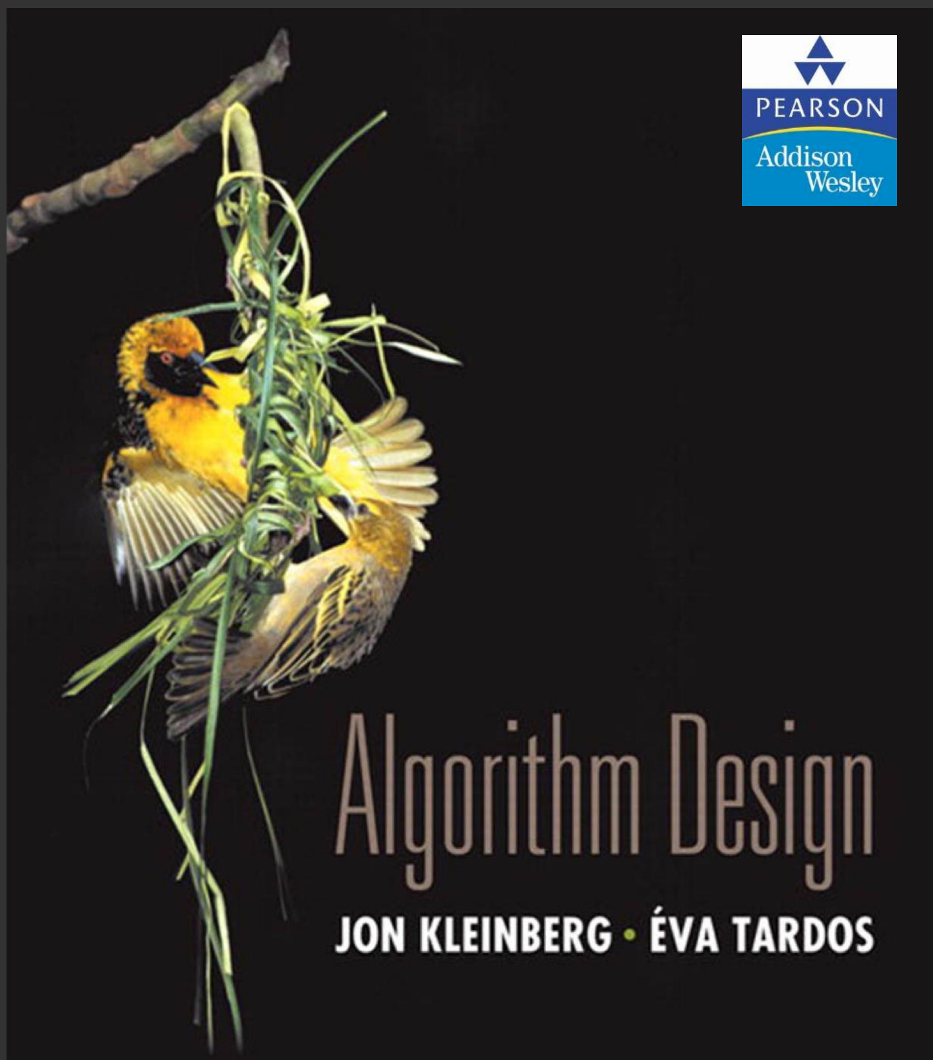
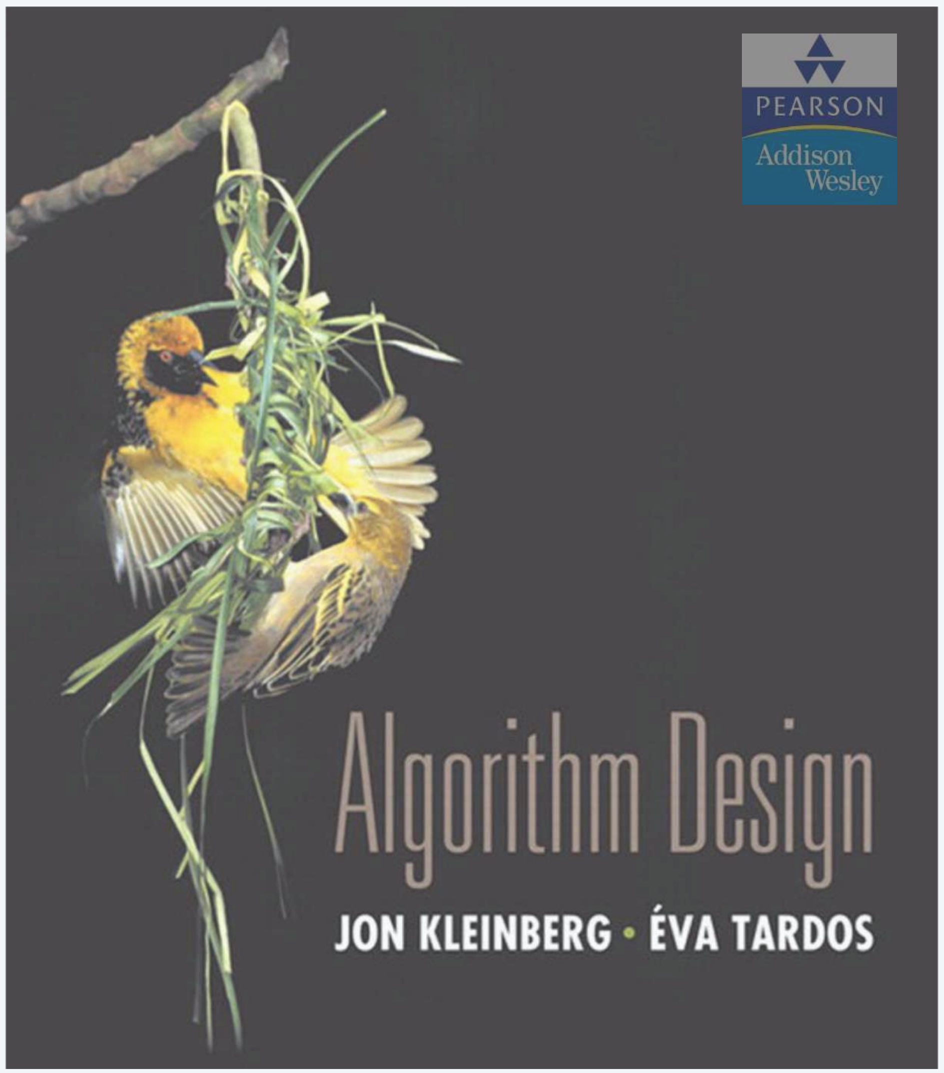


7. Network Flow I

- *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*

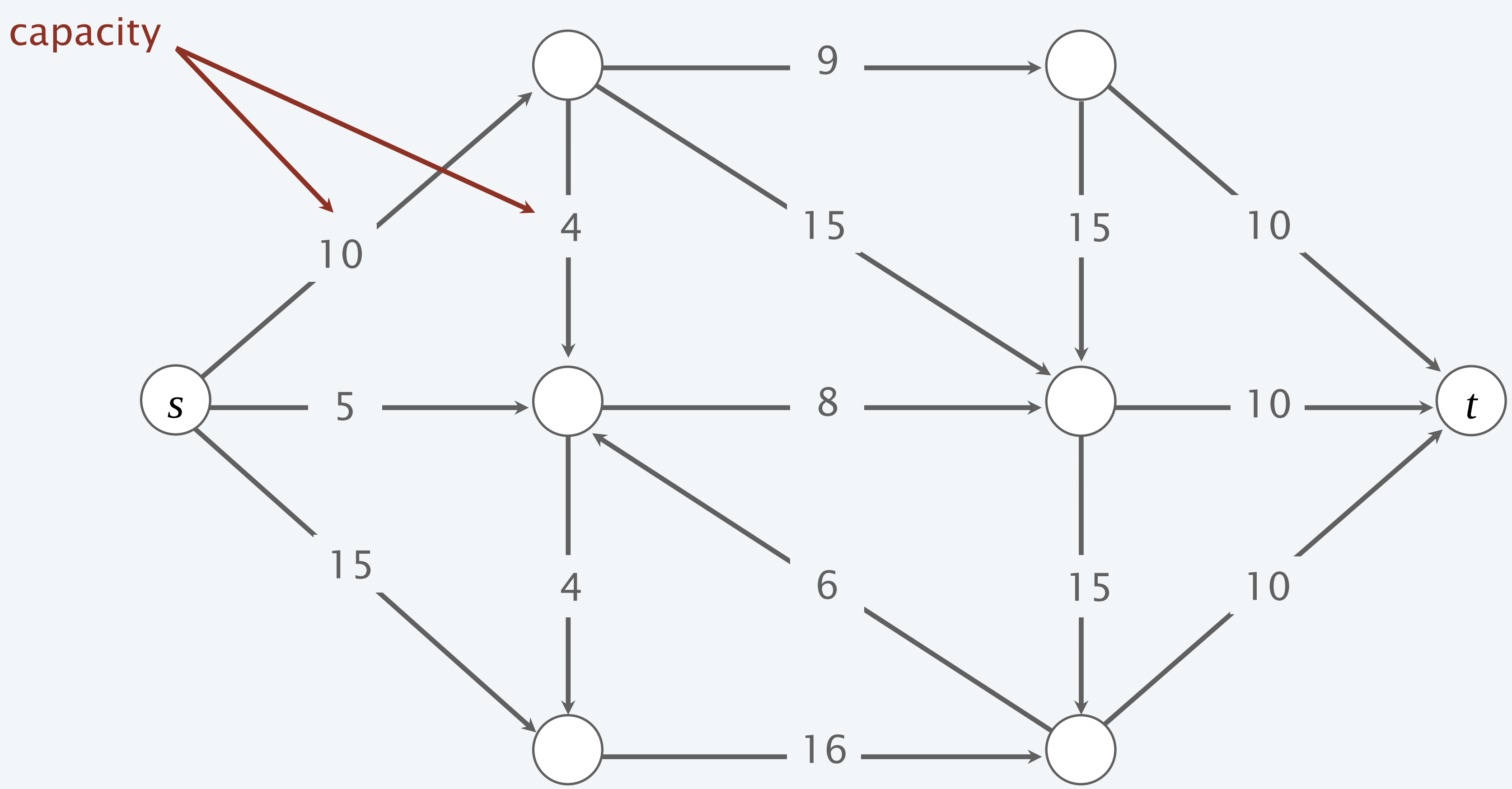
Flow network

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.

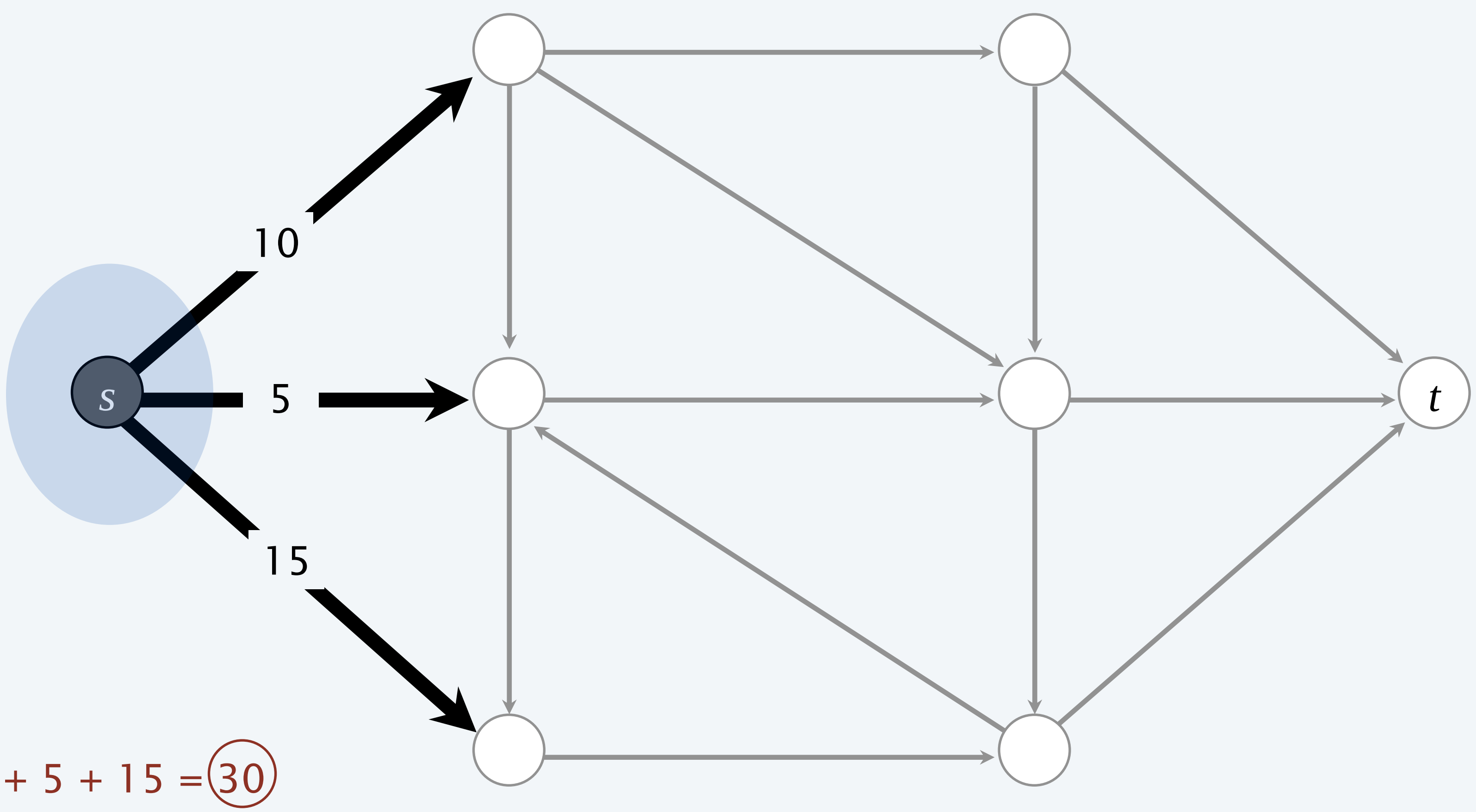


Minimum-cut problem

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.

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



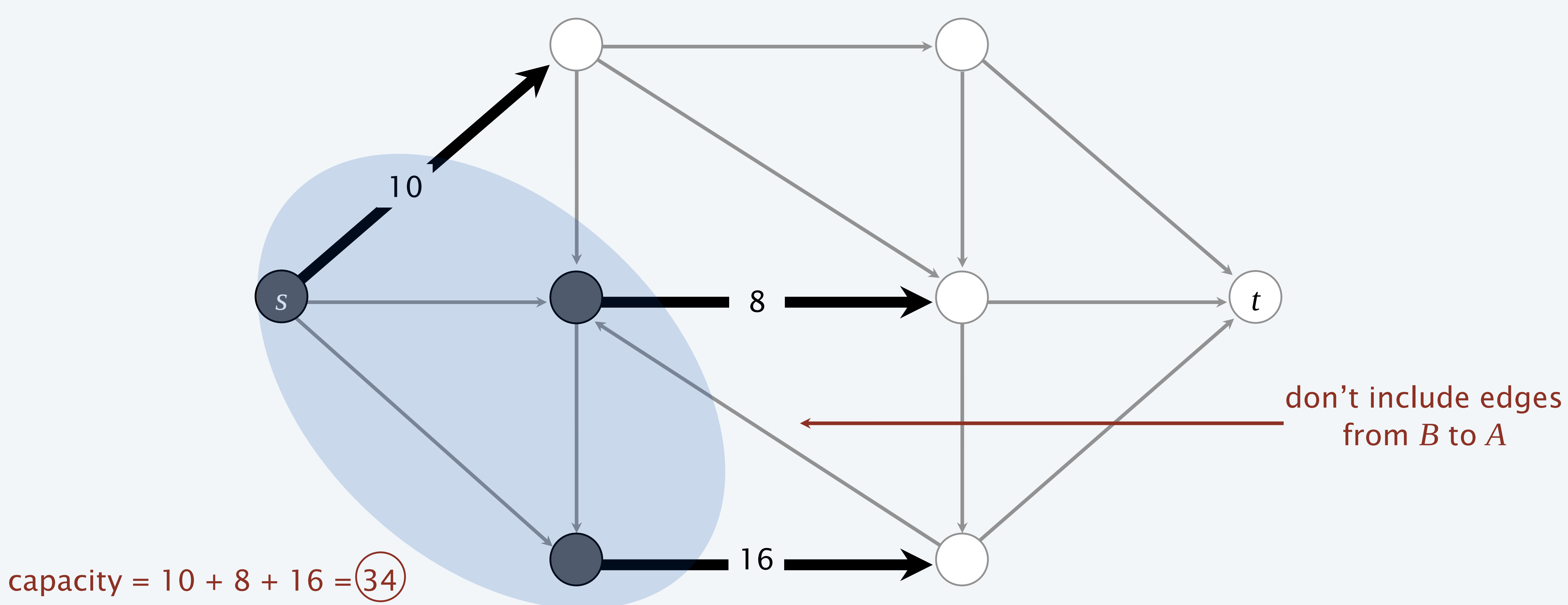
capacity = 10 + 5 + 15 = 30

Minimum-cut problem

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

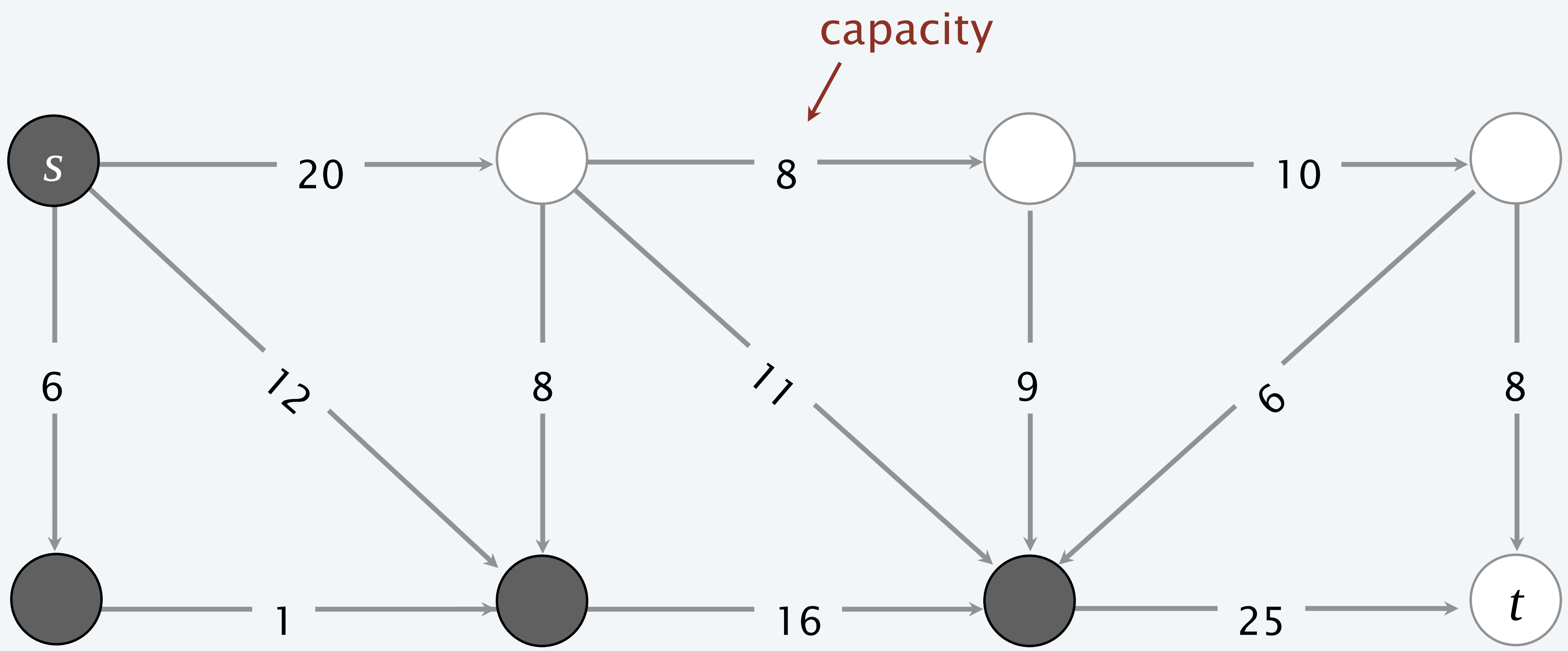
Def. Its capacity is the sum of the capacities of the edges from A to B.

