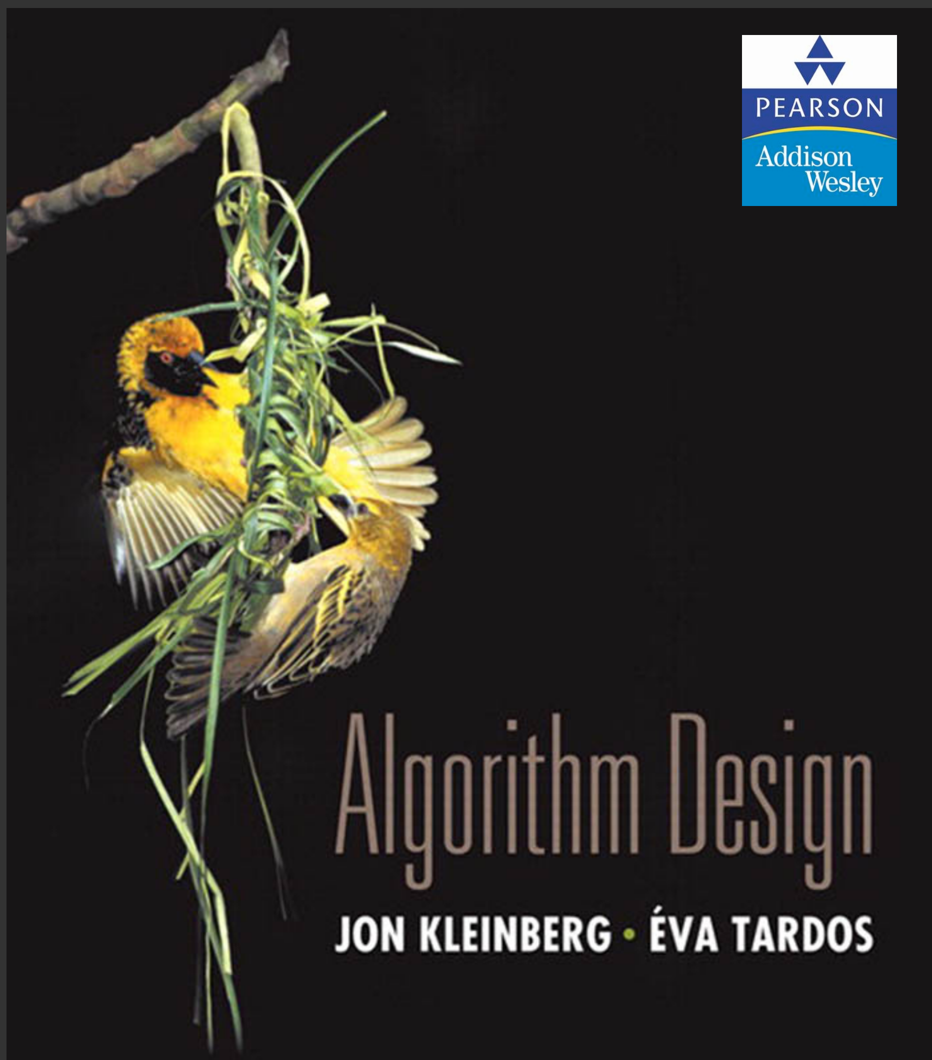


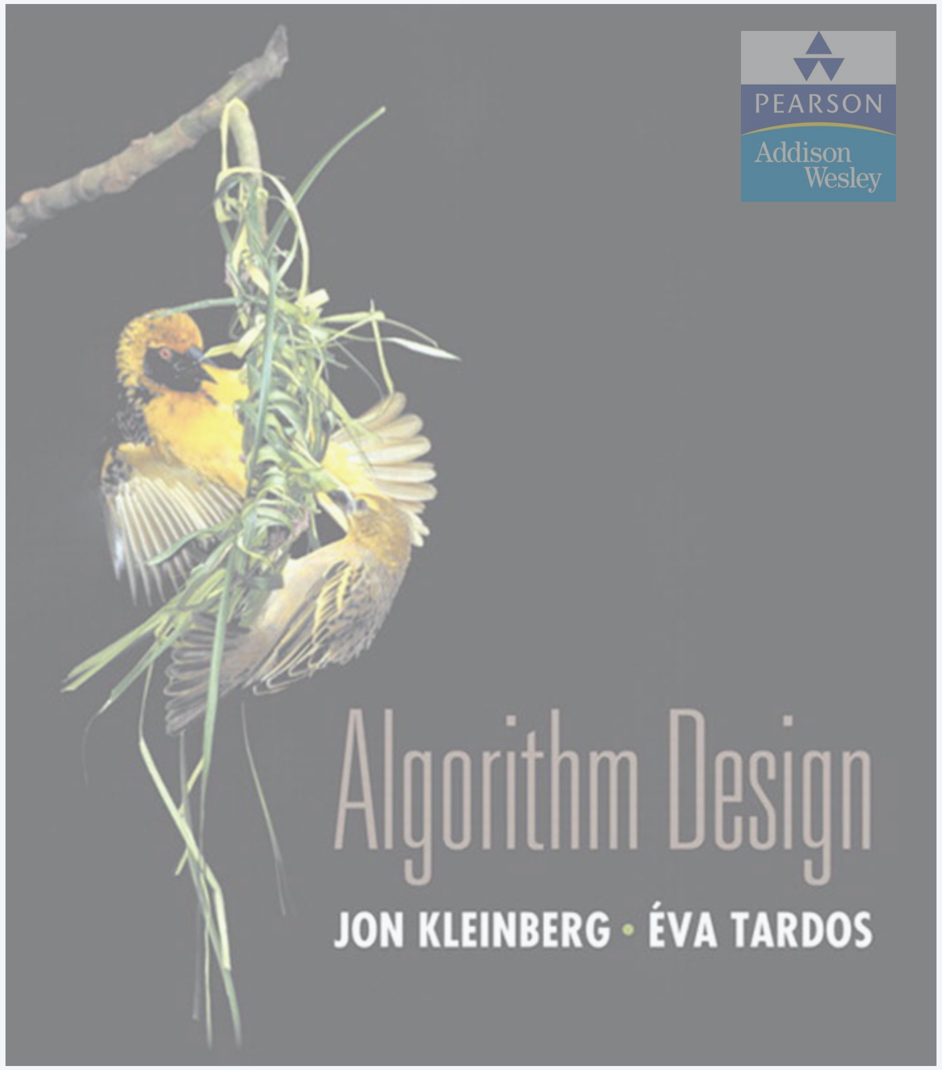

Lecture slides by Kevin Wayne

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*
- *Dinitz' algorithm*
- *simple unit-capacity networks*

