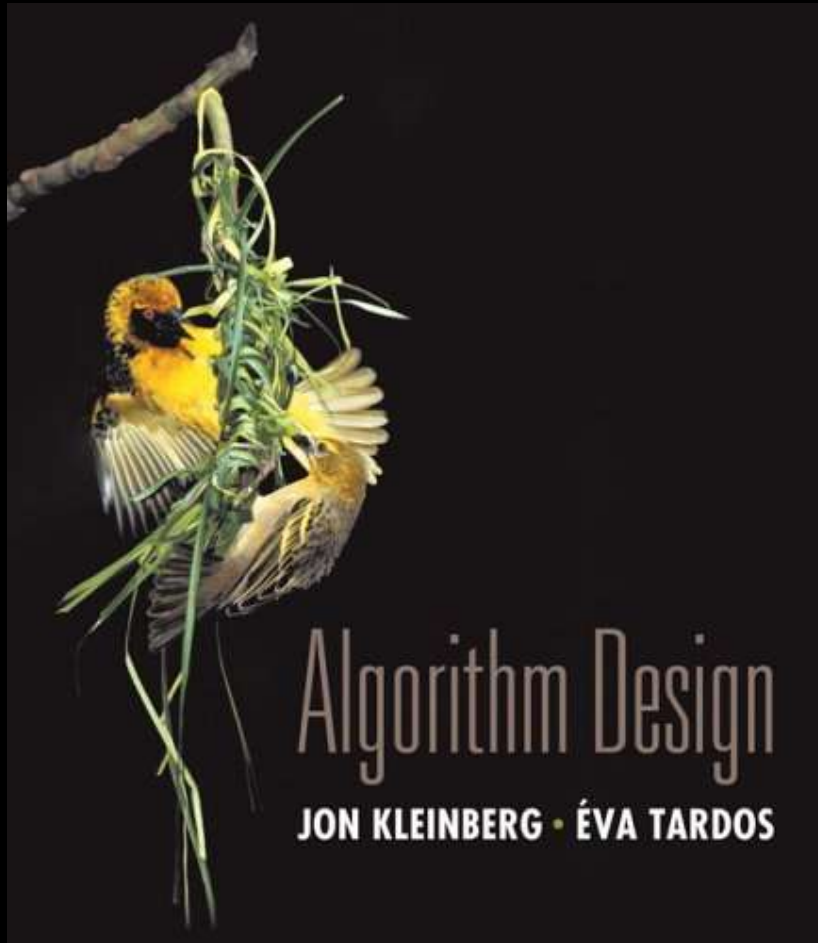




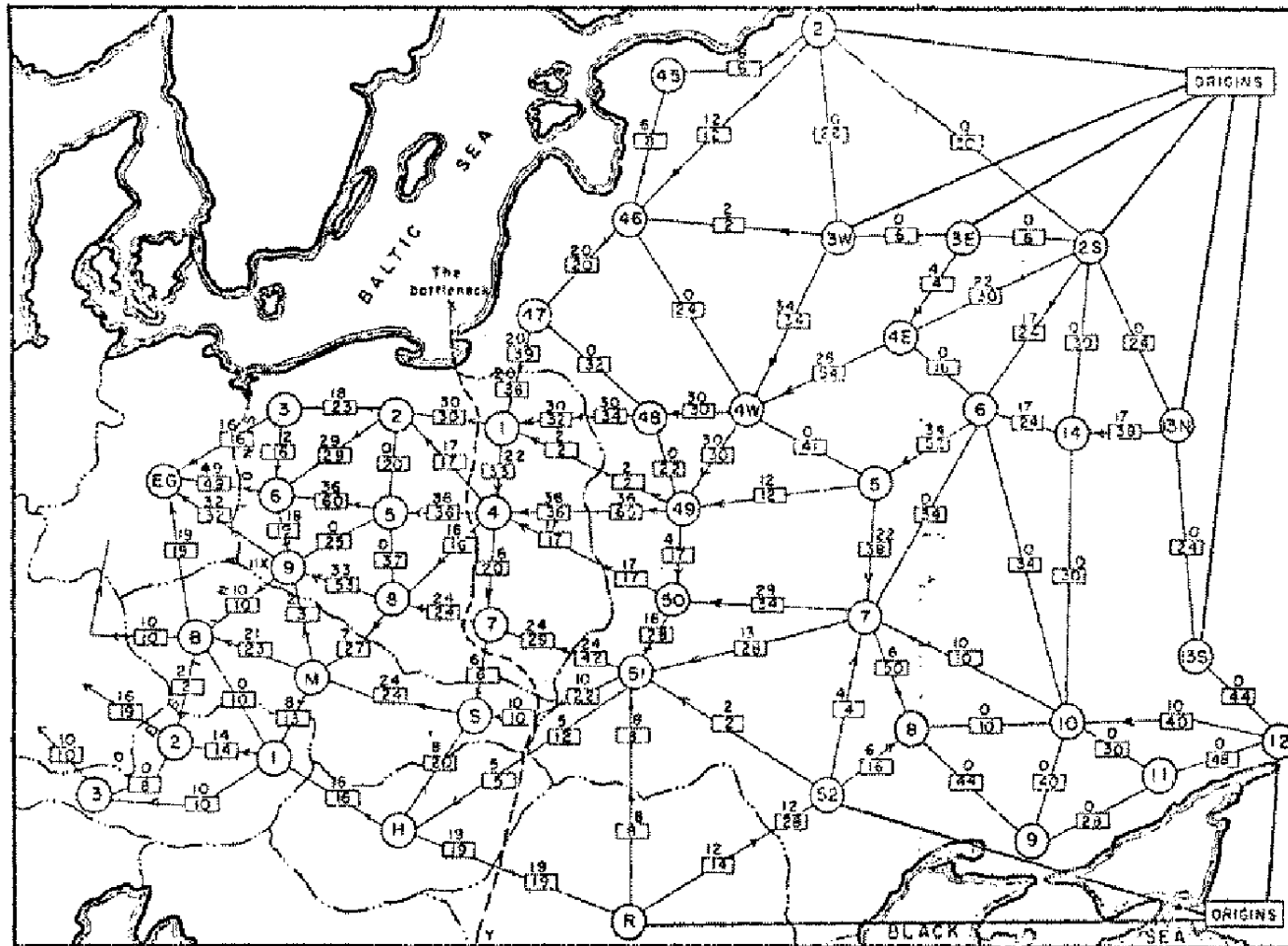
Chapter 7

Network Flow



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.
Updated by T. K. Prasad

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.