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



One more example

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



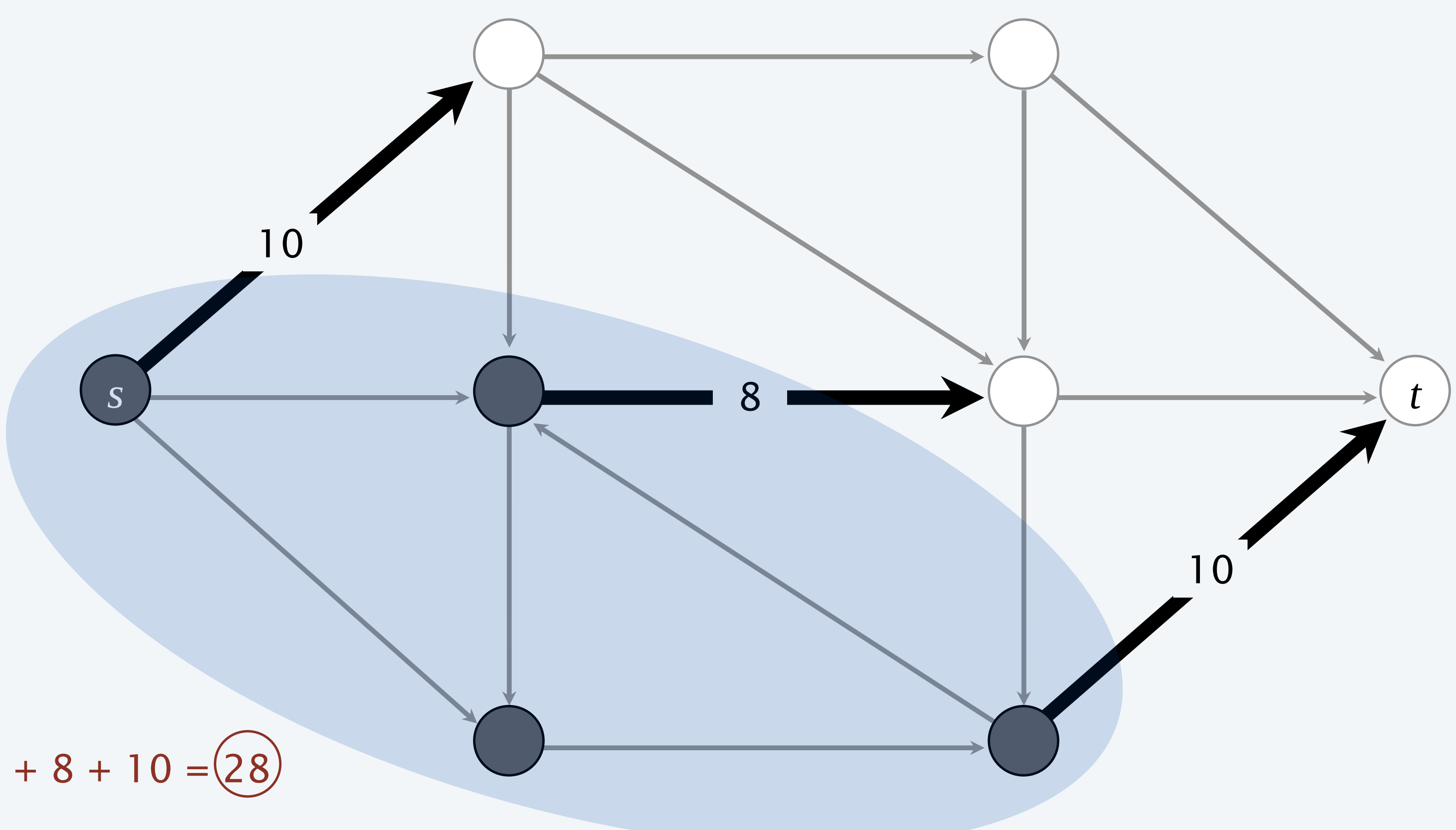
Minimum-cut problem

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.

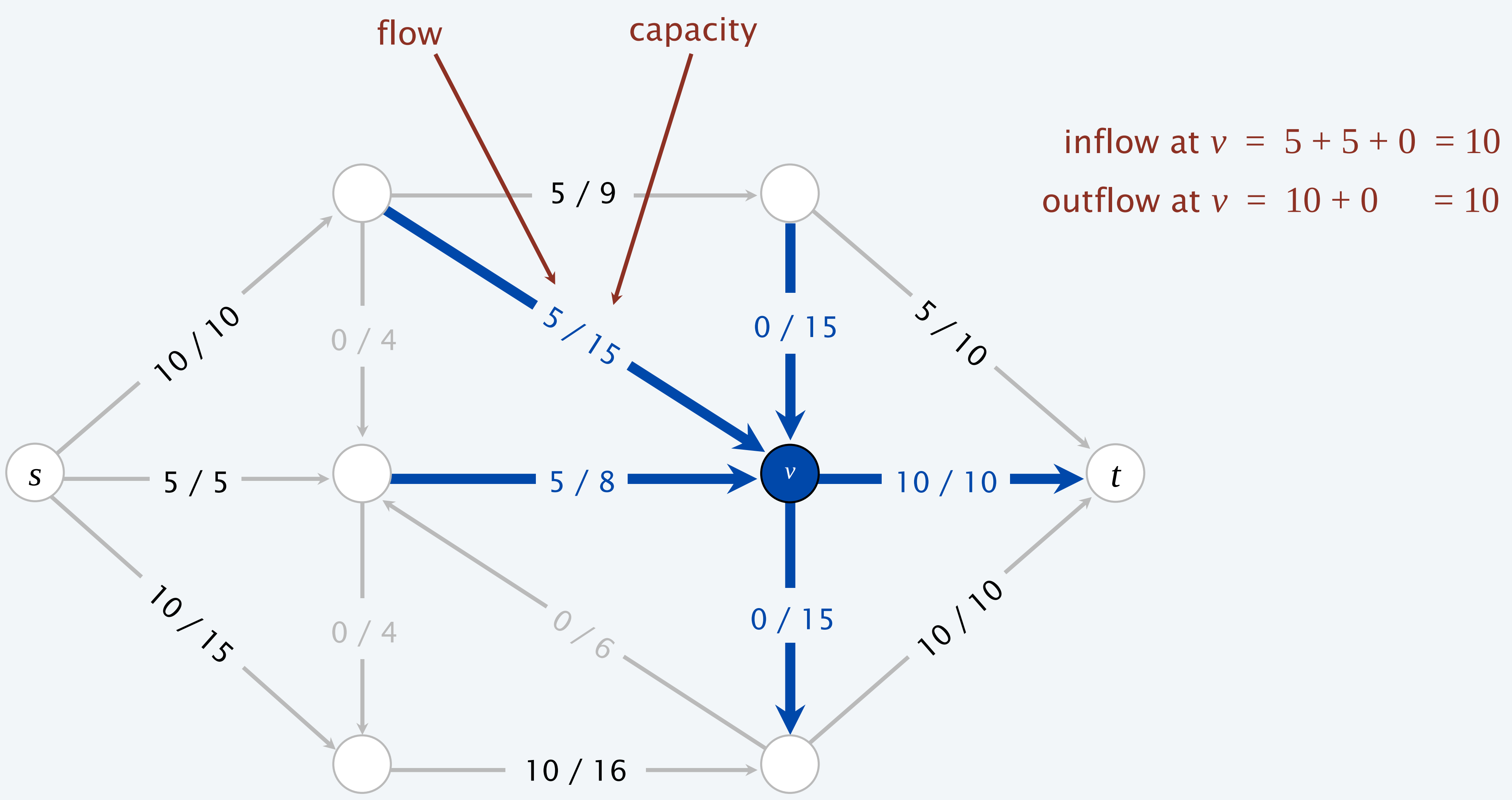


capacity = 10 + 8 + 10 = 28

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

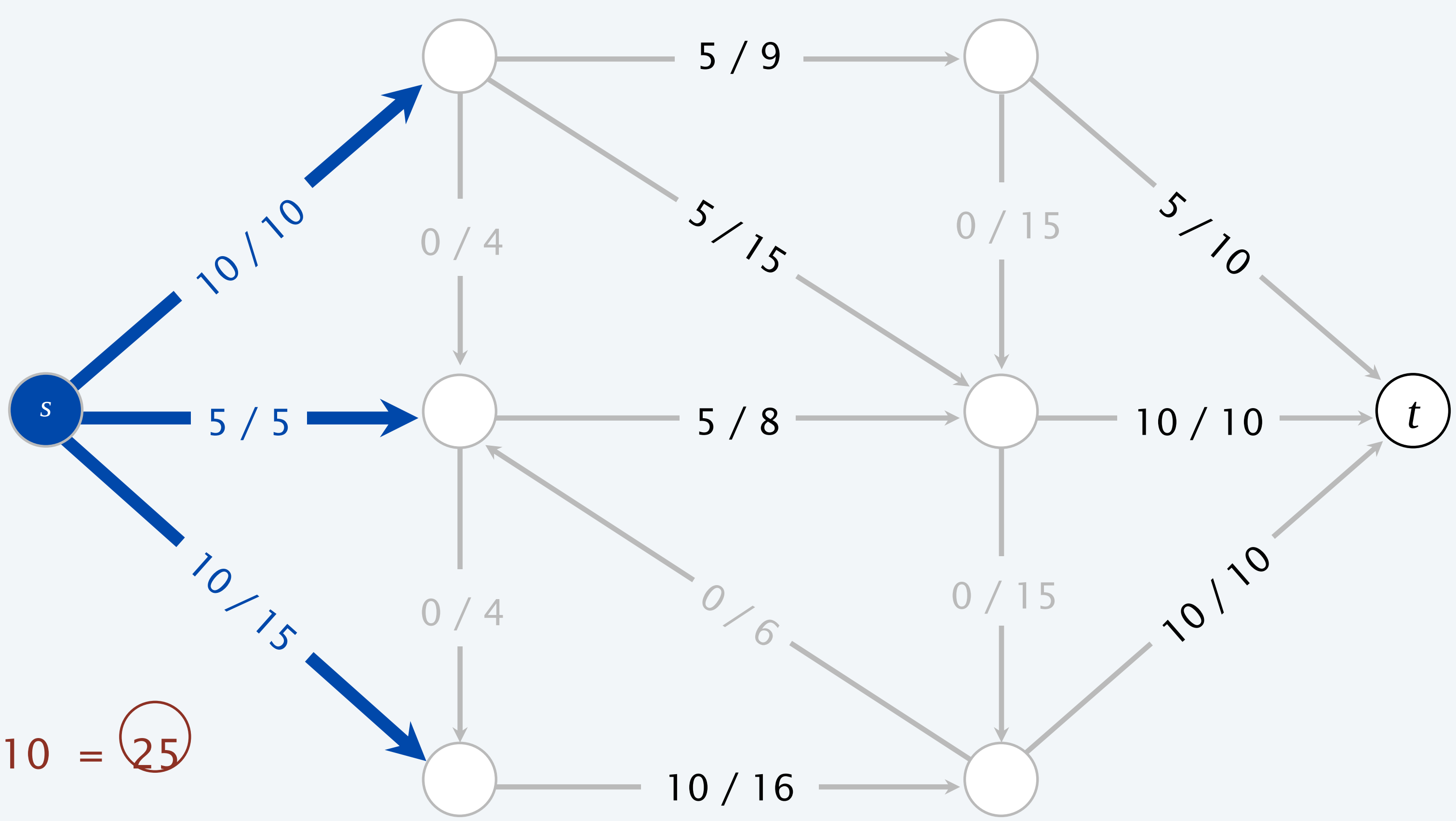


Maximum-flow problem

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

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



value = 5 + 10 + 10 = 25

Maximum-flow problem

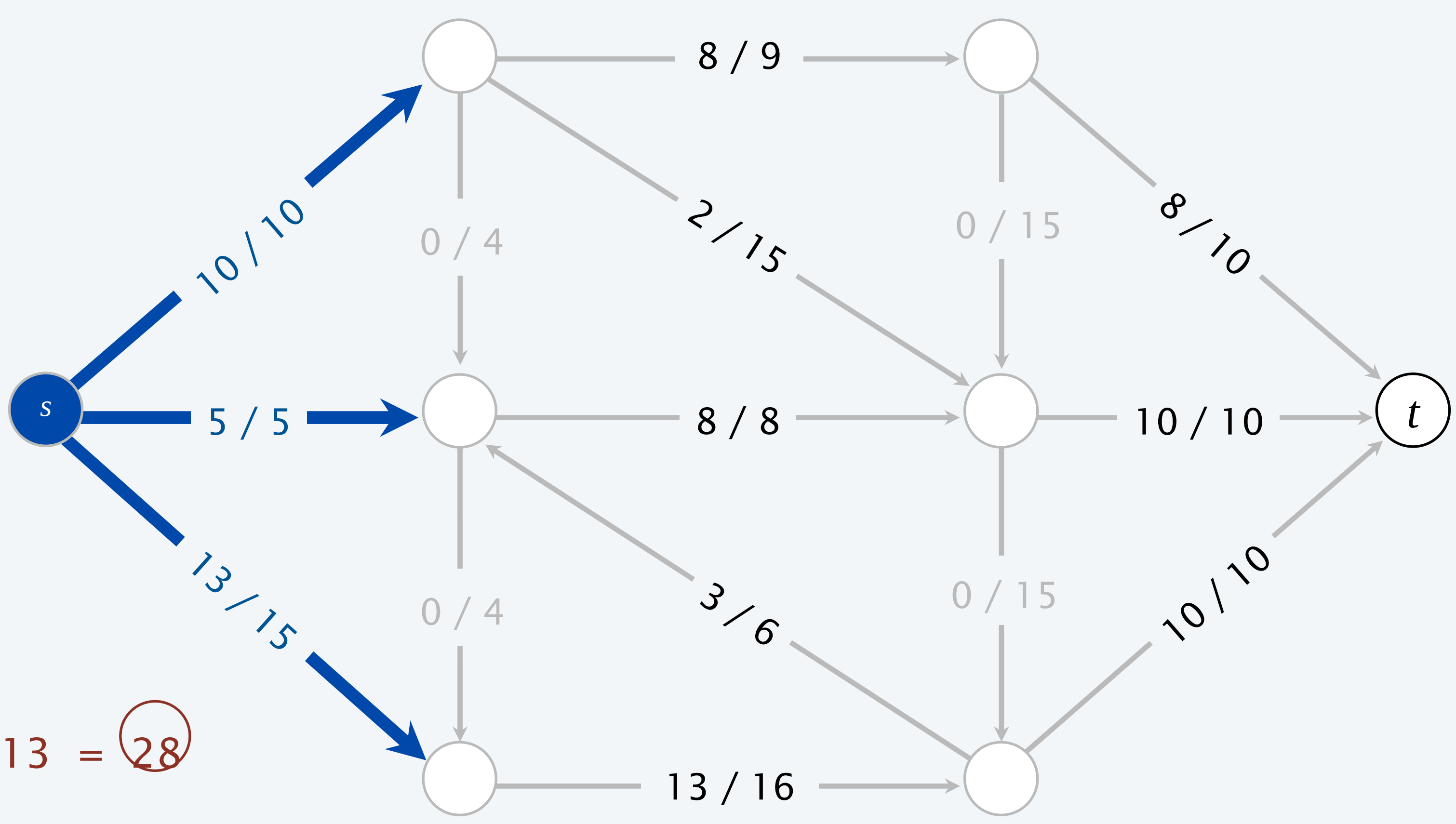
Def. An *st-flow (flow)* f is a function that satisfies:

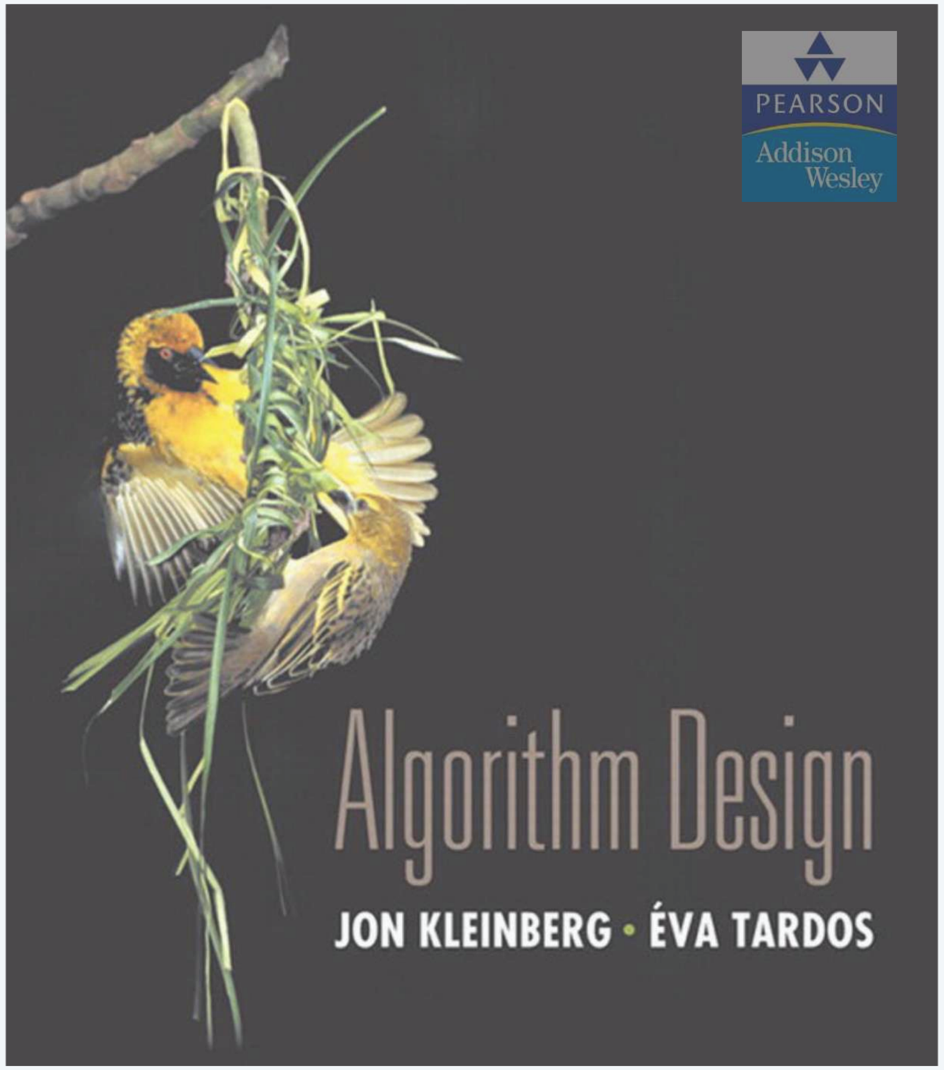
- For each $e \in E$: $\quad 0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

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

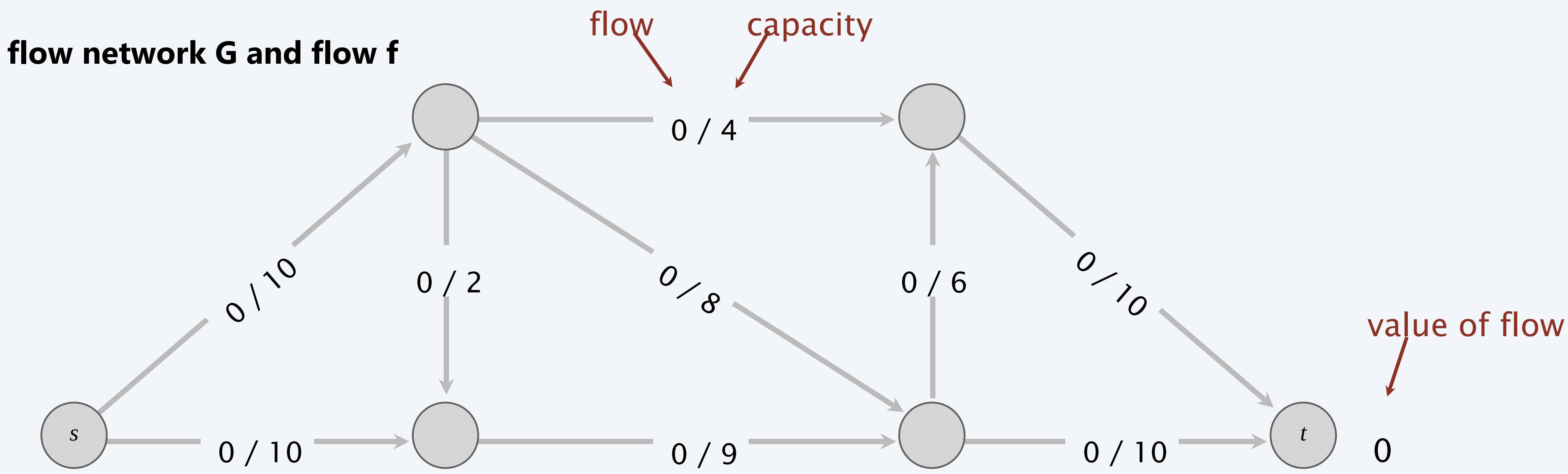
7. NETWORK FLOW I

- *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.