Last updated on 10/29/24 3:32PM




SECTION 7.1

# 7. NETWORK FLOW I

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

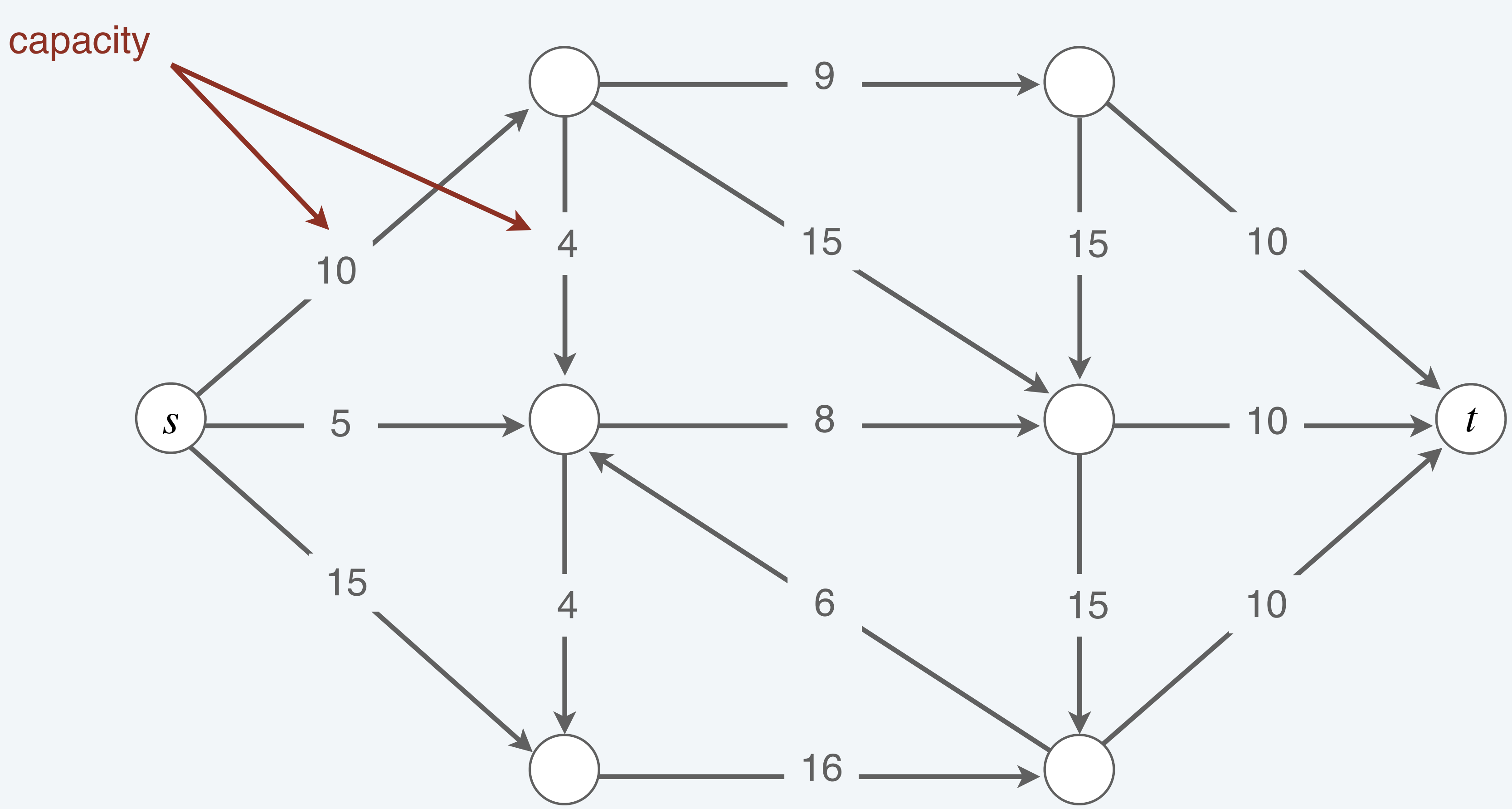
# Flow network

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

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

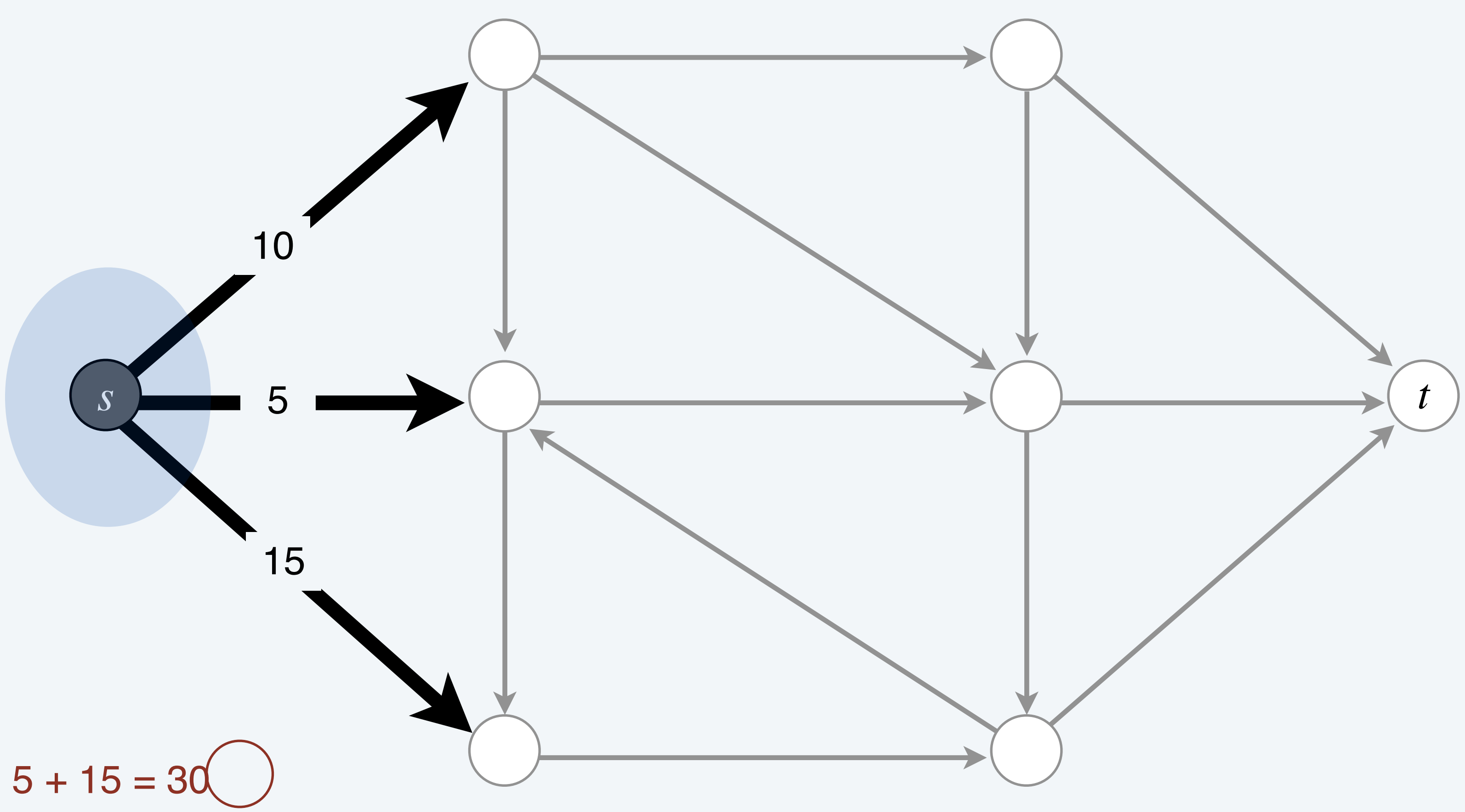


# Minimum-cut problem

**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 $B$.

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



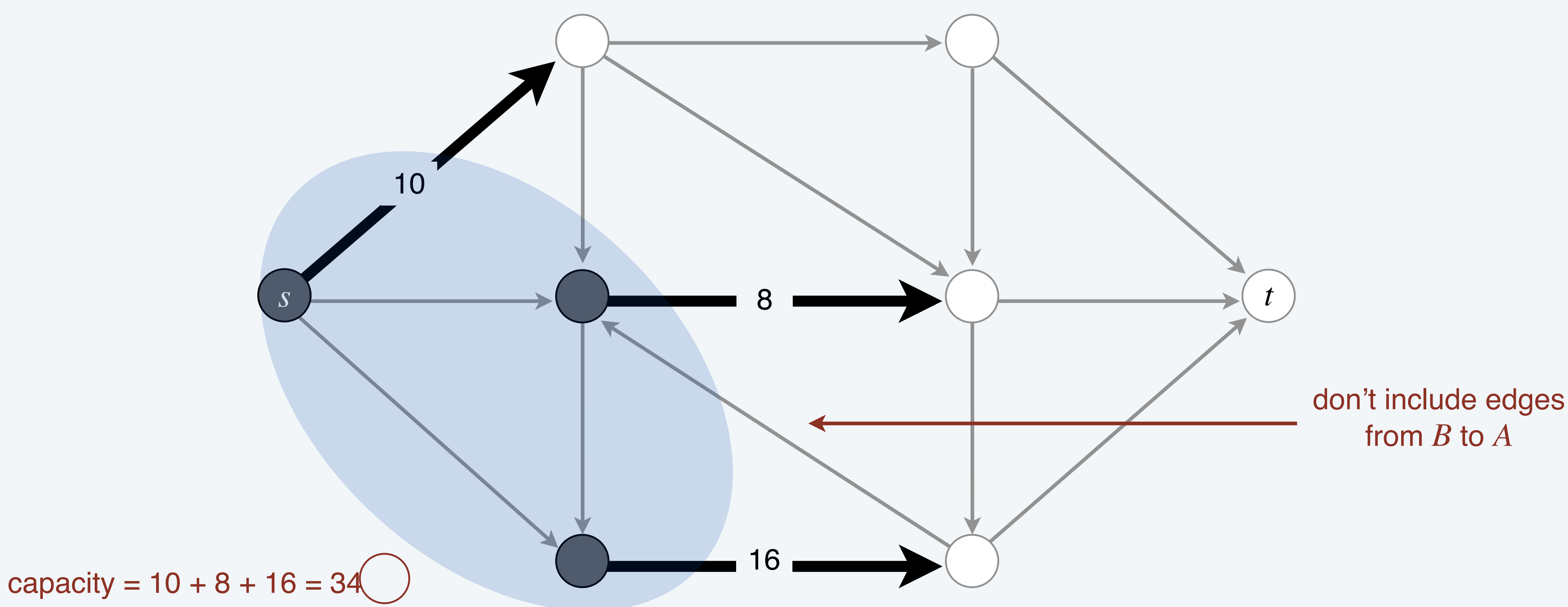

capacity = 10 + 5 + 15 = 30

# 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 $B$.

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



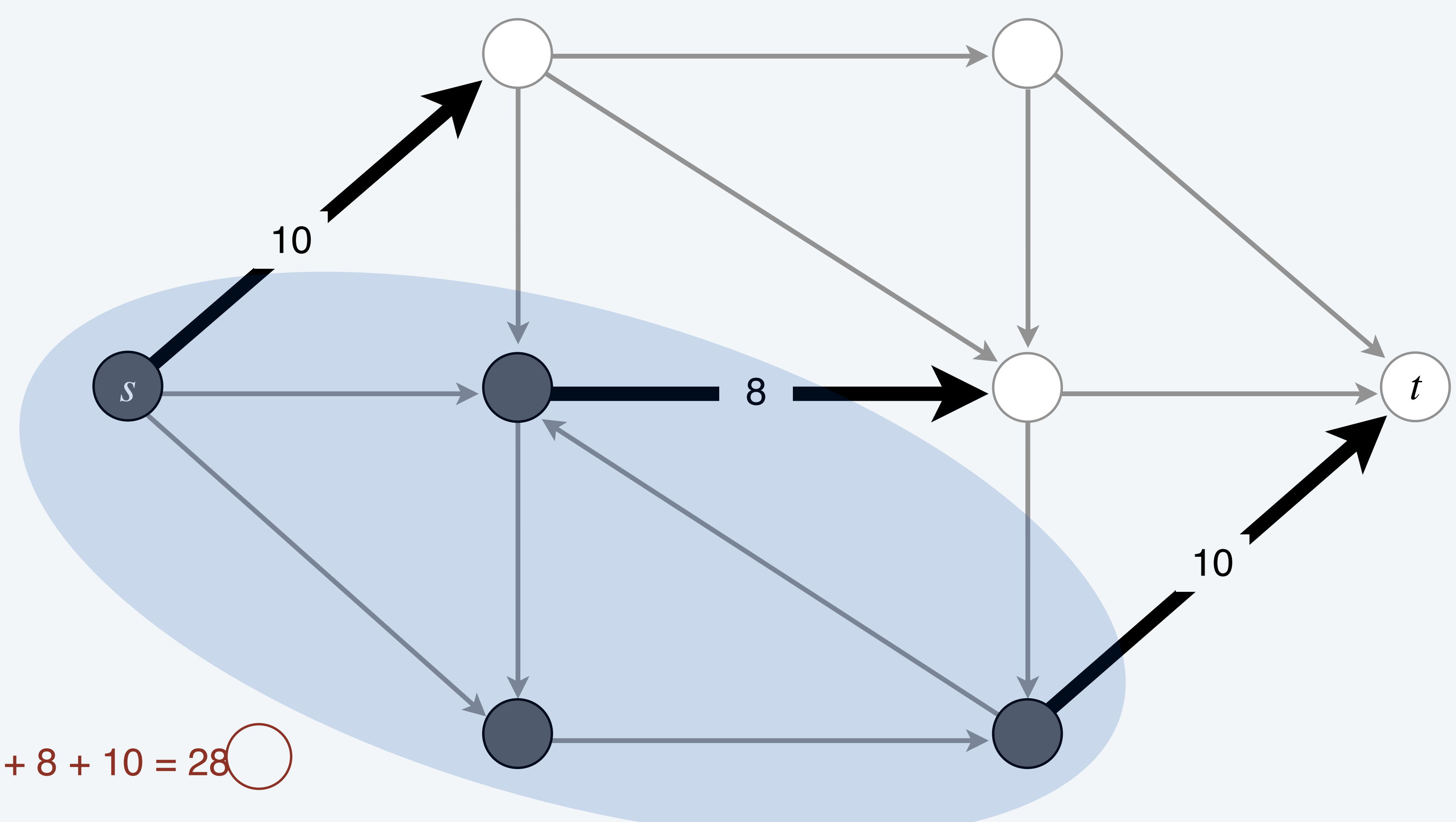
# Minimum-cut problem

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

# Network flow: quiz 1

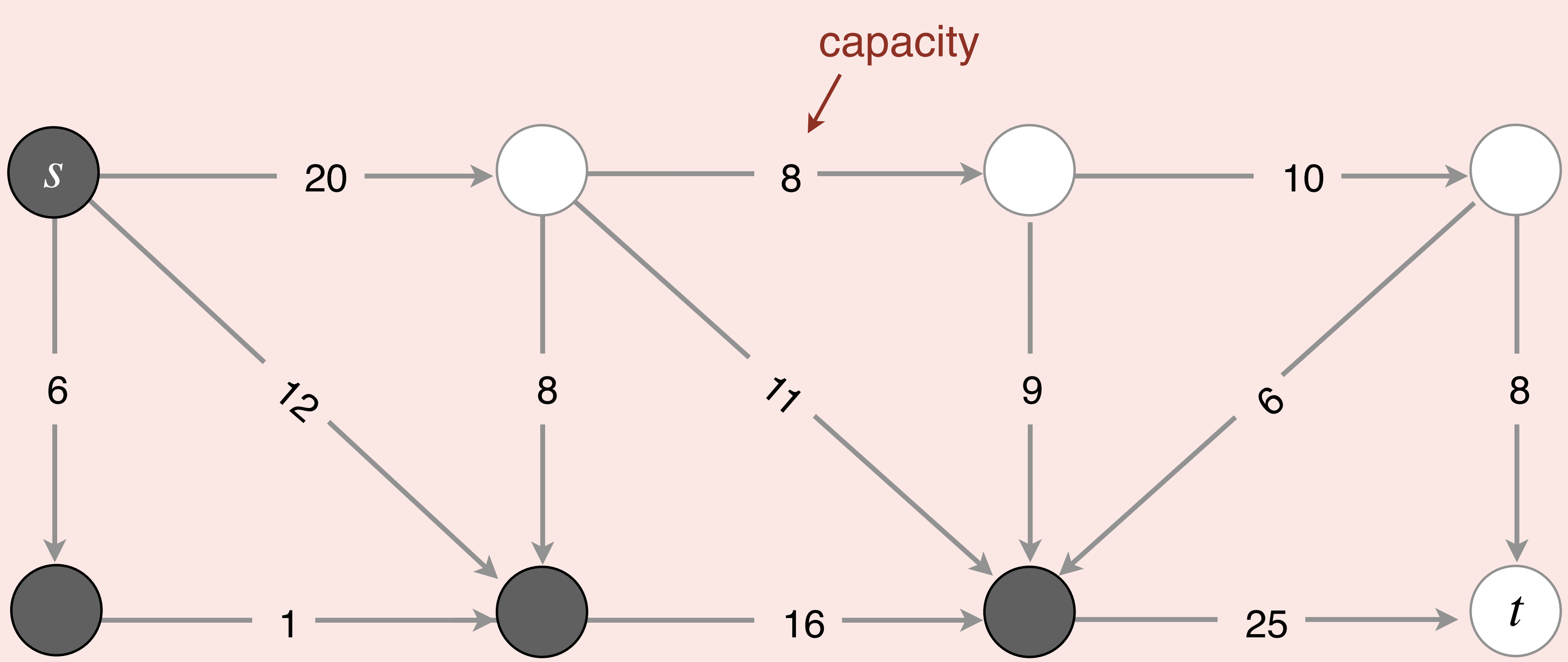
**Which is the capacity of the given *st*-cut?**

**A.** 11 (20 + 25 – 8 – 11 – 9 – 6)

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

**C.** 45 (20 + 25)

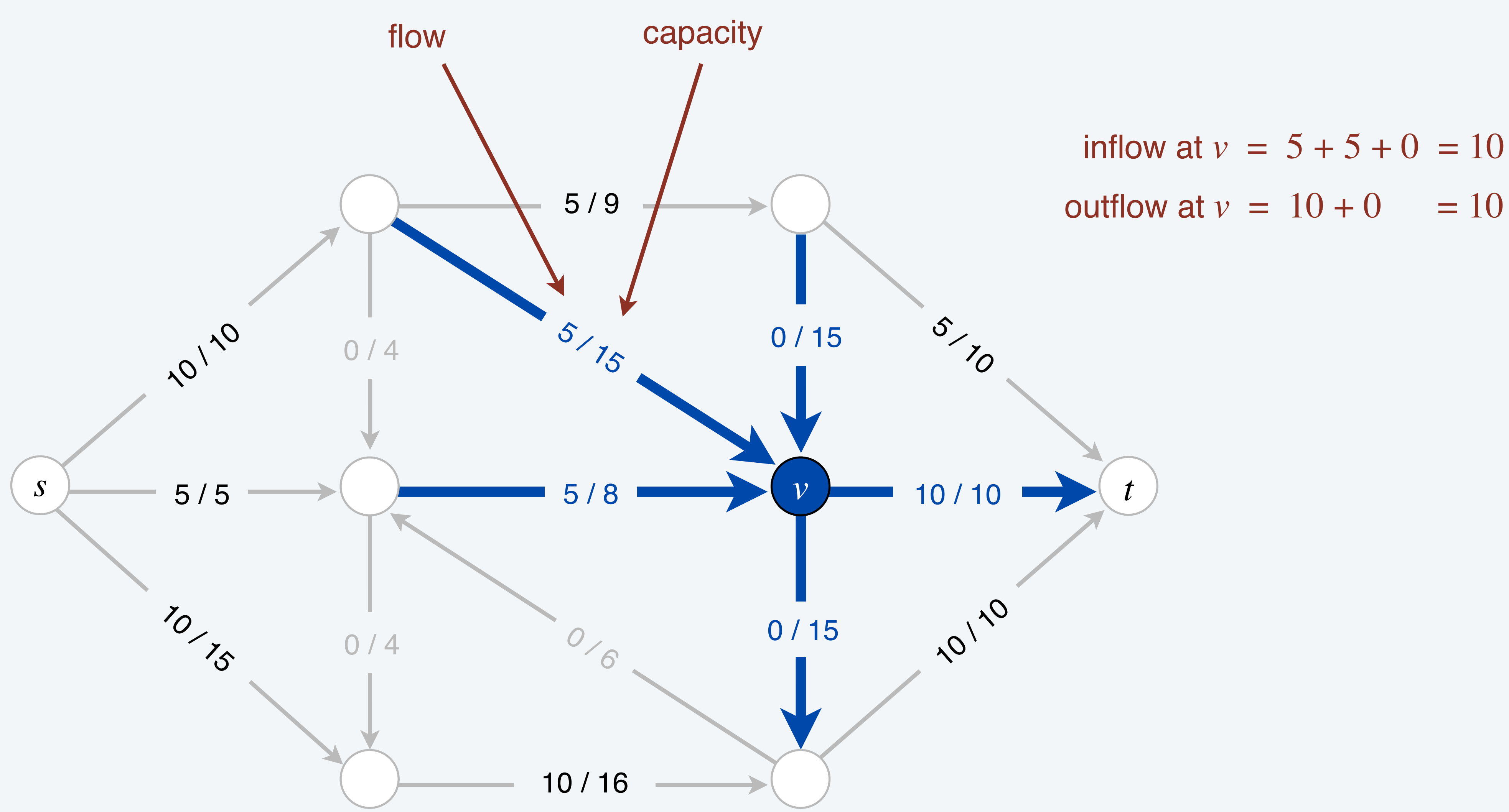
**D.** 79 (20 + 25 + 8 + 11 + 9 + 6)



# Maximum-flow problem

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

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

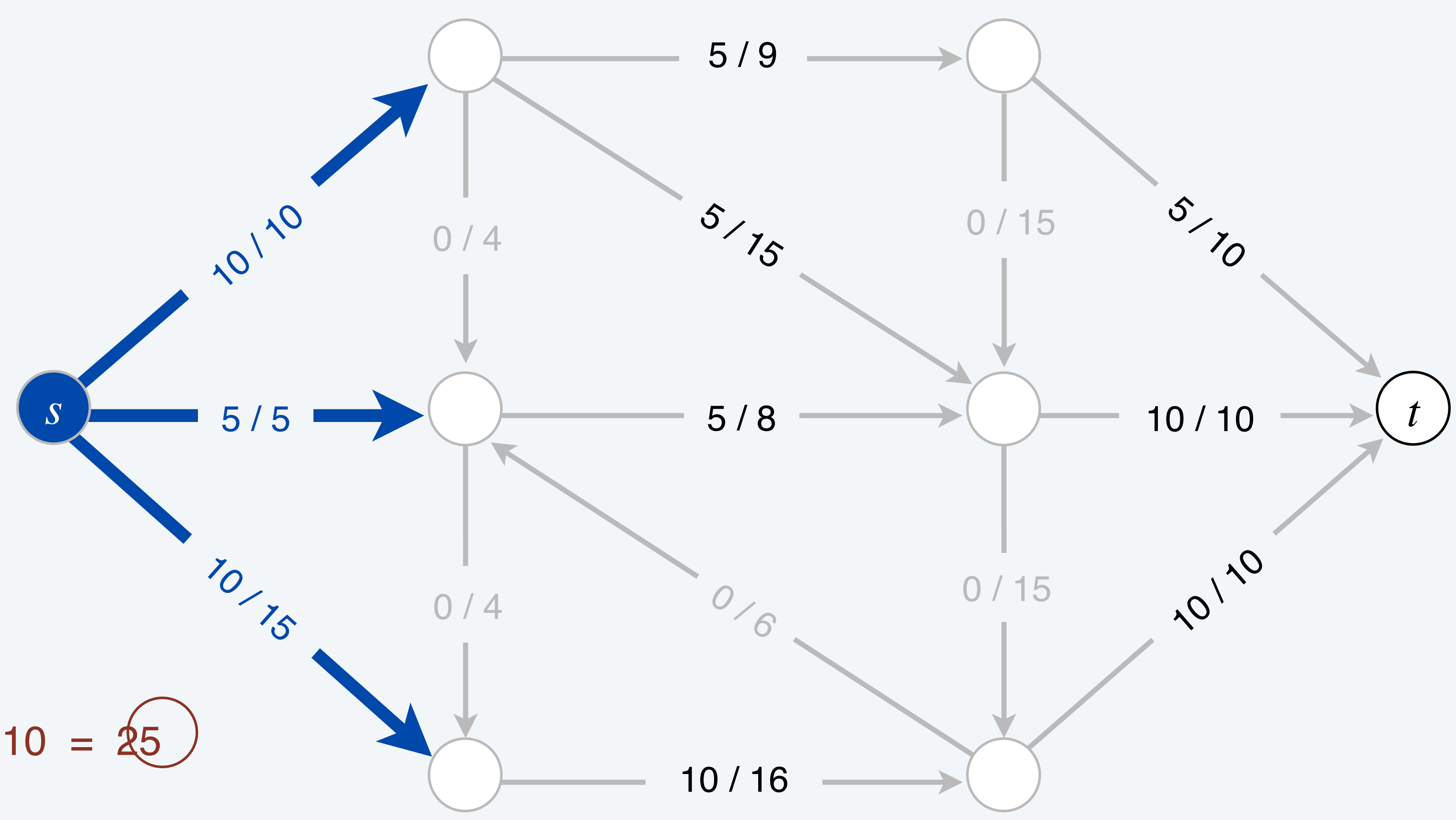


# Maximum-flow problem

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

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

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




value = 5 + 10 + 10 = 25

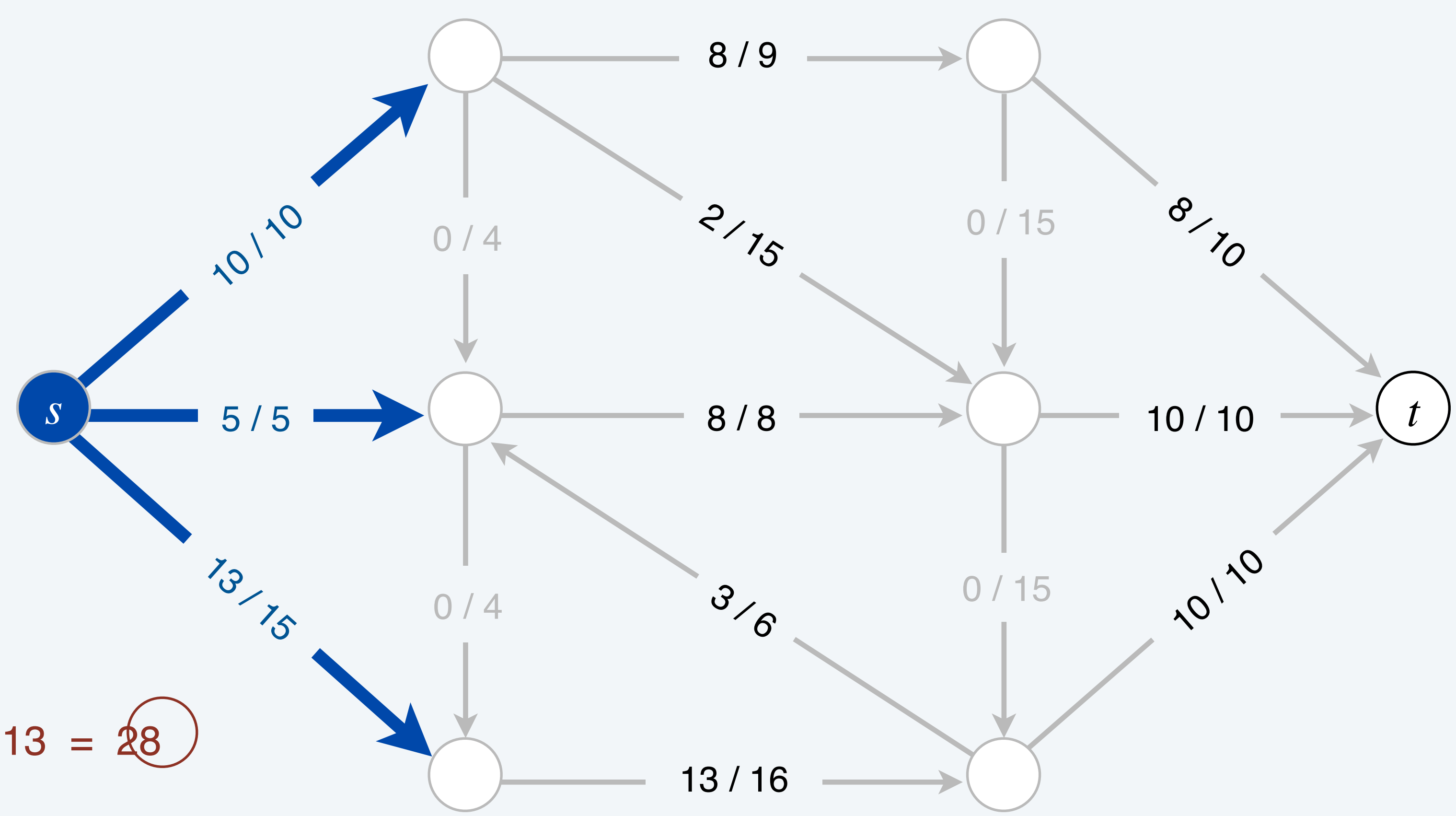
# Maximum-flow problem

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

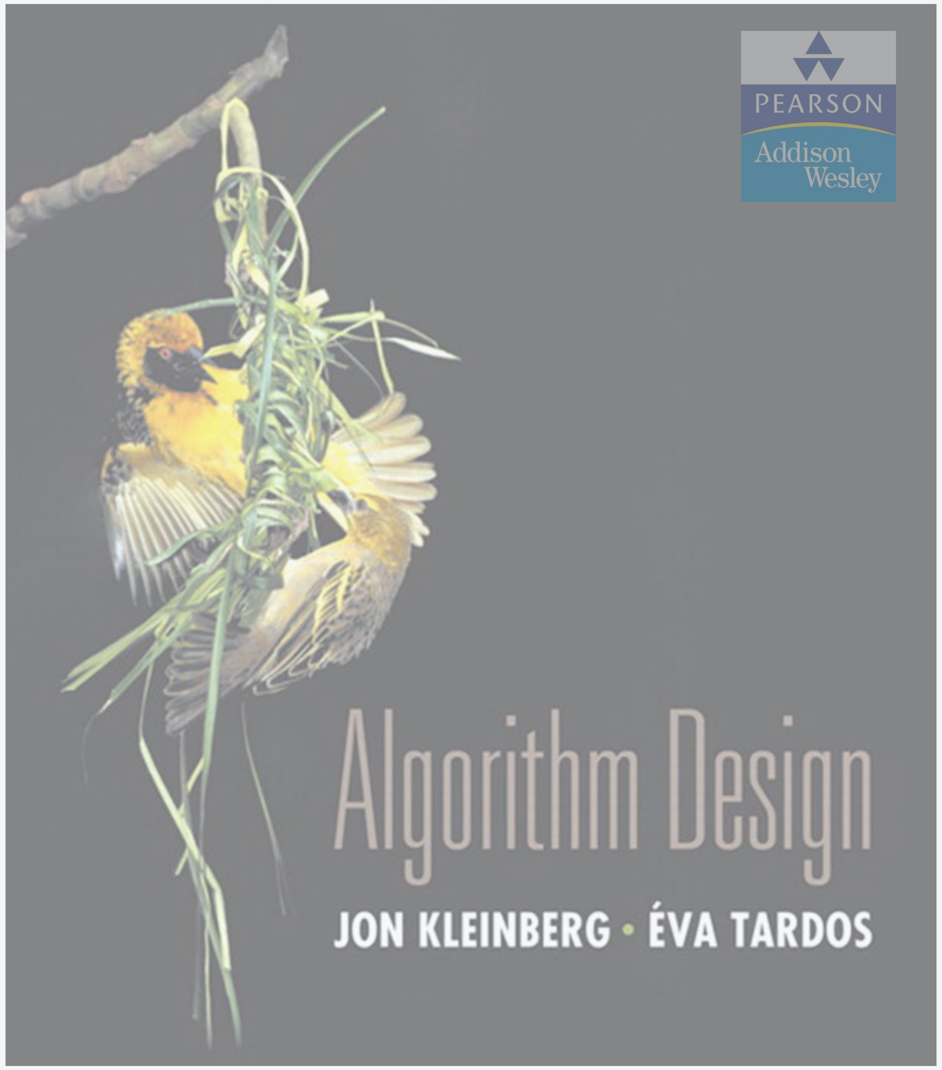
- For each $e \in E$: $\quad 0 \le f(e) \le c(e) \quad$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \quad$ [flow conservation]

