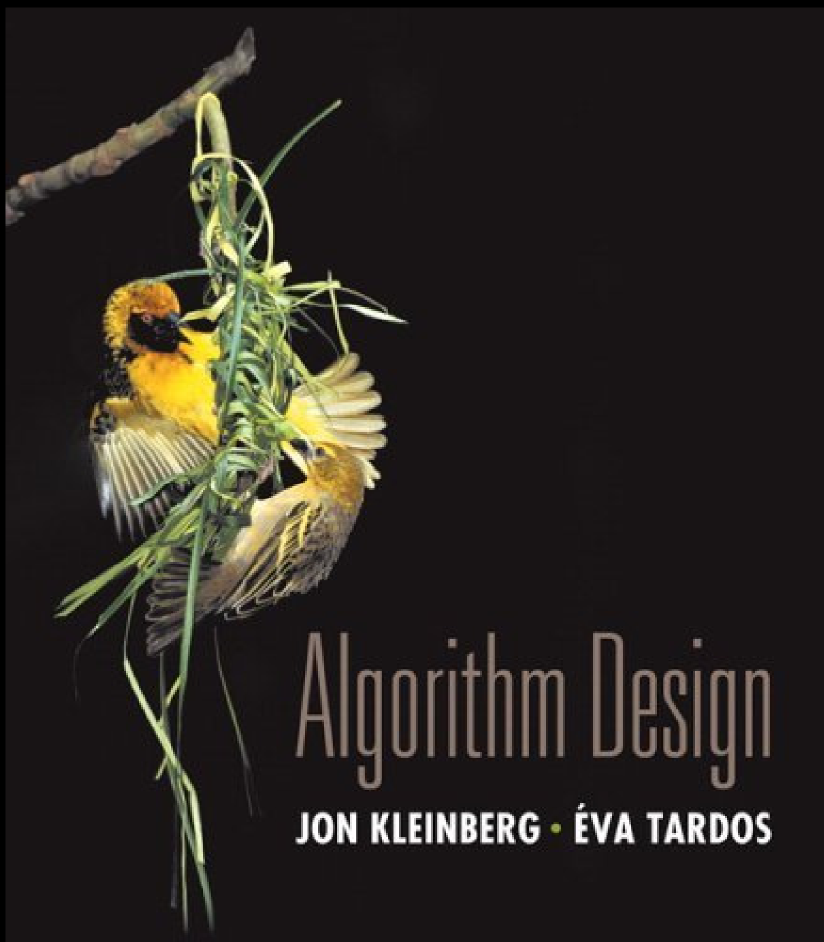


Network Flow

Slides by Kevin Wayne.

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.

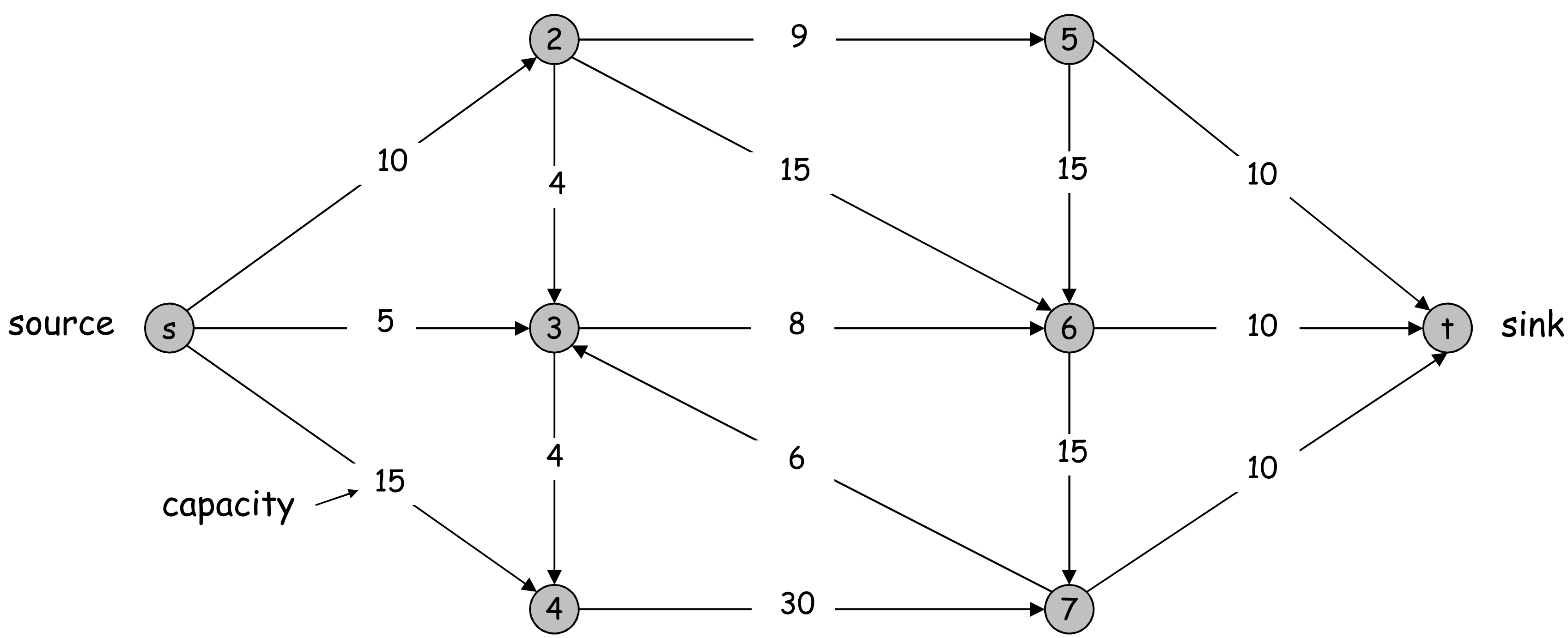
- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.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

7.1 Max-flow and Ford-Fulkerson Algorithm

Flows

Flow network.

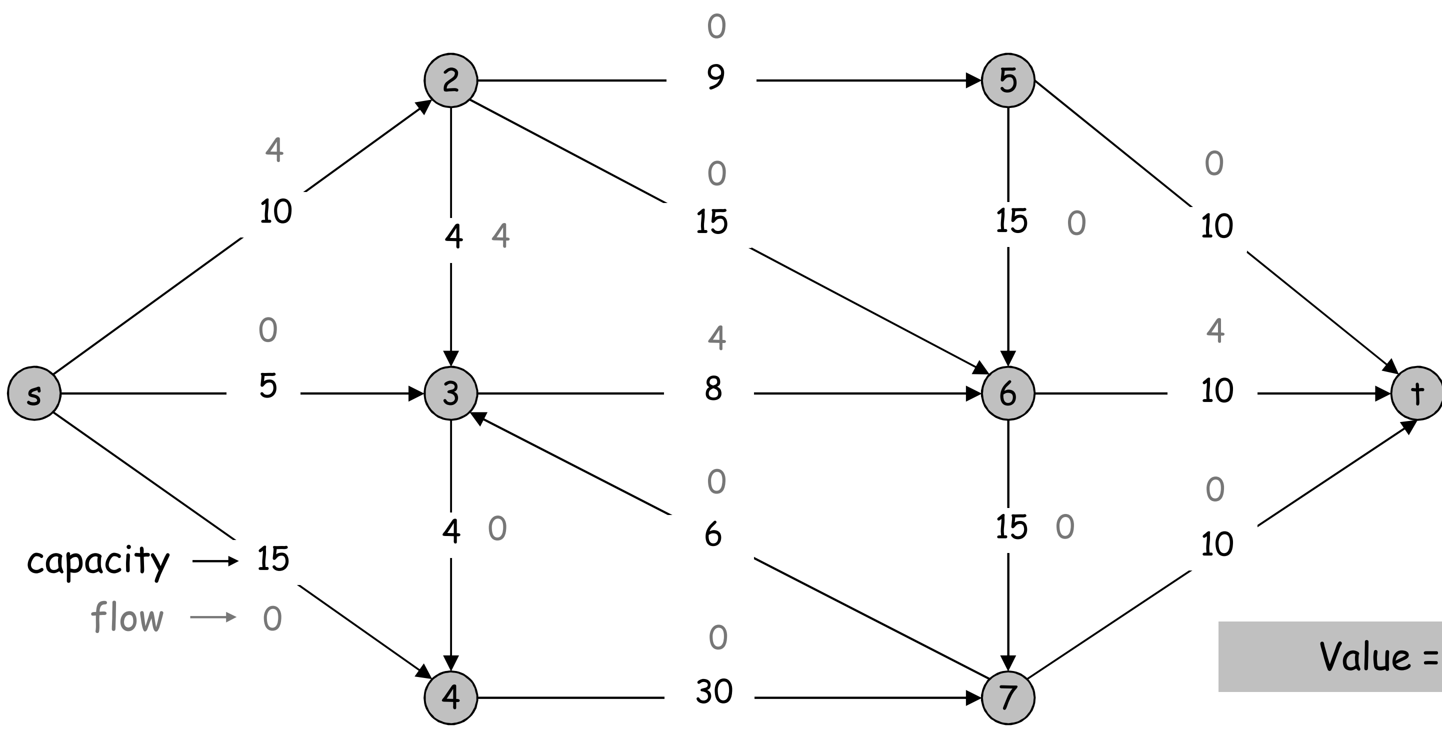
- Abstraction for material flowing through the edges.
- $G = (V, E)$ = directed graph
- Two distinguished nodes: s = source, t = sink.
- $c(e)$ = nonnegative capacity of edge e.



Flows

Def. An s-t flow is a function that satisfies:
- For each $e \in E$: $0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

