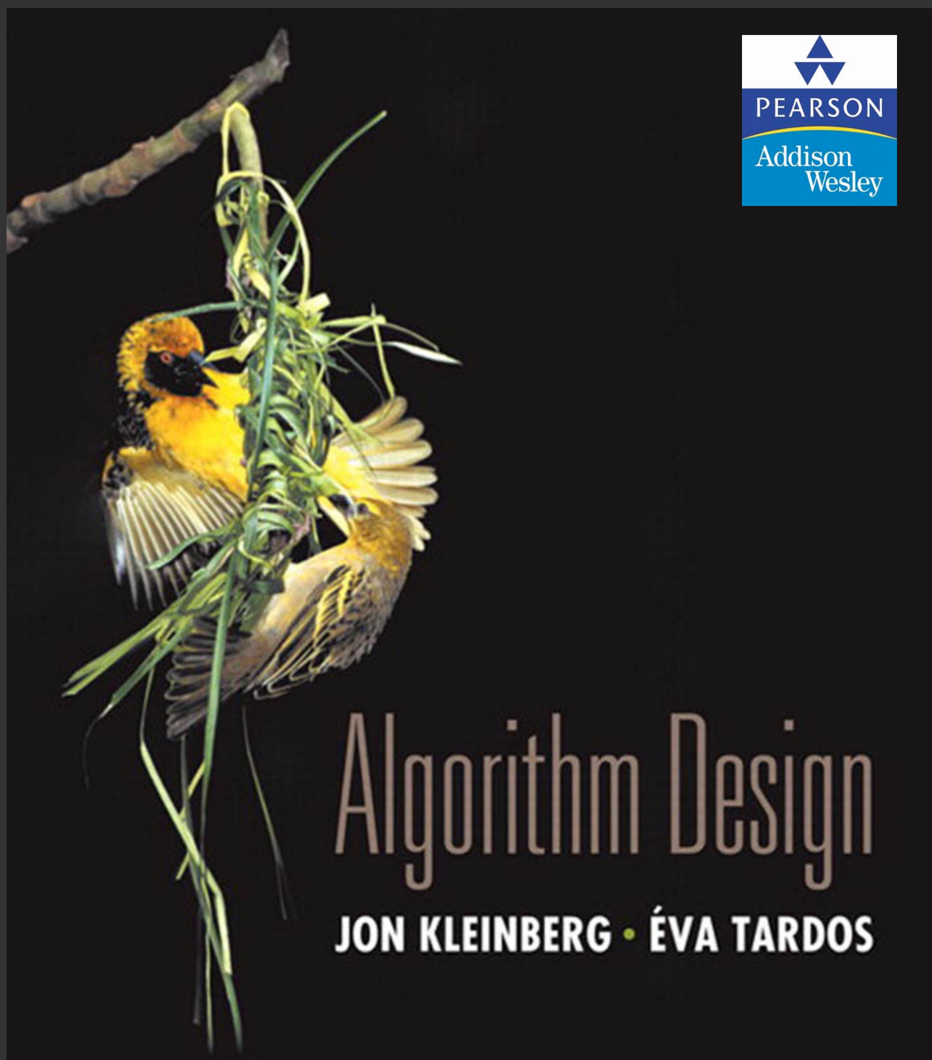


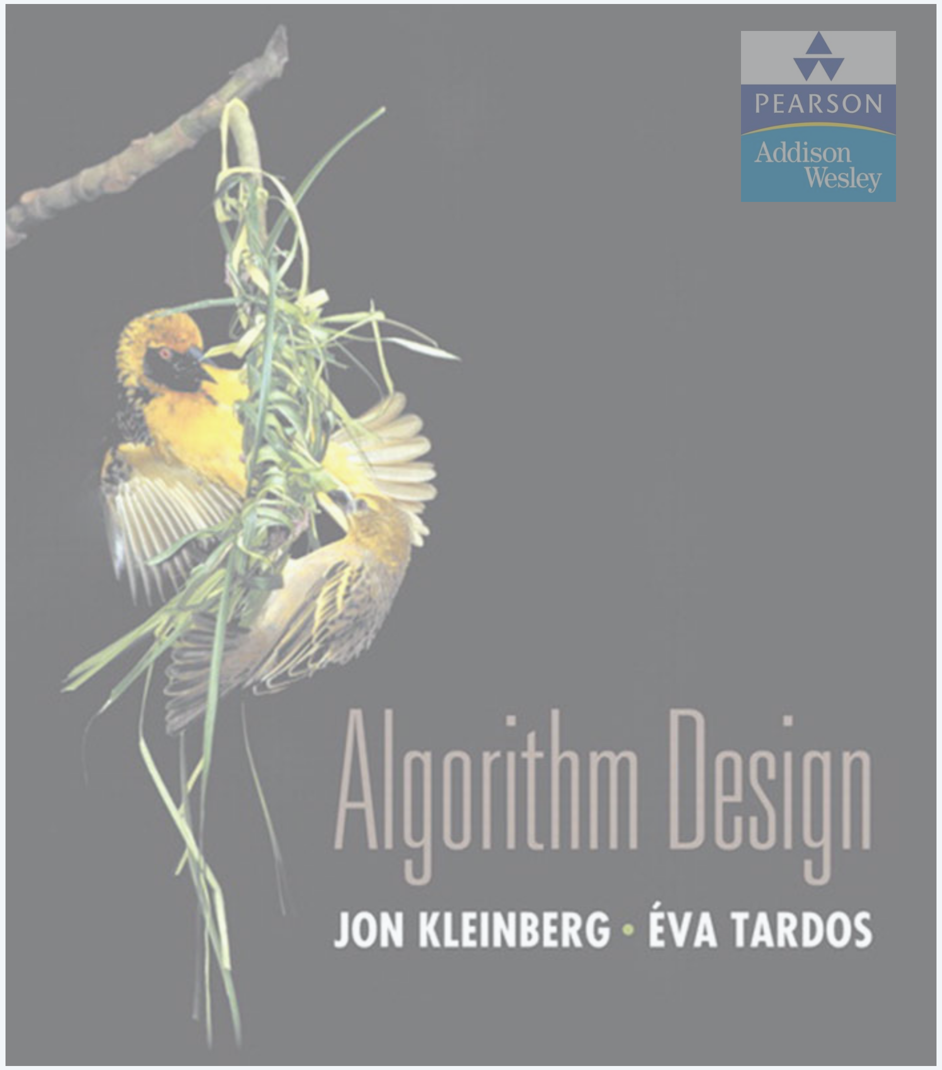
Network Flow

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*

Lecture slides by Kevin Wayne

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

Last updated on 23/12/2019 23:43



SECTION 7.1

NETWORK FLOW

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*

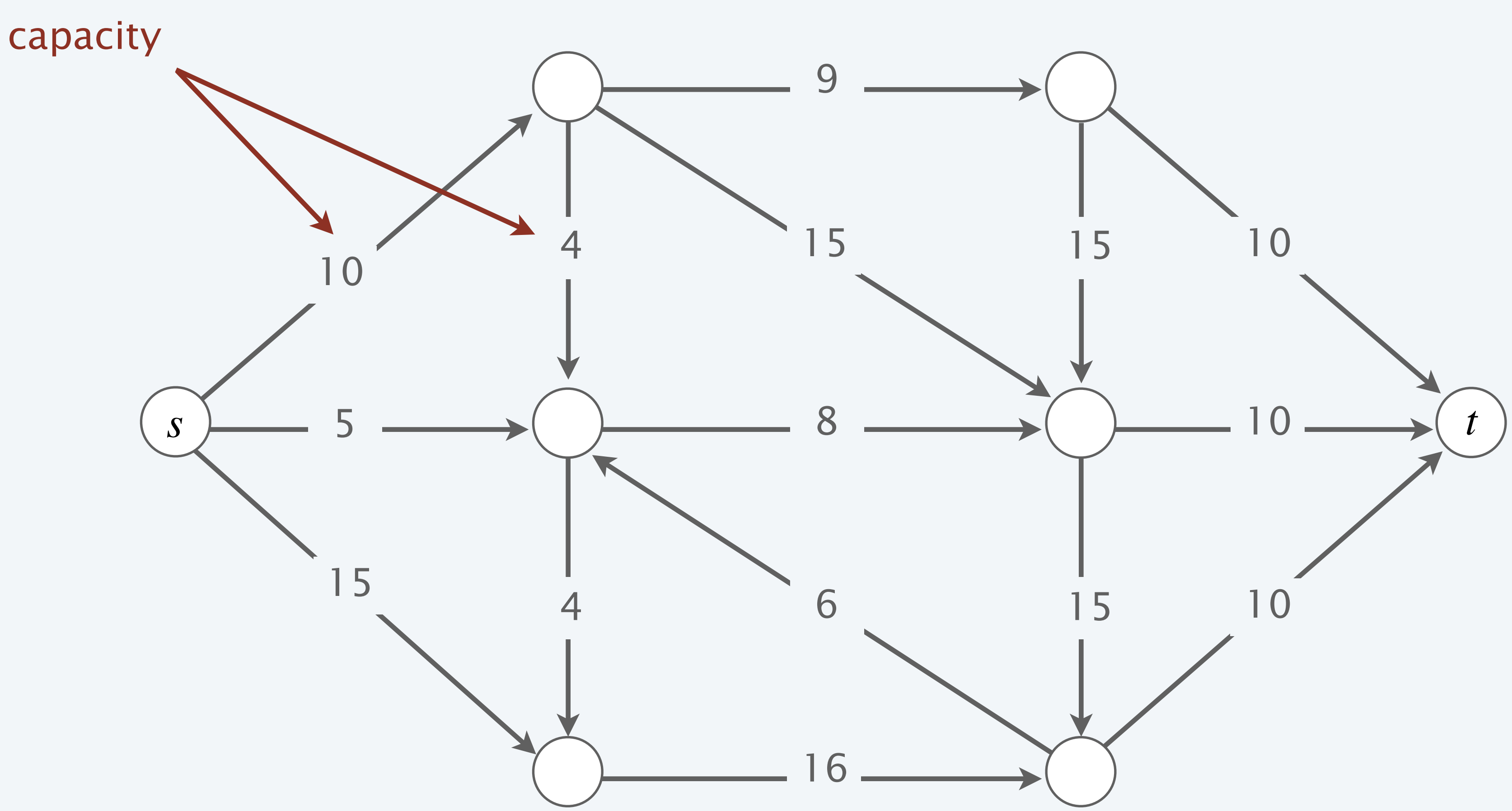
Flow network

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.

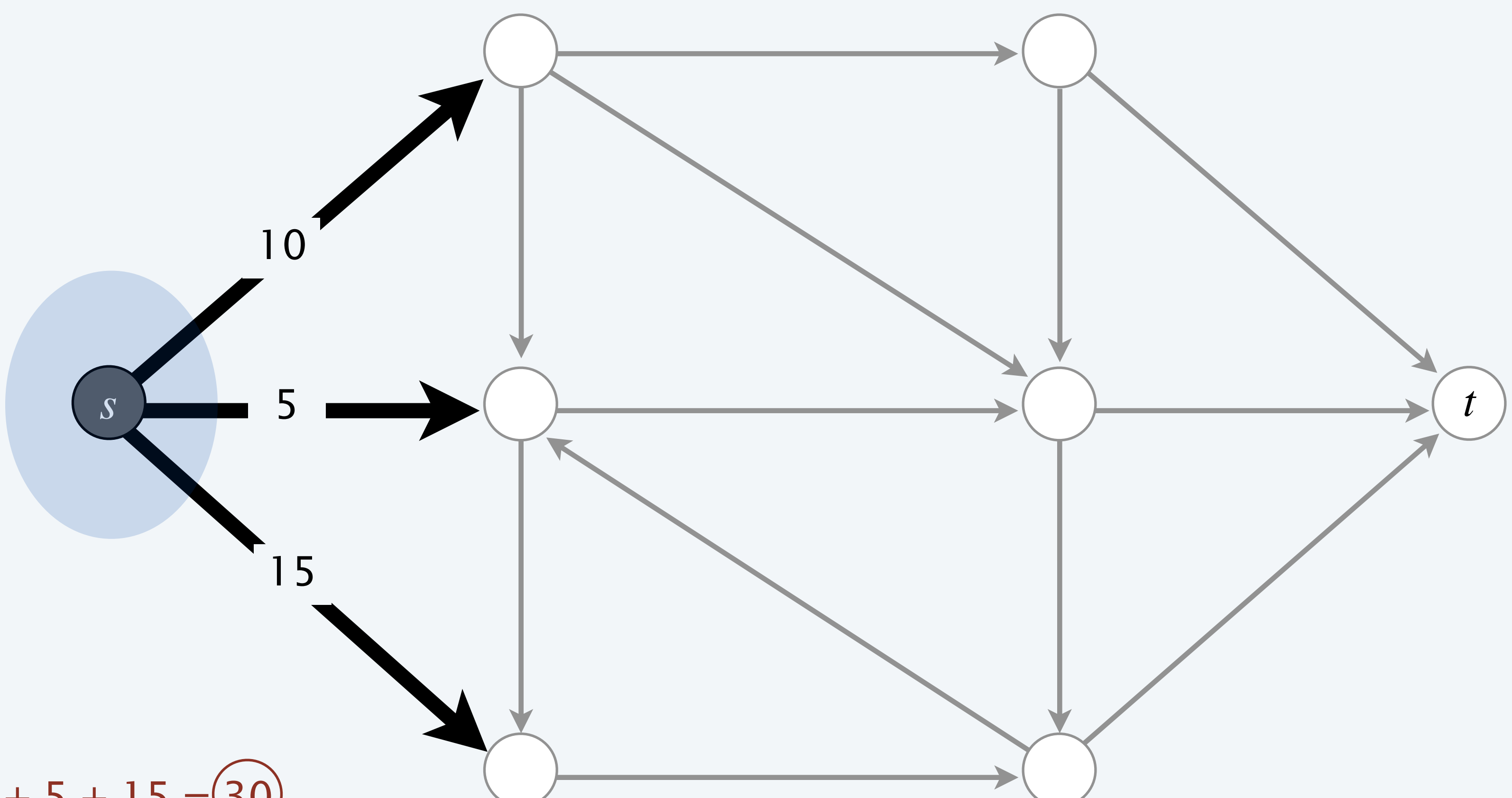


Minimum-cut problem

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.

