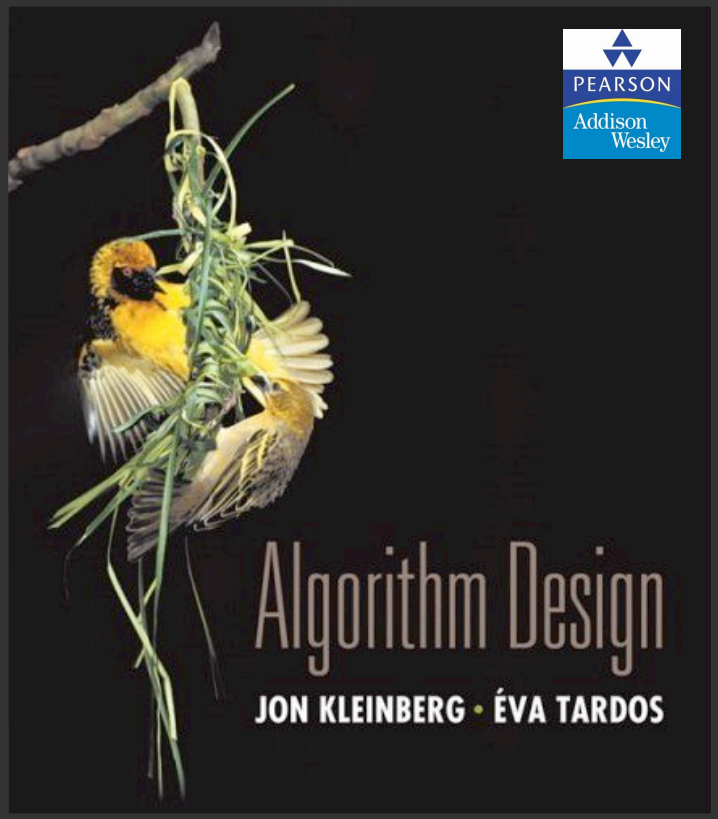


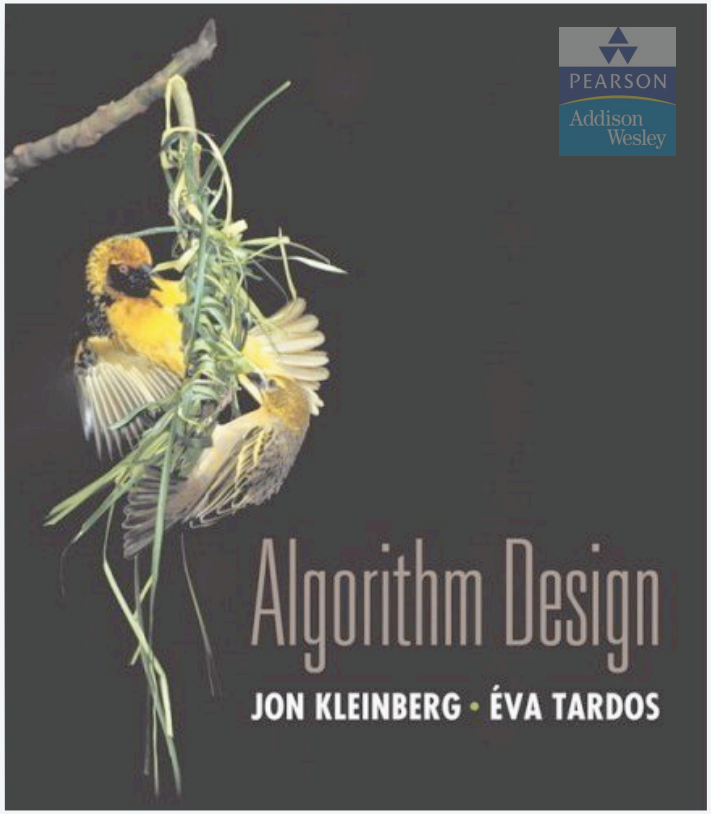
7. Network Flow I

- *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*

Lecture slides by Kevin Wayne

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

Last updated on Sep 8, 2013 6:40 AM



SECTION 7.1

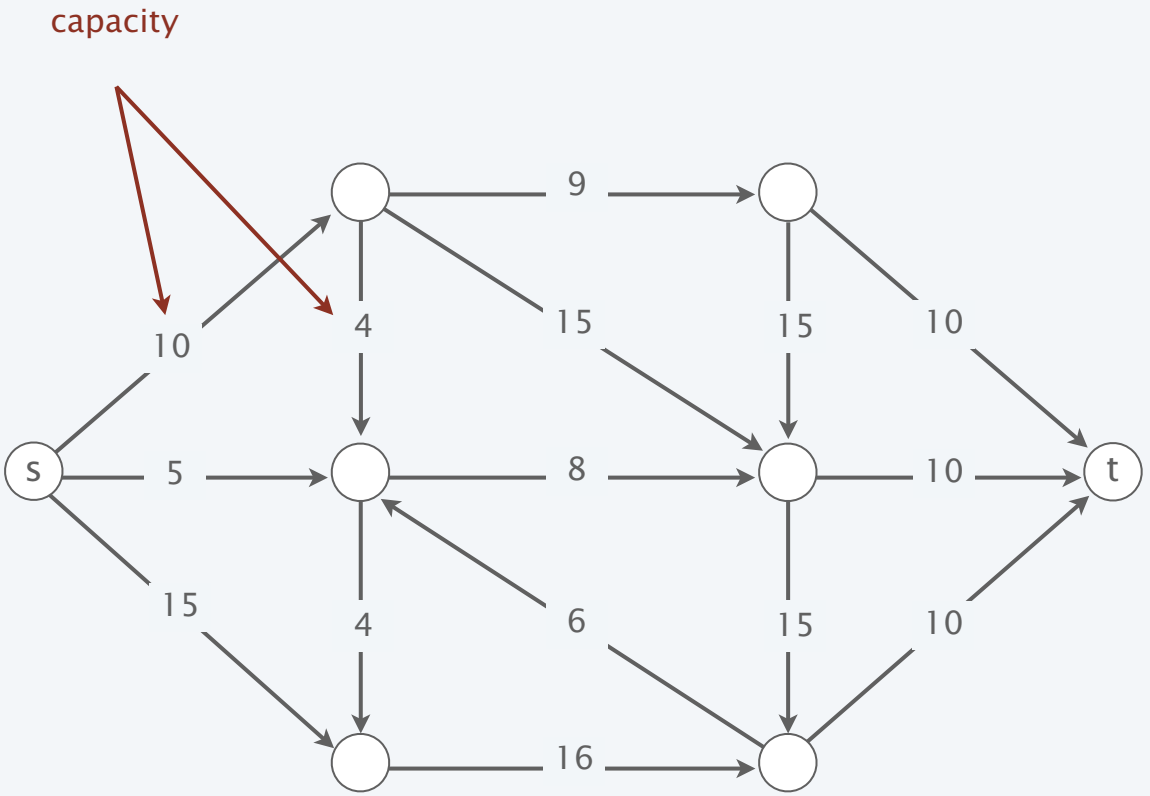
7. NETWORK FLOW I

- *max-flow and min-cut problems*
- *Ford-Fulkerson algorithm*
- *max-flow min-cut theorem*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *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 $e \in E$.

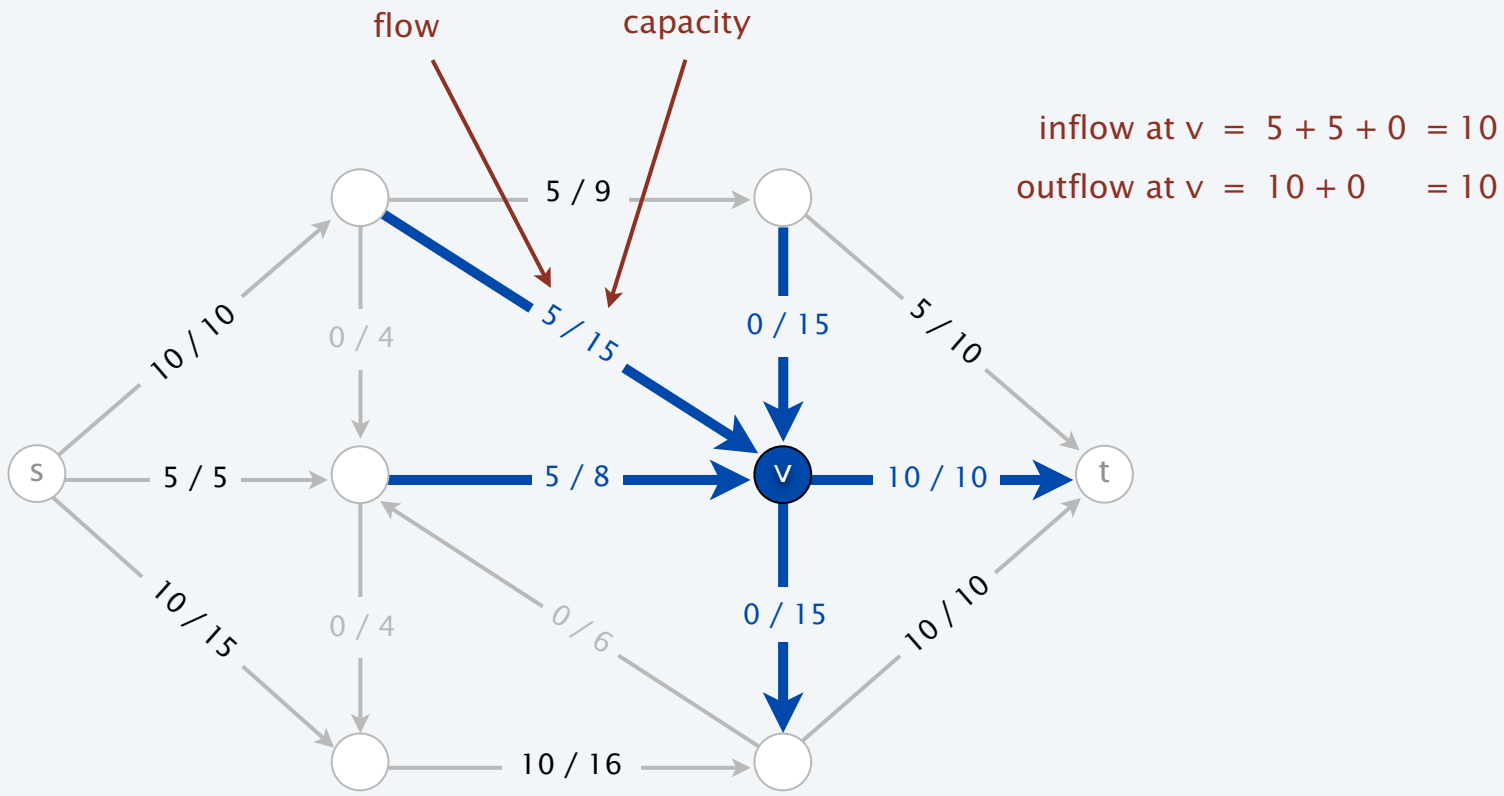
no parallel edges
no edge enters s
no edge leaves t



Maximum flow problem

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

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

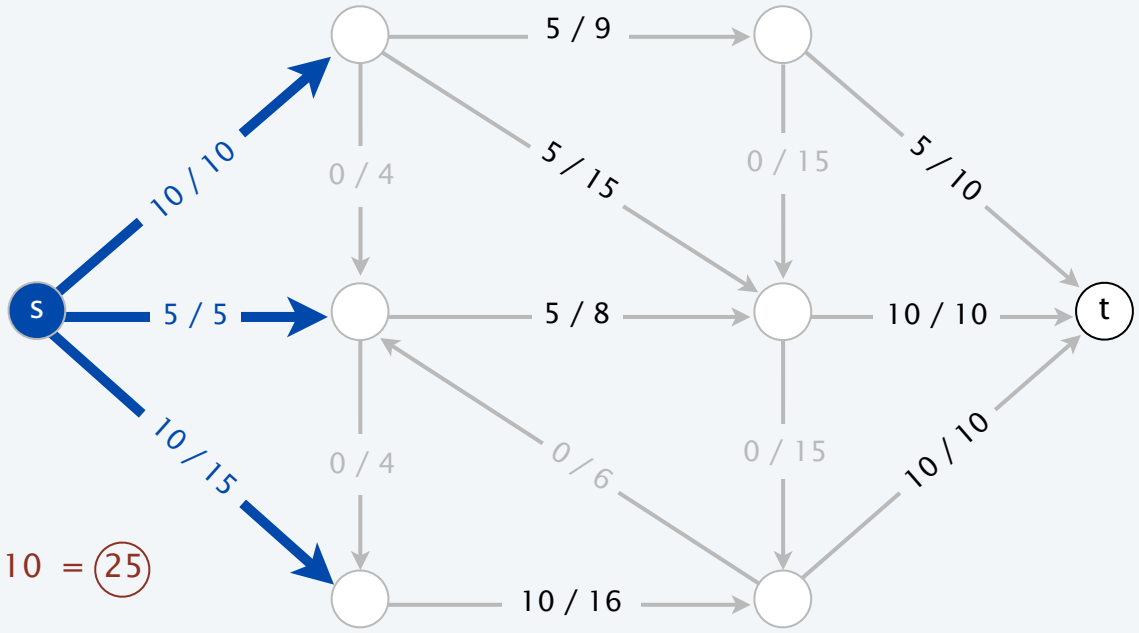


Maximum flow problem

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

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



value = 5 + 10 + 10 = 25

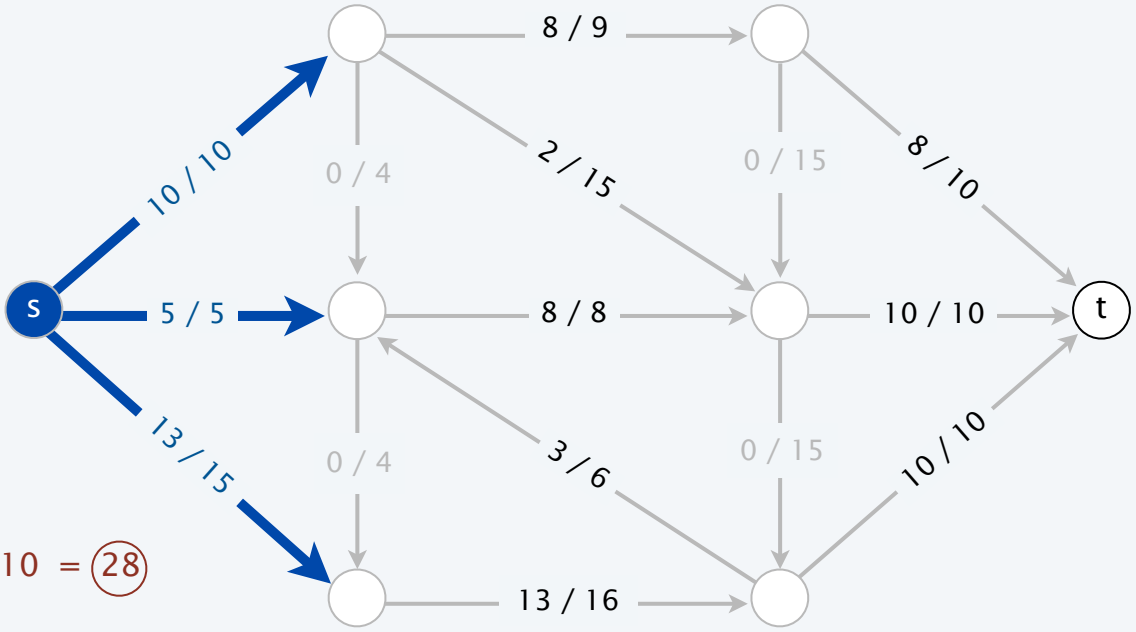
Maximum flow problem

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

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



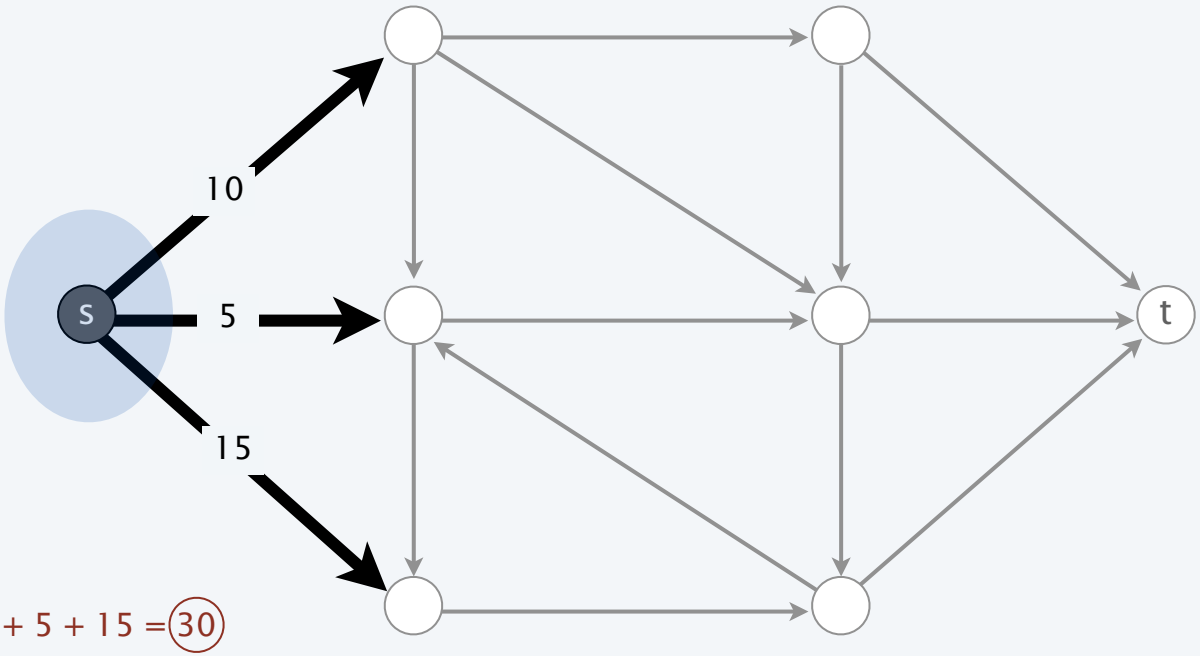
value = 8 + 10 + 10 = 28

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.

