Maximum flow - Minimum cut (Part I)

Quan Hoang

November 15, 2018

Outline

The problem in the Cold War

Definition of the problem

Ford-Fulkerson algorithm

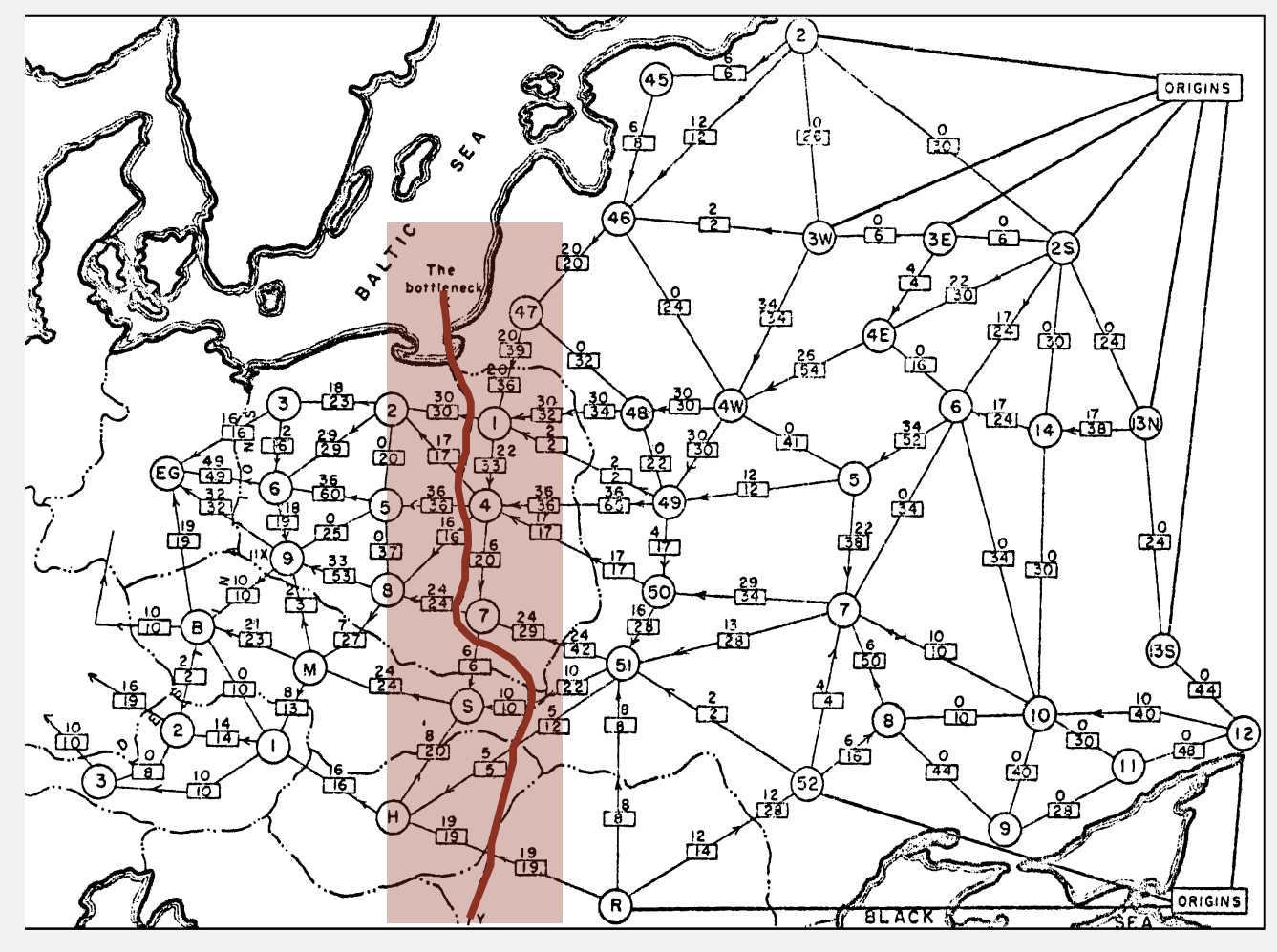
How to improve Ford-Fulkerson algorithm

Better algorithm for max flow problems

The problem in the Cold War

Mincut application (RAND 1950s)

"Free world" goal. Cut supplies (if cold war turns into real war).



rail network connecting Soviet Union with Eastern European countries
(map declassified by Pentagon in 1999)

Definition of the problem

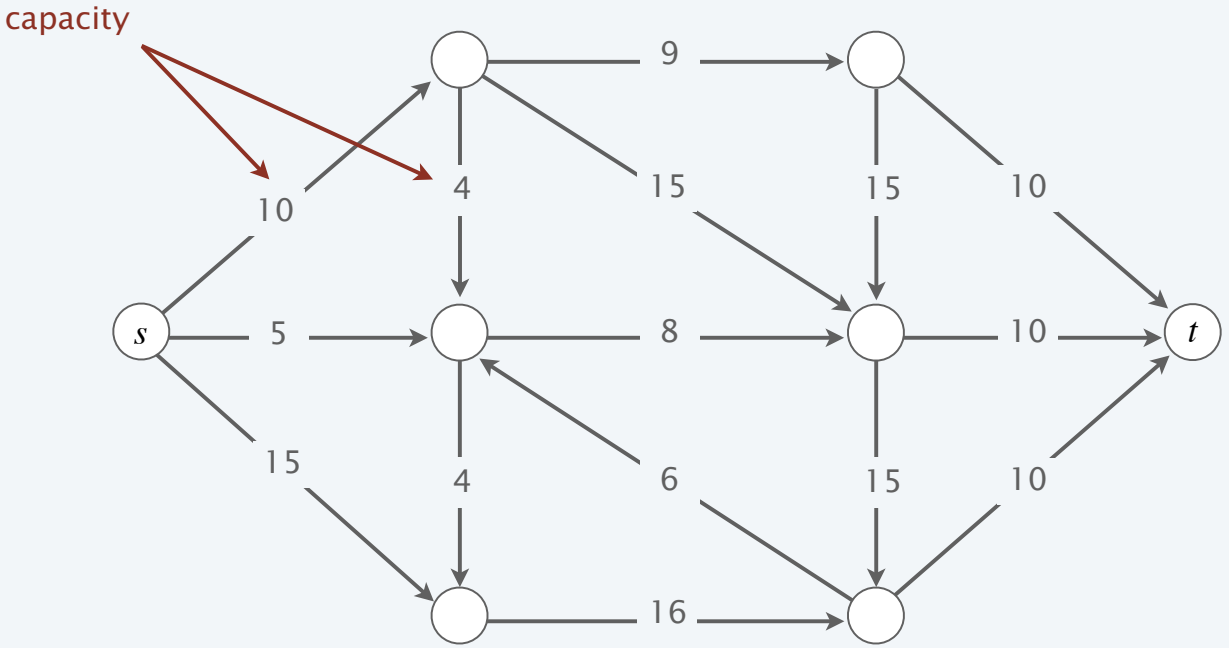
Flow network

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 $c(e) > 0$ 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.

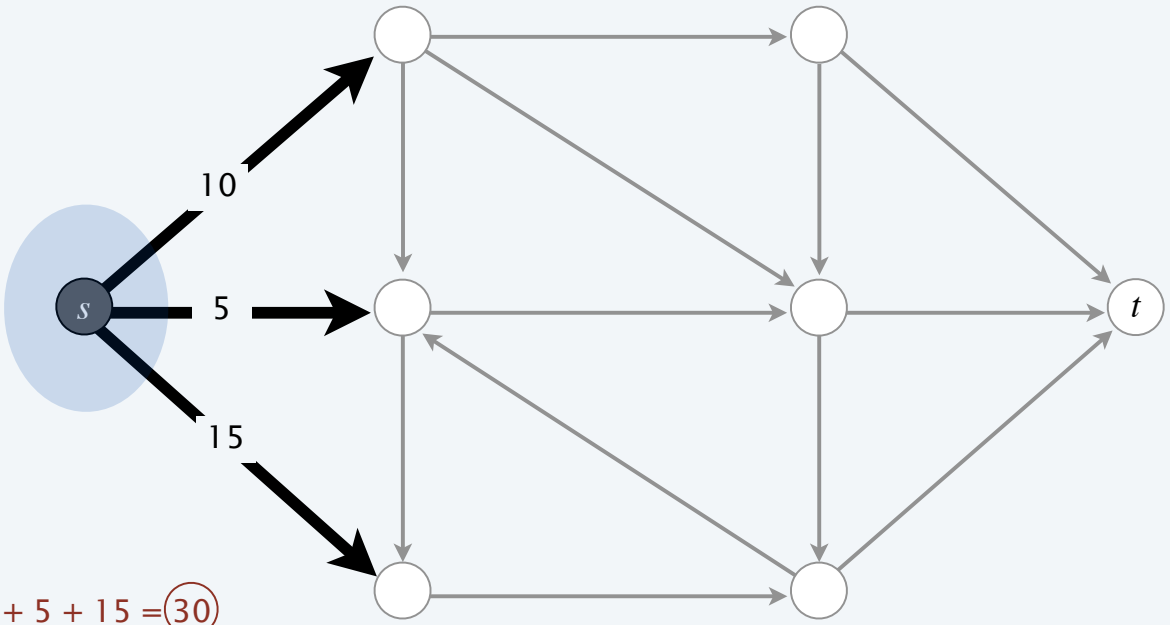


Minimum-cut problem

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.

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



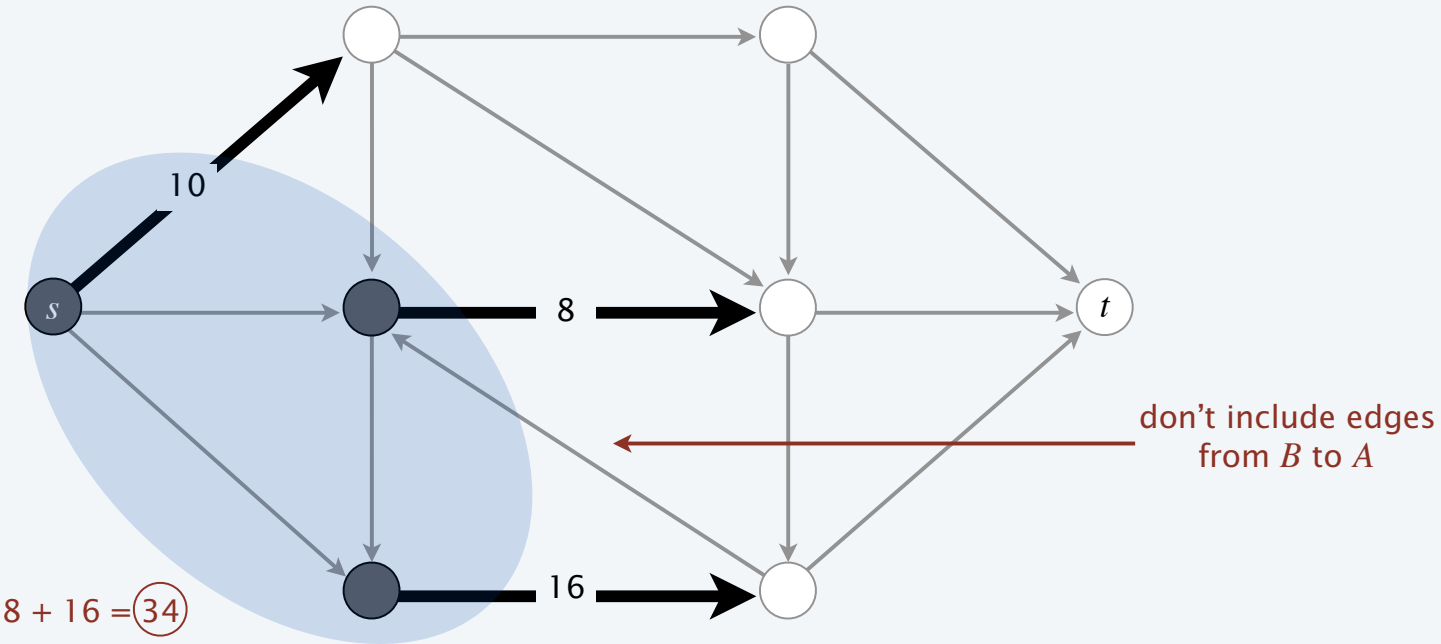
capacity = 10 + 5 + 15 = 30

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.

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



capacity = 10 + 8 + 16 = 34

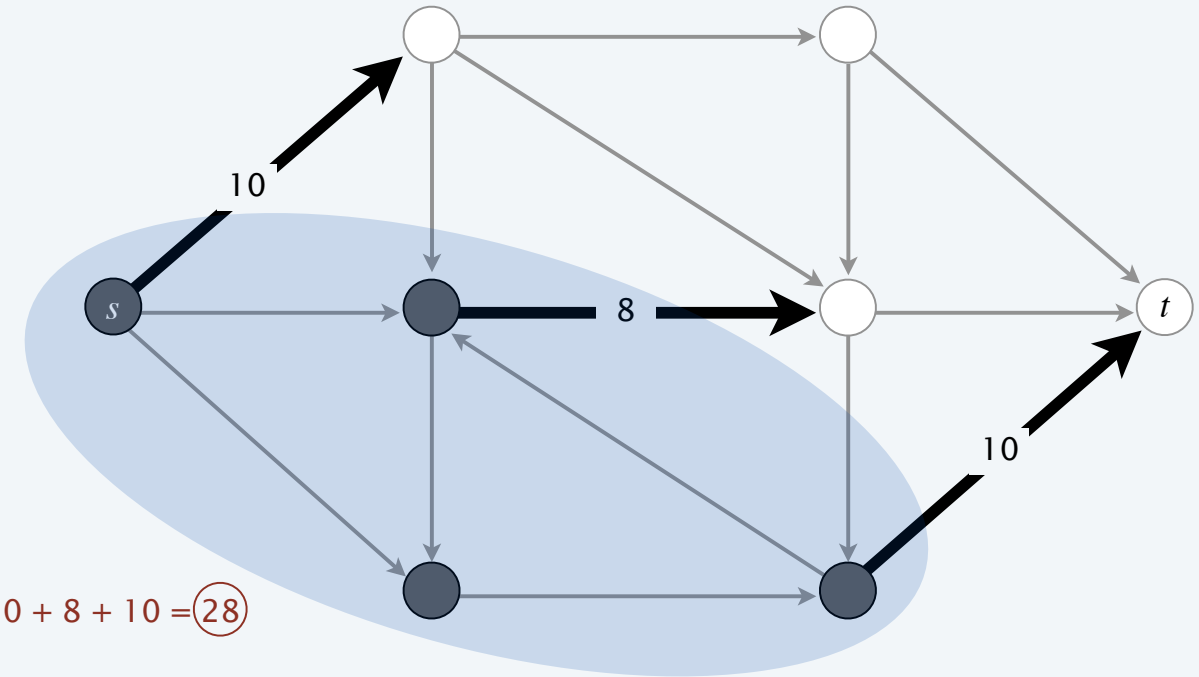
Minimum-cut problem

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

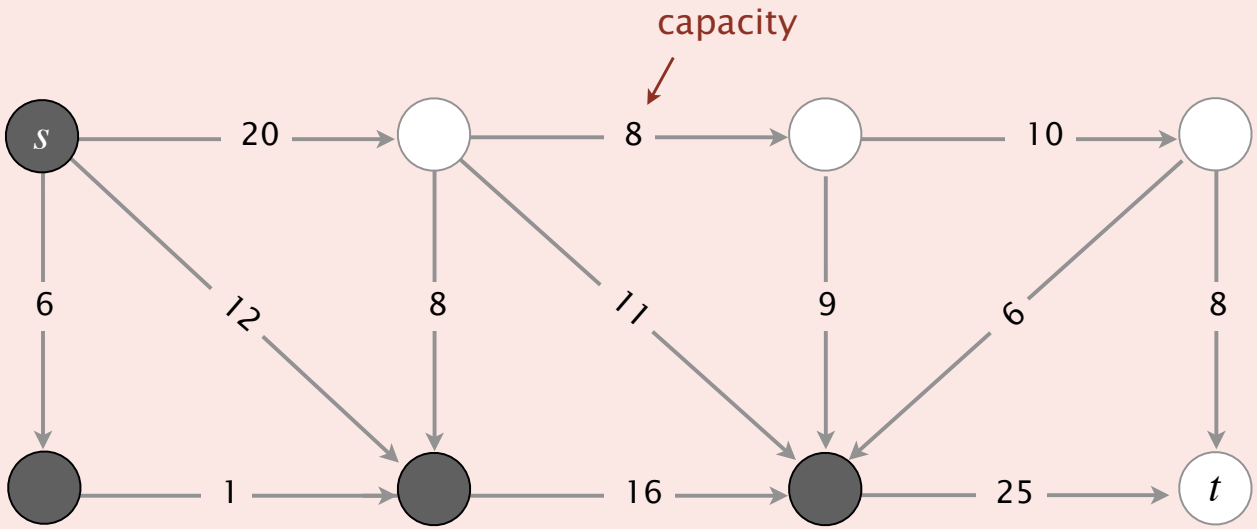
Which is the capacity of the given *st*-cut?

