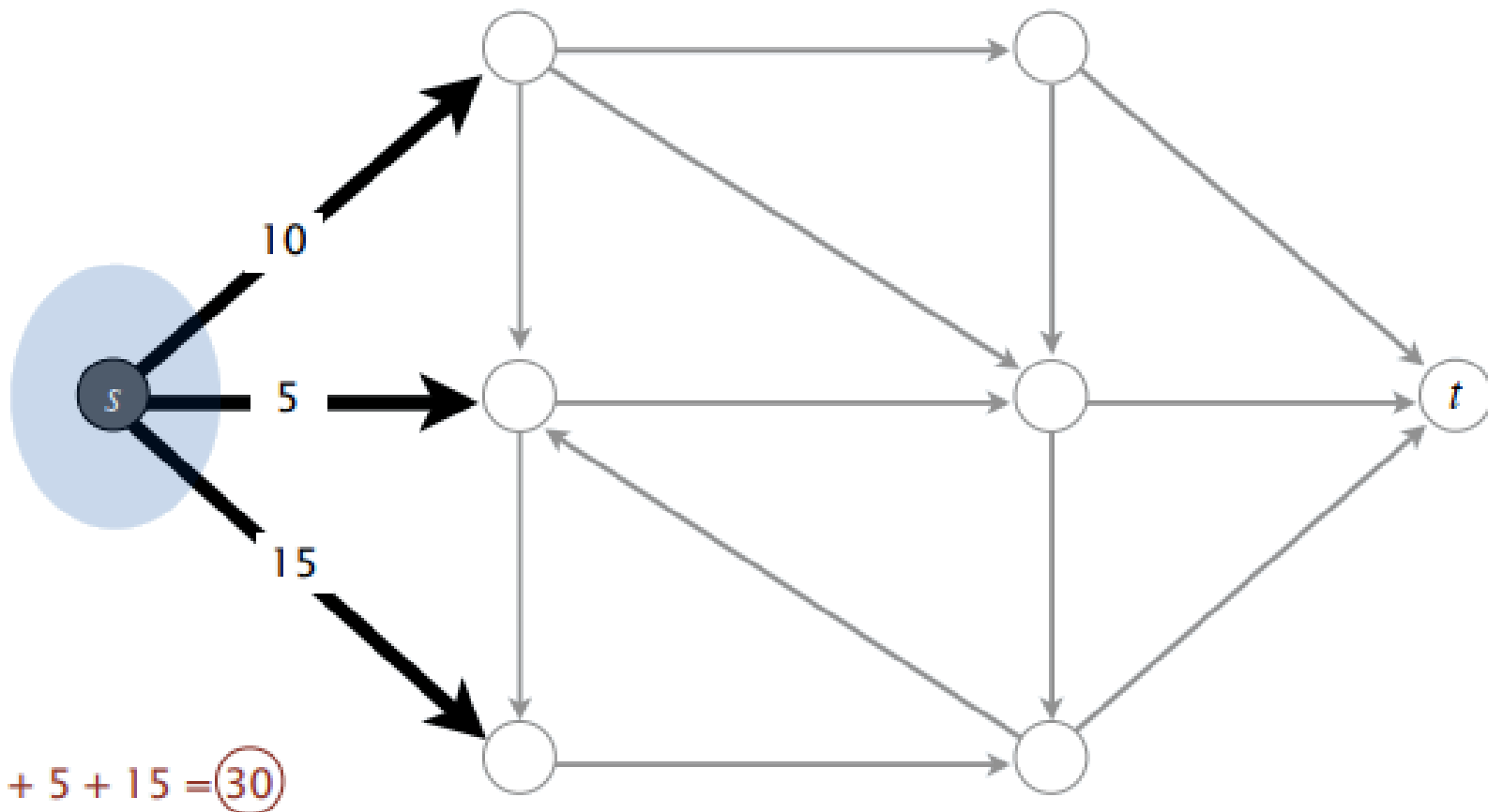


最小割问题

定义: st 割(cut)是一个划分 (A, B) , 其中 $s \in A$, $t \in B$

定义: 割的容量是从 A 到 B 的所有边的容量之和

$$cap(A, B) = \sum_{e \text{ 离开 } A} c(e)$$

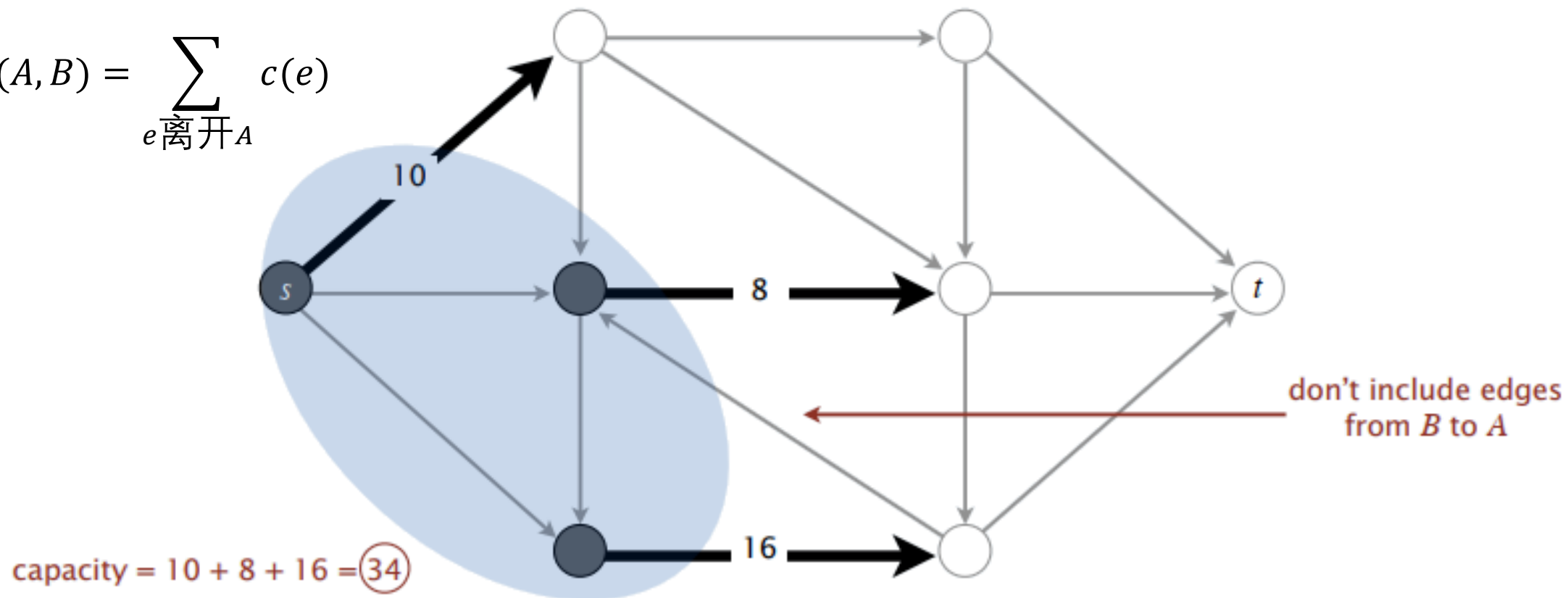


最小割问题

定义: st 割(cut)是一个划分 (A, B) , 其中 $s \in A$, $t \in B$

定义: 割的容量是从 A 到 B 的所有边的容量之和

$$cap(A, B) = \sum_{e \text{ 离开 } A} c(e)$$



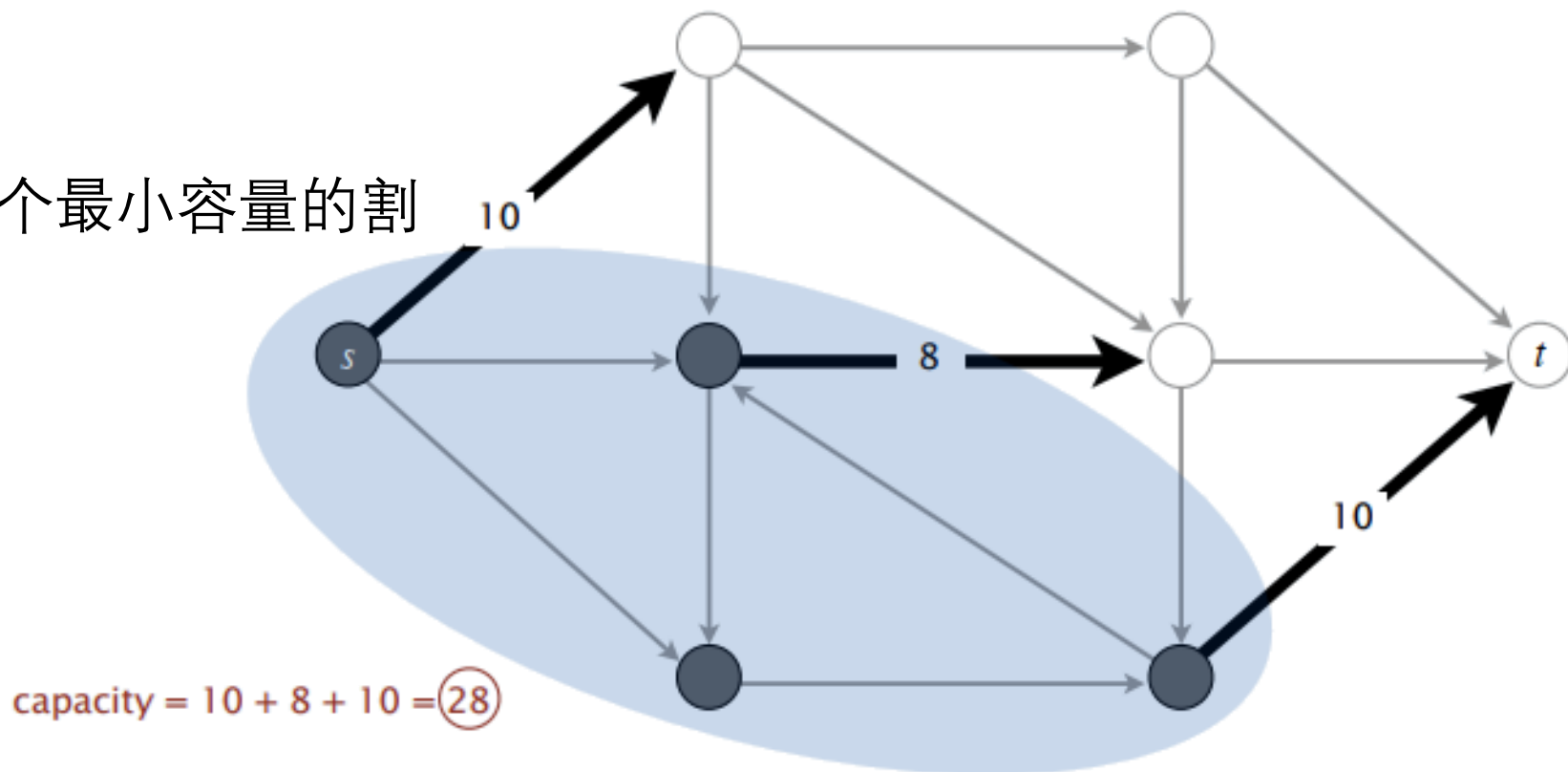
最小割问题

定义: st 割(cut)是一个划分 (A, B) , 其中 $s \in A$, $t \in B$

定义: 割的容量是从 A 到 B 的所有边的容量之和

$$cap(A, B) = \sum_{e \text{ 离开 } A} c(e)$$

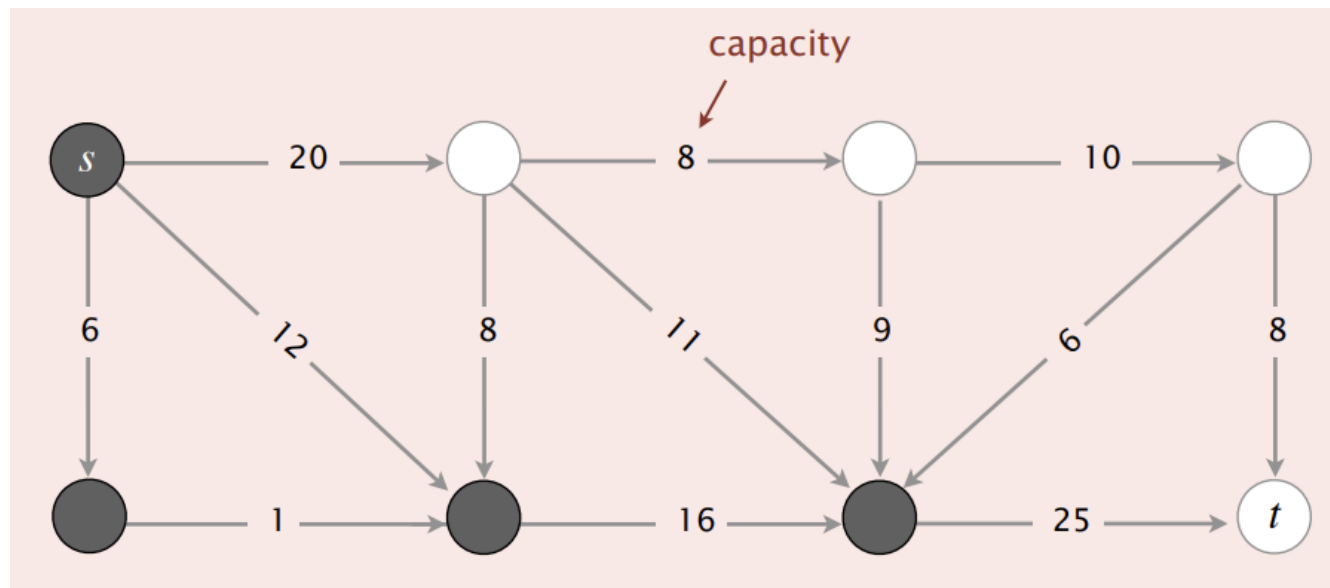
最小割问题: 找到一个最小容量的割



网络流： 问题 1

下列哪个选项是给定st割的容量？

- A.** 11 ($20 + 25 - 8 - 11 - 9 - 6$)
- B.** 34 ($8 + 11 + 9 + 6$)
- C.** 45 ($20 + 25$)
- D.** 79 ($20 + 25 + 8 + 11 + 9 + 6$)



最大流问题

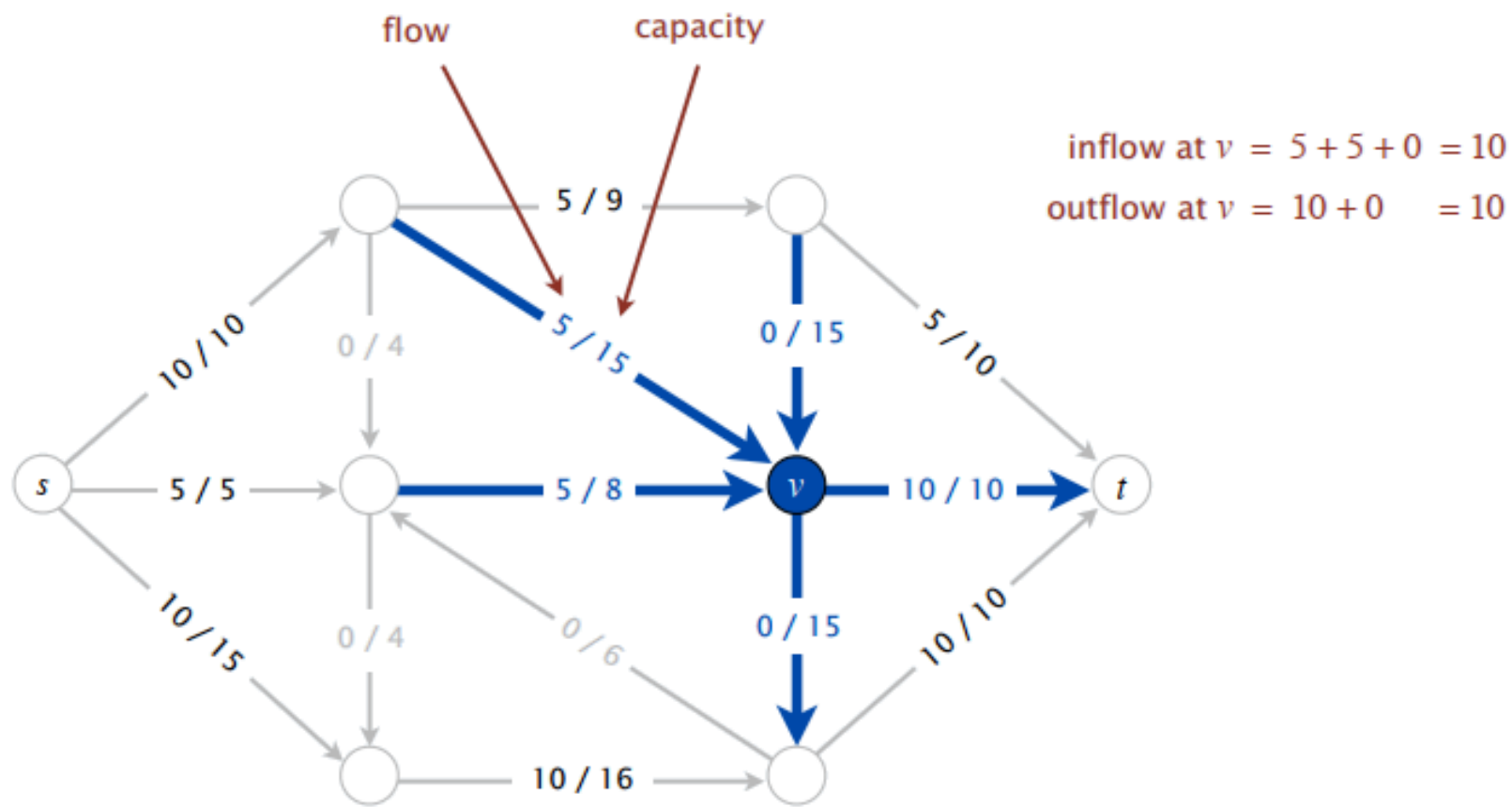
定义: st 流 (flow) f 是一个函数, 满足:

- 对每一条边 $e \in E$, $0 \leq f(e) \leq c_e$

[容量条件]

- 对每一个节点 $v \in V - \{s, t\}$, $\sum_{e \text{ 进入 } v} f(e) = \sum_{e \text{ 离开 } v} f(e)$

[守恒条件]



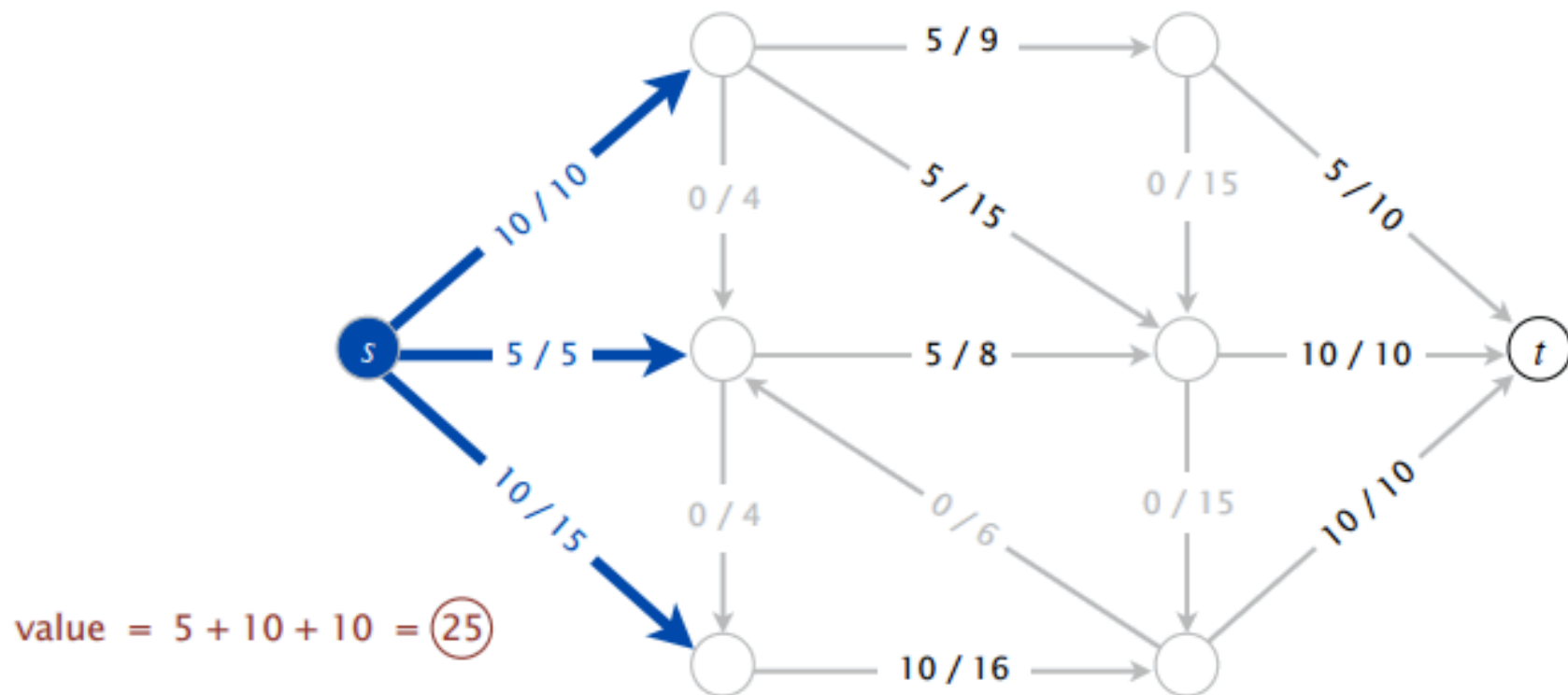
最大流问题

定义: st 流 (流) f 是一个函数, 满足:

• 对每一条边 $e \in E$, $0 \leq f(e) \leq c_e$ [容量条件]

• 对每一个节点 $v \in V - \{s, t\}$, $\sum_{e \text{ 进入 } v} f(e) = \sum_{e \text{ 离开 } v} f(e)$ [守恒条件]

定义: 流 f 的值 (value): $val(f) = \sum_{e \text{ 离开 } s} f(e) - \sum_{e \text{ 进入 } s} f(e)$



最大流问题

定义: st 流 (流) f 是一个函数, 满足:

- 对每一条边 $e \in E$, $0 \leq f(e) \leq c_e$

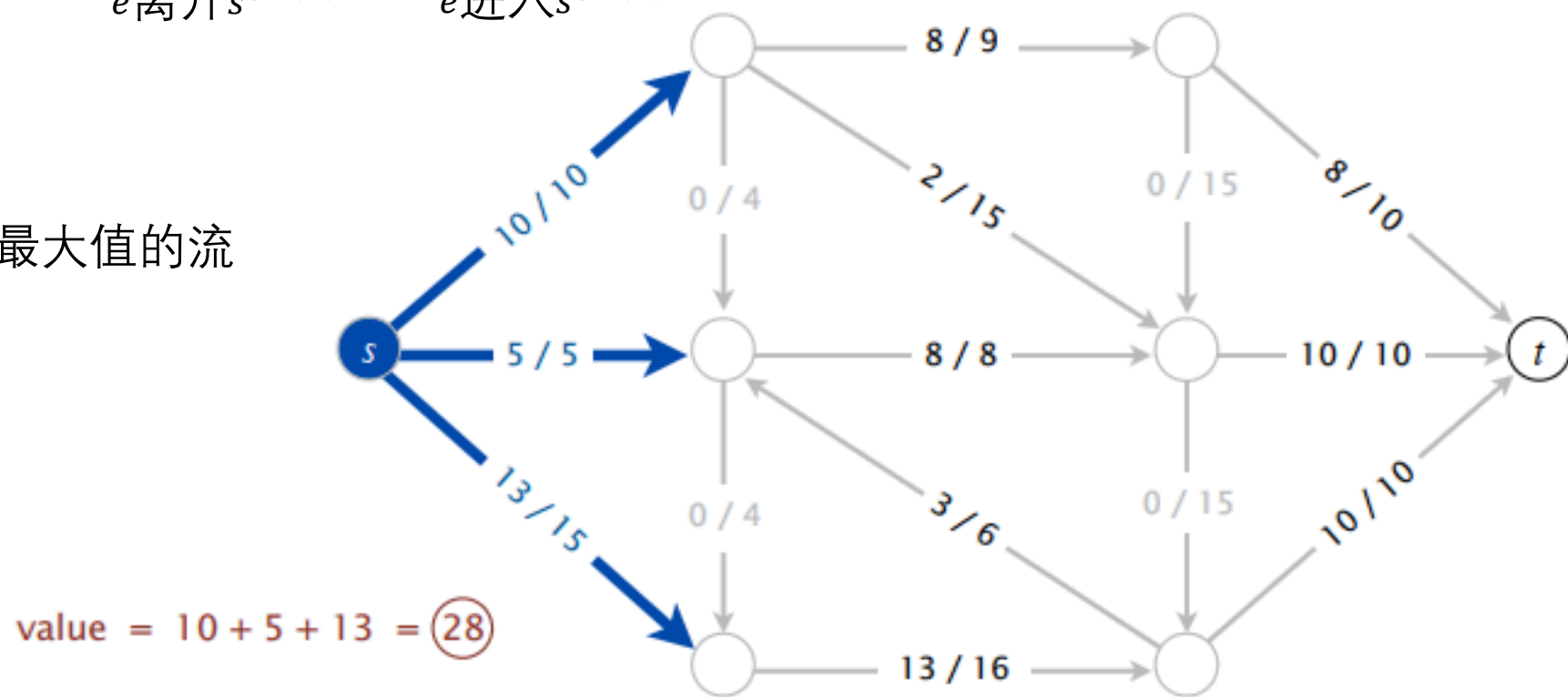
[容量条件]

- 对每一个节点 $v \in V - \{s, t\}$, $\sum_{e \text{ 进入 } v} f(e) = \sum_{e \text{ 离开 } v} f(e)$

[守恒条件]

定义: 流 f 的值: $val(f) = \sum_{e \text{ 离开 } s} f(e) - \sum_{e \text{ 进入 } s} f(e)$

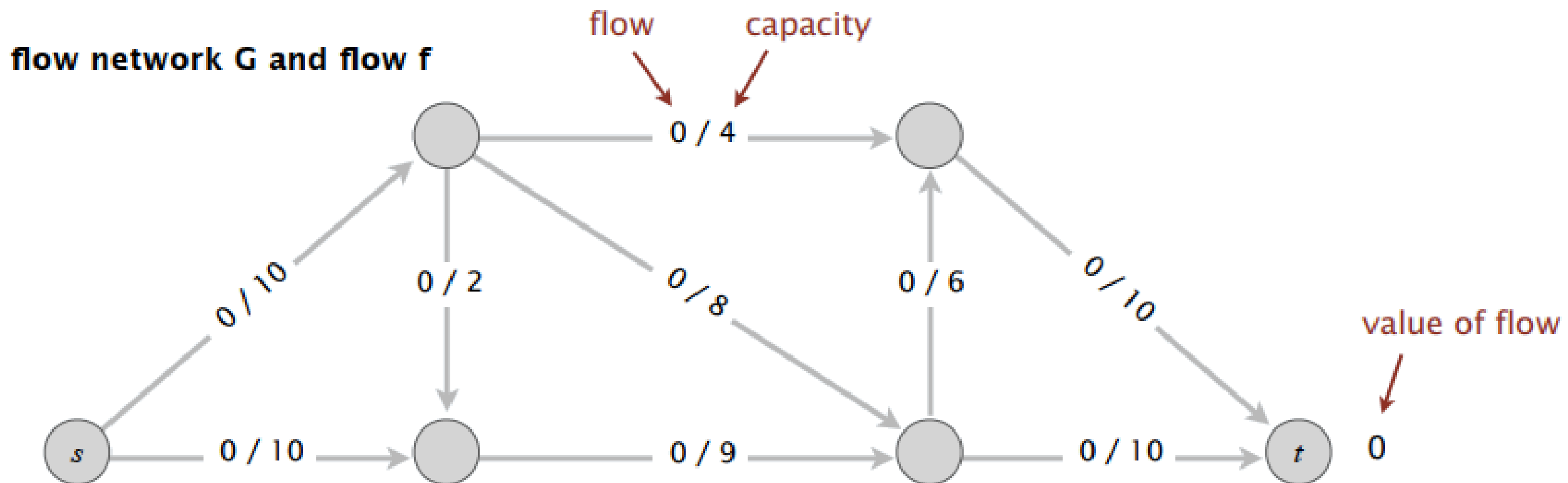
最大流问题: 找到一个最大值的流



课程提要

- 最大流和最小割问题
- Ford- Fulkerson 算法
- 最大流最小割定理
- 容量缩放算法
- 最短增广路经

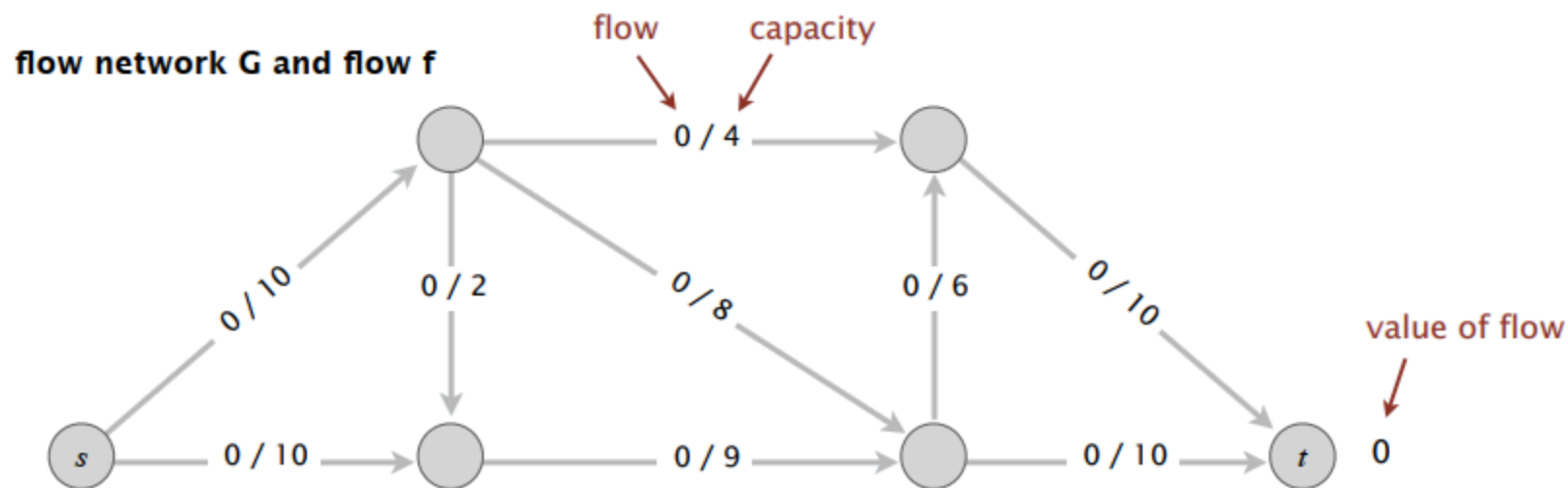
实现最大流算法



实现最大流算法

贪心算法:

- 对每一条边 $e \in E$, 从 $f(e) = 0$ 开始

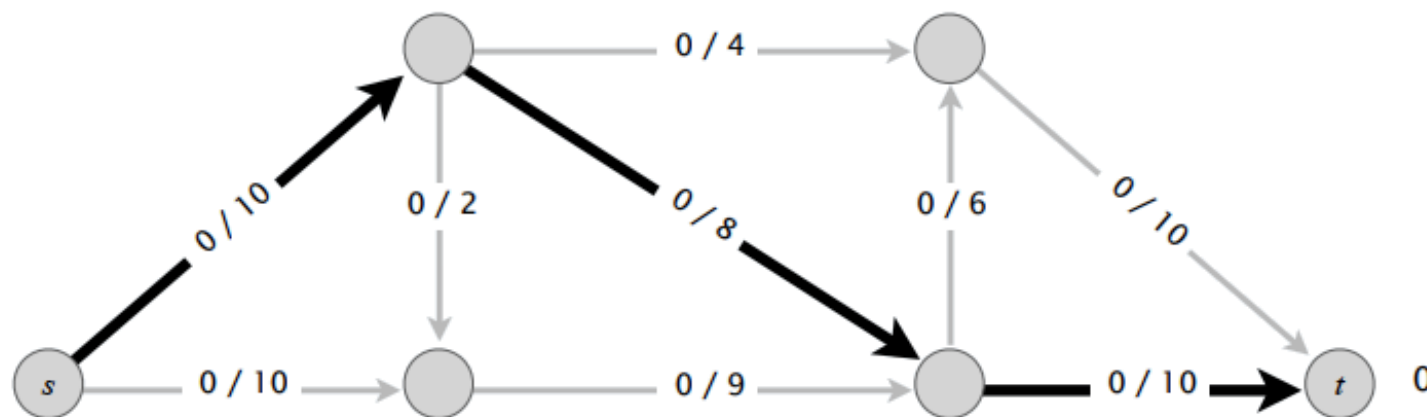


实现最大流算法

贪心算法：

- 对每一条边 $e \in E$ ，从 $f(e) = 0$ 开始
- 找到一条 $s \rightsquigarrow t$ 路径 P ，其中，每条边满足 $f(e) < c(e)$

flow network G and flow f

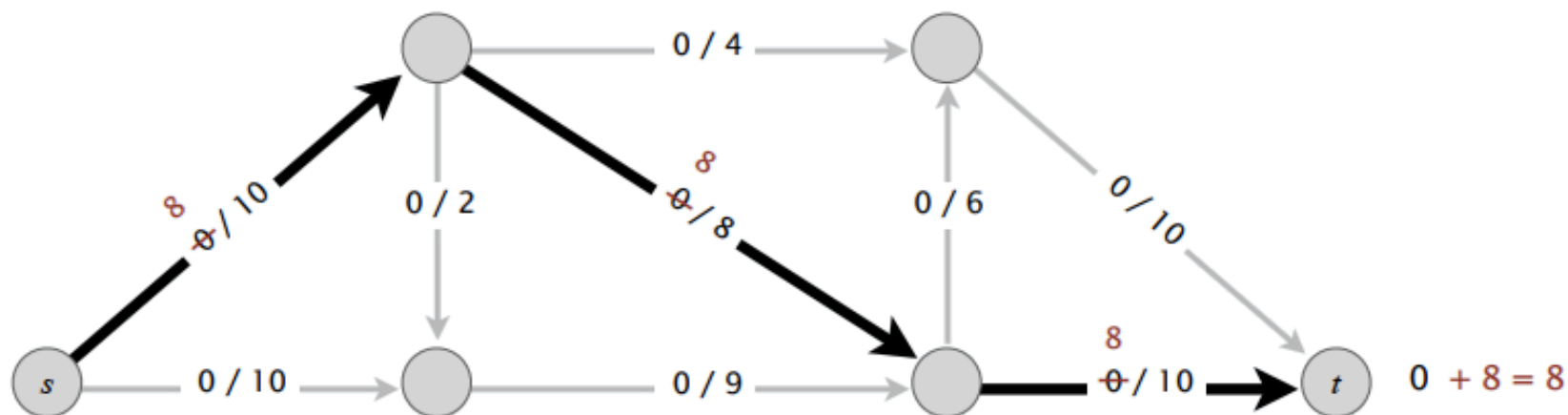


实现最大流算法

贪心算法：

- 对每一条边 $e \in E$ ，从 $f(e) = 0$ 开始
- 找到一条 $s \rightsquigarrow t$ 路径 P ，其中，每条边满足 $f(e) < c(e)$
- 沿着路径 P 增加流量

flow network G and flow f

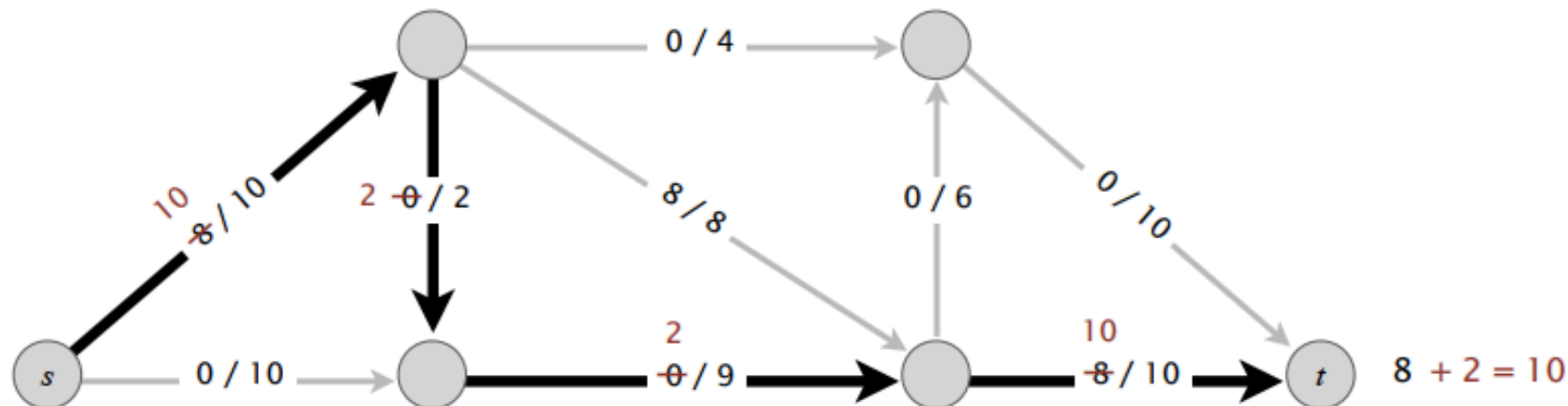


实现最大流算法

贪心算法:

- 对每一条边 $e \in E$, 从 $f(e) = 0$ 开始
- 找到一条 $s \rightsquigarrow t$ 路径 P , 其中, 每条边满足 $f(e) < c(e)$
- 沿着路径 P 增加流量
- **重复进行, 直到阻塞**

flow network G and flow f

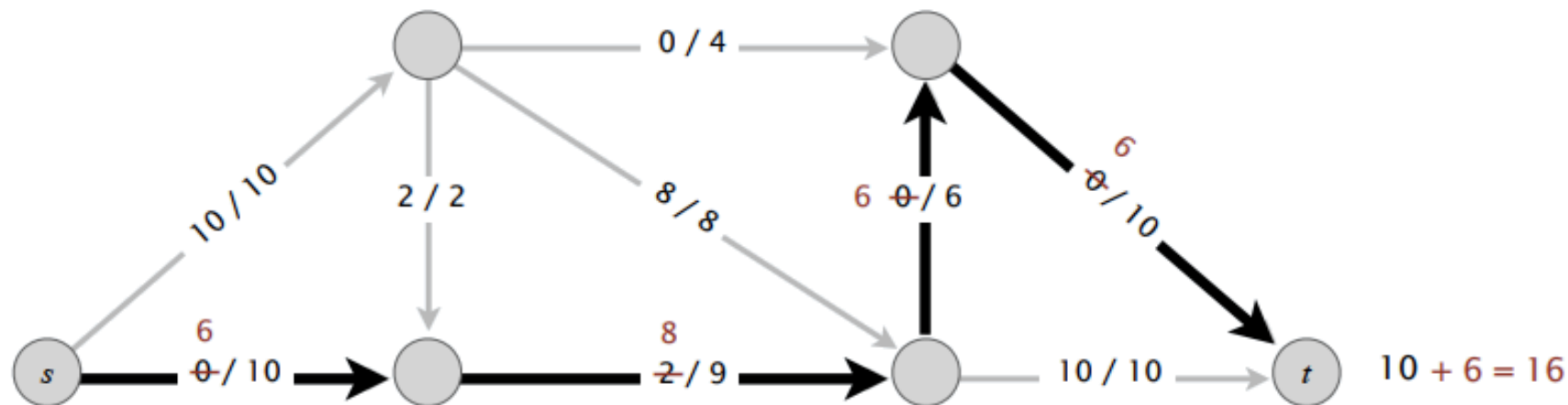


实现最大流算法

贪心算法:

- 对每一条边 $e \in E$, 从 $f(e) = 0$ 开始
- 找到一条 $s \rightsquigarrow t$ 路径 P , 其中, 每条边满足 $f(e) < c(e)$
- 沿着路径 P 增加流量
- **重复进行, 直到阻塞**

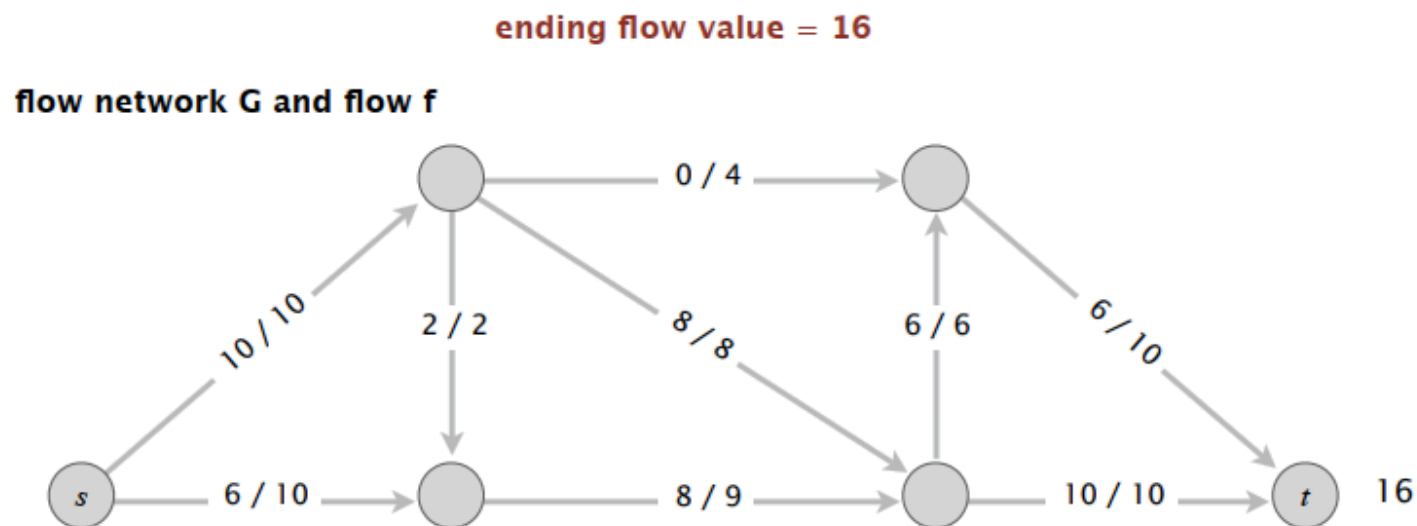
flow network G and flow f



实现最大流算法

贪心算法:

- 对每一条边 $e \in E$, 从 $f(e) = 0$ 开始
- 找到一条 $s \rightsquigarrow t$ 路径 P , 其中, 每条边满足 $f(e) < c(e)$
- 沿着路径 P 增加流量
- 重复进行, 直到阻塞



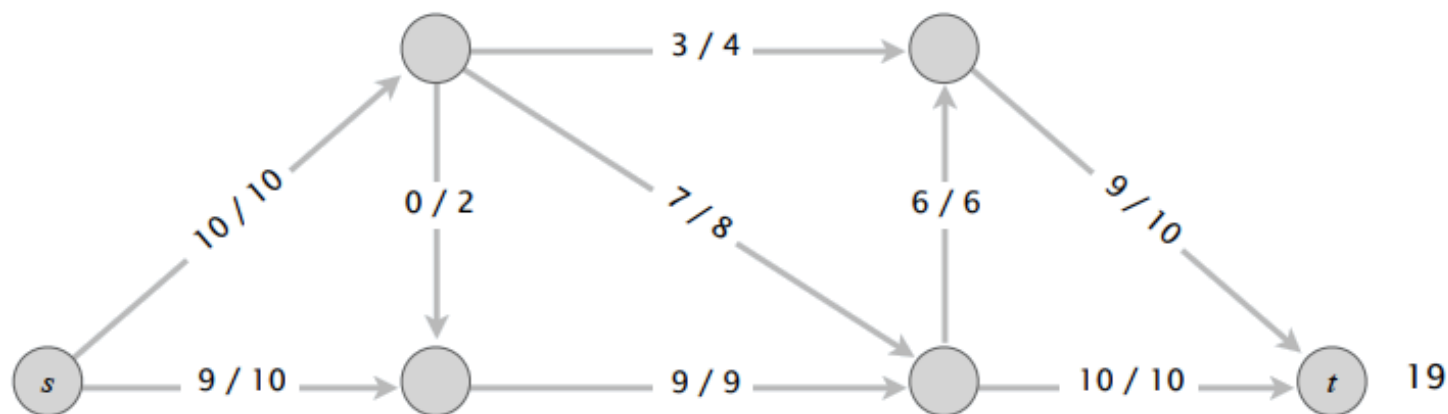
实现最大流算法

贪心算法:

- 对每一条边 $e \in E$, 从 $f(e) = 0$ 开始
- 找到一条 $s \rightsquigarrow t$ 路径 P , 其中, 每条边满足 $f(e) < c(e)$
- 沿着路径 P 增加流量
- 重复进行, 直到阻塞

but max-flow value = 19

flow network G and flow f



为什么贪心算法会失败

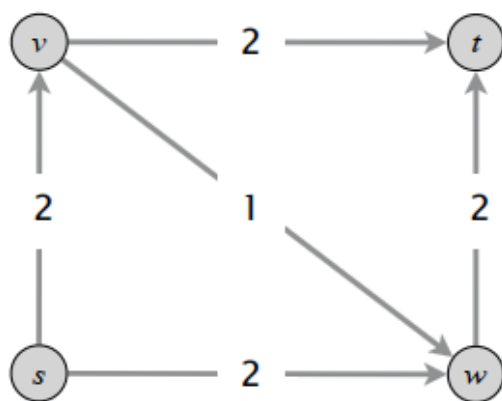
Q: 为什么贪心算法会失败?