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



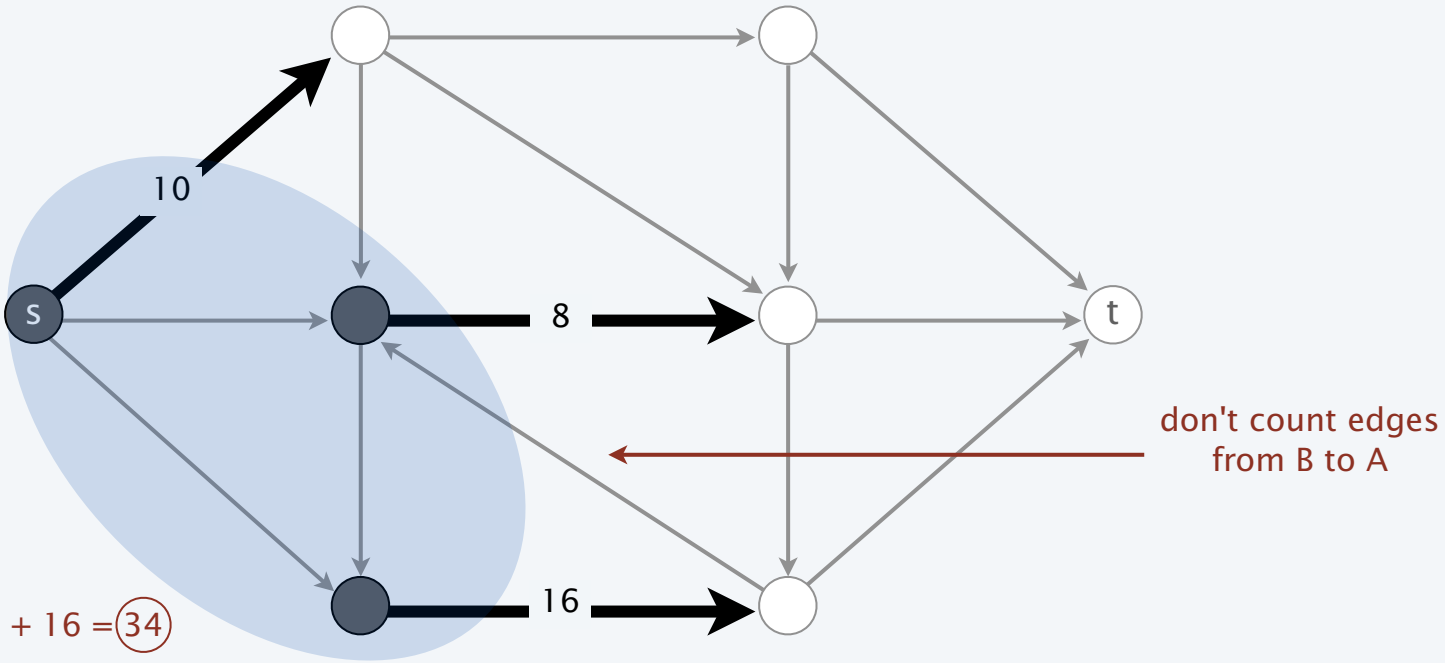
capacity = 10 + 5 + 15 = 30

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

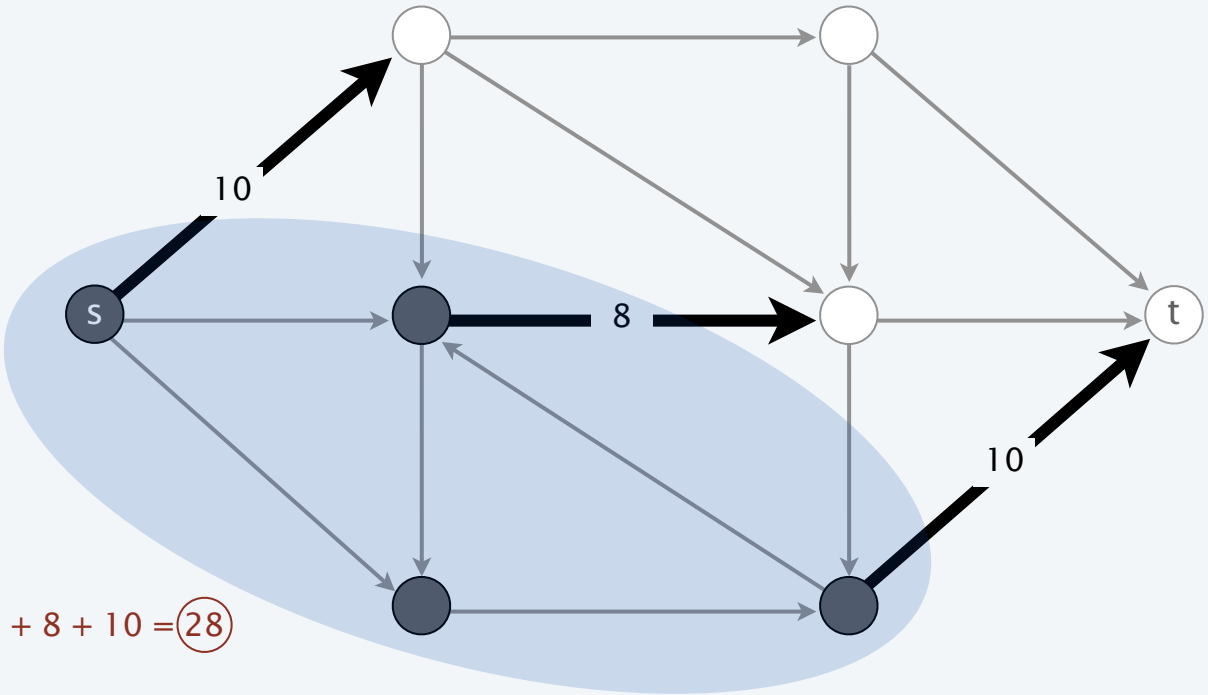
Minimum cut problem

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

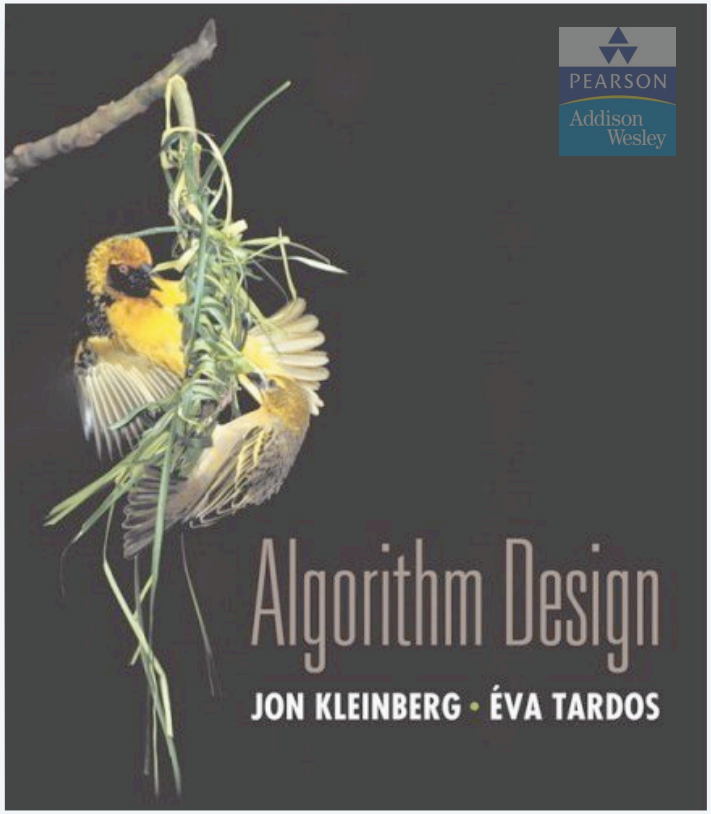
Def. Its capacity is the sum of the capacities of the edges from A to B.

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

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



capacity = 10 + 8 + 10 = 28



SECTION 7.1

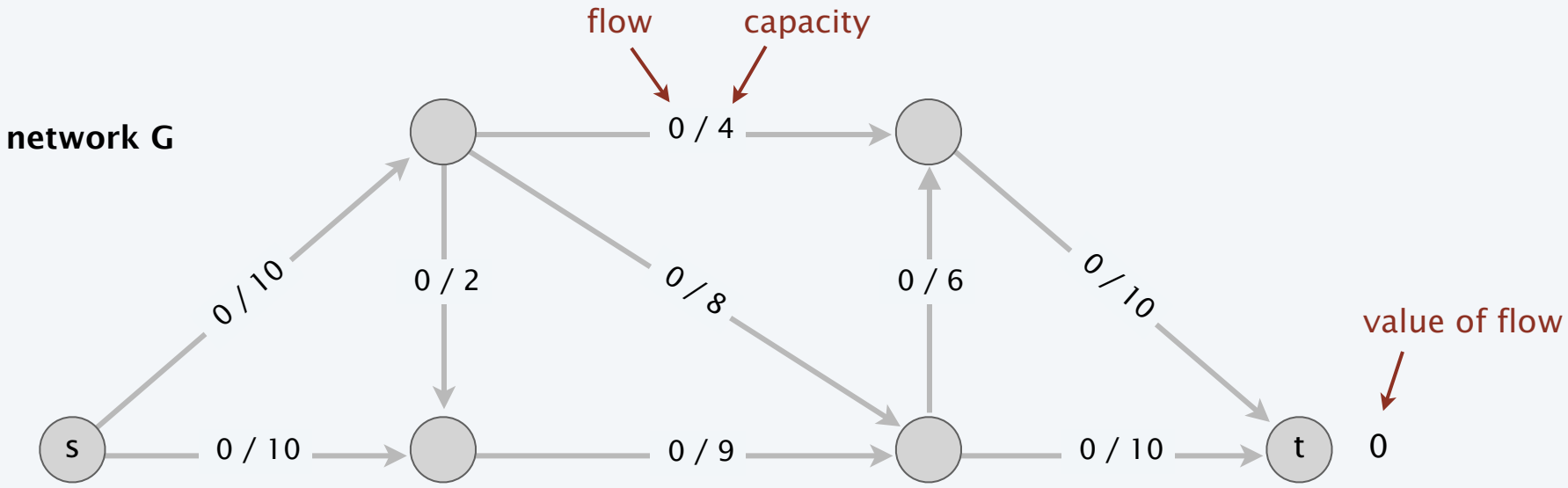
7. NETWORK FLOW I

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

Towards a max-flow algorithm

Greedy algorithm.

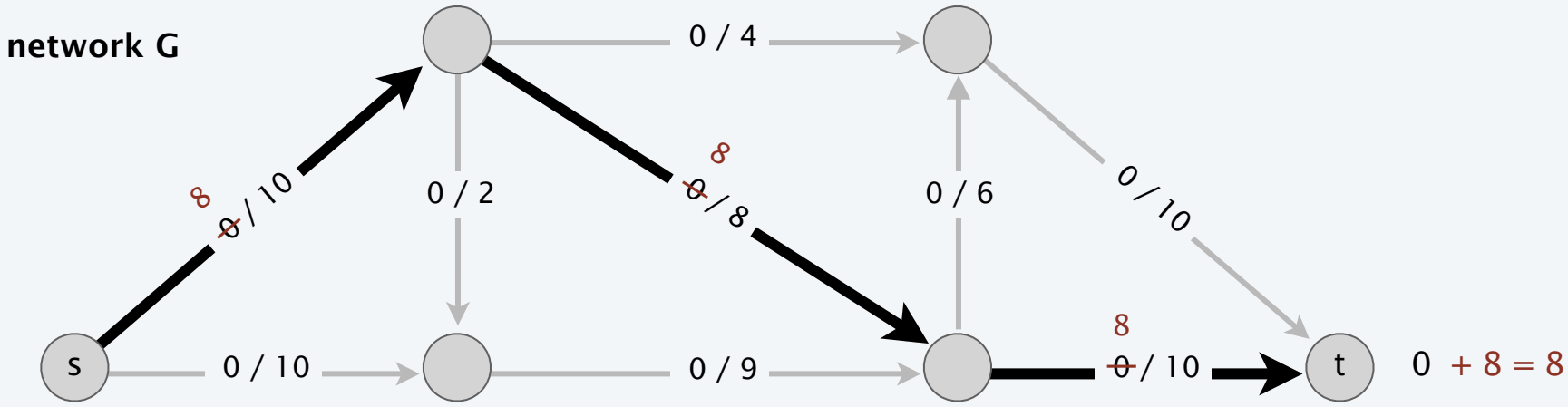
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Towards a max-flow algorithm

Greedy algorithm.

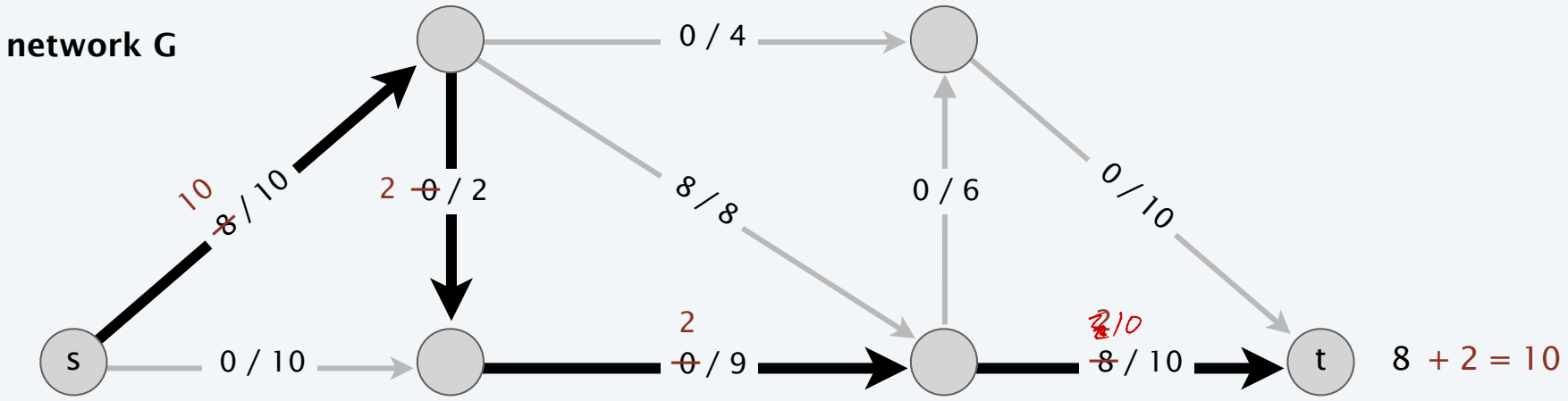
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Towards a max-flow algorithm

Greedy algorithm.