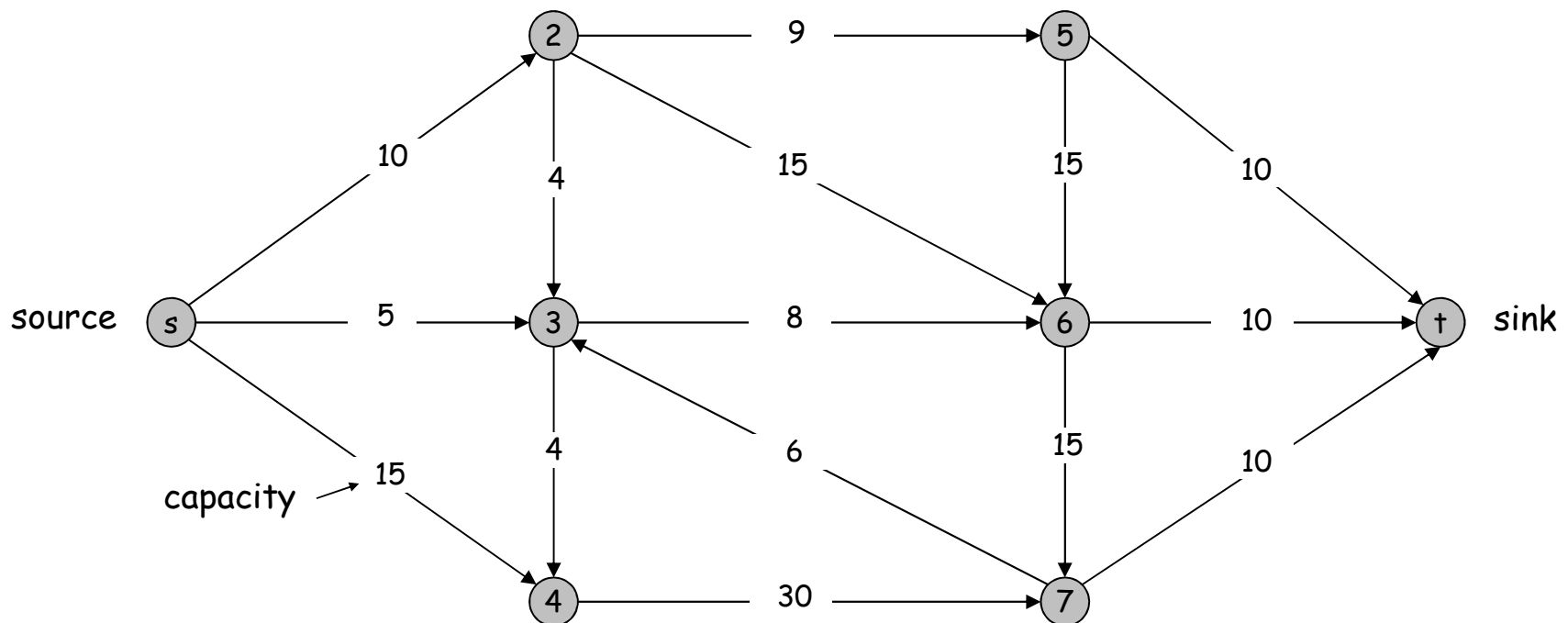
- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.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

Minimum Cut Problem

Flow network.

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

no parallel edges
no edge enters s
no edge leaves t

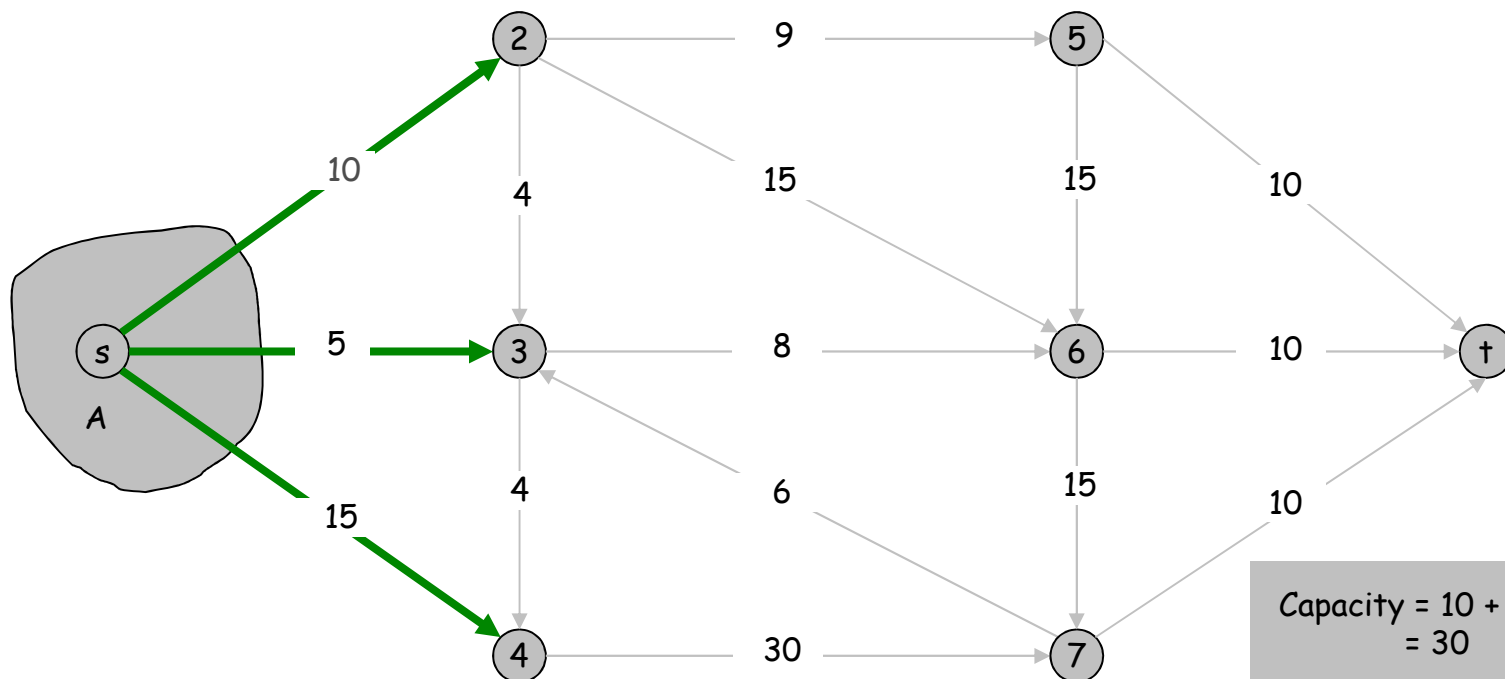


Cuts

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 the sum of the capacities of the edges from A to B :

