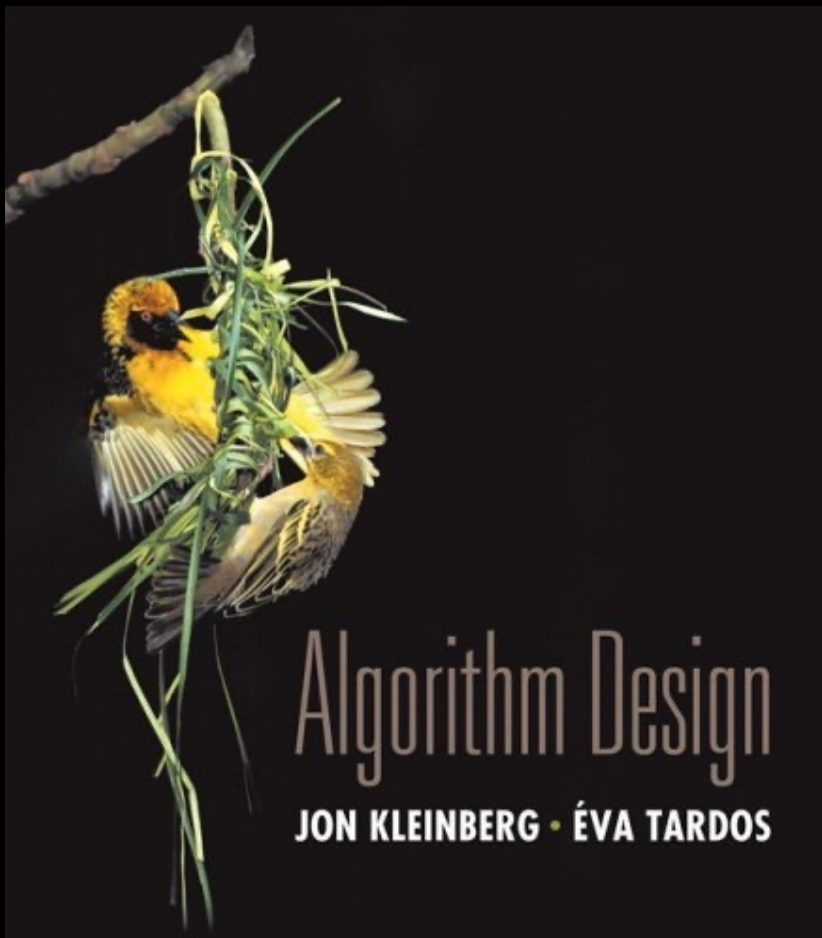
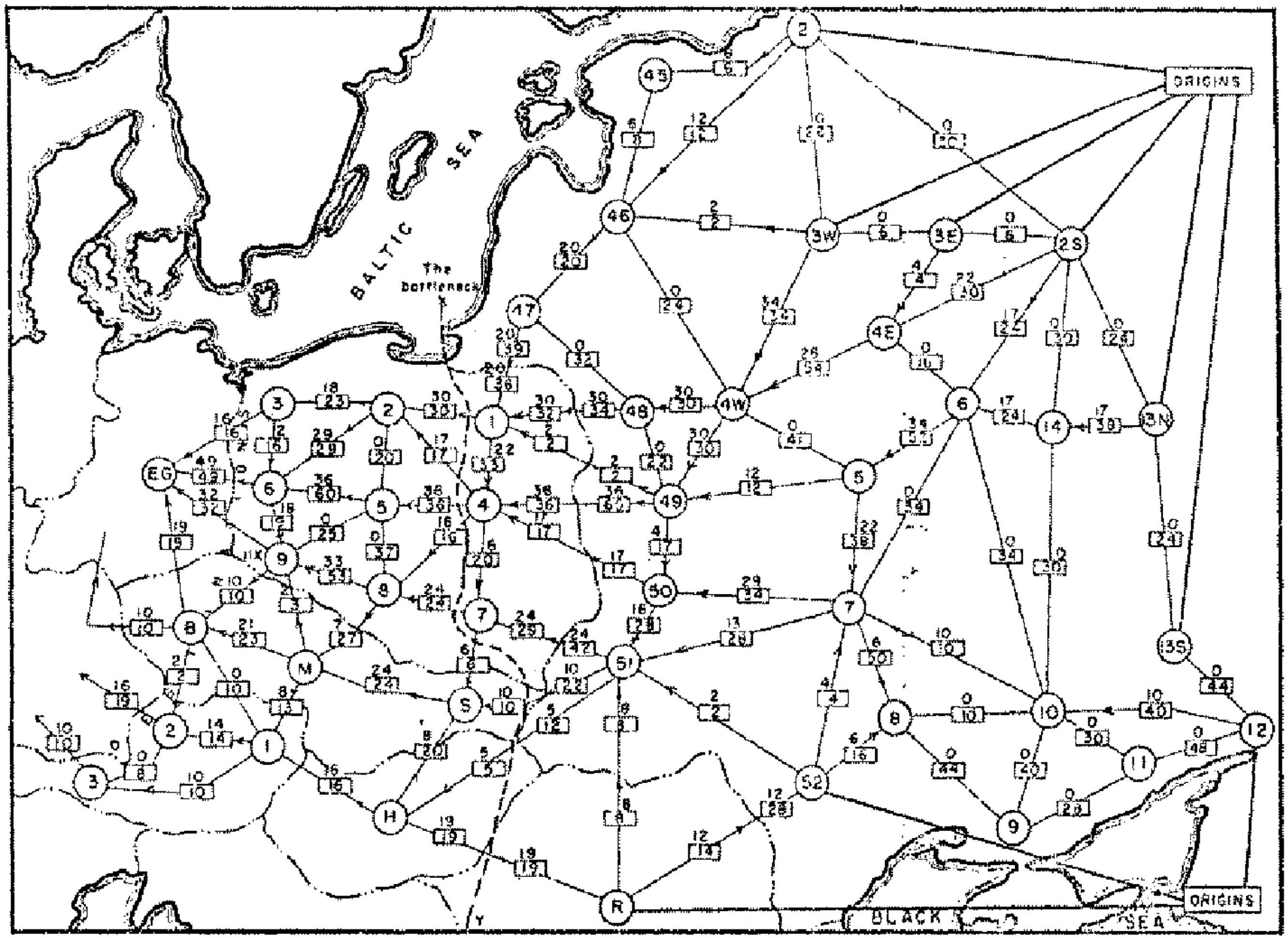


Chapter 7

Network Flow

Annotated and modified by Yang Xu (徐炀)
Contact: xuyang@sustech.edu.cn

Soviet Rail Network, 1955



Reference: *On the history of the transportation and maximum flow problems*.
Alexander Schrijver in Math Programming, 91: 3, 2002.

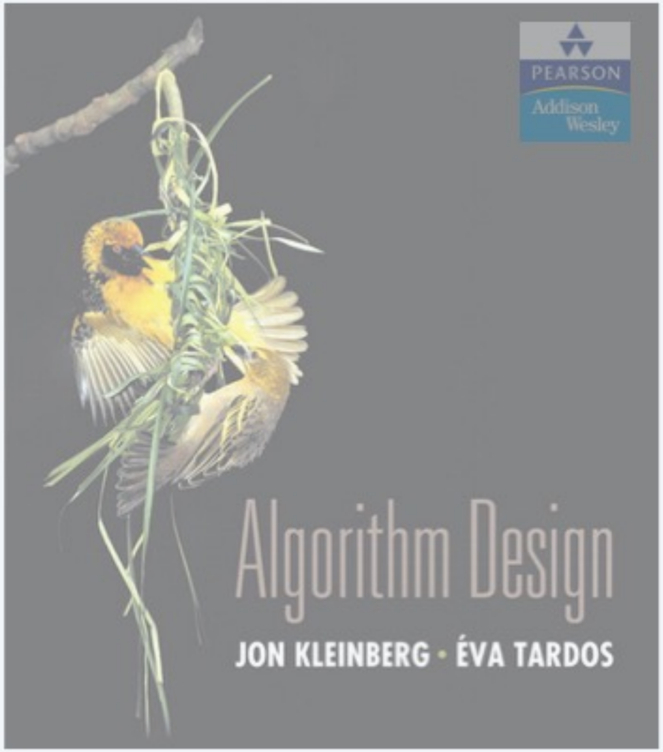
Maximum Flow and Minimum Cut

Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

Nontrivial applications / reductions.

- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more …



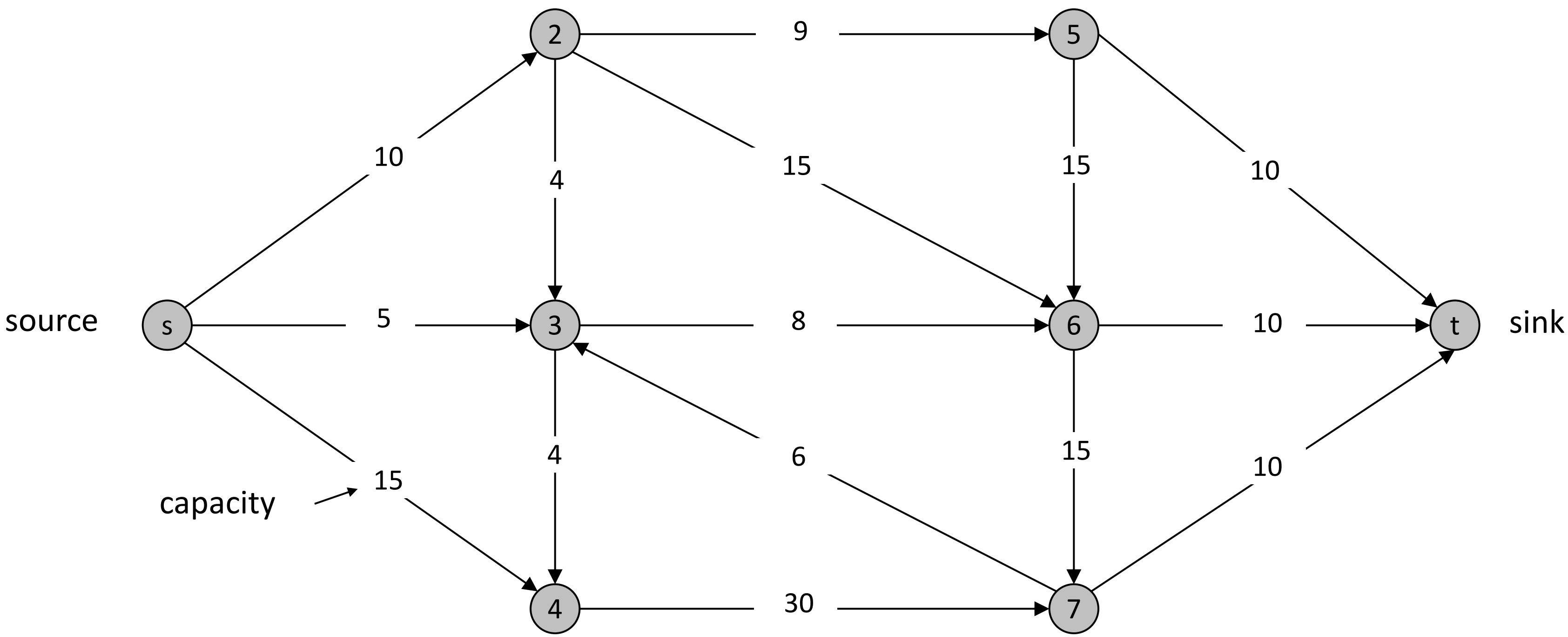
SECTION 7.1

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.

- Abstraction for material flowing through the edges.
- G = (V, E) = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- c(e) = capacity of edge e.

