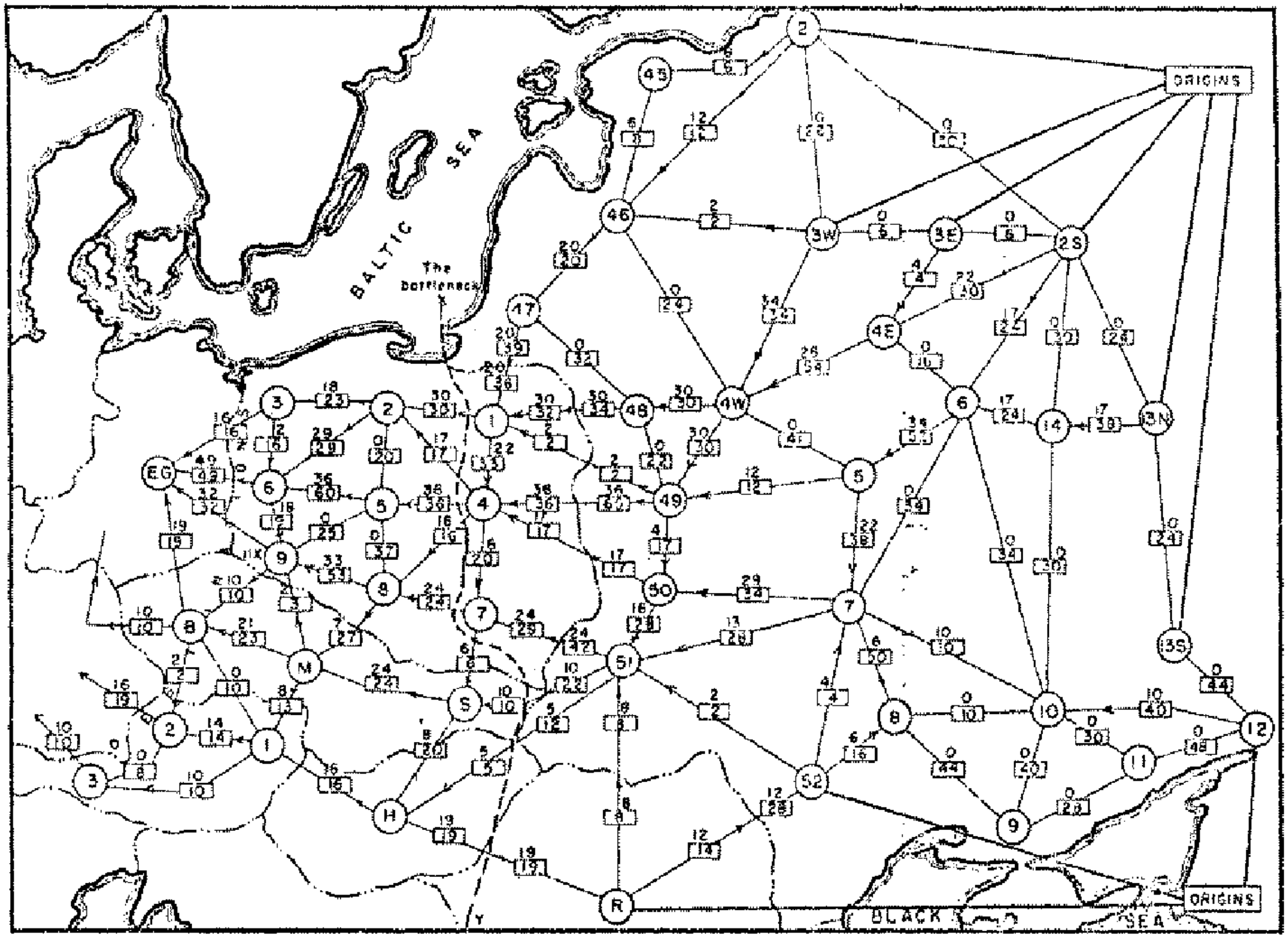
Network Flow

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.

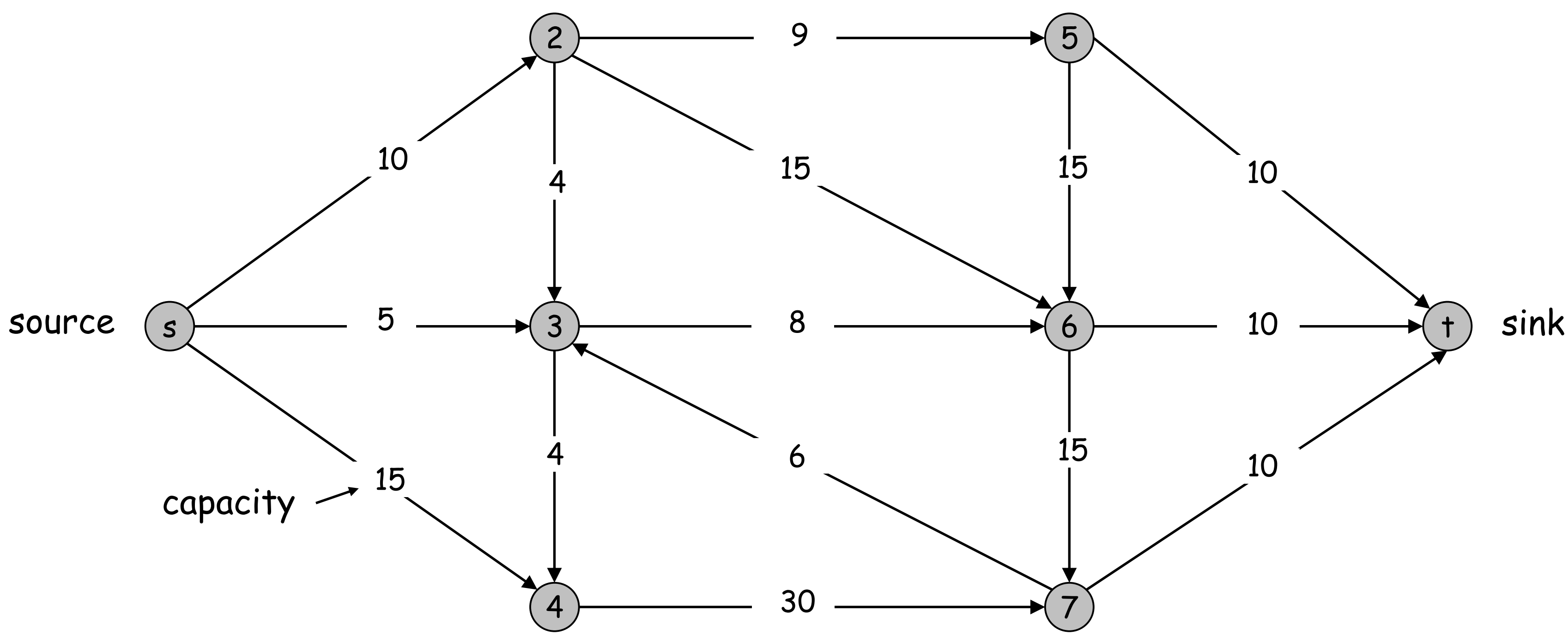
Nontrivial applications / reductions.

- 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 …

Minimum Cut Problem

Flow network.

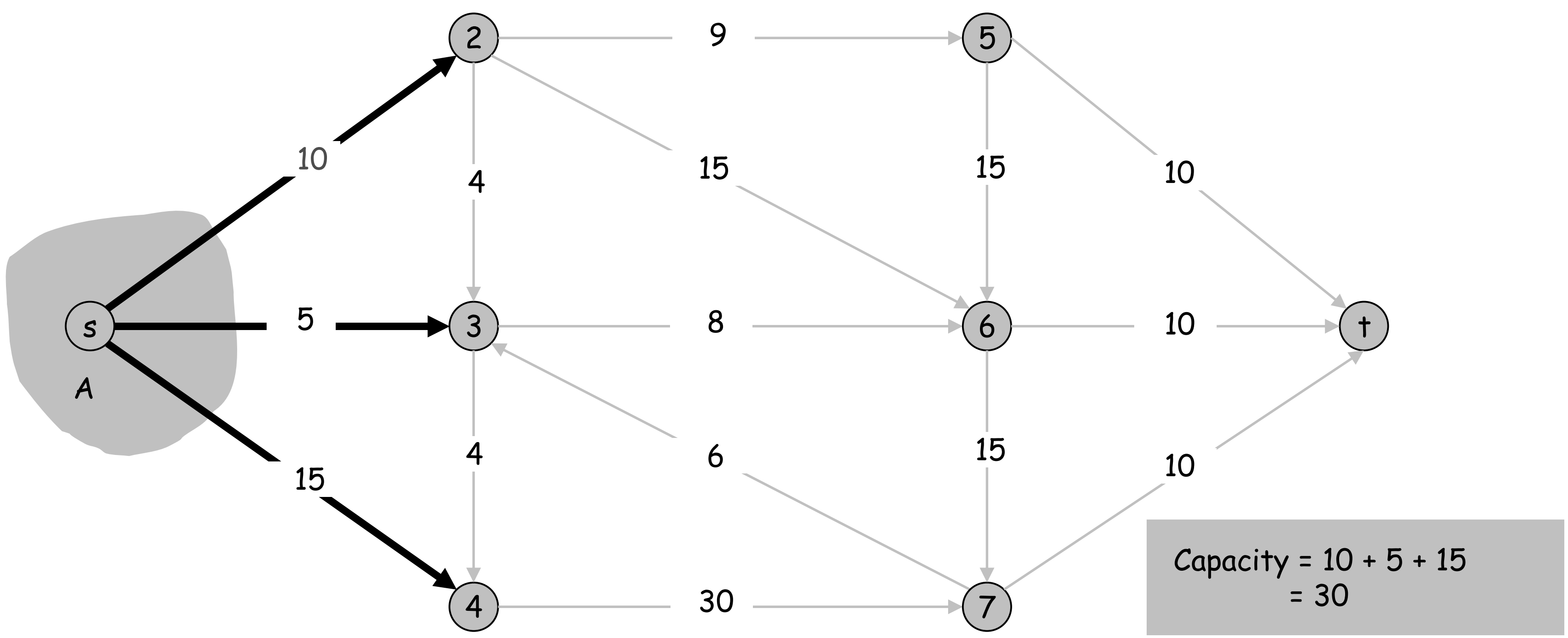
- Abstraction for material flowing through the edges.
- G = (V, E) = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- All nodes have at least ne incident edge.
- c(e) = capacity of edge e.



Cuts

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

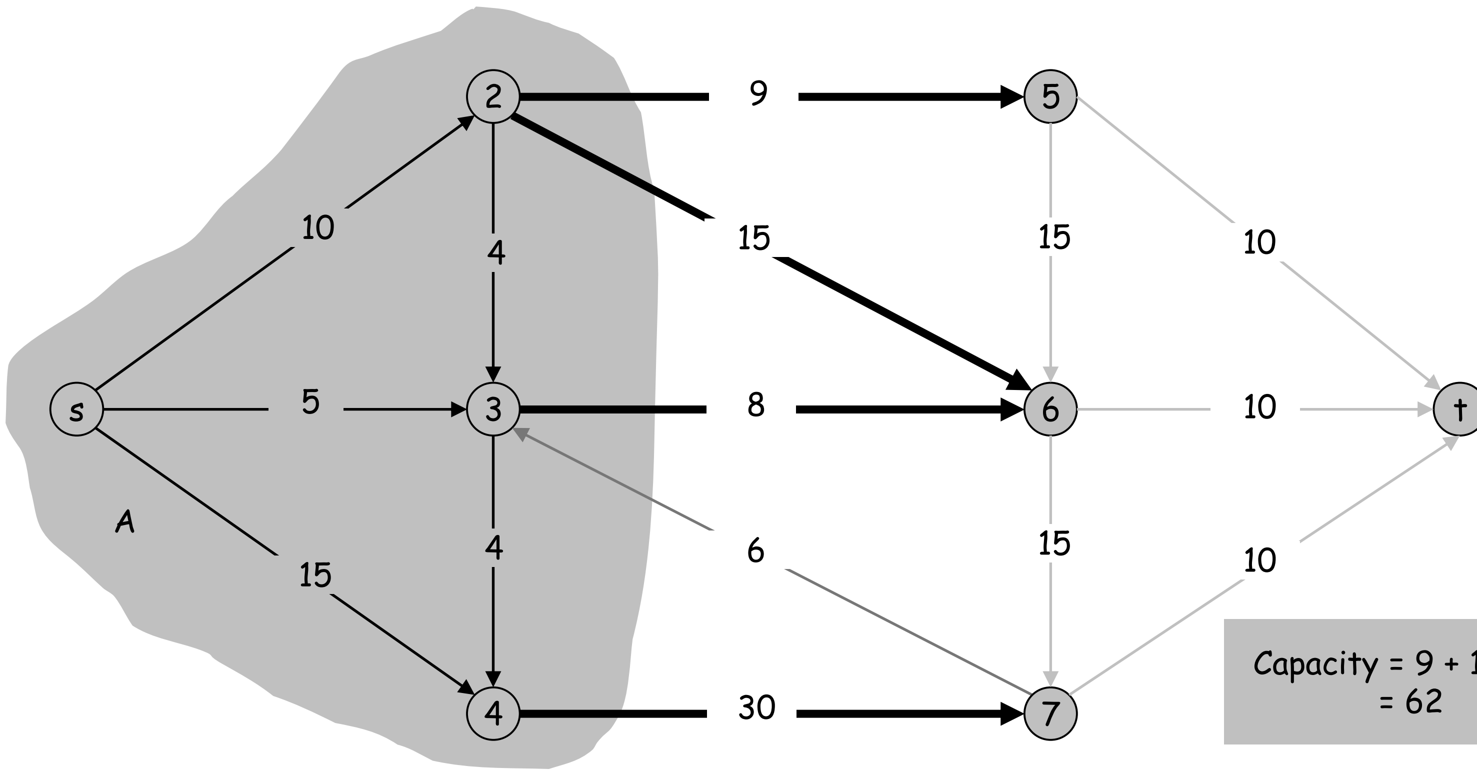
Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Cuts