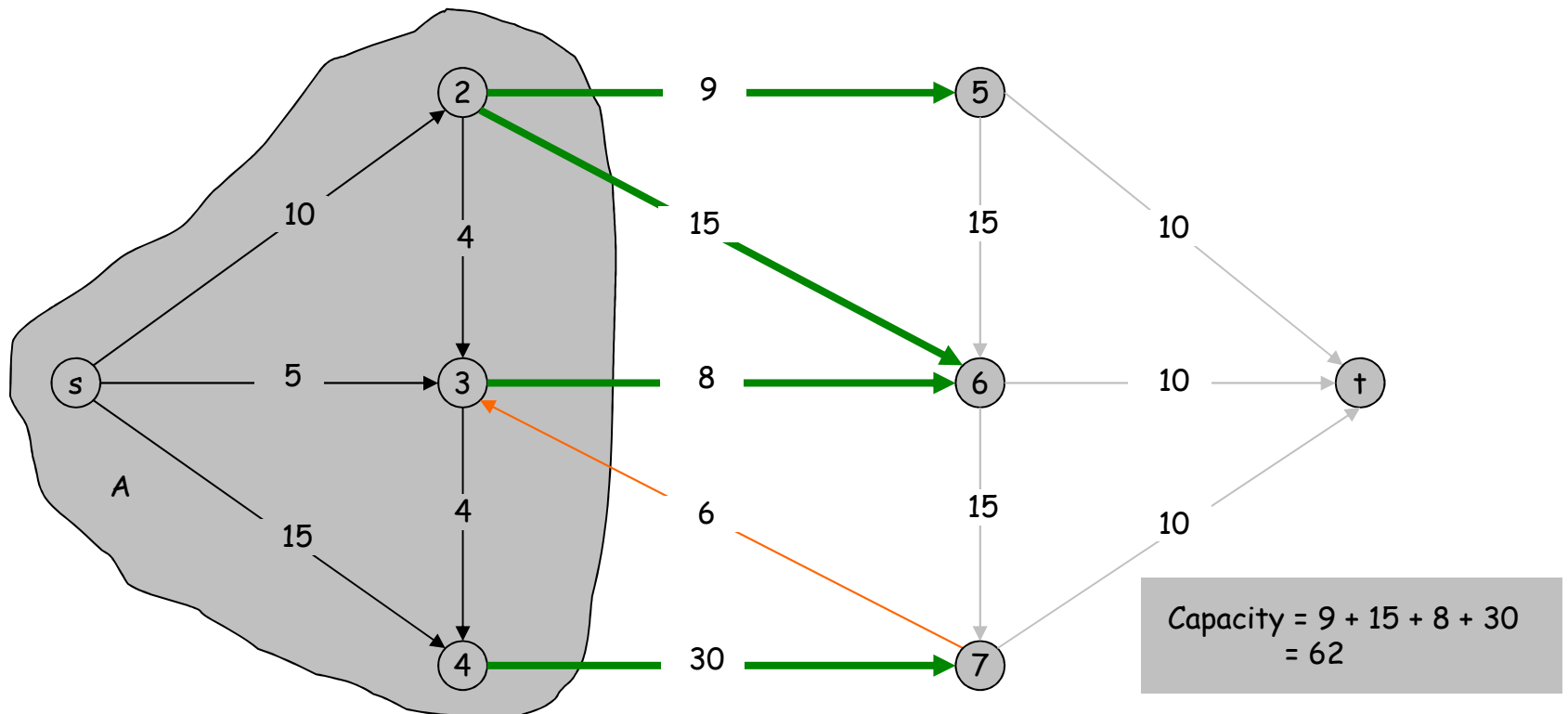
$$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Cuts

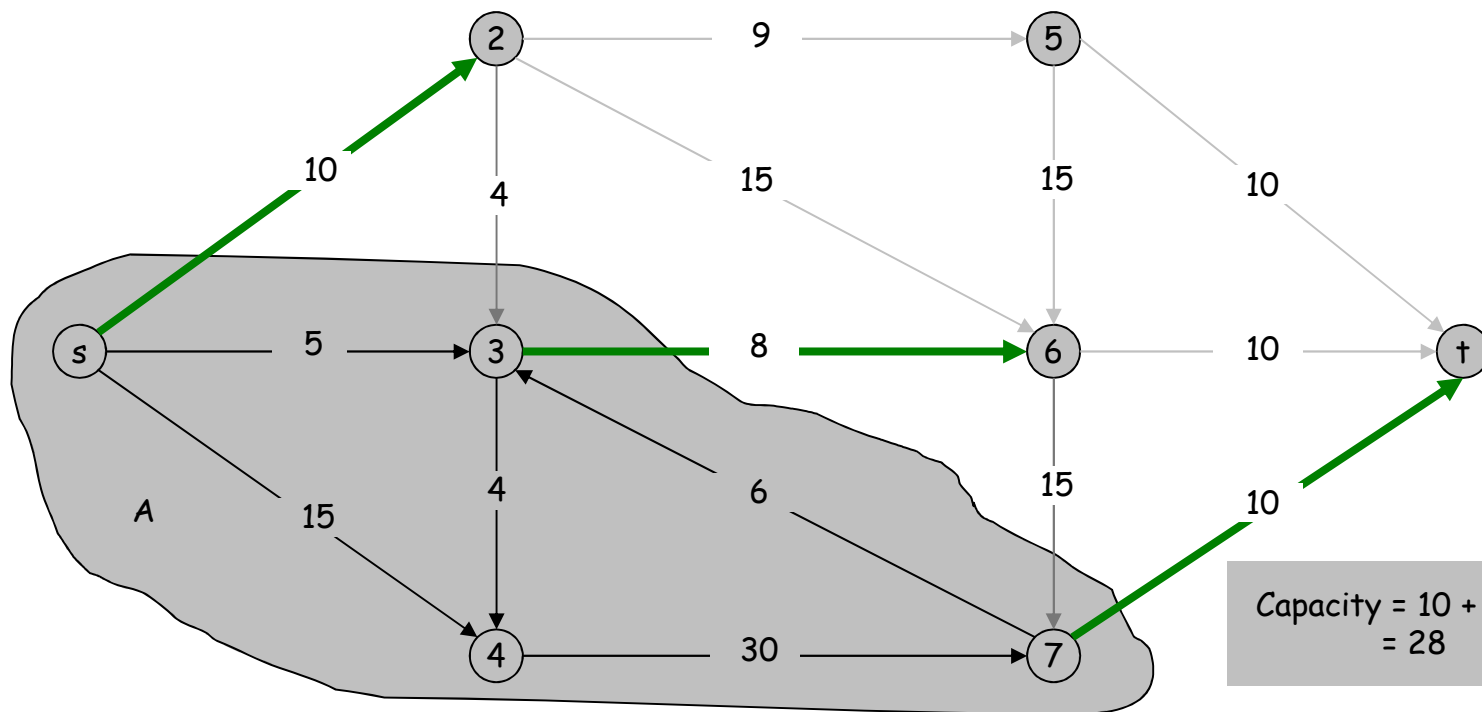
Def. An **s-t cut** is a partition (A, B) of V with $s \in A$ and $t \in B$.

