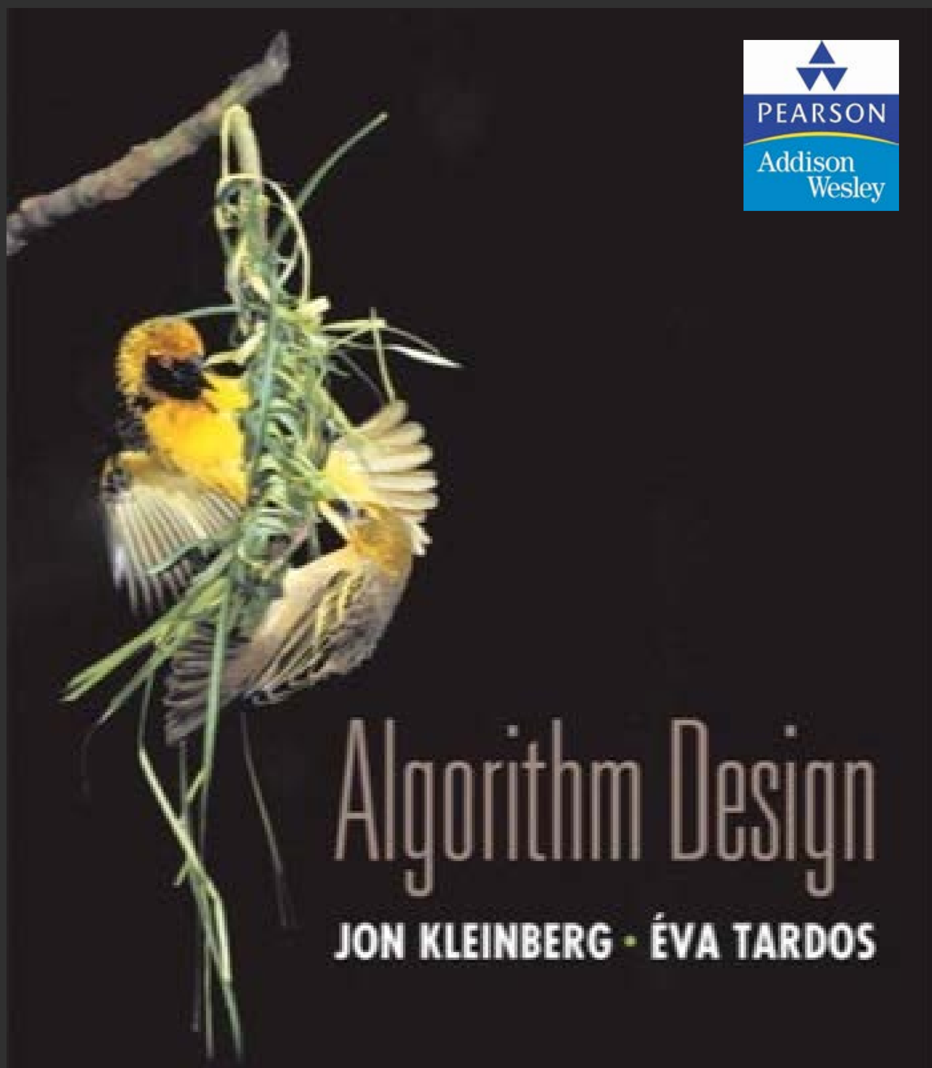
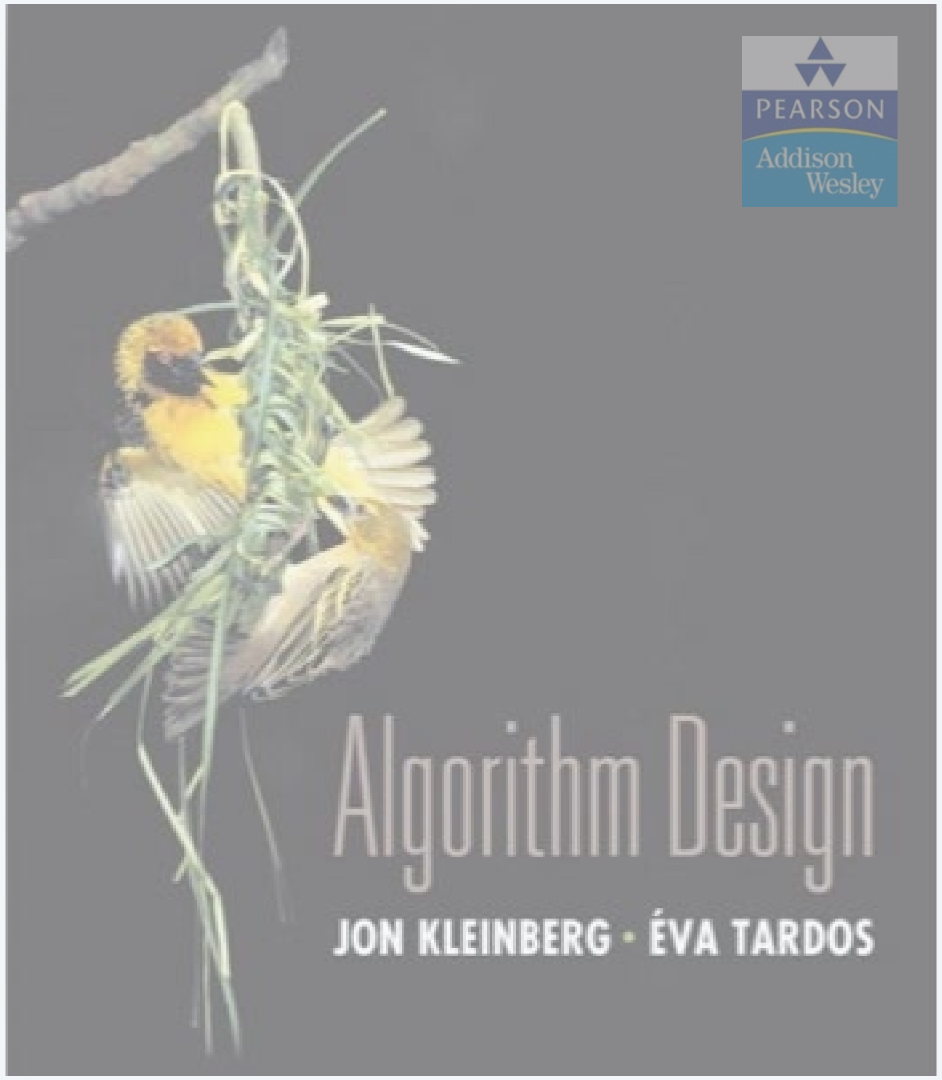


Lecture slides by Kevin Wayne

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

7. Network Flow I

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



7. Network Flow I

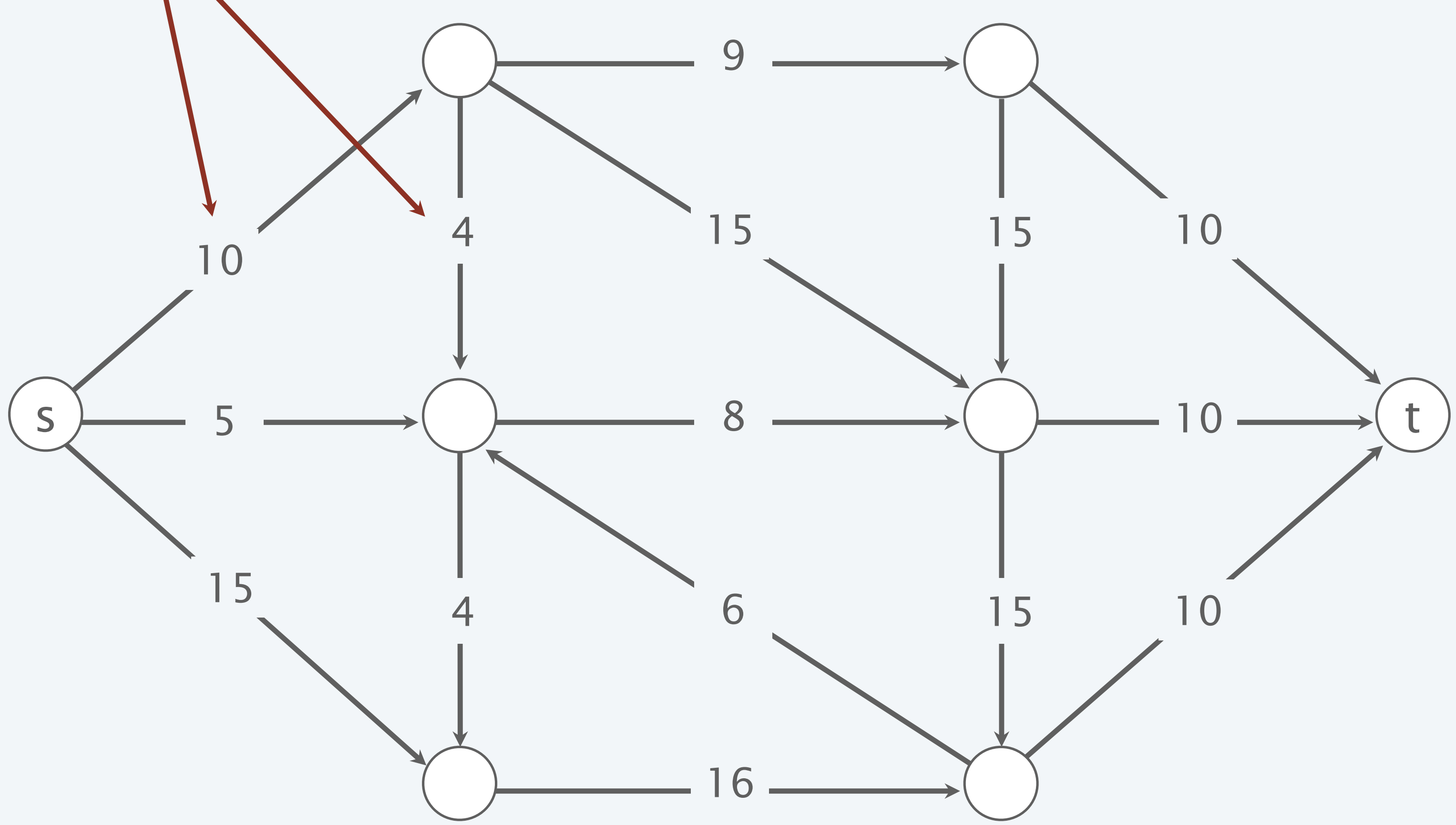
- *max-flow and min-cut problems*
- *Ford-Fulkerson algorithm*
- *max-flow min-cut theorem*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *blocking-flow algorithm*
- *unit-capacity simple networks*

Flow network

- Abstraction for material flowing through the edges.
- Digraph $G = (V, E)$ with source $s \in V$ and sink $t \in V$.
- Nonnegative integer capacity $c(e)$ for each $e \in E$.