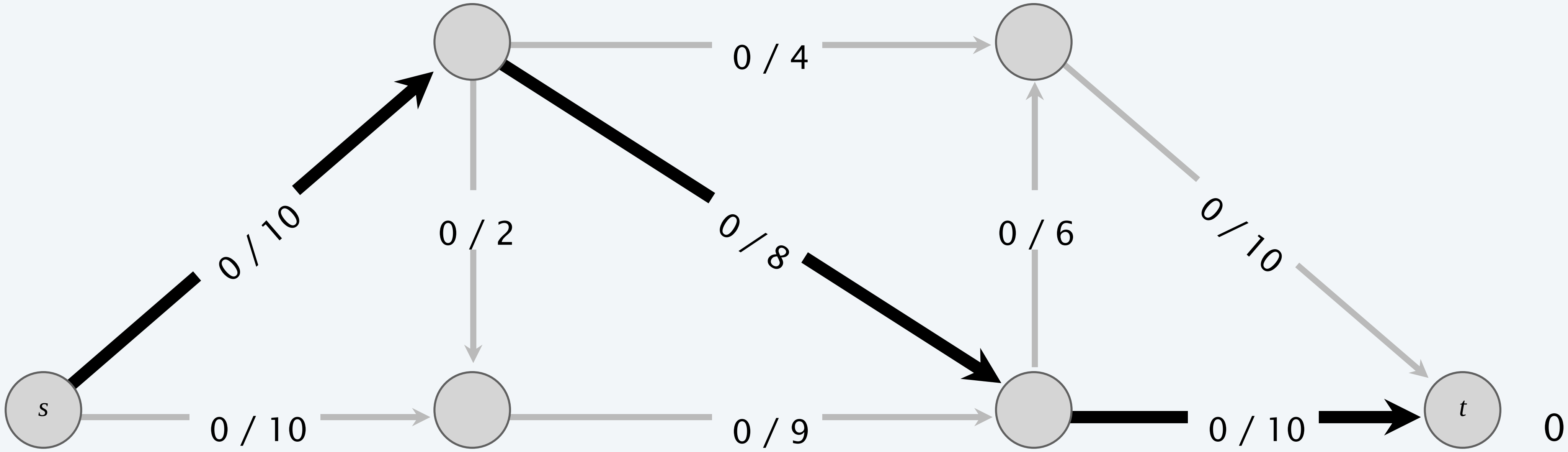


Toward a max-flow algorithm

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

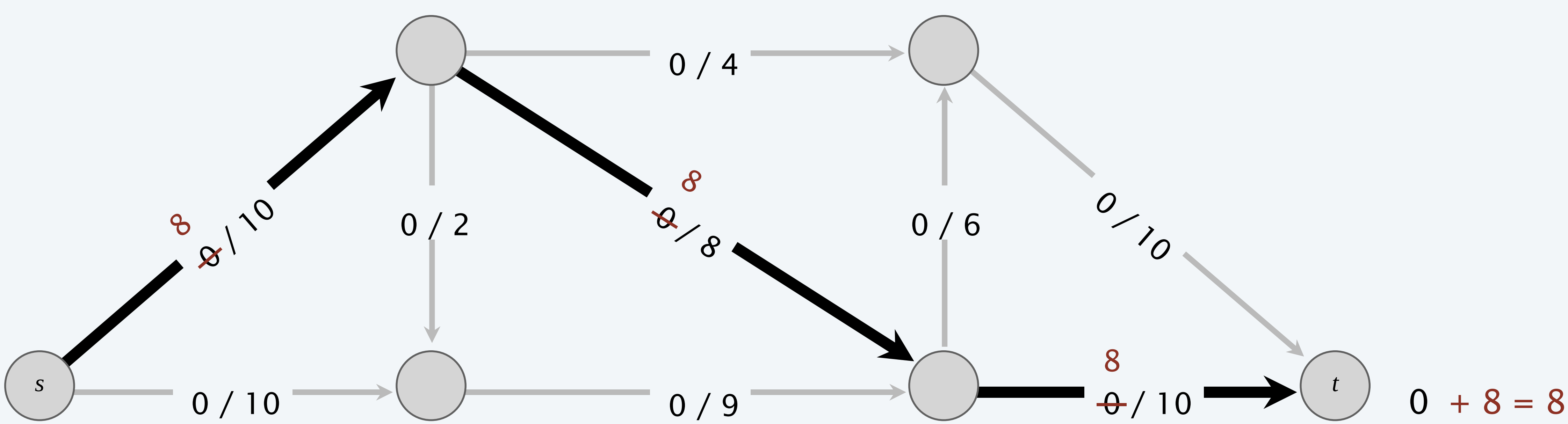


Toward a max-flow algorithm

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

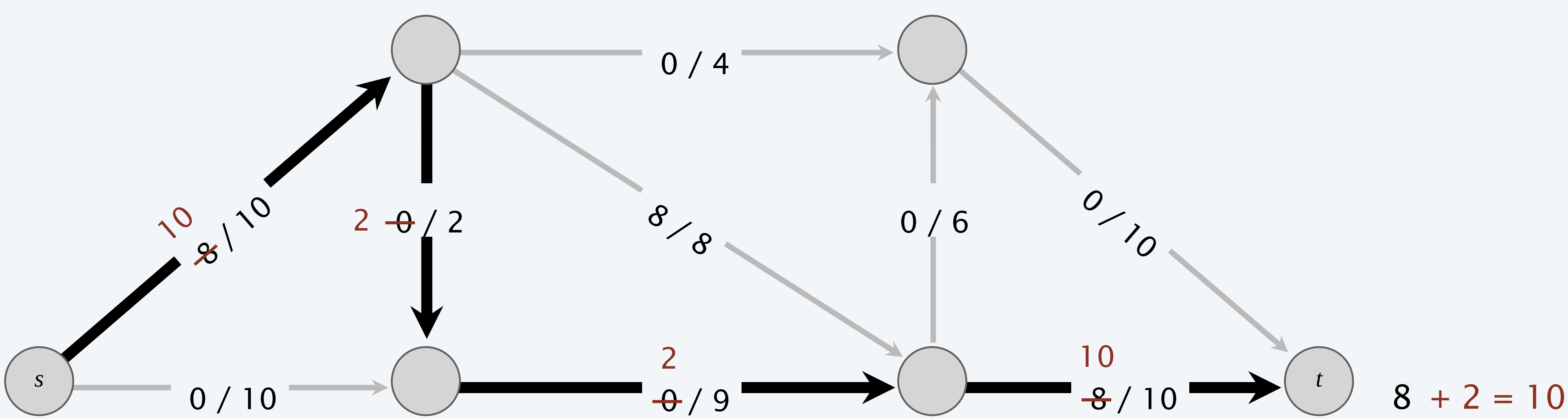


Toward a max-flow algorithm

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.

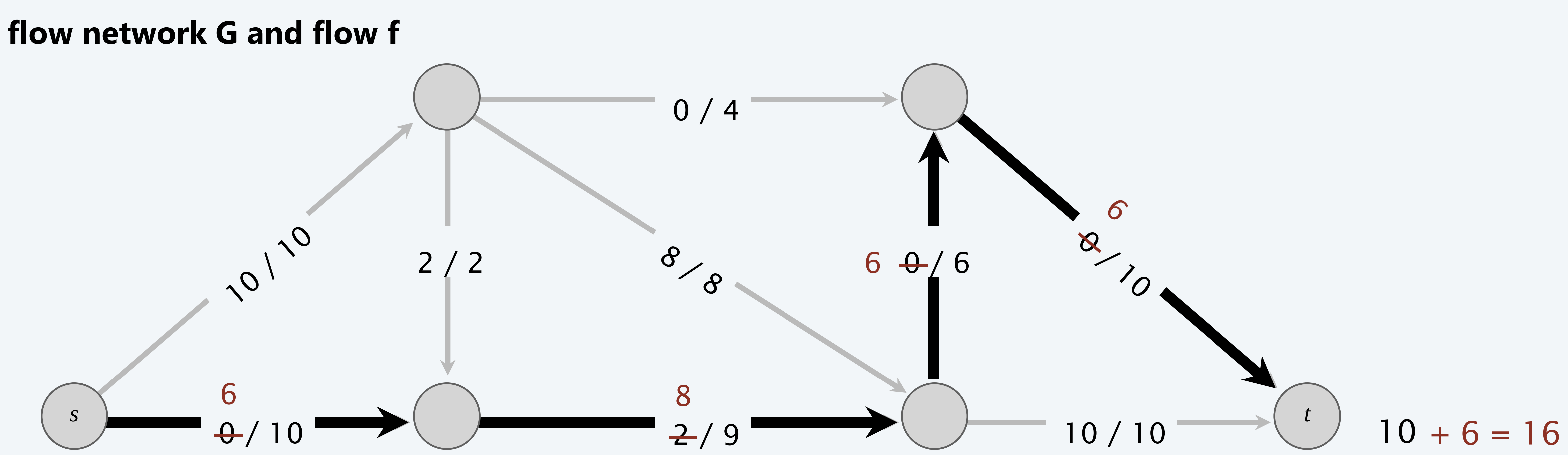
flow network G and flow f



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.



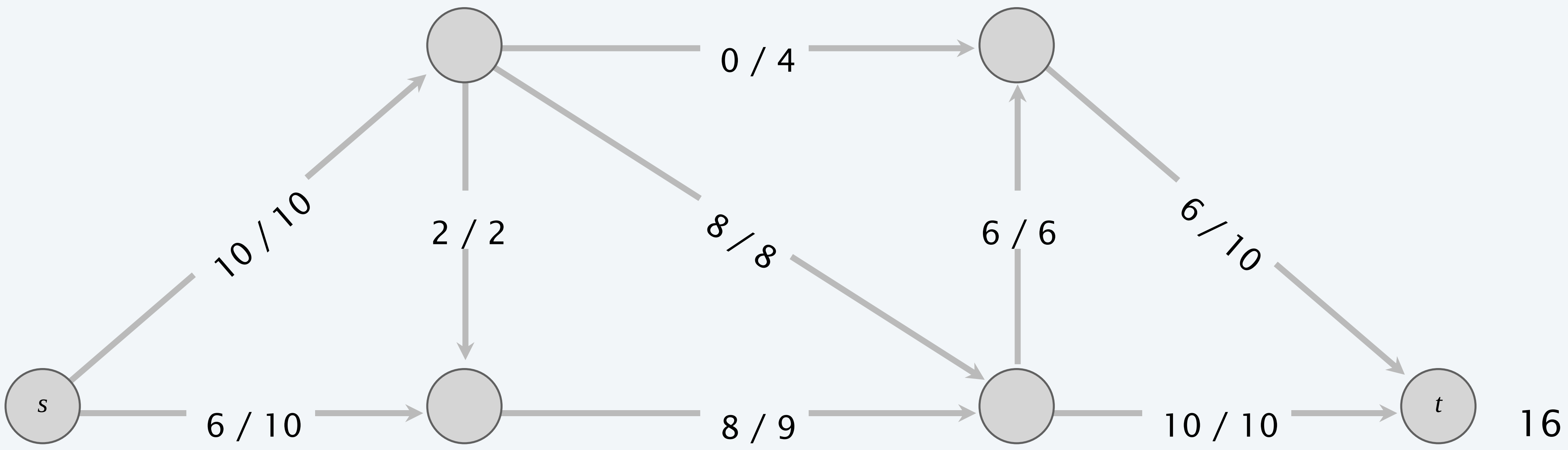
Toward a max-flow algorithm

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



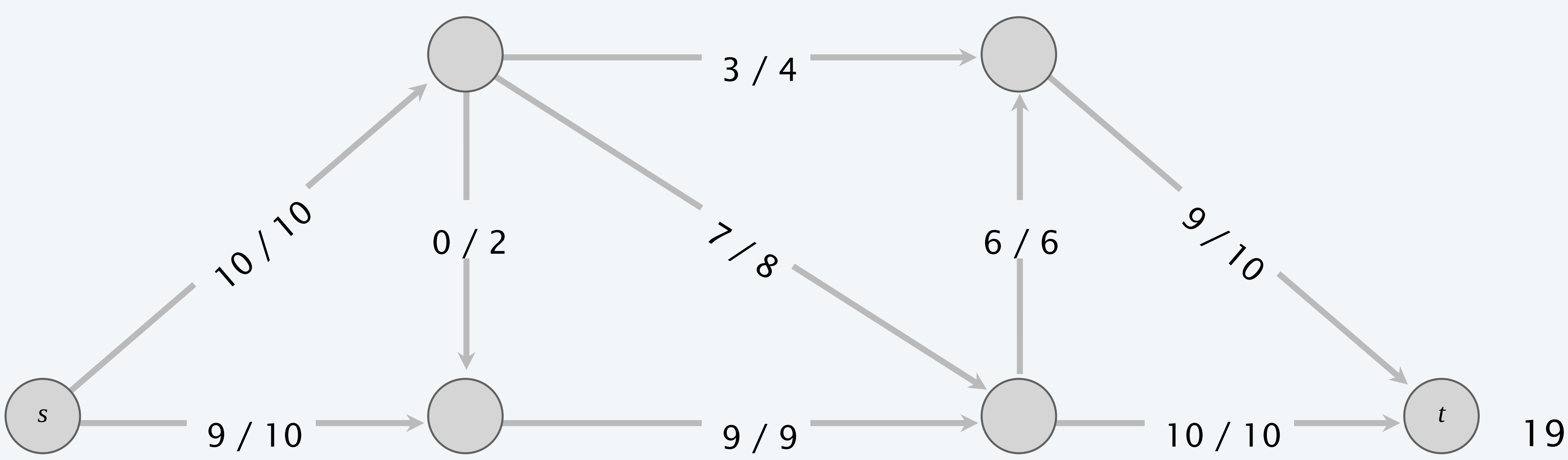
Toward a max-flow algorithm

Greedy algorithm.

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



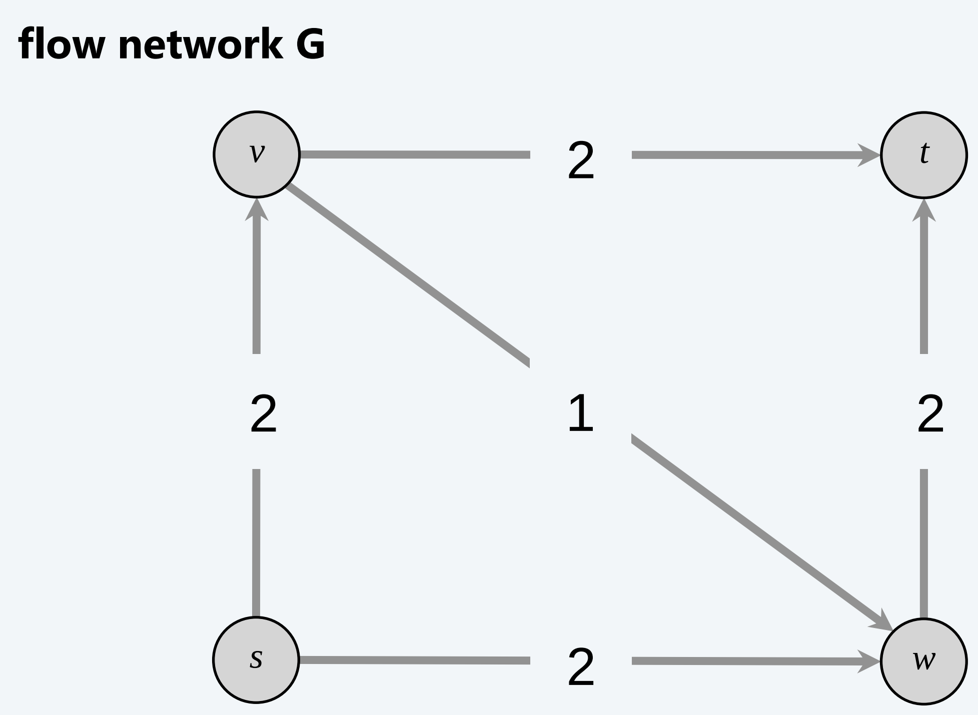
Why the greedy algorithm fails

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 $s \to v \to w \to t$ as first path.



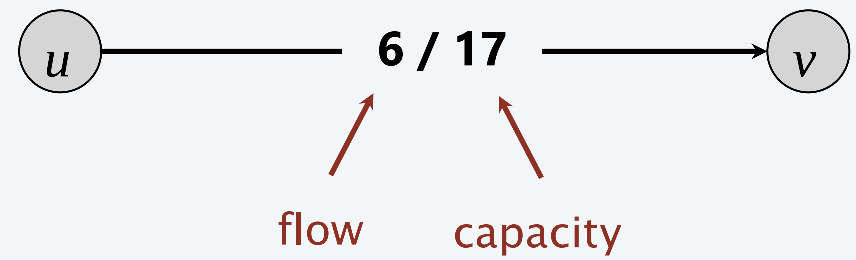
Bottom line. Need some mechanism to "undo" a bad decision.

Residual network

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

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

original flow network G



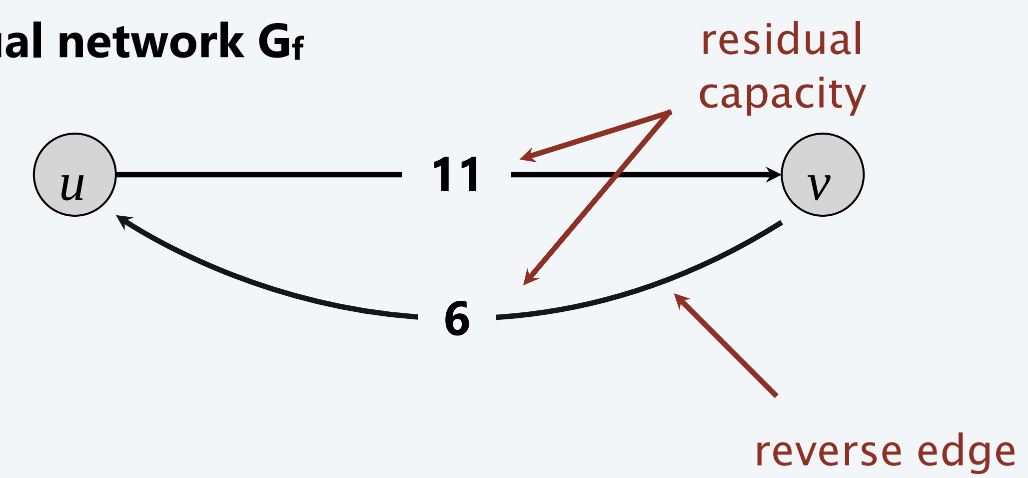
Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "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

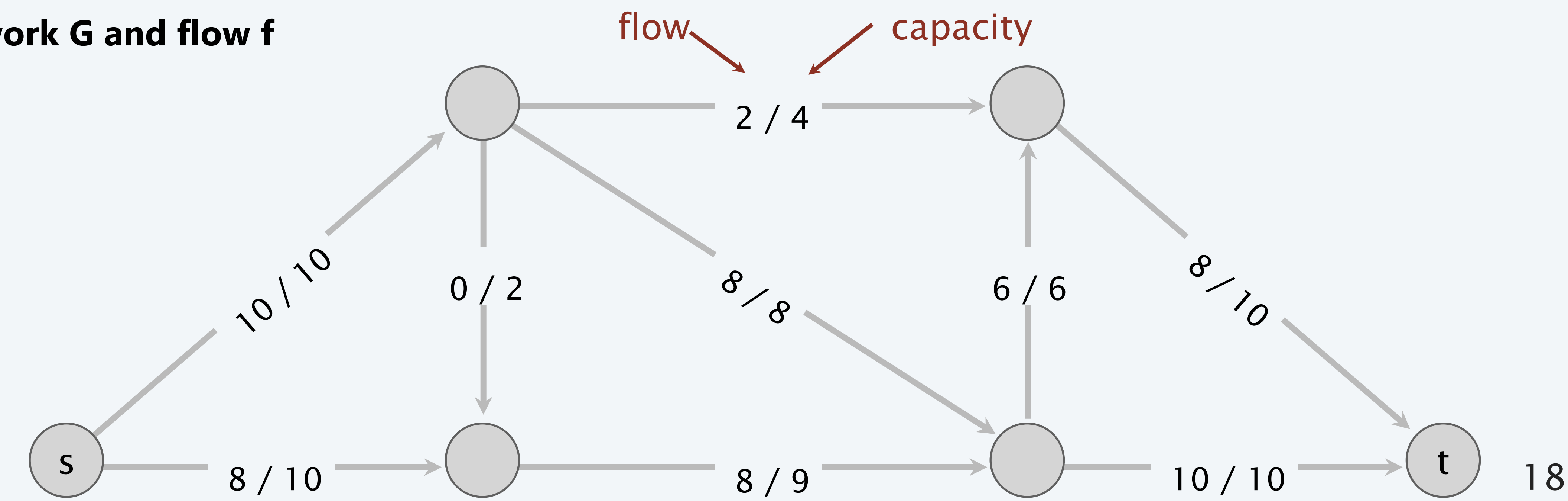
where flow on a reverse edge negates flow on corresponding forward edge

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

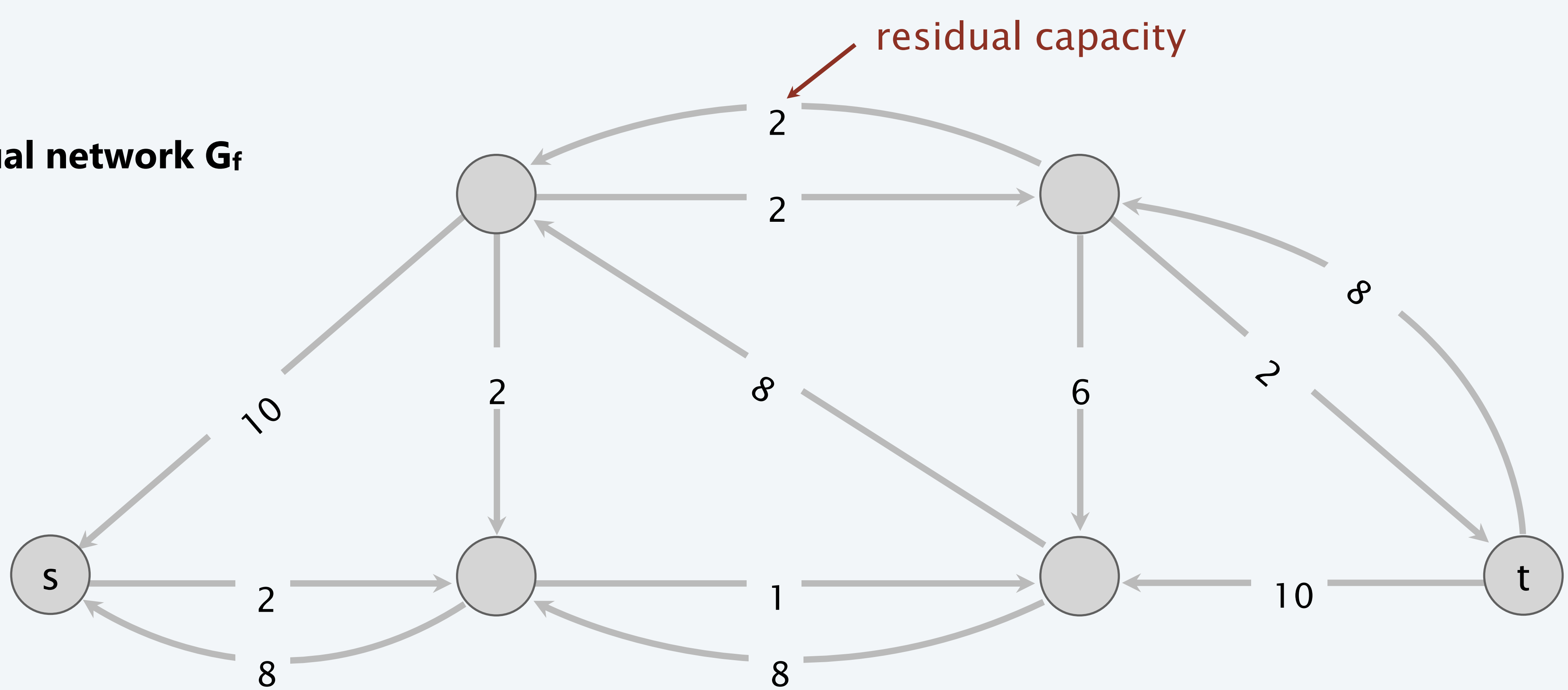
- $E_f = \{e : f(e) < c(e)\} \cup \{e : f(e^{\text{reverse}}) > 0\}$.
- Key property: f' is a flow in G_f iff $f + f'$ is a flow in G.