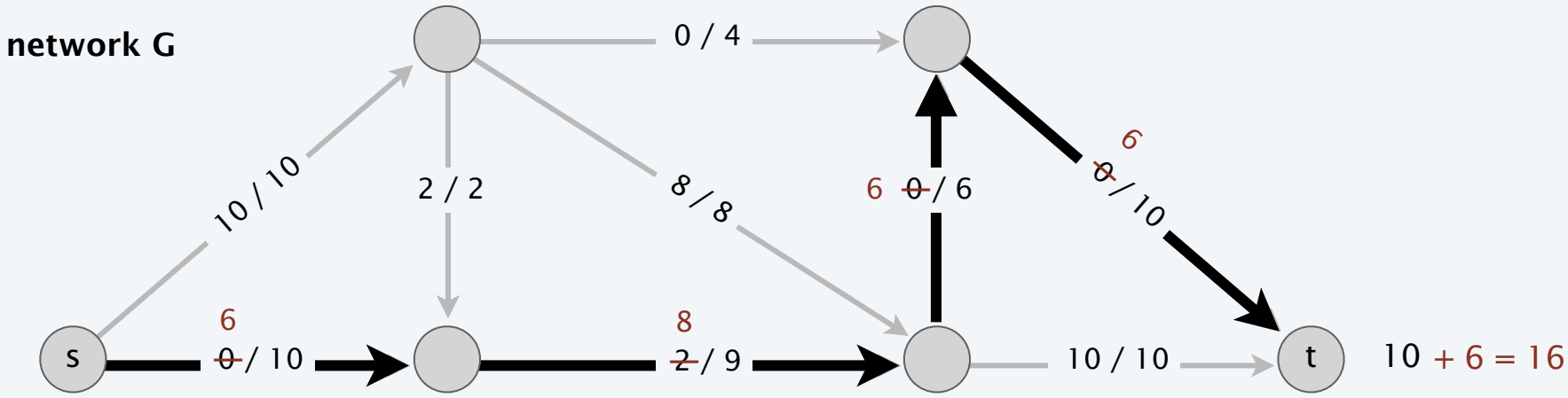
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.



Towards a max-flow algorithm

Greedy algorithm.

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

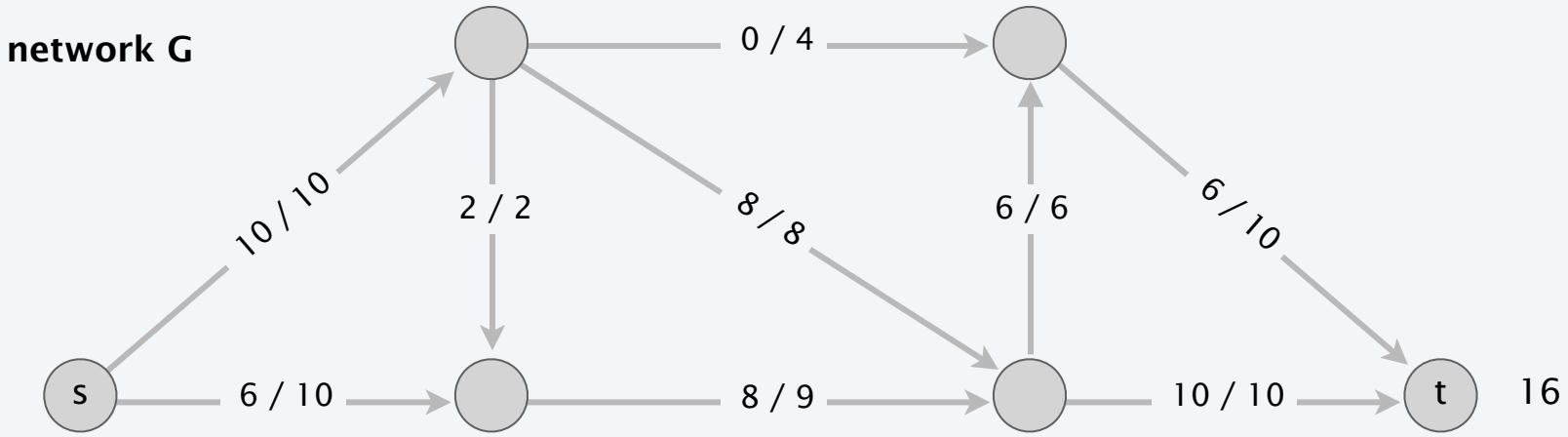


Towards a max-flow algorithm

Greedy algorithm.

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

ending flow value = 16

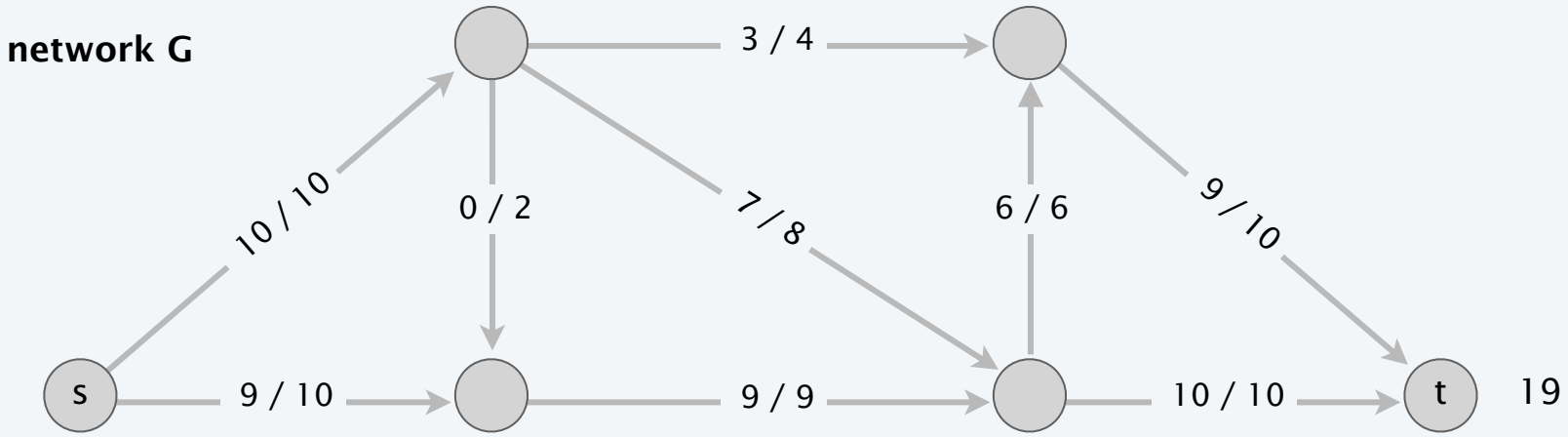


Towards a max-flow algorithm

Greedy algorithm.

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

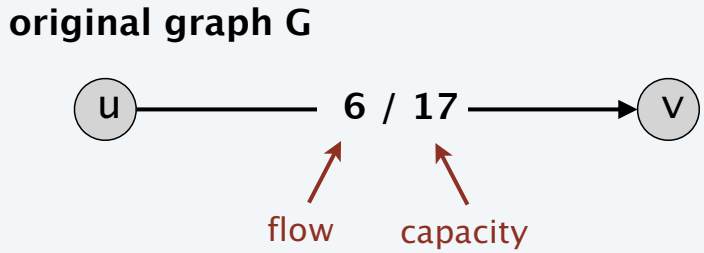
but max-flow value = 19



Residual graph

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

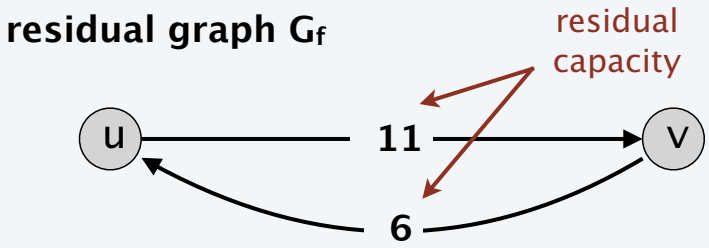
- Flow $f(e)$.
- Capacity $c(e)$.



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

Augmenting path

Def. An augmenting path is a simple $s \to t$ 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)

$b \leftarrow$ bottleneck capacity of path P.

FOREACH edge $e \in P$

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

  ELSE $f(e^R) \leftarrow f(e^R) - b$.

RETURN f.

amount of flow identifying the sent along each edge in f'

e is a forward edge

e^R is a backward edge

e' in the notation from the blackboard

Ford-Fulkerson algorithm

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)

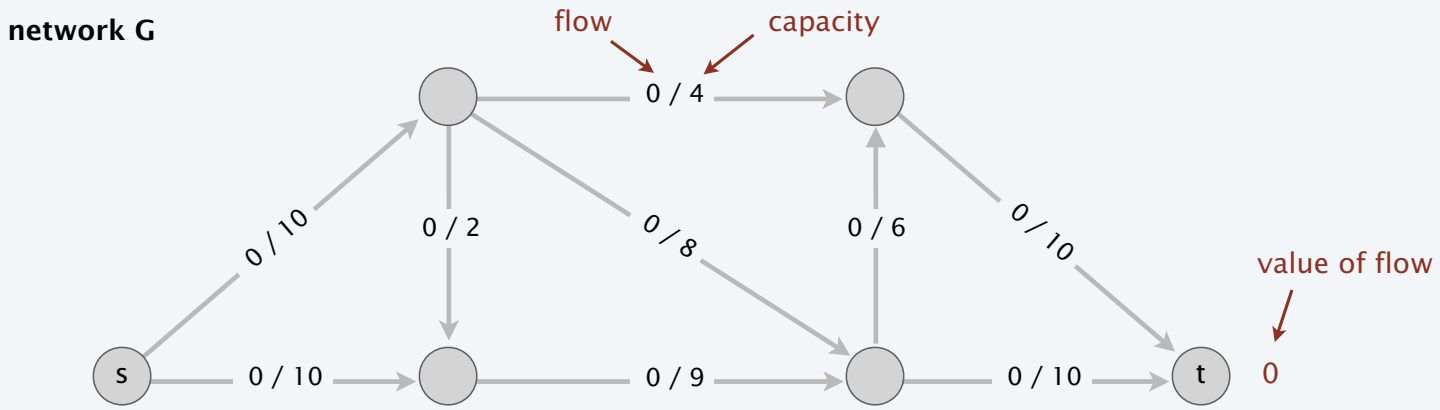
    $f \leftarrow$ AUGMENT (f, c, P).

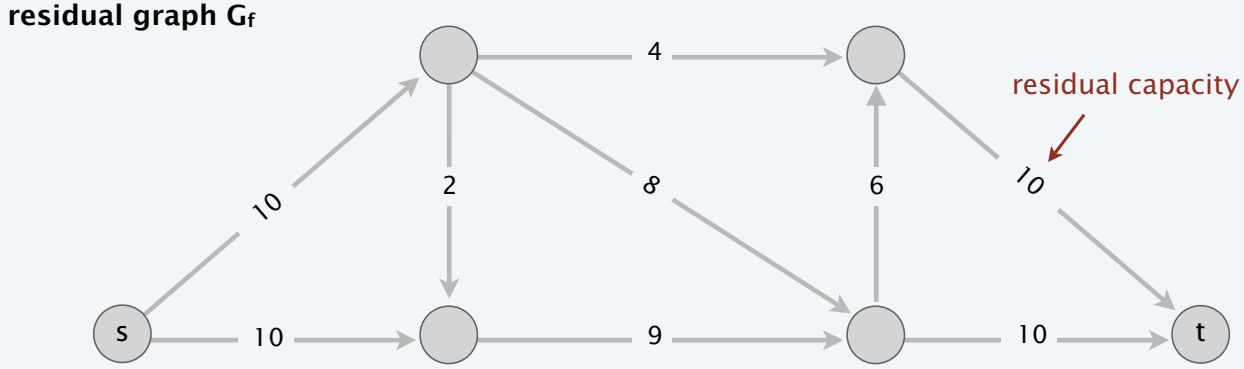
    Update G_f.

  RETURN f.

