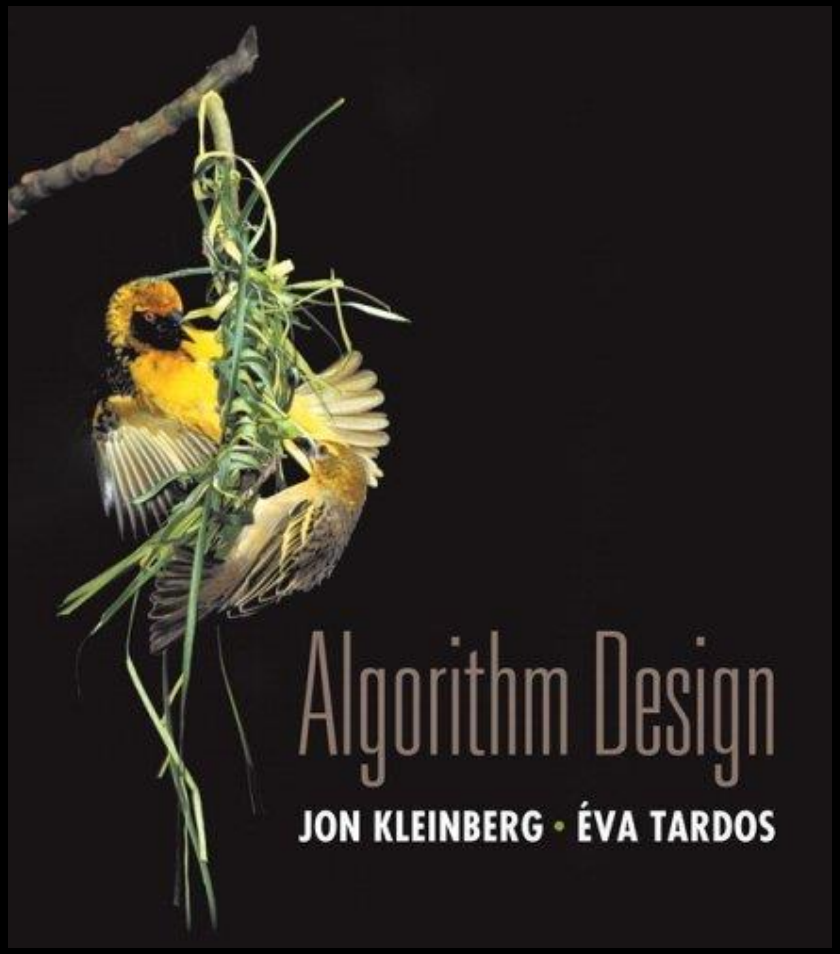


Chapter 7

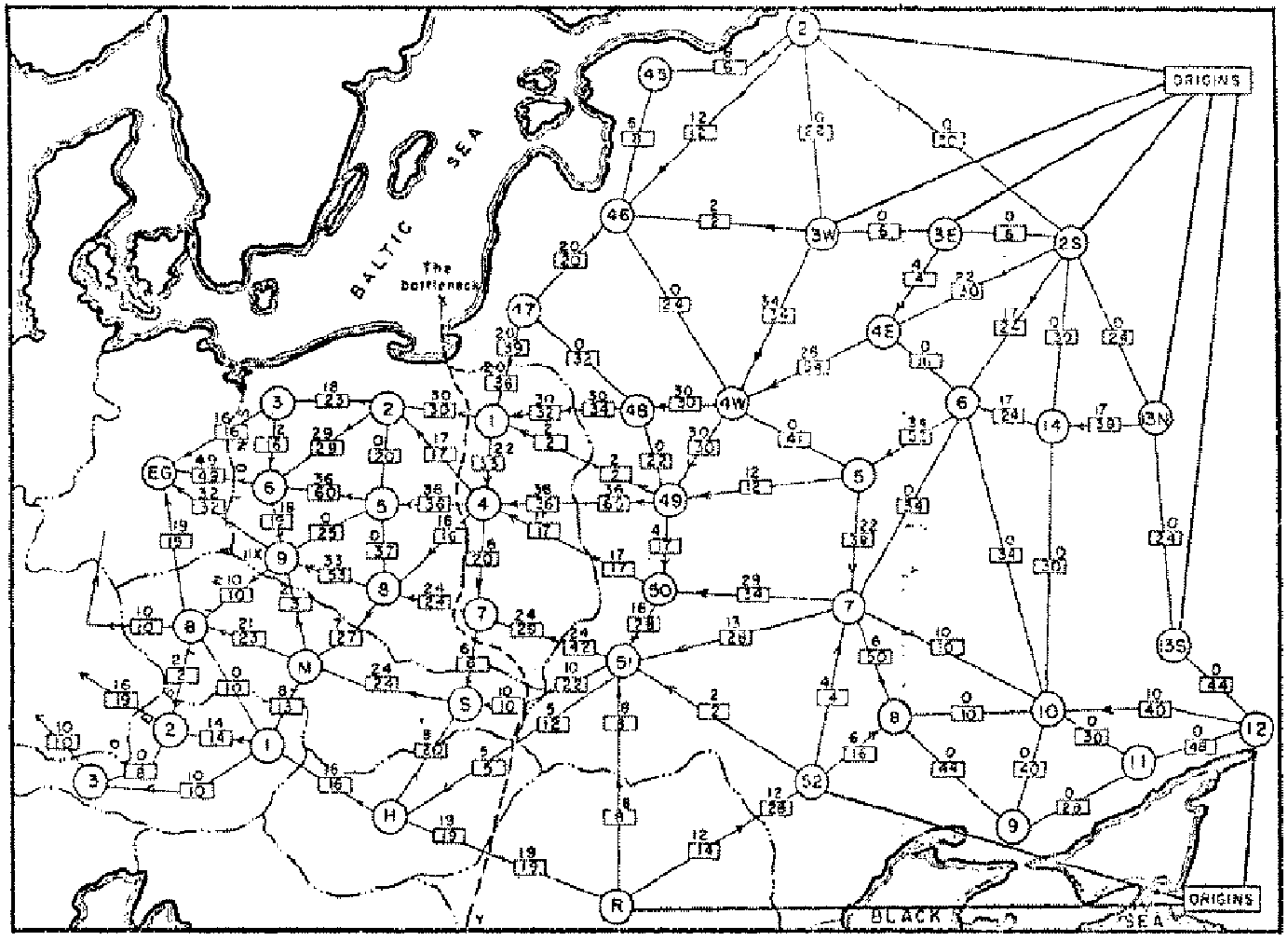
Network Flow

Maximum Flow

Goals/why study max flow

- Has many non-trivial application
- A taste of more advanced optimization algorithms
- **Duality**: powerful way of proving that a solution is optimal

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.

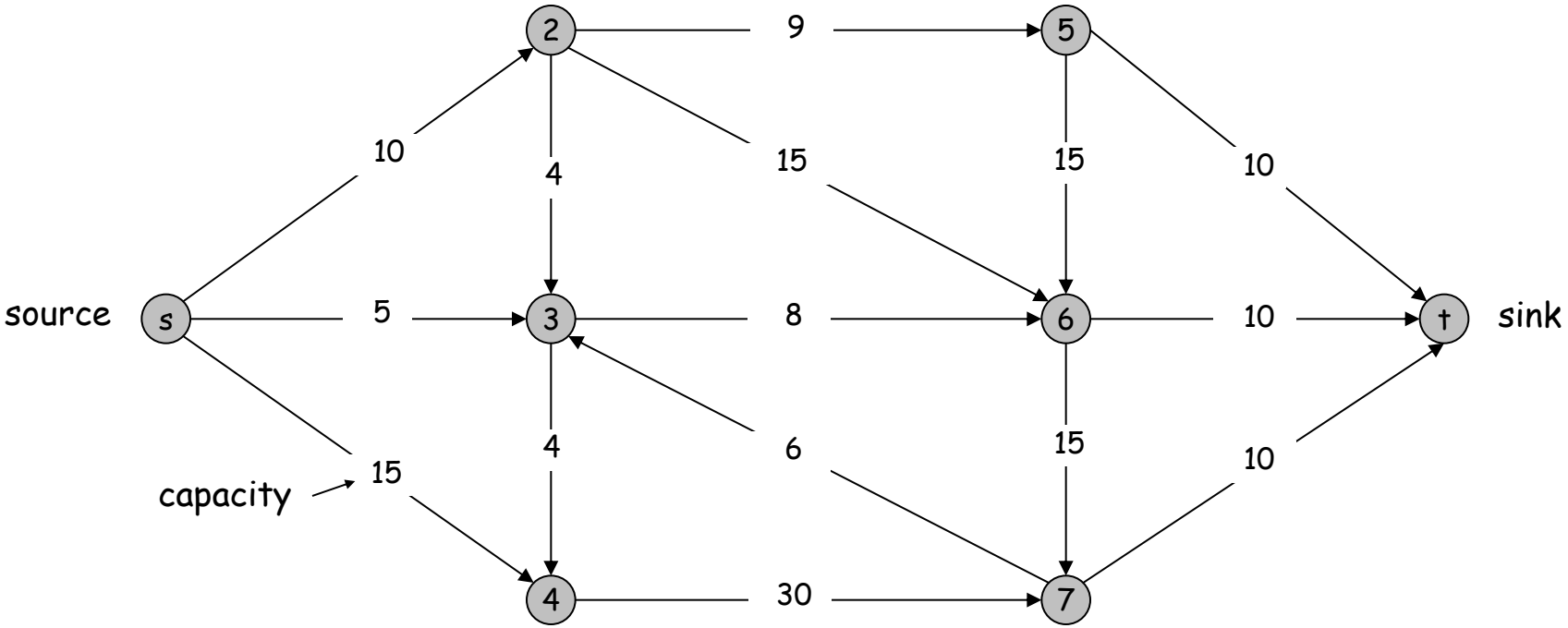
Nontrivial applications / reductions.

