

## 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ ~~*shortest augmenting paths*~~
- ▶ ~~*Dinitz' algorithm*~~
- ▶ ~~*simple unit-capacity networks*~~

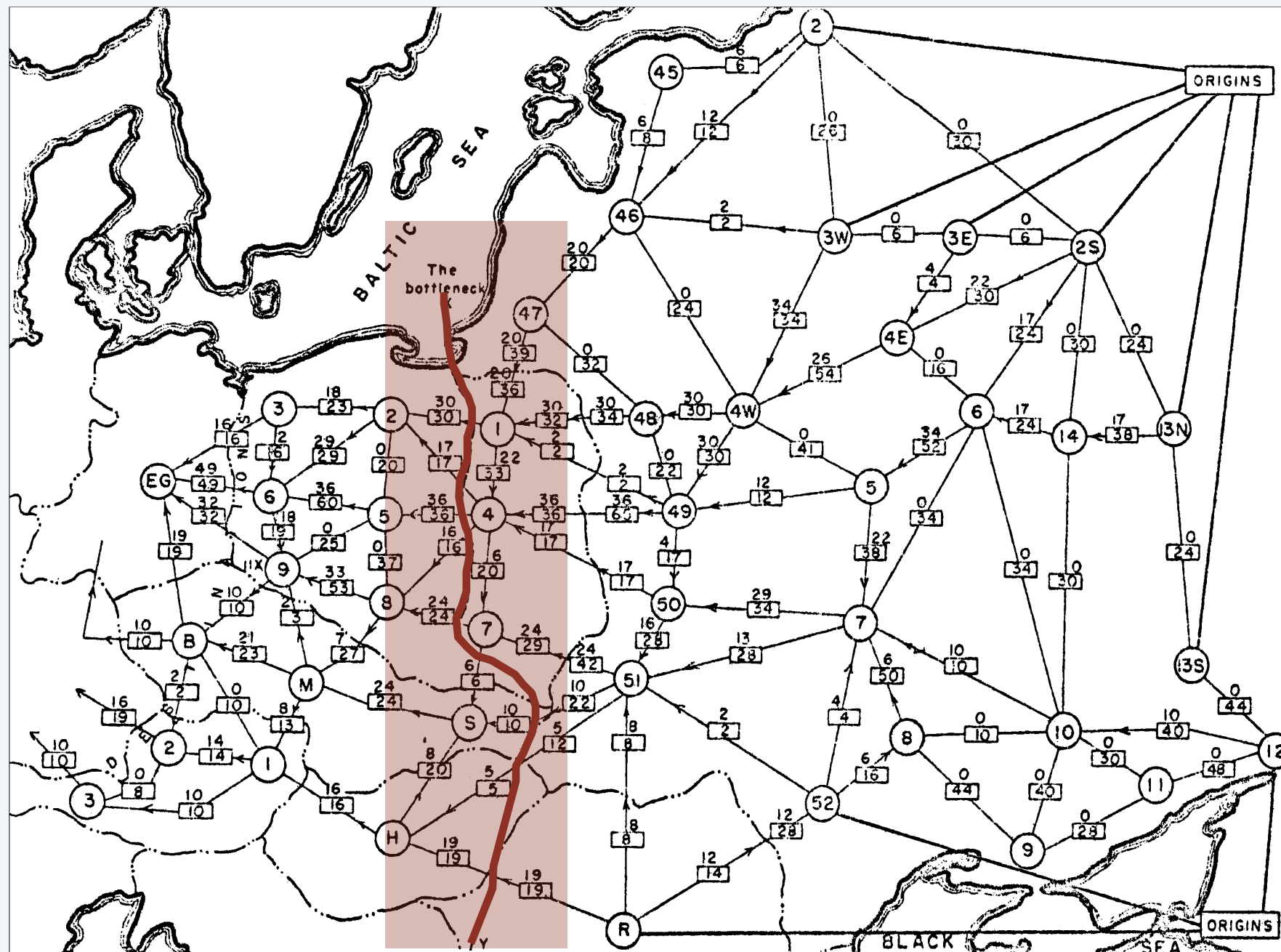
Lecture slides by Kevin Wayne

Copyright © 2005 Pearson–Addison Wesley

<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>

# Minimum cut application (RAND 1950s)

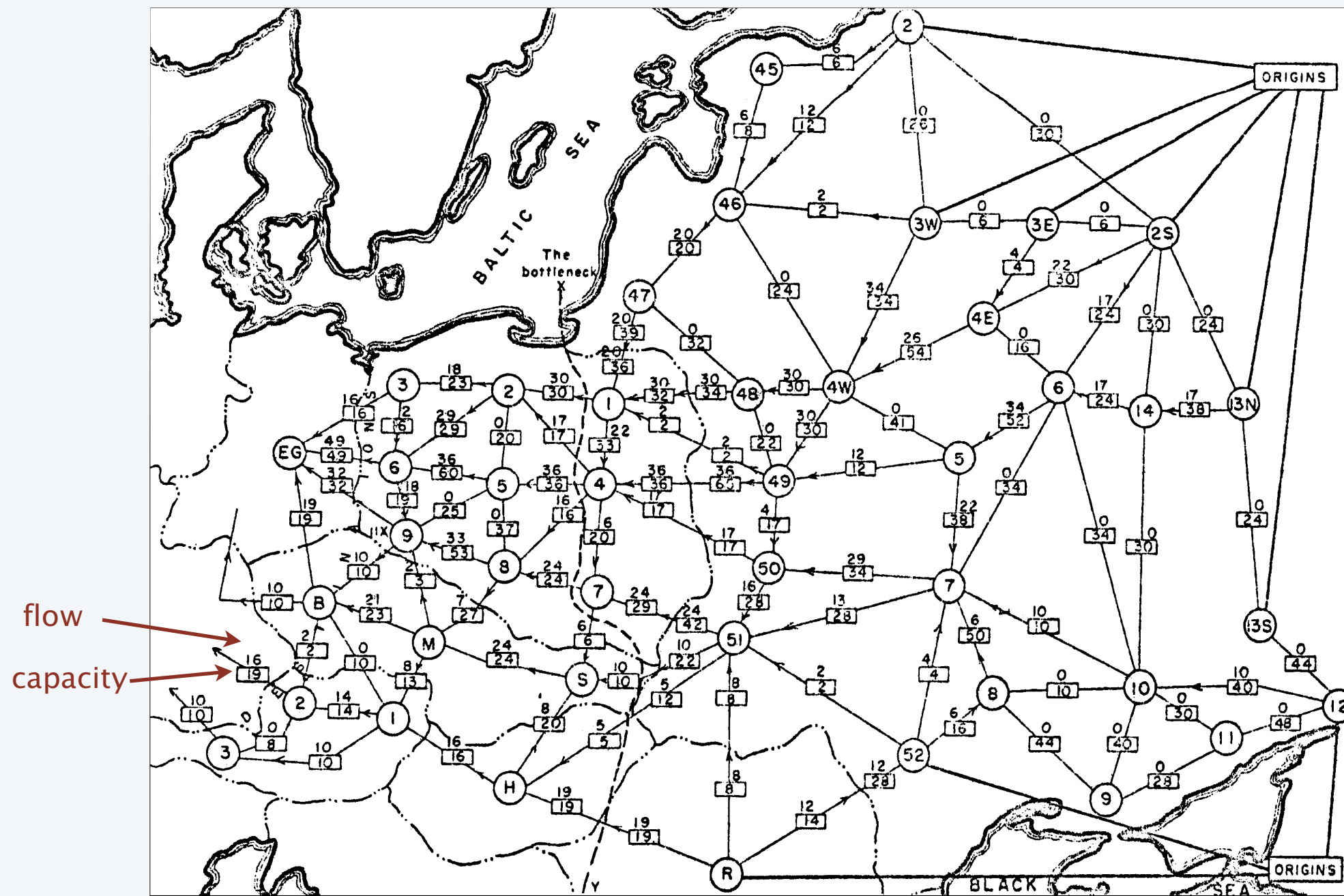
“Free world” goal. Cut supplies (if Cold War turns into real war).



rail network connecting Soviet Union with Eastern European countries  
(map declassified by Pentagon in 1999)

# Maximum flow application (Tolstoï 1930s)

Soviet Union goal. Maximize flow of supplies to Eastern Europe.



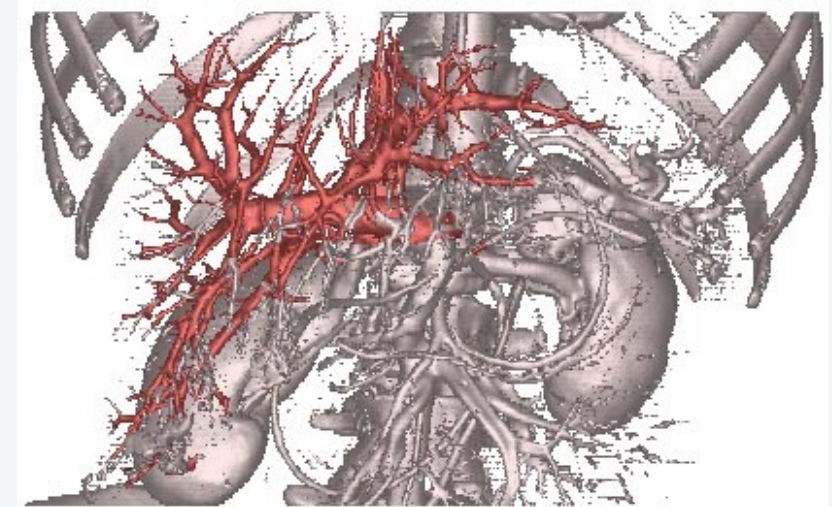
rail network connecting Soviet Union with Eastern European countries  
(map declassified by Pentagon in 1999)

# Max-flow and min-cut applications

---

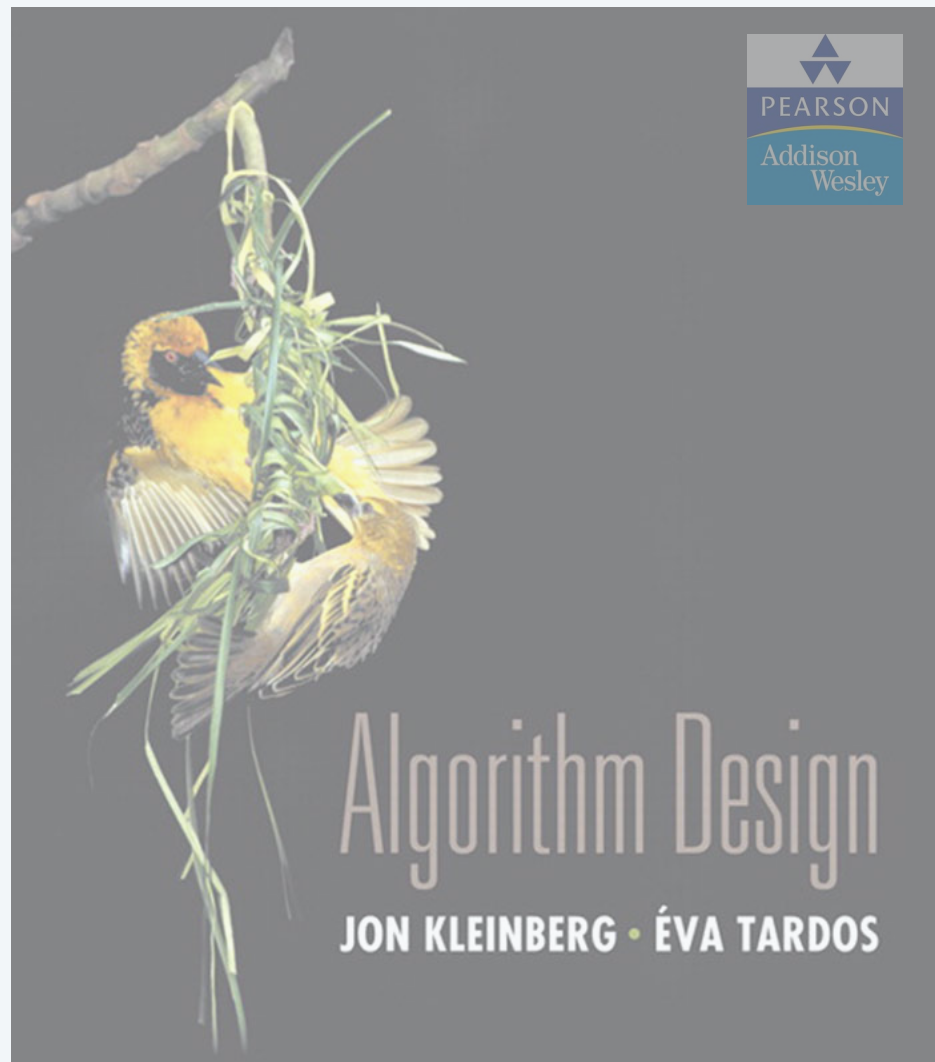
Max-flow and min-cut problems are widely applicable model.

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Markov random fields.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.



**liver and hepatic vascularization segmentation**





## SECTION 7.1

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *Dinitz' algorithm*
- ▶ *simple unit-capacity networks*

# Flow network

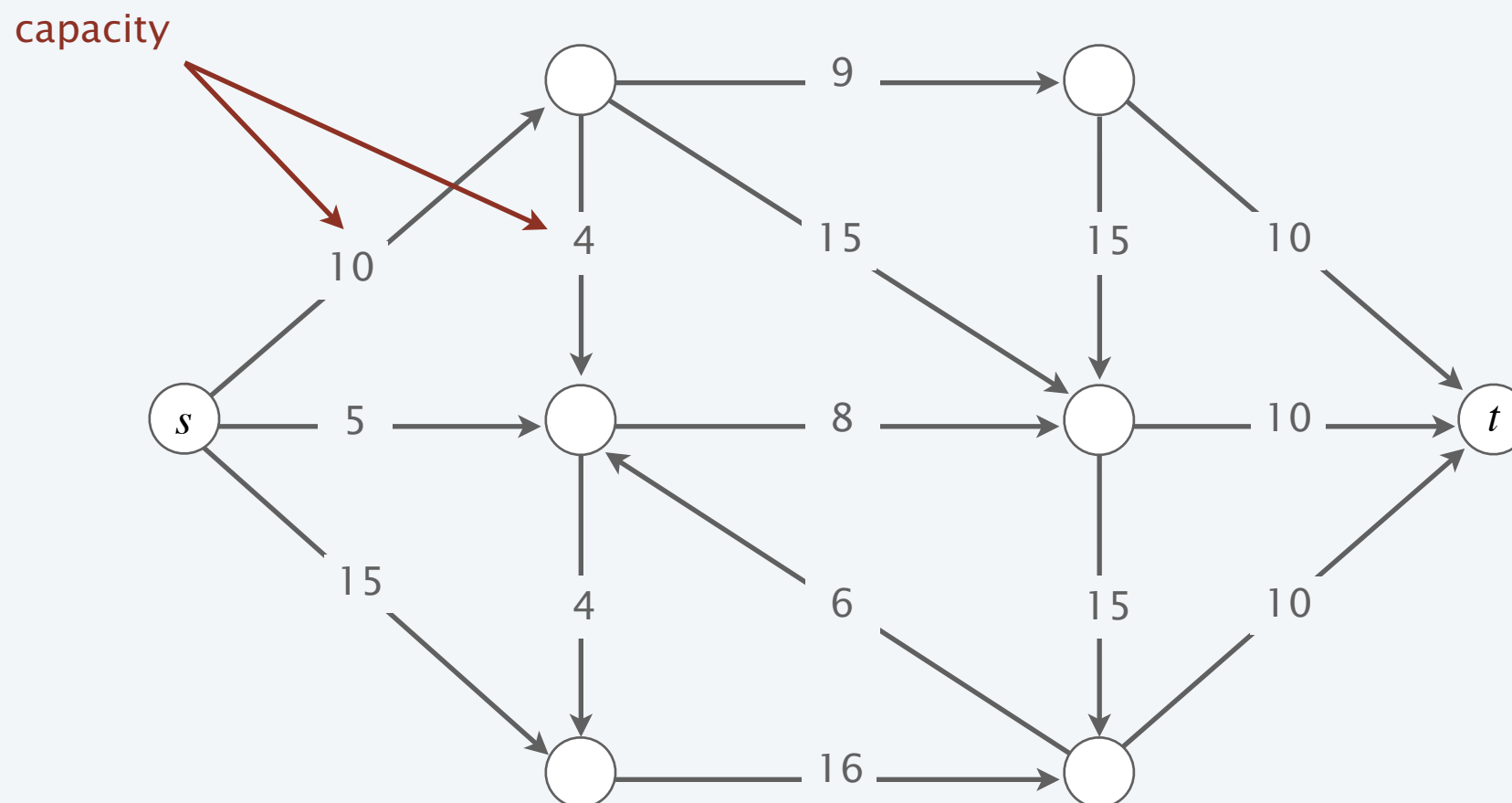
---

A **flow network** is a tuple  $G = (V, E, s, t, c)$ .