no parallel edges
no edge enters s
no edge leaves t

capacity

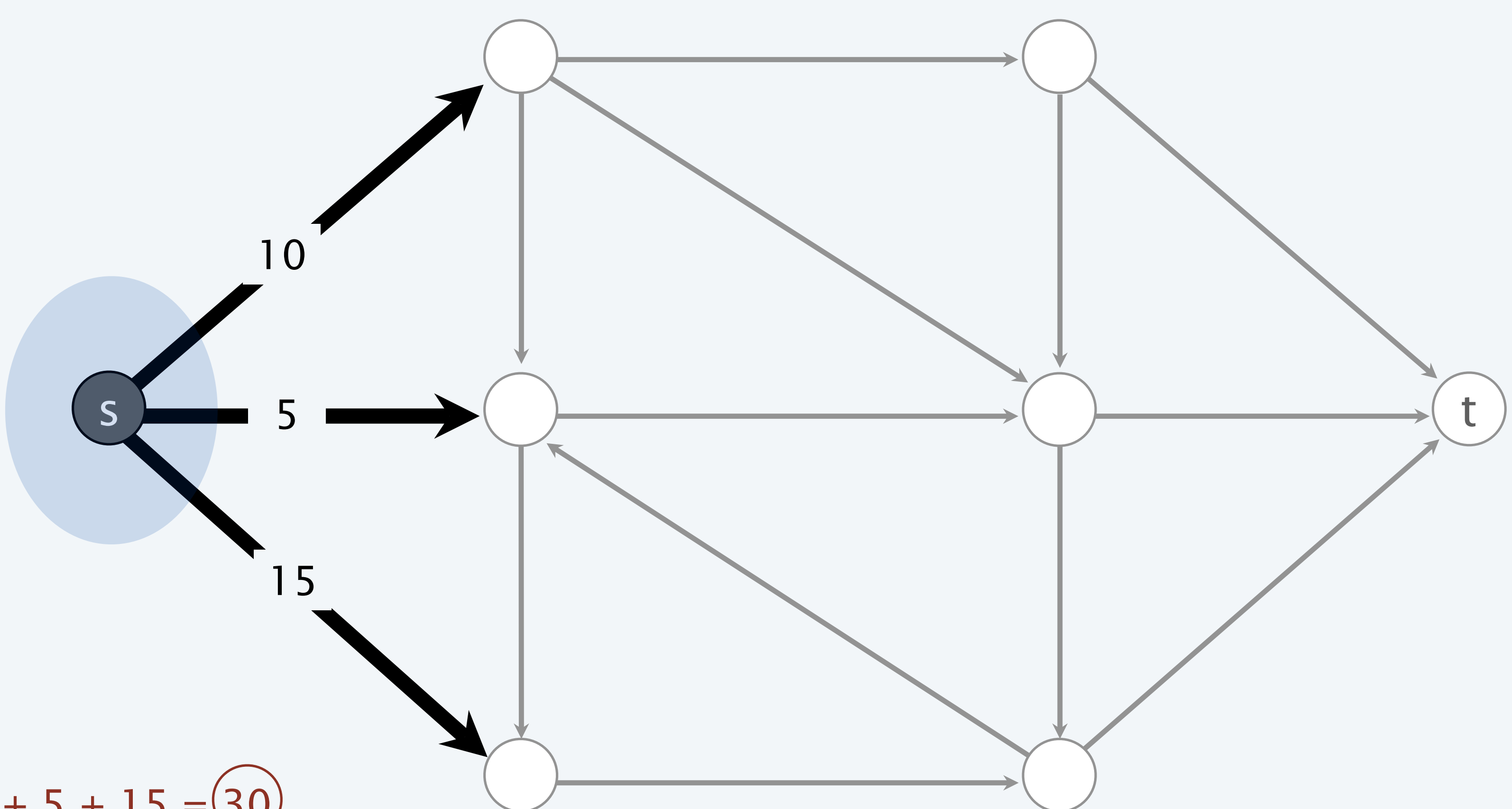


Minimum cut problem

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

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

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



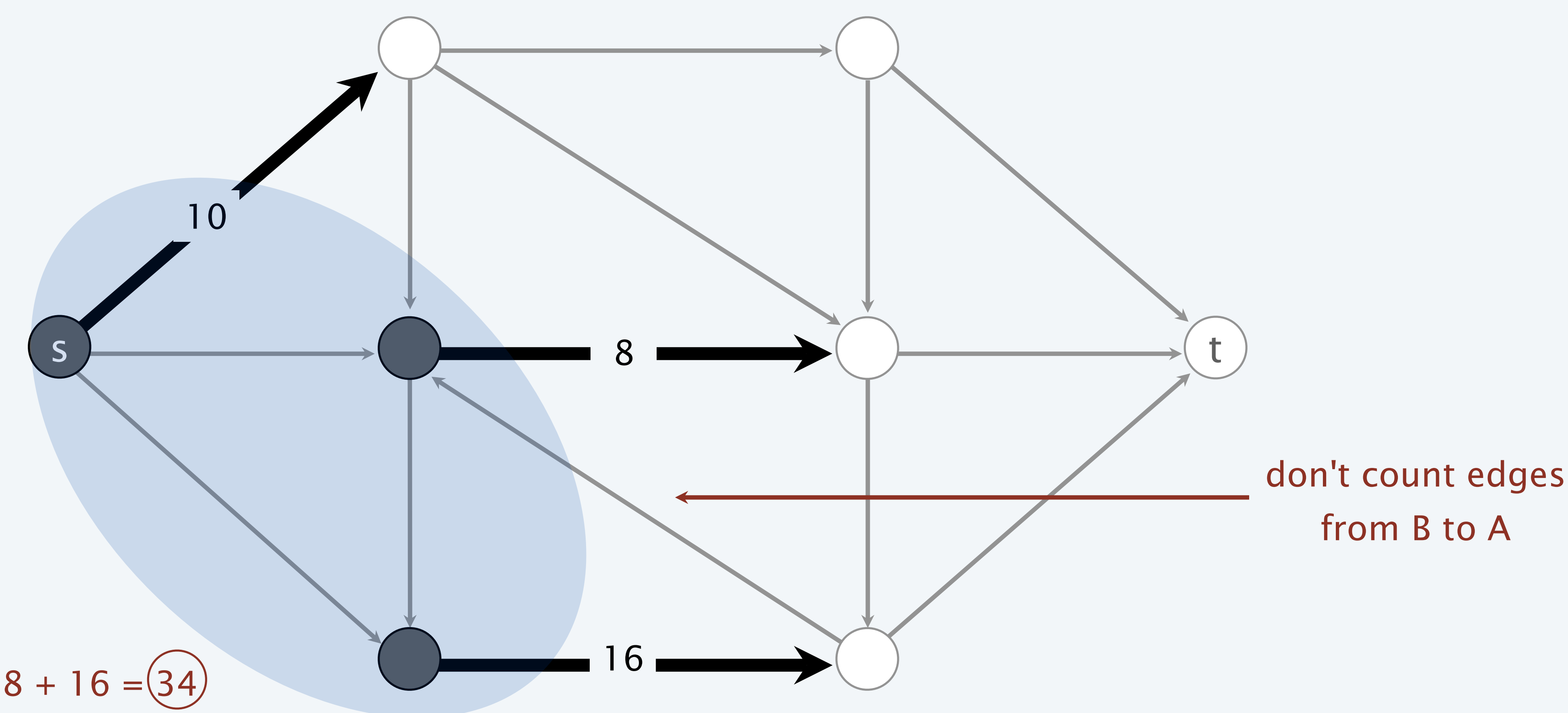
capacity = 10 + 5 + 15 = 30

Minimum cut problem

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

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

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



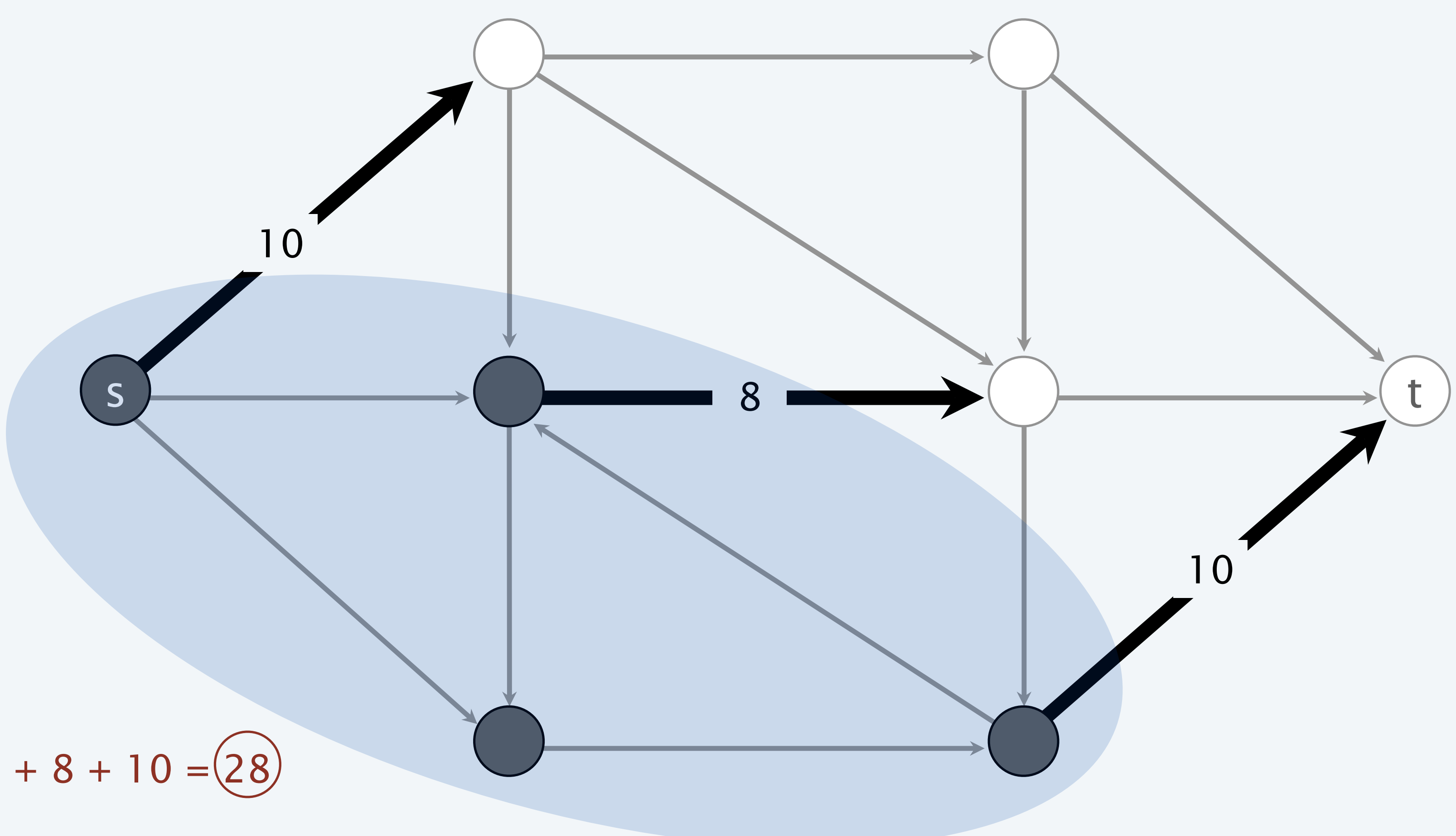
capacity = 10 + 8 + 16 = 34

Def. A *st*-cut (cut) is a partition (A, B) of the vertices 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.

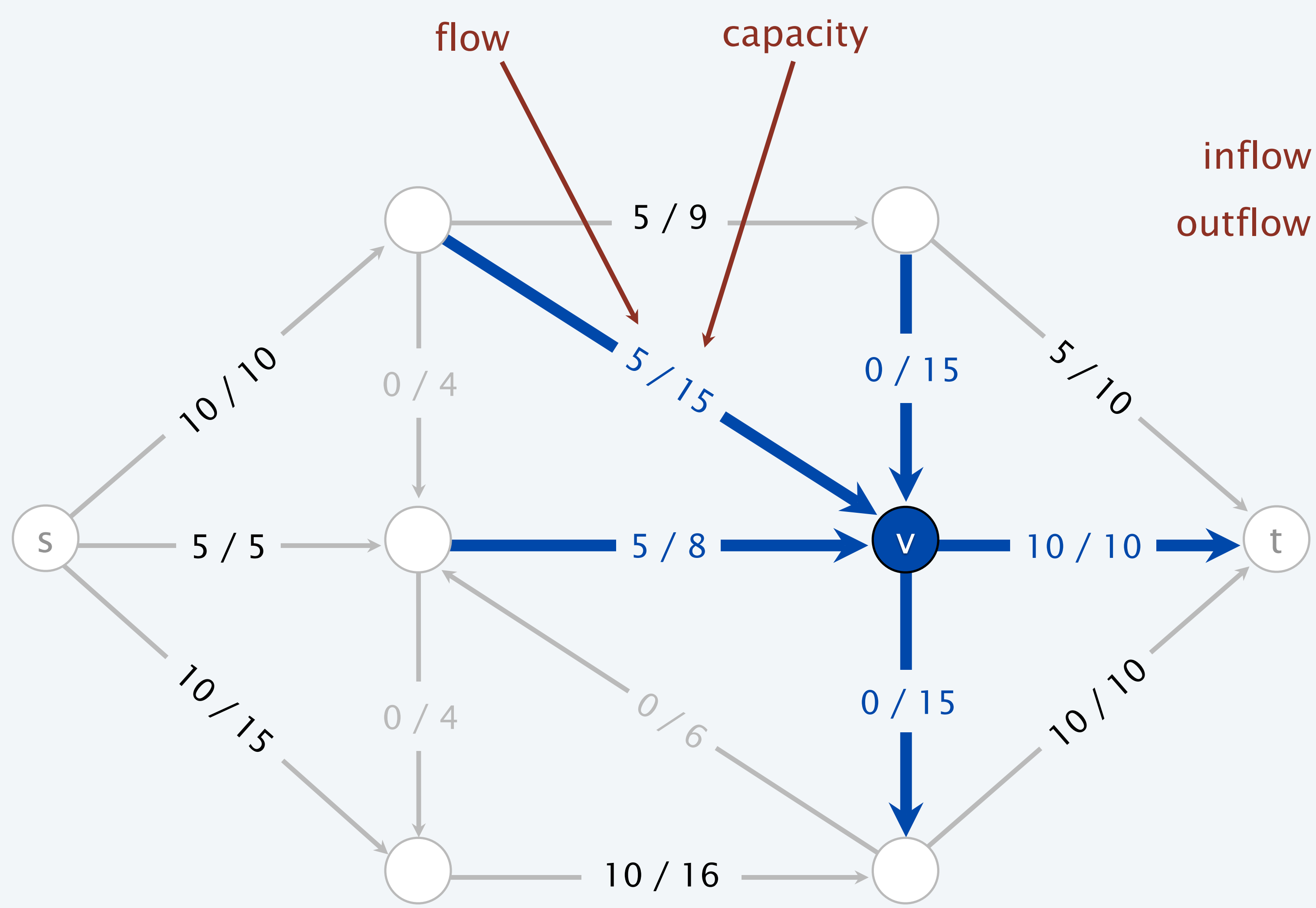


capacity = 10 + 8 + 10 = 28

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



inflow at v = 5 + 5 + 0 = 10

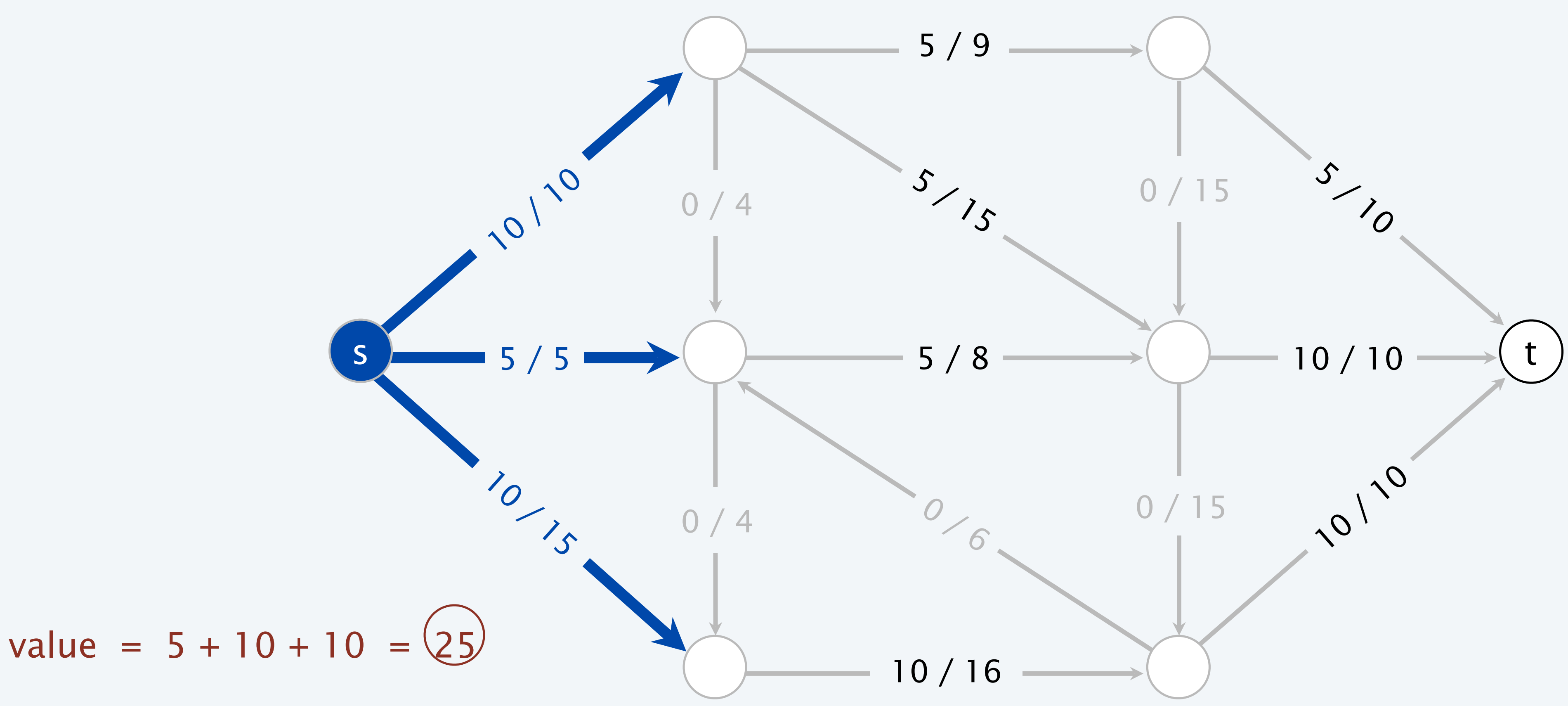
outflow at v = 10 + 0 = 10

Maximum flow problem

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

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [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: $val(f) = \sum_{e \text{ out of } s} f(e)$.