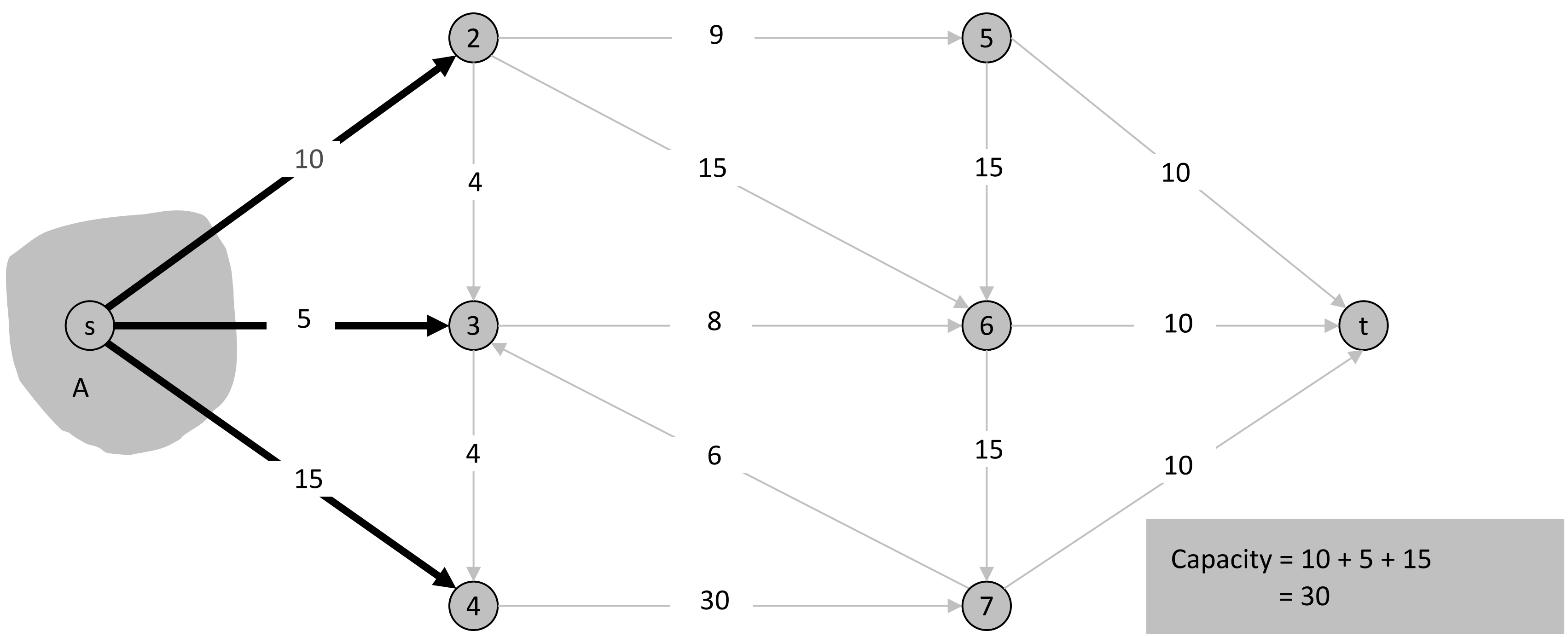


Cuts

Def. An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

Def. The capacity of a cut (A, B) is:

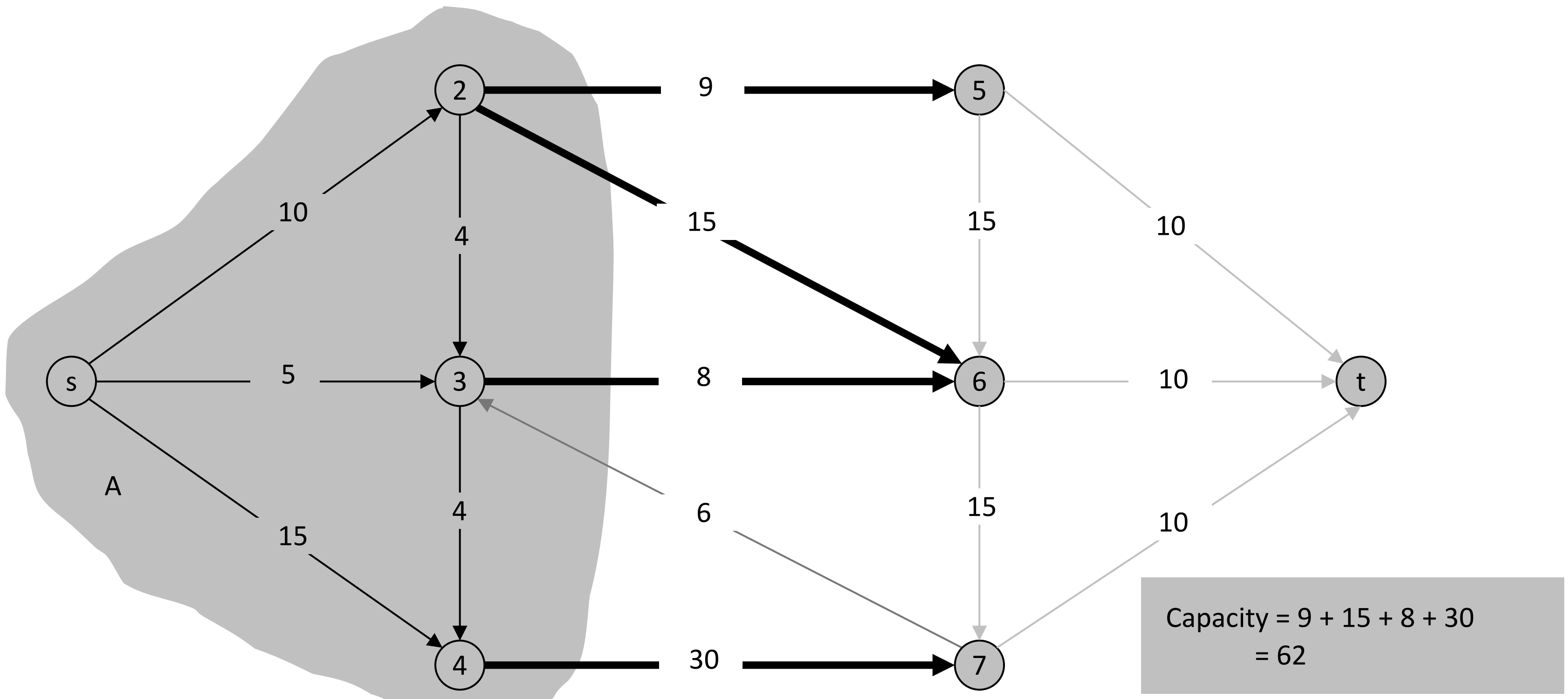
$$cap(A,B) = \sum_{e \text{ out of } A} c(e)$$



Cuts

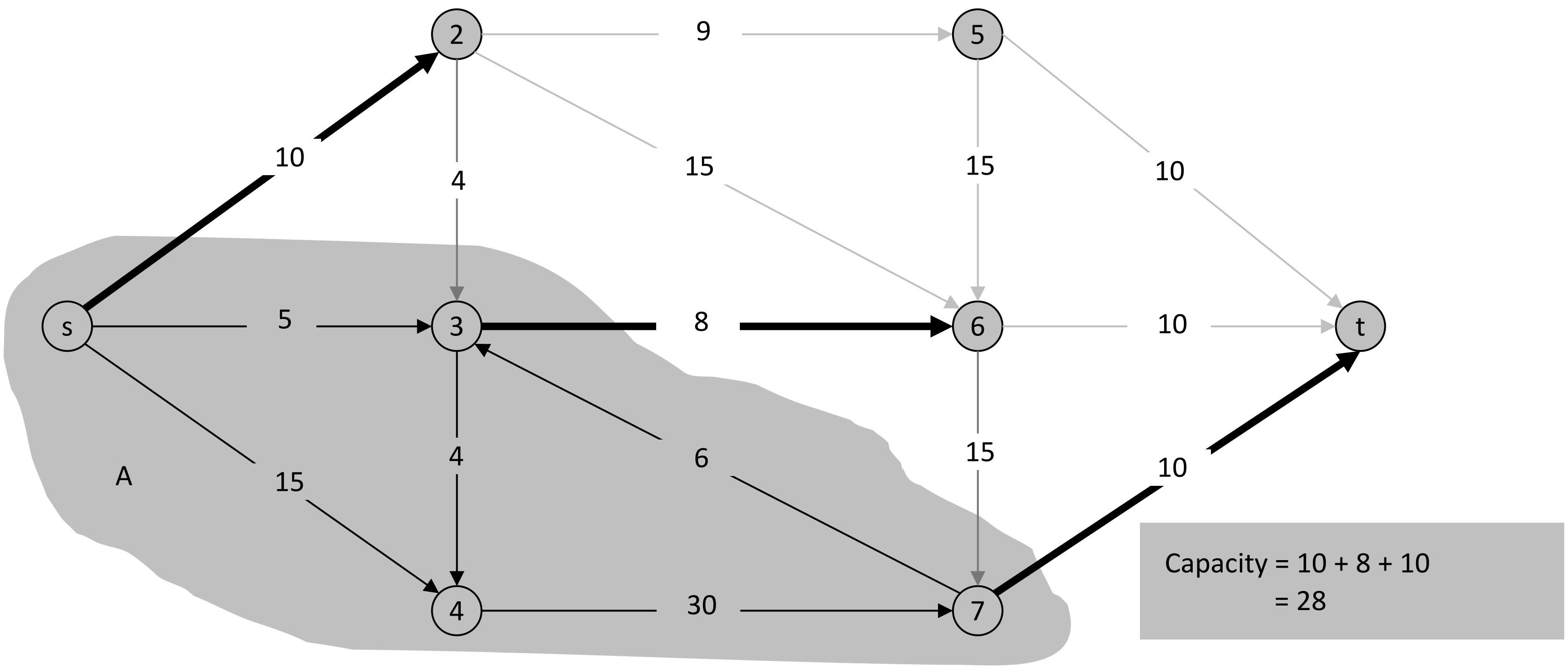
Def. An s-t cut is a partition (A, B) of V with s ∈ A and t ∈ B.

Def. The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Minimum Cut Problem

Min s-t cut problem. Find an s-t cut of minimum capacity.

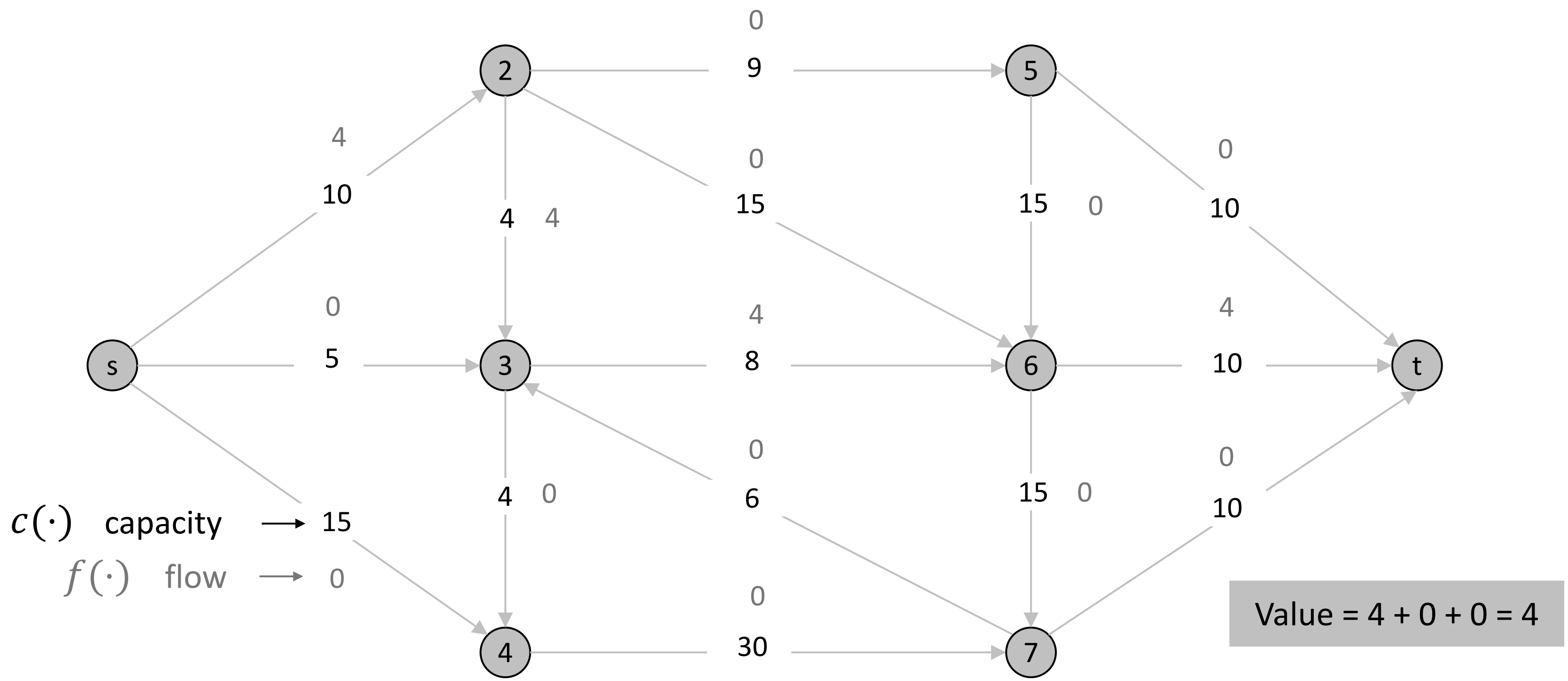


Flows

Def. An s-t flow is a function that satisfies:

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

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



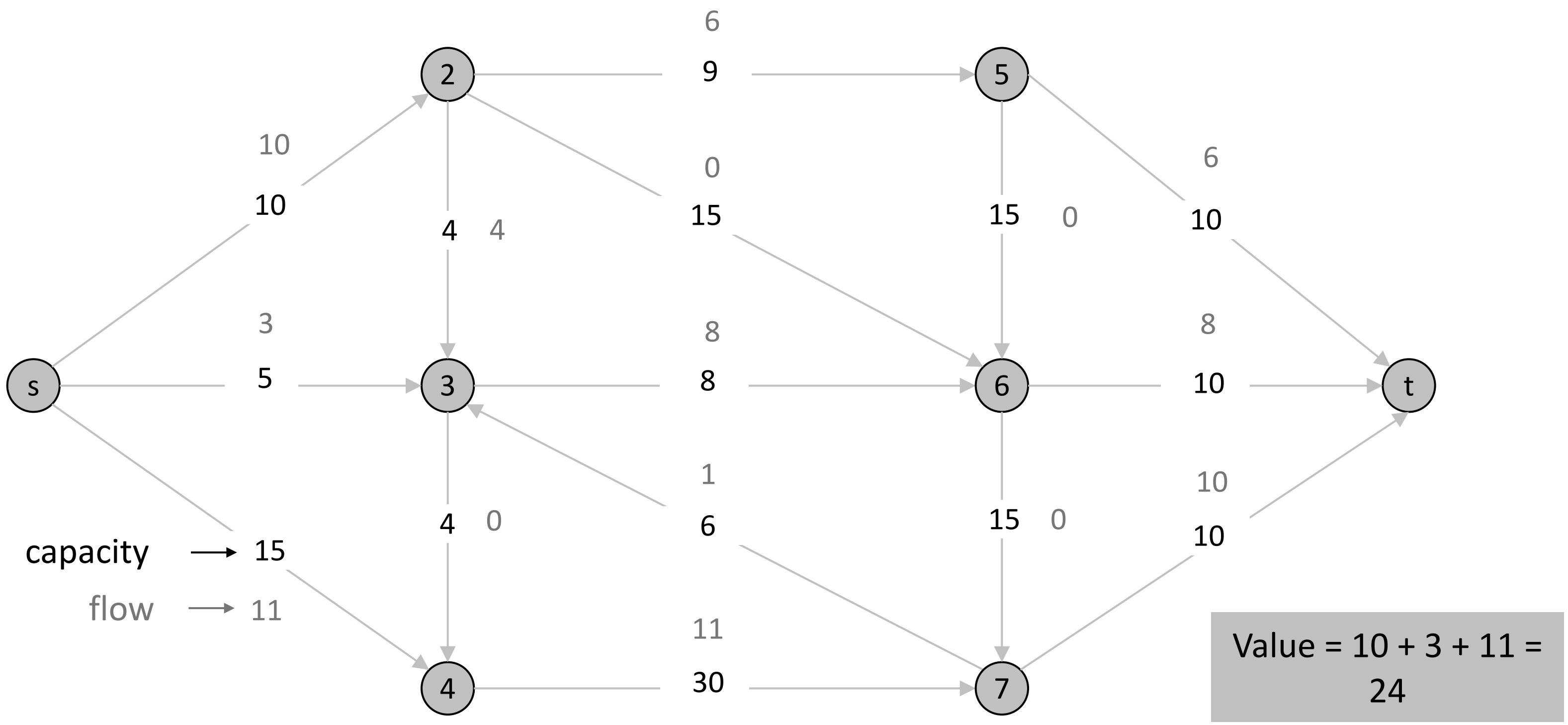
Flows

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

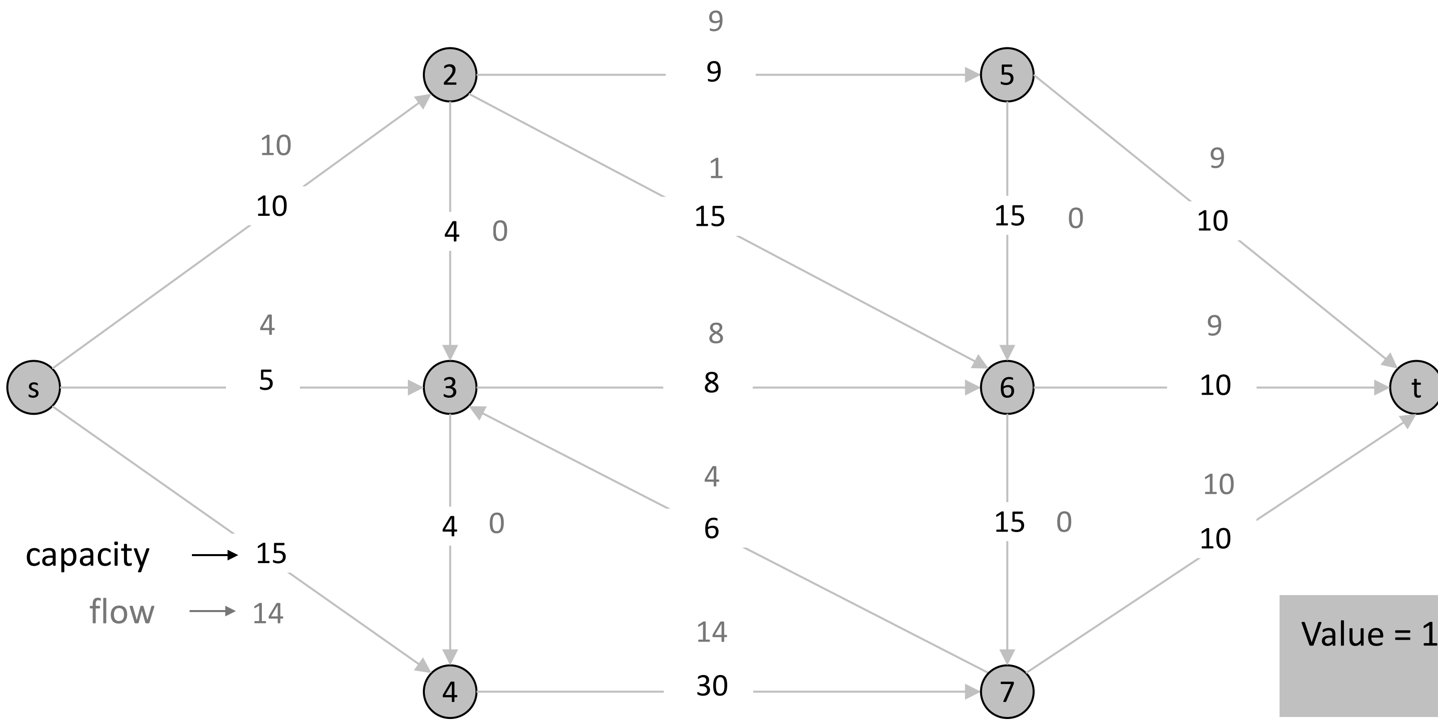
Def. The value of a flow f is:

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



Maximum Flow Problem

Max flow problem. Find s-t flow of maximum value.



Value = 10 + 4 + 14 = 28

Network flow: quiz 1

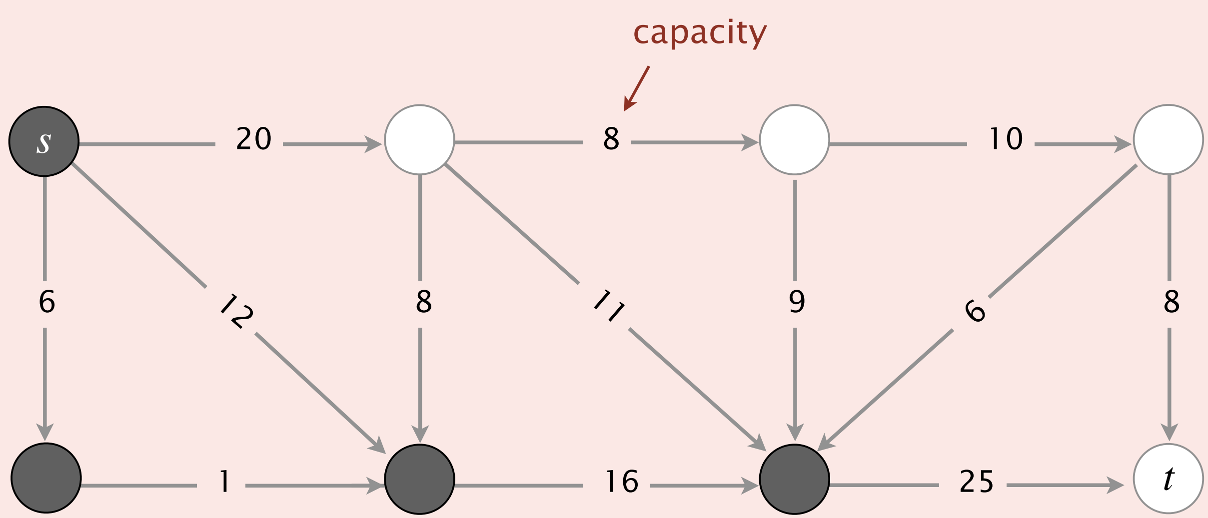
Which is the capacity of the given *st*-cut?

A. 11 (20 + 25 − 8 − 11 − 9 − 6)

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

C. 45 (20 + 25)

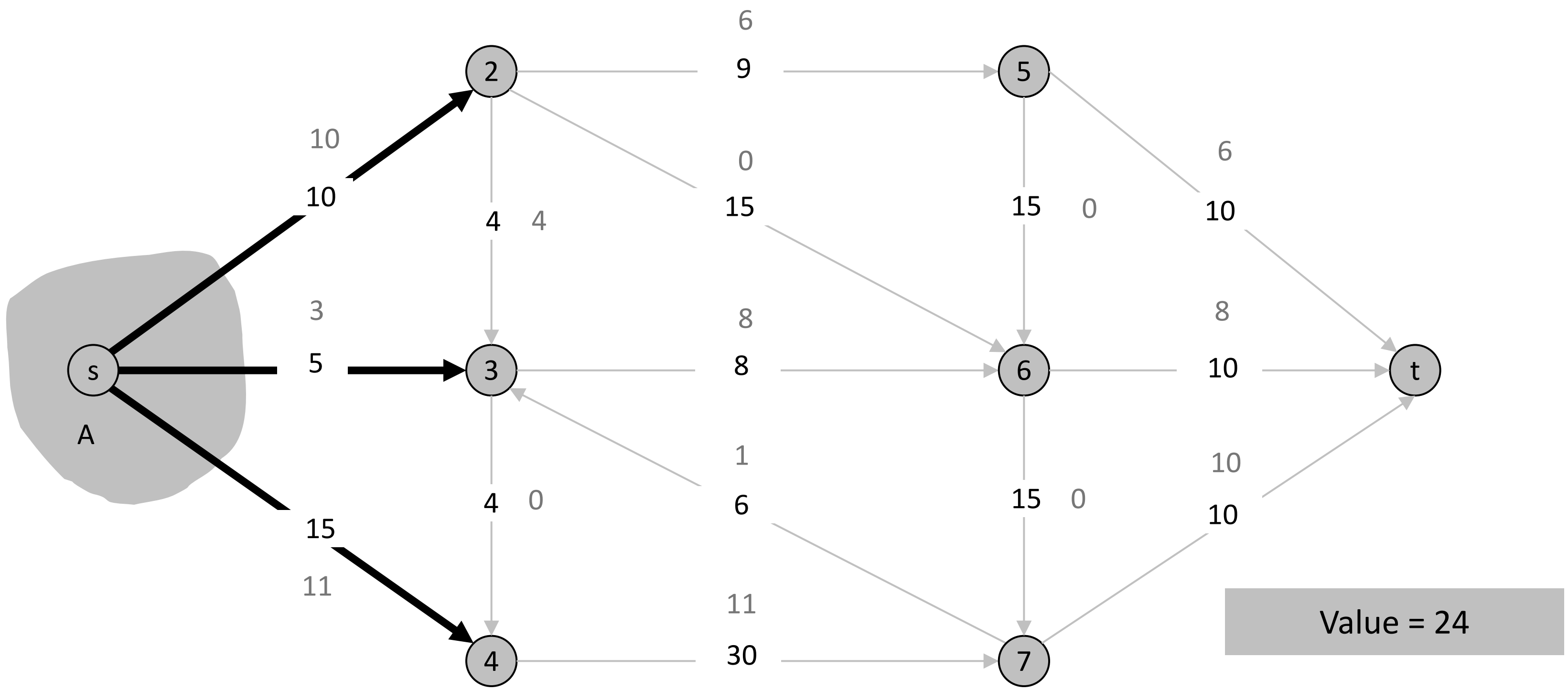
D. 79 (20 + 25 + 8 + 11 + 9 + 6)



Flows and Cuts

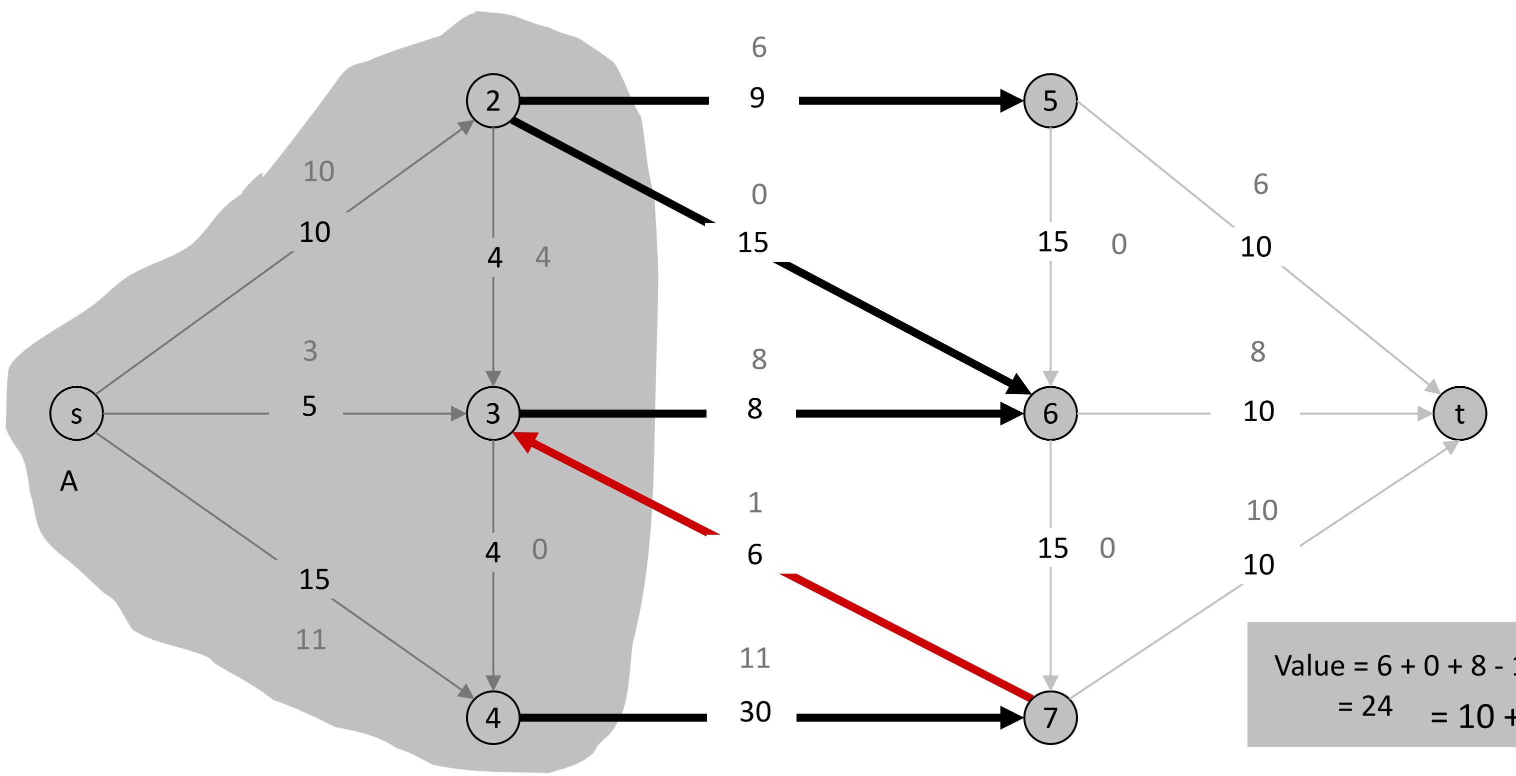
Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the **net flow** sent across the cut is equal to the amount leaving s.

