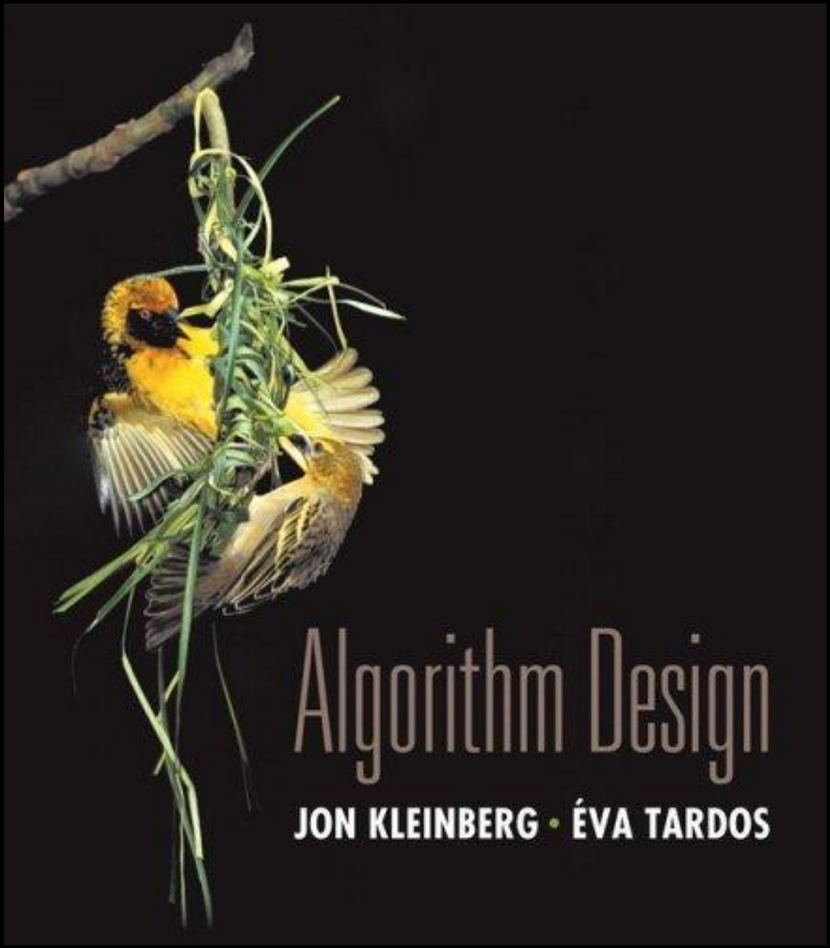
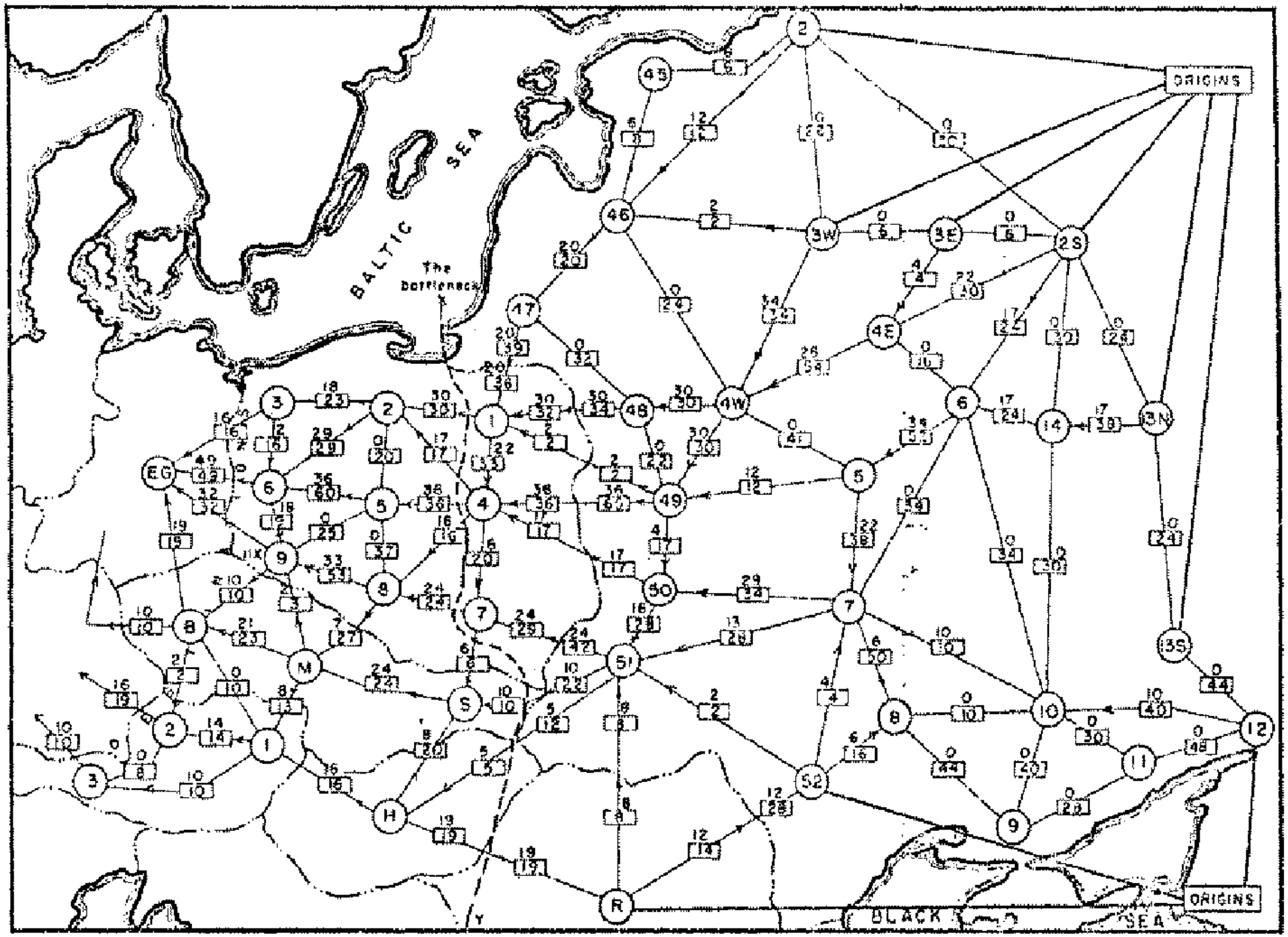


Chapter 7

Network Flow

Slides by Kevin Wayne.

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.

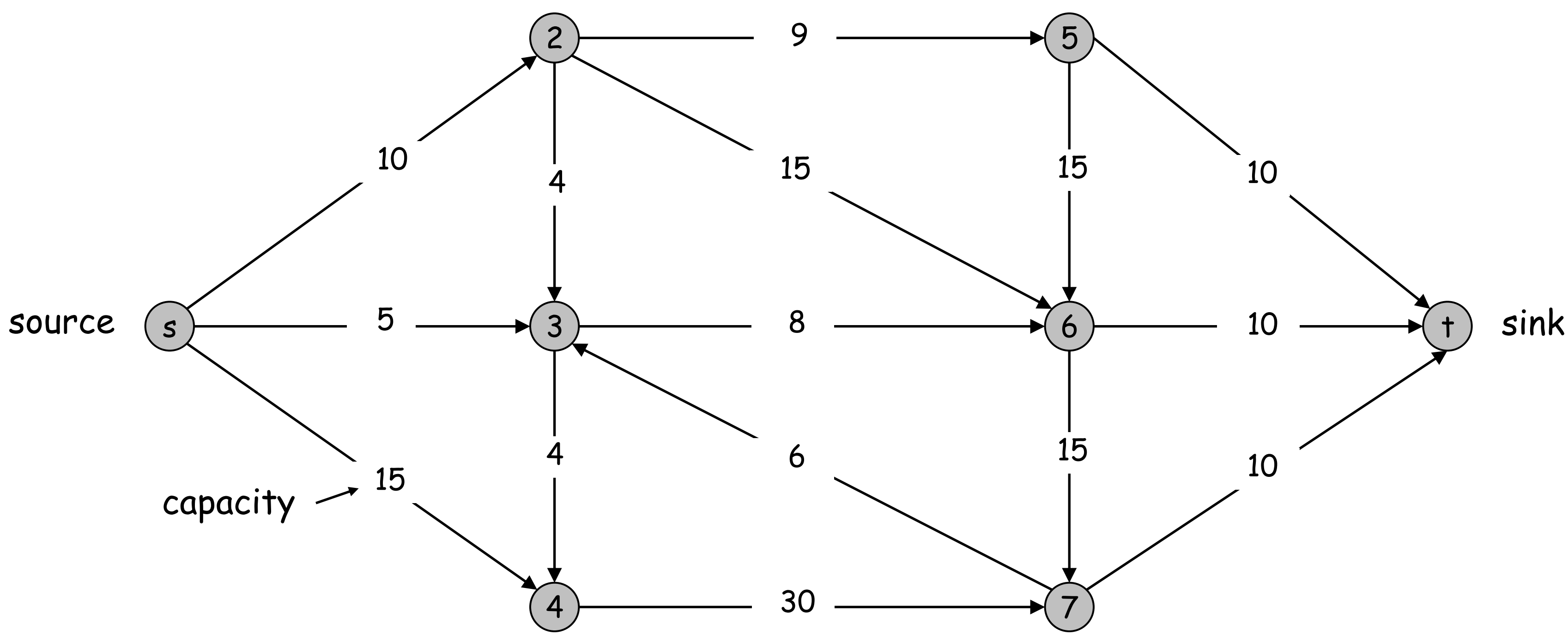
- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

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.
- c(e) = capacity of edge e.