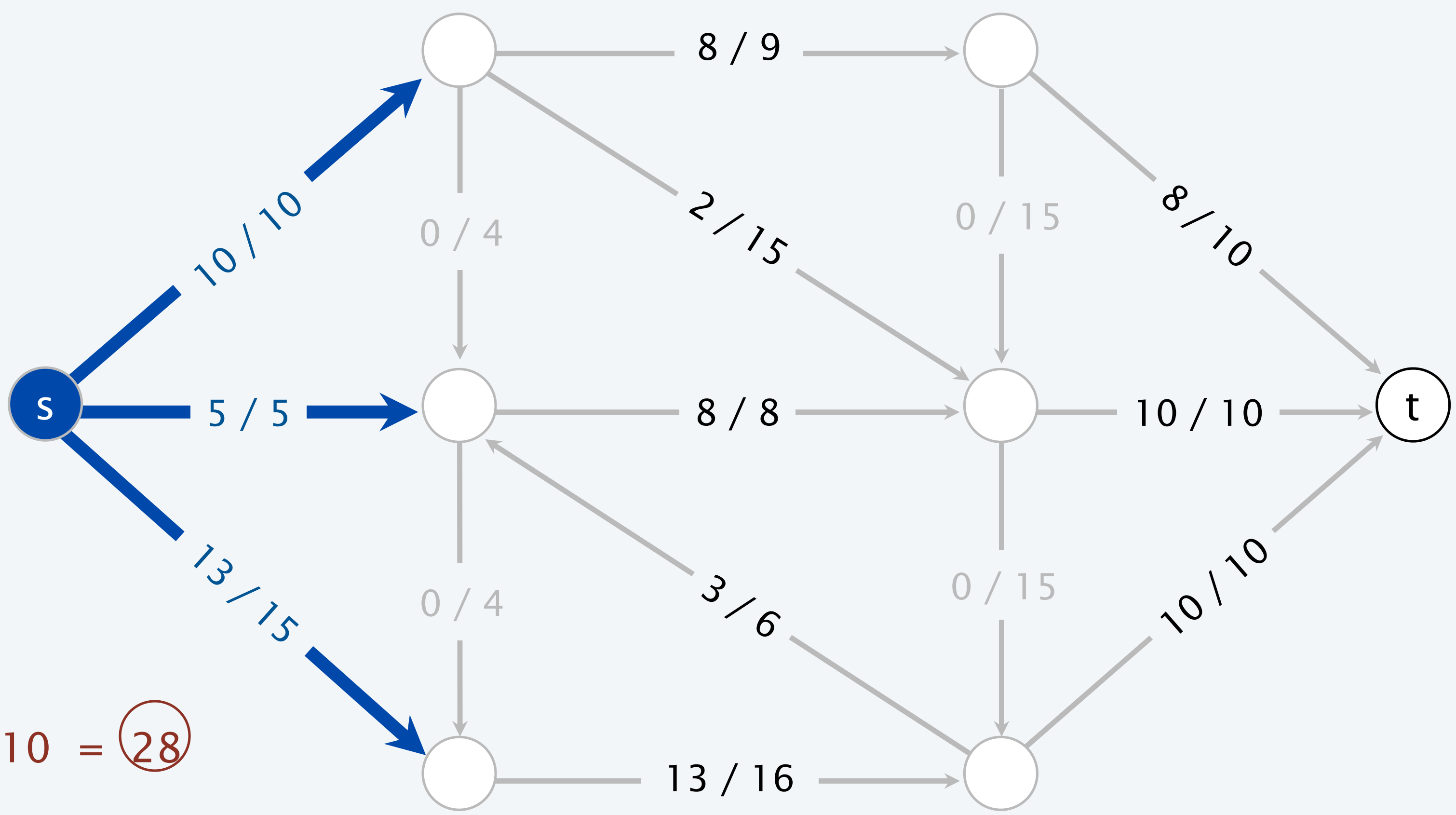
Maximum flow problem

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

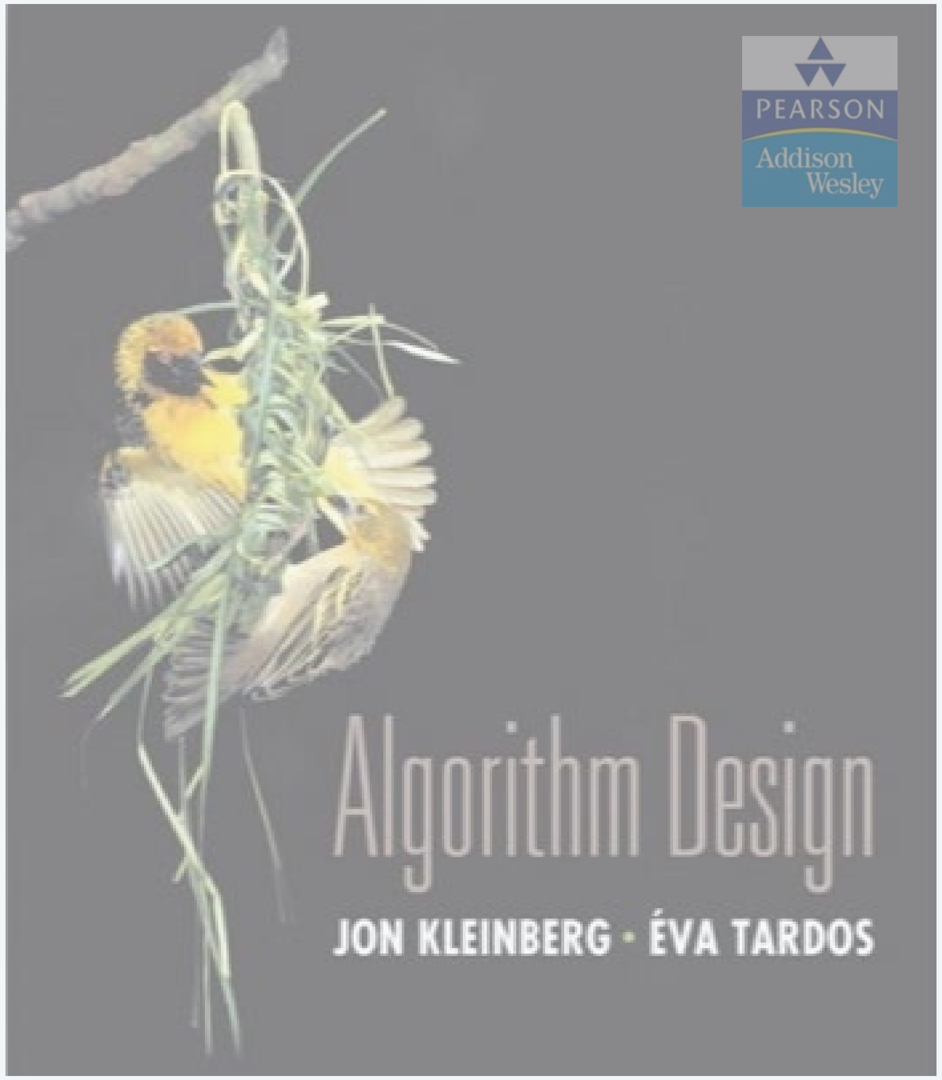
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

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

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



value = 8 + 10 + 10 = 28

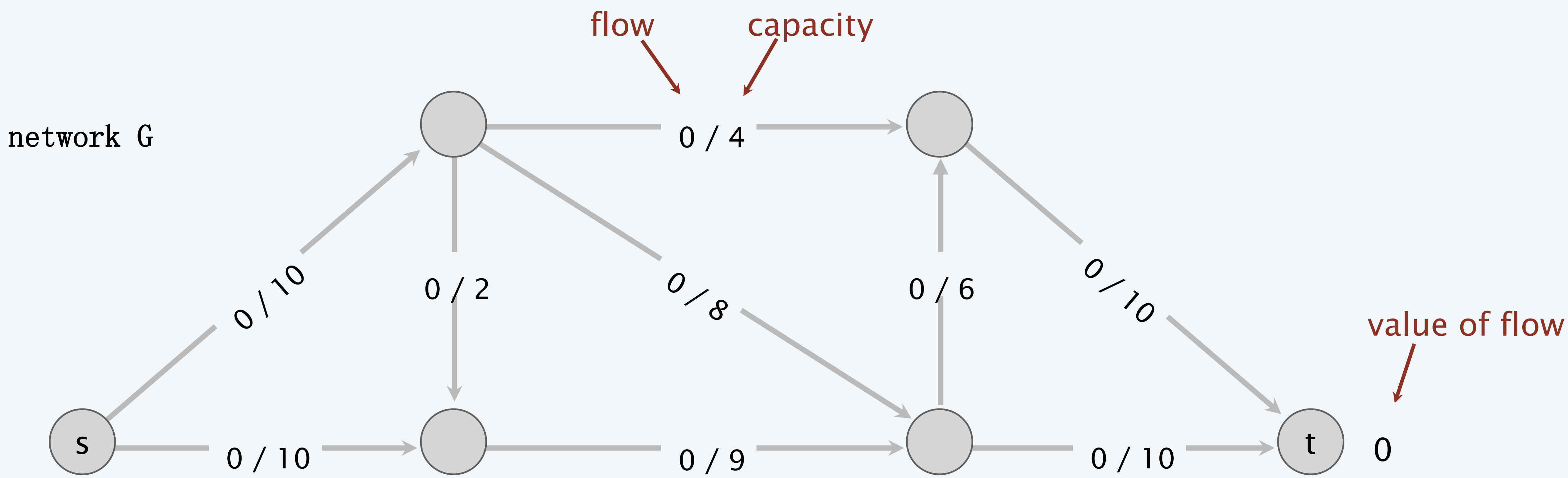


7. NETWORK FLOW I

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

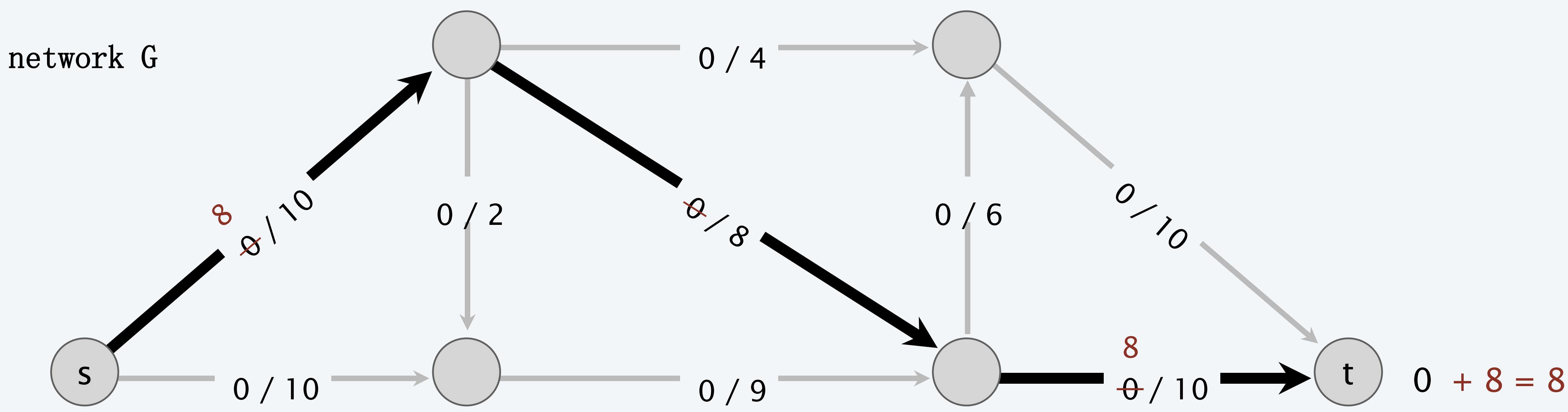
Greedy algorithm.

- Start with $f(e) = 0$ for all 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.



Greedy algorithm.

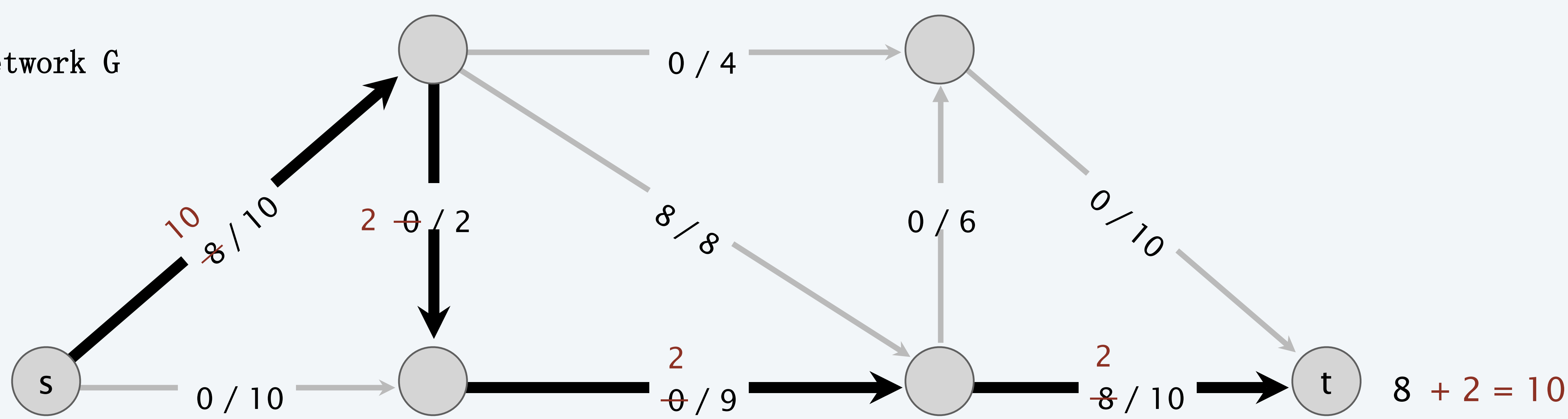
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Greedy algorithm.

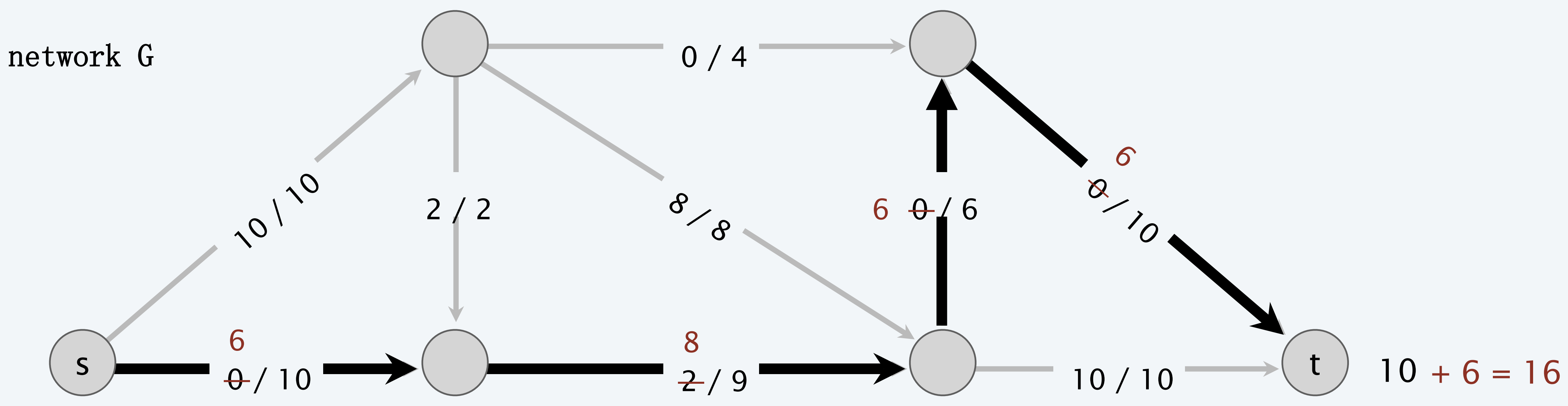
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

network G



Greedy algorithm.