A. 11 (20 + 25 – 8 – 11 – 9 – 6)

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

C. 45 (20 + 25)

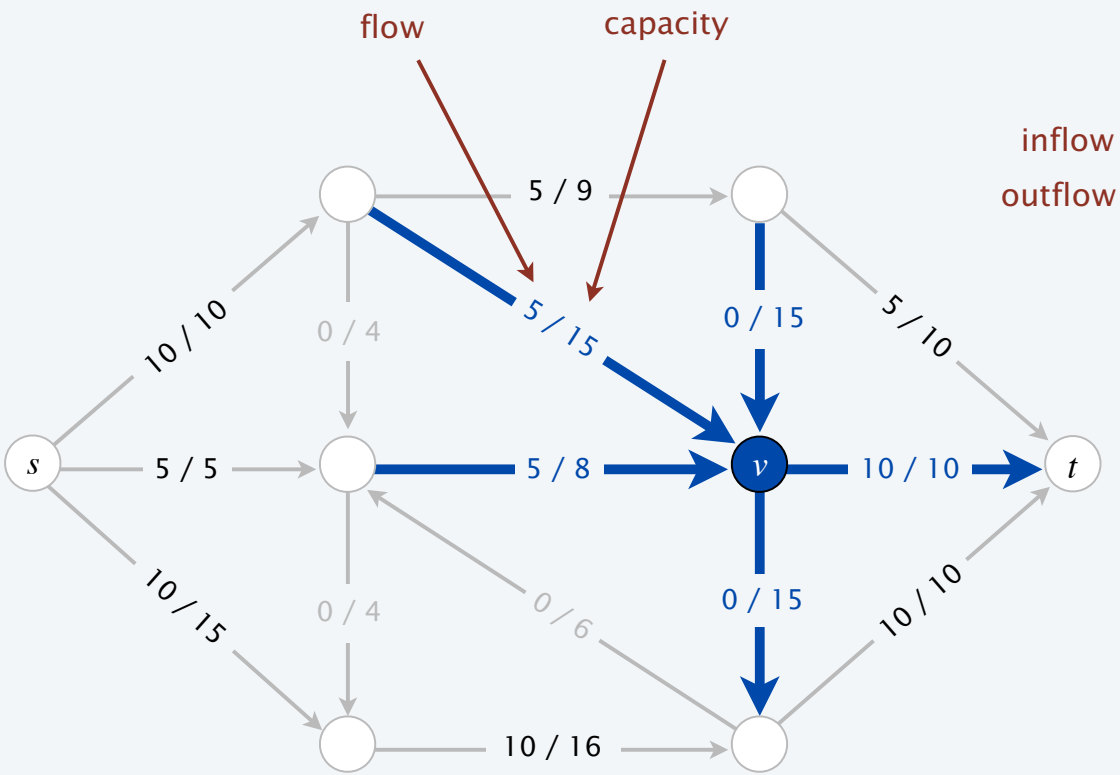
D. 79 (20 + 25 + 8 + 11 + 9 + 6)



Maximum-flow problem

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

- For each $e \in E$: $0 \le f(e) \le c(e)$ [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]



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

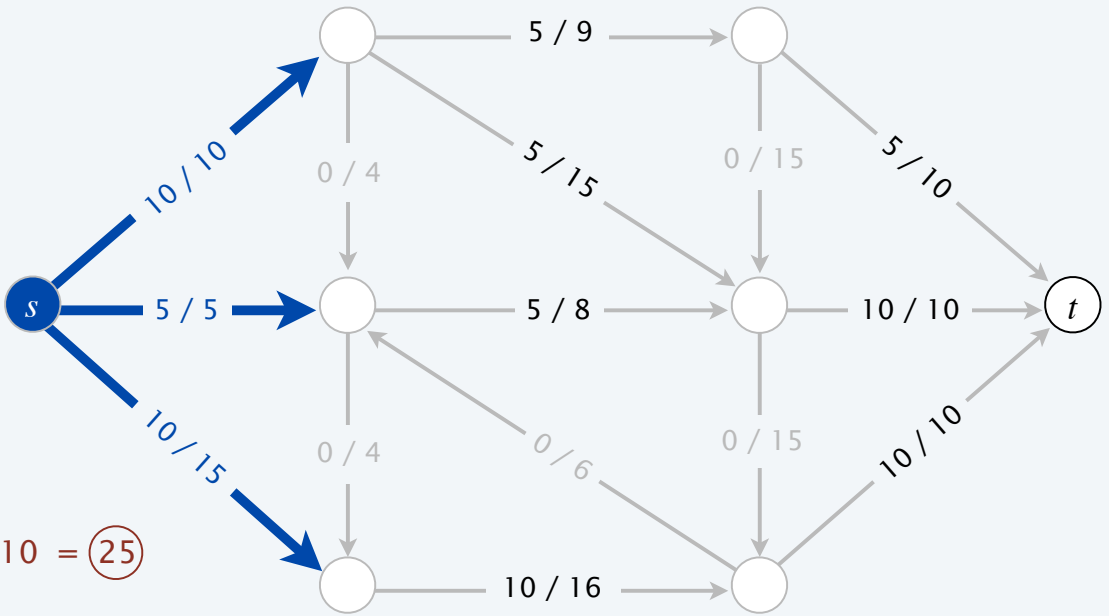
outflow at $v = 10 + 0 = 10$

Maximum-flow problem

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

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



value = 5 + 10 + 10 = 25

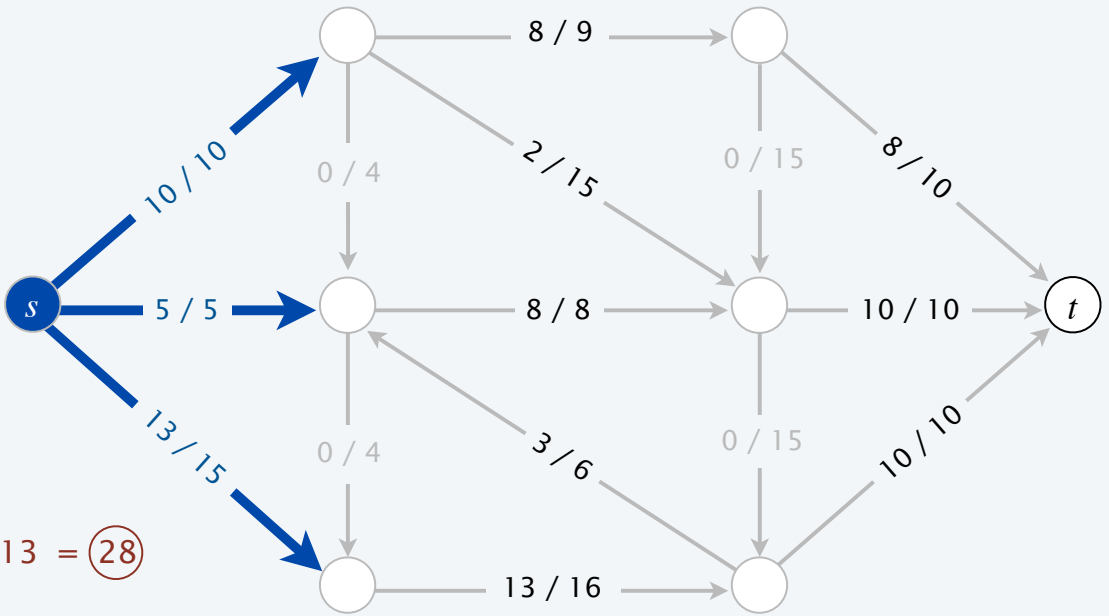
Maximum-flow problem

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

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

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



value = 10 + 5 + 13 = 28

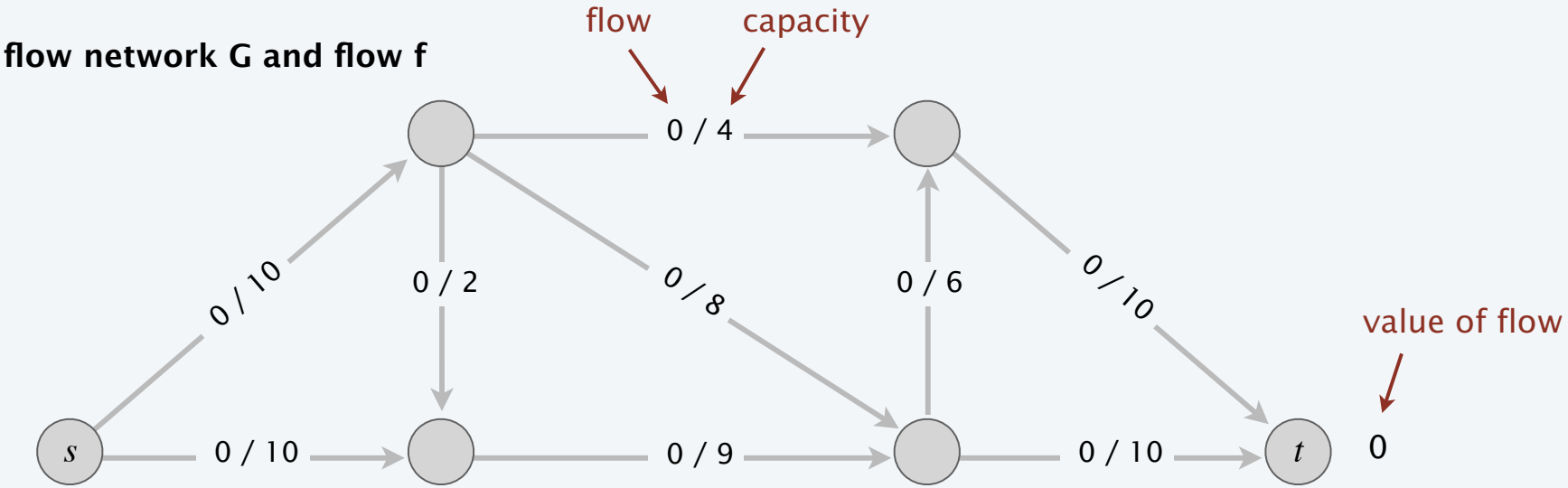
Ford-Fulkerson algorithm

Greedy Algorithm

Toward a max-flow algorithm

Greedy algorithm.

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

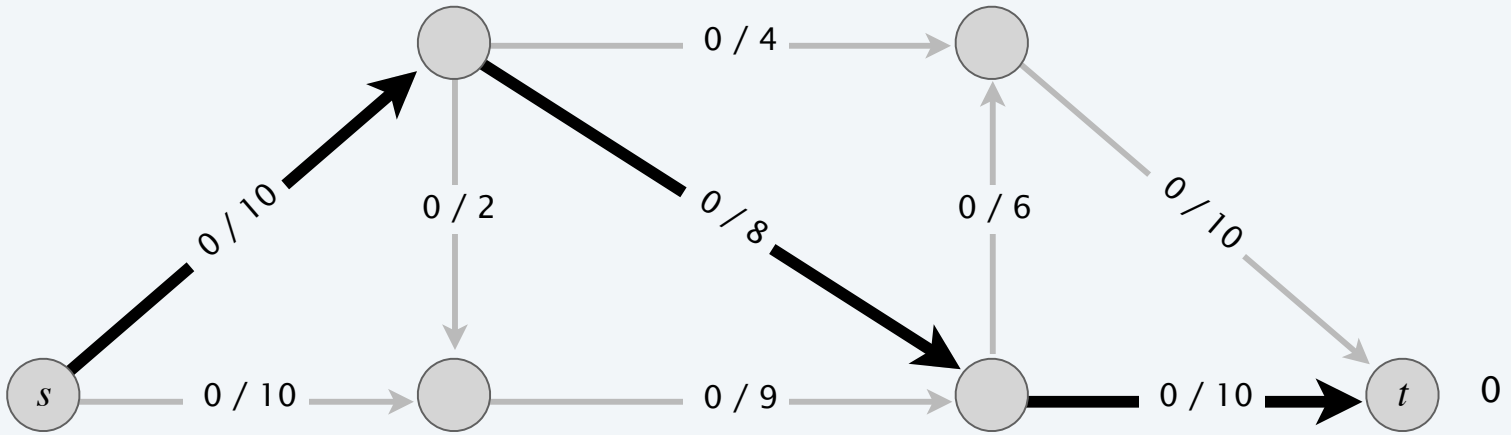


Toward a max-flow algorithm

Greedy algorithm.

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

flow network G and flow f

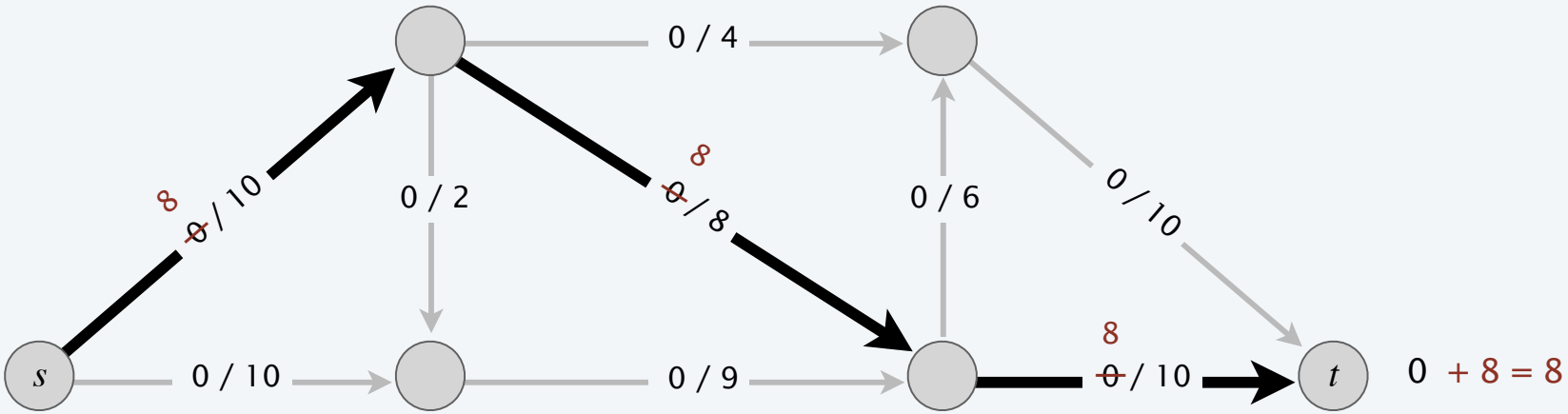


Toward a max-flow algorithm

Greedy algorithm.

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

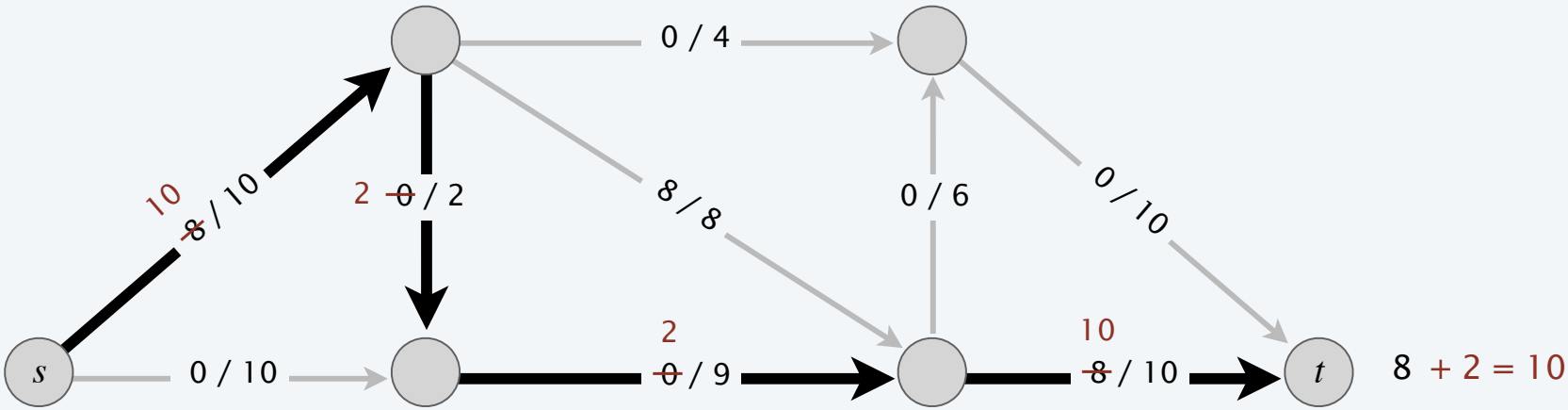


Toward a max-flow algorithm

Greedy algorithm.

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

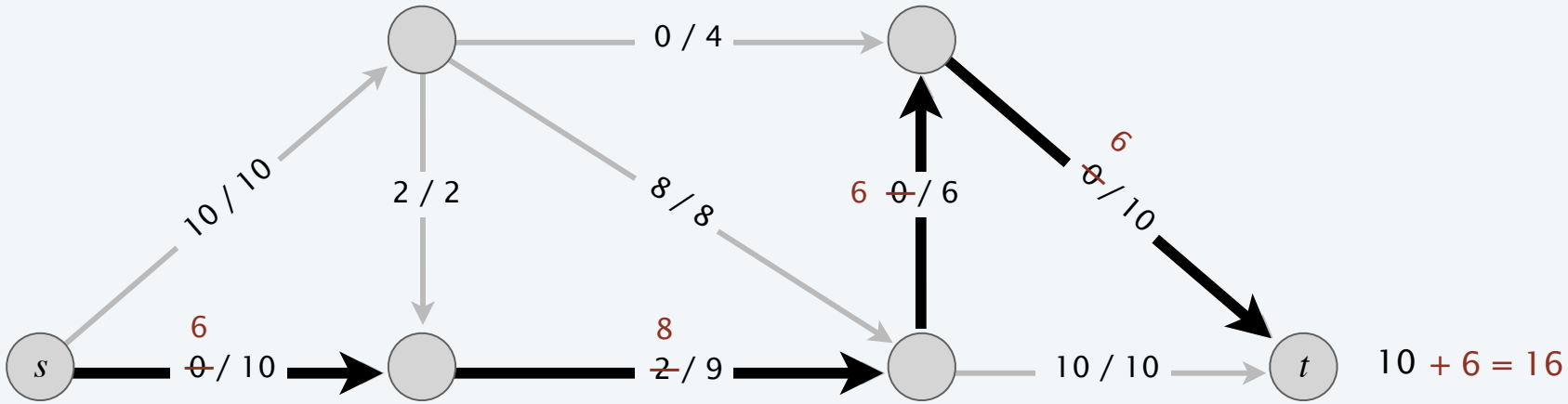


Toward a max-flow algorithm

Greedy algorithm.

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



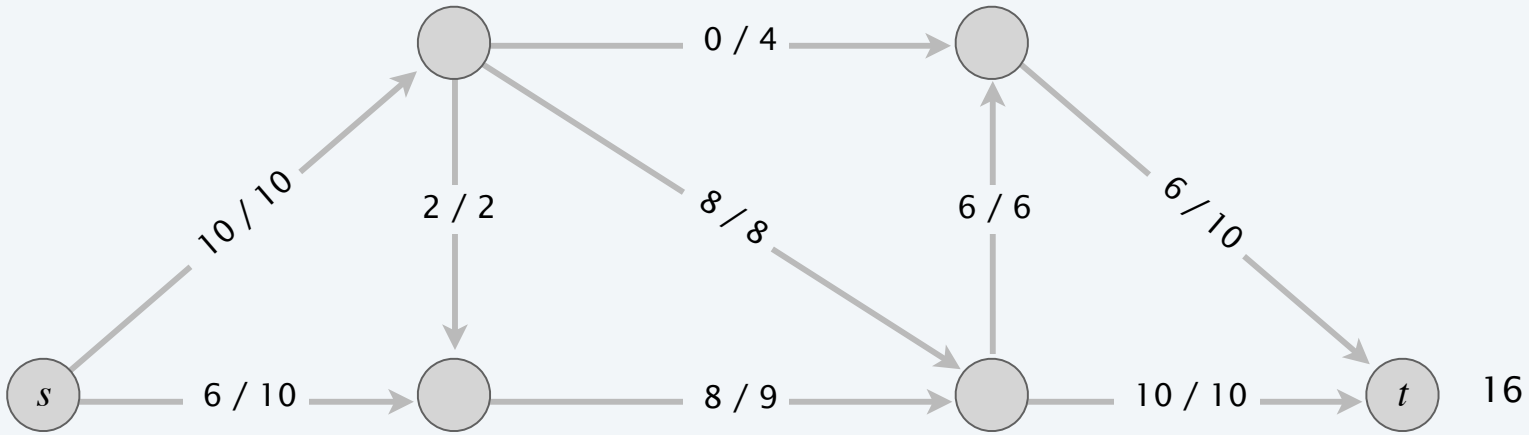
Toward a max-flow algorithm

Greedy algorithm.