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 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)

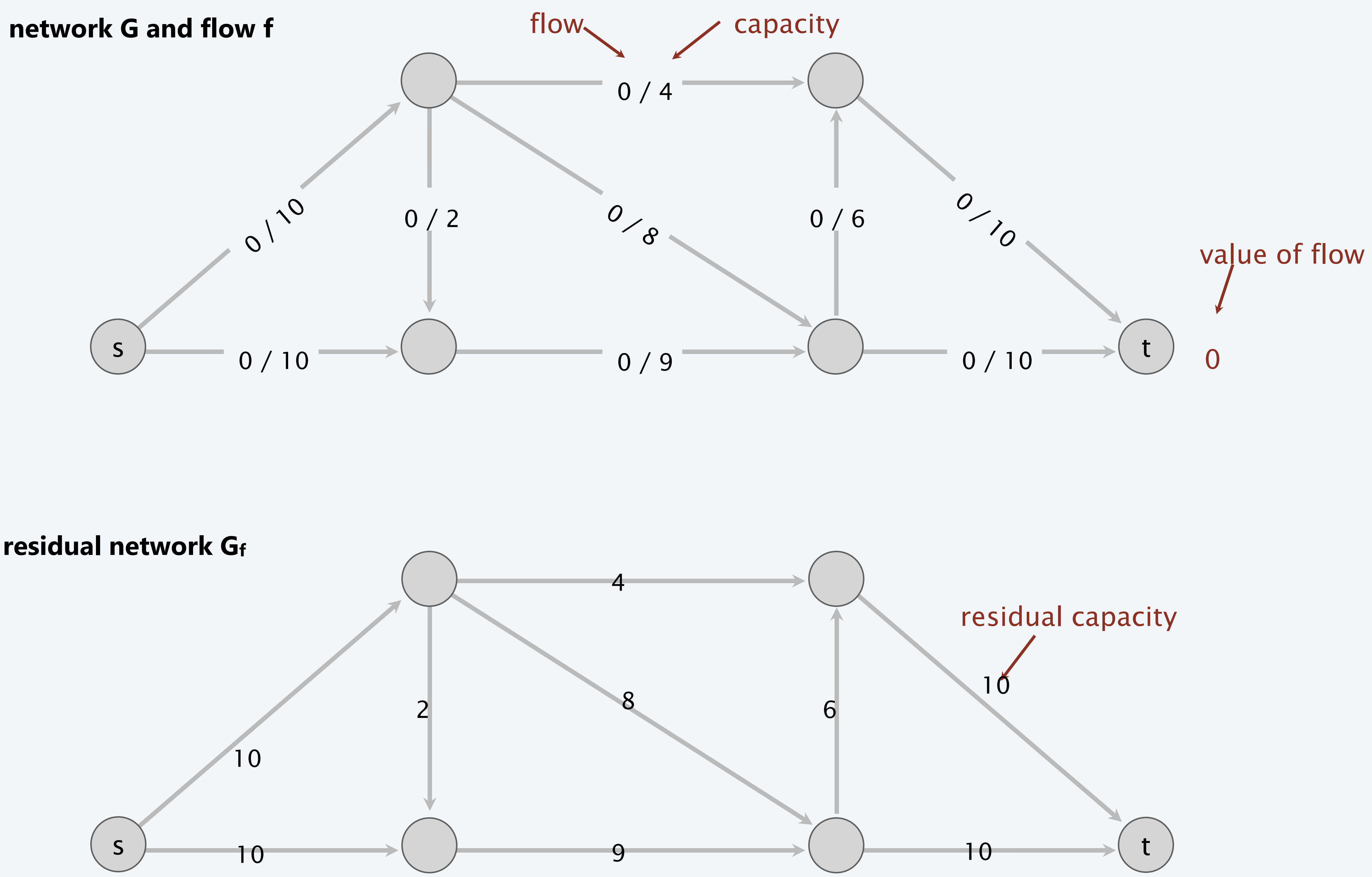
  $f \leftarrow$ AUGMENT(f, c, P).

  Update G_f.

RETURN f.