- 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 . . .

Max Flow 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 [no edges into s, out of t]
- c(e) = capacity of edge e.

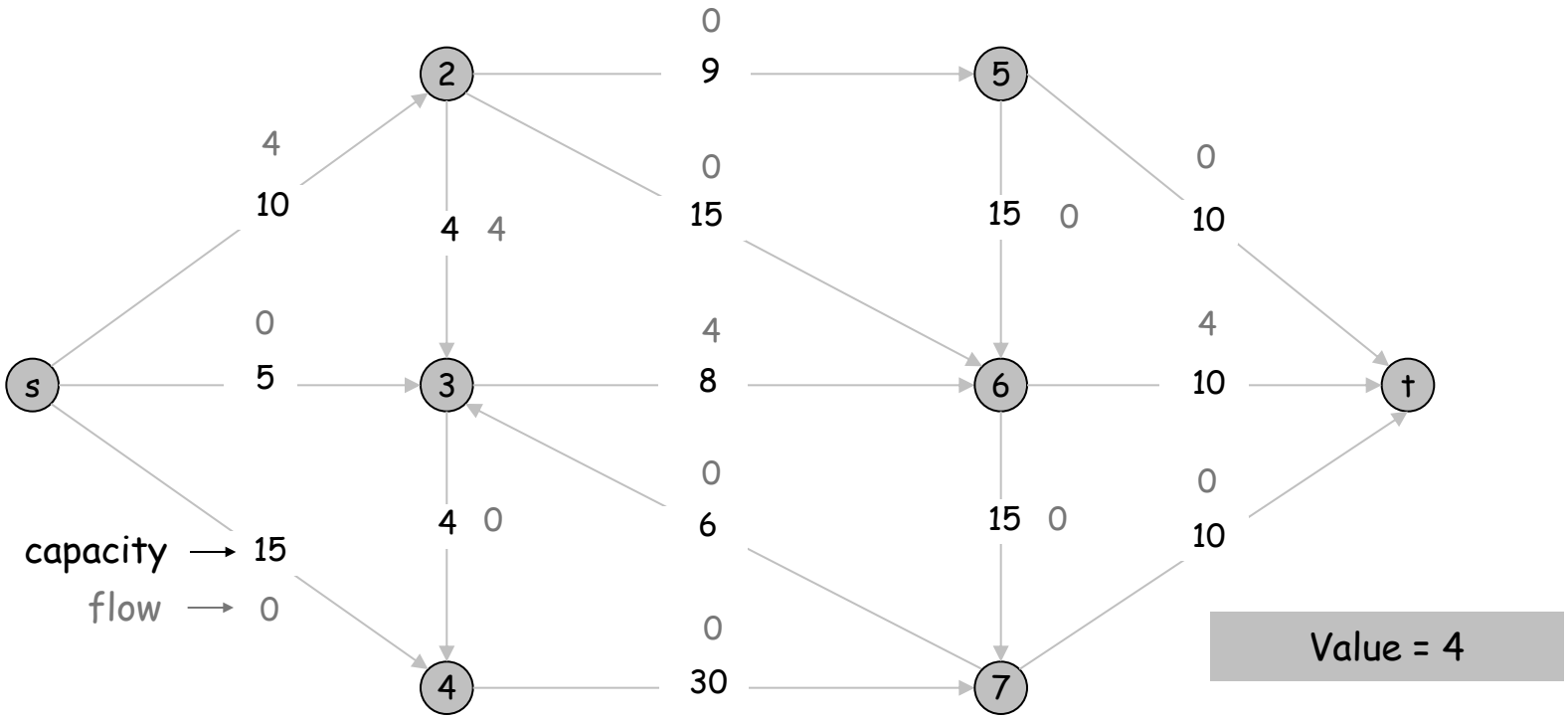


Flows

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

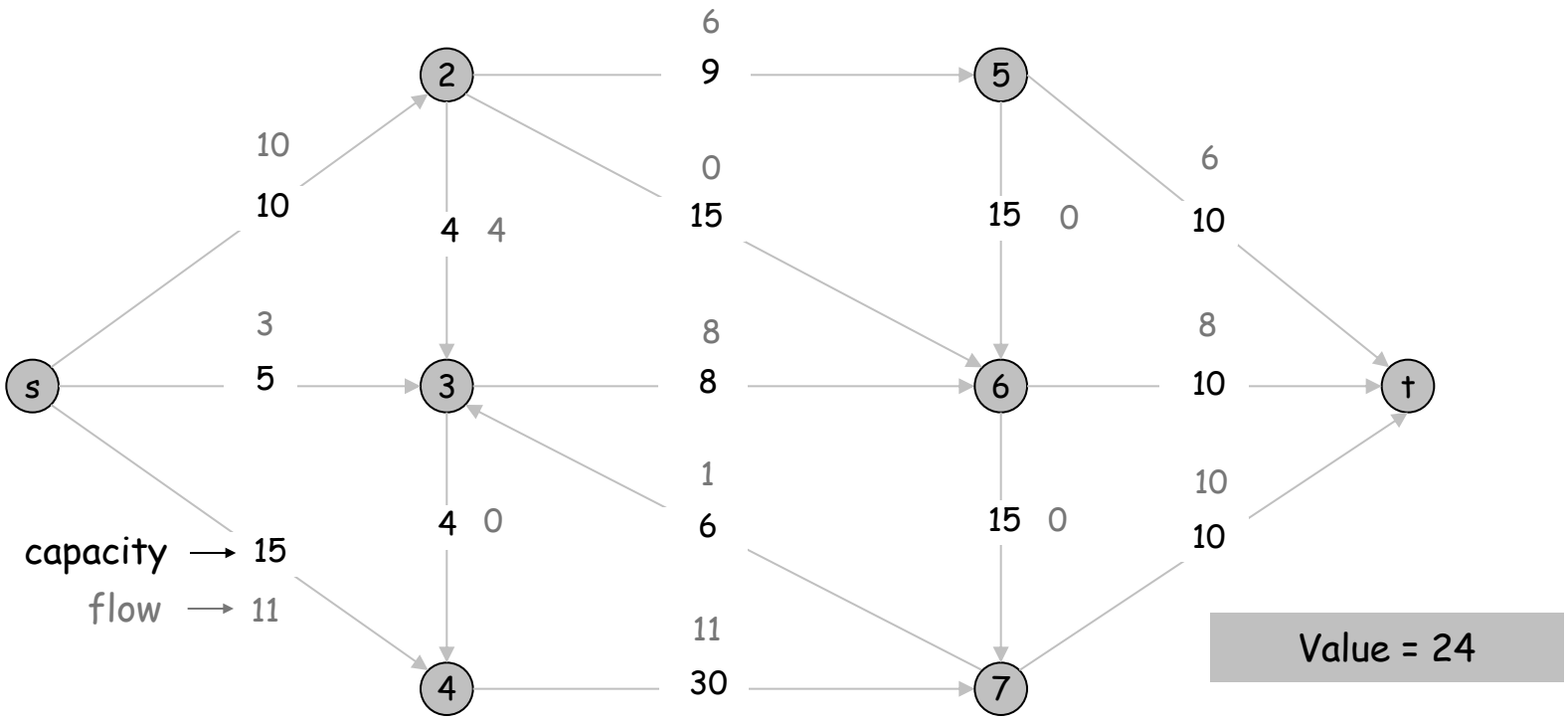


Flows

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

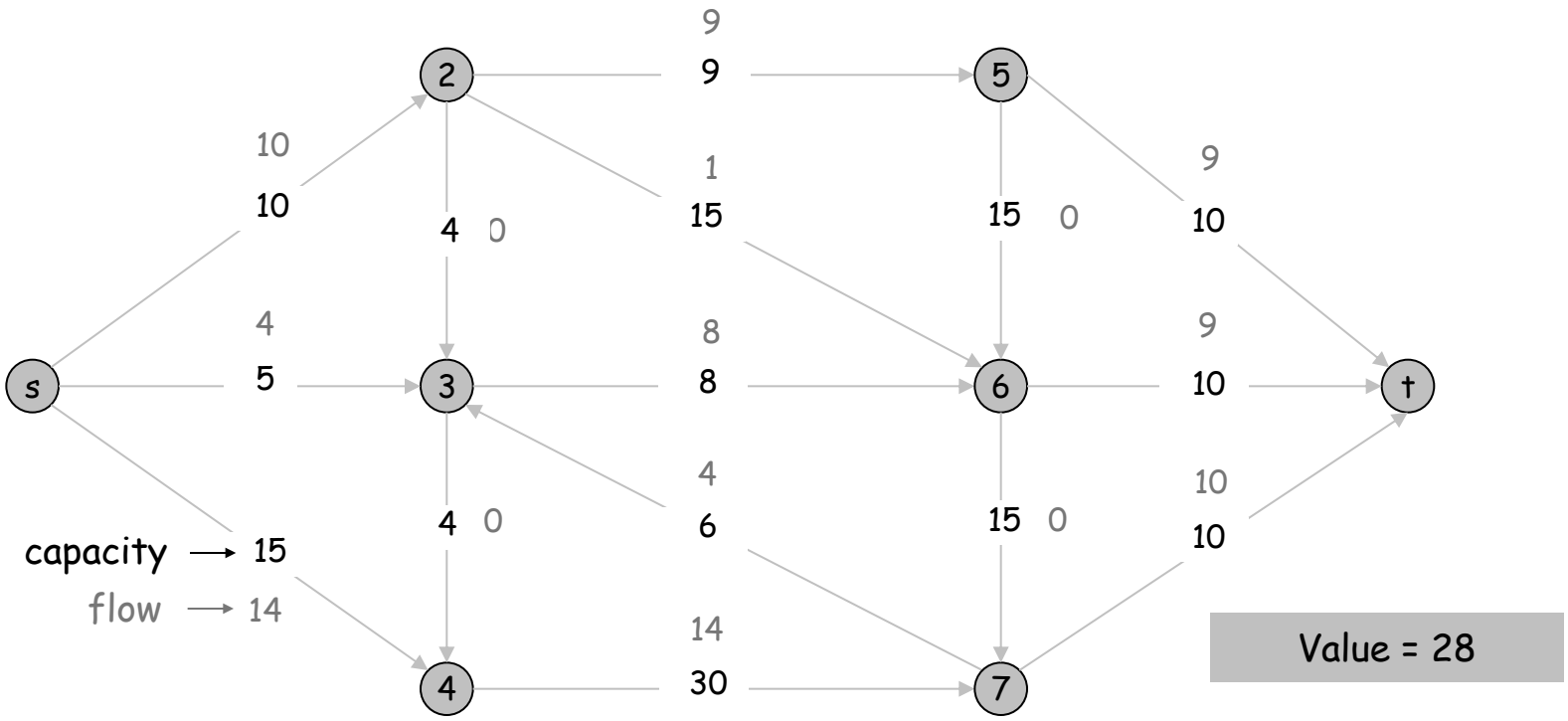
- For each e ∈ E: $0 \le f(e) \le c(e)$ (capacity)
- For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