A: 一旦贪心算法增加了一条边上的流量, 就不会再减少

Ex: 考虑流网络 G

- 唯一的最大流 f^* , 满足: $f^*(v, w) = 0$
- 贪心算法选择 $s \rightarrow v \rightarrow w \rightarrow t$, 作为第一条路径

flow network G



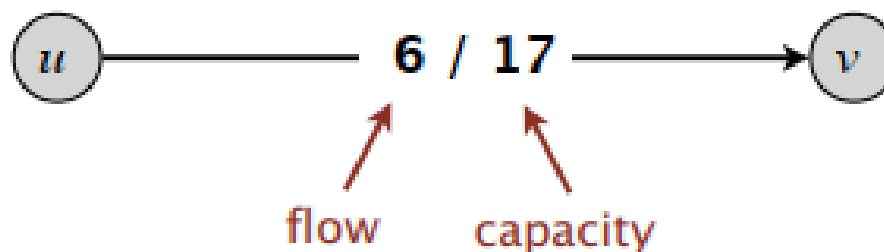
归根结底: 需要一些机制来“撤销”错误的决定

剩余网络 (Residual Network)

原始边: $e = (u, v) \in E$

- 流量 $f(e)$
- 容量 $c(e)$

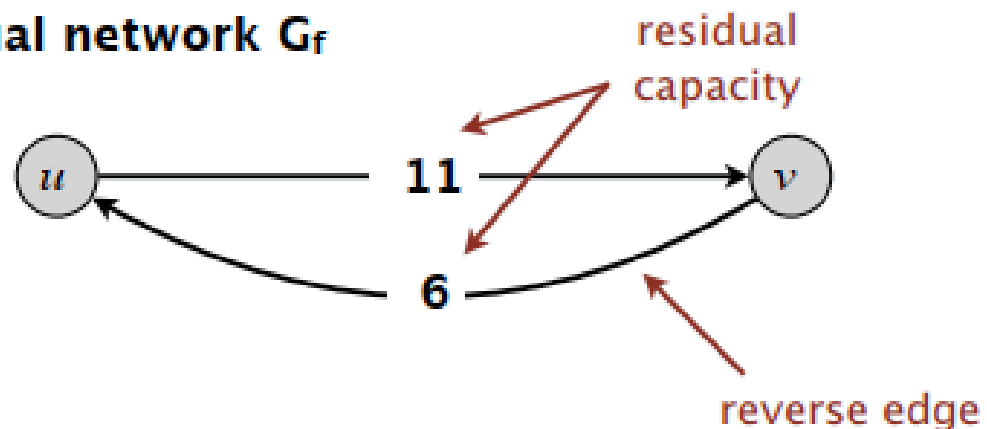
original flow network G



反向边: $e^{reverse} = (v, u)$

- 撤销已添加流量

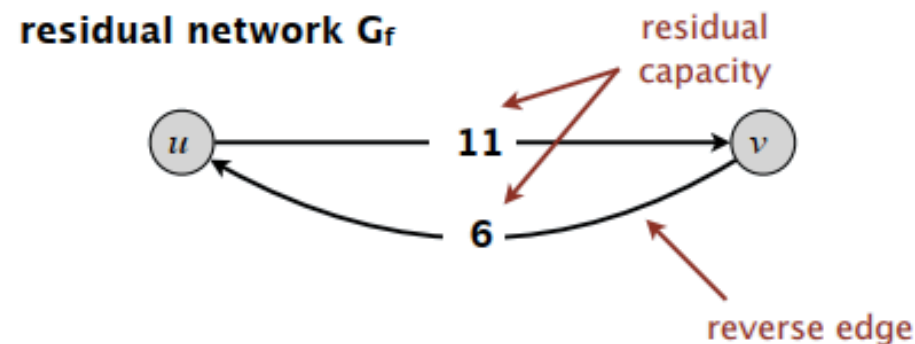
residual network G_f



剩余网络

剩余流量:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e^{reverse}) & \text{if } e^{reverse} \in E \end{cases}$$



剩余网络: $G_f = (V, E_f, s, t, c_f)$

- $E_f = \{e: f(e) < c(e)\} \cup \{e: f(e^{reverse}) > 0\}$
- 关键性质: f' 是 G_f 上的一个流当且仅当 $f + f'$ 是 G 上的一个流

剩余流量大于0的边

其中反向边上的流量否定了相应的正向边上的流量

增广路径

定义：增广路径是剩余网络 G_f 中的一条简单路径 $s \rightsquigarrow t$

定义：增广路径 P 的bottleneck容量是 P 中任意边的最小剩余容量

关键性质：设 f 是一个流， P 是 G_f 上的一条增广路径。则调用

$f' \leftarrow \text{AUGMENT}(f, c, P)$ 之后， f' 是一个流，并且 $\text{val}(f') = \text{val}(f) + \text{bottleneck}(G_f, P)$

AUGMENT(f, c, P)

$\delta \leftarrow$ bottleneck capacity of augmenting path P .

FOREACH edge $e \in P$:

IF ($e \in E$) $f(e) \leftarrow f(e) + \delta$.

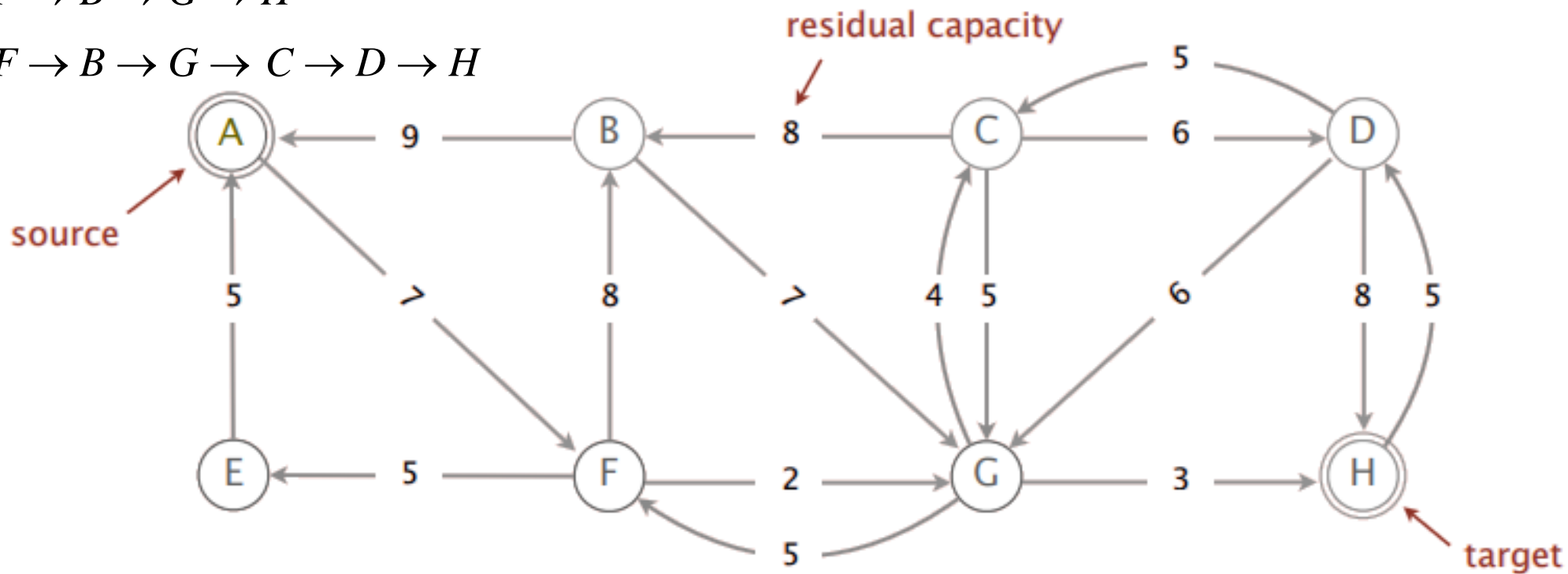
ELSE $f(e^{\text{reverse}}) \leftarrow f(e^{\text{reverse}}) - \delta$.

RETURN f .

网络流： 问题 2

下列选项哪一个是具有最大**bottleneck**容量的增广路径？

- A. $A \rightarrow F \rightarrow G \rightarrow H$
- B. $A \rightarrow B \rightarrow C \rightarrow D \rightarrow H$
- C. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow H$
- D. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow C \rightarrow D \rightarrow H$



Ford-Fulkerson算法

Ford-Fulkerson增广路径算法:

- 对每一条边 $e \in E$, 从 $f(e) = 0$ 开始
- 在剩余网络 G_f 中找到一条 $s \rightsquigarrow t$ 路径 P
- 沿着路径 P 增加流量
- 重复进行, 直到阻塞

FORD-FULKERSON(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f .

WHILE (there exists an $s \rightsquigarrow t$ path P in G_f)

$f \leftarrow$ **AUGMENT**(f, c, P).

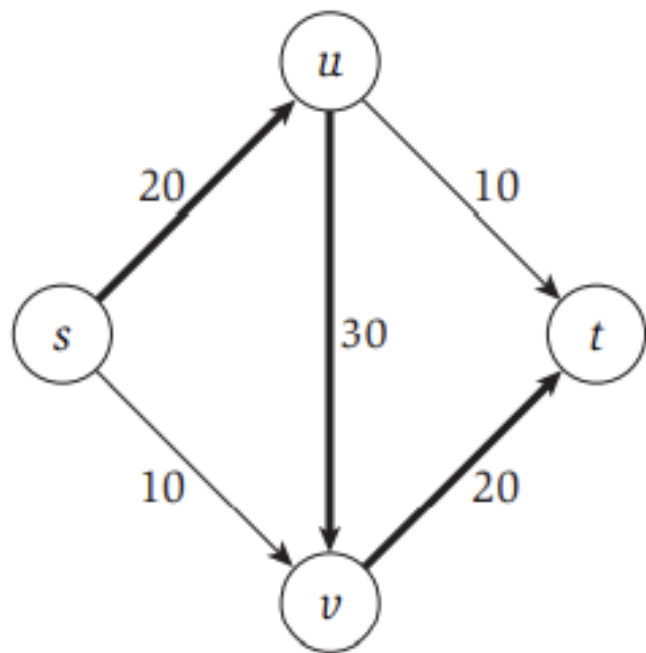
Update G_f .

RETURN f .

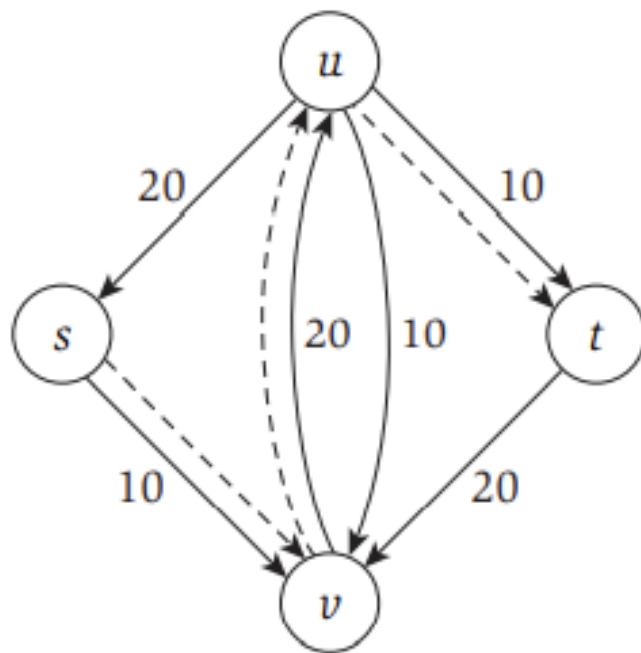
augmenting path



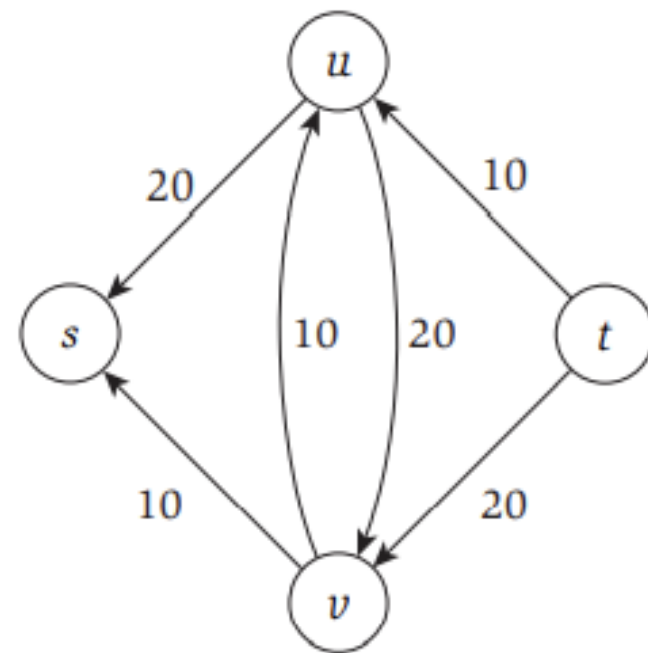
Ford-Fulkerson算法



(a)



(b)



(c)