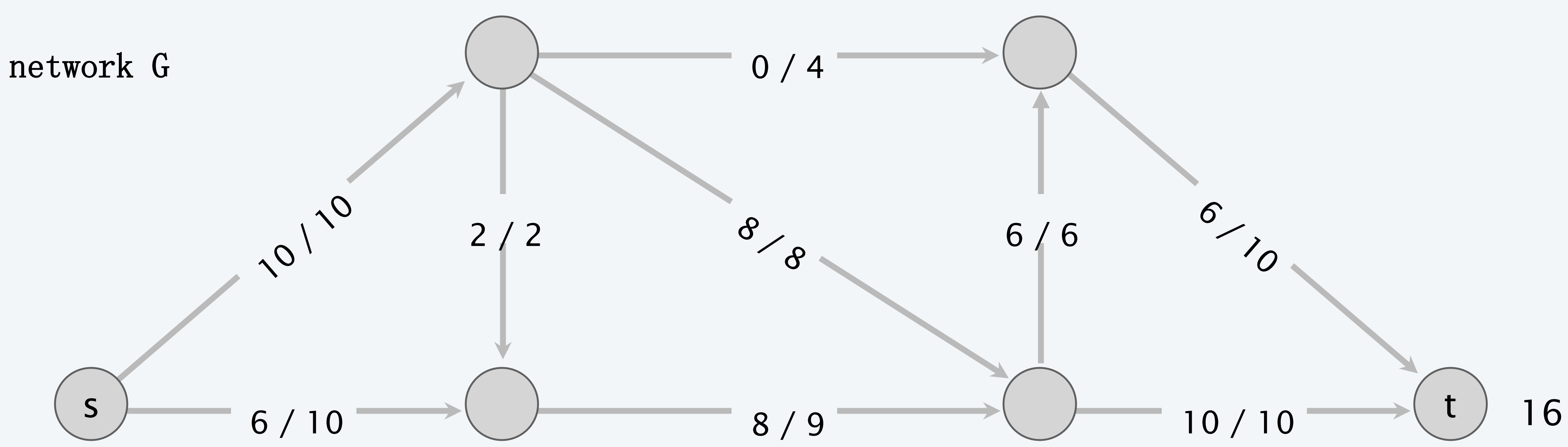
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \leadsto t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Greedy algorithm.

- Start with $f(e) = 0$ for all 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.

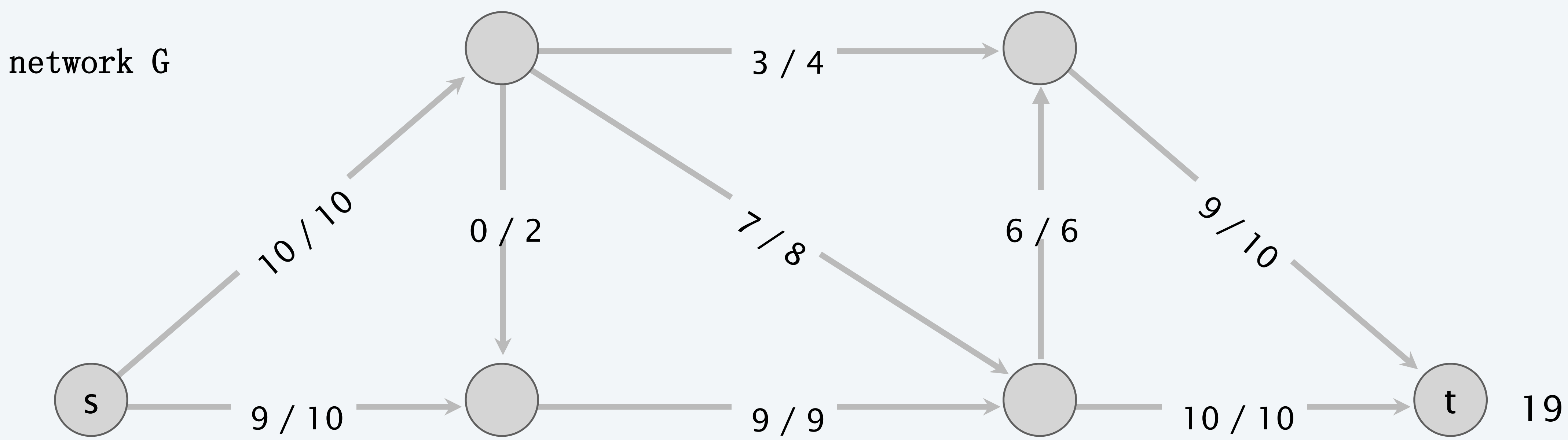
ending flow value = 16



Greedy algorithm.

- Start with $f(e) = 0$ for all 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.

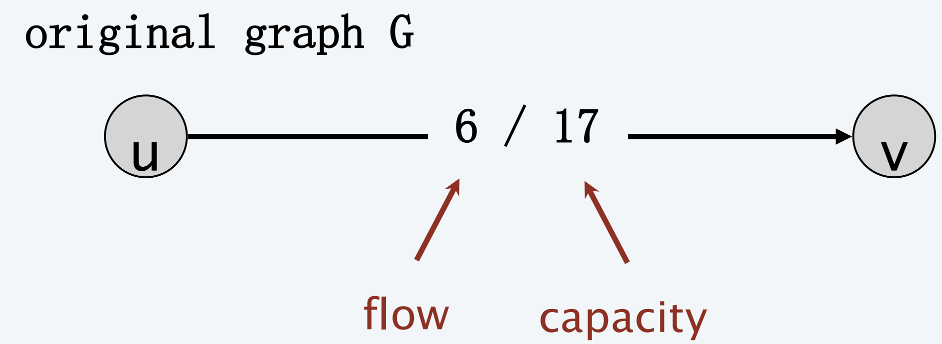
but max-flow value = 19



Residual graph

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

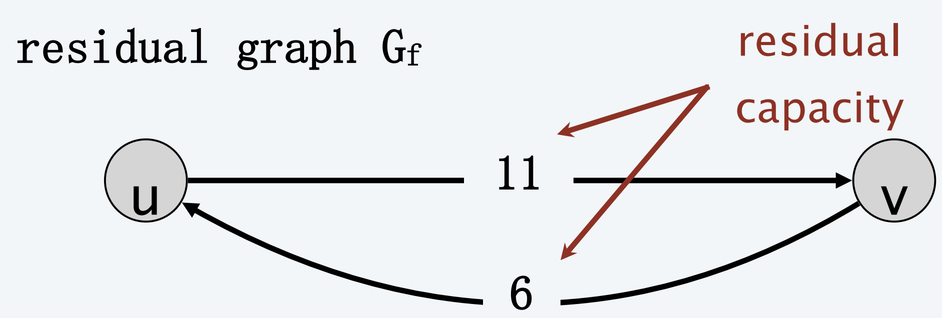
- Flow $f(e)$.
- Capacity $c(e)$.



Residual edge.

- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

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



Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

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

Def. An augmenting path is a simple $s \rightsquigarrow t$ path P in the residual graph G_f.

Def. The bottleneck capacity of an augmenting 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 f' is a flow and $val(f') = val(f) + bottleneck(G_f, P)$.

```
AUGMENT (f, c, P)
-----------------------------------------
b ← bottleneck capacity of path P.
FOREACH edge e ∈ P
   IF (e ∈ E ) f(e) ← f(e) + b.
   ELSE        f(e^R) ← f(e^R) – b.
RETURN f.
-----------------------------------------
```

Ford-Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an augmenting path P in the residual graph G_f .
- Augment flow along path P.
- Repeat until you get stuck.

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

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

$G_f \leftarrow$ residual graph.

WHILE (there exists an augmenting path P in G_f)

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

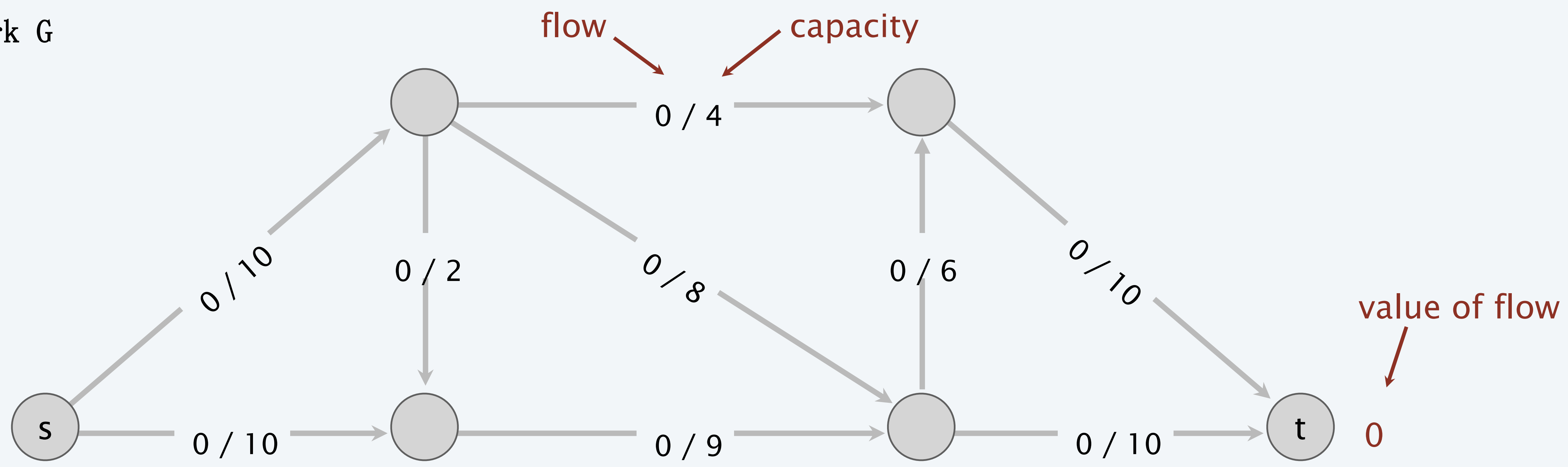
Update G_f.

RETURN f.