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

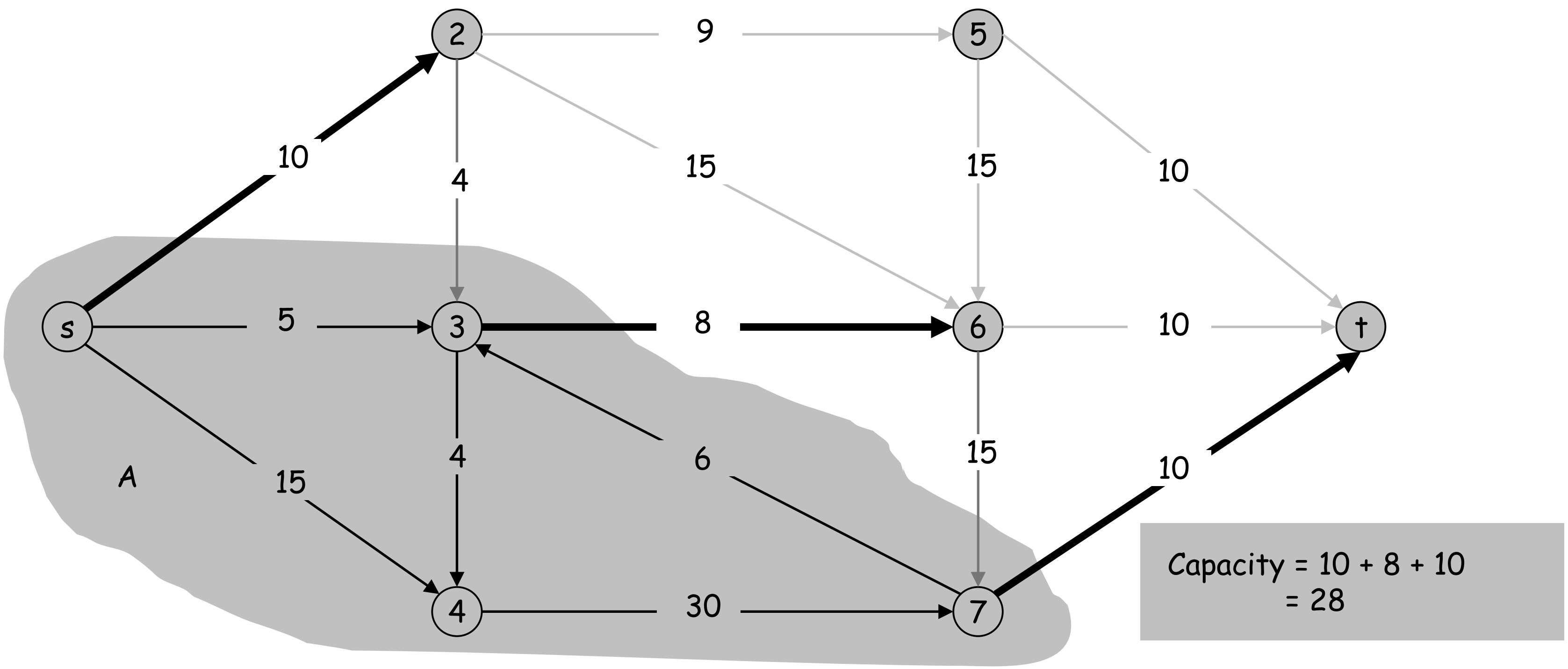
Def. The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

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

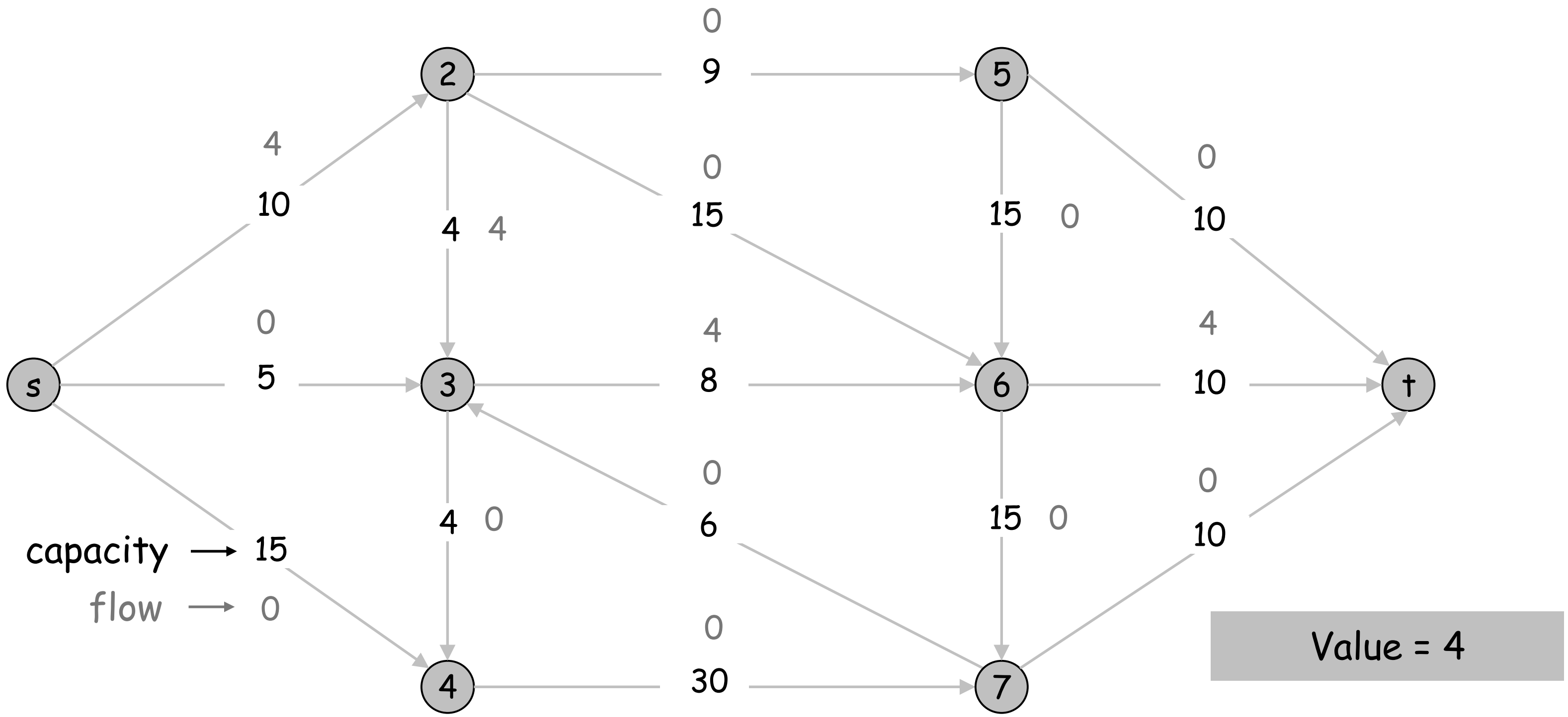


Flows

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

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

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

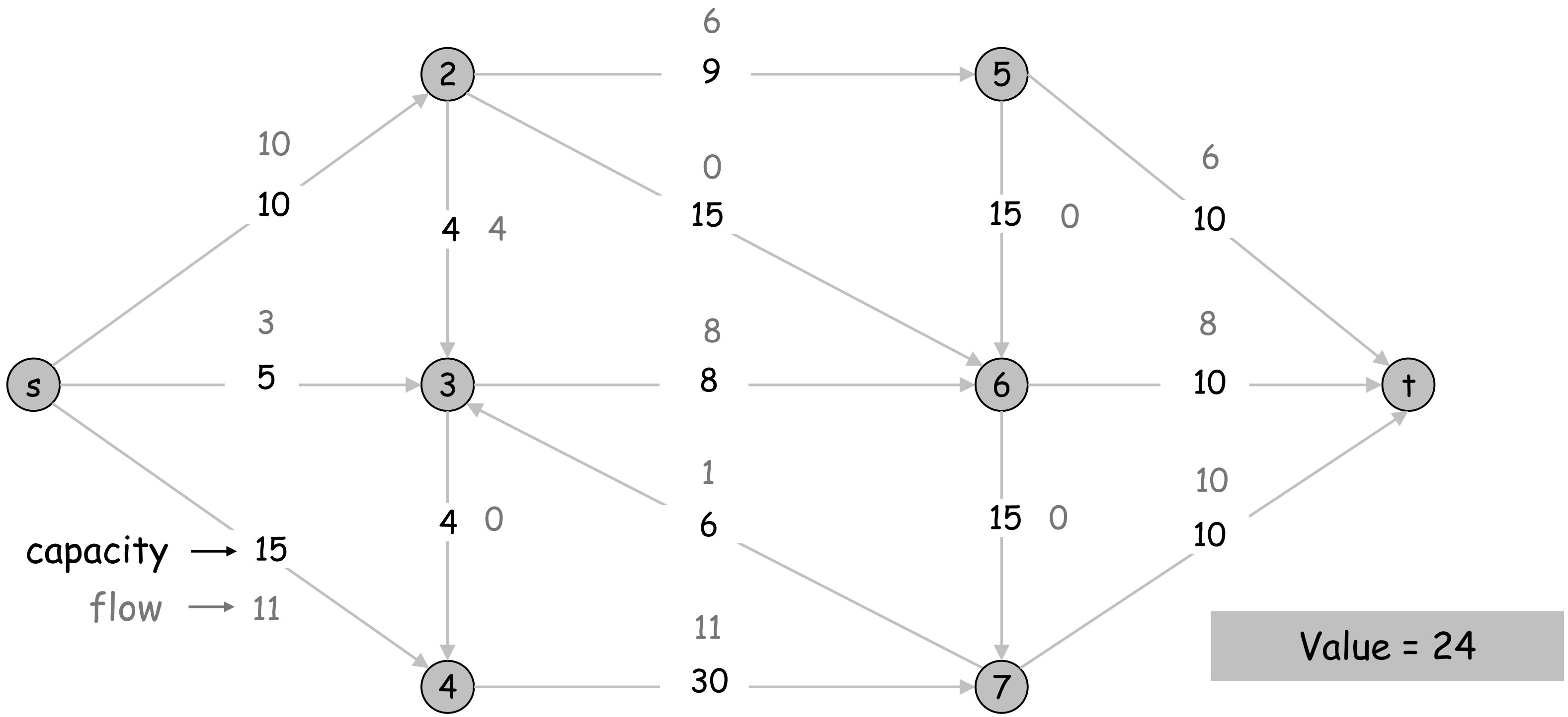


Flows

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

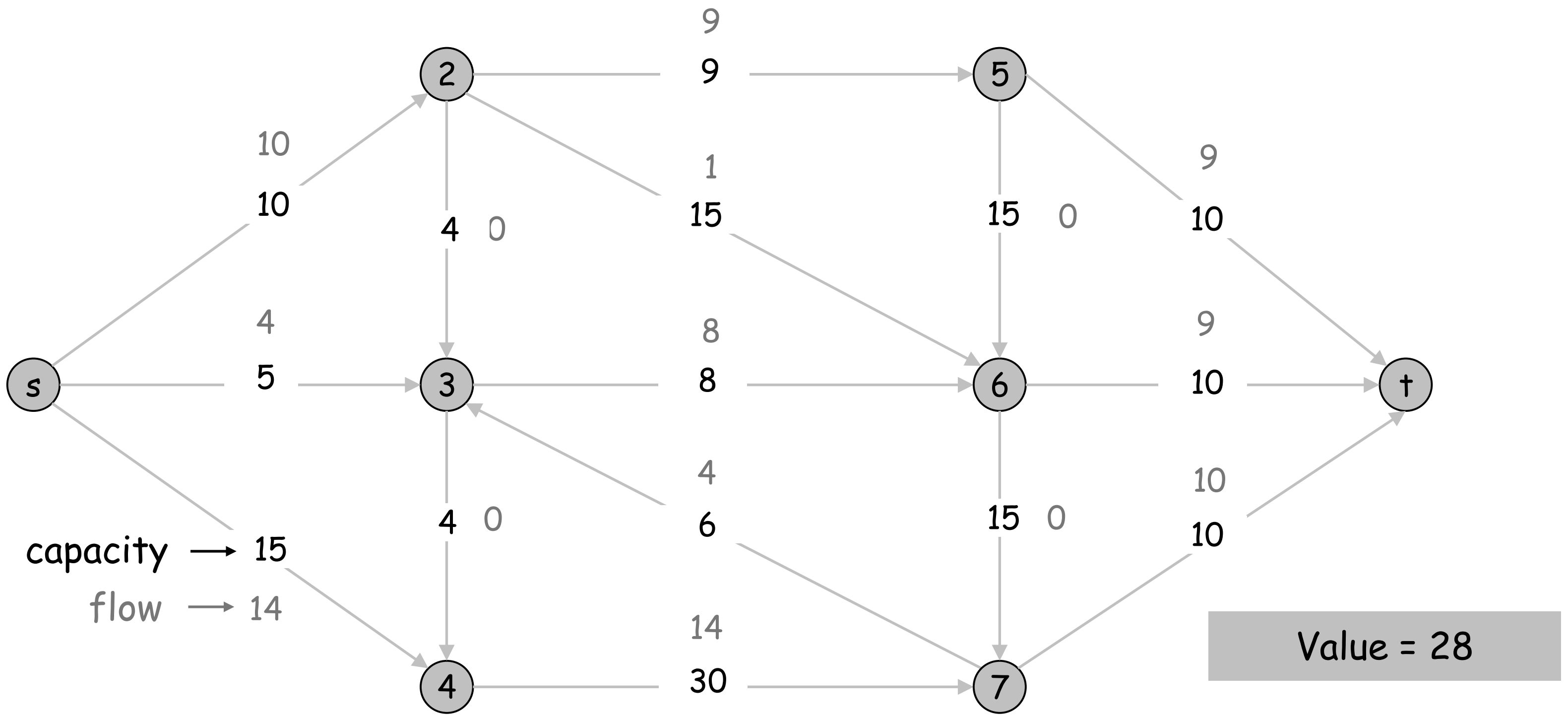
- For each $e \in E$: $\quad 0 \le f(e) \le c(e)$ $\quad$ [capacity]
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ $\quad$ [conservation]