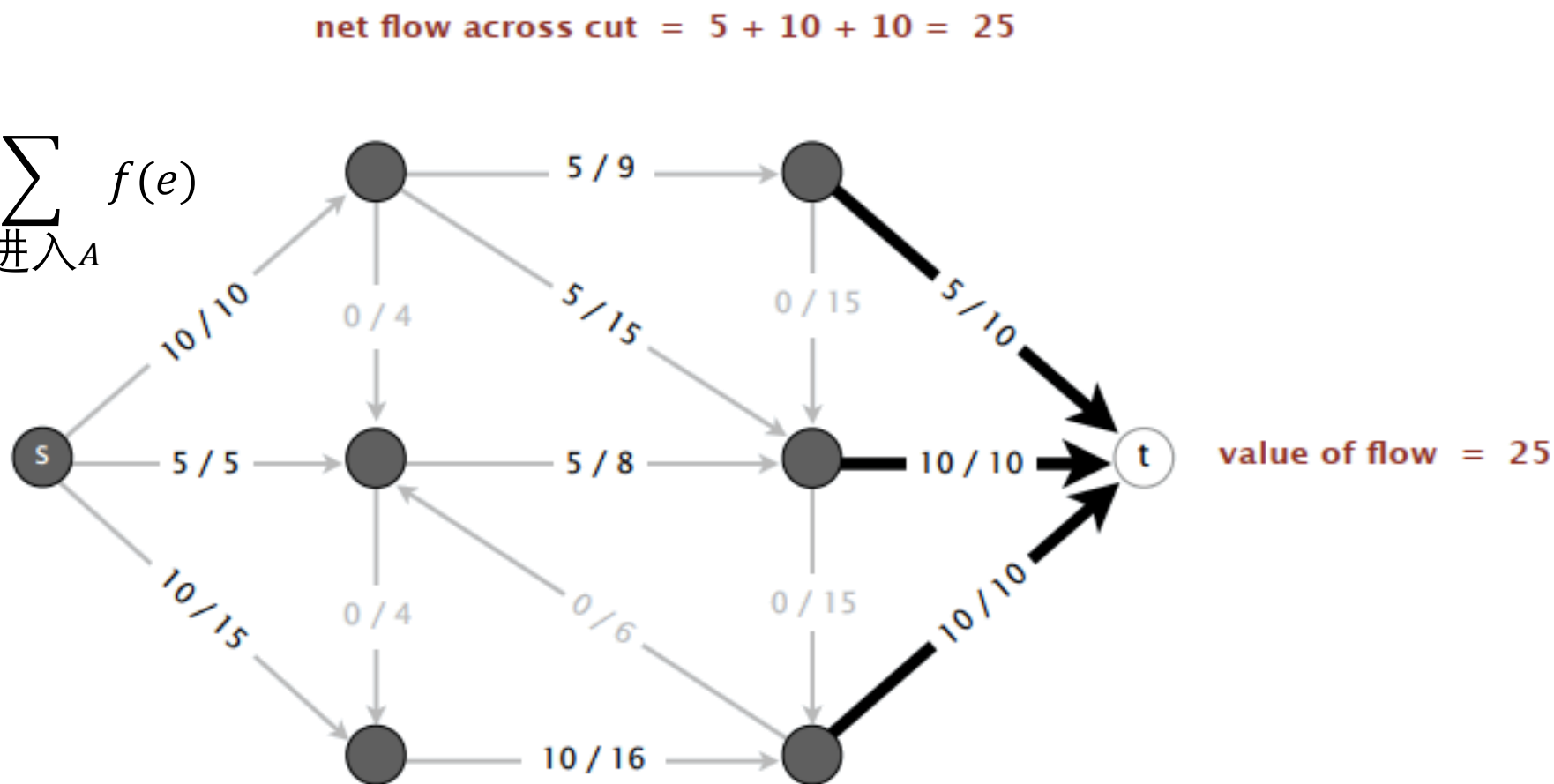
课程提要

- 最大流和最小割问题
- Ford- Fulkerson 算法
- 最大流最小割定理
- 容量缩放算法
- 最短增广路经

流和割之间的关系

流量值引理： 设 f 为任意流， (A,B) 为任意割。则流 f 的值等于穿过割 (A,B) 的网络流。

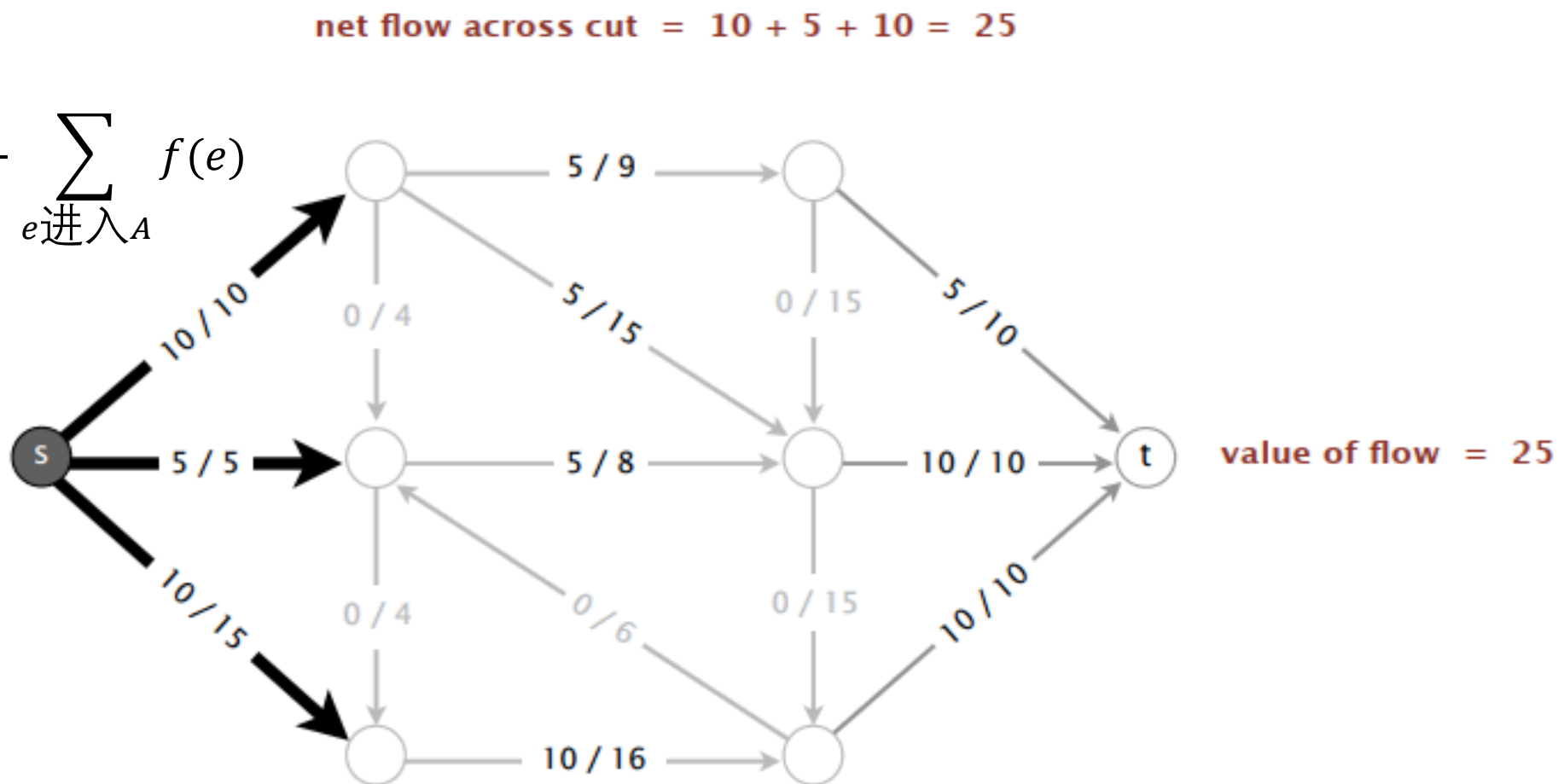
$$val(f) = \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e)$$



流和割之间的关系

流量值引理： 设 f 为任意流， (A,B) 为任意割。则流 f 的值等于穿过割 (A,B) 的网络流。

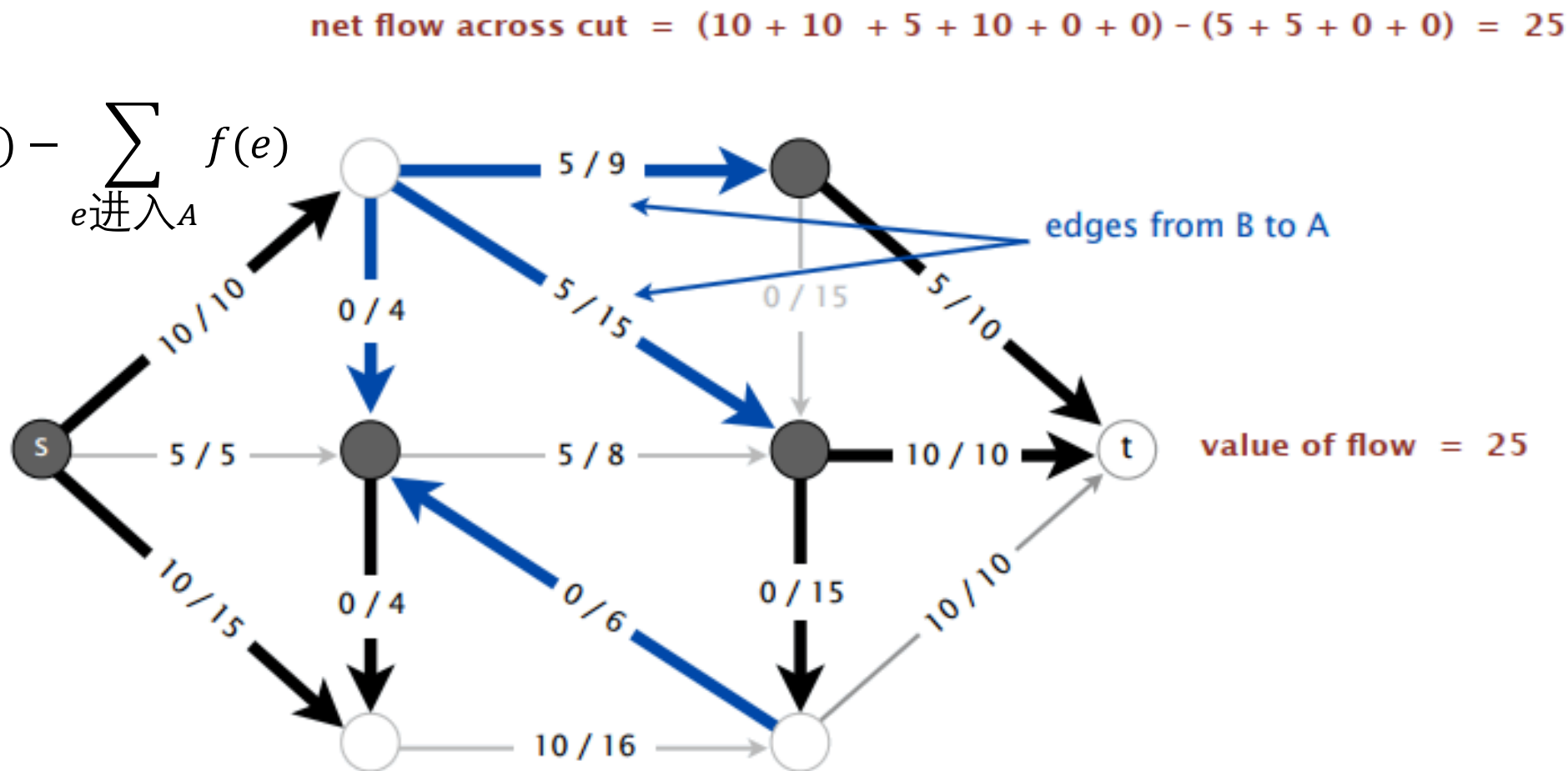
$$val(f) = \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e)$$



流和割之间的关系

流量值引理： 设 f 为任意流， (A,B) 为任意割。则，流 f 的值等于穿过割 (A,B) 的网络流。

$$val(f) = \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e)$$



网络流： 问题 3

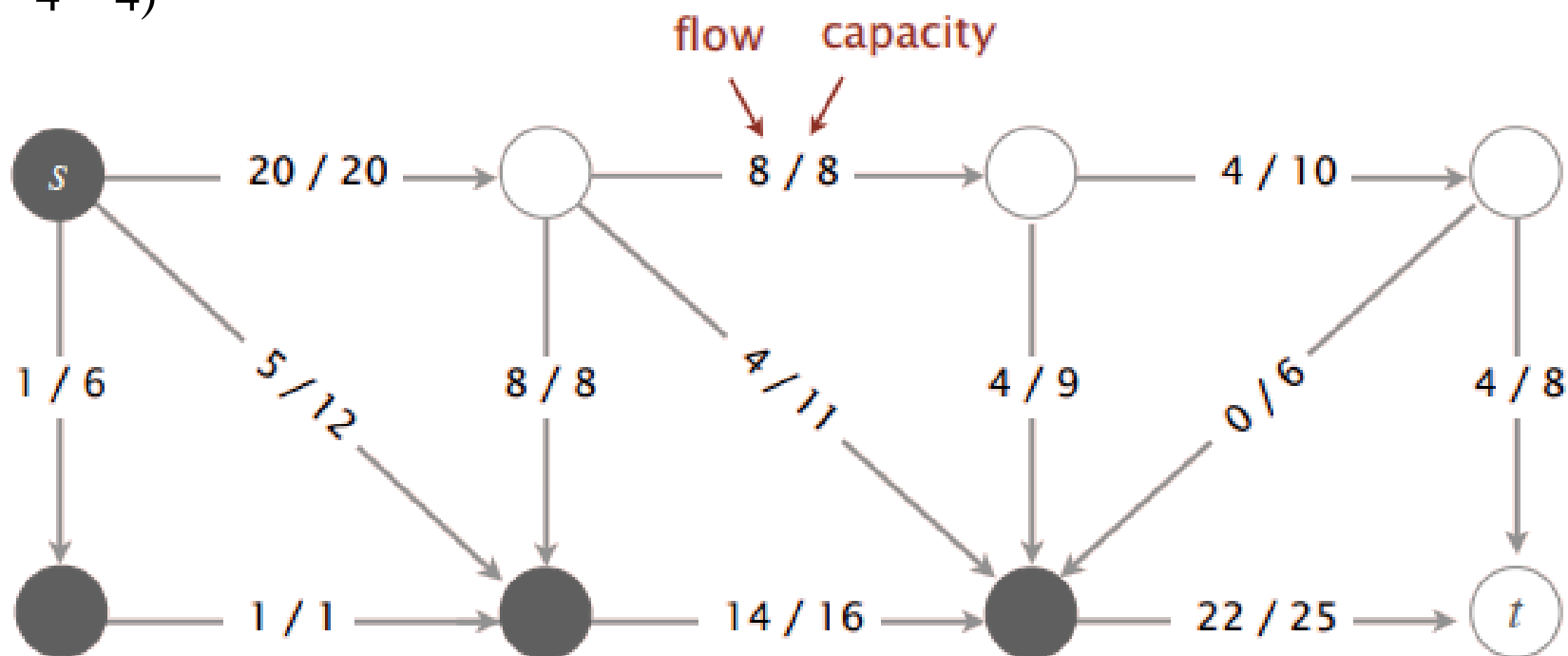
下列选项哪一个是穿过给定割的网络流？

A. 11 $(20 + 25 - 8 - 11 - 9 - 6)$

B. 26 $(20 + 22 - 8 - 4 - 4)$

C. 42 $(20 + 22)$

D. 45 $(20 + 25)$



流和割之间的关系

流量值引理： 设 f 为任意流， (A,B) 为任意割。则，流 f 的值等于穿过割 (A,B) 的网络流。
 $val(f) = \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e)$

证明：

根据流量守恒，所有
项除 $v=s$ 外，都是0



$$\begin{aligned} val(f) &= \sum_{e \text{ 离开 } s} f(e) - \sum_{e \text{ 进入 } s} f(e) \\ &= \sum_{v \in A} \left(\sum_{e \text{ 离开 } v} f(e) - \sum_{e \text{ 进入 } v} f(e) \right) \\ &= \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e) \end{aligned}$$

流和割之间的关系

弱对偶性Weak Duality: f 是任意流, (A, B) 是任意割。则, $val(f) \leq cap(A, B)$

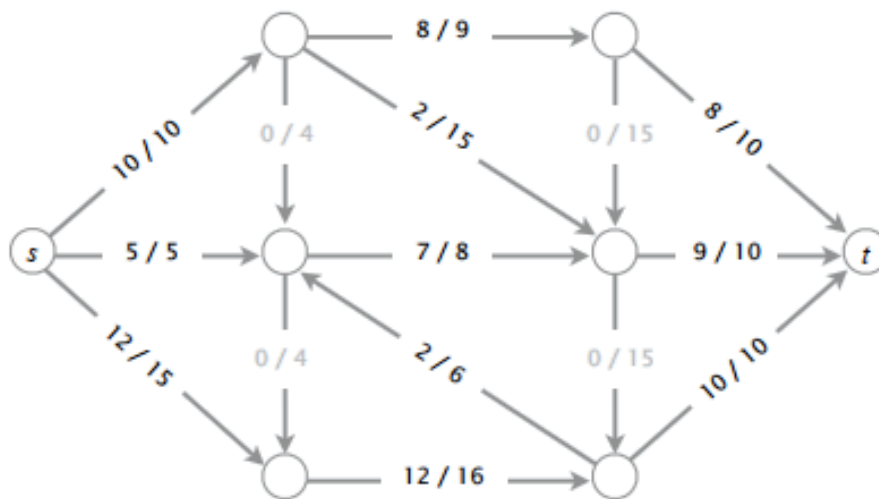
证明:

流量值引理

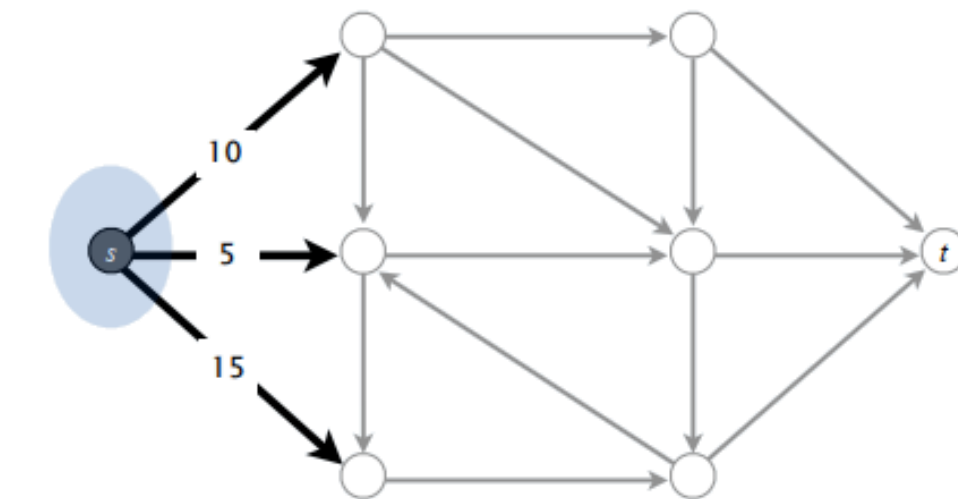
$$val(f) = \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e)$$

$$\leq \sum_{e \text{ 离开 } A} f(e)$$

$$\leq \sum_{e \text{ 离开 } A} c(e) = cap(A, B)$$



value of flow = 27



\leq

capacity of cut = 30

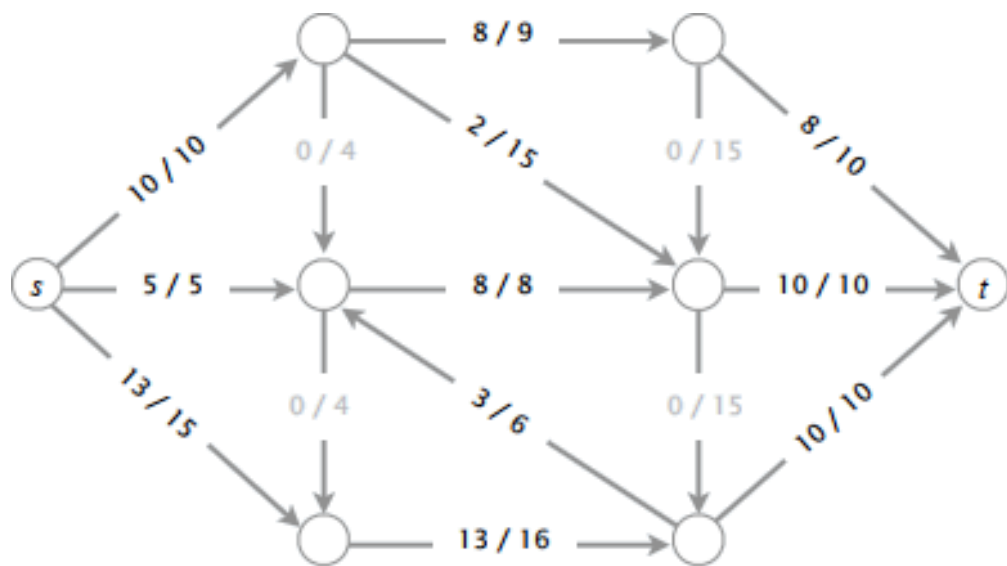
最优性证明

推论: f 是一个流, (A, B) 是任意割。如果 $val(f) = cap(A, B)$, 则,
 f 是一个最大流, (A, B) 是一个最小割

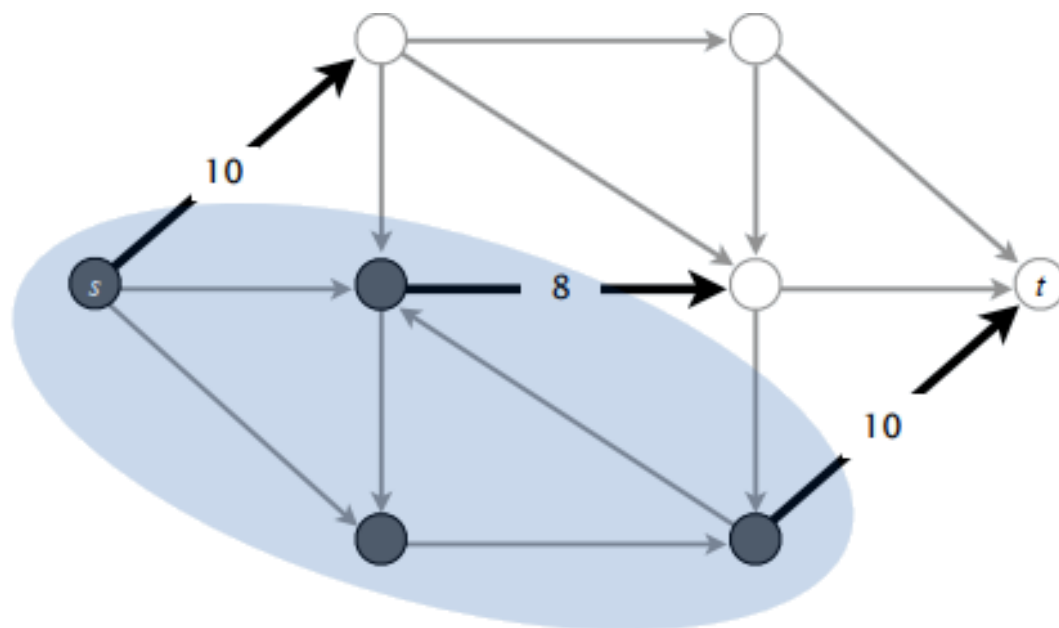
证明:

- 对任意流 f' : $val(f') \leq cap(A, B) = val(f)$
- 对任意割 (A', B') : $cap(A', B') \geq val(f) = cap(A, B)$

弱对偶性



value of flow = 28



=

capacity of cut = 28

最大流最小割定理

最大流最小割定理：最大流的值等于最小割的容量 **强对偶性Strong Duality**

MAXIMAL FLOW THROUGH A NETWORK

L. R. FORD, JR. AND D. R. FULKERSON

Introduction. The problem discussed in this paper was formulated by T. Harris as follows:

"Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other."

ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS

G. B. Dantzig
D. R. Fulkerson

P-826

April 15, 1955

A Note on the Maximum Flow Through a Network*

P. ELIAS†, A. FEINSTEIN‡, AND C. E. SHANNON§

Summary—This note discusses the problem of maximizing the rate of flow from one terminal to another, through a network which consists of a number of branches, each of which has a limited capacity. The main result is a theorem: The maximum possible flow from left to right through a network is equal to the minimum value among all simple cut-sets. This theorem is applied to solve a more general problem, in which a number of input nodes and a number of output nodes are used.

from one terminal to the other in the original network passes through at least one branch in the cut-set. In the network above, some examples of cut-sets are (d, e, f) , and (b, c, e, g, h) , (d, g, h, i) . By a *simple cut-set* we will mean a cut-set such that if any branch is omitted it is no longer a cut-set. Thus (d, e, f) and (b, c, e, g, h) are simple cut-sets while (d, e, h, i) is not. When a simple cut-set is

最大流最小割定理

最大流最小割定理：最大流的值等于最小割的容量

增广路径定理：流 f 是最大流当且仅当没有增广路径

证明：以下三个条件对任意流 f 来说等价：

- (1) 存在一个割 (A, B) ，使得 $cap(A, B) = val(f)$
 - (2) f 是最大流
 - (3) 不存在关于 f 的增广路径
- ← 如果Ford-Fulkerson终止
则 f 是最大流

(1) \Rightarrow (2) 弱对等性推论

最大流最小割定理

最大流最小割定理：最大流的值等于最小割的容量

增广路径定理：流 f 是最大流当且仅当没有增广路径

证明：以下三个条件对任意流 f 来说等价：

- (1)存在一个割 (A, B) ，使得 $cap(A, B) = val(f)$
- (2) f 是最大流
- (3)不存在关于 f 的增广路径

(2) \Rightarrow (3) 反证法： $\neg(3) \Rightarrow \neg(2)$

- (1)假设存在一个关于 f 的增广路径
- (2)沿着这条路径输送流量来改善 f
- (3)因此， f 不是最大流

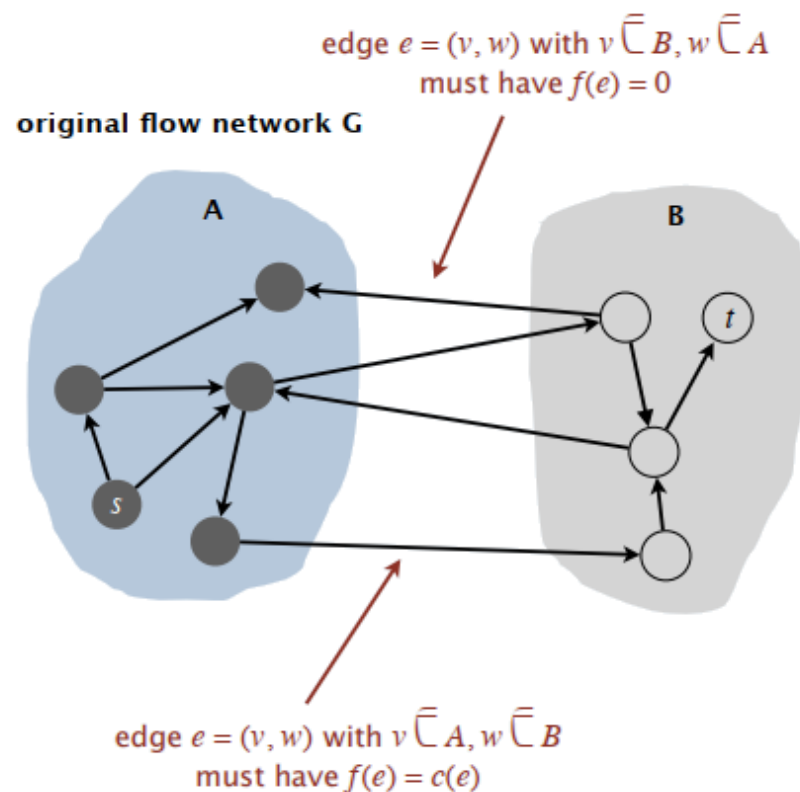
最大流最小割定理

(3) \Rightarrow (1)

- (1) 设流 f 没有增广路径
- (2) 设 A 是剩余网络 G_f 中从 s 可达的节点的集合
- (3) 由 A 的定义可知: $s \in A$
- (4) 由 f 的定义可知: $t \notin A$

流值引理

$$\begin{aligned} \text{val}(f) &= \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e) \\ &= \sum_{e \text{ 离开 } A} c(e) - 0 \\ &= \text{cap}(A, B) \end{aligned}$$

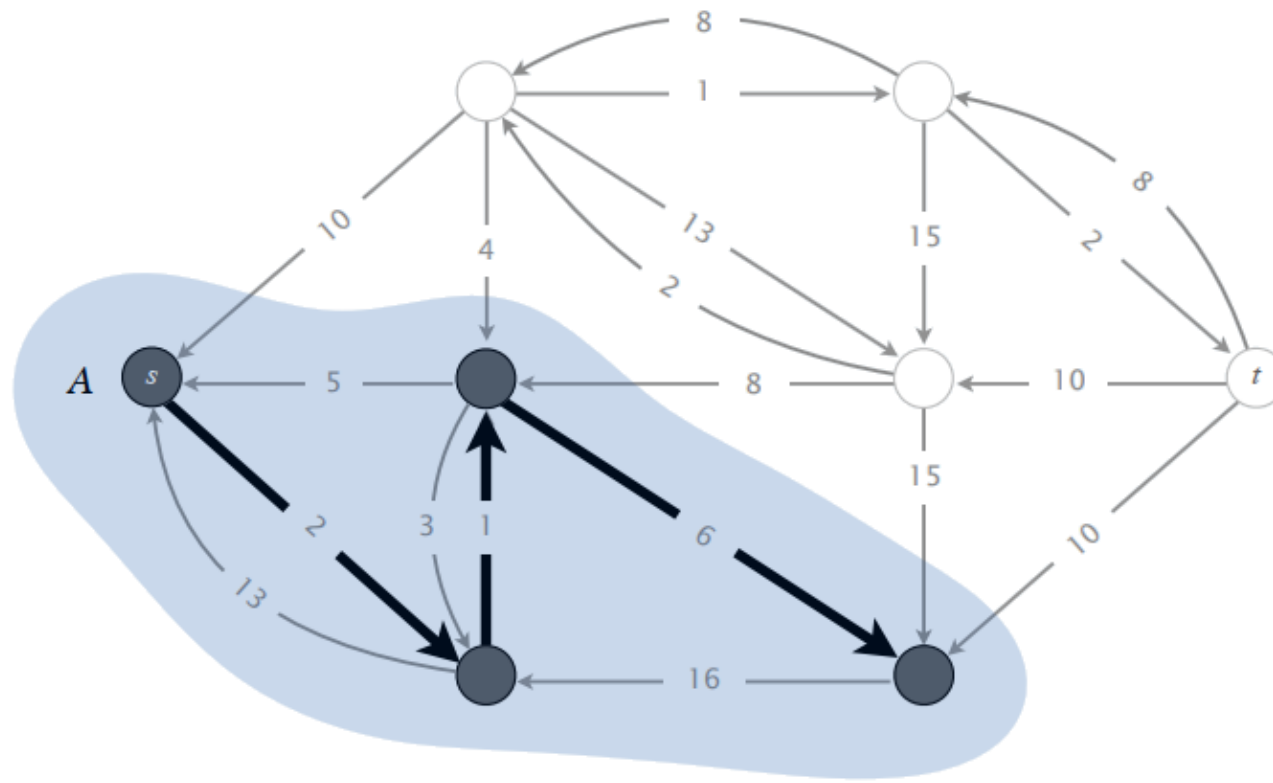


从最大流中计算最小割

定理：给定任意最大流 f ，可以在 $O(m)$ 时间内计算最小割 (A, B)

证明：设 A 是剩余网络 G_f 中从 s 可达的节点的集合

上一张幻灯片的论证意味着， (A, B) 的容量=流量值 f



课程提要

- 最大流和最小割问题
- Ford- Fulkerson 算法
- 最大流最小割定理
- 容量缩放算法
- 最短增广路经


Ford-Fulkerson算法的分析 (整数值容量)

假设: 每条边的容量 $c(e)$ 是1到 C 之间的一个整数

整数不变性: 整个Ford-Fulkerson, 每条边的流量 $f(e)$ 和剩余容量 $c_f(e)$ 是一个整数

证明: 通过对增广路径的数量进行归纳

考虑 cut $A = \{s\}$
(假设没有平行的边)




定理: Ford-Fulkerson最多在 $val(f^*) \leq nC$ 条增广路径后停止, 其中 f^* 是最大流

证明: 每一次增量都会使流量值至少增加1

推论: Ford-Fulkerson的运行时间是 $O(mnC)$

证明: 可以在 $O(m)$ 时间内, 使用BFS或者DFS找到一条增广路径

对于每条边 e , $f(e)$ 是一个整数



整数定理: 存在一个整数值的最大流 f^*

证明: 由于Ford-Fulkerson终止, 定理由整数不变性 (和增广路径定理) 得出

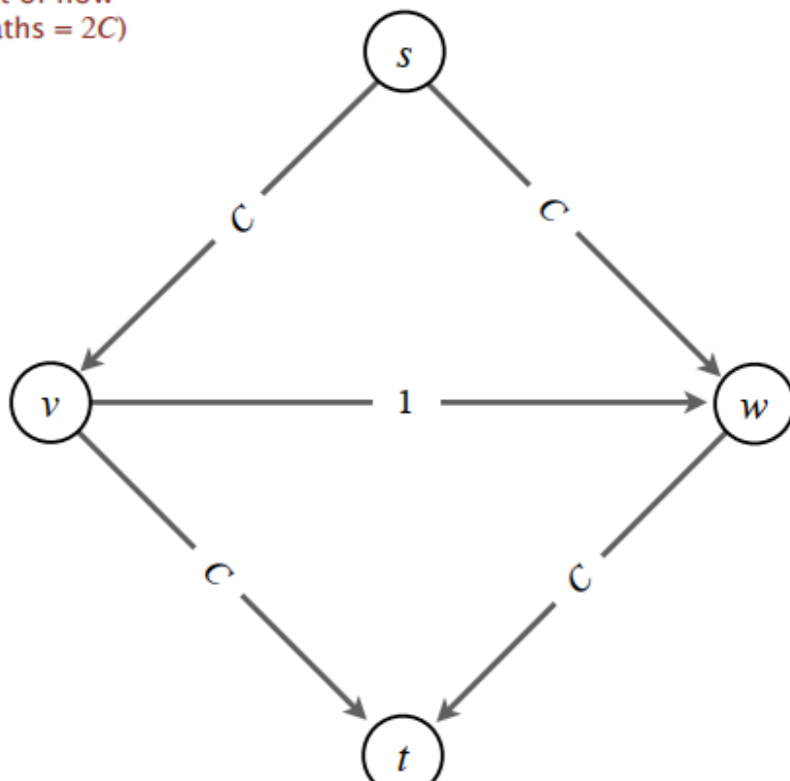
Ford-Fulkerson: 指数型例子

Q: Ford-Fulkerson算法 $O(mnC)$ 的复杂度界好吗?

A: 如果最大容量是 C , 则算法迭代次数 $\geq C$

- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- ...
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$

← each augmenting path
sends only 1 unit of flow
(# augmenting paths = $2C$)



网络流： 问题 4

如果边容量是(), Ford-Fulkerson算法保证会终止

- A. Rational numbers.
- B. Real numbers.
- C. Both A and B.
- D. Neither A nor B.

选择好的增广路径

在选择增广路径时要注意：每条边的容量 $c(e)$ 是1到 C 之间的一个整数

- 一些选择会导致指数级算法
- 聪明的选择导致多项式级算法

病态：当边容量是无理数时，不能保证Ford-Fulkerson终止（或收敛到最大流）

目标：选择增广路径，使得：

- 有效地找到增广路径
- 迭代次数少

选择好的增广路径

选择增广路径:

- 最大bottleneck容量 (“fattest”)
- 足够大的bottleneck容量
- 最少的边

Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems

JACK EDMONDS

University of Waterloo, Waterloo, Ontario, Canada

AND

RICHARD M. KARP

University of California, Berkeley, California

ABSTRACT. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

Edmonds-Karp 1972 (USA)

Dokl. Akad. Nauk SSSR
Tom 194 (1970), No. 4

Soviet Math. Dokl.
Vol. 11 (1970), No. 5

ALGORITHM FOR SOLUTION OF A PROBLEM OF MAXIMUM FLOW IN A NETWORK WITH POWER ESTIMATION

UDC 518.5

E. A. DINIC

Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this algorithm requires preliminary rounding off of the initial data, i.e. only an approximate solution of the problem is possible. In this connection the rapidity of convergence of the algorithm is inversely proportional to the relative precision.