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



Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

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

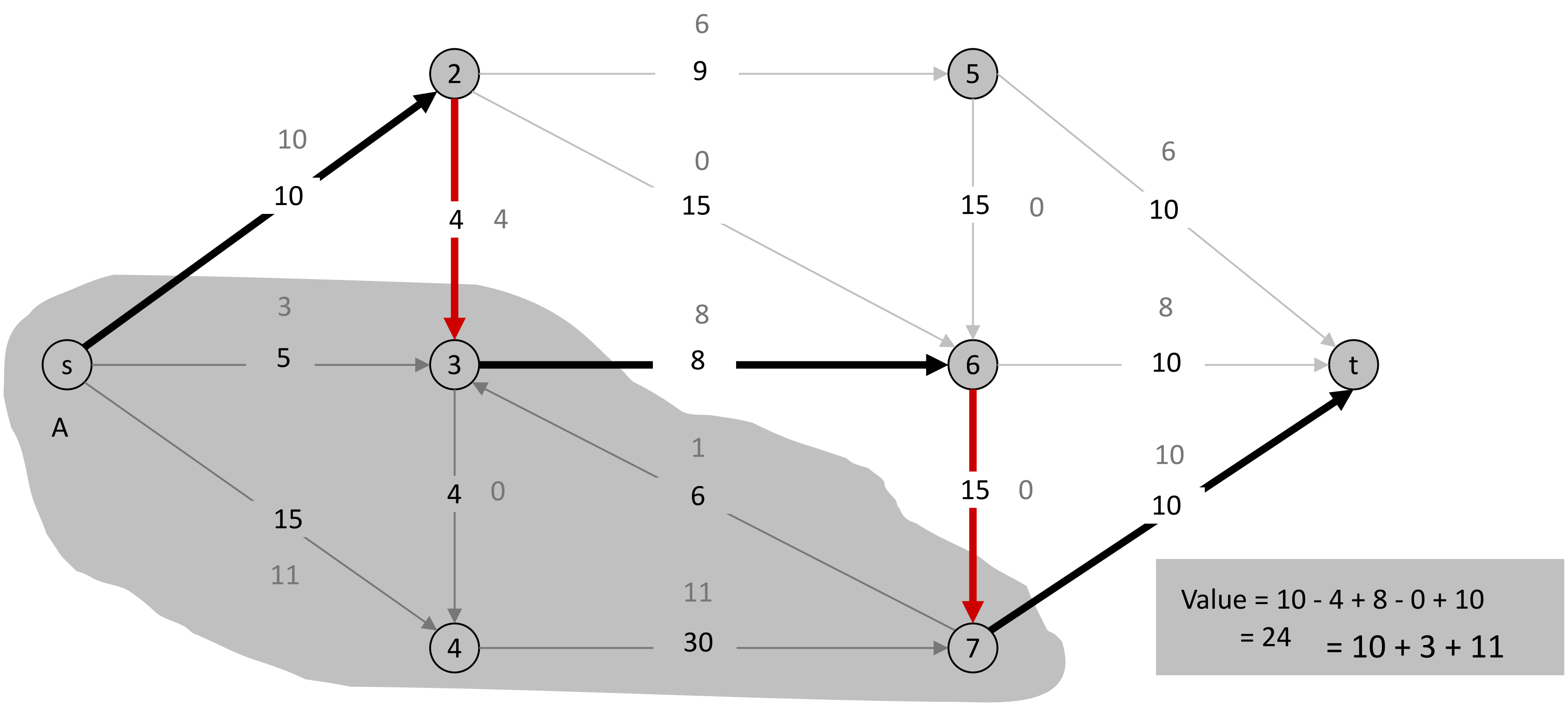


Value = 6 + 0 + 8 - 1 + 11
= 24 = 10 + 3 + 11

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then, the net flow sent across the cut is equal to the amount leaving s.

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



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then

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

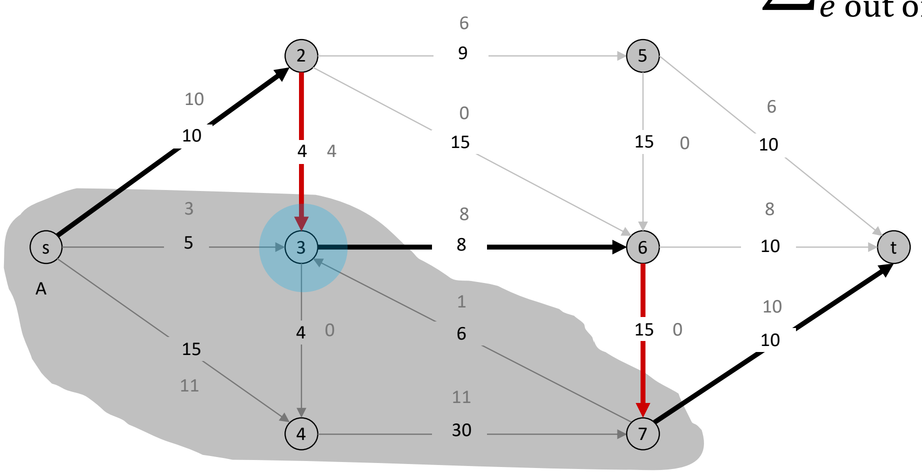
Pf.

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

by flow conservation, all terms except v = s are 0 $\longrightarrow$

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

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

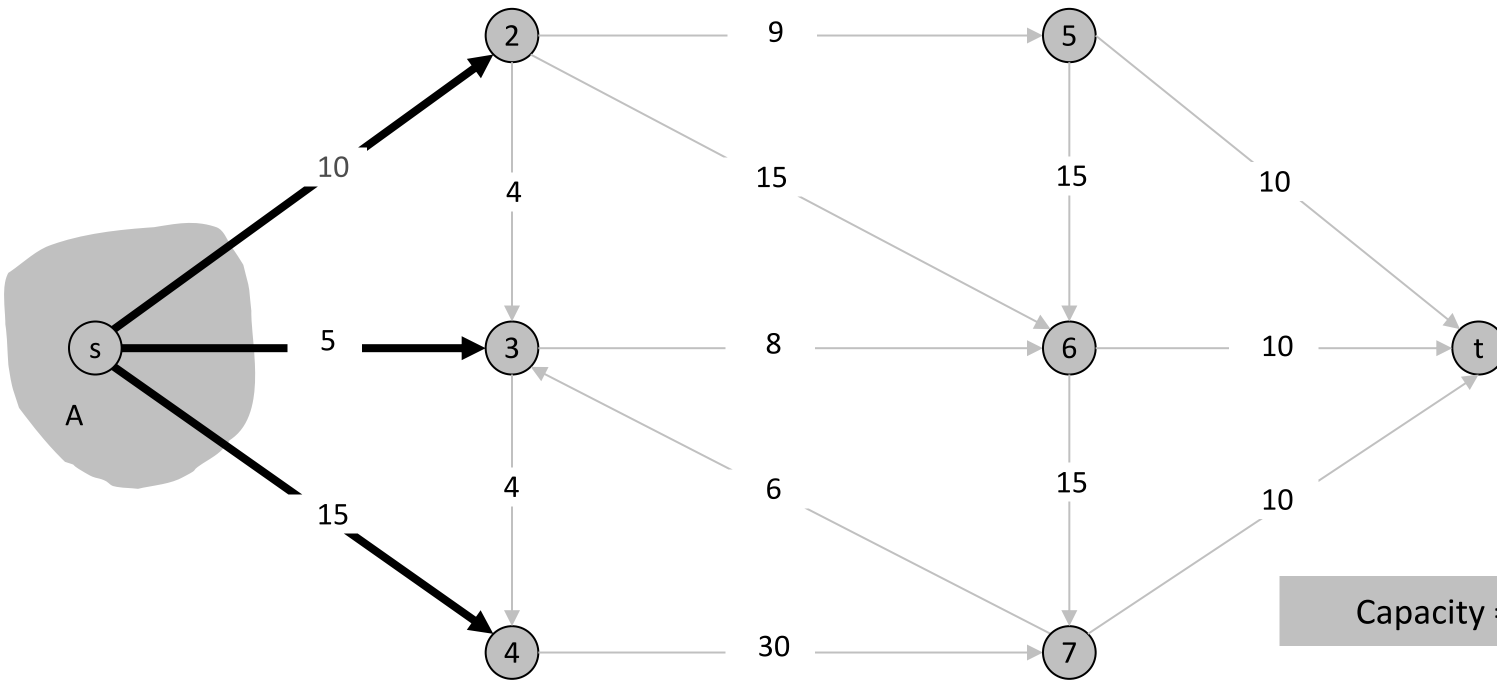


vertex 3: $f_{out} = 8 + 0 = f_{in} = 3 + 4 + 1$

Flows and Cuts

Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30



Capacity = 30

Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have $v(f) \leq cap(A, B)$.

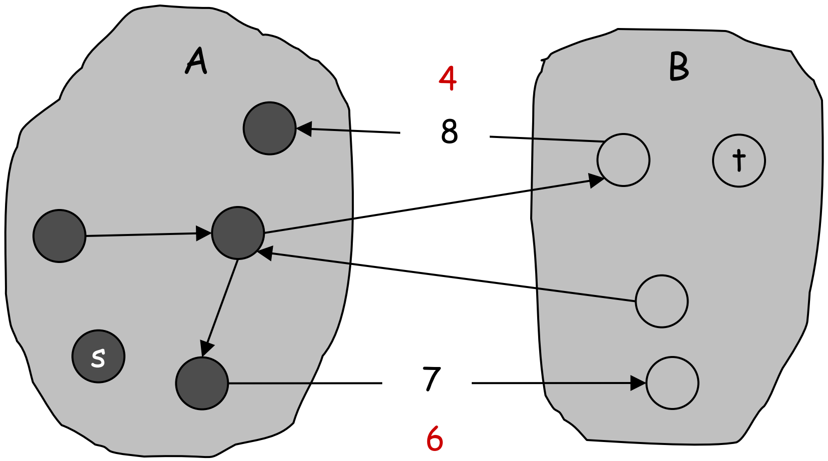
Pf.