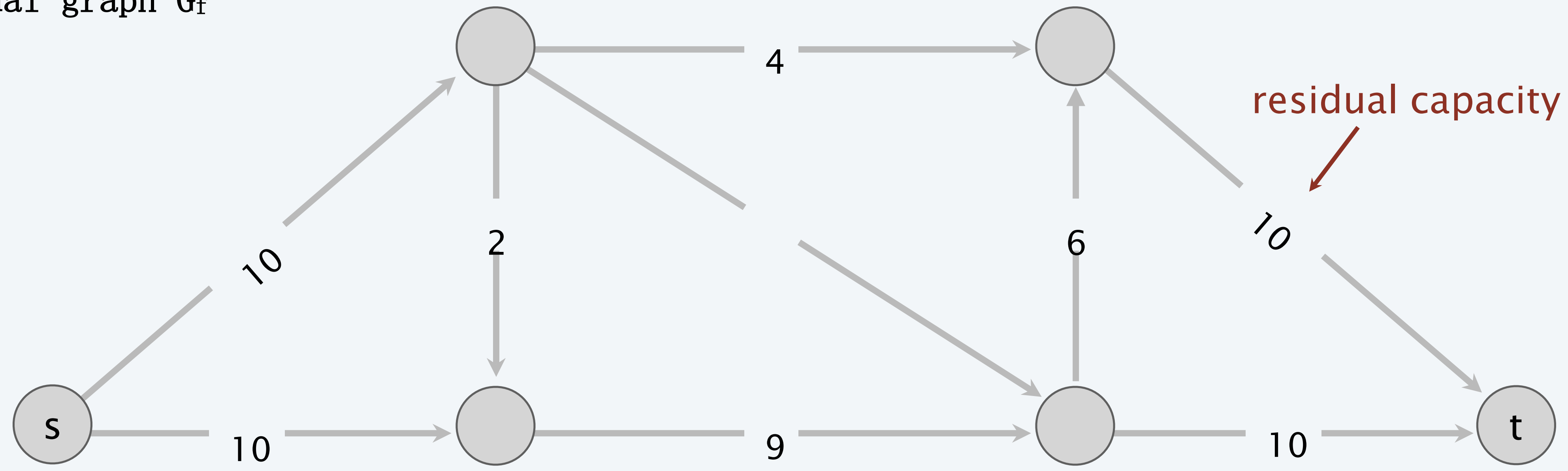
}

Ford-Fulkerson algorithm demo

network G

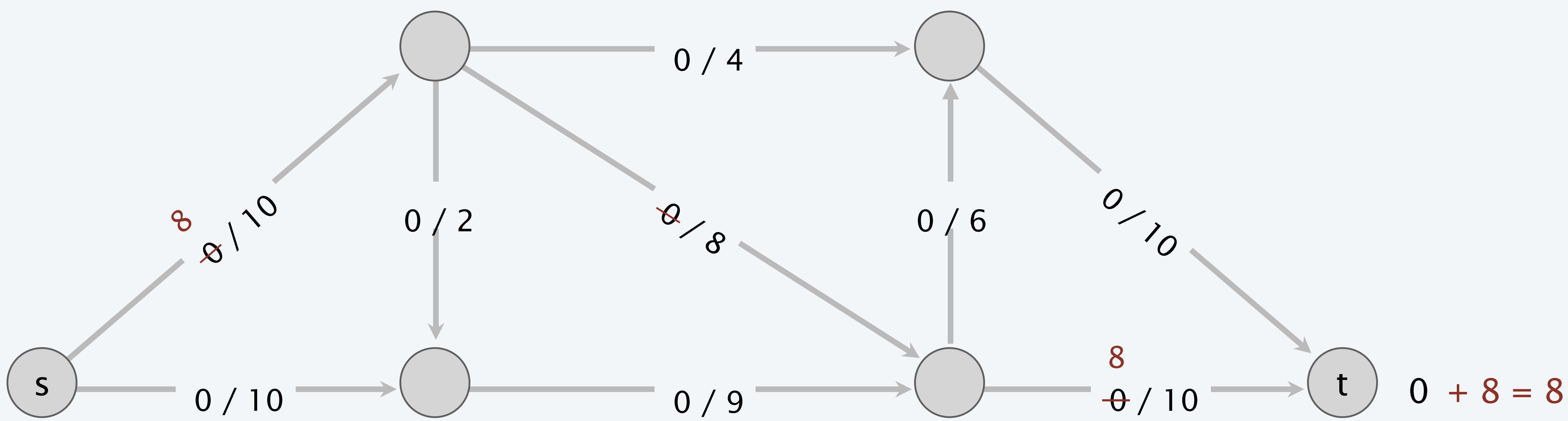


residual graph G_f

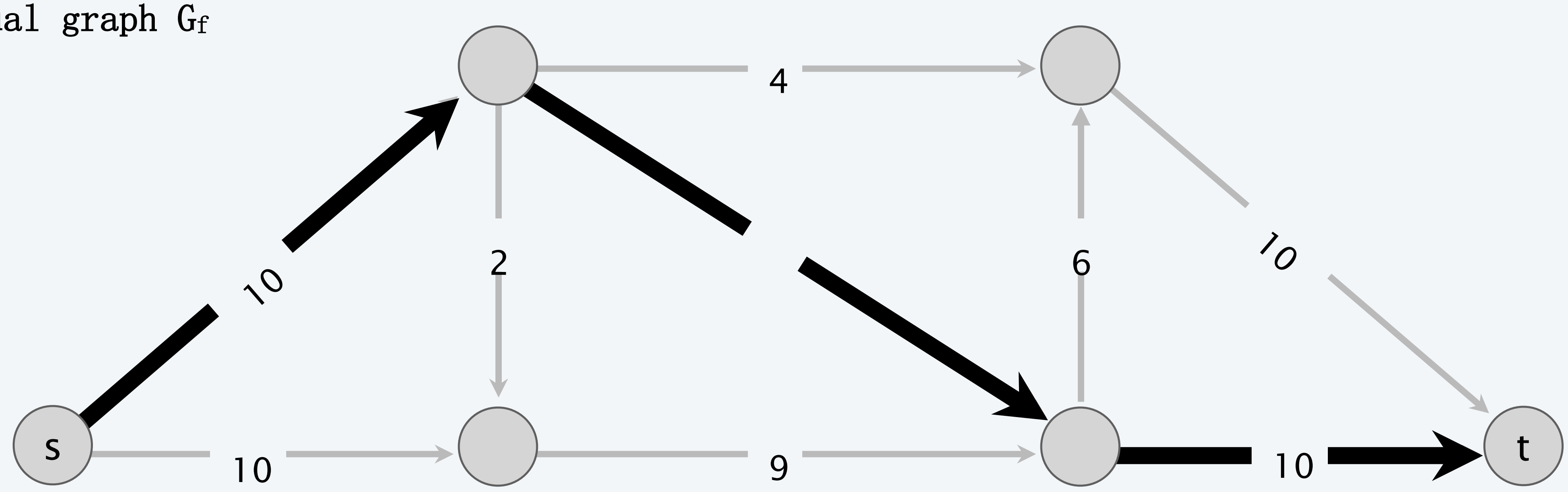


Ford-Fulkerson algorithm demo

network G

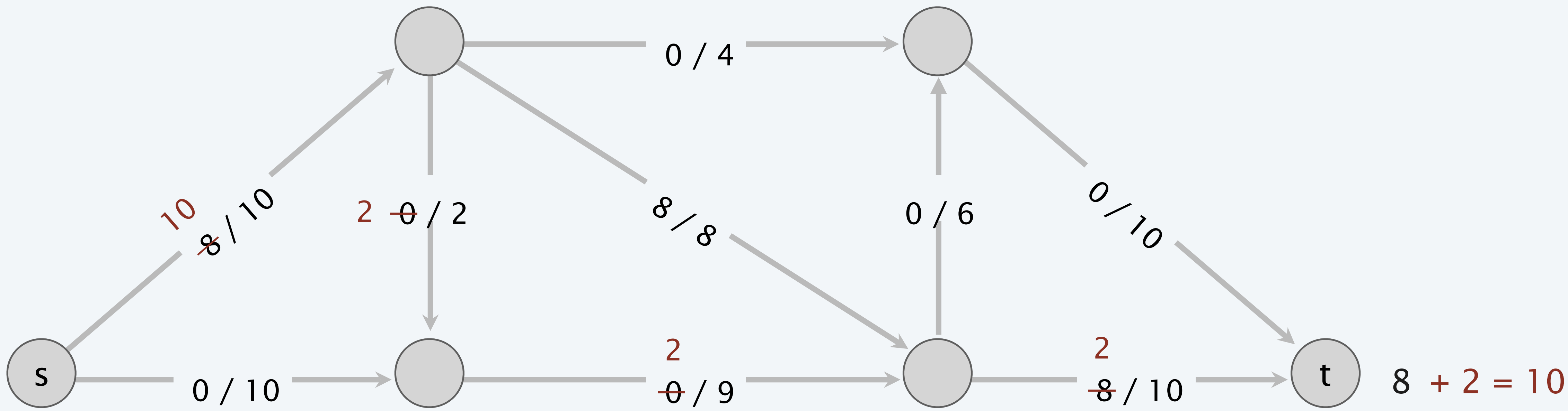


residual graph G_f

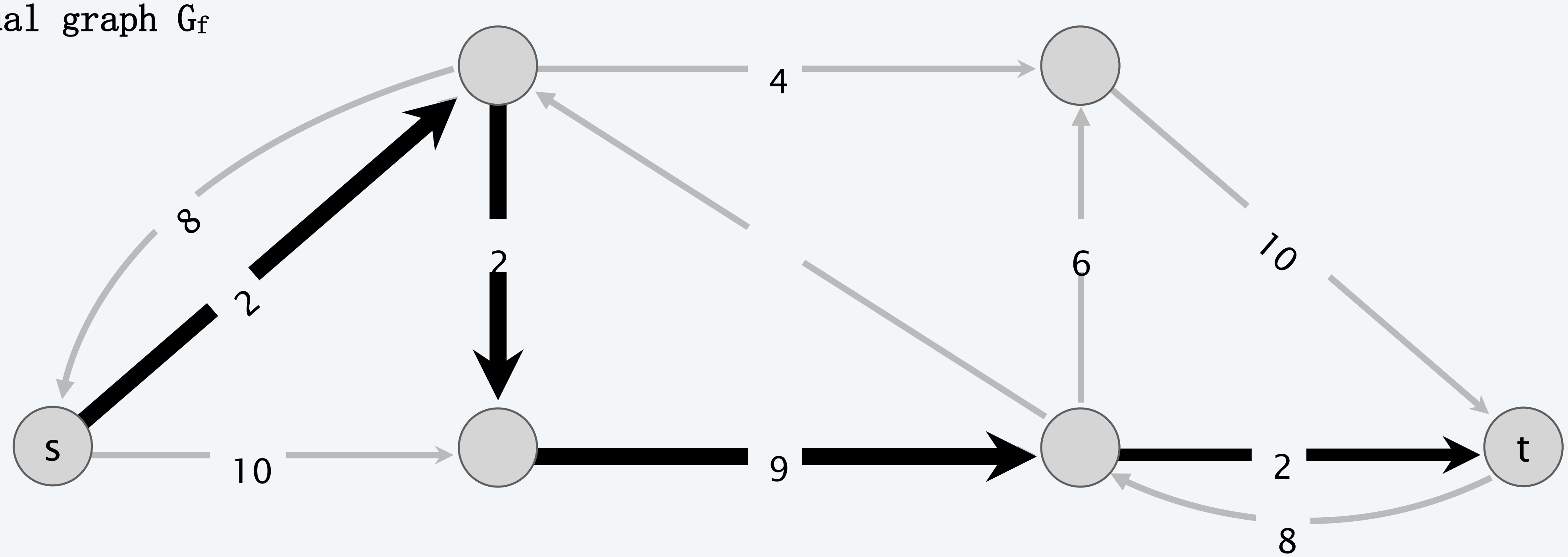


Ford-Fulkerson algorithm demo

network G

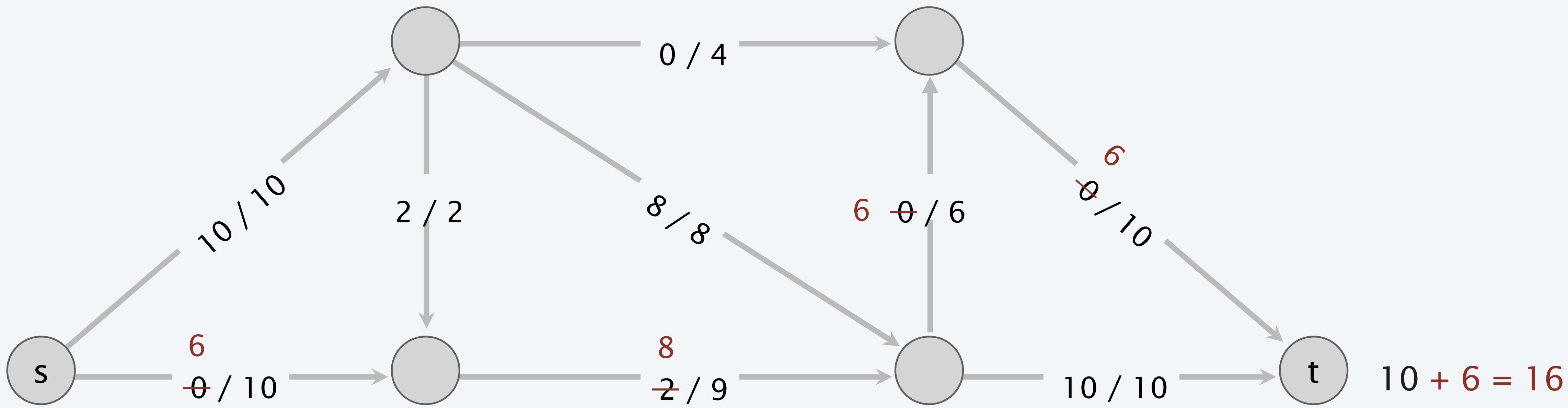


residual graph G_f

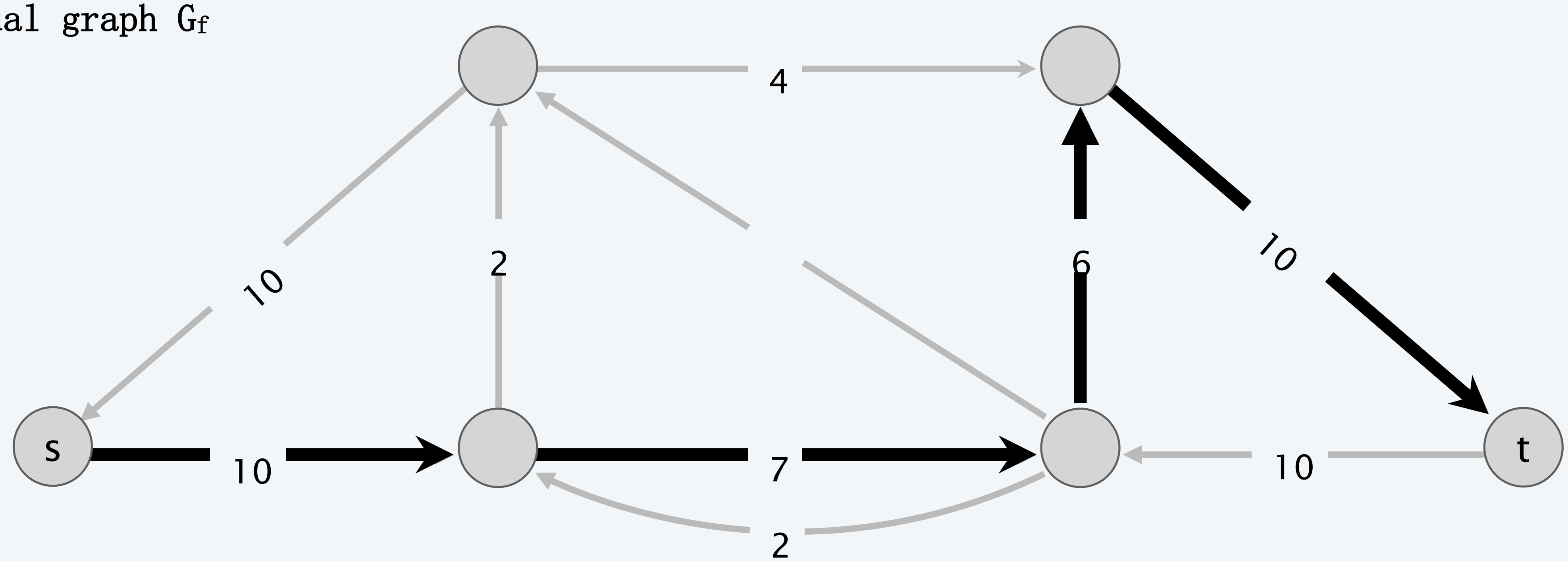


Ford-Fulkerson algorithm demo

network G

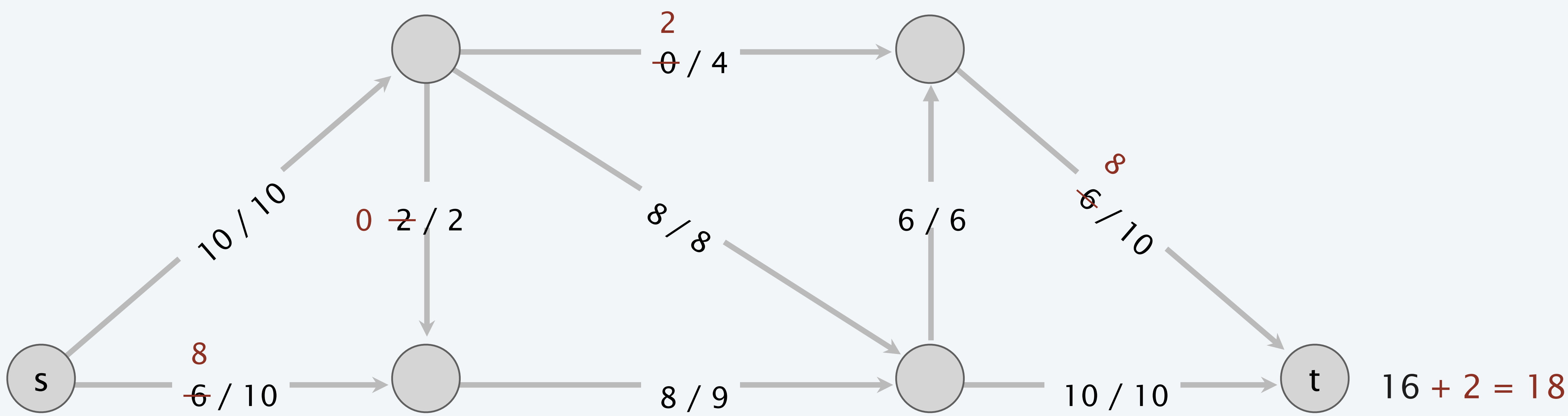


residual graph G_f

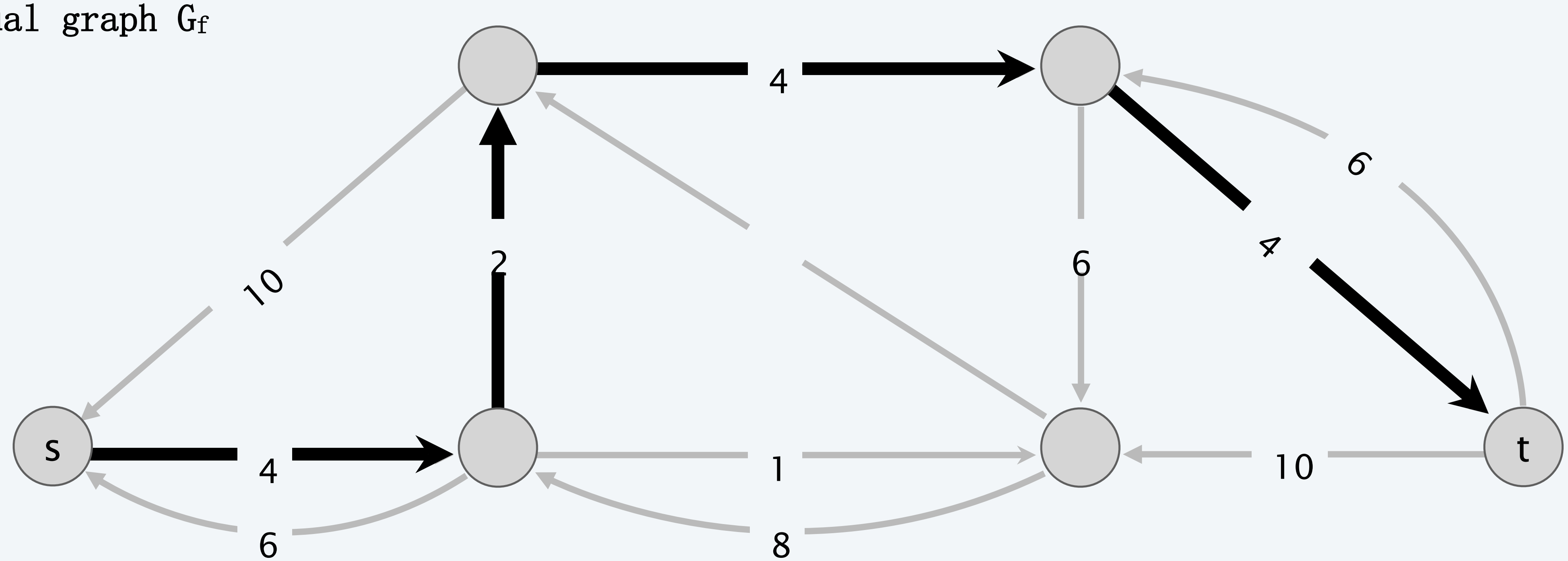


Ford-Fulkerson algorithm demo

