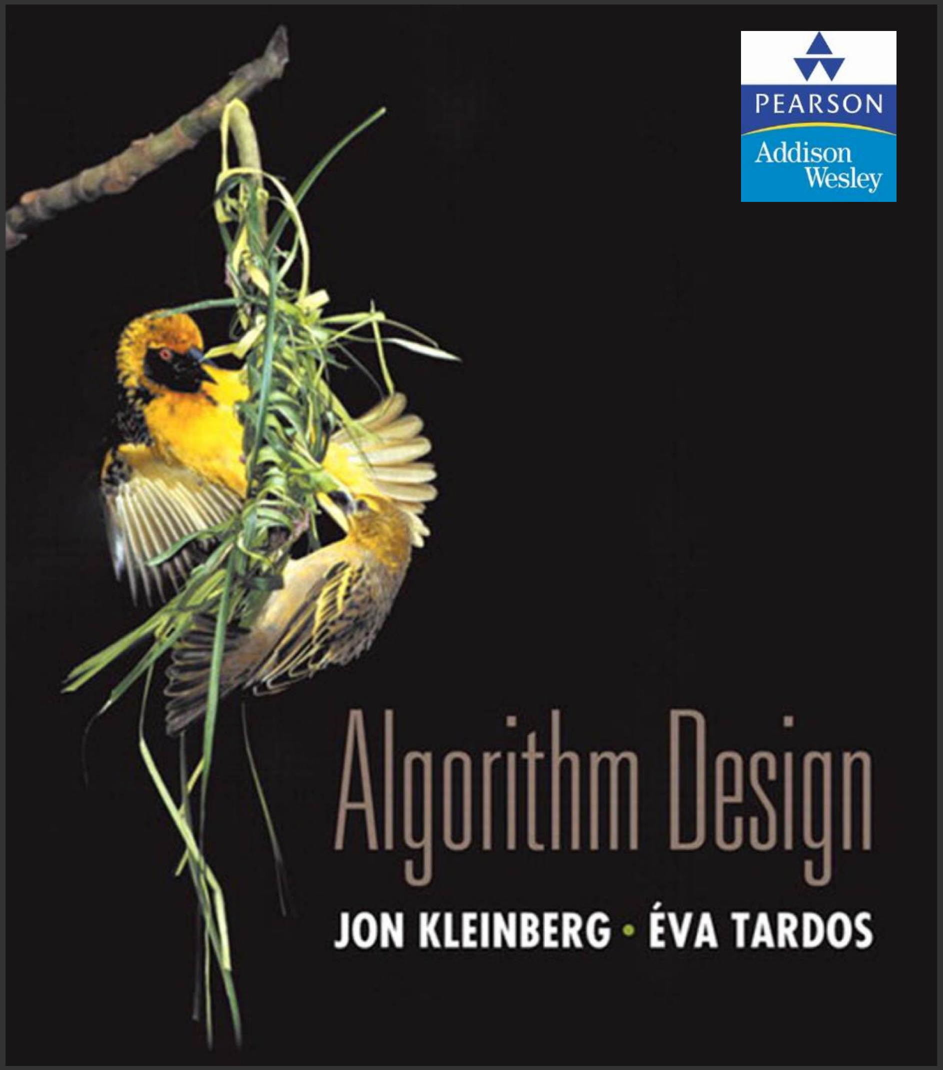
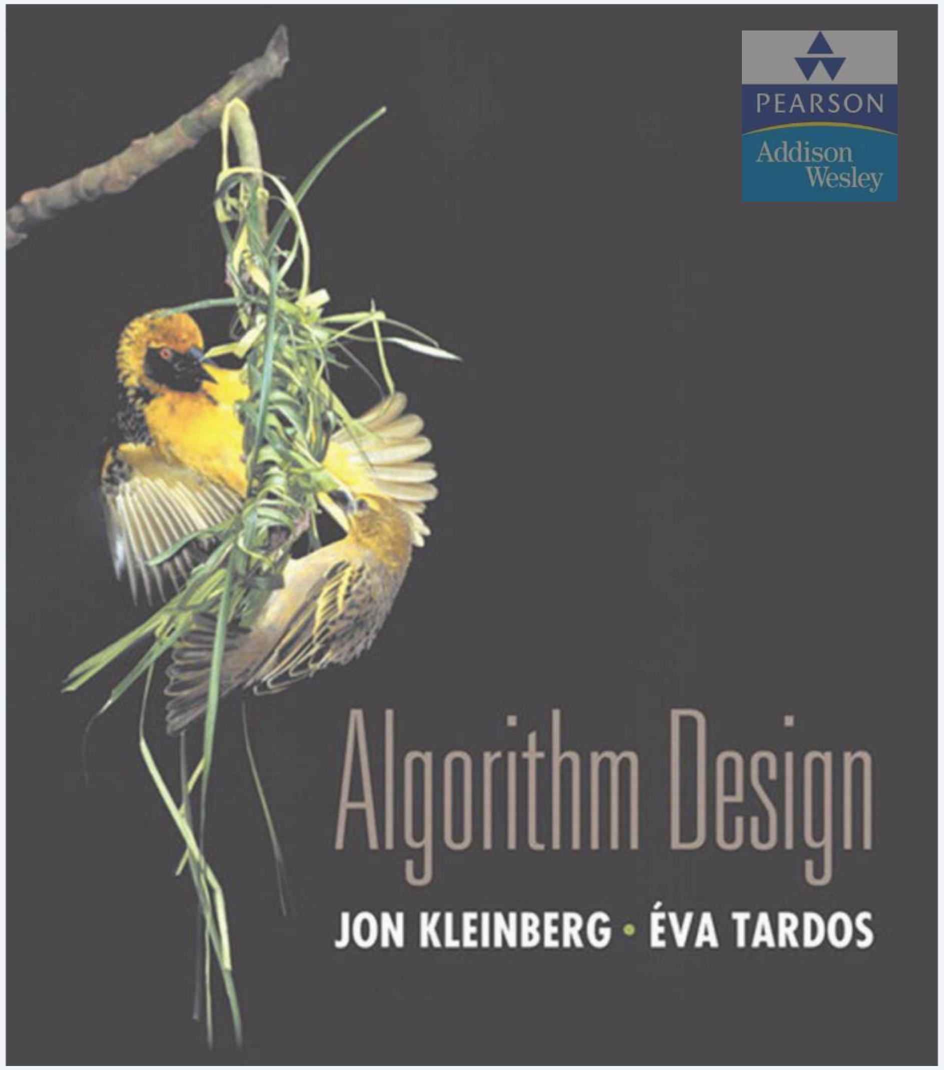


7. Network Flow I

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *choosing good augmenting paths*

Lecture slides by Kevin Wayne

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



SECTION 7.1

7. NETWORK FLOW I

- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *choosing good augmenting paths*

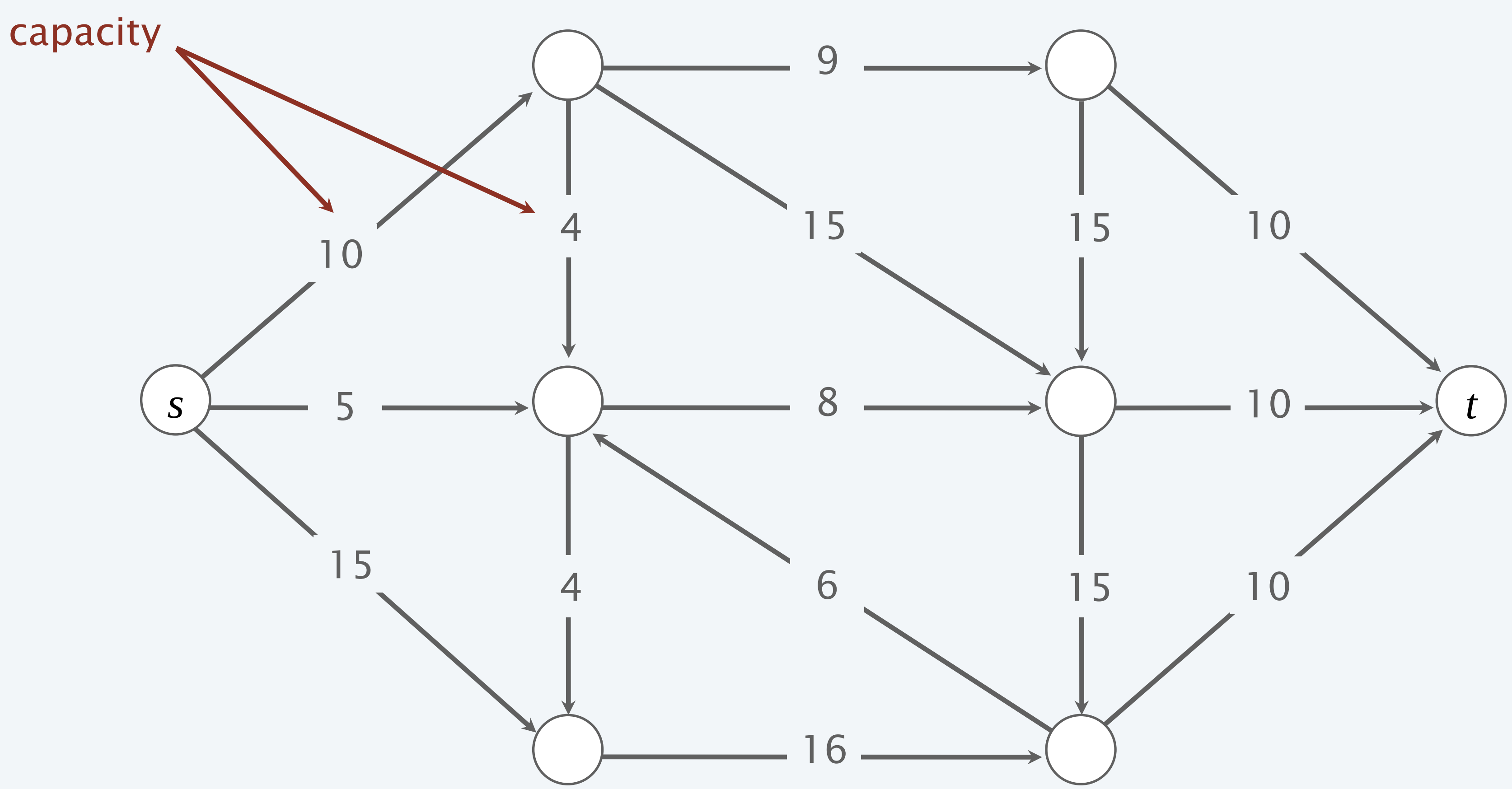
Flow network

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 $c(e) \geq 0$ 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.