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

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

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

$$= cap(A, B)$$

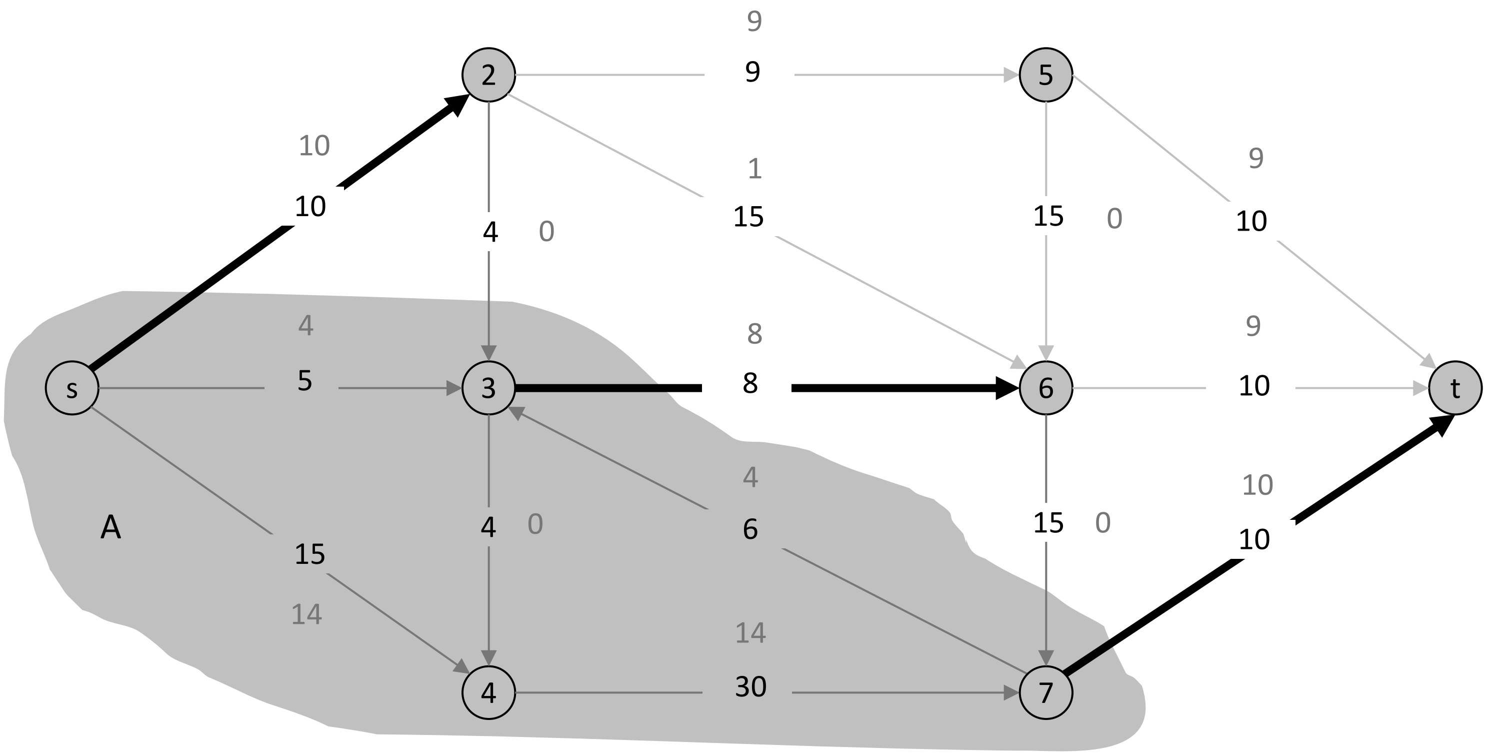


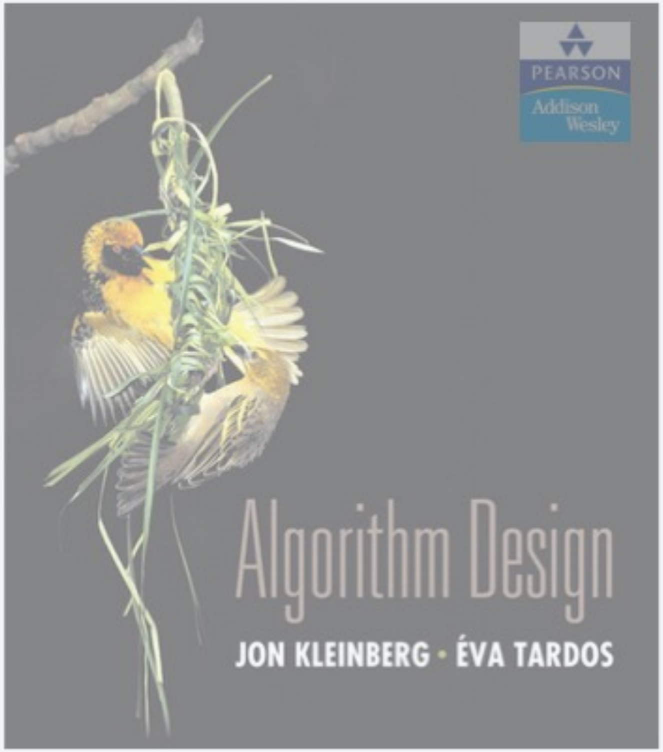
Certificate of Optimality

Corollary. Let f be any flow, and let (A, B) be any cut.

If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28





SECTION 7.1

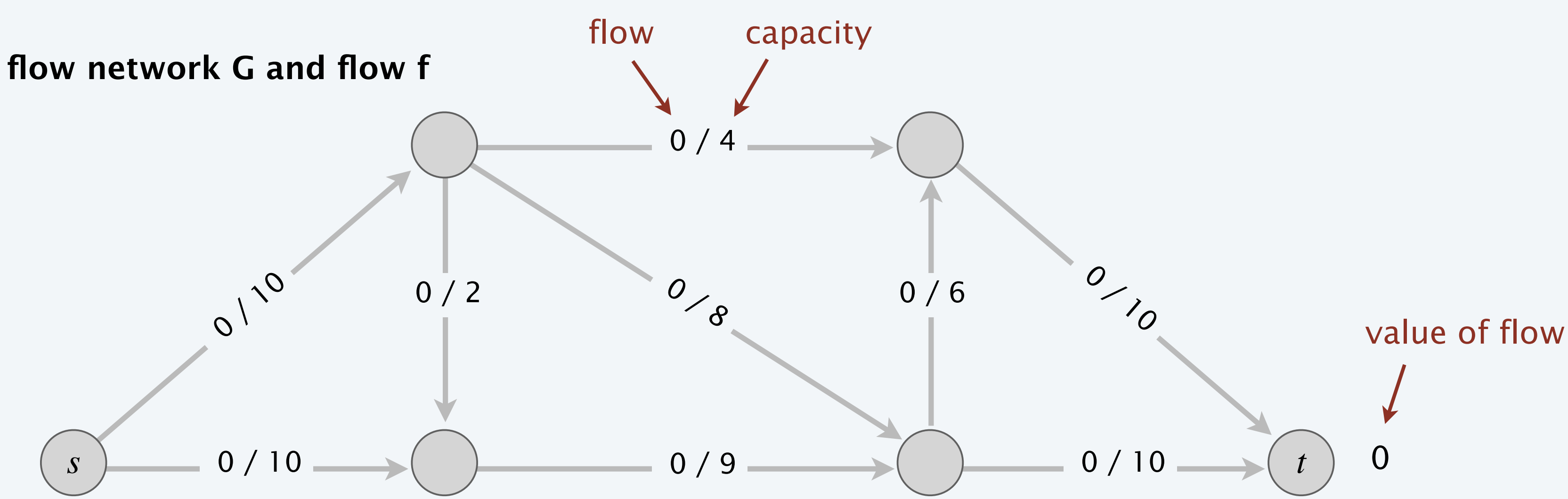
7. NETWORK FLOW I

- 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.