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

ending flow value = 16

flow network G and flow f



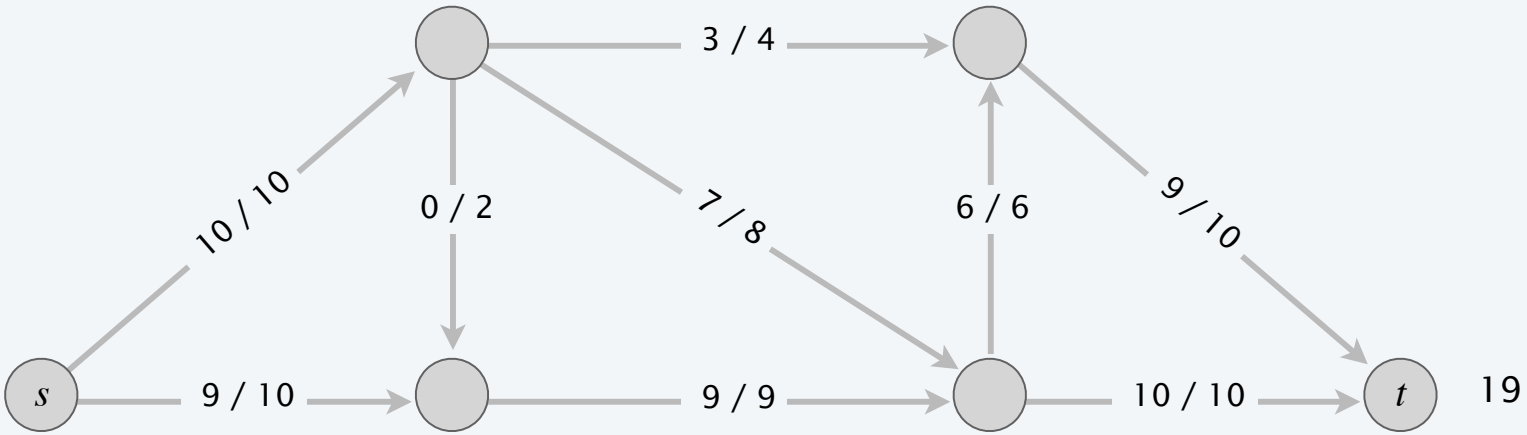
Toward a max-flow algorithm

Greedy algorithm.

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

flow network G and flow f



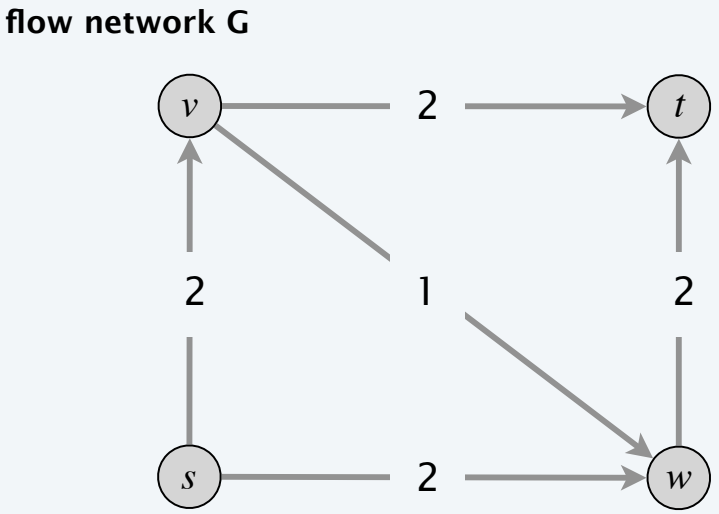
Why the greedy algorithm fails

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



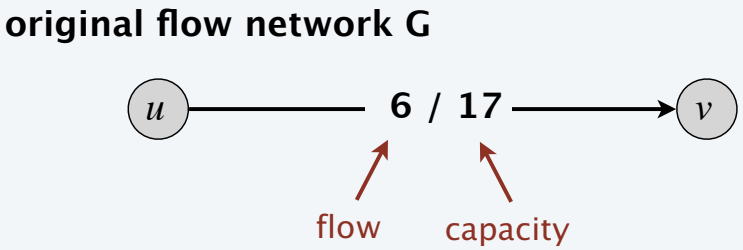
Bottom line. Need some mechanism to "undo" a bad decision.

Residual graph - Augmenting path

Residual network

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

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

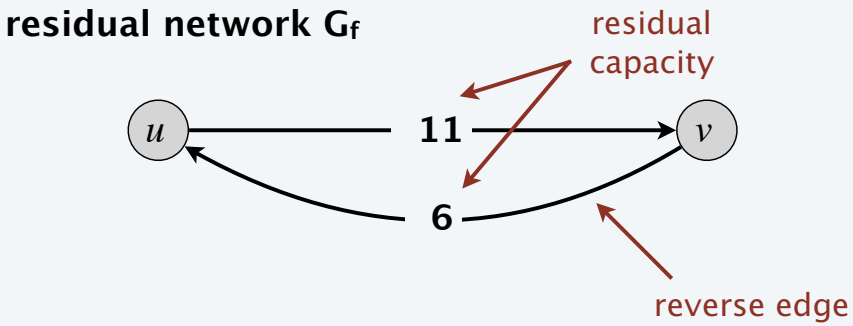


Reverse edge. $e^{\text{reverse}} = (v, u)$.

- "Undo" flow sent.

Residual capacity.

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



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

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

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 $f' \leftarrow \text{AUGMENT}(f, c, P)$, 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

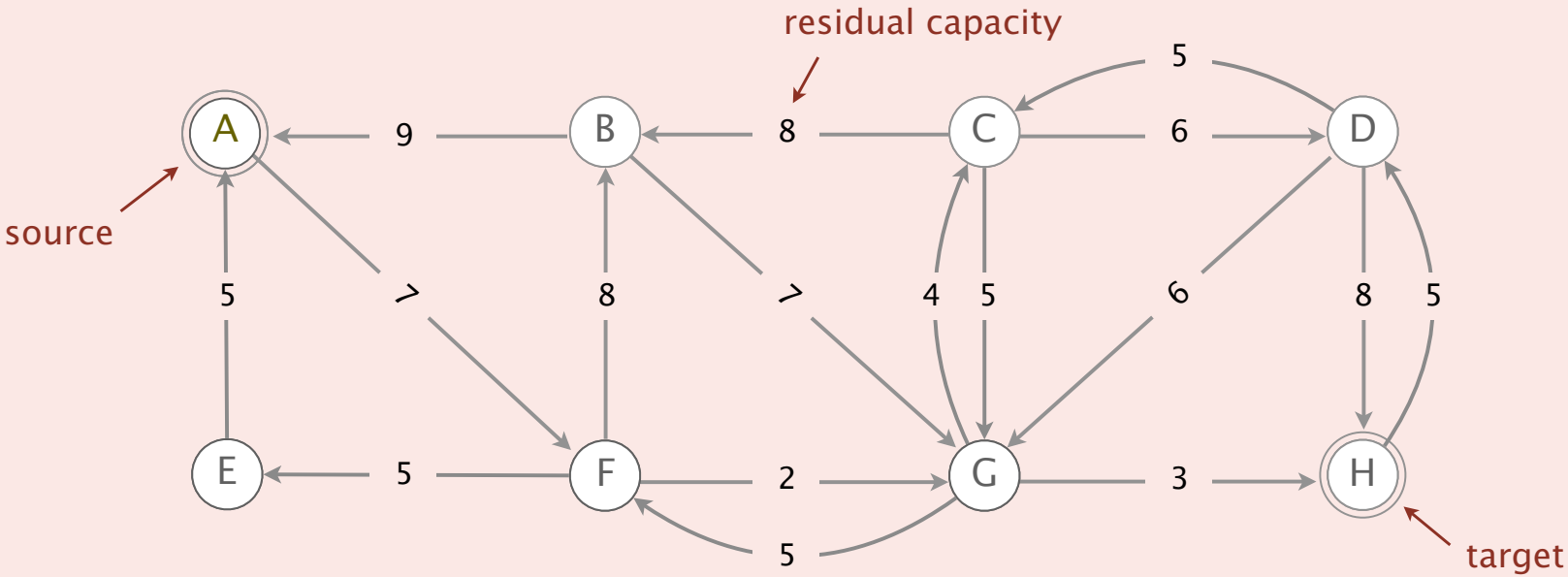
Which is the augmenting path of highest bottleneck capacity?

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 $s \rightarrow t$ 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.

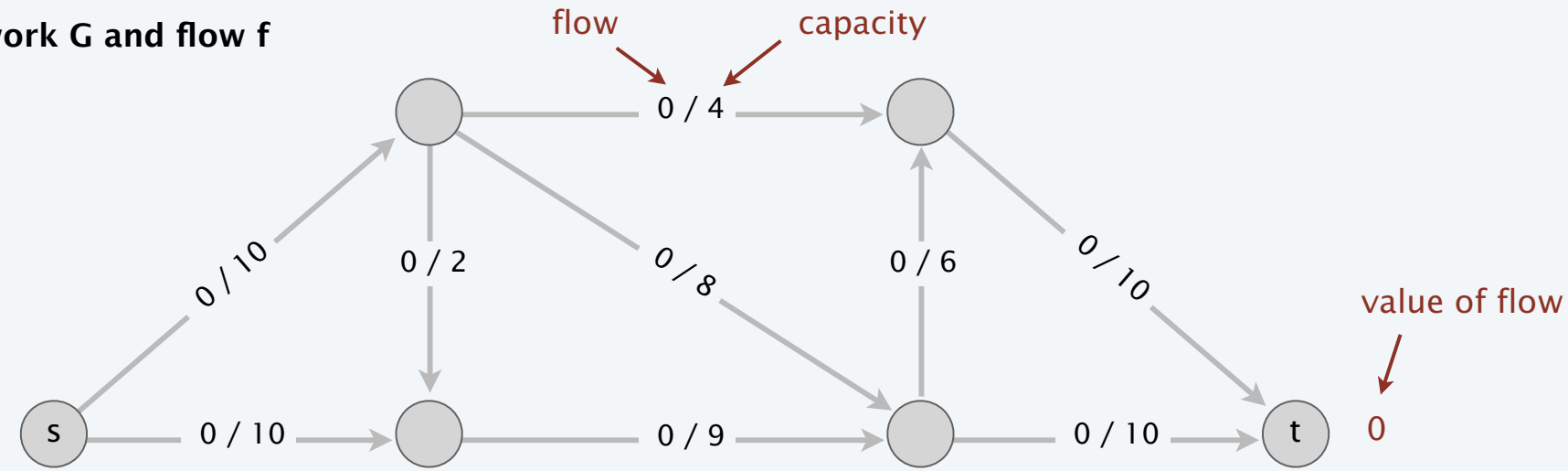
RETURN f.