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



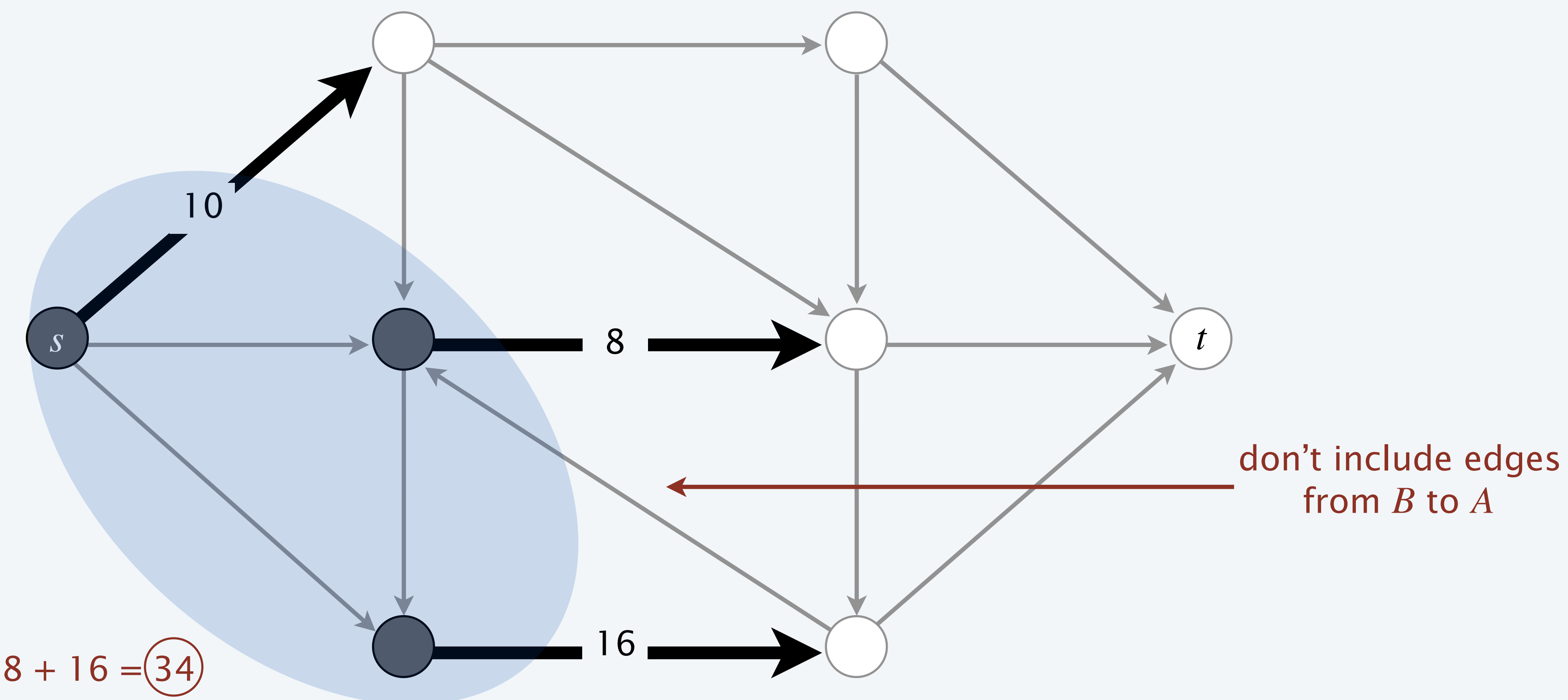
capacity = 10 + 5 + 15 = 30

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.

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



capacity = 10 + 8 + 16 = 34

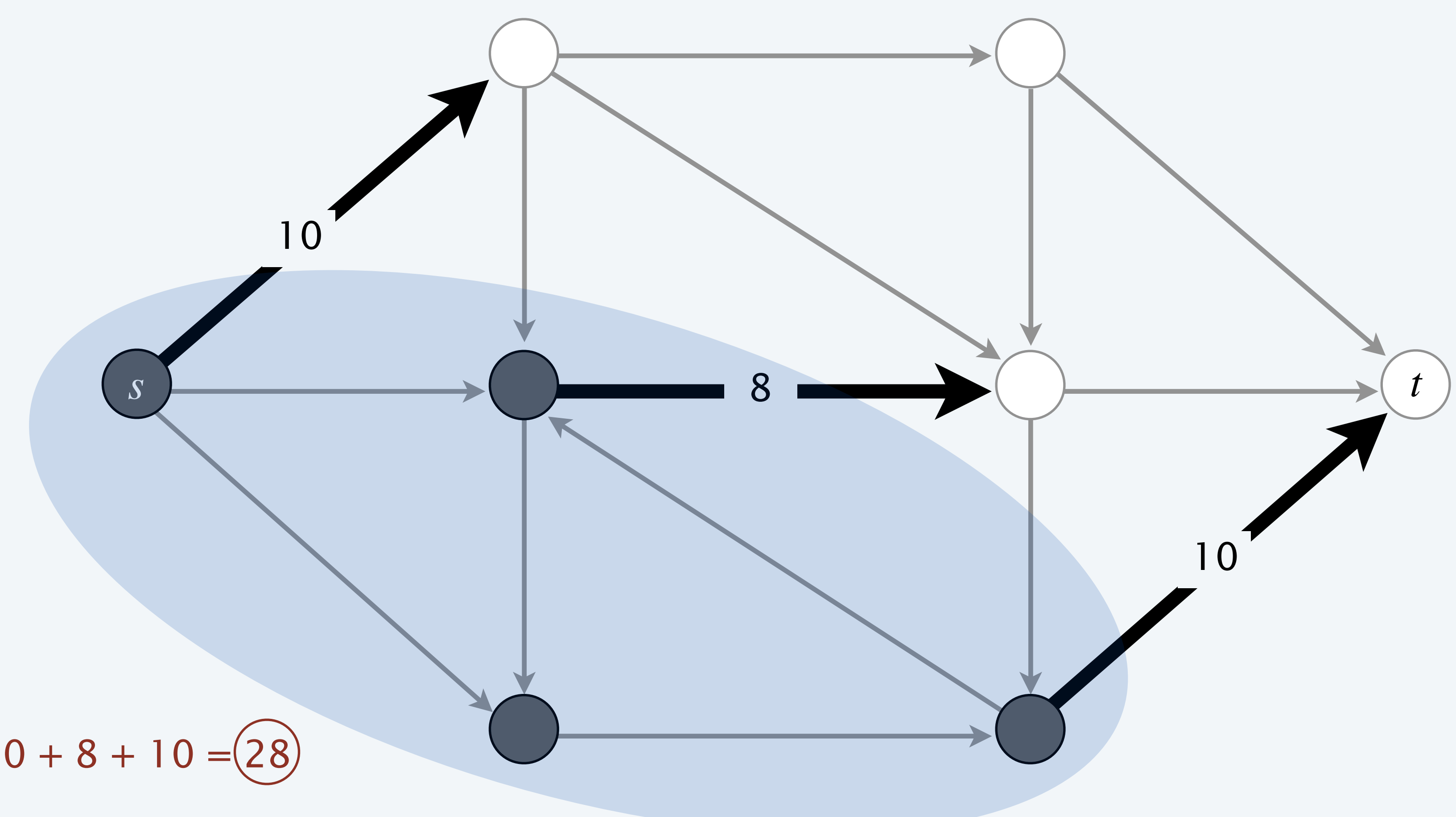
Minimum-cut problem

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.

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

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



capacity = 10 + 8 + 10 = 28

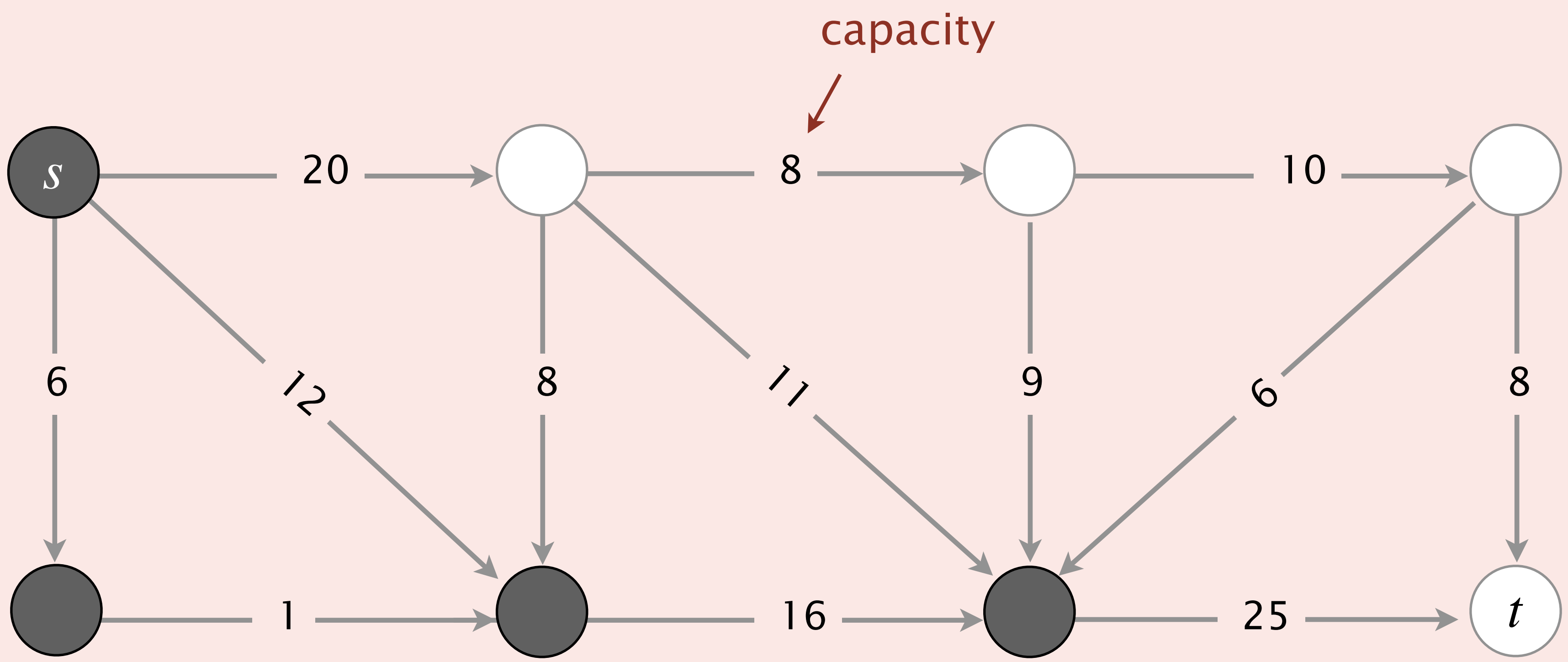
Which is the capacity of the given st-cut?

A. 11 (20 + 25 − 8 − 11 − 9 − 6)

B. 34 (8 + 11 + 9 + 6)

C. 45 (20 + 25)

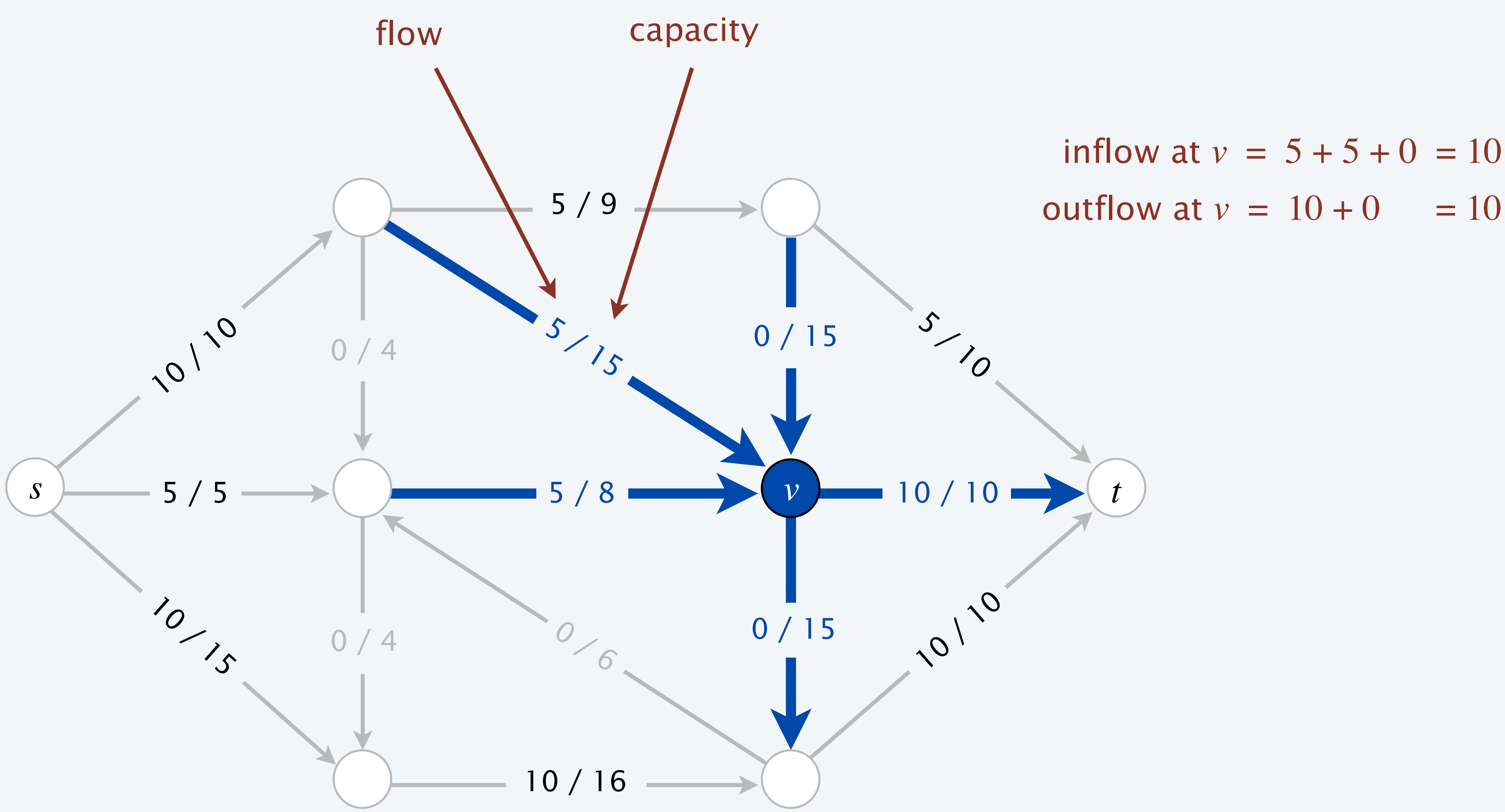
D. 79 (20 + 25 + 8 + 11 + 9 + 6)