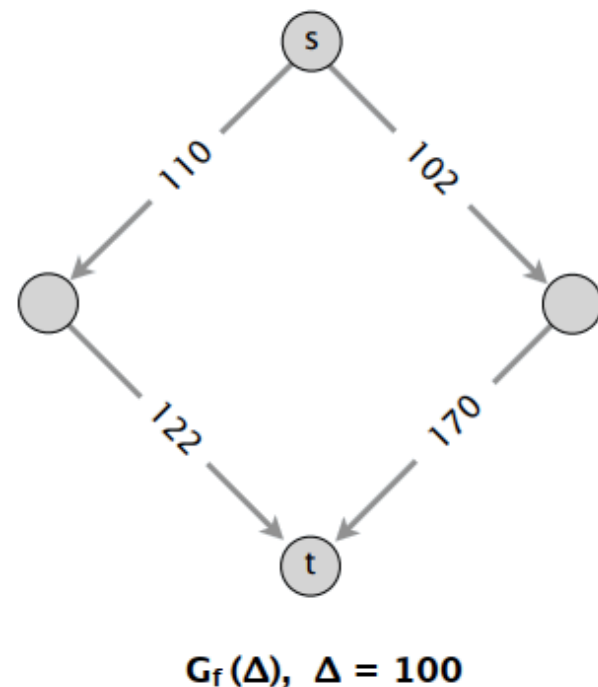
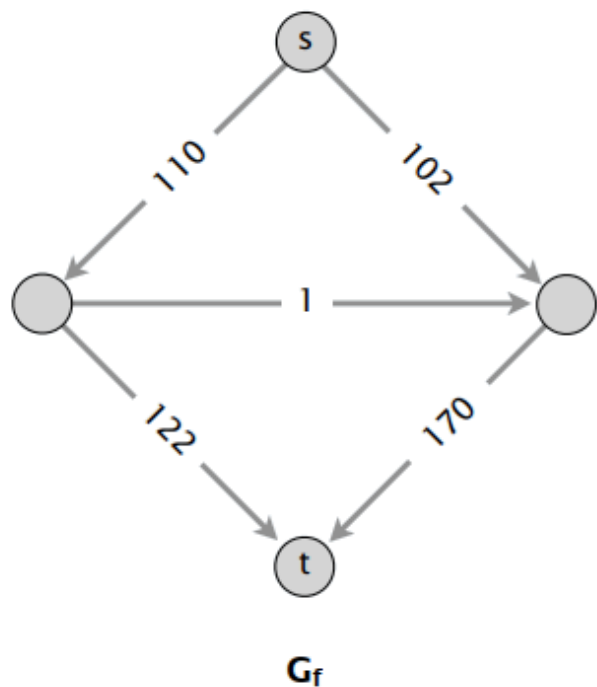
Dinitz 1970 (Soviet Union)

invented in response to a class
exercises by Adel'son-Vel'skiĭ

容量缩放算法

概述：选择“大”bottleneck容量的增广路径

- 保持缩放参数 Δ
- 设 $G_f(\Delta)$ 是剩余网络的一部分，只包含容量 $\geq \Delta$ 的边
- $G_f(\Delta)$ 中任何增广路径的bottleneck容量 $\geq \Delta$



容量缩放算法

CAPACITY-SCALING(G)

FOREACH edge $e \in E$: $f(e) \leftarrow 0$.

$\Delta \leftarrow$ largest power of 2 $\leq C$.

WHILE ($\Delta \geq 1$)

$G_f(\Delta) \leftarrow \Delta$ -residual network of G with respect to flow f .

WHILE (there exists an $s \rightsquigarrow t$ path P in $G_f(\Delta)$)

$f \leftarrow \text{AUGMENT}(f, c, P)$.

Update $G_f(\Delta)$.

$\Delta \leftarrow \Delta / 2$.

Δ -scaling phase

RETURN f .

容量缩放算法——正确性证明

假设：所有边的容量都是1到 C 之间的整数

不变量：缩放参数 Δ 是2的幂

证明：初始是2的幂，每个阶段将 Δ 除以2

整数不变性：整个算法中，每条边的流量 $f(e)$ 和剩余容量 $c_f(e)$ 是一个整数

证明：和通用的Ford-Fulkerson一样

定理：如果容量缩放算法终止，则 f 是最大流

证明：i. 由整数不变性，当 $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$

ii. $\Delta = 1$ 阶段结束时，没有增广路径

iii. 结果由增广路径定理得出

容量缩放算法——运行时间分析

引理 1: 存在 $1 + \lceil \log_2 C \rceil$ 个放缩阶段

证明: 初始 $C/2 \leq \Delta \leq C$; 每次迭代 Δ 按 2 的因子下降

引理 2: 设 f 是 Δ 缩放阶段结束时的流, 则最大流的值 $val(f) + m\Delta$

证明: 见下一张幻灯片

引理 3: 缩放阶段的增广次数 $\leq 2m$

证明: i. 设 f 是 Δ 缩放阶段开始时的流

ii. 引理 2 \Rightarrow 最大流的值 $val(f) \leq m(2\Delta)$

iii. Δ 的每一次增量都会使 $val(f)$ 至少增加 Δ

定理: 容量缩放算法运行时间为 $O(m^2 \log C)$

证明: i. 引理 1 + 引理 3 $\Rightarrow O(m \log C)$ 增量

ii. 找到一条增广路径花费时间 $O(m)$

容量缩放算法——运行时间分析

引理 2: 设 f 是 Δ 缩放阶段结束时的流, 则最大流的值 $val(f) + m\Delta$

证明: i. 我们证明存在割 (A,B) , 使得 $cap(A,B) \leq val(f) + m\Delta$

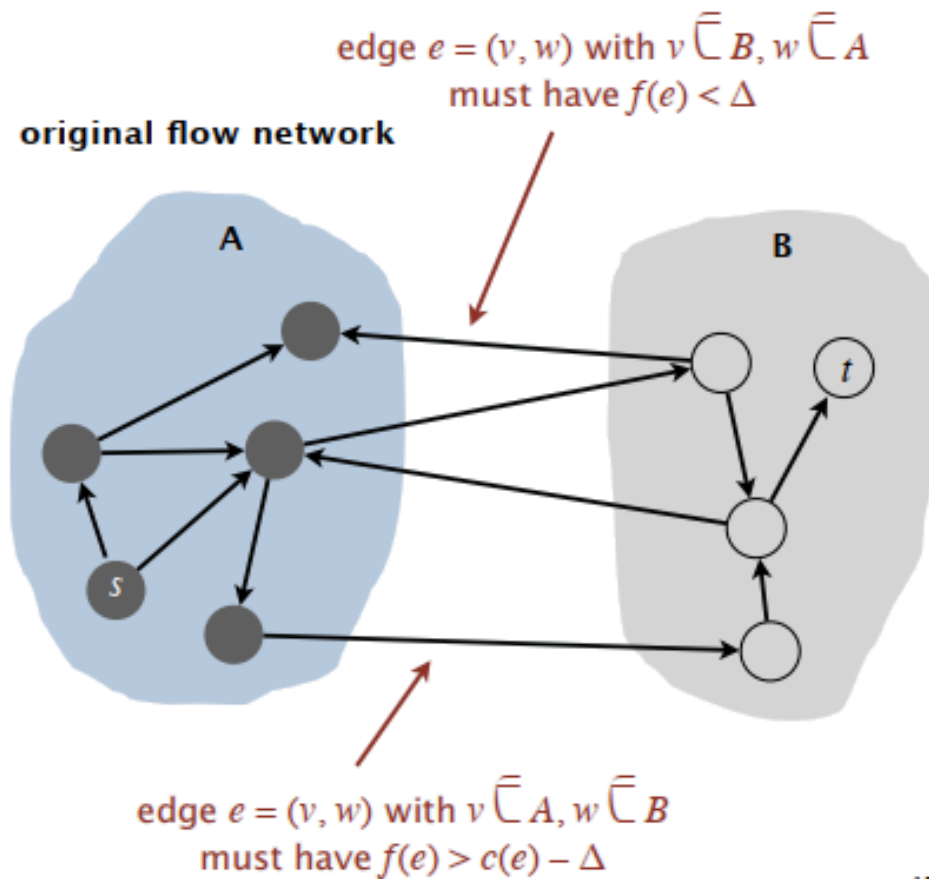
ii. 选择 $G_f(\Delta)$ 中从 s 可达的点的集合 A

iii. 由 A 的定义可知: $s \in A$

iv. 由流 f 的定义可知: $t \notin A$

流值引理

$$\begin{aligned} val(f) &= \sum_{e \text{ 离开 } A} f(e) - \sum_{e \text{ 进入 } A} f(e) \\ &\geq \sum_{e \text{ 离开 } A} (c(e) - \Delta) - \sum_{e \text{ 进入 } A} \Delta \\ &\geq \sum_{e \text{ 离开 } A} c(e) - \sum_{e \text{ 离开 } A} \Delta - \sum_{e \text{ 进入 } A} \Delta \\ &\geq cap(A,B) - m\Delta \end{aligned}$$



课程提要

- 最大流和最小割问题
- Ford- Fulkerson 算法
- 最大流最小割定理
- 容量缩放算法
- 最短增广路经

最短增广路径

Q: 在Ford-Fulkerson中, 如何选择下一个增广路径

A: 用最少的边来选择 可以通过BFS找到

SHORTEST-AUGMENTING-PATH(G)

FOREACH $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f .

WHILE (there exists an $s \rightsquigarrow t$ path in G_f)

$P \leftarrow \text{BREADTH-FIRST-SEARCH}(G_f)$.

$f \leftarrow \text{AUGMENT}(f, c, P)$.

Update G_f .

RETURN f .

最短增广路径——分析概述

引理 1: 最短增广路径的长度不会减少

边的数量

引理 2: 在最多 m 次最短路径增广后, 最短增广路径的长度严格增加。

定理: 最短增广路径算法运行时间为 $O(m^2n)$

证明: i. 通过BFS找到一条最短增广路径用时 $O(m)$

ii. 增广次数 $\leq mn$

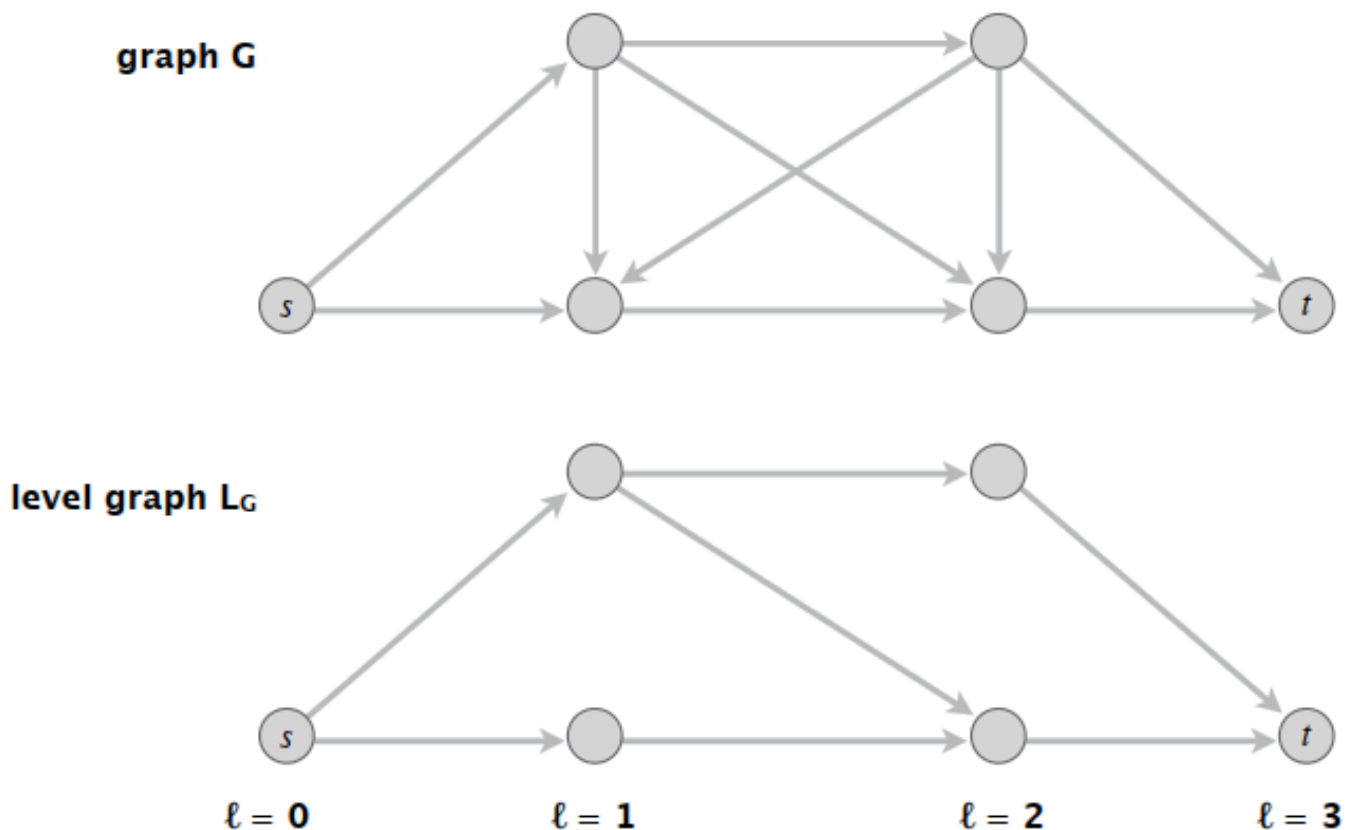
- 最多 m 条长度为 k 的增广路径 ← Lemma 1+Lemma 2

- 最多 $n - 1$ 个不同长度

最短增广路径——分析

定义： 给定源节点为 s 的有向图 $G = (V, E)$ ，它的**水平图**定义如下：

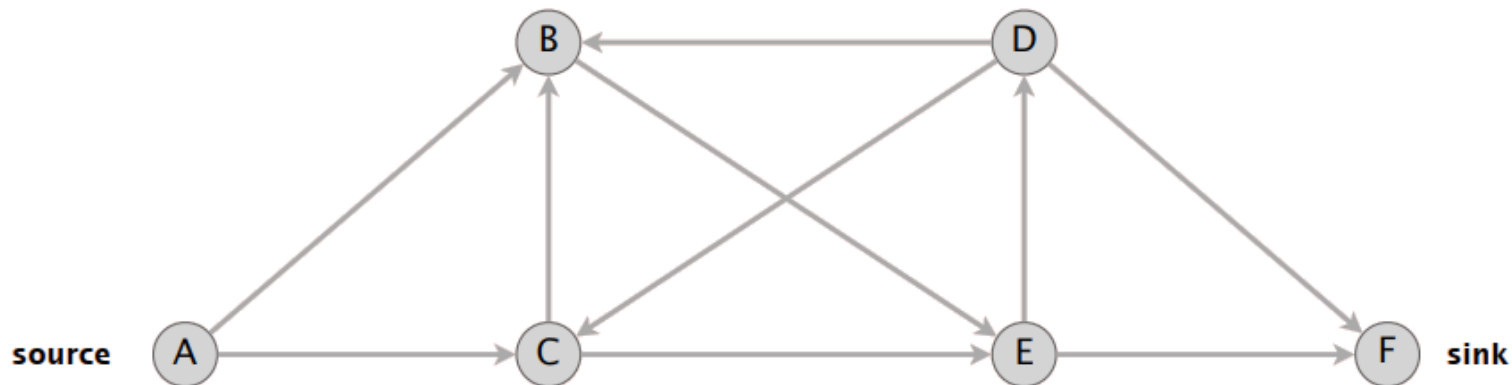
- $l(v)$ = 最短路径 $s \rightsquigarrow v$ 中边的数量
- $L_G = (V, E_G)$ 是 G 的子图，只包含边 $(v, w) \in E$ ，且 $l(w) = l(v) + 1$



网络流： 问题 5

哪些边位于以下有向图的水平图中？

- A. $D \rightarrow F$.
- B. $E \rightarrow F$.
- C. Both A and B.
- D. Neither A nor B.