- Digraph  $(V, E)$  with source  $s \in V$  and sink  $t \in V$ .
- Capacity  $c(e) > 0$  for each  $e \in E$ .

assume all nodes are reachable from  $s$

**Intuition.** Material flowing through a transportation network; material originates at source and is sent to sink.



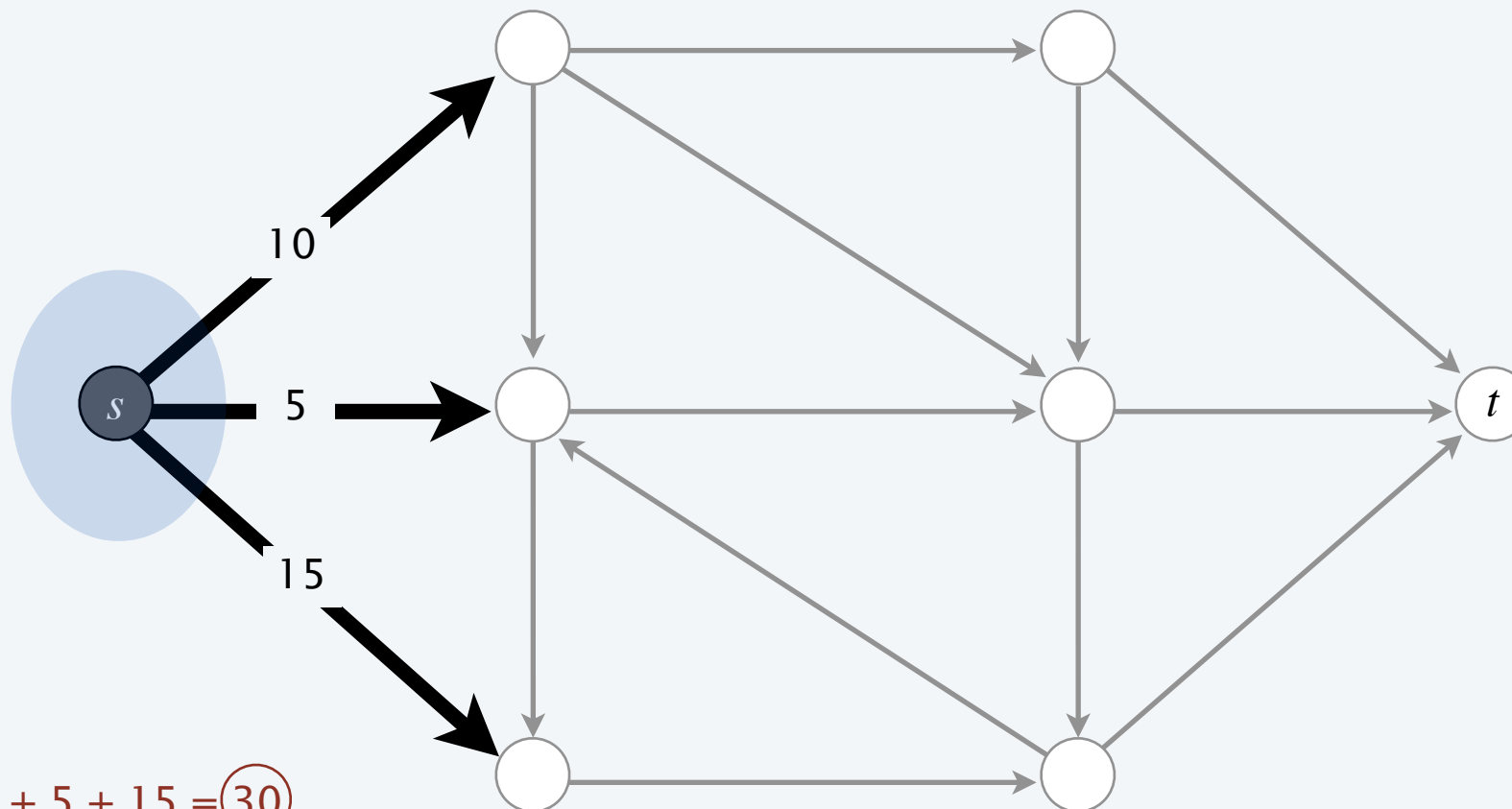
# Minimum-cut problem

---

**Def.** An *st-cut (cut)* is a partition  $(A, B)$  of the nodes with  $s \in A$  and  $t \in B$ .

**Def.** Its *capacity* is the sum of the capacities of the edges from  $A$  to  $B$ .

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



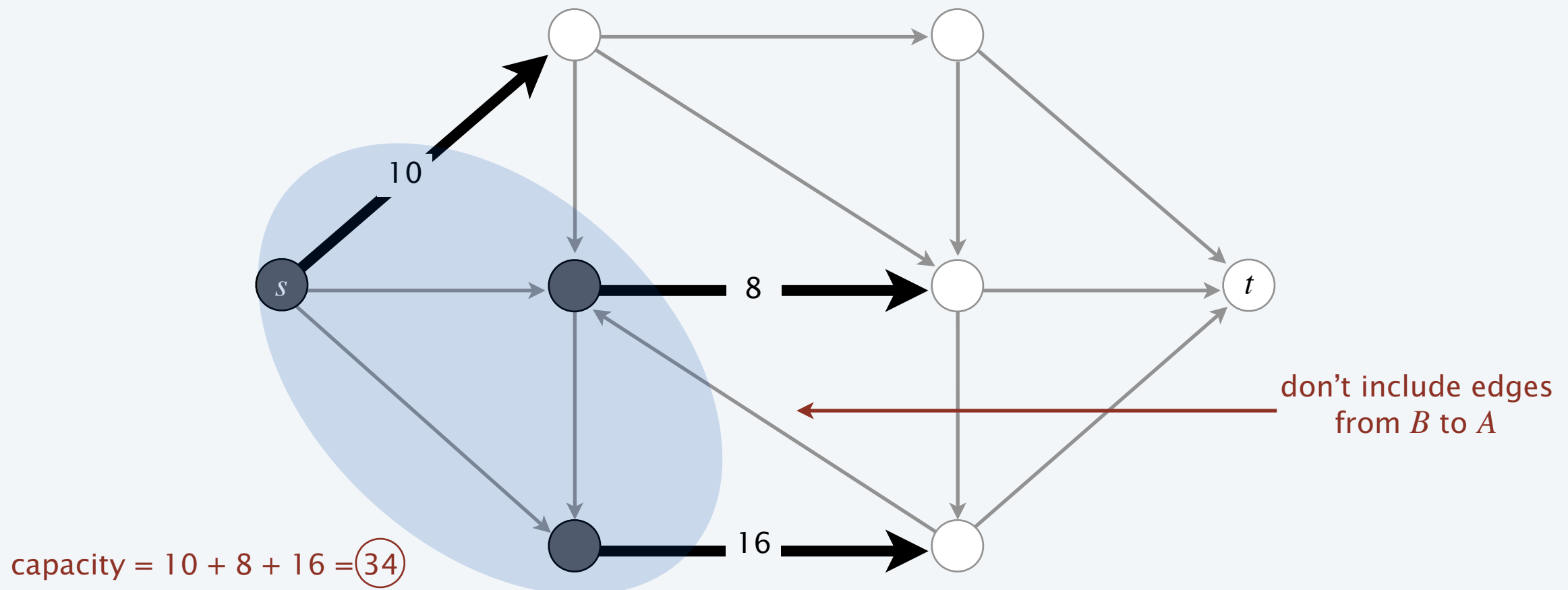
capacity =  $10 + 5 + 15 = 30$

# Minimum-cut problem

Def. An *st-cut (cut)* is a partition  $(A, B)$  of the nodes with  $s \in A$  and  $t \in B$ .

Def. Its *capacity* is the sum of the capacities of the edges from  $A$  to  $B$ .

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





# Minimum-cut problem

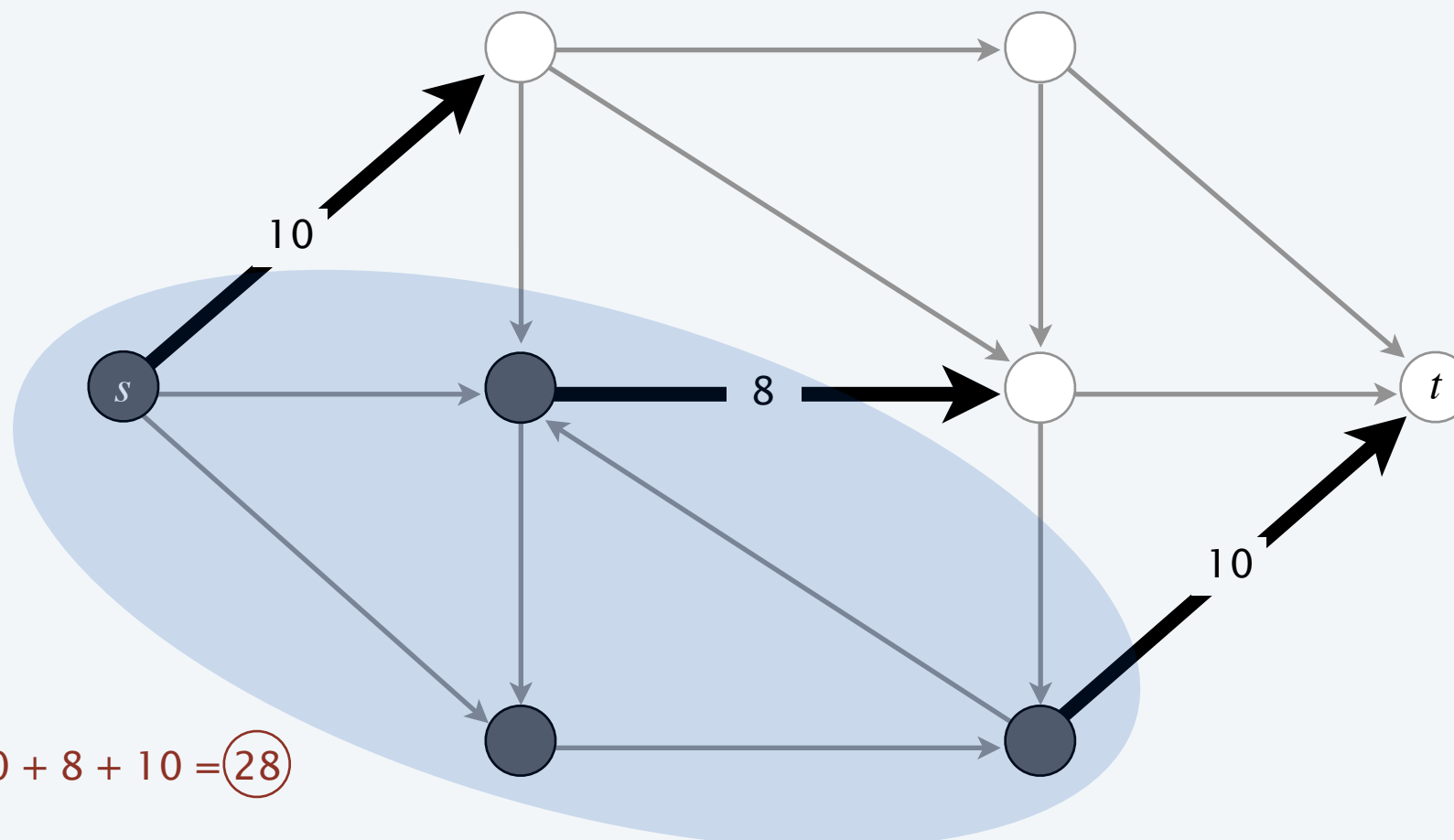
---

**Def.** An *st-cut (cut)* is a partition  $(A, B)$  of the nodes with  $s \in A$  and  $t \in B$ .

**Def.** Its *capacity* is the sum of the capacities of the edges from  $A$  to  $B$ .

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

**Min-cut problem.** Find a cut of minimum capacity.

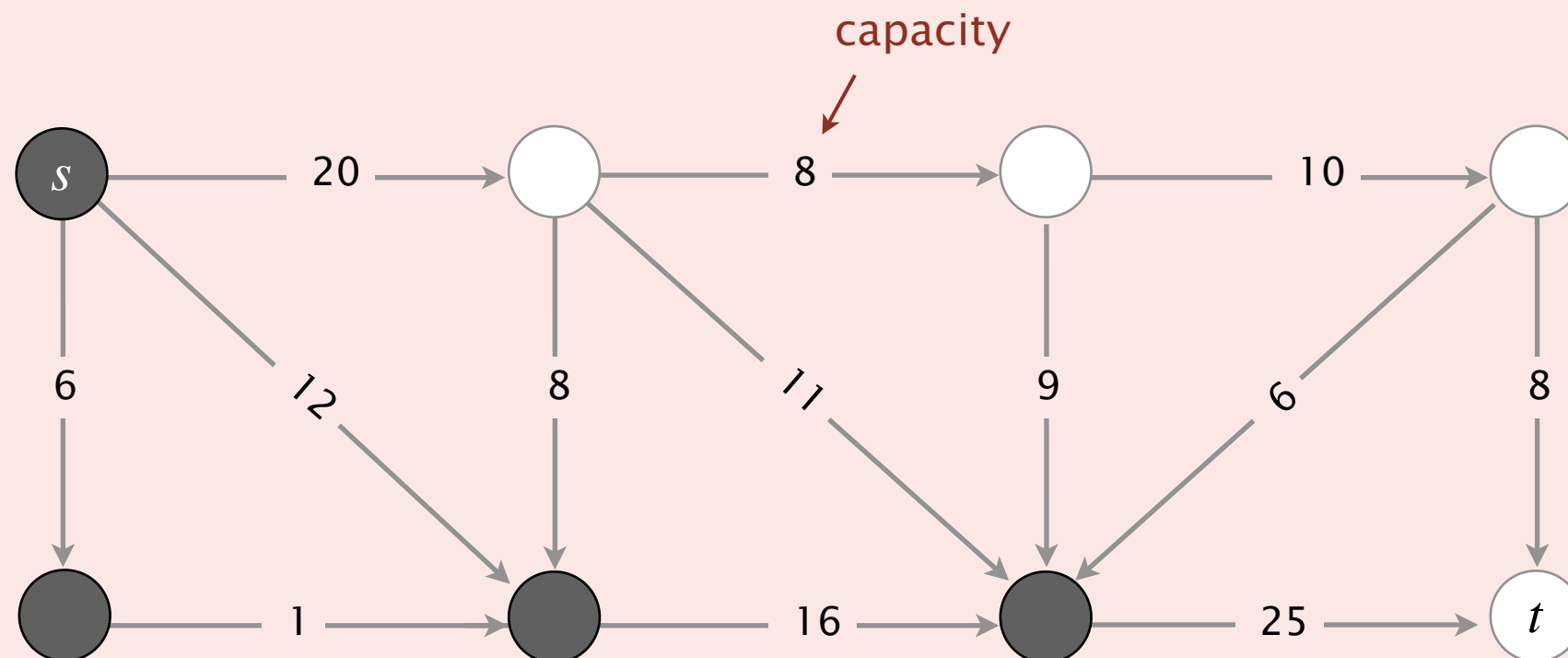


capacity =  $10 + 8 + 10 = 28$



Which is the capacity of the given  $st$ -cut?