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

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




value = 10 + 5 + 13 = 28




SECTION 7.1

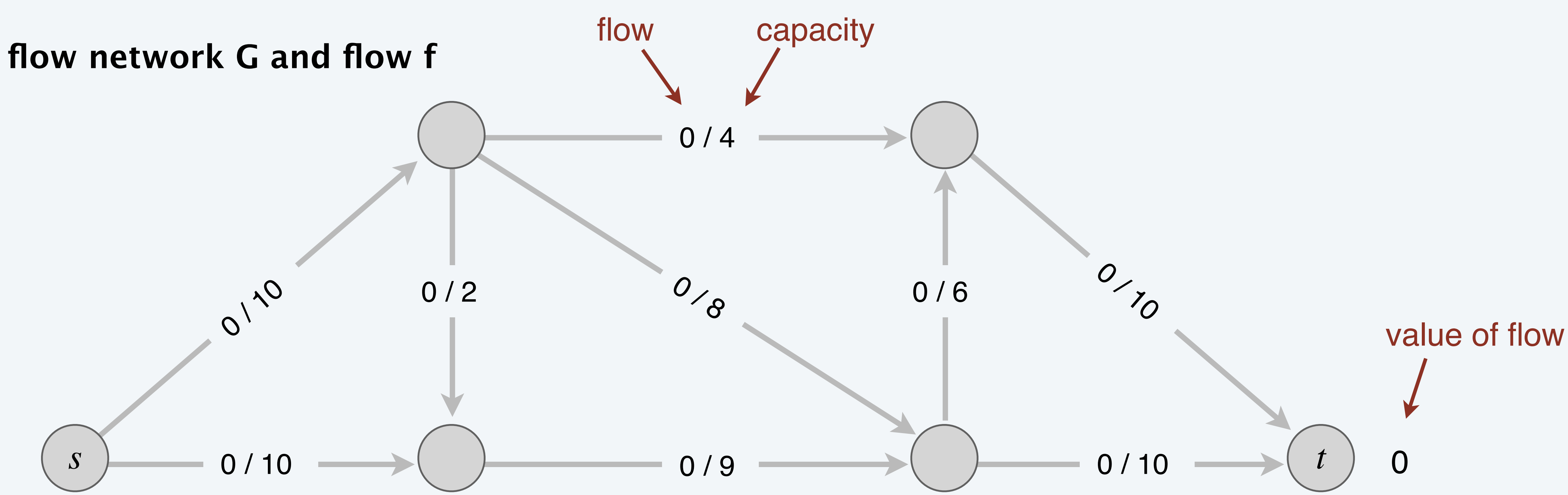
# 7. NETWORK FLOW I

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

# Toward a max-flow algorithm

Greedy algorithm.

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

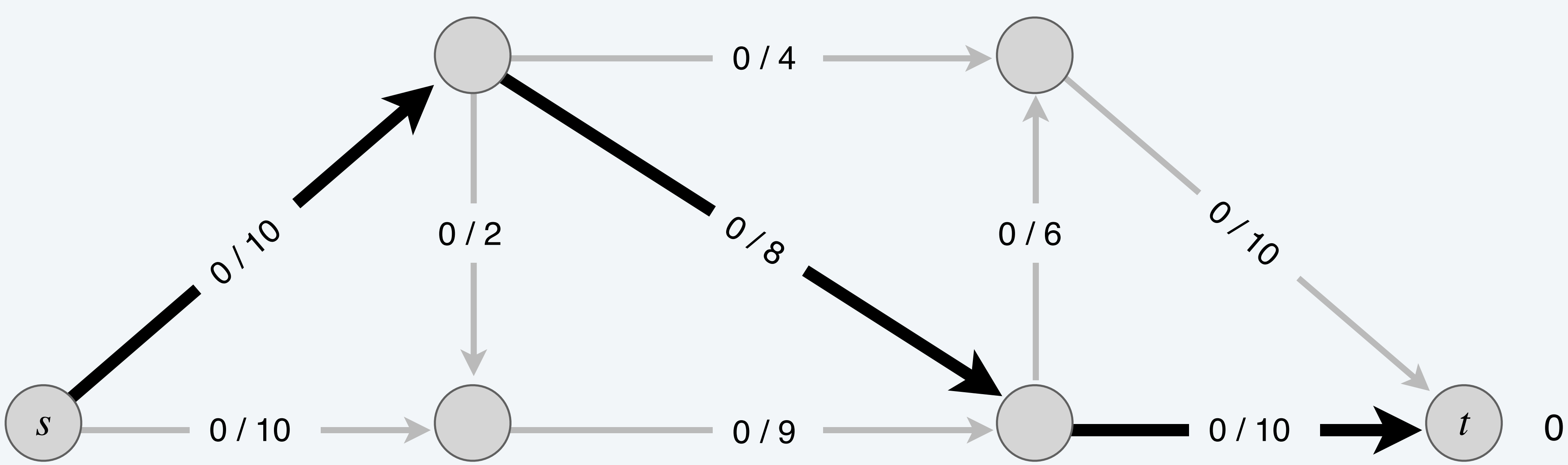


# Toward a max-flow algorithm

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.

flow network G and flow f

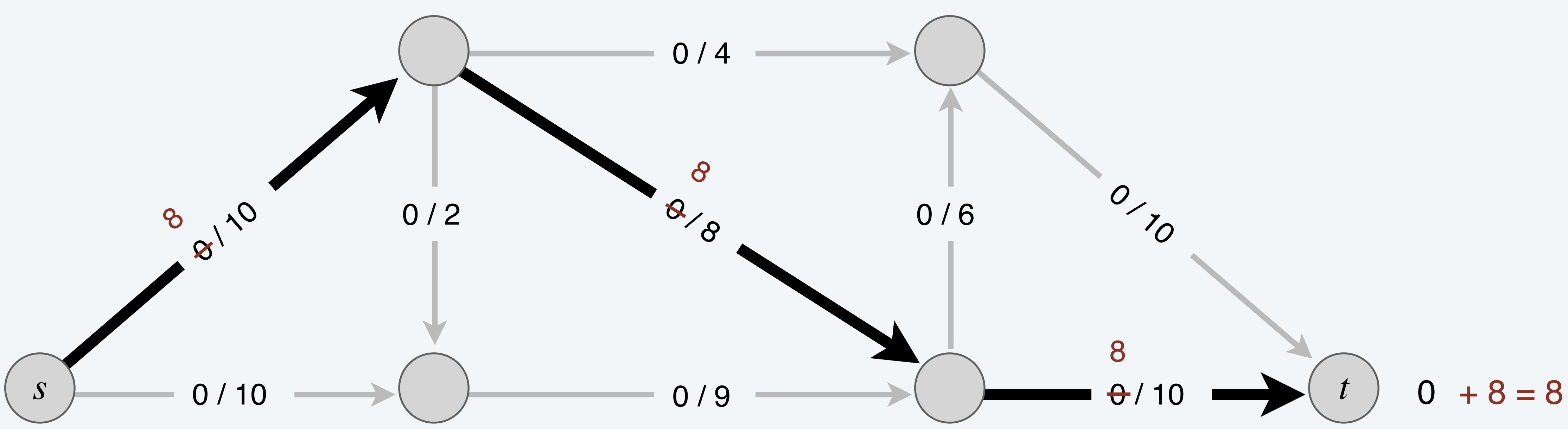


# Toward a max-flow algorithm

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**

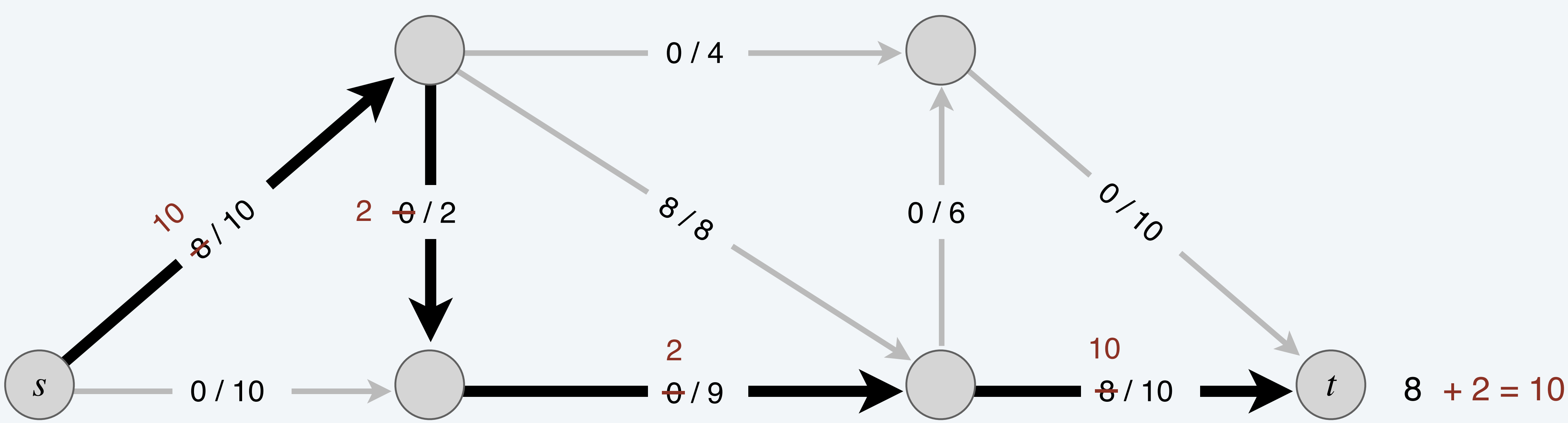


# Toward a max-flow algorithm

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

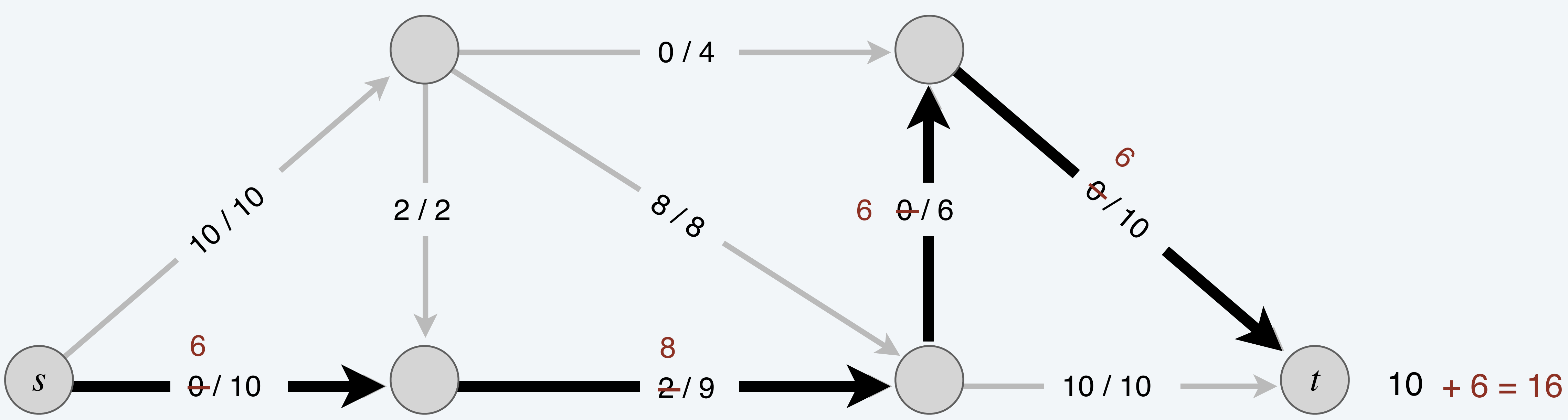


# Toward a max-flow algorithm

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**

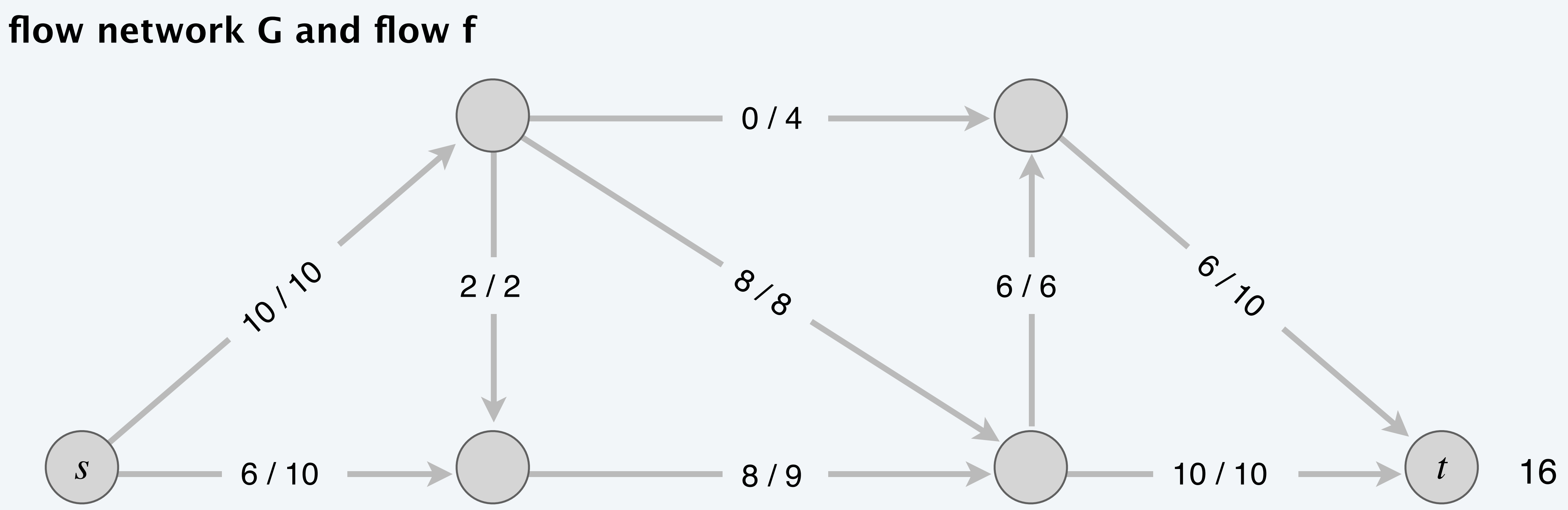


# Toward a max-flow algorithm

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



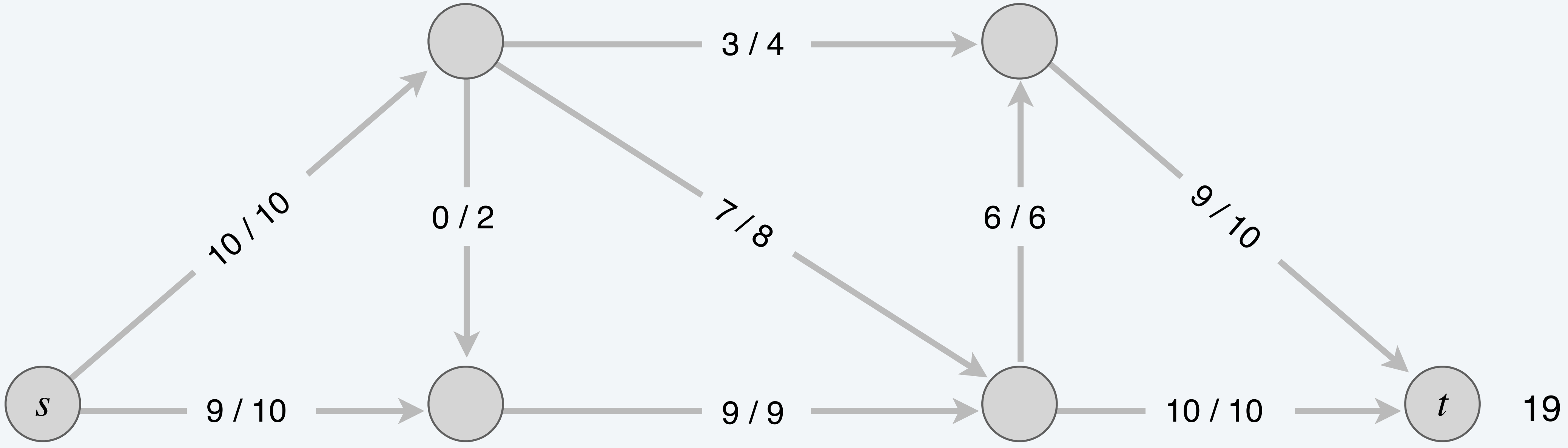
# Toward a max-flow algorithm

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

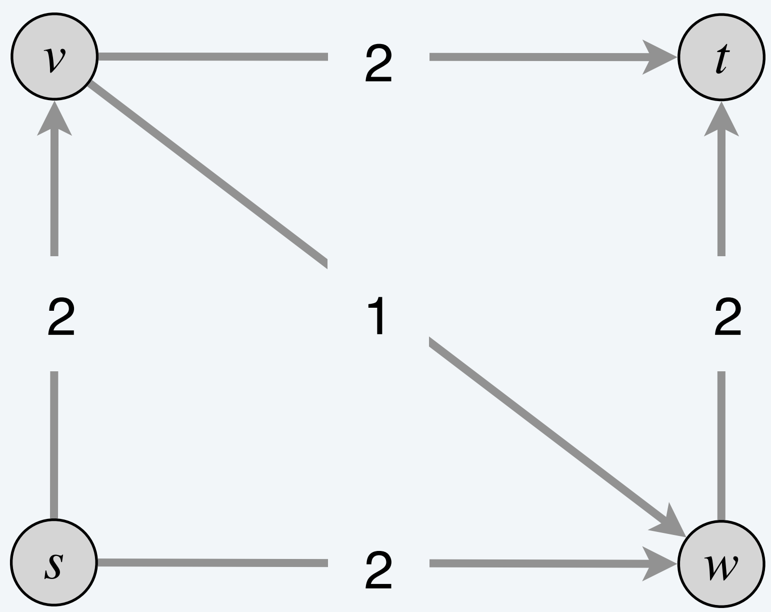
Q. Why does the greedy algorithm fail?

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

Ex. Consider flow network $G$.