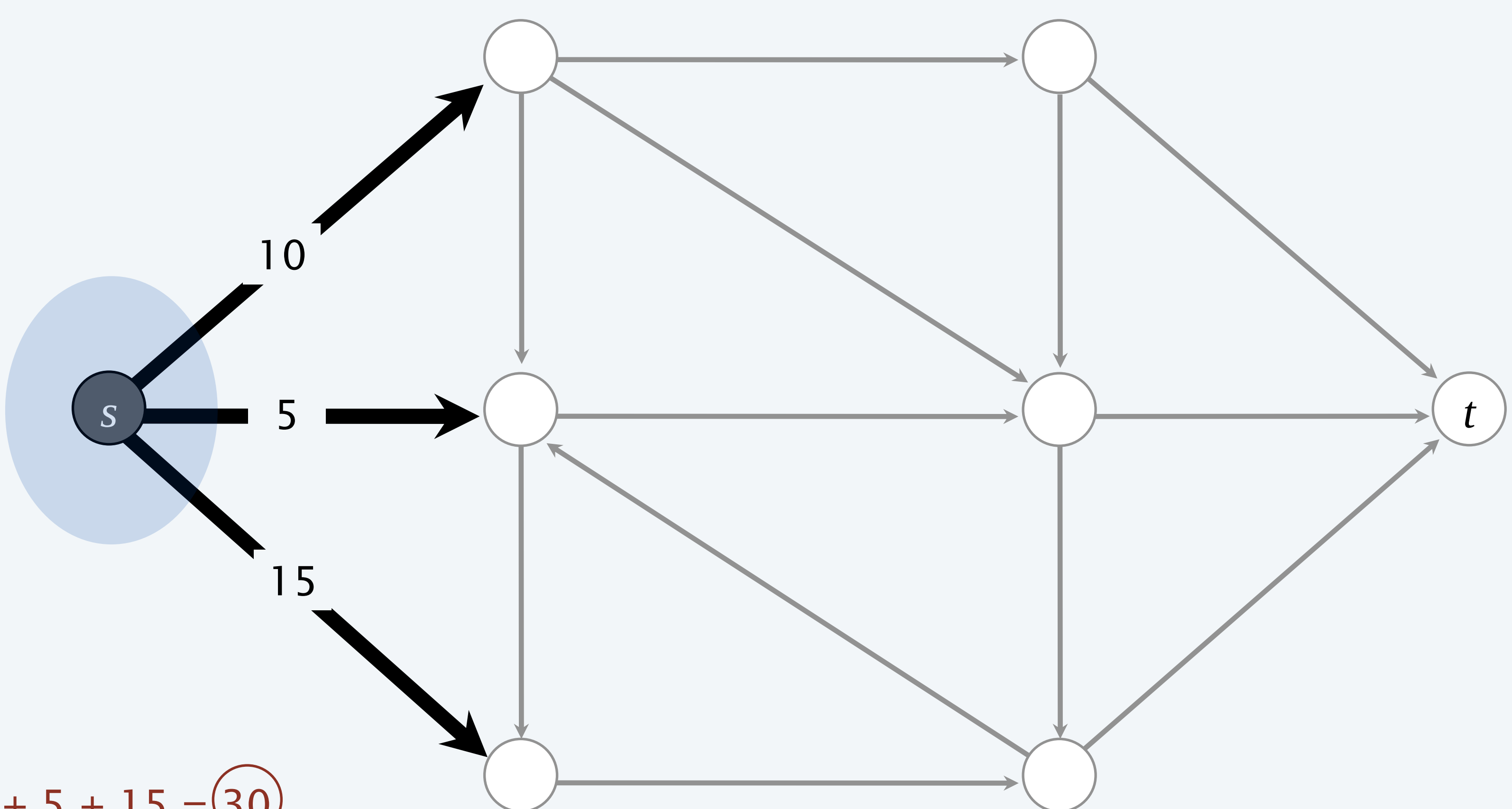


Minimum-cut problem

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.

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



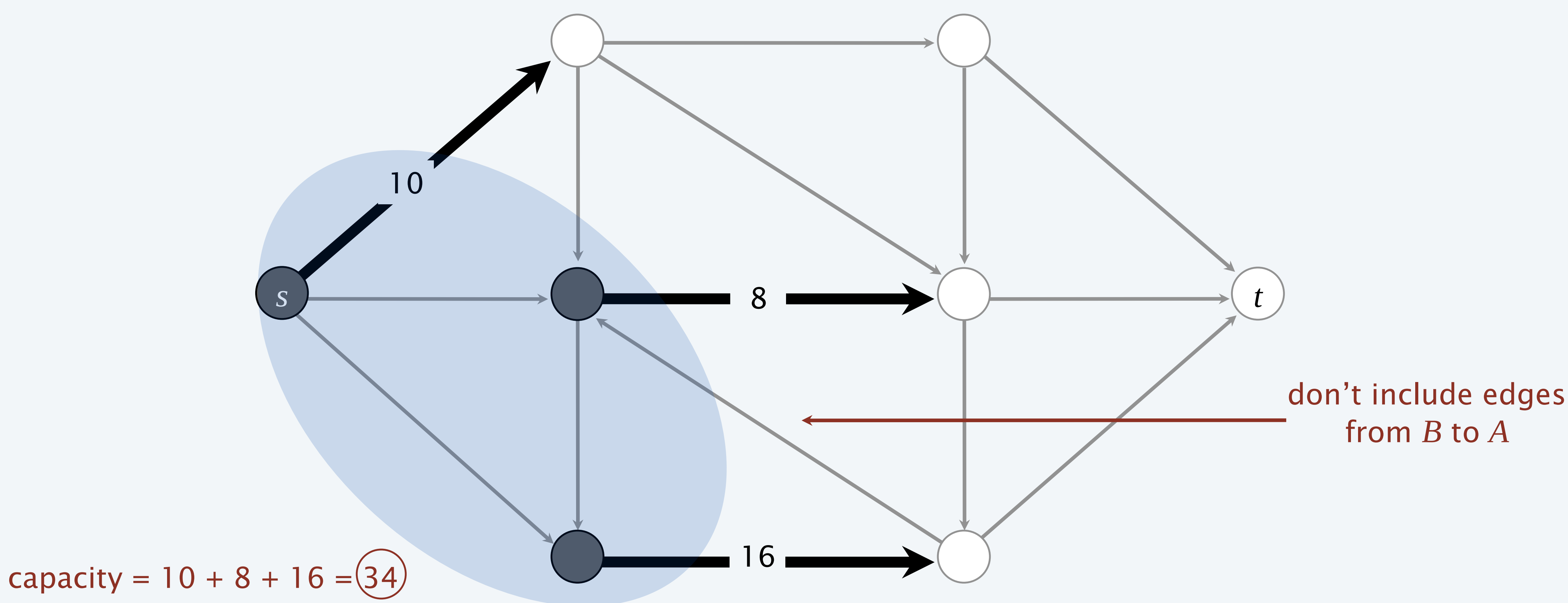
capacity = 10 + 5 + 15 = 30

Minimum-cut problem

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

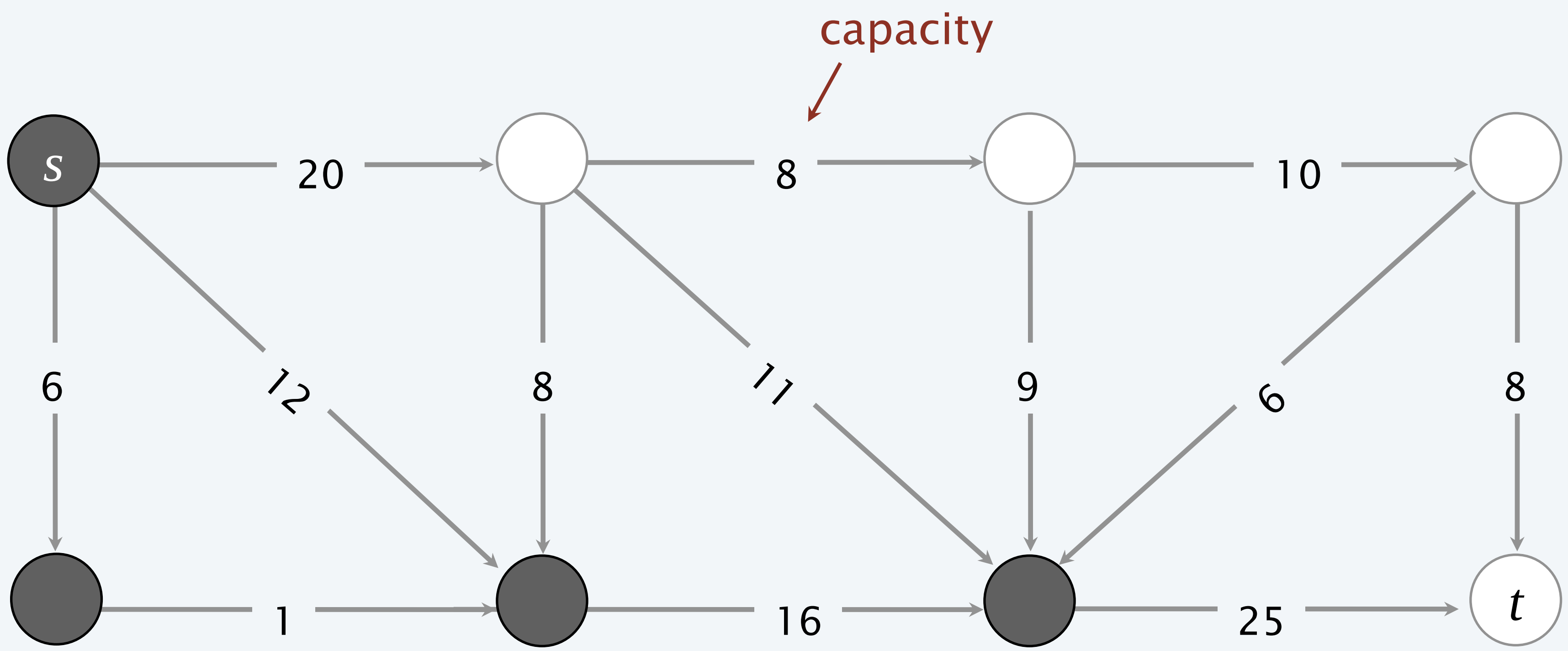
Def. Its capacity is the sum of the capacities of the edges from A to B.

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



One more example

Capacity of the given *st*-cut: 20+25=45



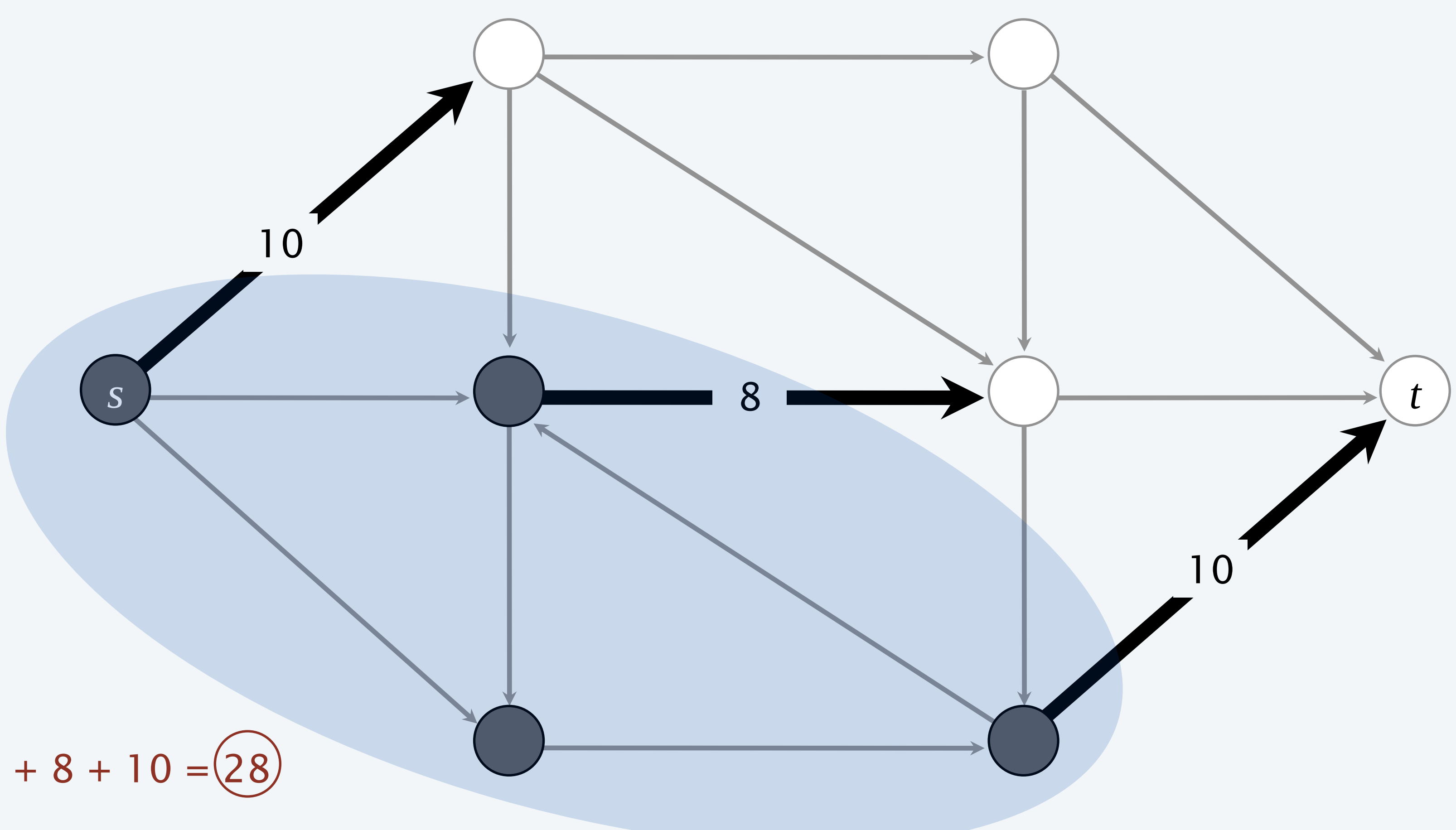
Minimum-cut problem

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.