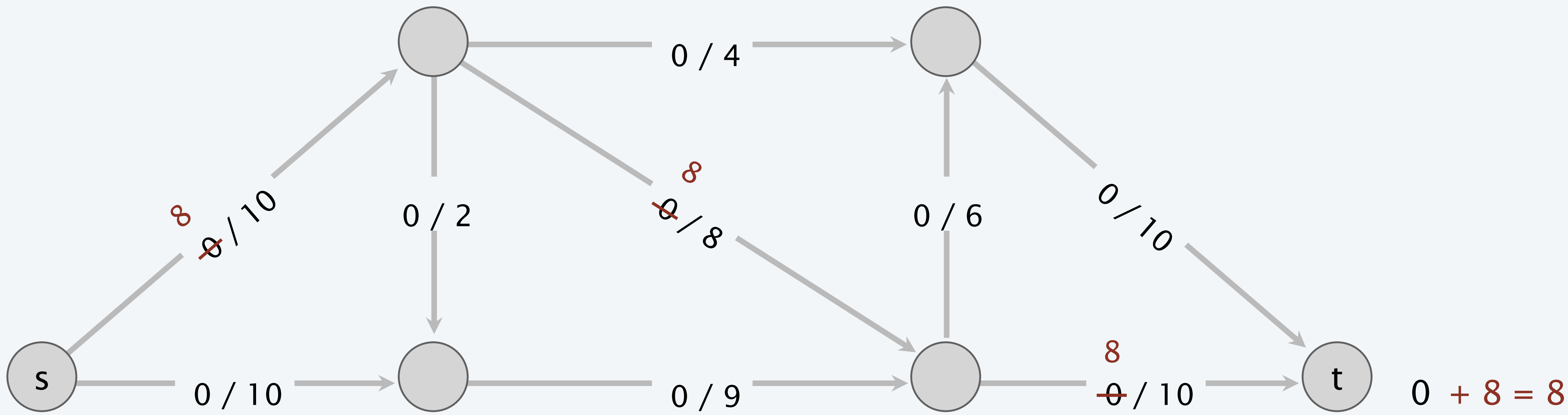
augmenting path

Ford–Fulkerson algorithm demo

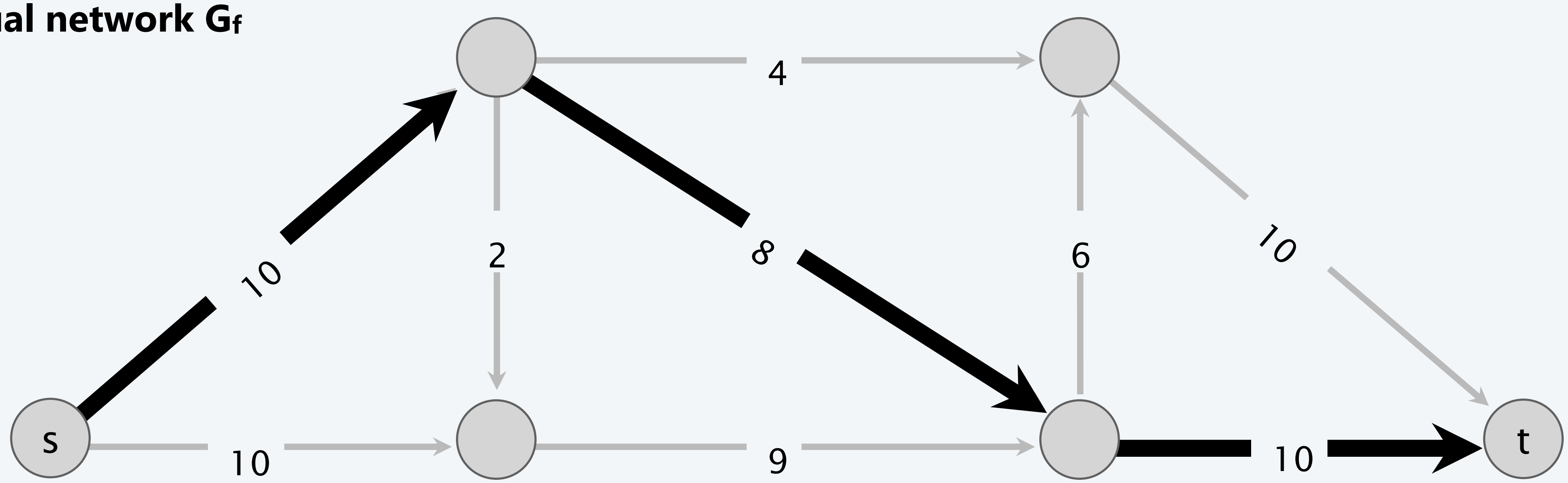


Ford–Fulkerson algorithm demo

network G and flow f

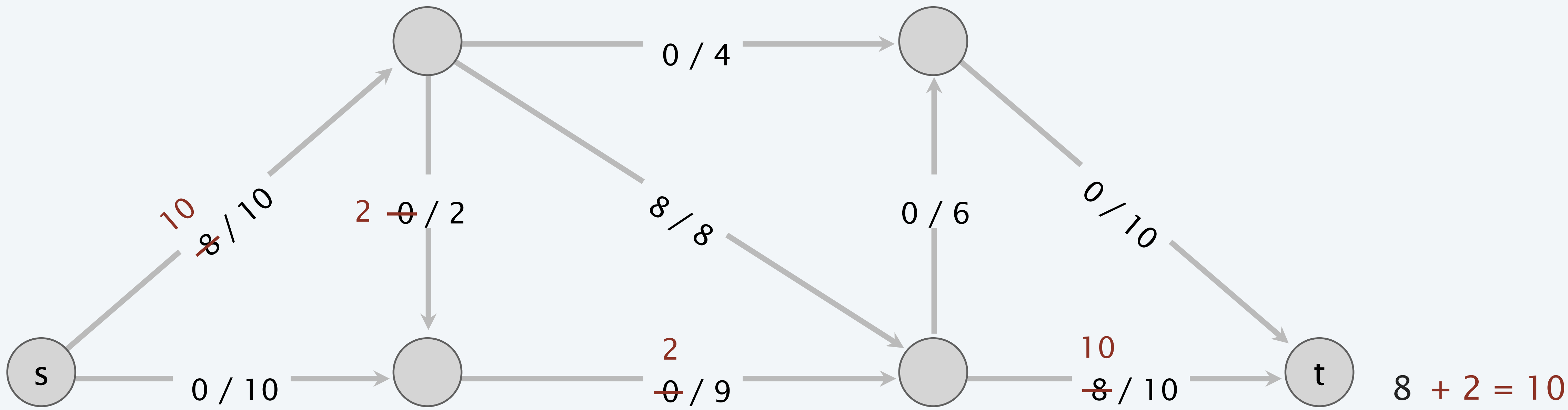


residual network G_f

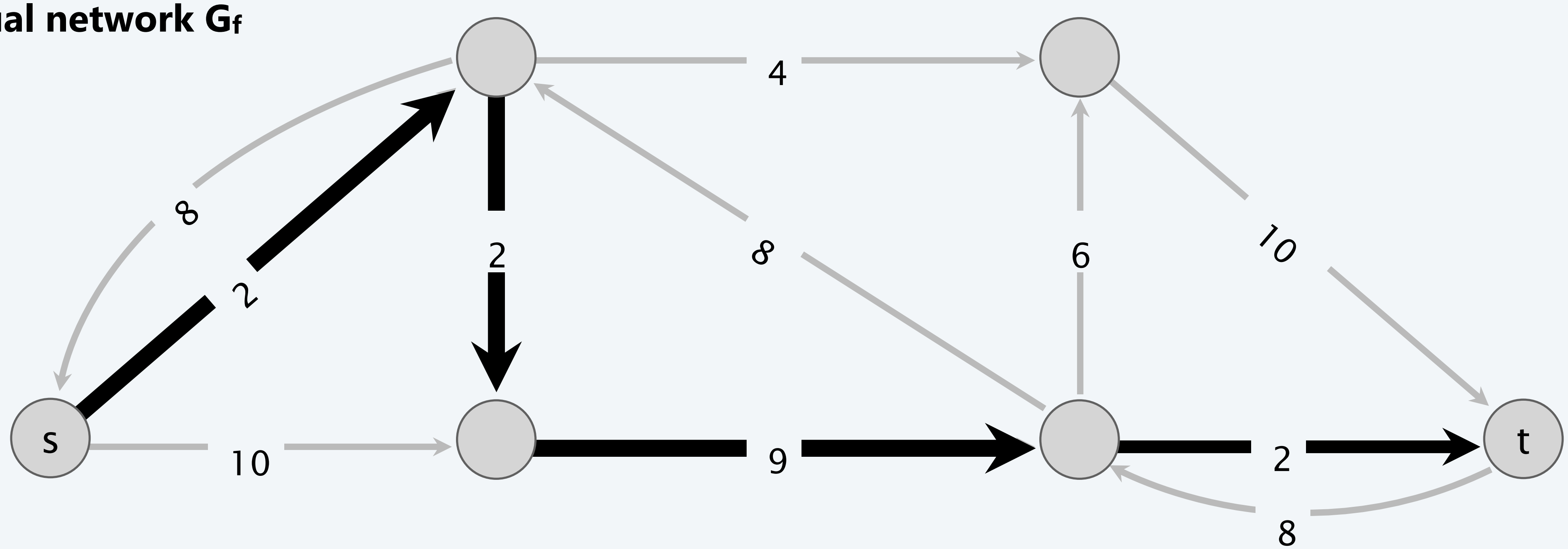


Ford–Fulkerson algorithm demo

network G and flow f

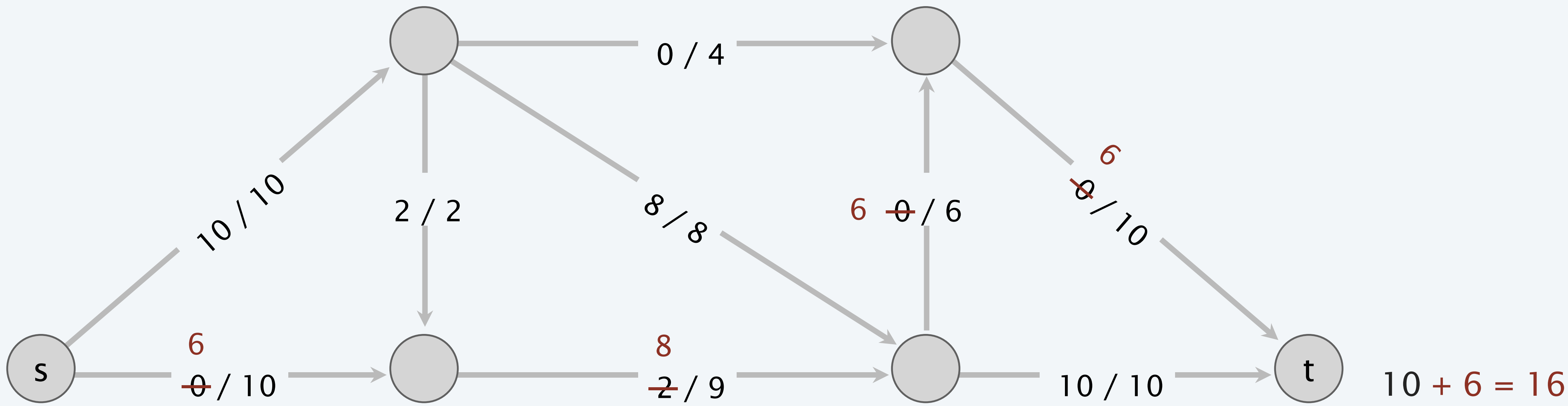


residual network G_f

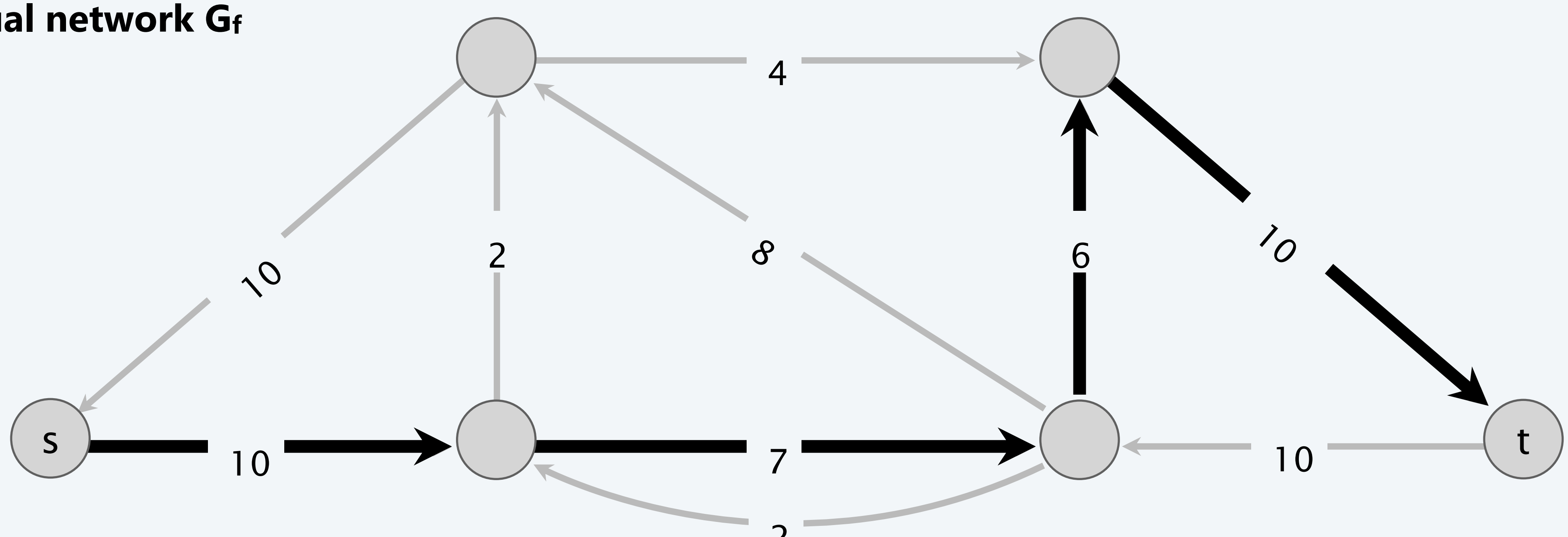


Ford–Fulkerson algorithm demo

network G and flow f

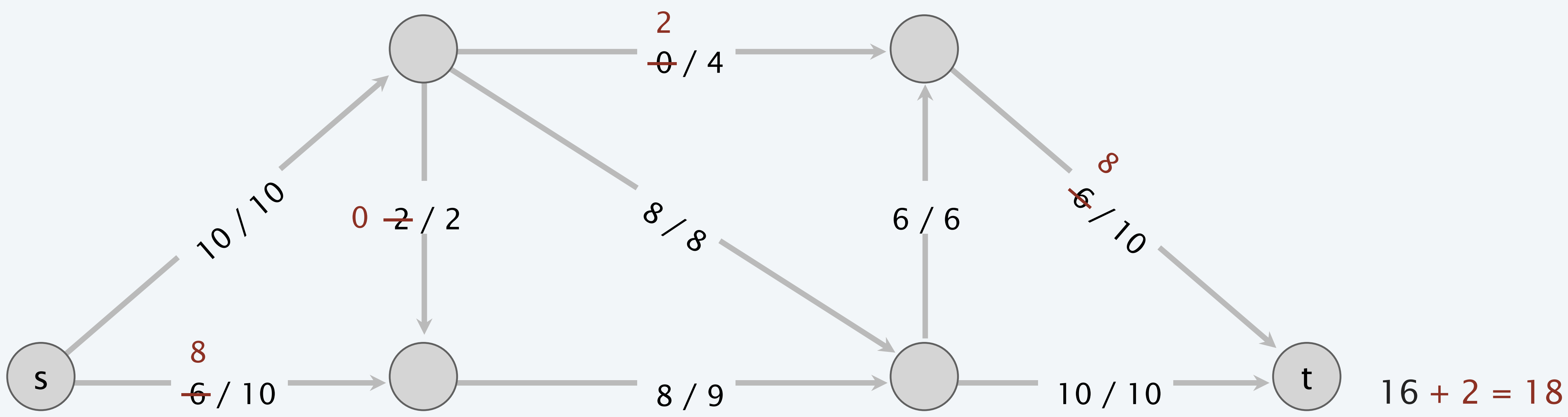


residual network G_f



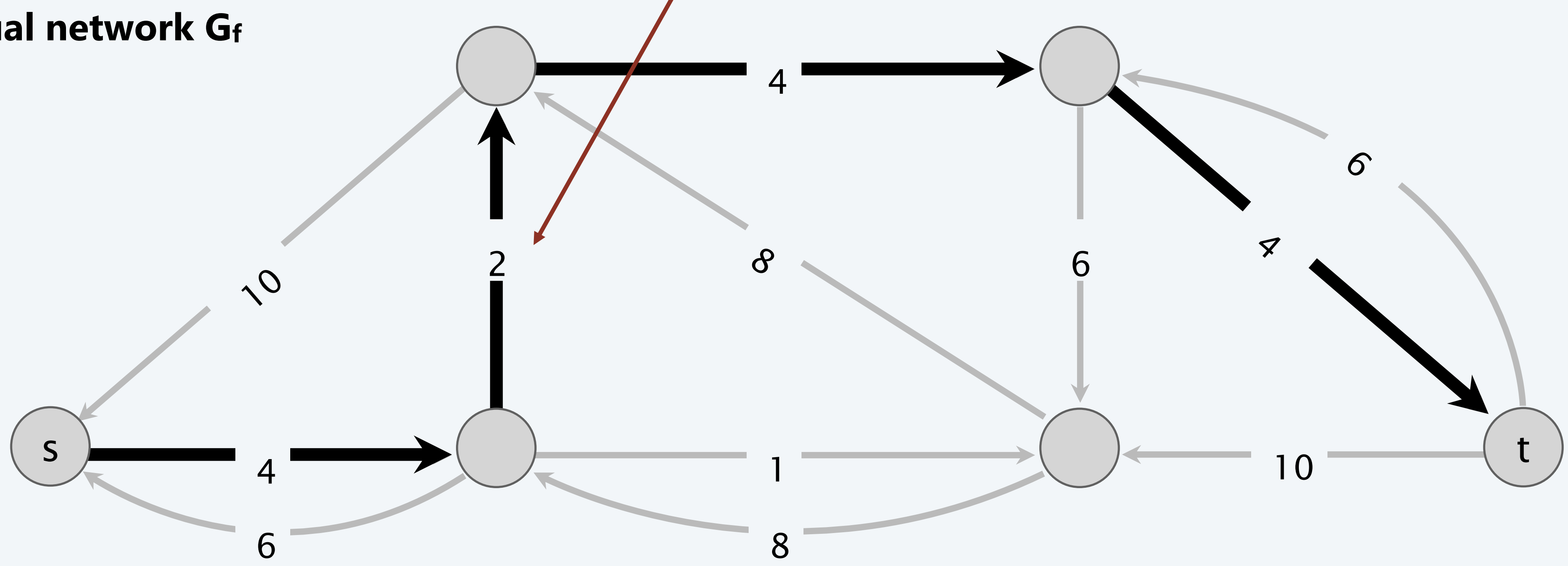
Ford–Fulkerson algorithm demo

network G and flow f



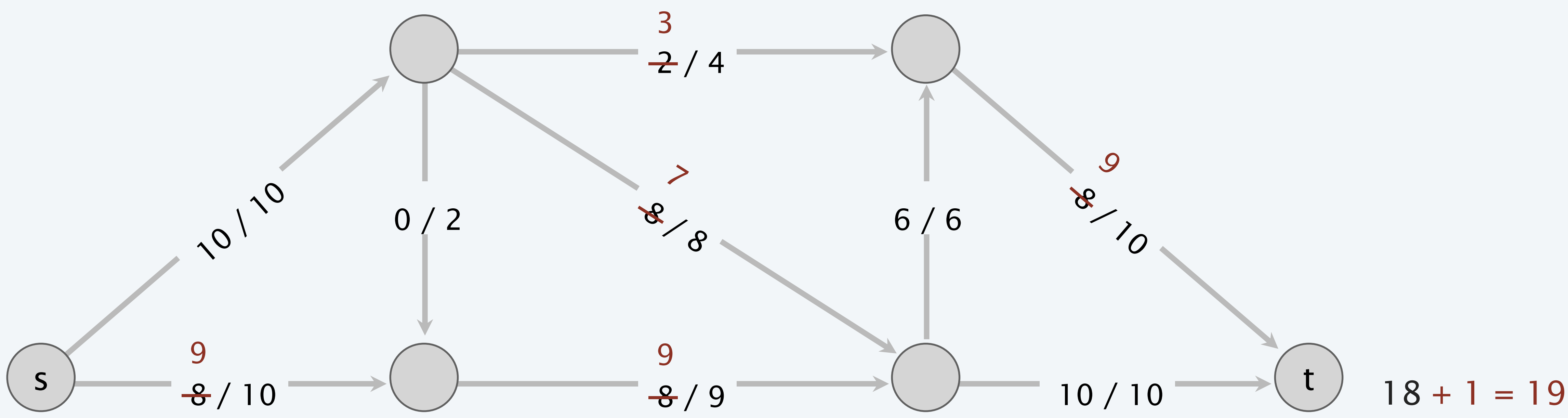
fixes mistake from
second augmenting path

residual network G_f

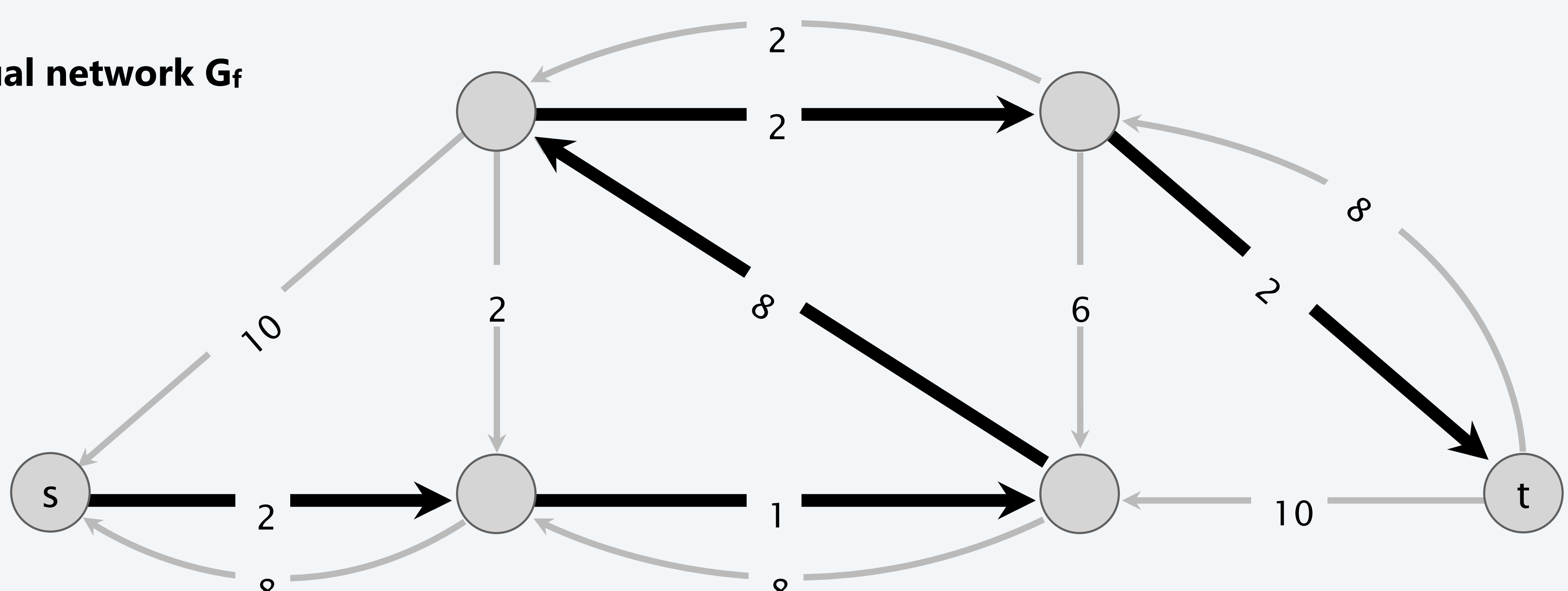


Ford–Fulkerson algorithm demo

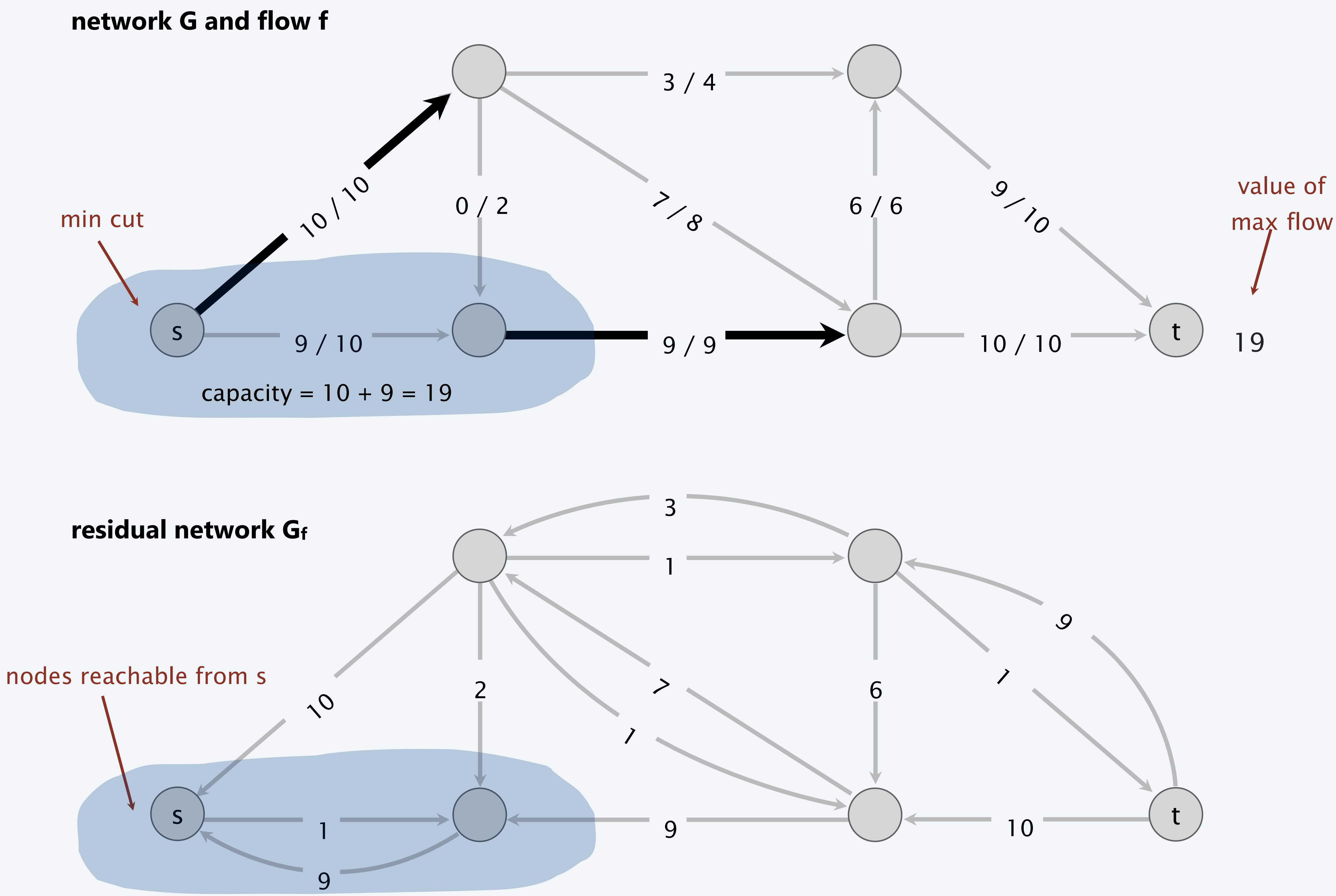
network G and flow f

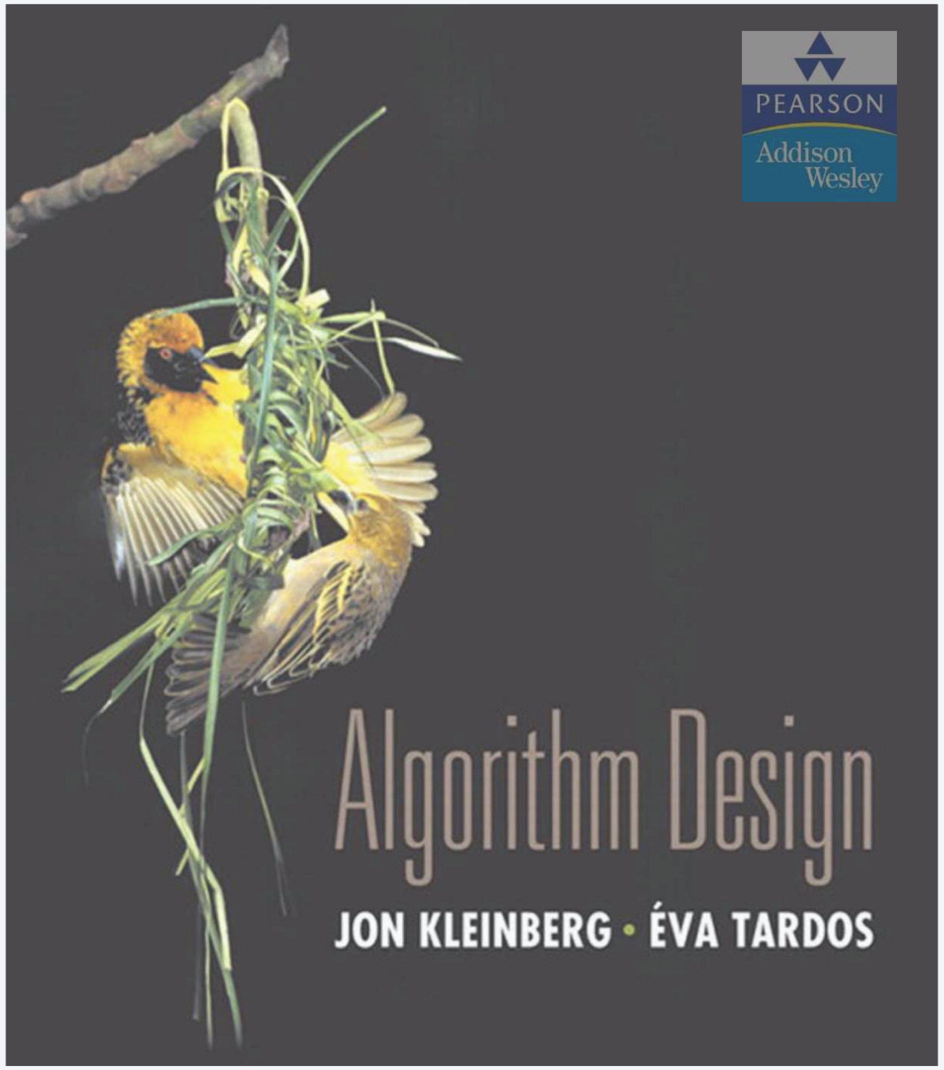


residual network G_f



Ford–Fulkerson algorithm demo





SECTION 7.2

7. NETWORK FLOW I

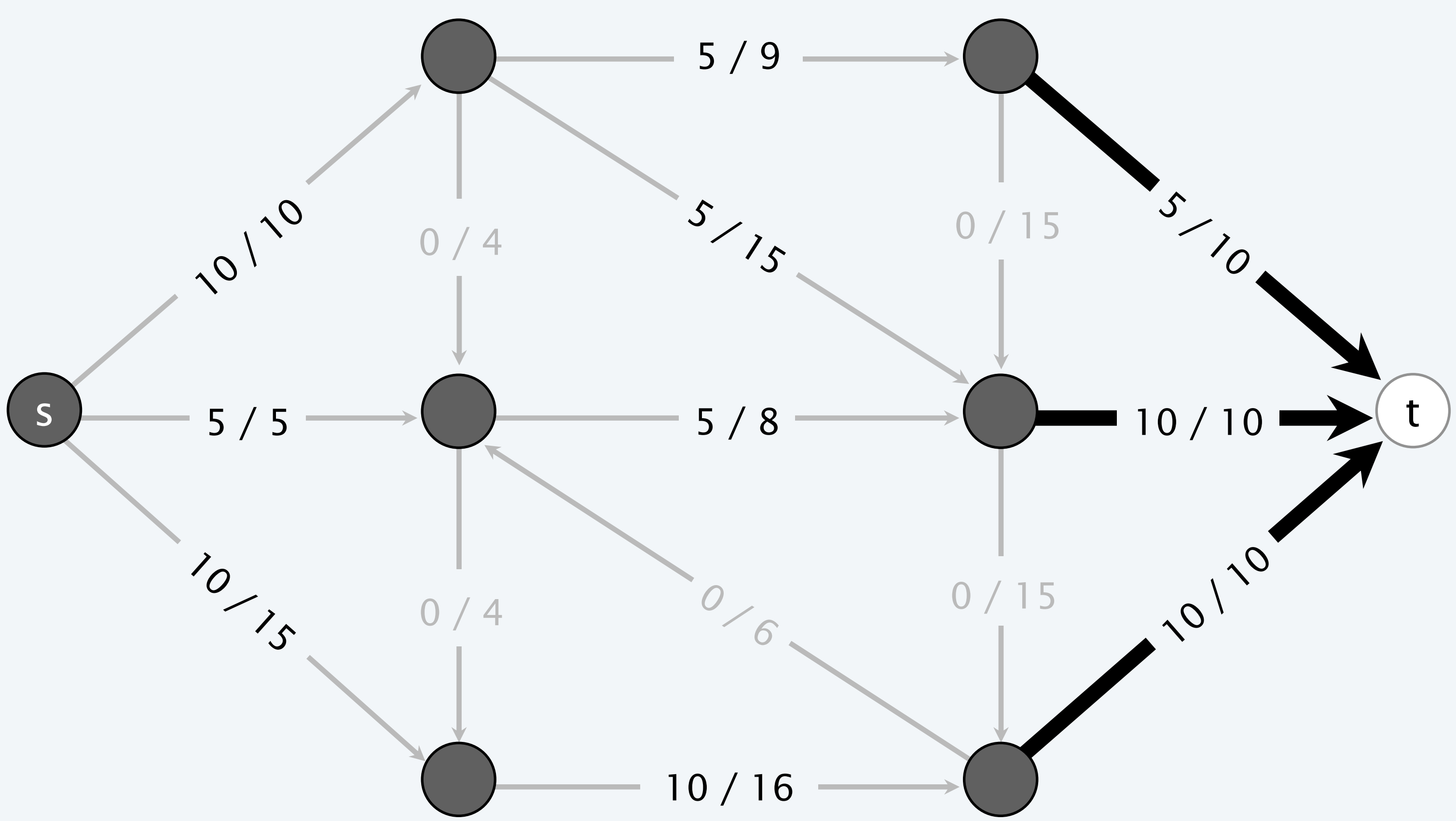
- *max-flow and min-cut problems*
- *Ford–Fulkerson algorithm*
- *max-flow min-cut theorem*
- *choosing good augmenting paths*

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

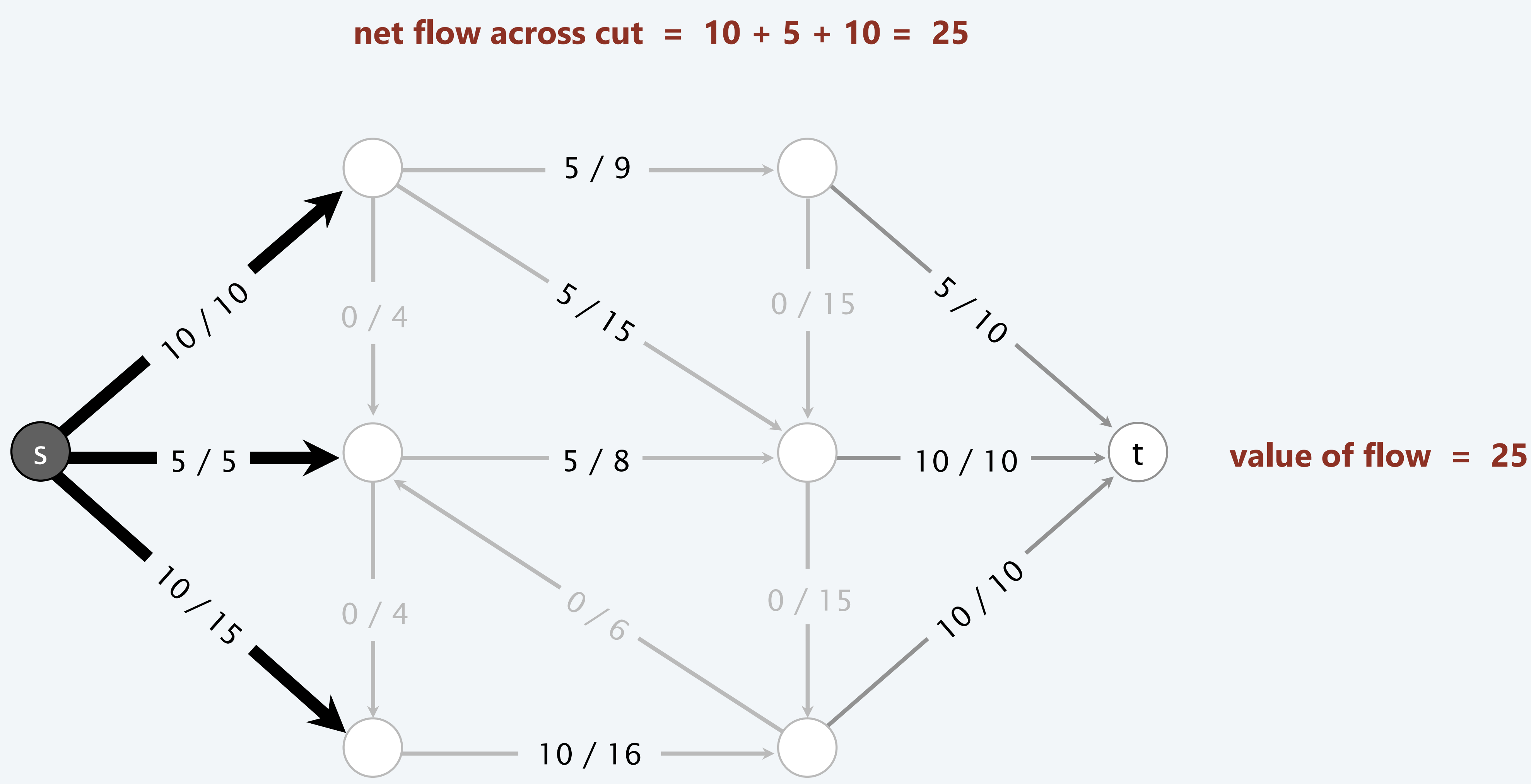


value of flow = 25

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).