}

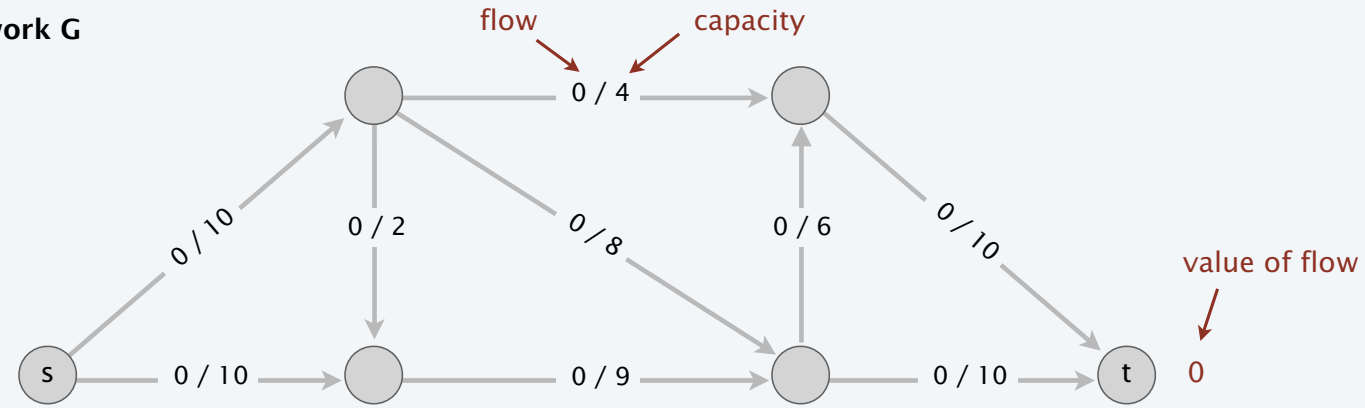
Ford-Fulkerson algorithm demo



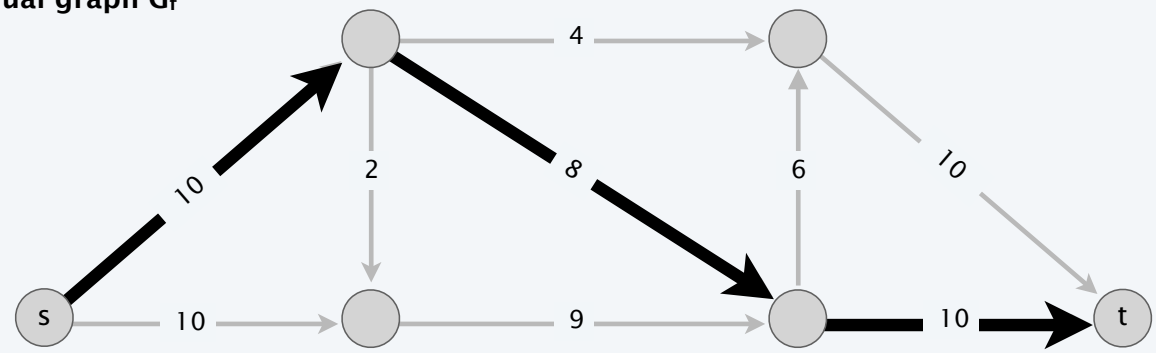


Ford-Fulkerson algorithm demo

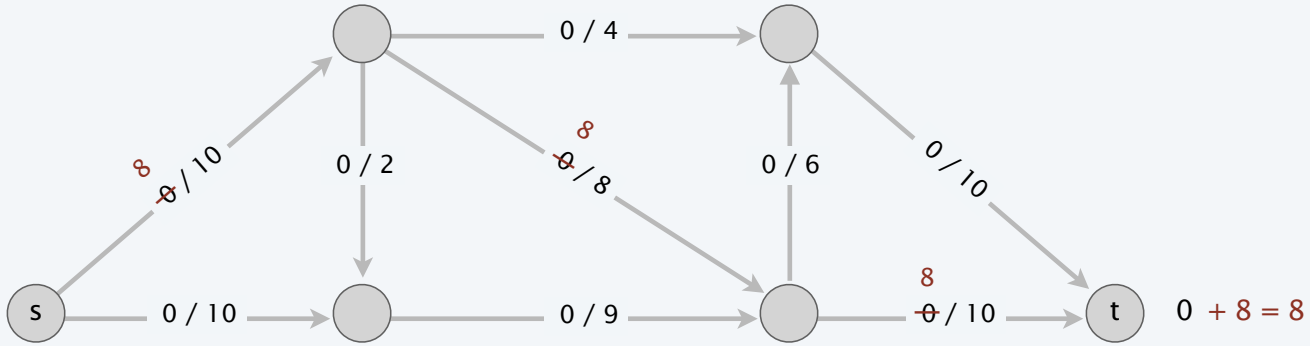
network G



residual graph G_f

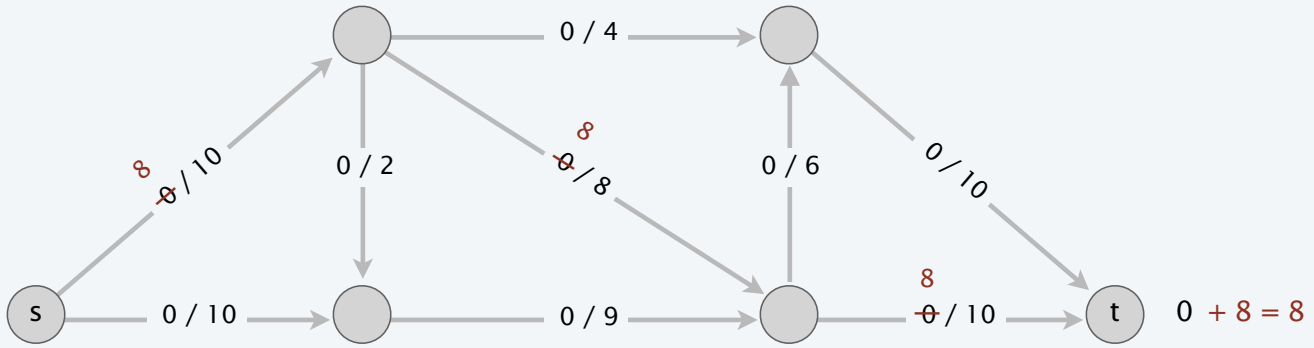


network G

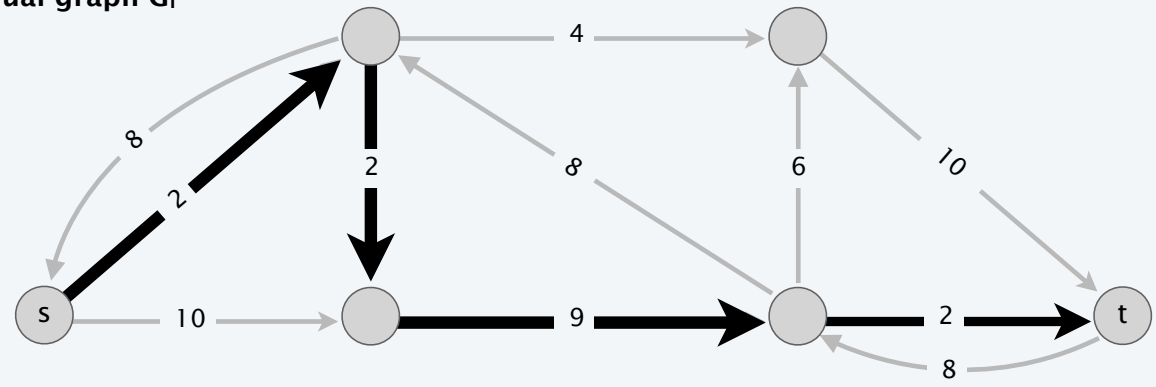


Ford-Fulkerson algorithm demo

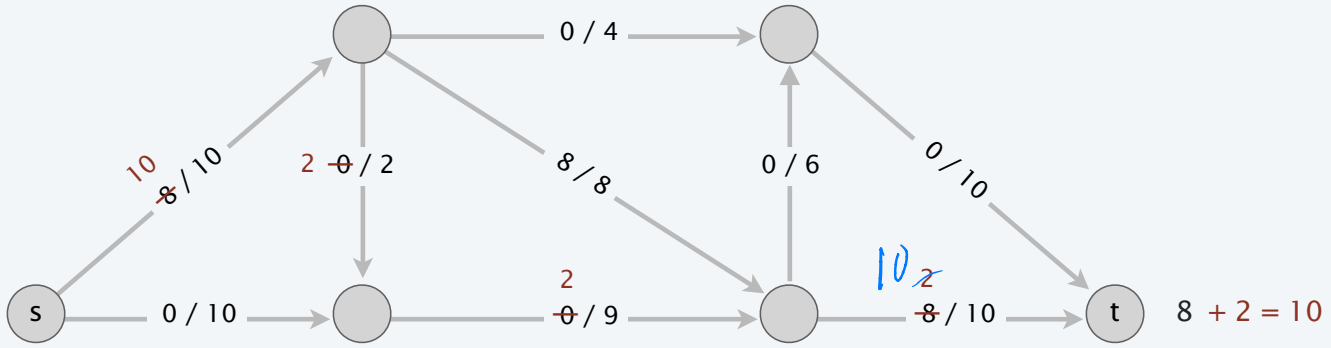
network G



residual graph G_f

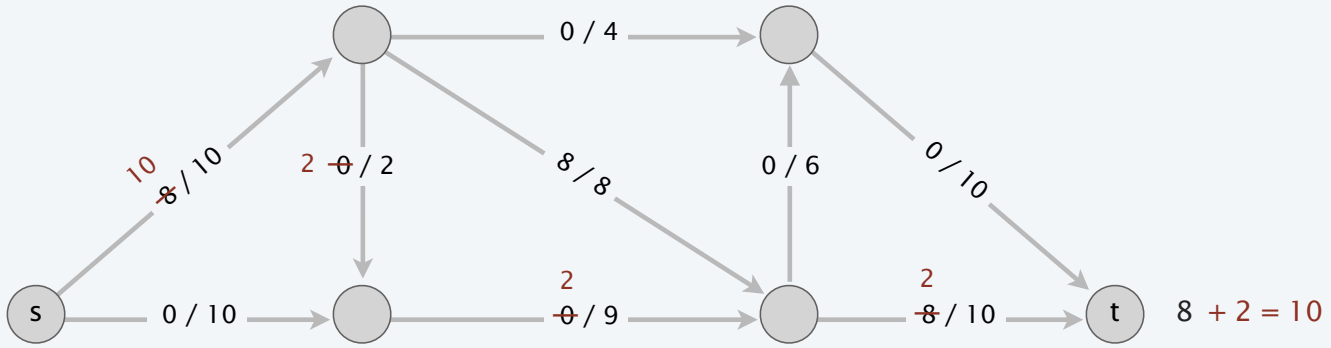


network G

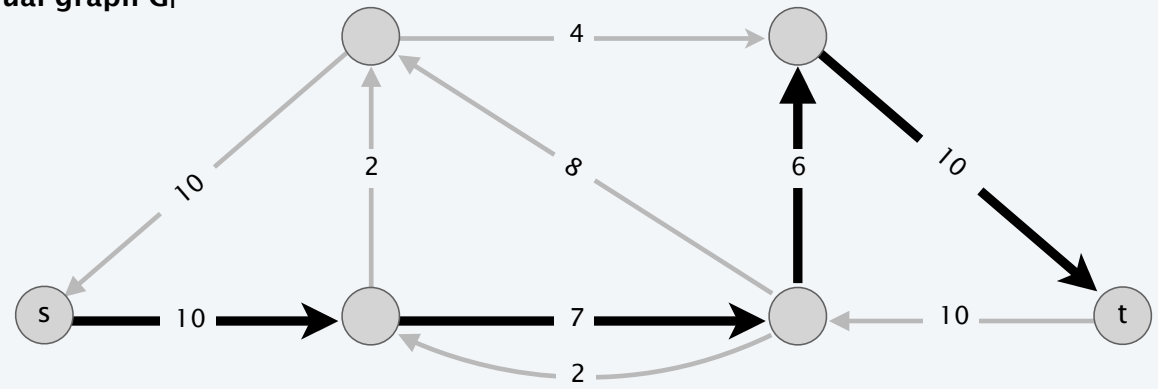


Ford-Fulkerson algorithm demo

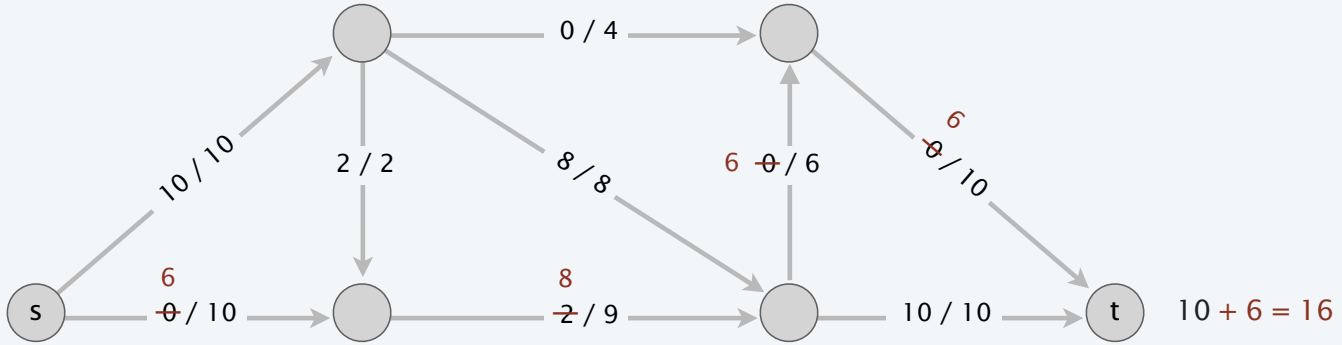
network G



residual graph G_f

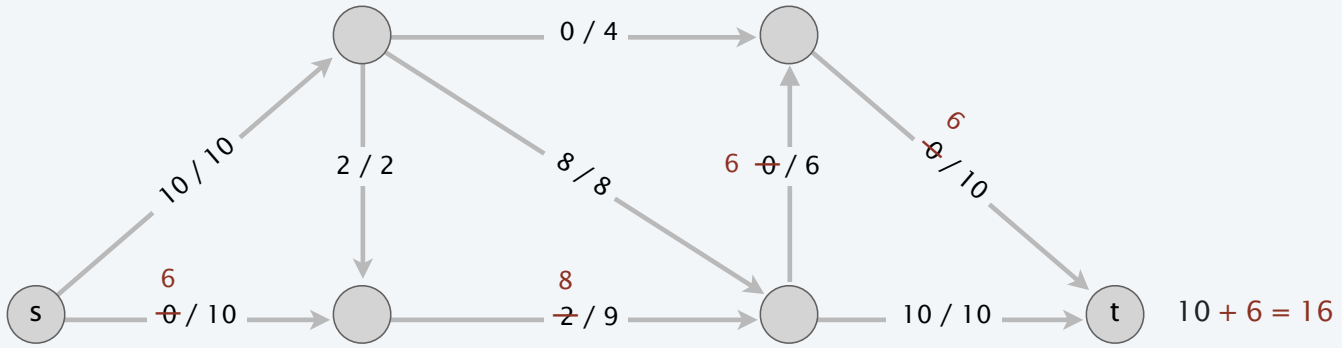


network G

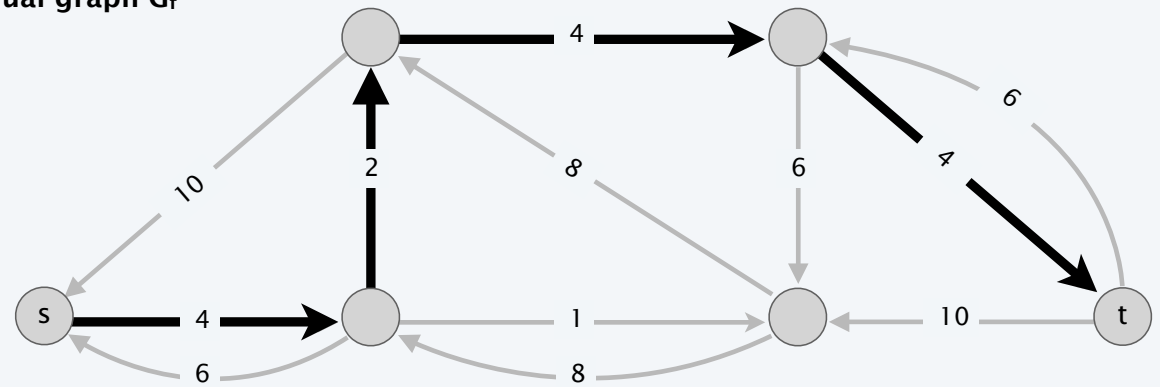


Ford-Fulkerson algorithm demo

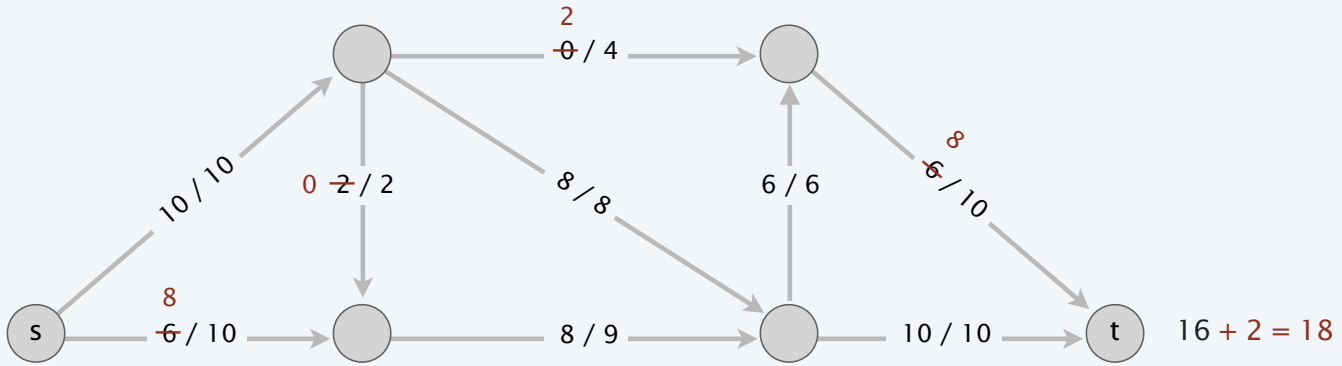
network G



residual graph G_f

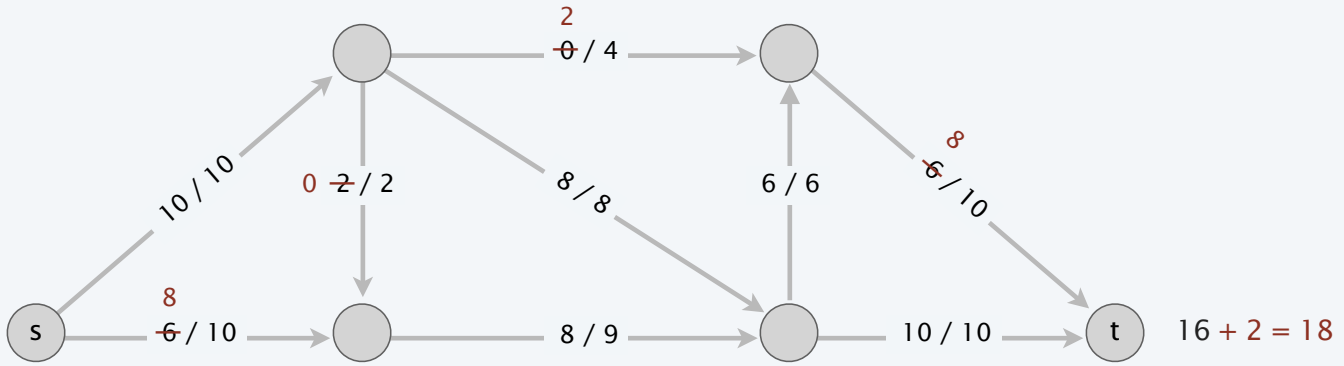


network G

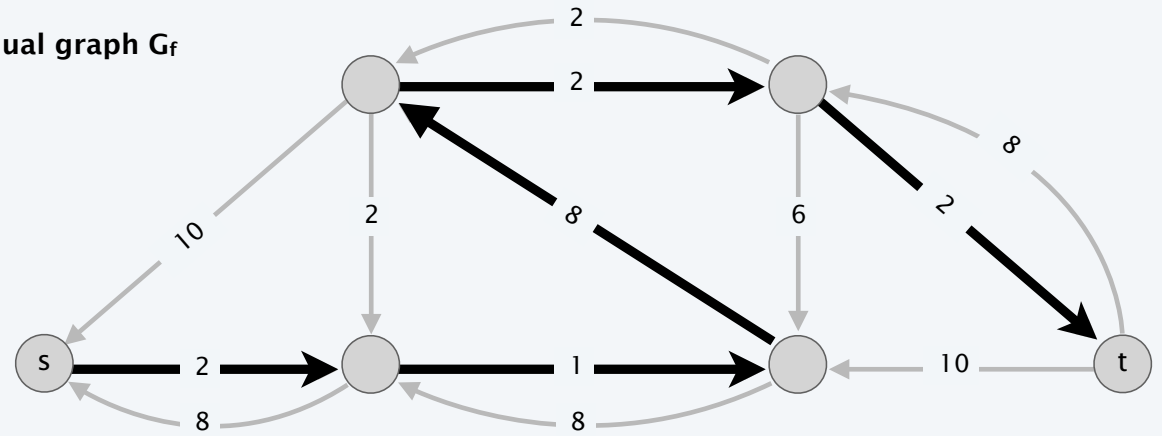


Ford-Fulkerson algorithm demo

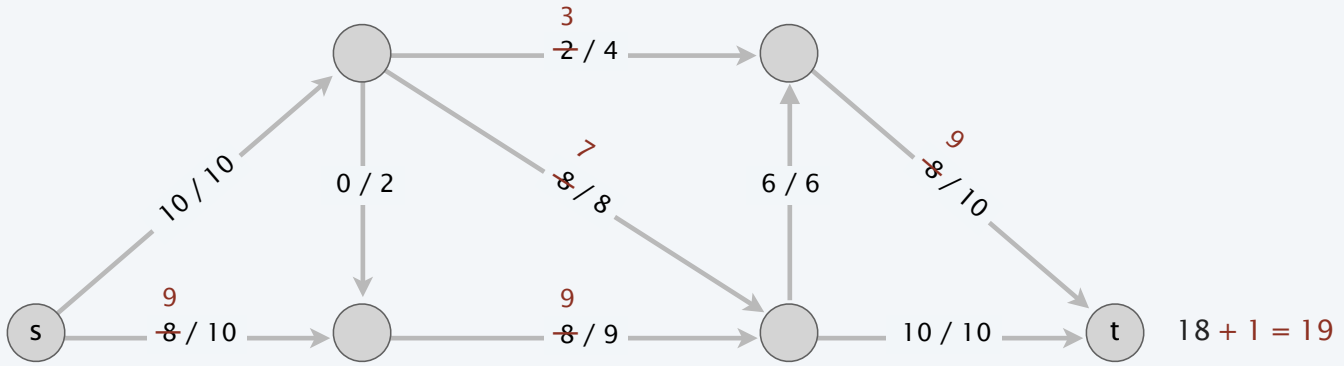
network G



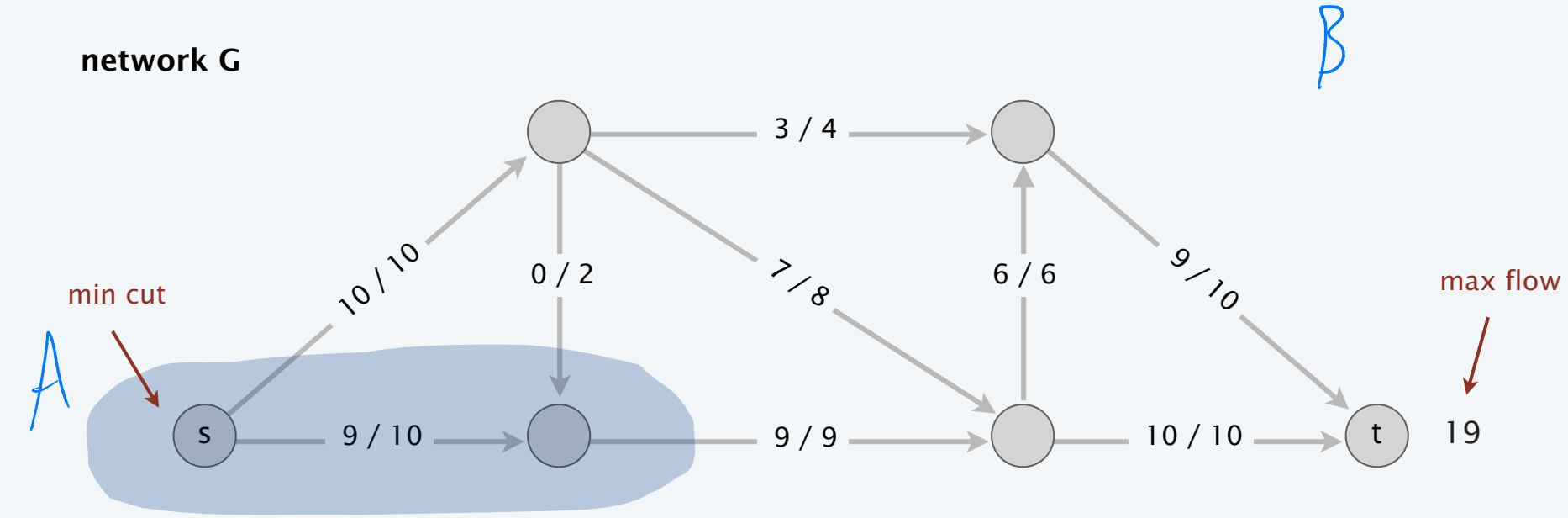
residual graph G_f

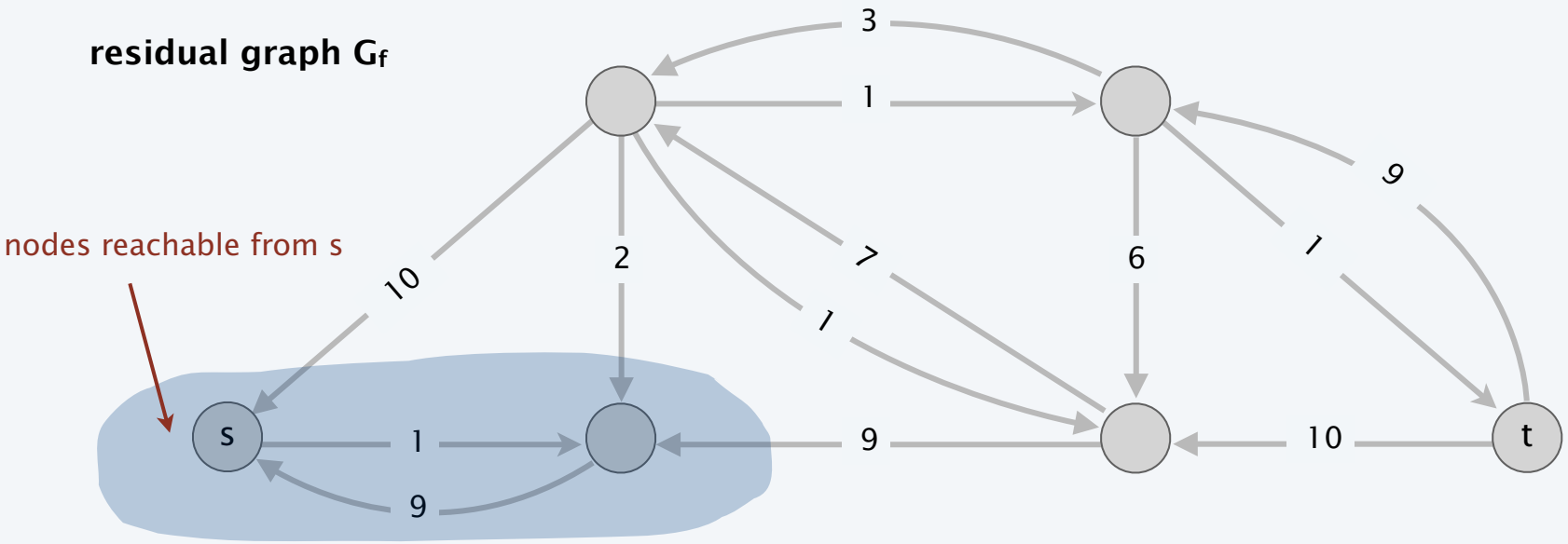


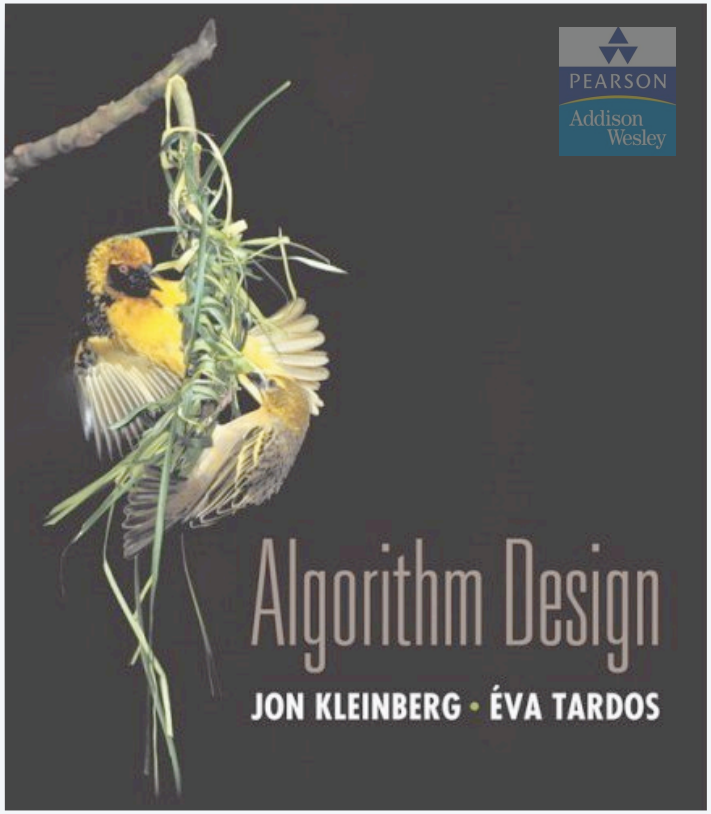
network G



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*
- *capacity-scaling algorithm*