Maximum-flow problem

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

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

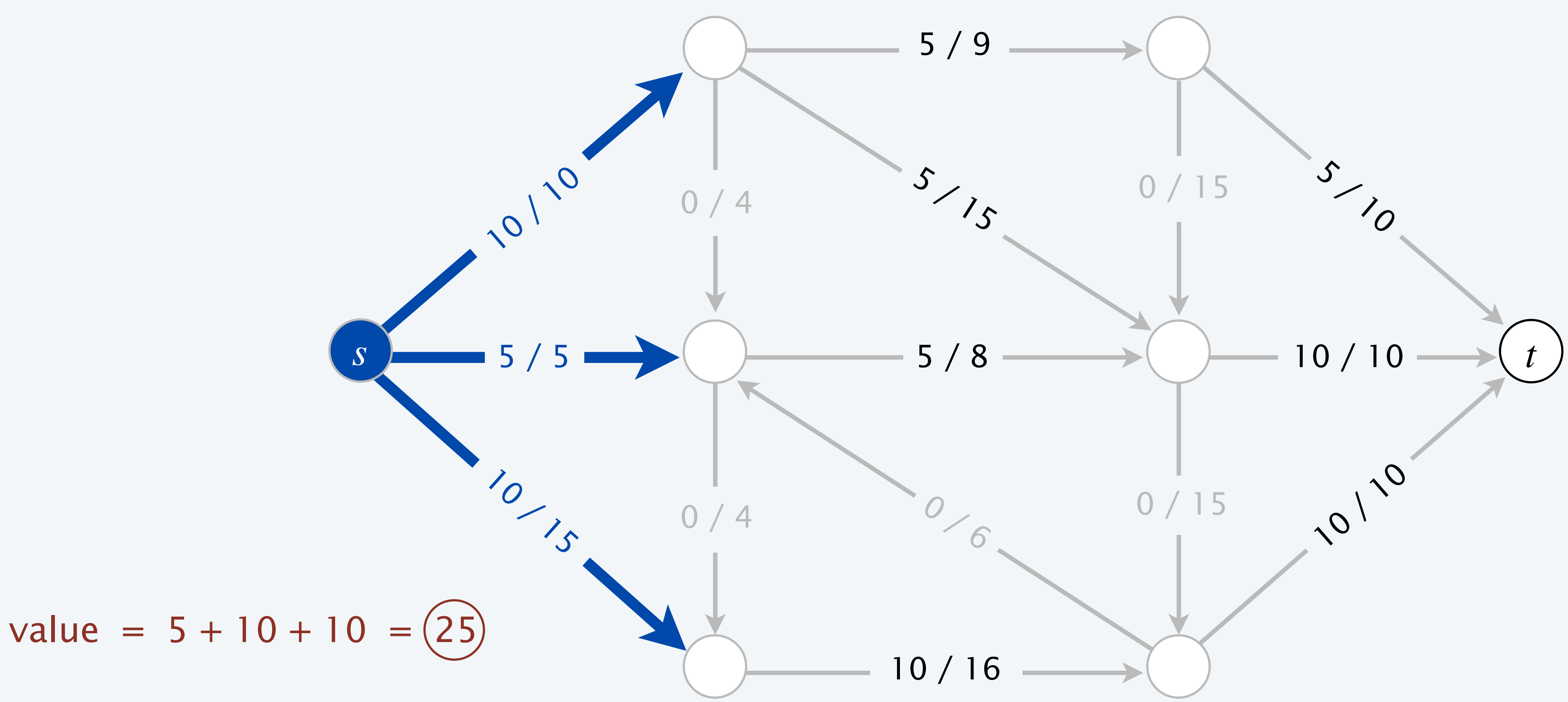


Maximum-flow problem

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



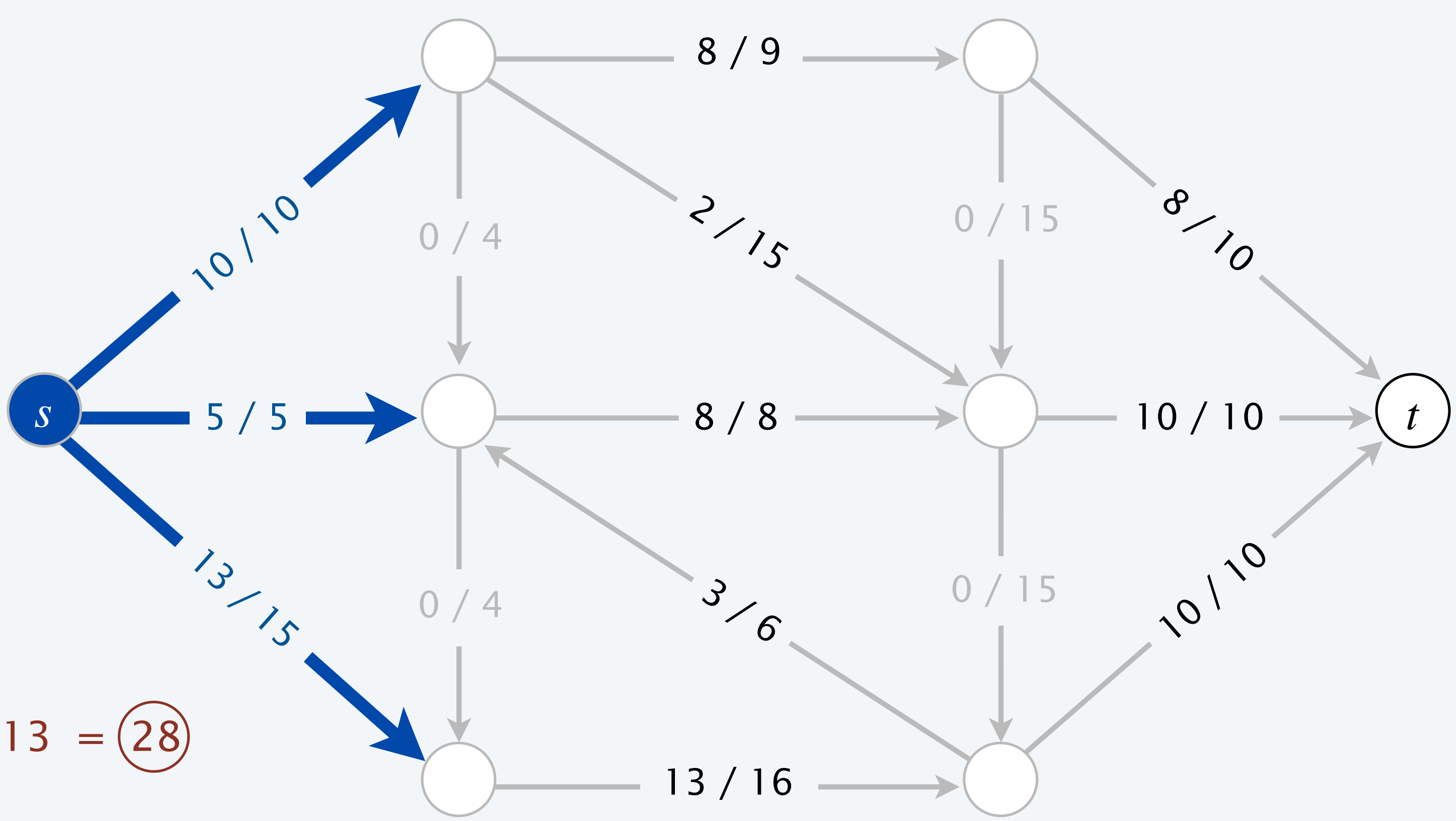
Maximum-flow problem

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

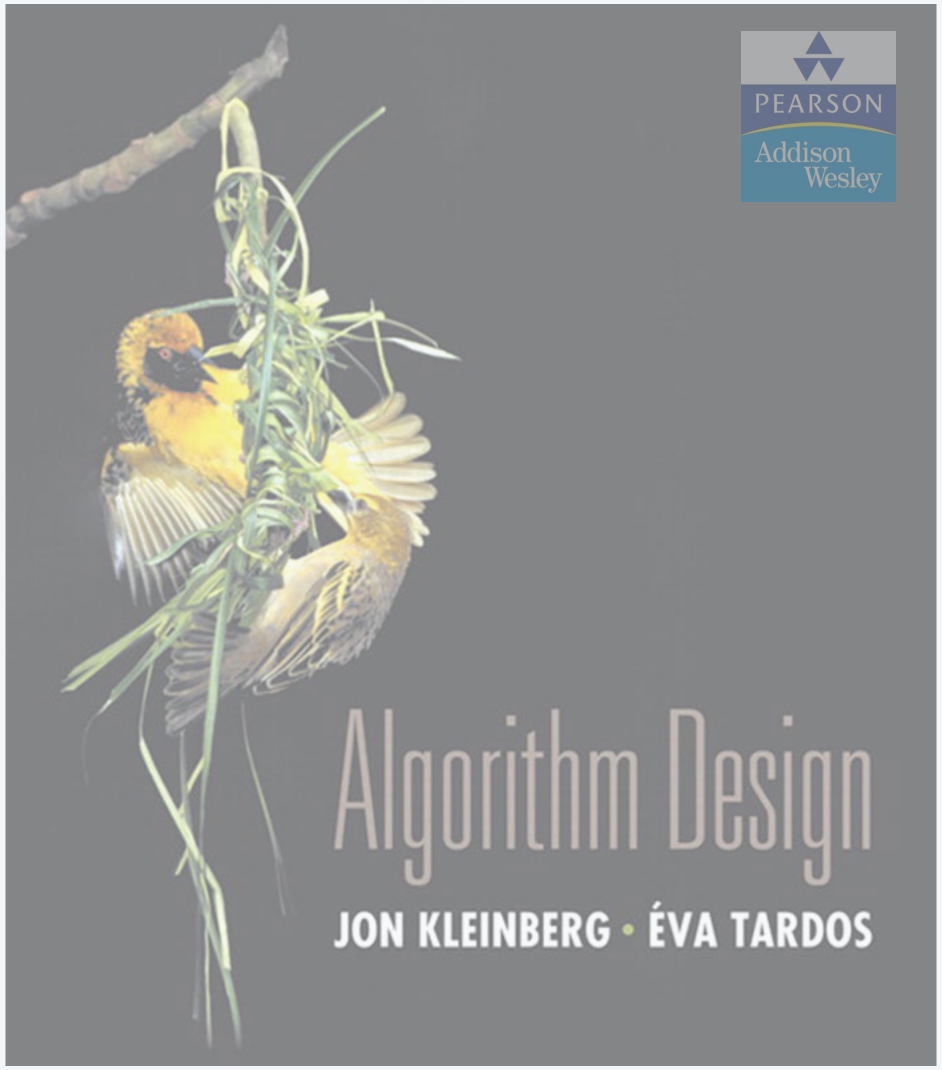
- For each $e \in E$: $\quad 0 \leq f(e) \leq c(e) \quad$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \displaystyle\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \quad$ [flow conservation]

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

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



value = 10 + 5 + 13 = 28



SECTION 7.1

NETWORK FLOW

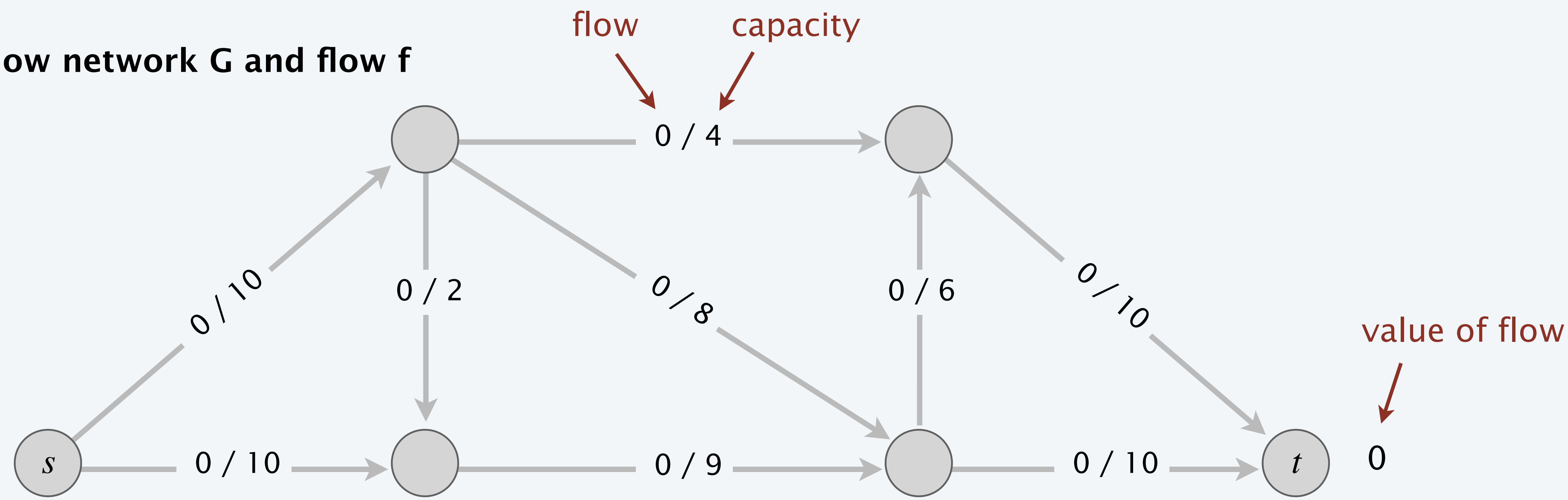
- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*

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

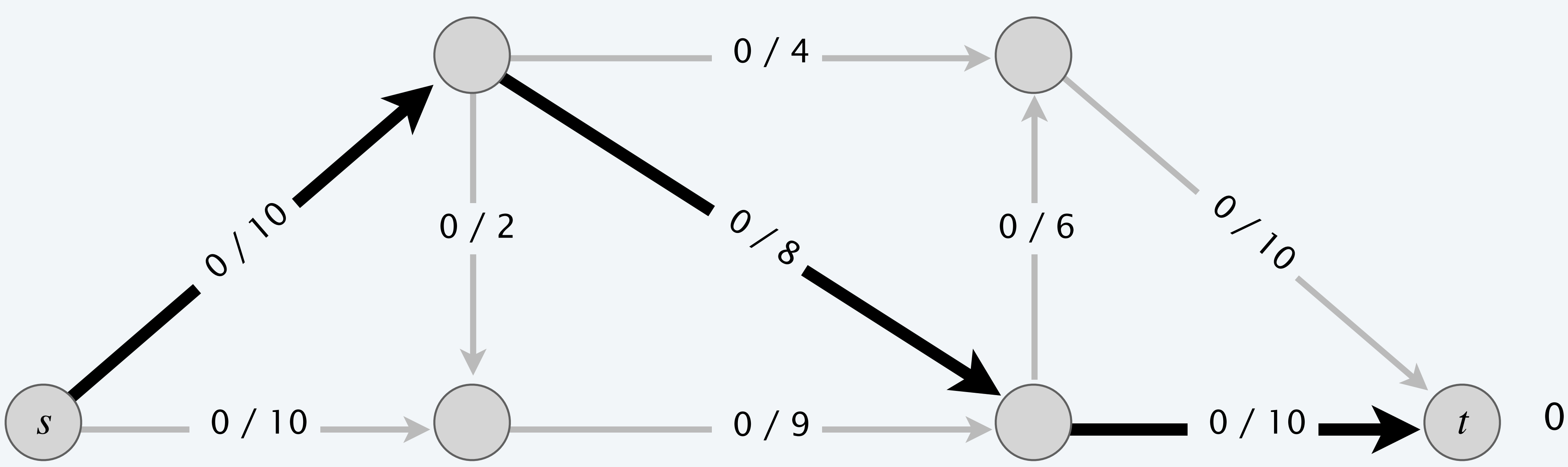


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ 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

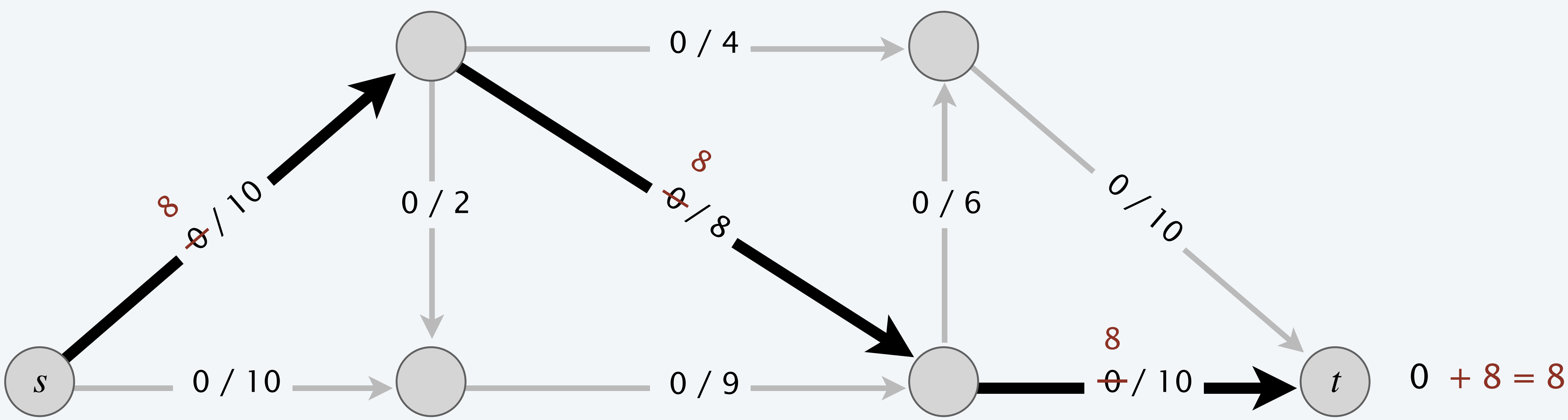


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ 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