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



Minimum Cut Problem

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

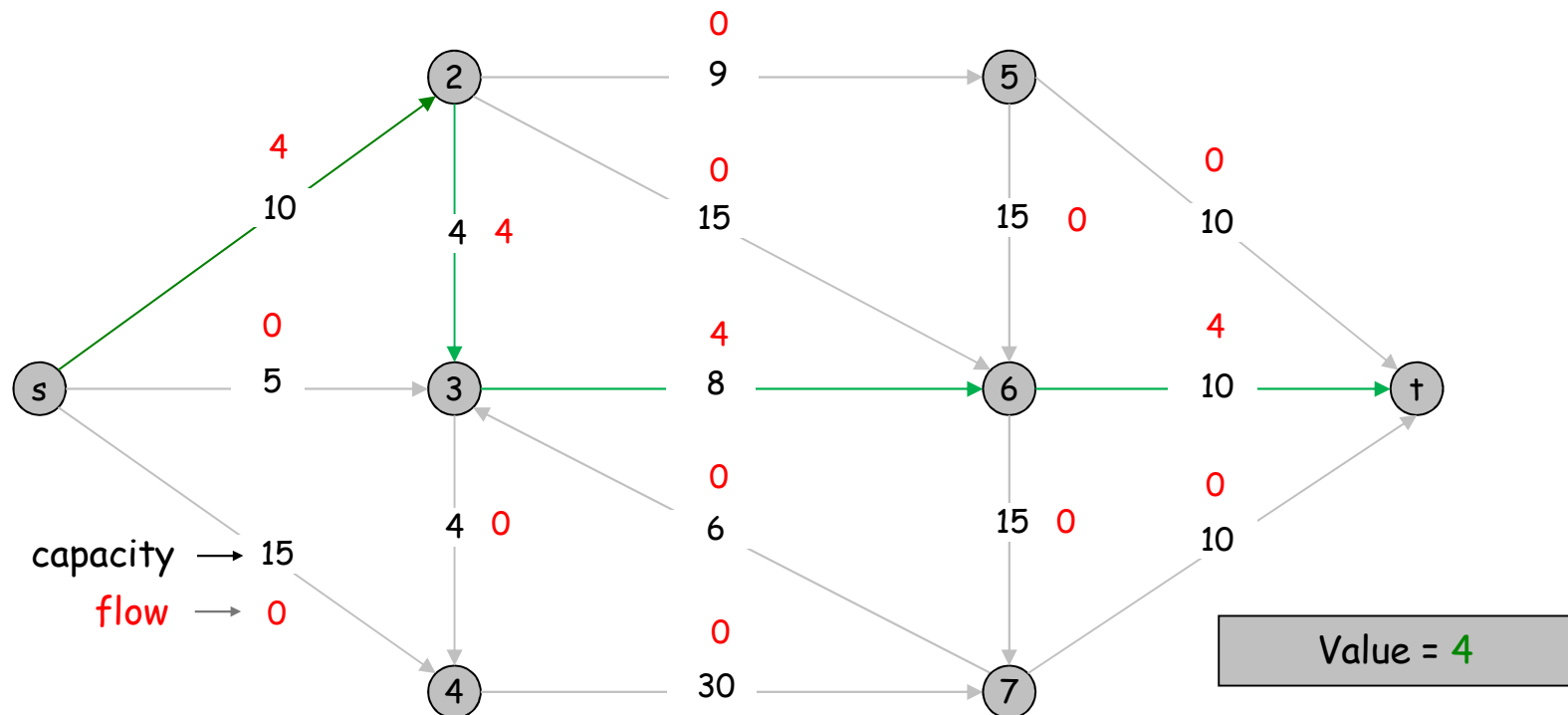


Flows

Def. An **s-t flow** f 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: $v(f) = \sum_{e \text{ out of } s} f(e)$.

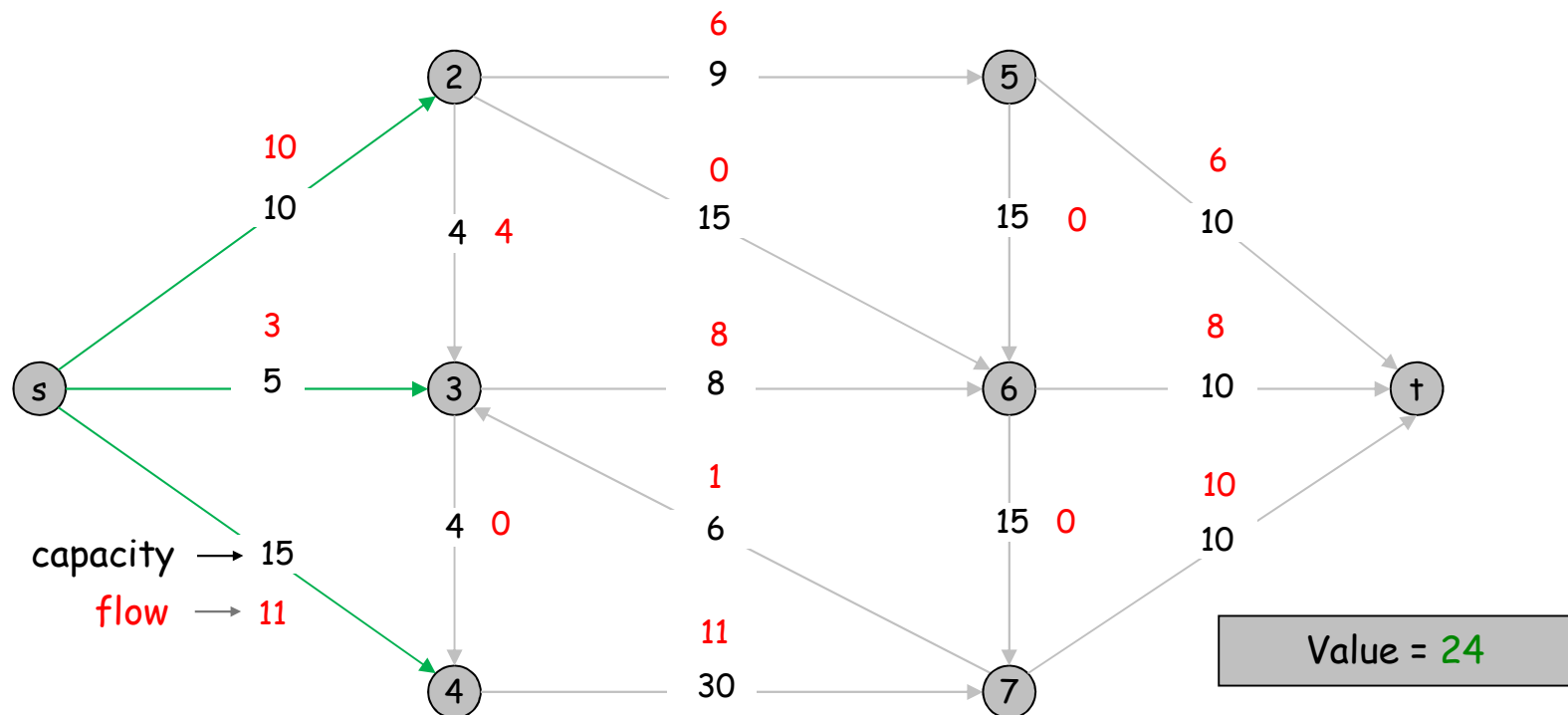


Flows

Def. An **s-t flow** f 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)