augmenting path

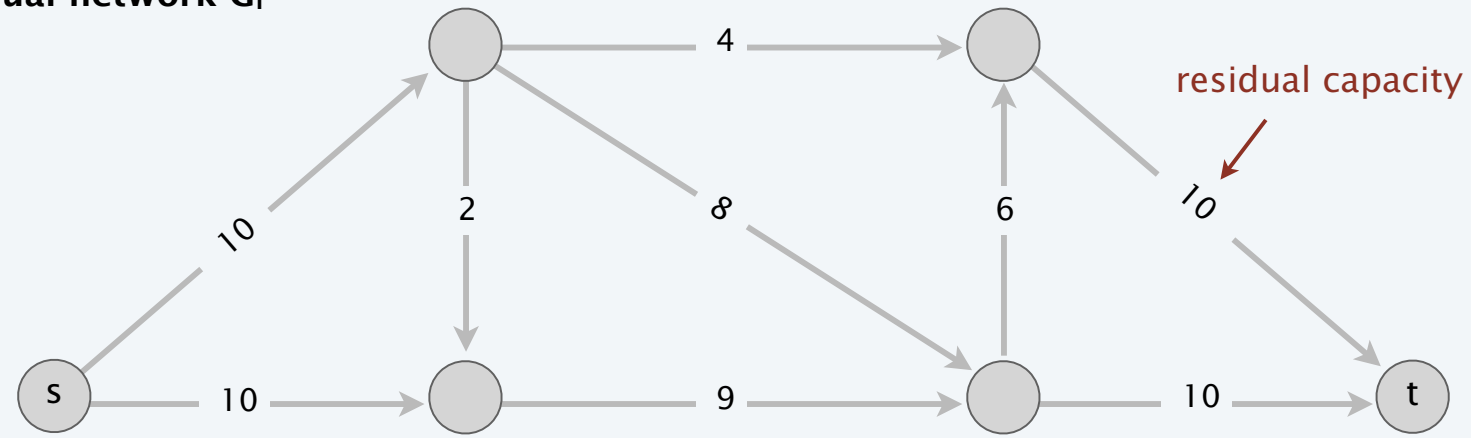
Example

Ford–Fulkerson algorithm demo

network G and flow f

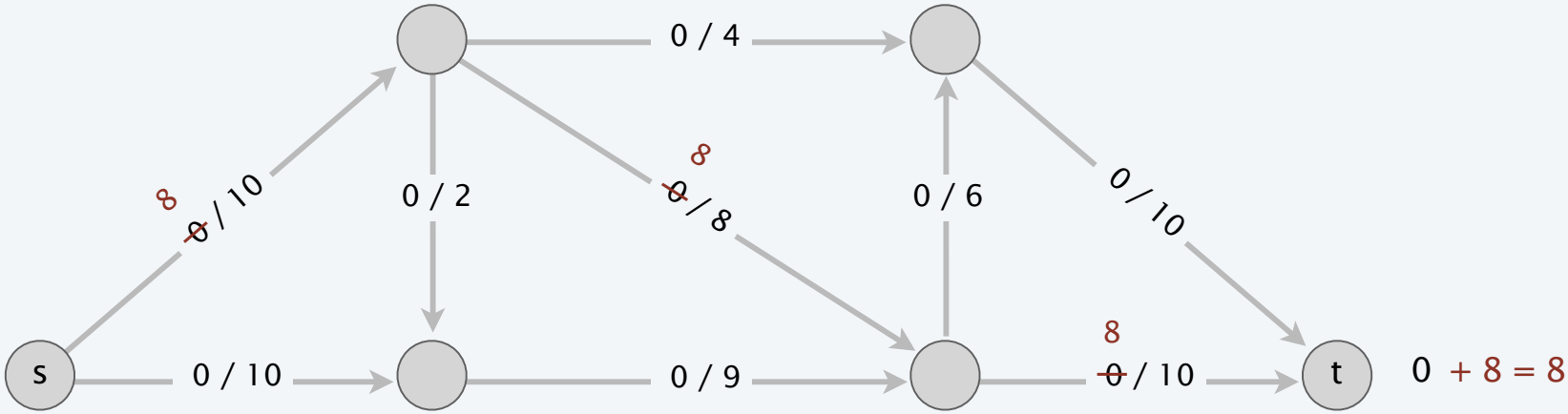


residual network G_f

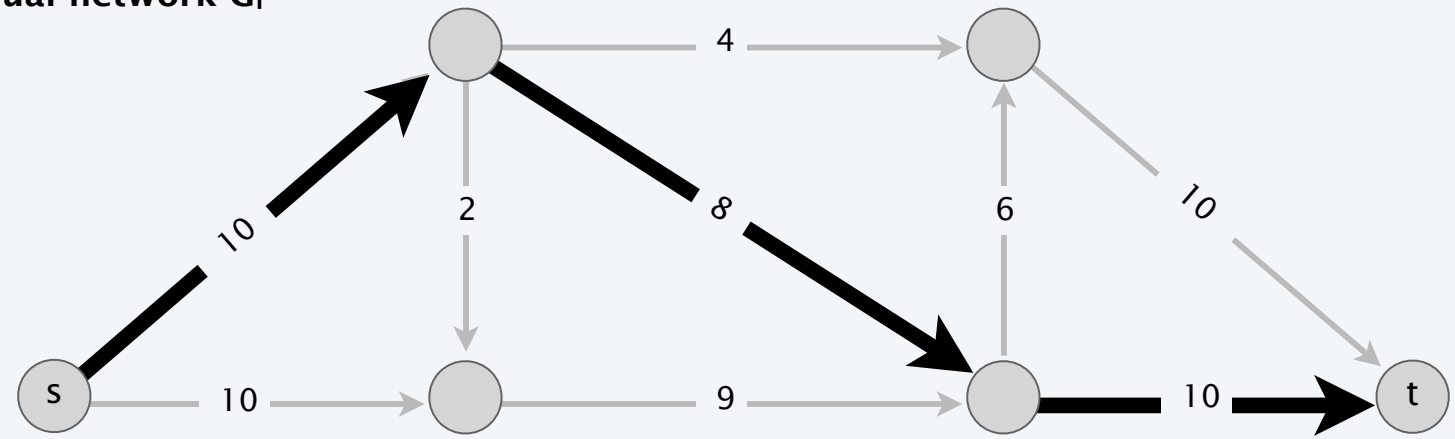


Ford–Fulkerson algorithm demo

network G and flow f

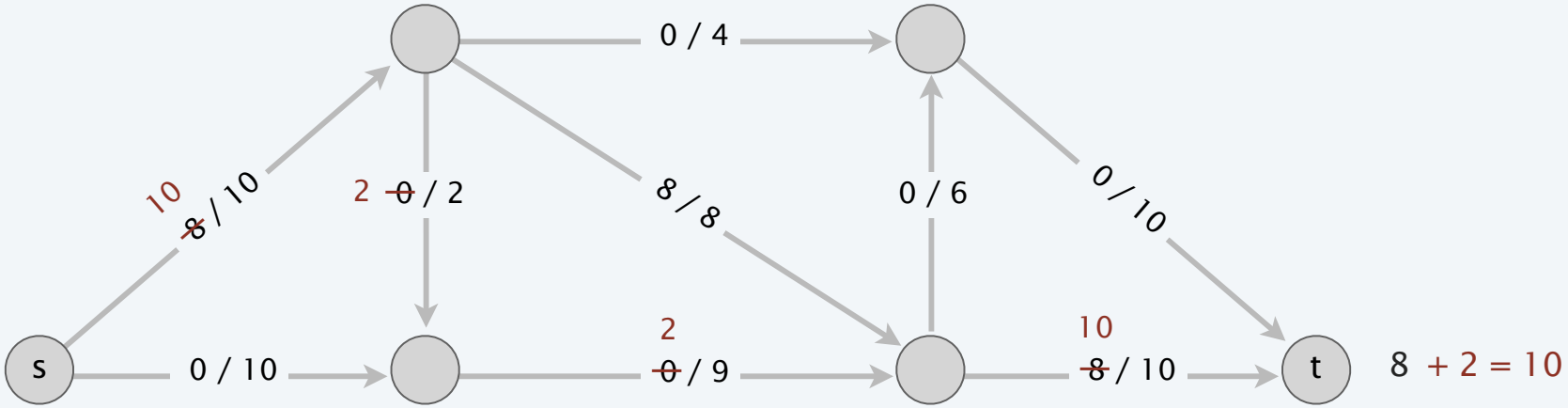


residual network G_f

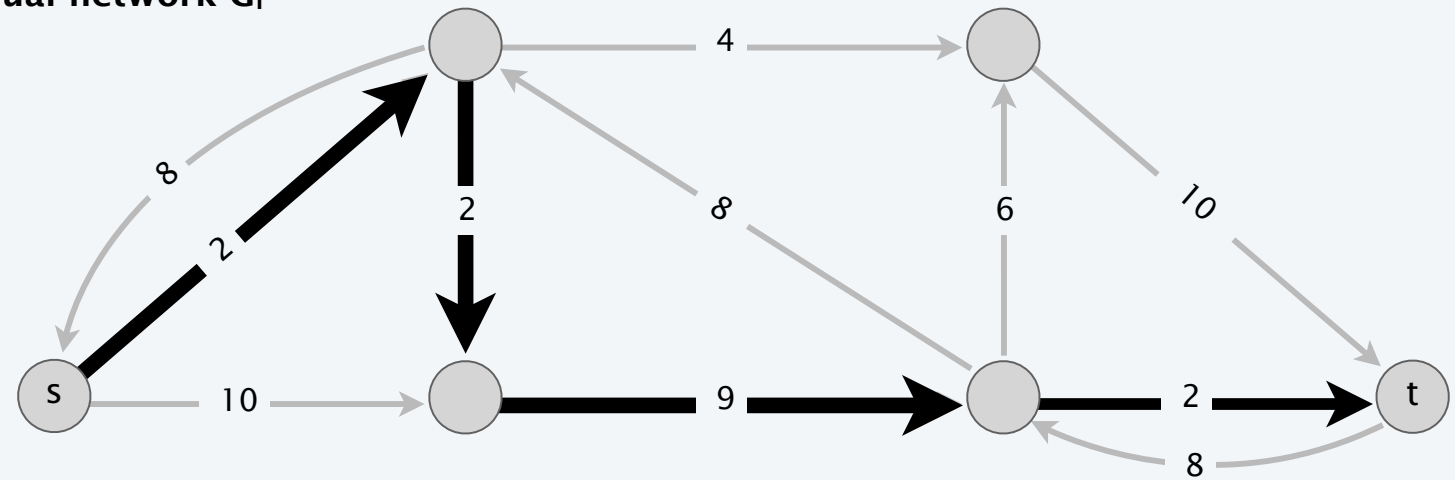


Ford–Fulkerson algorithm demo

network G and flow f

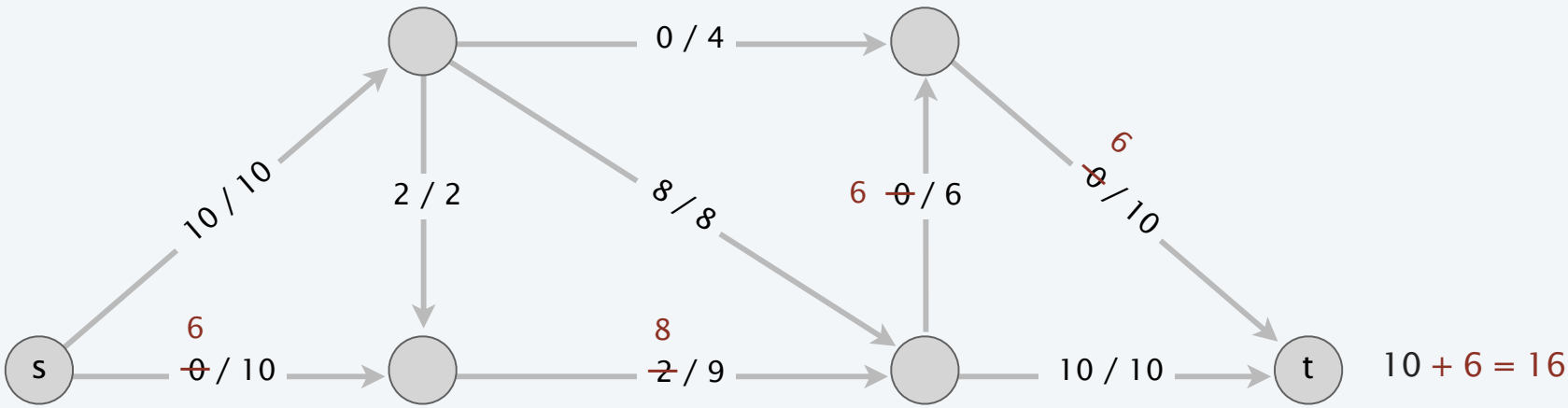


residual network G_f

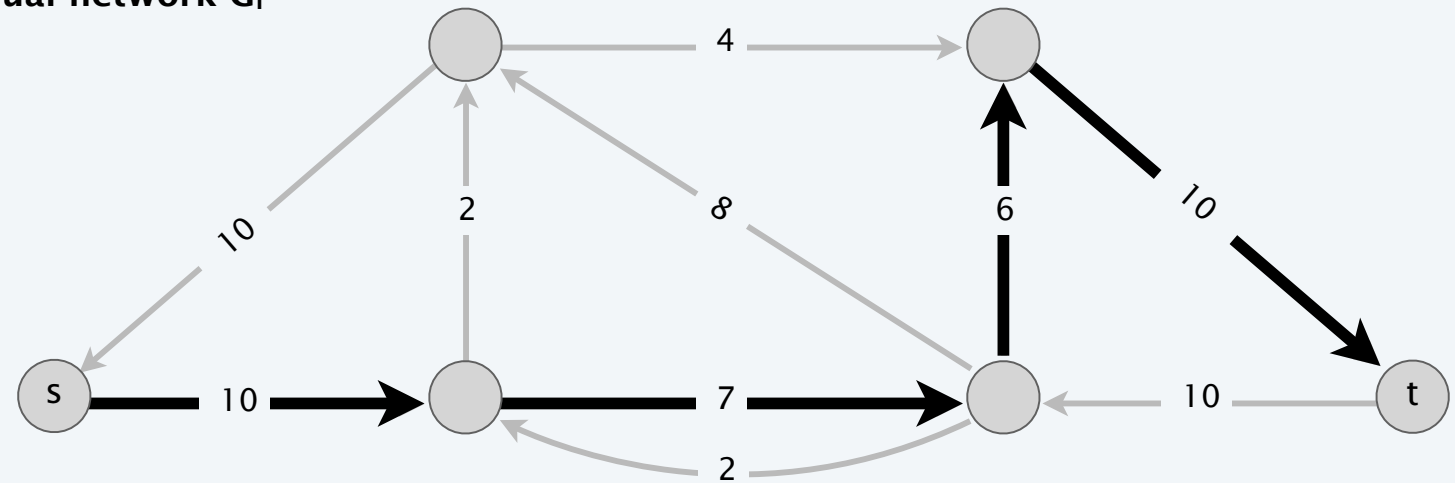


Ford–Fulkerson algorithm demo

network G and flow f

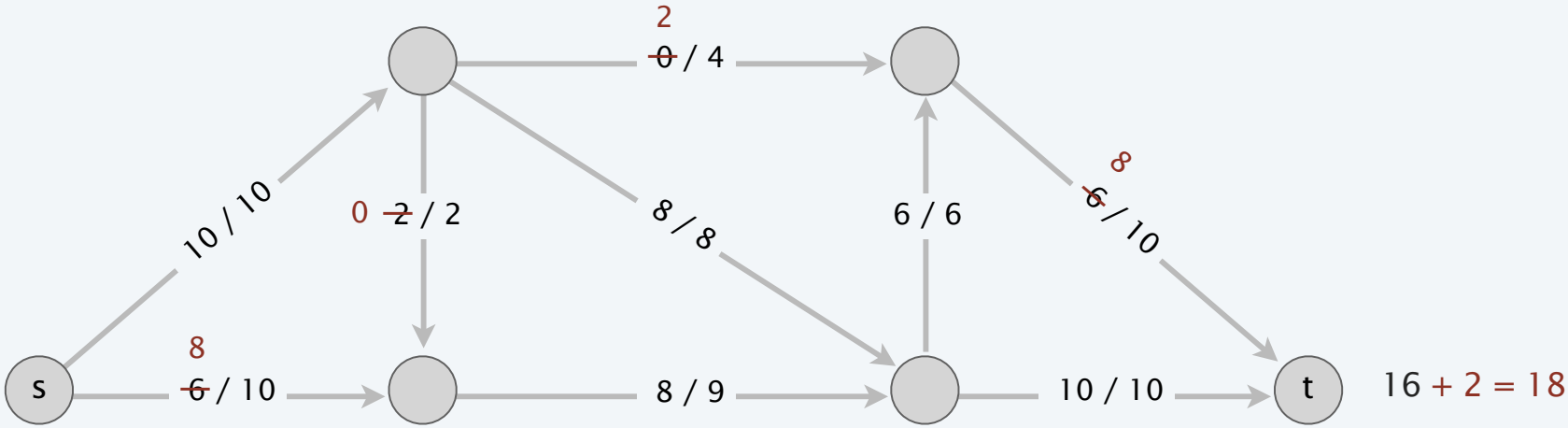


residual network G_f



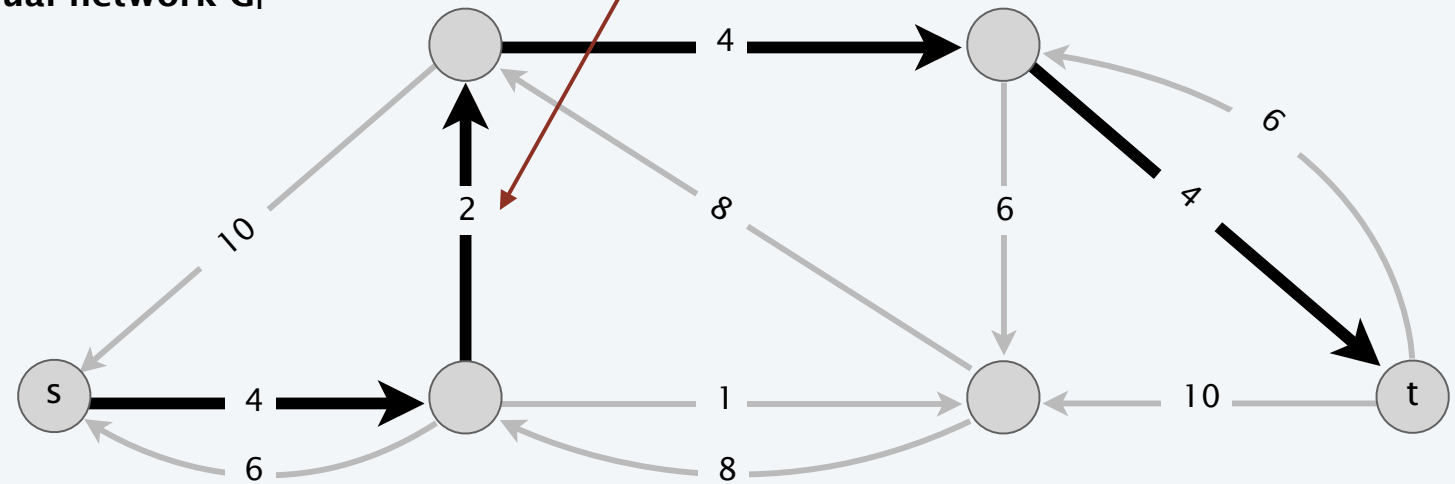
Ford–Fulkerson algorithm demo

network G and flow f



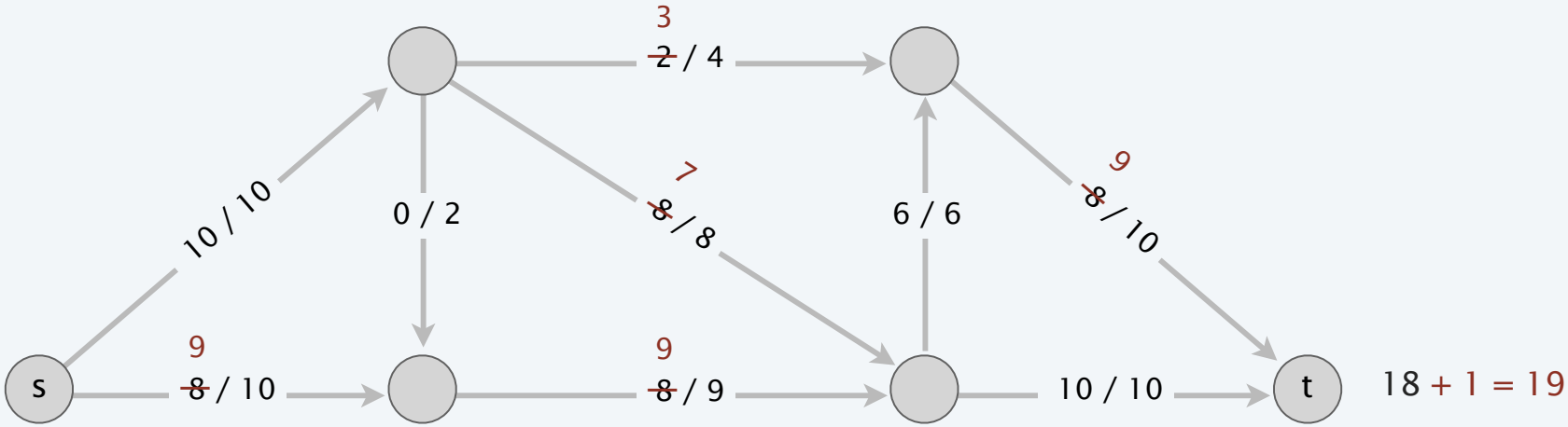
fixes mistake from
second augmenting path