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



Maximum Flow Problem

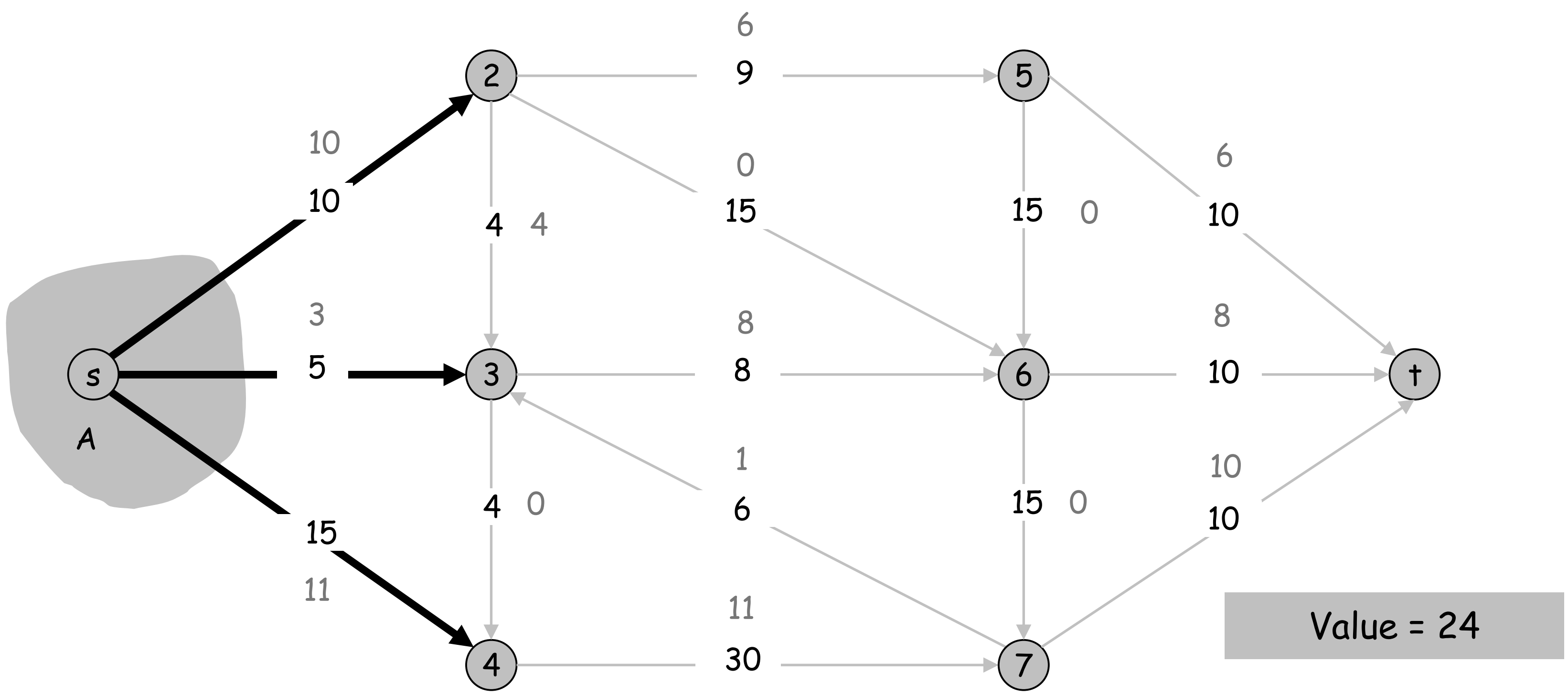
Max flow problem. Find s-t flow of maximum value.



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.

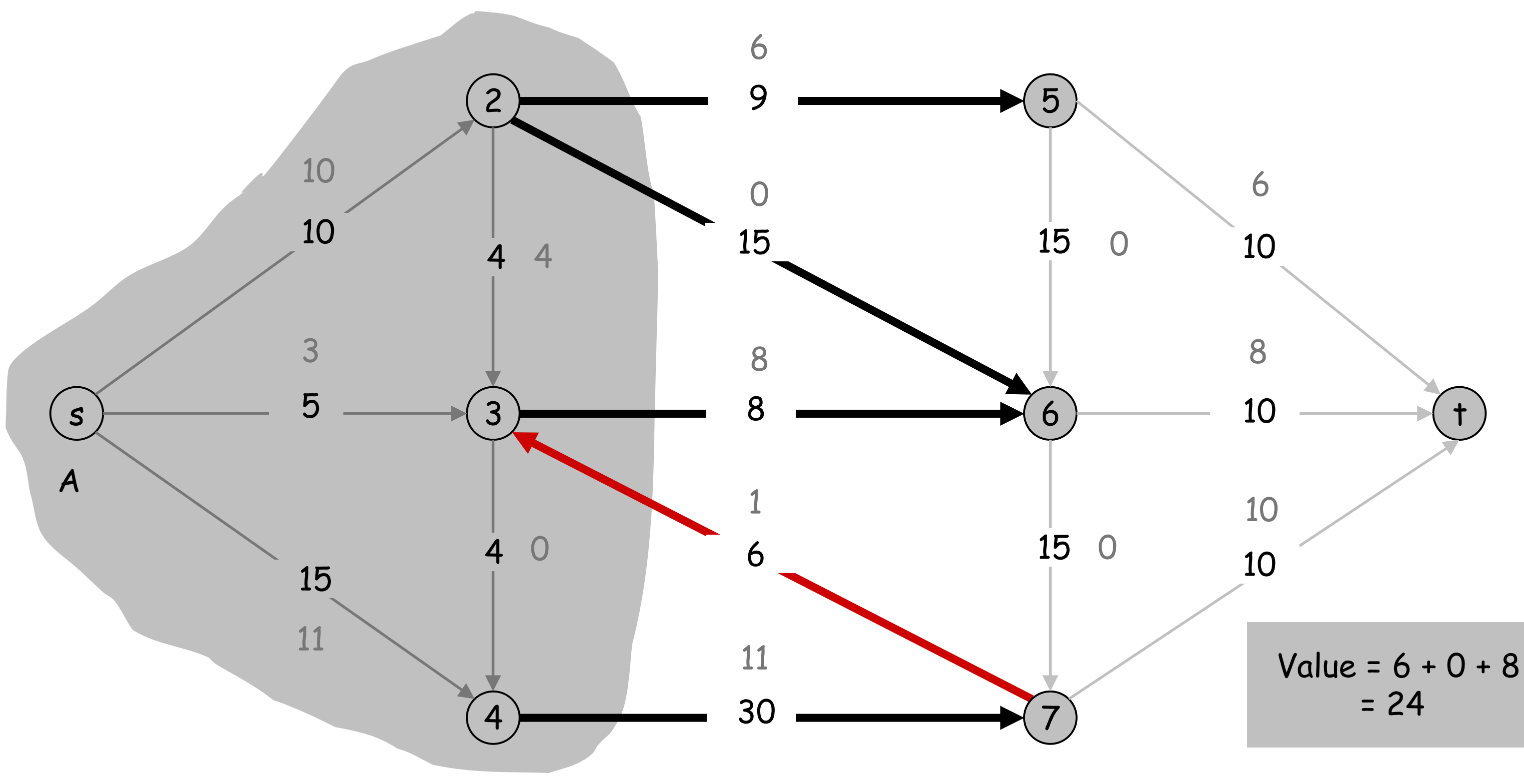
$$\sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;=\; v(f)$$



Flows and Cuts

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

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

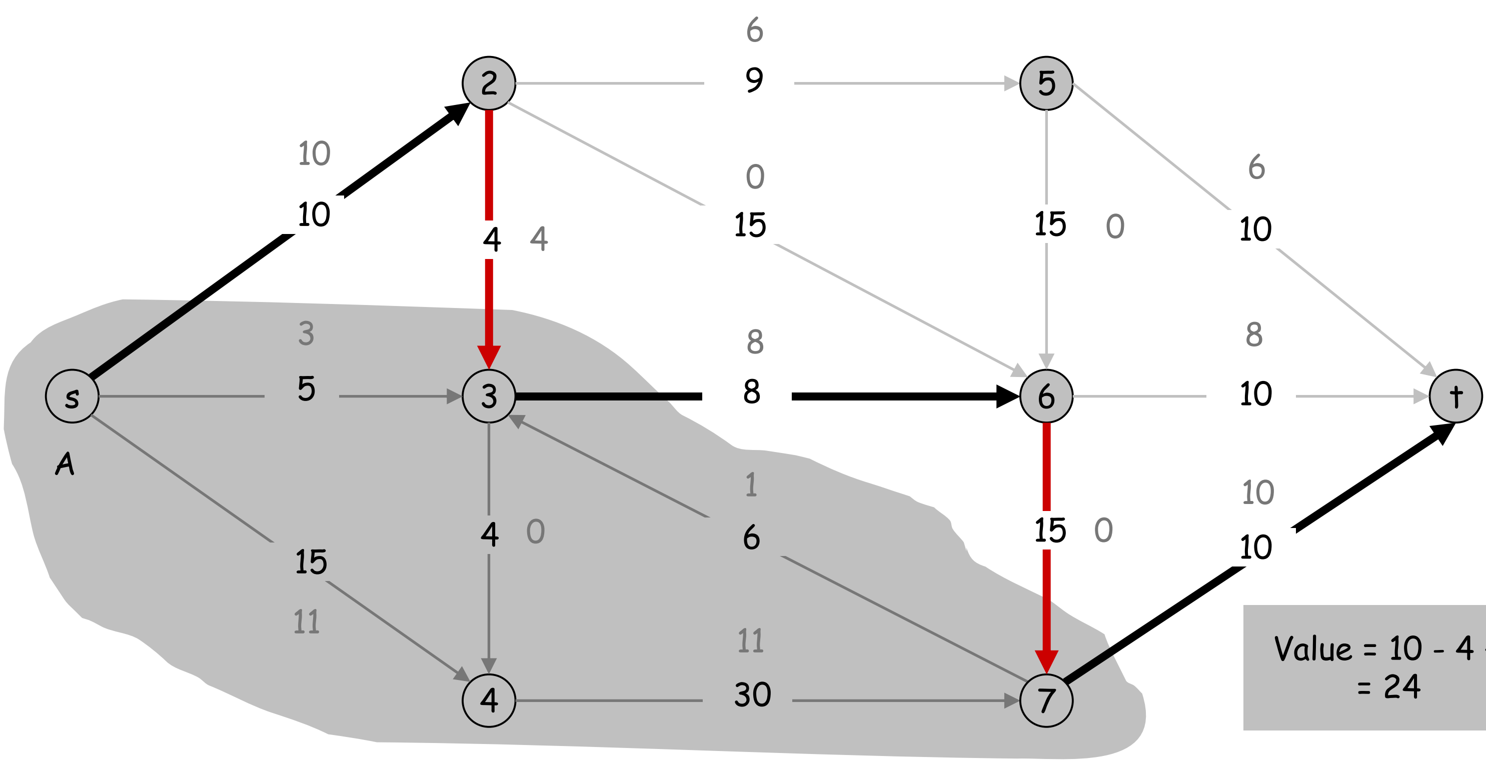


Value = 6 + 0 + 8 - 1 + 11
= 24

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.

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



Value = 10 - 4 + 8 - 0 + 10
= 24

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.