- The unique max flow $f^*$ has $f^*(v, w) = 0$.
- Greedy algorithm could choose $s \to v \to w \to t$ 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 $c(e)$.

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

- "Undo" flow sent.

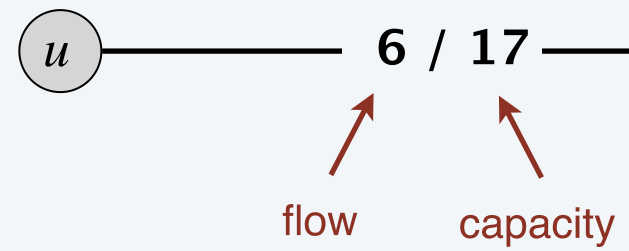
**Residual capacity.**

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

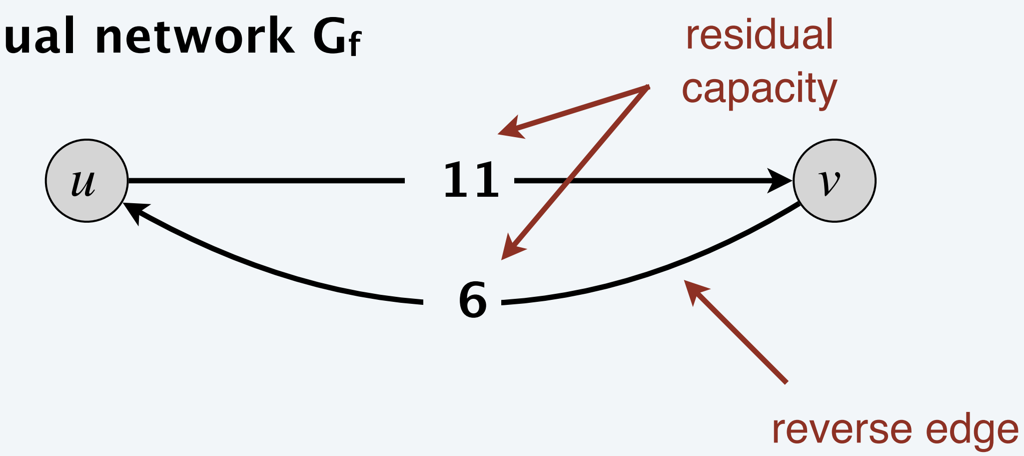
**Residual network.** $G_f = (V, E_f, s, t, c_f)$.

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

**original flow network G**




**residual network Gf**




edges with positive residual capacity

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

# Augmenting path

Def. An augmenting path is a simple $s \rightsquigarrow t$ 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($f, c, P$)

---

$\delta \leftarrow$ bottleneck capacity of augmenting path $P$.

FOREACH edge $e \in P$ :

  IF ($e \in E$) $f(e) \leftarrow f(e) + \delta$.

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

RETURN $f$.

---

# Network flow: quiz 2

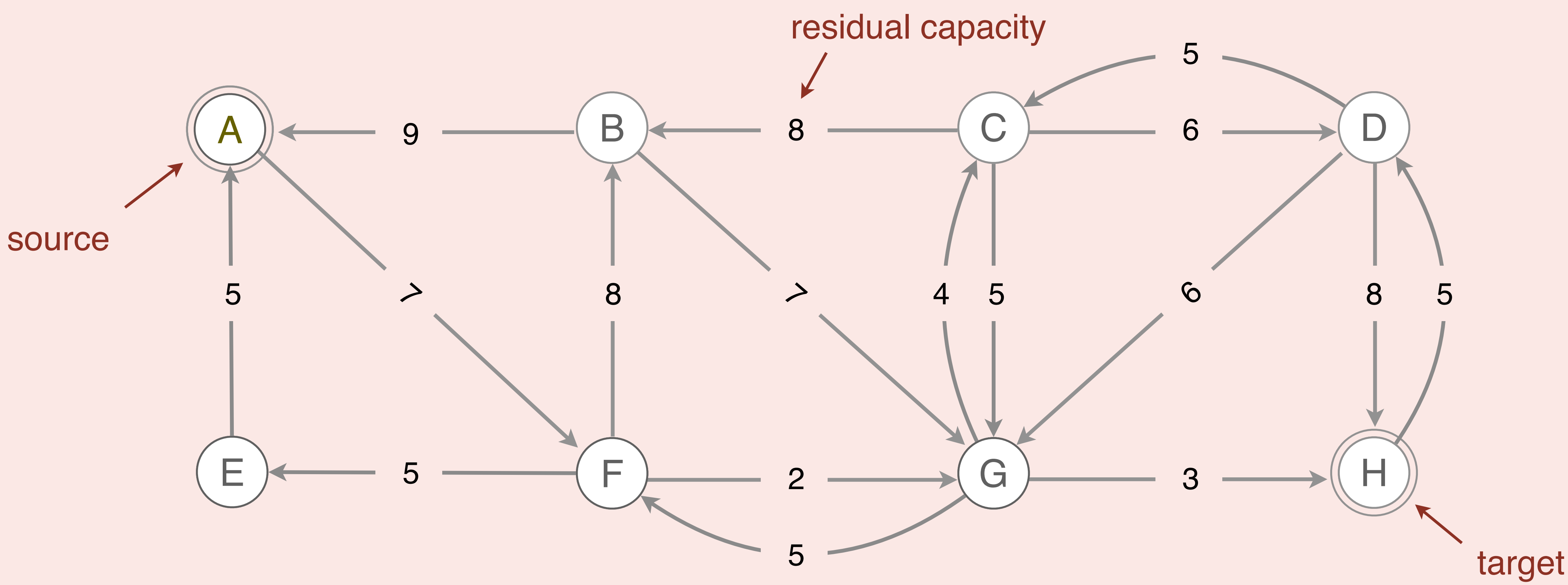
**Which is the augmenting path of highest bottleneck capacity?**

**A.** $A \to F \to G \to H$

**B.** $A \to B \to C \to D \to H$

**C.** $A \to F \to B \to G \to H$

**D.** $A \to F \to B \to G \to C \to D \to H$



# 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 $f$.

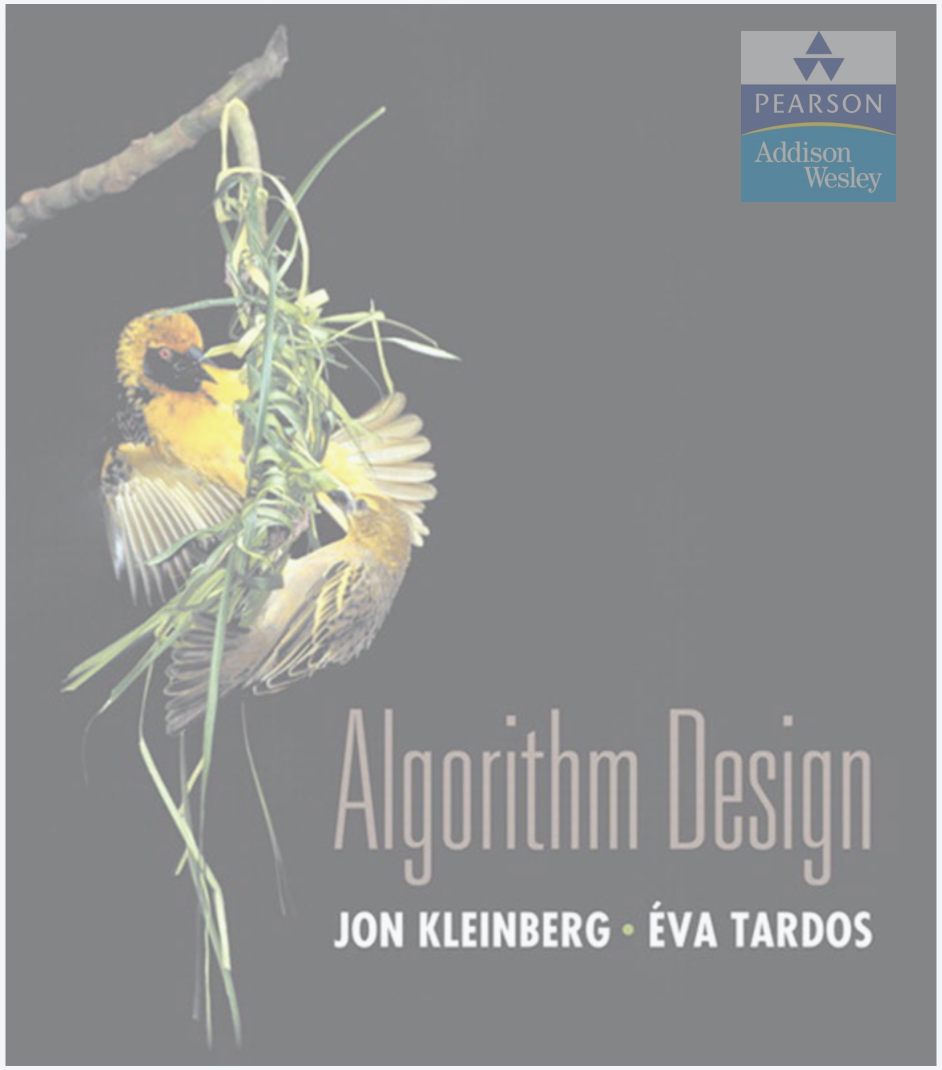
WHILE (there exists an s⤳t path $P$ in $G_f$)

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

  Update $G_f$.

RETURN $f$.