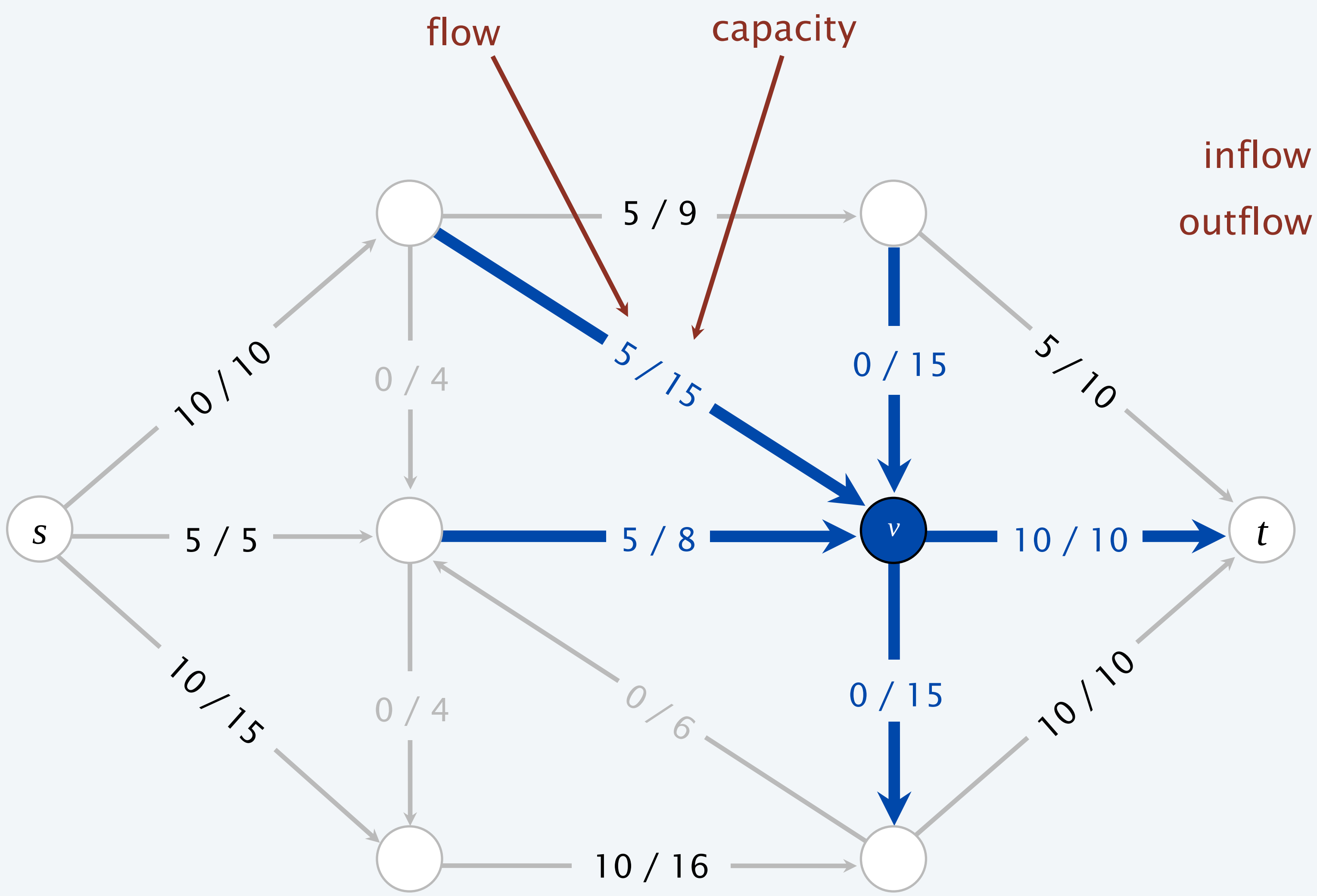


capacity = 10 + 8 + 10 = 28

Maximum-flow problem

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

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



inflow at v = $5 + 5 + 0$ = 10

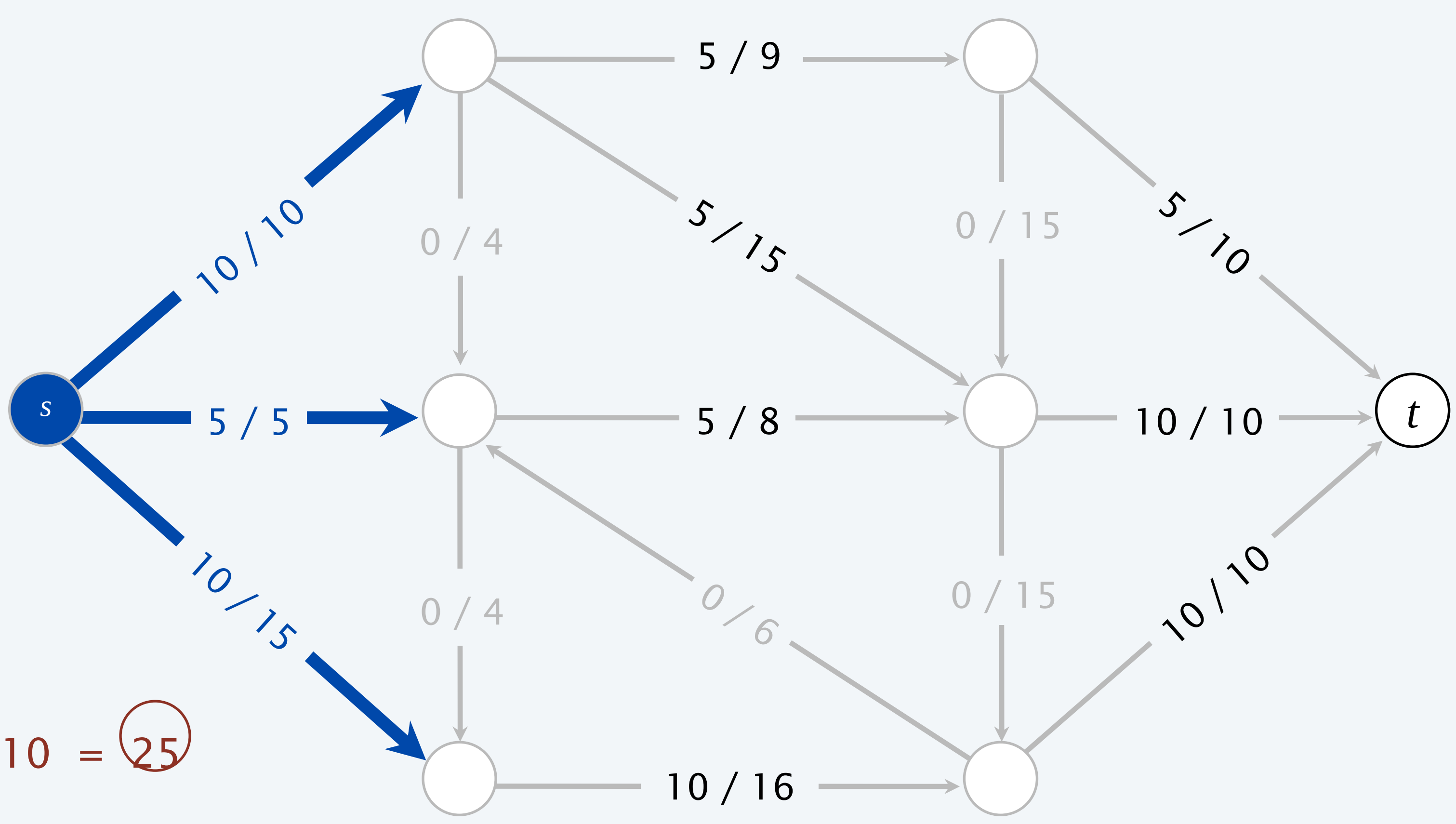
outflow at v = $10 + 0$ = 10

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 $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \quad$ [flow conservation]