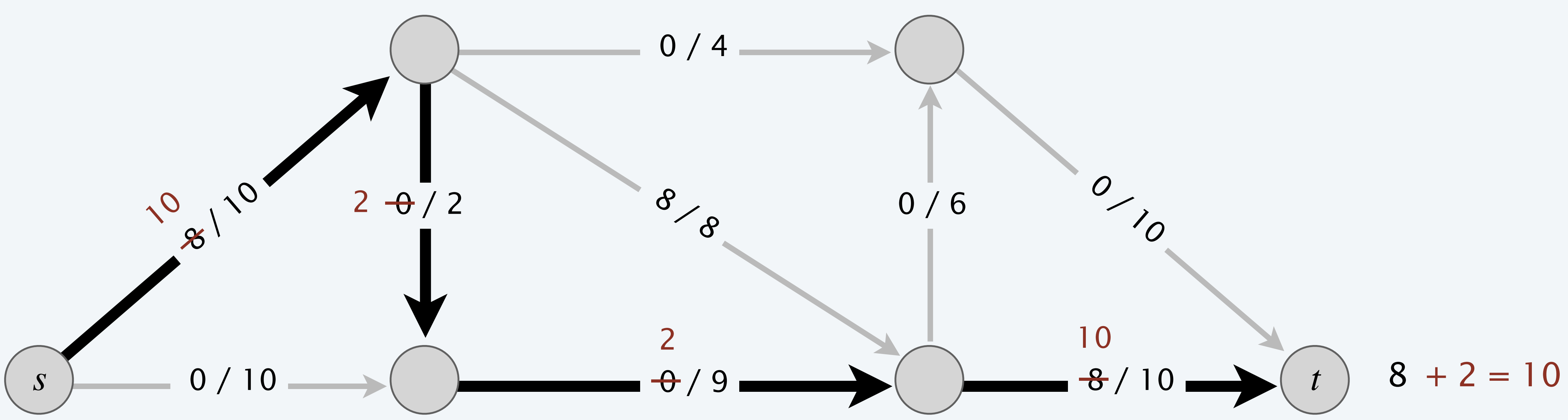


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ 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

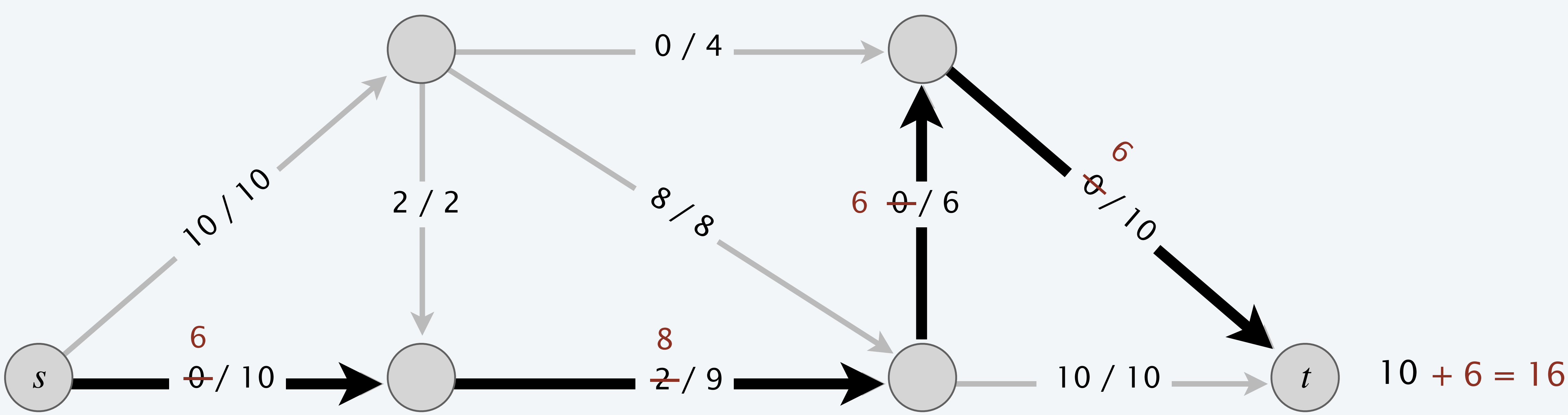


Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ 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



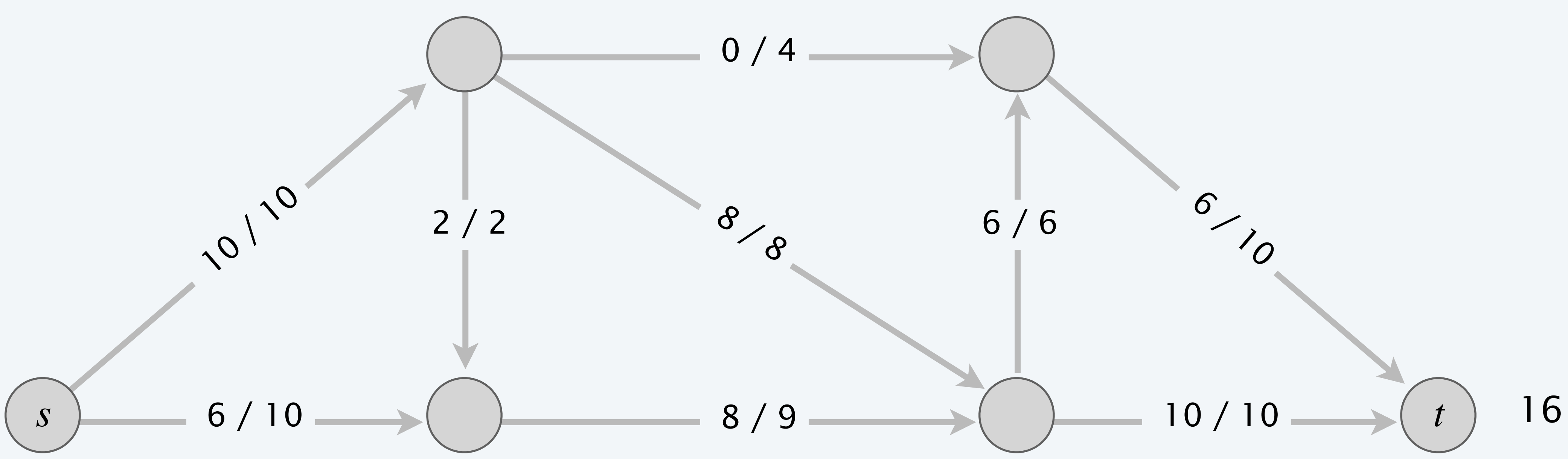
Toward a max-flow algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ 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



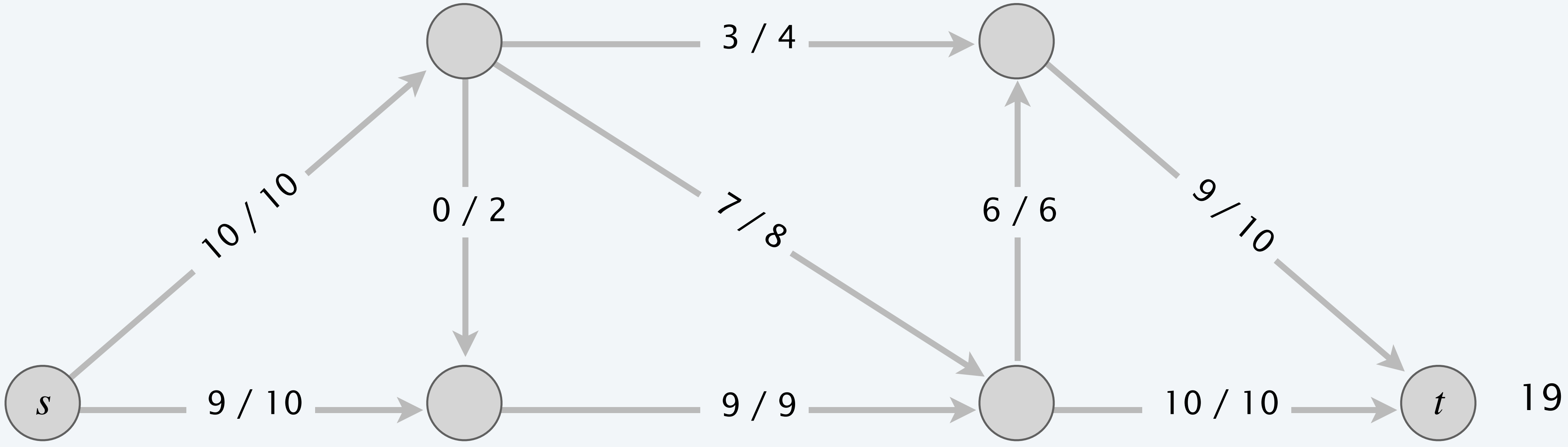
Toward a max-flow algorithm

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

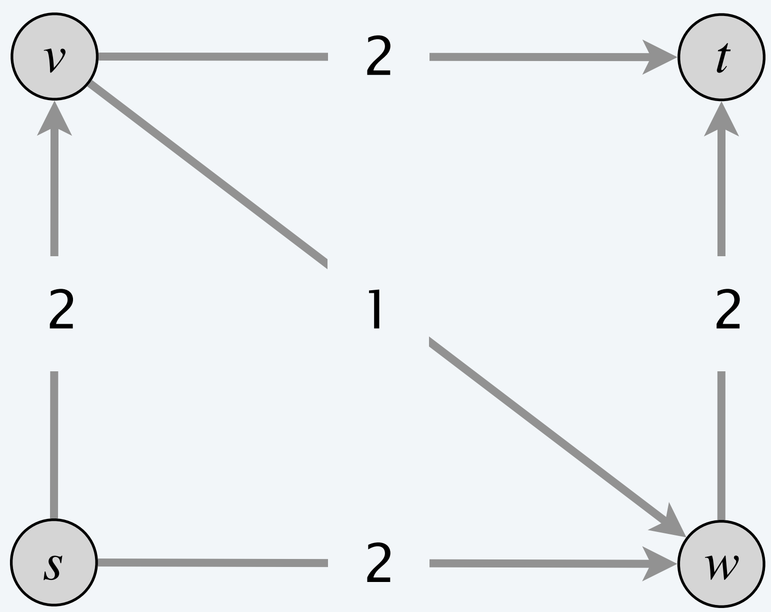
Q. Why does the greedy algorithm fail?

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



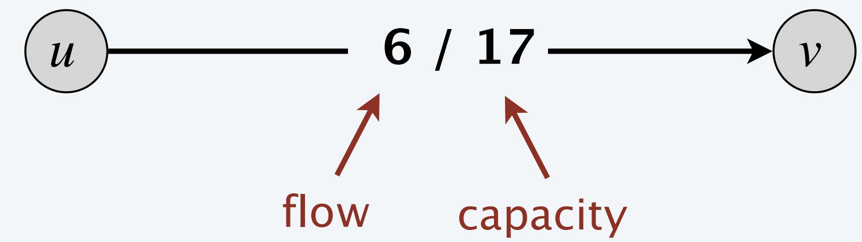
Bottom line. Need some mechanism to "undo" a bad decision.

Residual network

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

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

original flow network G



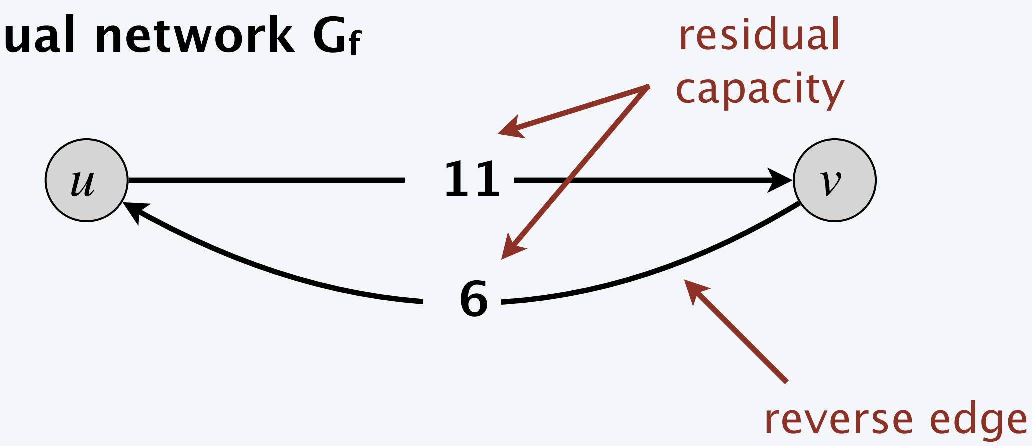
Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "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 Gf



edges with positive residual capacity

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

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.

Augmenting path

Def. An augmenting path is a simple $s \rightsquigarrow t$ 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 $f' \leftarrow$ AUGMENT(f, c, P), the resulting f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

```
AUGMENT(f, c, P)
____________________________________________
δ ← bottleneck capacity of augmenting path P.
FOREACH edge e ∈ P :
   IF (e ∈ E) f(e) ← f(e) + δ.
   ELSE       f(e^reverse) ← f(e^reverse) – δ.
RETURN f.
____________________________________________
```

Network flow: quiz 2

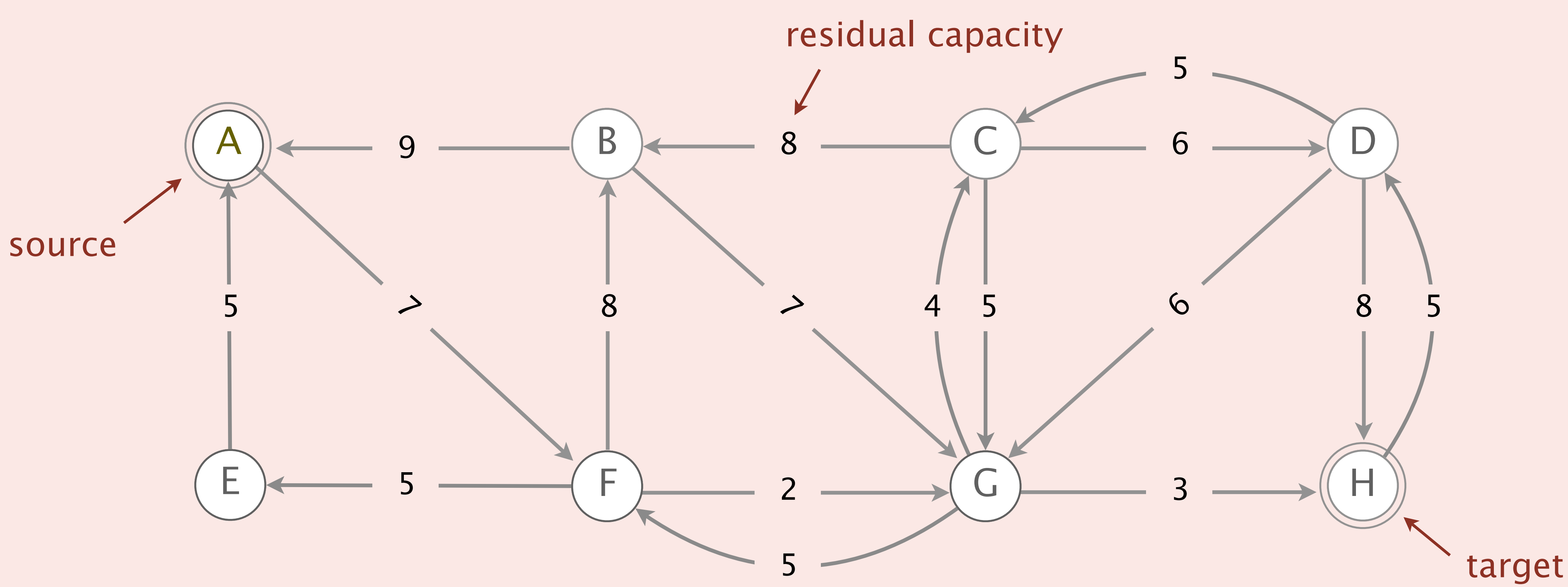
Which is the augmenting path of highest bottleneck capacity?