Conservation of Flow

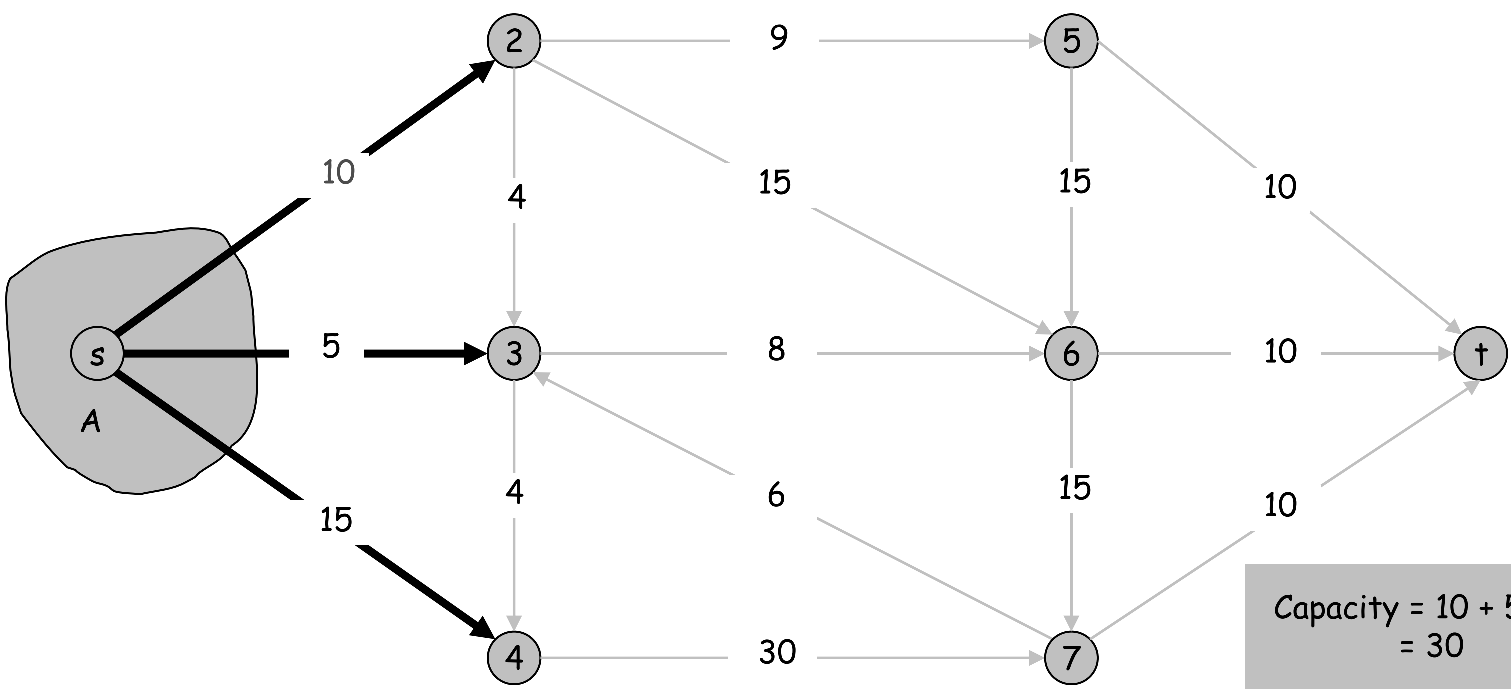
For the nodes other than s and t.

Total-flow-into-node-i=Total-flow-out-of-node-i

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

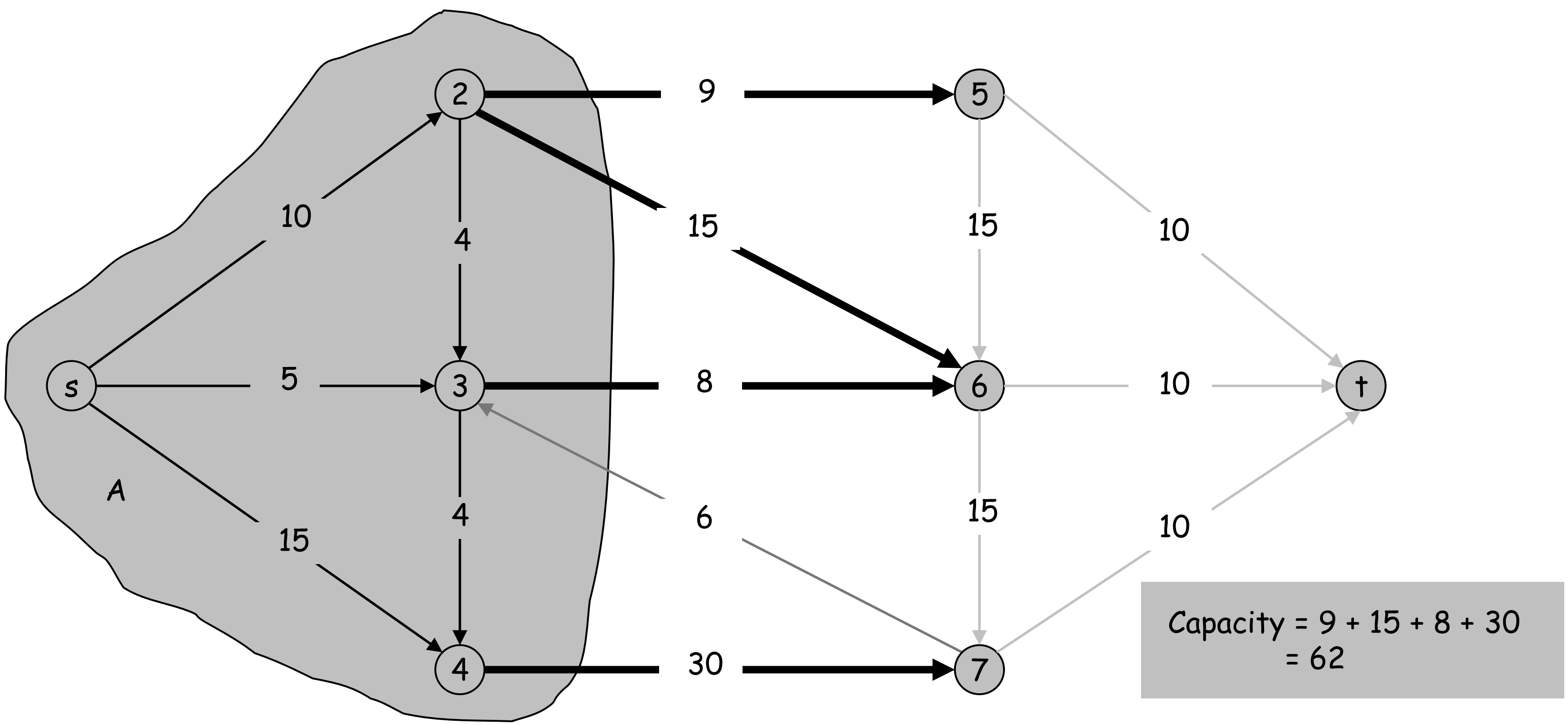


Capacity = 10 + 5 + 15
= 30

Cuts

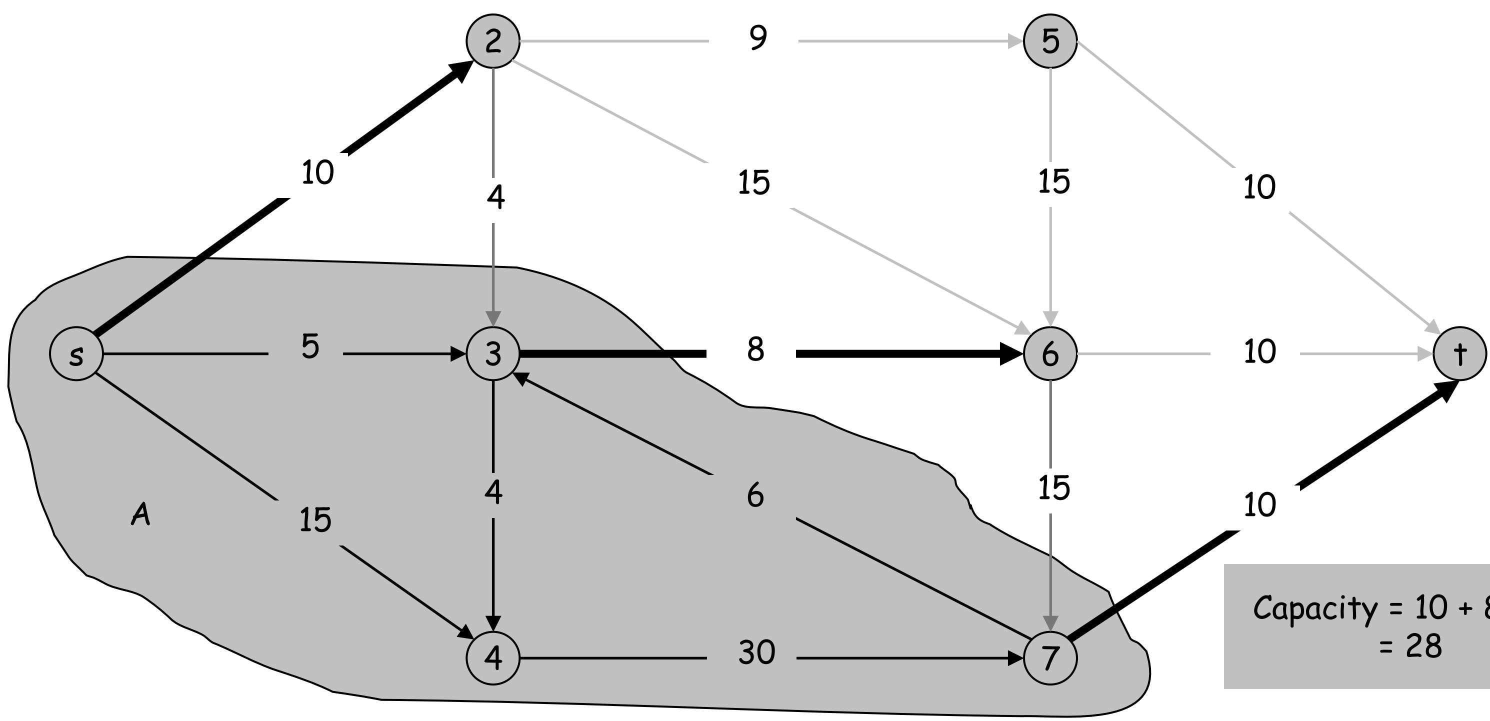
Def. An s-t cut is a partition (A, B) of V with s $\in$ A and t $\in$ B.