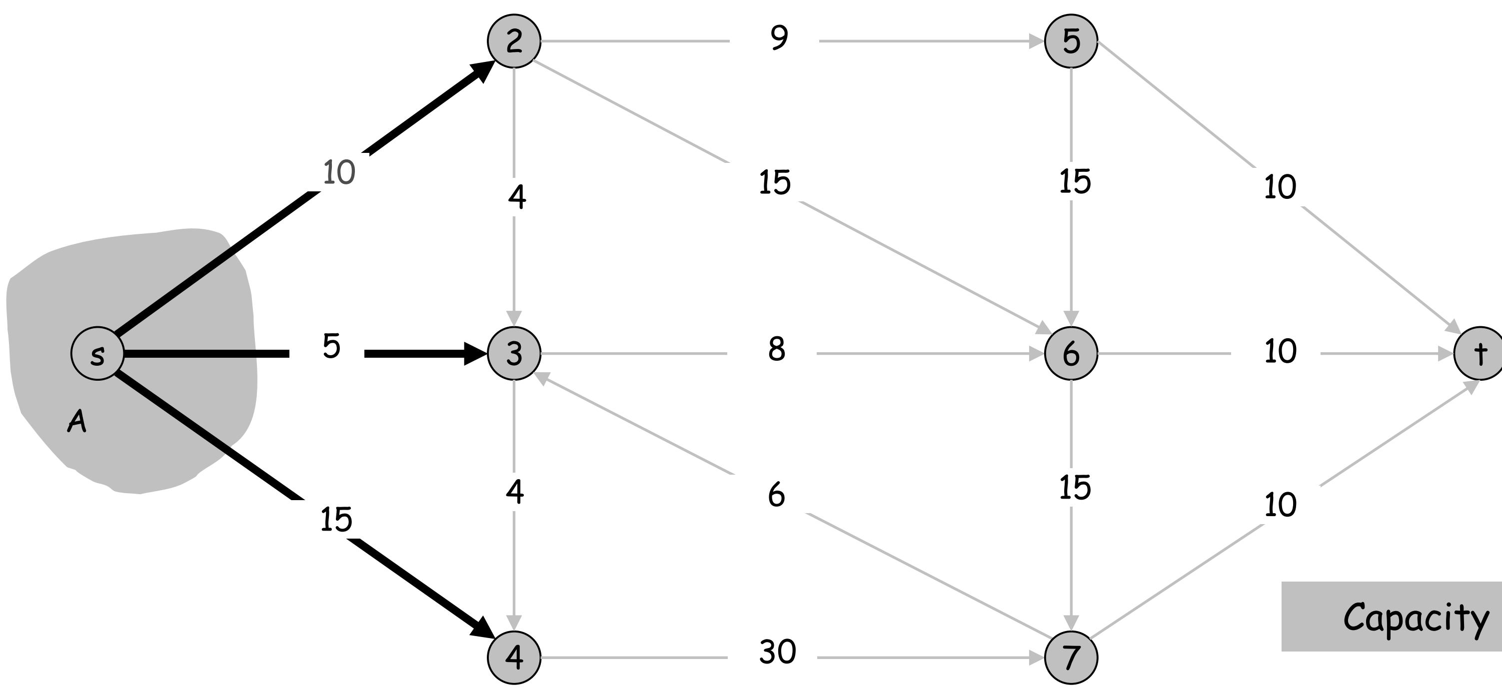
$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } s} f(e) \\
&= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right) \\
&= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).
\end{aligned}
$$

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

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



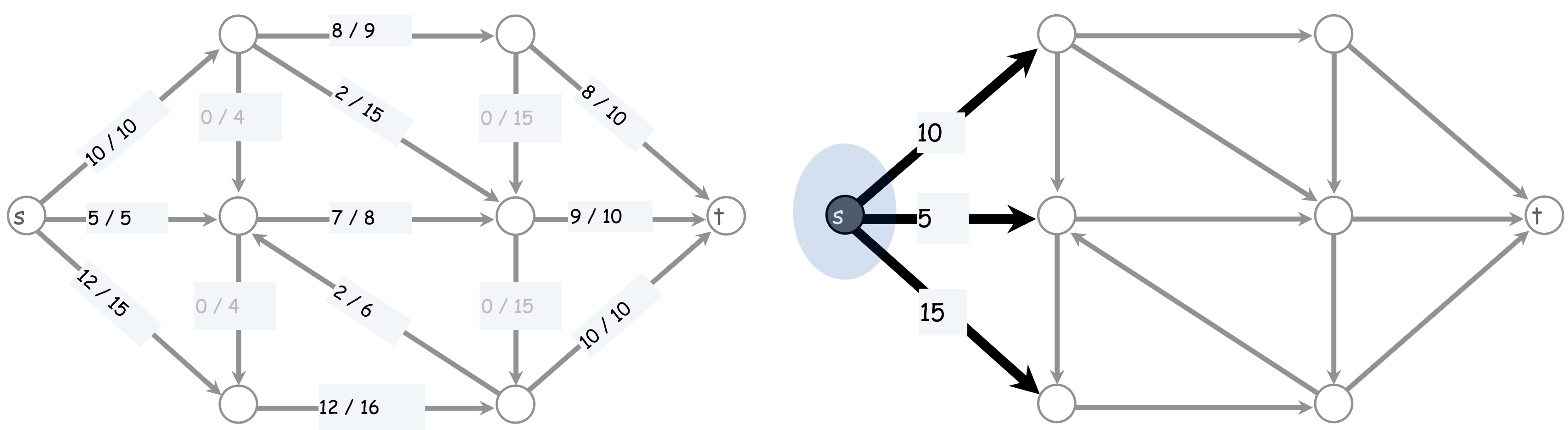
Capacity = 30

Relationship between flows and cuts

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

Pf.

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

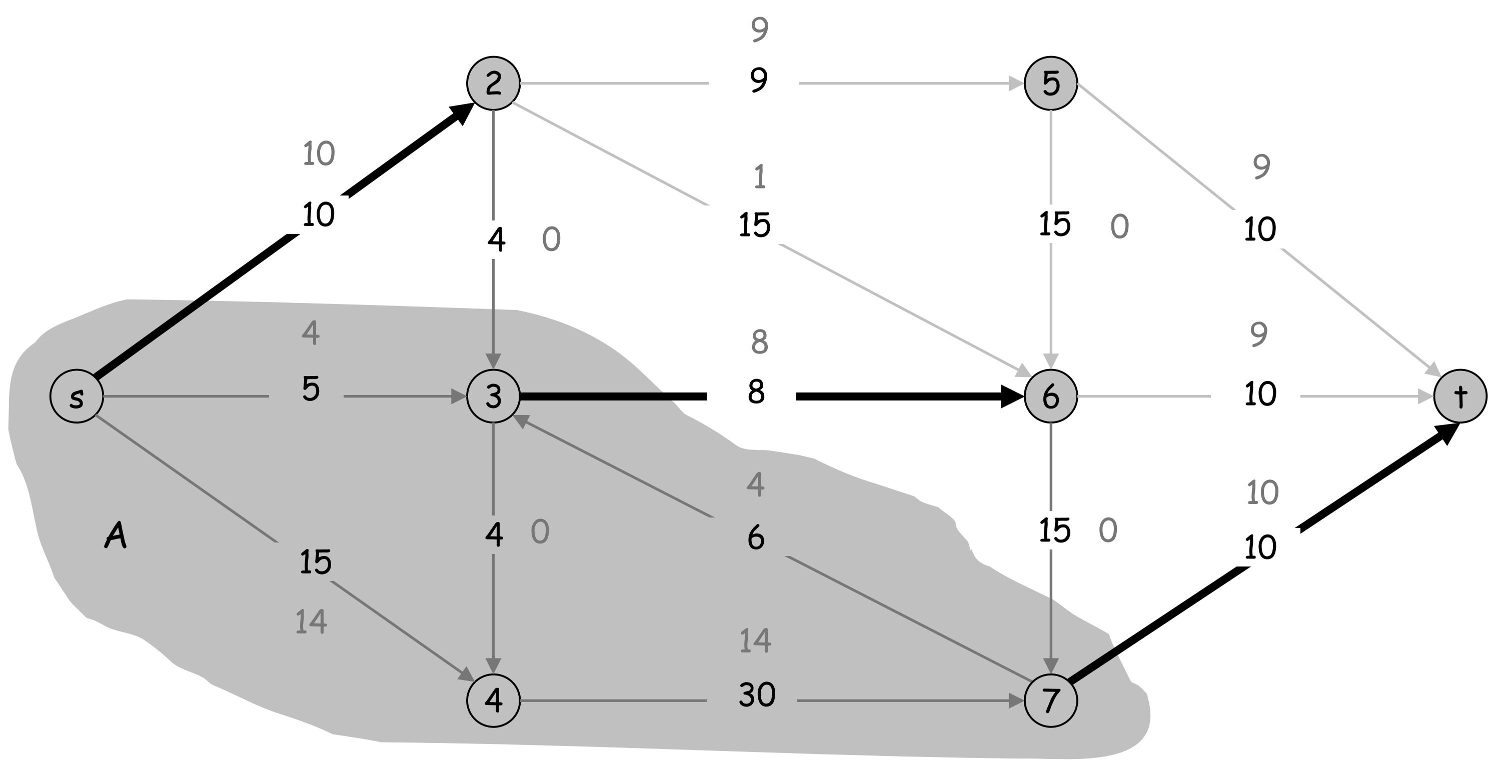


value of flow = 27 $\leqslant$ capacity of cut = 30

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.

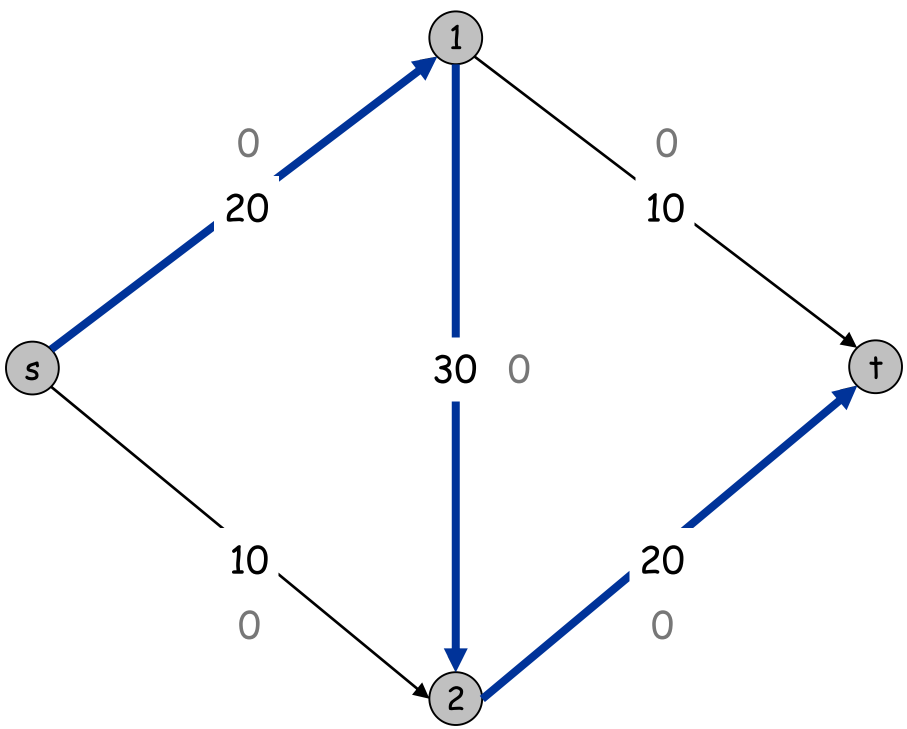
Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



Towards a Max Flow Algorithm

Greedy algorithm.