augmenting path




SECTION 7.2

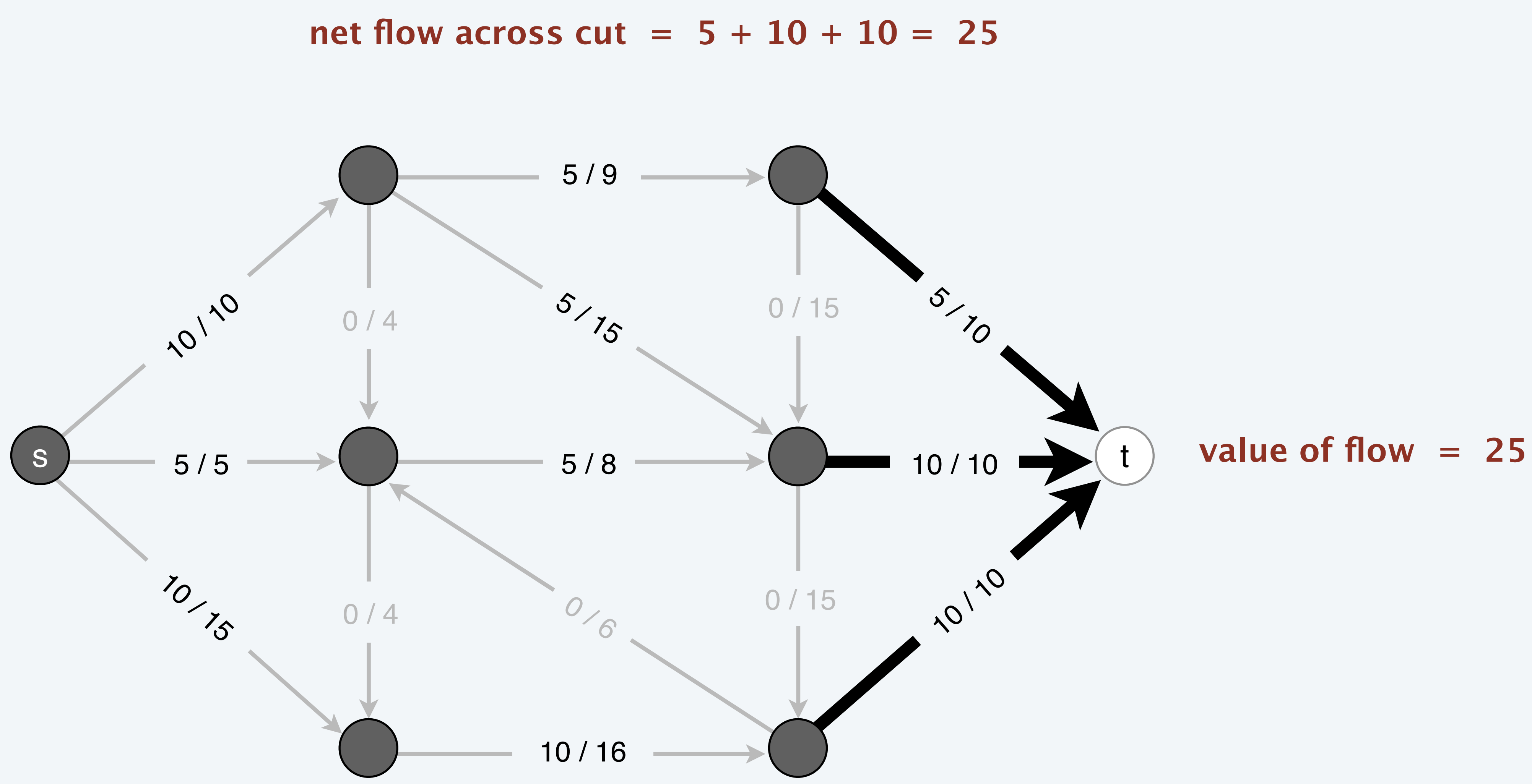
# 7. NETWORK FLOW I

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

# Relationship between flows and cuts

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

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

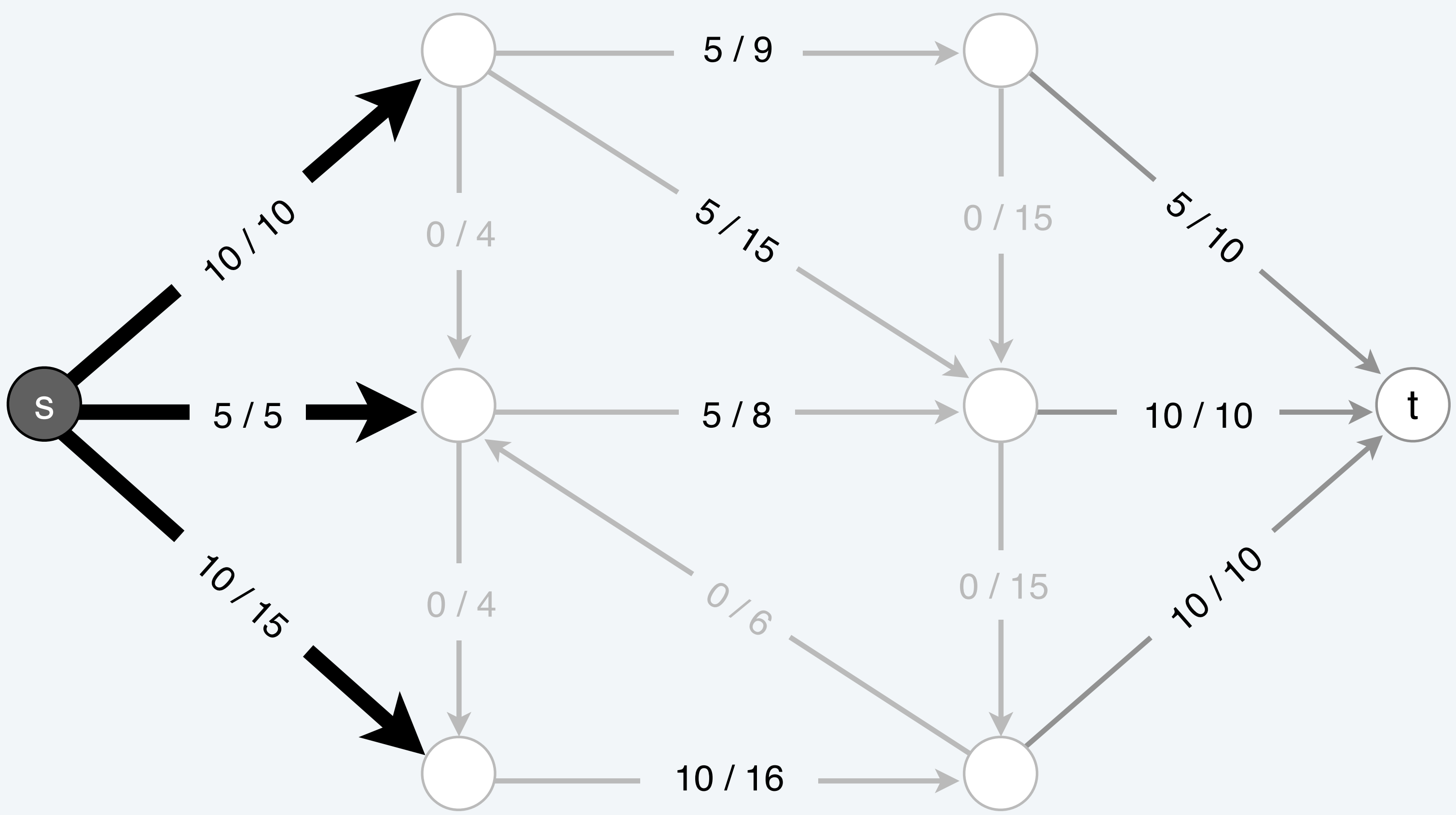


# Relationship between flows and cuts

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

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

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



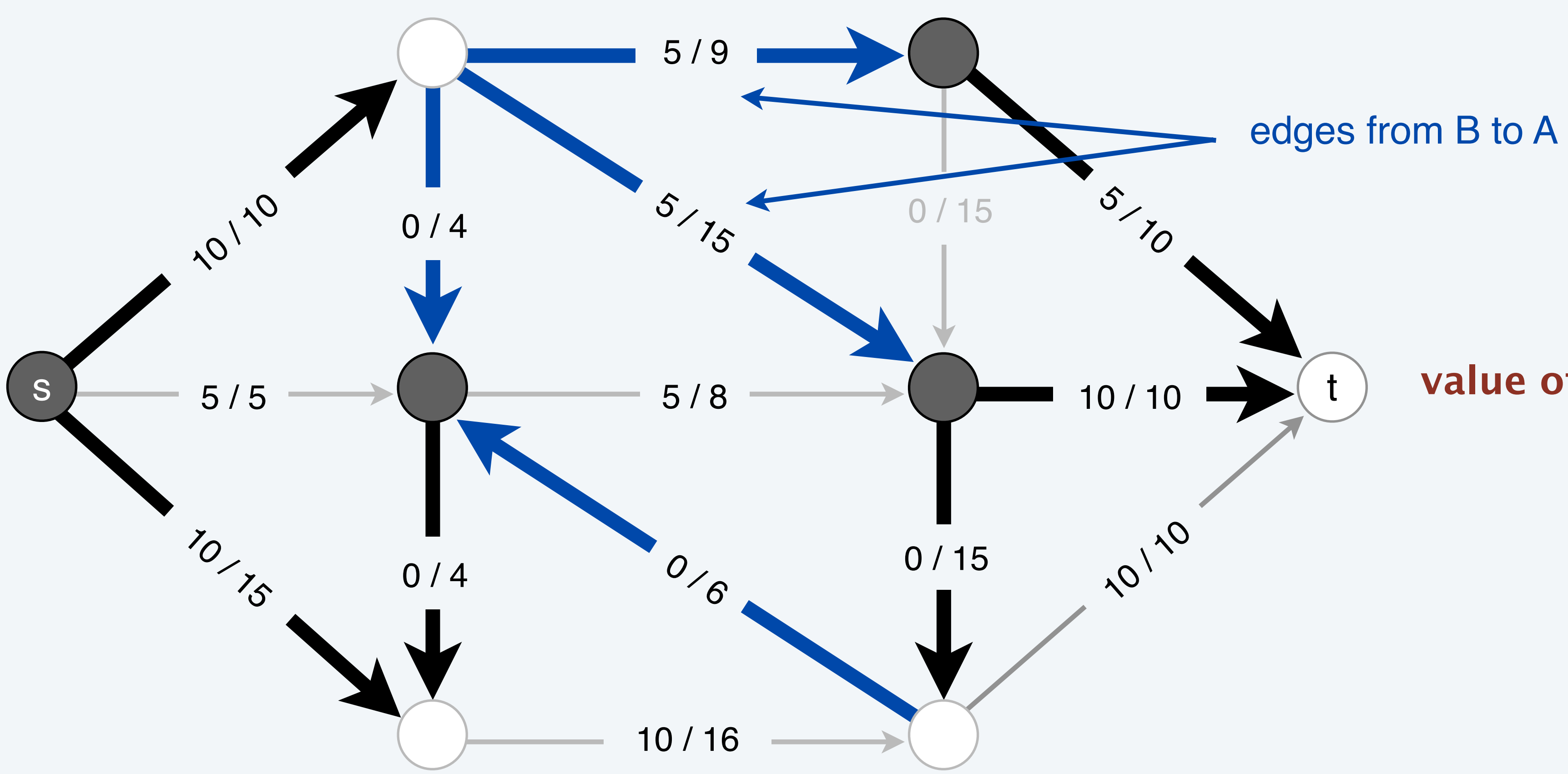

value of flow = 25

# 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 $(A, B)$.

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

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




**value of flow = 25**

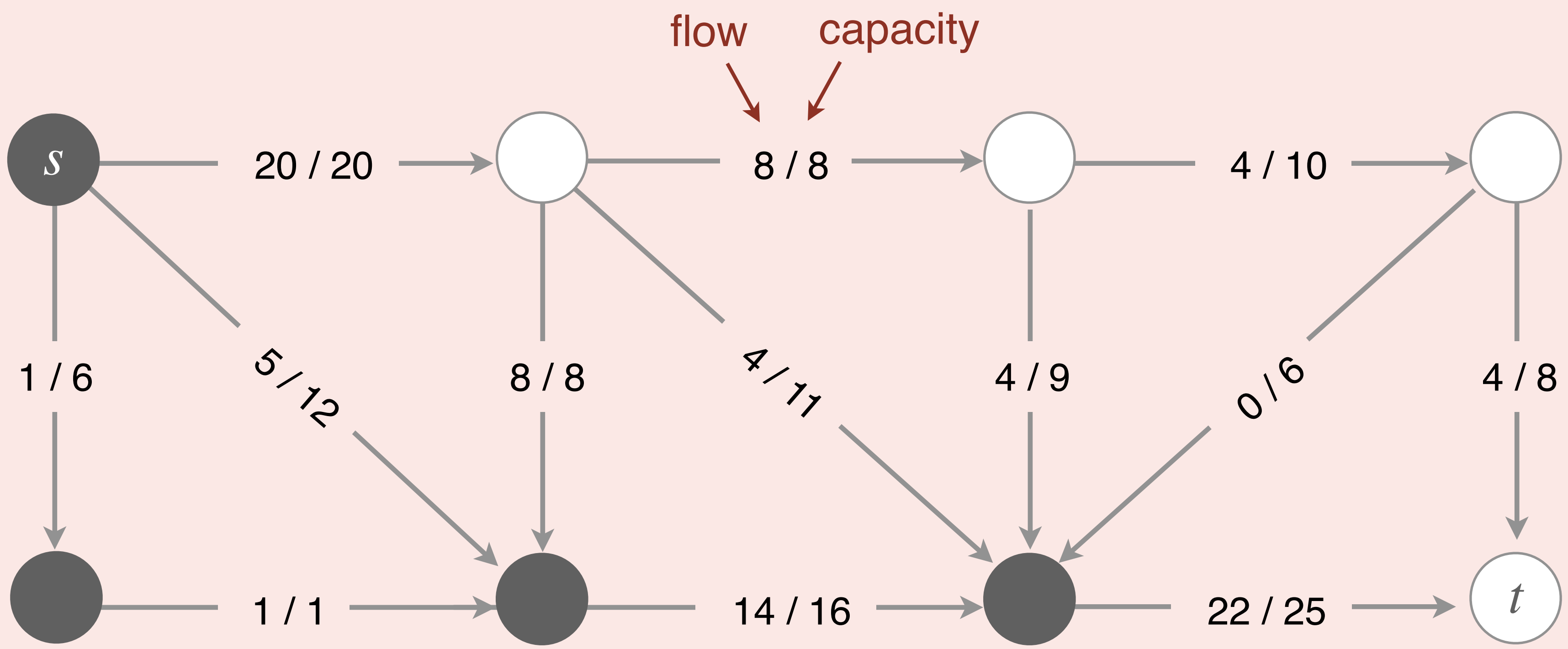
**Which is the net flow across the given cut?**

**A.** 11 (20 + 25 − 8 − 11 − 9 − 6)

**B.** 26 (20 + 22 − 8 − 4 − 4)

**C.** 42 (20 + 22)

**D.** 45 (20 + 25)



# Relationship between flows and cuts

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

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

Pf.

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

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

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

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

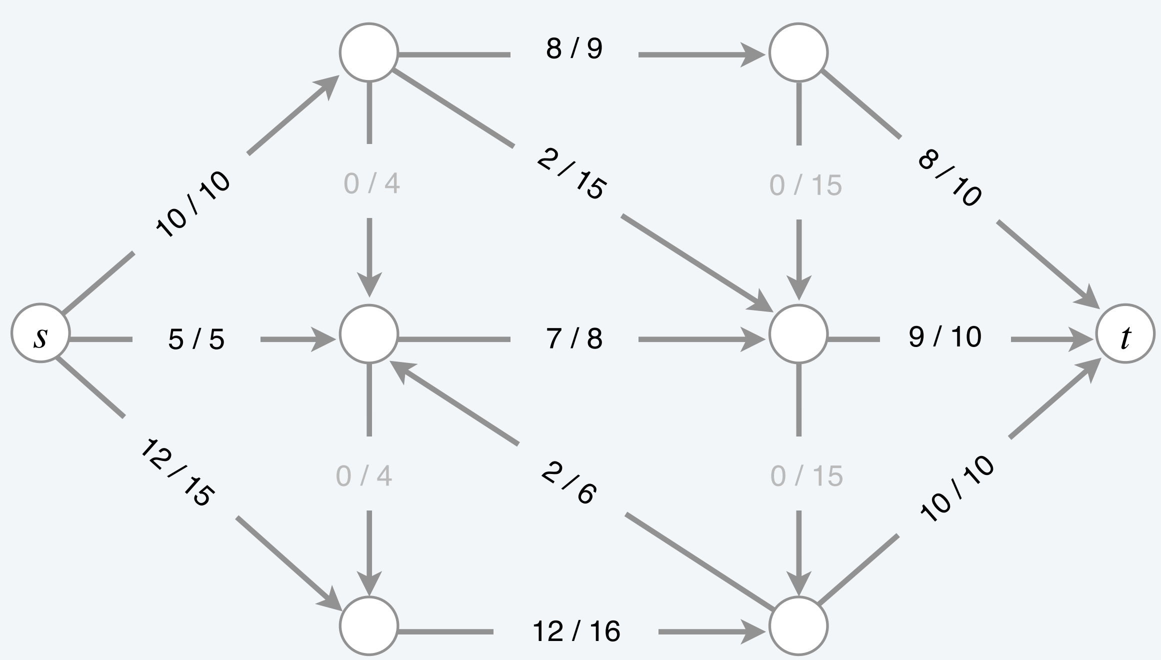
# Relationship between flows and cuts

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

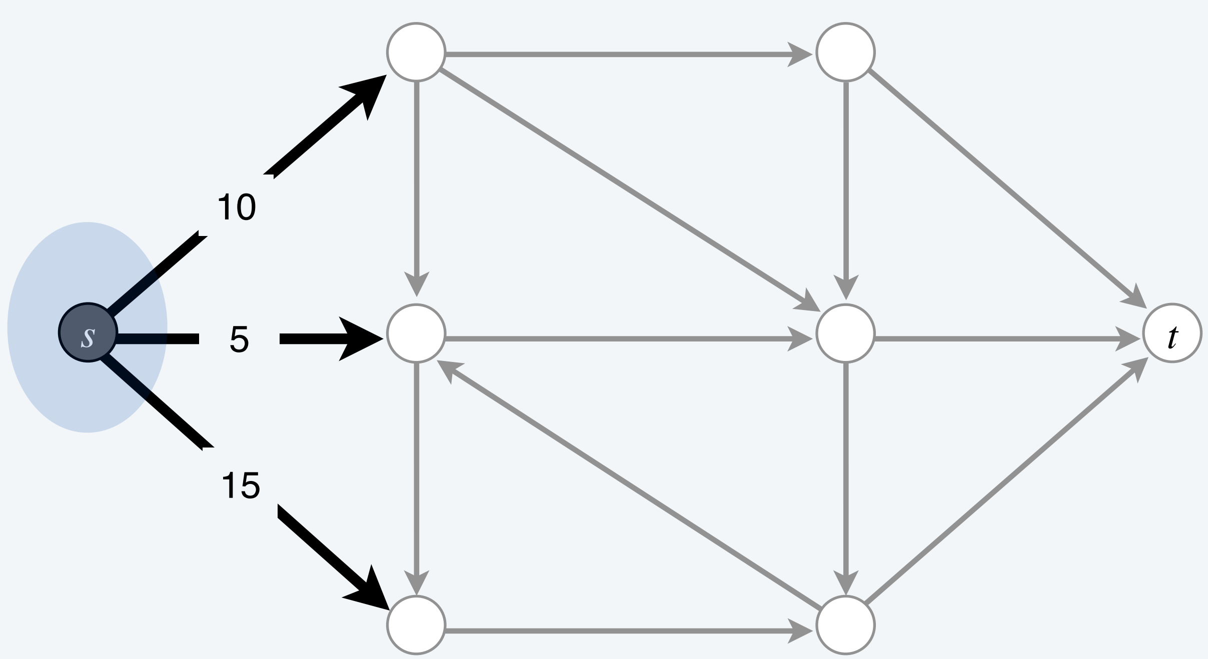
Pf.

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

flow value lemma




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



# Certificate of optimality

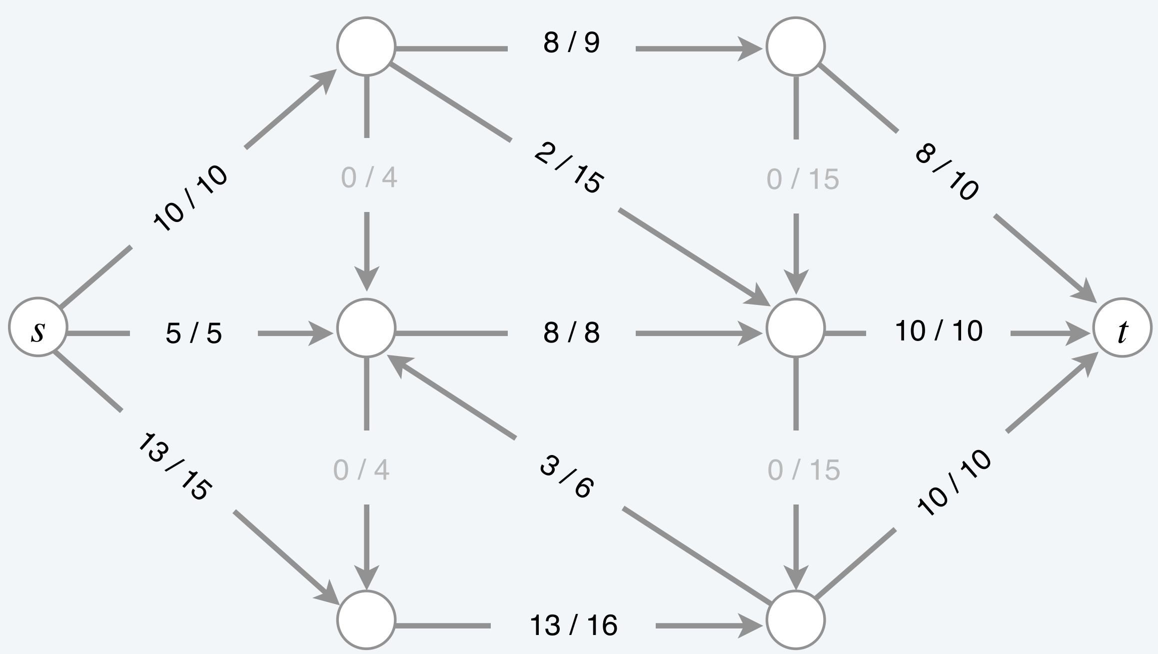
Corollary. Let $f$ be a flow and let $(A, B)$ be any cut.
If $val(f) = cap(A, B)$, then $f$ is a max flow and $(A, B)$ is a min cut.

Pf.

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

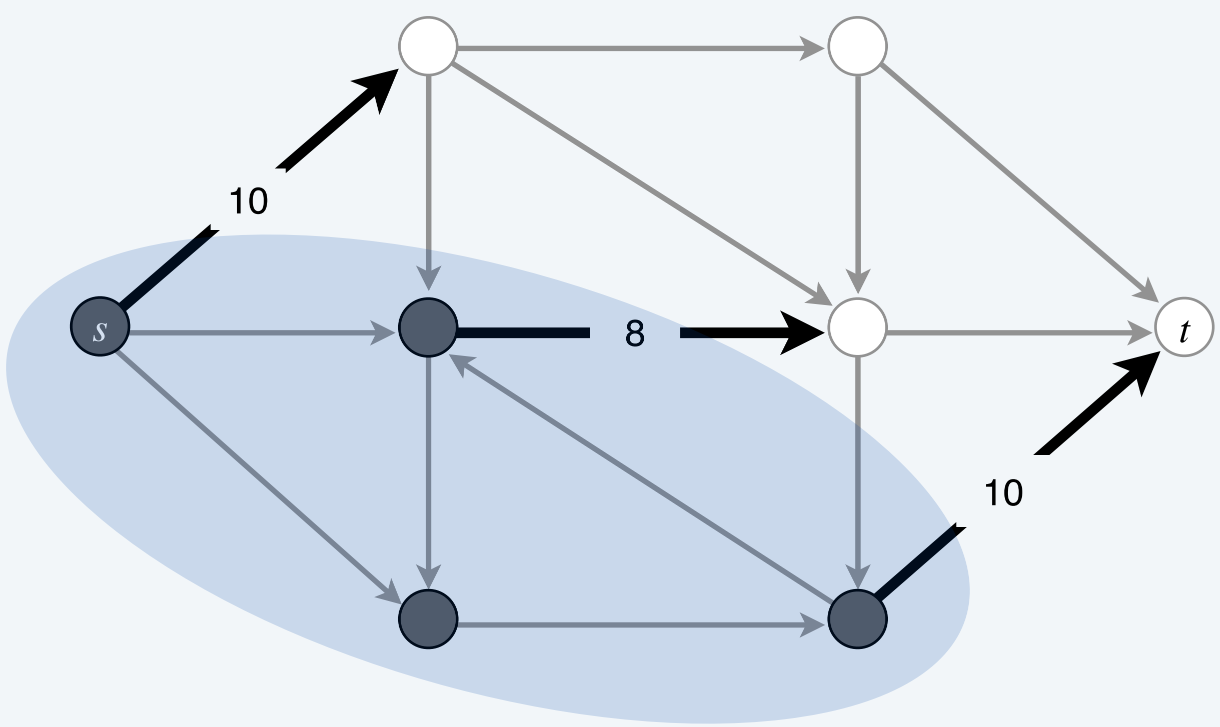
weak duality

weak duality




value of flow = 28

=




capacity of cut = 28

# Max-flow min-cut theorem

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

strong duality

**MAXIMAL FLOW THROUGH A NETWORK**

L. R. FORD, JR. AND D. R. FULKERSON

**Introduction.** The problem discussed in this paper was formulated by T. Harris as follows:

"Consider a rail network connecting two cities by way of a number of intermediate cities, where each link of the network has a number assigned to it representing its capacity. Assuming a steady state condition, find a maximal flow from one given city to the other."

**ON THE MAX FLOW MIN CUT THEOREM OF NETWORKS**

G. B. Dantzig
D. R. Fulkerson

P-826

April 15, 1955

## A Note on the Maximum Flow Through a Network*

P. ELIAS†, A. FEINSTEIN‡, AND C. E. SHANNON§

*Summary*—This note discusses the problem of maximizing the rate of flow from one terminal to another, through a network which consists of a number of branches, each of which has a limited capacity. The main result is a theorem: The maximum possible flow from left to right through a network is equal to the minimum value among all simple cut-sets. This theorem is applied to solve a more general problem, in which a number of input nodes and a number of output nodes are used.

from one terminal to the other in the original network passes through at least one branch in the cut-set. In the network above, some examples of cut-sets are $(d, e, f)$, and $(b, c, e, g, h)$, $(d, g, h, i)$. By a *simple cut-set* we will mean a cut-set such that if any branch is omitted it is no longer a cut-set. Thus $(d, e, f)$ and $(b, c, e, g, h)$ are simple