network G

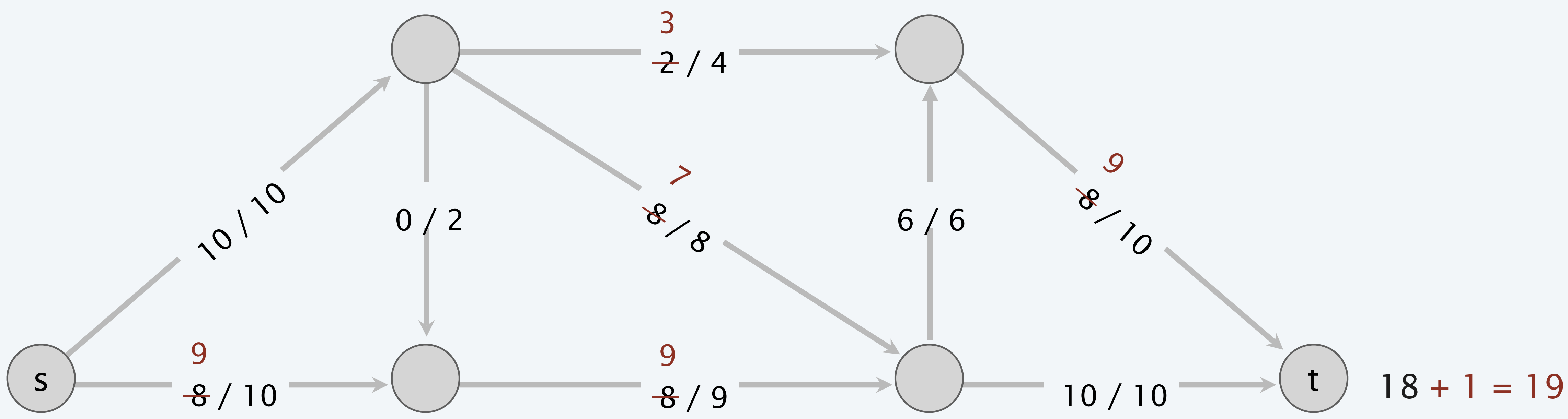


residual graph G_f

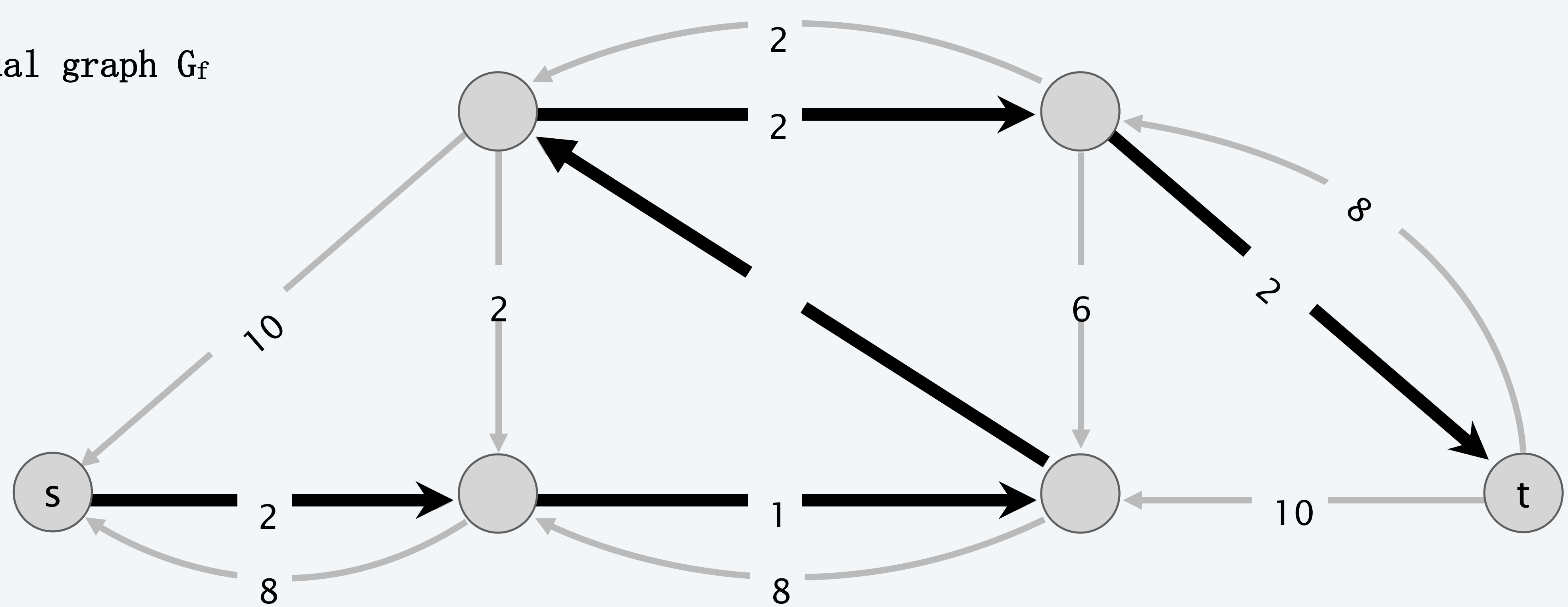


Ford-Fulkerson algorithm demo

network G

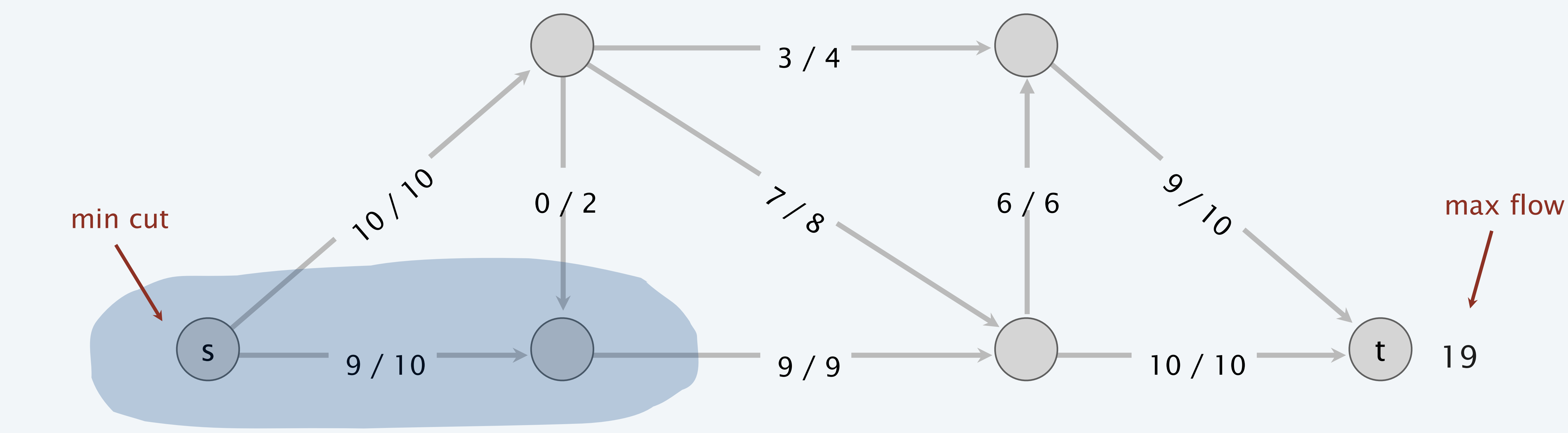


residual graph G_f

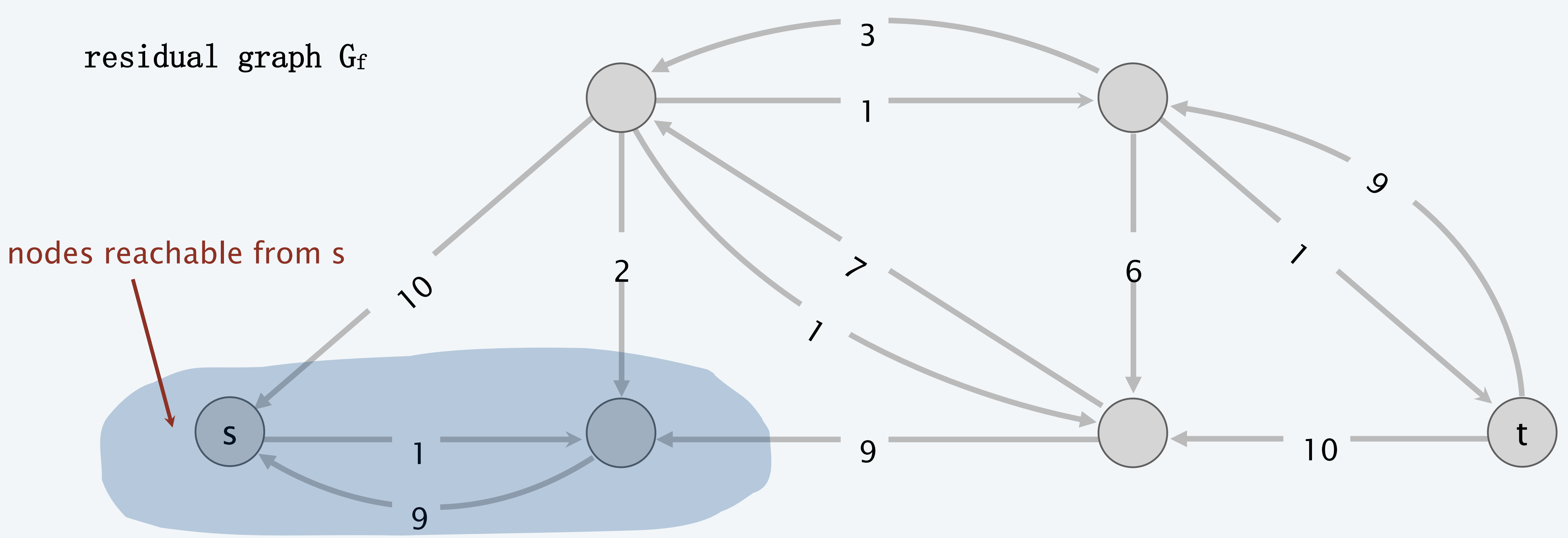


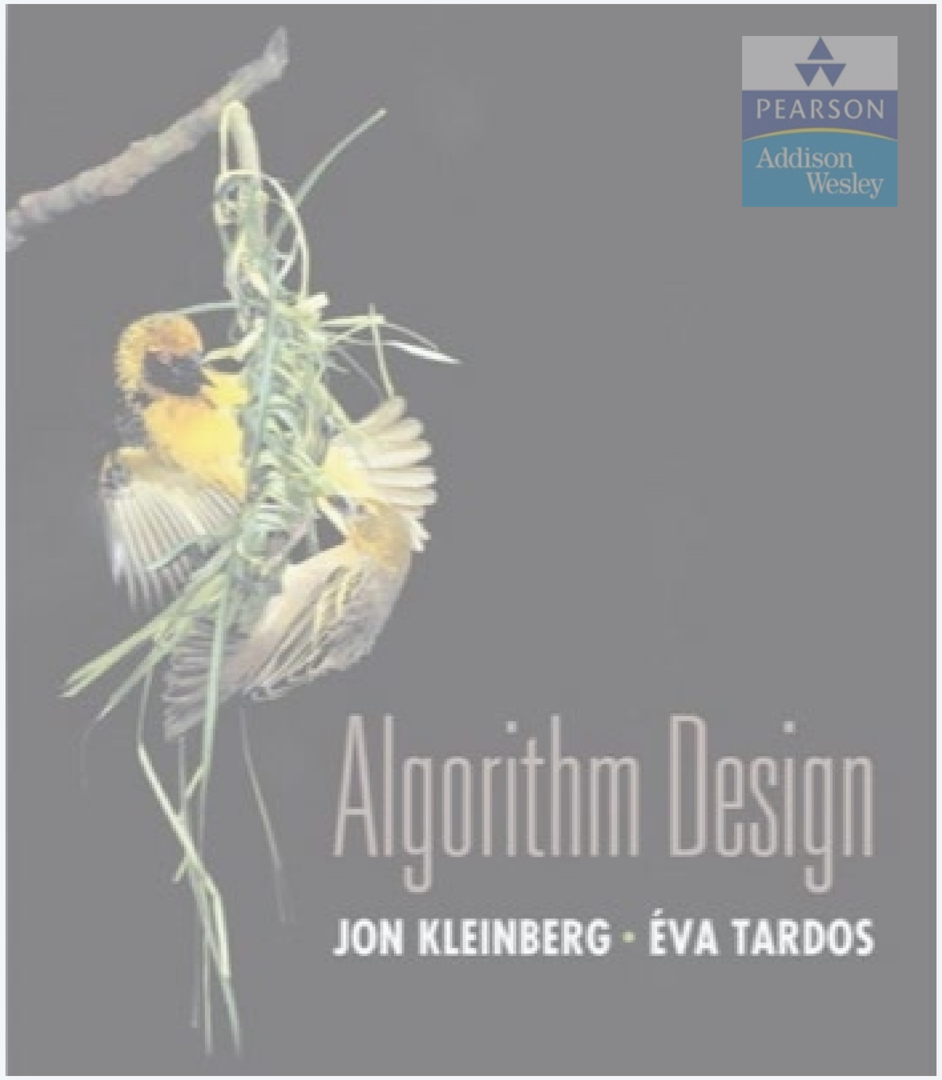
Ford-Fulkerson algorithm demo

network G



residual graph G_f





7. Network Flow I

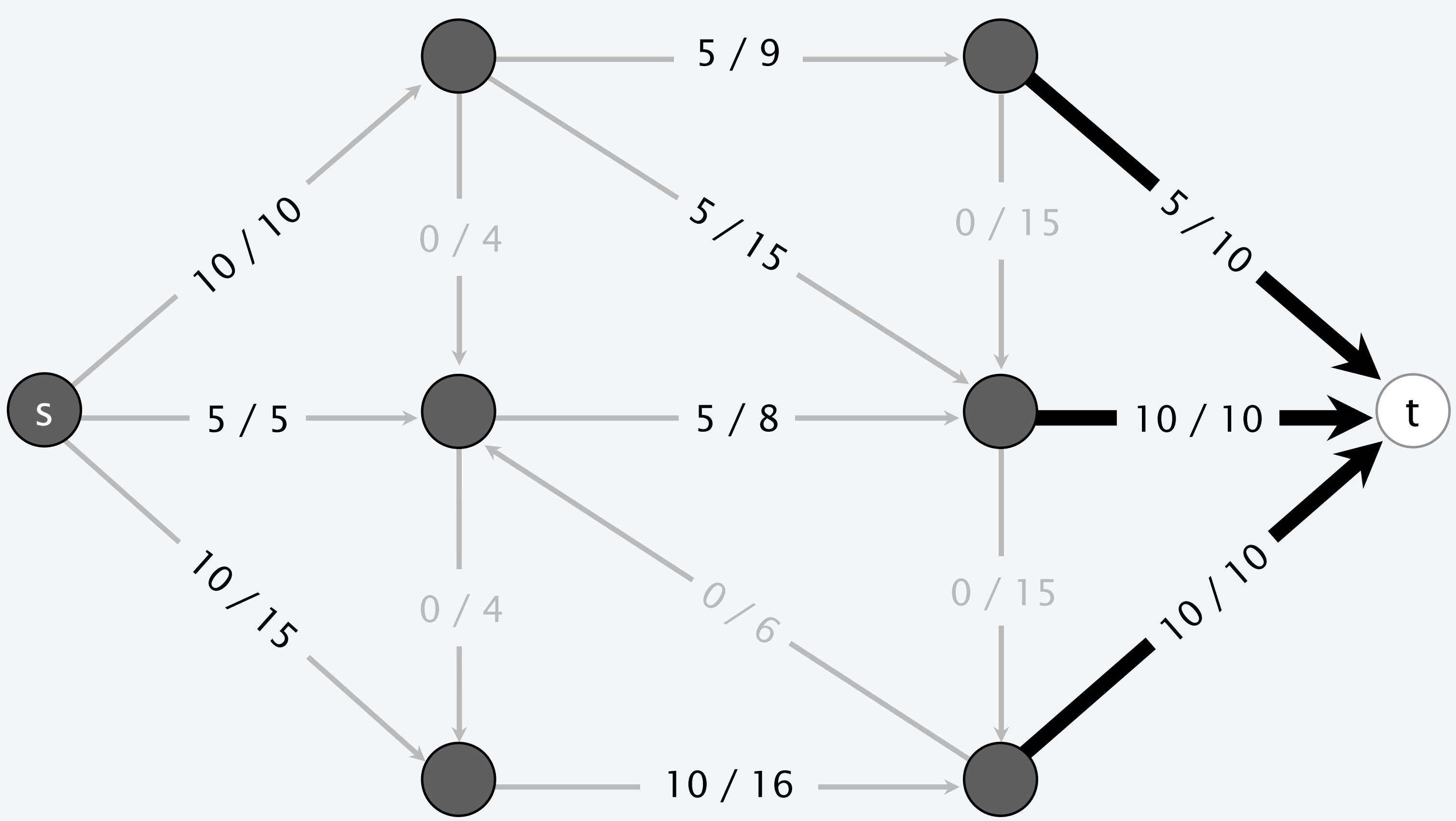
- *max-flow and min-cut problems*
- *Ford-Fulkerson algorithm*
- *max-flow min-cut theorem*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *blocking-flow algorithm*
- *unit-capacity simple networks*