residual network G_f

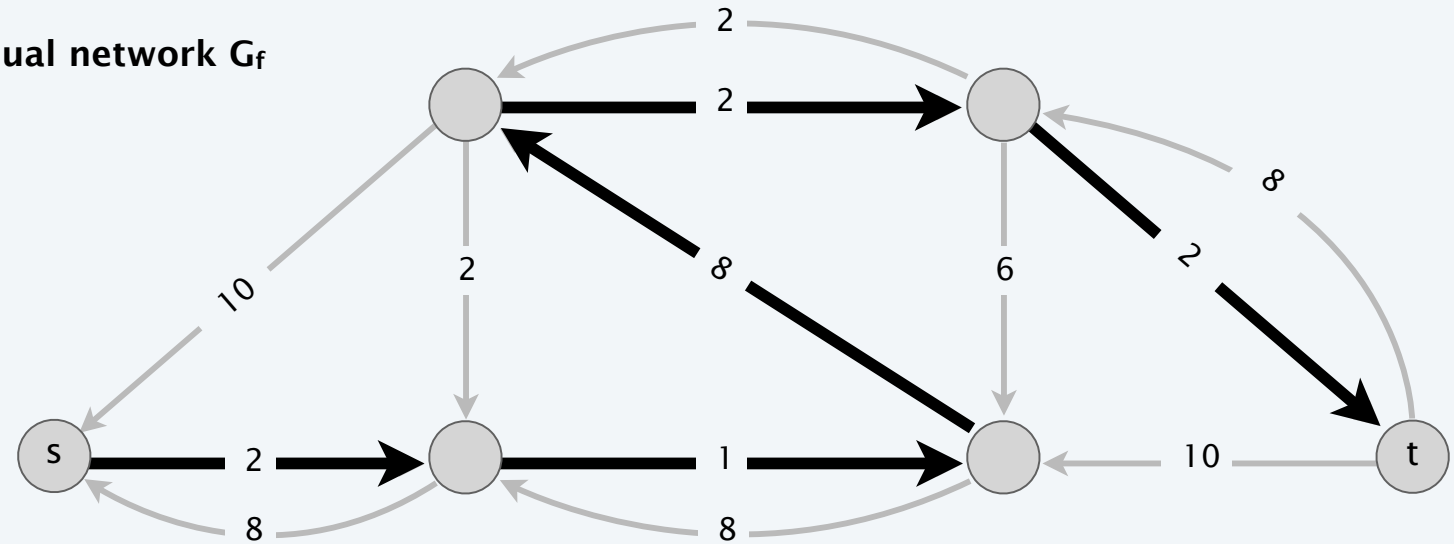


Ford–Fulkerson algorithm demo

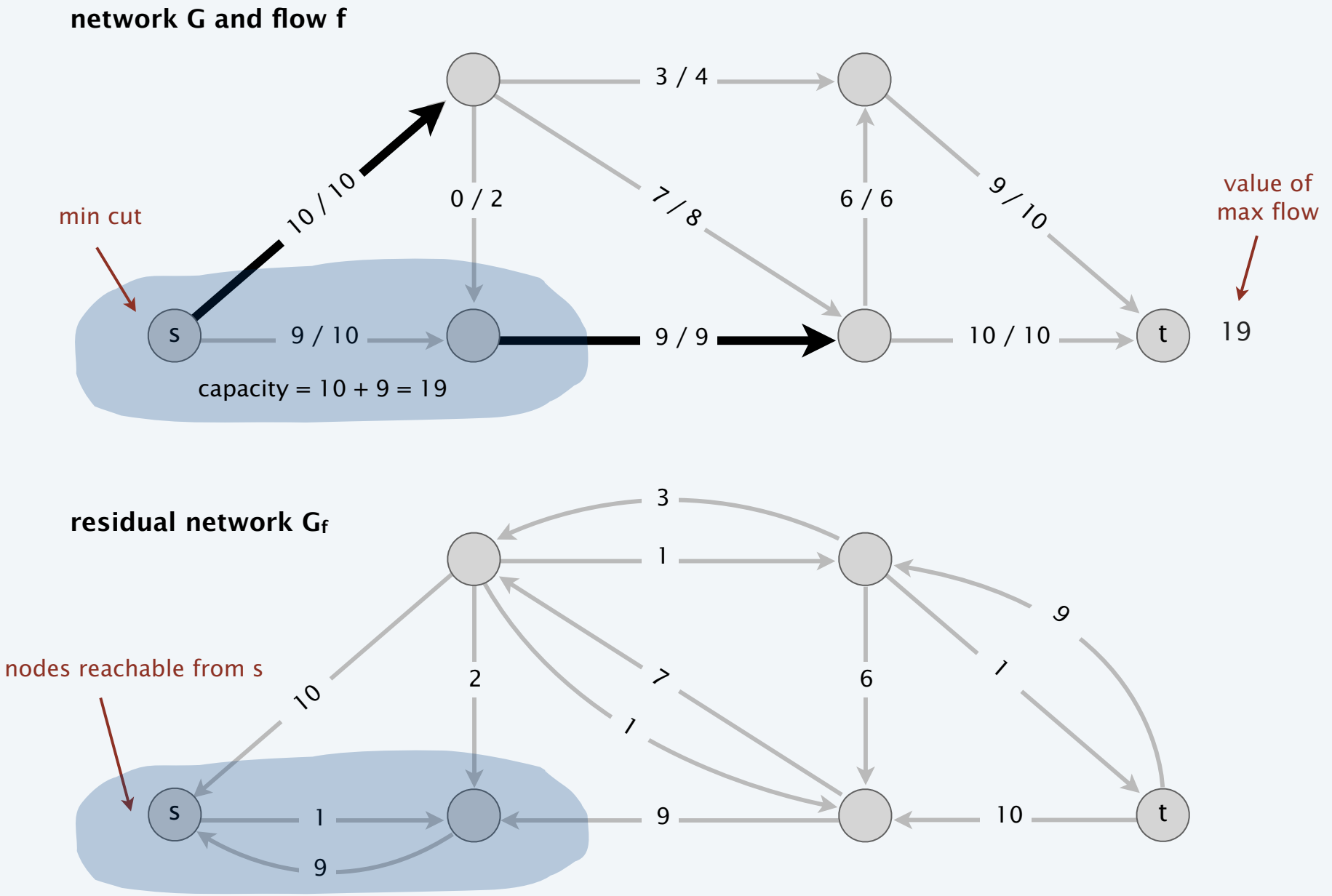
network G and flow f



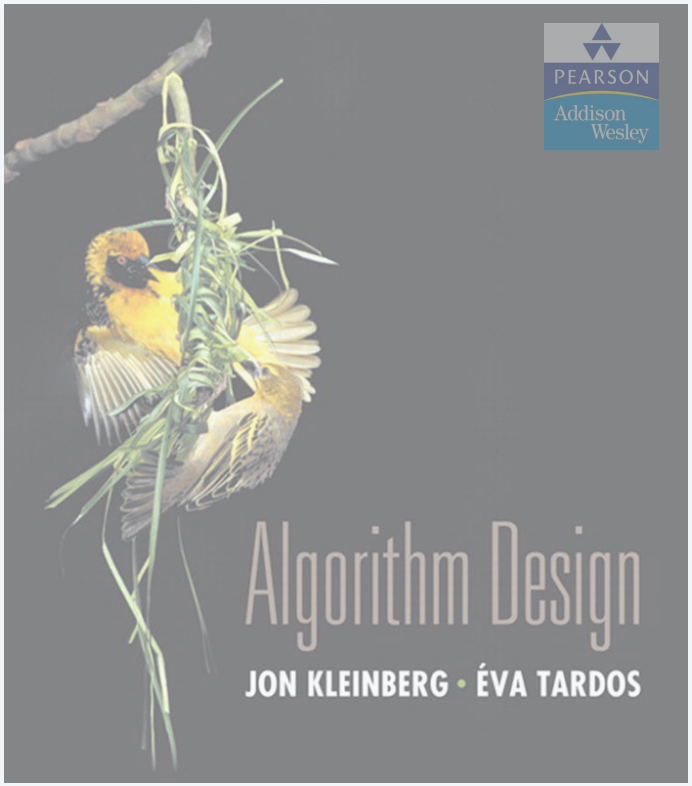
residual network G_f



Ford–Fulkerson algorithm demo



The correctness of the algorithm



SECTION 7.2

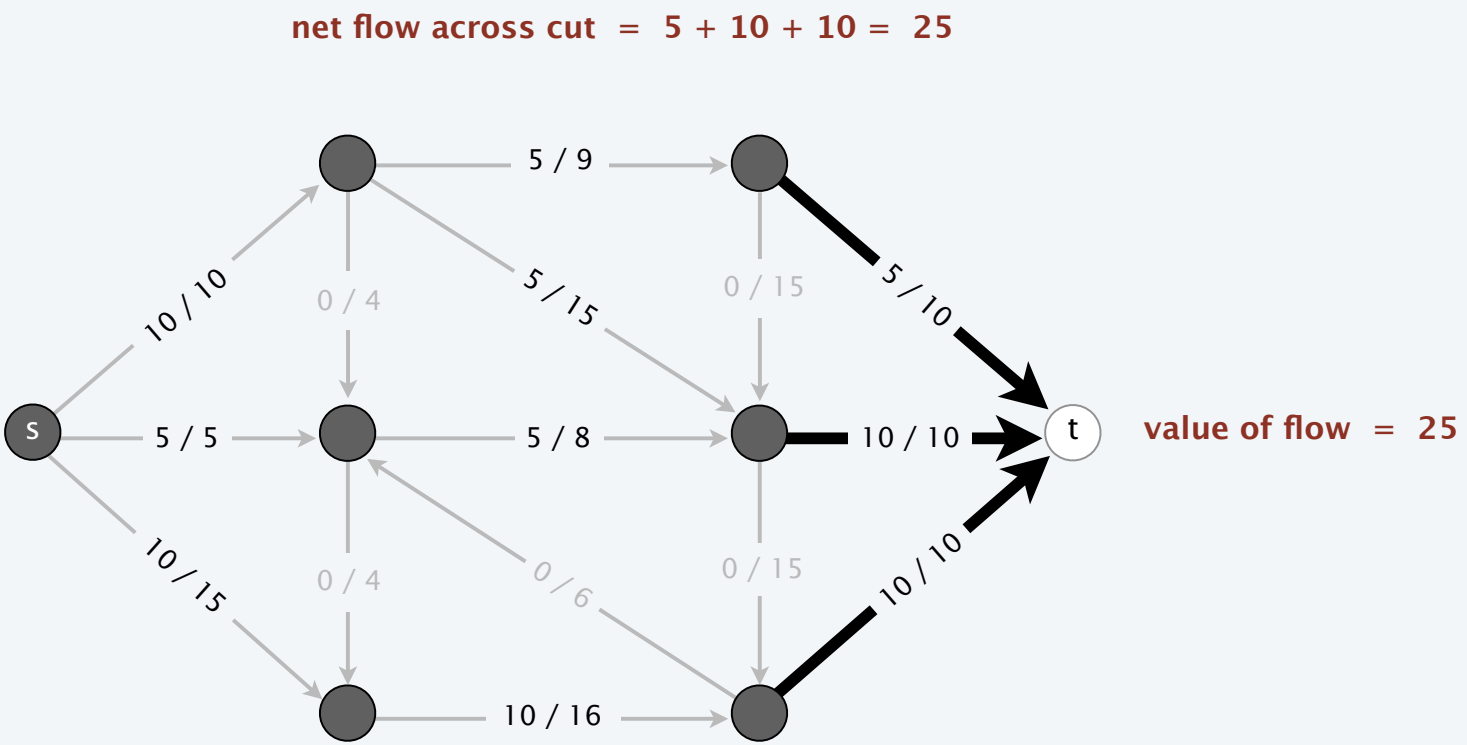
7. NETWORK FLOW I

- *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).

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

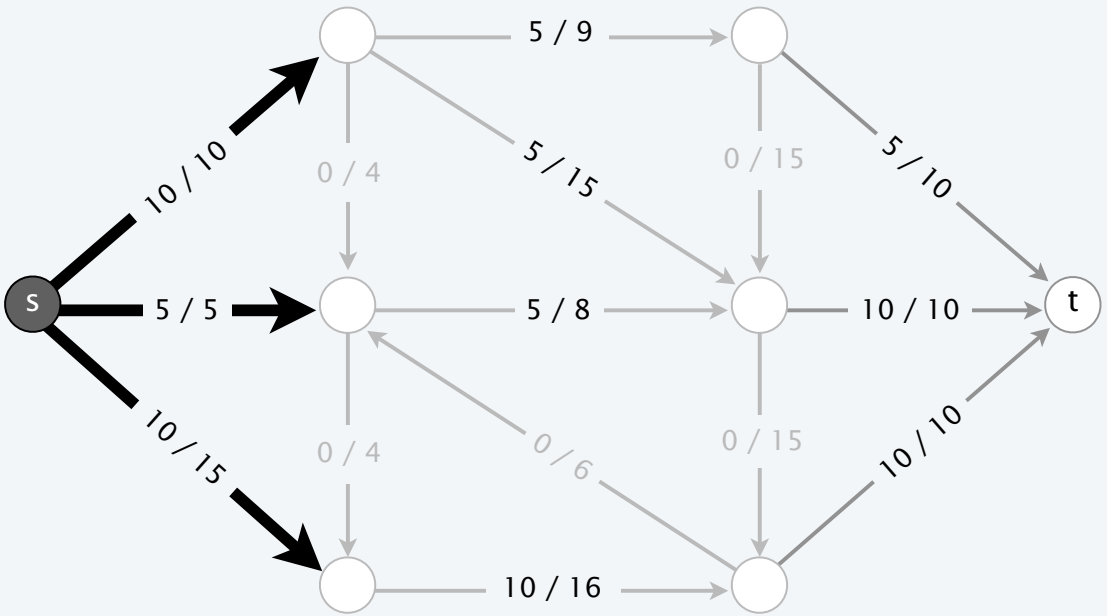


Relationship between flows and cuts

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

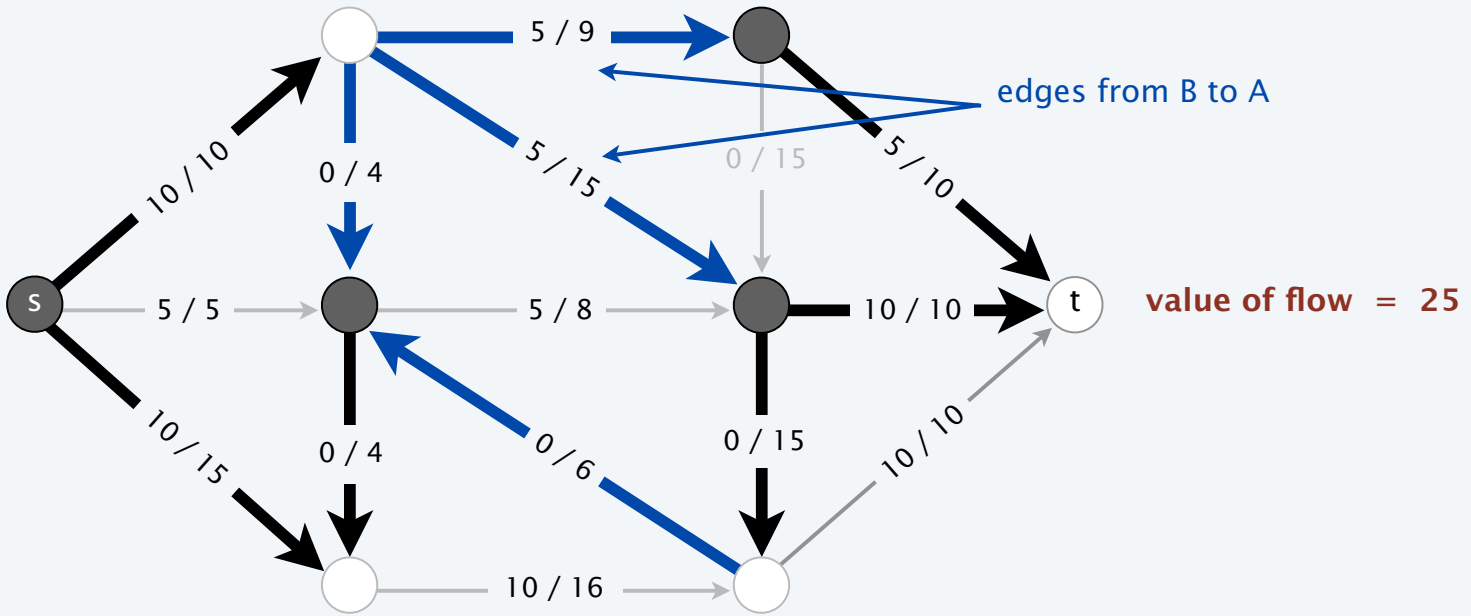


Relationship between flows and cuts

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



Network flow: quiz 3

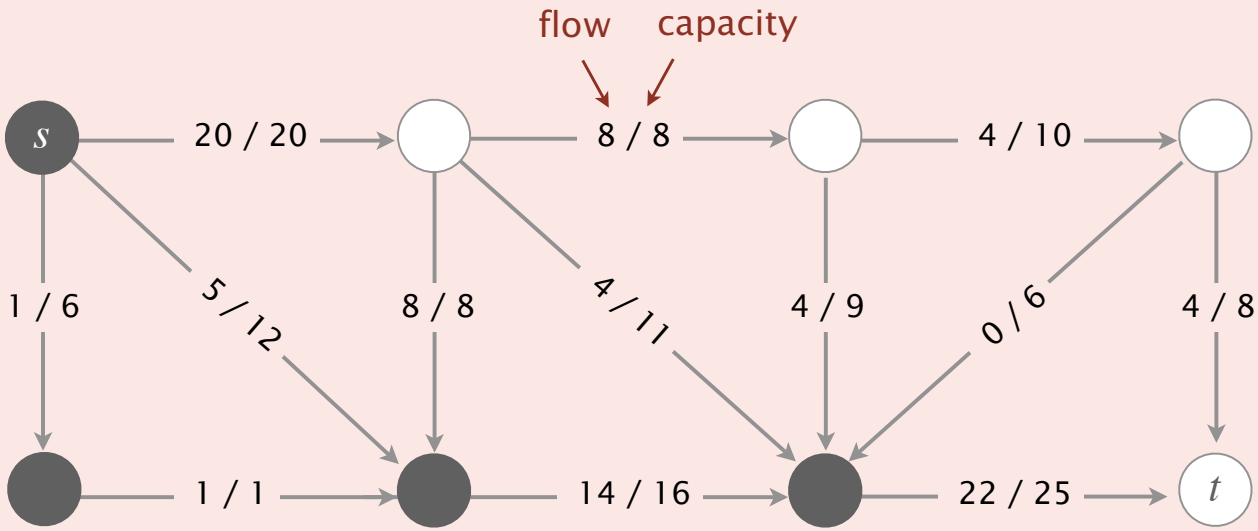
Which is the net flow across the given cut?

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 →

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

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

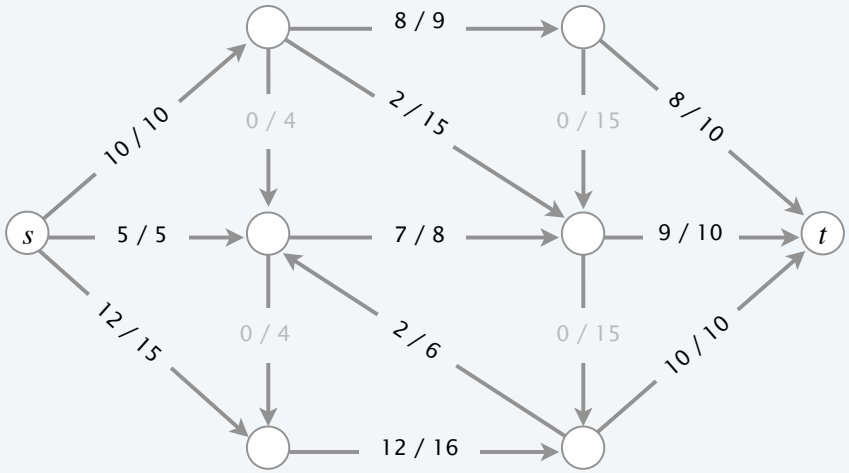
Relationship between flows and cuts

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

Pf.

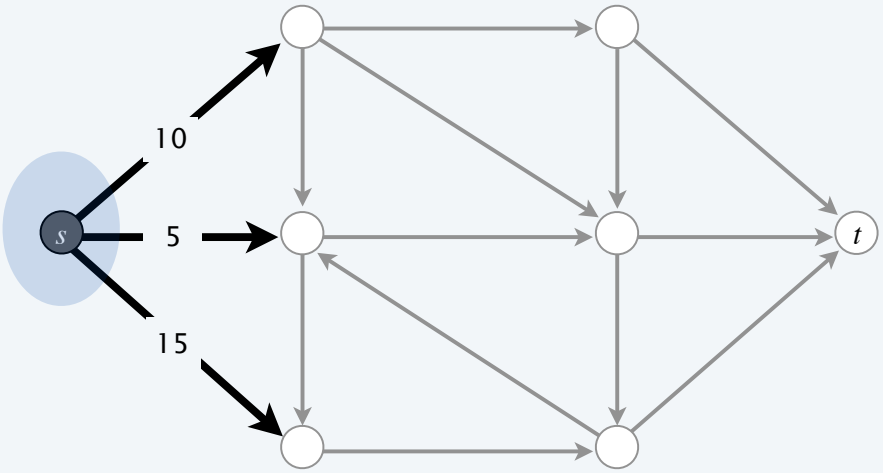
$$\begin{aligned} val(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\le \sum_{e \text{ out of } A} f(e) \\ &\le \sum_{e \text{ out of } A} c(e) \\ &= cap(A, B) \quad \blacksquare \end{aligned}$$

flow value lemma



value of flow = 27

≤



capacity of cut = 30

Certificate of optimality

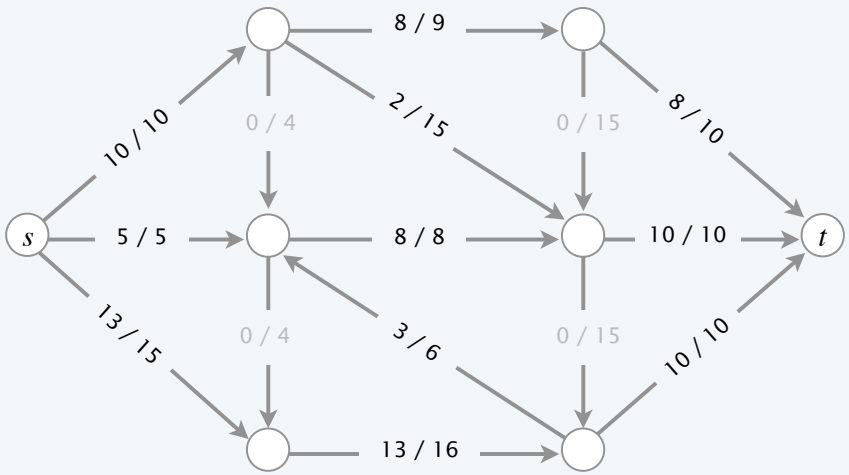
Corollary. Let f be a flow and let (A, B) be any cut.
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

weak duality

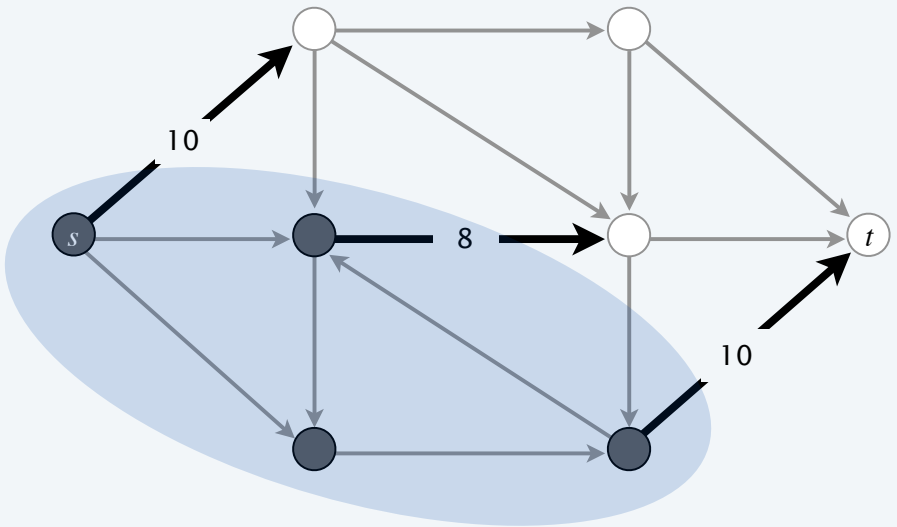
- For any flow f': $val(f') \leq cap(A, B) = val(f)$.
- For any cut (A', B'): $cap(A', B') \geq val(f) = cap(A, B)$. ▪

weak duality



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.