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



value = 5 + 10 + 10 = 25

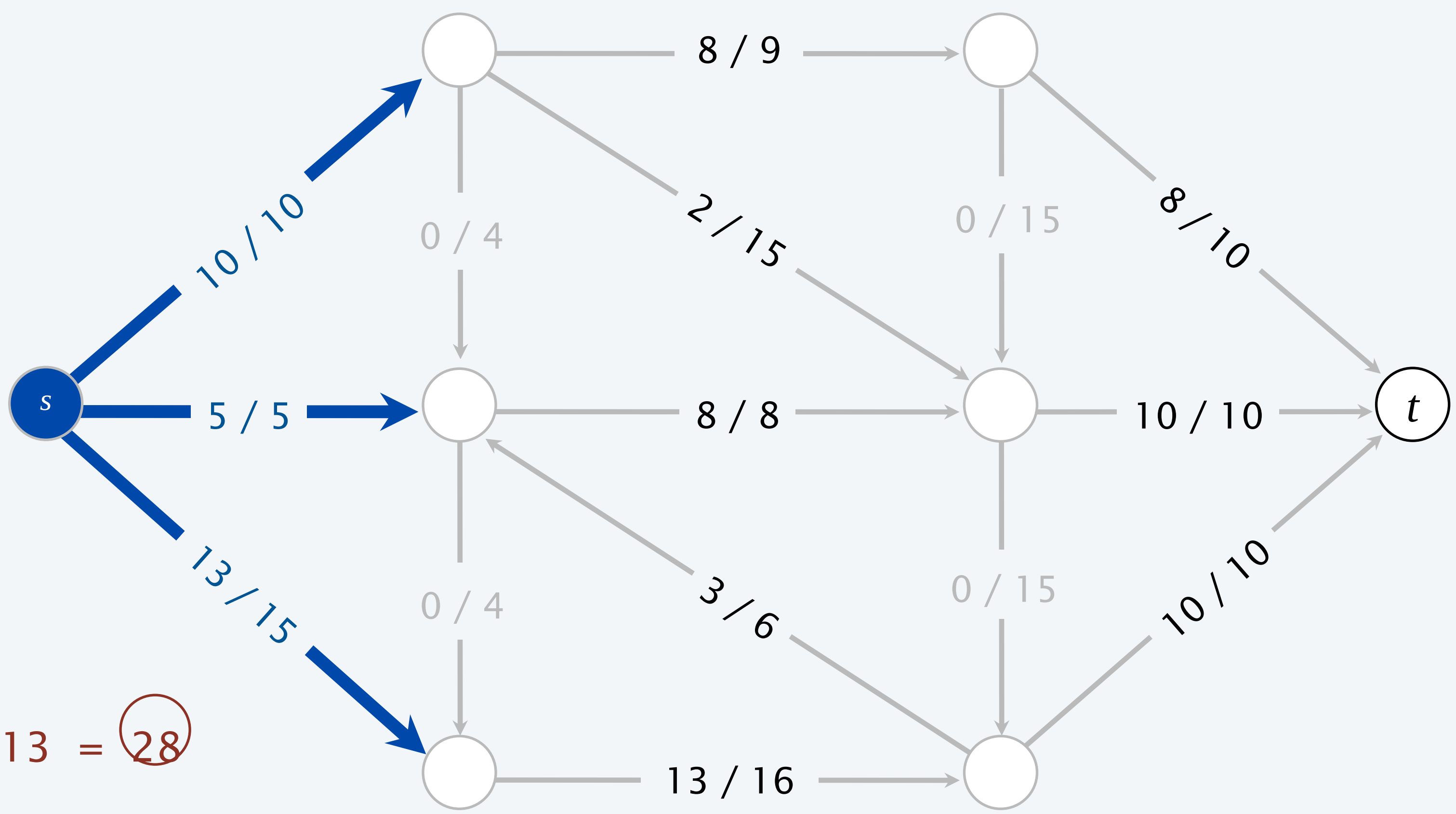
Maximum-flow problem

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

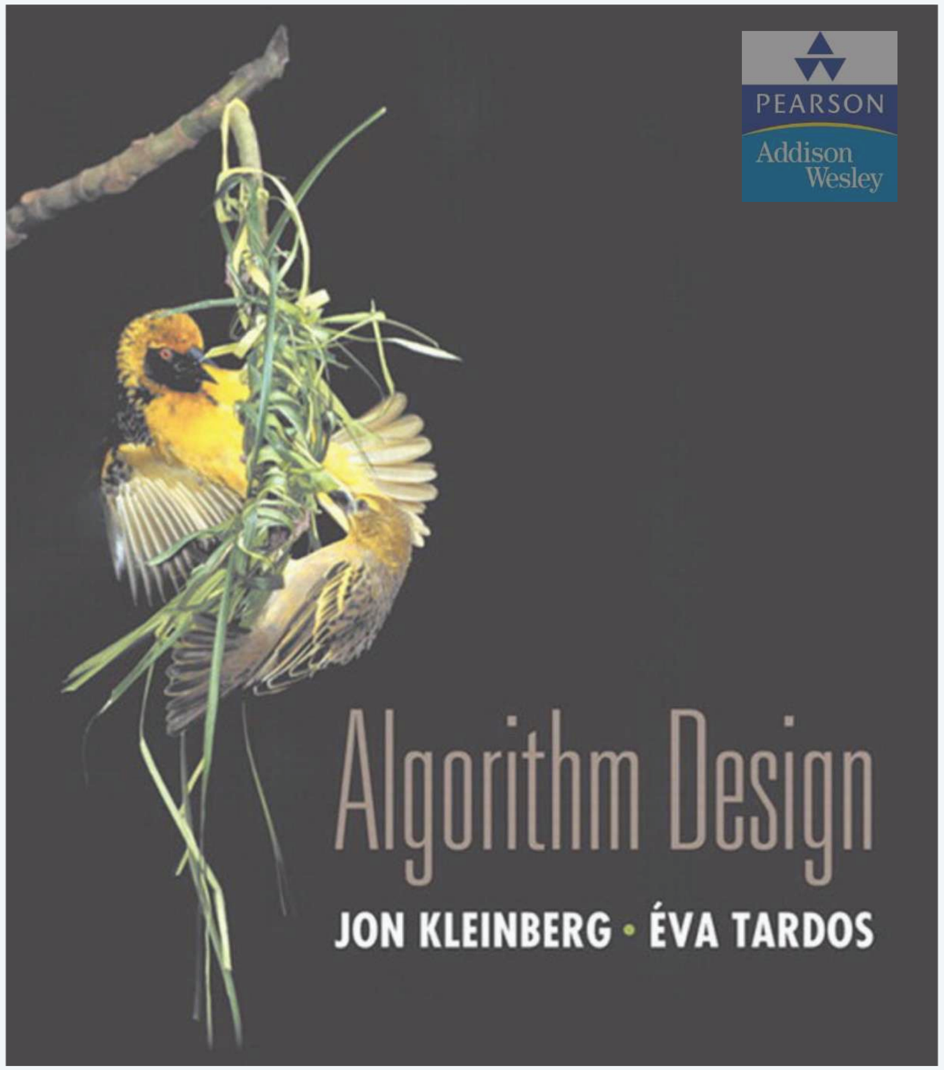
- For each $e \in E$: $\quad 0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

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.



value = 10 + 5 + 13 = 28



SECTION 7.1

7. NETWORK FLOW I

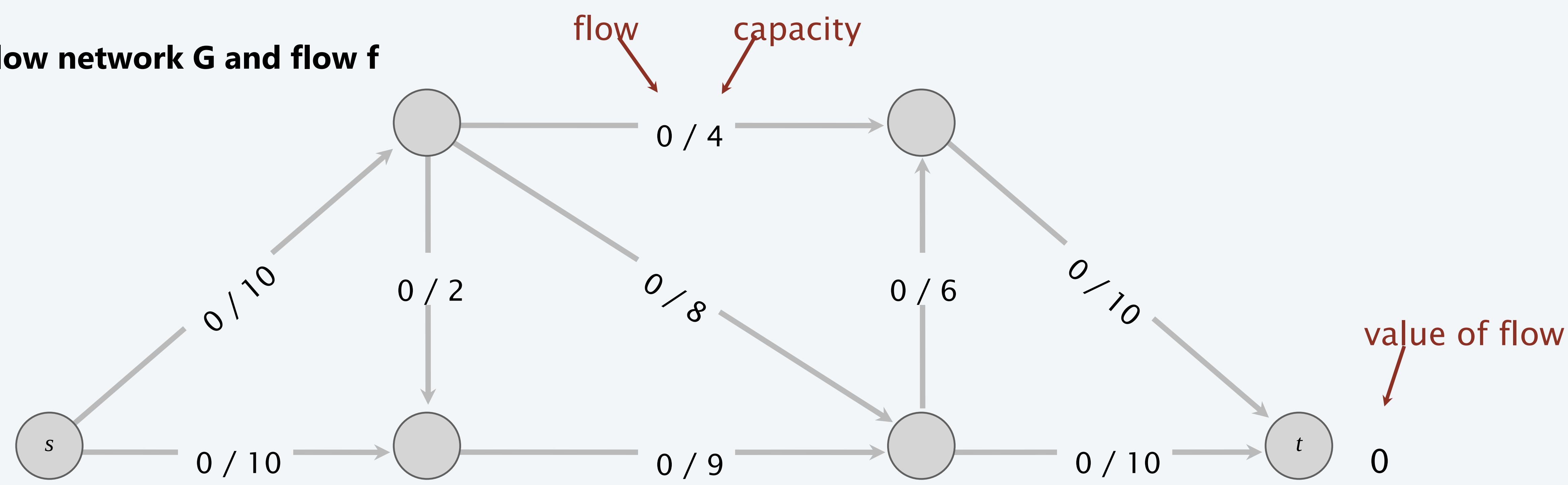
- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *choosing good augmenting paths*

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

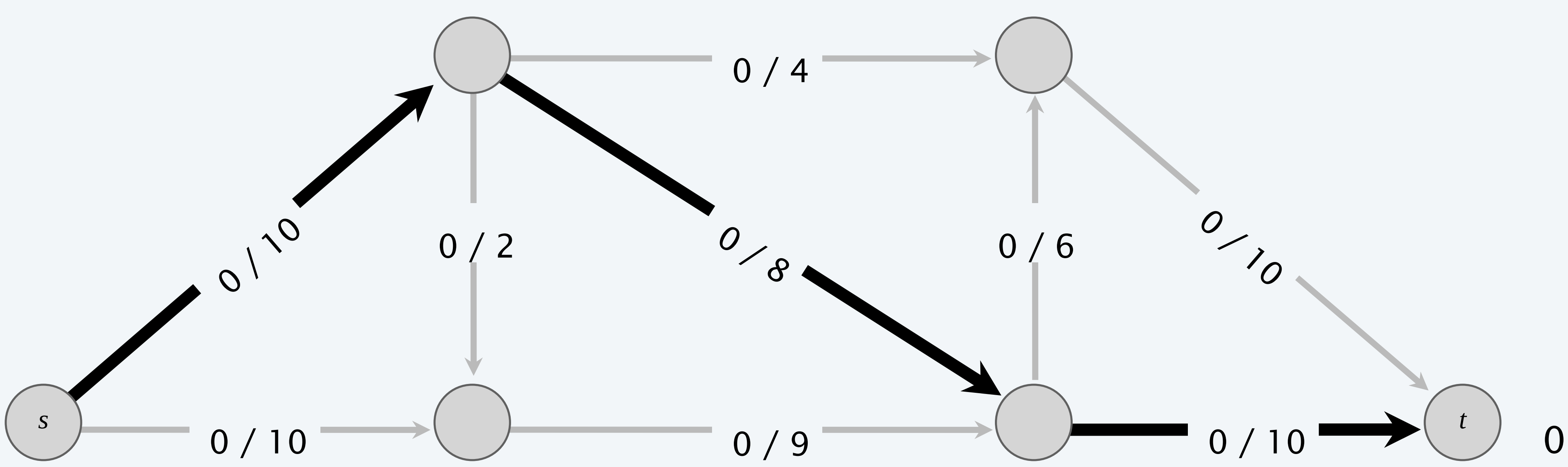


Toward a max-flow algorithm

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

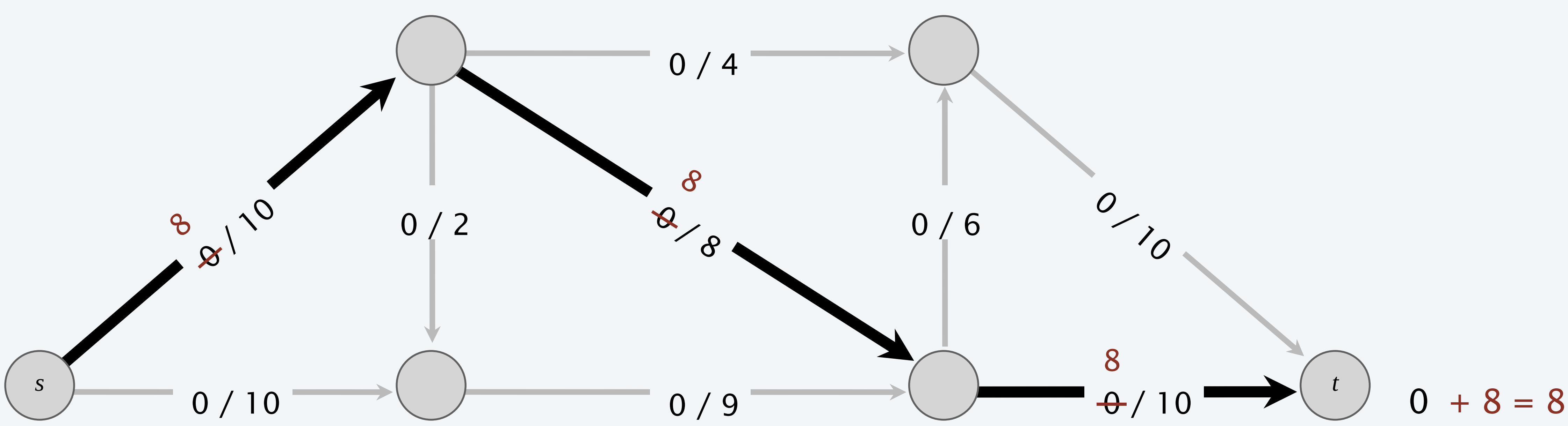


Toward a max-flow algorithm

Greedy algorithm.