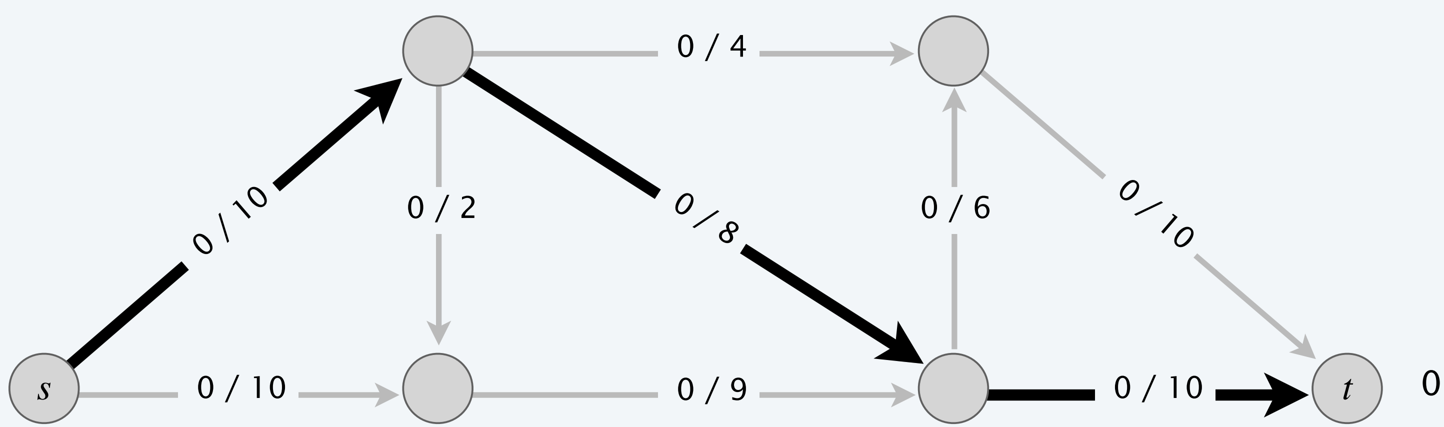


Toward a max-flow algorithm

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

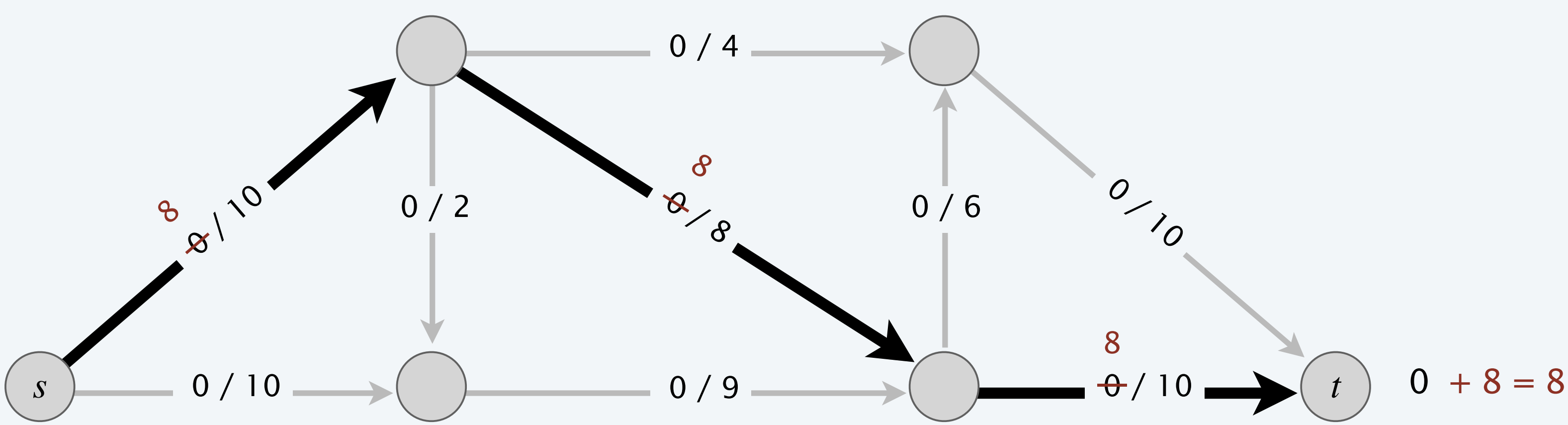


Toward a max-flow algorithm

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

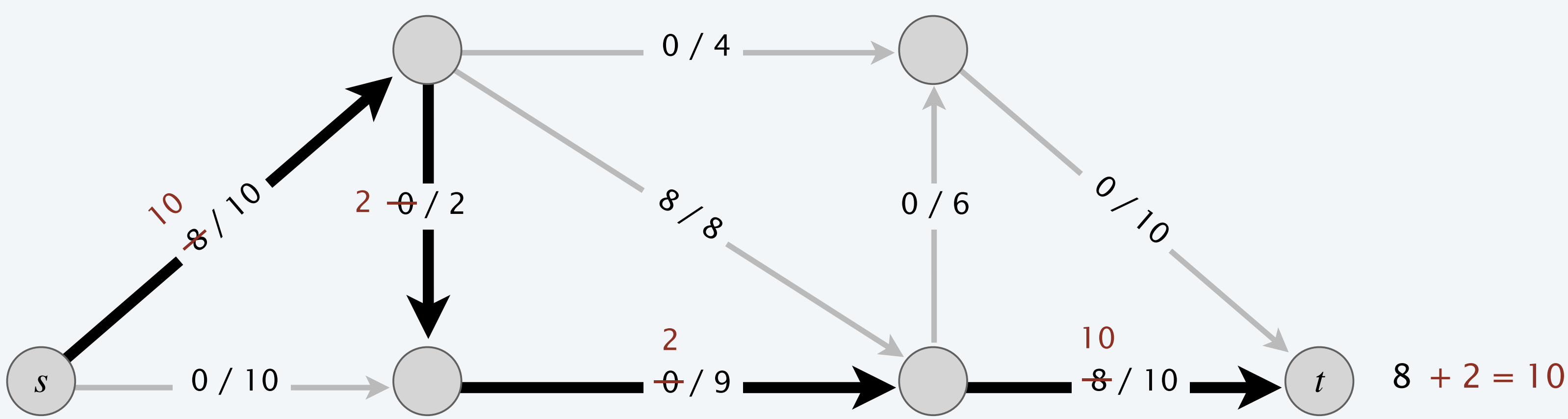


Toward a max-flow algorithm

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

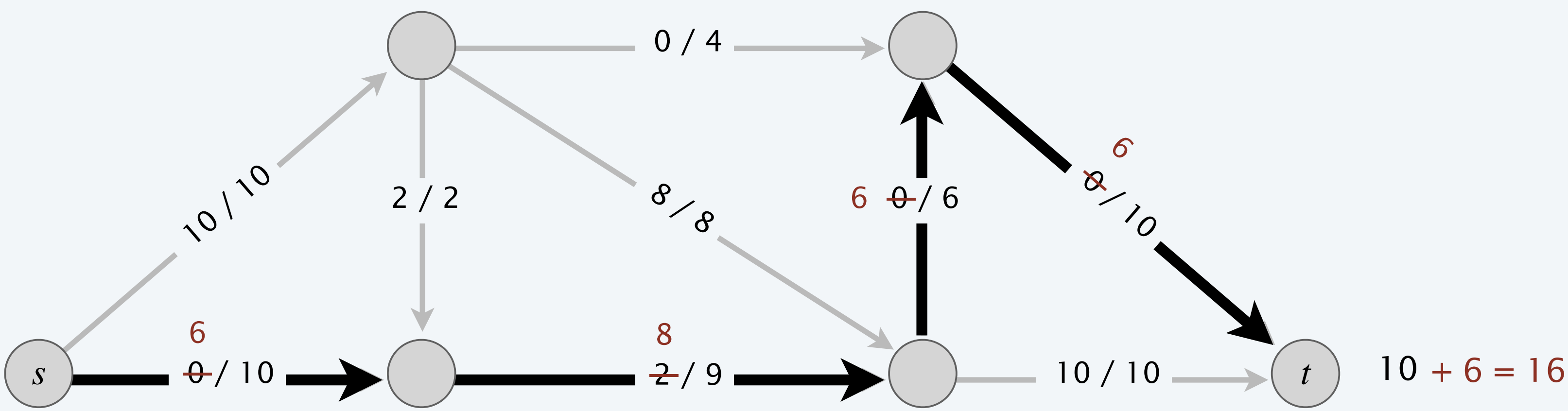


Toward a max-flow algorithm

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



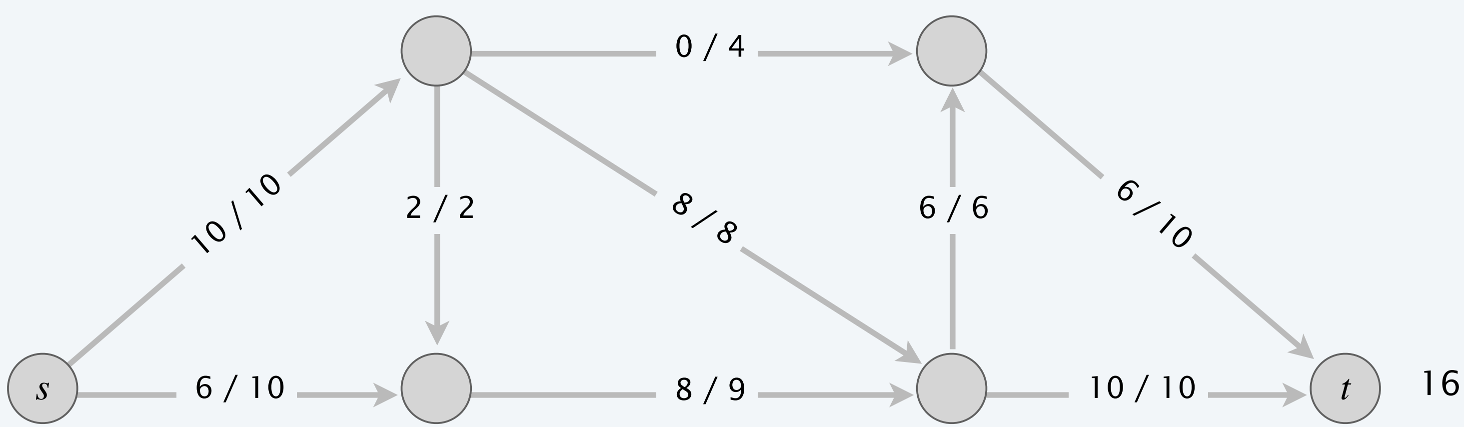
Toward a max-flow algorithm

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

flow network G and flow f



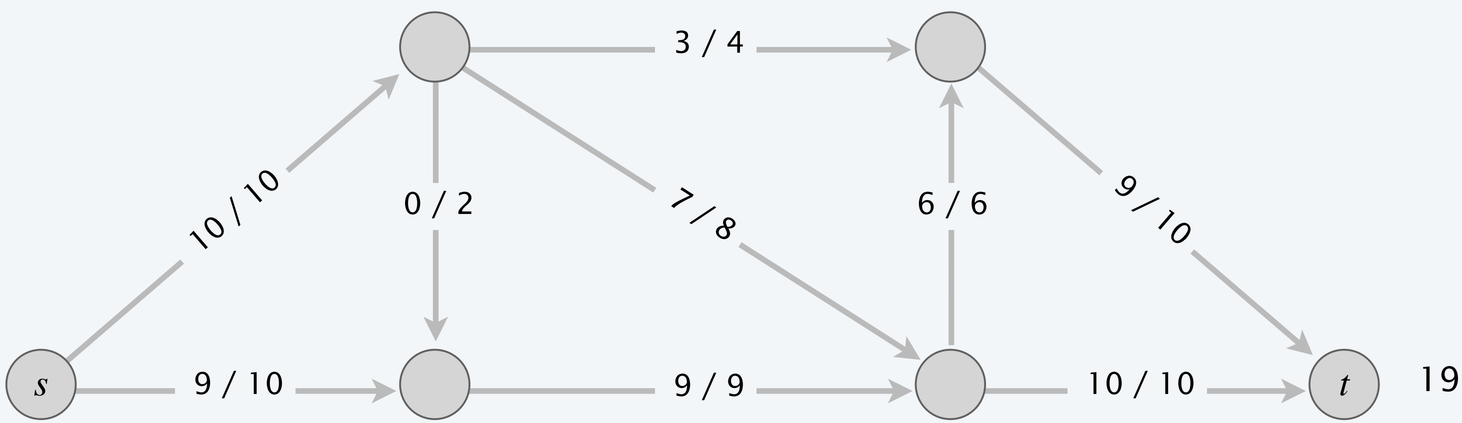
Toward a max-flow algorithm

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.

but max-flow value = 19

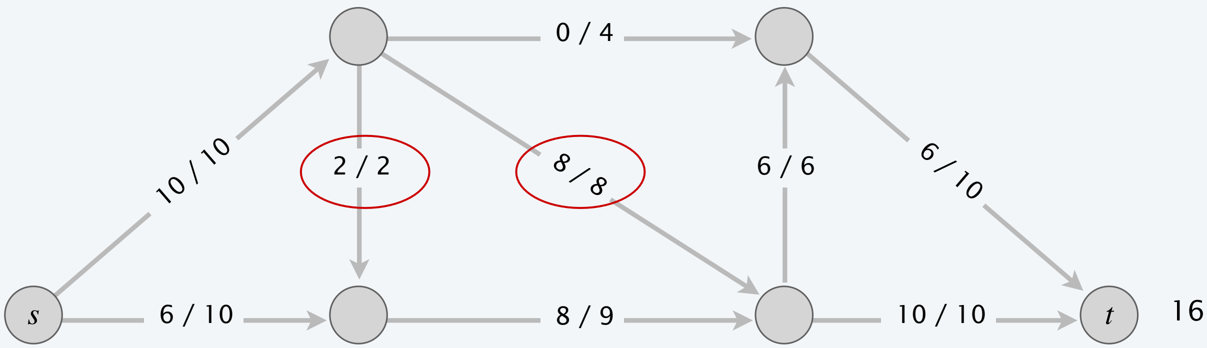
flow network G and flow f



Compare the two solutions

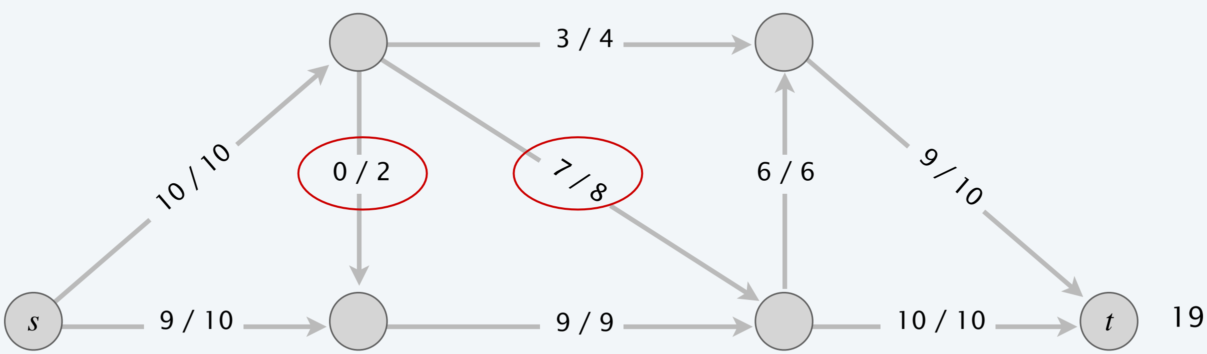
ending flow value = 16

flow network G and flow f



but max-flow value = 19

flow network G and flow f



2/2 8/8

decrease

0/2 7/8

locally optimality
≠
global optimality

Why the greedy algorithm fails

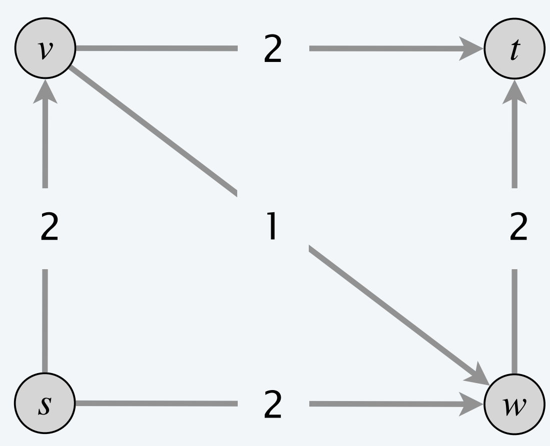
Q. Why does the greedy algorithm fail?

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.

that is, $v(f) = 4$



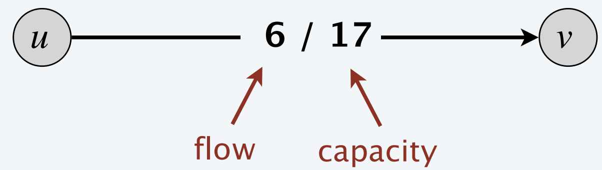
Bottom line. Need some mechanism to "undo" a bad decision.

Residual network (Residual Graph)

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

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

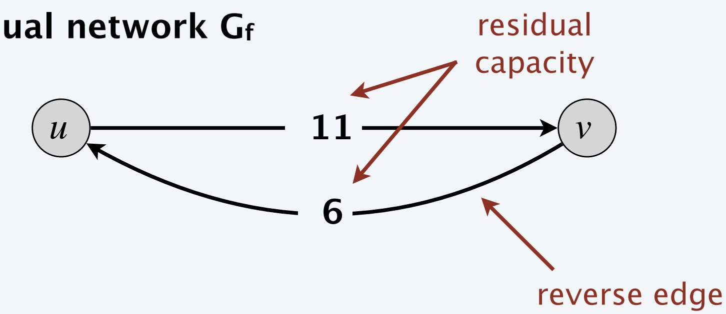
original flow network G



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

- "Undo" flow sent.

residual network G_f



Residual capacity.

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

edges with positive residual capacity

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

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

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

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

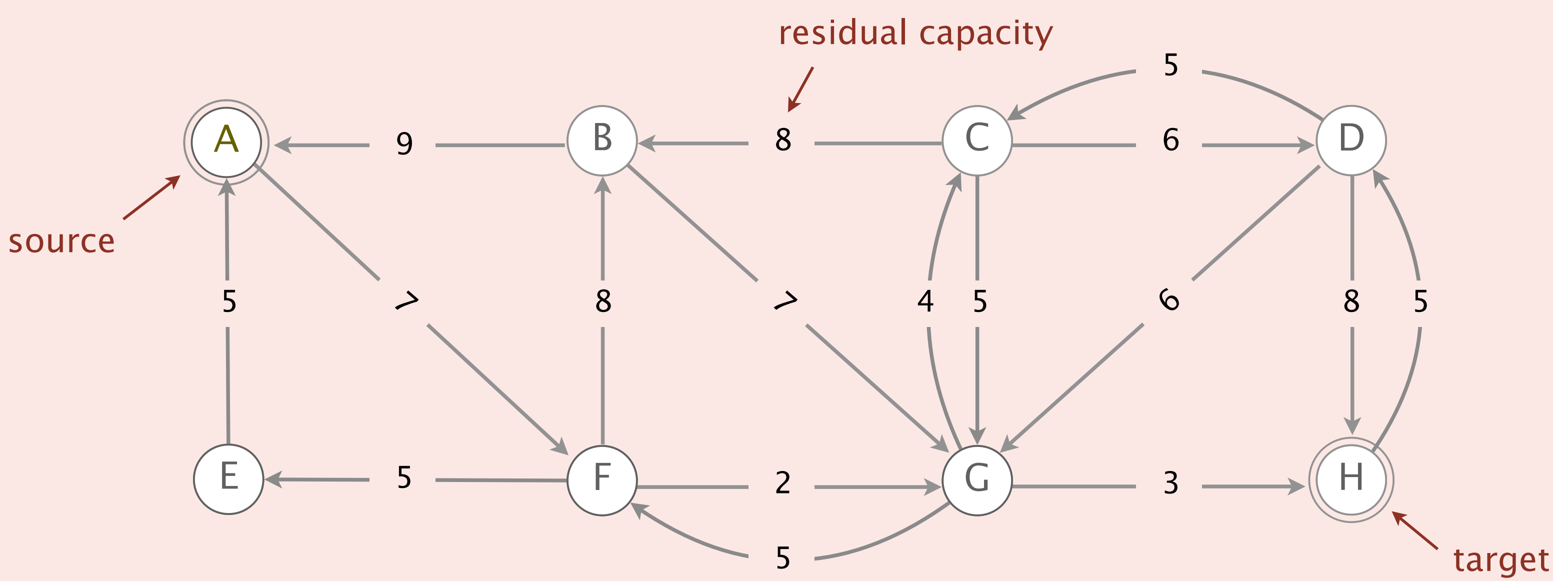
Which is the augmenting path of highest bottleneck capacity?