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

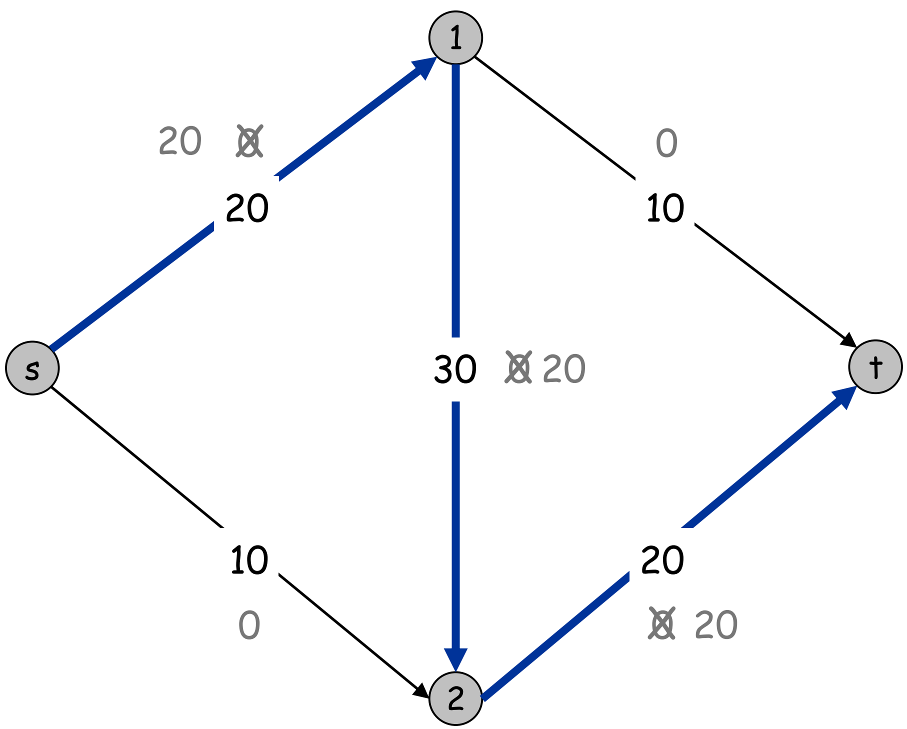


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

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

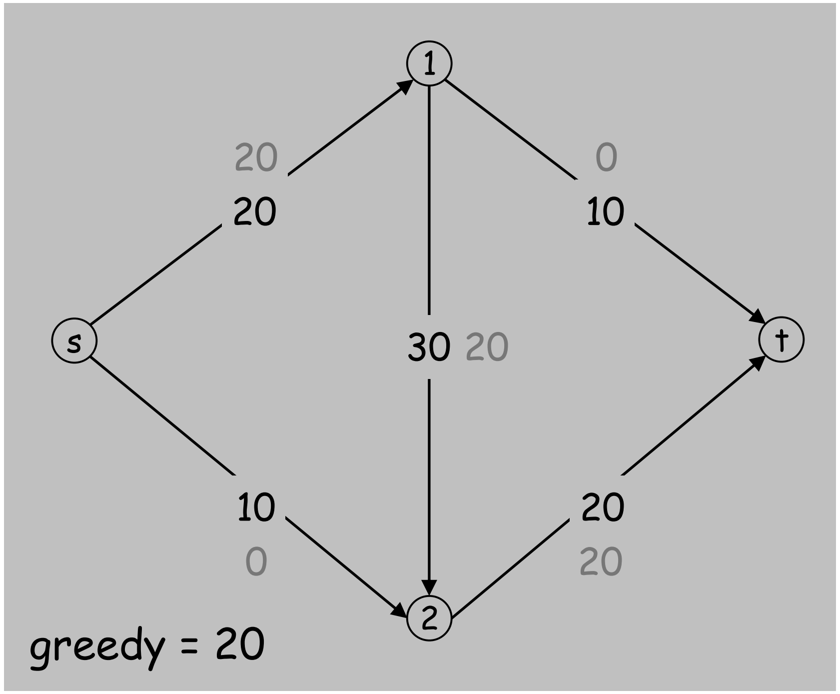


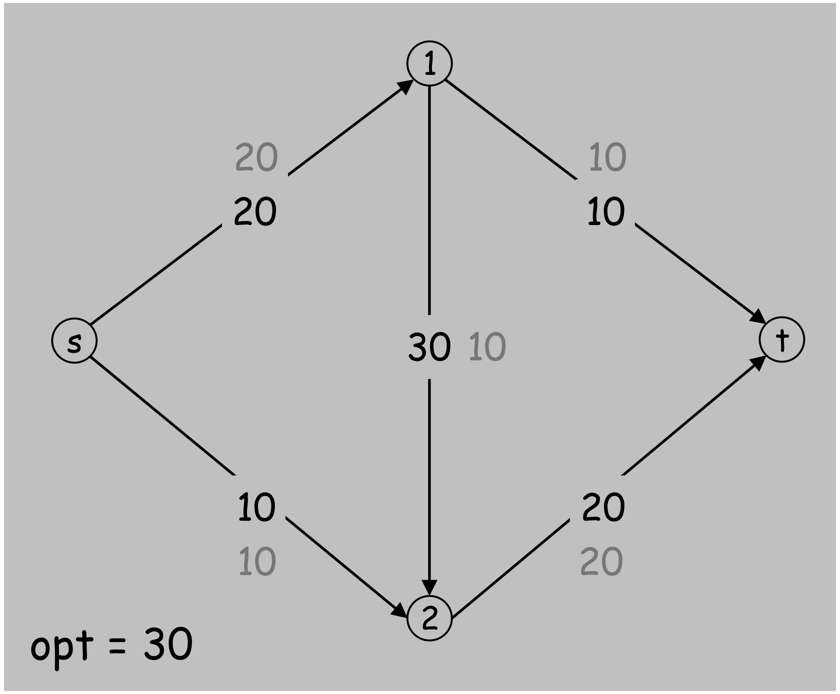
Flow value = 20

Towards a Max Flow Algorithm

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

locally optimality $\not\Rightarrow$ global optimality

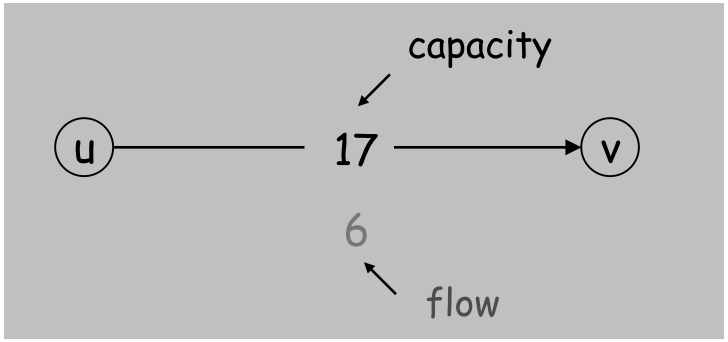




Residual Graph

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

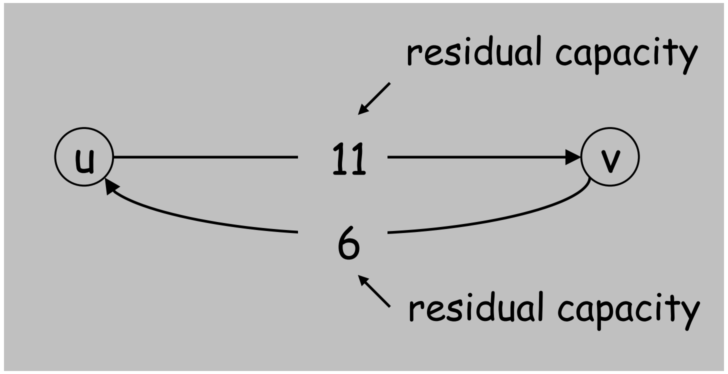
- Flow $f(e)$, capacity $c(e)$.



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.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Augmenting path

Def. An augmenting path is a simple $s \leadsto t$ 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 AUGMENT, the resulting f' is a flow and $v(f') = v(f) + bottleneck(G_f, P)$.

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 $f[e^{\text{reverse}}] \leftarrow f[e^{\text{reverse}}] - b$.

RETURN f.

Ford-Fulkerson Algorithm

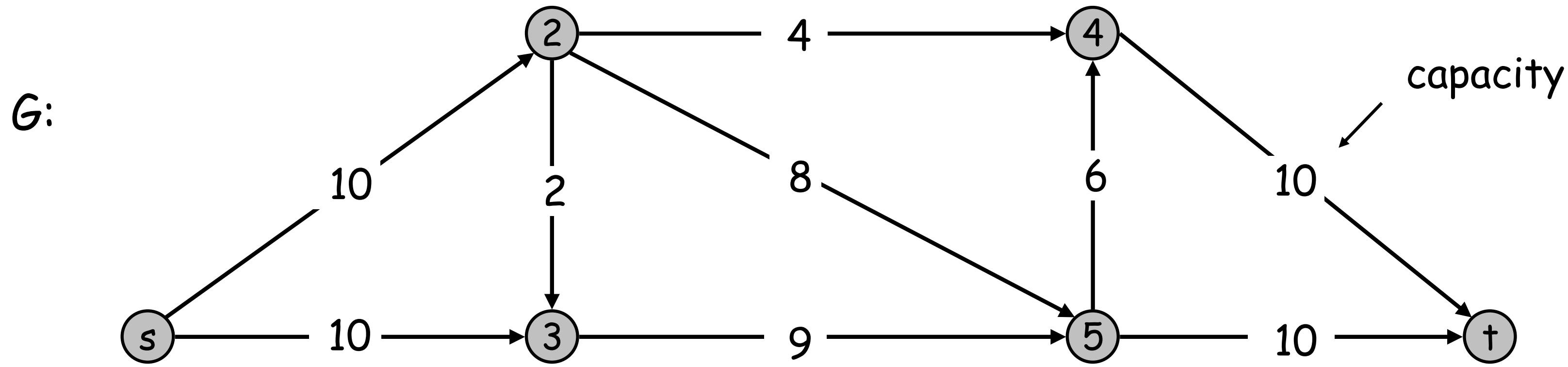
Ford-Fulkerson augmenting path algorithm.

- Start with f(e) = 0 for each edge e ∈ E.
- Find an s↝t path P in the residual network G_f
- Augment flow along path P.
- Repeat until you get stuck.

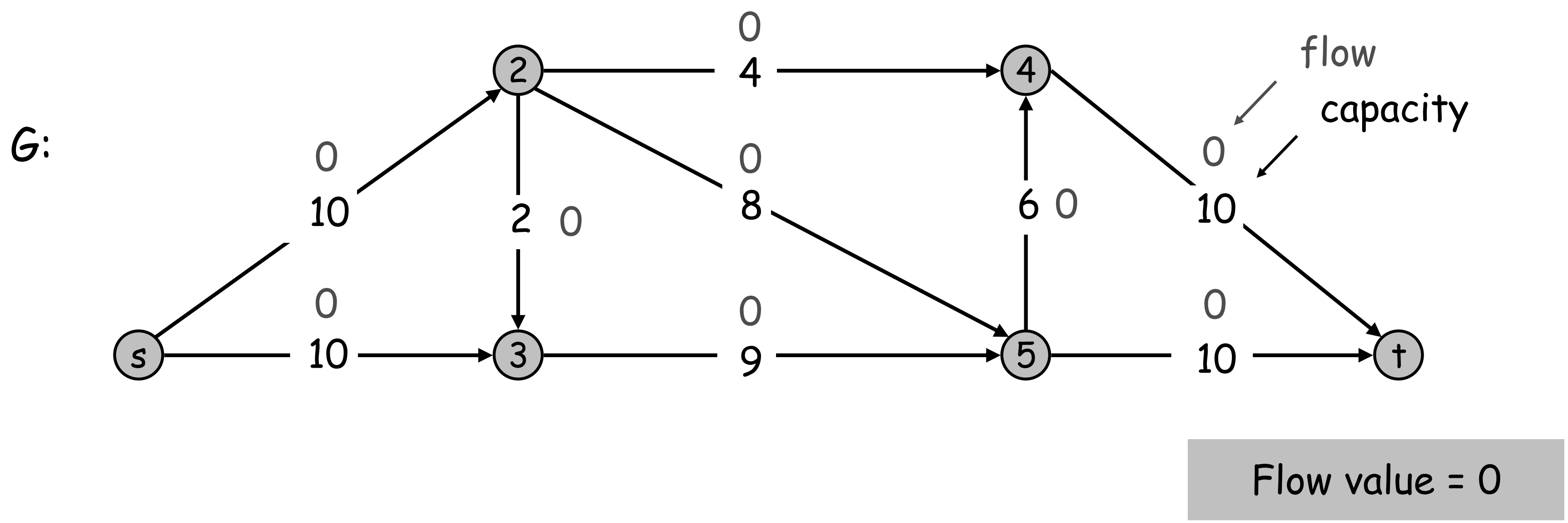
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