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

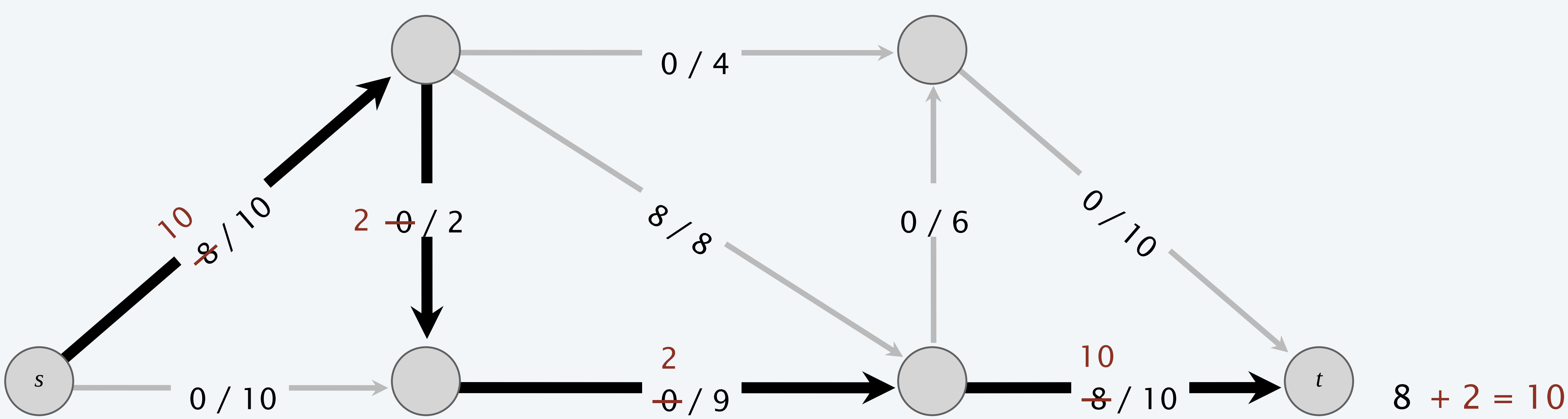


Toward a max-flow algorithm

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

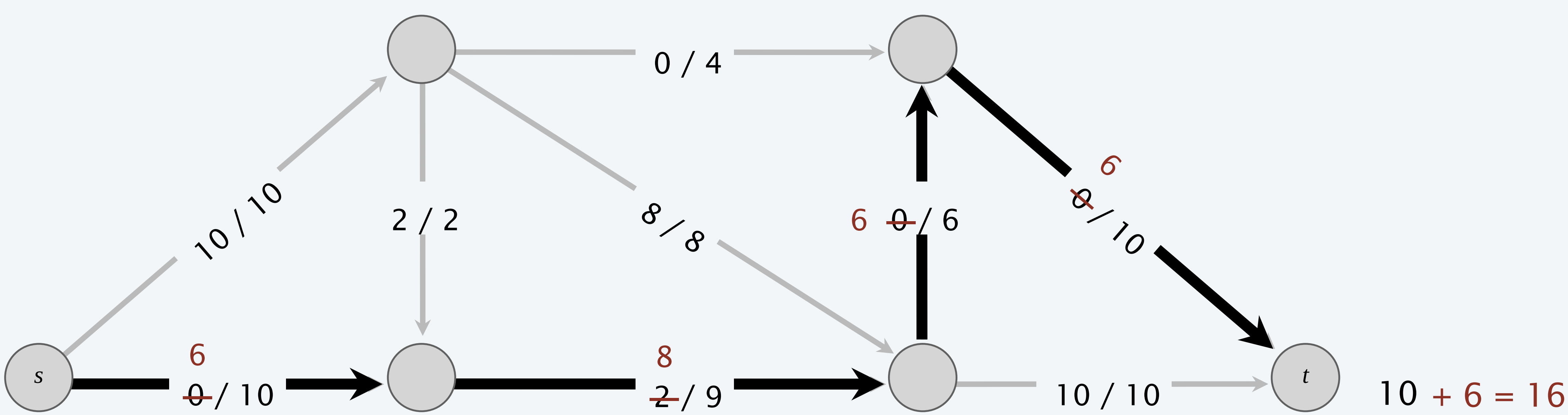


Toward a max-flow algorithm

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



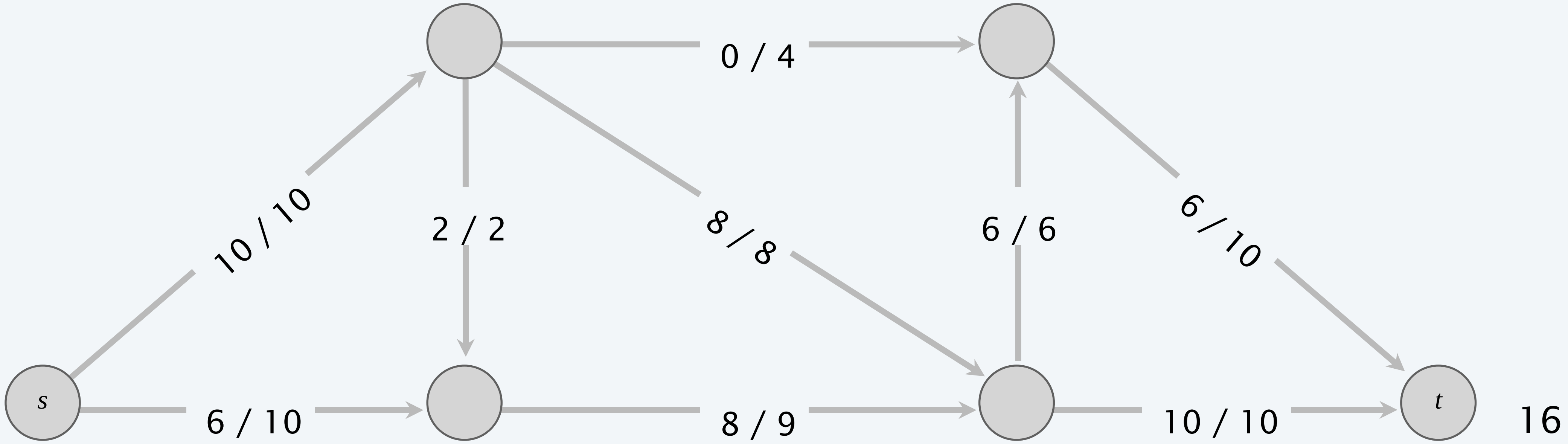
Toward a max-flow algorithm

Greedy algorithm.

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



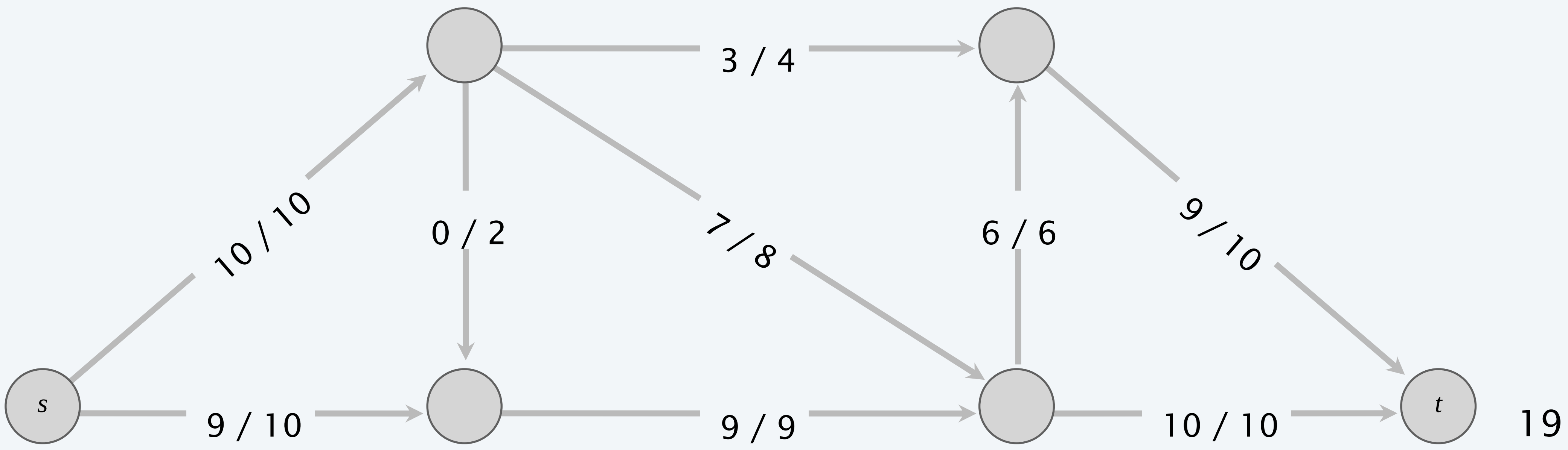
Toward a max-flow algorithm

Greedy algorithm.

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

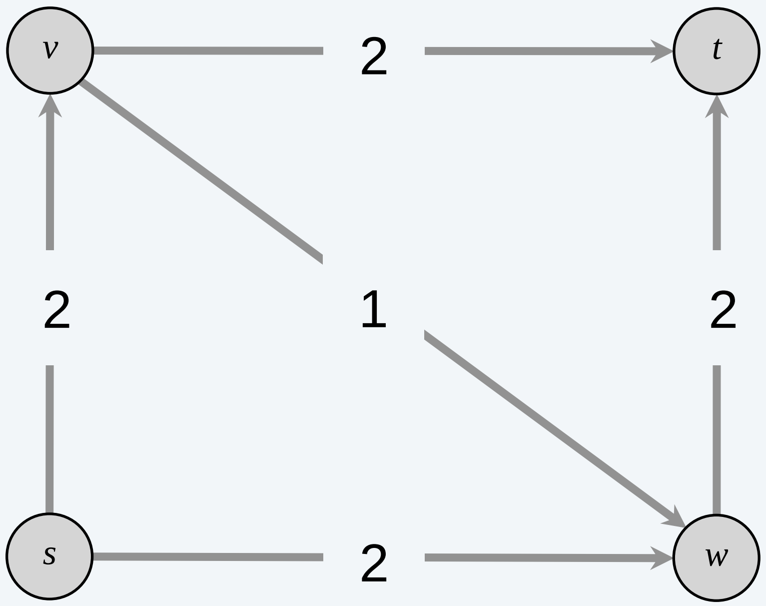
Q. Why does the greedy algorithm fail?

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

Ex. Consider flow network G.

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

flow network G



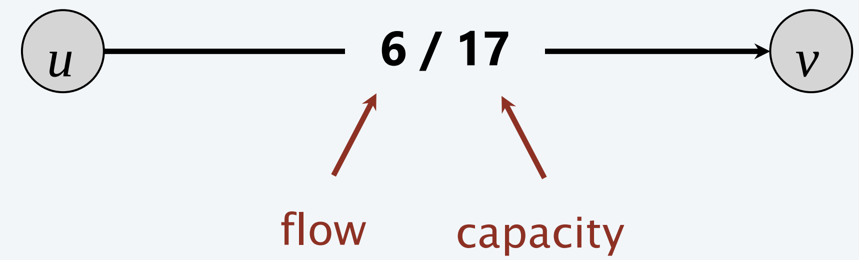
Bottom line. Need some mechanism to "undo" a bad decision.

Residual network

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

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

original flow network G



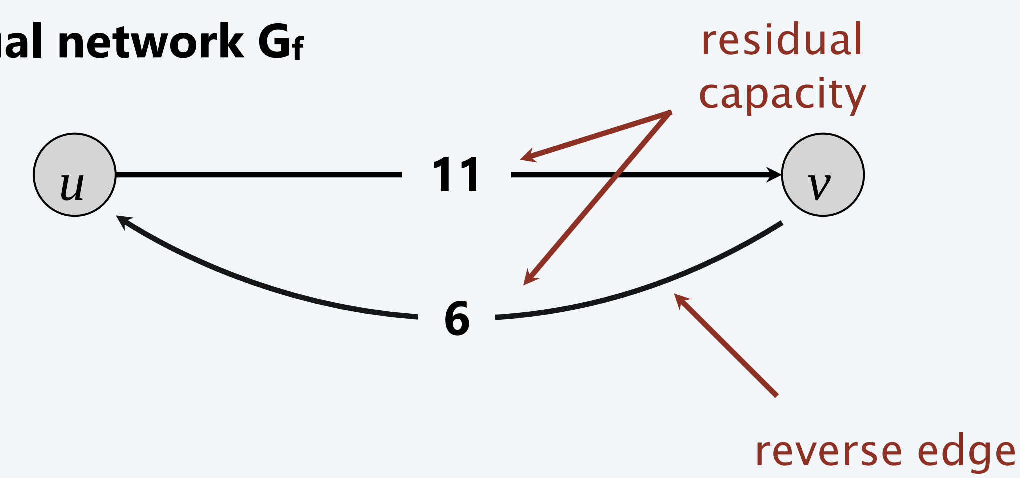
Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

Residual capacity.