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



Maximum Flow Problem

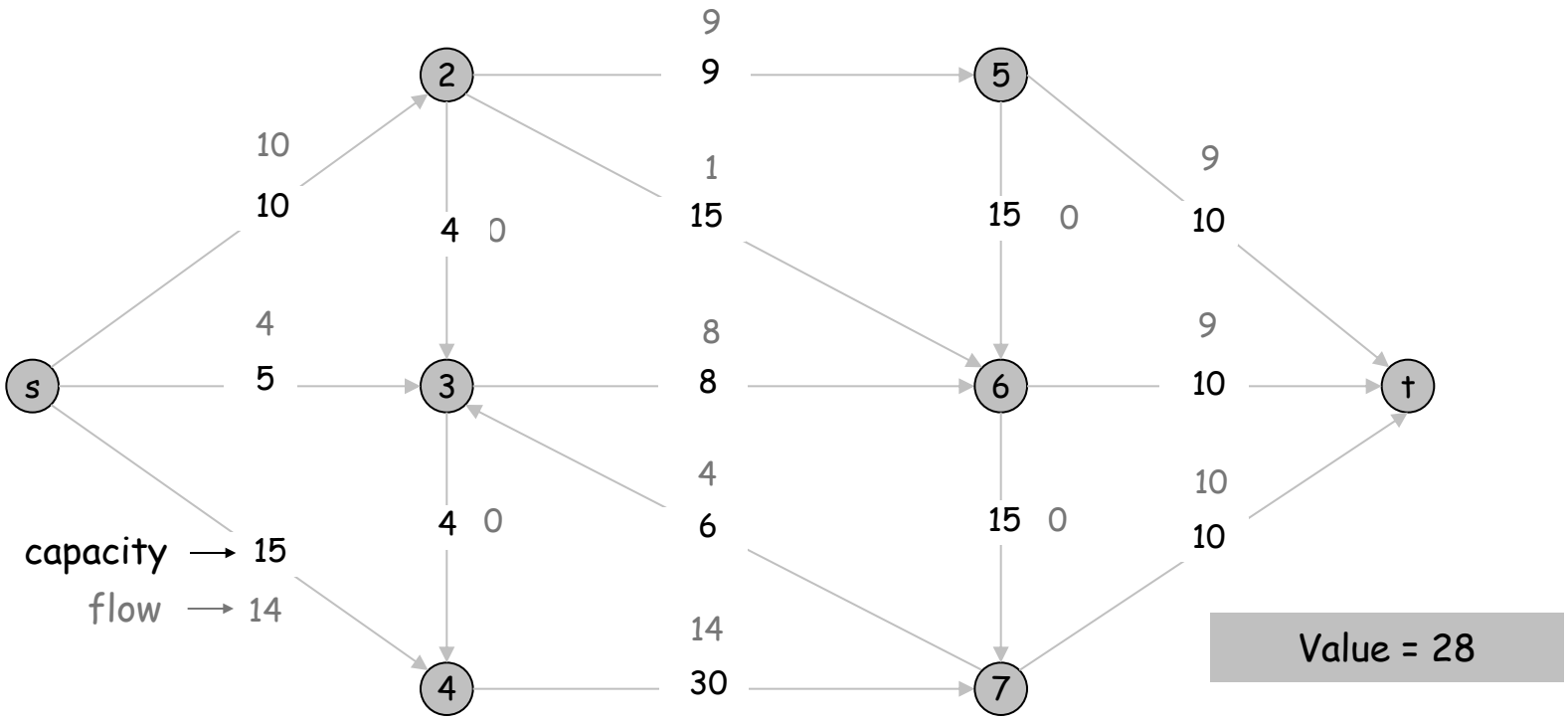
Max flow problem. Find s-t flow of maximum value.



Maximum Flow Problem

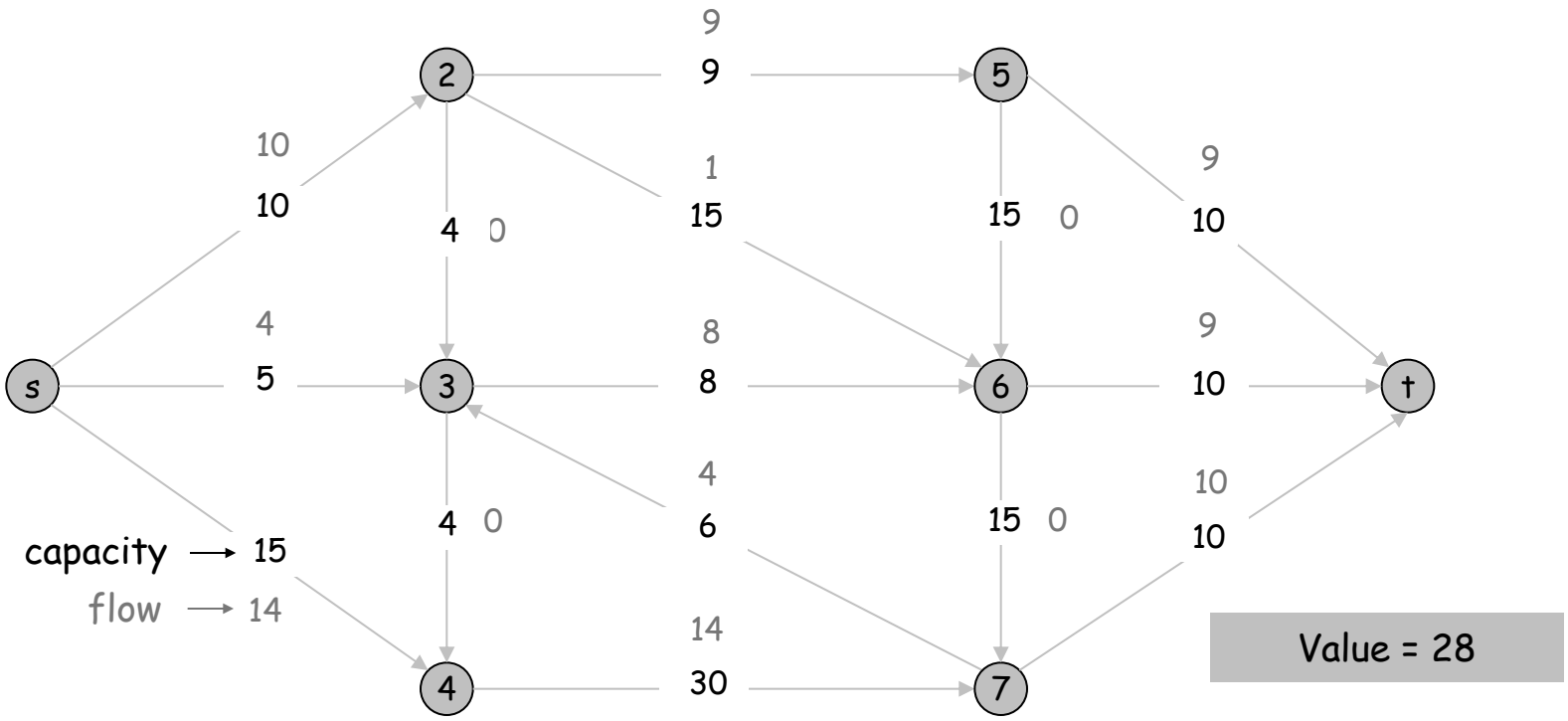
Very useful and concrete way of thinking about flows:

Flow decomposition: Every flow can be thought as union of path-flows and cycle-flows