A. $A \rightarrow F \rightarrow G \rightarrow H$

B. $A \rightarrow B \rightarrow C \rightarrow D \rightarrow H$

C. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow H$

D. $A \rightarrow F \rightarrow B \rightarrow G \rightarrow C \rightarrow D \rightarrow H$



Ford–Fulkerson algorithm

Ford–Fulkerson augmenting path algorithm.

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path P in the residual network G_f.
- Augment flow along path P.
- Repeat until you get stuck. (that is, can no longer find an s⤳t path)

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.

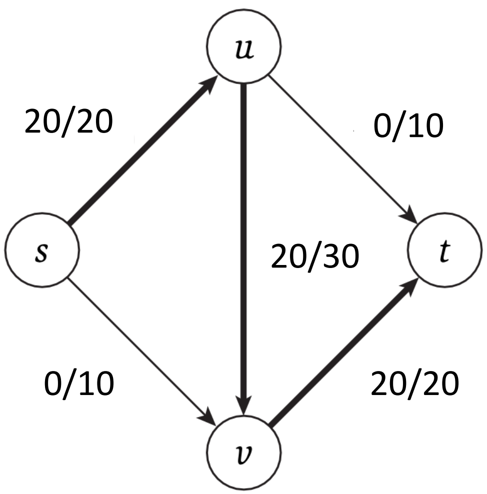
RETURN f.

augmenting path

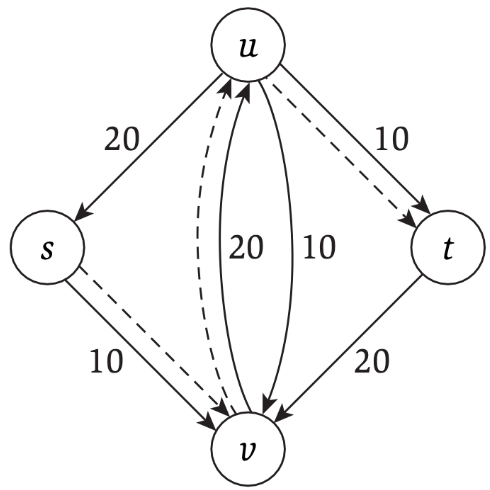
时间复杂度为O (E f)，其中E表示边的数量，f表示最终流量的大小。 如果使用最小割定理，时间复杂度可优化为O (E^2 log (C))，其中C为最大容量

Textbook Demo of Ford-Fulkerson Algorithm

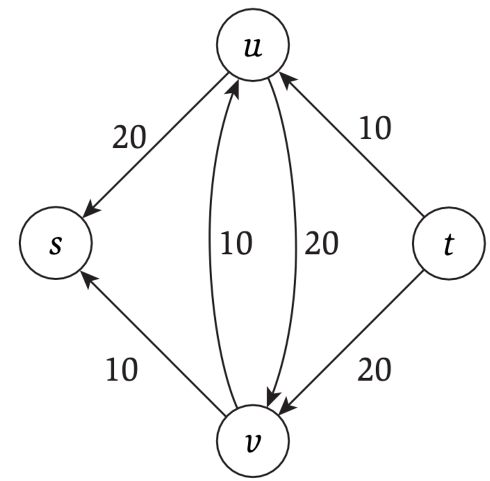
Page 342 demo



(a)



(b)

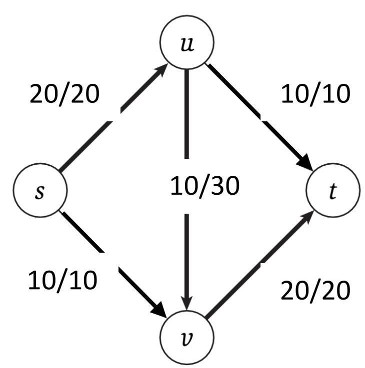


(c)

augmenting path: s -> v -> u -> t

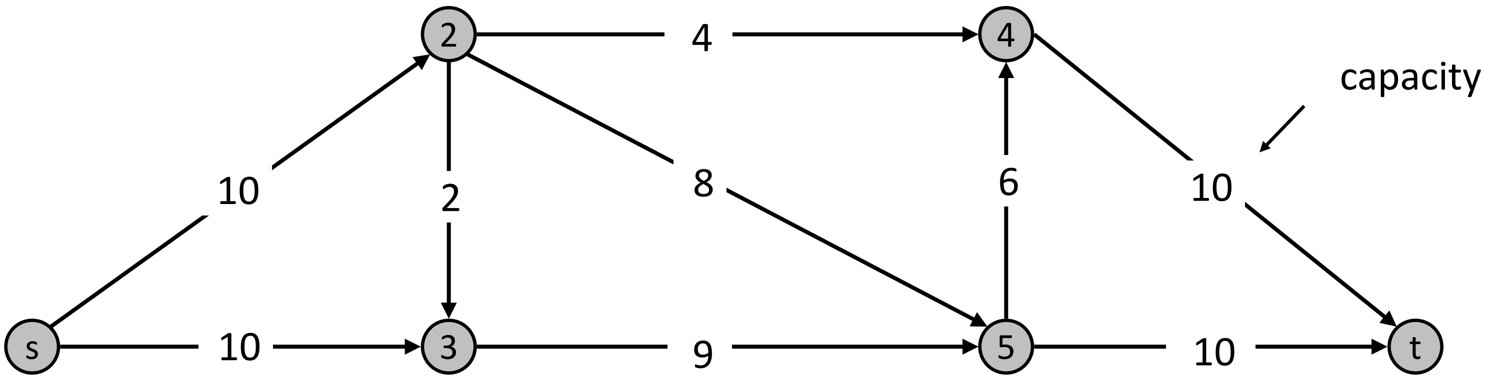
Updated G_f

Update f:



Ford-Fulkerson Algorithm

G:

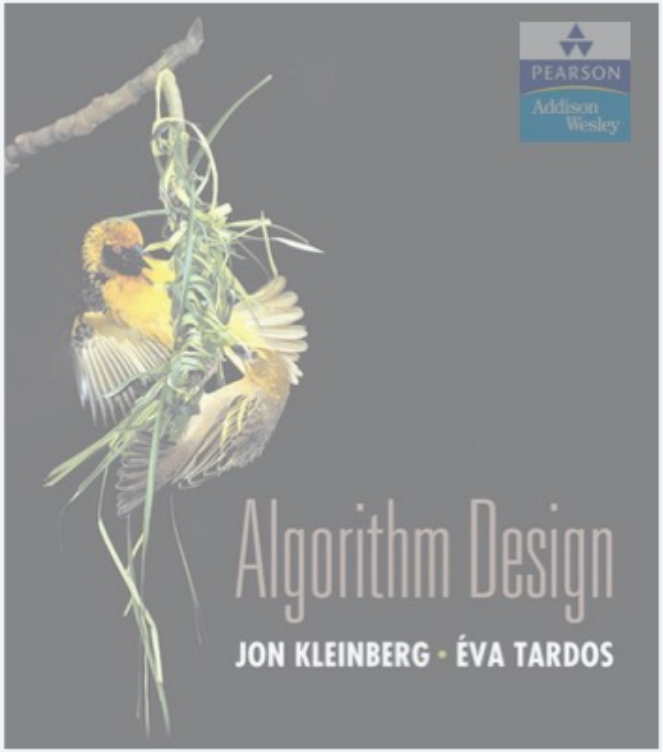


Augmenting Path Algorithm

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b        forward edge
      else       f(eᴿ)← f(eᴿ) - b       reverse edge
   }
   return f
}
```

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update Gf
   }
   return f
}
```



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*

Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow f is a max flow *iff* there are no augmenting paths.

Max-flow min-cut theorem. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Pf. We prove both simultaneously by showing TFAE (The Following Are Equivalent):

(i) There exists a cut (A, B) such that v(f) = cap(A, B).

(ii) Flow f is a max flow.

(iii) There is no augmenting path relative to f.

(i) $\Rightarrow$ (ii) This was the corollary to weak duality lemma.

(ii) $\Rightarrow$ (iii) We show contrapositive.

- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

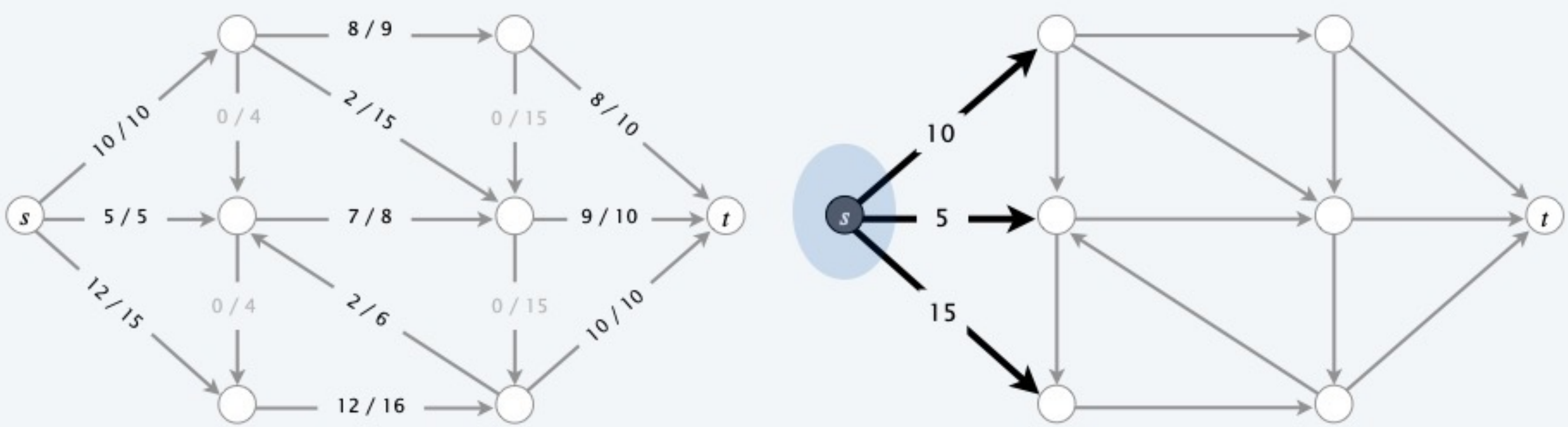
Relationship between flows and cuts

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

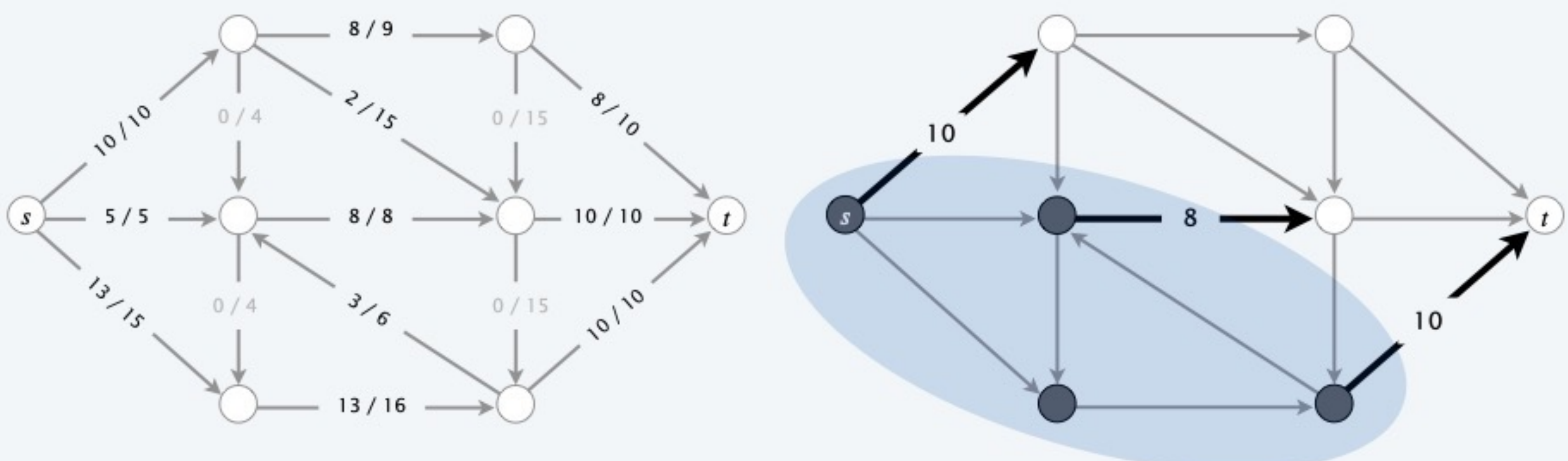
Corollary. Let f be a flow and let (A, B) be any cut.
If $val(f) = cap(A, B)$, then f is a max flow and (A, B) is a min cut.