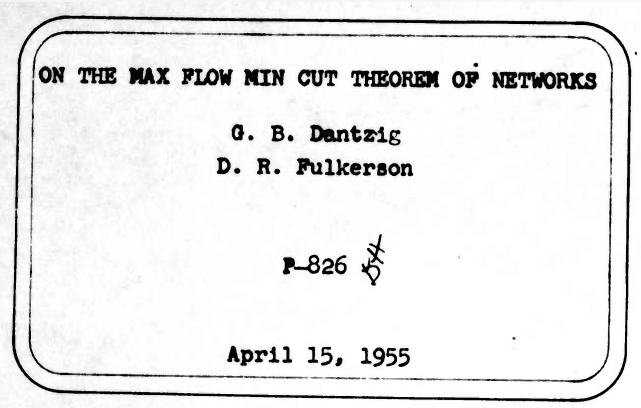
strong duality

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 cut-sets while (d, g, h, i) is not. When a simple cut-set is

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 $\Rightarrow$ 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 $\Rightarrow$ iii] We prove contrapositive: $\neg$ iii $\Rightarrow$ $\neg$ 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 $\Rightarrow$ i]

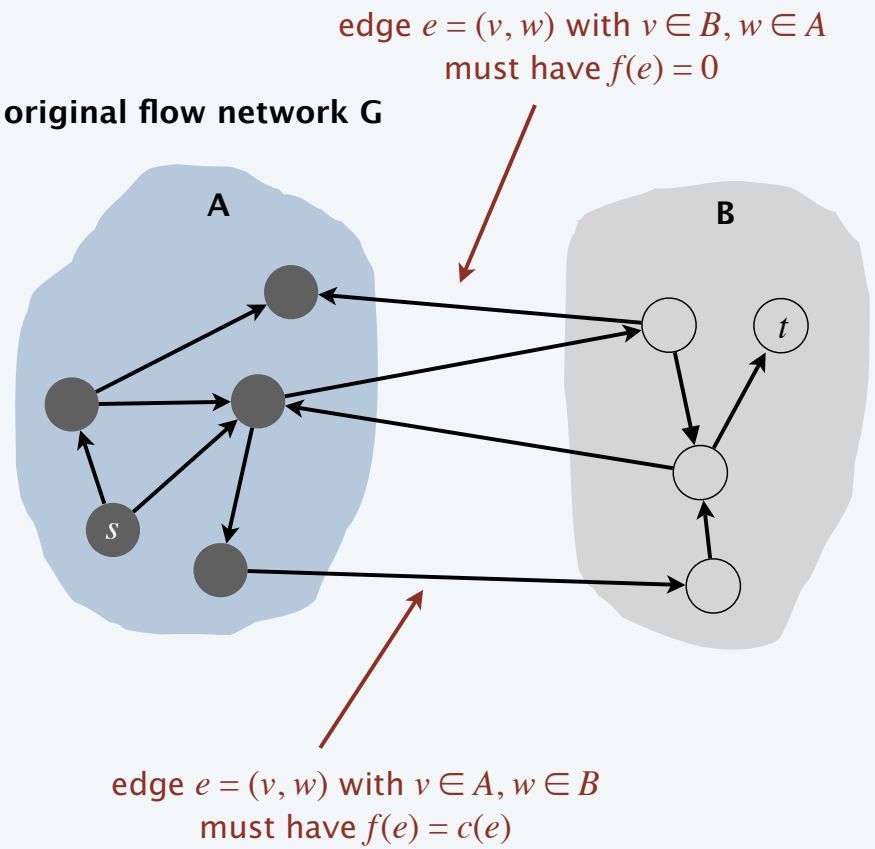
- Let f be a flow with no augmenting paths.
- Let A be set of nodes reachable from s in residual network G_f.
- By definition of A: $s \in A$.
- By definition of flow f: $t \notin A$.

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

flow value lemma

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

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



Runtime of the algorithm

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

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^*.

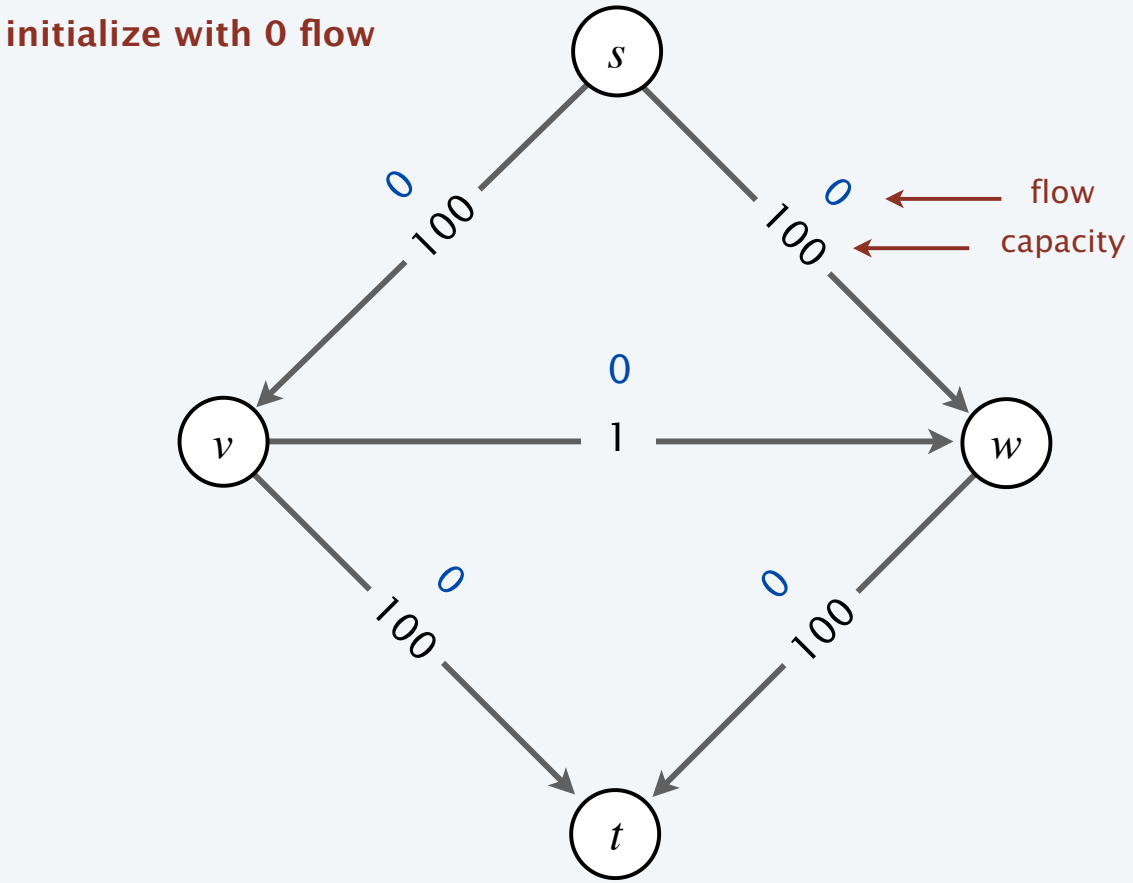
Pf. Since Ford–Fulkerson terminates, theorem follows from integrality invariant (and augmenting path theorem). ▪

How to improve Ford-Fulkerson algorithm

Why need to improve the algorithm?

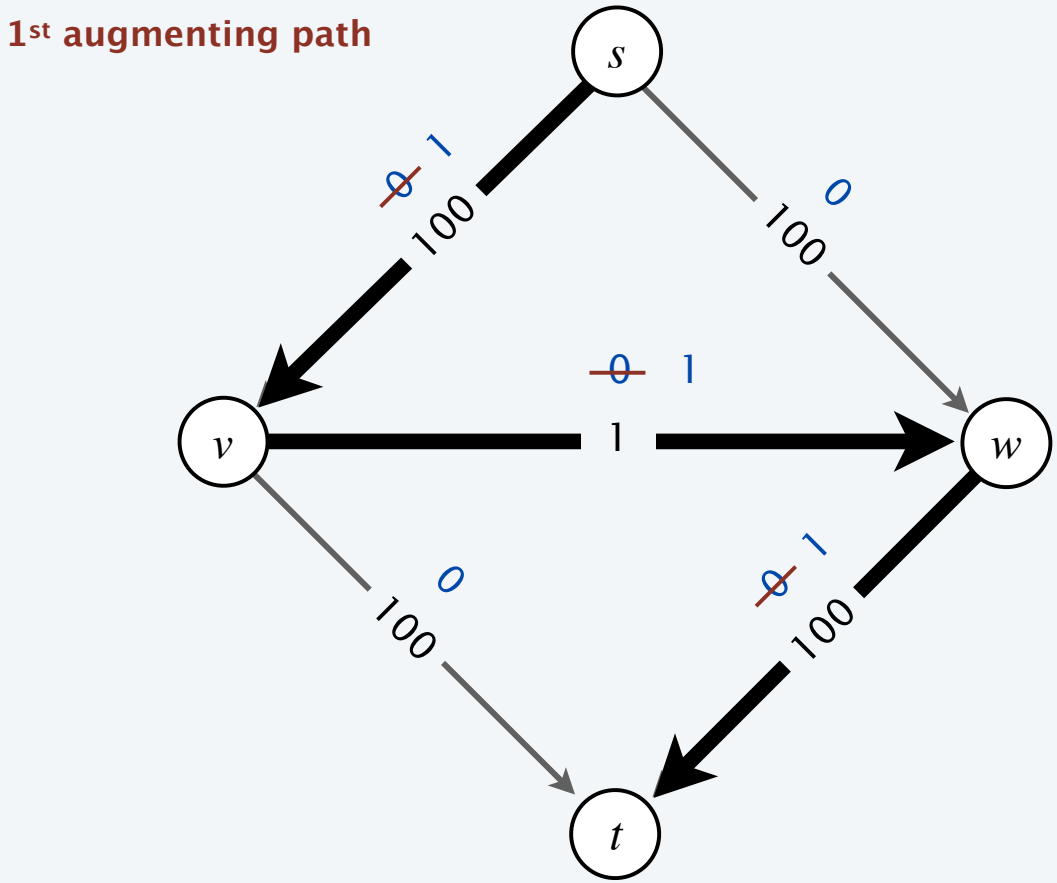
Ford–Fulkerson algorithm: exponential-time example

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



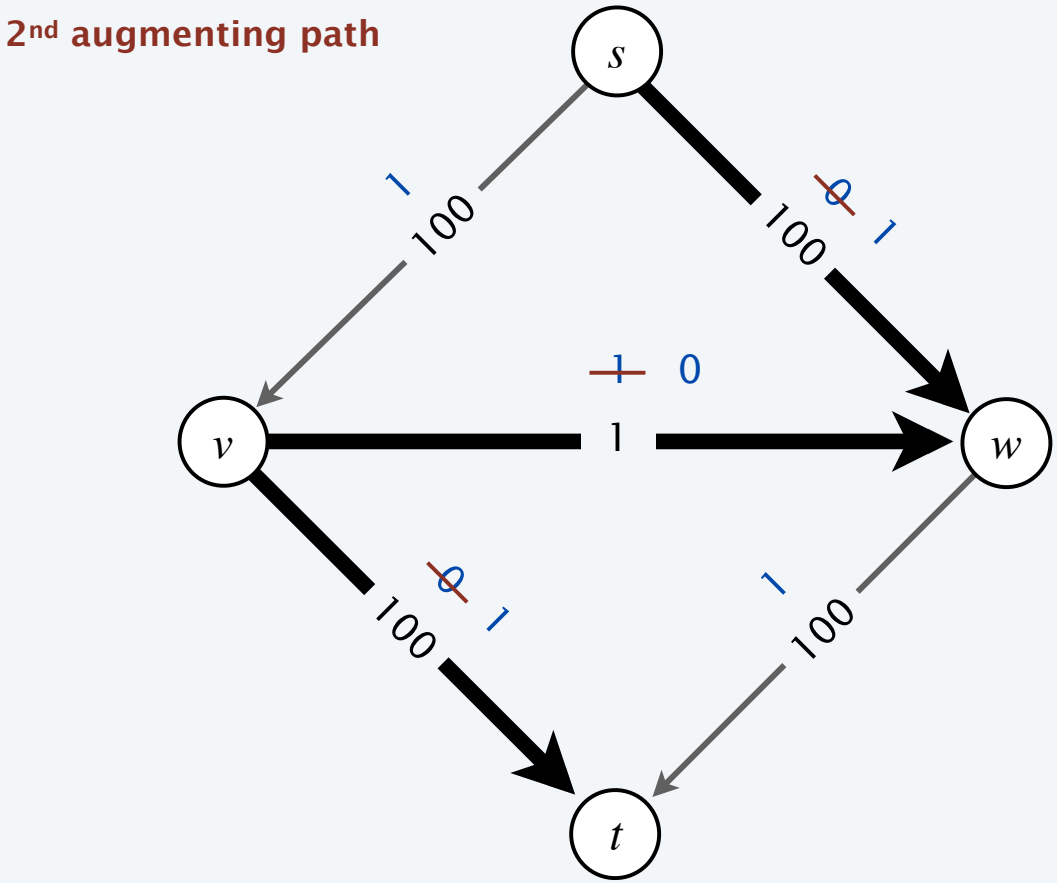
Ford–Fulkerson algorithm: exponential-time example

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



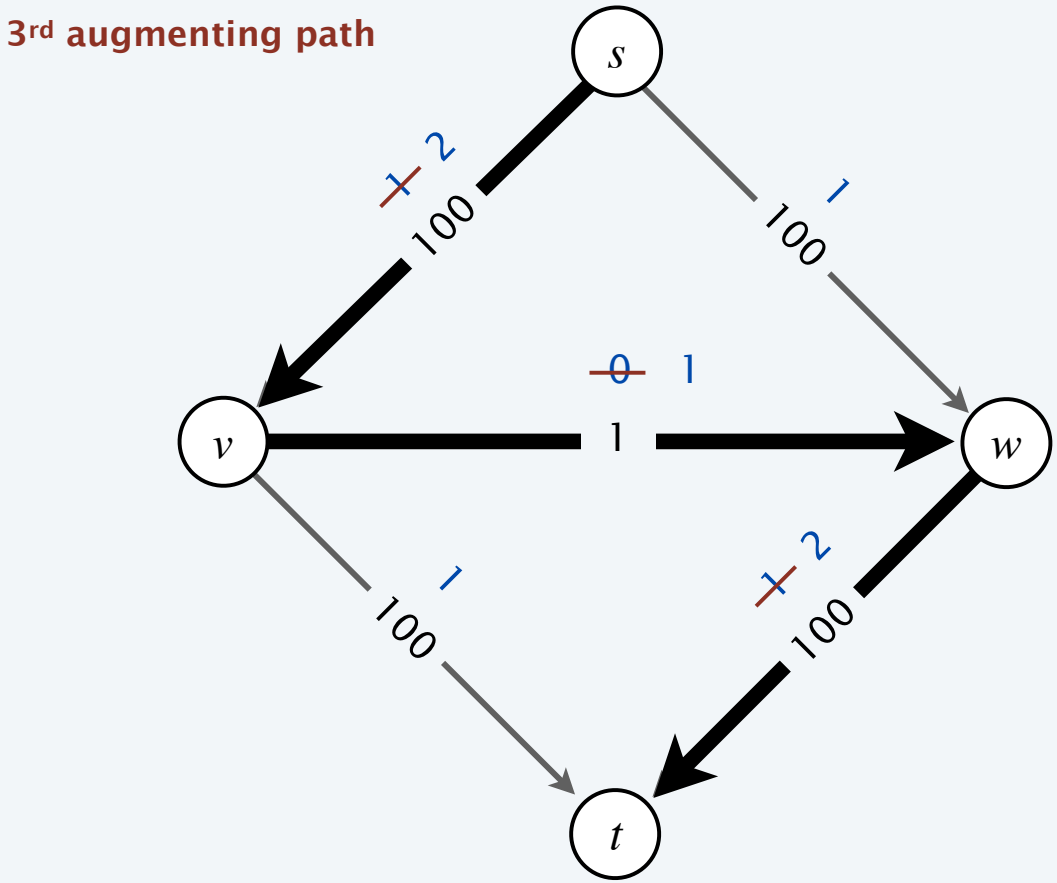
Ford–Fulkerson algorithm: exponential-time example