Maximum Flow Problem

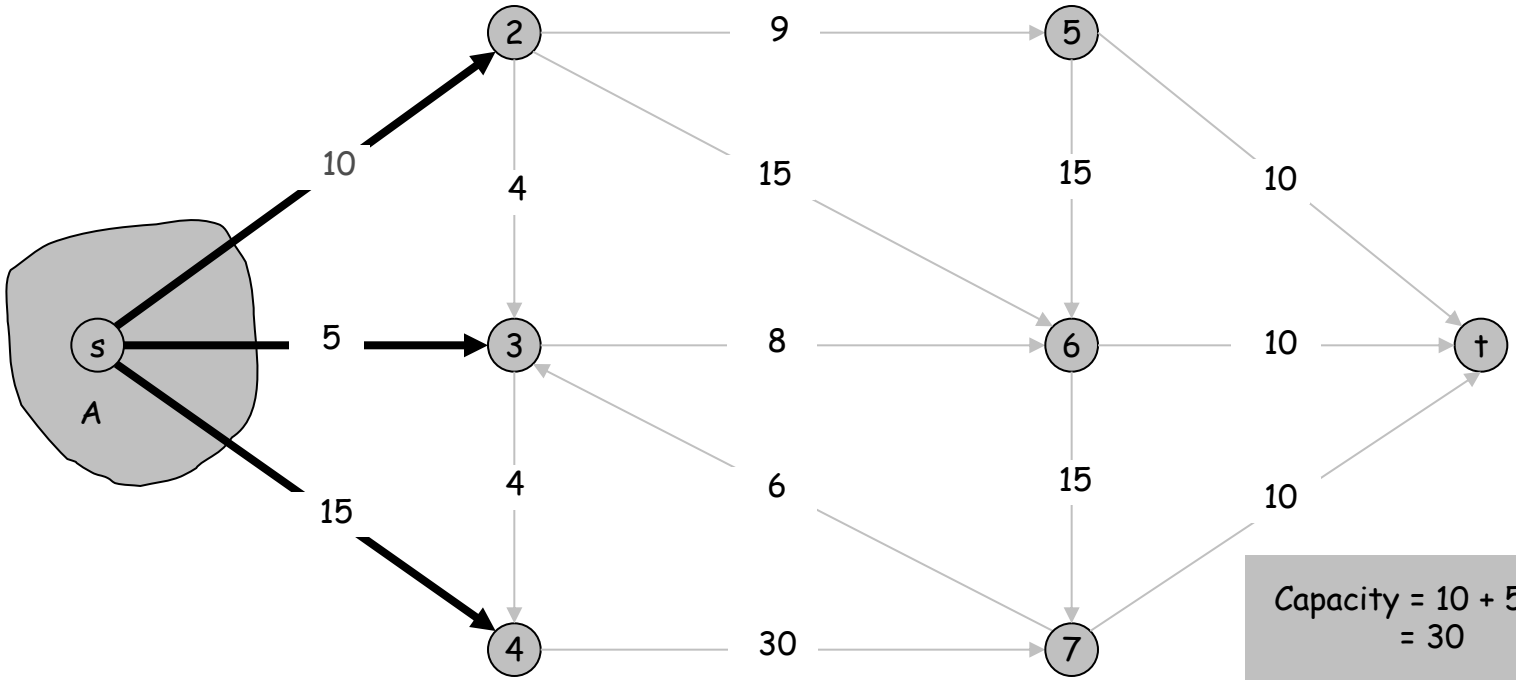
Q: Is this the max s-t flow?



Cuts

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

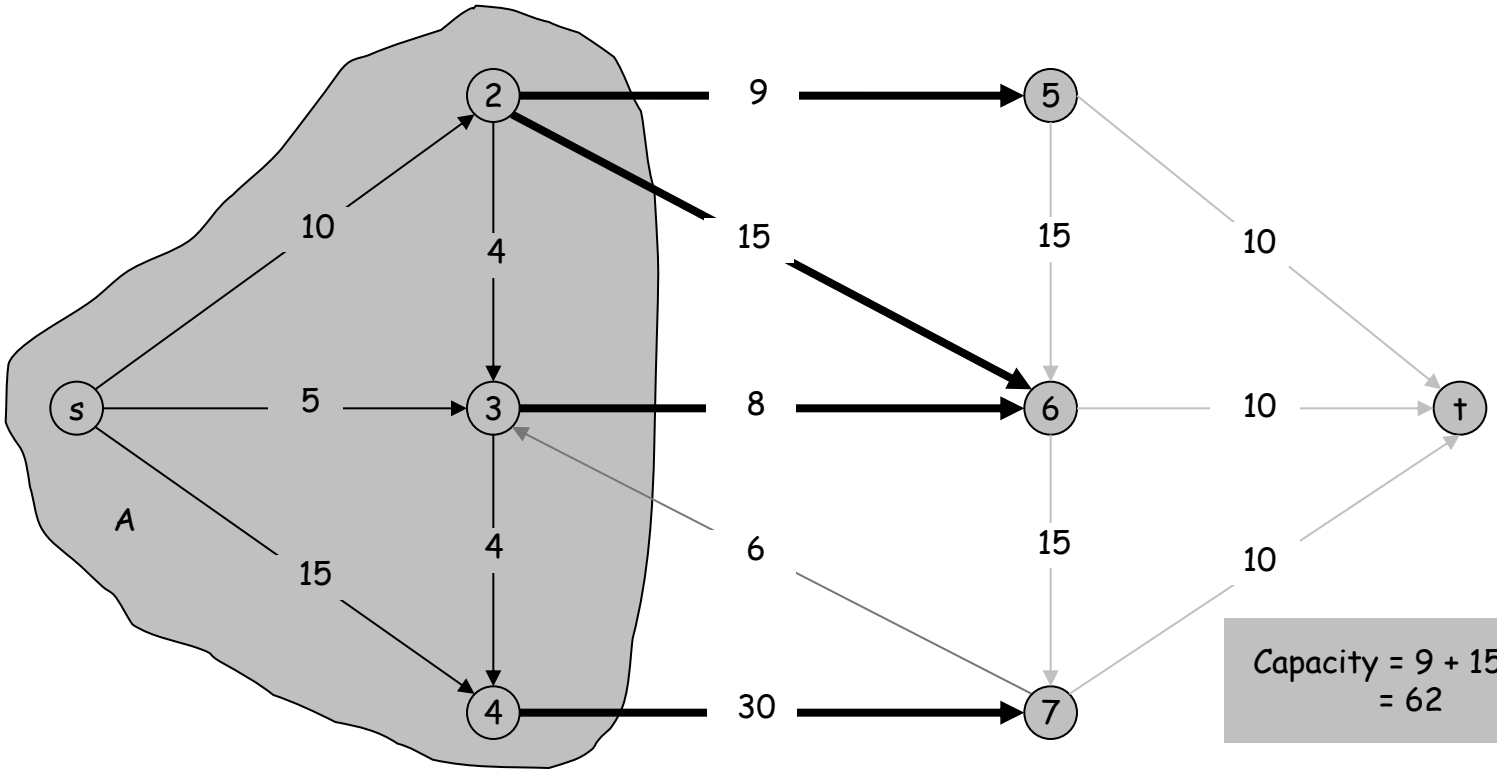
Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Cuts

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

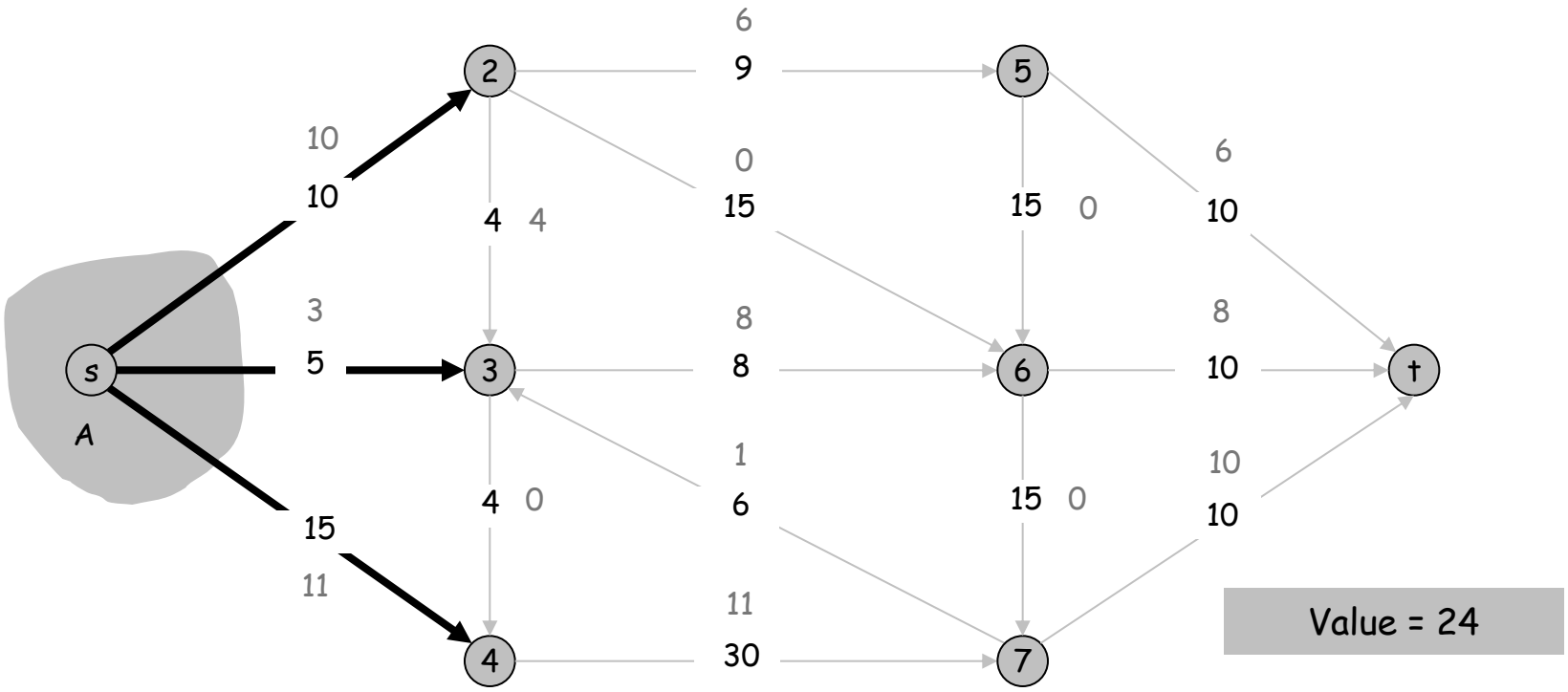
Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



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.