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

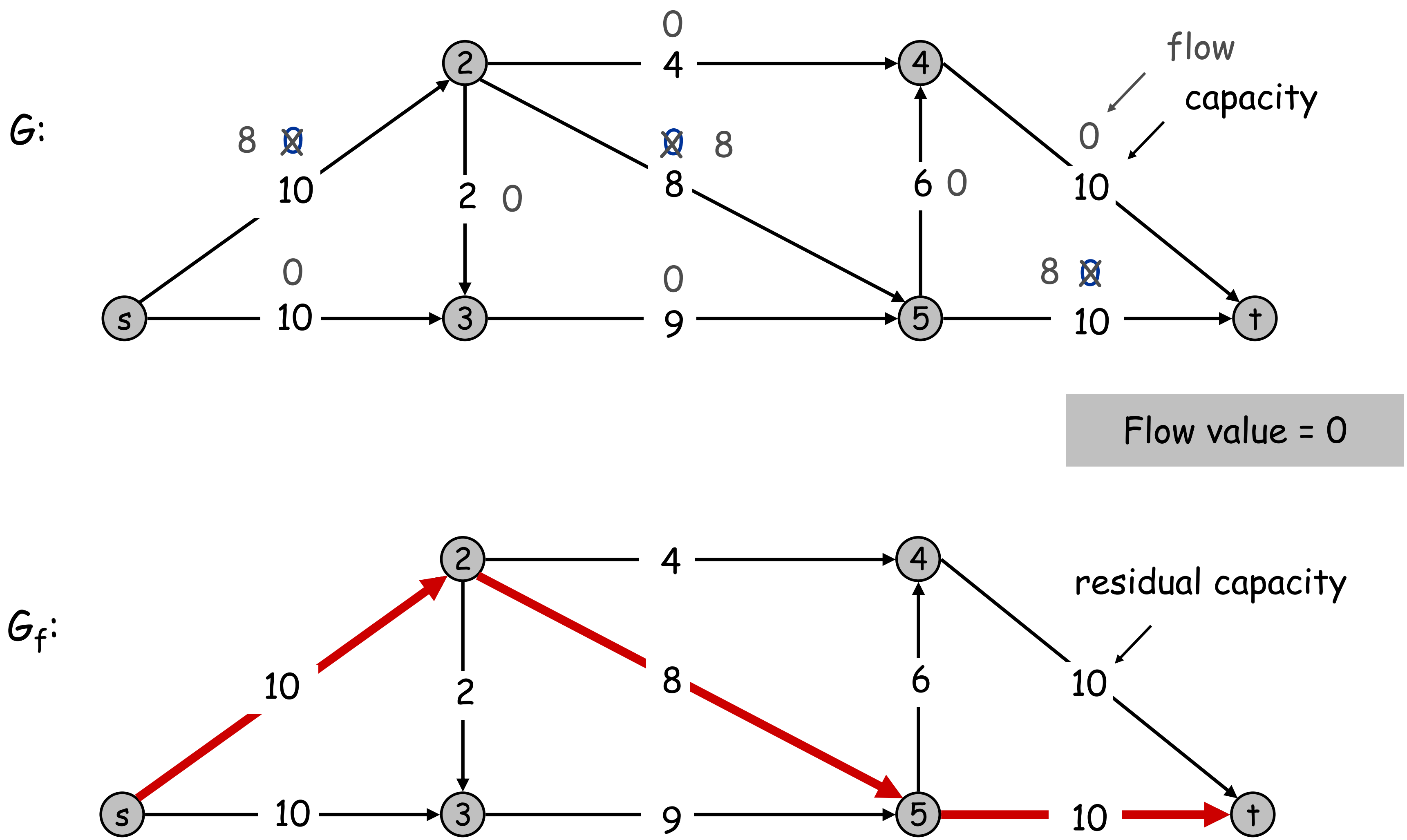
Ford-Fulkerson Algorithm



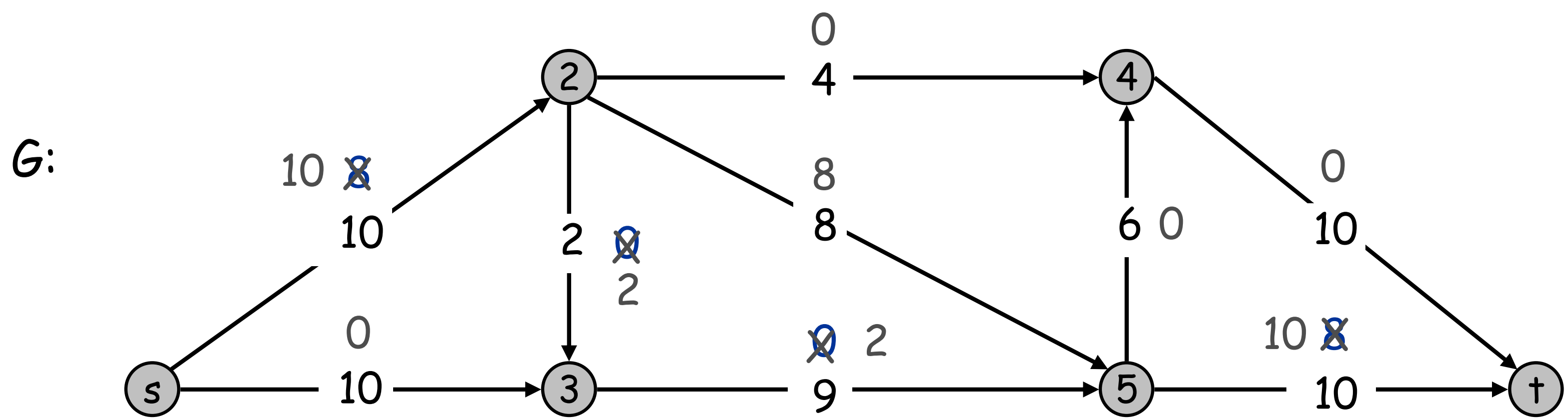
Ford-Fulkerson Algorithm



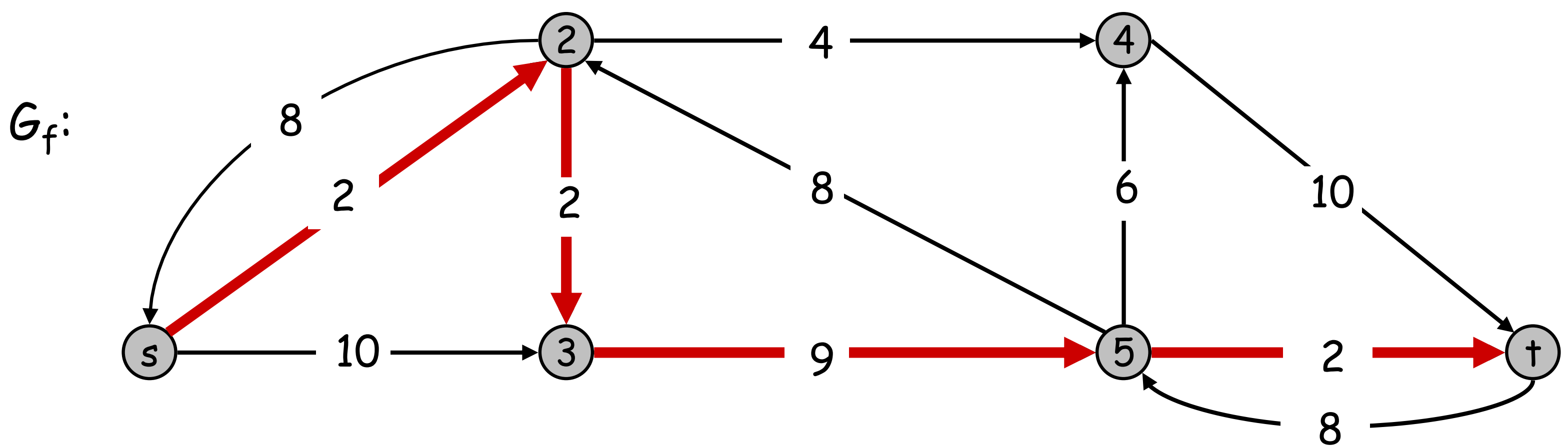
Ford-Fulkerson Algorithm



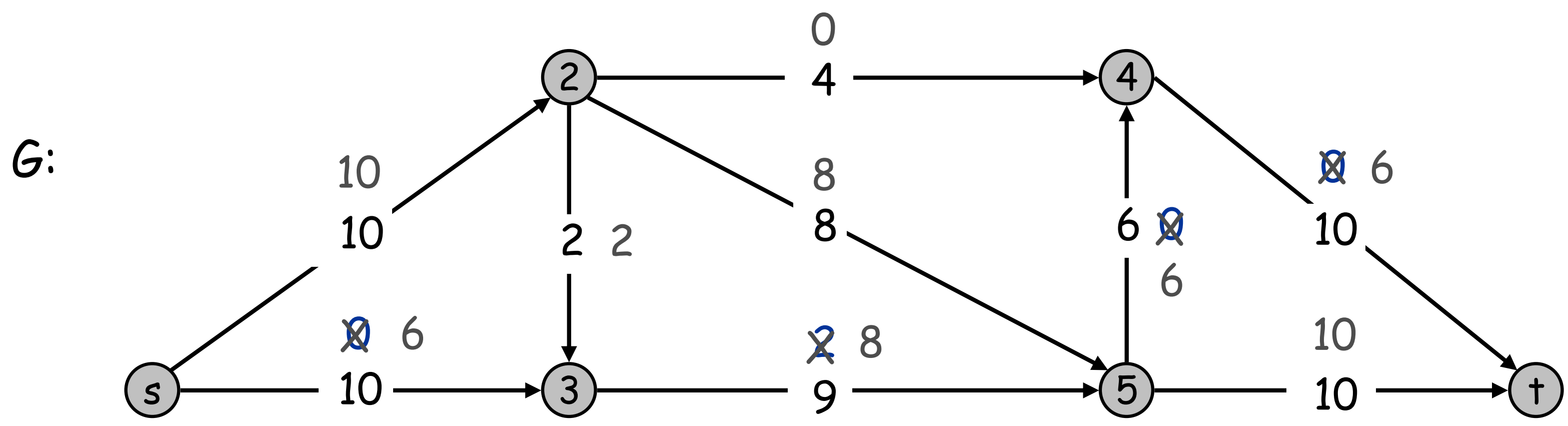
Ford-Fulkerson Algorithm



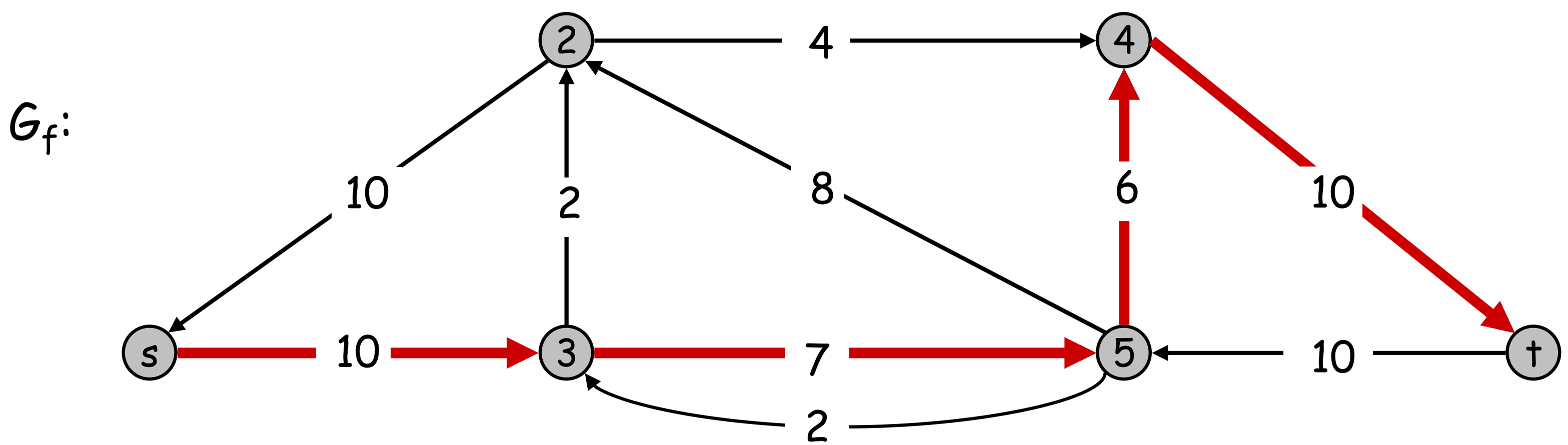
Flow value = 8



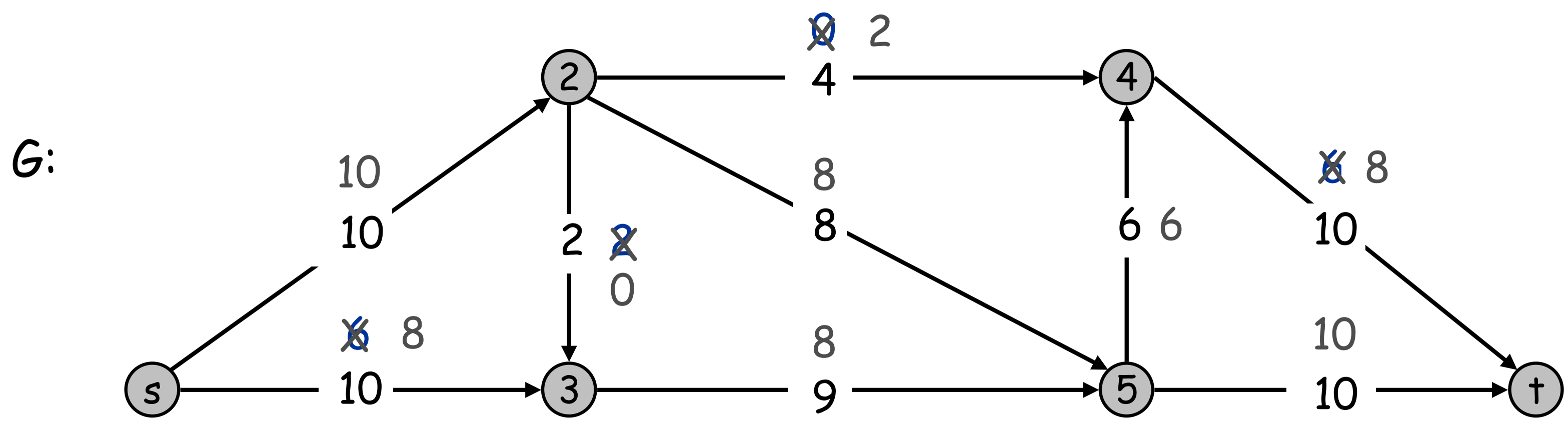
Ford-Fulkerson Algorithm



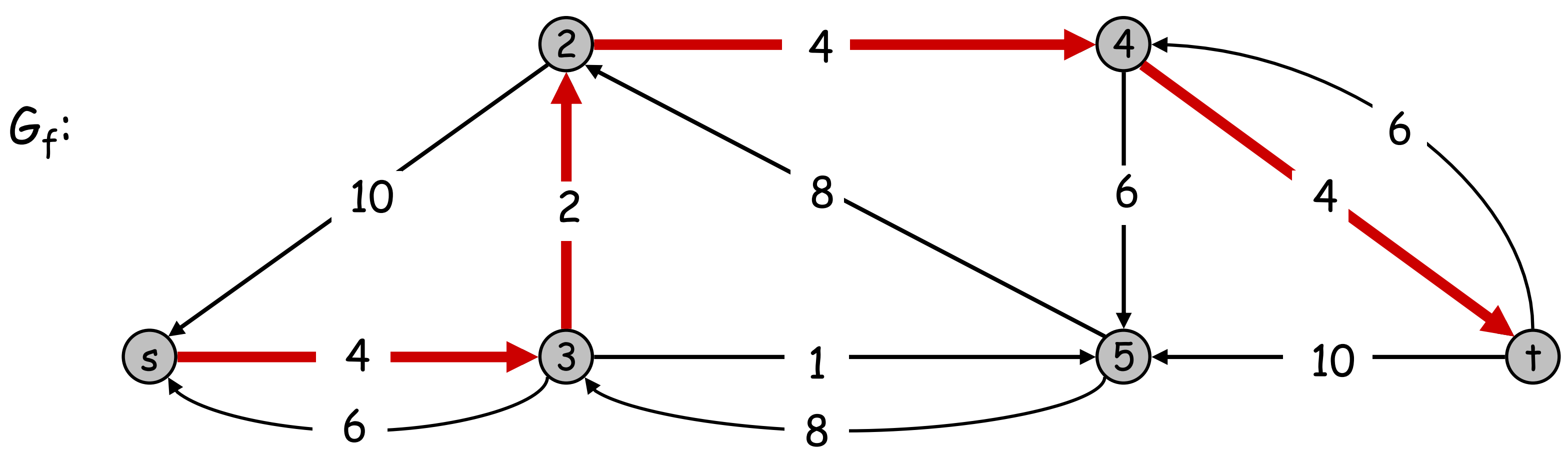
Flow value = 10



Ford-Fulkerson Algorithm

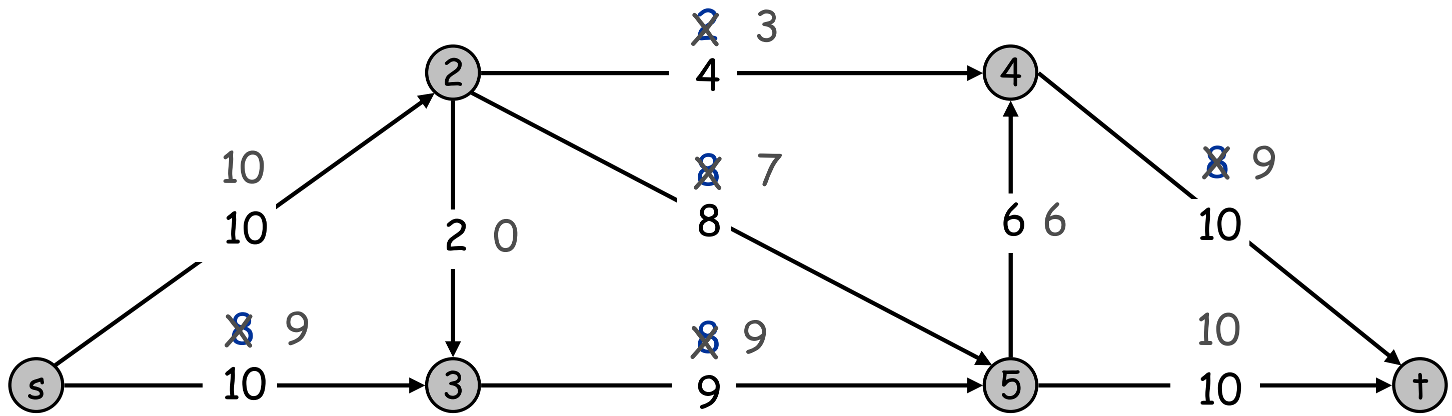


Flow value = 16



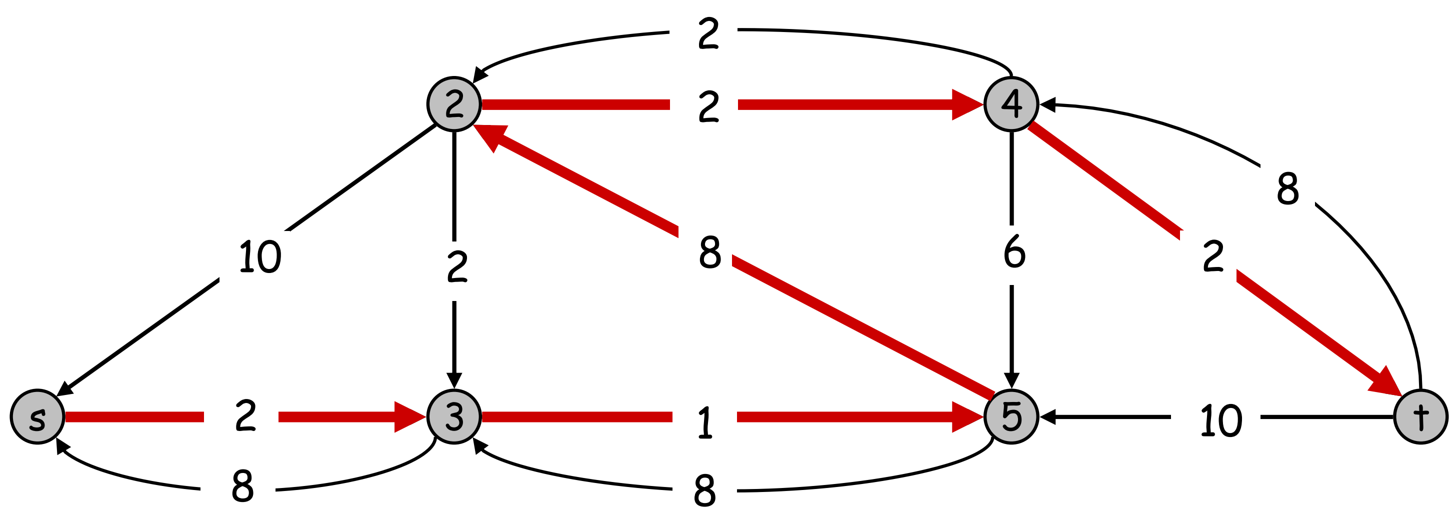
Ford-Fulkerson Algorithm

G:

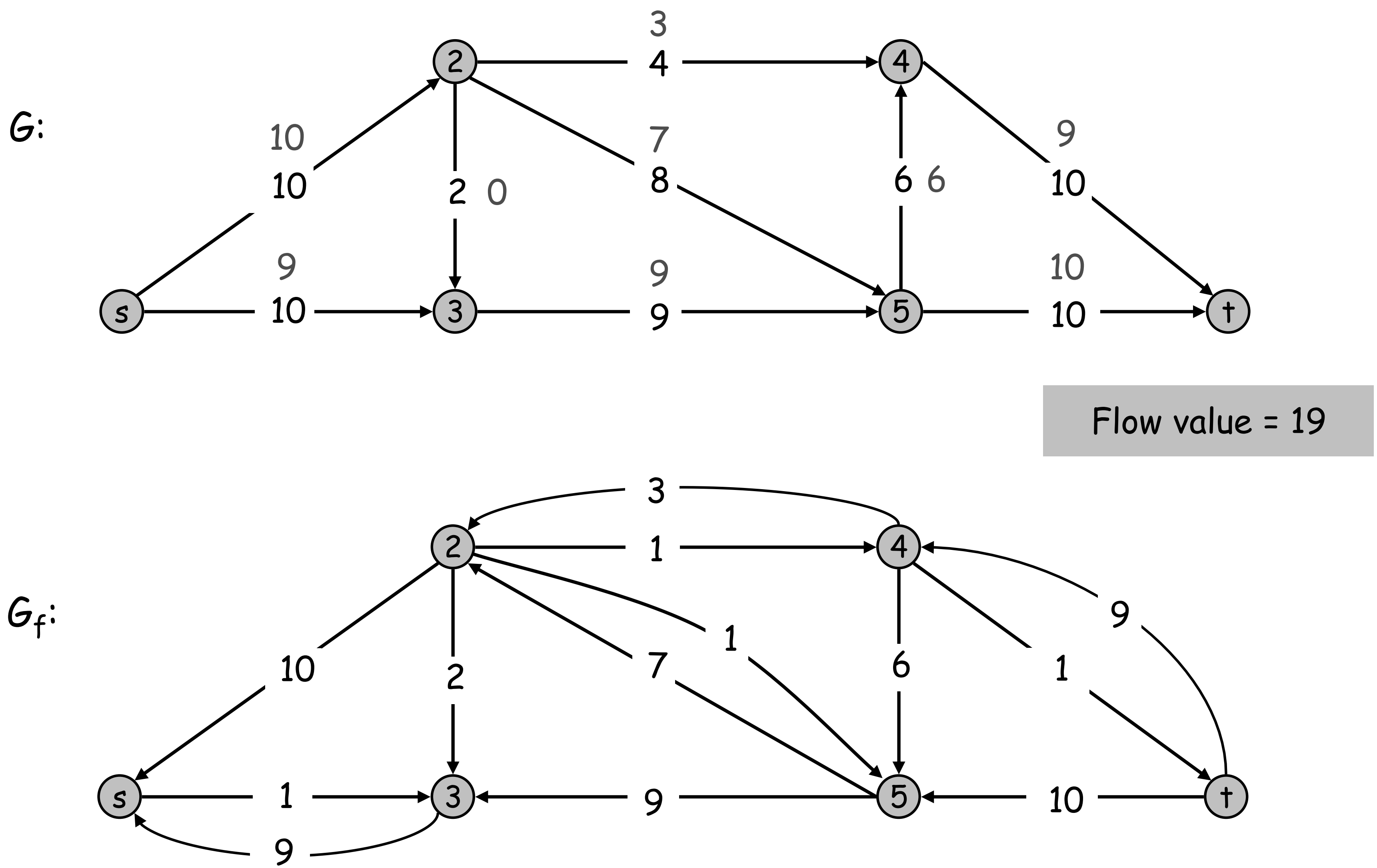


Flow value = 18

G_f:



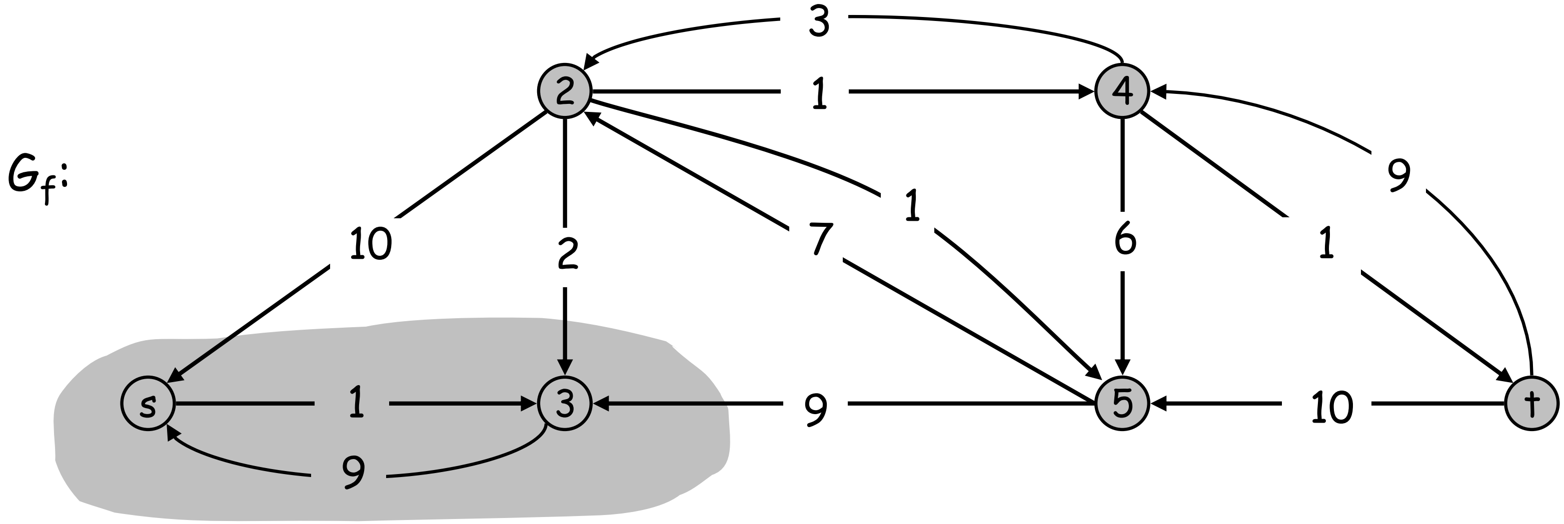
Ford-Fulkerson Algorithm



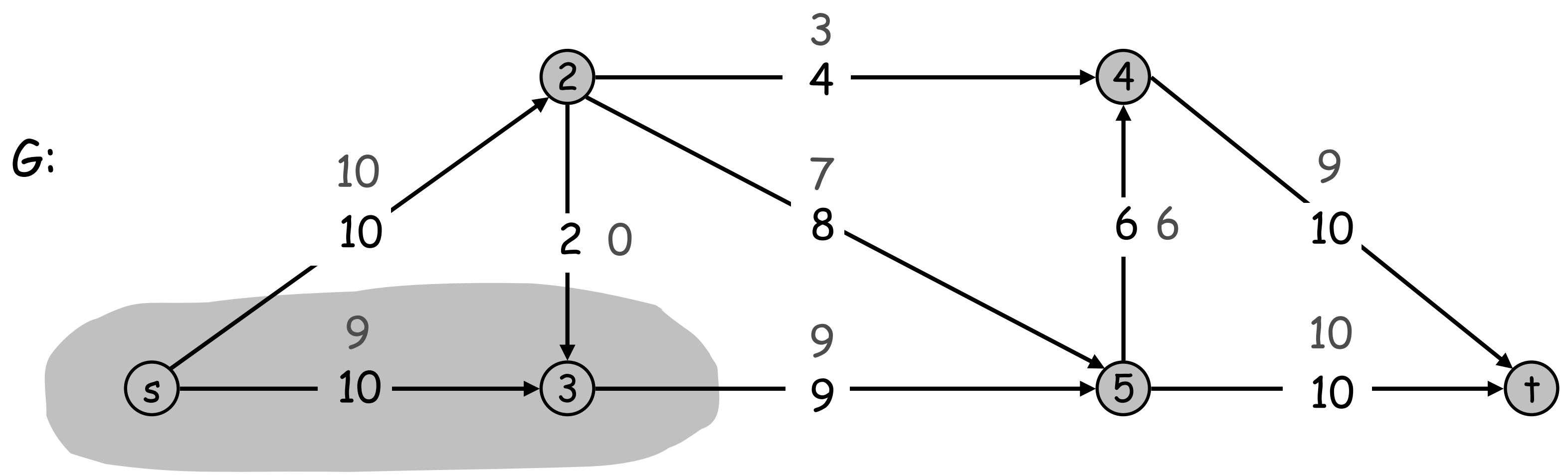
Computing a minimum cut from a maximum flow

To compute a min cut (A, B) from a max flow f^* :

- By augmenting path theorem, no augmenting paths in G_{f^*}.
- Compute A = set of nodes reachable from s in G_{f^*}.

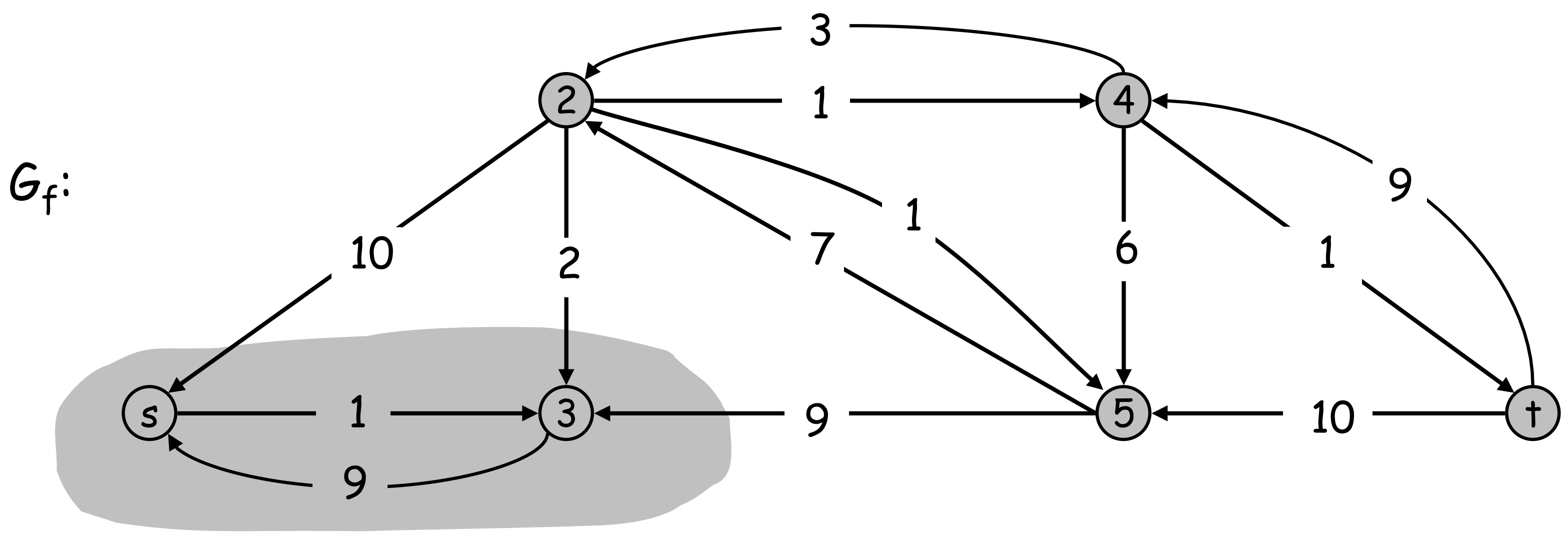


Ford-Fulkerson Algorithm



Cut capacity = 19

Flow value = 19



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. The following three conditions are equivalent for any flow f

(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.