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

residual network G_f



edges with positive residual capacity

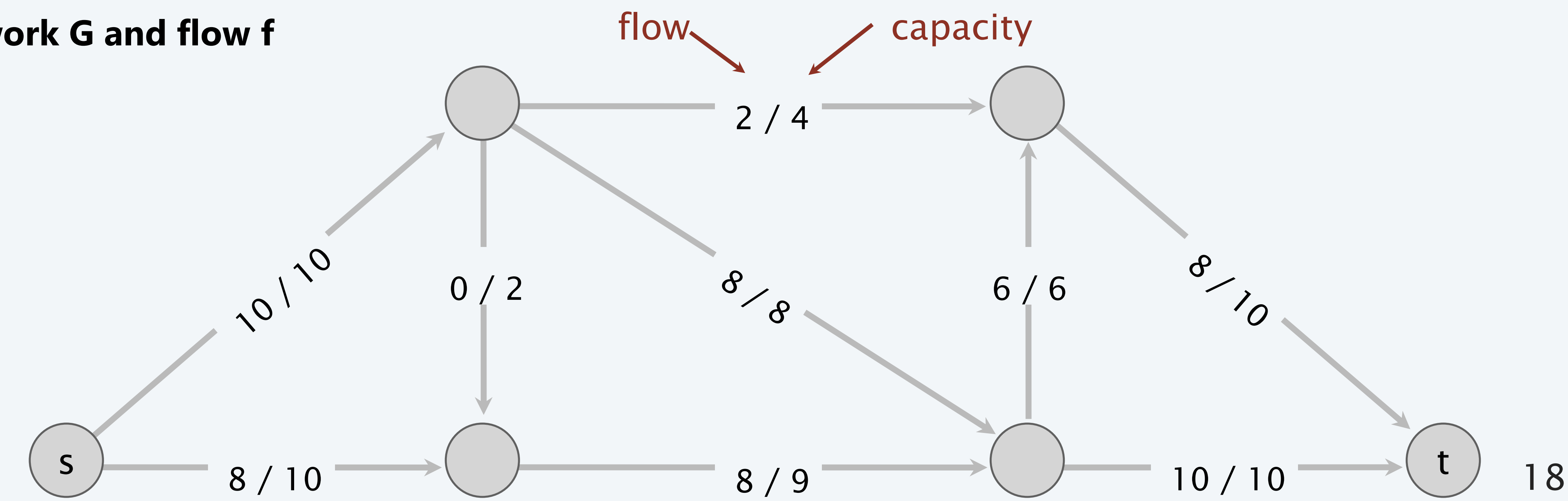
where flow on a reverse edge negates flow on corresponding forward edge

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

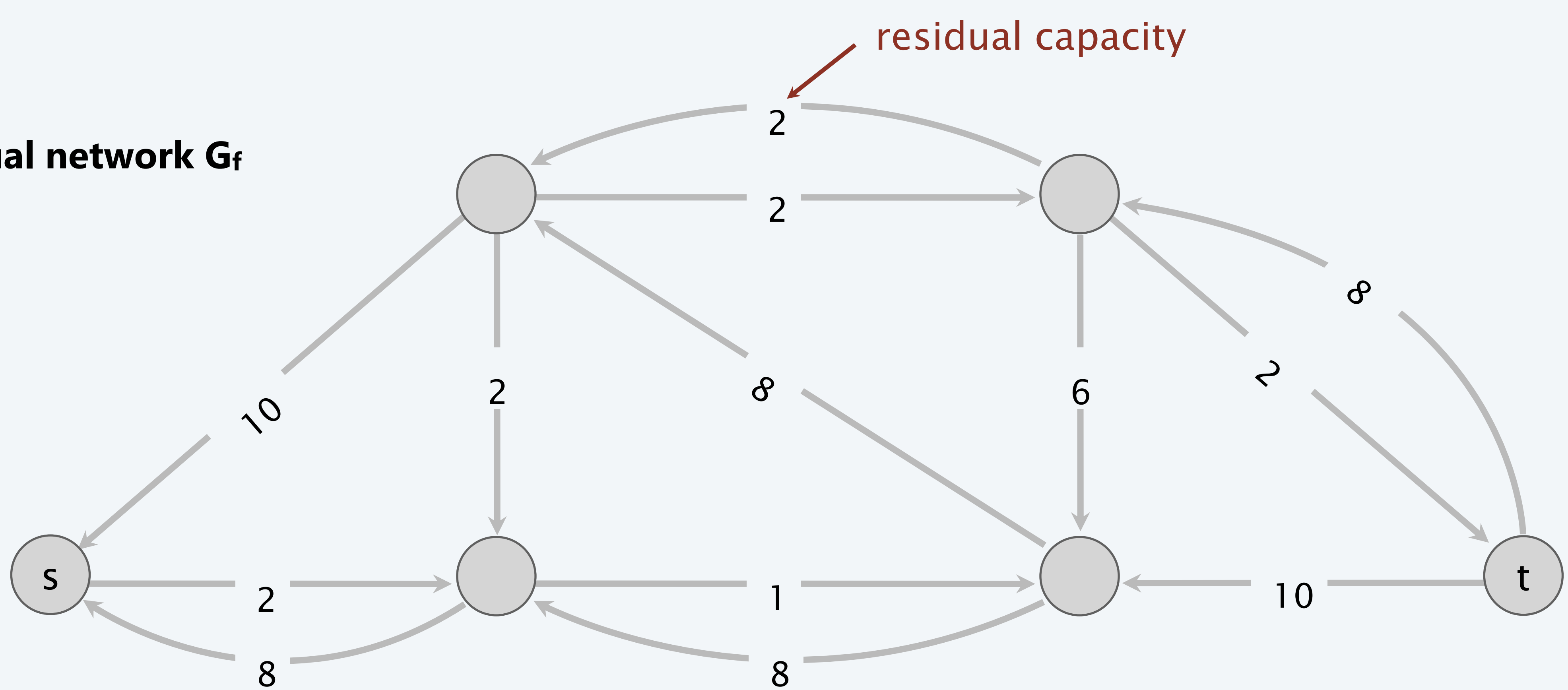
- $E_f = \{e : f(e) < c(e)\} \cup \{e : f(e^{\text{reverse}}) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