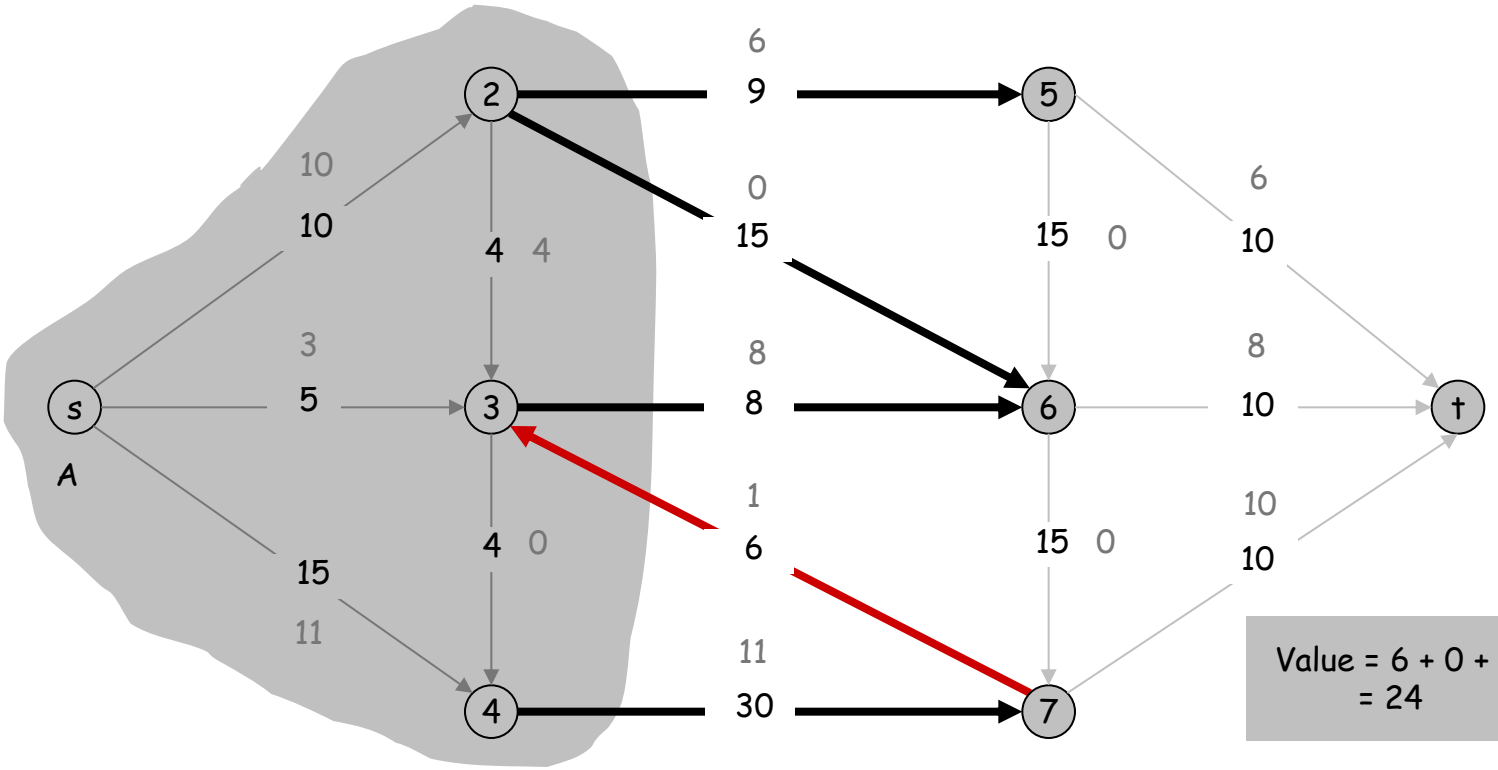
$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



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.

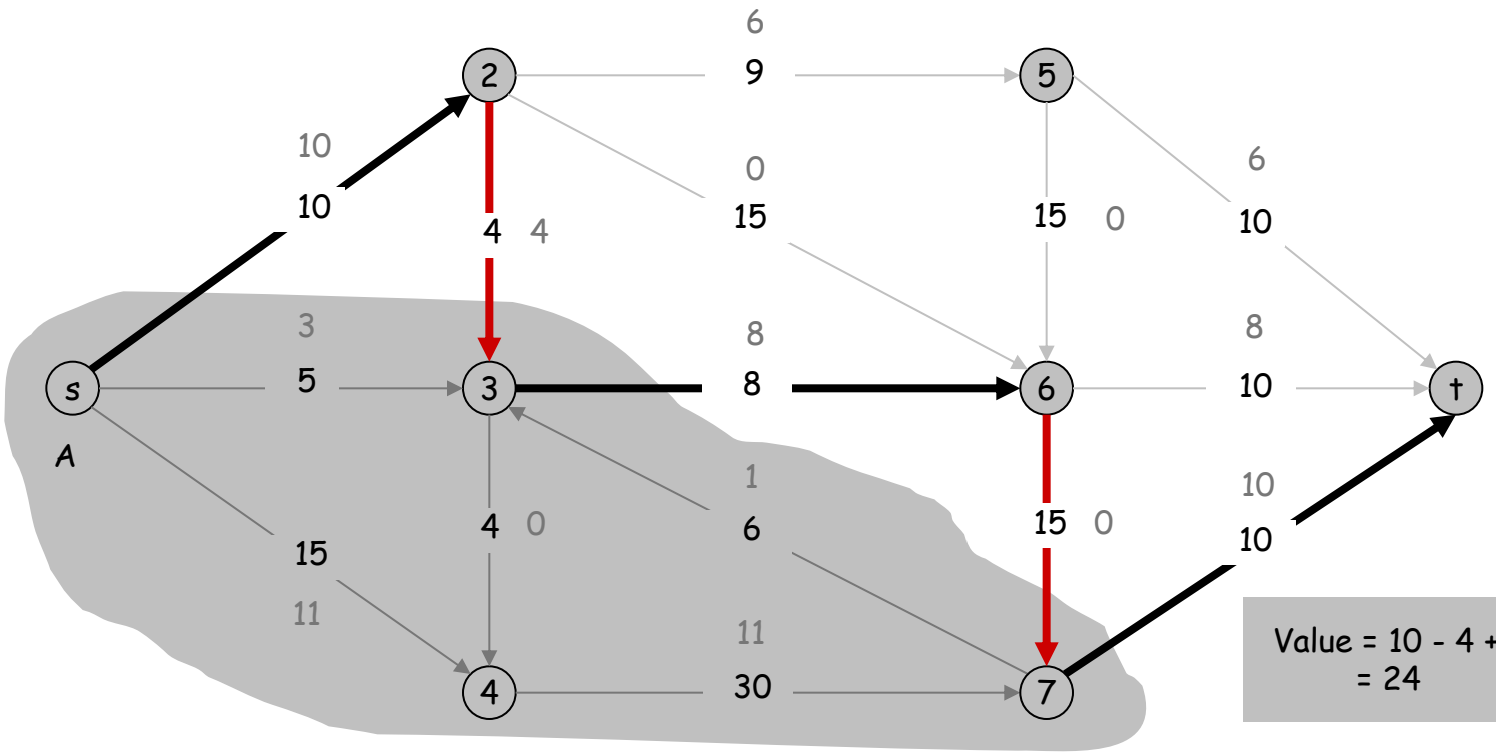
$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



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.

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



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 $\longrightarrow$

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

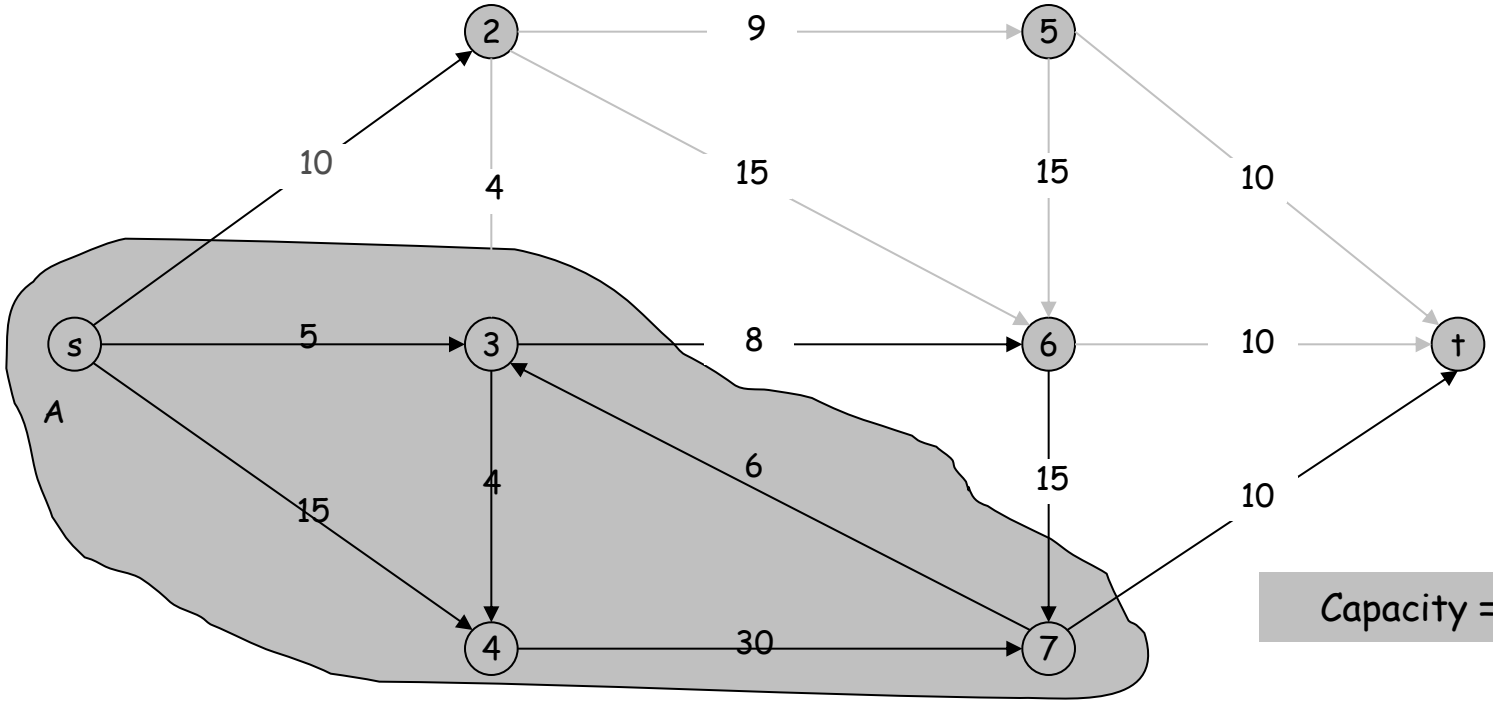
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e).$$