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



Minimum Cut Problem

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



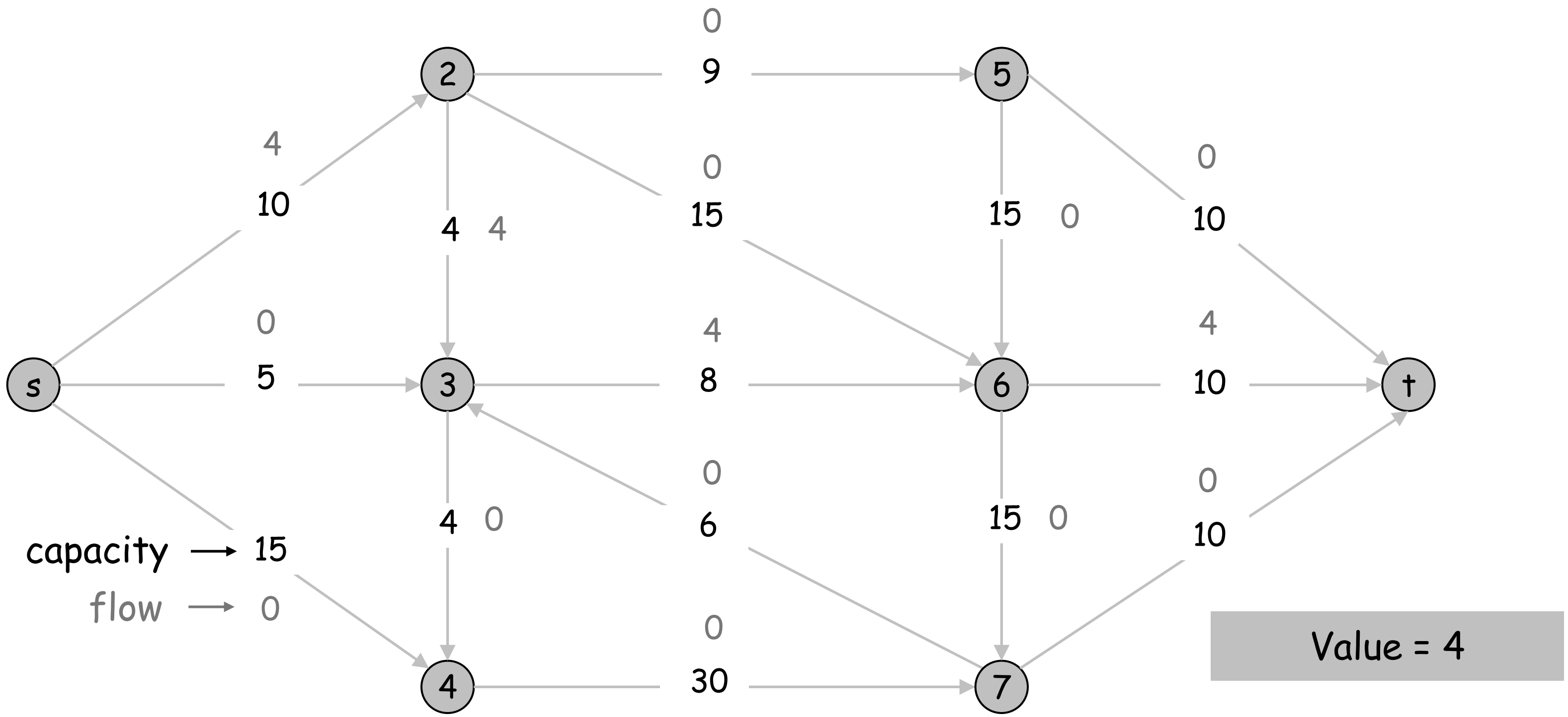
Capacity = 10 + 8 + 10
= 28

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)

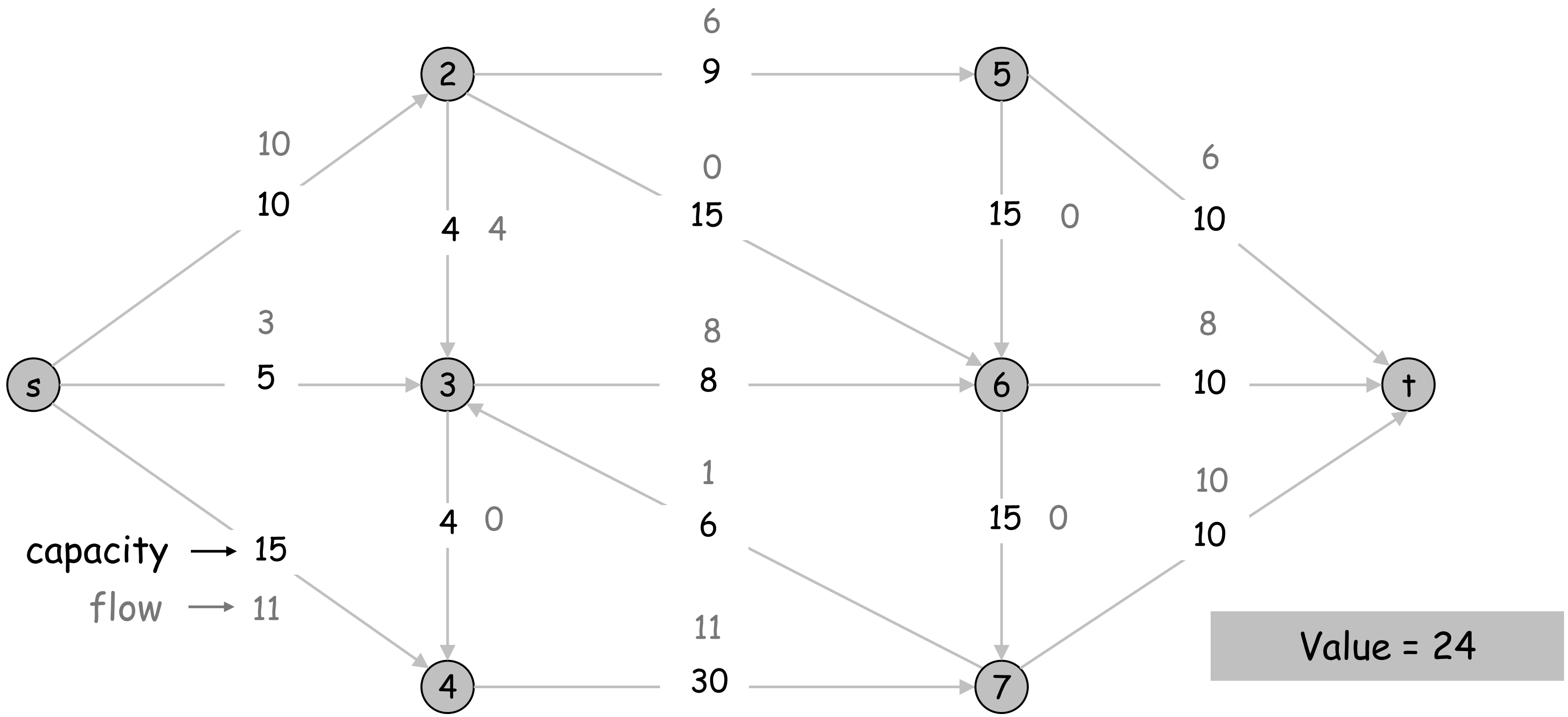
Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Flows