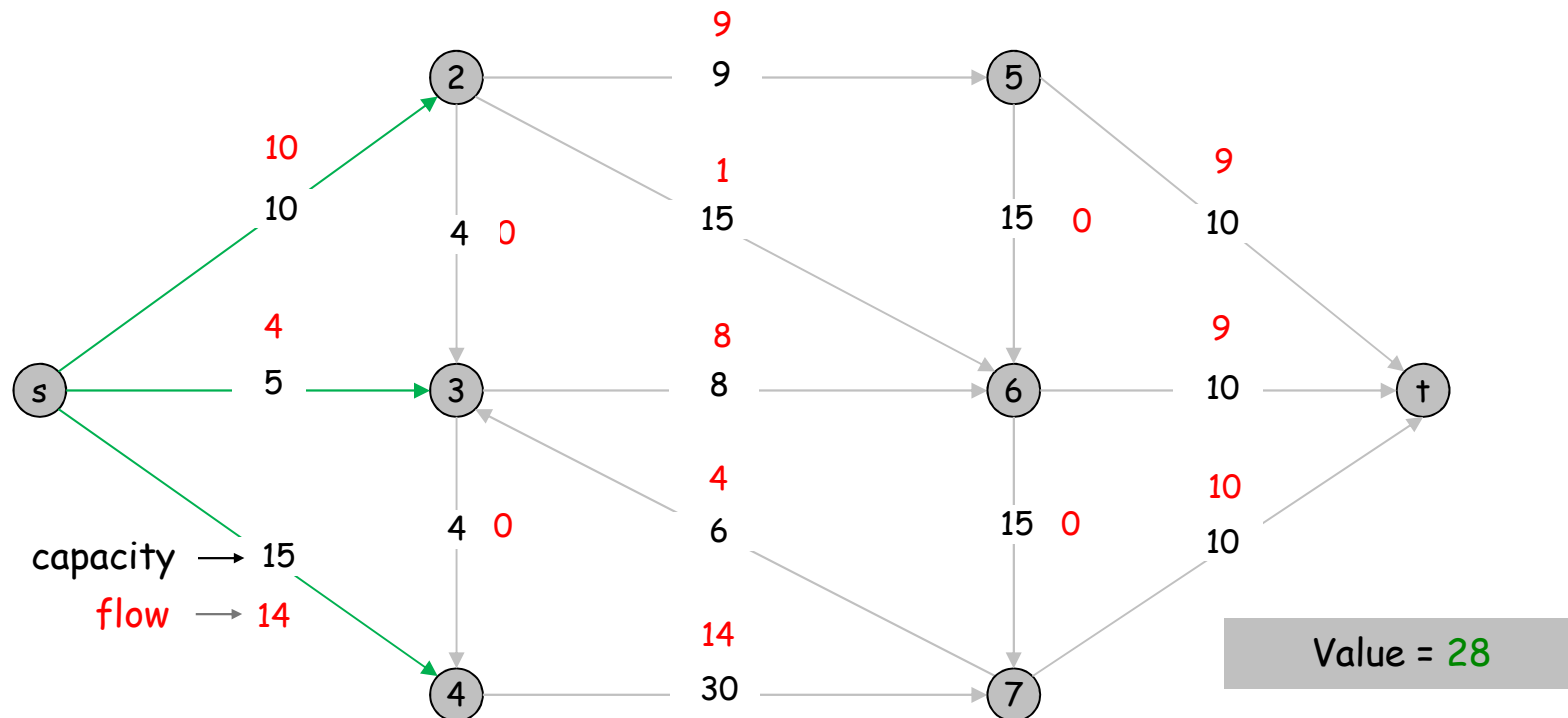
Def. The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Maximum Flow Problem

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

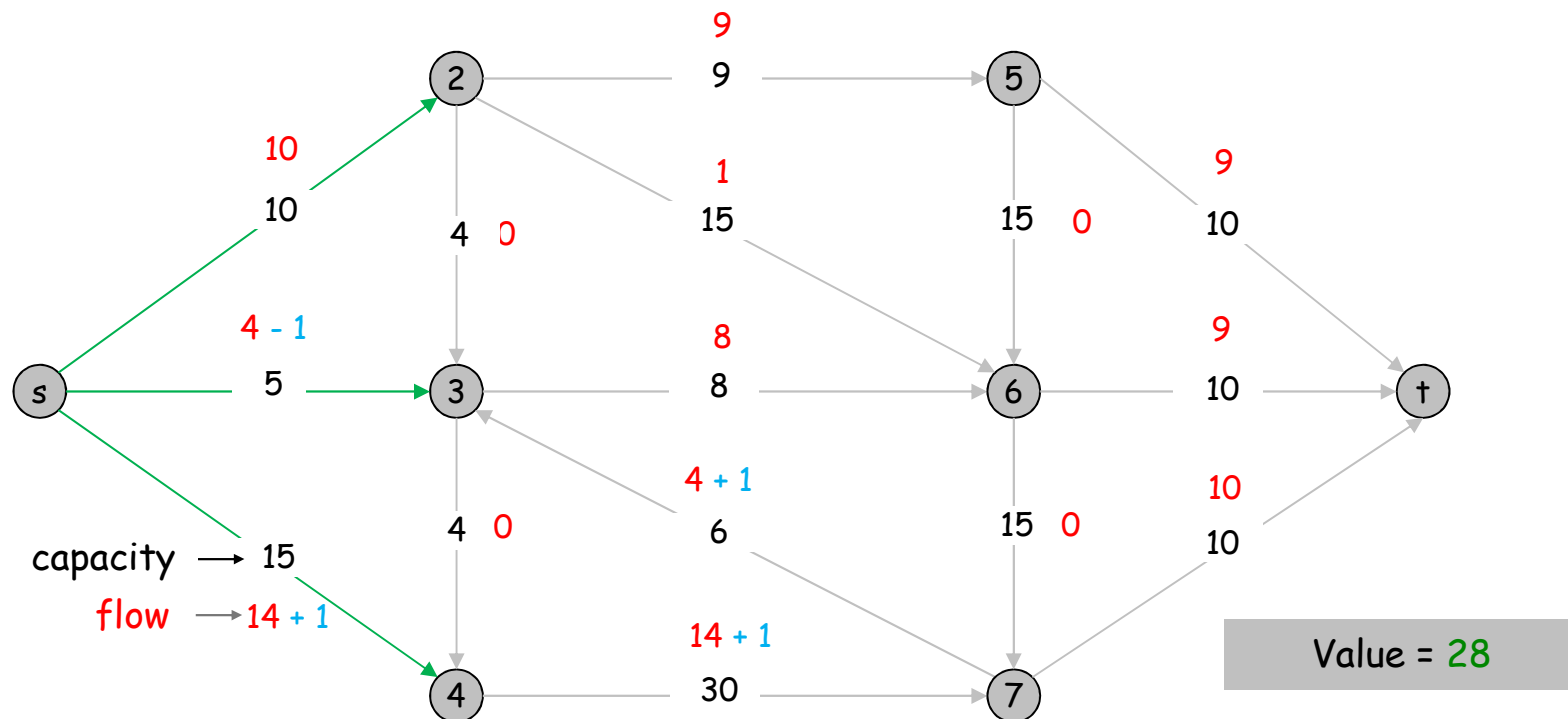
Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Maximum Flow Problem