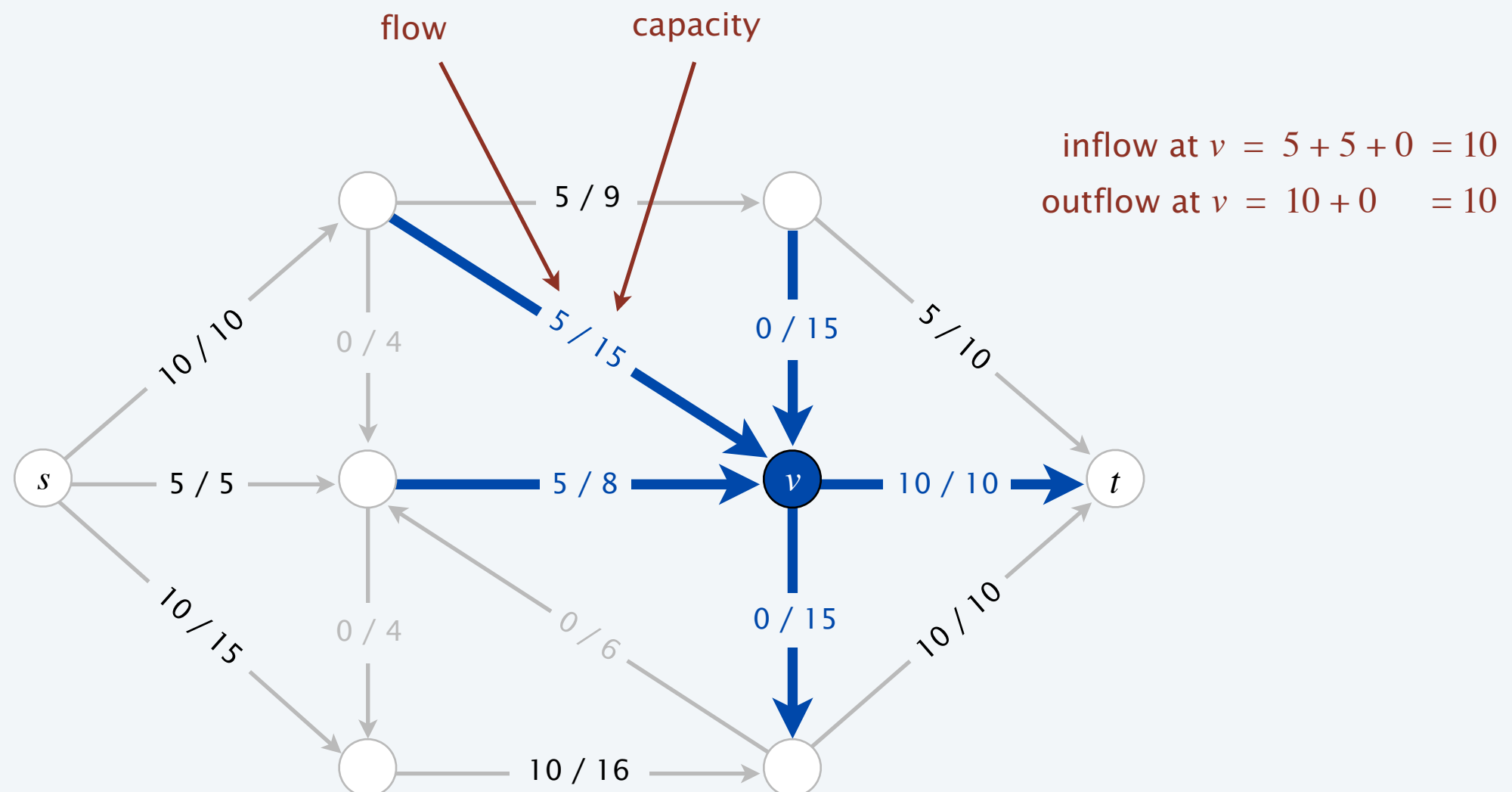
- A. 11 ( $20 + 25 - 8 - 11 - 9 - 6$ )
- B. 34 ( $8 + 11 + 9 + 6$ )
- C. 45 ( $20 + 25$ )
- D. 79 ( $20 + 25 + 8 + 11 + 9 + 6$ )



# Maximum-flow problem

**Def.** An *st-flow* (flow)  $f$  is a function that satisfies:

- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  [capacity]
- For each  $v \in V - \{s, t\}$ :  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$  [flow conservation]

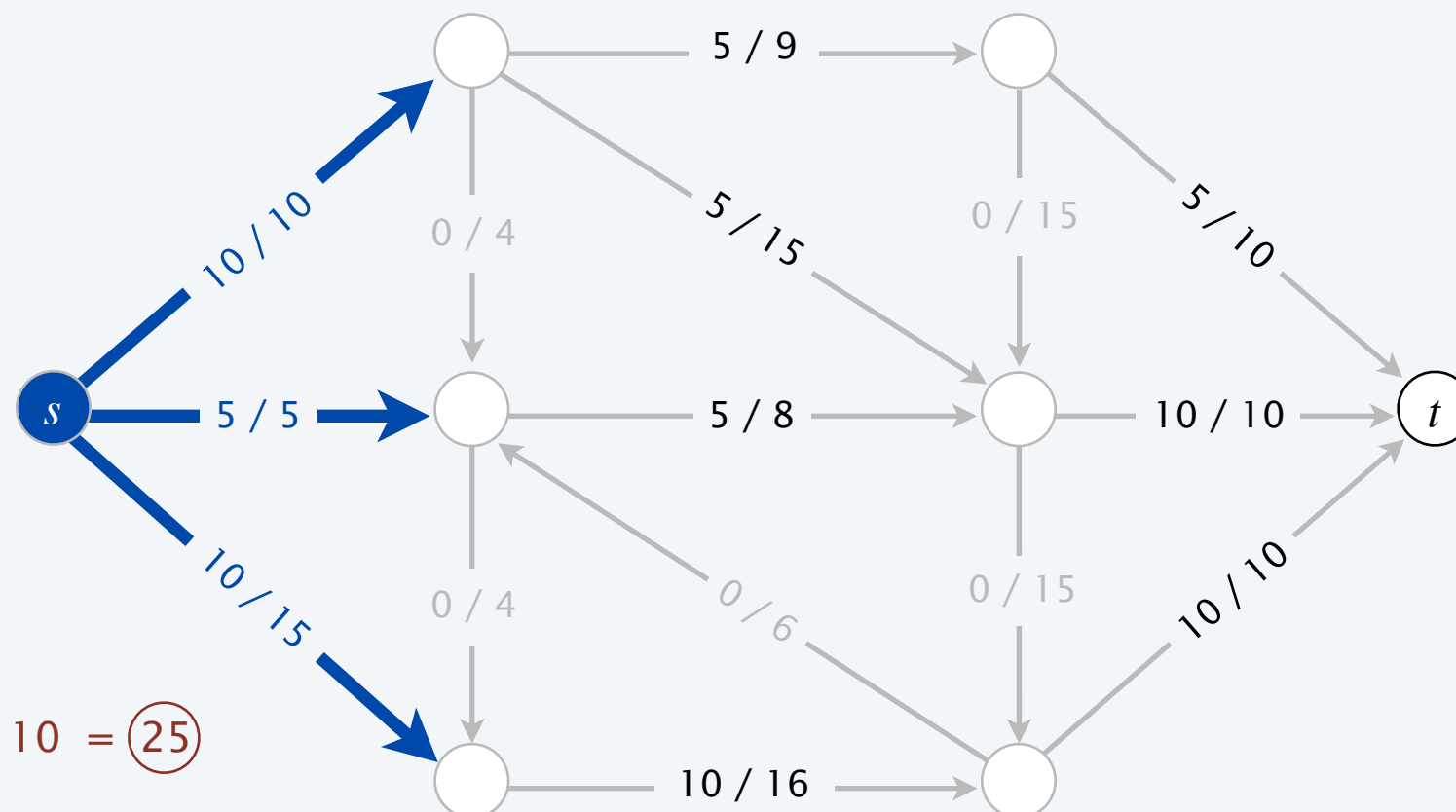


# Maximum-flow problem

Def. An *st-flow* (flow)  $f$  is a function that satisfies:

- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  [capacity]
- For each  $v \in V - \{s, t\}$ :  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$  [flow conservation]

Def. The *value* of a flow  $f$  is:  $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$



value =  $5 + 10 + 10 = 25$

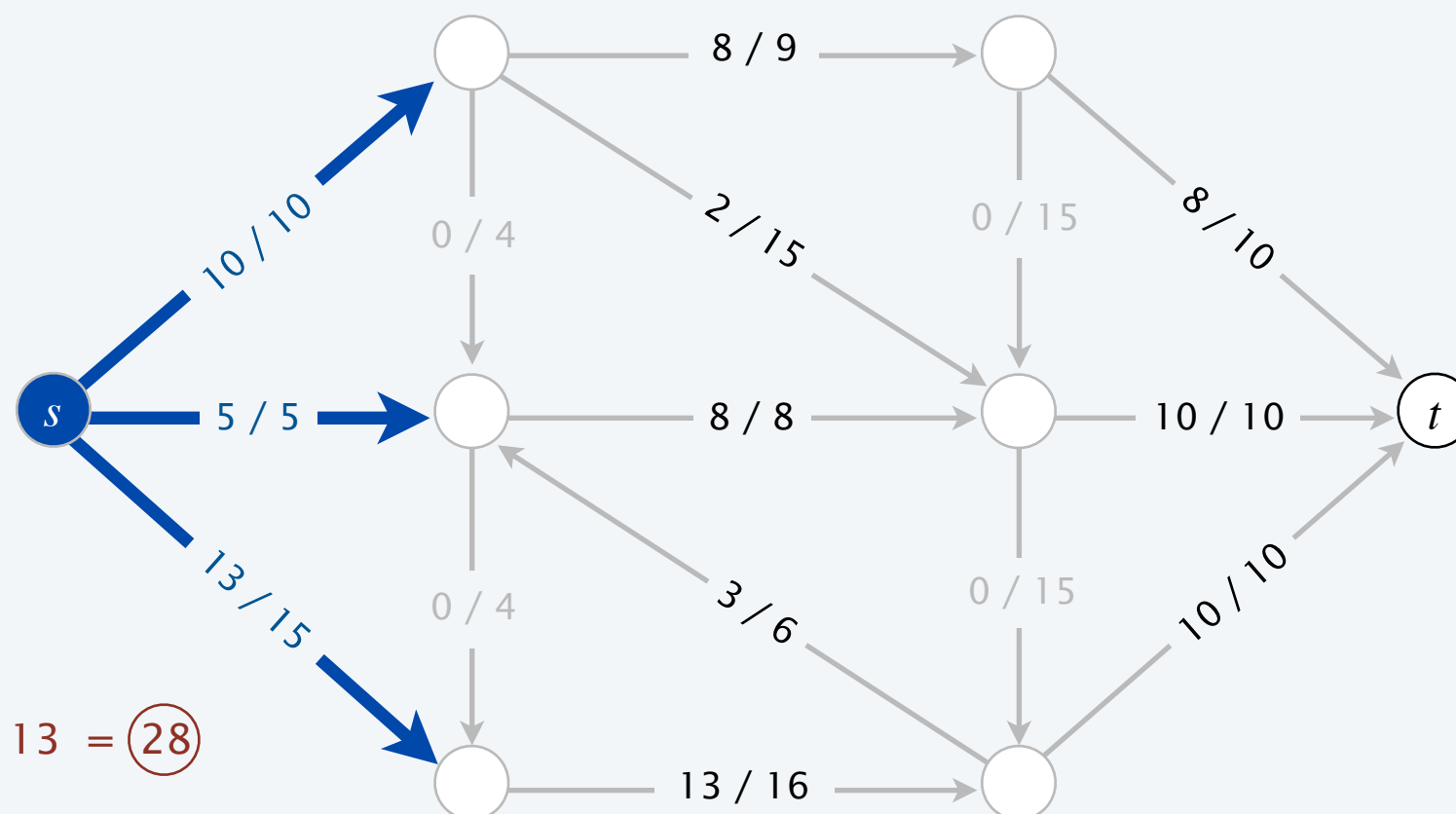
# Maximum-flow problem

Def. An *st-flow (flow)*  $f$  is a function that satisfies:

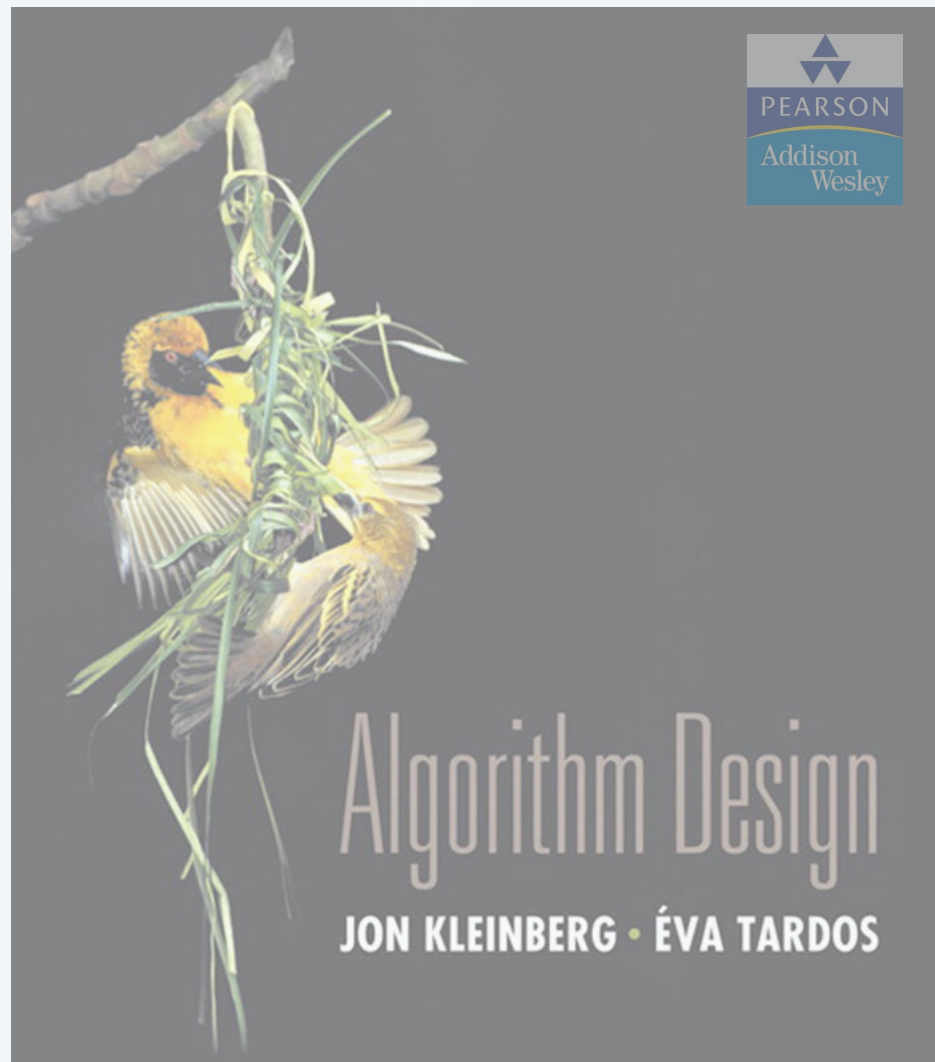
- For each  $e \in E$ :  $0 \leq f(e) \leq c(e)$  [capacity]
- For each  $v \in V - \{s, t\}$ :  $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$  [flow conservation]

Def. The *value* of a flow  $f$  is:  $val(f) = \sum_{e \text{ out of } s} f(e) - \sum_{e \text{ in to } s} f(e)$

Max-flow problem. Find a flow of maximum value.



$$\text{value} = 10 + 5 + 13 = \textcircled{28}$$



## SECTION 7.1

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ **Ford–Fulkerson algorithm**
- ▶ *max-flow min-cut theorem*
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *Dinitz' algorithm*
- ▶ *simple unit-capacity networks*