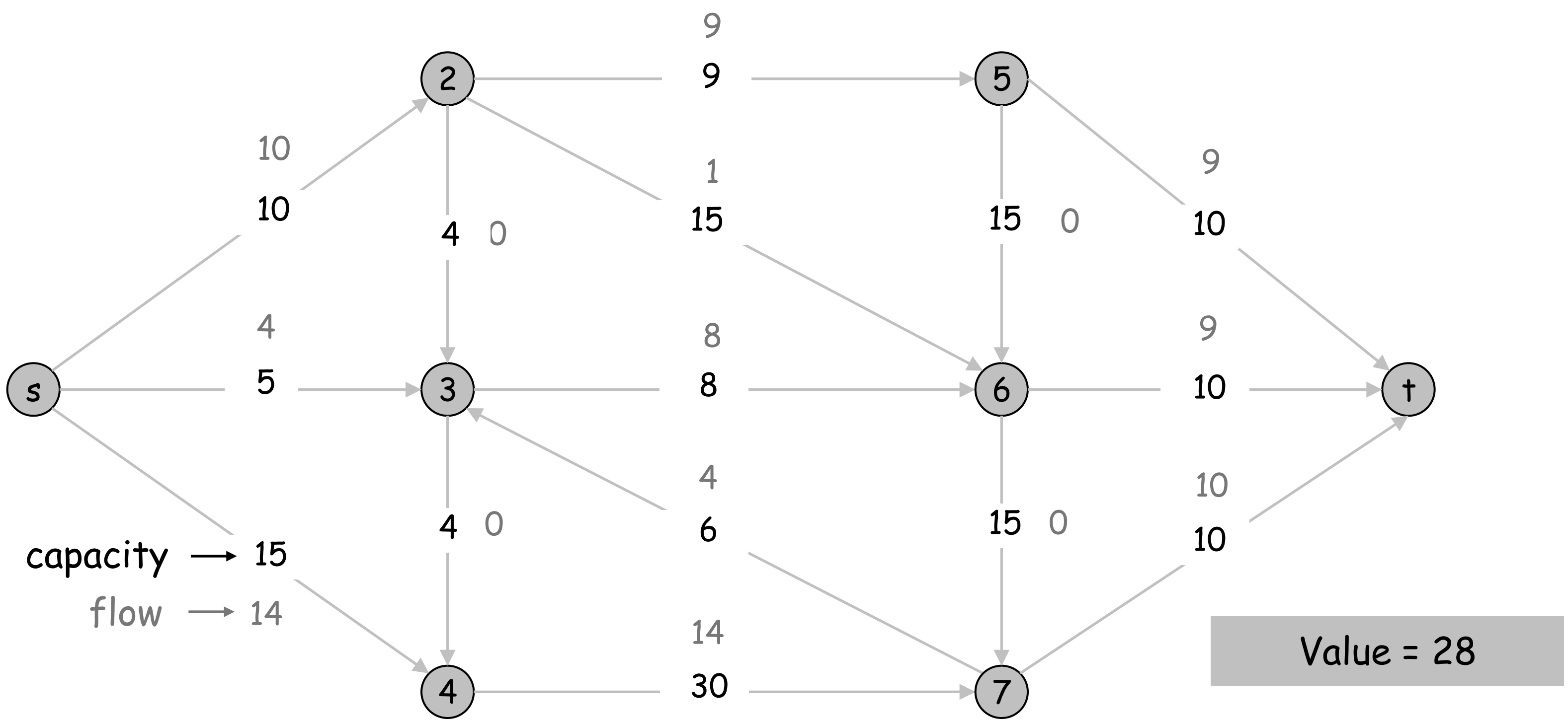
Def. An s-t flow is a function that satisfies:
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

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



Maximum Flow Problem

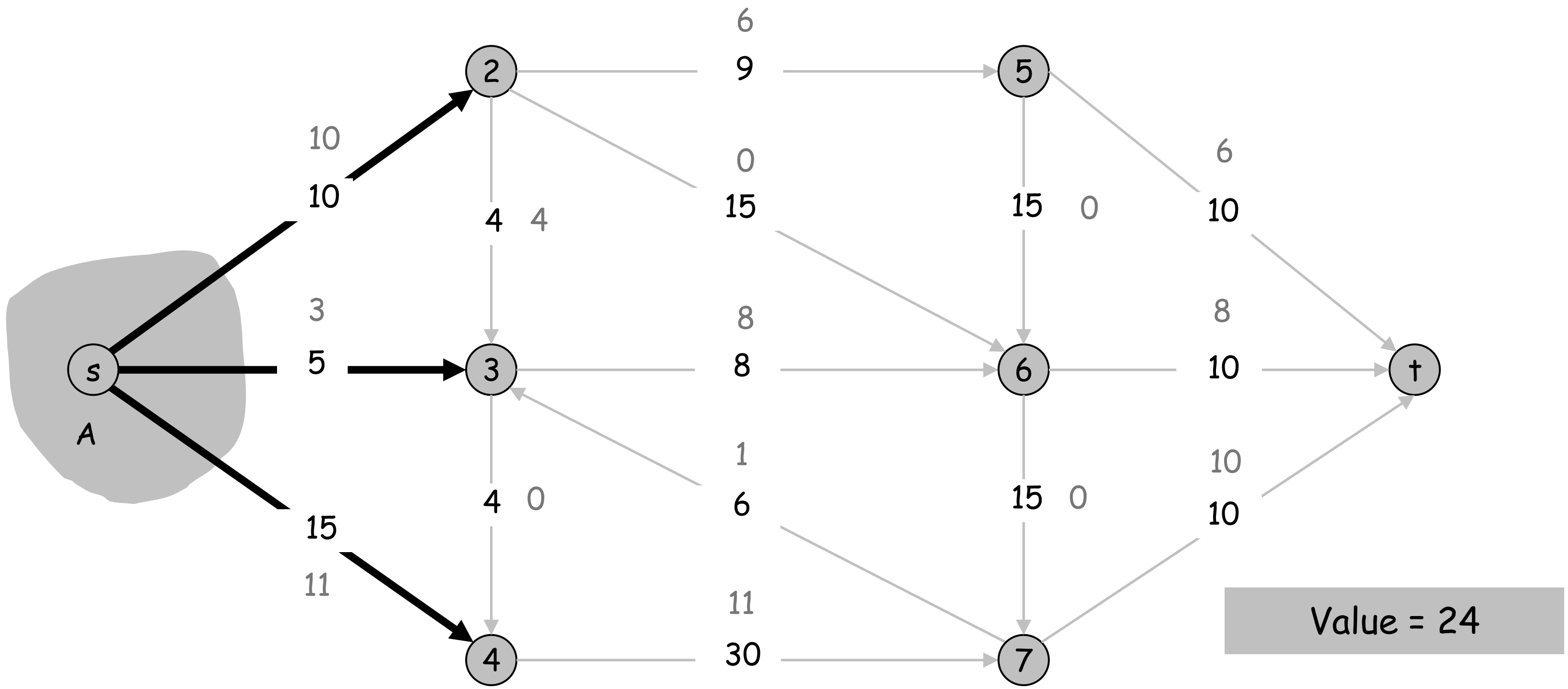
Max flow problem. Find s-t flow of maximum value.



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.

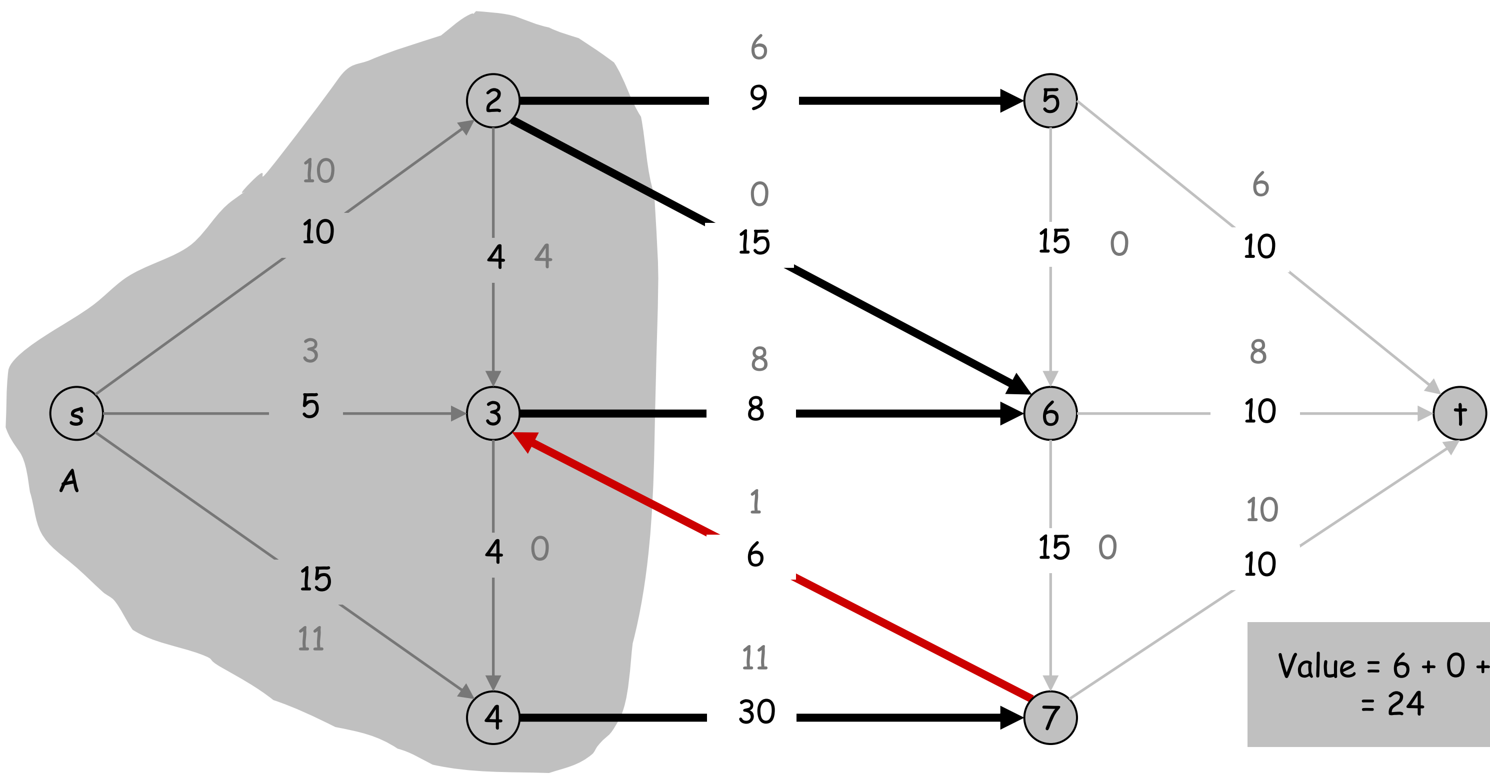
$$\sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;=\; v(f)$$



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.