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⇝t path P in G_f)

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

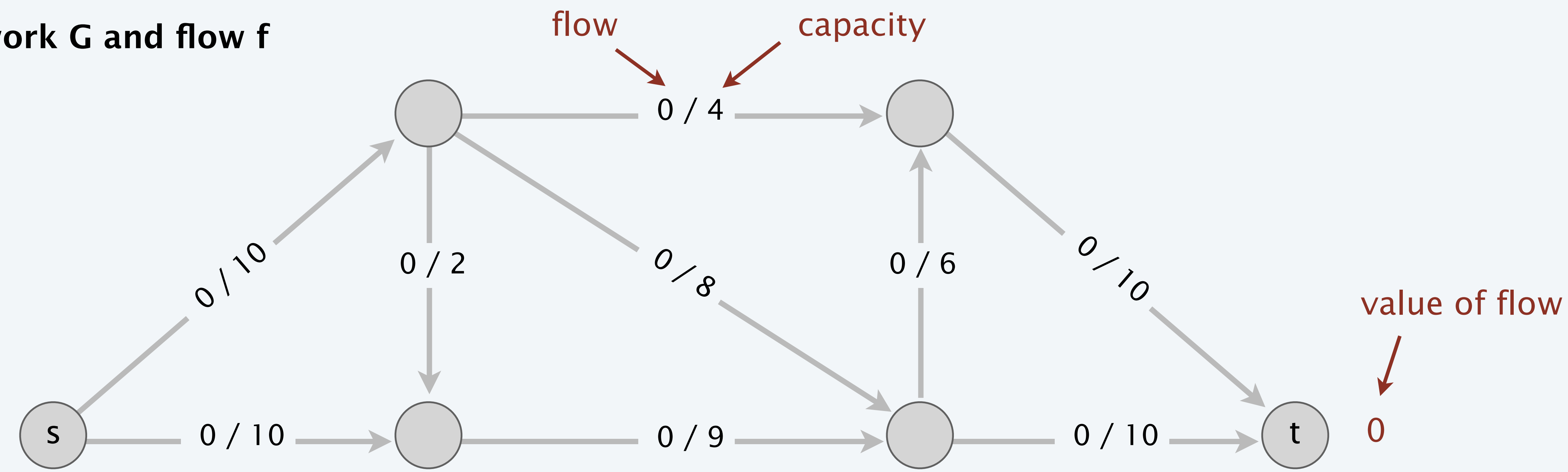
  Update G_f.

RETURN f.