Relationship between flows and cuts

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

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

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

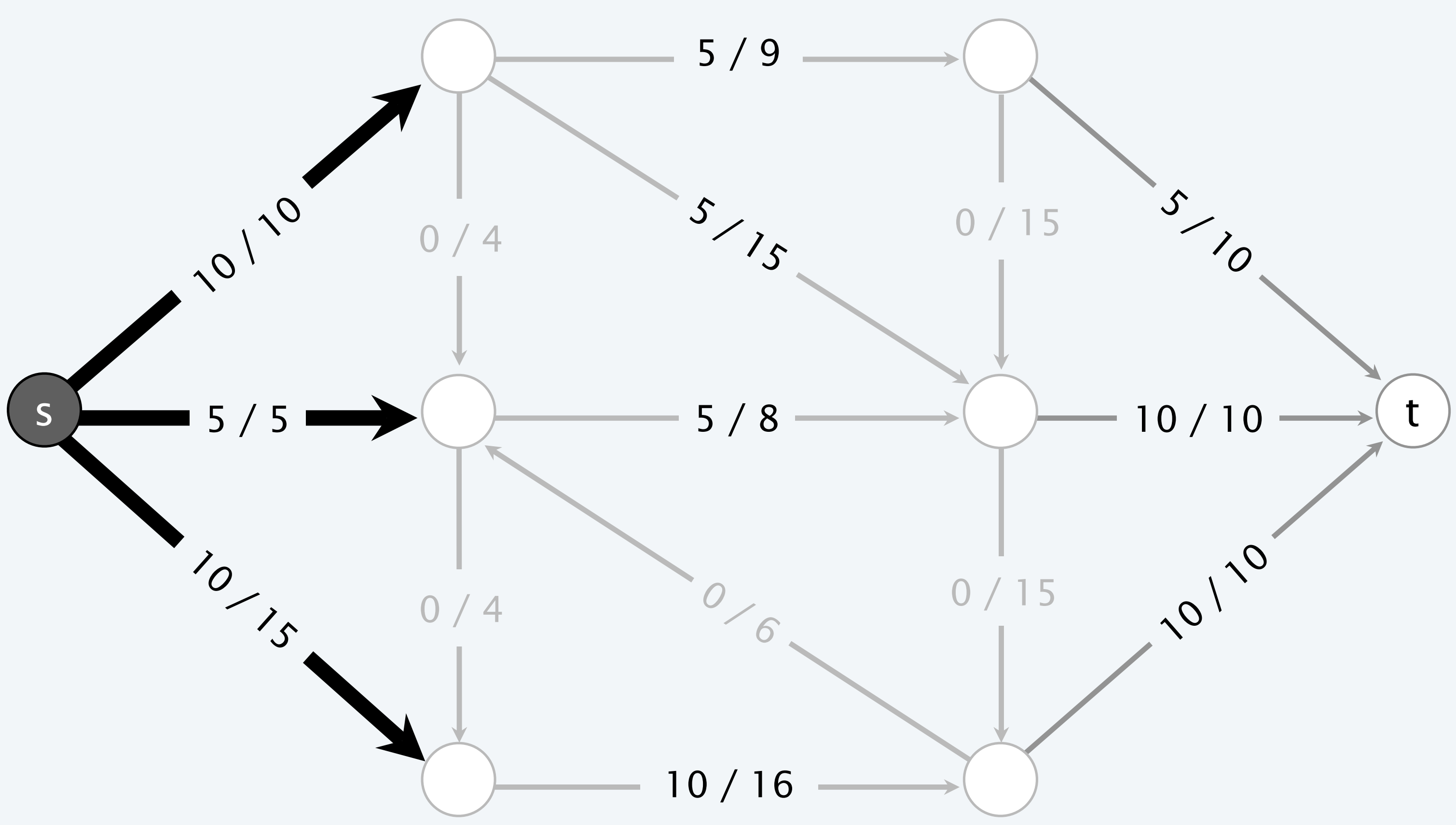


value of flow = 25

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

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

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



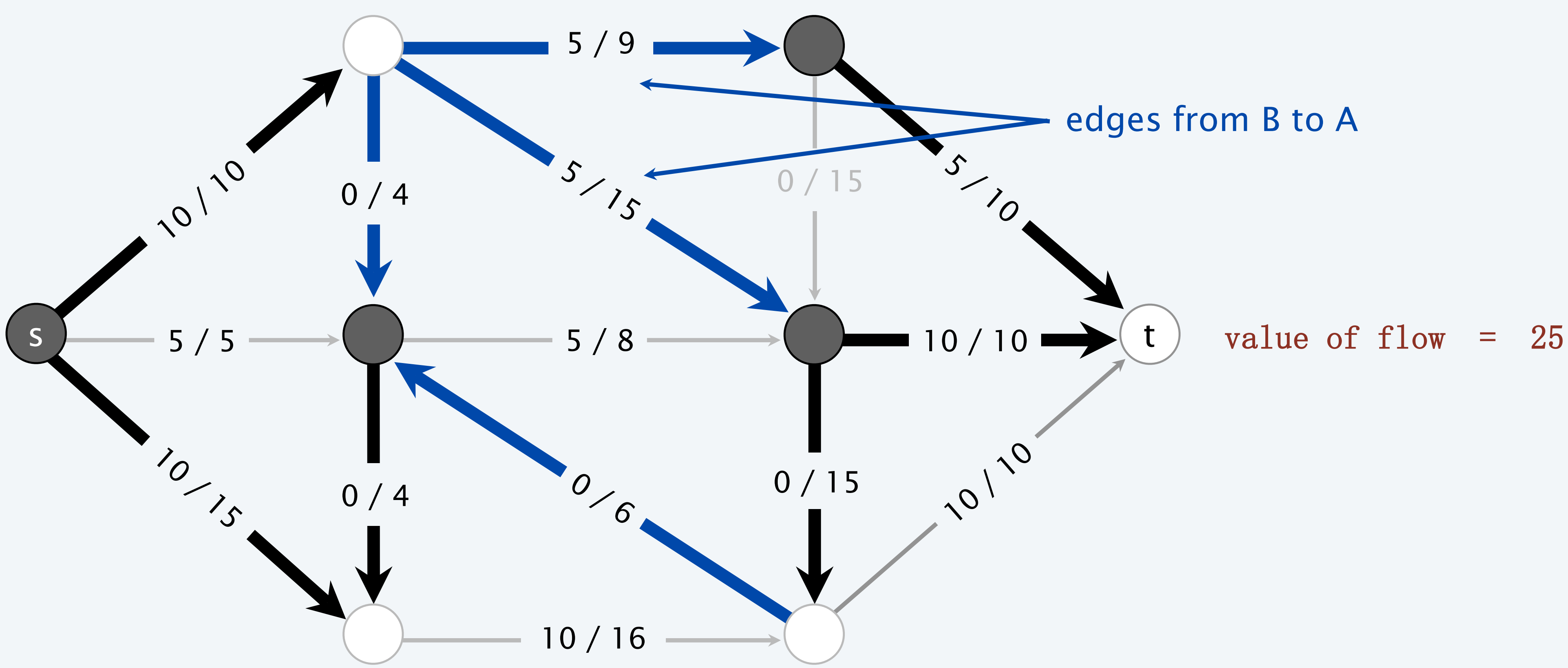
value of flow = 25

Relationship between flows and cuts

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

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

net flow across cut = (10 + 10 + 5 + 10 + 0 + 0) − (5 + 5 + 0 + 0) = 25



Relationship between flows and cuts

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

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

Pf.

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

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

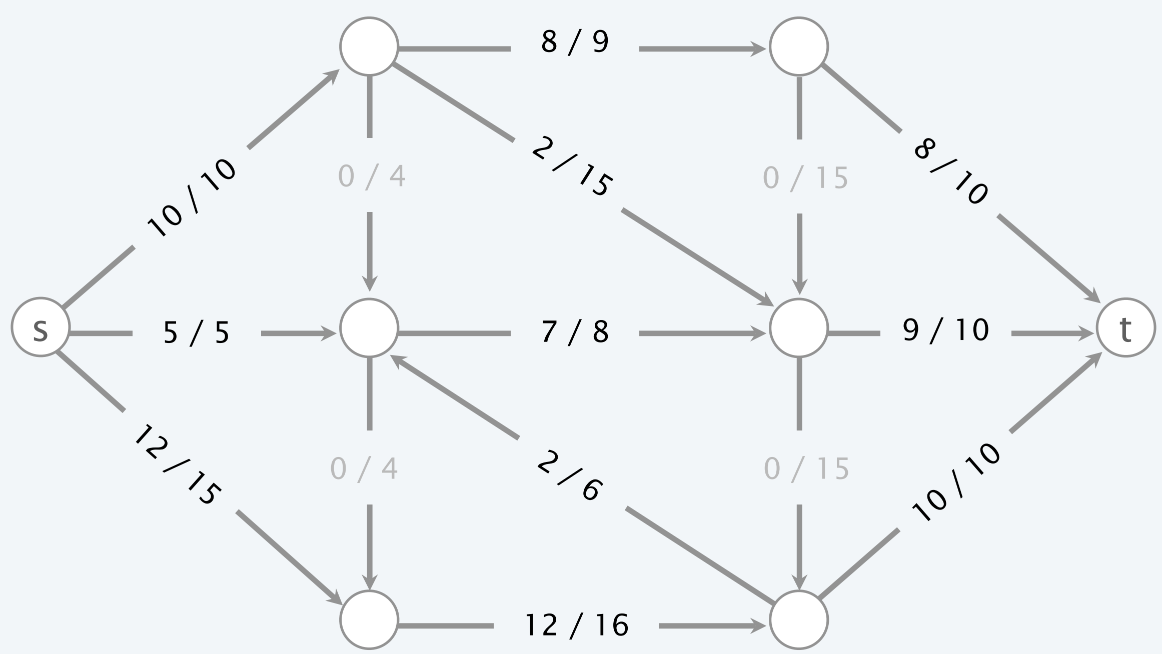
Relationship between flows and cuts

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

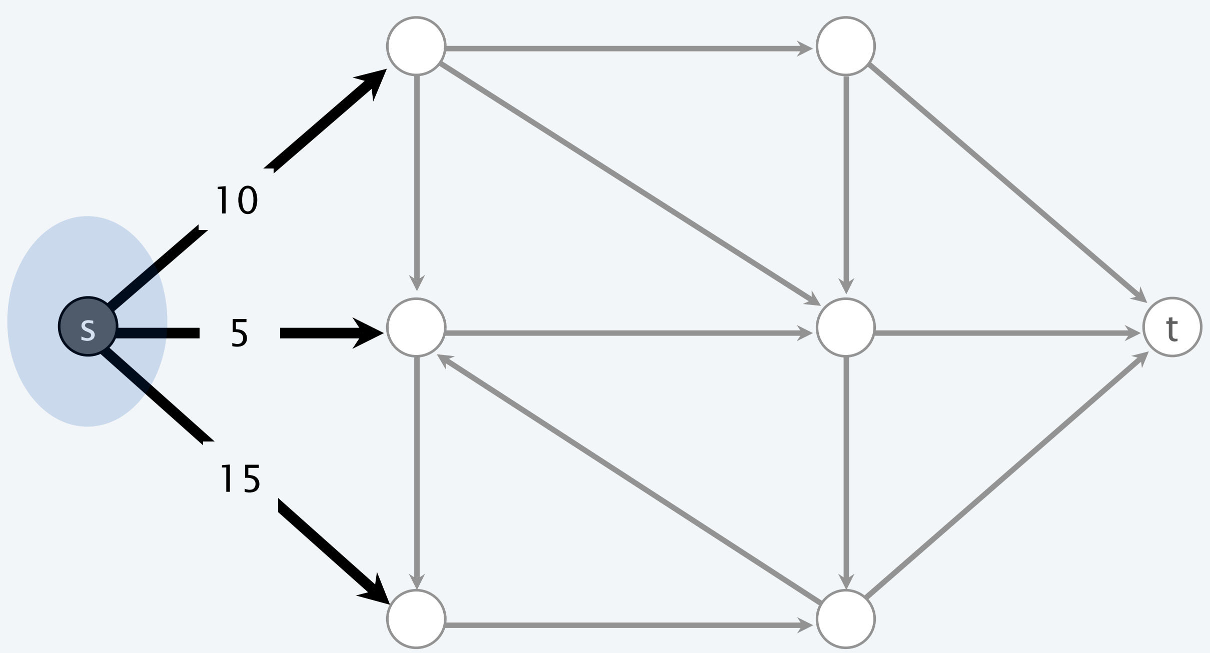
Pf.

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

flow-value lemma



value of flow = 27 $\leq$ capacity of cut = 30



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

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

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.