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



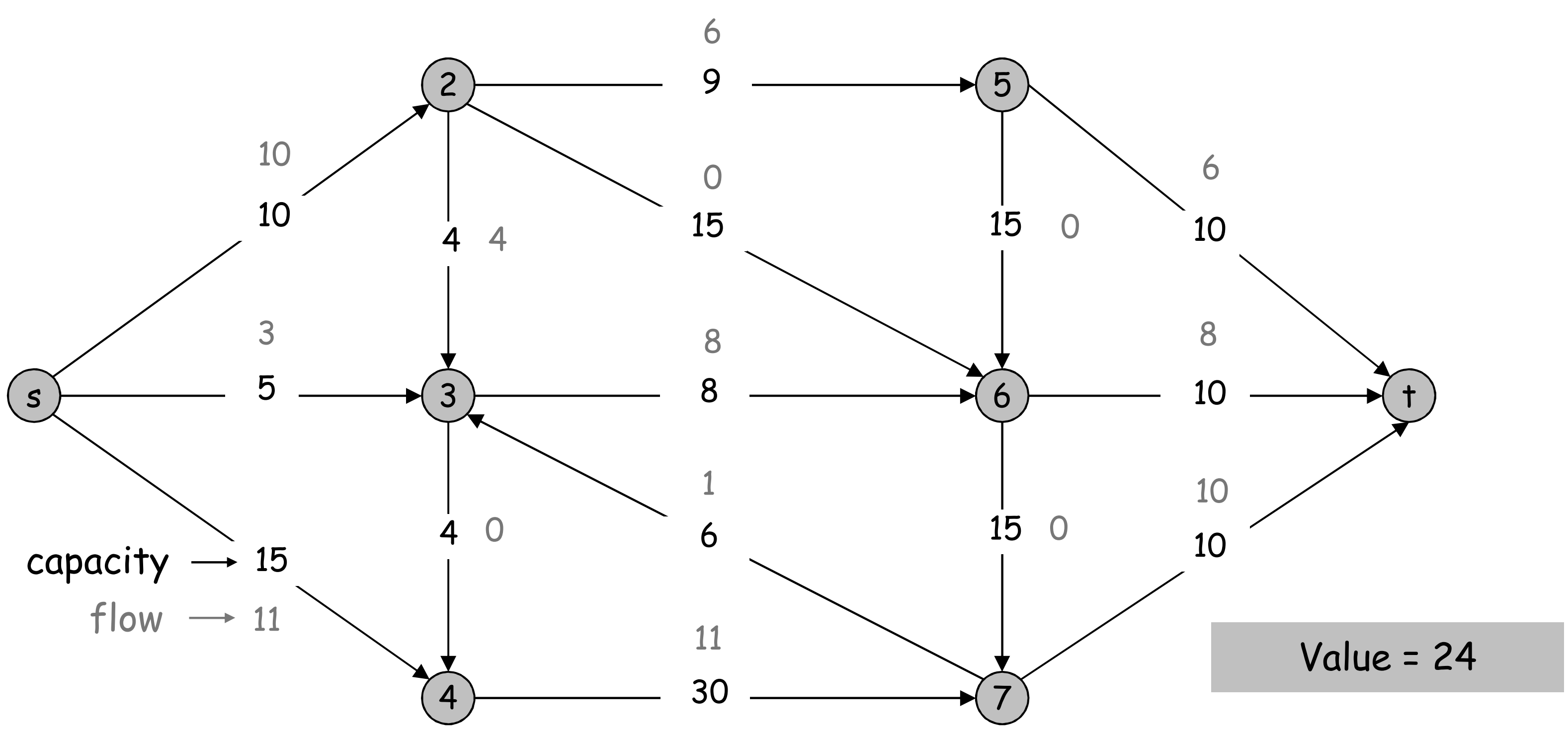
Value = 4

Flows

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

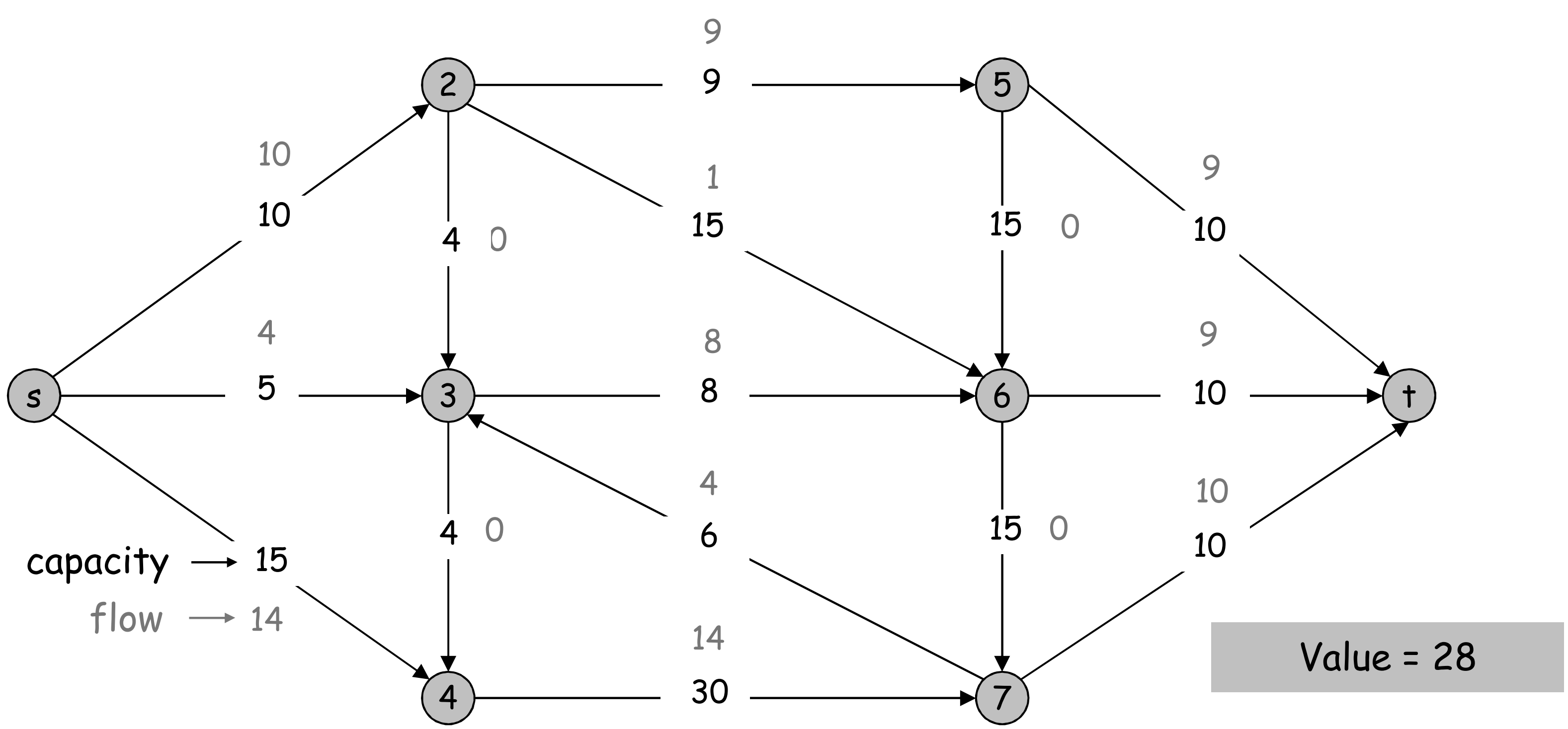
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

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



Maximum Flow Problem

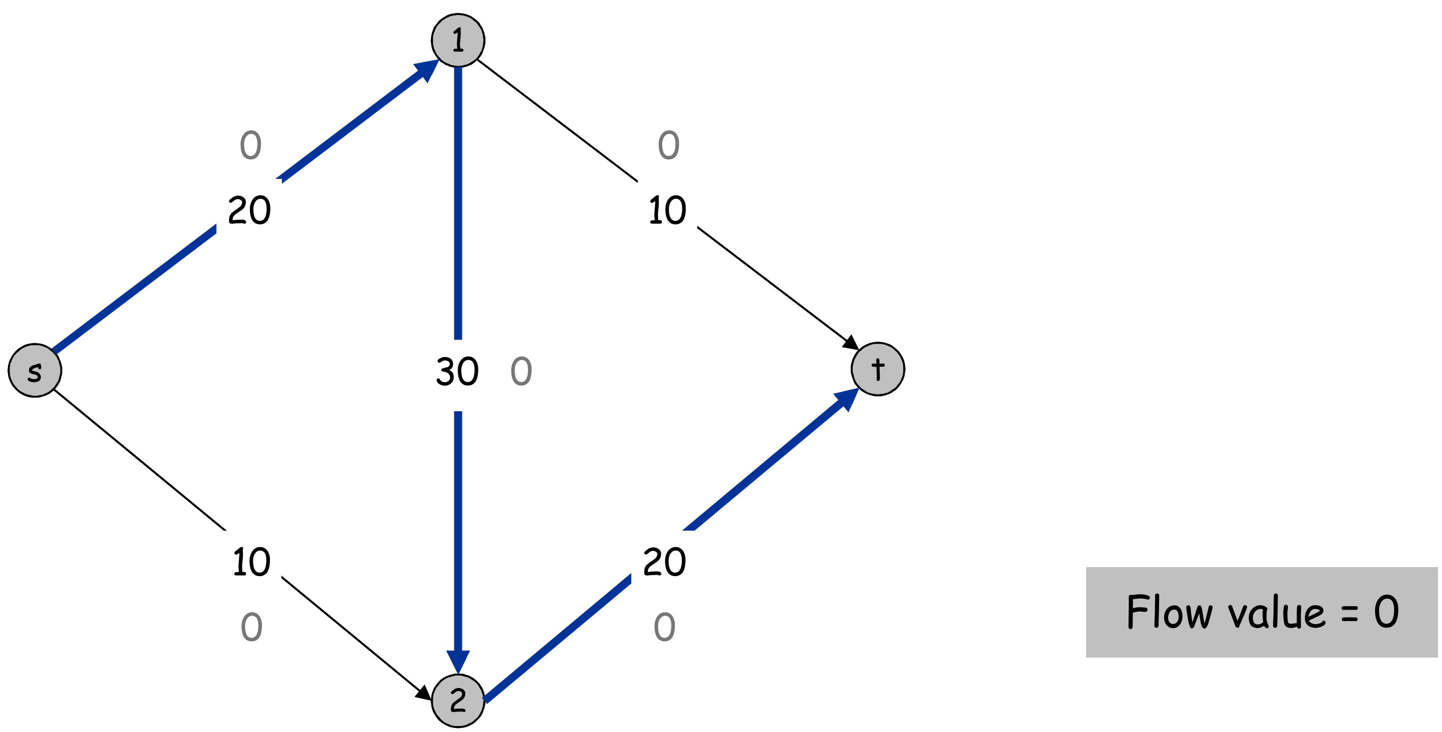
Max flow problem. Find s-t flow of maximum value.



Towards a Max Flow Algorithm

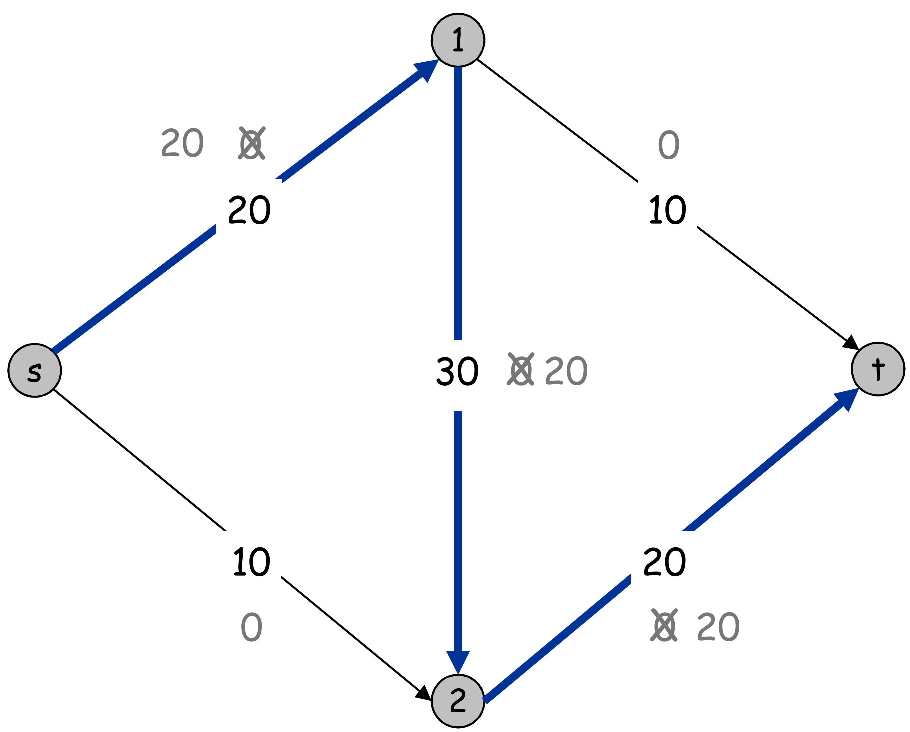
Greedy algorithm.

- 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.



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.

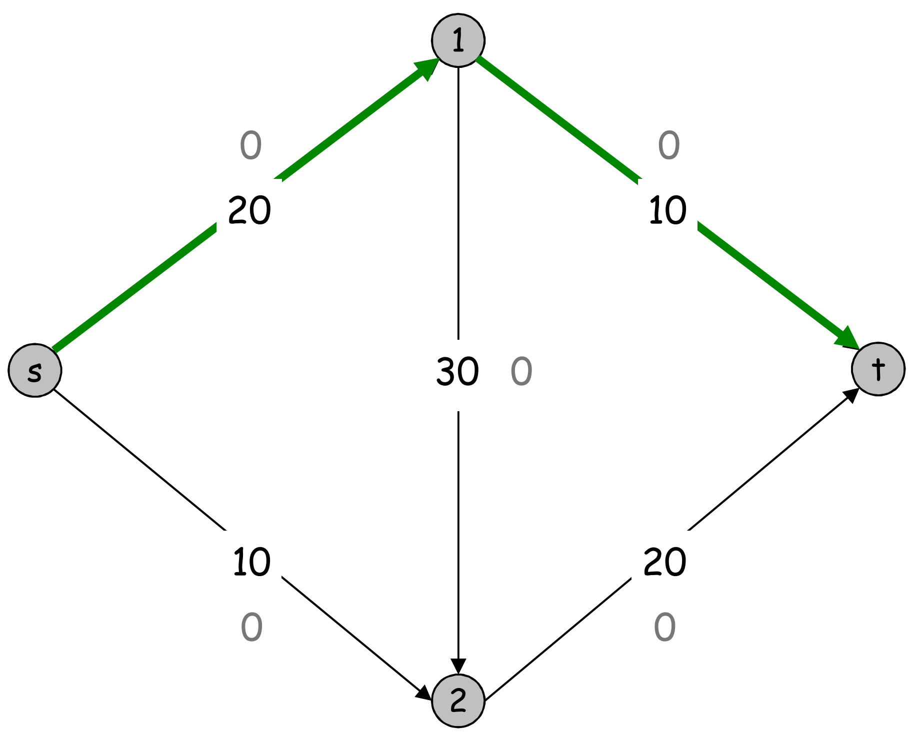


Flow value = 20

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.

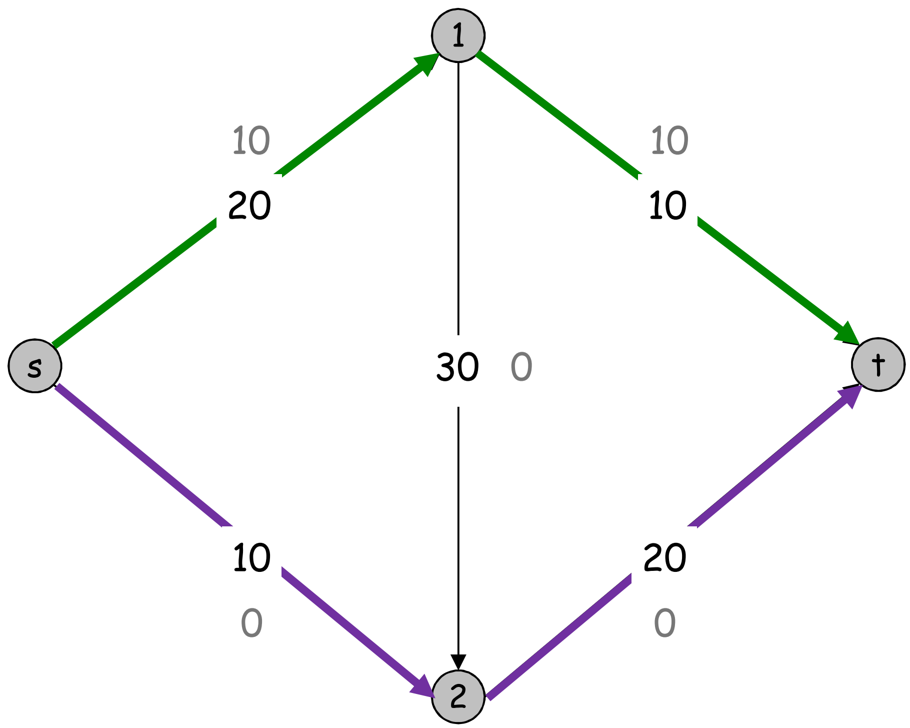


Flow value = 0

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.

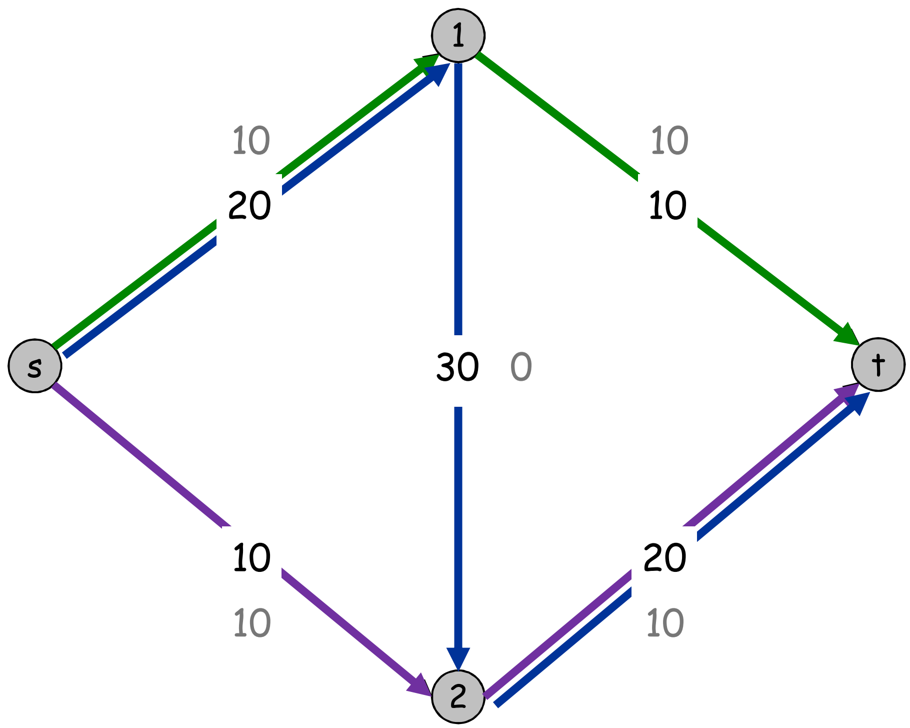


Flow value = 10

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.