Pf.

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

[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: $t \notin A$.

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

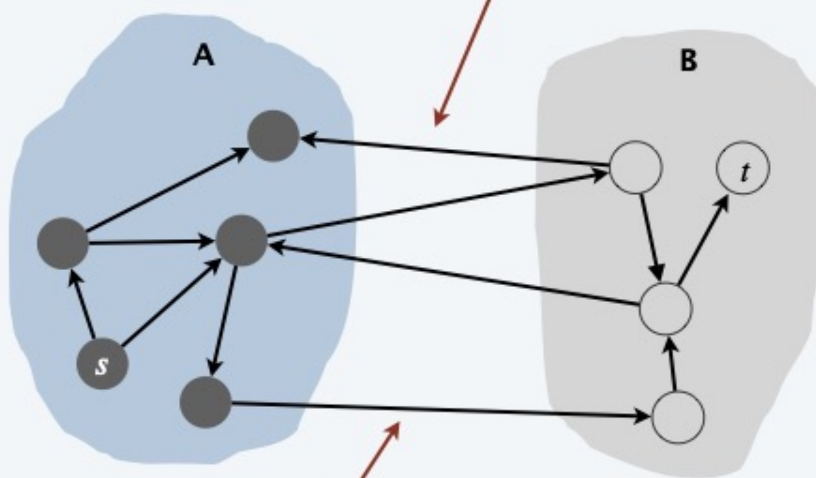
flow value lemma

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

$$= cap(A, B) \quad ▪$$

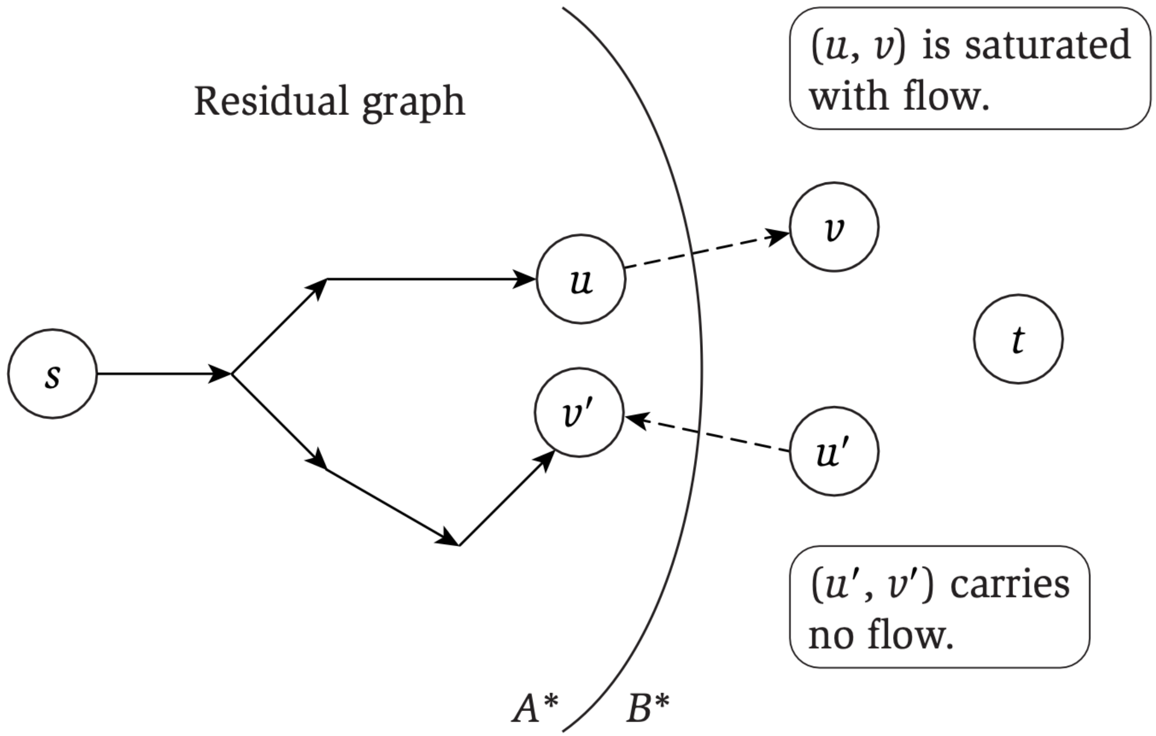
edge $e = (v, w)$ with $v \in B, w \in A$
must have $f(e) = 0$

original flow network G



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

Proof of Max-Flow Min-Cut Theorem



If f is an *s*-*t* flow such that there is no *s*-*t* path in the residual graph G_f , then there is an *s*-*t* cut (A∗,B∗) in G for which v(f)=c(A∗,B∗)

(Textbook, page 349)

Running Time

Assumption. All capacities are integers between 1 and C.

Invariant. Every flow value f(e) and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.

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

Corollary. If C = 1, Ford-Fulkerson runs in O(mn) time.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.
Pf. Since algorithm terminates, theorem follows from invariant. ▪

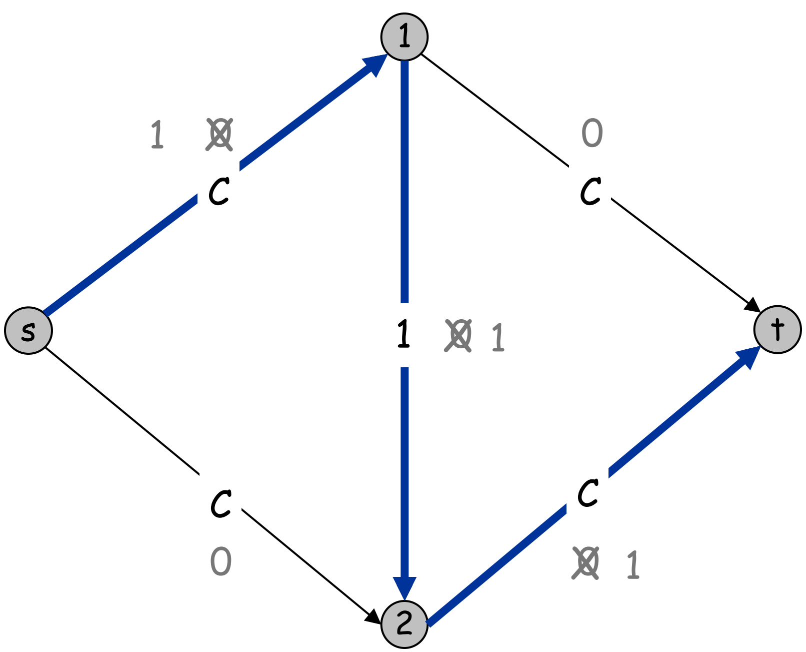
7.3 Choosing Good Augmenting Paths

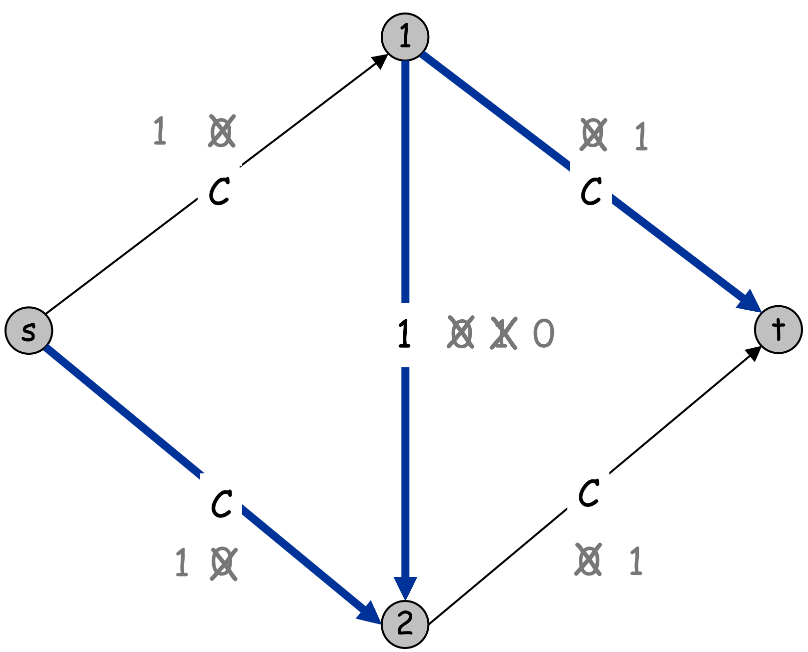
Ford-Fulkerson: Exponential Number of Augmentations

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

m, n, and log C

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





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.

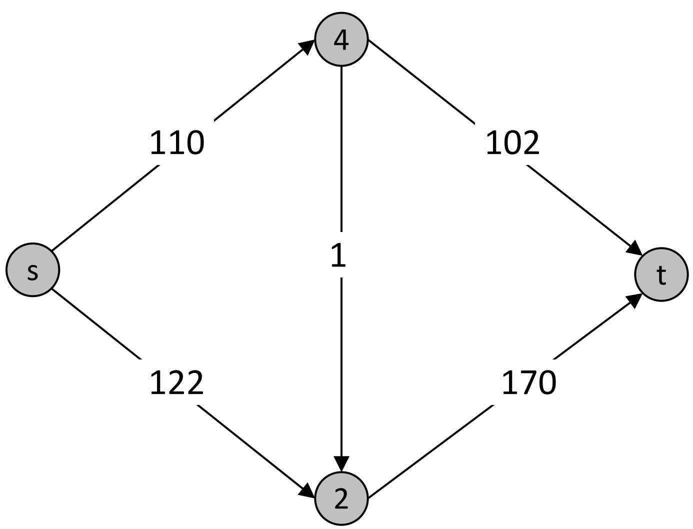
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

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

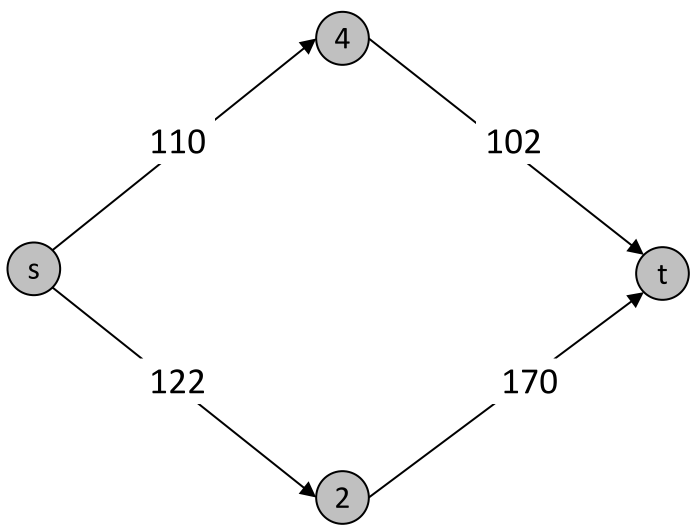
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- 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 of only edges with capacity at least Δ.



G_f



$G_f(100)$

Capacity Scaling

Let C be the maximum capacity out of s: $\Delta \leq \max_{e \text{ out of } s} c_e$

```
Scaling-Max-Flow(G, s, t, C) {
   foreach e ∈ E  f(e) ← 0
   Δ ← largest power of 2 smaller than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

Capacity Scaling: Correctness

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

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

Correctness. If the 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: Running Time

Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.

Pf. Initially $C/2 \leq \Delta < C$. Δ decreases by a factor of 2 each iteration. ▪

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

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

- Let f be the flow at the end of the previous scaling phase.
- L2 $\Rightarrow$ $v(f^*) \leq v(f) + m\,(2\Delta)$.
- Each augmentation in a Δ-phase increases $v(f)$ 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. ▪

Capacity Scaling: Running Time

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.

Pf. (almost identical to proof of max-flow min-cut theorem)

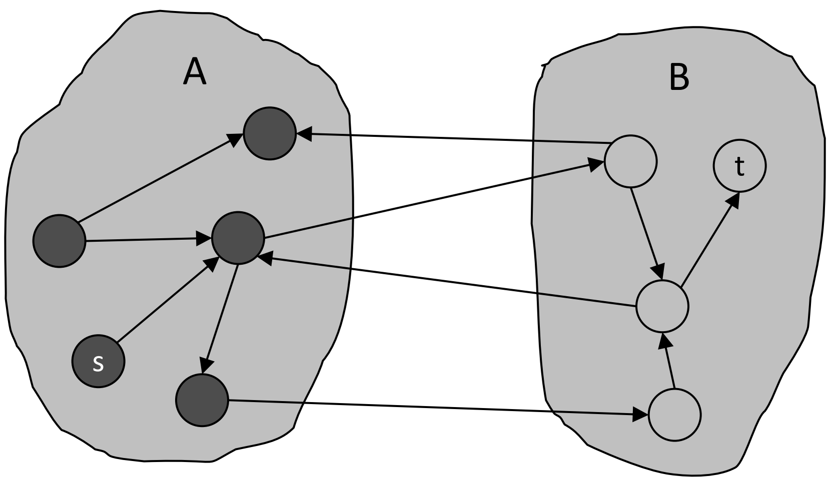
- We show that at the end of a Δ-phase, there exists a cut (A, B) such that cap(A, B) $\leq$ $v(f) + m\,\Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

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

$$\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta$$

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

$$\geq cap(A, B) - m\Delta \quad \blacksquare$$



original network