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

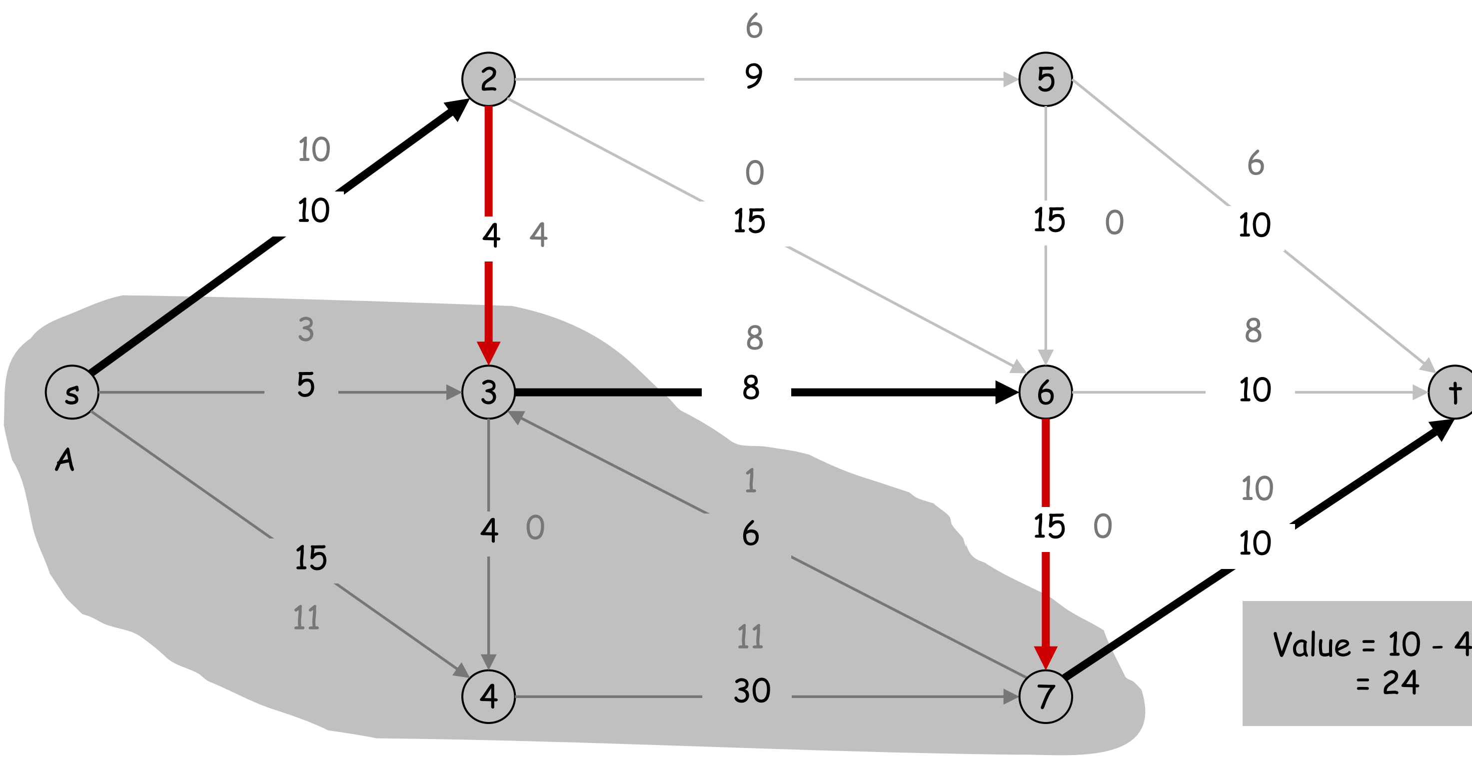


Value = 6 + 0 + 8 - 1 + 11
= 24

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.

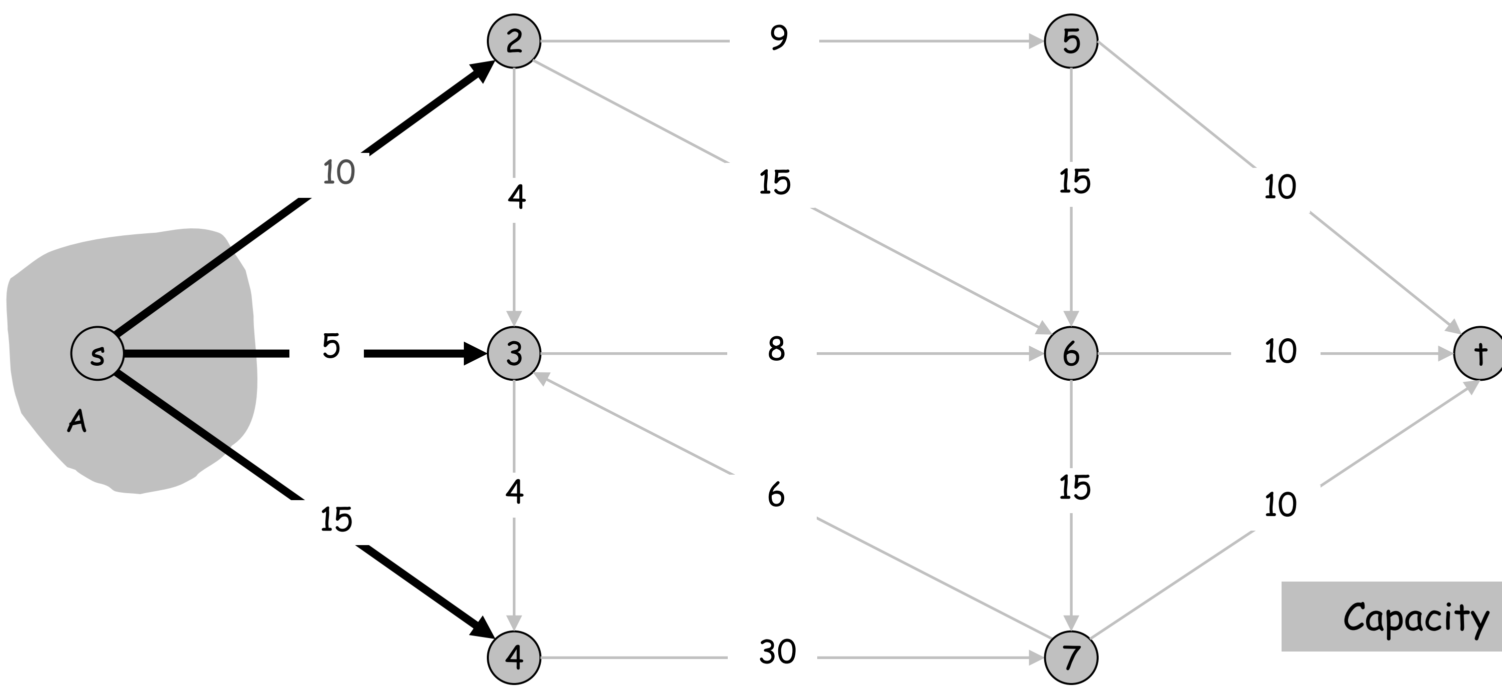
$$
\begin{aligned}
v(f) &= \sum_{e \text{ out of } s} f(e) \\
&= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \right) \\
&= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e).
\end{aligned}
$$

by flow conservation, all terms except v = s are 0 →

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



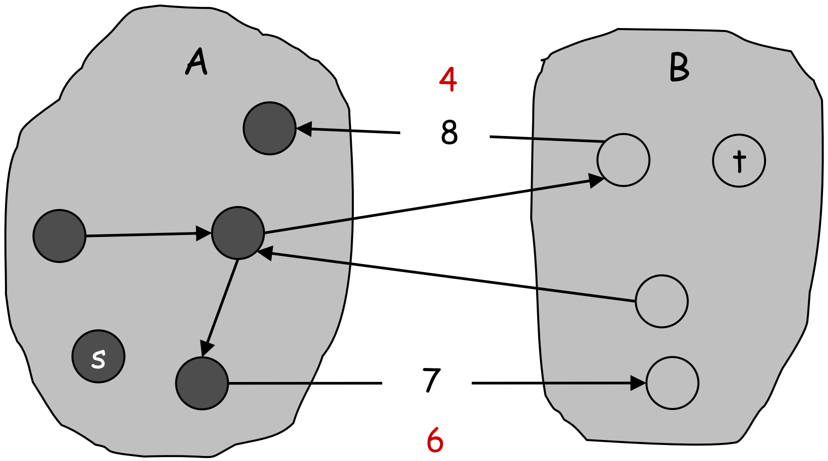
Capacity = 30

Flows and Cuts

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

Pf.