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

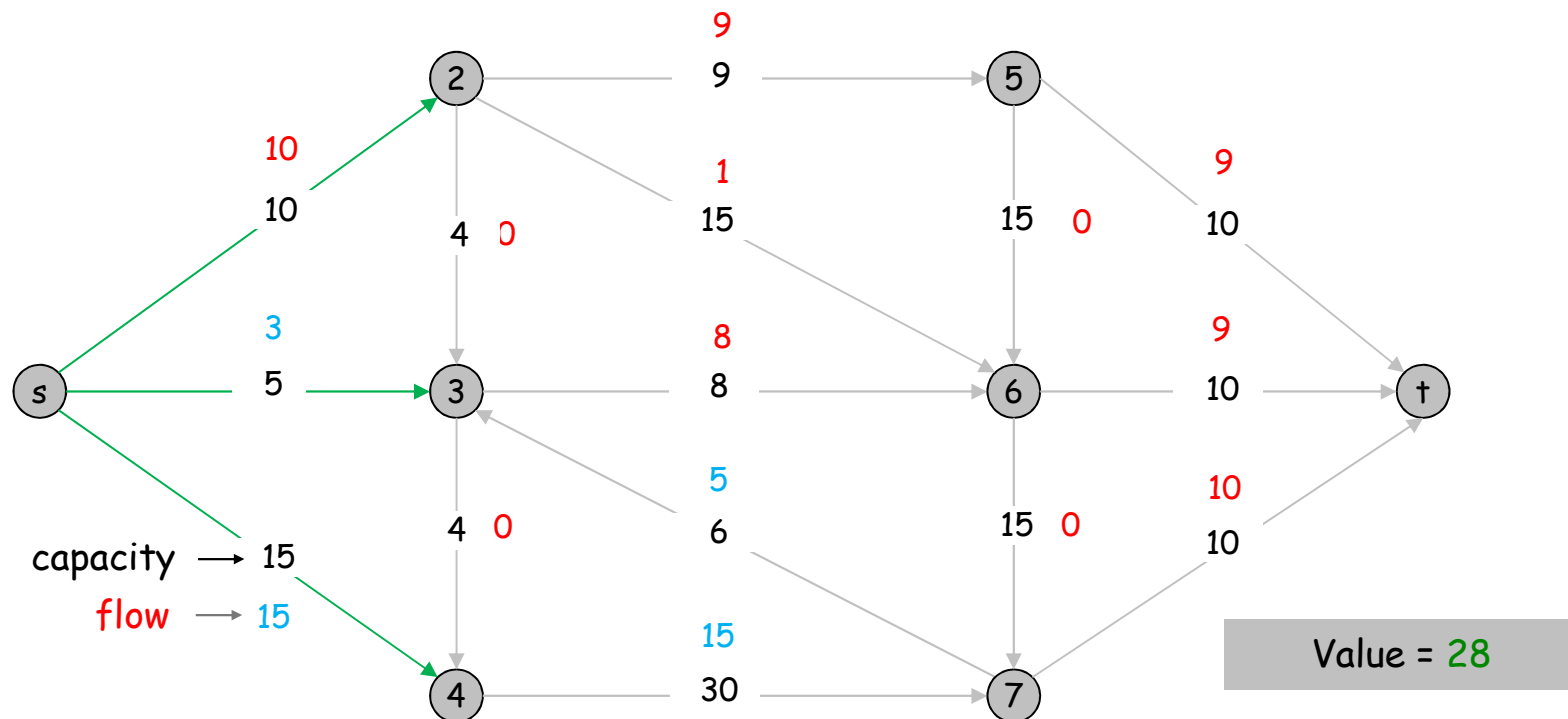
Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Maximum Flow Problem

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

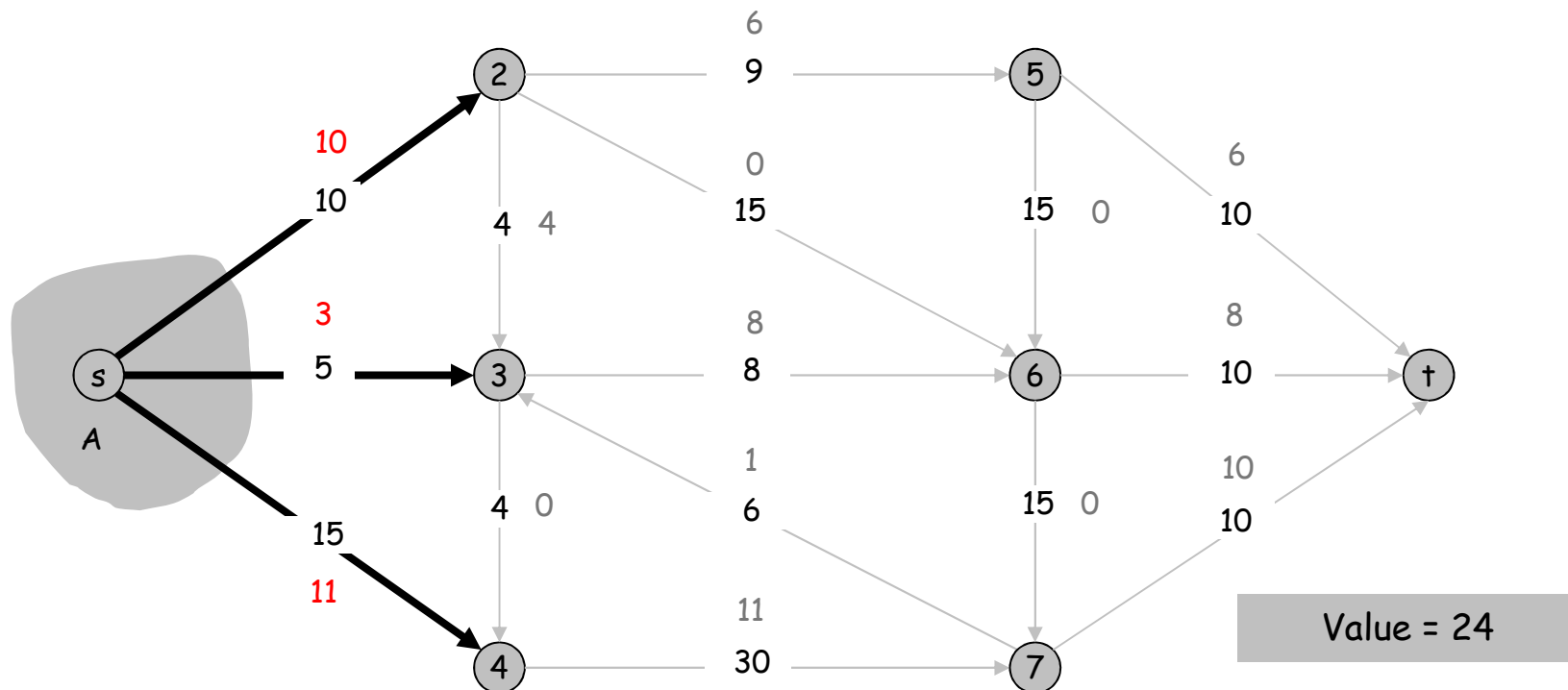
Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