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

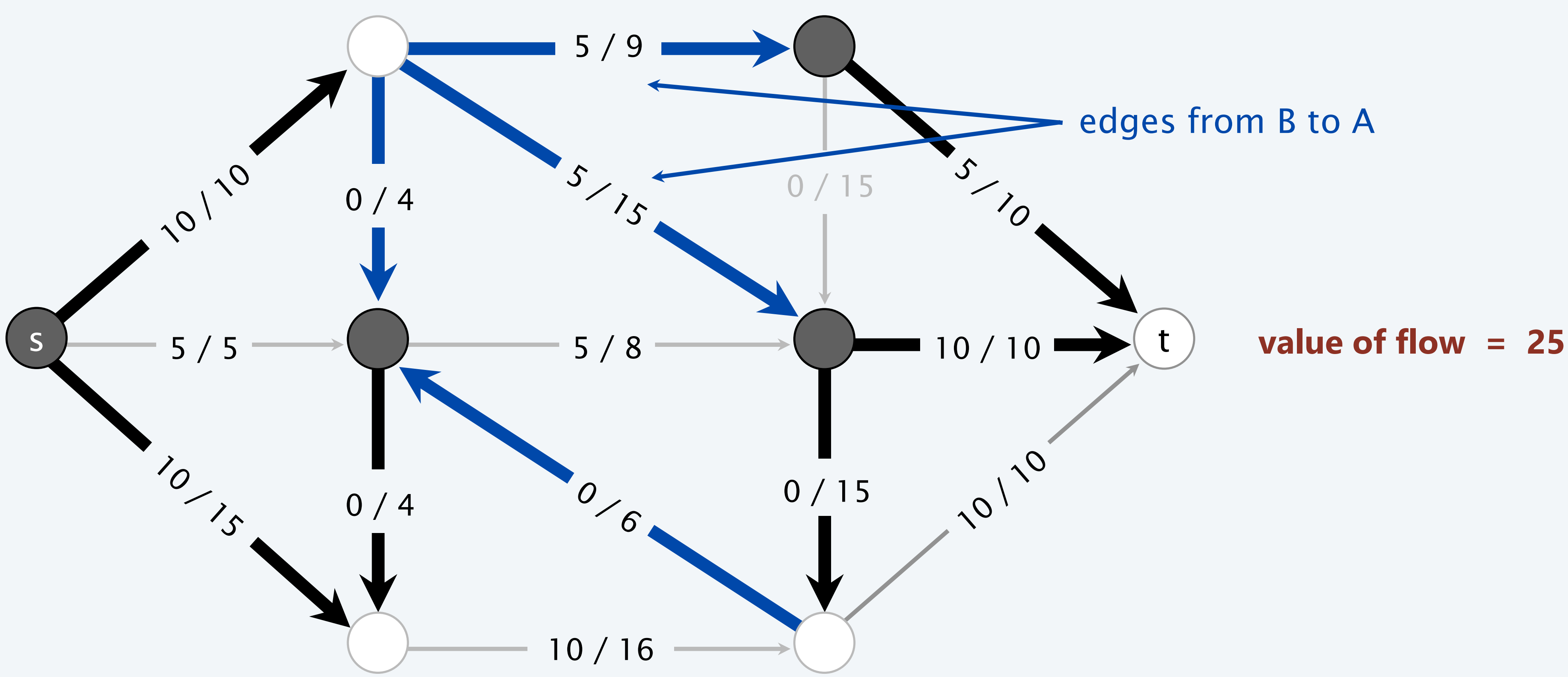


Relationship between flows and cuts

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).

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

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 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

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

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

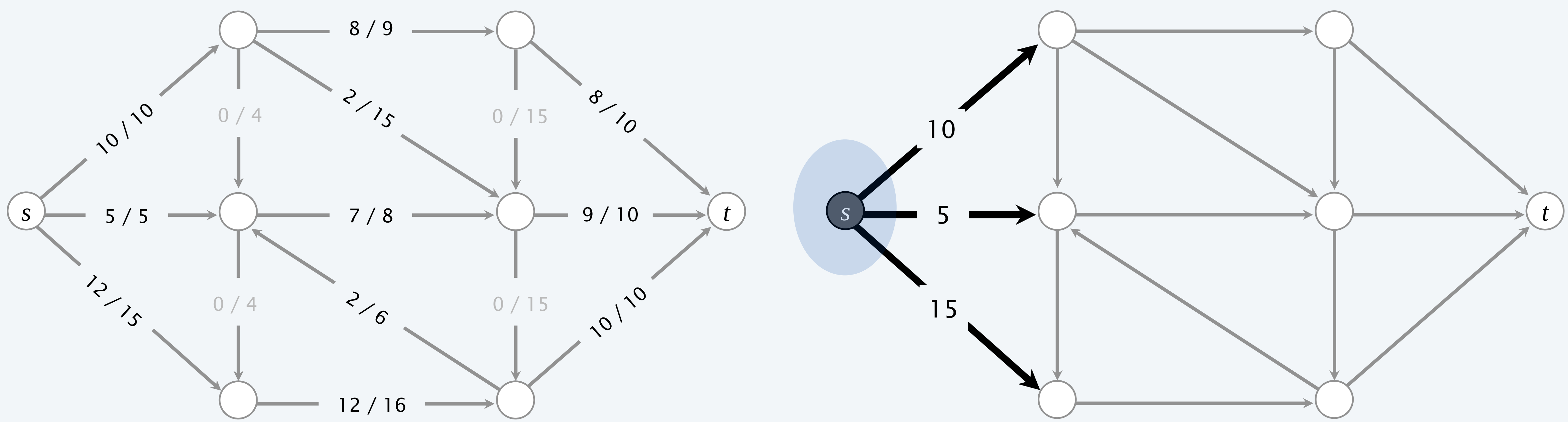
Relationship between flows and cuts

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



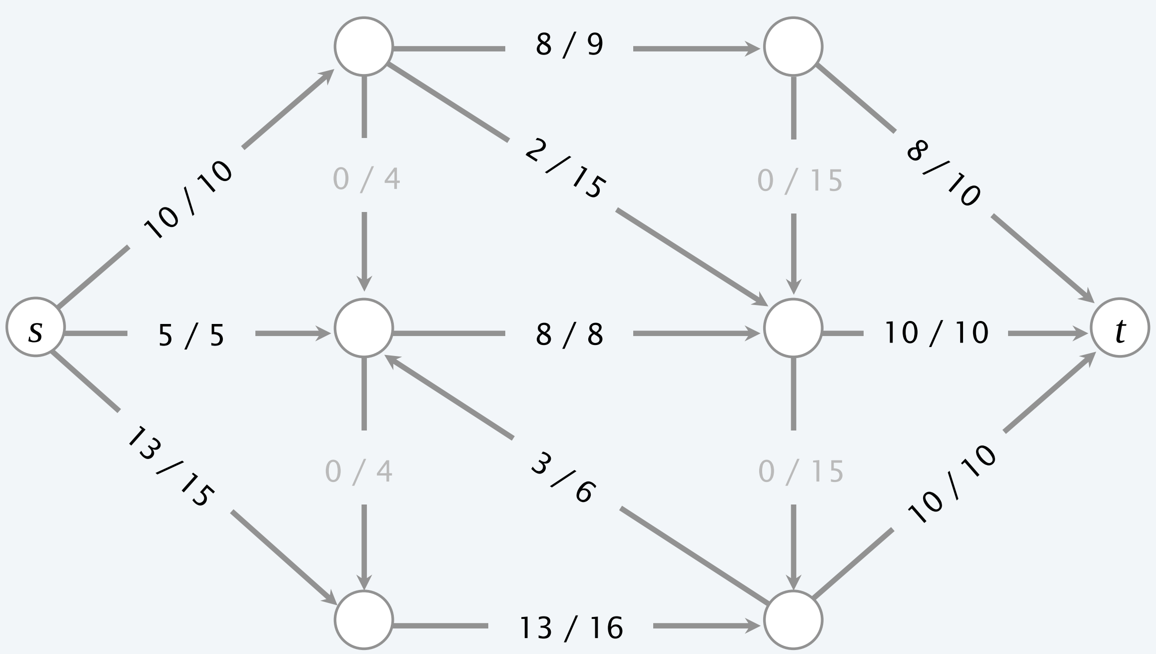
value of flow = 27 $\leq$ **capacity of cut = 30**

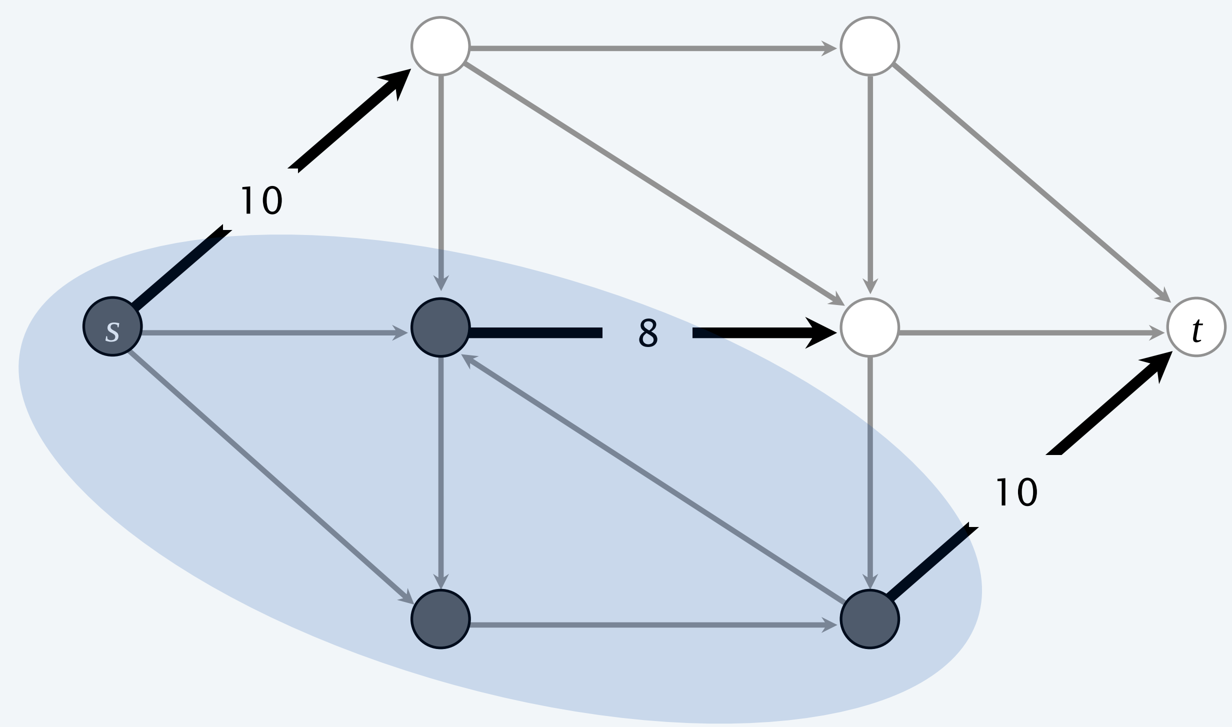
Certificate of optimality

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': $val(f') \leq cap(A, B) = val(f)$.
  weak duality
- 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

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 ⇒ 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 ⇒ 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 = set of nodes reachable from s in residual network G_f.
- By definition of A: $s \in A$.
- By definition of flow f: $t \notin A$.

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

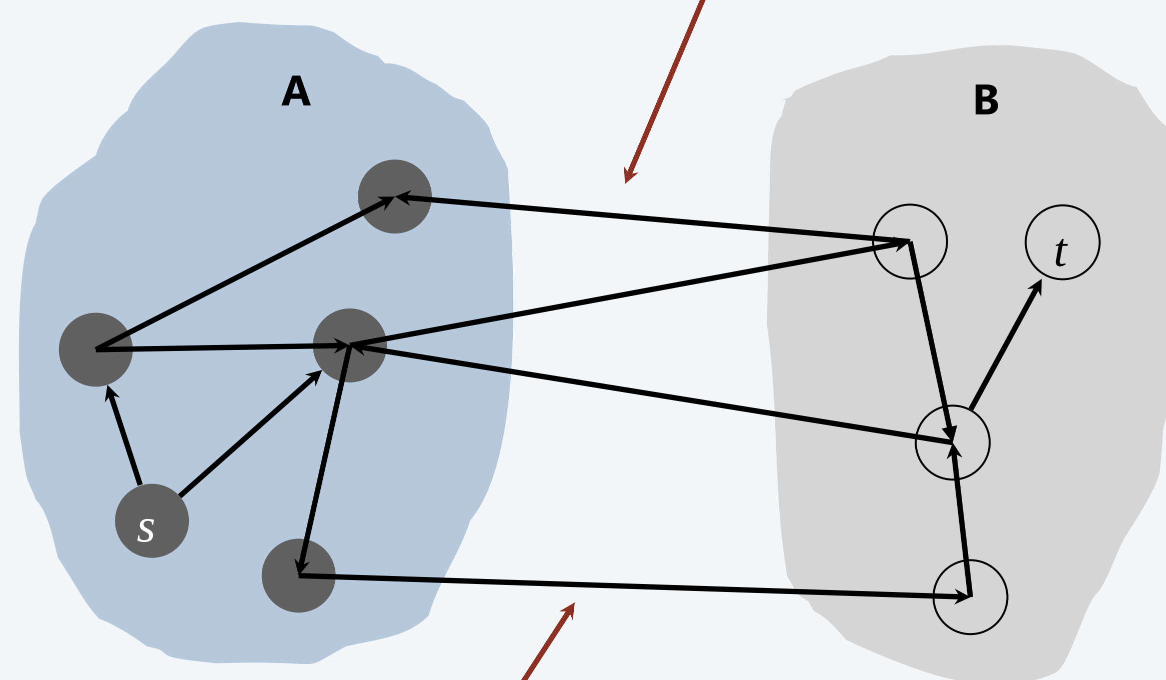
$$= \sum_{e \text{ out of } A} c(e) - 0$$

$$= cap(A, B) \quad \blacksquare$$

flow value lemma

edge $e = (v, w)$ with $v \in B$, $w \in A$
must have $f(e) = 0$

original flow network G



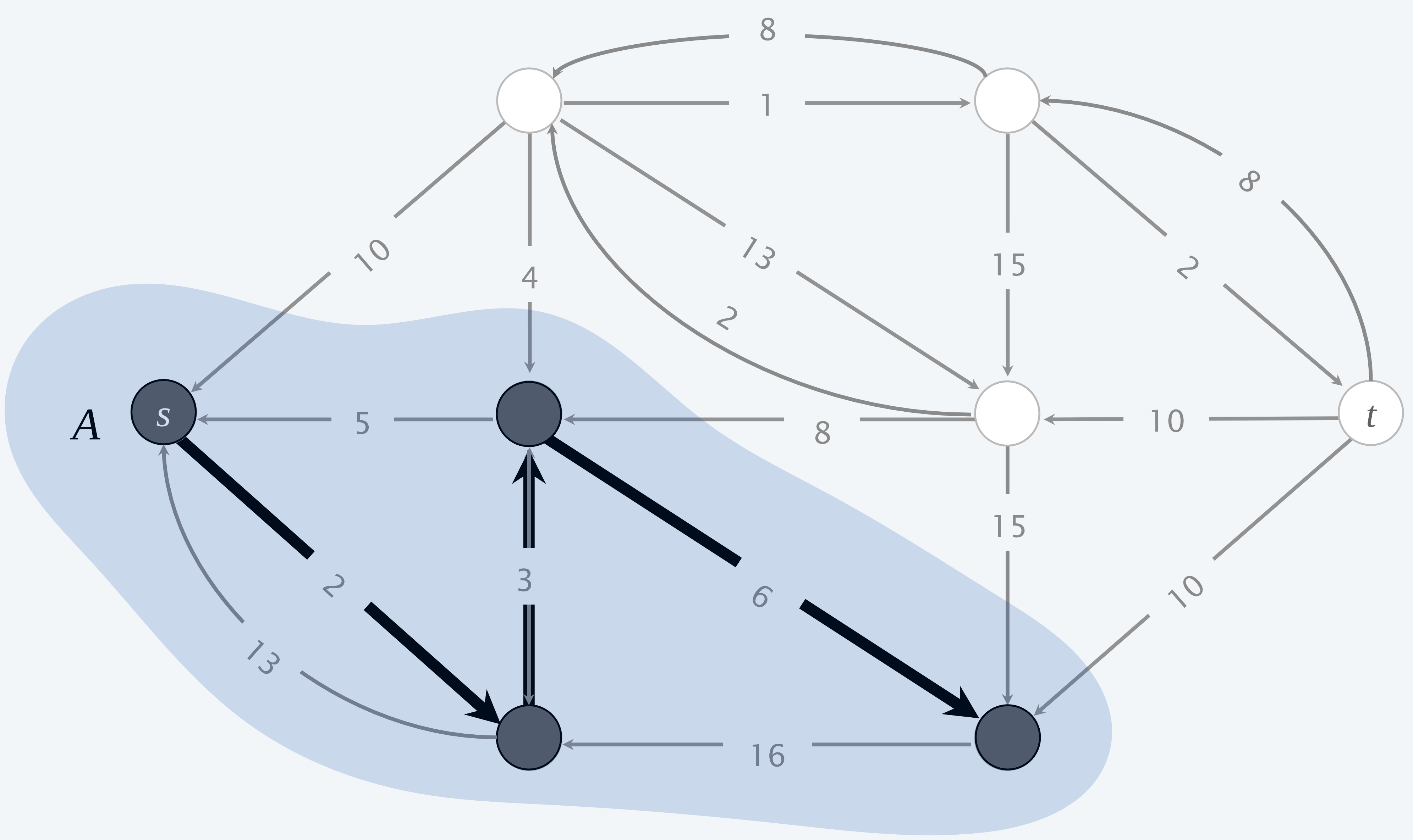
edge $e = (v, w)$ with $v \in A$, $w \in B$
must have $f(e) = c(e)$

Computing a minimum cut from a maximum flow

Theorem. Given any max flow f, can compute a min cut (A, B) in $O(m)$ time.