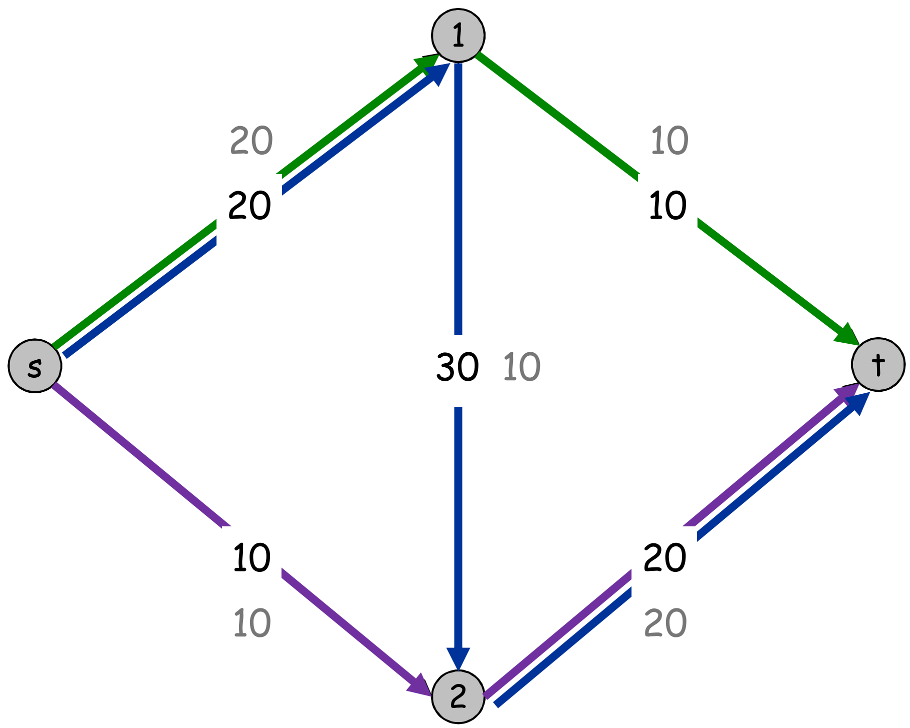


Flow value = 20

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.



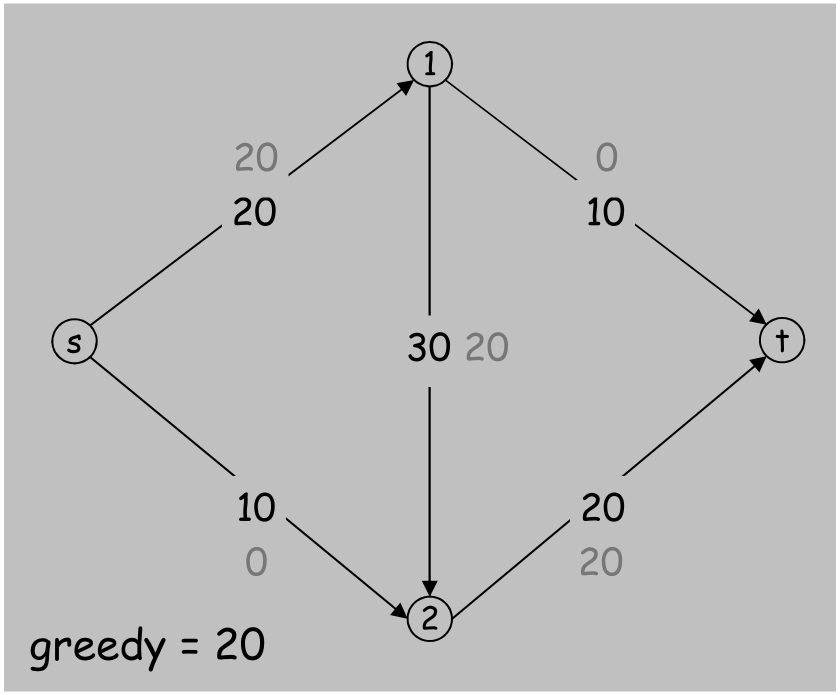
Flow value = 30

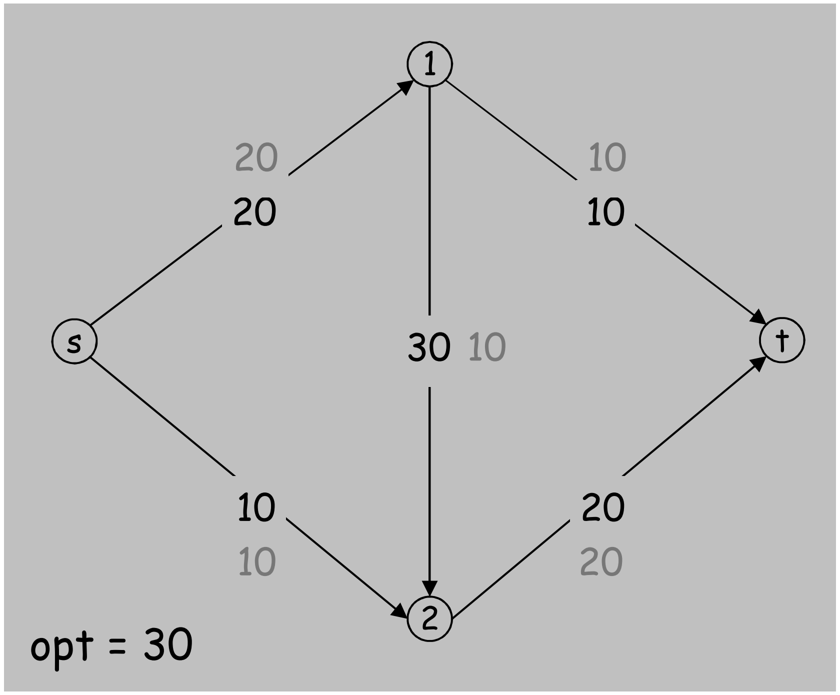
Towards a Max Flow Algorithm

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

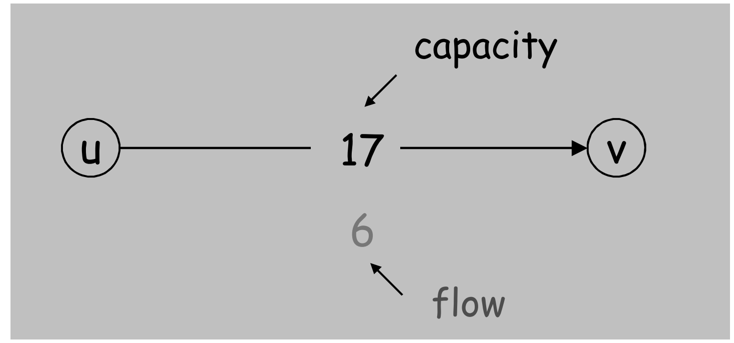




Residual Graph

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

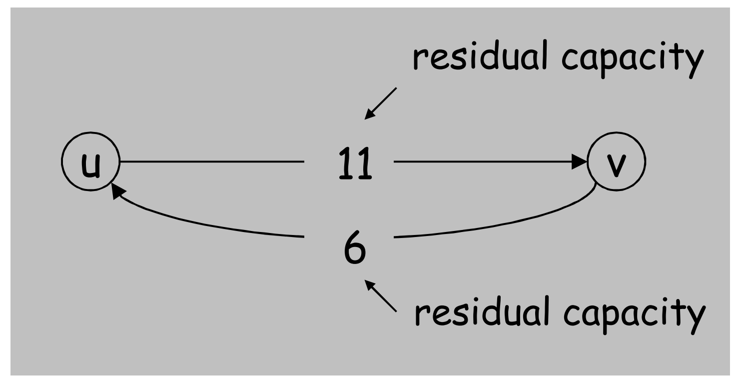
- Flow $f(e)$, capacity $c(e)$.



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

Augmenting path: a simple s-t path P in the residual graph G_f

Bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P

```
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
}
```

Claim: After augmentation, f is still a flow.

Ford-Fulkerson Algorithm

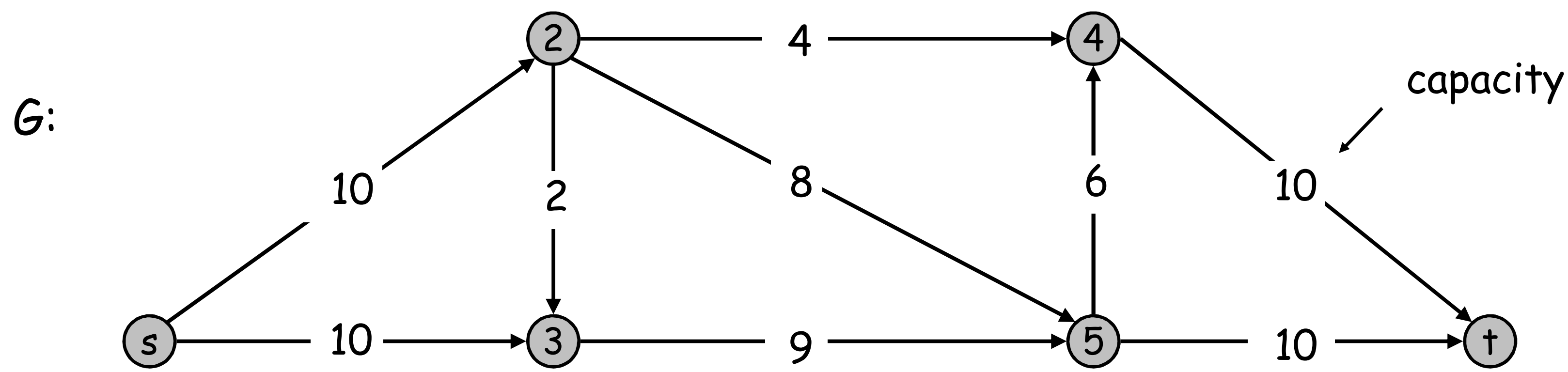
Ford-Fulkerson Algorithm

- 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.

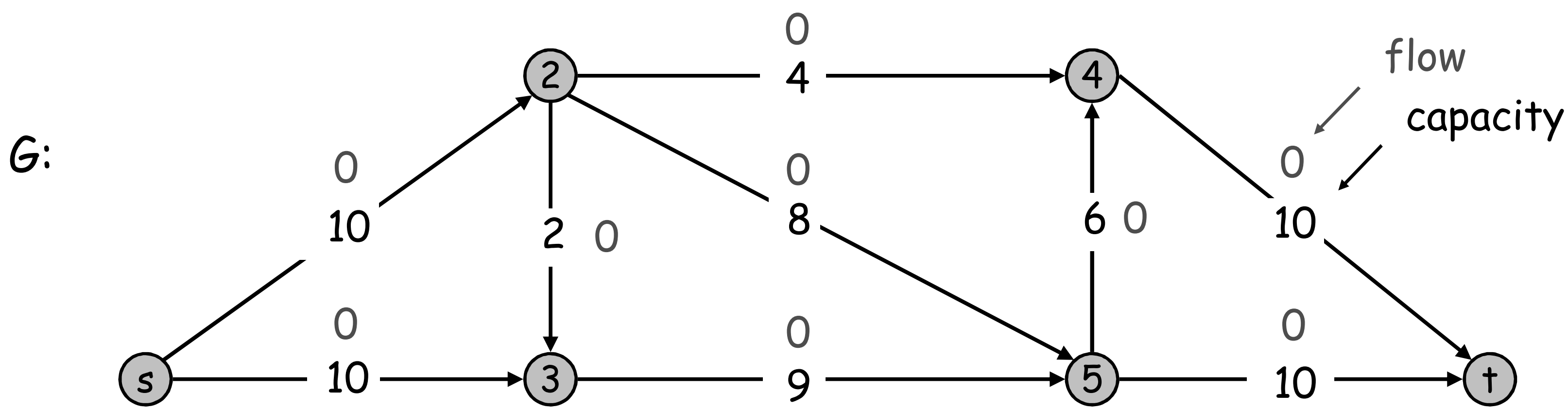
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