- *shortest augmenting paths*
- *blocking-flow algorithm*
- *unit-capacity simple networks*

Flow Value Lemma

Let f be a flow and let (A, B) be an s-t cut. Then:

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

Flow Value Lemma

Given an s-t cut (A, B),

$$v(f) = f^{(out)}(A) - f^{(in)}(A)$$

Let $U \subseteq V$.

Define $f^{(out)}(U) = \sum_{e \text{ out of } U} f(e)$

$$f^{(in)}(U) = \sum_{e \text{ into } U} f(e)$$

For $v \in V$: $f^{(out)}(v) = \sum_{e \text{ out of } v} f(e)$

$$f^{(in)}(v) = \sum_{e \text{ into } v} f(e)$$

Proof:

By definition: $v(f) = f^{(out)}(s)$

$$= f^{(out)}(s) - \underbrace{f^{(in)}(s)}_{=0}$$

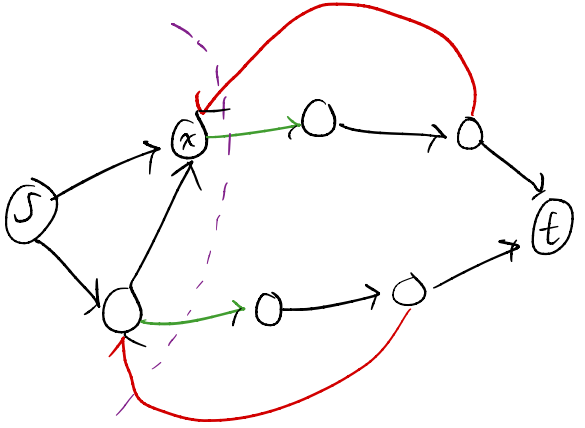
$$= f^{(out)}(s) - f^{(in)}(s) + \underbrace{\sum_{v \in A \setminus \{s\}} \left(f^{(out)}(v) - f^{(in)}(v) \right)}_{=0}$$

conservation of flow

$$f^{(out)}(s) - f^{(in)}(s) + \underbrace{\sum_{v \in A \setminus \{s\}} \left(f^{(out)}(v) - f^{(in)}(v) \right)}_{=0} = \sum_{v \in A} \left[f^{(out)}(v) - f^{(in)}(v) \right]$$

$$= f^{(out)}(A) - f^{(in)}(A)$$

A



B

$+$?? $= 0$

For (u,v) with $u,v \in A$

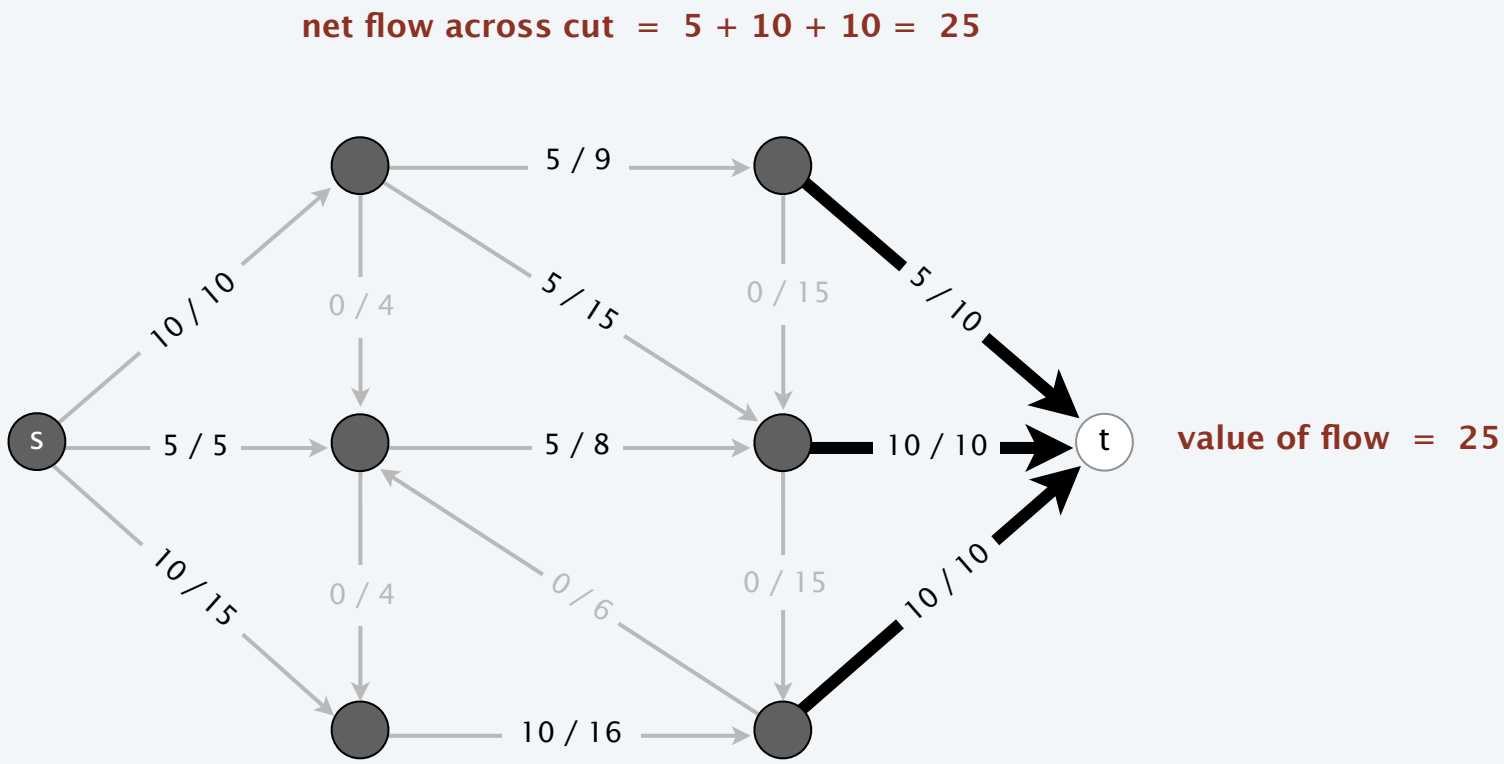
(u,v) is counted with $+$ for $f^{(out)}(u)$

and counted with $-$ for $f^{(in)}(v)$

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.