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.

Proof: Flow out of s = flow out of cut (A,B) <= capacity of cut (A,B)

Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28

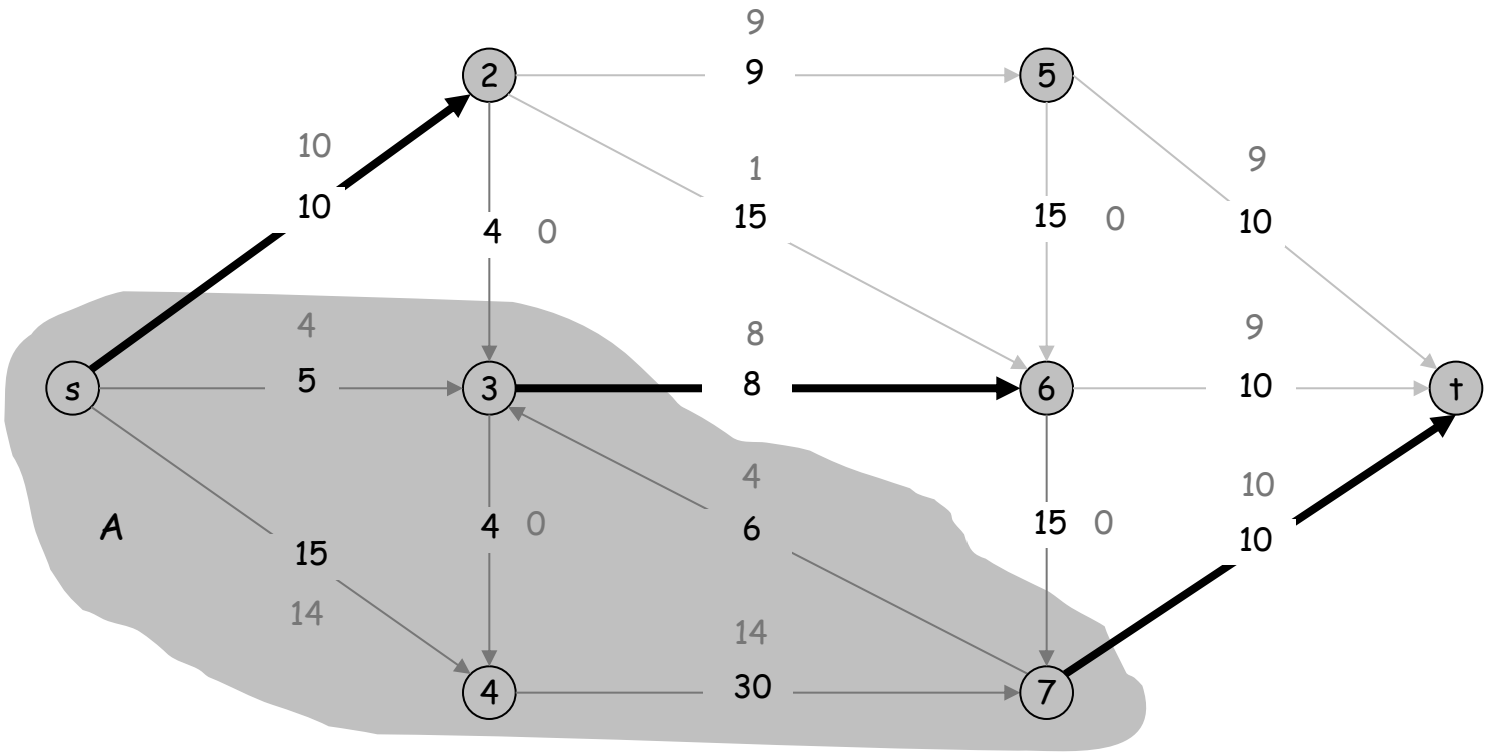


Certificate of Optimality

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.

This shows that the flow for our initial example is optimal:

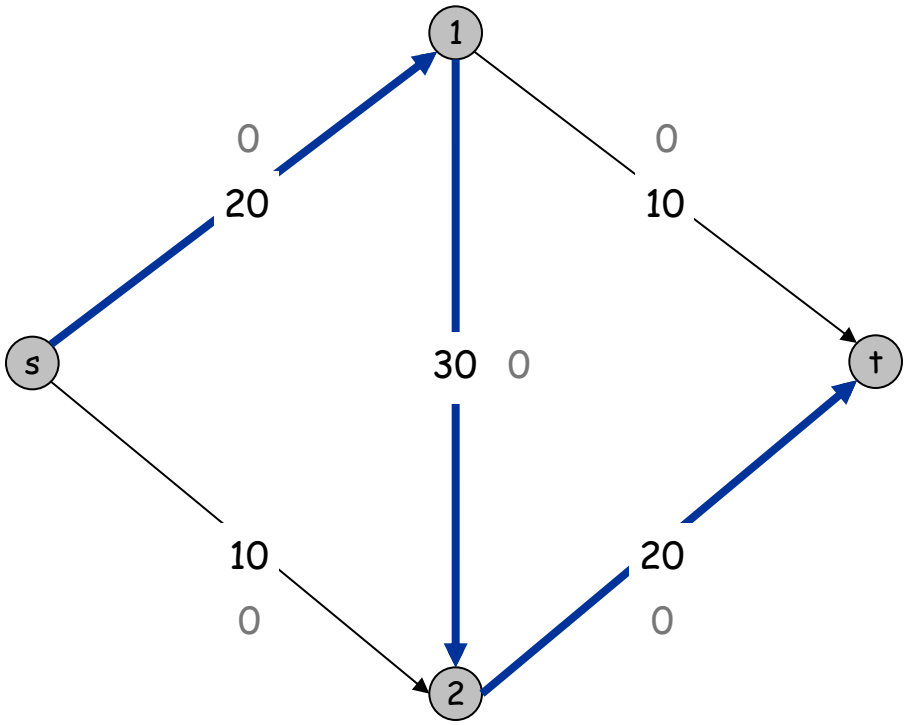
Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



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.

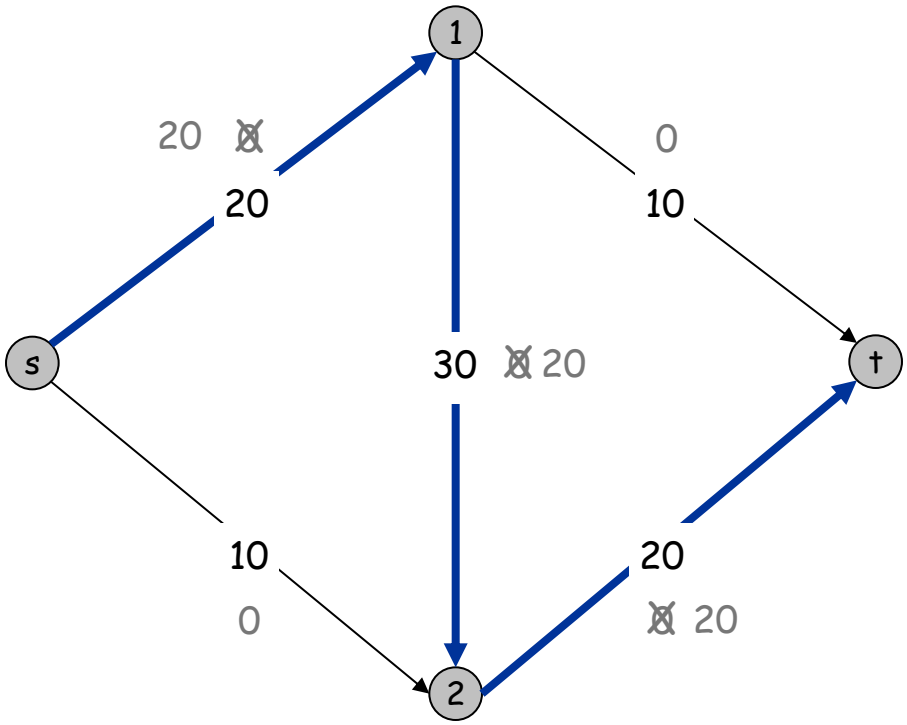


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with f(e) = 0 for all edge e ∈ 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.