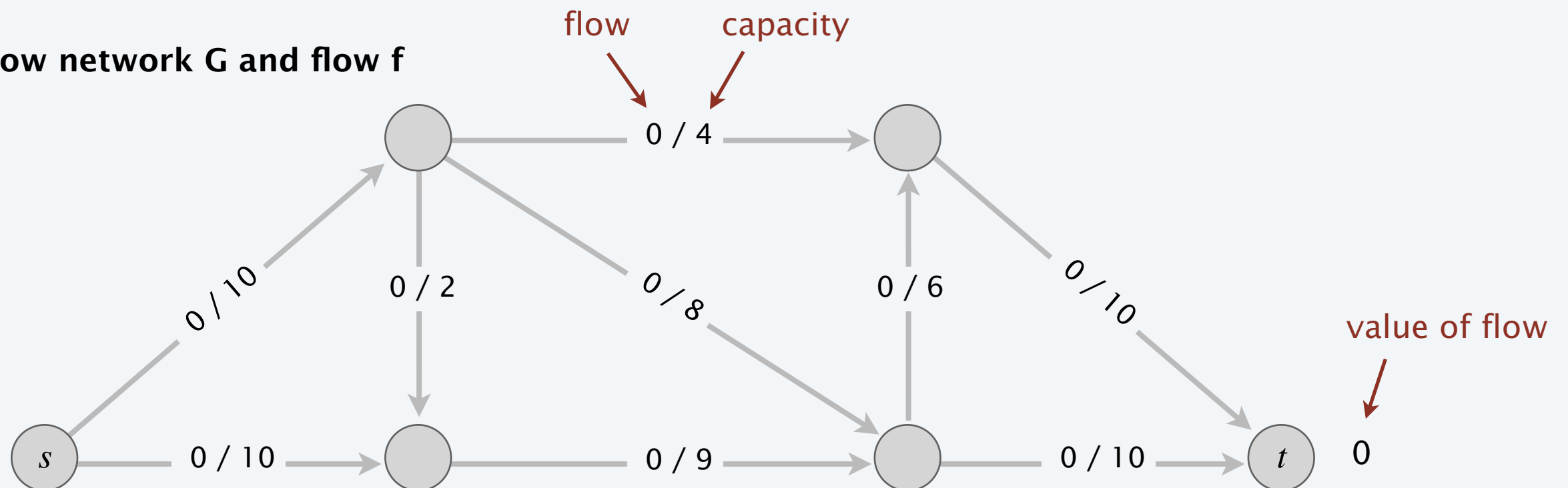


# Toward a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \leadsto t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

flow network  $G$  and flow  $f$



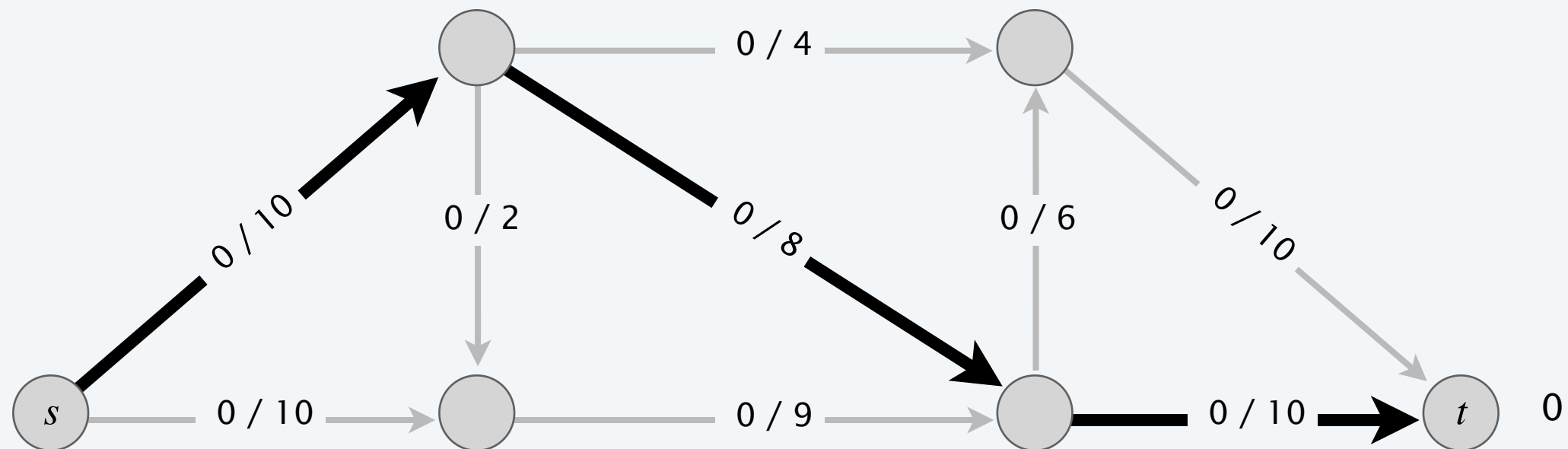
# Toward a max-flow algorithm

---

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \rightsquigarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

flow network  $G$  and flow  $f$

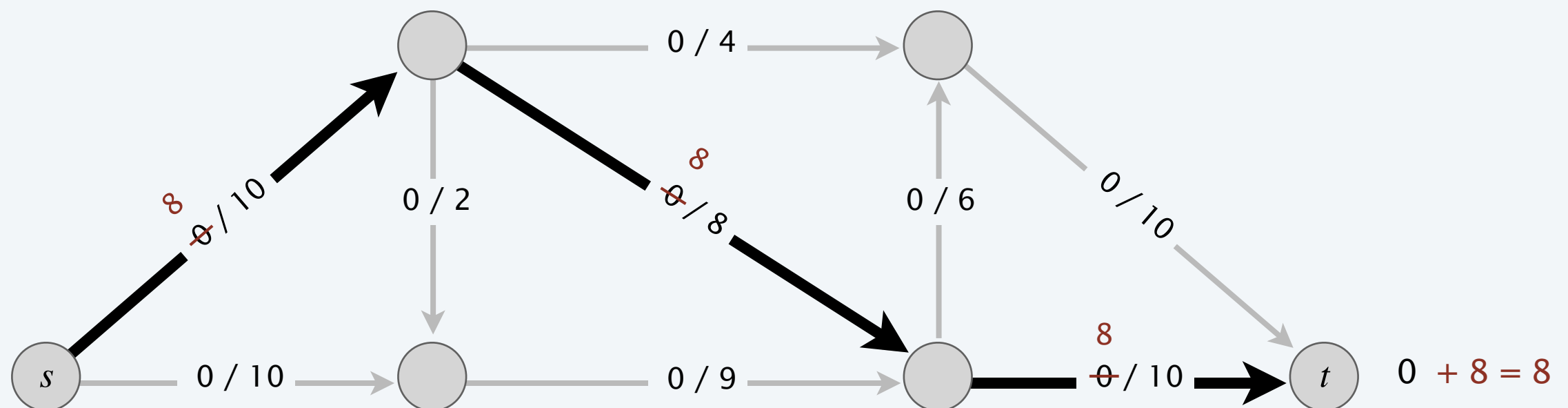


# Toward a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \leadsto t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

flow network  $G$  and flow  $f$

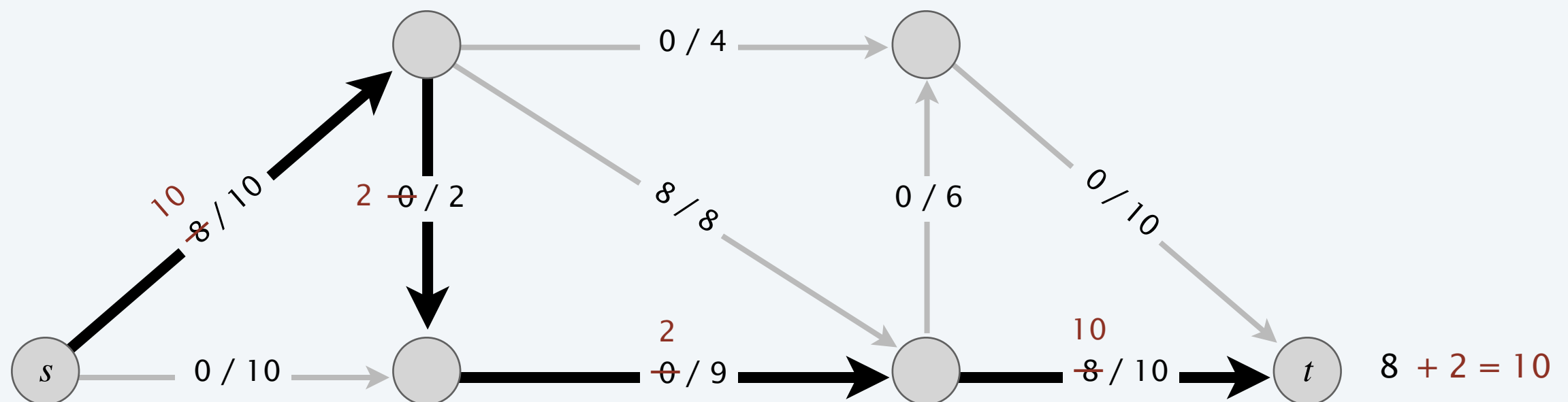


# Toward a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \rightsquigarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

flow network  $G$  and flow  $f$

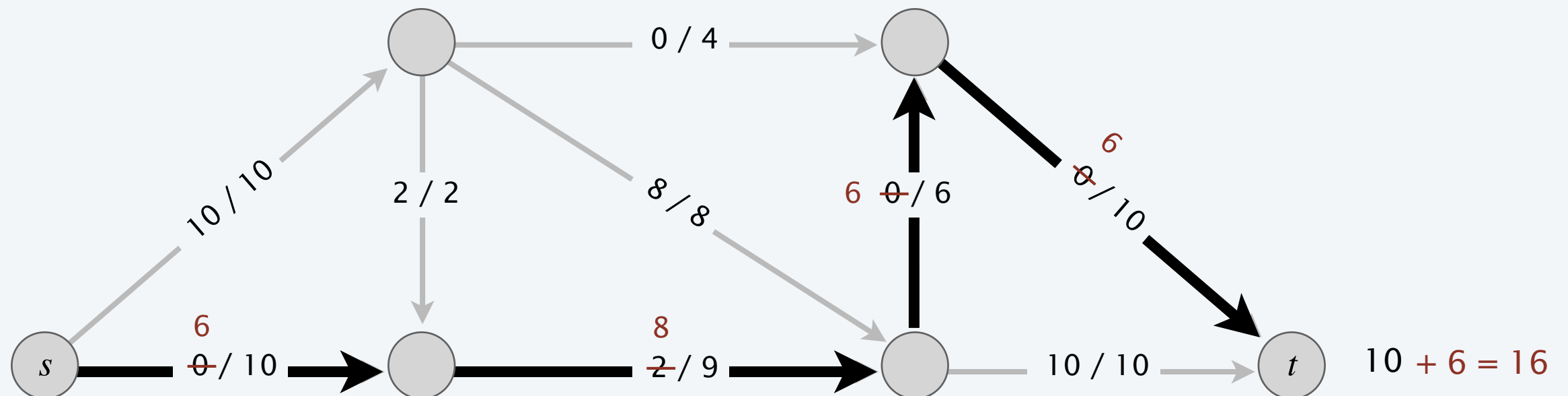


# Toward a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \leadsto t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

flow network  $G$  and flow  $f$



# Toward a max-flow algorithm

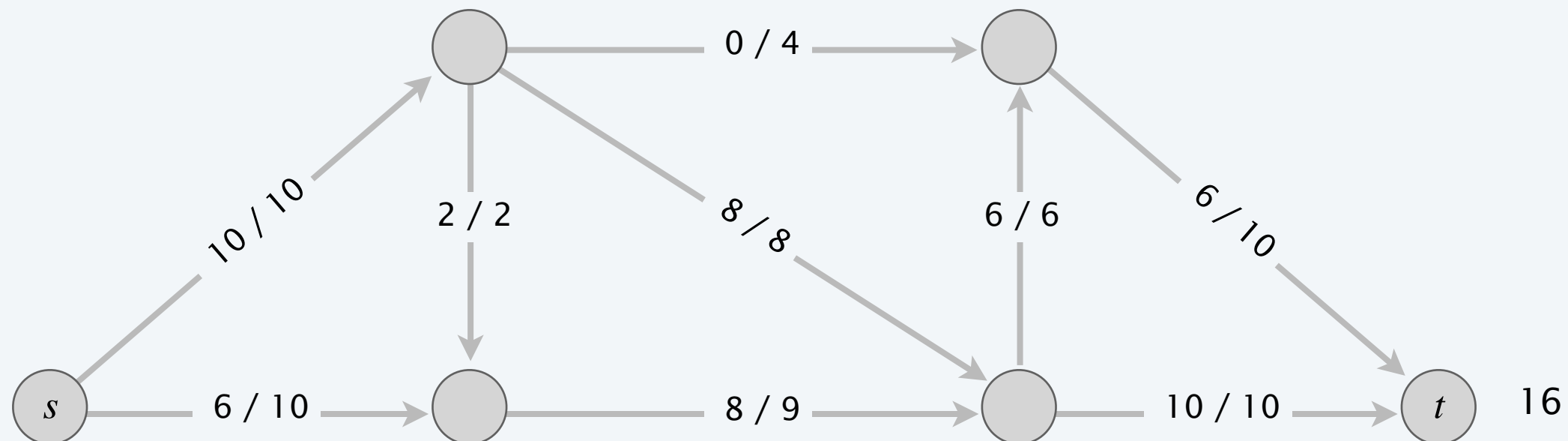
---

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \leadsto t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

ending flow value = 16

flow network  $G$  and flow  $f$





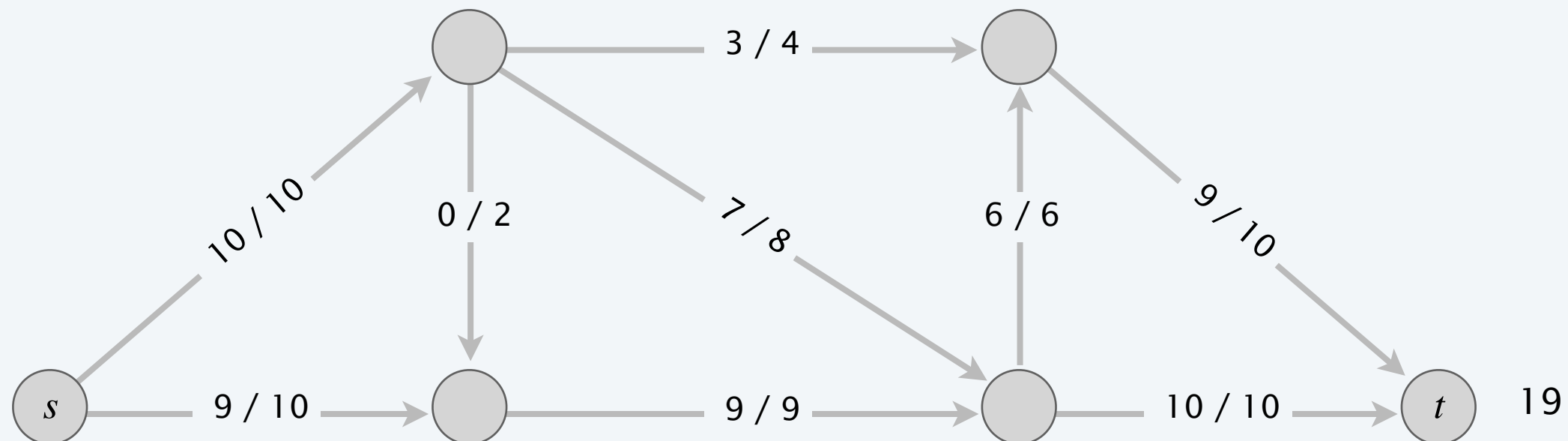
# Toward a max-flow algorithm

## Greedy algorithm.

- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \rightsquigarrow t$  path  $P$  where each edge has  $f(e) < c(e)$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

**but max-flow value = 19**

**flow network G and flow f**



# Why the greedy algorithm fails

---

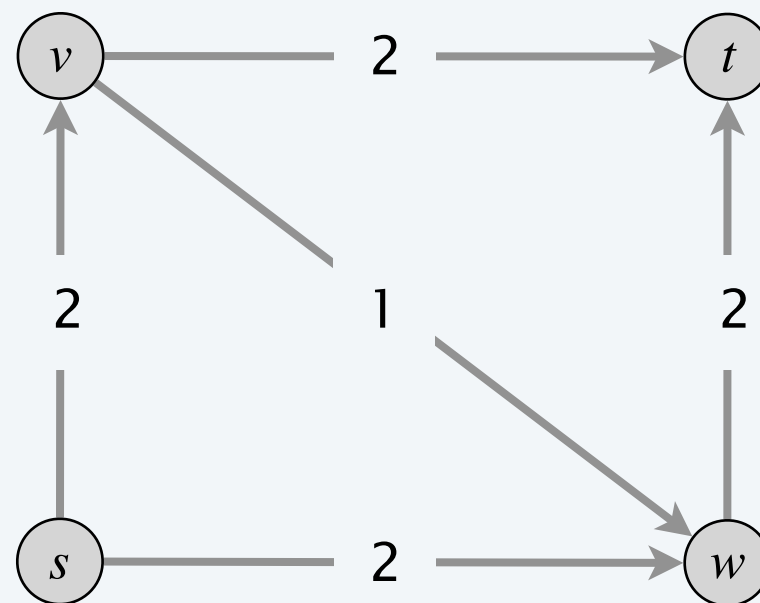
**Q.** Why does the greedy algorithm fail?