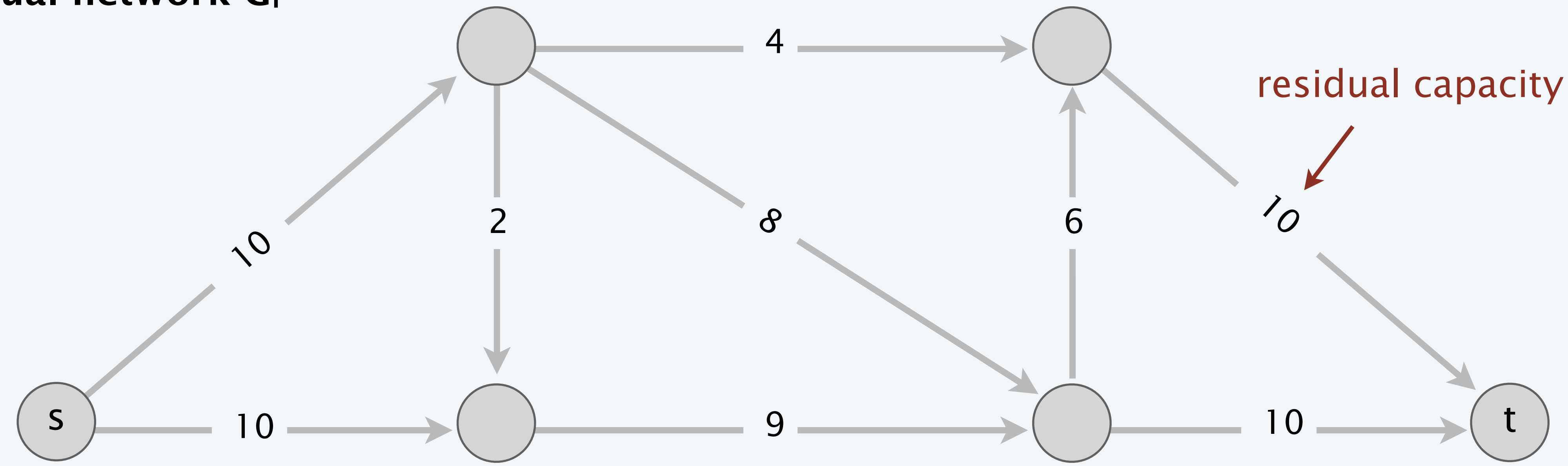
augmenting path

Ford–Fulkerson algorithm demo

network G and flow f

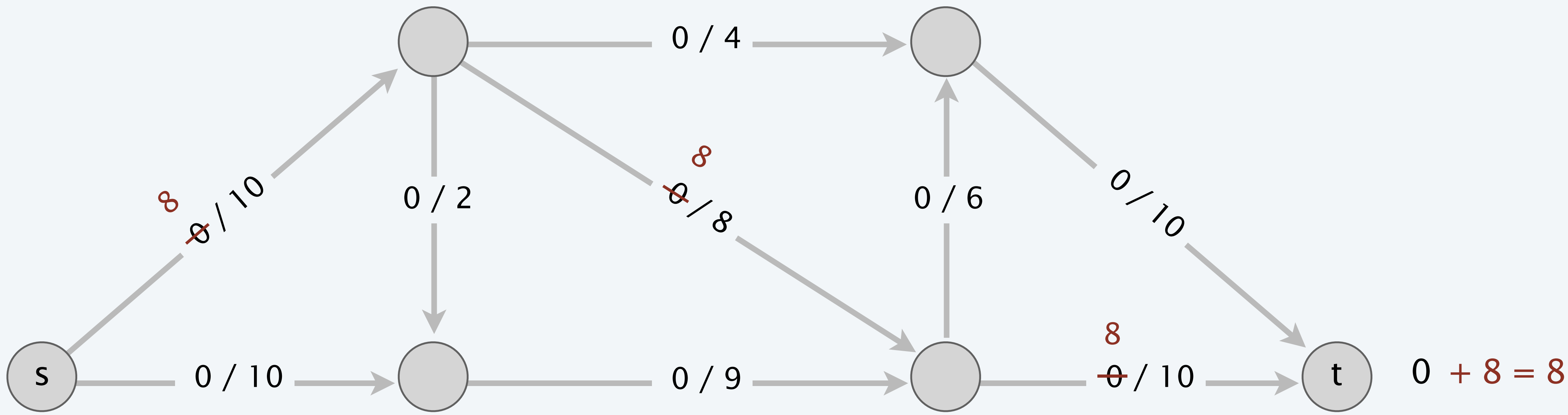


residual network G_f

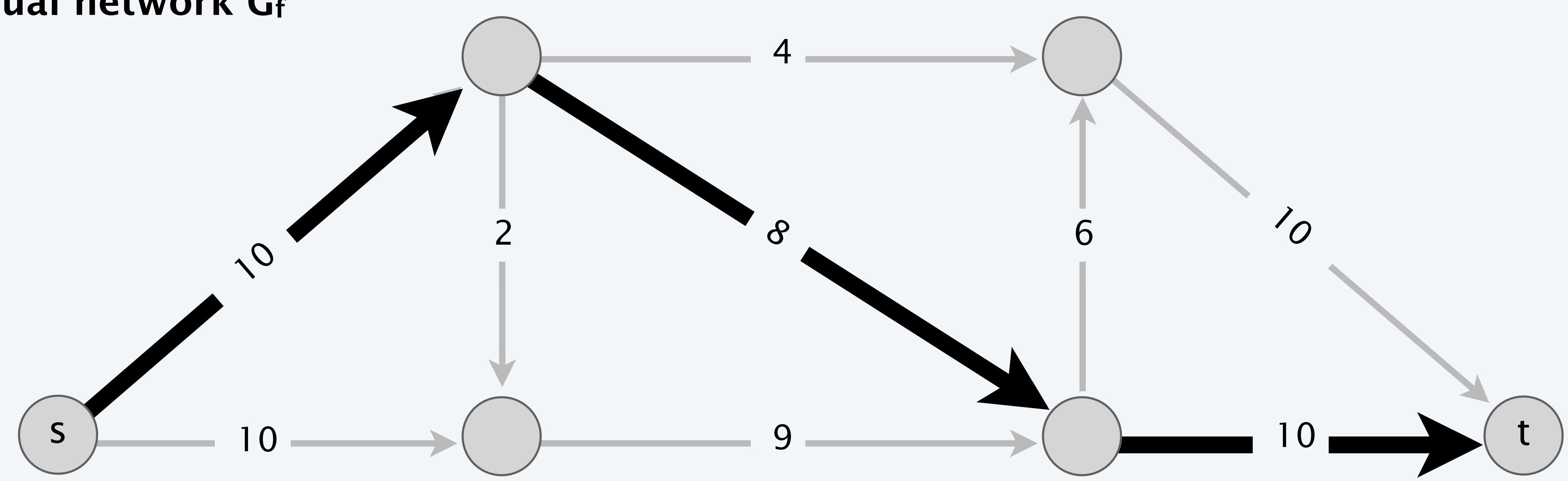


Ford–Fulkerson algorithm demo

network G and flow f

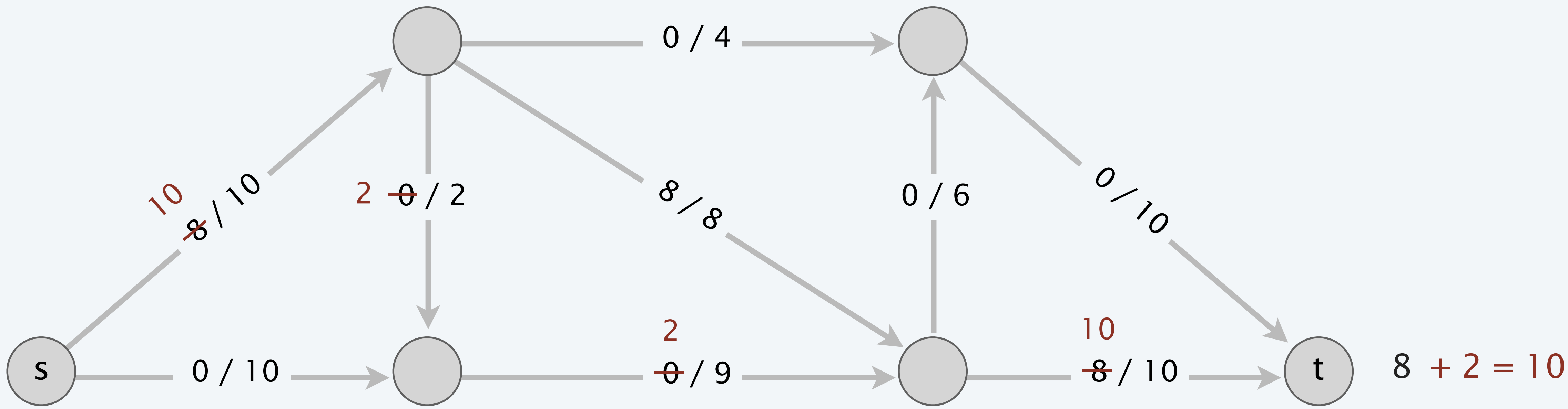


residual network G_f

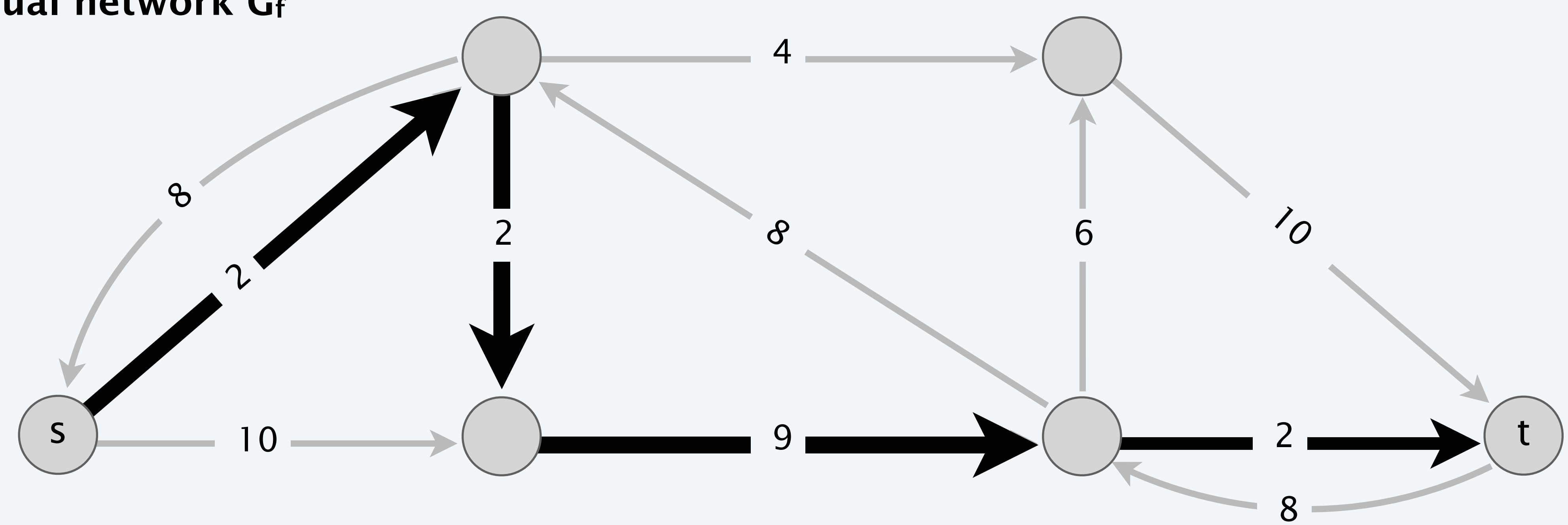


Ford–Fulkerson algorithm demo

network G and flow f

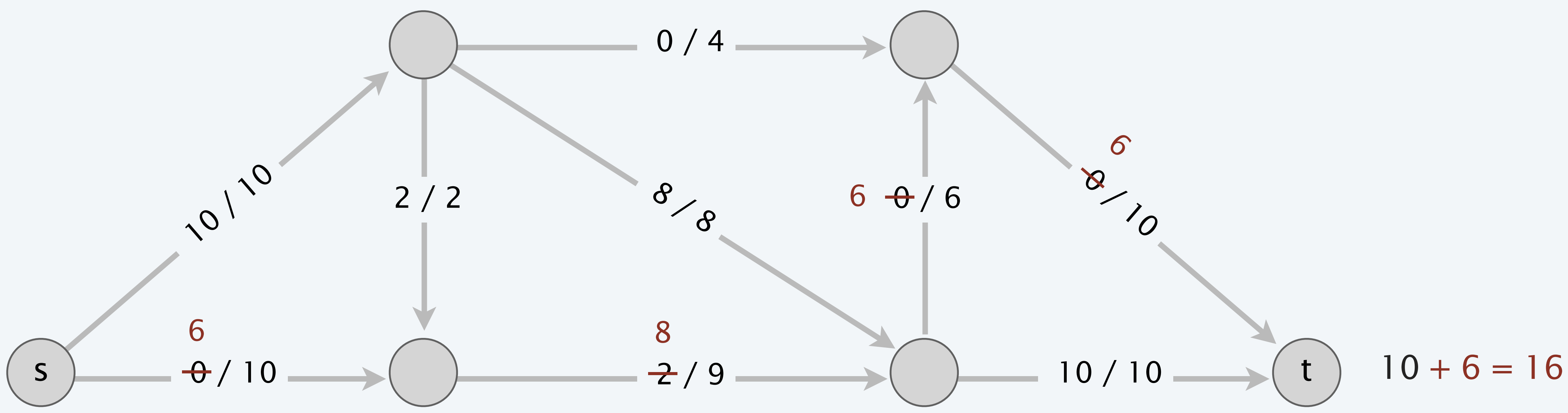


residual network G_f

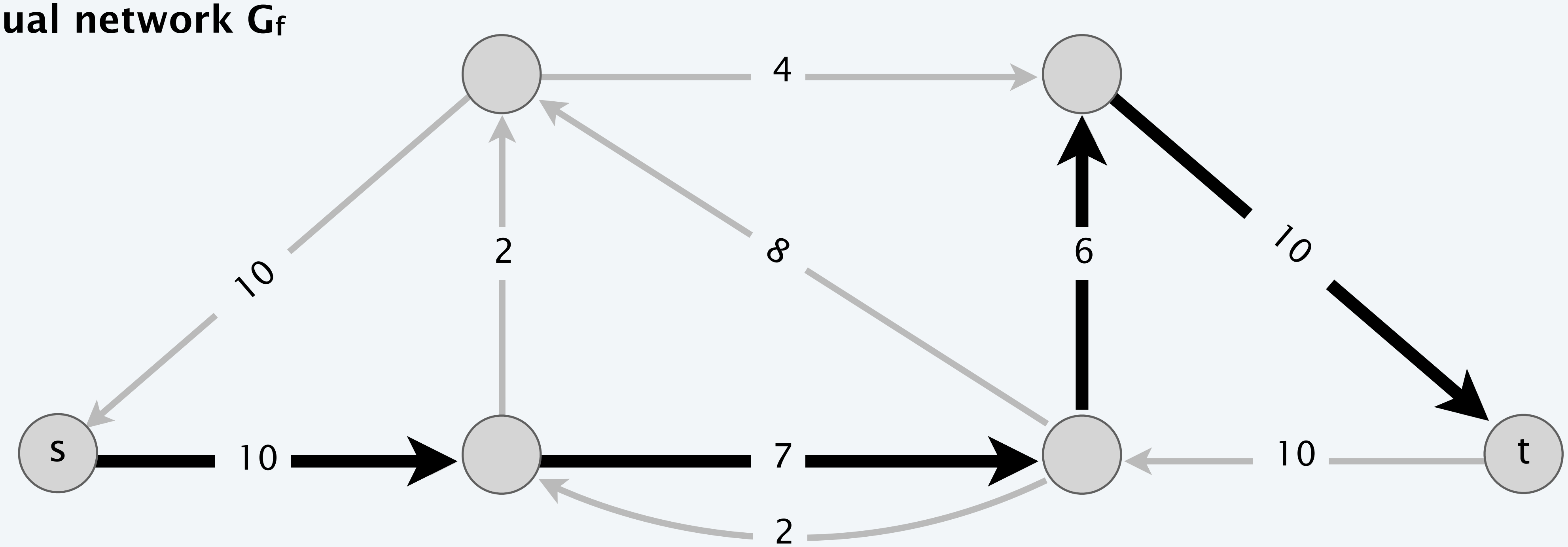


Ford–Fulkerson algorithm demo

network G and flow f

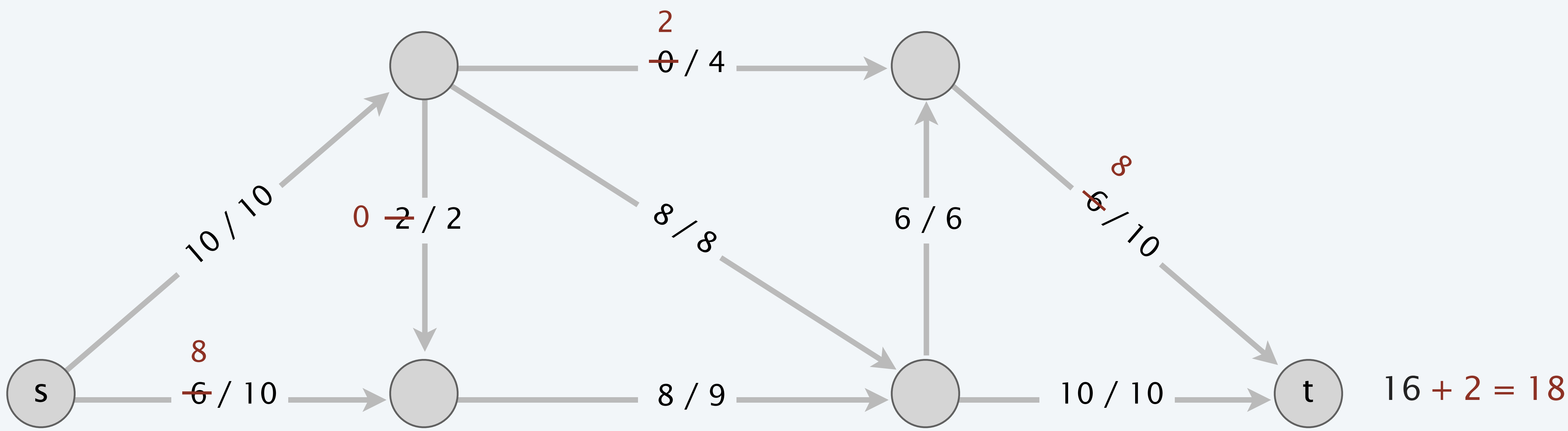


residual network G_f



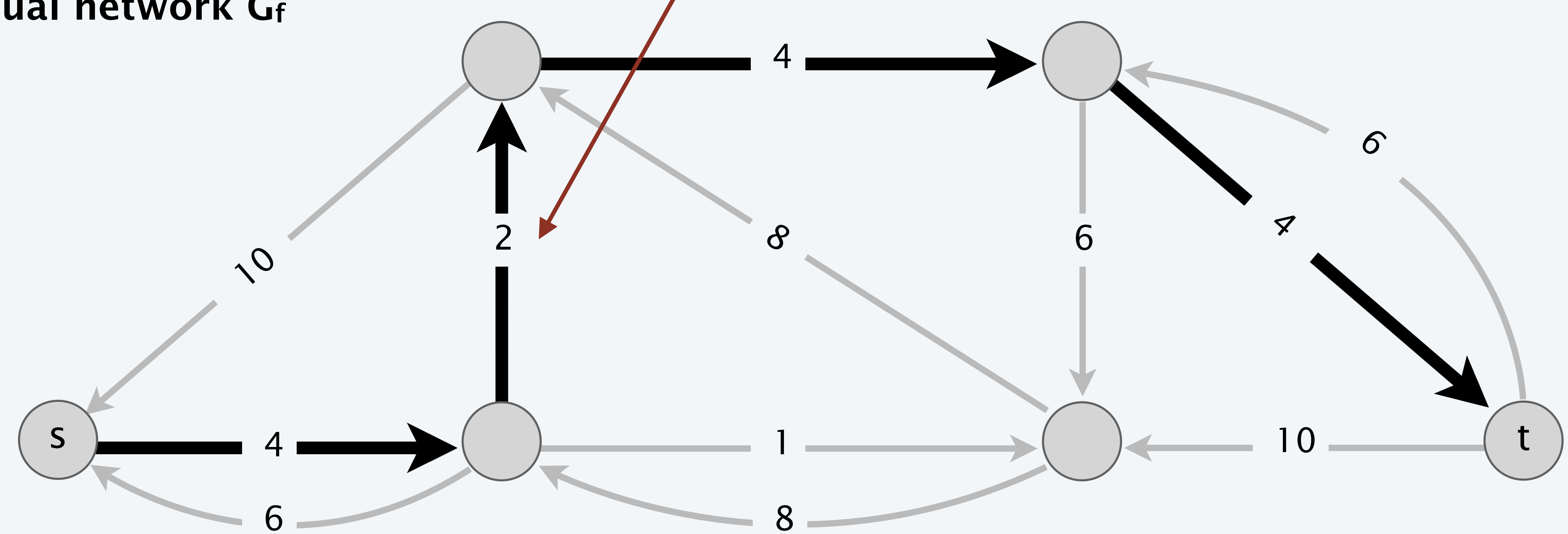
Ford–Fulkerson algorithm demo

network G and flow f



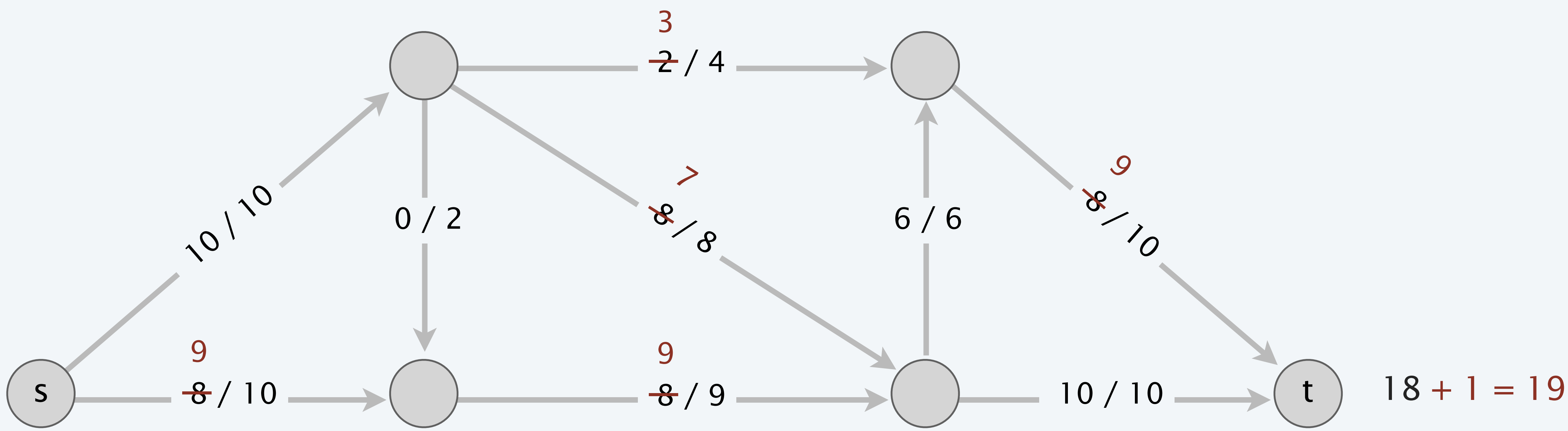
fixes mistake from
second augmenting path