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

Ford-Fulkerson Algorithm

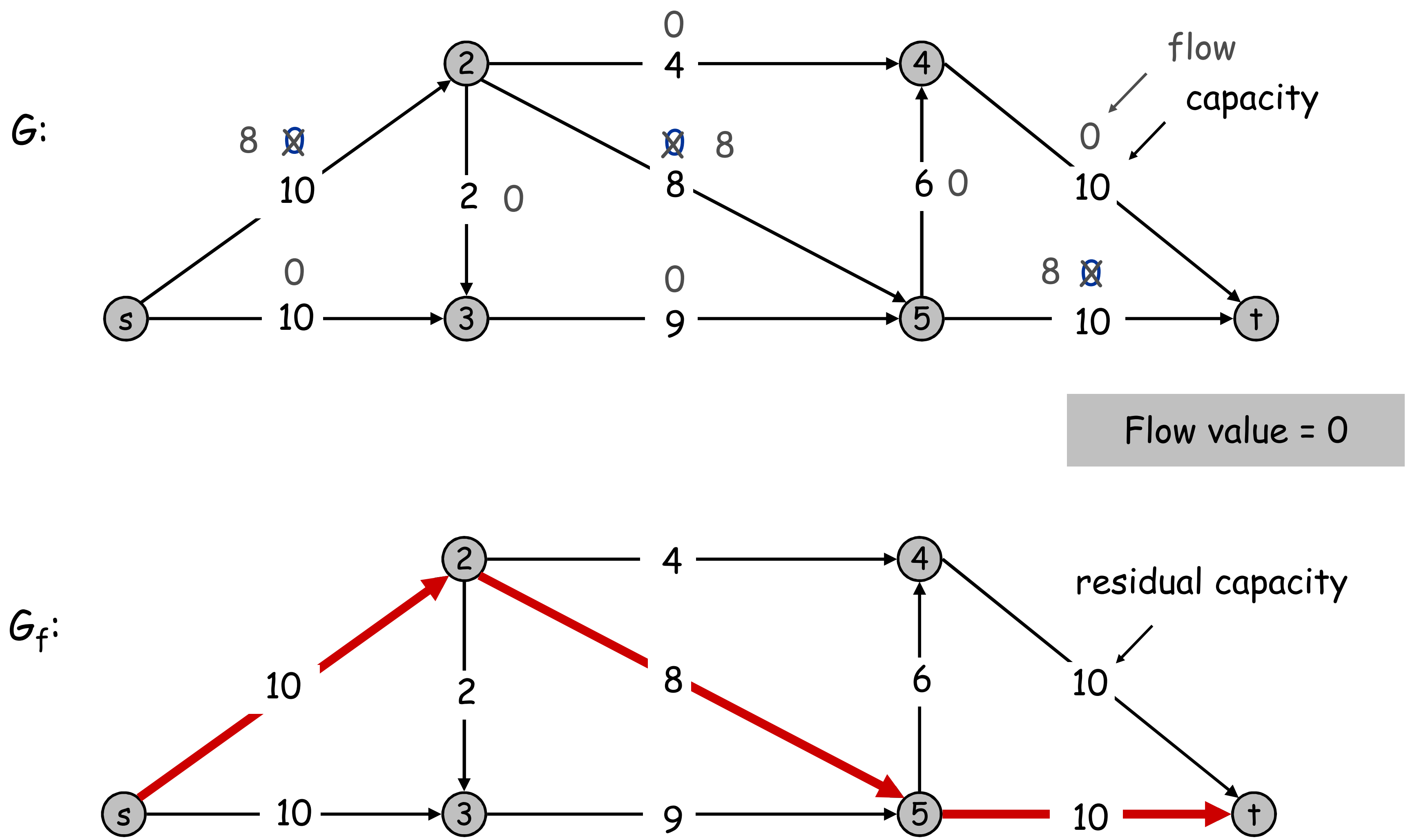


Ford-Fulkerson Algorithm

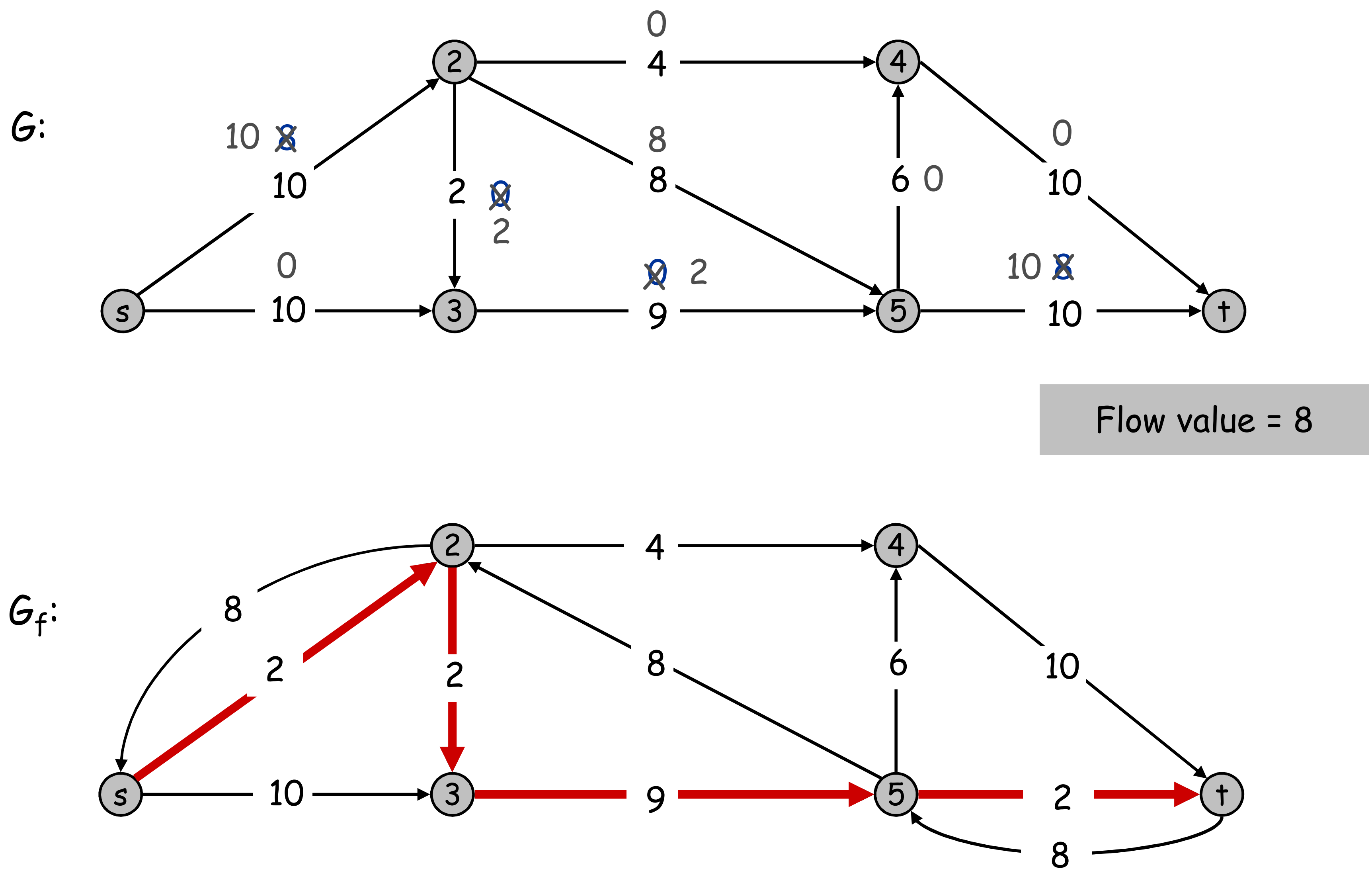


Flow value = 0

Ford-Fulkerson Algorithm

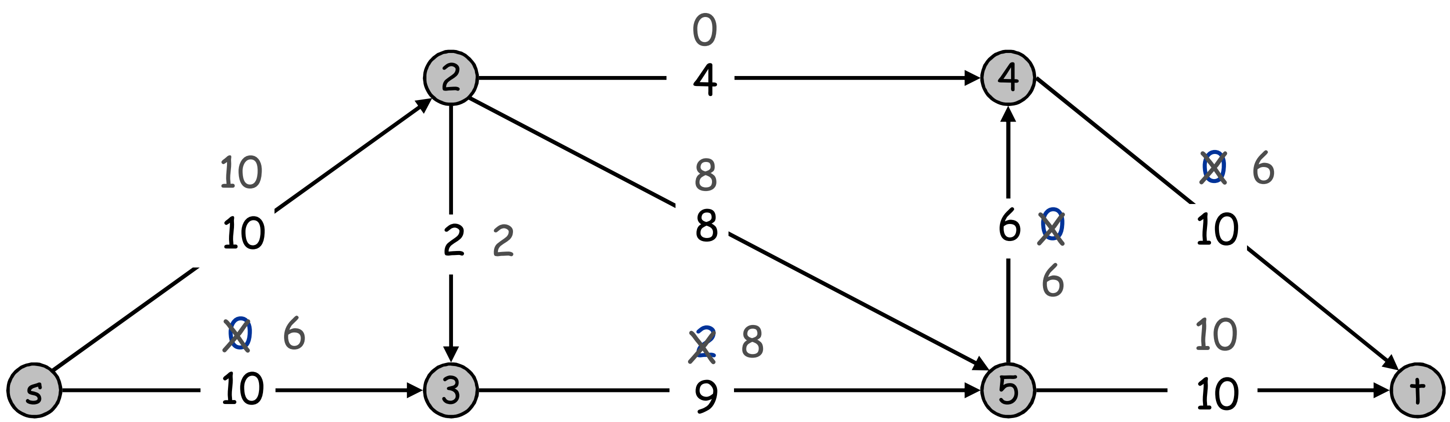


Ford-Fulkerson Algorithm



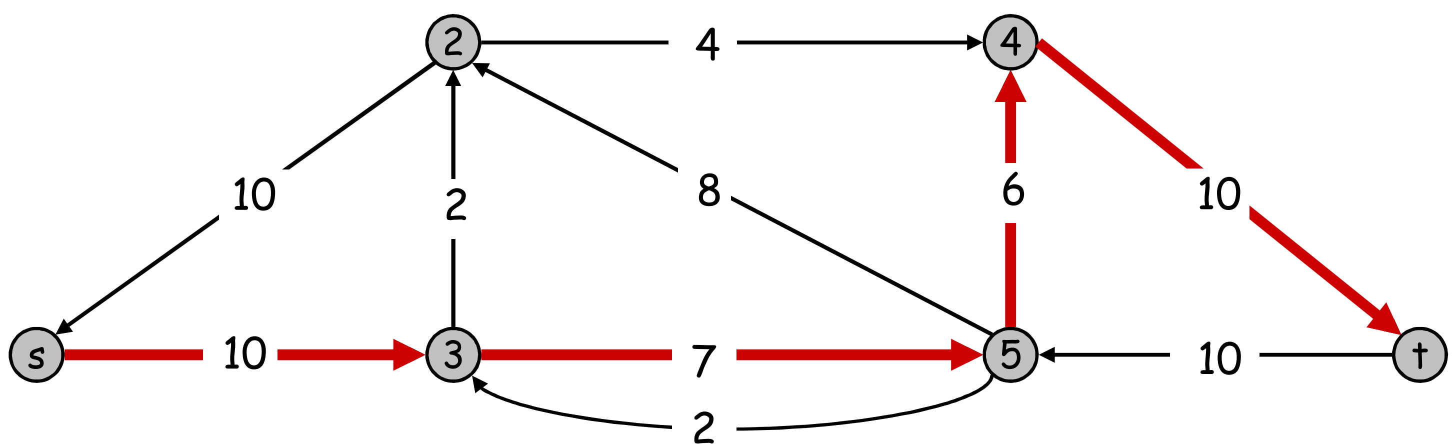
Ford-Fulkerson Algorithm

G:

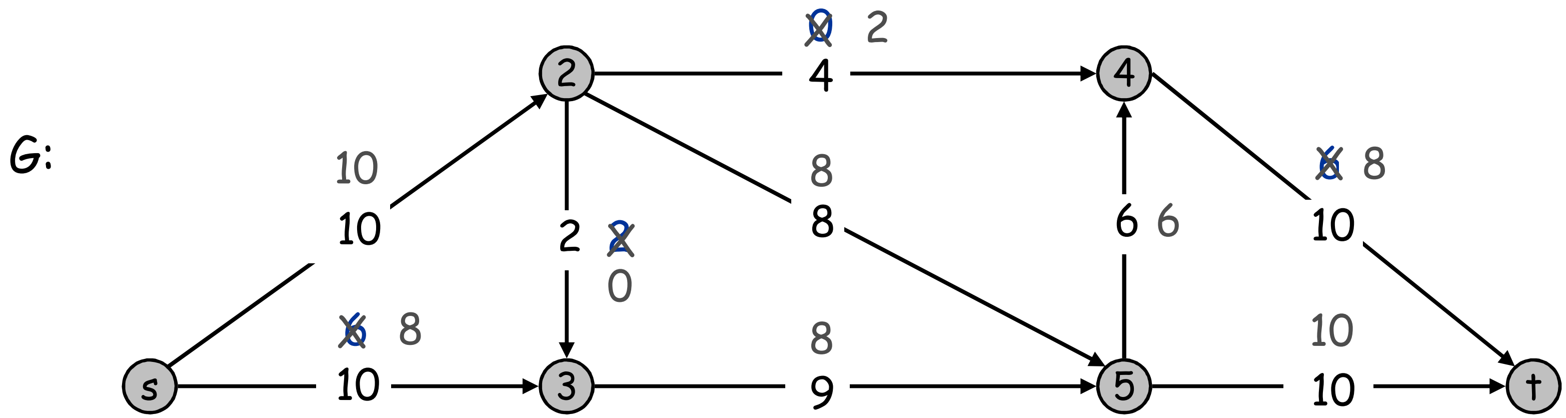


Flow value = 10

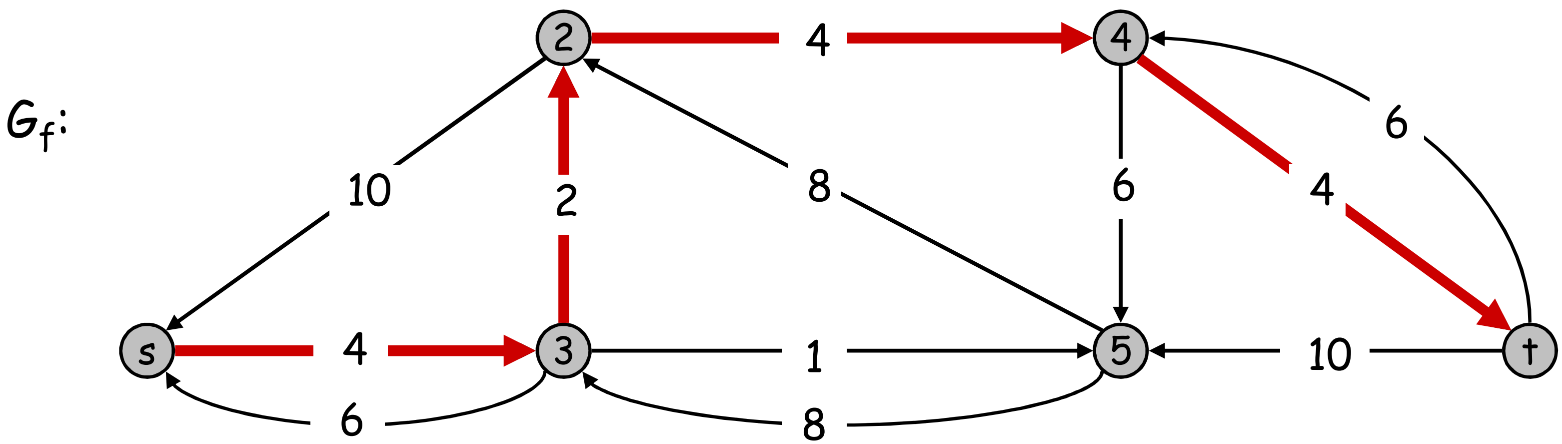
G_f:



Ford-Fulkerson Algorithm

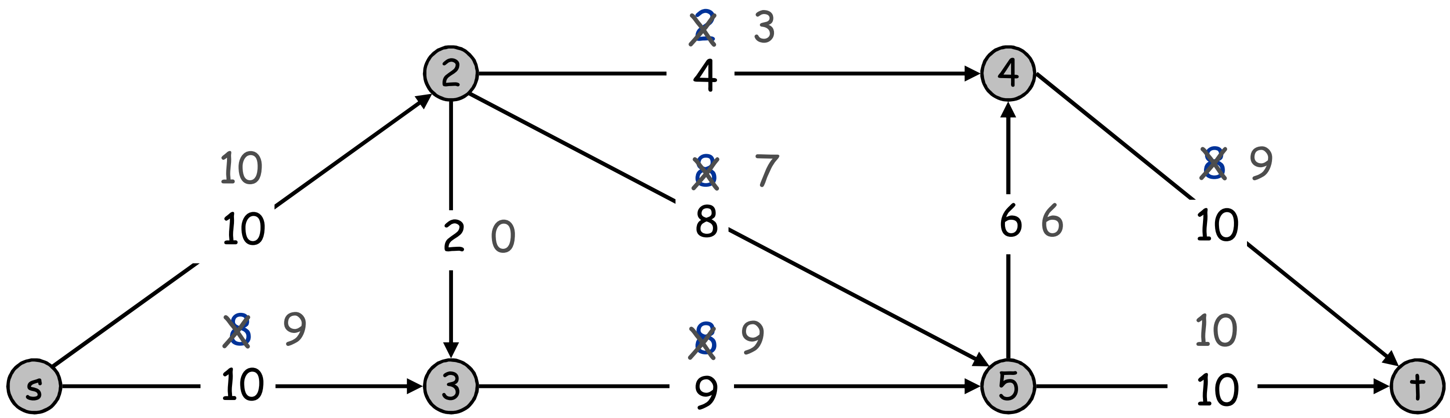


Flow value = 16



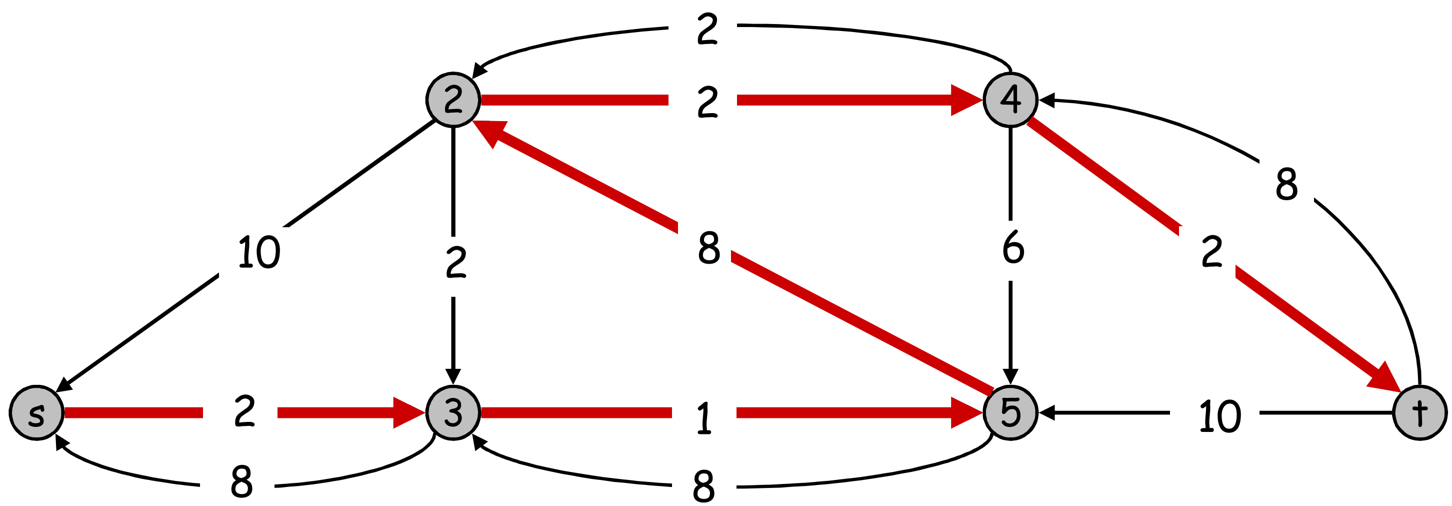
Ford-Fulkerson Algorithm

G:

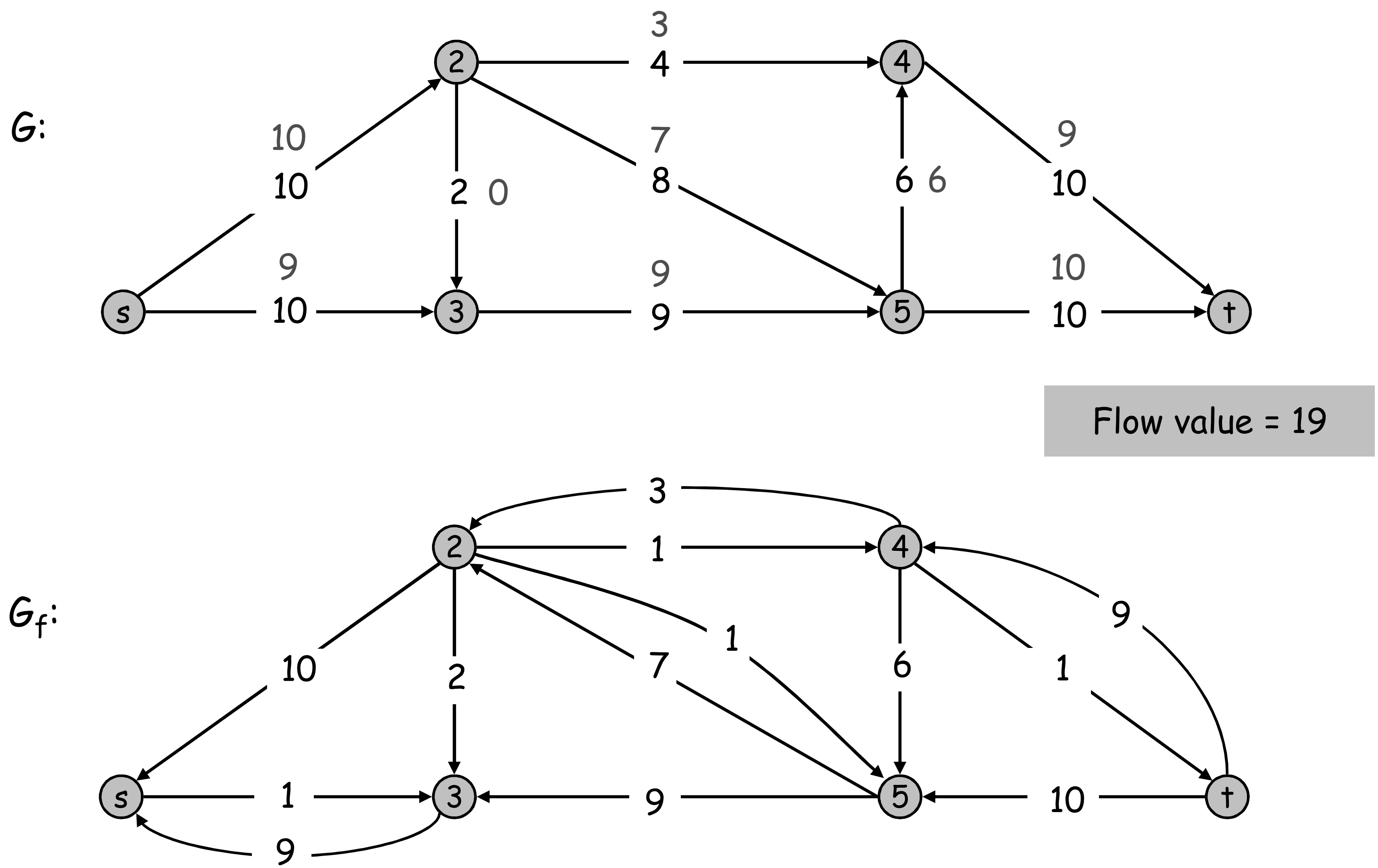


Flow value = 18

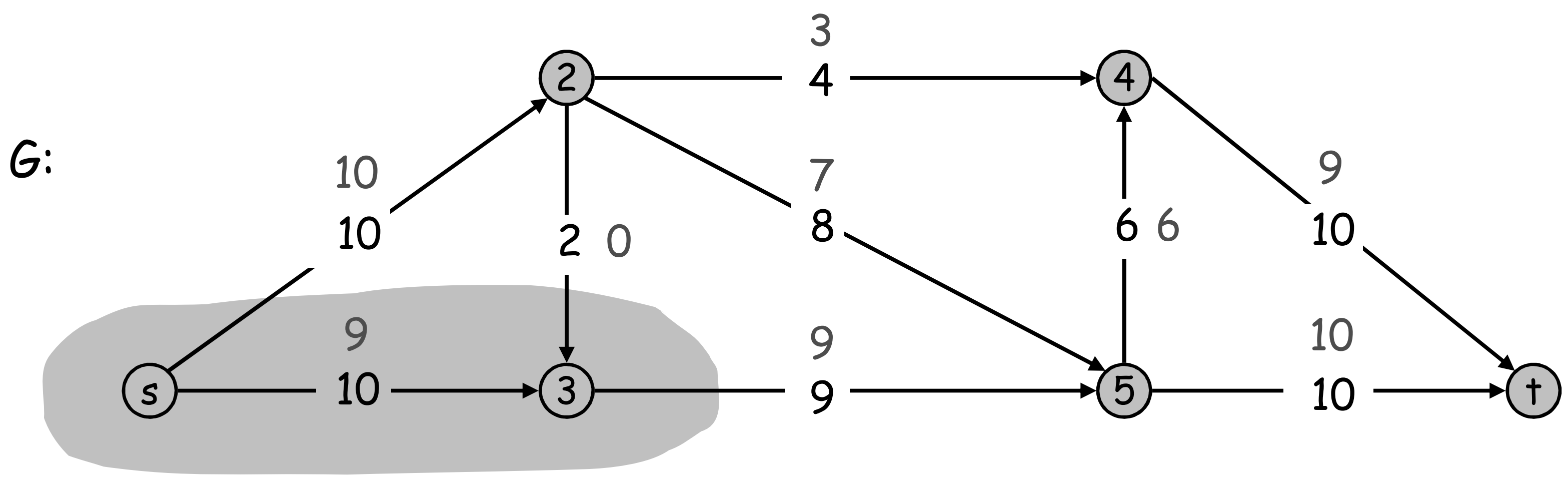
G_f:



Ford-Fulkerson Algorithm

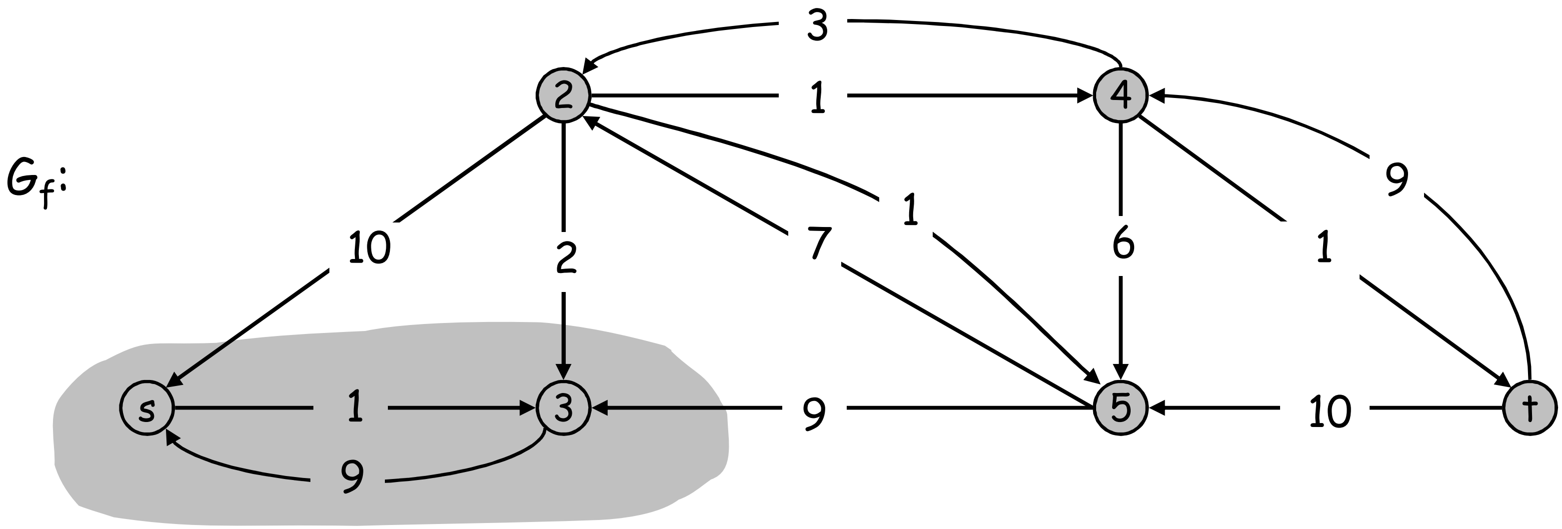


Ford-Fulkerson Algorithm



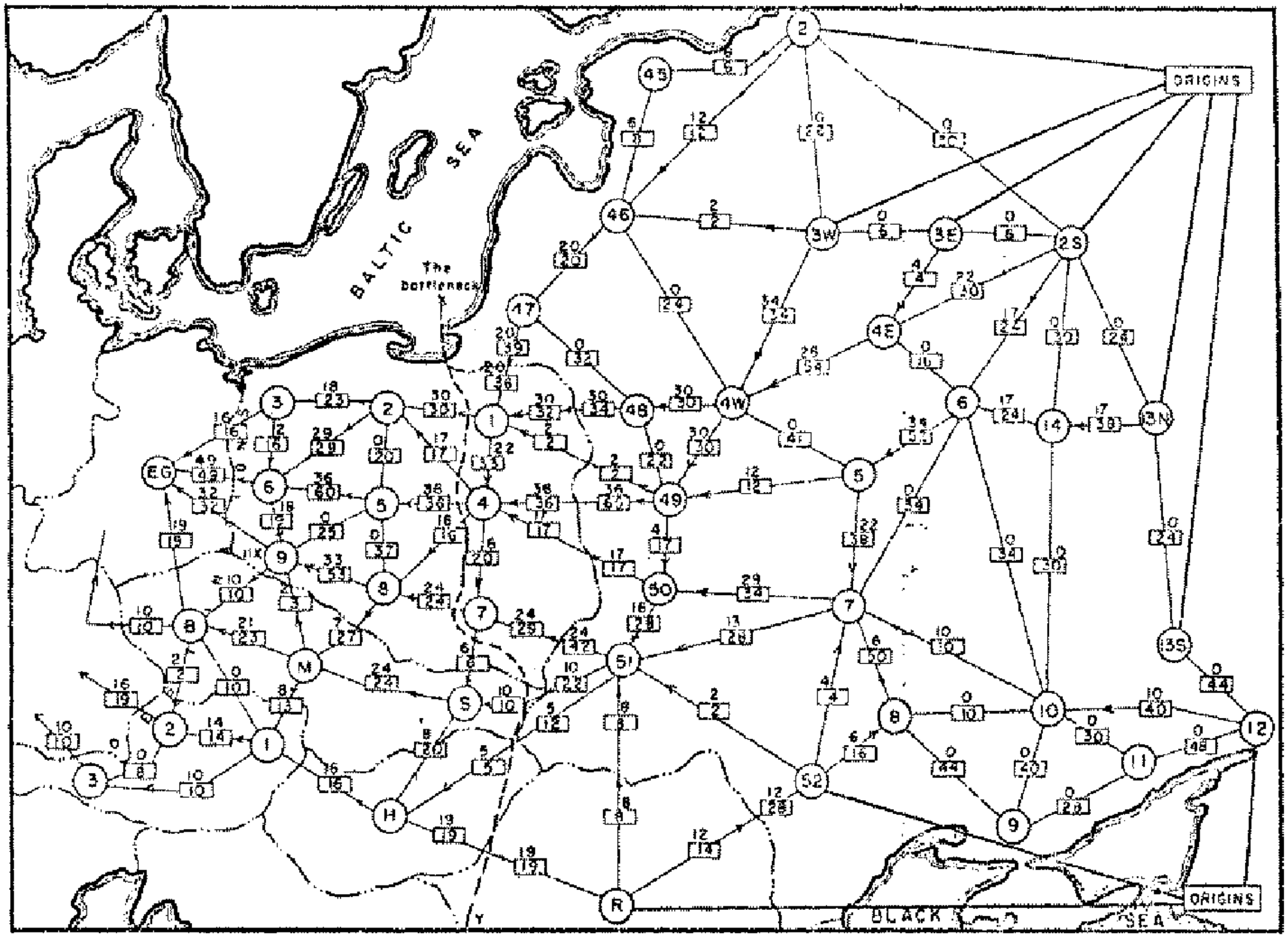
Cut capacity = 19

Flow value = 19



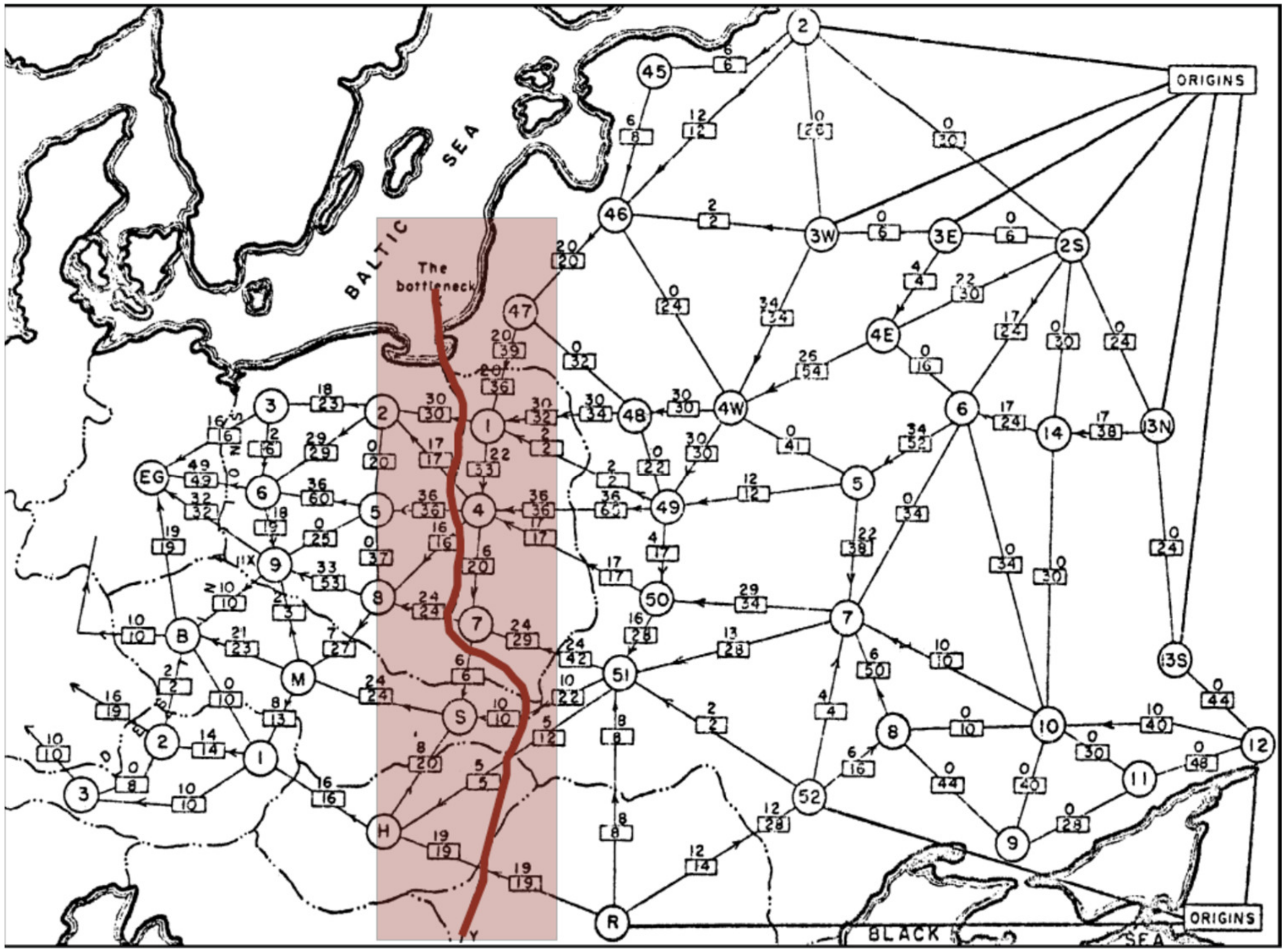
7.2 Max-flow and Min-cut

Soviet Rail Network, 1955



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

Soviet Rail Network, 1955

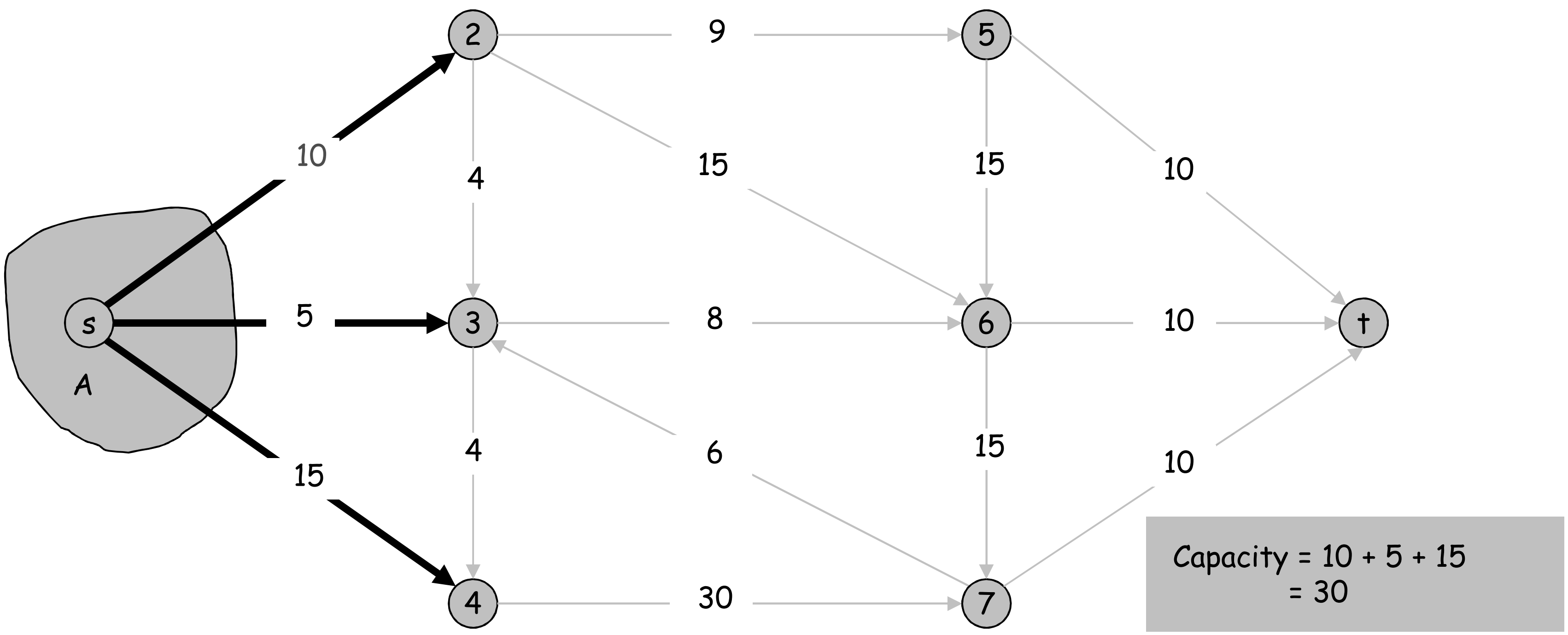


Reference: *On the history of the transportation and maximum flow problems.*
Alexander Schrijver in Math Programming, 91: 3, 2002.

Cuts

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

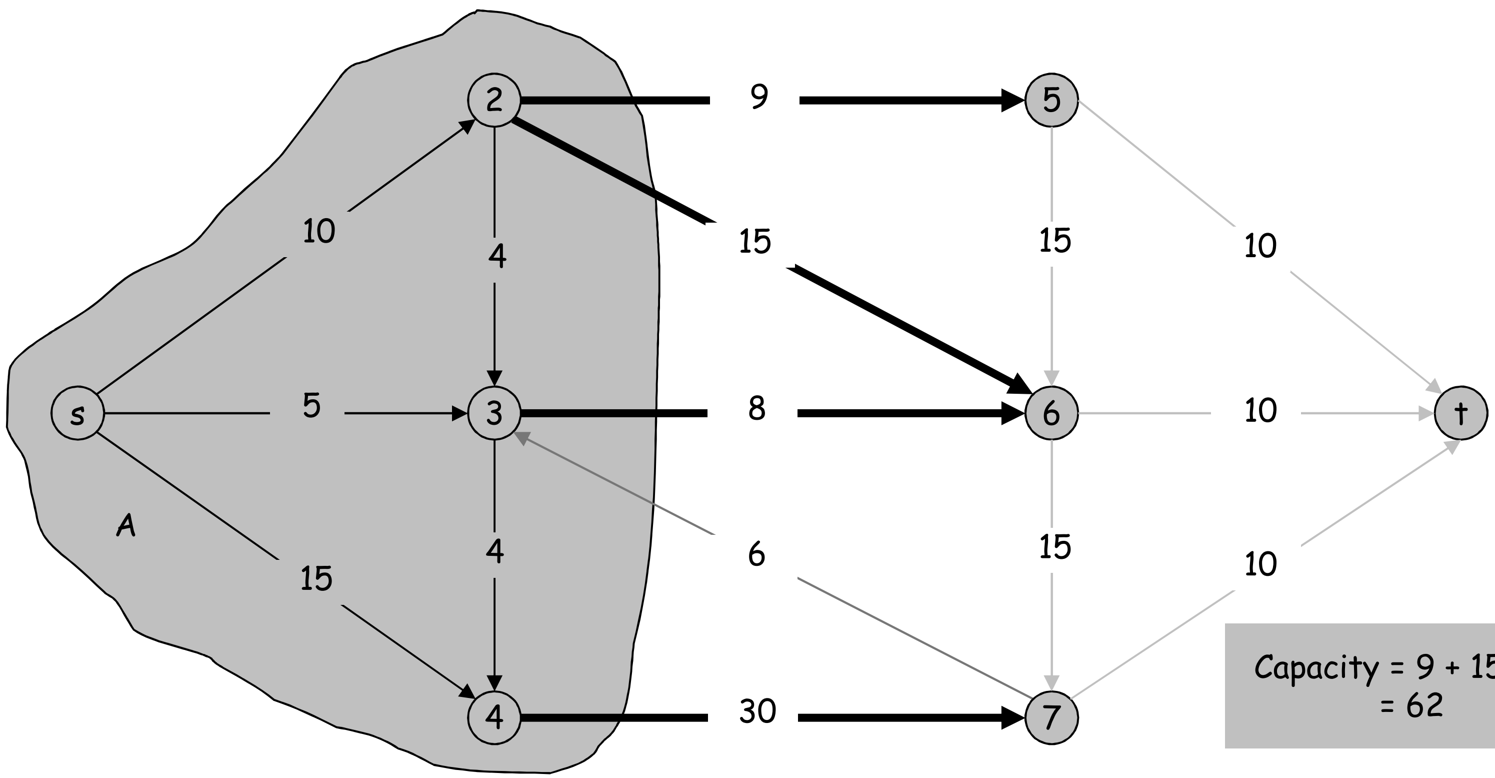
Def. The capacity of a cut (A, B) is: $cap(A, B) \ = \sum_{e \text{ out of } A} c(e)$



Cuts

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

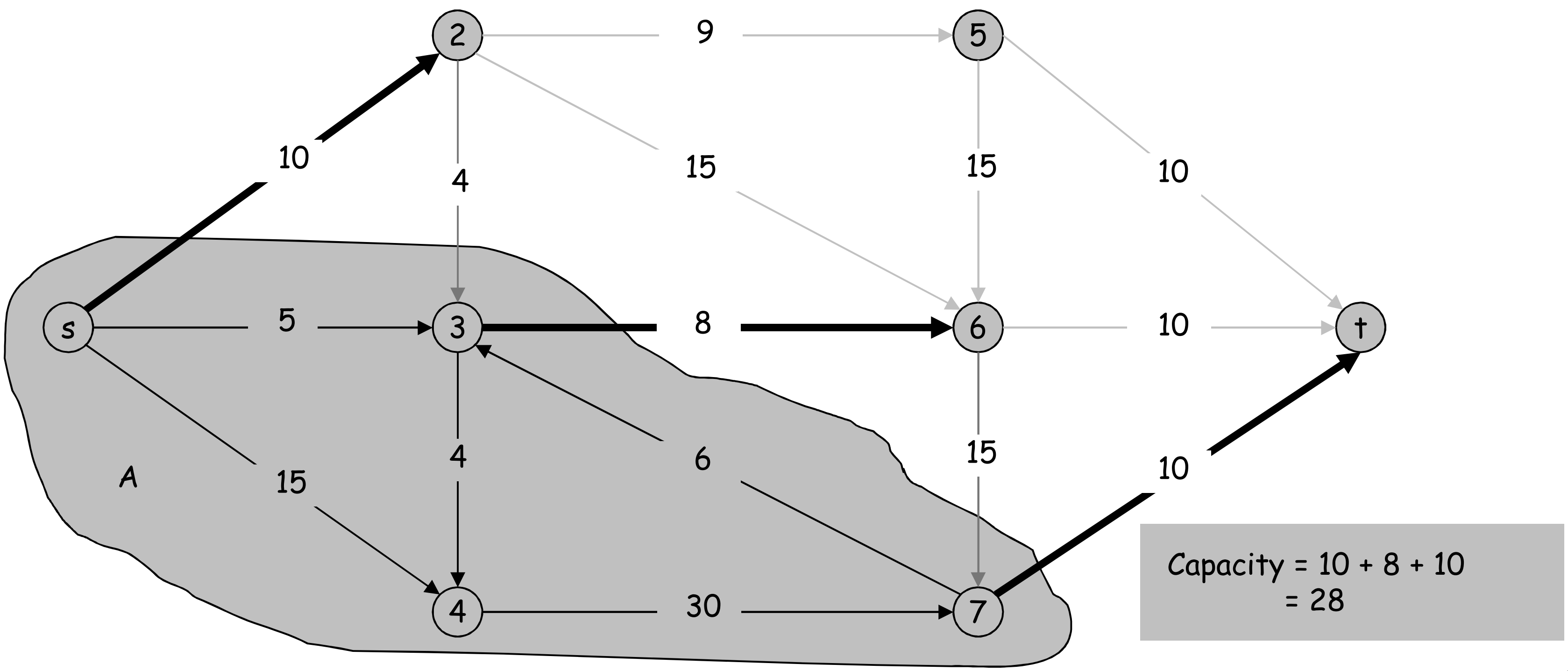
Def. The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