# 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 $f$: $t \notin A$.

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

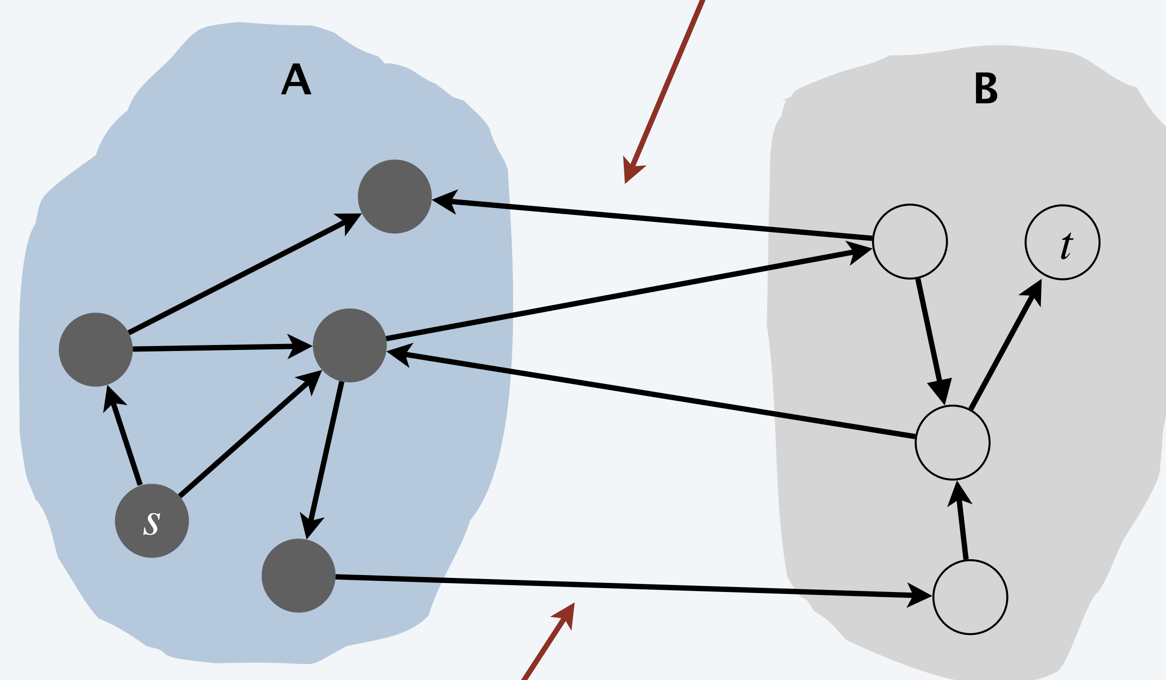
flow value lemma

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

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

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

original flow network G



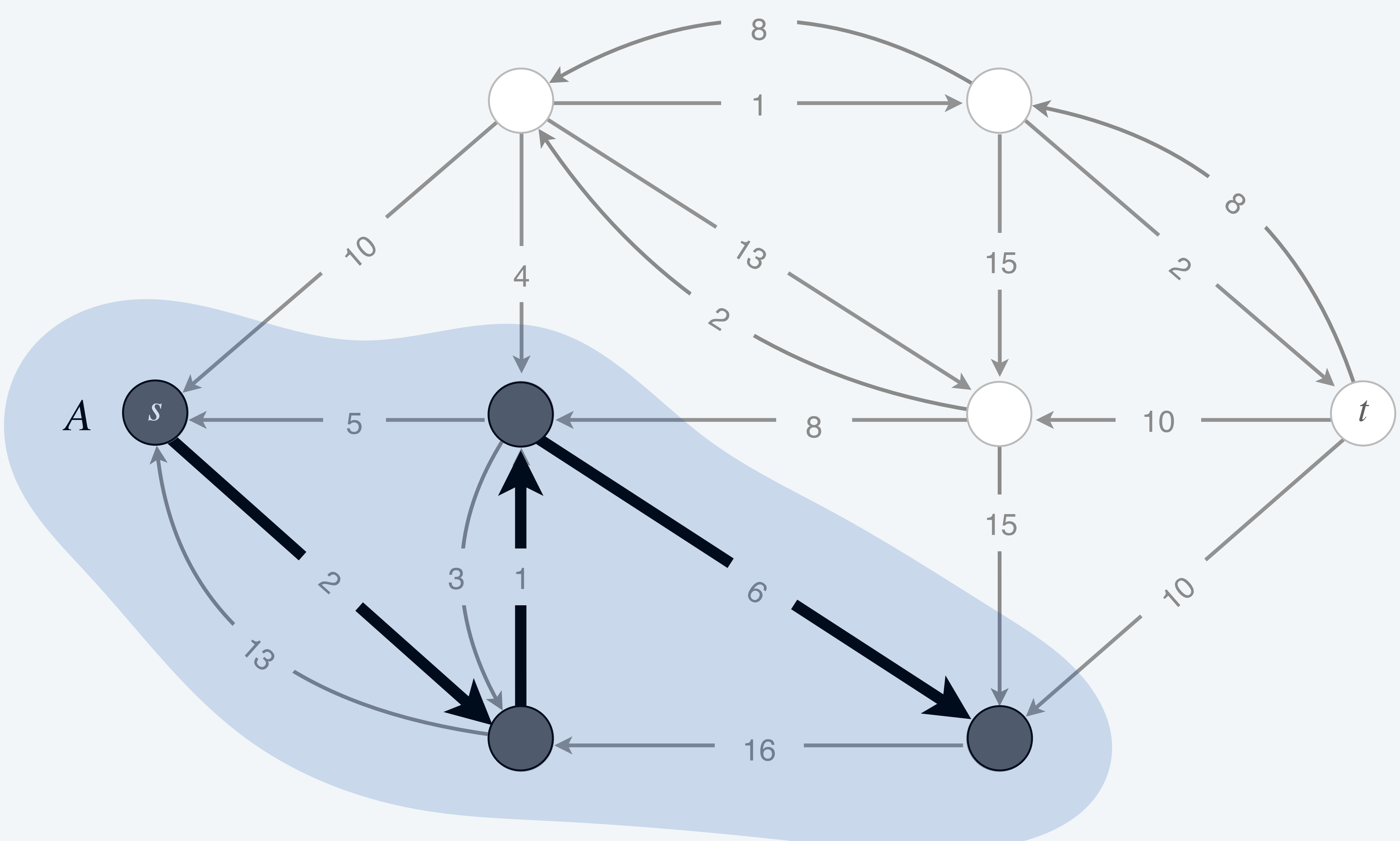

edge $e = (v, w)$ with $v \in A, w \in B$
must have $f(e) = c(e)$

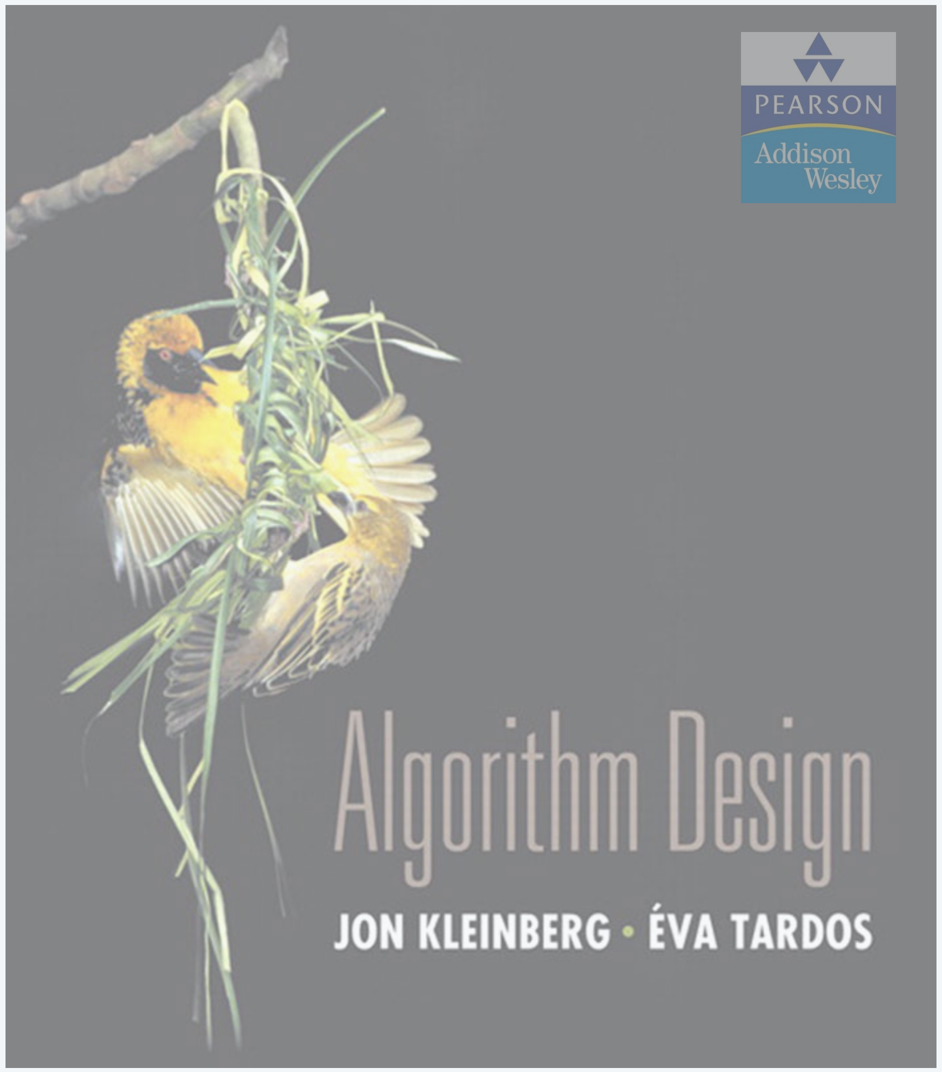
# Computing a minimum cut from a maximum flow

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 $f$






SECTION 7.3

# 7. NETWORK FLOW I

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

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

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

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

Pf. By induction on the number of augmenting paths. ▪

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

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

Pf. Each augmentation increases the value of the flow by at least $1$. ▪

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

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

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

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

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

# Ford–Fulkerson: exponential example

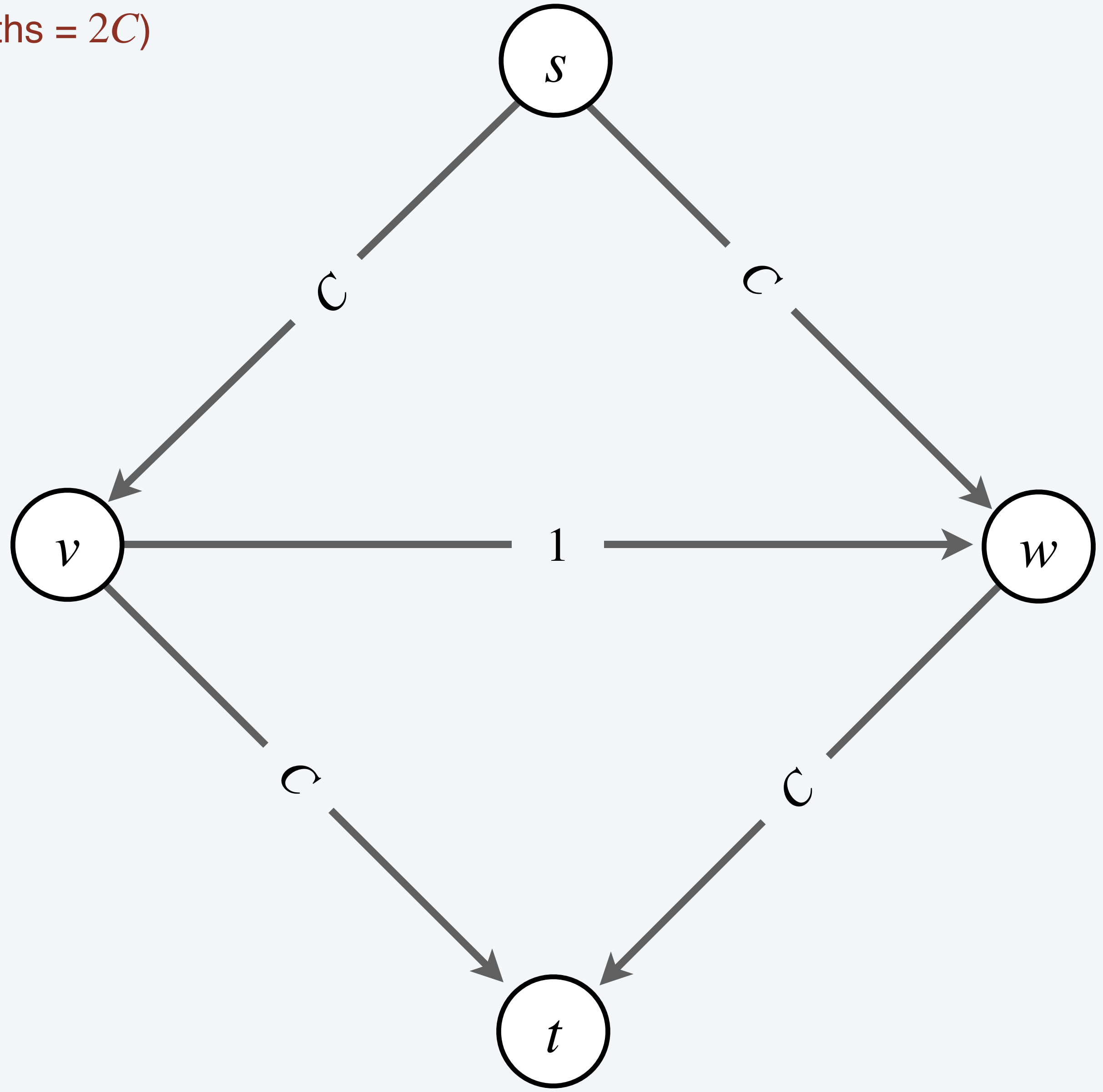
Q. Is generic Ford–Fulkerson algorithm poly-time in input size?

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

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

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

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



# Network flow: quiz 4

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

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

# Choosing good augmenting paths

Use care when selecting augmenting paths.

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

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

Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

# Choosing good augmenting paths

Choose augmenting paths with:

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

**Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems**

JACK EDMONDS

*University of Waterloo, Waterloo, Ontario, Canada*

AND

RICHARD M. KARP

*University of California, Berkeley, California*

ABSTRACT. This paper presents new algorithms for the maximum flow problem, the Hitchcock transportation problem, and the general minimum-cost flow problem. Upper bounds on the numbers of steps in these algorithms are derived, and are shown to compare favorably with upper bounds on the numbers of steps required by earlier algorithms.

**Edmonds-Karp 1972 (USA)**

Dokl. Akad. Nauk SSSR
Tom 194 (1970), No. 4

Soviet Math. Dokl.
Vol. 11 (1970), No. 5

**ALGORITHM FOR SOLUTION OF A PROBLEM OF MAXIMUM FLOW IN A NETWORK WITH POWER ESTIMATION**

UDC 518.5

E. A. DINIC

Different variants of the formulation of the problem of maximal stationary flow in a network and its many applications are given in [1]. There also is given an algorithm solving the problem in the case where the initial data are integers (or, what is equivalent, commensurable). In the general case this algorithm requires preliminary rounding off of the initial data, i.e. only an approximate solution of the problem is possible. In this connection the rapidity of convergence of the algorithm is inversely proportional to the relative precision.

**Dinitz 1970 (Soviet Union)**

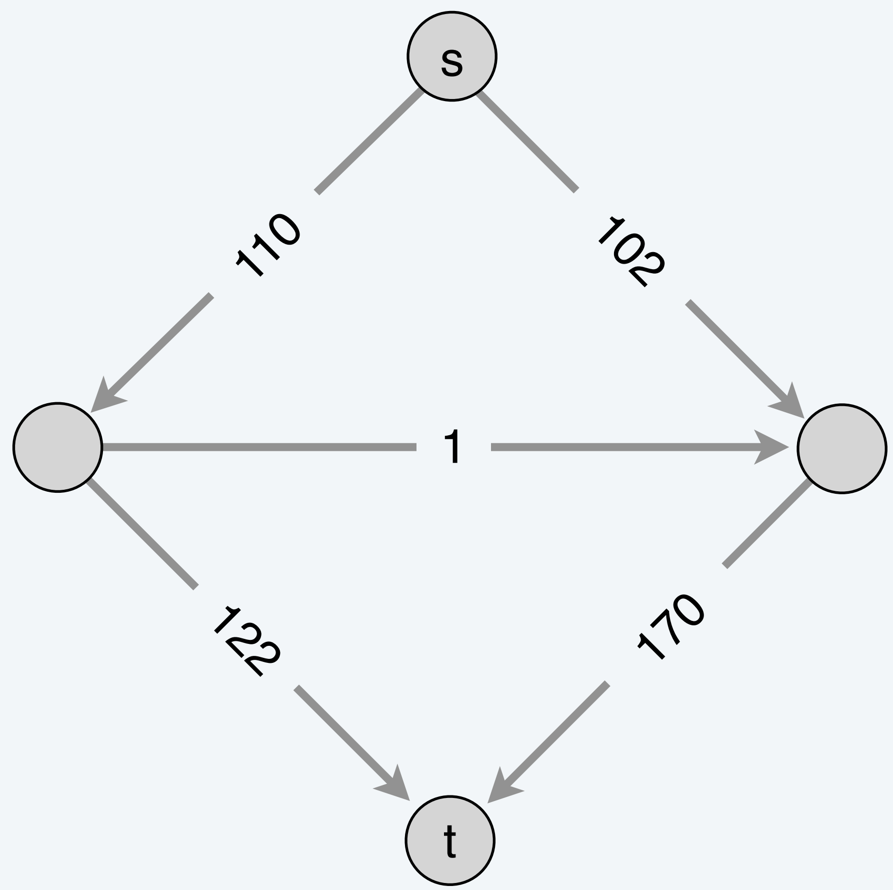
↑ invented in response to a class exercises by Adel'son-Vel'skiĭ

# Capacity-scaling algorithm

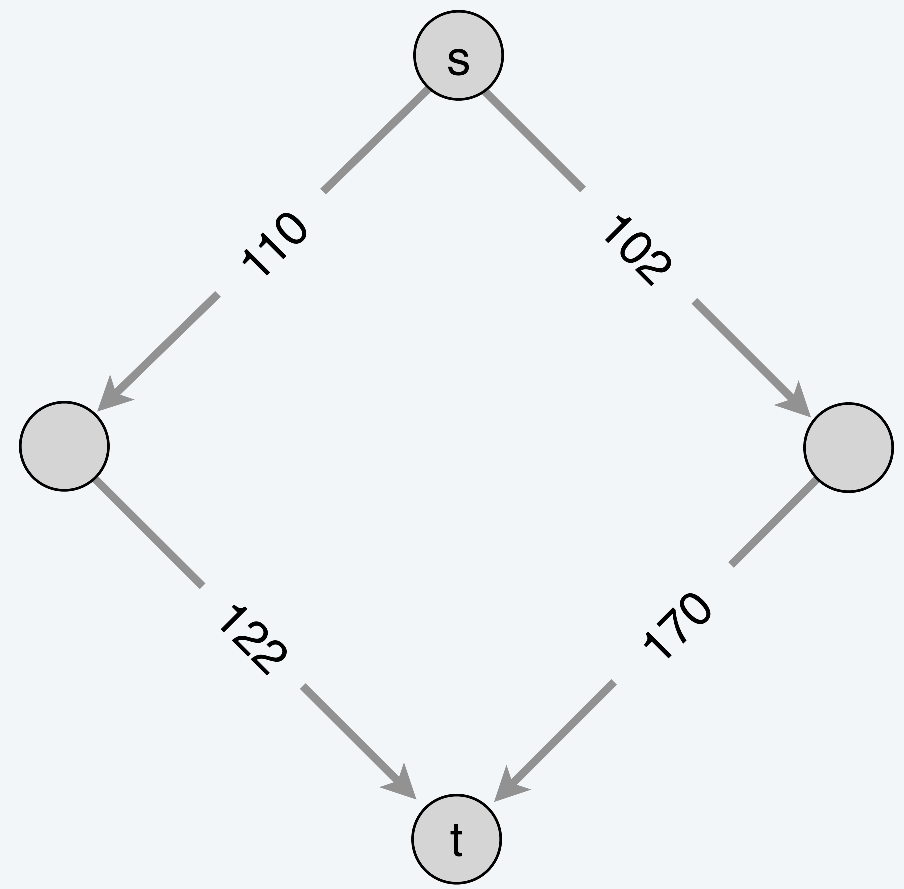
Overview. Choosing augmenting paths with "large" bottleneck capacity.

though not necessarily largest

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




$G_f$




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

# Capacity-scaling algorithm

CAPACITY-SCALING($G$)

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

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

WHILE ($\Delta \geq 1$)

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

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

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

    Update $G_f(\Delta)$.

  $\Delta \leftarrow \Delta / 2$.

Δ-scaling phase

RETURN $f$.

# Capacity-scaling algorithm: proof of correctness

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

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

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

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

Pf. Same as for generic Ford–Fulkerson. ▪

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

Pf.