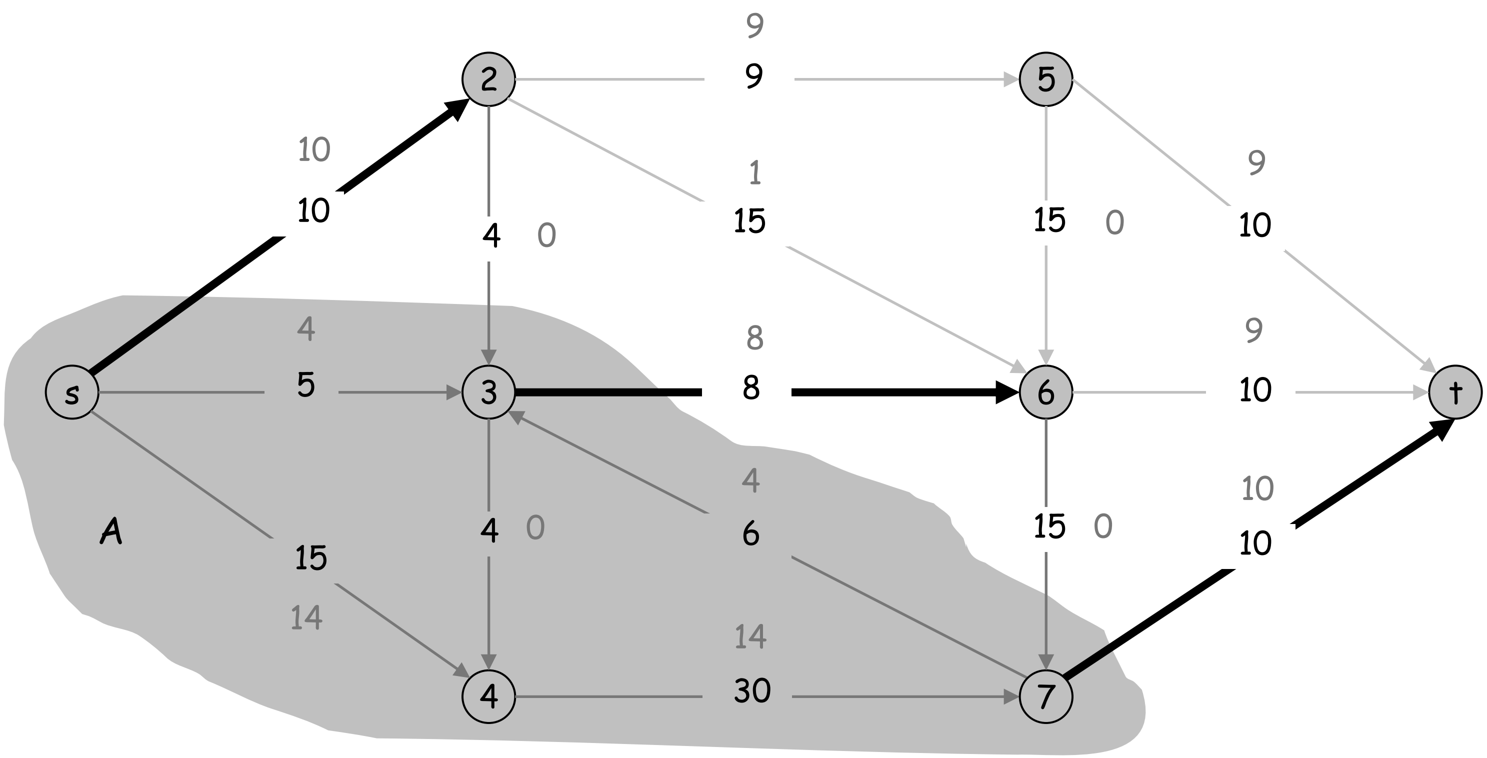
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\le \sum_{e \text{ out of } A} f(e) \\ &\le \sum_{e \text{ out of } A} c(e) \\ &= \operatorname{cap}(A, B) \quad \blacksquare \end{aligned}$$



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.

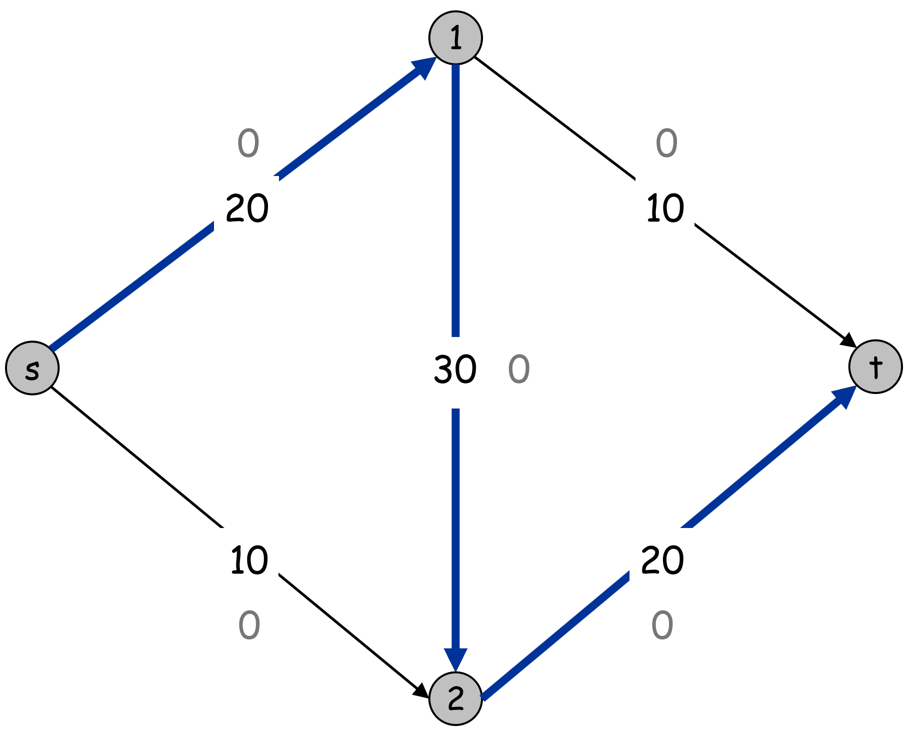
Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



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.

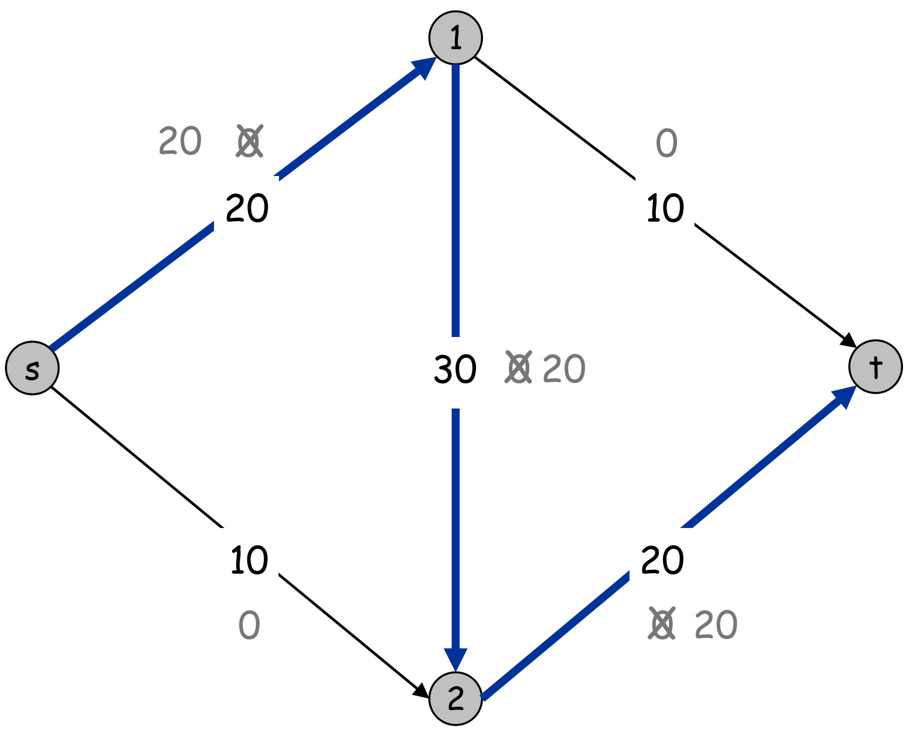


Flow value = 0

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.