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

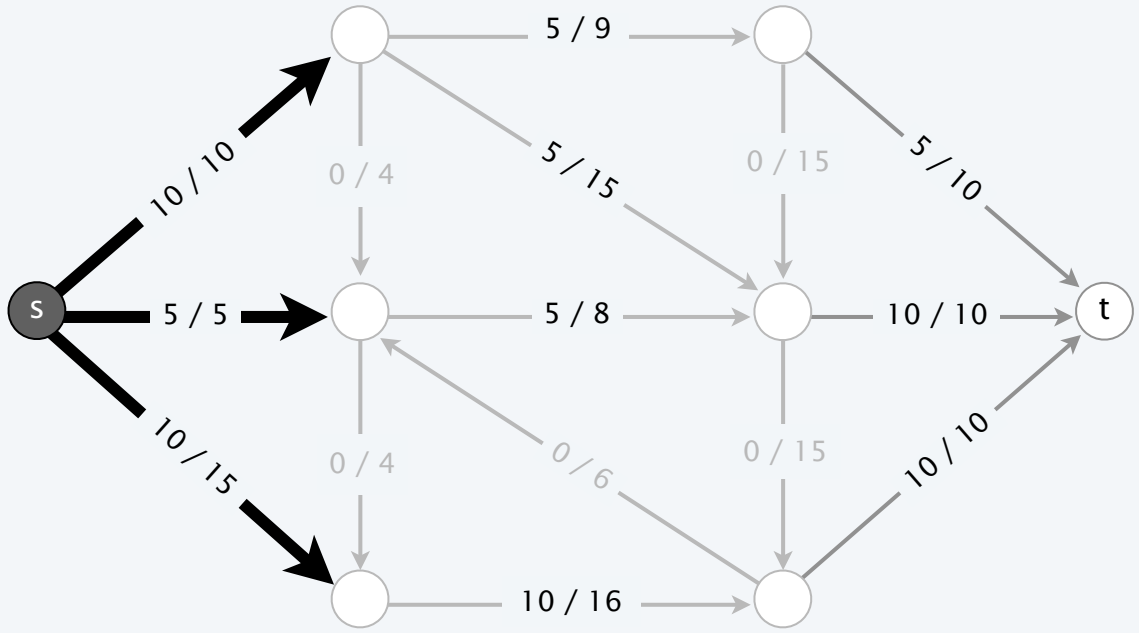


Relationship between flows and cuts

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 + 5 + 10 = 25



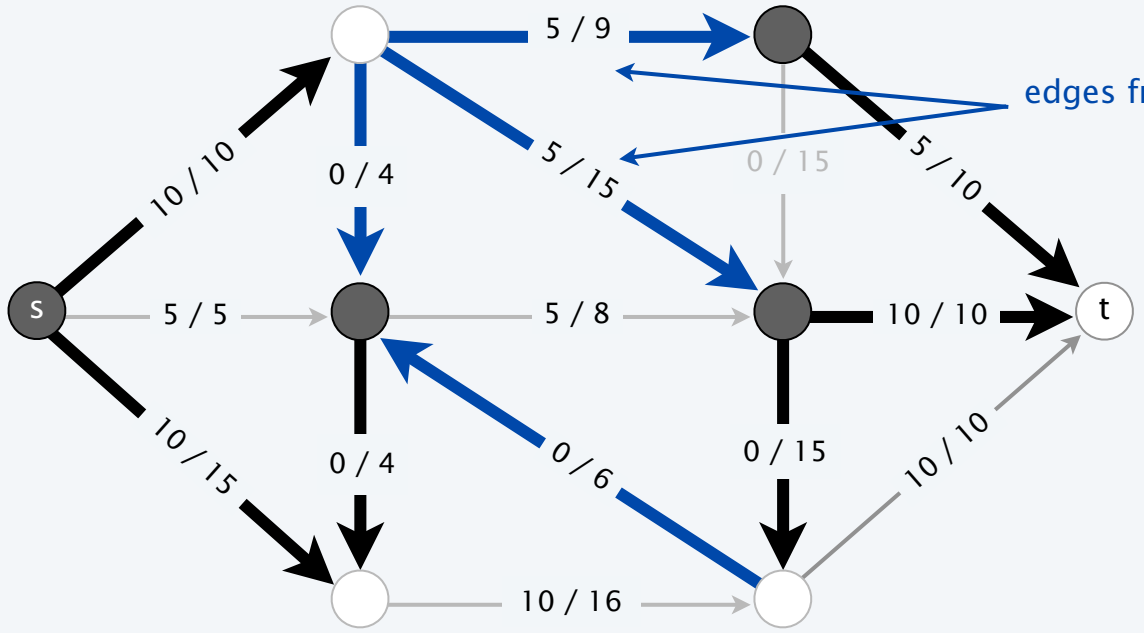
value of flow = 25

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.

$$v(f) \; = \sum_{e \text{ out of } s} f(e)$$

by flow conservation, all terms except 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, $v(f) \le cap(A, B)$.

Pf.

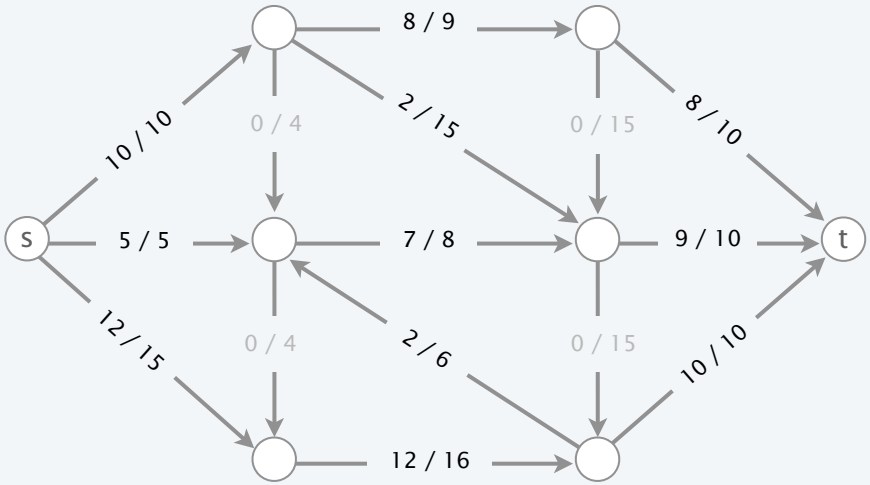
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e)$$

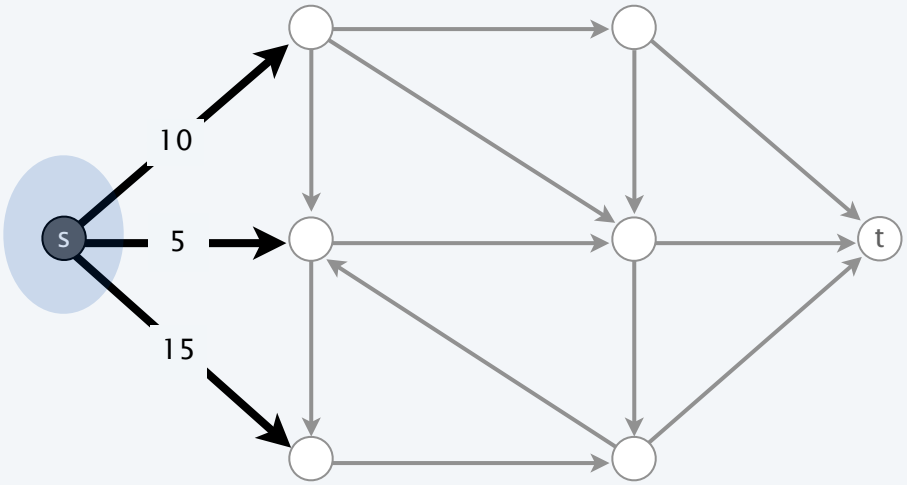
flow-value lemma

$$\le \sum_{e \text{ out of } A} f(e)$$

$$\le \sum_{e \text{ out of } A} c(e)$$

$$= \text{cap}(A, B) \quad \blacksquare$$





value of flow = 27 ≤ **capacity of cut = 30**

Max-flow min-cut theorem

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 $\Rightarrow$ ii]

- Suppose that (A, B) is a cut such that $cap(A, B) = val(f)$.
- Then, for any flow f', $val(f') \leq cap(A, B) = val(f)$.
- Thus, f is a max-flow. ▪

weak duality

by assumption

Max-flow min-cut theorem

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]

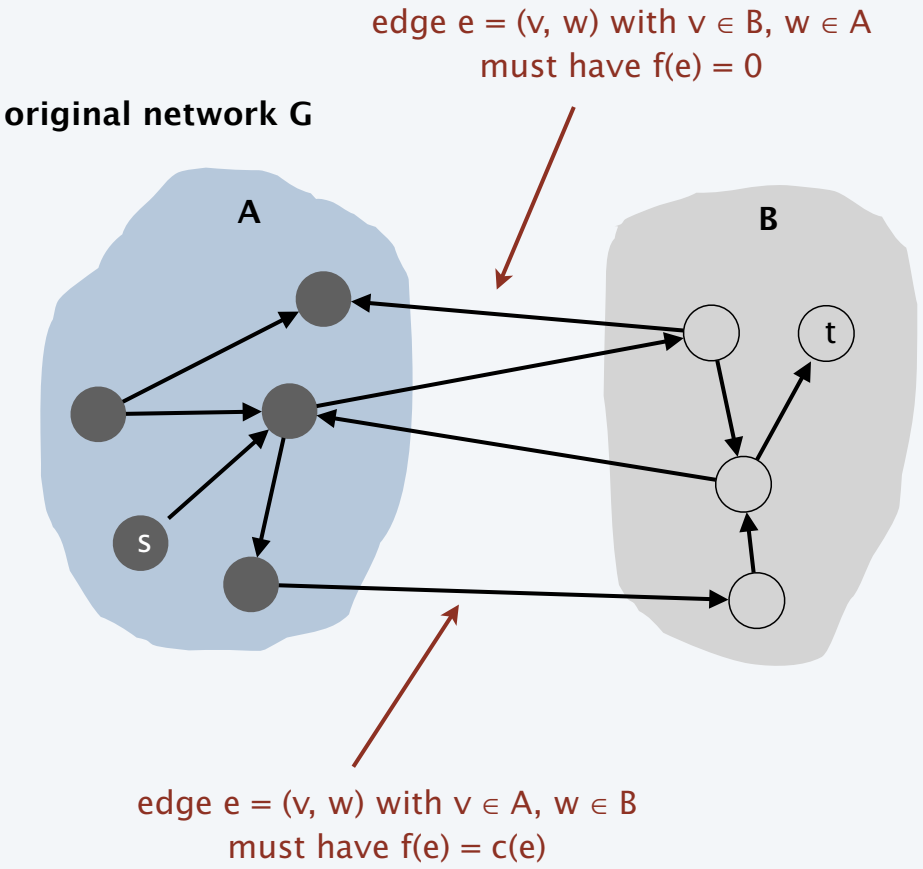
- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual graph G_f.
- By definition of cut A, $s \in A$.
- By definition of flow f, $t \notin A$.

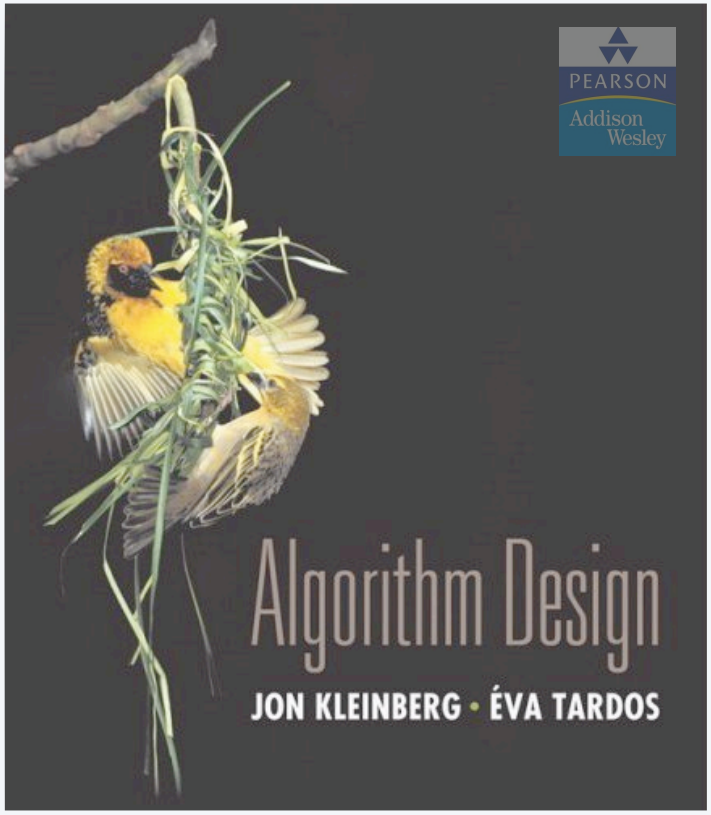
$$v(f) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e)$$

flow-value lemma

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

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





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*
- *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

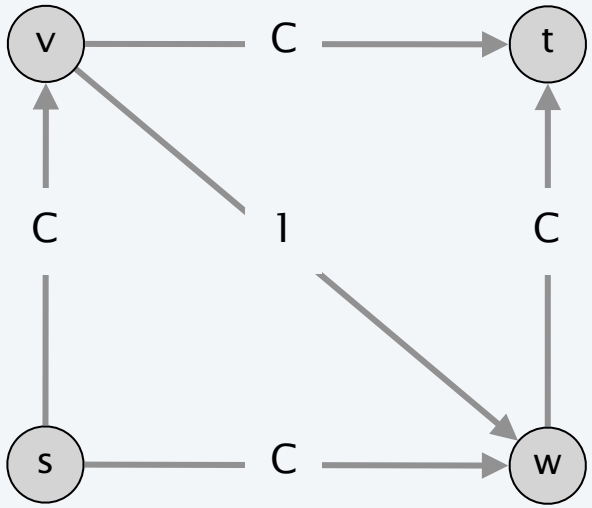
Q. Is generic Ford-Fulkerson algorithm poly-time in input size?

m, n, and log C

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

- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- …
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$

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!