residual network G_f

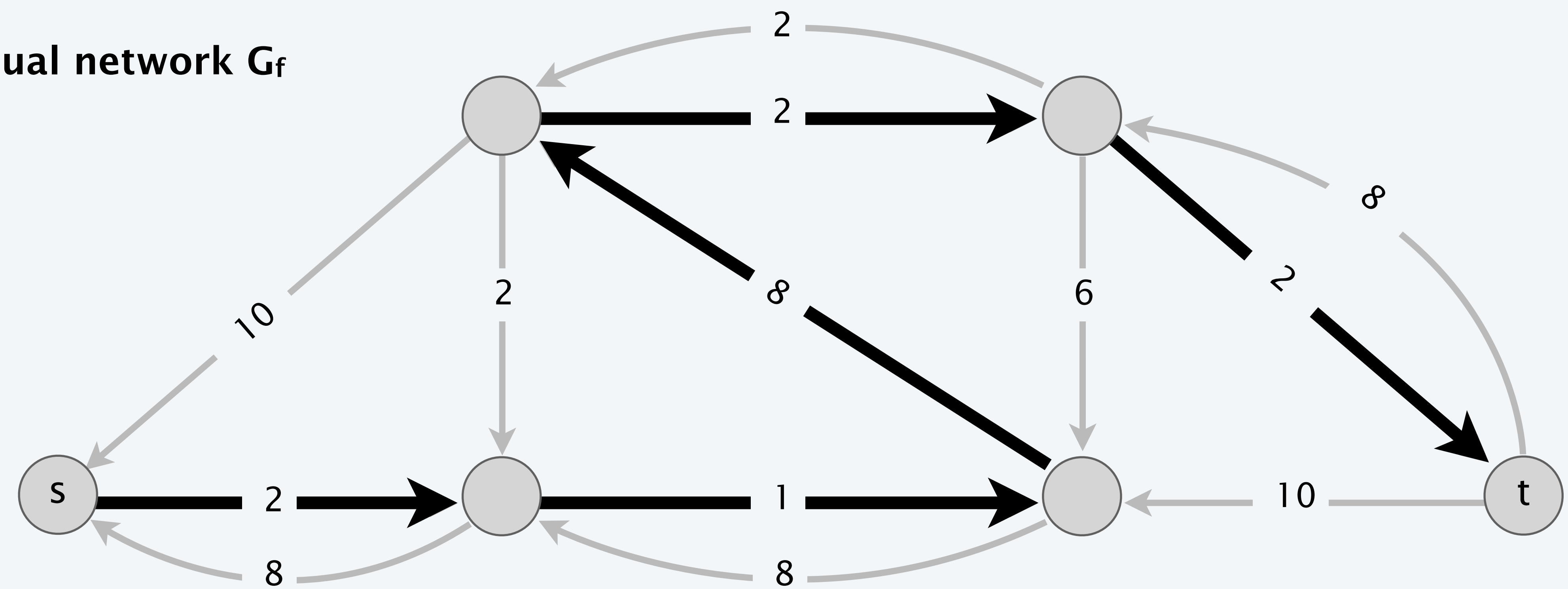


Ford–Fulkerson algorithm demo

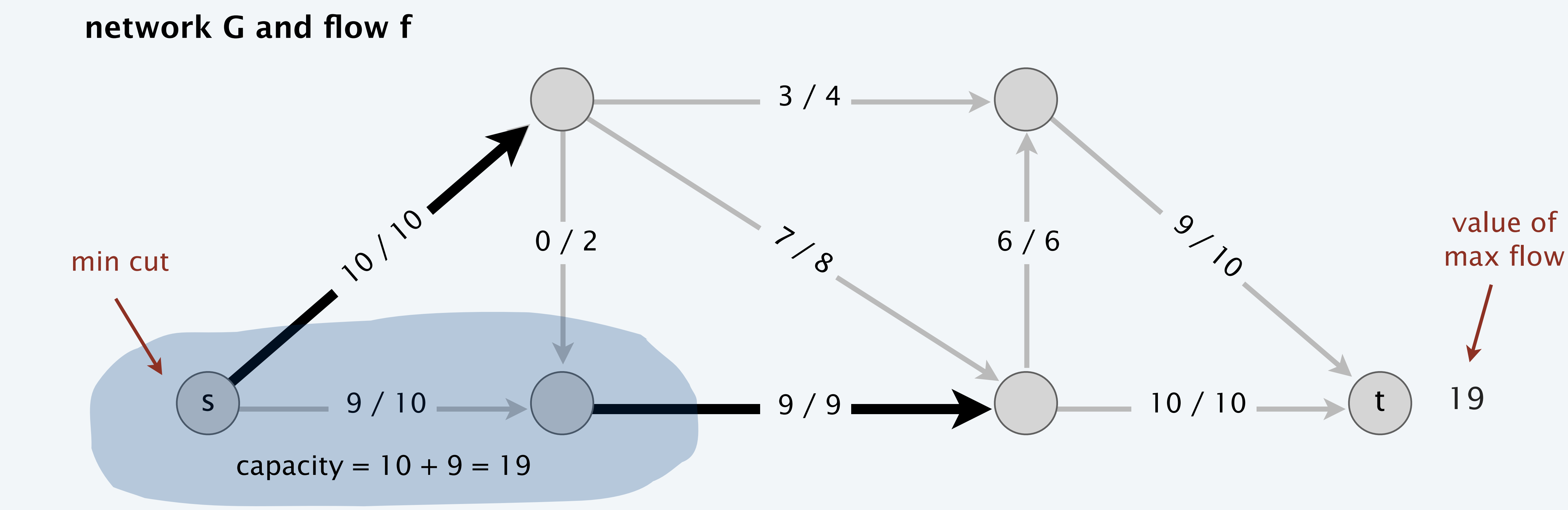
network G and flow f

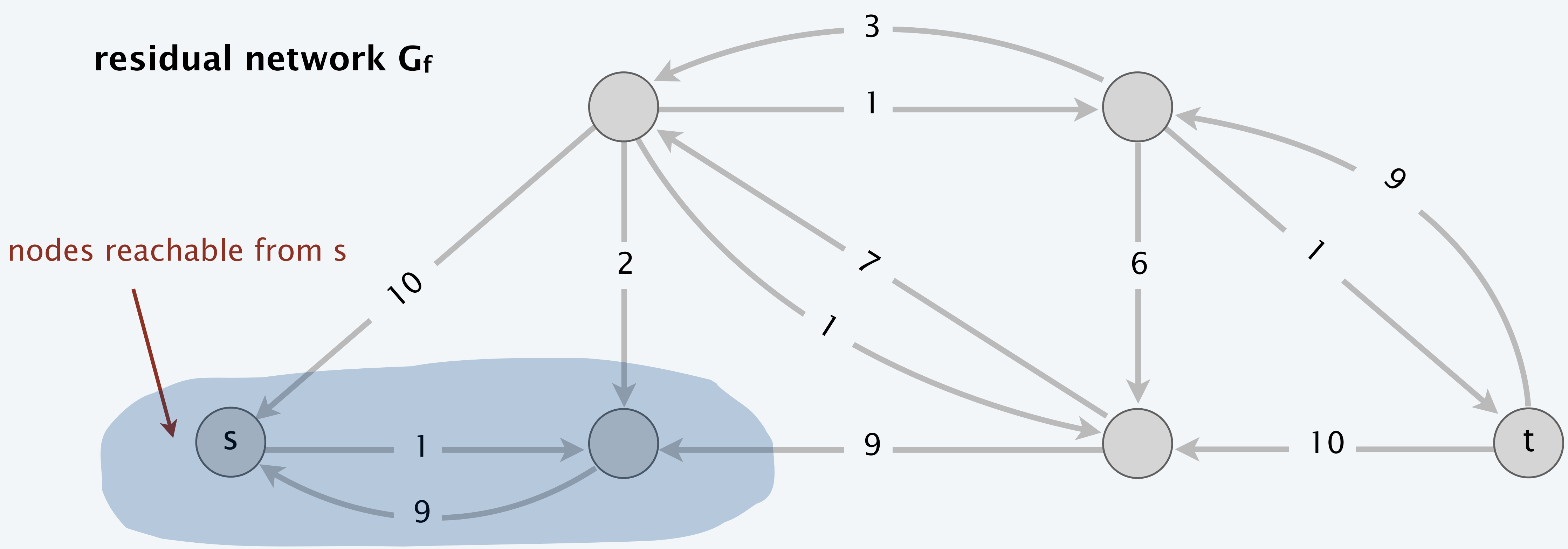


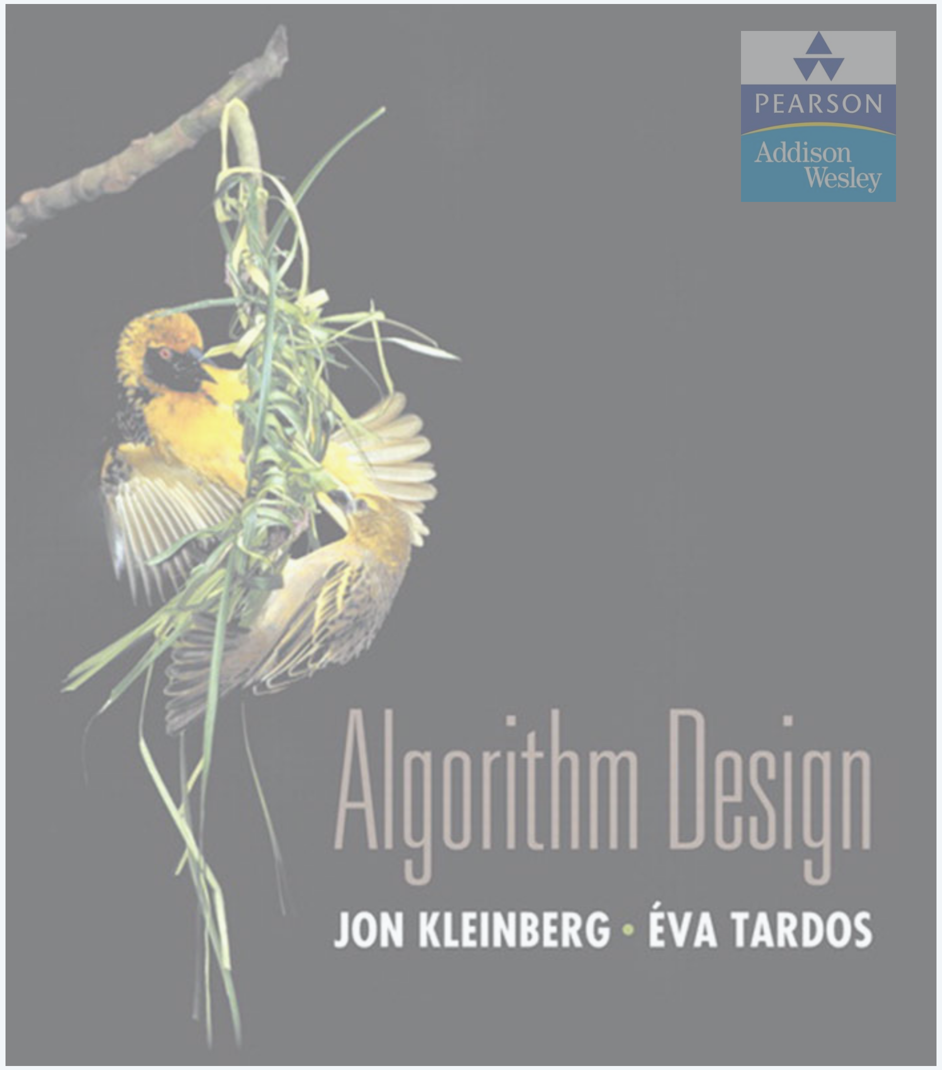
residual network G_f



Ford–Fulkerson algorithm demo







SECTION 7.2

NETWORK FLOW

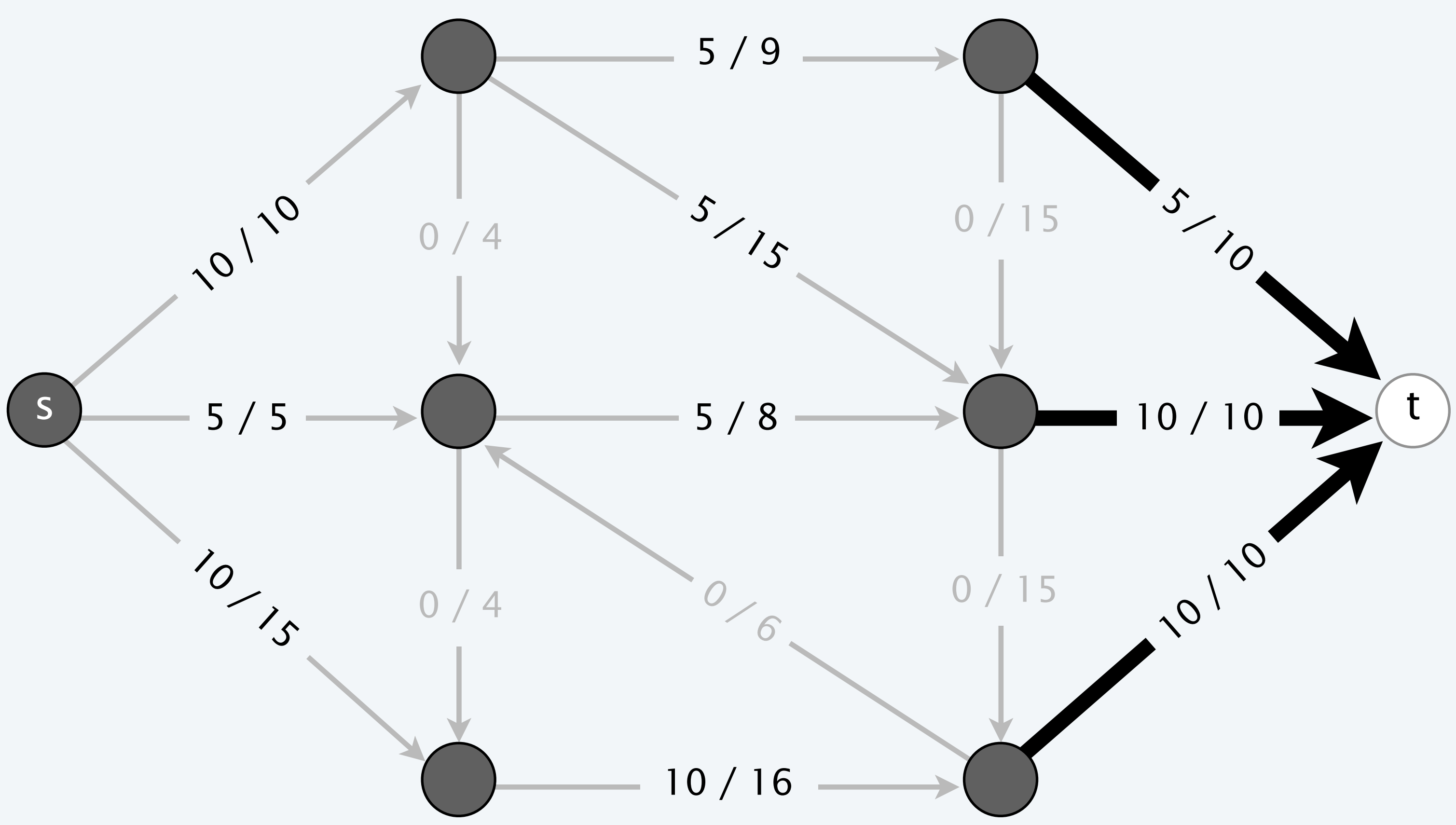
- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*

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

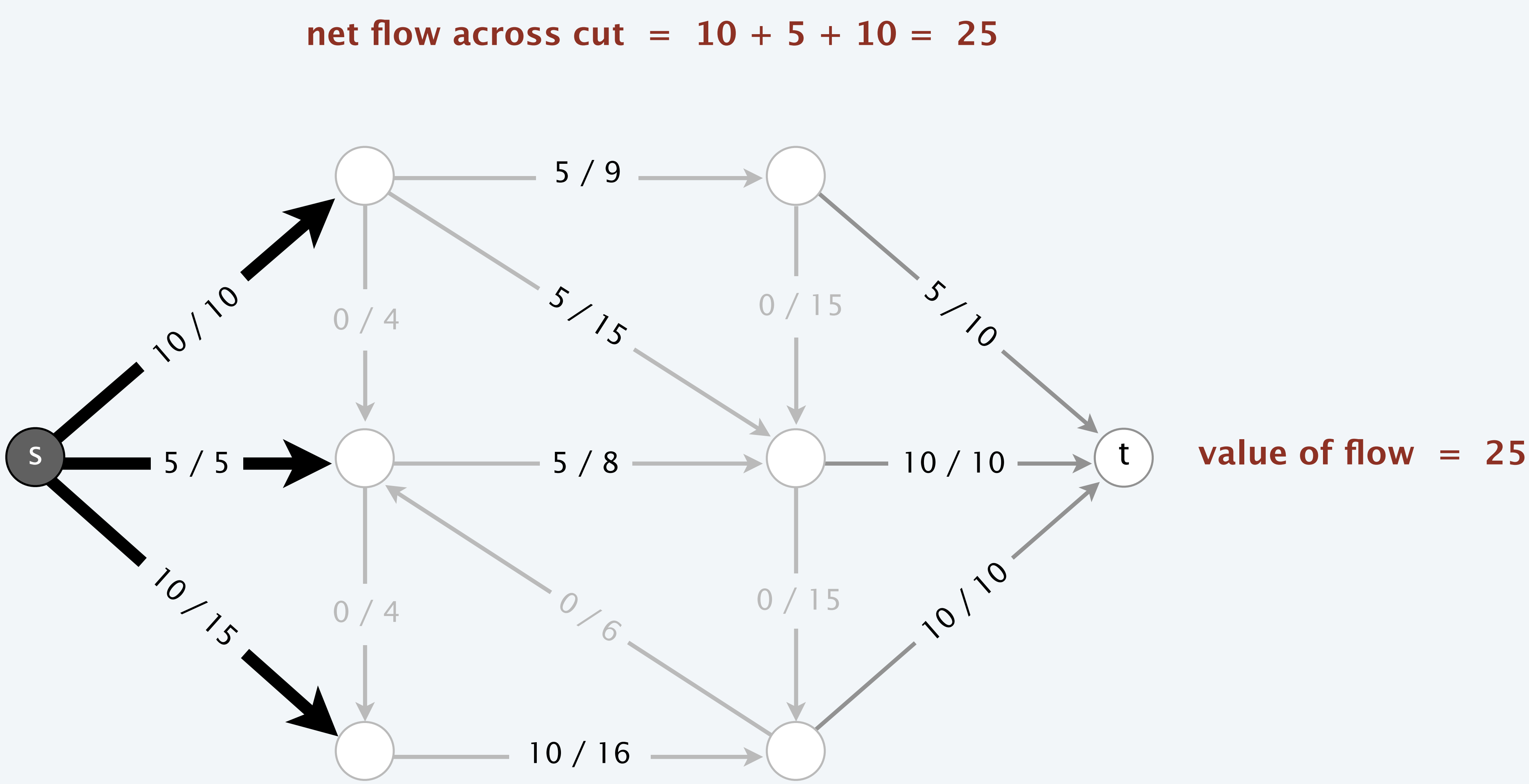


value of flow = 25

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).

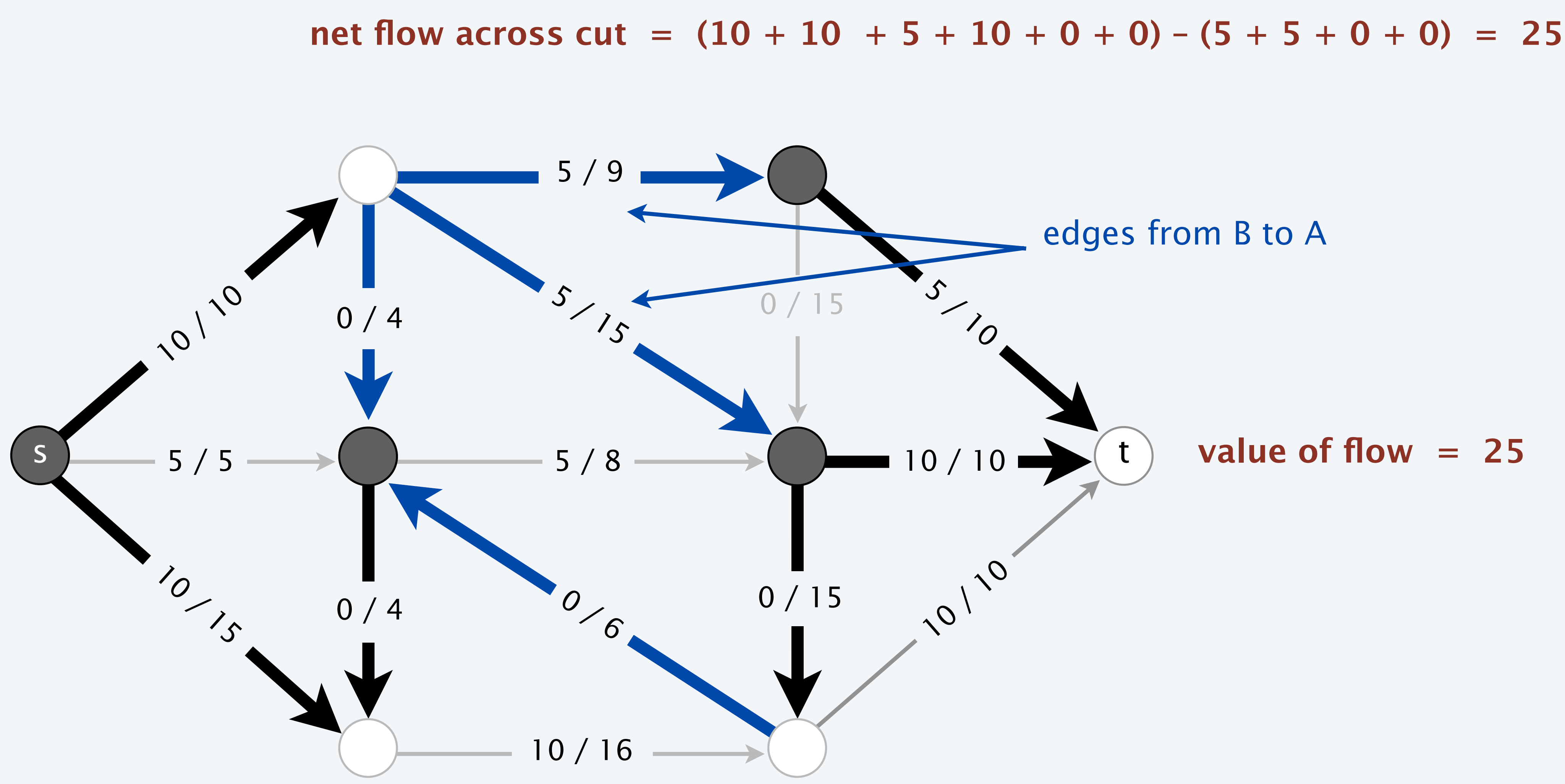
$$val(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$



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).