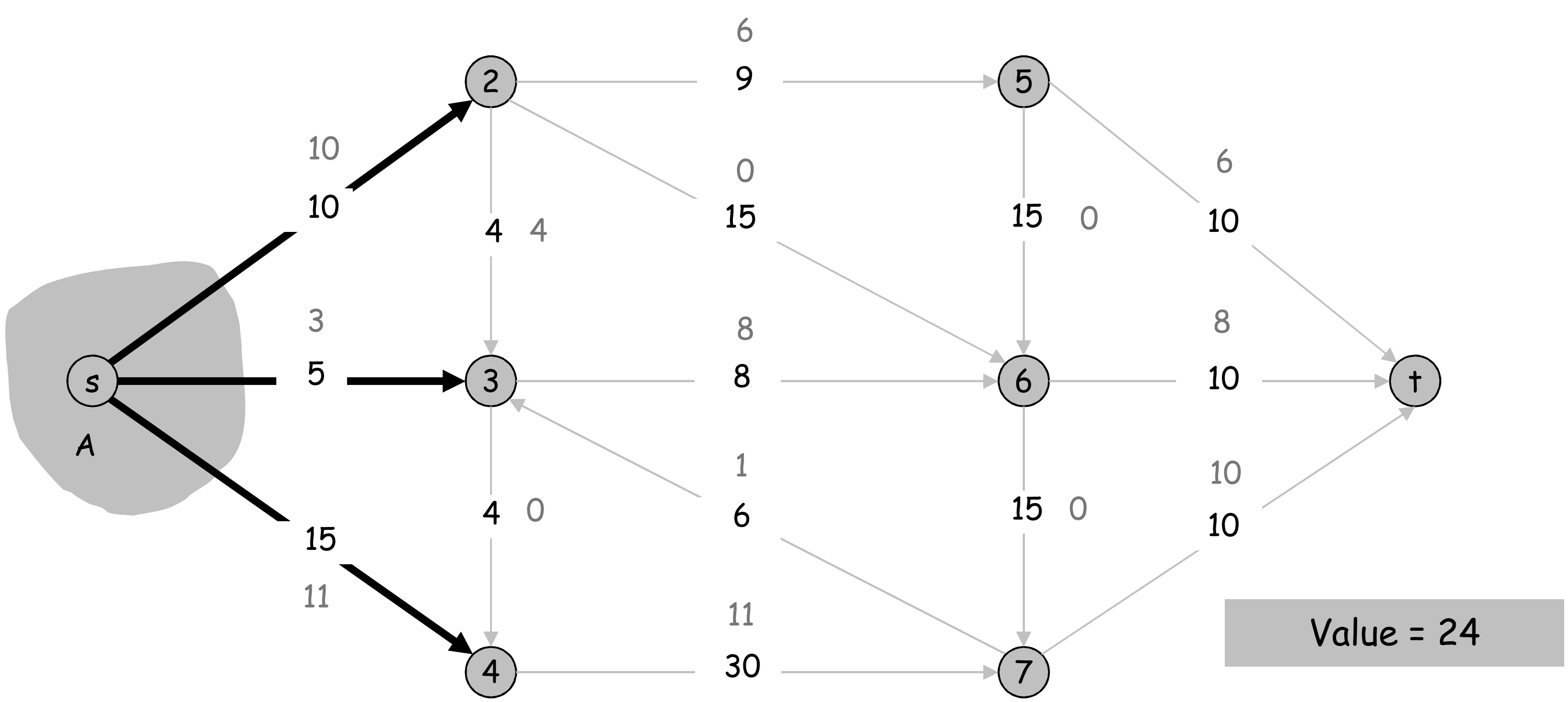
Min s-t cut problem. Find an s-t cut of minimum capacity.



Flows 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.

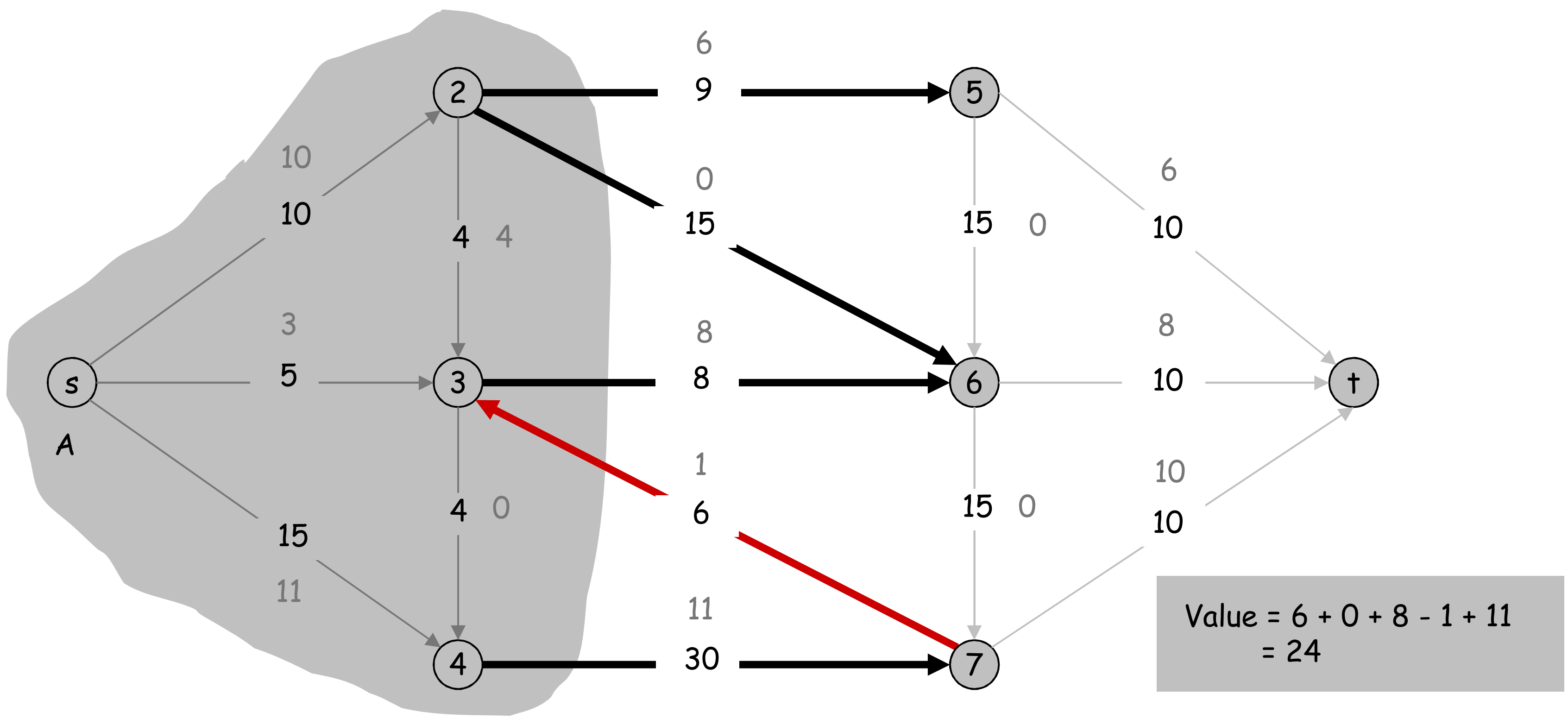
$$\sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;=\; v(f)$$



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.

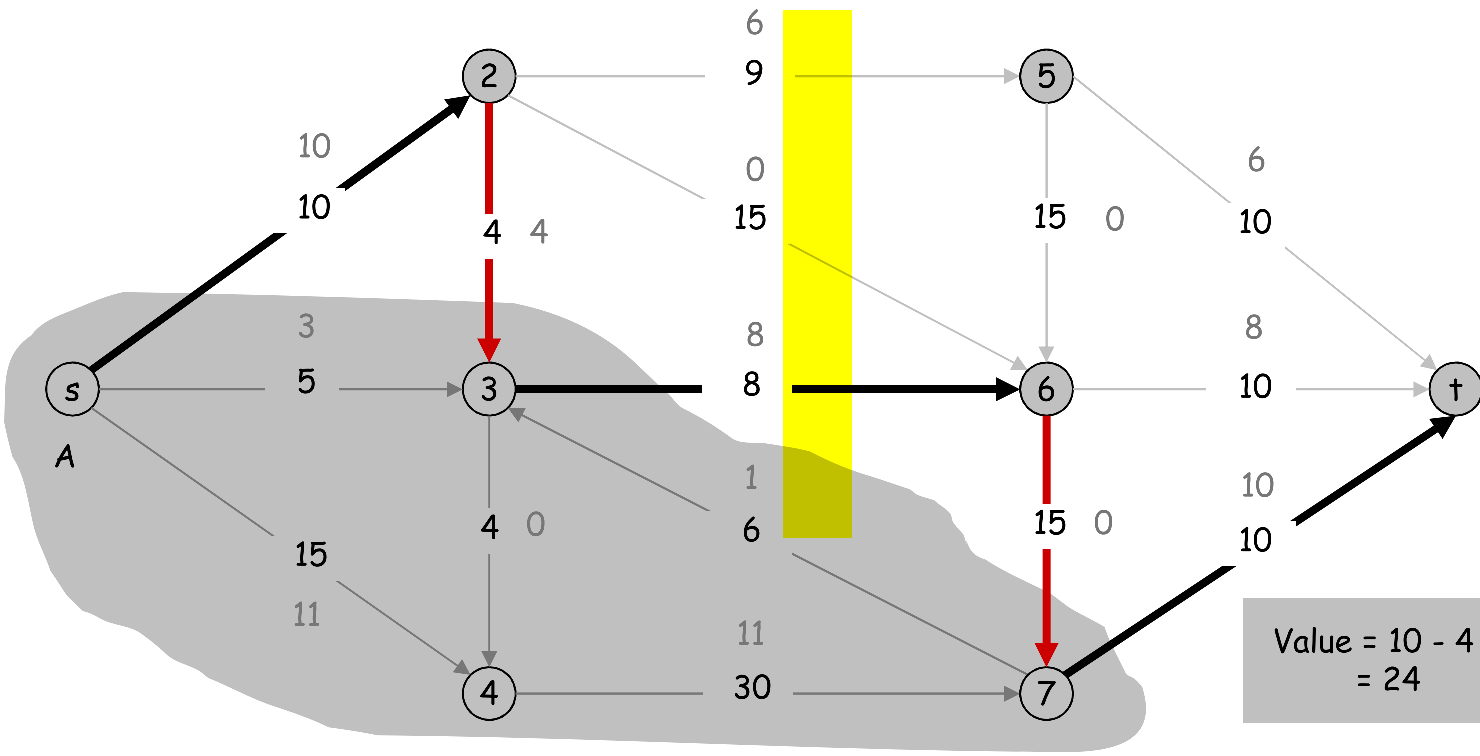
$$\sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;=\; v(f)$$



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.

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

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

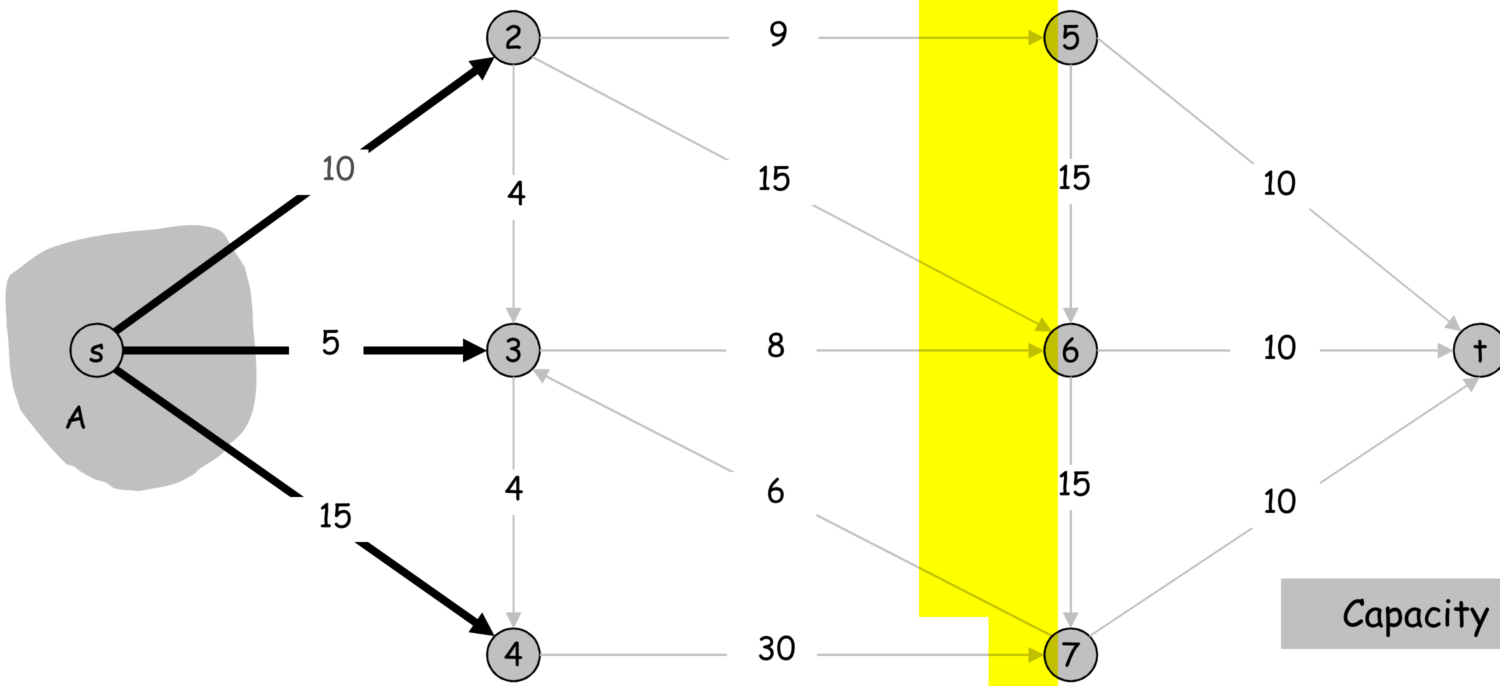
$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \right)$$