Residual network: an example

network G and flow f



residual network G_f



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.
```

Ford–Fulkerson algorithm

Ford–Fulkerson augmenting path algorithm.

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

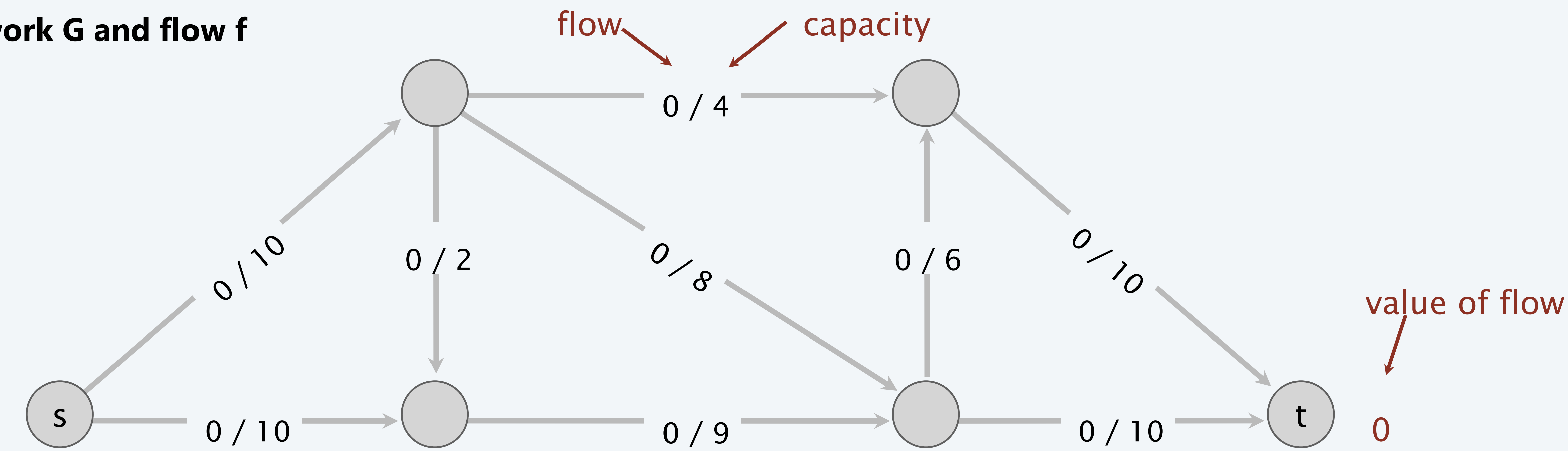
    Update G_f.

RETURN f.

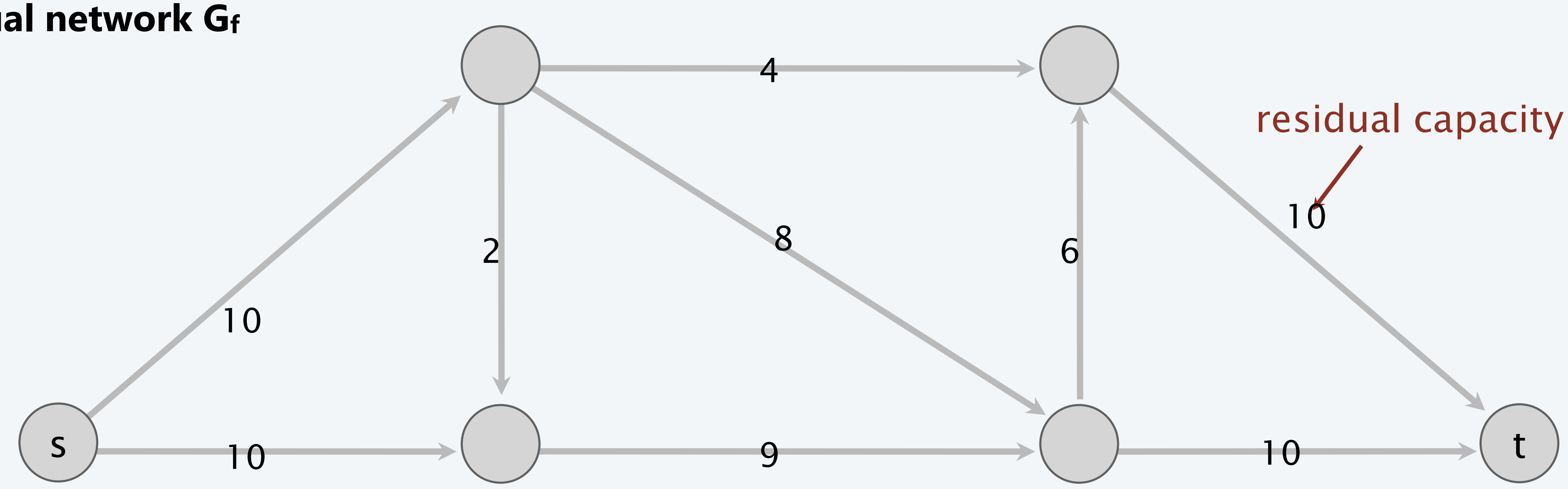
augmenting path

Ford–Fulkerson algorithm demo

network G and flow f

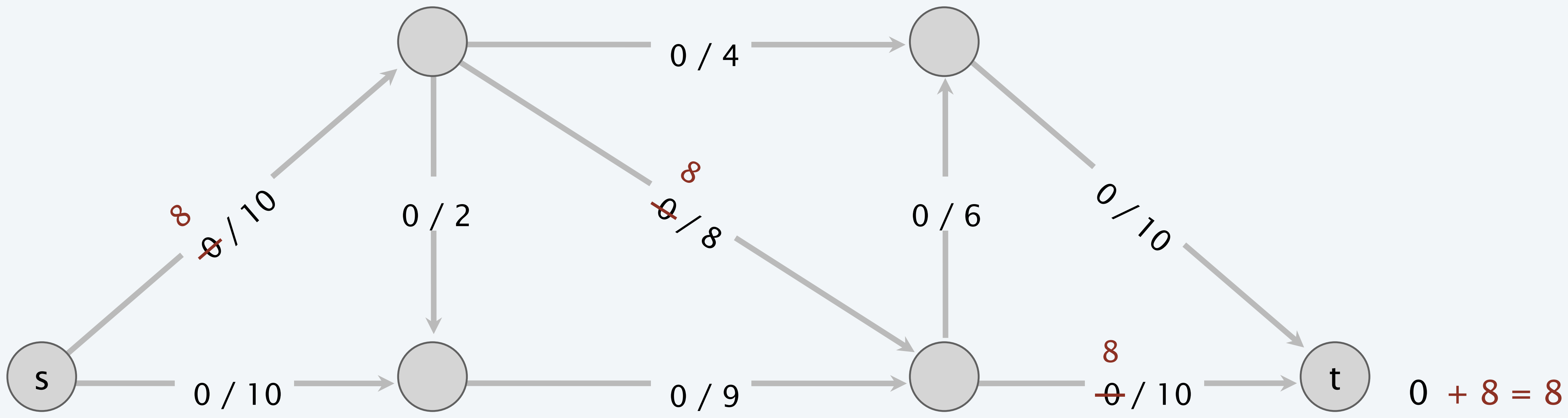


residual network G_f

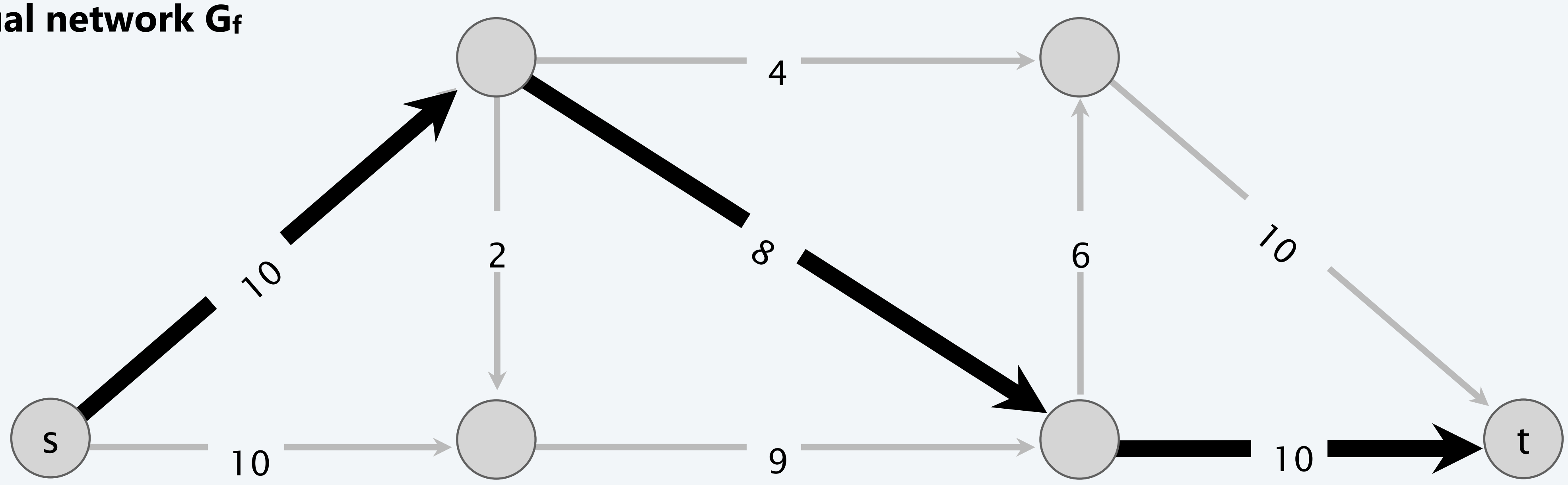


Ford–Fulkerson algorithm demo

network G and flow f

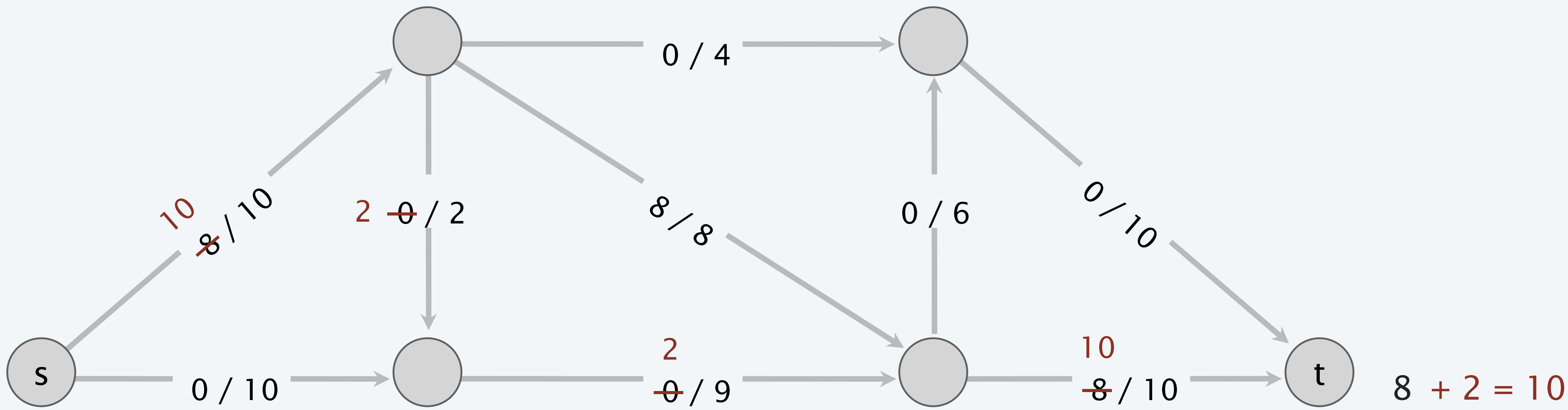


residual network G_f

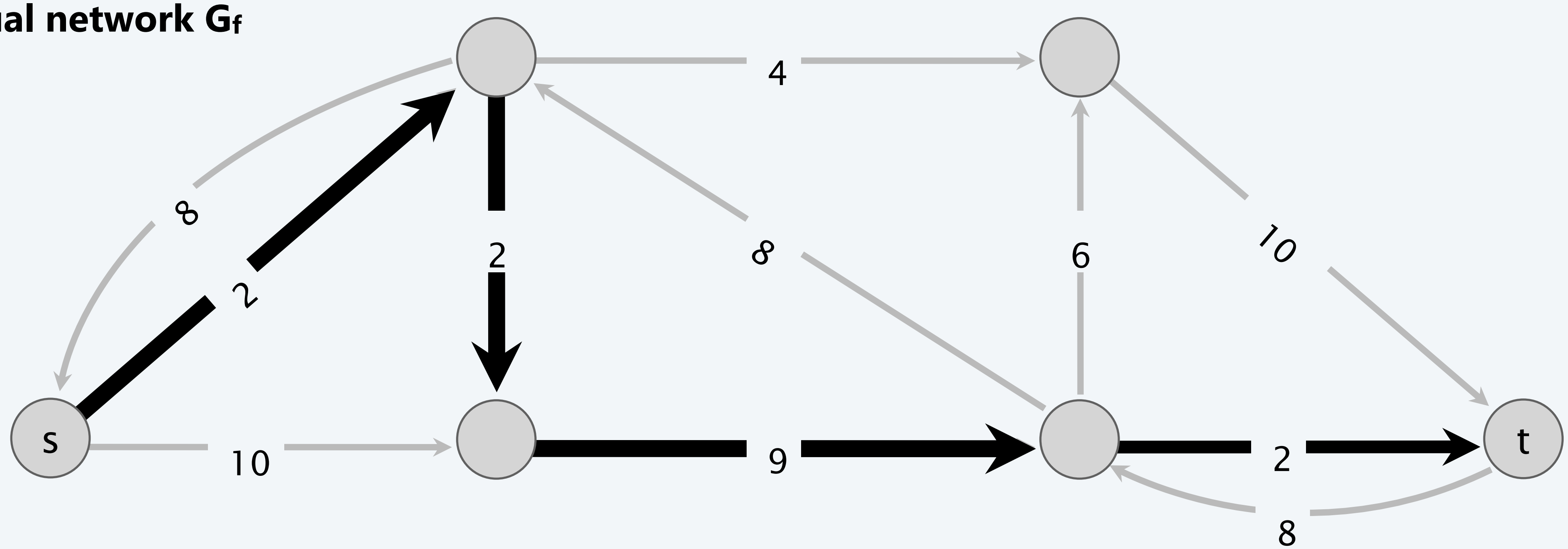