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



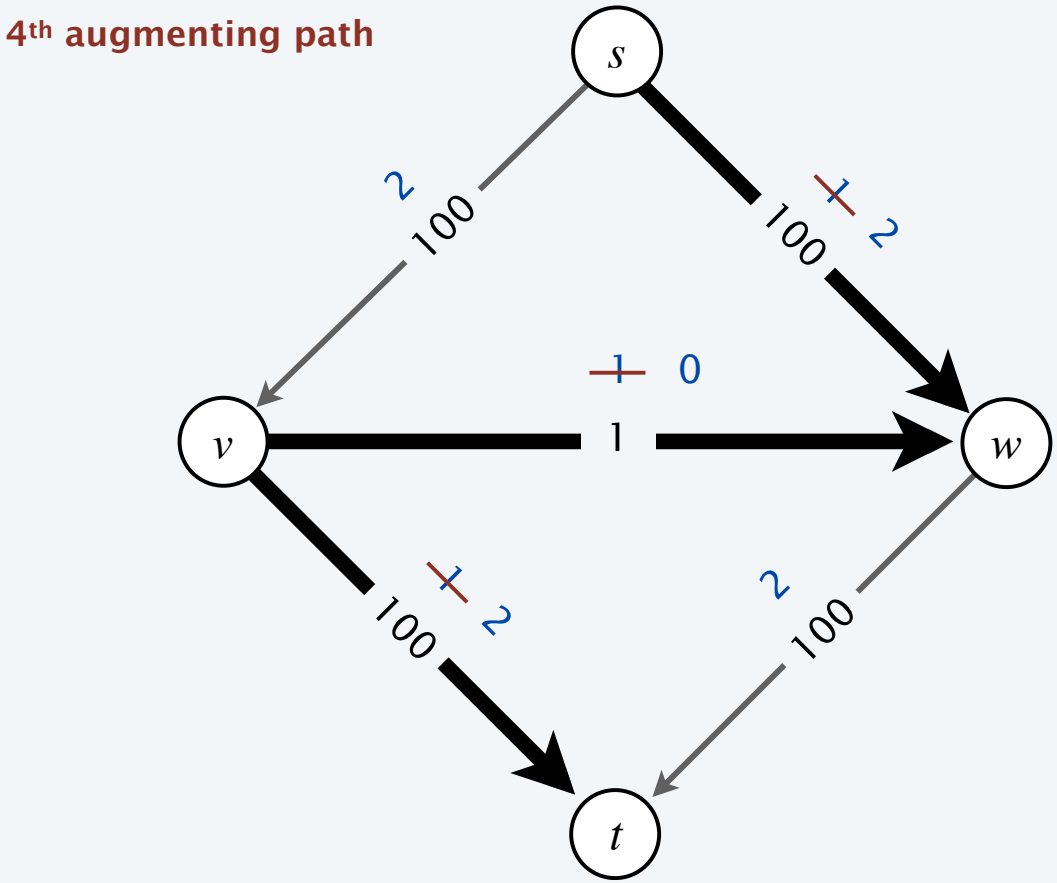
Ford–Fulkerson algorithm: exponential-time example

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.

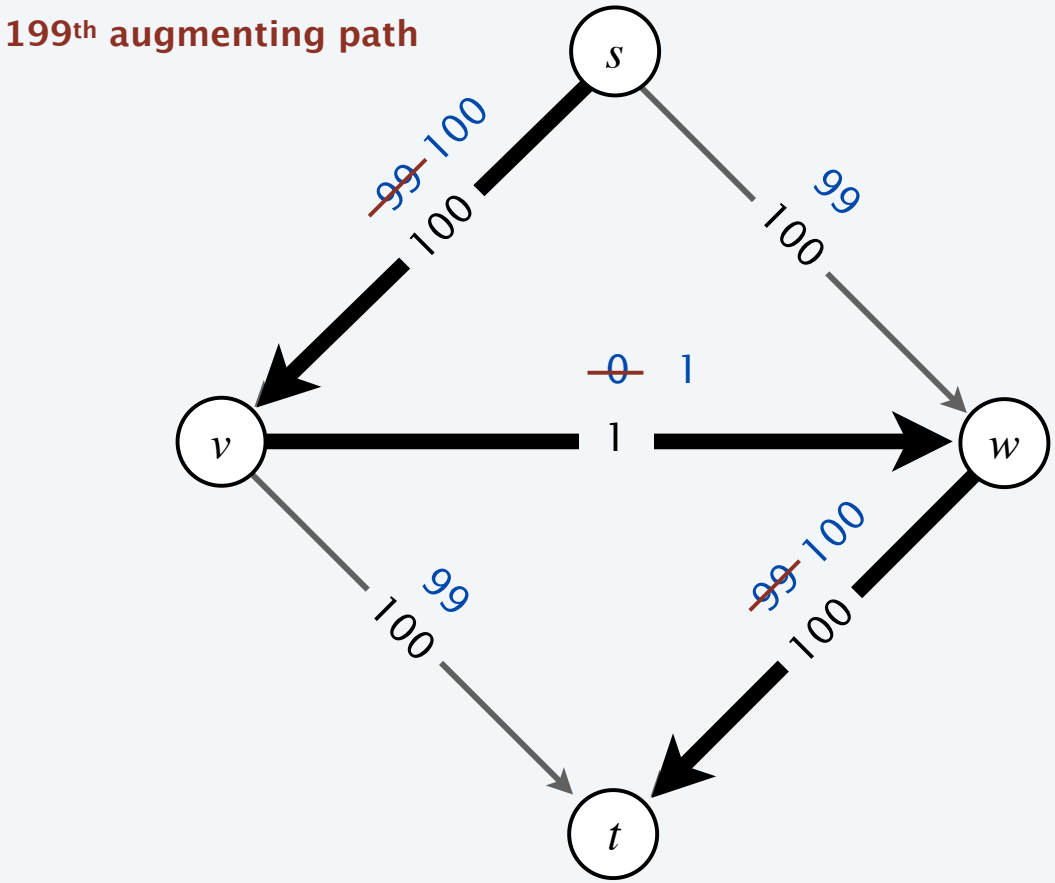


Ford–Fulkerson algorithm: exponential-time example

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

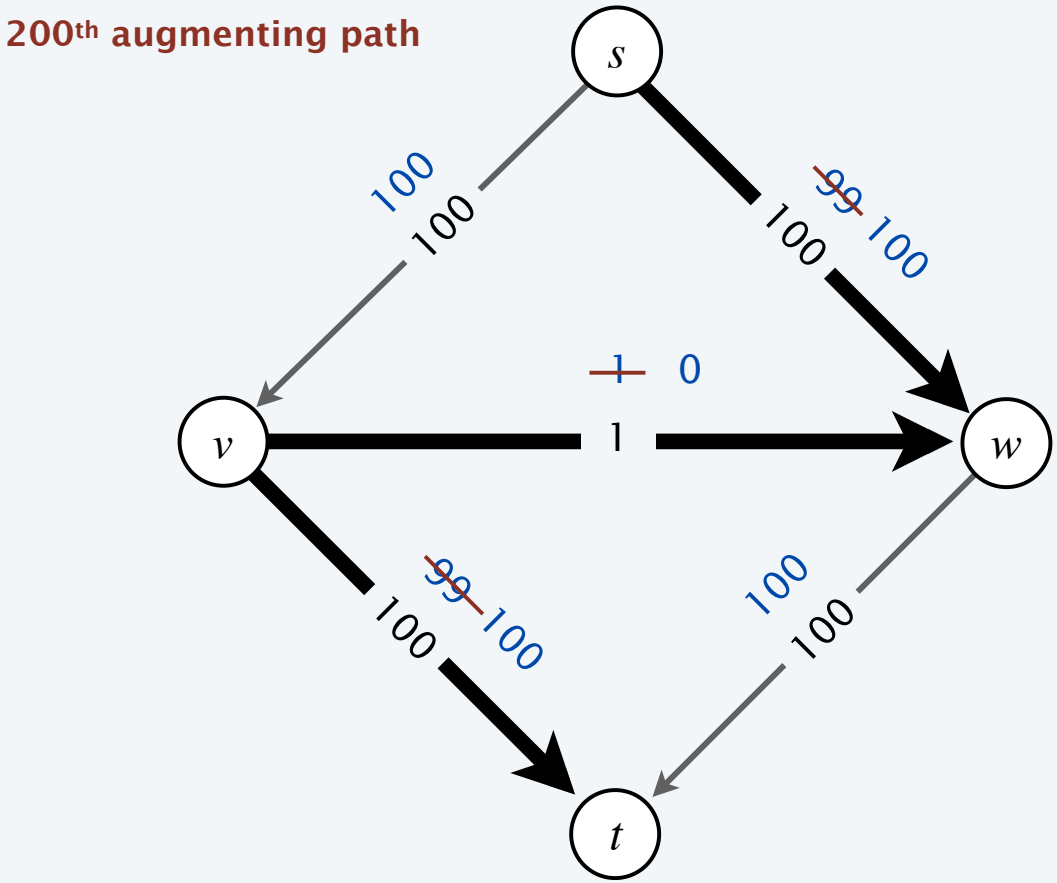
Ford–Fulkerson algorithm: exponential-time example

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



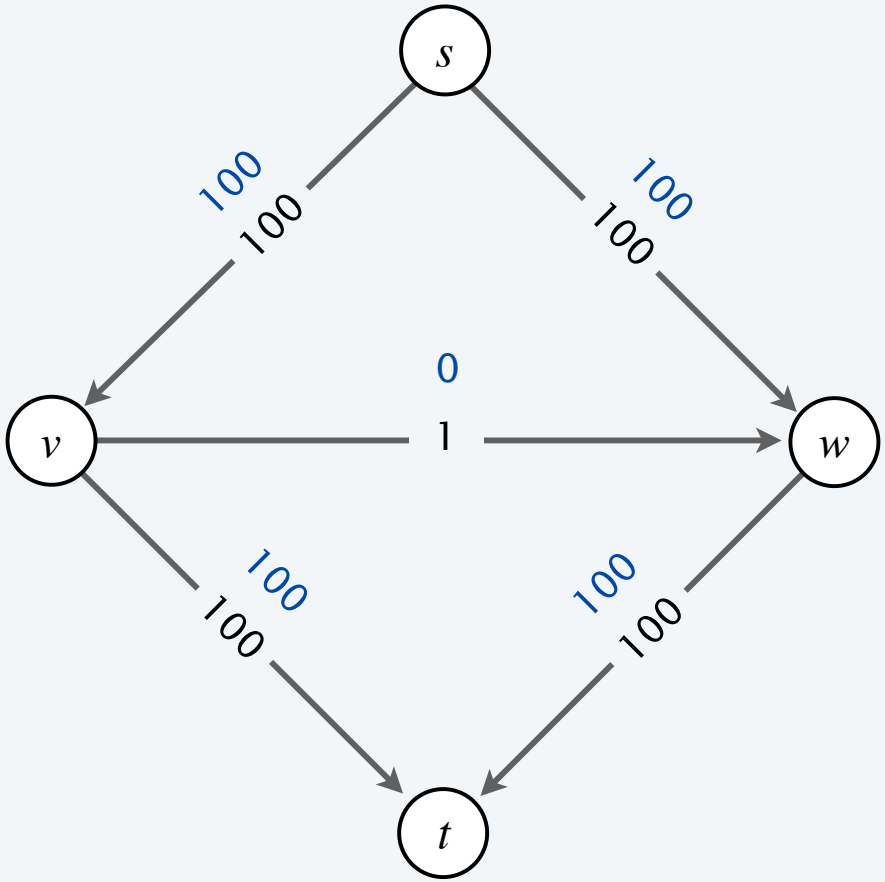
Ford–Fulkerson algorithm: exponential-time example

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



Ford–Fulkerson algorithm: exponential-time example

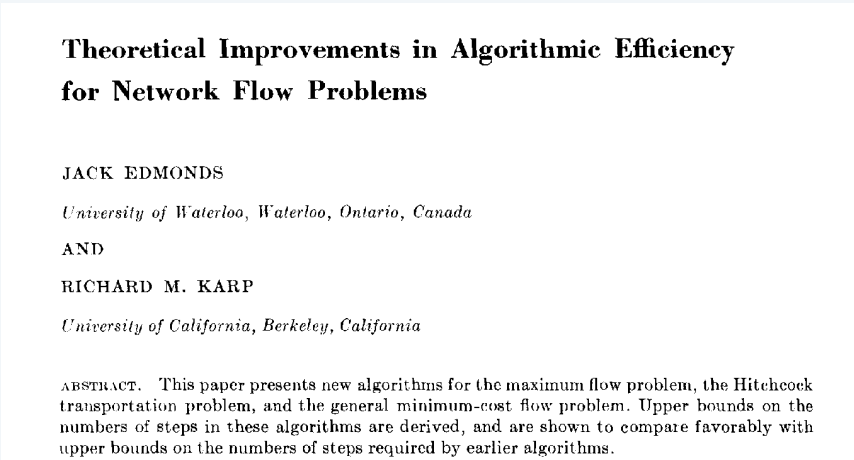
Bad news. Number of augmenting paths can be exponential in input size.



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

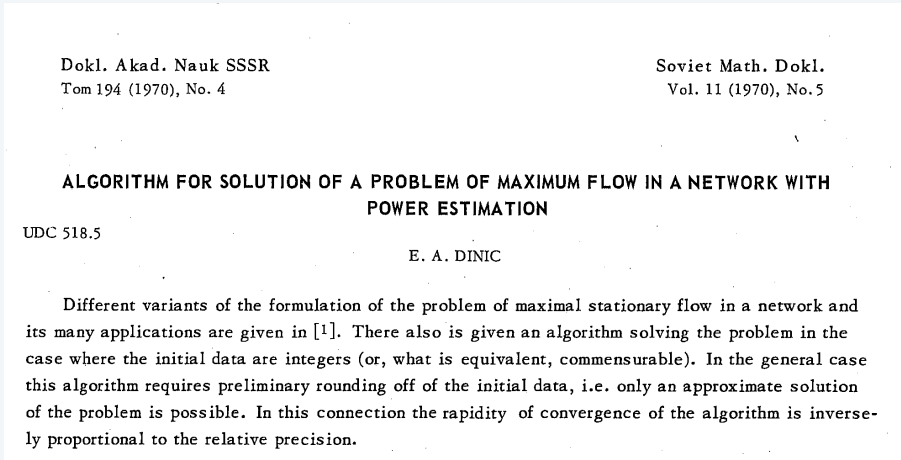
AND

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

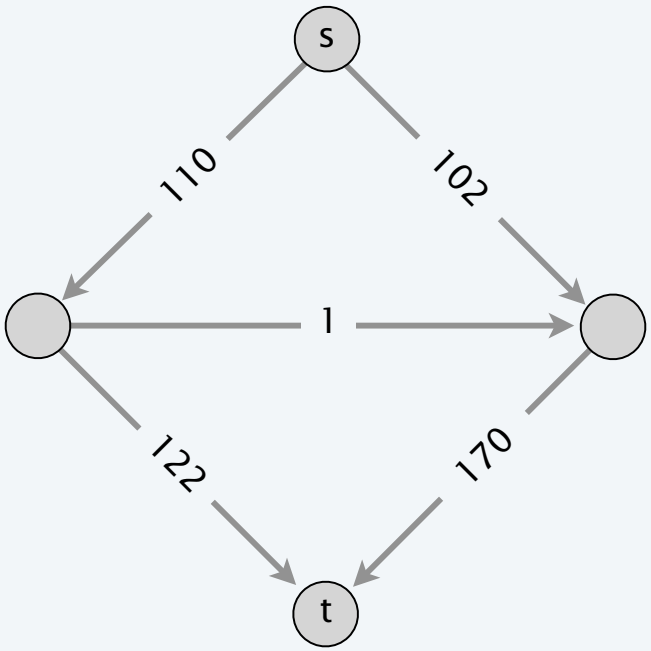
Capacity scaling algorithm

Capacity-scaling algorithm

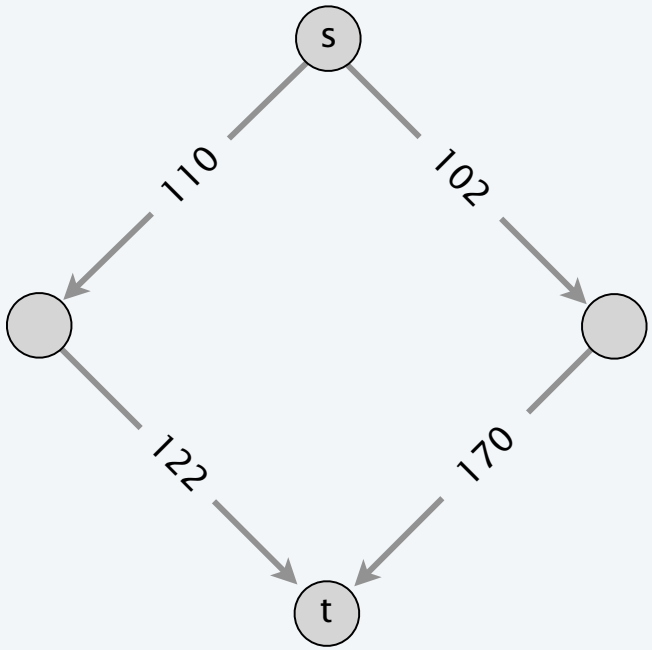
Overview. Choosing augmenting paths with "large" bottleneck capacity.

- 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 $\geq \Delta$.

though not necessarily largest



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 Δ is a power of 2.

Pf. Initially a power of 2; each phase divides Δ 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 $C/2 < \Delta \leq C$; 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 max-flow value $\leq val(f) + m\,\Delta$.

Pf. Next slide.

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

Pf.

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

or equivalently, at the end of a 2Δ-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. ▪

Shortest argumenting paths

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 $s \rightarrow t$ 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: overview of analysis

Lemma 1. The length of a shortest augmenting path never decreases.

Pf. Ahead.

number of edges

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Pf. Ahead.

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

Pf.