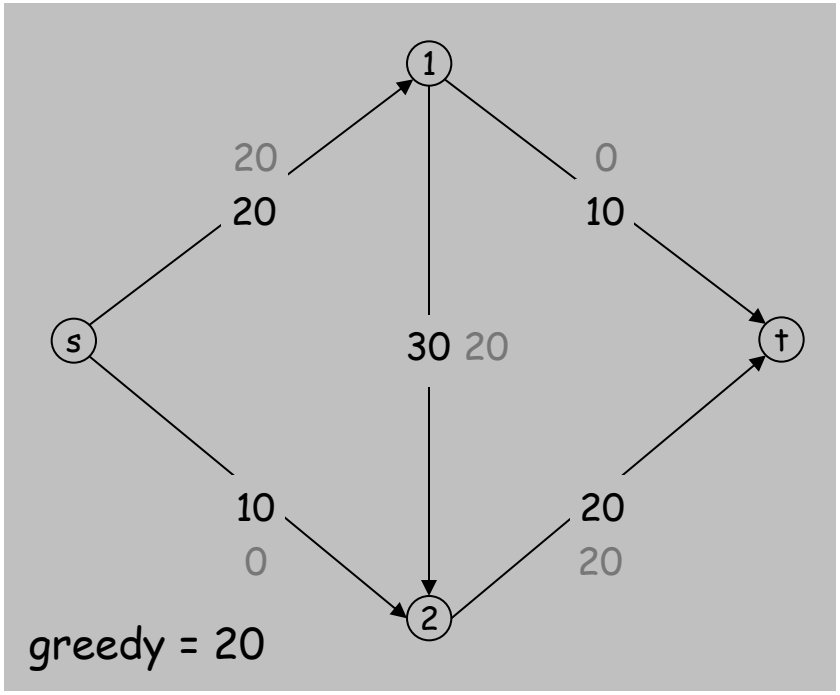
Flow value = 20

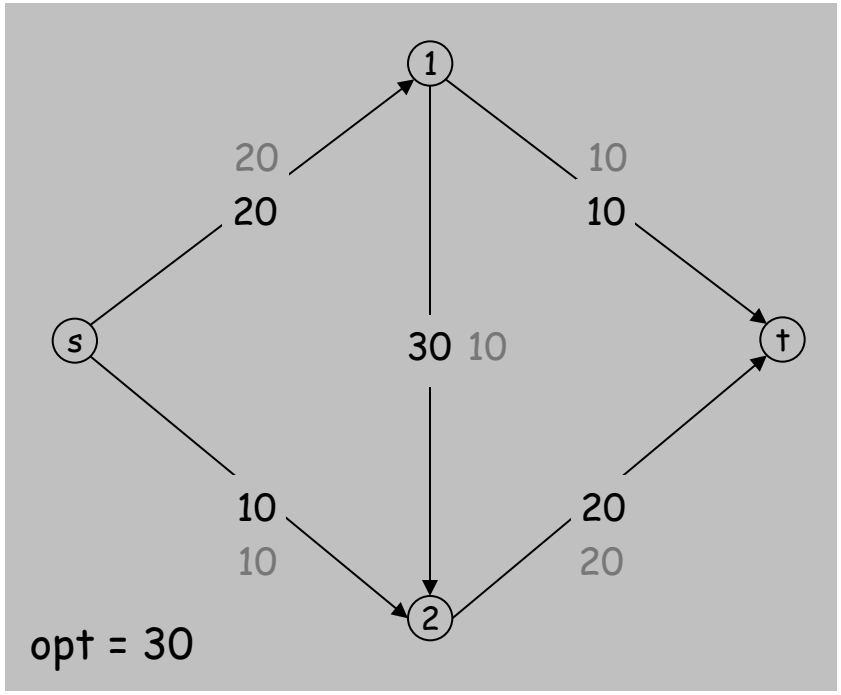
Towards a Max Flow Algorithm

Greedy algorithm.

- Start with f(e) = 0 for all edge e ∈ 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 ⇏ global optimality

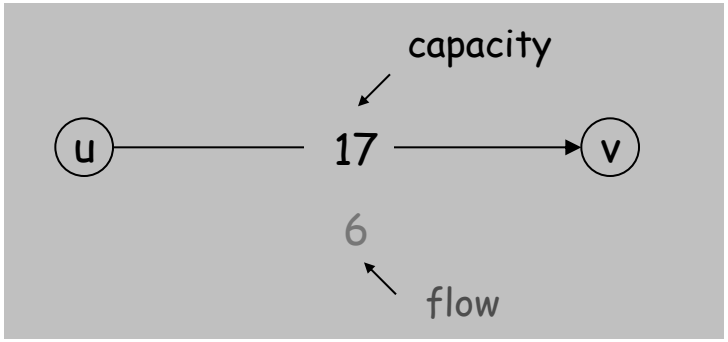




Residual Graph

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

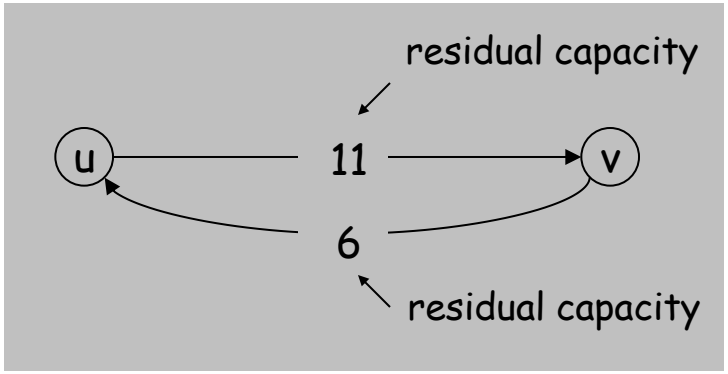
- Flow f(e), capacity c(e).



Residual edges.

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

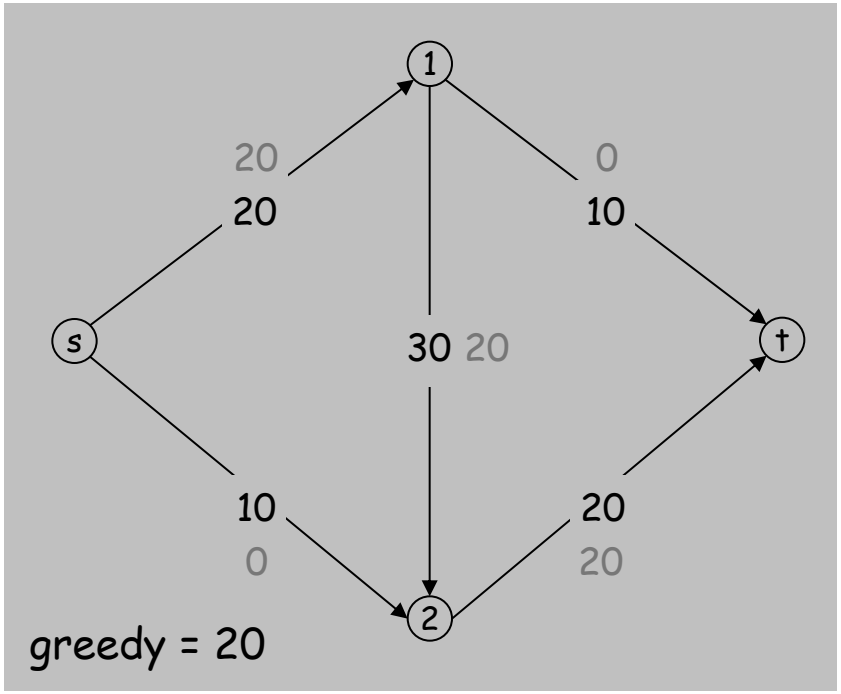
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

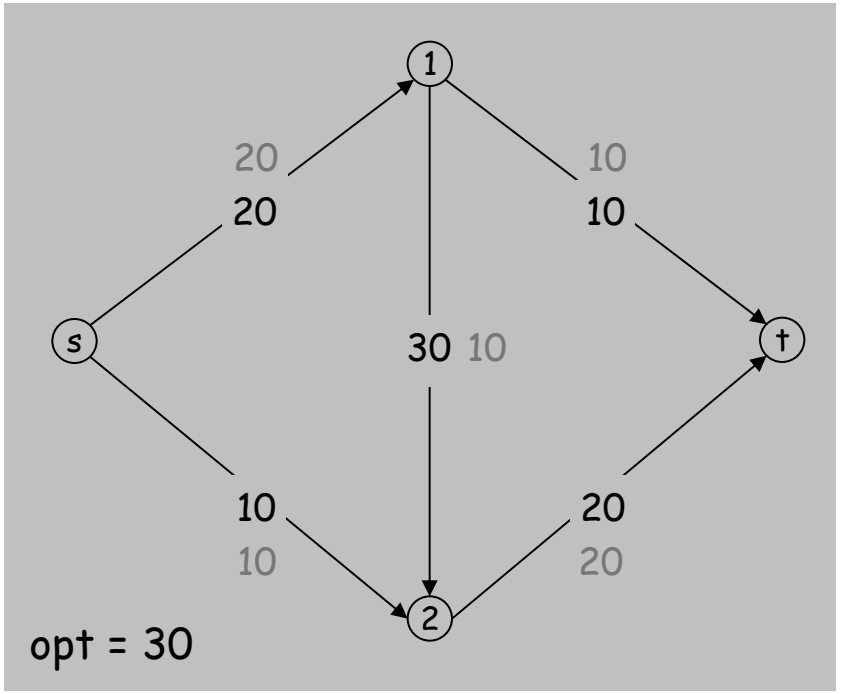


Residual graph: $G_f = (V, E_f)$.