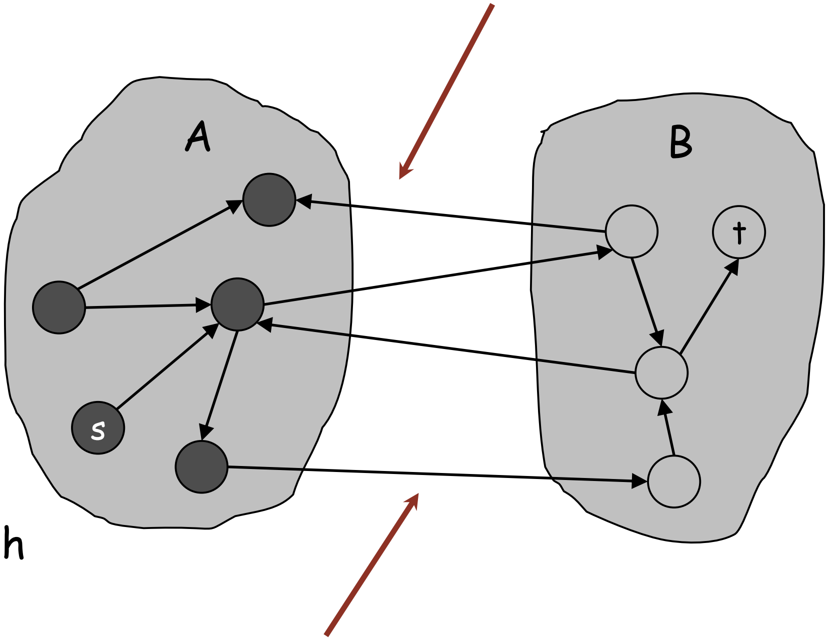
Proof of Max-Flow Min-Cut Theorem

(iii) $\Rightarrow$ (i)

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

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

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



original graph

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

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. ▪

Ford-Fulkerson Algorithm

Ford-Fulkerson augmenting path algorithm can be implemented to run in O(mnC) time.

- Start with f(e) = 0 for each edge $e \in E$.
- Find an s⤳t path P in the residual network G_f
- Augment flow along path P.
- Repeat until you get stuck.

O(nC)

```
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
}
```

O(m+n)

O(n)

O(m)

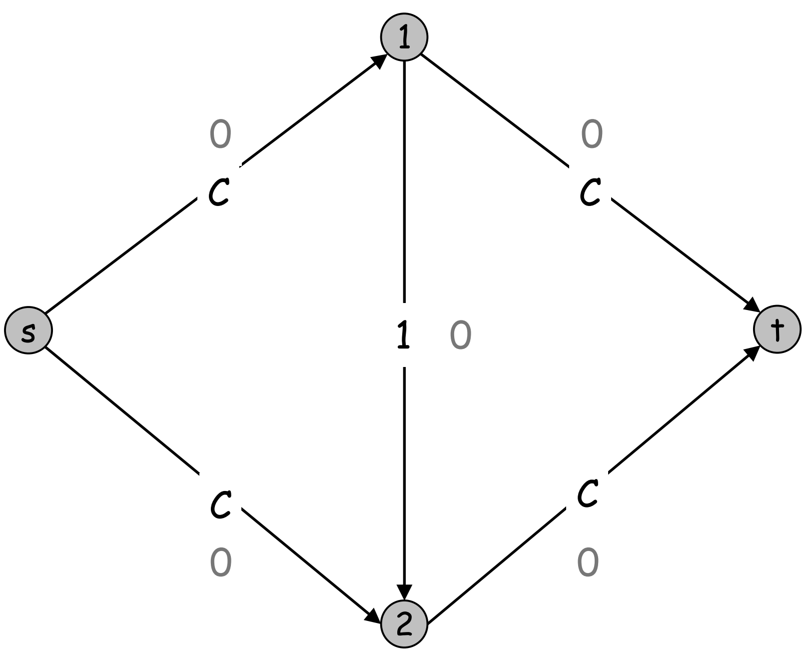
O(m+n) => O(m) since each node has an incident edge, then m ≥ n/2

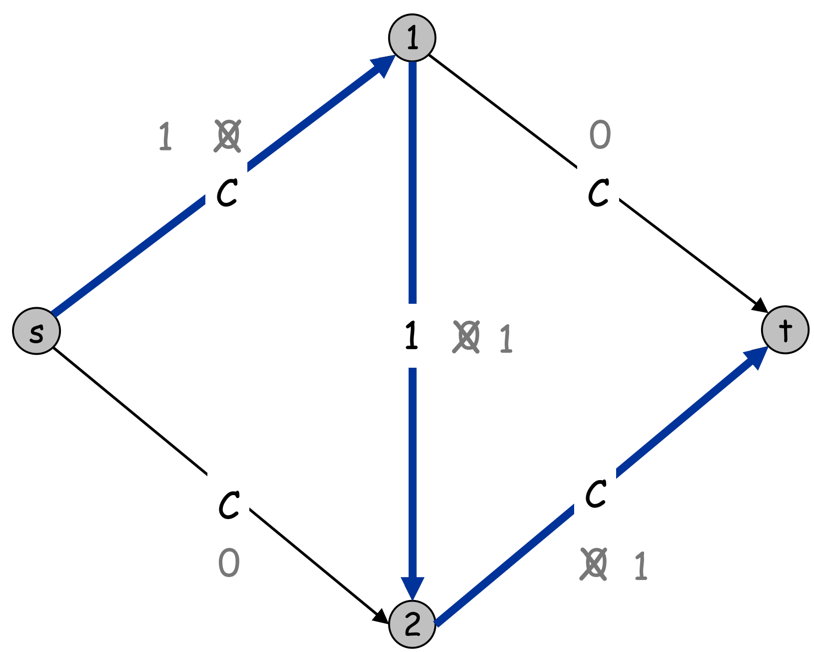
Ford-Fulkerson: Exponential Number of Augmentations

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.



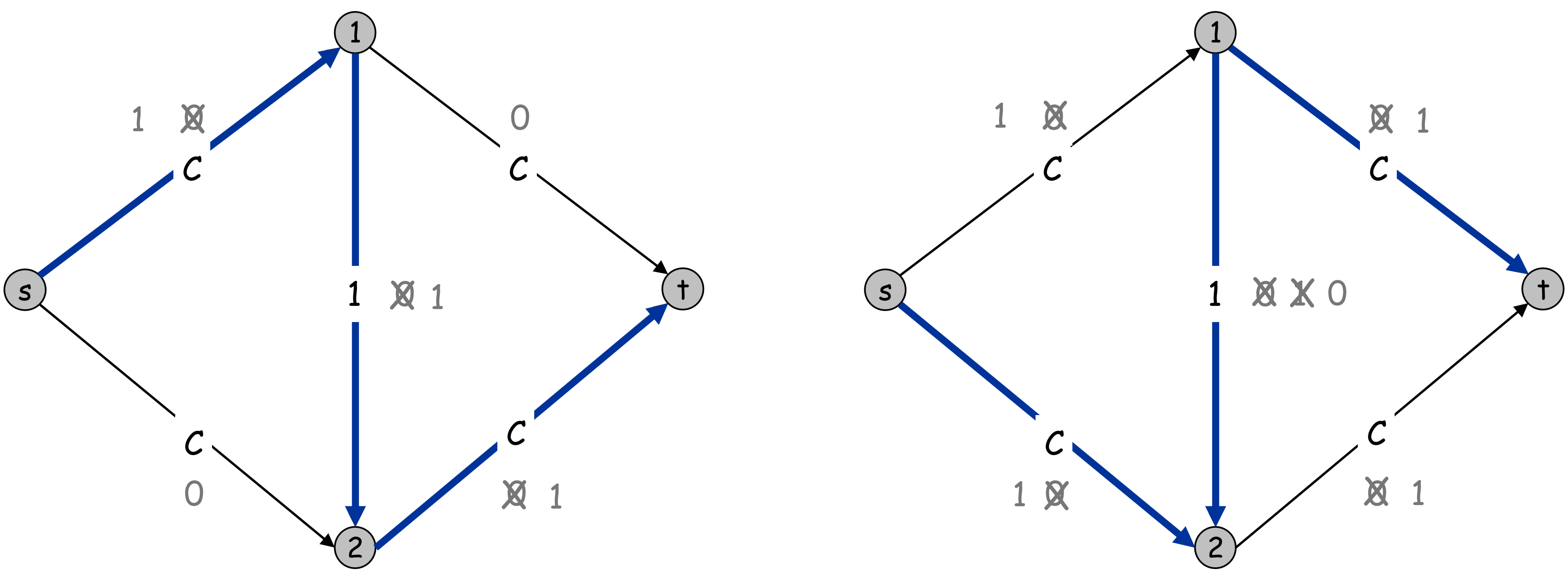


Ford-Fulkerson: Exponential Number of Augmentations

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.

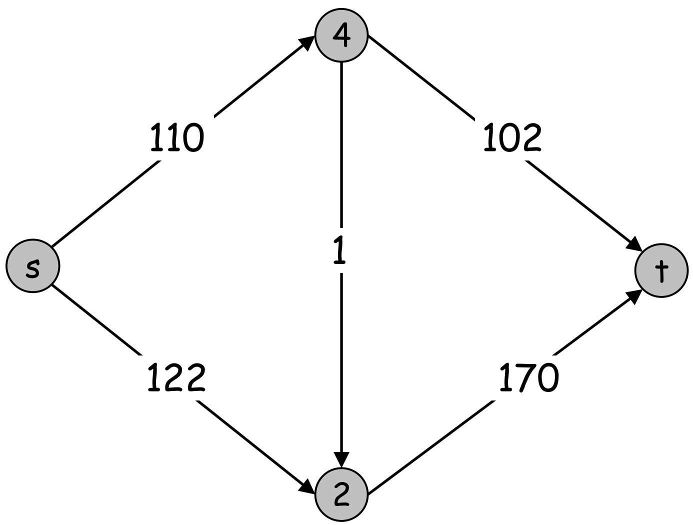
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

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

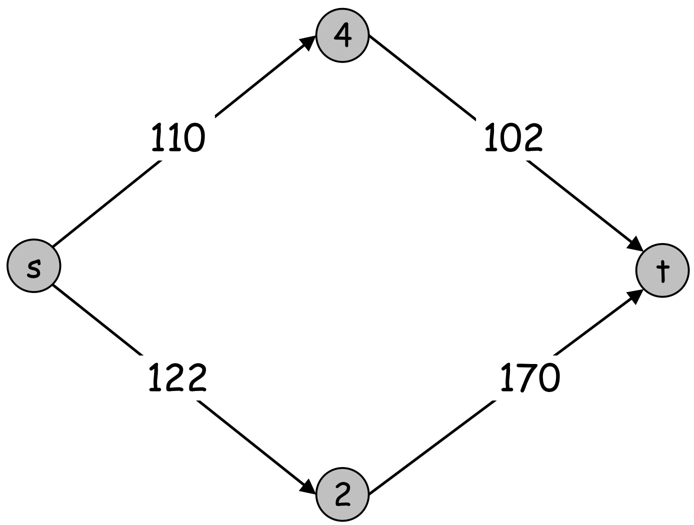
Capacity Scaling

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 arcs with capacity at least Δ.



G_f



$G_f\,(100)$

Capacity Scaling

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 $G_f(\Delta)$.

$\Delta \leftarrow \Delta / 2$.

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 $C/2 < \Delta \leq C$. 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. ▪

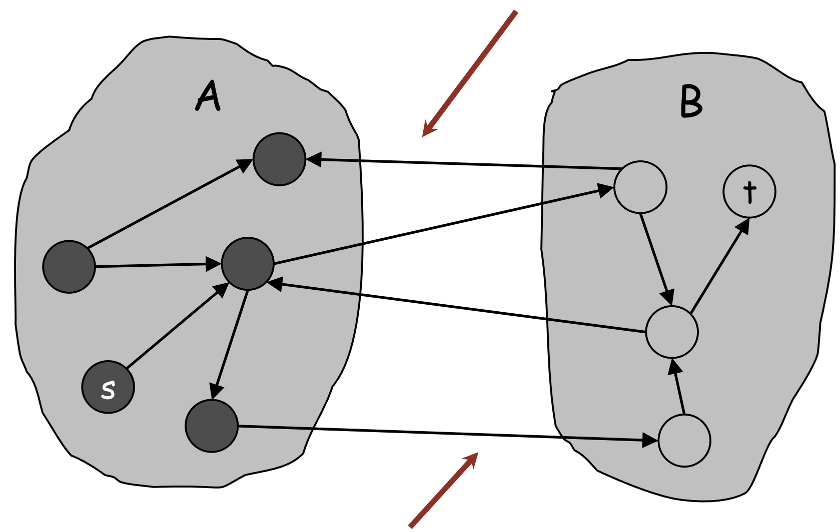
Capacity Scaling: Running Time

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 $cap(A, B) \le v(f) + m\,\Delta$.
- 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, $t \notin A$.

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



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

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