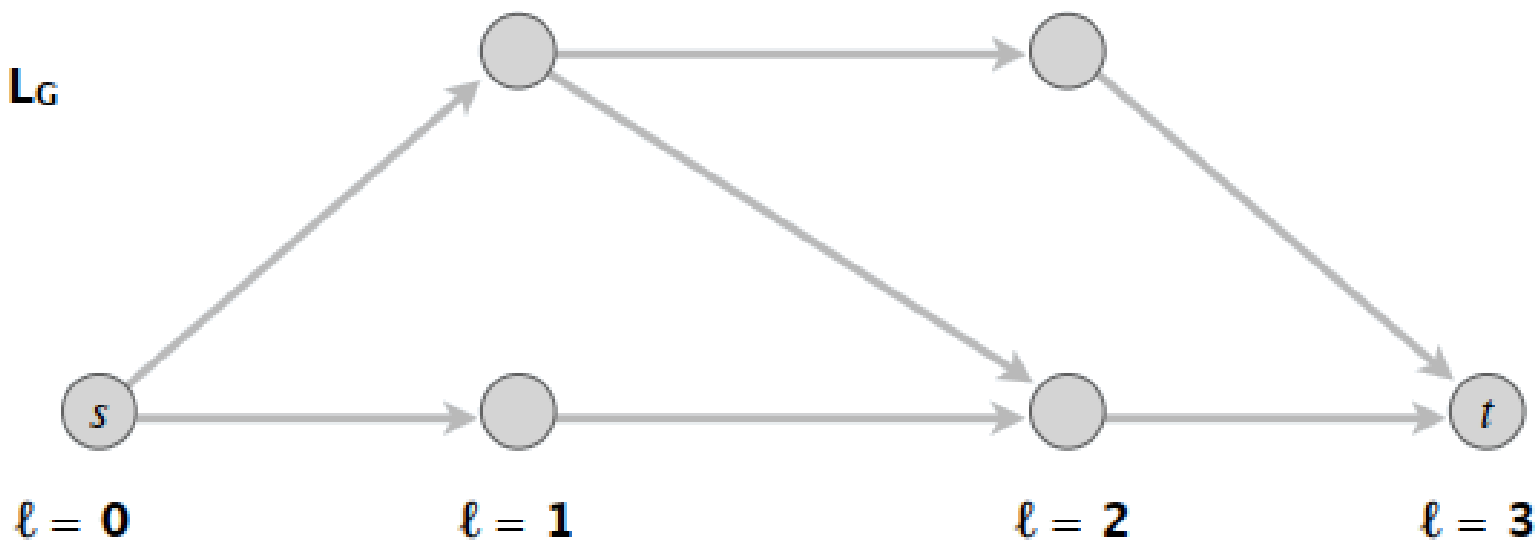
最短增广路径——分析

定义：给定源节点为 s 的有向图 $G = (V, E)$ ，它的**水平图**定义如下：

- $l(v)$ = 最短 $s \rightsquigarrow v$ 路径中边的数量
- $L_G = (V, E_G)$ 是 G 的子图，只包含边 $(v, w) \in E$ ，且 $l(w) = l(v) + 1$

关键性质： P 是 G 中的一条最短 $s \rightsquigarrow v$ 路径当且仅当 P 是 L_G 中的一条 $s \rightsquigarrow v$ 路径

level graph L_G

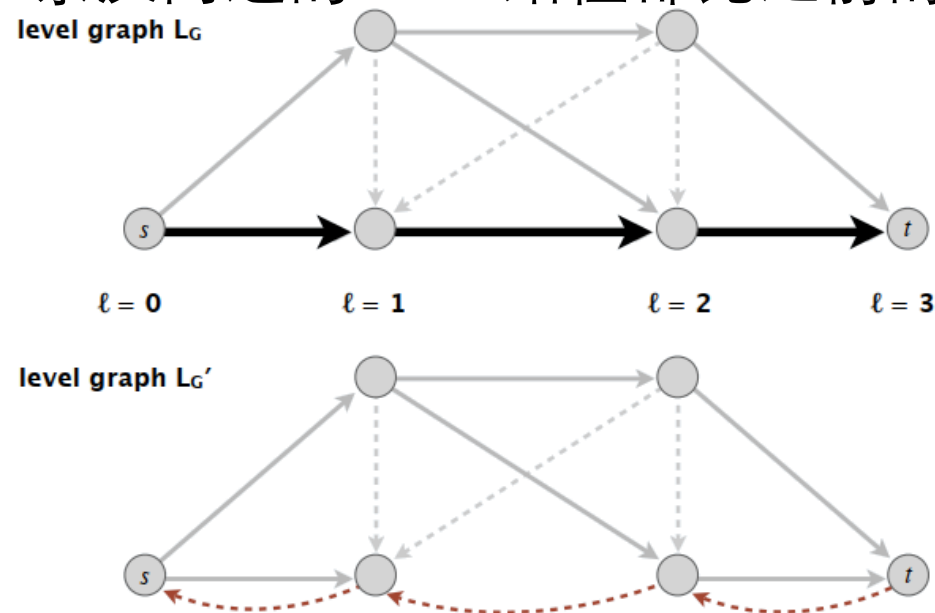


最短增广路径——分析

引理 1: 最短增广路径的长度不会减少

- 设 f 和 f' 分别是一条最短增广路径之前和之后的流
- 设 L_G 和 $L_{G'}$ 分别是 G_f 和 $G_{f'}$ 的水平图
- 只有反向边增加到了 $G_{f'}$

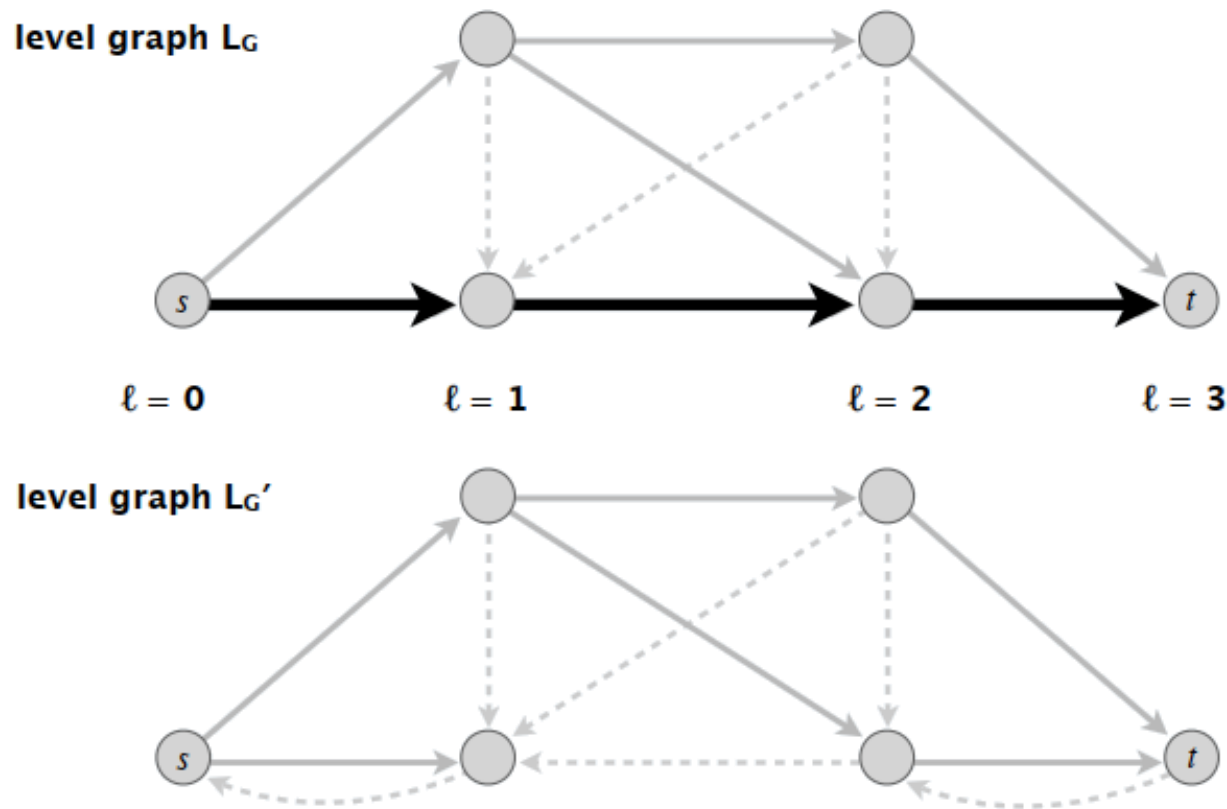
(任意使用了一条反向边的 $s \rightsquigarrow t$ 路径都比之前的路径长)



最短增广路径——分析

引理 2: 在最多 m 次最短路径增广后，最短增广路径的长度严格增加。

- 每次增广中，至少一条(bottleneck)边是从 L_G 中删除
- 没有新的边增加到 L_G ，直到最短路径长度严格增加



最短增广路径——分析回顾

引理 1: 整个算法中, 最短增广路径的长度不会减少

引理 2: 在最多 m 次最短路径增广后, 最短增广路径的长度严格增加。

定理: 最短增广路径算法运行时间为 $O(m^2n)$

最短增广路径——改进运行时间

注意: $\Theta(mn)$ 的增量对一些流网络来说是必要的

- 尝试在每一次增广中减少时间
- 简单的方法 $\Rightarrow O(mn^2)$ [Dinitz 1970]
- 动态树 $\Rightarrow O(mn \log n)$ [Sleator-Tarjan 1983]

A Data Structure for Dynamic Trees

DANIEL D. SLEATOR AND ROBERT ENDRE TARJAN

Bell Laboratories, Murray Hill, New Jersey 07974

Received May 8, 1982; revised October 18, 1982

A data structure is proposed to maintain a collection of vertex-disjoint trees under a sequence of two kinds of operations: a *link* operation that combines two trees into one by adding an edge, and a *cut* operation that divides one tree into two by deleting an edge. Each operation requires $O(\log n)$ time. Using this data structure, new fast algorithms are obtained for the following problems:

- (1) Computing nearest common ancestors.
- (2) Solving various network flow problems including finding maximum flows, blocking flows, and acyclic flows.
- (3) Computing certain kinds of constrained minimum spanning trees.
- (4) Implementing the network simplex algorithm for minimum-cost flows.


The most significant application is (2); an $O(mn \log n)$ -time algorithm is obtained to find a maximum flow in a network of n vertices and m edges, beating by a factor of $\log n$ the fastest algorithm previously known for sparse graphs.

增广路径算法：总结

year	method	# augmentations	running time	
1955	augmenting path	$n C$	$O(m n C)$	
1972	fattest path	$m \log (mC)$	$O(m^2 \log n \log (mC))$	fat paths
1972	capacity scaling	$m \log C$	$O(m^2 \log C)$	
1985	improved capacity scaling	$m \log C$	$O(m n \log C)$	
1970	shortest augmenting path	$m n$	$O(m^2 n)$	shortest paths
1970	level graph	$m n$	$O(m n^2)$	
1983	dynamic trees	$m n$	$O(m n \log n)$	

augmenting-path algorithms with m edges, n nodes, and integer capacities between 1 and C

最大流算法：理论亮点

year	method	worst case	discovered by
1951	simplex	$O(m n^2 C)$	Dantzig
1955	augmenting paths	$O(m n C)$	Ford–Fulkerson
1970	shortest augmenting paths	$O(m n^2)$	Edmonds–Karp, Dinitz
1974	blocking flows	$O(n^3)$	Karzanov
1983	dynamic trees	$O(m n \log n)$	Sleator–Tarjan
1985	improved capacity scaling	$O(m n \log C)$	Gabow
1988	push-relabel	$O(m n \log (n^2 / m))$	Goldberg–Tarjan
1998	binary blocking flows	$O(m^{3/2} \log (n^2 / m) \log C)$	Goldberg–Rao
2013	compact networks	$O(m n)$	Orlin
2014	interior-point methods	$\tilde{O}(m n^{1/2} \log C)$	Lee–Sidford
2016	electrical flows	$\tilde{O}(m^{10/7} C^{1/7})$	Mądry
20xx			

max-flow algorithms with m edges, n nodes, and integer capacities between 1 and C

最大流算法： 实践

Push-relabel 算法(SECTION 7.4): [Goldberg-Tarjan 1988]

每一次增加一条边的流量， 而不是每一次增加一条增广路径

A New Approach to the Maximum-Flow Problem

ANDREW V. GOLDBERG

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

ROBERT E. TARJAN

Princeton University, Princeton, New Jersey, and AT&T Bell Laboratories, Murray Hill, New Jersey

Abstract. All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the *preflow* concept of Karzanov is introduced. A preflow is like a flow, except that the total amount flowing into a vertex is allowed to exceed the total amount flowing out. The method maintains a preflow in the original network and pushes local flow excess toward the sink along what are estimated to be shortest paths. The algorithm and its analysis are simple and intuitive, yet the algorithm runs as fast as any other known method on dense graphs, achieving an $O(n^3)$ time bound on an n -vertex graph. By incorporating the dynamic tree data structure of Sleator and Tarjan, we obtain a version of the algorithm running in $O(nm \log(n^2/m))$ time on an n -vertex, m -edge graph. This is as fast as any known method for any graph density and faster on graphs of moderate density. The algorithm also admits efficient distributed and parallel implementations. A parallel implementation running in $O(n^2 \log n)$ time using n processors and $O(m)$ space is obtained. This time bound matches that of the Shiloach-Vishkin algorithm, which also uses n processors but requires $O(n^2)$ space.

最大流算法： 实践

计算机视觉： 对于在计算机视觉应用中出现的一些密集问题， 不同的算法效果更好。

An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision

Yuri Boykov and Vladimir Kolmogorov*

Abstract

After [15, 31, 19, 8, 25, 5] minimum cut/maximum flow algorithms on graphs emerged as an increasingly useful tool for exact or approximate energy minimization in low-level vision. The combinatorial optimization literature provides many min-cut/max-flow algorithms with different polynomial time complexity. Their practical efficiency, however, has to date been studied mainly outside the scope of computer vision. The goal of this paper is to provide an experimental comparison of the efficiency of min-cut/max flow algorithms for applications in vision. We compare the running times of several standard algorithms, as well as a new algorithm that we have recently developed. The algorithms we study include both Goldberg-Tarjan style “push-relabel” methods and algorithms based on Ford-Fulkerson style “augmenting paths”. We benchmark these algorithms on a number of typical graphs in the contexts of image restoration, stereo, and segmentation. In many cases our new algorithm works several times faster than any of the other methods making near real-time performance possible. An implementation of our max-flow/min-cut algorithm is available upon request for research purposes.

VERMA, BATRA: MAXFLOW REVISITED

1

MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems

Tanmay Verma
tanmay08054@iitd.ac.in
Dhruv Batra
dbatra@ttic.edu

IIT-Delhi
Delhi, India
TTI-Chicago
Chicago, USA

Abstract

Algorithms for finding the maximum amount of flow possible in a network (or max-flow) play a central role in computer vision problems. We present an empirical comparison of different max-flow algorithms on modern problems. Our problem instances arise from energy minimization problems in Object Category Segmentation, Image Deconvolution, Super Resolution, Texture Restoration, Character Completion and 3D Segmentation. We compare 14 different implementations and find that the most popularly used implementation of Kolmogorov [5] is no longer the fastest algorithm available, especially for dense graphs.

最大流算法： Matlab



Documentation



CONTENTS

maxflow

R2018a

Maximum flow in graph

[collapse all in page](#)

Syntax

```
mf = maxflow(G,s,t)
mf = maxflow(G,s,t,algorithm)
[mf,GF] = maxflow(__)
[mf,GF,cs,ct] = maxflow(__)
```

Description

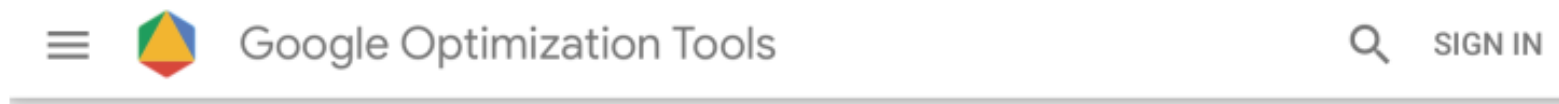
`mf = maxflow(G,s,t)` returns the [maximum flow](#) between nodes `s` and `t`. If graph `G` is unweighted (that is, `G.Edges` does not contain the variable `Weight`), then `maxflow` treats all graph edges as having a weight equal to 1.

[example](#)

`mf = maxflow(G,s,t,algorithm)` specifies the maximum flow algorithm to use. This syntax is only available if `G` is a directed graph.

[example](#)

最大流算法：Google



Products > Optimization > Reference

Contents

Classes



C++ Reference: max_flow

This documentation is automatically generated.

An implementation of a push-relabel algorithm for the max flow problem.

In the following, we consider a graph $G = (V, E, s, t)$ where V denotes the set of nodes (vertices) in the graph, E denotes the set of arcs (edges). s and t denote distinguished nodes in G called source and target. $n = |V|$ denotes the number of nodes in the graph, and $m = |E|$ denotes the number of arcs in the graph.

Each arc (v, w) is associated a capacity $c(v, w)$.