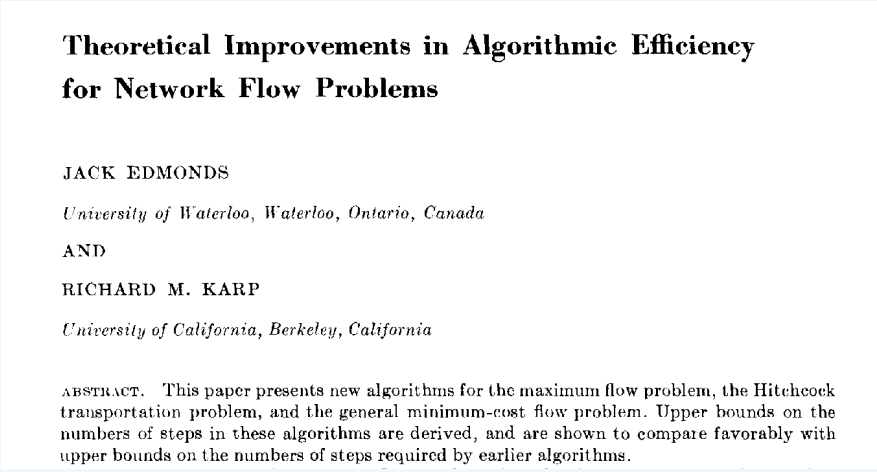
Goal. Choose augmenting paths so that:

- 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

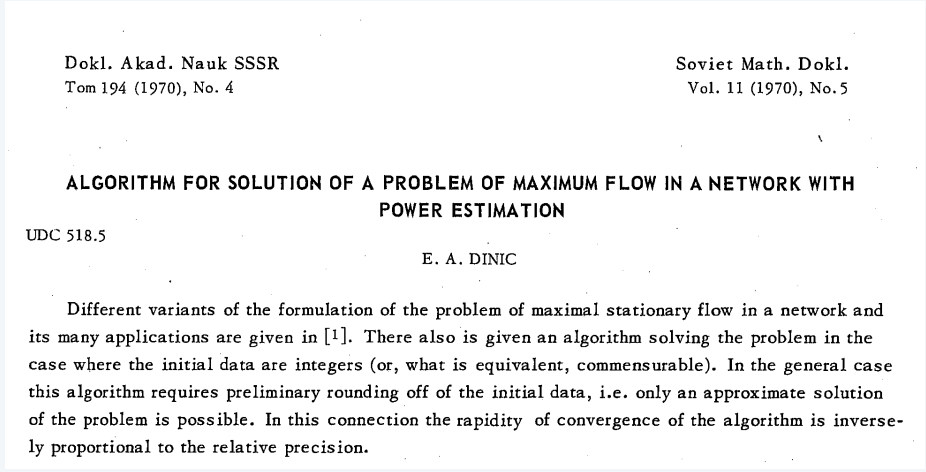
AND

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.

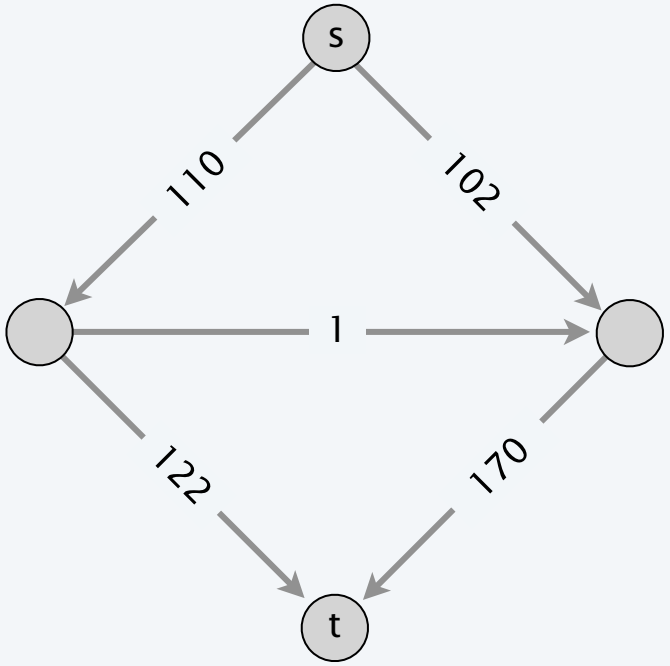
Dinic 1970 (Soviet Union)

Capacity-scaling algorithm

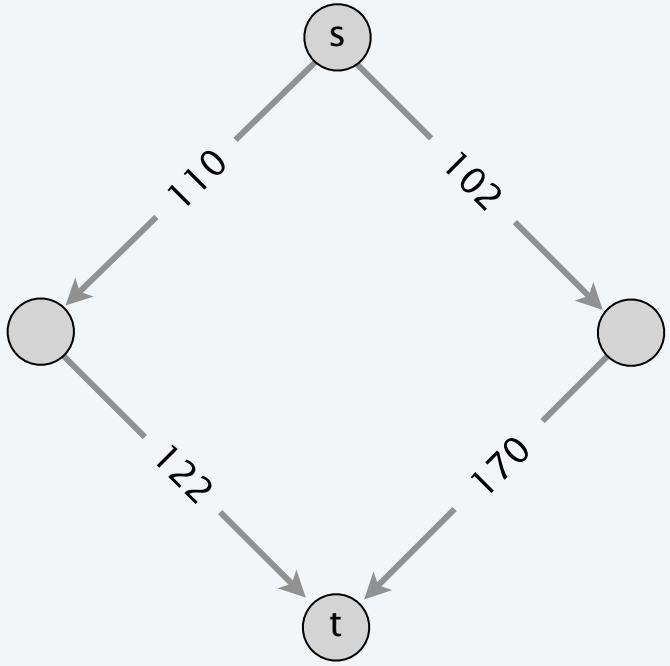
Note: we did not cover capacity-scaling algorithm in class

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 algorithm

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 $C / 2 < \Delta \leq C$; 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 $\leq val(f) + m \Delta$. ⟵ 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 $\Rightarrow$ $val(f^*) \leq val(f) + 2 m \Delta$.
- 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. It can be implemented to run in $O(m^2 \log C)$ time.

Pf. Follows from LEMMA 1 and LEMMA 3. ▪

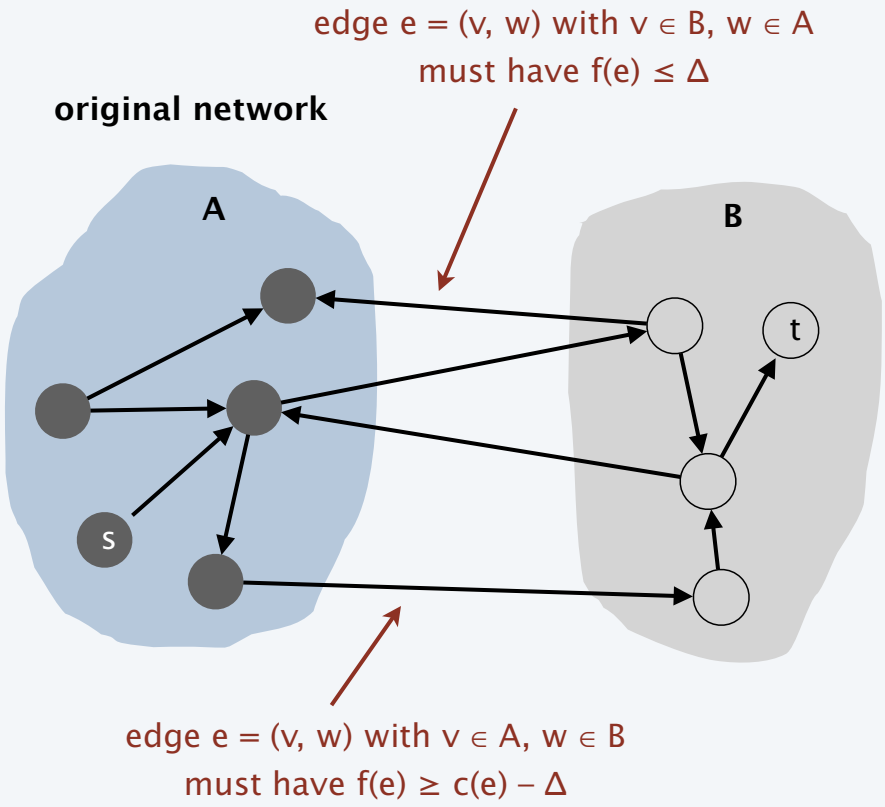
Capacity-scaling algorithm: analysis of running time

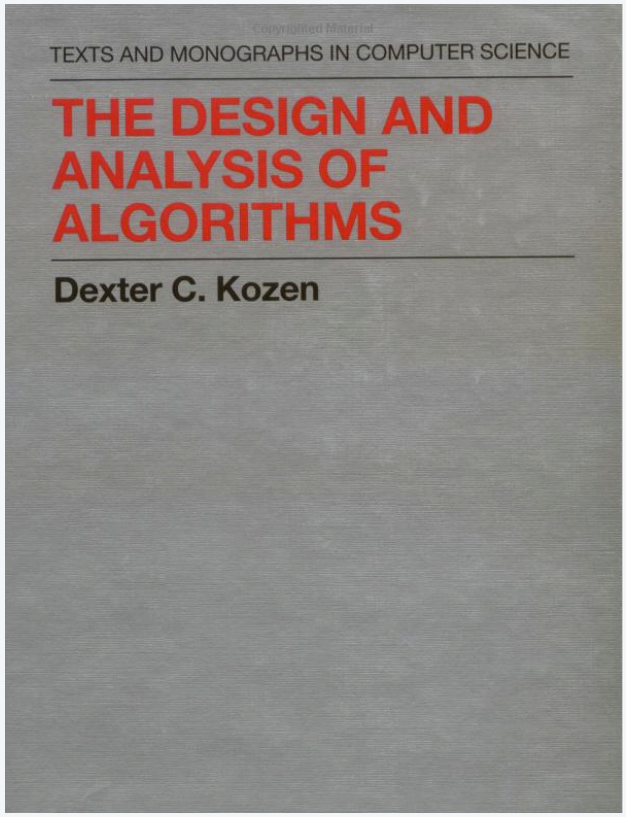
Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then, the value of the max-flow $\leq val(f) + m\,\Delta$.

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, $t \notin A$.

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





SECTION 17.2

7. NETWORK FLOW I

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

Shortest augmenting path

Q. Which augmenting path?

A. The one with the fewest number of edges.

SHORTEST-AUGMENTING-PATH(G, s, t, c)

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

  $G_f \leftarrow$ residual graph.

  WHILE (there exists an augmenting path in G_f)

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

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

    Update G_f.

RETURN f.

For flow f and vertex v, let $\delta_f(s, v)$ be length of shortest s-v path in G_f ("shortest" means least number of edges)

Lemma

If Edmonds-Karp is run on a flow network, then throughout the algorithm, for all vertices $v \in V \setminus \{s, t\}$, the shortest path distance $\delta_f(s, v)$ never decreases.

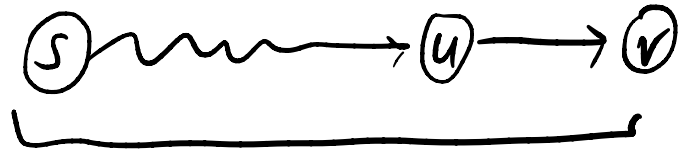
Proof

Let f be the flow just prior to first augmentation that decreases some (shortest path) distance, and let f' be the next flow

Among all vertices whose distance decreases from G_f to $G_{f'}$,
let v be the vertex with minimum $\delta_{f'}(s, v)$

Let P be shortest s-v path in $G_{f'}$, and let u be predecessor of v in P

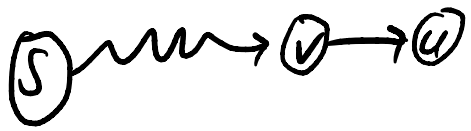
After augmentation:



P is path in $G_{f'}$

(1) $\delta_{f'}(s,v) = \delta_{f'}(s,u) + 1$

Because $\delta_{f'}(s,u) < \delta_{f'}(s,v) \Rightarrow \delta_{f'}(s,u) \geq \delta_f(s,u)$ (2)

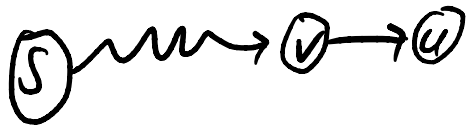
Claim: In G_f, shortest s-u path is of the form 

Claim $\Rightarrow$ $\delta_f(s,u) = \delta_f(s,v) + 1$ (3)

Proof continued

(1) $\delta_{f'}(s,v) = \delta_{f'}(s,u) + 1$

Because $\delta_{f'}(s,u) < \delta_{f'}(s,v) \Longrightarrow \delta_{f'}(s,u) \geq \delta_{f}(s,u)$ (2)

Claim: In G_f, shortest s-u path is of the form 

Claim $\Longrightarrow$ $\delta_f(s,u) = \delta_f(s,v) + 1$ (3)

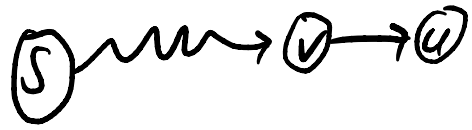
$$\delta_{f'}(s,v) = \delta_{f'}(s,u) + 1 \quad (1)$$

$$\geq \delta_f(s,u) + 1 \quad (2)$$

$$= \delta_f(s,v) + 2 \quad (3)$$

$\delta_{f'}(s,v)$ ↔ shortest path dist. from s to v actually increased by 2.

Proof continued

Claim: In G_f, shortest s-u path is of the form 

Proof:

First, we claim (u,v) is not an edge in G_f.

Suppose (for contradiction) that (u,v) is edge in G_f.

$$\begin{aligned} \delta_f(s,v) &\leq \delta_f(s,u) + 1 && \text{(triangle inequality)} \\ &\leq \delta_{f'}(s,u) + 1 && (2) \\ &= \delta_{f'}(s,v) && (1) \quad \lightning \end{aligned}$$

Indeed, (u,v) is not in G_f. But! (u,v) is in $G_{f'}$.

$\Rightarrow$ (v,u) belongs to the path along which flow was augmented in G_f.

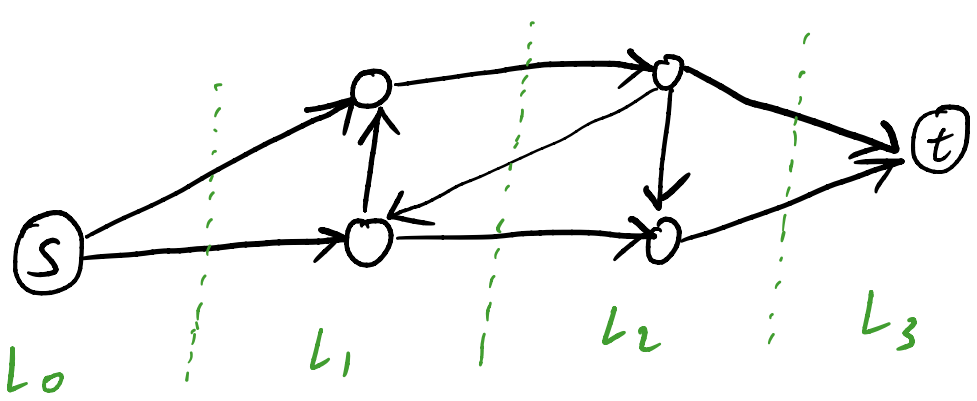
$\Rightarrow$ (v,u) is edge in s.p. from s to u.

Theorem

m = #edges n = #vertices

If Edmonds-Karp is run on a flow network, then the algorithm performs $O(mn)$ flow augmentations.

Proof



G_f

Let p be augmenting path in G_f s.t. $p = (v_0, v_1, \ldots, v_j)$

B/c p is a shortest path, $\forall i : v_i \in L_i$

At least one edge (v_i, v_{i+1}) in p will be "bottleneck edge" — augmentation of flow uses all of this edge's residual capacity.

After augment: (v_i, v_{i+1}) is removed! and we add backward edge (v_{i+1}, v_i)

Let P be augmenting path in G_f s.t. $P = (v_0, v_1, \ldots, v_j)$

B/c P is a shortest path, $\forall i : v_i \in L_i$

At least one edge (v_i, v_{i+1}) in P will be "bottleneck edge" — augmentation of flow uses all of this edge's residual capacity.

After augment: (v_i, v_{i+1}) is removed! and we add backward edge (v_{i+1}, v_i)

Suppose that later, in some new residual graph G_f', the edge (v_i, v_{i+1}) comes back after augment flow in G_f'.

$\Rightarrow (v_{i+1}, v_i)$ belongs to shortest s-t path in G_f'.

$$\begin{aligned} \delta_{f'}(s, v_i) &= \delta_{f'}(s, v_{i+1}) + 1 \\ &\geq \delta_f(s, v_{i+1}) + 1 \\ &= \delta_f(s, v_i) + 2 \end{aligned}$$

Each time edge (u,v) is removed and comes back,

shortest path distance from s to u $\uparrow$ by 2.