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

# Capacity-scaling algorithm: analysis of running time

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

Pf. Initially $C/2 < \Delta \le C$; $\Delta$ decreases by a factor of $2$ in each iteration. ▪

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

Then, the max-flow value $\le val(f) + m\,\Delta$.

Pf. Next slide.

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

Pf.

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

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

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

Pf.

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

# Capacity-scaling algorithm: analysis of running time

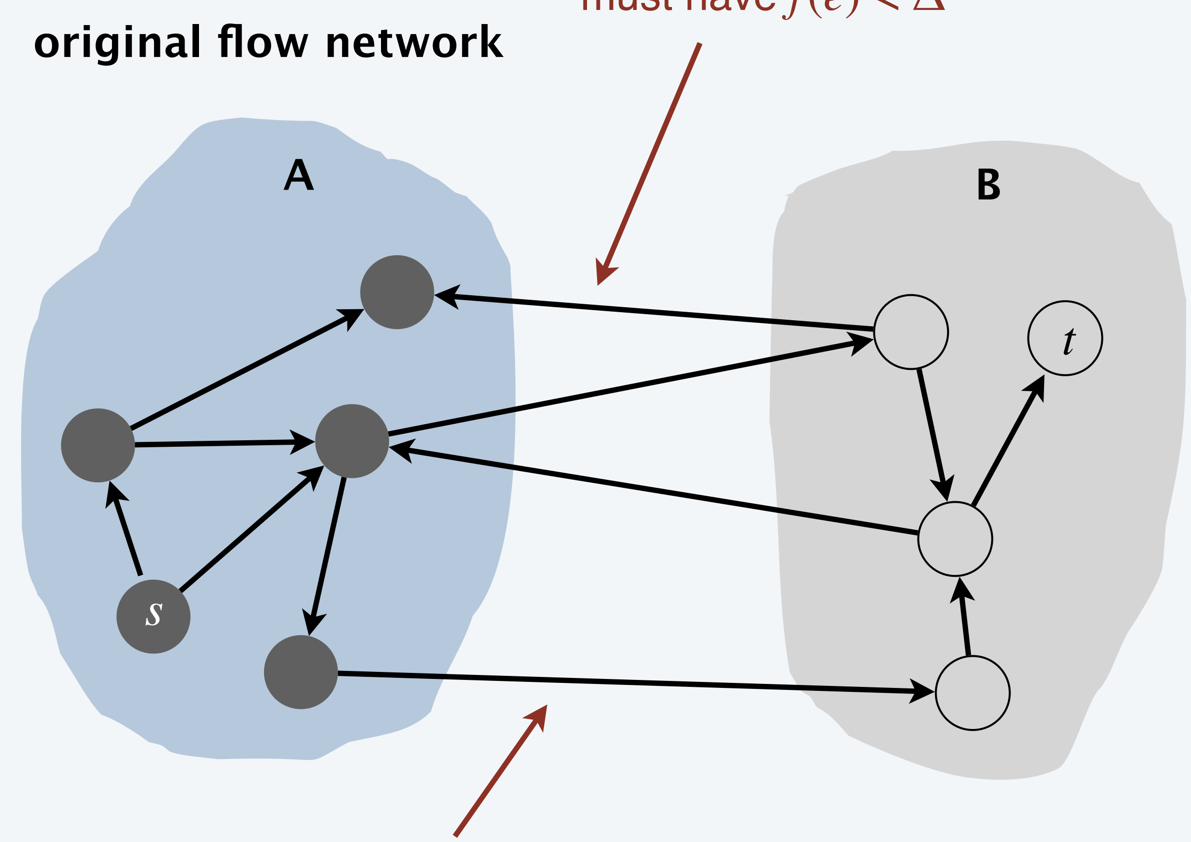
**Lemma 2.** Let $f$ be the flow at the end of a $\Delta$-scaling phase.
Then, the max-flow value $\leq val(f) + m\,\Delta$.

**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 $A$: $s \in A$.
- By definition of flow $f$: $t \notin A$.

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

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*
- *Dinitz' algorithm*
- ***simple unit-capacity networks***

# Network flow: quiz 7

**Which max-flow algorithm to use for bipartite matching?**

**A.** Ford–Fulkerson: $O(m\,n\,C)$.

**B.** Capacity scaling: $O(m^2 \log C)$.

**C.** Shortest augmenting path: $O(m^2\,n)$.

**D.** Dinitz' algorithm: $O(m\,n^2)$.

# Simple unit-capacity networks

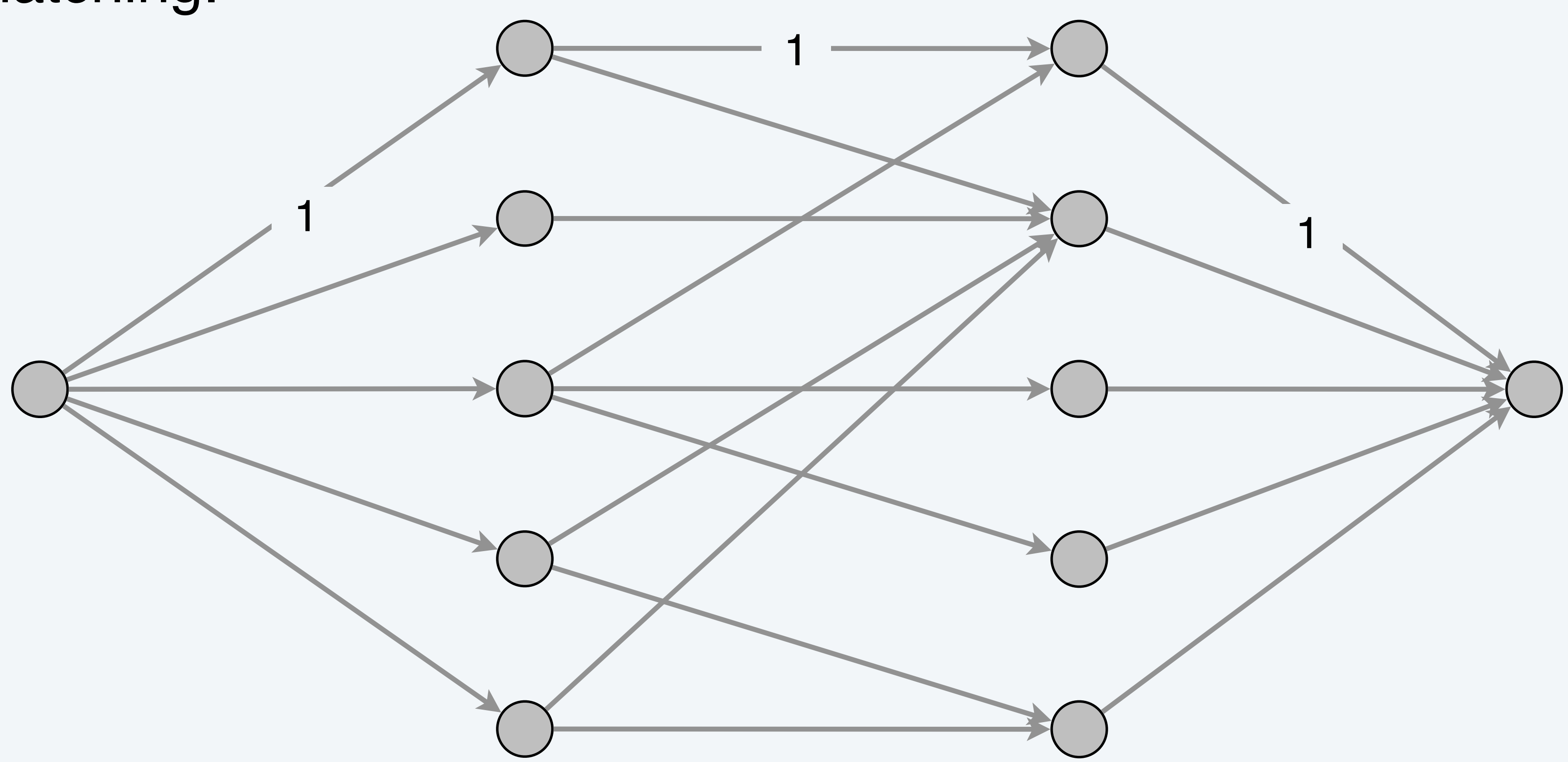
Def. A flow network is a simple unit-capacity network if:

- Every edge has capacity $1$.
- Every node (other than $s$ or $t$) has exactly one entering edge, or exactly one leaving edge, or both.

node capacity = 1

Property. Let $G$ be a simple unit-capacity network and let $f$ be a $0$–$1$ flow. Then, residual network $G_f$ is also a simple unit-capacity network.

Ex. Bipartite matching.



# Simple unit-capacity networks

Shortest-augmenting-path algorithm.

- Normal augmentation: length of shortest path does not change.
- Special augmentation: length of shortest path strictly increases.

Theorem. [Even–Tarjan 1975] In simple unit-capacity networks, Dinitz' algorithm computes a maximum flow in $O(m\, n^{1/2})$ time.

Pf.

- Lemma 1. Each phase of normal augmentations takes $O(m)$ time.
- Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.
- Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal. ▪

Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal.

Pf. Each augmentation increases flow value by at least 1. ▪

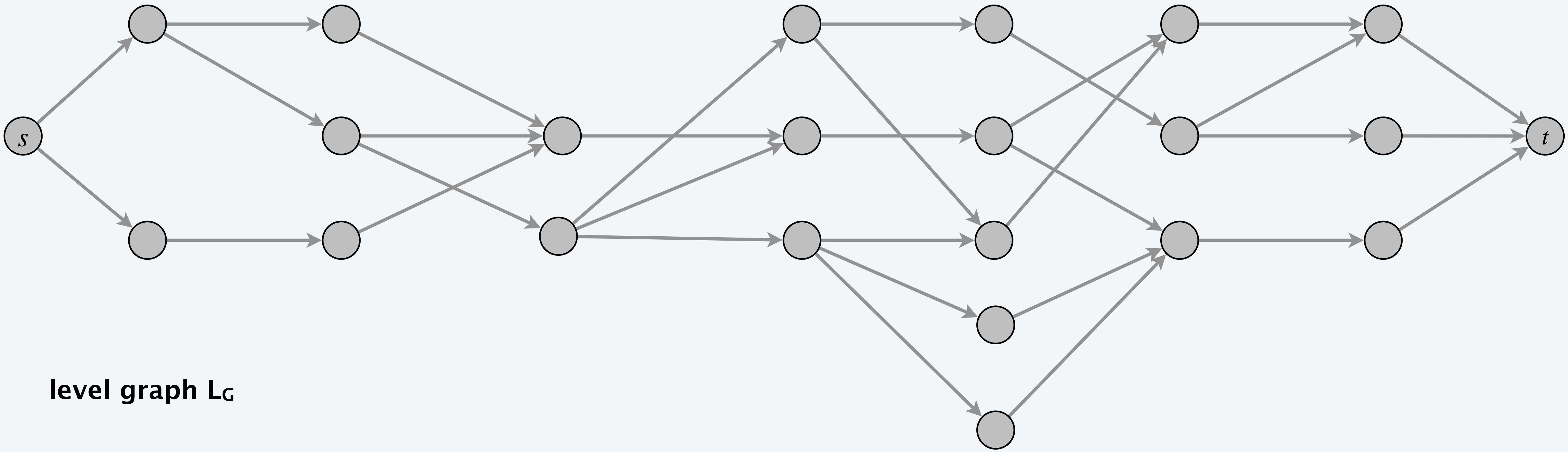
Lemma 1 and Lemma 2. Ahead.