**A.** Once greedy algorithm increases flow on an edge, it never decreases it.

**Ex.** Consider flow network  $G$ .

- The unique max flow has  $f^*(v, w) = 0$ .
- Greedy algorithm could choose  $s \rightarrow v \rightarrow w \rightarrow t$  as first augmenting path.

flow network  $G$



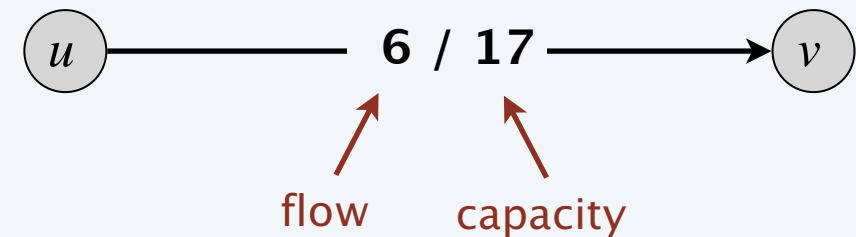
**Bottom line.** Need some mechanism to “undo” a bad decision.

# Residual network

**Original edge.**  $e = (u, v) \in E$ .

- Flow  $f(e)$ .
- Capacity  $c(e)$ .

**original flow network G**



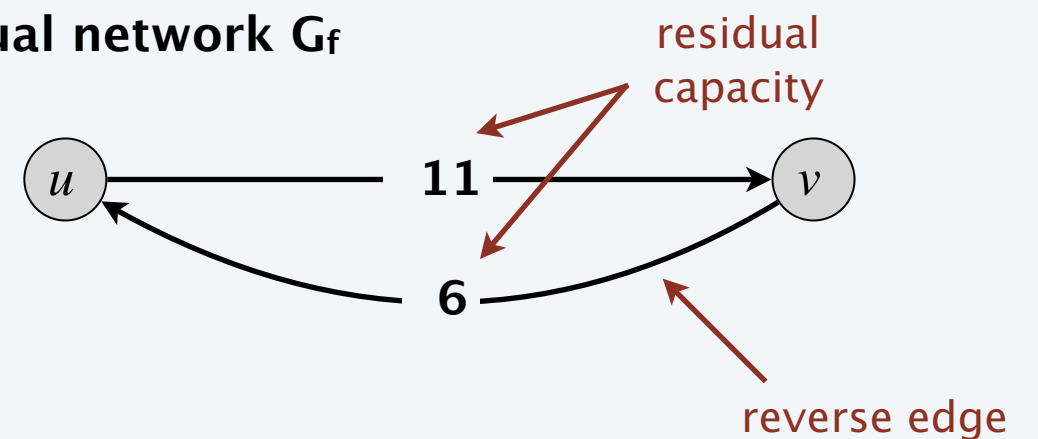
**Reverse edge.**  $e^{\text{reverse}} = (v, u)$ .

- “Undo” flow sent.

**Residual capacity.**

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

**residual network  $G_f$**



**Residual network.**  $G_f = (V, E_f, s, t, c_f)$ .

- $E_f = \{e : f(e) < c(e)\} \cup \{e^{\text{reverse}} : f(e) > 0\}$ .
- Key property:  $f'$  is a flow in  $G_f$  iff  $f + f'$  is a flow in  $G$ .

edges with positive  
residual capacity

where flow on a reverse edge  
negates flow on  
corresponding forward edge

# Augmenting path

---

**Def.** An **augmenting path** is a simple  $s \rightarrow t$  path in the residual network  $G_f$ .

**Def.** The **bottleneck capacity** of an augmenting path  $P$  is the minimum residual capacity of any edge in  $P$ .

**Key property.** Let  $f$  be a flow and let  $P$  be an augmenting path in  $G_f$ . Then, after calling  $f' \leftarrow \text{AUGMENT}(f, c, P)$ , the resulting  $f'$  is a flow and  $\text{val}(f') = \text{val}(f) + \text{bottleneck}(G_f, P)$ .

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

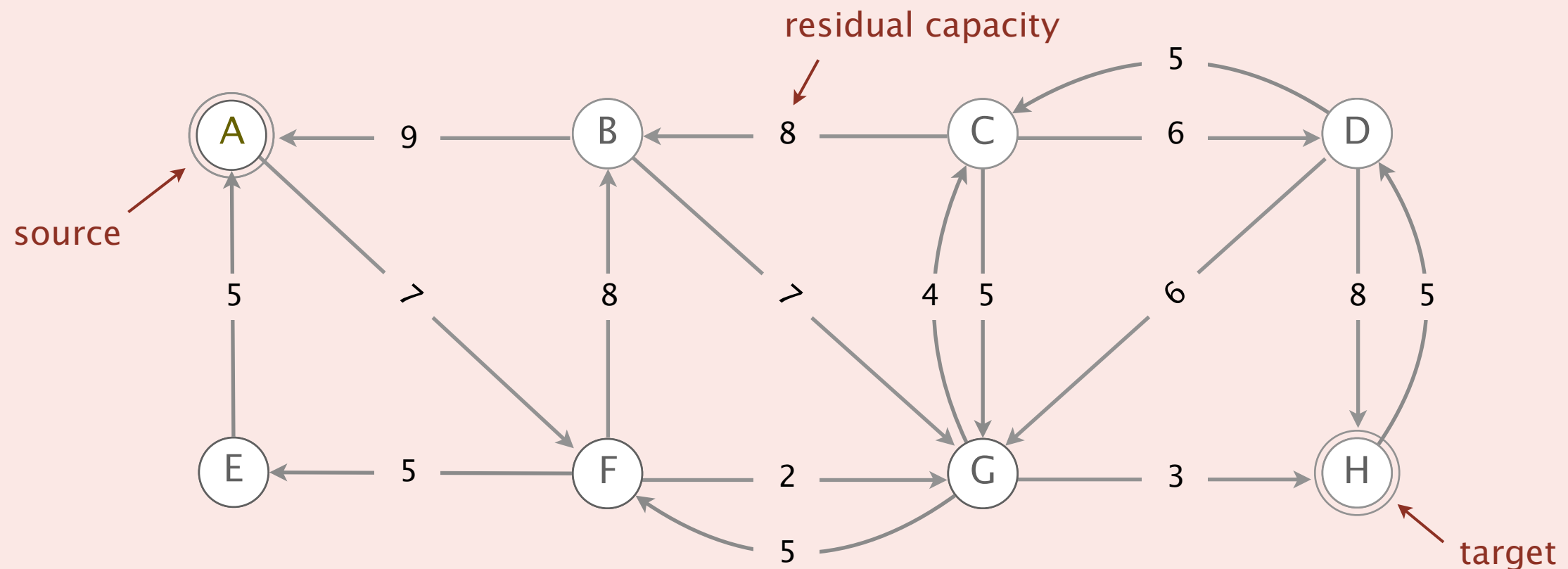
**RETURN**  $f$ .

---



Which is the augmenting path of highest bottleneck capacity?

- 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 algorithm

---

## Ford–Fulkerson augmenting path algorithm.



- Start with  $f(e) = 0$  for each edge  $e \in E$ .
- Find an  $s \rightsquigarrow t$  path  $P$  in the residual network  $G_f$ .
- Augment flow along path  $P$ .
- Repeat until you get stuck.

### 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  $G_f$ .

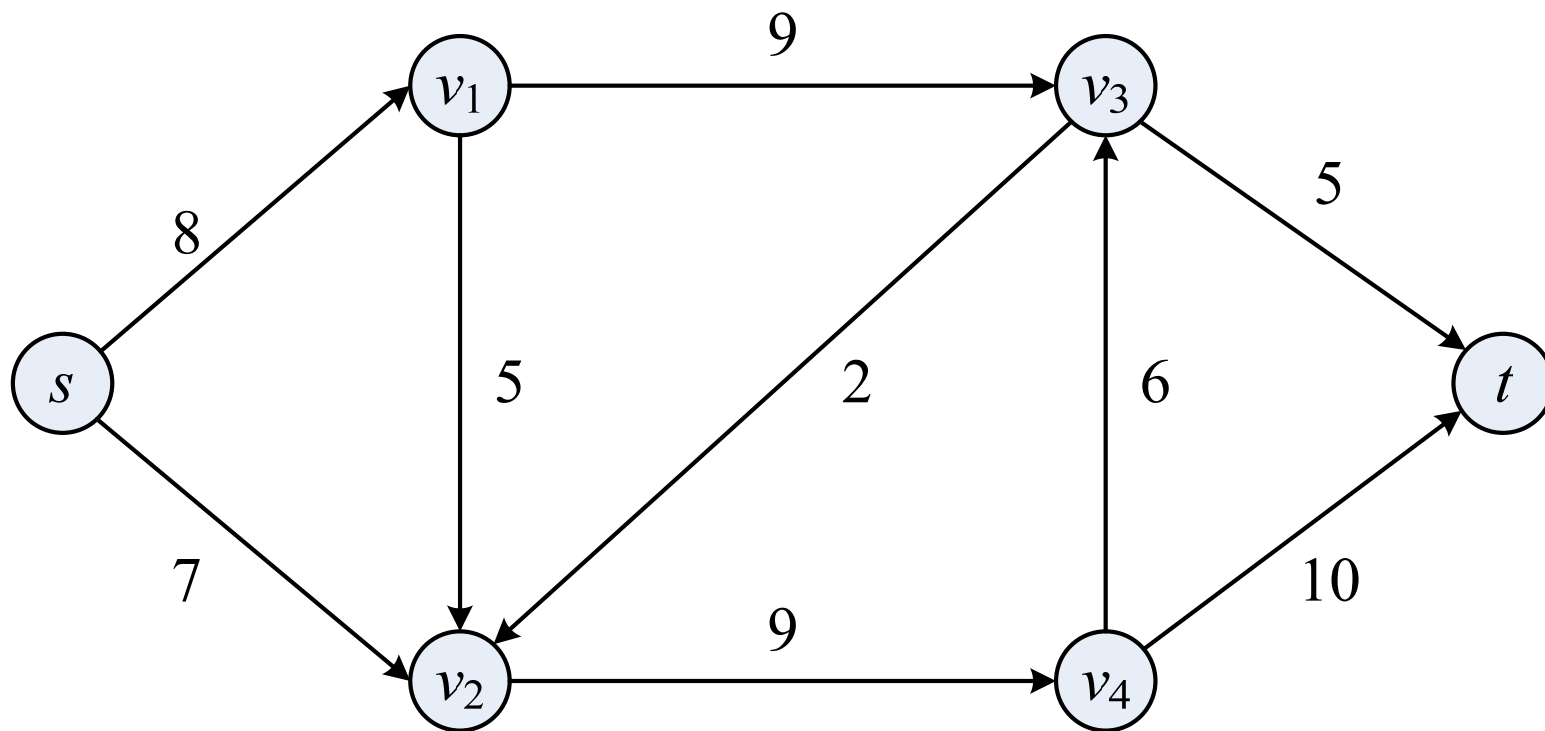
RETURN  $f$ .

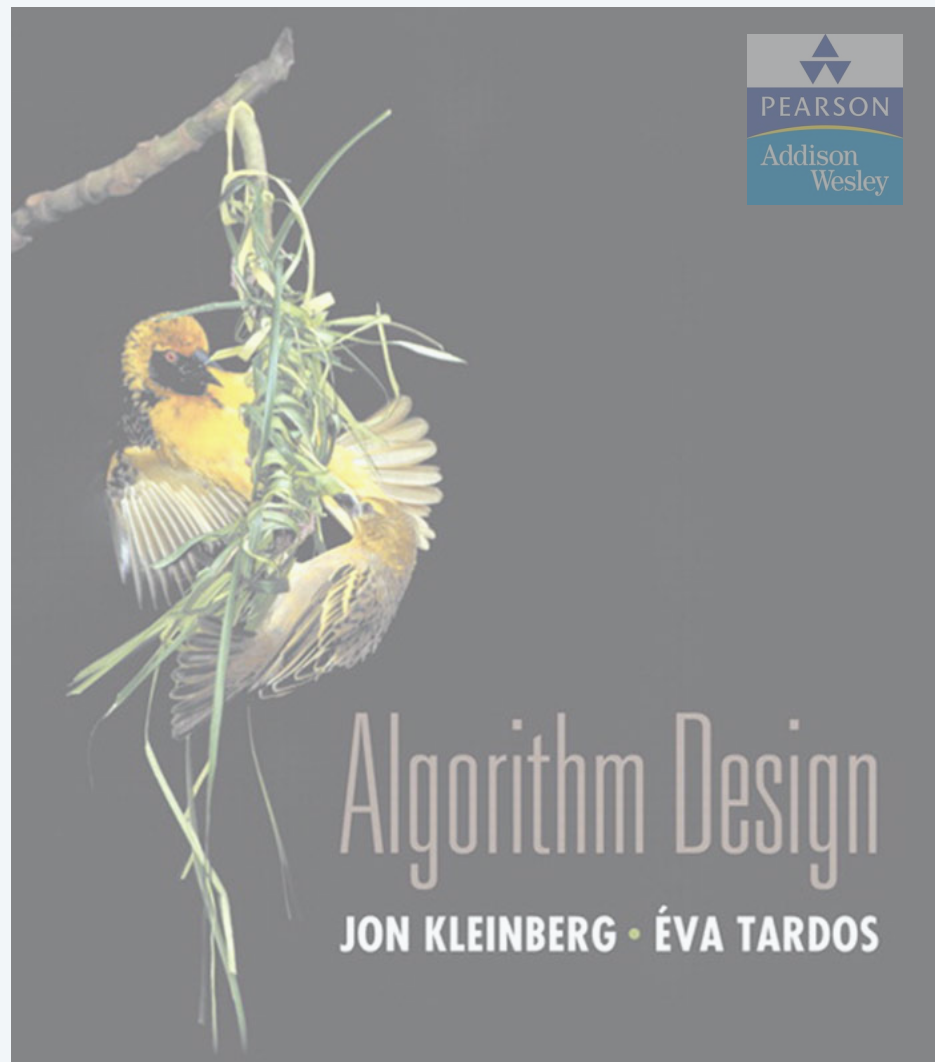
augmenting path

A red arrow pointing from the text 'augmenting path' to the variable  $P$  in the line 'WHILE (there exists an  $s \rightsquigarrow t$  path  $P$  in  $G_f$ )'.