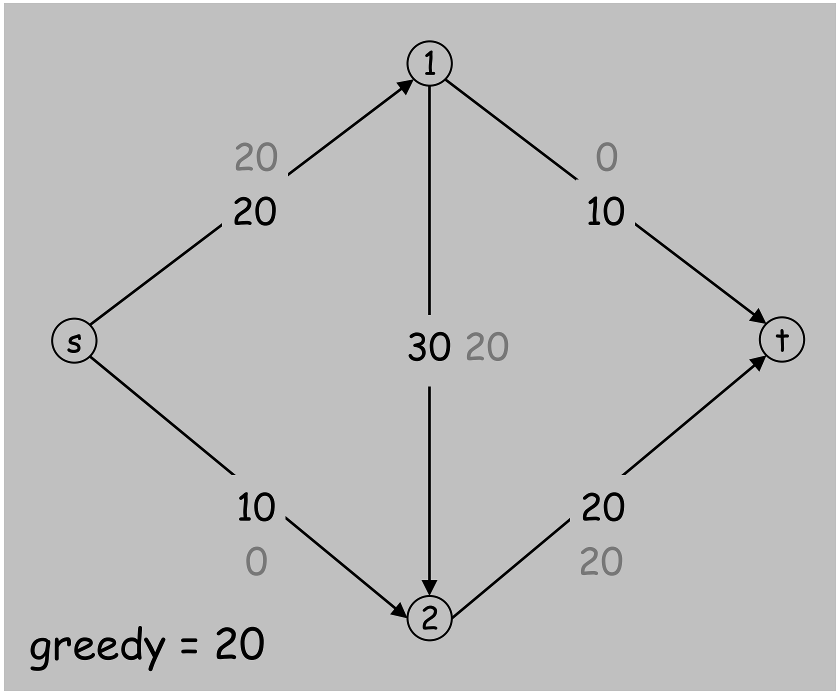


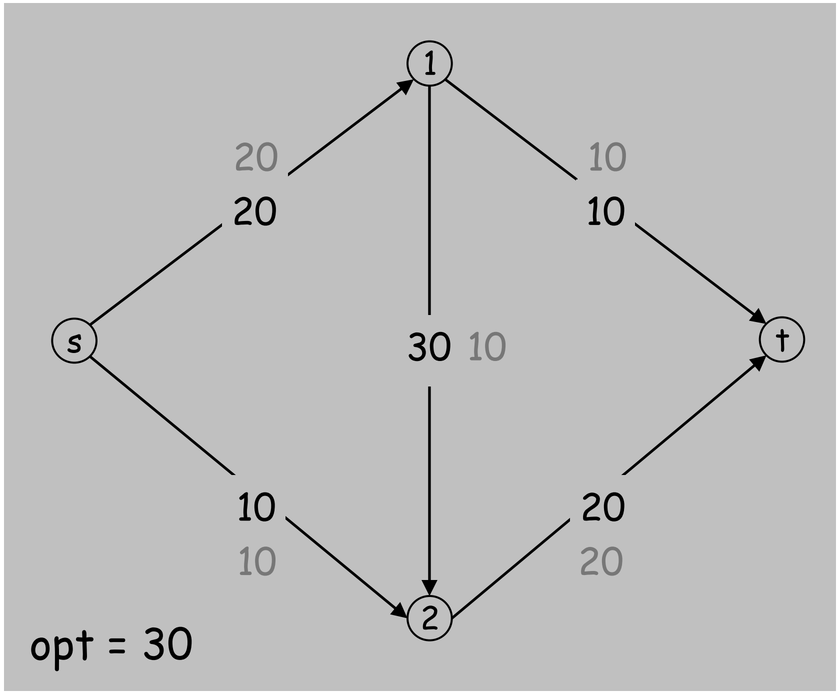
Flow value = 20

Towards a Max Flow Algorithm

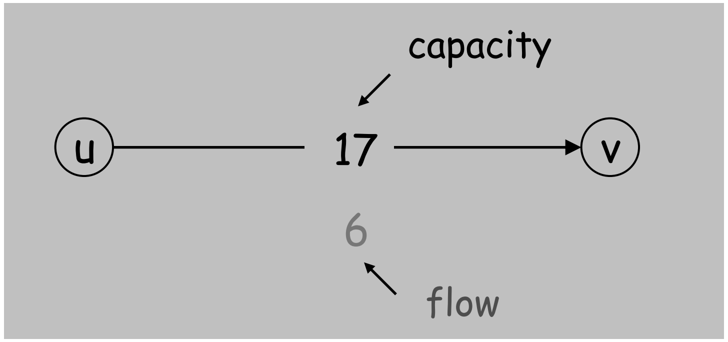
Greedy algorithm.
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

locally optimality $\not\Rightarrow$ global optimality



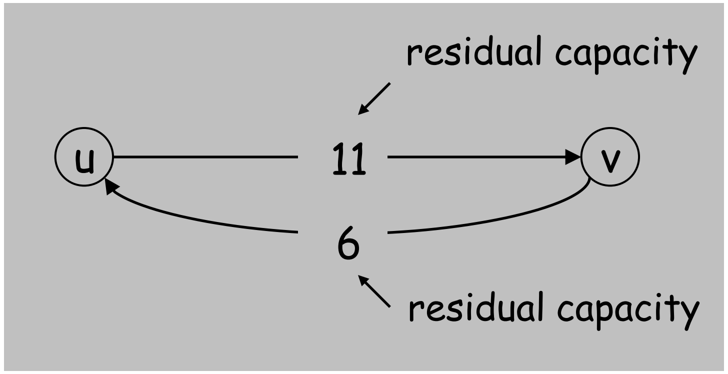


Residual Graph



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

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



Residual edge.

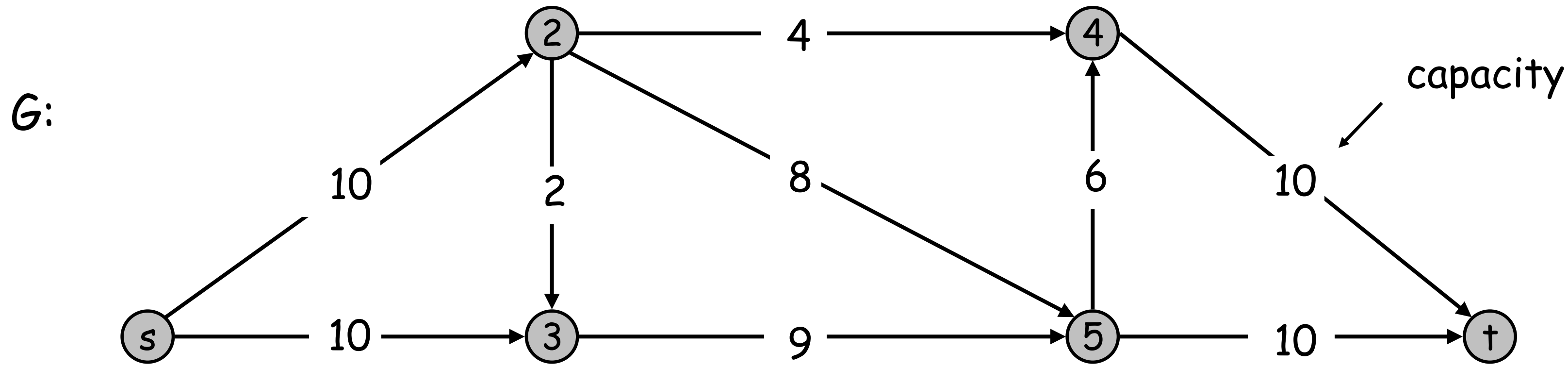
- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity:

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

Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : c(e) > 0\}$.

Ford-Fulkerson Algorithm



Augmenting Path Algorithm

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

forward edge

reverse edge

```
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
}
```

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 TFAE:

- (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.

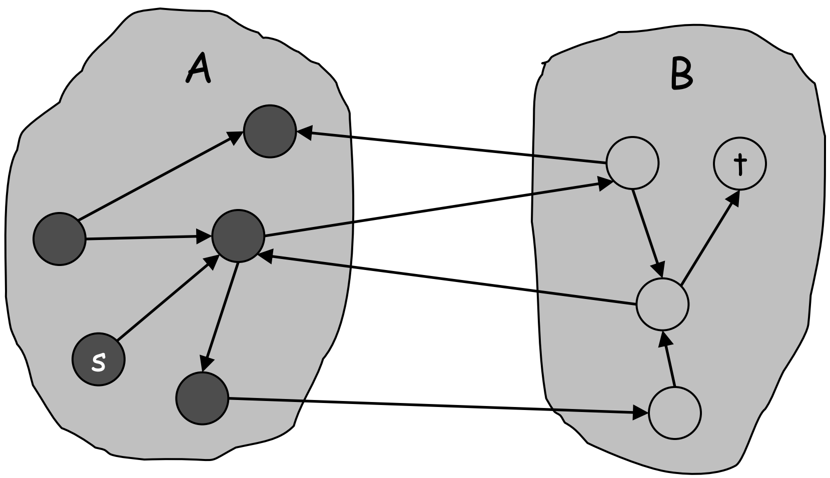
- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

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.

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



original network

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.

Theorem. The algorithm terminates in at most C iterations.
Pf. Each augmentation increase value by at least 1. ▪

Corollary. If $C = 1$, Ford-Fulkerson Algorithm can be implemented to run in $O(mC)$ 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. ▪

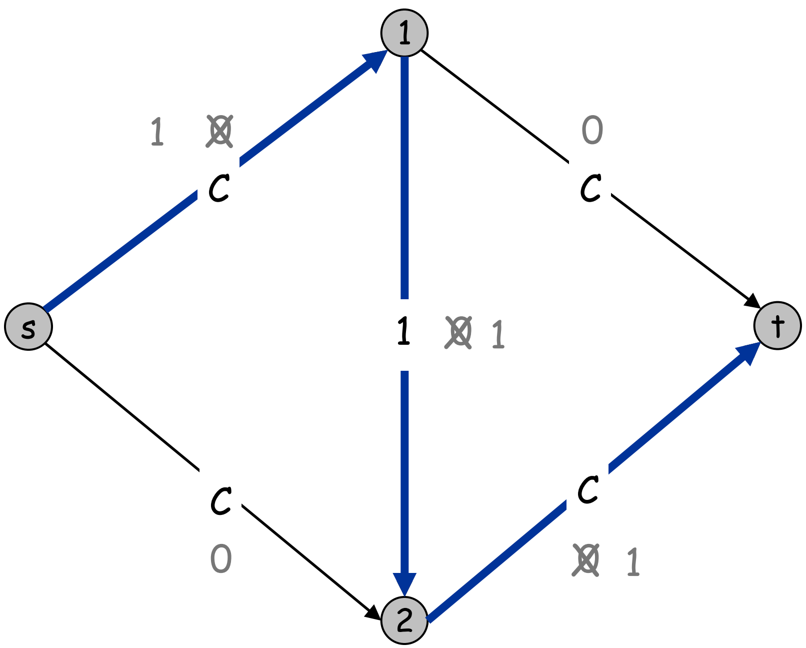
7.3 Choosing Good Augmenting Paths

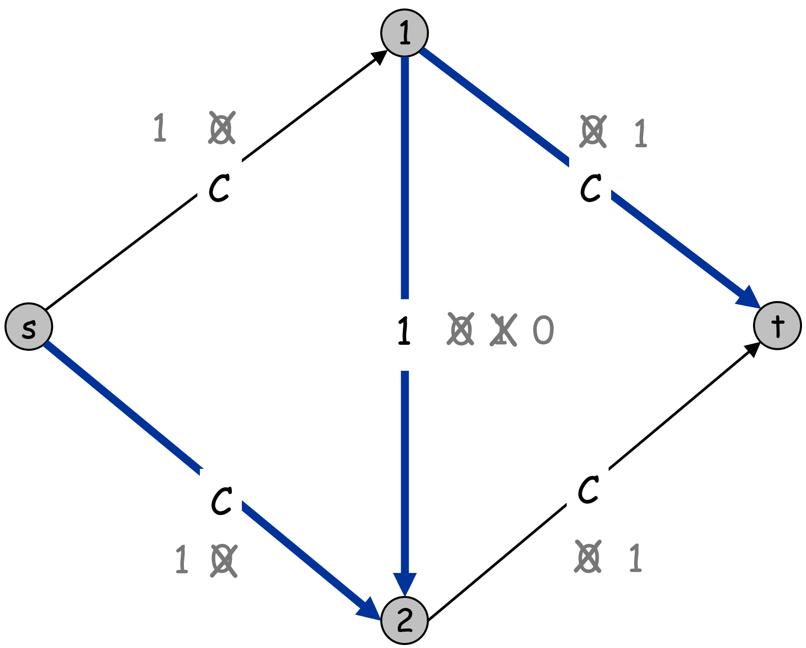
Ford-Fulkerson: Exponential Number of Augmentations

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.