# Simple unit-capacity networks

**Phase of normal augmentations.** ← within a phase, length of shortest augmenting path does not change

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

**construct level graph**



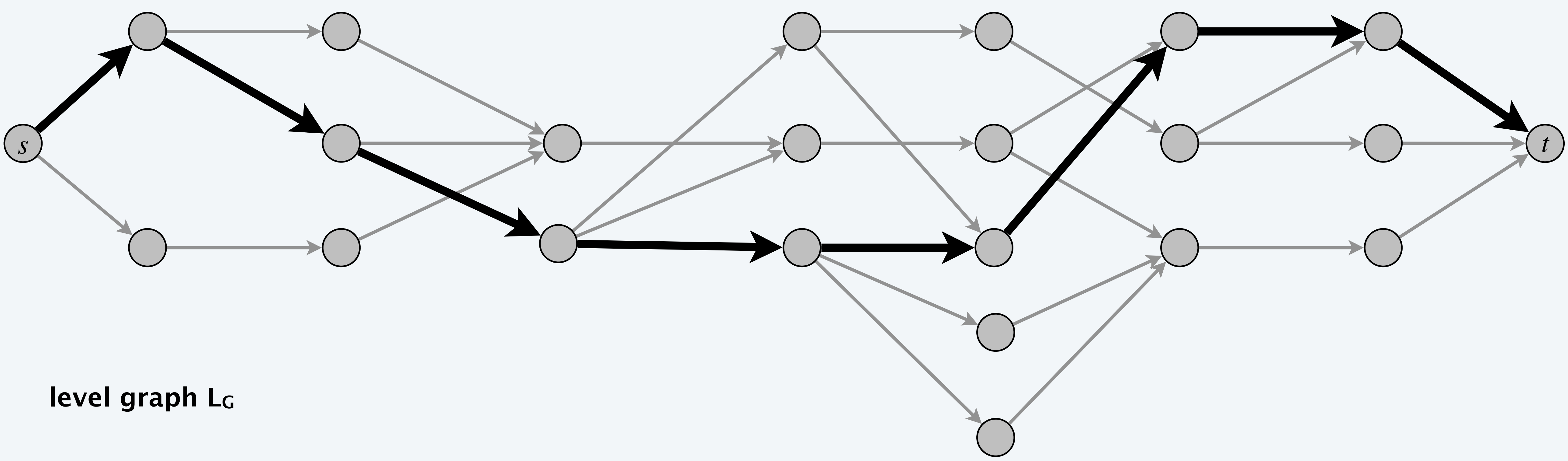

**level graph $L_G$**

# Simple unit-capacity networks

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

**advance**

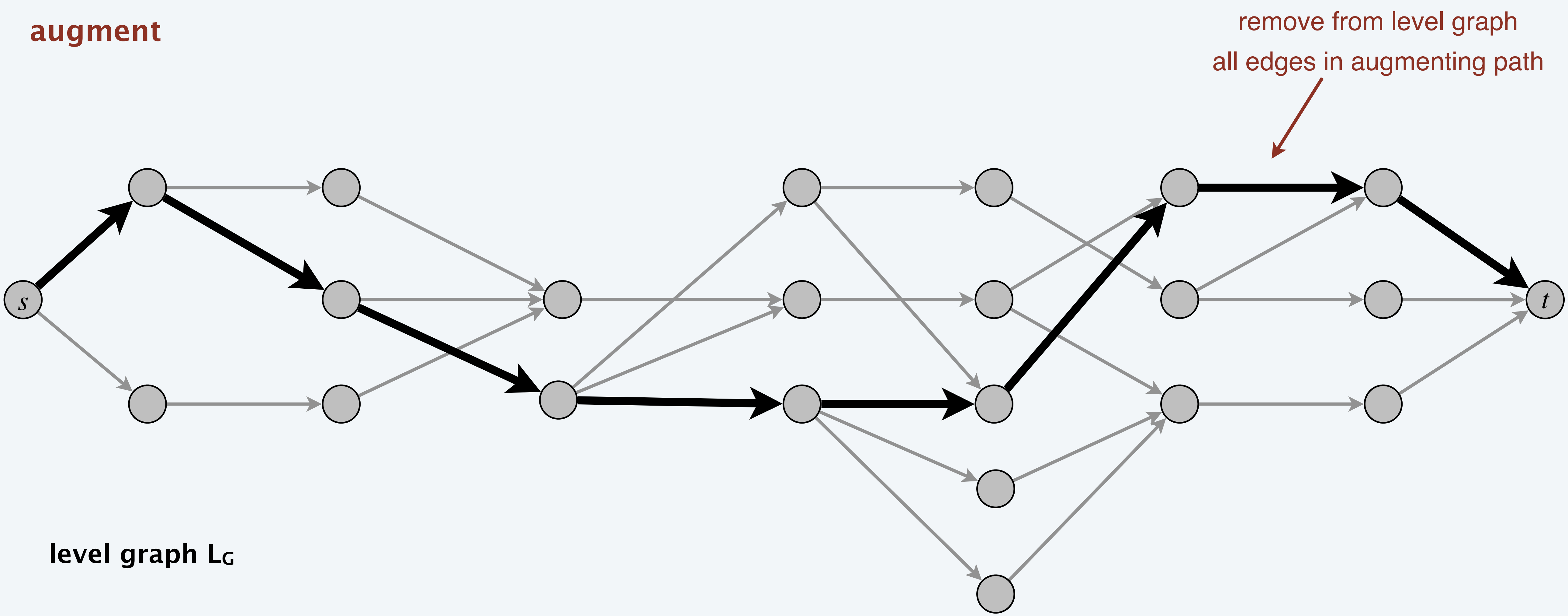



**level graph $L_G$**

# Simple unit-capacity networks

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

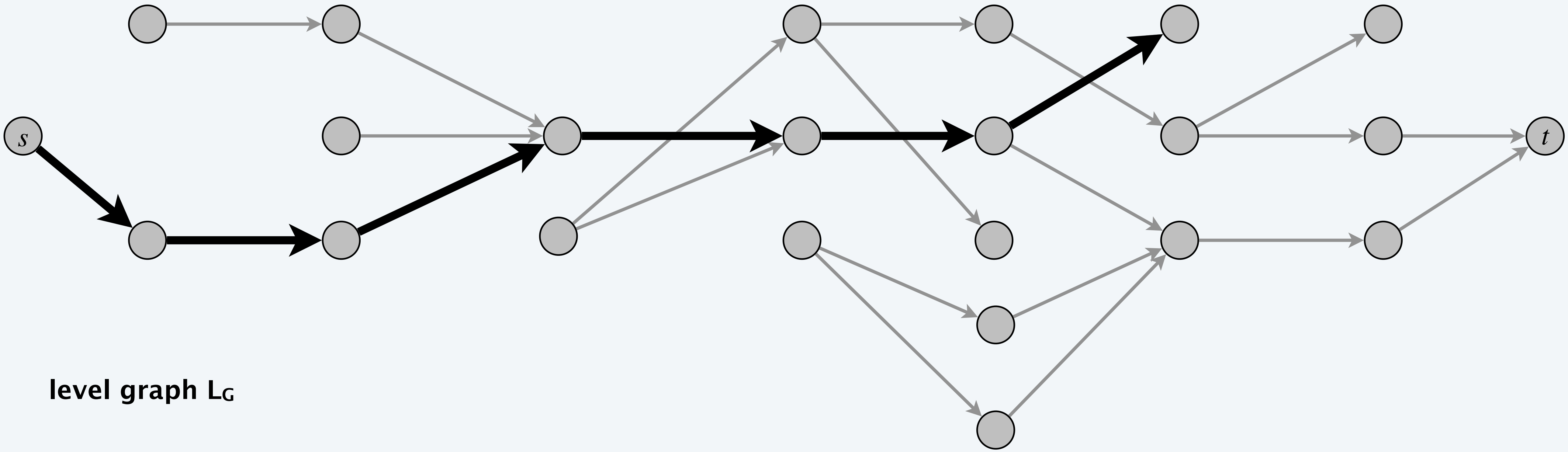


# Simple unit-capacity networks

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

advance



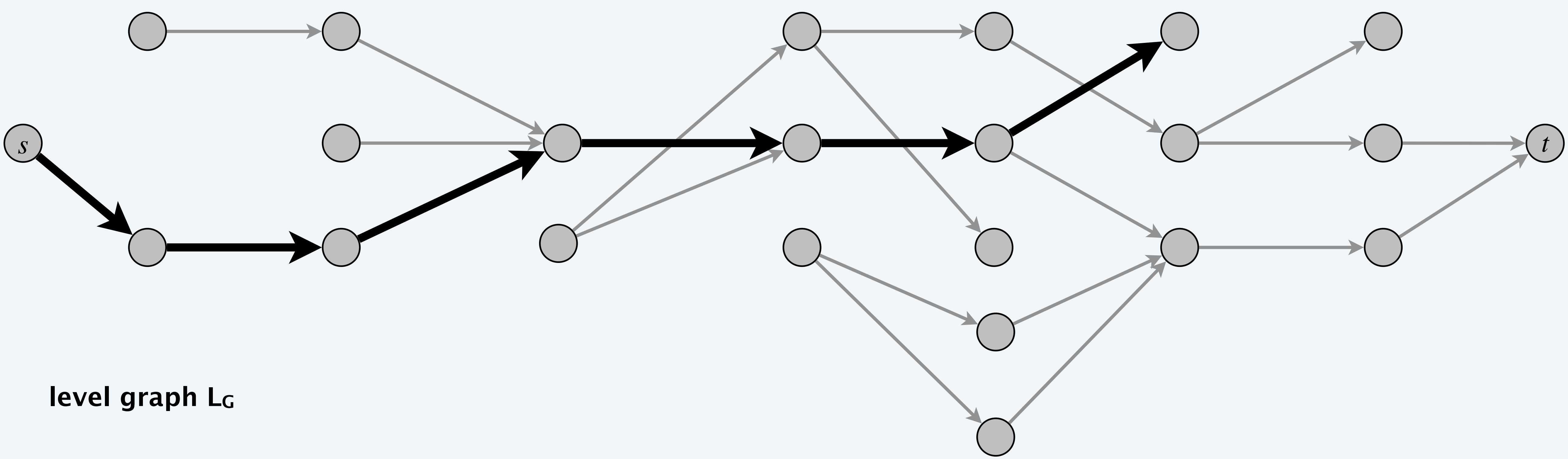

level graph $L_G$

# Simple unit-capacity networks

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

**retreat**



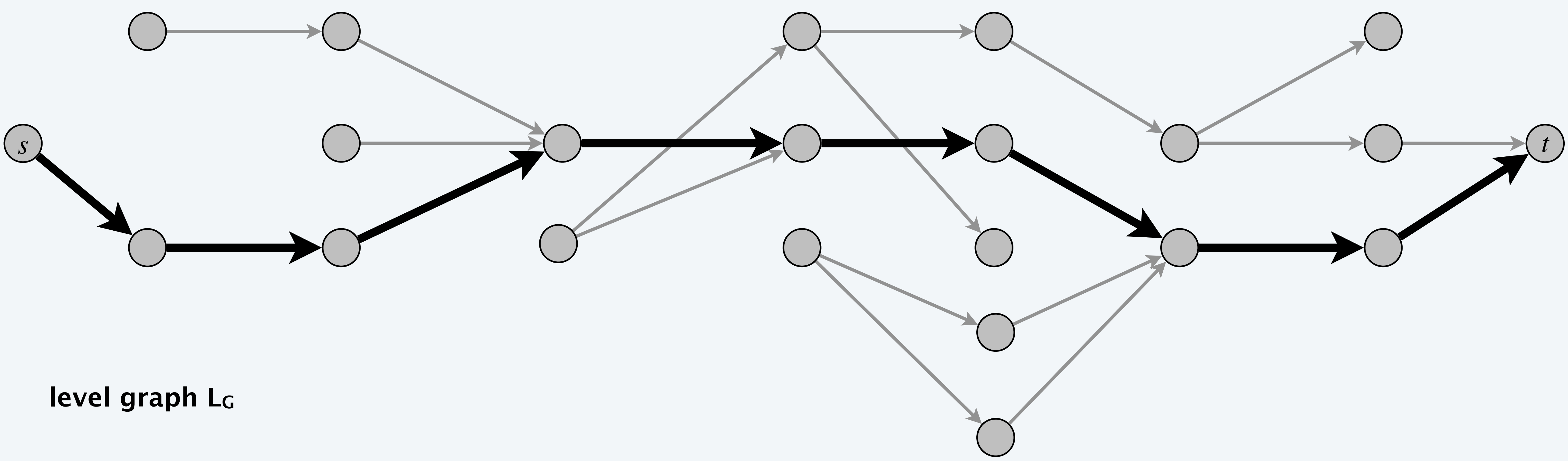

**level graph $L_G$**

# Simple unit-capacity networks

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

advance

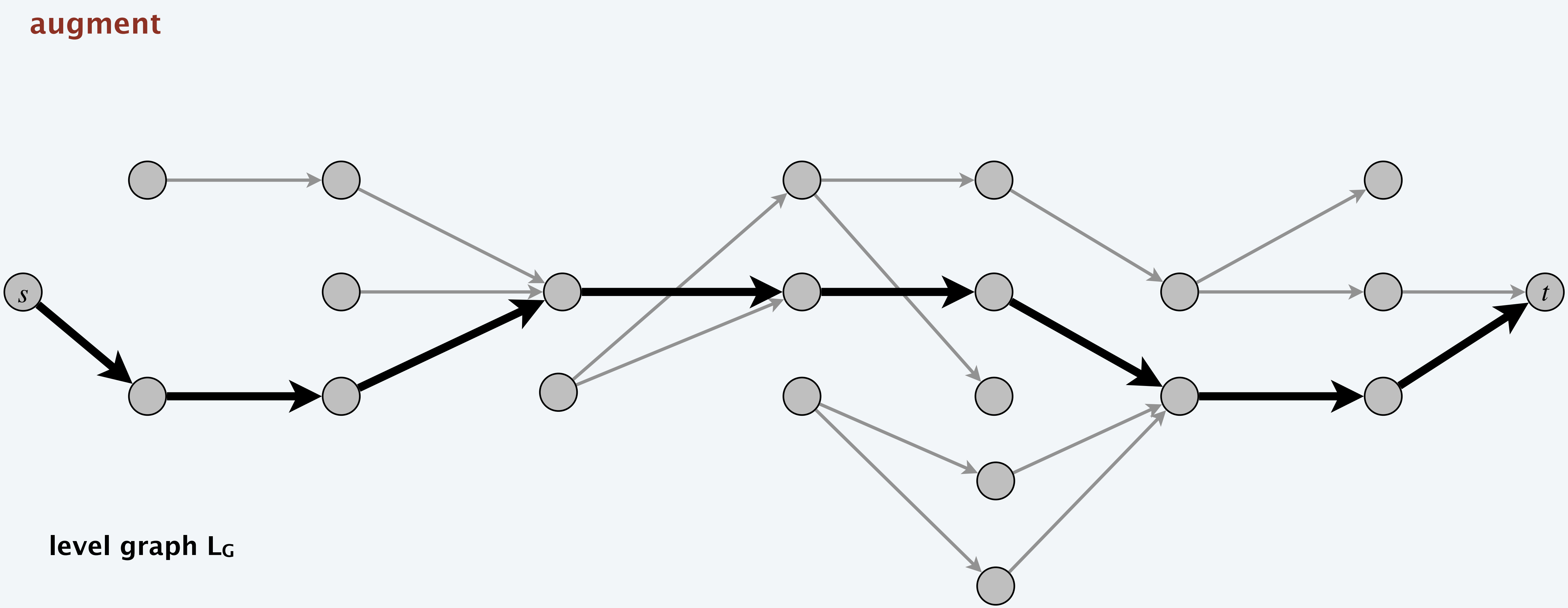



level graph $L_G$

# Simple unit-capacity networks

Phase of normal augmentations.

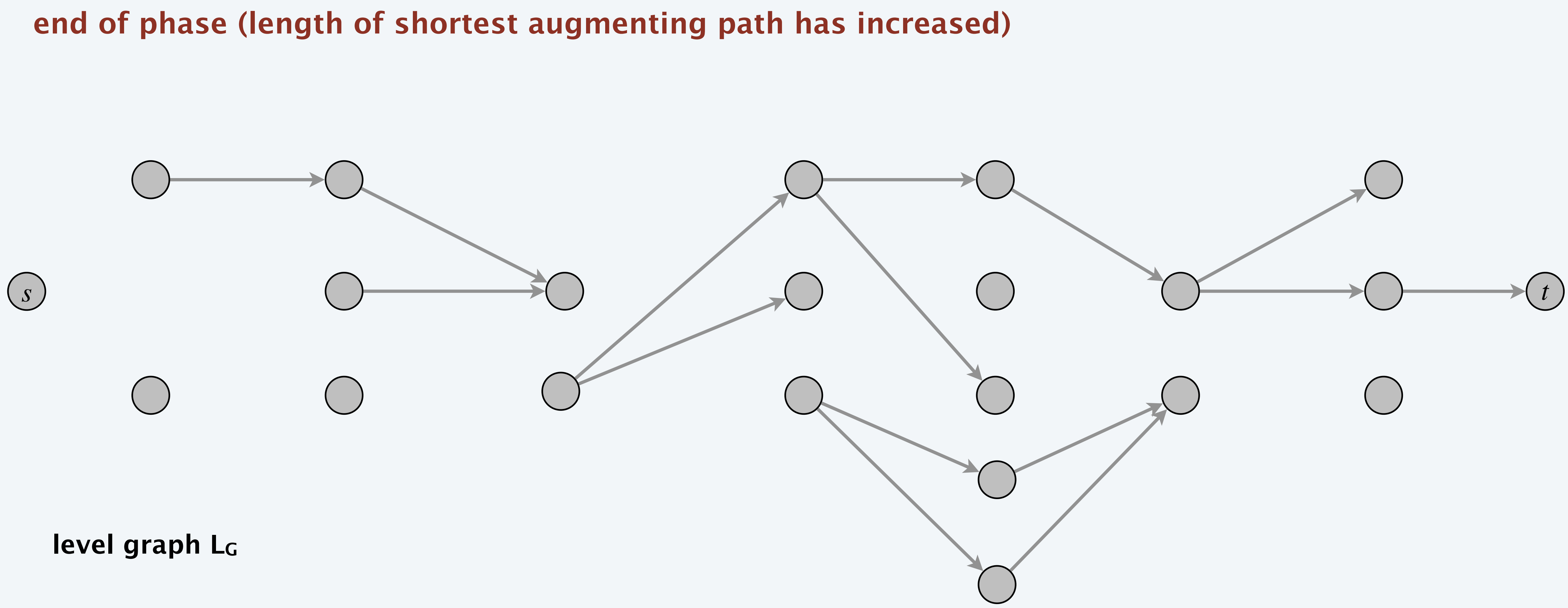
- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.



# Simple unit-capacity networks

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.



# Simple unit-capacity networks: analysis

Phase of normal augmentations.

- Construct level graph $L_G$.
- Start at $s$, advance along an edge in $L_G$ until reach $t$ or get stuck.
- If reach $t$, augment flow; update $L_G$; and restart from $s$.
- If get stuck, delete node from $L_G$ and go to previous node.

Lemma 1. A phase of normal augmentations takes $O(m)$ time.

Pf.

- $O(m)$ to create level graph $L_G$.
- $O(1)$ per edge (each edge involved in at most one advance, retreat, and augmentation).
- $O(1)$ per node (each node deleted at most once). ▪

**Consider running advance-retreat algorithm in a unit-capacity network (but not necessarily a simple one). What is running time?**

both indegree and outdegree
of a node can be larger than 1

**A.** $O(m)$.

**B.** $O(m^{3/2})$.

**C.** $O(m\,n)$.

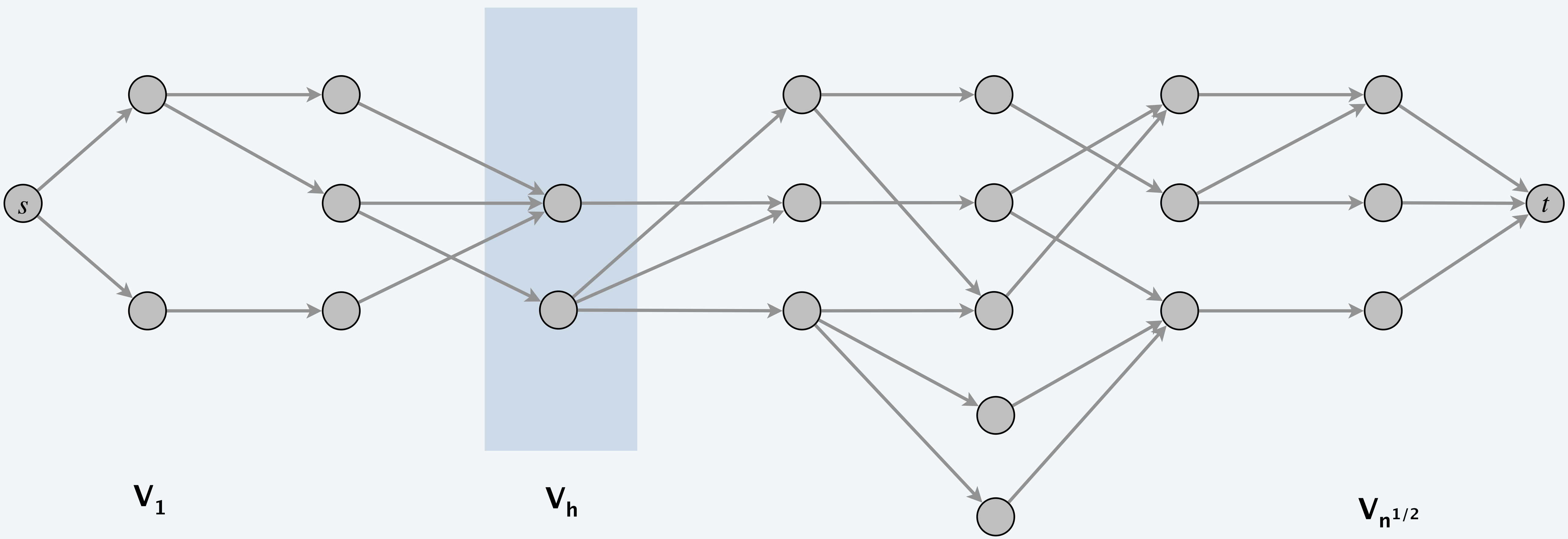
**D.** May not terminate.

# Simple unit-capacity networks: analysis

Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.

- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Thus, level graph has $\geq n^{1/2}$ levels (not including levels for $s$ or $t$).
- Let $1 \leq h \leq n^{1/2}$ be a level with min number of nodes $\Rightarrow |V_h| \leq n^{1/2}$.

level graph $L_G$ for flow f

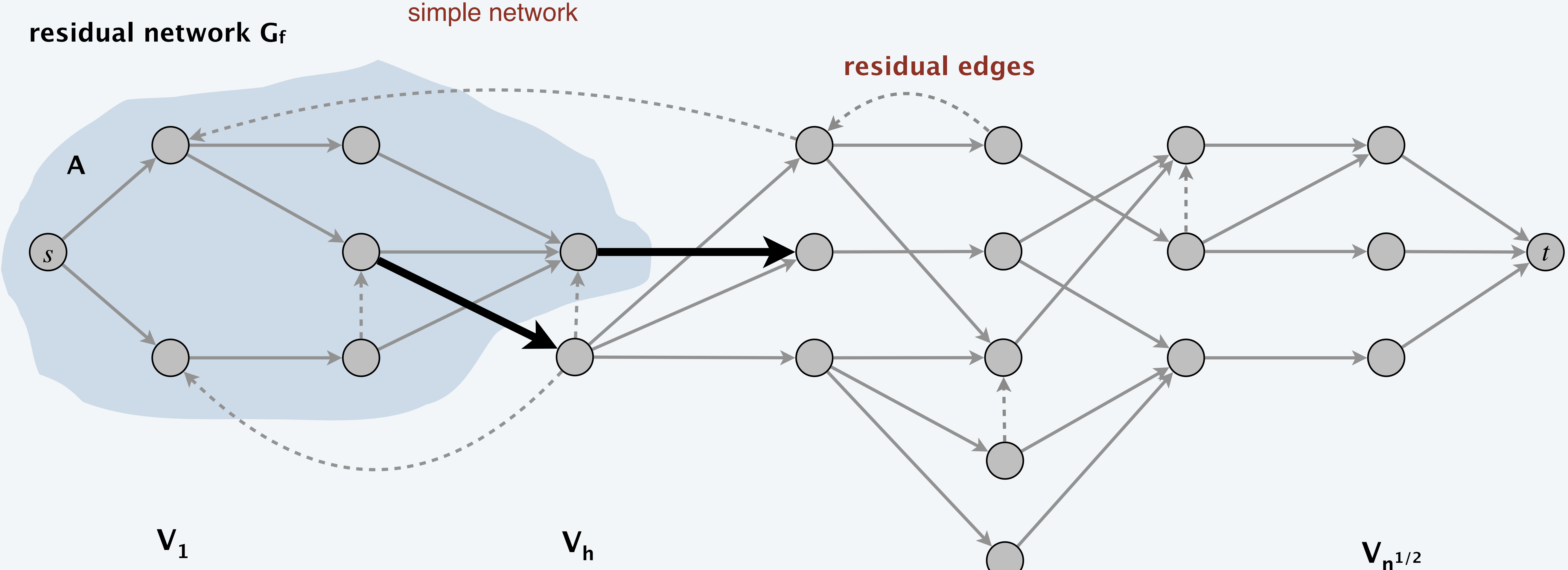


# Simple unit-capacity networks: analysis

Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.

- After $n^{1/2}$ phases, length of shortest augmenting path is $> n^{1/2}$.
- Thus, level graph has $\geq n^{1/2}$ levels (not including levels for $s$ or $t$).
- Let $1 \leq h \leq n^{1/2}$ be a level with min number of nodes $\Rightarrow |V_h| \leq n^{1/2}$.
- Let $A = \{v : \ell(v) < h\} \cup \{v : \ell(v) = h$ and $v$ has $\leq 1$ outgoing residual edge$\}$.
- $cap_f(A, B) \leq |V_h| \leq n^{1/2} \Rightarrow val(f) \geq val(f^*) - n^{1/2}$. ▪

unit-capacity simple network



## Simple unit-capacity networks: review

Theorem. [Even–Tarjan 1975] In simple unit-capacity networks, Dinitz' algorithm computes a maximum flow in $O(m\, n^{1/2})$ time.

Pf.

- Lemma 1. Each phase takes $O(m)$ time.
- Lemma 2. After $n^{1/2}$ phases, $val(f) \geq val(f^*) - n^{1/2}$.
- Lemma 3. After $\leq n^{1/2}$ additional augmentations, flow is optimal. ▪

Corollary. Dinitz' algorithm computes max-cardinality bipartite matching in $O(m\, n^{1/2})$ time.