Practice: solve the max flow problem of the next network





## SECTION 7.2

# 7. NETWORK FLOW I

---

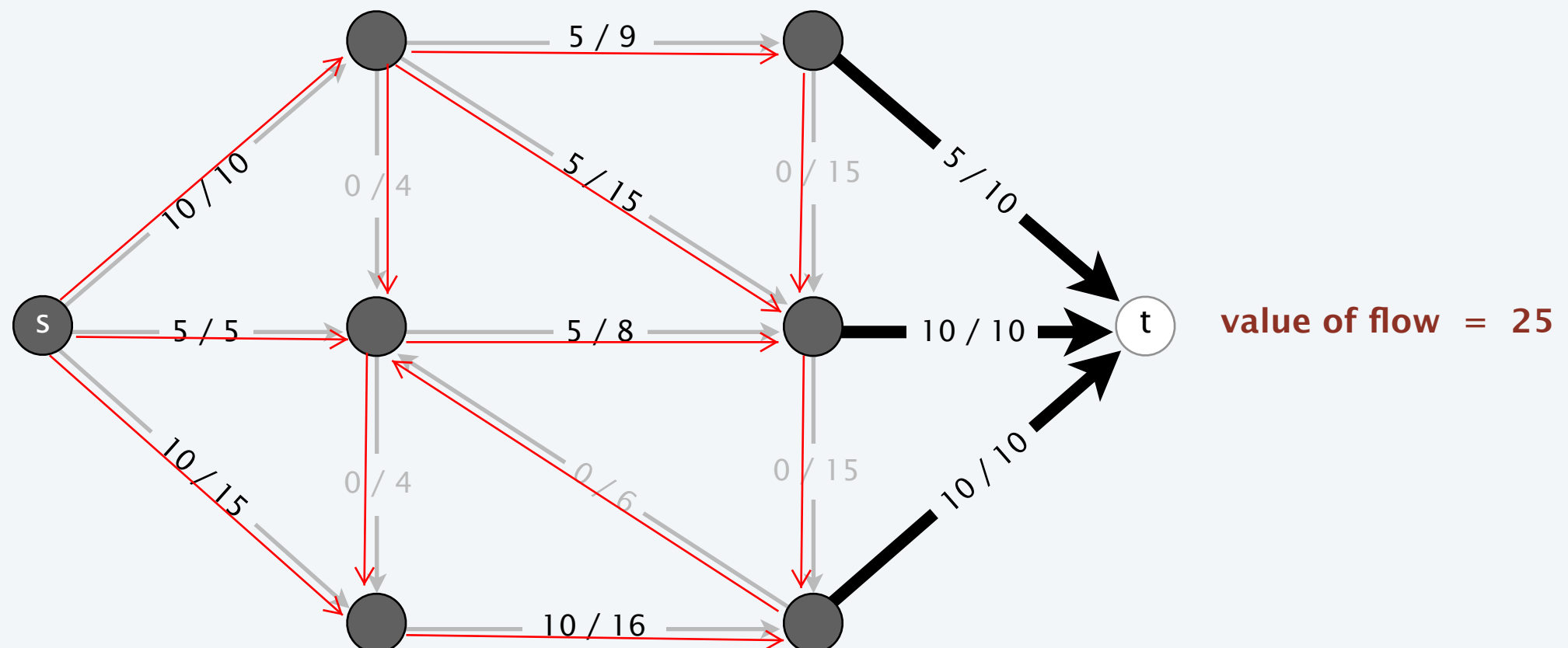
- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ ***max-flow min-cut theorem***
- ▶ *capacity-scaling algorithm*
- ▶ *shortest augmenting paths*
- ▶ *Dinitz' algorithm*
- ▶ *simple unit-capacity networks*

# Relationship between flows and cuts

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the value of the flow  $f$  equals the net flow across the cut  $(A, B)$ .

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

**net flow across cut = 5 + 10 + 10 = 25**

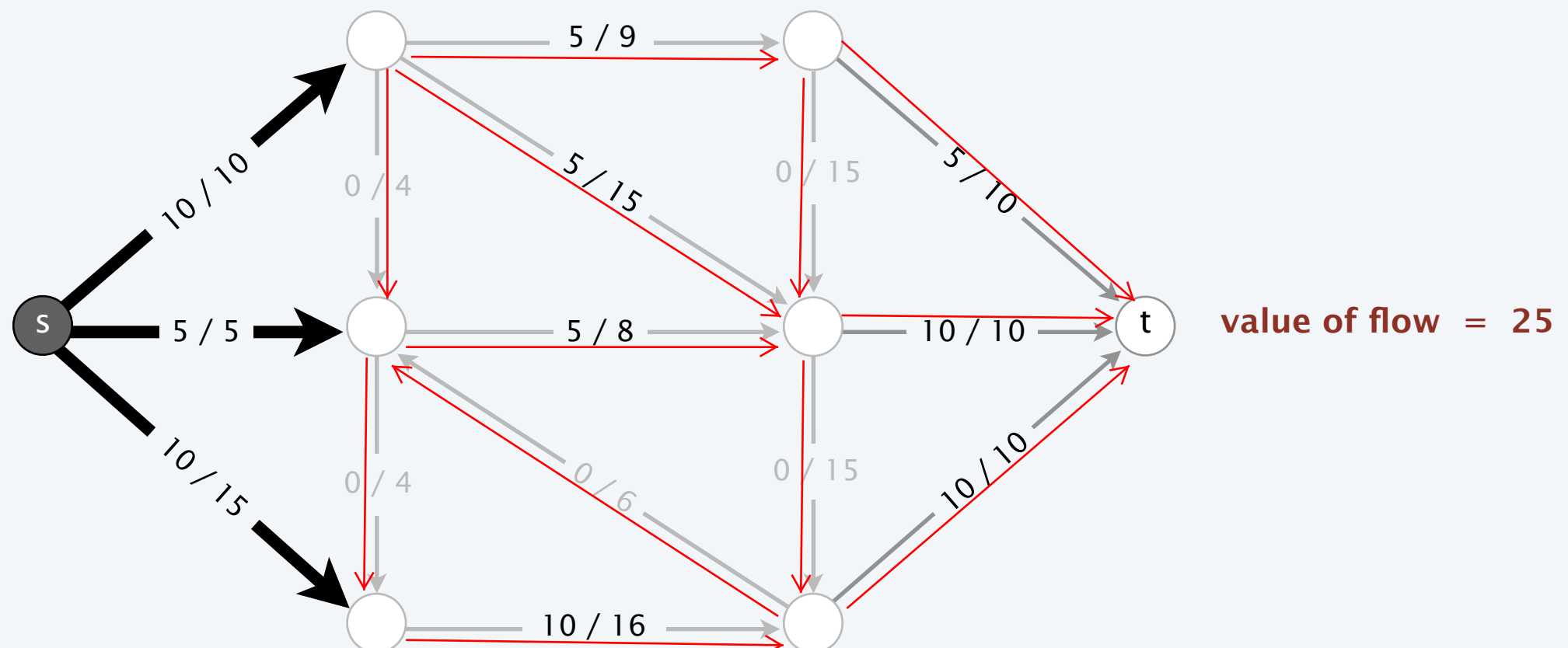


# Relationship between flows and cuts

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the value of the flow  $f$  equals the net flow across the cut  $(A, B)$ .

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

**net flow across cut = 10 + 5 + 10 = 25**

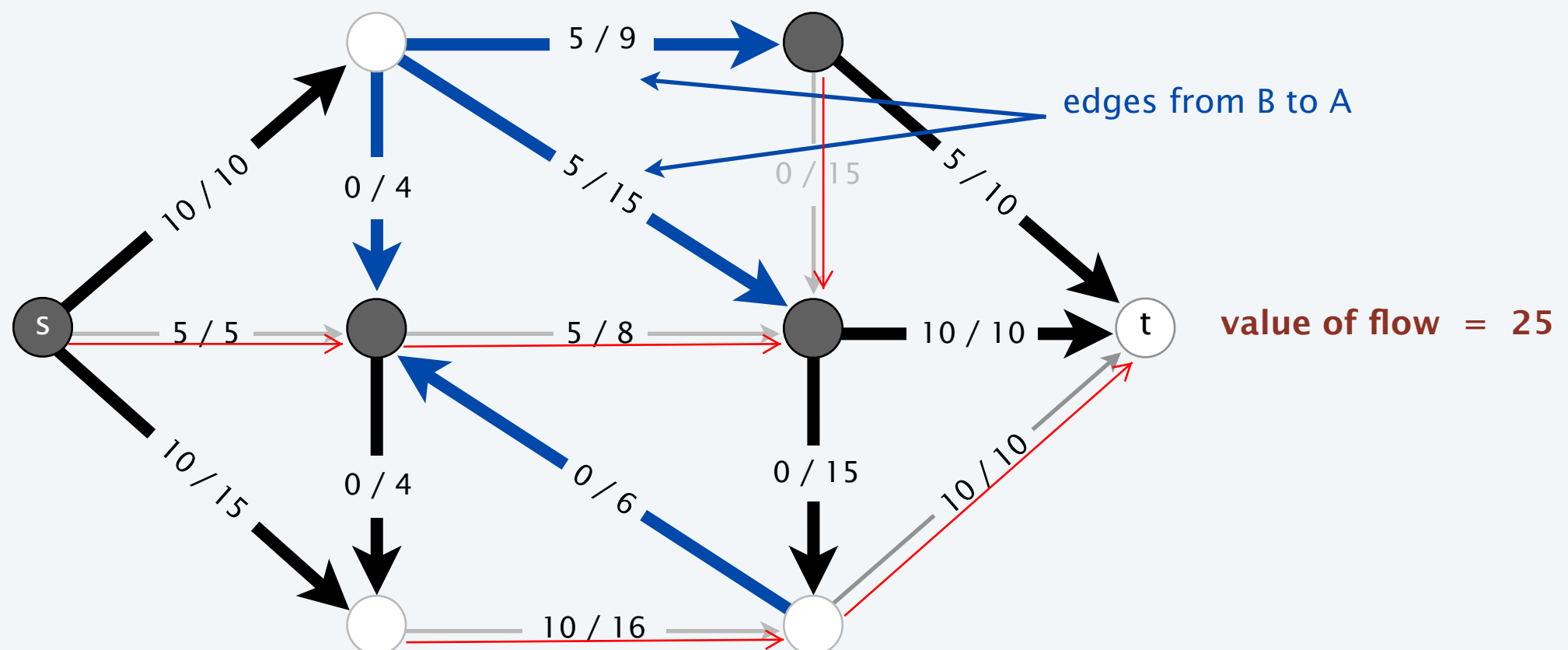


# Relationship between flows and cuts

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the value of the flow  $f$  equals the net flow across the cut  $(A, B)$ .

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

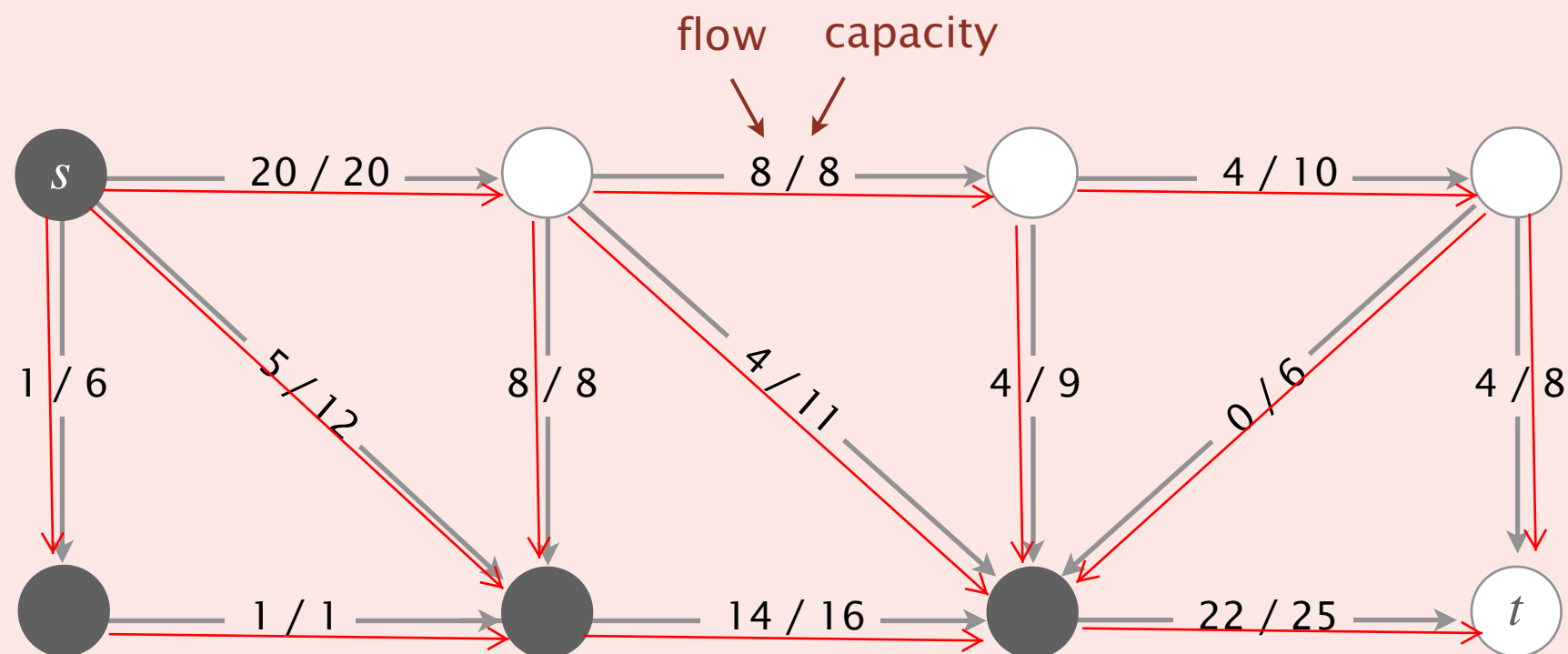
$$\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5 + 0 + 0) = 25$$





Which is the net flow across the given cut?

- A. 11 ( $20 + 25 - 8 - 11 - 9 - 6$ )
- B. 26 ( $20 + 22 - 8 - 4 - 4$ )
- C. 42 ( $20 + 22$ )
- D. 45 ( $20 + 25$ )





# Relationship between flows and cuts

---

**Flow value lemma.** Let  $f$  be any flow and let  $(A, B)$  be any cut. Then, the value of the flow  $f$  equals the net flow across the cut  $(A, B)$ .