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

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

Flows and Cuts

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

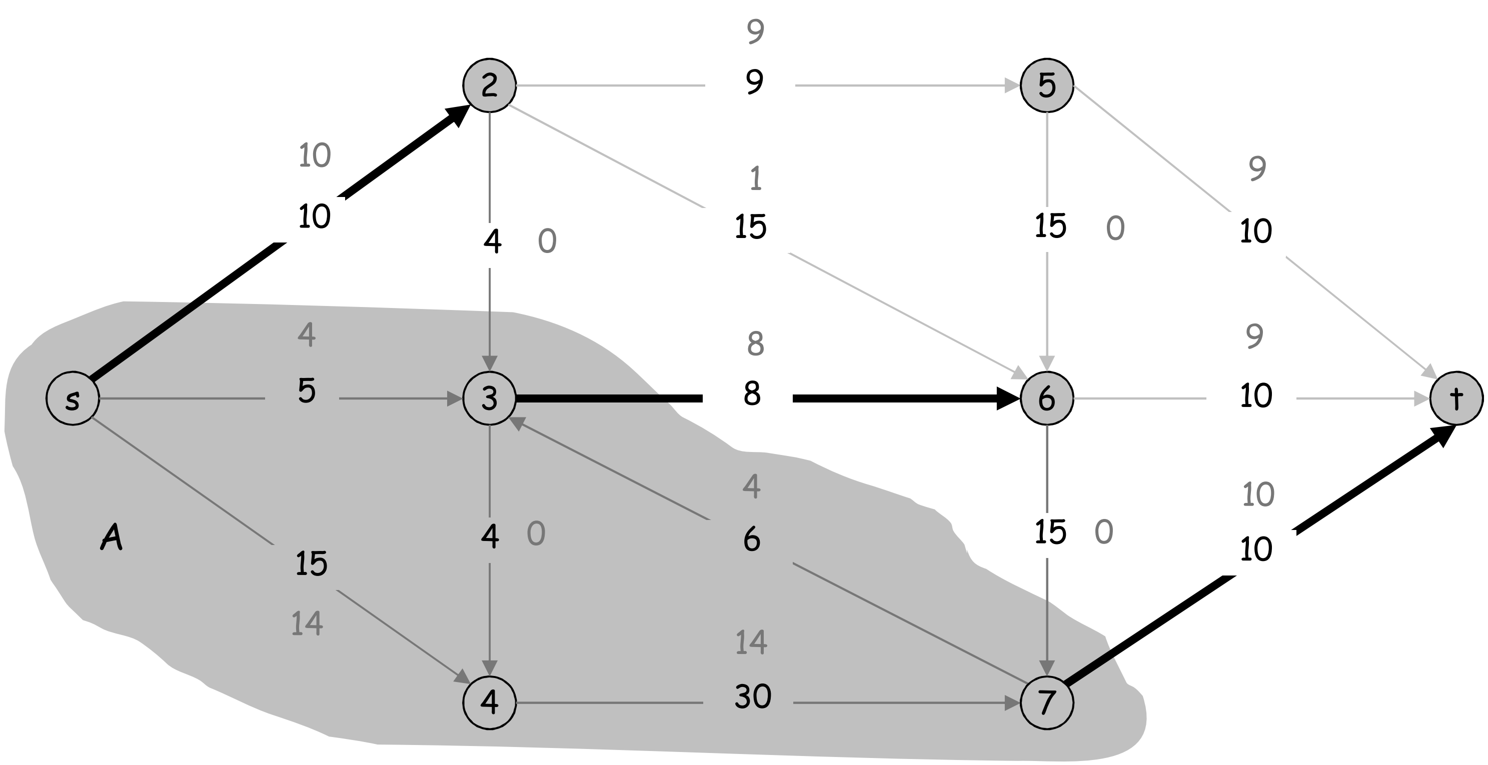
Pf.

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

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



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. [Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Proof strategy. We prove both simultaneously by showing the equivalence of the following three conditions 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.

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

Proof of Max-Flow Min-Cut Theorem

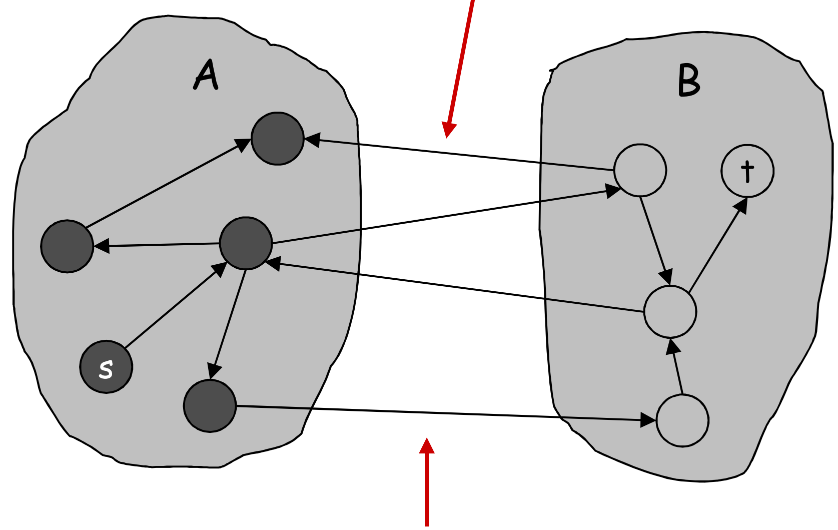
(iii) $\Rightarrow$ (i)

- 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, $t \notin A$.

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

original network

B->A edge must have f(e) = 0



A->B edge must have f(e) = c(e)

7.3 Choosing Good Augmenting Paths

Running Time

Assumption. All capacities are integers between 1 and C.

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

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. ▪

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

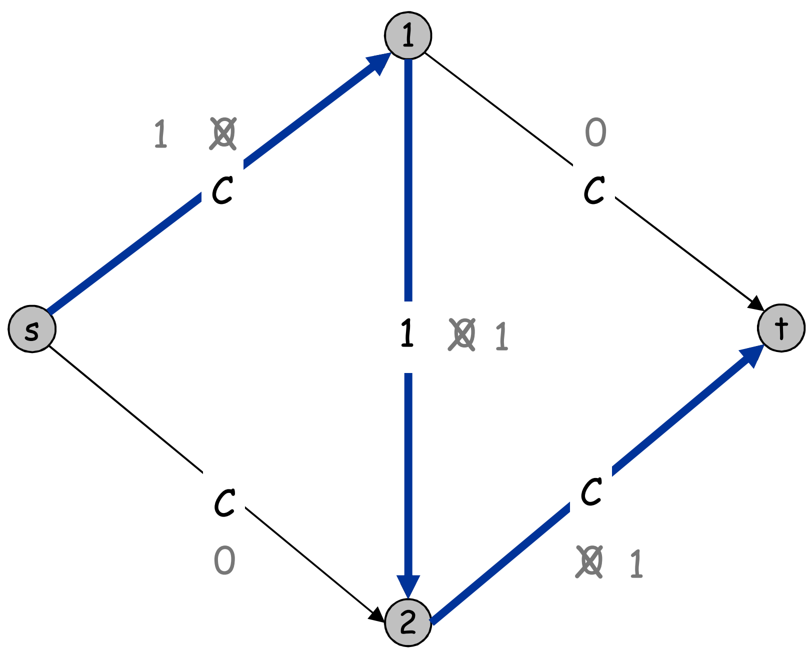
Corollary. Running time of Ford-Fulkerson is $O(mnC)$ ⟵ Polynomial?

Ford-Fulkerson: Exponential Number of Augmentations

Generic Ford-Fulkerson algorithm is not polynomial in input size?

m, n, and log C

An example: If max capacity is C, then the algorithm can take ≥ C iterations.

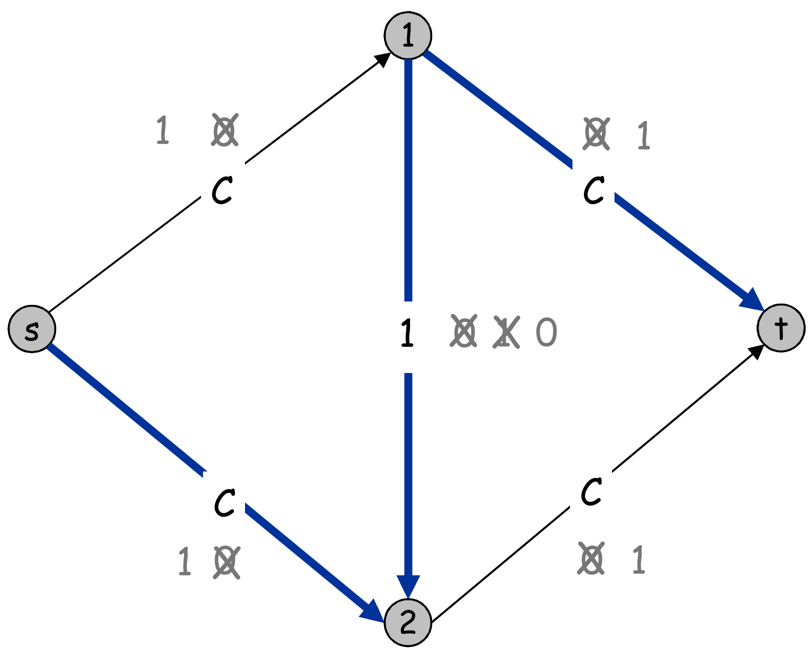


Ford-Fulkerson: Exponential Number of Augmentations

Generic Ford-Fulkerson algorithm is not polynomial in input size?

m, n, and log C

An example: If max capacity is C, then the algorithm can take $\geq C$ iterations.

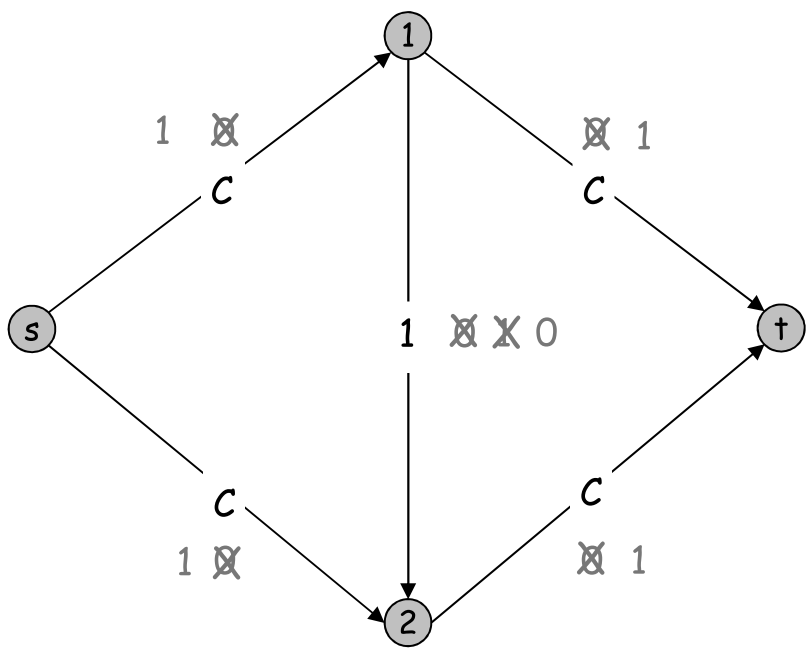


Ford-Fulkerson: Exponential Number of Augmentations

Generic Ford-Fulkerson algorithm is not polynomial in input size?

m, n, and log C

An example: If max capacity is C, then the algorithm can take $\geq C$ iterations.



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.

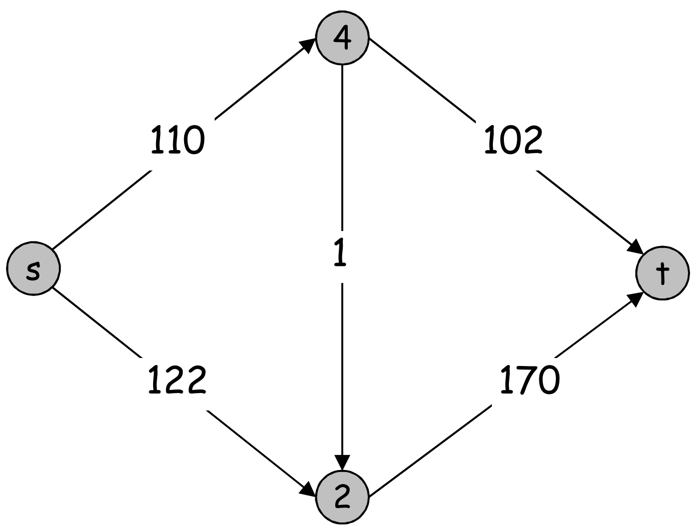
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

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

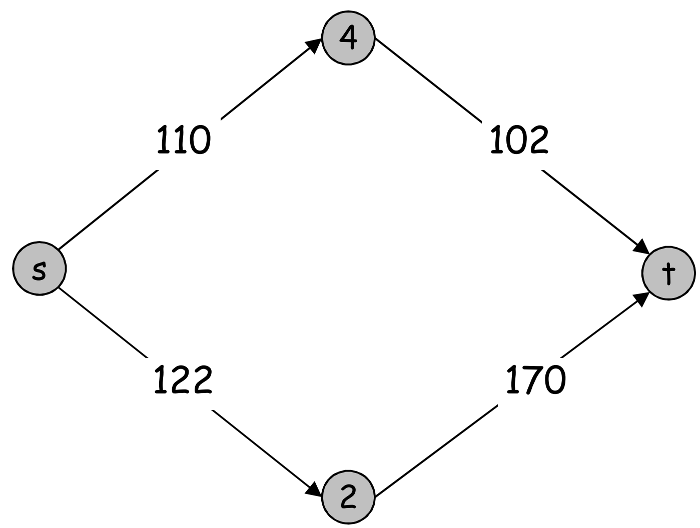
Capacity Scaling

Intuition. Choosing path with high bottleneck capacity

- 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

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← largest power of 2 ≤ C

   while (Δ ≥ 1) {
      G_f(Δ) ← Δ-residual graph
      while (there exists augmenting path P in G_f(Δ)) {
         f ← augment(f, c, P)
         update G_f(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 $1 + \lfloor \log_2 C \rfloor$ times.
Pf. Initially $C/2 < \Delta \le C$. 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 $\Rightarrow$ $v(f^*) \le v(f) + m\,(2\Delta)$.
- 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, $t \notin A$.

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