Ford–Fulkerson algorithm demo

network G and flow f

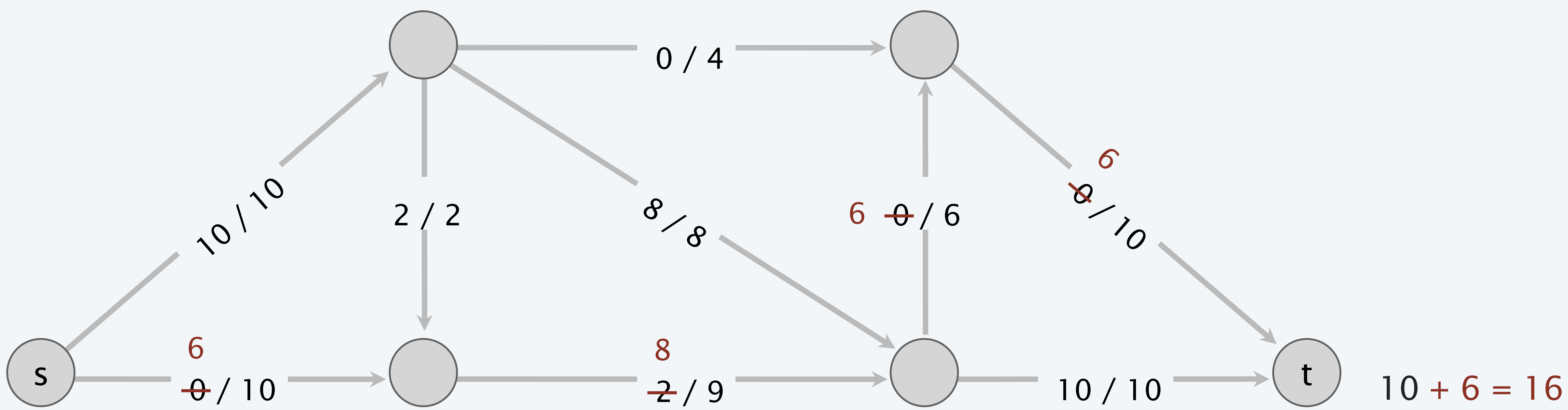


residual network G_f

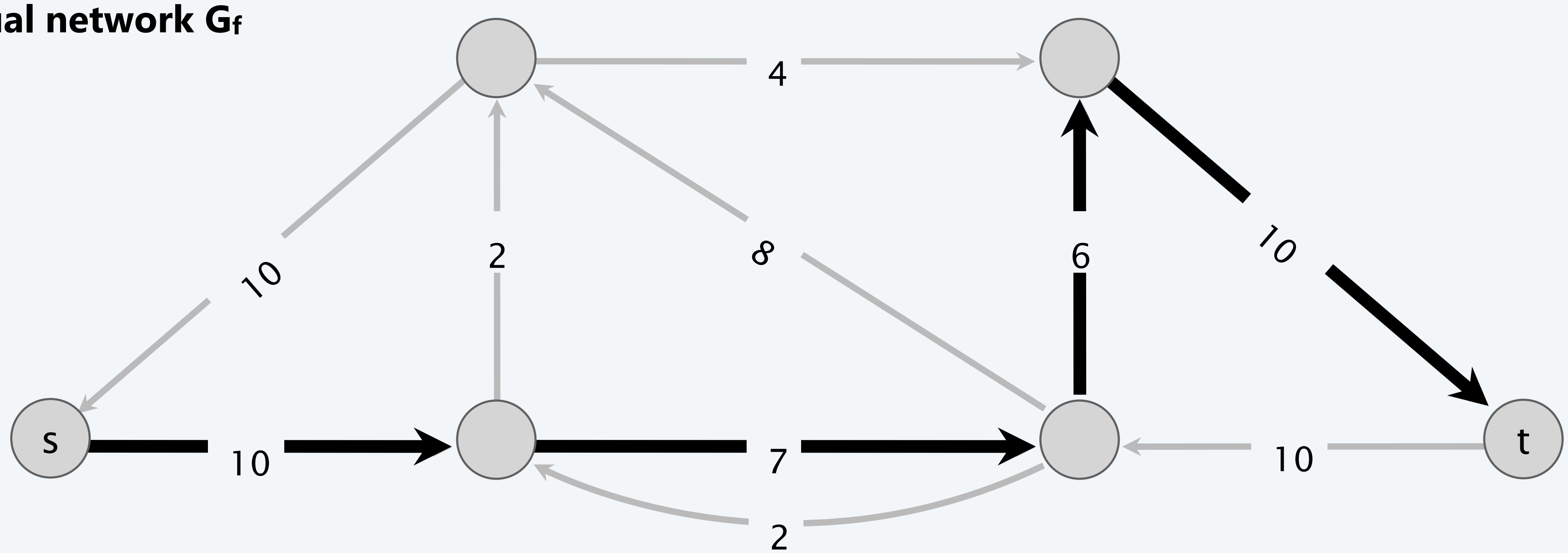


Ford–Fulkerson algorithm demo

network G and flow f

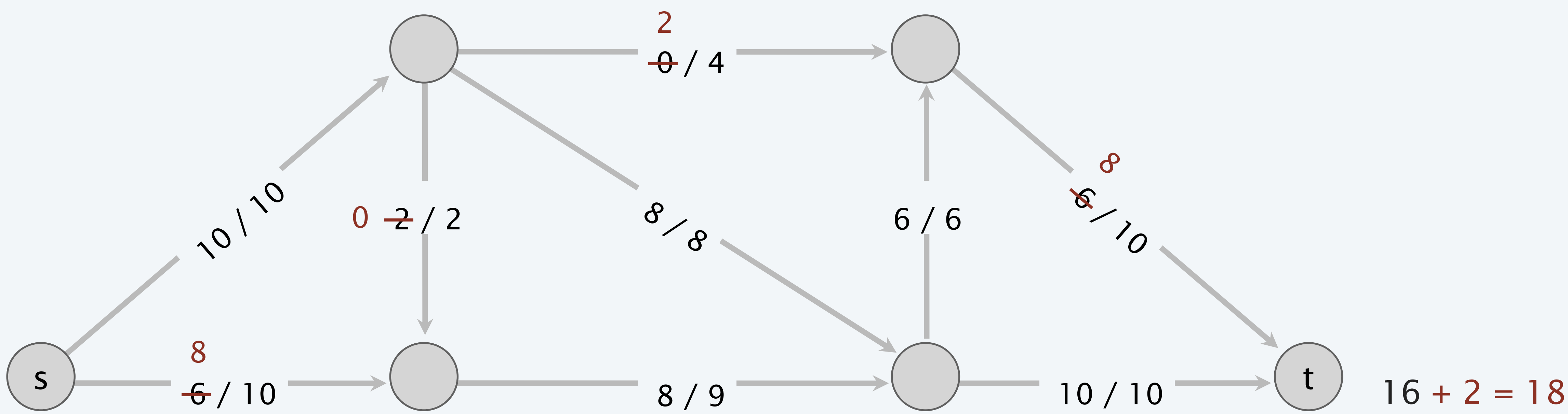


residual network G_f

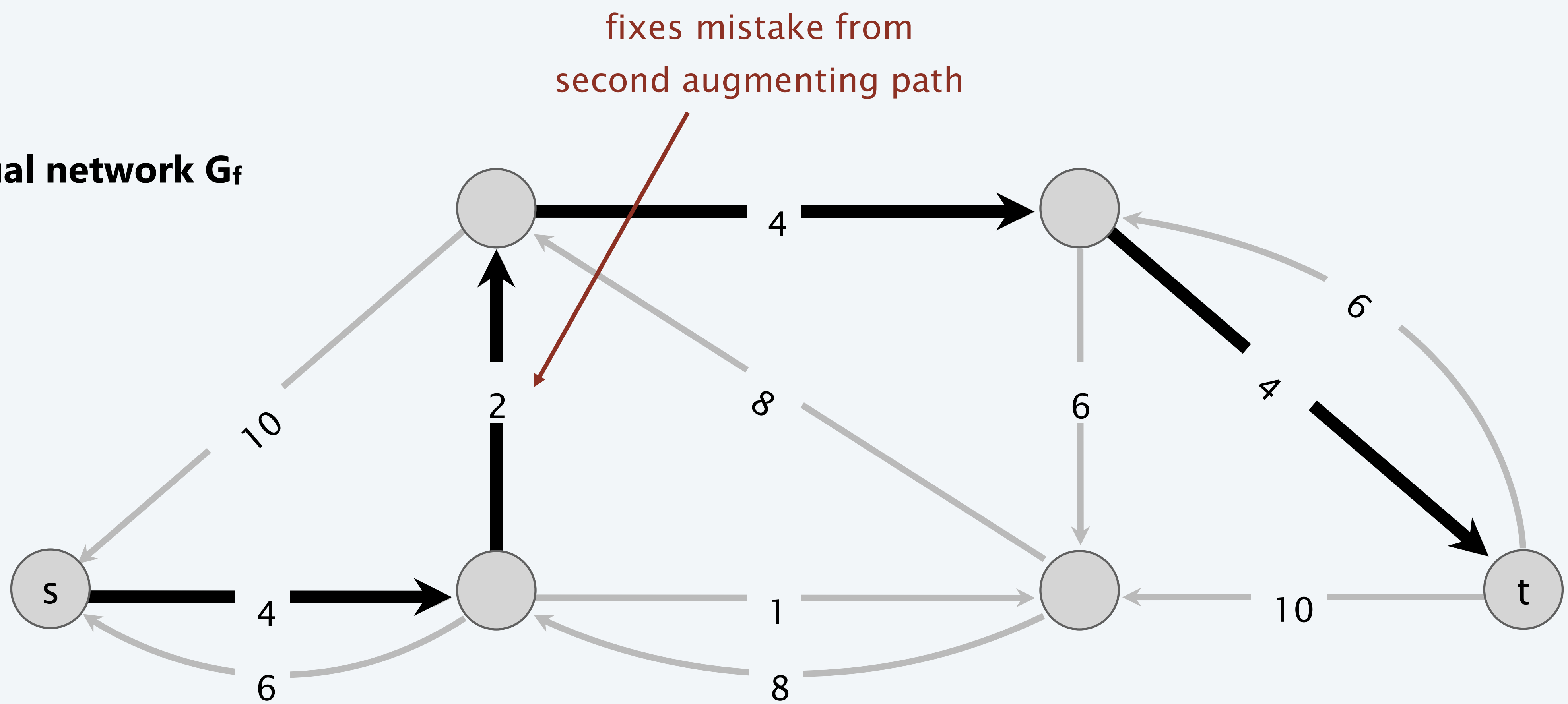


Ford–Fulkerson algorithm demo

network G and flow f

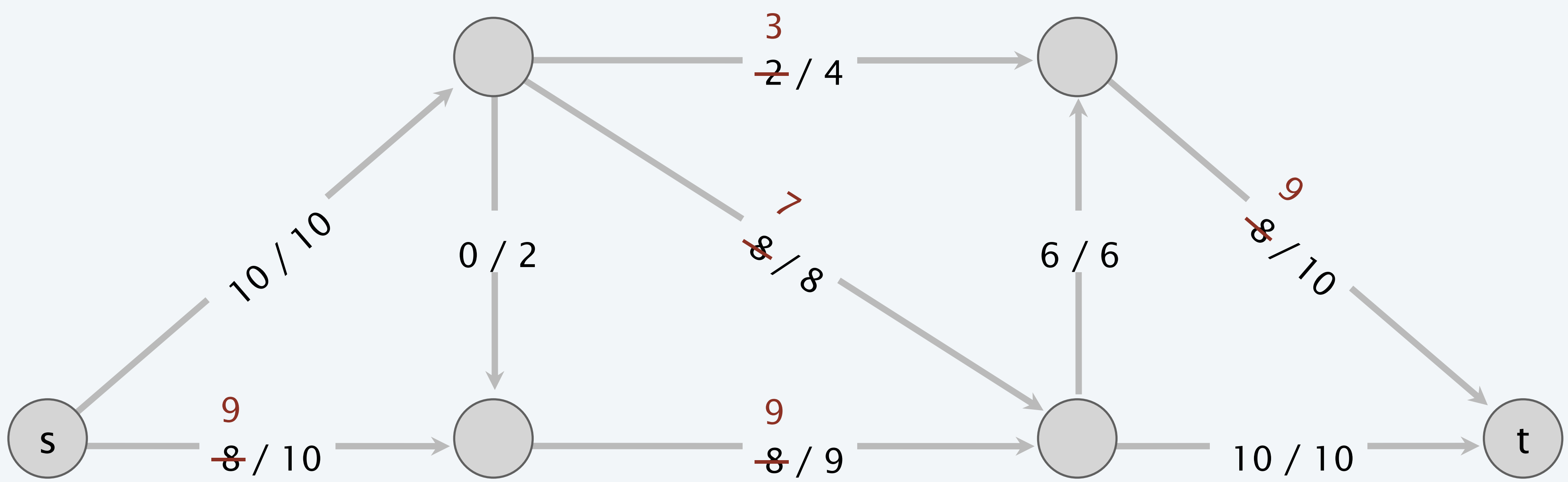


residual network Gf



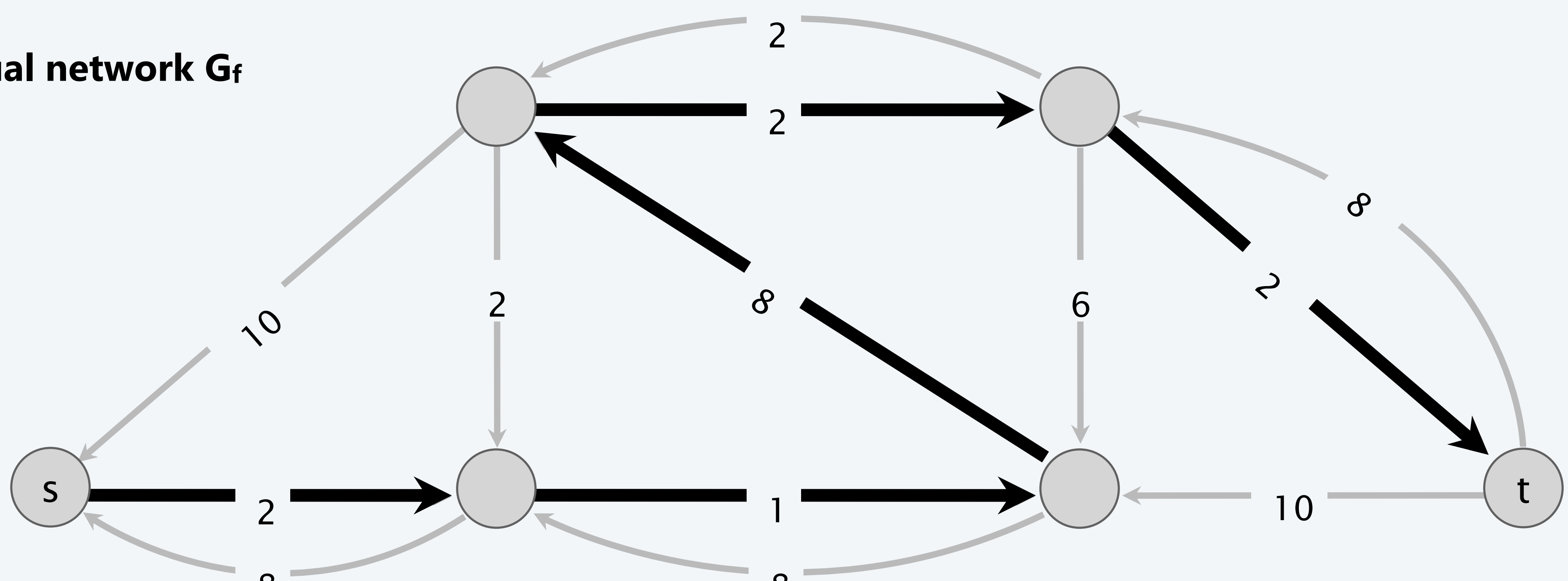
Ford–Fulkerson algorithm demo

network G and flow f



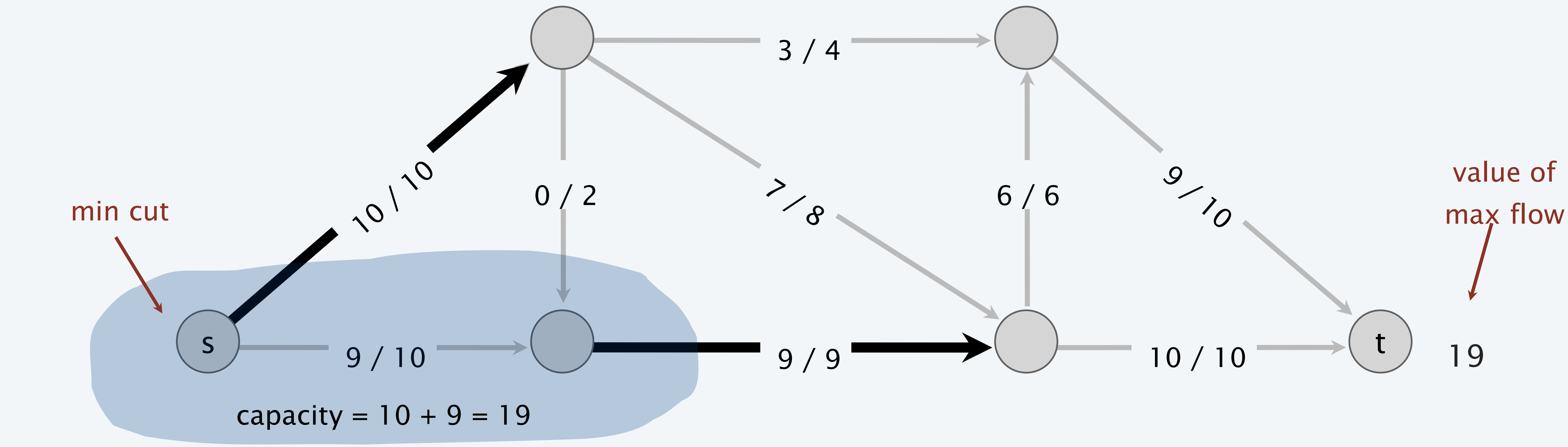
18 + 1 = 19

residual network G_f



Ford–Fulkerson algorithm demo

network G and flow f



residual network G_f