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

**Pf.**

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

by flow conservation, all terms  
except for  $v = s$  are 0  $\longrightarrow$

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

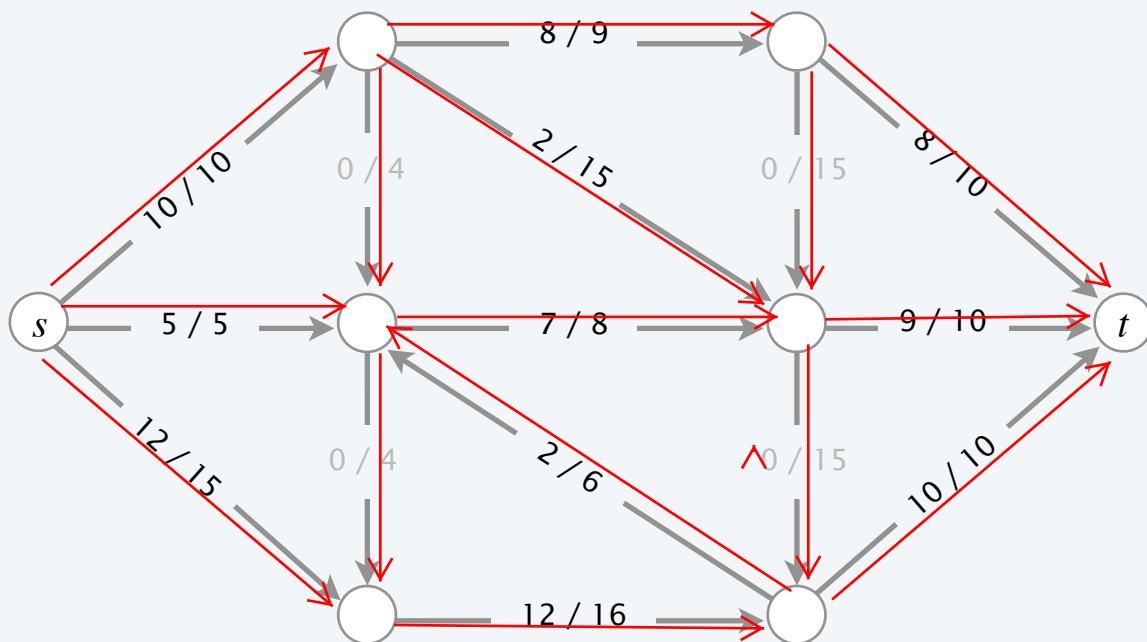
# Relationship between flows and cuts

**Weak duality.** Let  $f$  be any flow and  $(A, B)$  be any cut. Then,  $val(f) \leq cap(A, B)$ .

**Pf.**

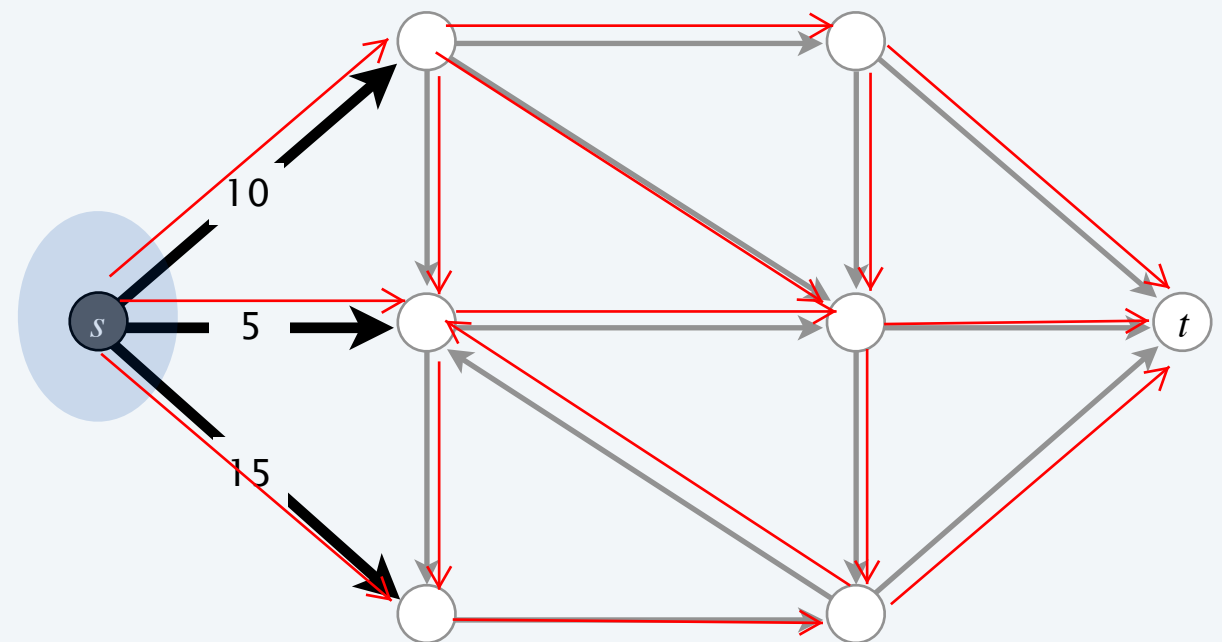
$$\begin{aligned}
 val(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &\leq \sum_{e \text{ out of } A} f(e) \\
 &\leq \sum_{e \text{ out of } A} c(e) \\
 &= cap(A, B) \quad \blacksquare
 \end{aligned}$$

flow value lemma



value of flow = 27

$\leq$



capacity of cut = 30

# Certificate of optimality

**Corollary.** Let  $f$  be a flow and let  $(A, B)$  be any cut.

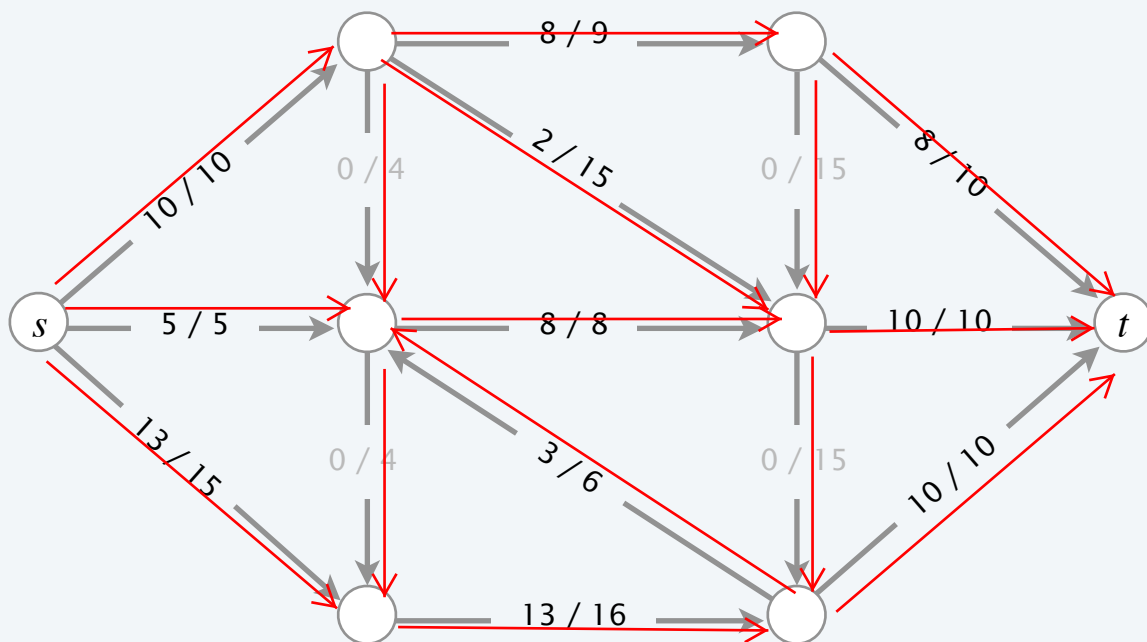
If  $val(f) = cap(A, B)$ , then  $f$  is a max flow and  $(A, B)$  is a min cut.

**Pf.**

- For any flow  $f'$ :  $val(f') \leq cap(A, B) = val(f)$ .
- For any cut  $(A', B')$ :  $cap(A', B') \geq val(f) = cap(A, B)$ . ■

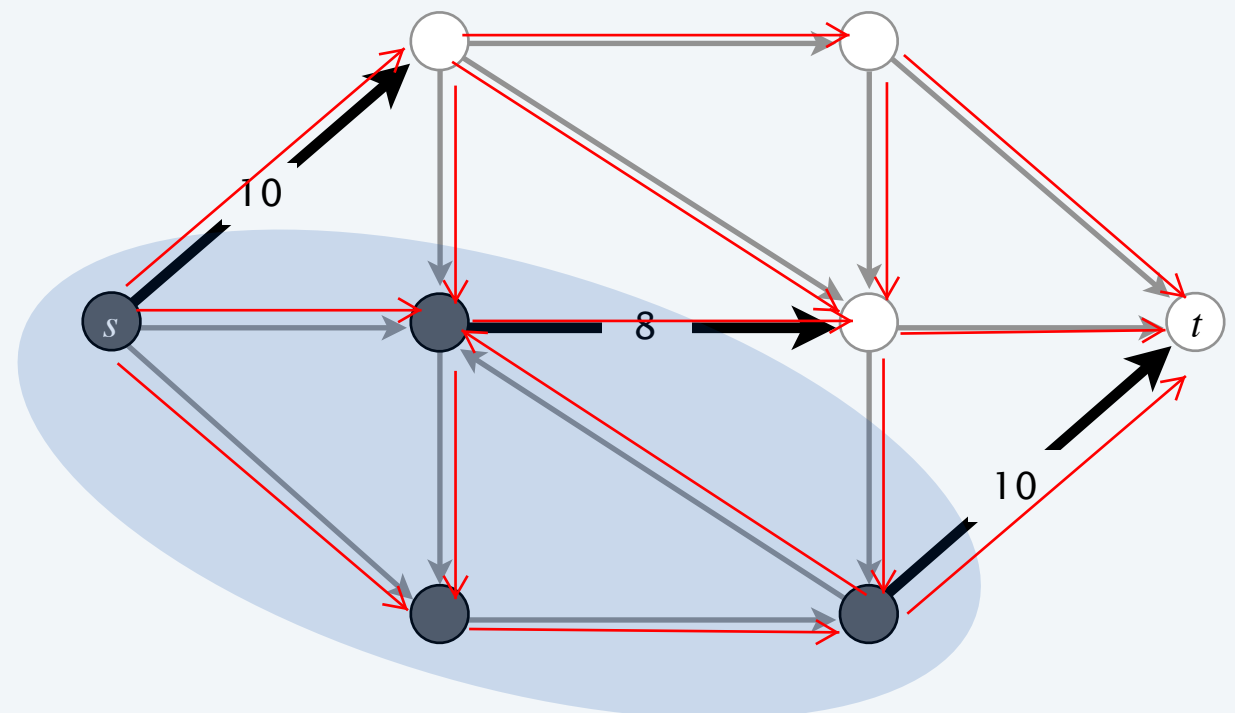
weak duality

weak duality



value of flow = 28

=



capacity of cut = 28

# Max-flow min-cut theorem

Max-flow min-cut theorem. Value of a max flow = capacity of a min cut.

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  $(d, a, b, c)$  is not. When a simple cut set is

# Max-flow min-cut theorem

---

**Max-flow min-cut theorem.** Value of a max flow = capacity of a min cut.

**Augmenting path theorem.** A flow  $f$  is a max flow iff no augmenting paths.

**Pf.** The following three conditions are equivalent for any flow  $f$ :

- i. There exists a cut  $(A, B)$  such that  $cap(A, B) = val(f)$ .
- ii.  $f$  is a max flow.
- iii. There is no augmenting path with respect to  $f$ . ← if Ford–Fulkerson terminates, then  $f$  is max flow

[ i  $\Rightarrow$  ii ]

- This is the weak duality corollary. ■

# Max-flow min-cut theorem

---

**Max-flow min-cut theorem.** Value of a max flow = capacity of a min cut.

**Augmenting path theorem.** A flow  $f$  is a max flow iff no augmenting paths.

**Pf.** The following three conditions are equivalent for any flow  $f$ :

- i. There exists a cut  $(A, B)$  such that  $cap(A, B) = val(f)$ .
- ii.  $f$  is a max flow.
- iii. There is no augmenting path with respect to  $f$ .

[ ii  $\Rightarrow$  iii ] We prove contrapositive:  $\neg$  iii  $\Rightarrow$   $\neg$  ii.

- Suppose that there is an augmenting path with respect to  $f$ .
- Can improve flow  $f$  by sending flow along this path.
- Thus,  $f$  is not a max flow. ■

# Max-flow min-cut theorem

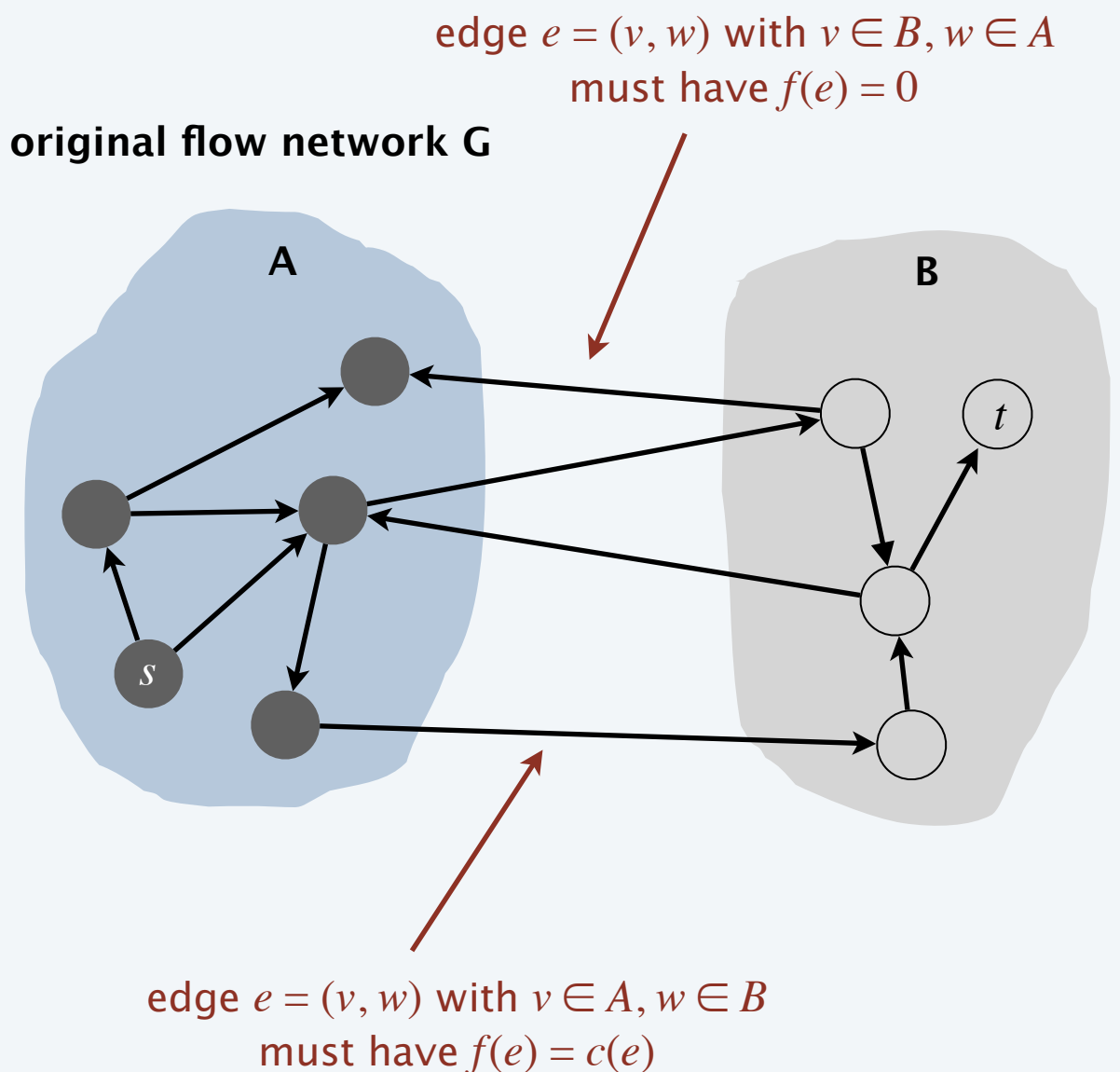
[ iii  $\Rightarrow$  i ]

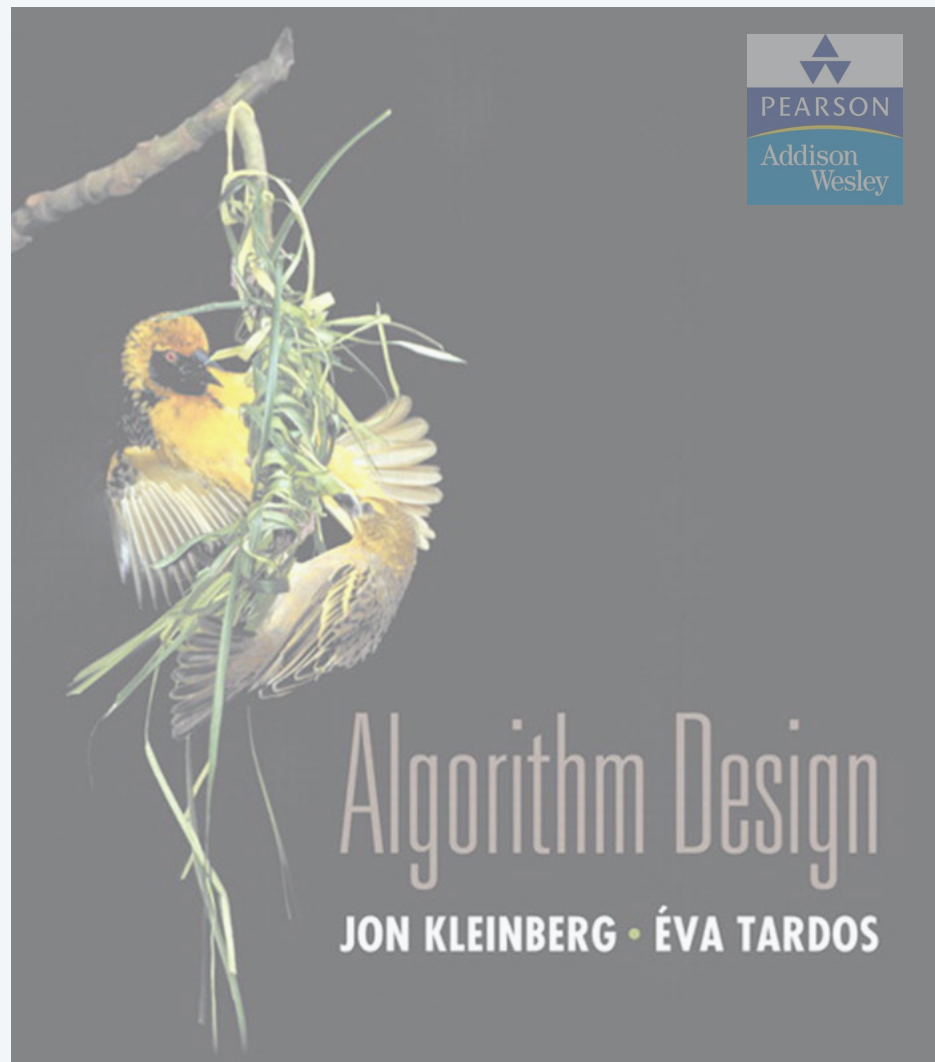
- Let  $f$  be a flow with no augmenting paths.
- Let  $A$  be set of nodes reachable from  $s$  in residual network  $G_f$ .
- By definition of  $A$ :  $s \in A$ .
- By definition of flow  $f$ :  $t \notin A$ .