[i ⇒ ii]

- Suppose that (A, B) is a cut such that $cap(A, B) = val(f)$.
- Then, for any flow f', $\underbrace{val(f') \leq cap(A, B)}_{\text{weak duality}} \underbrace{=}_{\text{by assumption}} val(f)$.
- Thus, f is a max-flow. ▪

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

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

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 ⇒ iii] We prove contrapositive: ~iii ⇒ ~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. ▪

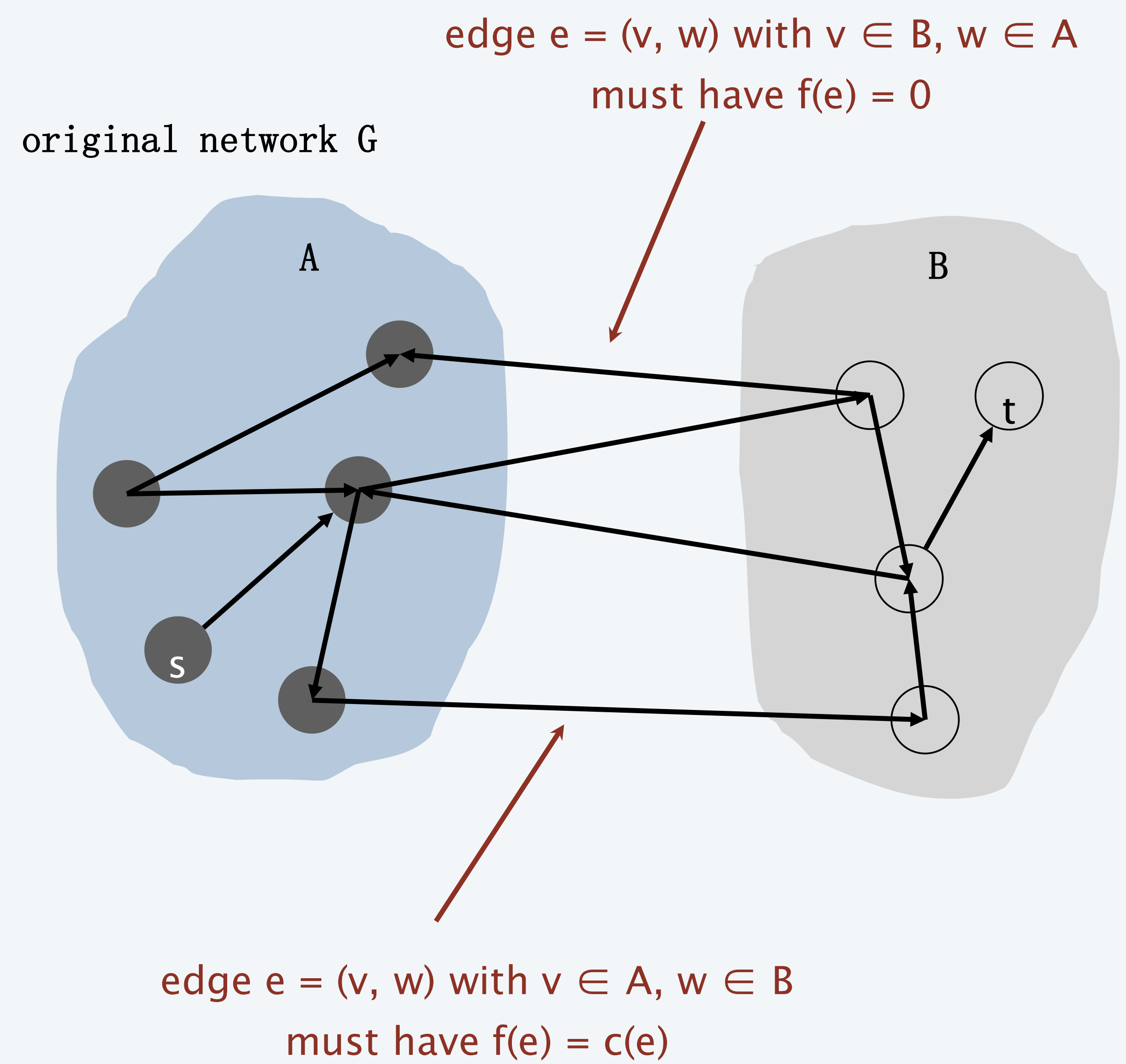
Max-flow min-cut theorem

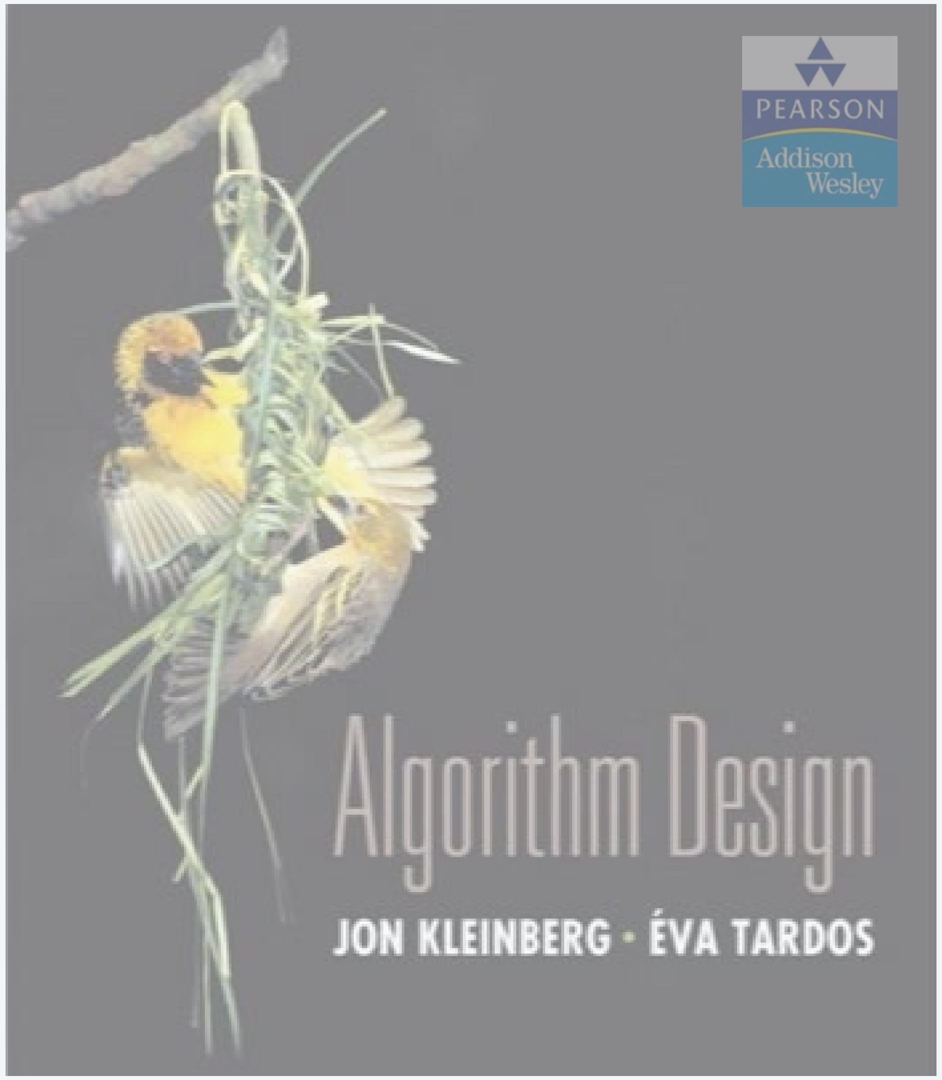
[iii ⇒ i]

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

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

flow-value lemma





7. Network Flow I

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

Running time

Assumption. Capacities are integers between 1 and C.

Integrality invariant. Throughout the algorithm, the flow values $f(e)$ and the residual capacities $c_f(e)$ are integers.

Theorem. The algorithm terminates in at most $val(f^*) \leq nC$ iterations.
Pf. Each augmentation increases the value by at least 1. ▪

Corollary. The running time of Ford-Fulkerson is $O(mnC)$.
Corollary. If $C = 1$, the running time of Ford-Fulkerson is $O(mn)$.

Integrality theorem. Then exists a max-flow f^* for which every flow value $f^*(e)$ is an integer.
Pf. Since algorithm terminates, theorem follows from invariant. ▪

Bad case for Ford-Fulkerson

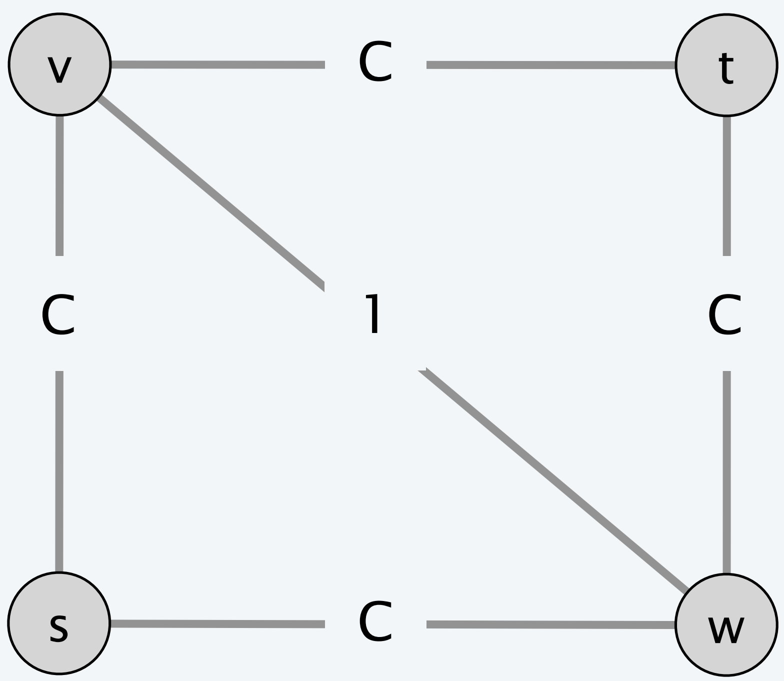
Q. Is generic Ford-Fulkerson algorithm poly-time in input size?

m, n, and log C

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

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

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



Choosing good augmenting paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

Choosing good augmenting paths

Choose augmenting paths with:

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

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

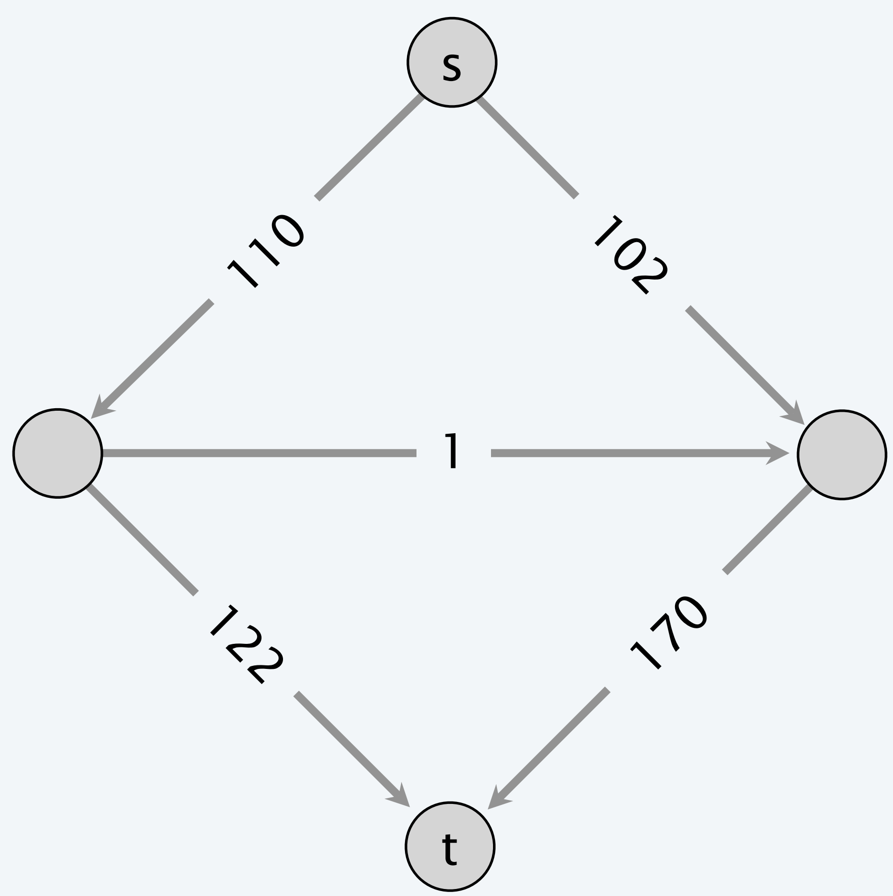
E. A. DINIC

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.

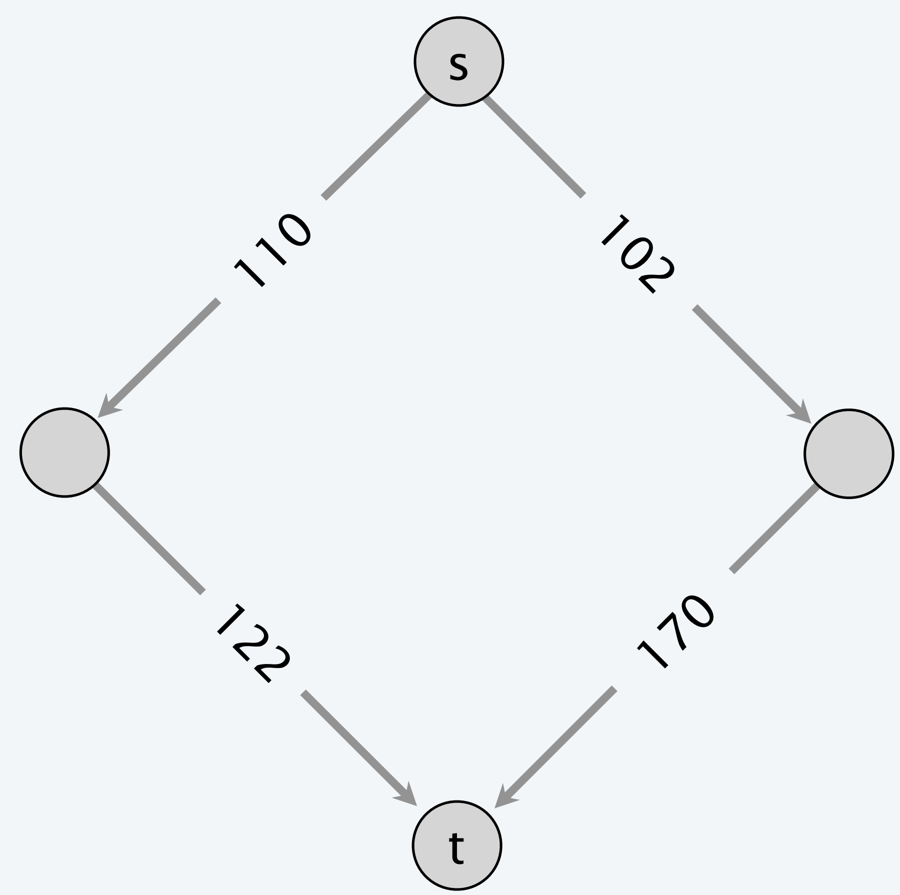
Dinic 1970 (Soviet Union)

Intuition. Choose augmenting path with highest bottleneck capacity: it increases flow by max possible amount in given iteration.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting only of arcs with capacity $\geq \Delta$.



G_f



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

CAPACITY-SCALING(G, s, t, c)

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 graph.

  WHILE (there exists an augmenting path P in $G_f(\Delta)$)

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

    Update $G_f(\Delta)$.

  $\Delta \leftarrow \Delta / 2$.

RETURN f.

Capacity-scaling algorithm: proof of correctness

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

Integrality invariant. All flow and residual capacity values are integral.

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. ▪

Capacity-scaling algorithm: analysis of running time

Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.
Pf. Initially $C/2 < \Delta \le C$; Δ decreases by a factor of 2 in each iteration. ▪

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the value of the max-flow $\le$ $val(f) + m\,\Delta$. ← proof on next slide

Lemma 3. There are at most $2m$ augmentations per scaling phase.
Pf.

- Let f be the flow at the end of the previous scaling phase.
- LEMMA 2 $\Rightarrow$ $val(f^*) \le val(f) + 2\,m\,\Delta$.
- Each augmentation in a Δ-phase increases $val(f)$ by at least Δ. ▪

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. An augmentation need $O(m)$, including setup of graph and finding a path. It can be implemented to run in $O(m^2 \log C)$ time.
Pf. Follows from LEMMA 1 and LEMMA 3. ▪

Capacity-scaling algorithm: analysis of running time

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the value of the max-flow $\leq$ $val(f) + m\,\Delta$.

Pf.

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

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