Pf. Let A = set of nodes reachable from s in residual network G_f. ▪

argument from previous slide implies that capacity of (A, B) = value of flow f



Analysis of Ford–Fulkerson algorithm (for integral capacities)

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 $val(f^*) \le n C$ 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 $O(m\ val(f^*)) = O(m n C)$.
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

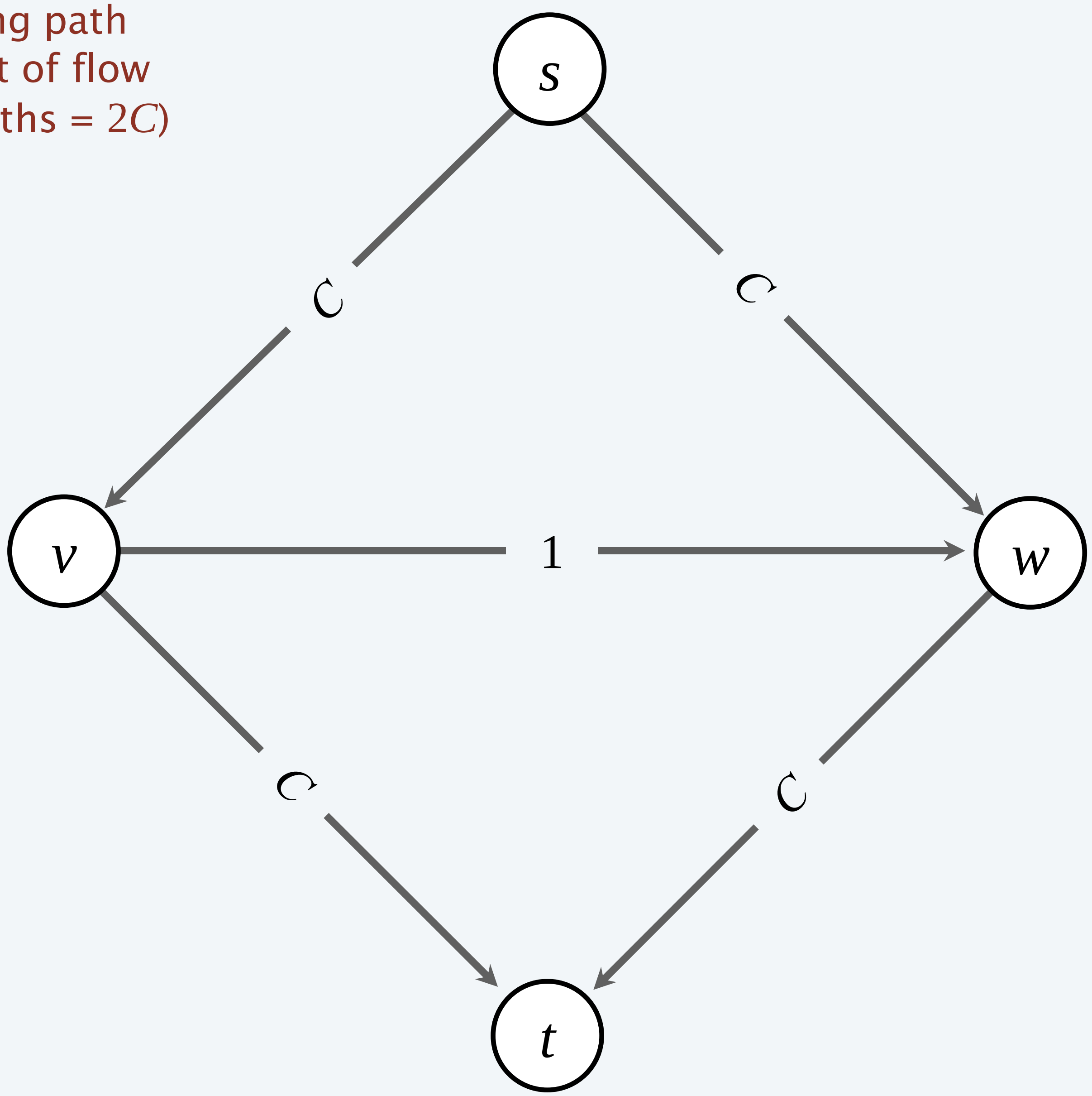
Q. Is generic Ford–Fulkerson algorithm poly-time in input size?

m, n, and $\log C$

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

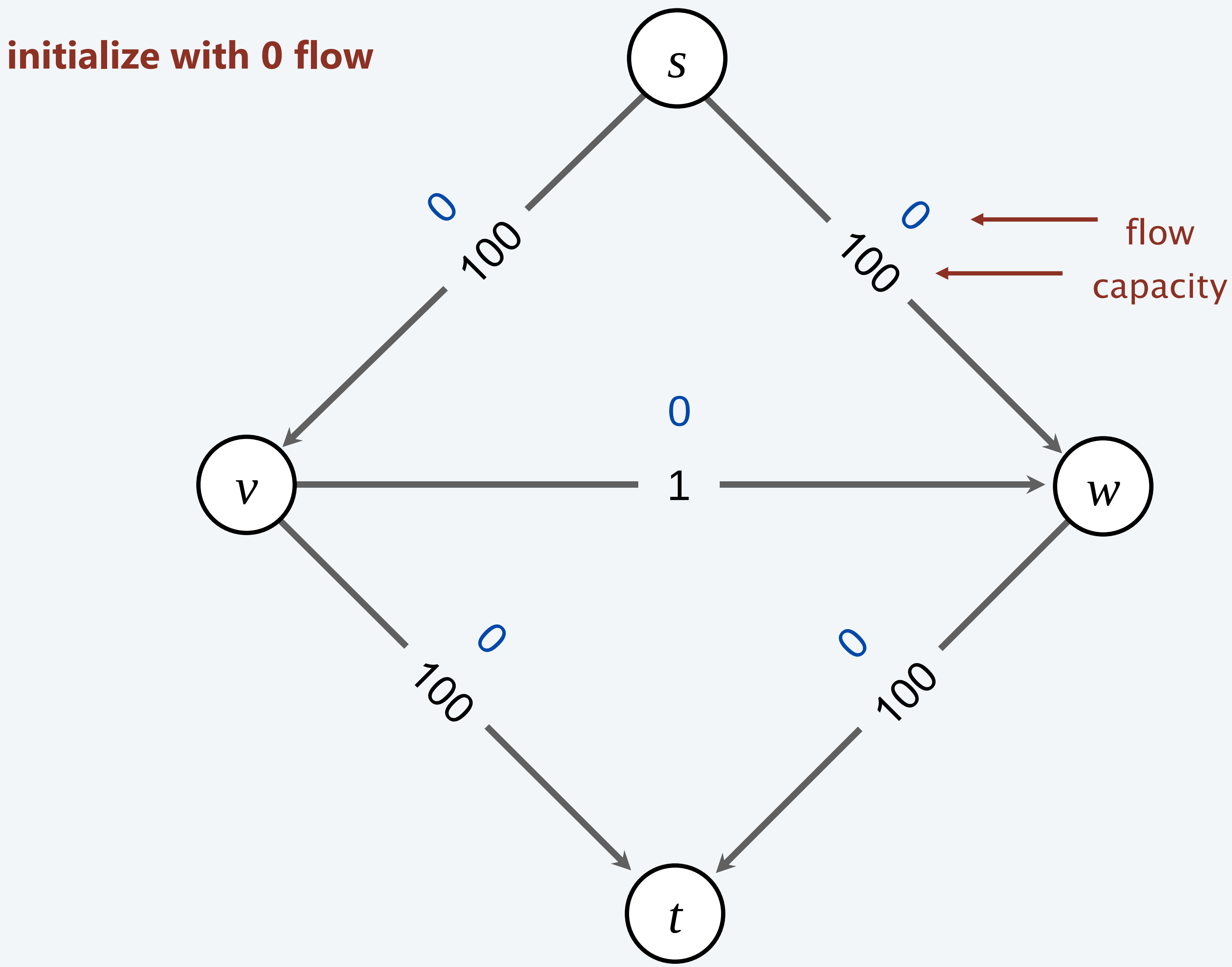
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- $s \to v \to w \to t$
- $s \to w \to v \to t$
- ...
- $s \to v \to w \to t$
- $s \to w \to v \to t$

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



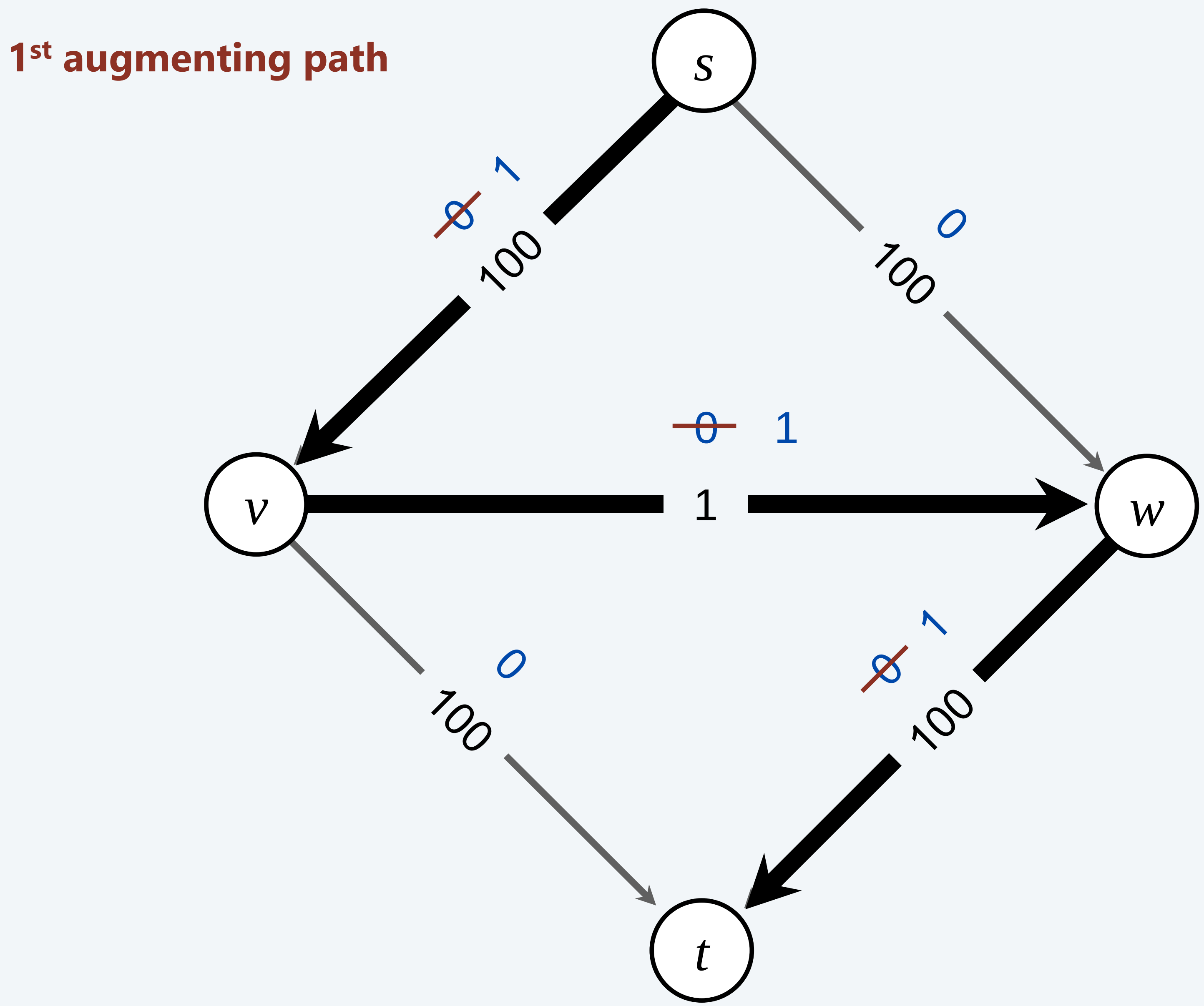
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



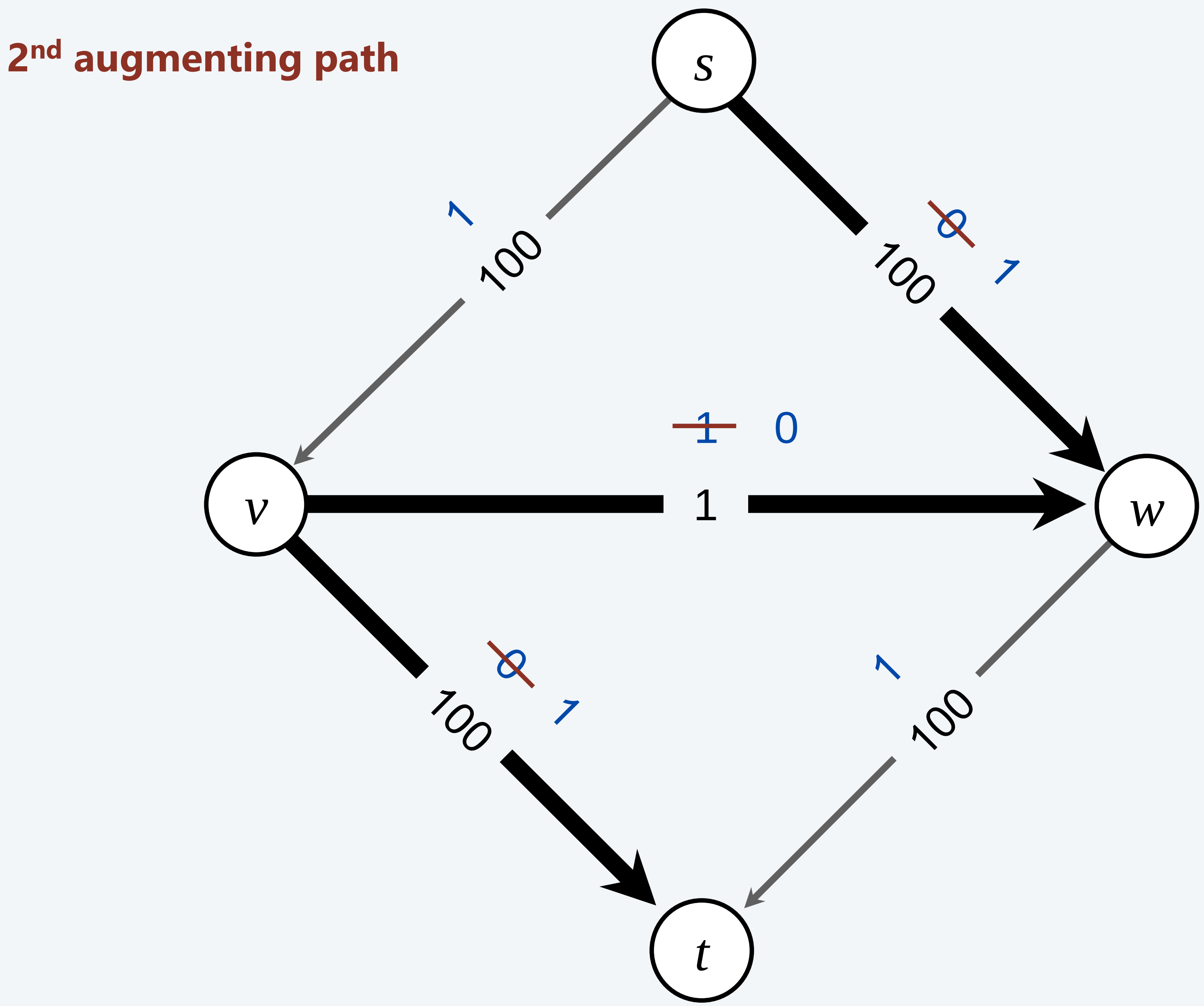
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



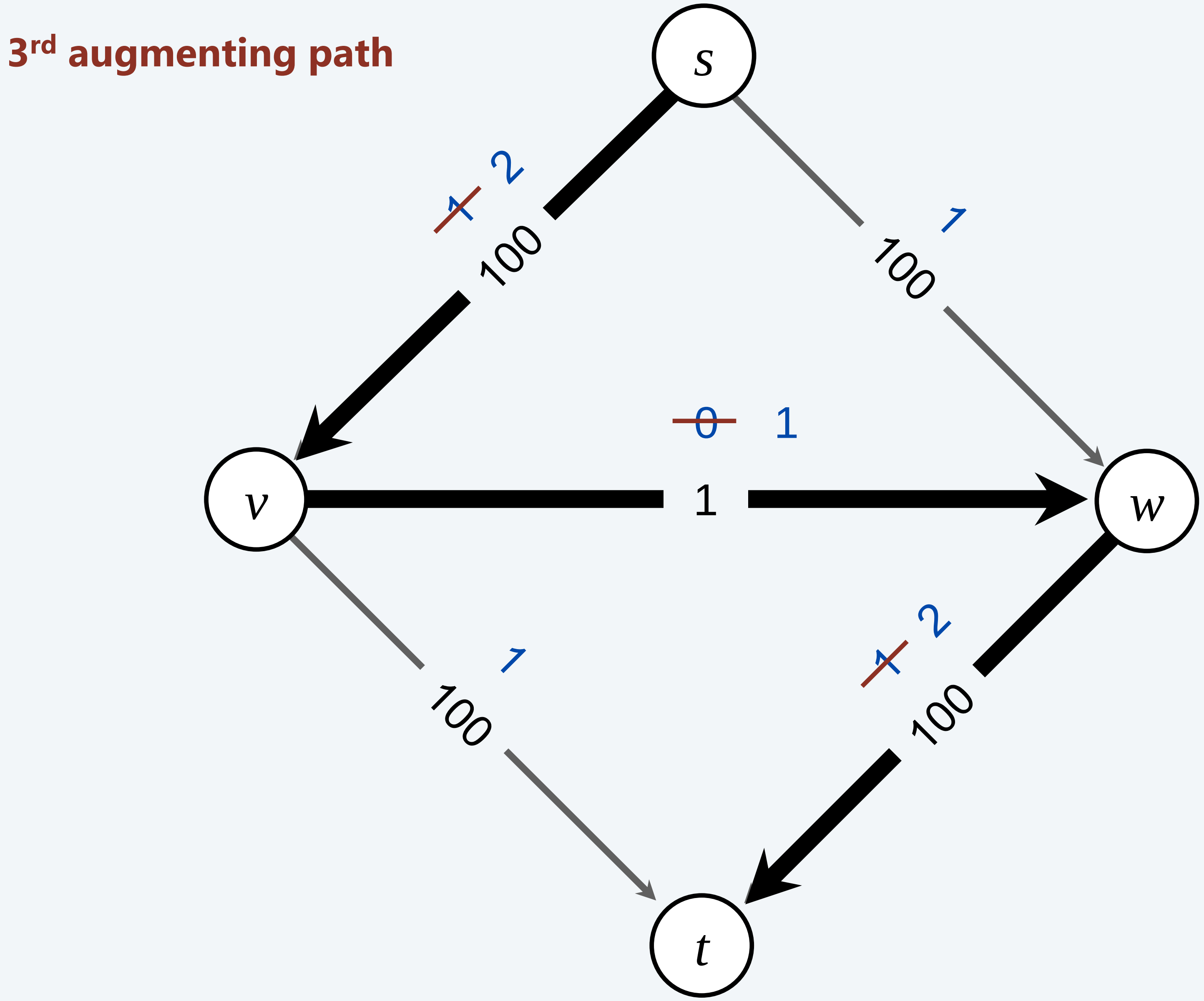
Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.



Ford–Fulkerson algorithm: exponential-time example

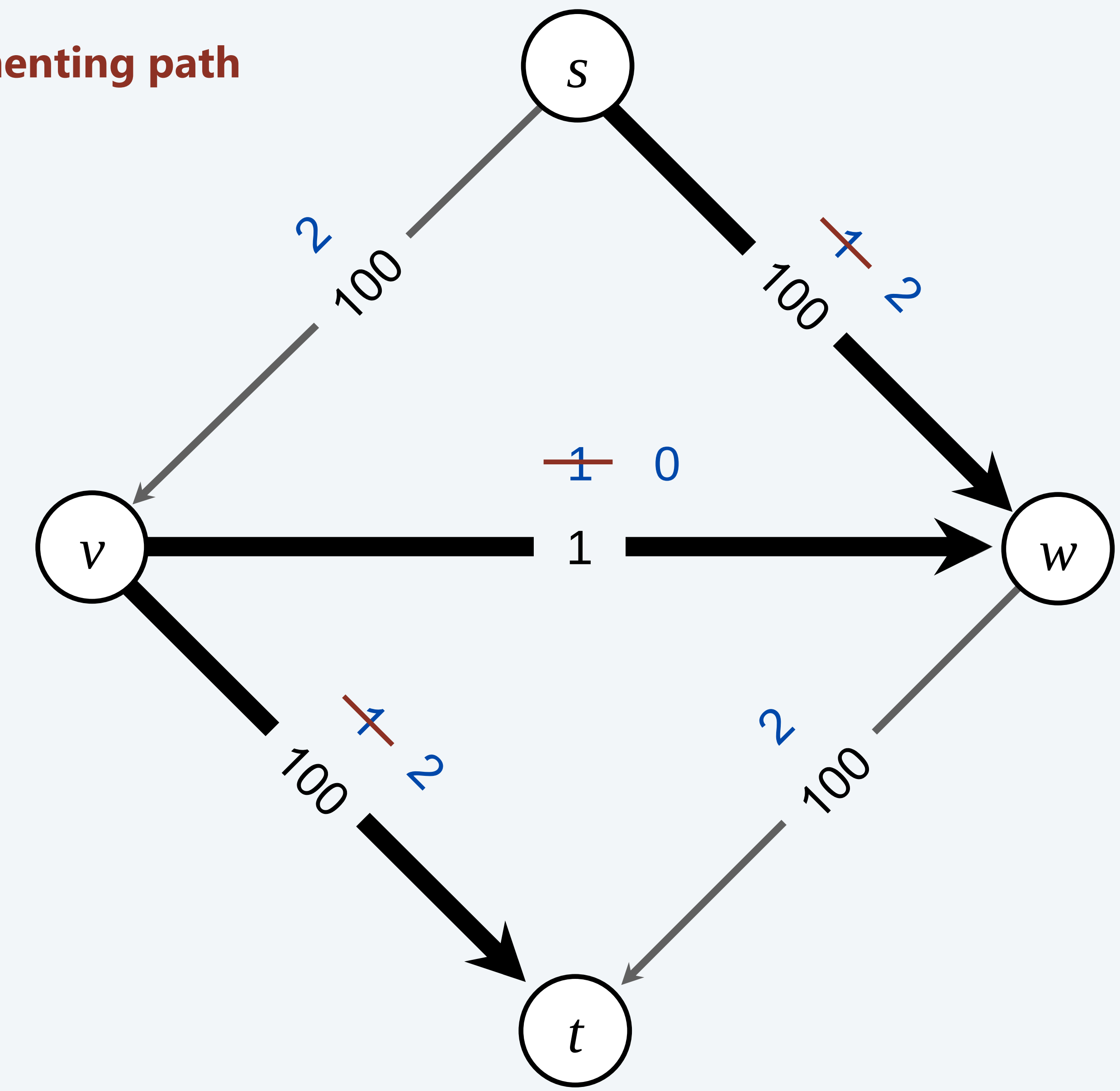
Bad news. Number of augmenting paths can be exponential in input size.



Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.