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 .

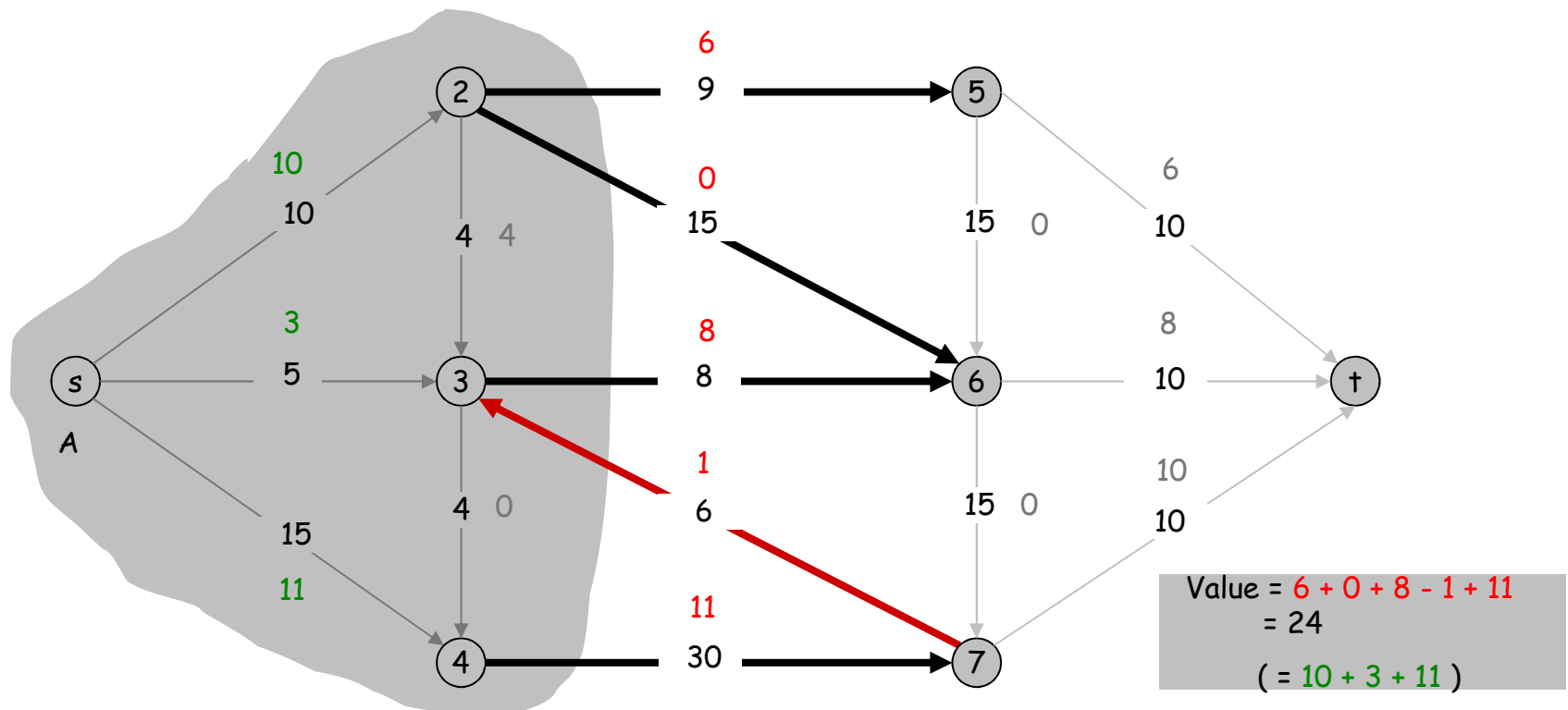
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



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 .

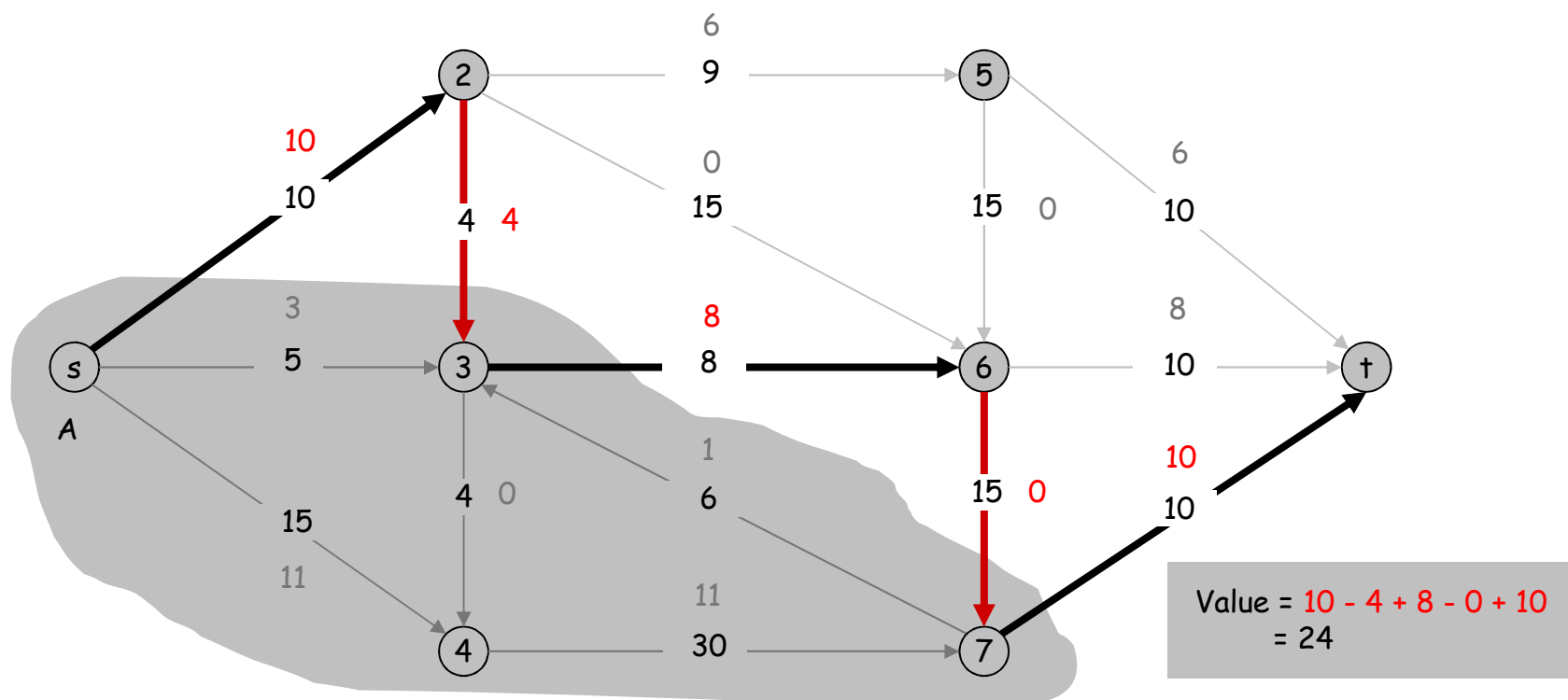
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