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



Network flow: quiz 3

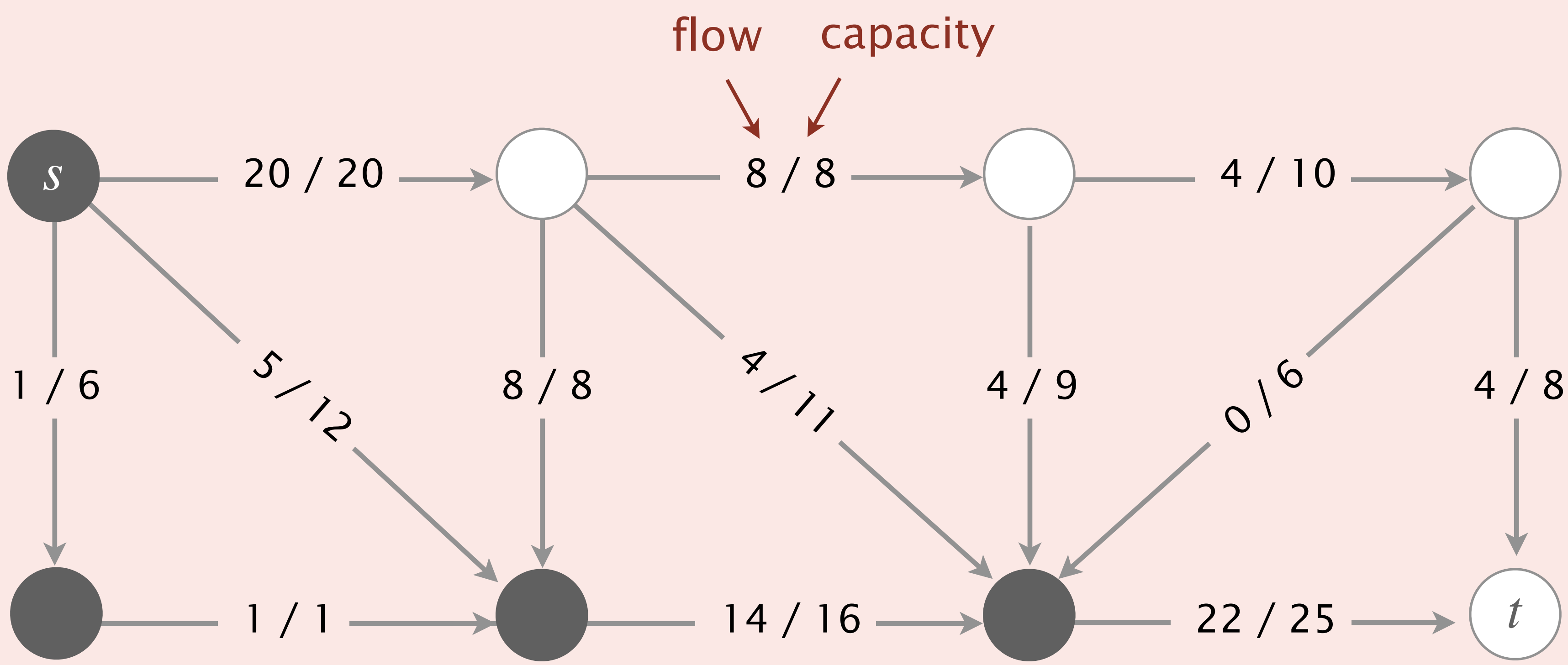
Which is the net flow across the given cut?

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.

$$\begin{aligned} val(f) &= \sum_{e \text{ out of } s} f(e) \;-\; \sum_{e \text{ in to } s} f(e) \\ &= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) \;-\; \sum_{e \text{ in to } v} f(e) \right) \\ &= \sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;\;\blacksquare \end{aligned}$$

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

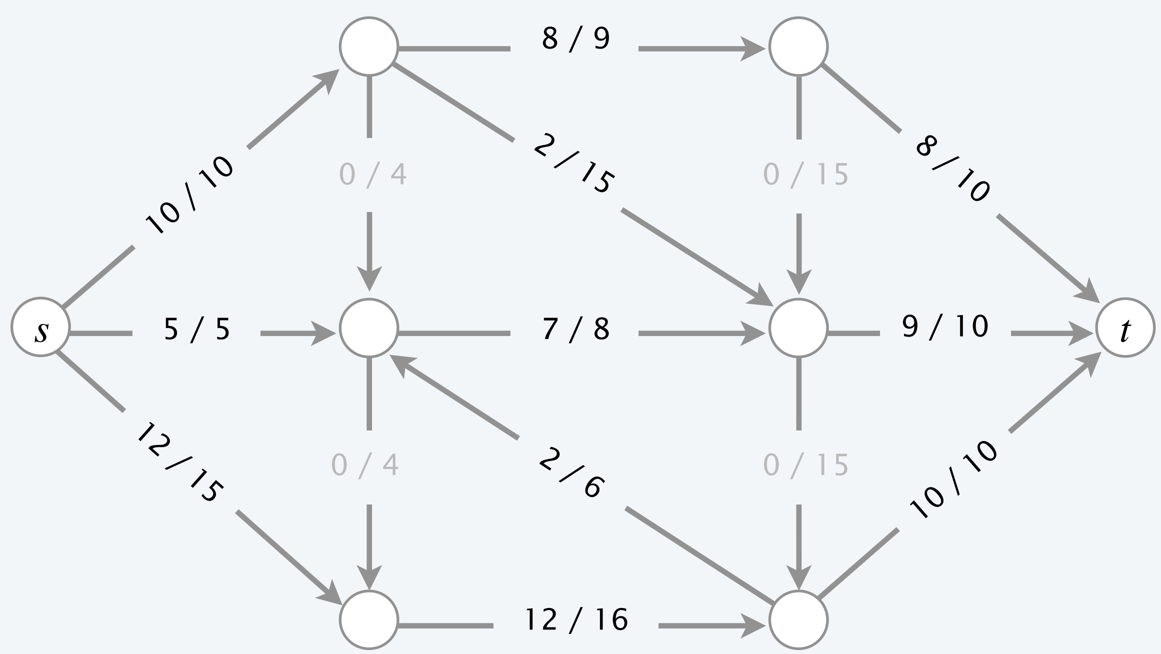
Relationship between flows and cuts

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

Pf.

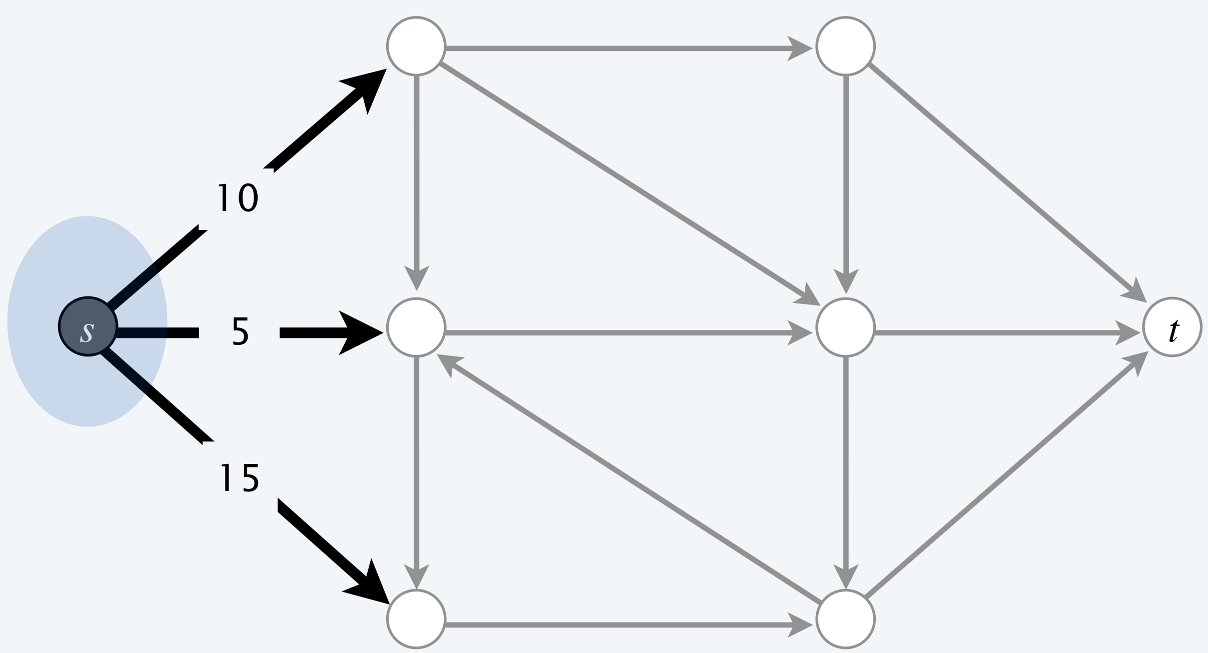
$$
\begin{aligned}
val(f) &= \sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to } A} f(e) \\
&\leq \sum_{e \text{ out of } A} f(e) \\
&\leq \sum_{e \text{ out of } A} c(e) \\
&= cap(A, B) \quad \blacksquare
\end{aligned}
$$

flow value lemma



value of flow = 27

$\leq$



capacity of cut = 30

Certificate of optimality

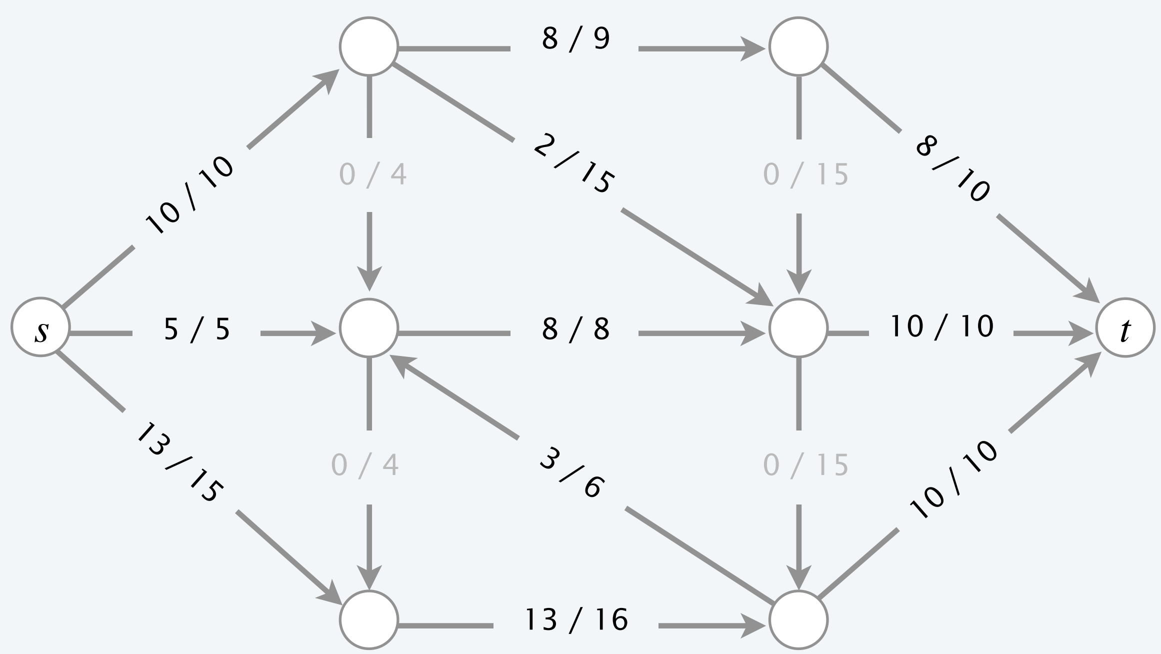
Corollary. Let f be a flow and let (A, B) be any cut.
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

weak duality

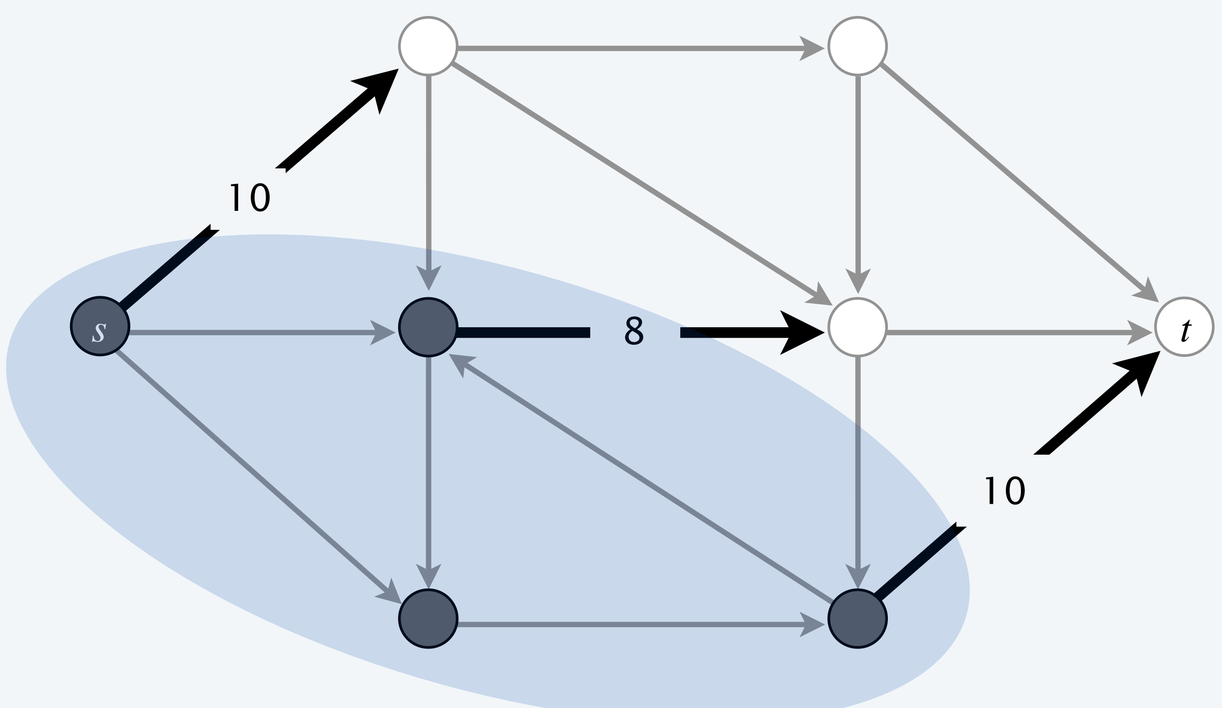
- For any flow f': $val(f') \leq cap(A, B) = val(f)$.
- For any cut (A', B'): $cap(A', B') \geq val(f) = cap(A, B)$. ▪

weak duality



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*, *g*, *h*, *i*) is not. When a simple cut-set is