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

Residual Graph

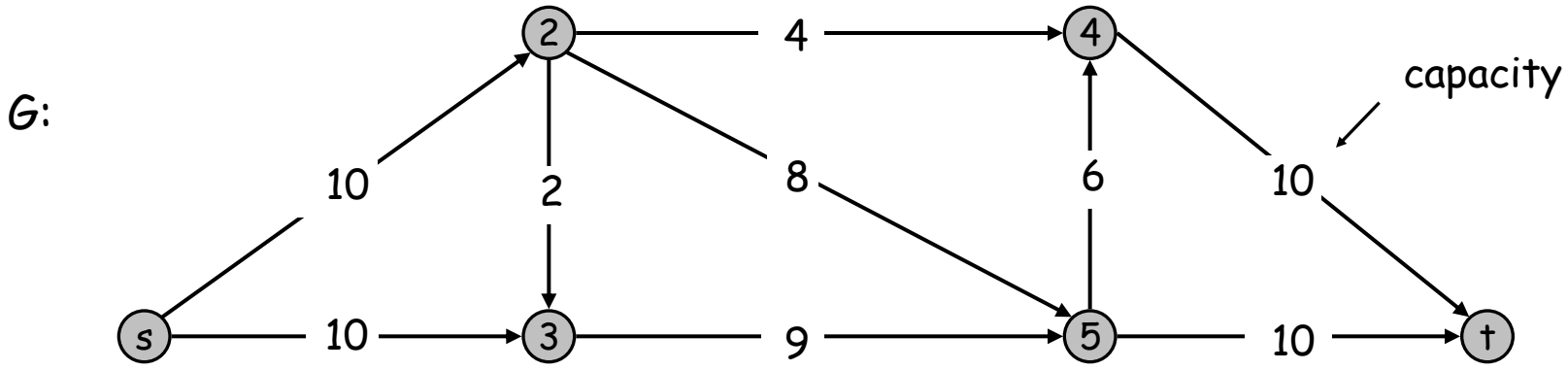




Ford-Fulkerson Algorithm

[Ford-Fulkerson '56] Keep sending flow in the residual graph until get stuck

- Now takes into account possibility of undoing previous moves



Ford-Fulkerson Algorithm

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

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

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

Recap

Max-flow problem: Given a directed graph with capacities on the edges. Want to send max amount of flow from s to t respecting capacities (= max flow out of s)

For any s-t cut:

flow out of s = effective flow out of cut = flow out cut – flow in cut (*)

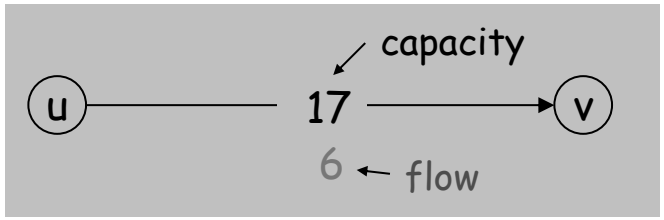
Flows vs. Cuts: Flow out of s is always at most the capacity of any s-t cut

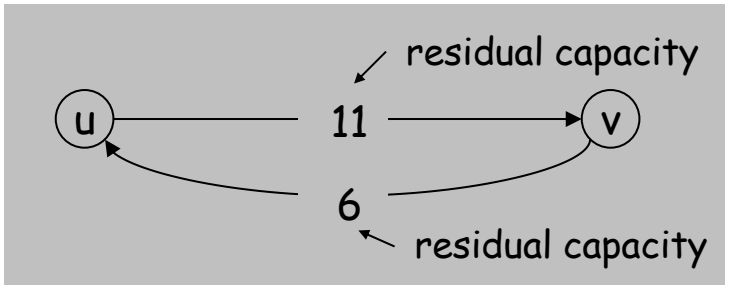
flow out of s <= capacity of cut

Duality (certificate of optimality): If have a flow with total flow out of s = V and found cut with same capacity V, then your flow is maximum

Recap

Residual graph:





Ford-Fulkerson algorithm for max-flow: Keep finding path from s to t in residual graph and push flow along this path

Ford-Fulkerson Algorithm: Analysis

Theorem: When the Ford-Fulkerson Algorithm terminates, it finds a max flow

Need to find cut with same capacity as flow out of s

Proof of Theorem:

- Let A be set of vertices reachable from s in residual graph at the end of algo
- Does A give an s-t cut?
- Crucial property: no edges going out of A in residual graph
- Look at capacity of this cut in the original graph
- What is the capacity?
- Is there flow incoming into A?
-

flow out of $s \overset{(*)}{=}$ effective flow out of A = flow out of A = capacity of cut

Ford-Fulkerson Algorithm: Analysis

Theorem: When the Ford-Fulkerson Algorithm terminates, it finds a max flow

Need to find cut with same capacity as flow out of s

Proof of Theorem:

- Let A be set of vertices reachable from s in residual graph at the end of algo
- Does A give an s-t cut? Yes: $s \in A$ and $t \notin A$ (no paths from s in res. graph)
- Crucial property: no edges going out of A in residual graph

- Look at capacity of this cut in the original graph
- What is the capacity? All outgoing edges are reversed in residual graph, so we're pushing flow in them up to capacity ➔ capacity = flow out of A

- Is there flow incoming into A? No, otherwise edges out of A in residual graph
- So

flow out of $s \overset{(*)}{=}$ effective flow out of A = flow out of A = capacity of cut

Ford-Fulkerson Algorithm: Running Time

Assumption. All capacities are integers between 1 and C.

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

Theorem. The algorithm terminates in at most $maxFlow \leq nC$ iterations.
Pf. Each augmentation increase value by at least 1. ▪

Running time: $O(maxFlow^*(m+n))$, which is $O(nC(m+n))$

Q: Is this polynomial in the input size?
A: No! Just like in knapsack, input size is ~ m log C

Consequences

Max-flow/Min-cut Theorem ['56]: There is a cut whose capacity equals the value of the max flow (it's the cut with minimum capacity)

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.

This theorem is very important for modeling applications

7.3 Choosing Good Augmenting Paths

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]

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

History of running times

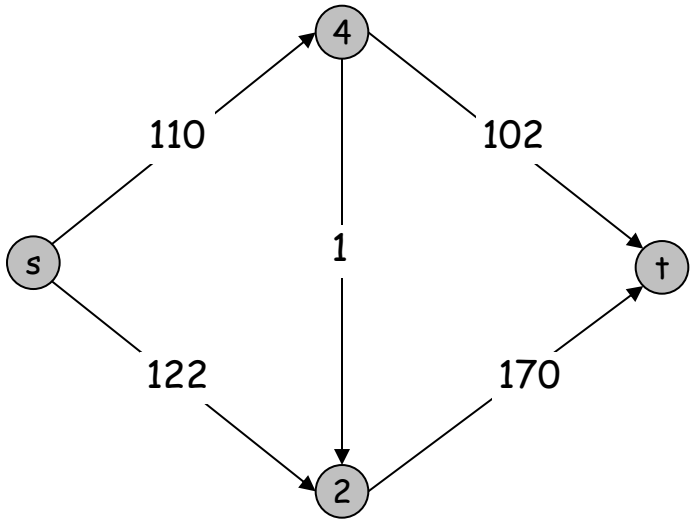
Year	Discoverer	Method	Asymptotic Time
1951	Dantzig	Simplex	$E V^2 U^{\dagger}$
1955	Ford, Fulkerson	Augmenting path	$E V U^{\dagger}$
1970	Edmonds-Karp	Shortest path	$E^2 V$
1970	Edmonds-Karp	Max capacity	$E \log U (E + V \log V)^{\dagger}$
1970	Dinitz	Improved shortest path	$E V^2$
1972	Edmonds-Karp, Dinitz	Capacity scaling	$E^2 \log U^{\dagger}$
1973	Dinitz-Gabow	Improved capacity scaling	$E V \log U^{\dagger}$
1974	Karzanov	Preflow-push	V^3
1983	Sleator-Tarjan	Dynamic trees	$E V \log V$
1986	Goldberg-Tarjan	FIFO preflow-push	$E V \log (V^2 / E)$
. . .	. . .	. . .	. . .
1997	Goldberg-Rao	Length function	$E^{3/2} \log (V^2 / E) \log U^{\dagger}$ $E V^{2/3} \log (V^2 / E) \log U^{\dagger}$

2013 [Orlin]: $O(nm)$

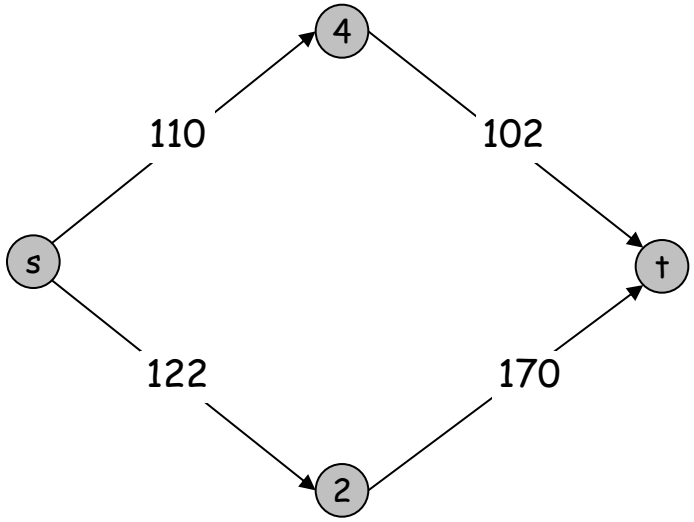
Capacity Scaling

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
}
```