Fact: any shortest path distance $\leq n-1$

times one edge can be removed and come back $= O(n)$

edges $= m$

paths $= O(mn)$

cost of BFS in each iteration $\downarrow$ is $O(m)$

Runtime of Edmonds-Karp-Dinic : $O(m^2 n)$

Shortest augmenting path: overview of analysis

L1. Throughout the algorithm, length of the shortest path never decreases.

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

Theorem. The shortest augmenting path algorithm runs in $O(m^2 n)$ time.

Pf.

- $O(m + n)$ time to find shortest augmenting path via BFS.
- $O(m)$ augmentations for paths of length k.
- If there is an augmenting path, there is a simple one.

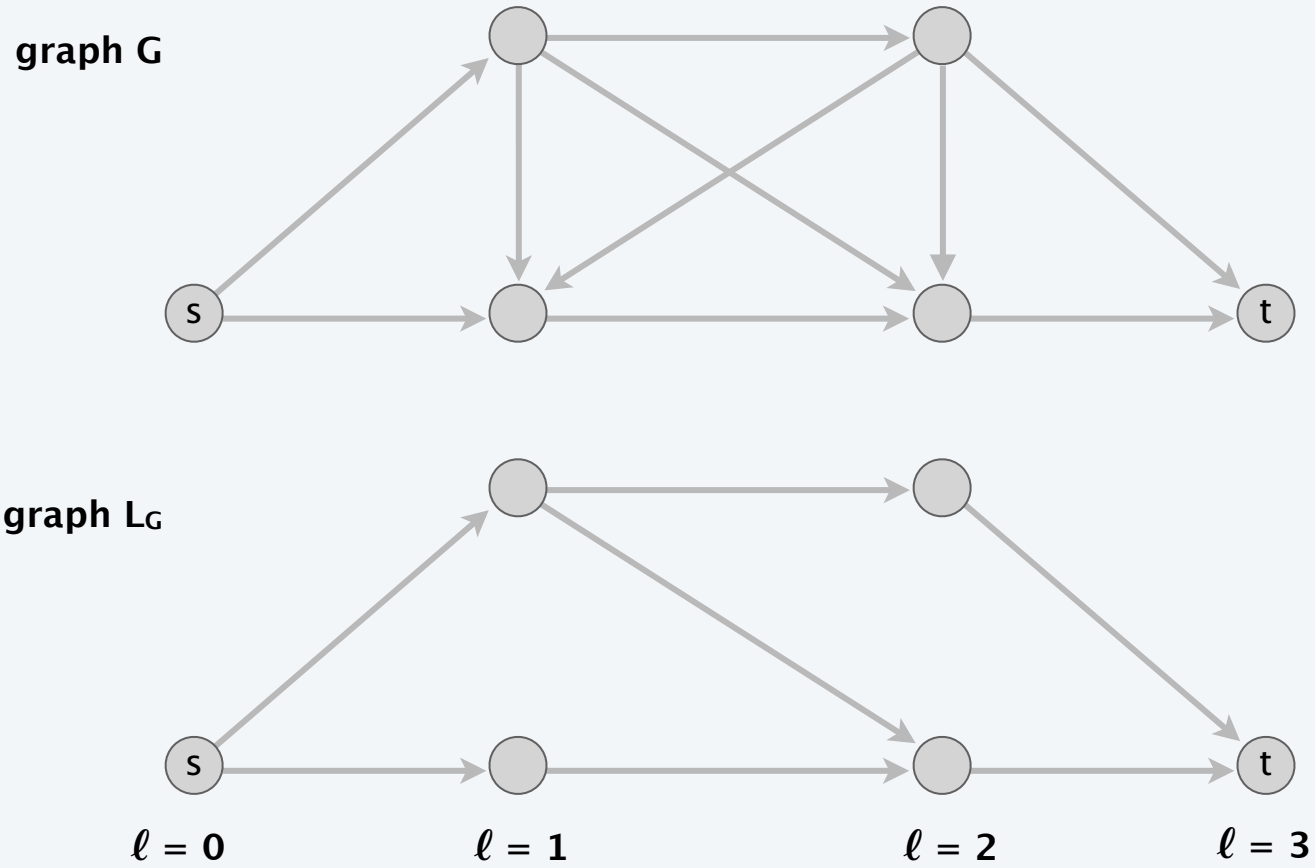
  $\Rightarrow \quad 1 \leq k < n$

  $\Rightarrow \quad O(m\,n)$ augmentations. ▪

Shortest augmenting path: analysis

Def. Given a digraph $G = (V, E)$ with source s, its level graph is defined by:

- $\ell(v) =$ number of edges in shortest path from s to v.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edges $(v,w) \in E$ with $\ell(w) = \ell(v) + 1$.



Shortest augmenting path: analysis

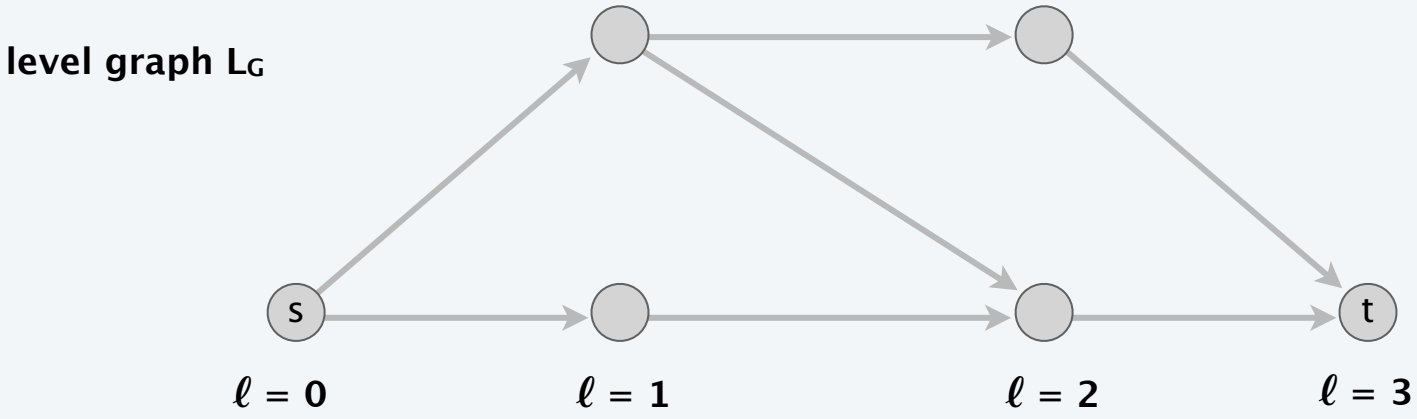
Def. Given a digraph $G = (V, E)$ with source s, its level graph is defined by:

- $\ell(v) =$ number of edges in shortest path from s to v.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edges $(v,w) \in E$ with $\ell(w) = \ell(v) + 1$.

Property. Can compute level graph in $O(m + n)$ time.

Pf. Run BFS; delete back and side edges.

Key property. P is a shortest $s \rightarrow v$ path in G iff P is an $s \rightarrow v$ path L_G.

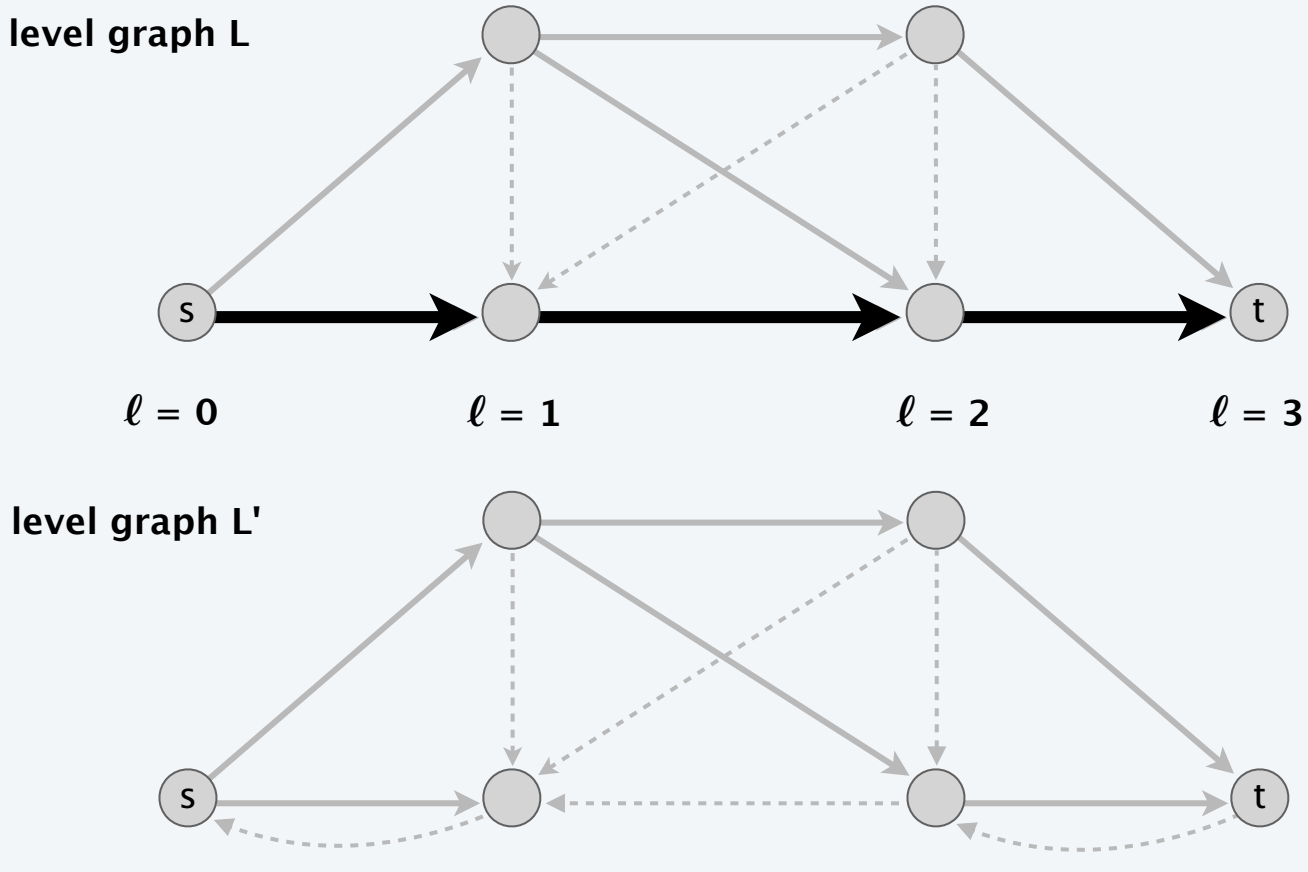


Shortest augmenting path: analysis

L1. Throughout the algorithm, length of the shortest path never decreases.

- Let f and f' be flow before and after a shortest path augmentation.
- Let L and L' be level graphs of G_f and $G_{f'}$.
- Only back edges added to $G_{f'}$

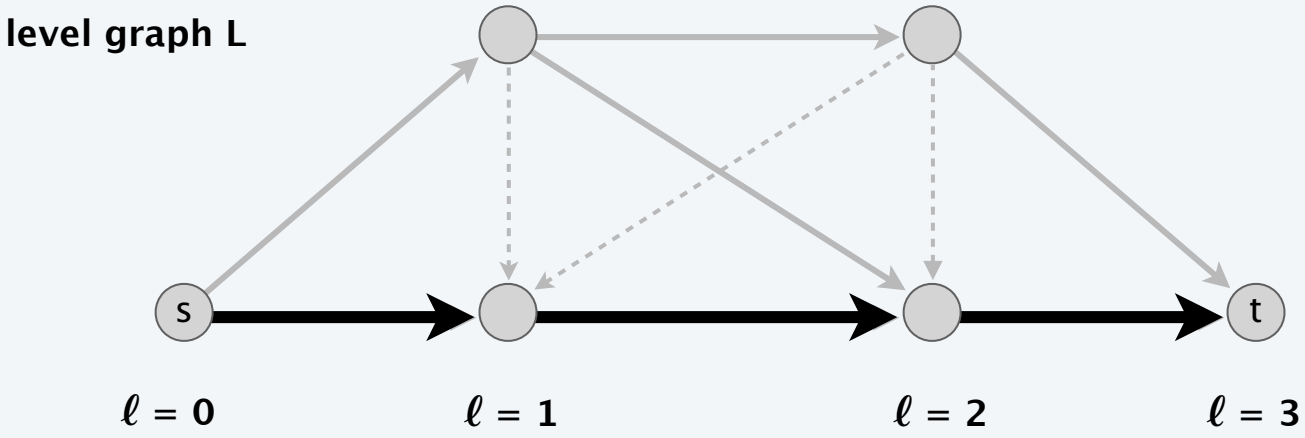
  (any path with a back edge is longer than previous length) ▪

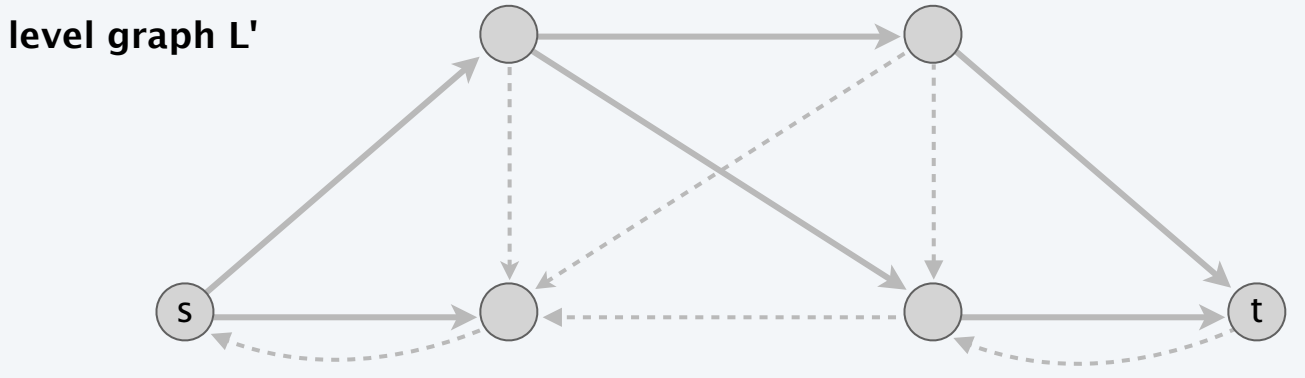


Shortest augmenting path: analysis

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

- The bottleneck edge(s) is deleted from L after each augmentation.
- No new edge added to L until length of shortest path strictly increases. ▪





Shortest augmenting path: review of analysis

L1. Throughout the algorithm, length of the shortest path never decreases.

L2. After at most m shortest path augmentations, the length of the shortest augmenting path strictly increases.

Theorem. The shortest augmenting path algorithm runs in $O(m^2 n)$ time.
Pf.

- $O(m + n)$ time to find shortest augmenting path via BFS.
- $O(m)$ augmentations for paths of exactly k edges.
- $O(m n)$ augmentations. ▪

Shortest augmenting path: improving the running time

Note. $\Theta(m\,n)$ augmentations necessary on some networks.

- Try to decrease time per augmentation instead.
- Simple idea $\Rightarrow$ $O(m\,n^2)$ [Dinic 1970]
- Dynamic trees $\Rightarrow$ $O(m\,n \log n)$ [Sleator-Tarjan 1983]

A Data Structure for Dynamic Trees

DANIEL D. SLEATOR AND ROBERT ENDRE TARJAN

Bell Laboratories, Murray Hill, New Jersey 07974

Received May 8, 1982; revised October 18, 1982

A data structure is proposed to maintain a collection of vertex-disjoint trees under a sequence of two kinds of operations: a *link* operation that combines two trees into one by adding an edge, and a *cut* operation that divides one tree into two by deleting an edge. Each operation requires $O(\log n)$ time. Using this data structure, new fast algorithms are obtained for the following problems:

(1) Computing nearest common ancestors.

(2) Solving various network flow problems including finding maximum flows, blocking flows, and acyclic flows.

(3) Computing certain kinds of constrained minimum spanning trees.

(4) Implementing the network simplex algorithm for minimum-cost flows.

The most significant application is (2); an $O(mn \log n)$-time algorithm is obtained to find a maximum flow in a network of n vertices and m edges, beating by a factor of $\log n$ the fastest algorithm previously known for sparse graphs.