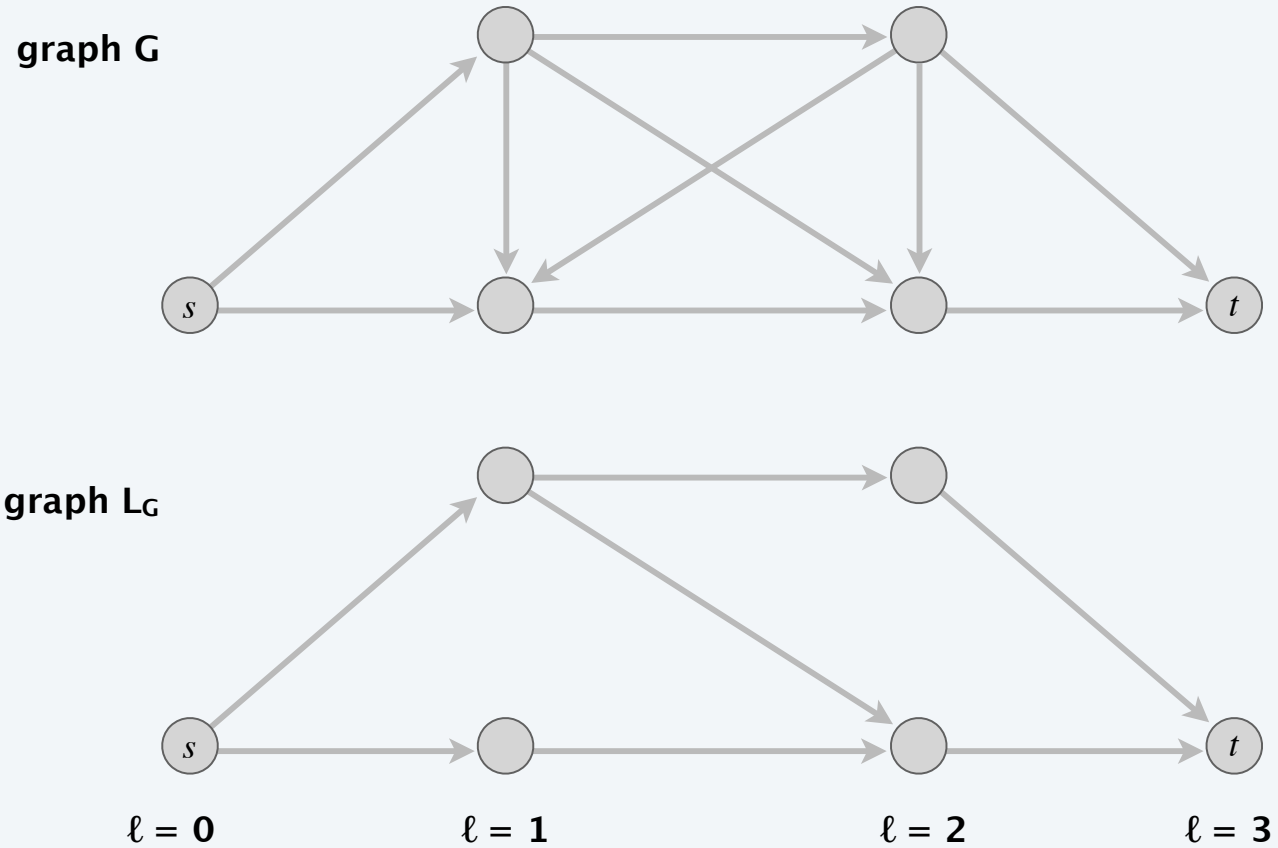
- $O(m)$ time to find a shortest augmenting path via BFS.
- There are $\leq m n$ augmentations.
  - at most m augmenting paths of length k ← Lemma 1 + Lemma 2
  - at most $n-1$ different lengths ▪

augmenting paths are simple paths

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



Network flow: quiz 5

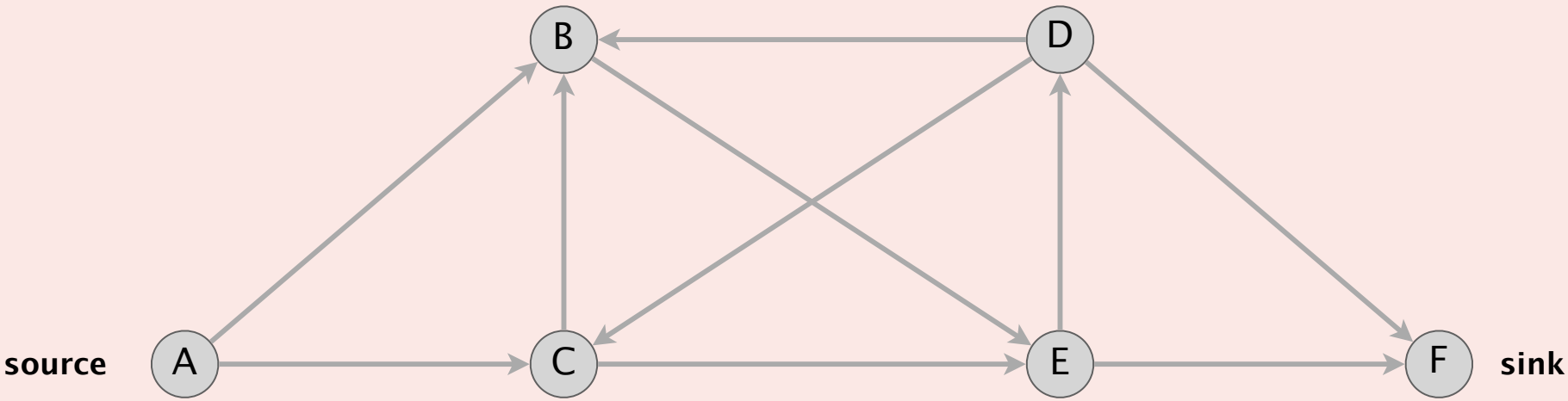
Which edges are in the level graph of the following digraph?

A. D→F.

B. E→F.

C. Both A and B.

D. Neither A nor B.

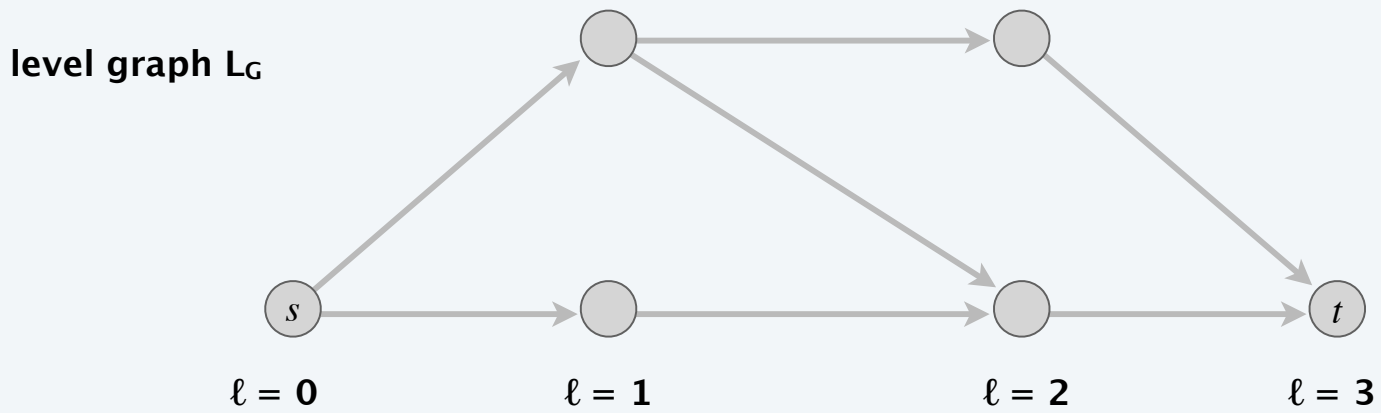


Shortest augmenting path: analysis

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

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

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

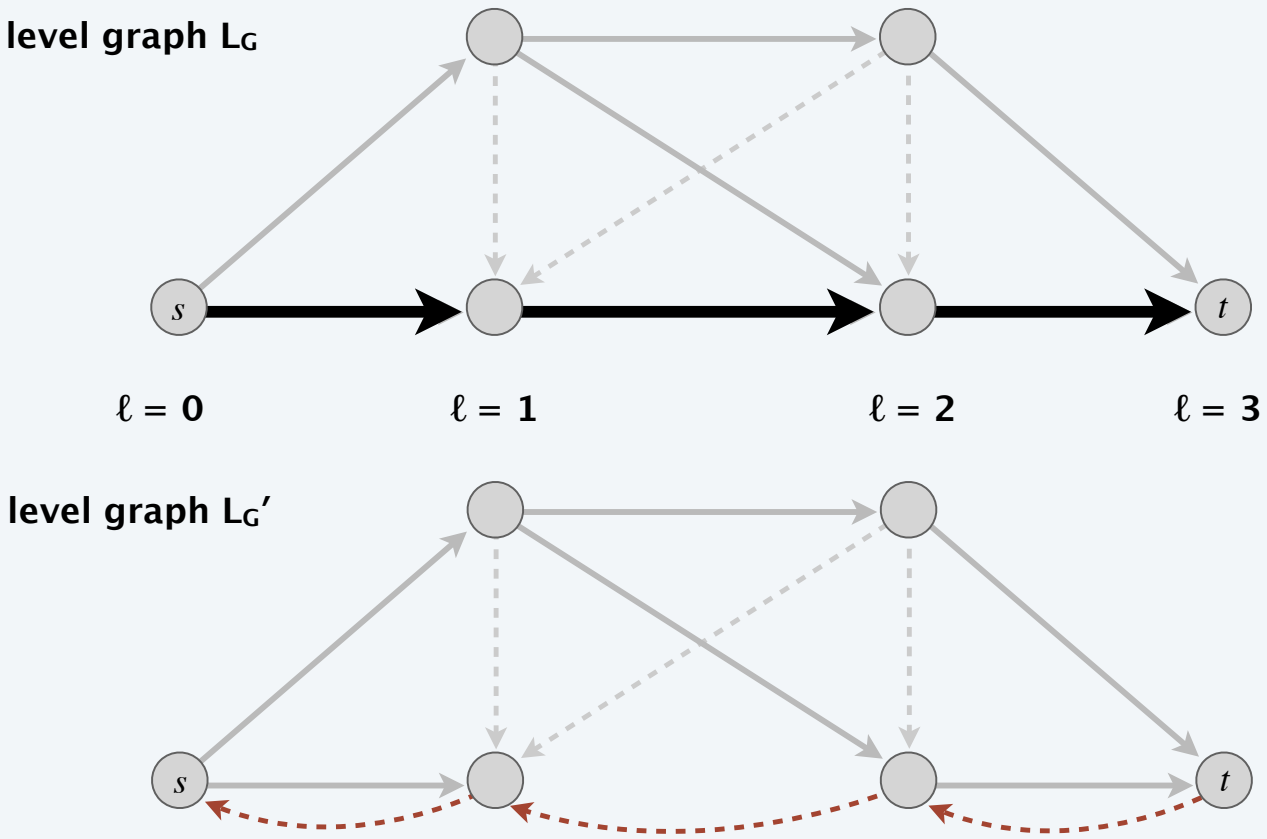


Shortest augmenting path: analysis

Lemma 1. The length of a shortest augmenting path never decreases.

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

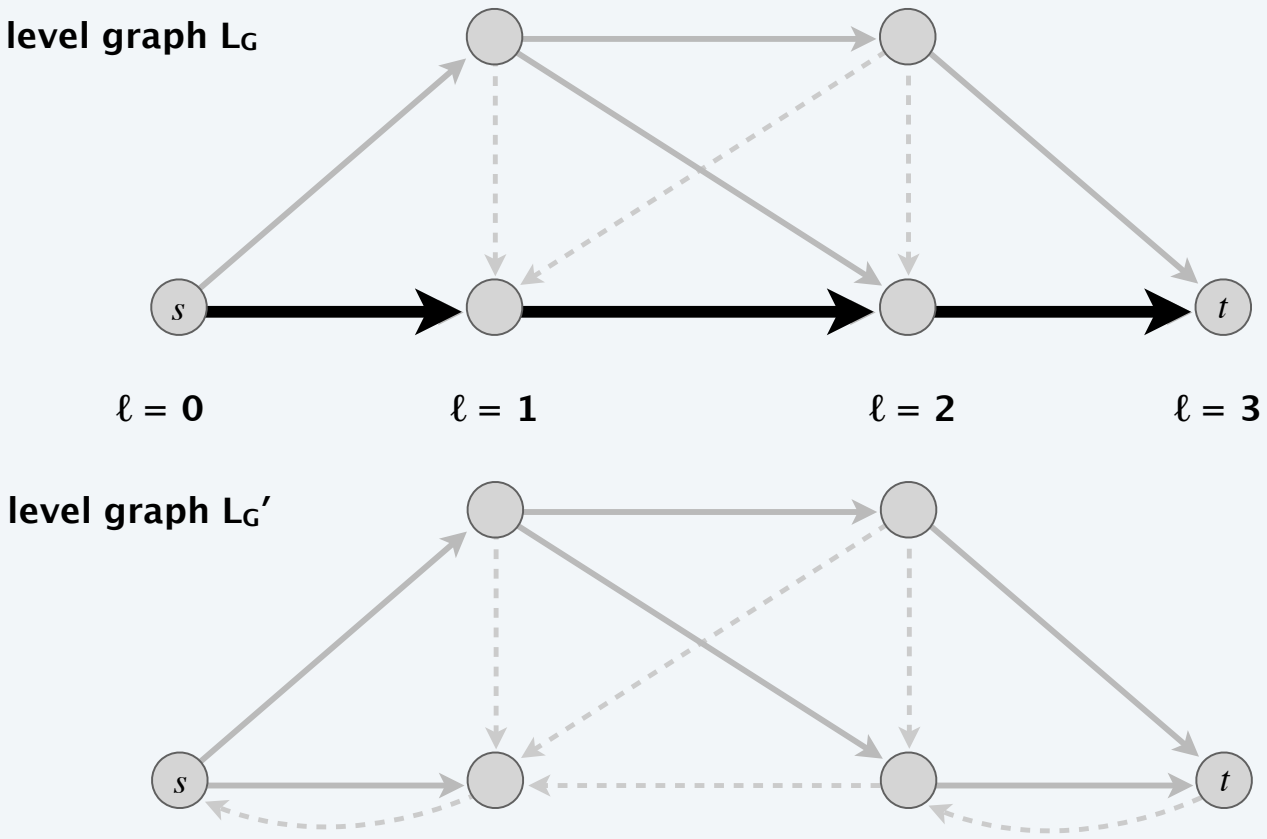
  (any $s \rightsquigarrow t$ path that uses a back edge is longer than previous length) ▪



Shortest augmenting path: analysis

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

- At least one (bottleneck) edge is deleted from L_G per augmentation.
- No new edge added to L_G until shortest path length strictly increases. ▪



Shortest augmenting path: review of analysis

Lemma 1. Throughout the algorithm, the length of a shortest augmenting path never decreases.

Lemma 2. After at most m shortest-path augmentations, the length of a shortest augmenting path strictly increases.

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

Better algorithm for max flow problems

Augmenting-path algorithms: summary

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

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

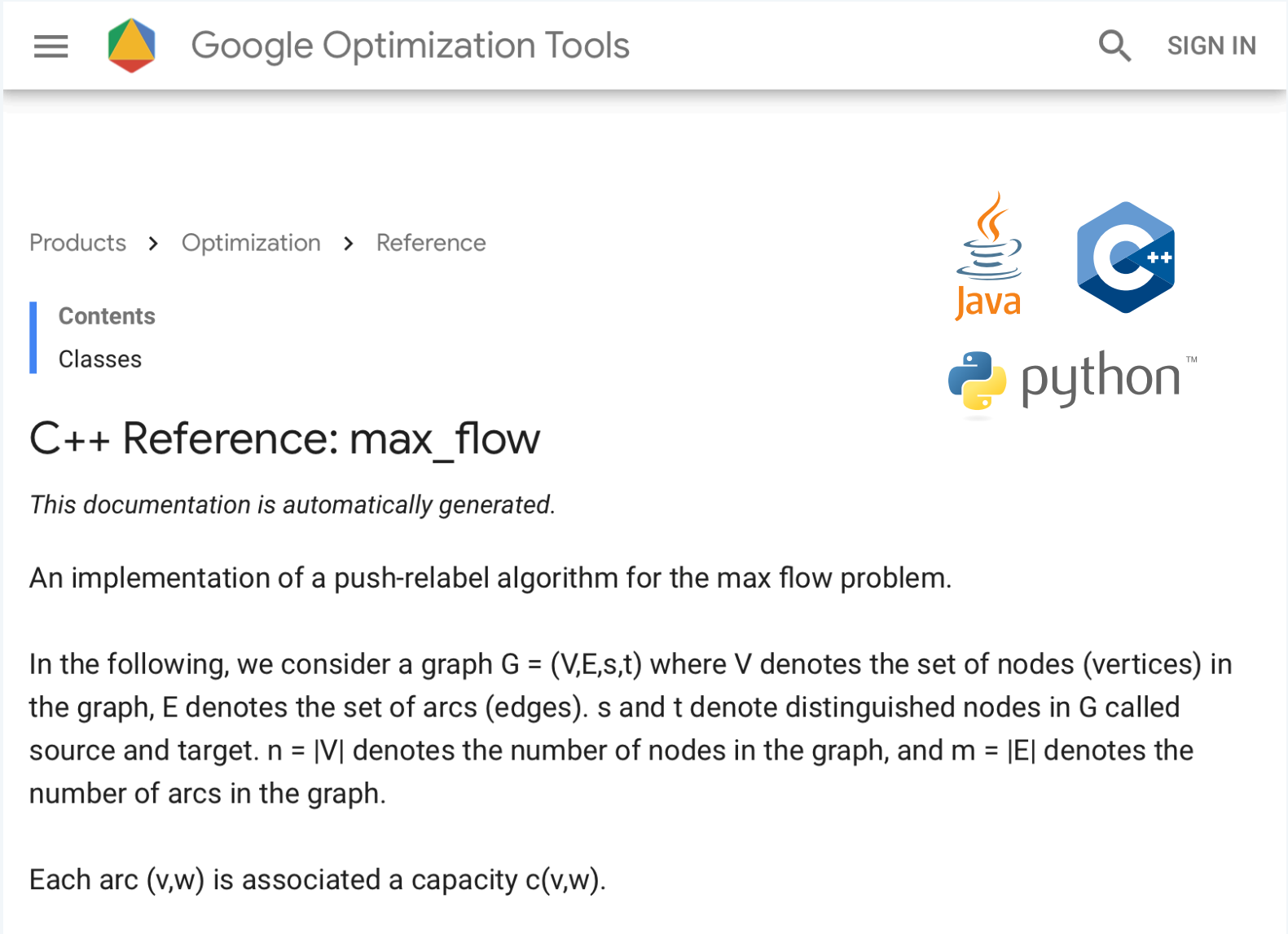
Maximum-flow algorithms: theory highlights

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**	$\tilde{O}(m\, m^{1/2} \log C)$	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

Google Optimazation Tools

Maximum-flow algorithms: Google



References


Robert Sedgewick & Kevin Wayne. Algorithms, 4th Edition. Addison-Wesley.
Slides at `https://algs4.cs.princeton.edu/lectures/`


Kleinberg, J., & Tardos, E. (2006). The algorithm design. Pearson, Addison-Wesley.
Slides at
`https://www.cs.princeton.edu/~wayne/kleinberg-tardos/`


Martin Skutella. An Introduction to Network Flows over time.