flow value lemma  $\nearrow$

$$\begin{aligned} \text{val}(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &= \sum_{e \text{ out of } A} c(e) - 0 \\ &= \text{cap}(A, B) \quad \blacksquare \end{aligned}$$

original flow network  $G$





## SECTION 7.3

# 7. NETWORK FLOW I

---

- ▶ *max-flow and min-cut problems*
- ▶ *Ford–Fulkerson algorithm*
- ▶ *max-flow min-cut theorem*
- ▶ ***capacity-scaling algorithm***
- ▶ *shortest augmenting paths*
- ▶ *Dinitz' algorithm*
- ▶ *simple unit-capacity networks*



# Analysis of Ford–Fulkerson algorithm (when capacities are integral)

---

**Assumption.** Every edge capacity  $c(e)$  is an integer between 1 and  $C$ .

**Integrality invariant.** Throughout Ford–Fulkerson, every edge flow  $f(e)$  and residual capacity  $c_f(e)$  is an integer.

**Pf.** By induction on the number of augmenting paths. ■

consider cut  $A = \{ s \}$   
(assumes no parallel edges)

**Theorem.** Ford–Fulkerson terminates after at most  $val(f^*) \leq nC$  augmenting paths, where  $f^*$  is a max flow.

**Pf.** Each augmentation increases the value of the flow by at least 1. ■

**Corollary.** The running time of Ford–Fulkerson is  $O(mnC)$ .

**Pf.** Can use either BFS or DFS to find an augmenting path in  $O(m)$  time. ■

$f(e)$  is an integer for every  $e$

**Integrality theorem.** There exists an integral max flow  $f^*$ .

**Pf.** Since Ford–Fulkerson terminates, theorem follows from integrality invariant (and augmenting path theorem). ■

# Ford–Fulkerson: exponential example

Q. Is generic Ford–Fulkerson algorithm poly-time in input size?

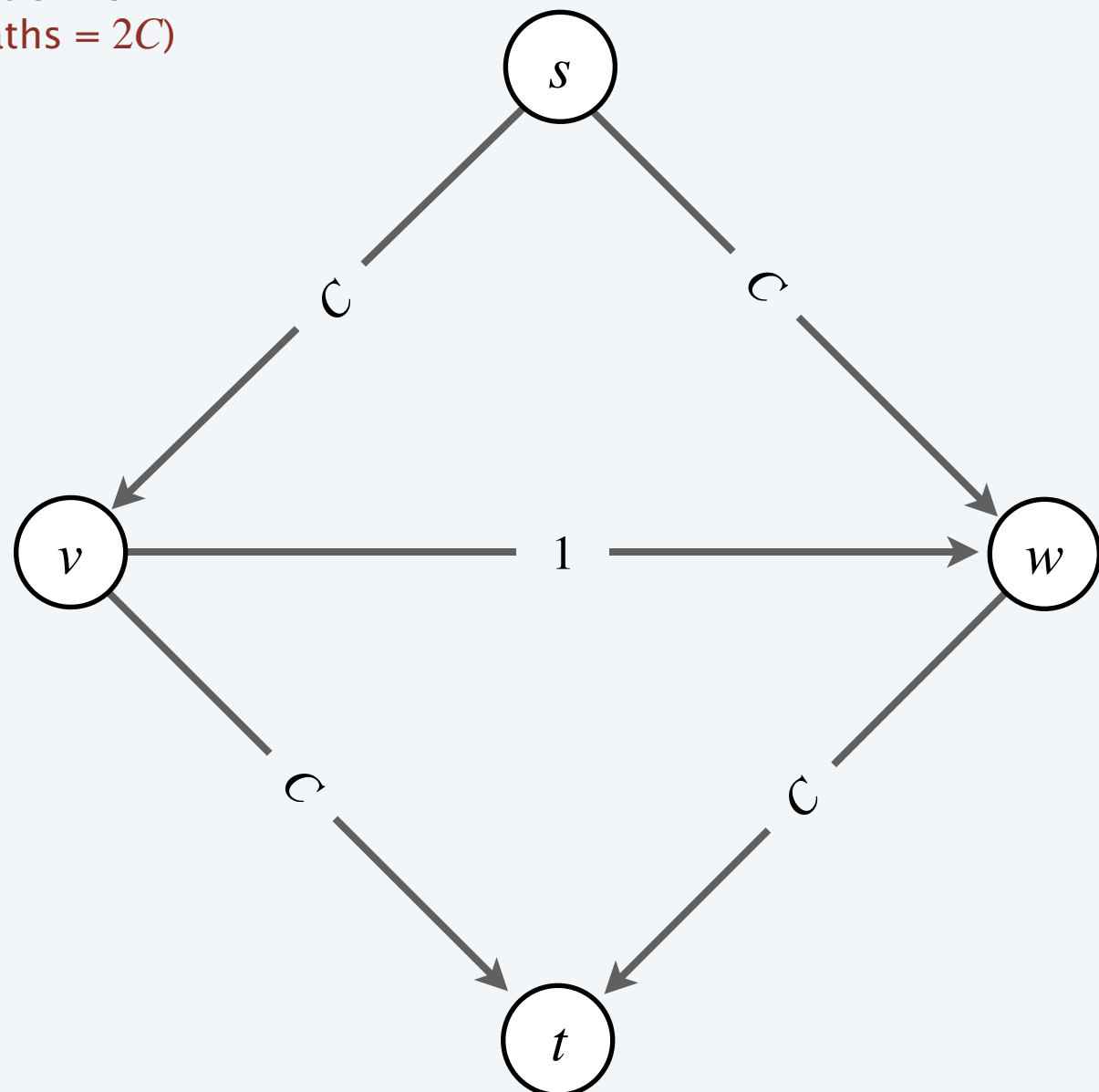
$m, n$ , and  $\log C$



A. No. If max capacity is  $C$ , then algorithm can take  $\geq C$  iterations.

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

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





The Ford–Fulkerson algorithm is guaranteed to terminate if the edge capacities are ...

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

# Choosing good augmenting paths

---

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.

**Pathology.** When edge capacities can be irrational, no guarantee that Ford–Fulkerson terminates (or converges to a maximum flow)!



**Goal.** Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

# Choosing good augmenting paths

## Choose augmenting paths with:

- Max bottleneck capacity (“fattest”). ← how to find?
- Sufficiently large bottleneck capacity. ← next
- Fewest edges. ← ahead

### 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)**

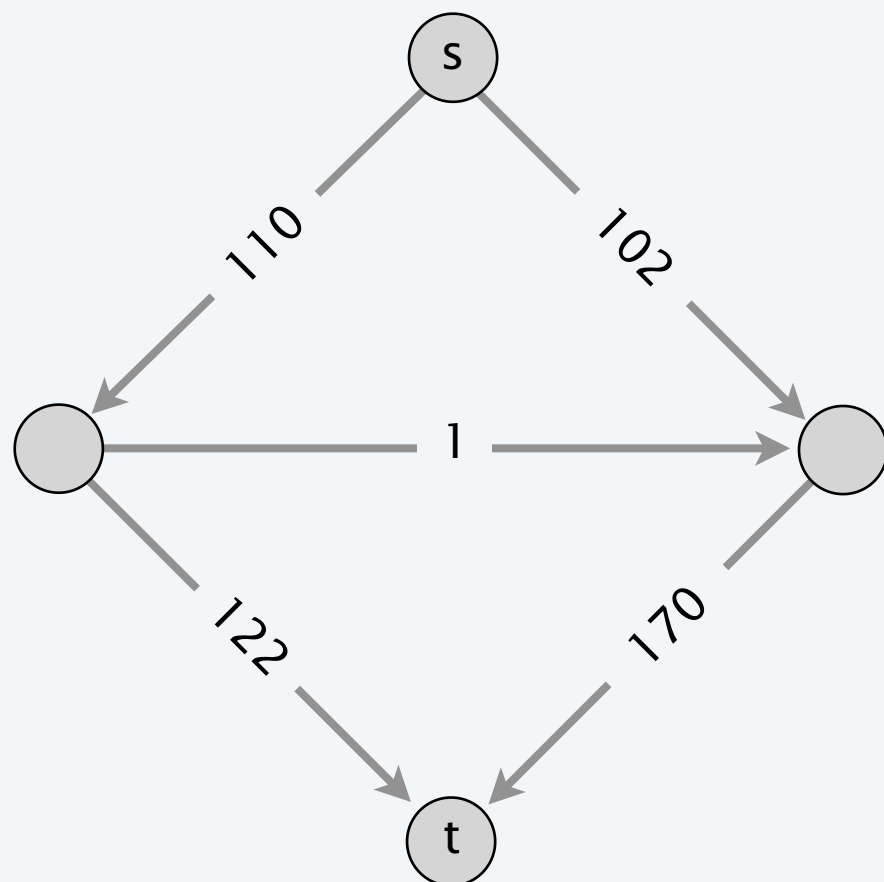
invented in response to a class  
exercises by Adel'son-Vel'skiĭ

# Capacity-scaling algorithm

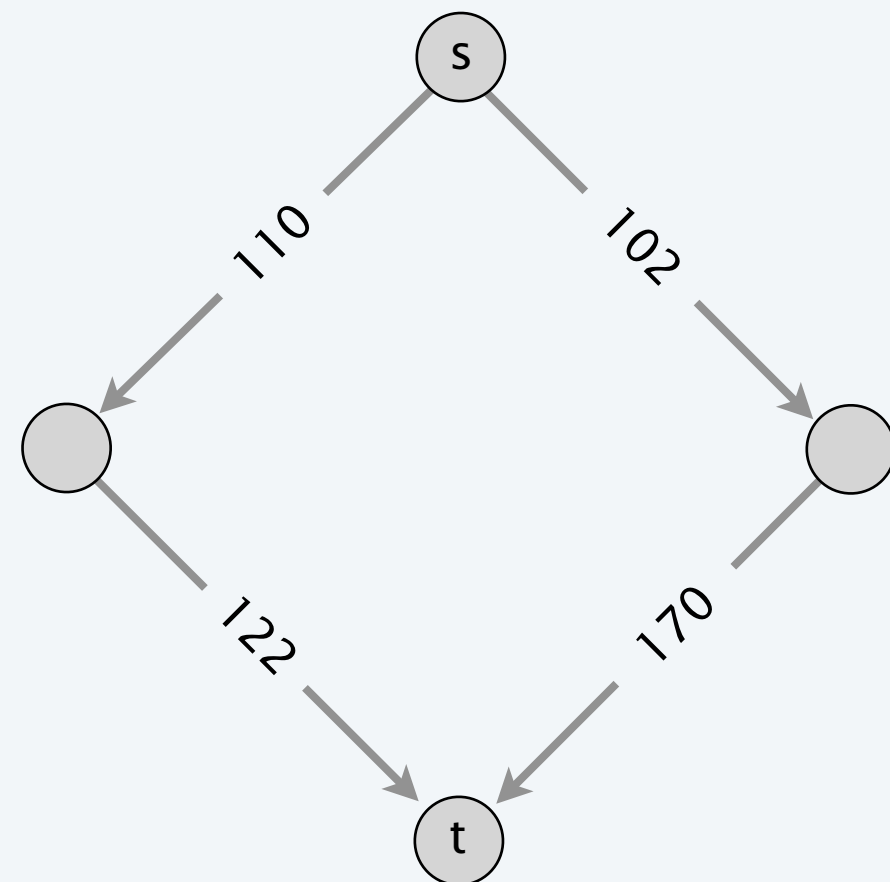
**Overview.** Choosing augmenting paths with “large” bottleneck capacity.

- Maintain scaling parameter  $\Delta$ .
- Let  $G_f(\Delta)$  be the part of the residual network containing only those edges with capacity  $\geq \Delta$ .
- Any augmenting path in  $G_f(\Delta)$  has bottleneck capacity  $\geq \Delta$ .

though not necessarily largest



$G_f$



$G_f(\Delta), \Delta = 100$

# Capacity-scaling algorithm

---

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  $s \rightarrow t$  path  $P$  in  $G_f(\Delta)$ )

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

Update  $G_f(\Delta)$ .

$\Delta \leftarrow \Delta / 2$ .

$\Delta$ -scaling phase

RETURN  $f$ .

---

# Capacity-scaling algorithm: proof of correctness

---

**Assumption.** All edge capacities are integers between 1 and  $C$ .

**Invariant.** The scaling parameter  $\Delta$  is a power of 2.

**Pf.** Initially a power of 2; each phase divides  $\Delta$  by exactly 2. ■

**Integrality invariant.** Throughout the algorithm, every edge flow  $f(e)$  and residual capacity  $c_f(e)$  is an integer.

**Pf.** Same as for generic Ford–Fulkerson. ■

**Theorem.** If capacity-scaling algorithm terminates, then  $f$  is a max flow.

**Pf.**

- By integrality invariant, when  $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$ .
- Upon termination of  $\Delta = 1$  phase, there are no augmenting paths.
- Result follows augmenting path theorem ■



# Capacity-scaling algorithm: analysis of running time

---

**Lemma 1.** There are  $1 + \lfloor \log_2 C \rfloor$  scaling phases.

**Pf.** Initially  $C/2 < \Delta \leq C$ ;  $\Delta$  decreases by a factor of 2 in each iteration. ■


**Lemma 2.** Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase.

Then, the max-flow value  $\leq \text{val}(f) + m \Delta$ .

**Pf.** Next slide.

**Lemma 3.** There are  $\leq 2m$  augmentations per scaling phase.

**Pf.**

- Let  $f$  be the flow at the beginning of a  $\Delta$ -scaling phase. 
- Lemma 2  $\Rightarrow$  max-flow value  $\leq \text{val}(f) + m (2 \Delta)$ .
- Each augmentation in a  $\Delta$ -phase increases  $\text{val}(f)$  by at least  $\Delta$ . ■

or equivalently,  
at the end  
of a  $2\Delta$ -scaling phase

**Theorem.** The capacity-scaling algorithm takes  $O(m^2 \log C)$  time.

**Pf.**

- Lemma 1 + Lemma 3  $\Rightarrow O(m \log C)$  augmentations.
- Finding an augmenting path takes  $O(m)$  time. ■

# Capacity-scaling algorithm: analysis of running time

**Lemma 2.** Let  $f$  be the flow at the end of a  $\Delta$ -scaling phase.

Then, the max-flow value  $\leq \text{val}(f) + m \Delta$ .

**Pf.**

- We show there exists a cut  $(A, B)$  such that  $\text{cap}(A, B) \leq \text{val}(f) + m \Delta$ .
- Choose  $A$  to be the set of nodes reachable from  $s$  in  $G_f(\Delta)$ .
- By definition of  $A$ :  $s \in A$ .
- By definition of flow  $f$ :  $t \notin A$ .

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