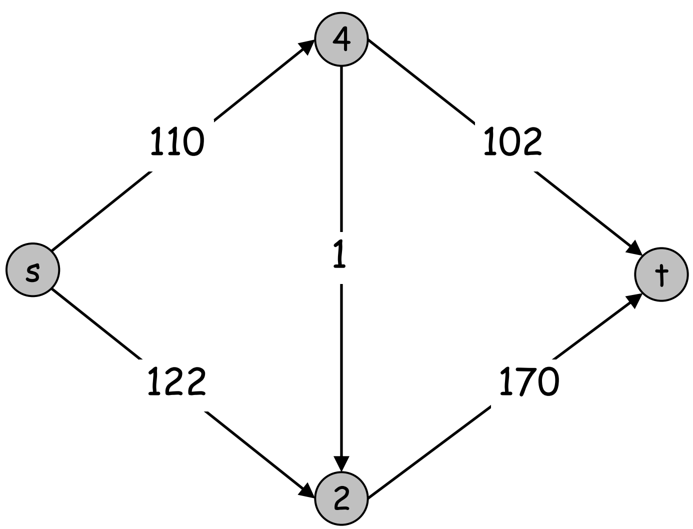
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

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

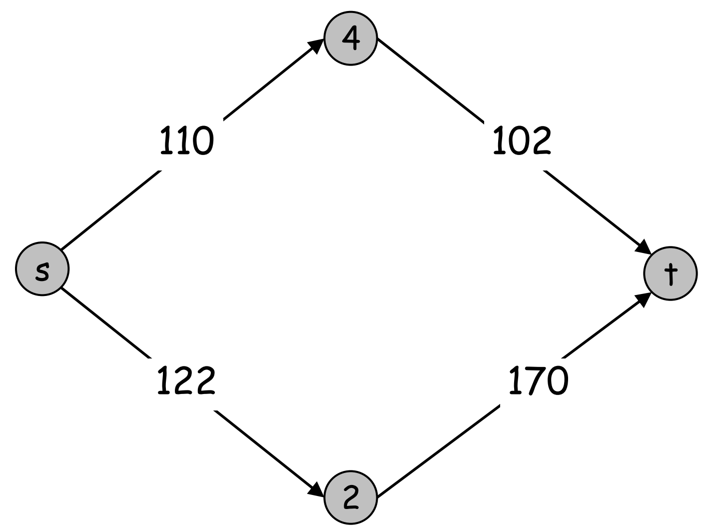
Capacity Scaling

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 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 ← smallest power of 2 greater than or equal to C
   G_f ← residual graph

   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 $1 + \lceil \log_2 C \rceil$ times.
Pf. Initially $C \le \Delta < 2C$. 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. ▪

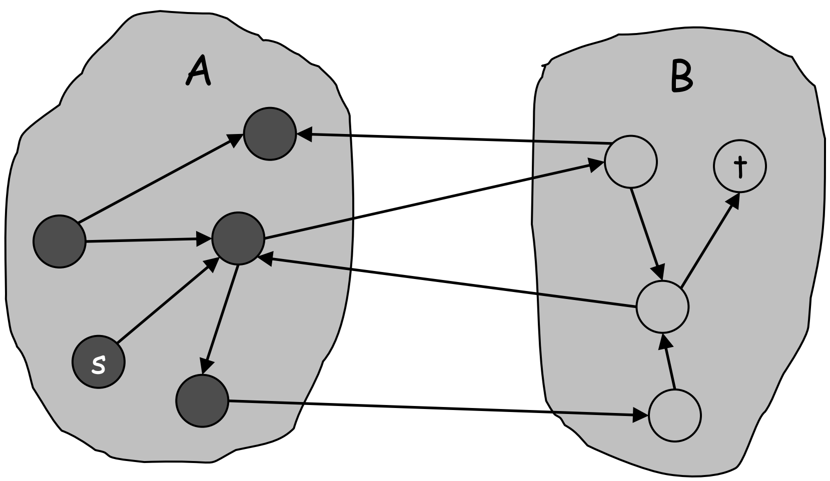
Capacity Scaling: Running Time

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 Δ.

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 Δ.
- 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.

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



original network

7.5 A First Application: The Bipartite Matching Problem

-Form the flow network (G') using the bipartite graph

-Suppose there is a flow f' in G' of value k. Since all capacities are 1, this meanf f(e) is equal to 0 or 1 for each edge e.

-Consider the set M' of edges of the form (x, y) on which the flow value is 1.

There are three simple facts about the set M':

1)M' contains k edges

2)Each edge in X is the tail of at most one edge in M'

3) Each node in Y is the head of at most one edge in M'

(7.38) The Ford-Fulkerson Algorithm can be used to find a maximum matching in a bipartite gramp in O(mn) time.

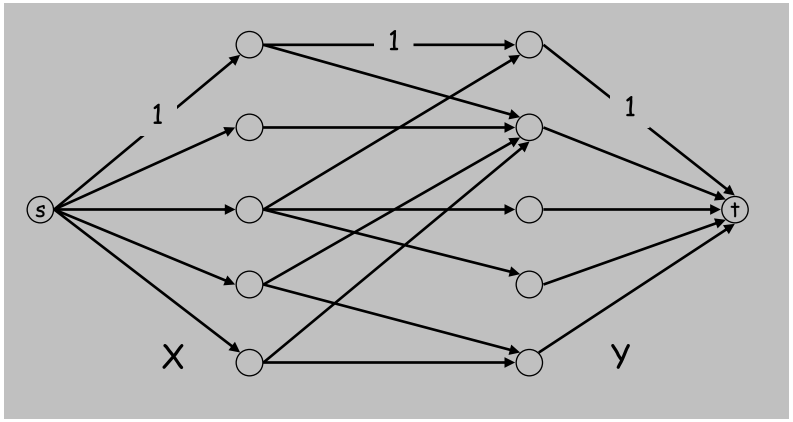
Bipartite Matching

Bipartite matching. Can solve via reduction to max flow.

Flow. During Ford-Fulkerson, all capacities and flows are 0/1. Flow corresponds to edges in a matching M.

Residual graph G_M simplifies to:

- If $(x, y) \notin M$, then (x, y) is in G_M.
- If $(x, y) \in M$, the (y, x) is in G_M.

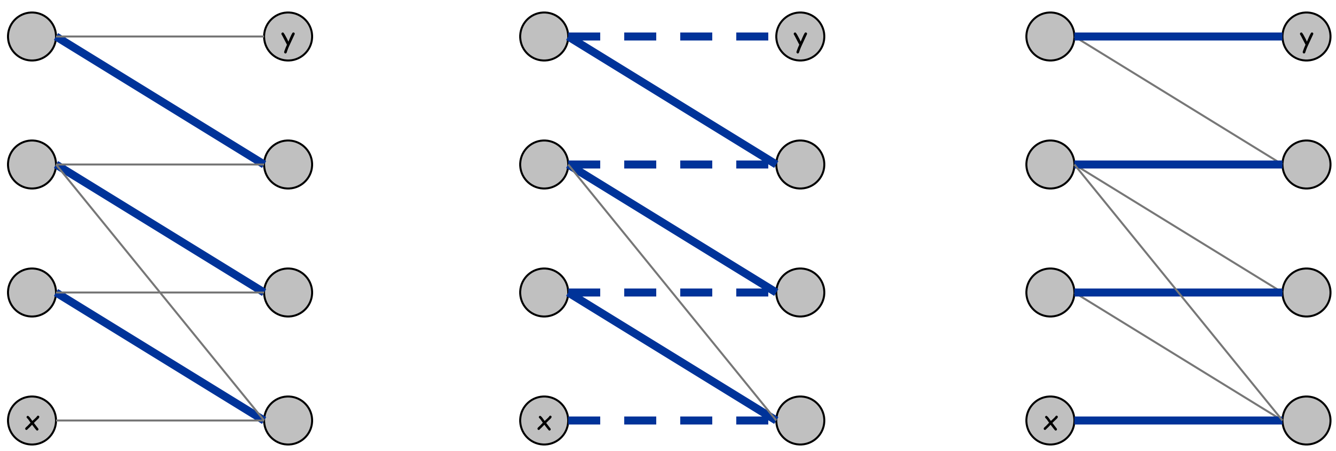


Augmenting path simplifies to:

- Edge from s to an unmatched node $x \in X$.
- Alternating sequence of unmatched and matched edges.
- Edge from unmatched node $y \in Y$ to t.

Alternating Path

Alternating path. Alternating sequence of unmatched and matched edges, from unmatched node $x \in X$ to unmatched node $y \in Y$.



matching M   alternating path   matching M'

Extensions: The Structure of Bipartite Graphs with No Perfect Matching

(7.39) If a bipartite graph $G=(V,E)$ with two sides X and Y has a perfect matching, then for all $A \subseteq X$ we must have $|\Gamma(A)| >= |A|$.

(7.40) Assume that the bipartite graph $G=(V, E)$ has two sides with X and Y such that $|X| = |Y|$. Then the graph G either has a perfect matching or there is a subset $A \subseteq X$ such that $|\Gamma(A)| < |A|$. A perfect matching or an appropriate matching can be found in $O(mn)$ time.