4th augmenting path

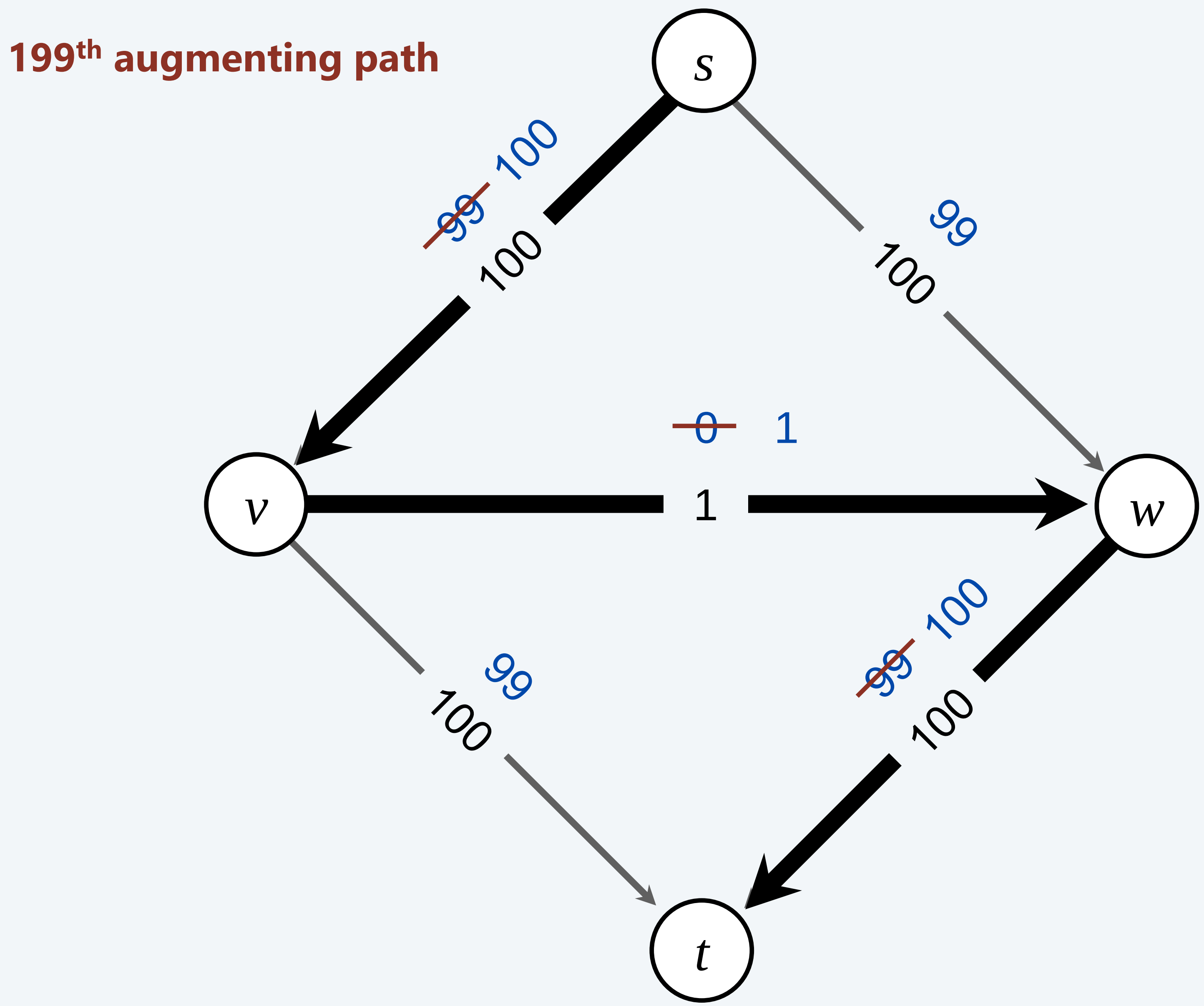


Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.

Ford–Fulkerson algorithm: exponential-time example

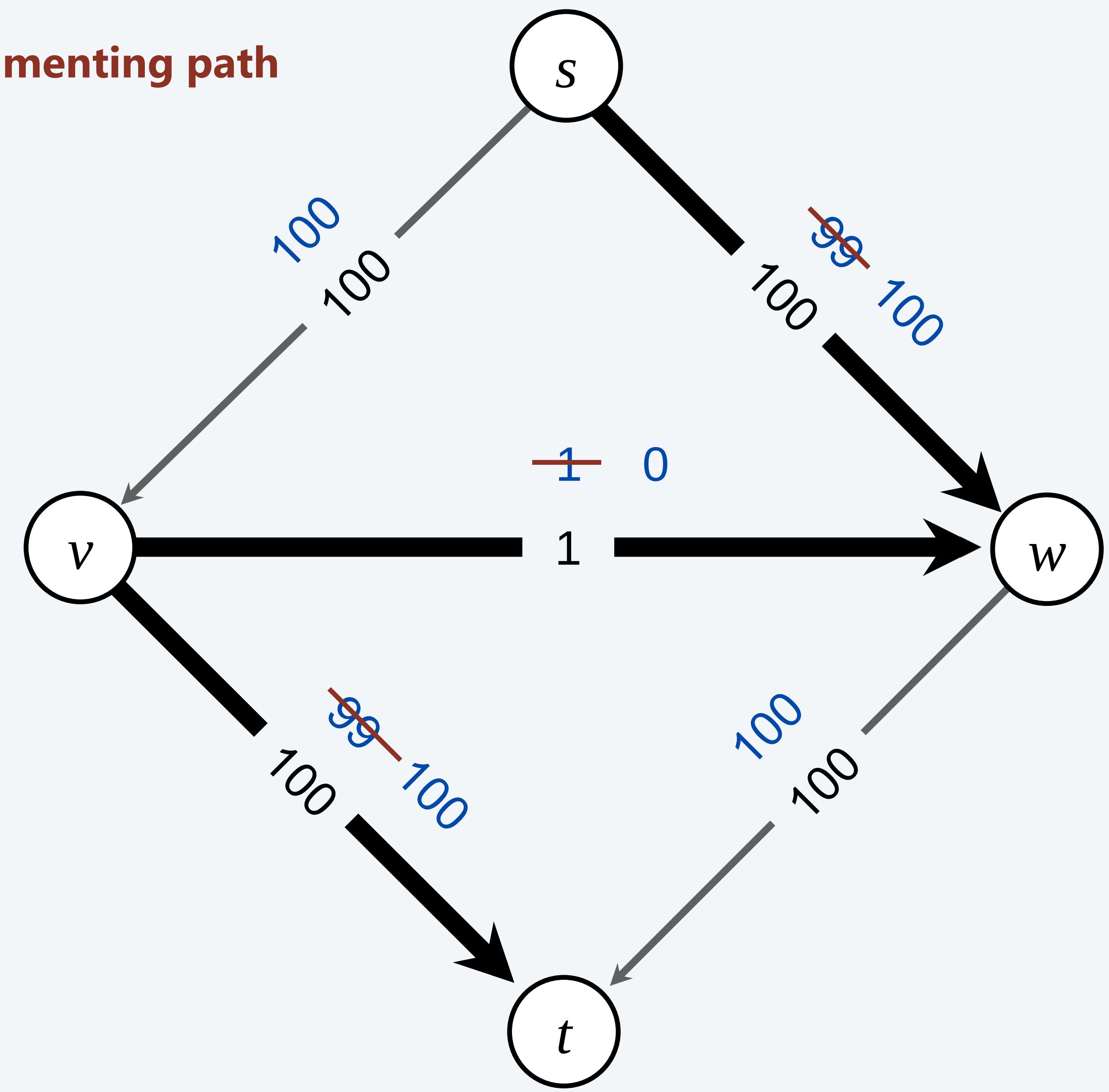
Bad news. Number of augmenting paths can be exponential in input size.



Ford–Fulkerson algorithm: exponential-time example

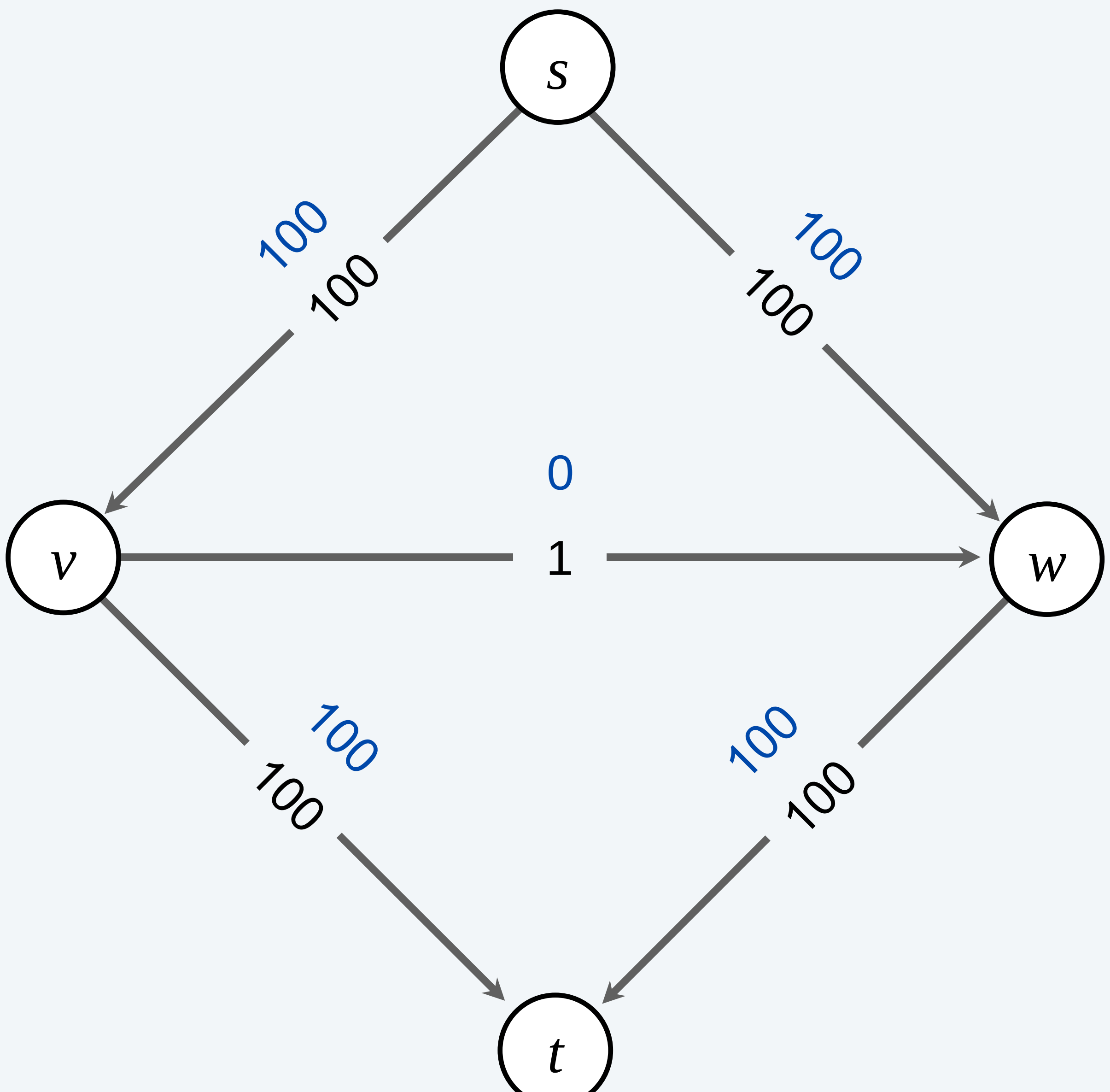
Bad news. Number of augmenting paths can be exponential in input size.

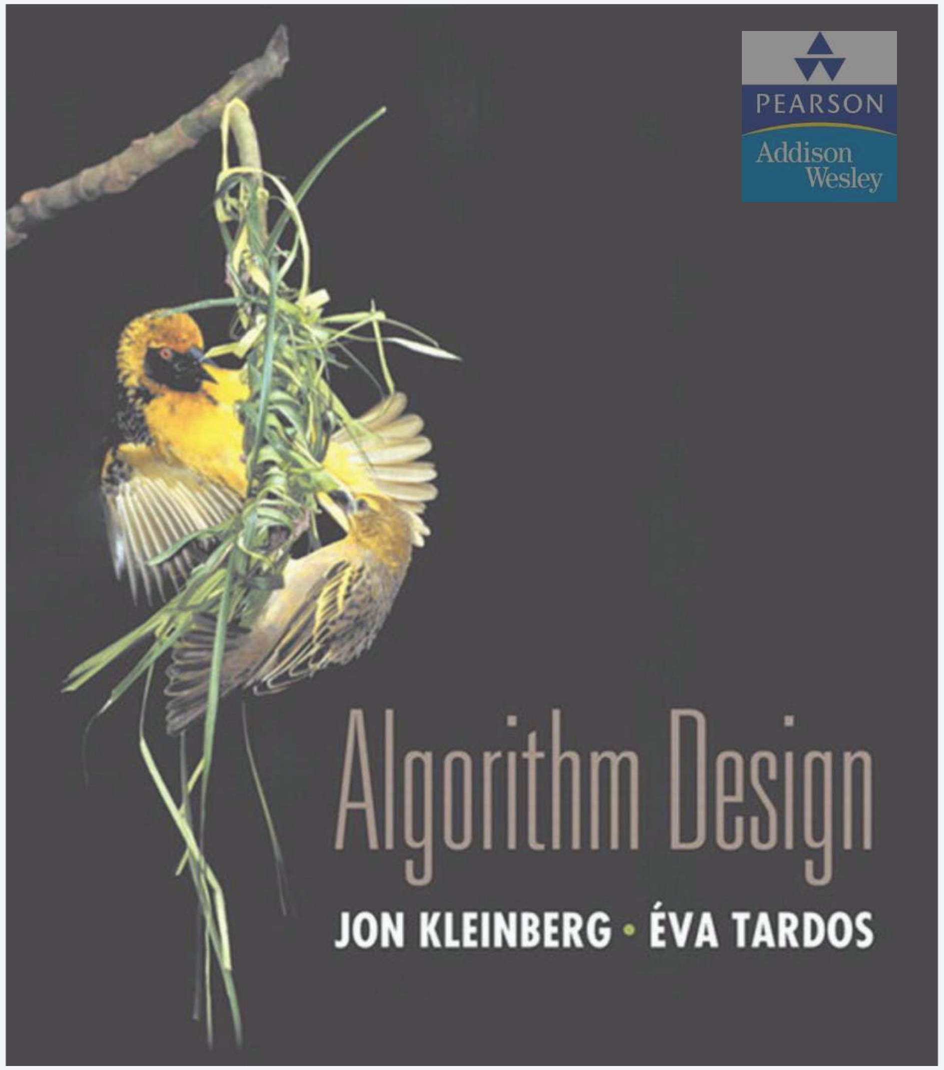
200th augmenting path



Ford–Fulkerson algorithm: exponential-time example

Bad news. Number of augmenting paths can be exponential in input size.





SECTION 7.3

7. NETWORK FLOW I

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

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:

- Sufficiently large bottleneck capacity.
- Fewest 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.

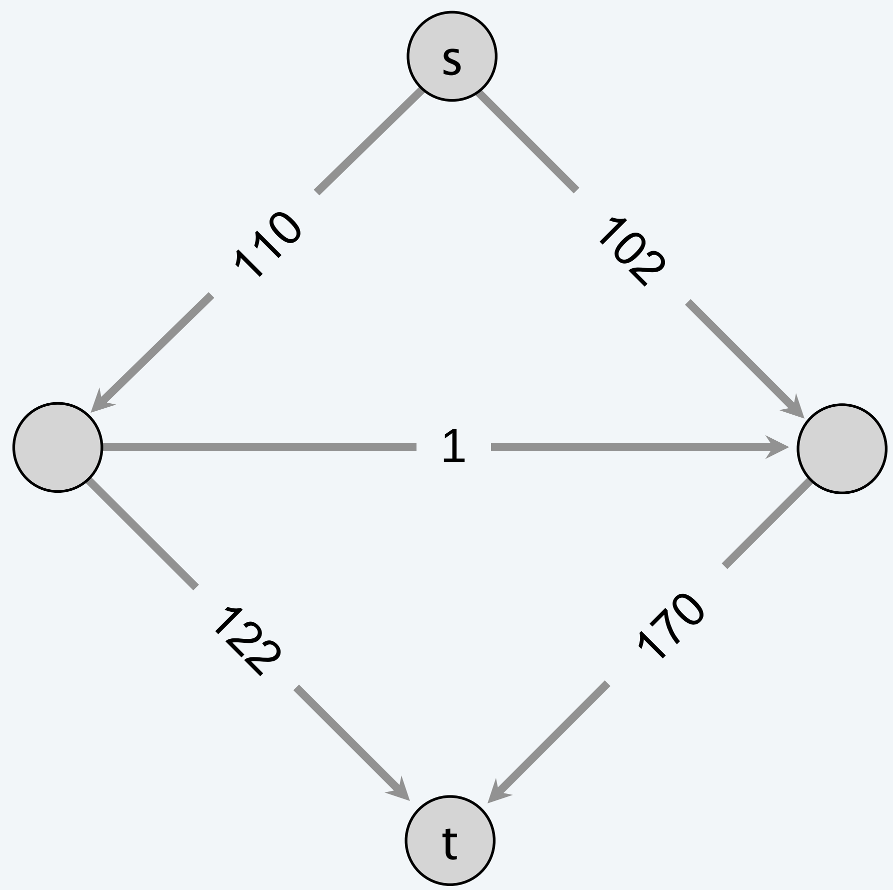
Edmonds-Karp 1972 (USA)

Capacity-scaling algorithm

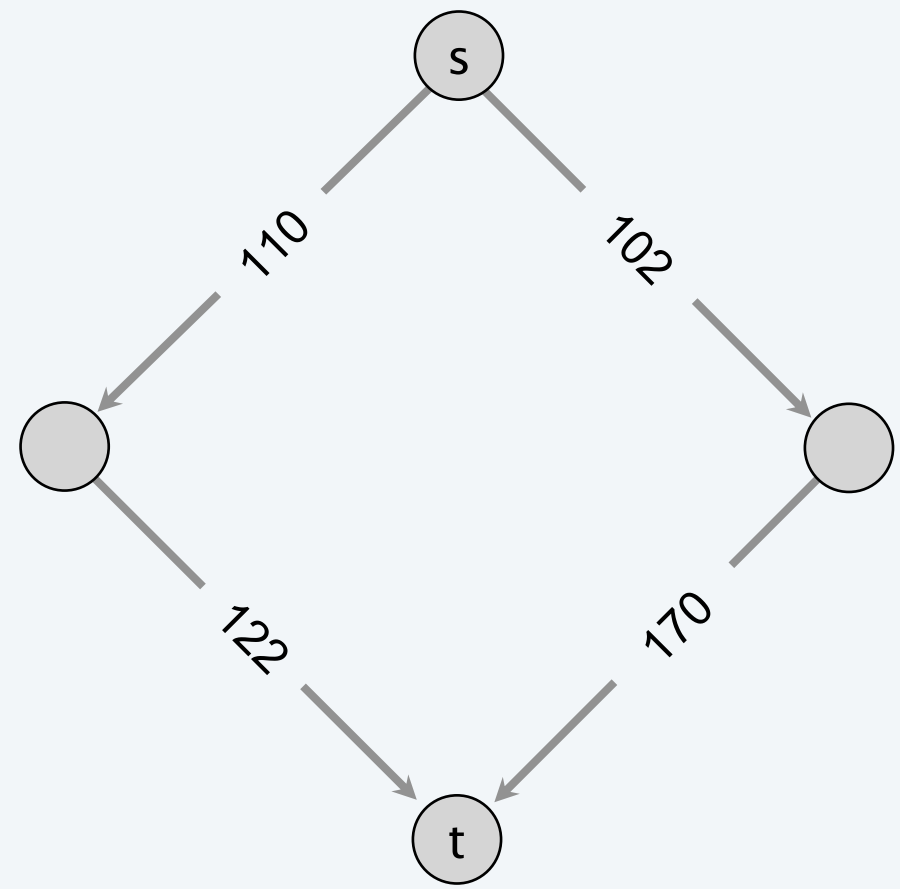
Overview. Choosing augmenting paths with "large" bottleneck capacity.

though not necessarily largest

- Maintain scaling parameter Δ.
- 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 $\geq \Delta$.



G_f



$G_f(\Delta)$, Δ = 100

Capacity-scaling algorithm

CAPACITY-SCALING(G)

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

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

WHILE $(\Delta \ge 1)$

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

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

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

    Update $G_f(\Delta)$.

  $\Delta \leftarrow \Delta / 2$.

RETURN f.

Δ-scaling phase

Capacity-scaling algorithm: analysis of running time (sketch)

It can be proved the following:

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

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

⇨ total number of augmentations: $O(m \log C)$

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

Shortest augmenting path

Q. How to choose next augmenting path in Ford–Fulkerson?

A. Pick one that uses the fewest edges.

can find via BFS

SHORTEST-AUGMENTING-PATH(G)

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

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

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

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

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

Update G_f.

RETURN f.

Shortest augmenting path: running time

It can be proved the following:

Lemma 1. The total number of augmentations is at most $m\, n$.

Theorem. The shortest-augmenting-path algorithm takes $O(m^2\, n)$ time.

Augmenting-path algorithms: summary

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

fat paths

shortest paths

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

Maximum-flow algorithms: theory highlights

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

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

Maximum-flow algorithms: theory highlights

FOCS 2022

1 / 112 125%

Maximum Flow and Minimum-Cost Flow in Almost-Linear Time

Li Chen*
Georgia Tech
lichen@gatech.edu

Rasmus Kyng†
ETH Zurich
kyng@inf.ethz.ch

Yang P. Liu‡
Stanford University
yangpliu@stanford.edu

Richard Peng
University of Waterloo §
y5peng@uwaterloo.ca

Maximilian Probst Gutenberg†
ETH Zurich
maxprobst@ethz.ch

Sushant Sachdeva¶
University of Toronto
sachdeva@cs.toronto.edu

April 26, 2022

Abstract

We give an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with m edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a new dynamic graph data structure.

Our framework extends to algorithms running in $m^{1+o(1)}$ time for computing flows that minimize general edge-separable convex functions to high accuracy. This gives almost-linear time algorithms for several problems including entropy-regularized optimal transport, matrix scaling, p-norm flows, and p-norm isotonic regression on arbitrary directed acyclic graphs.