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

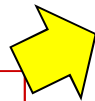
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$$

Pf.

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

by flow conservation and definition of source, all terms except for outflow from s are 0.

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



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

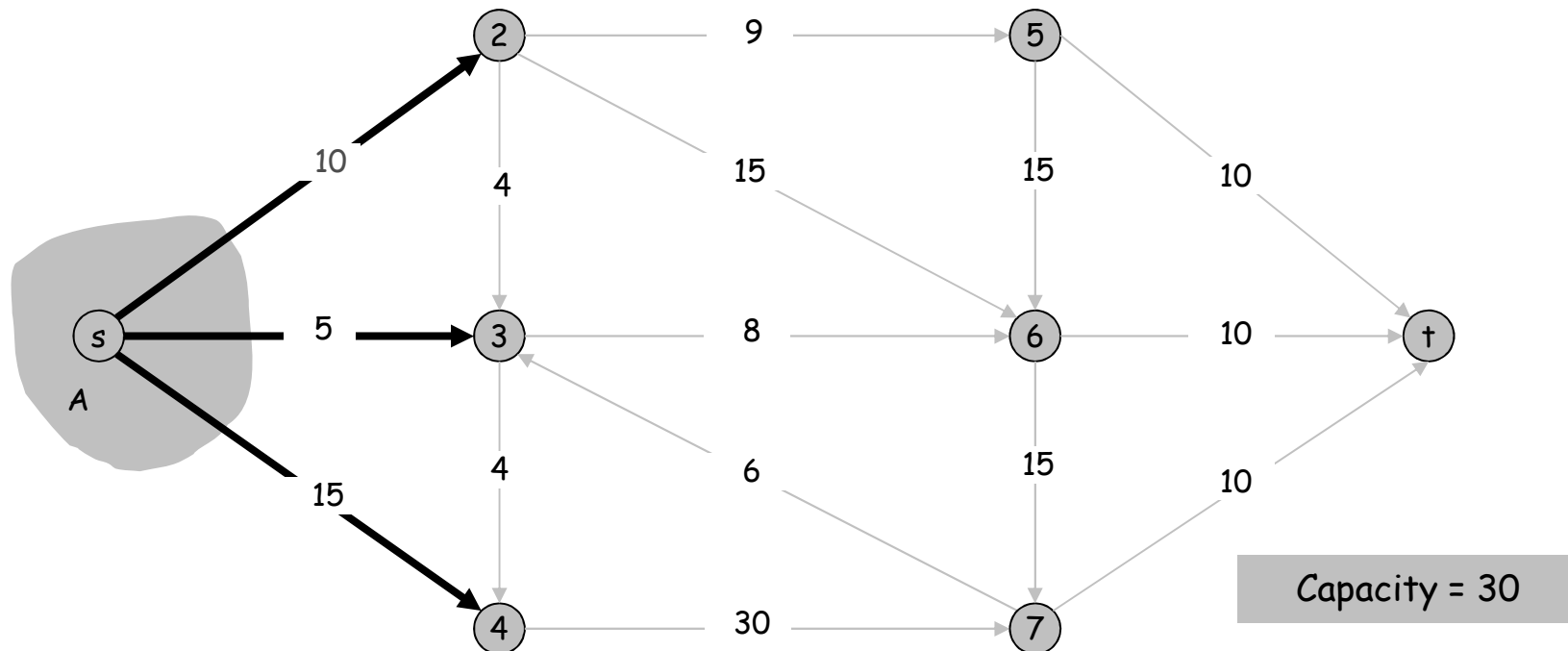
Edge e from/to v in A can be incident on (to/from) some u in A or B .
If in A , the two identical edges from the two summations cancel. Only if in B , the hanging edges in the two summations survive.

Edges connecting nodes within A do not contribute to any net outflow or inflow with respect to A .

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

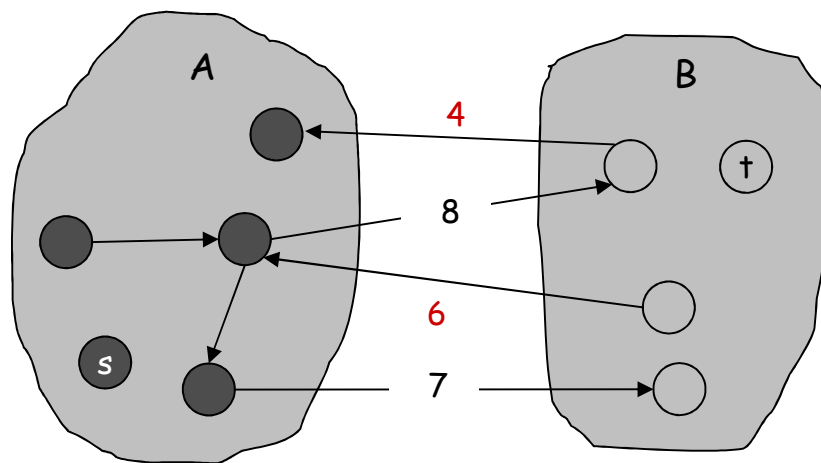


Flows and Cuts

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

Pf.

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

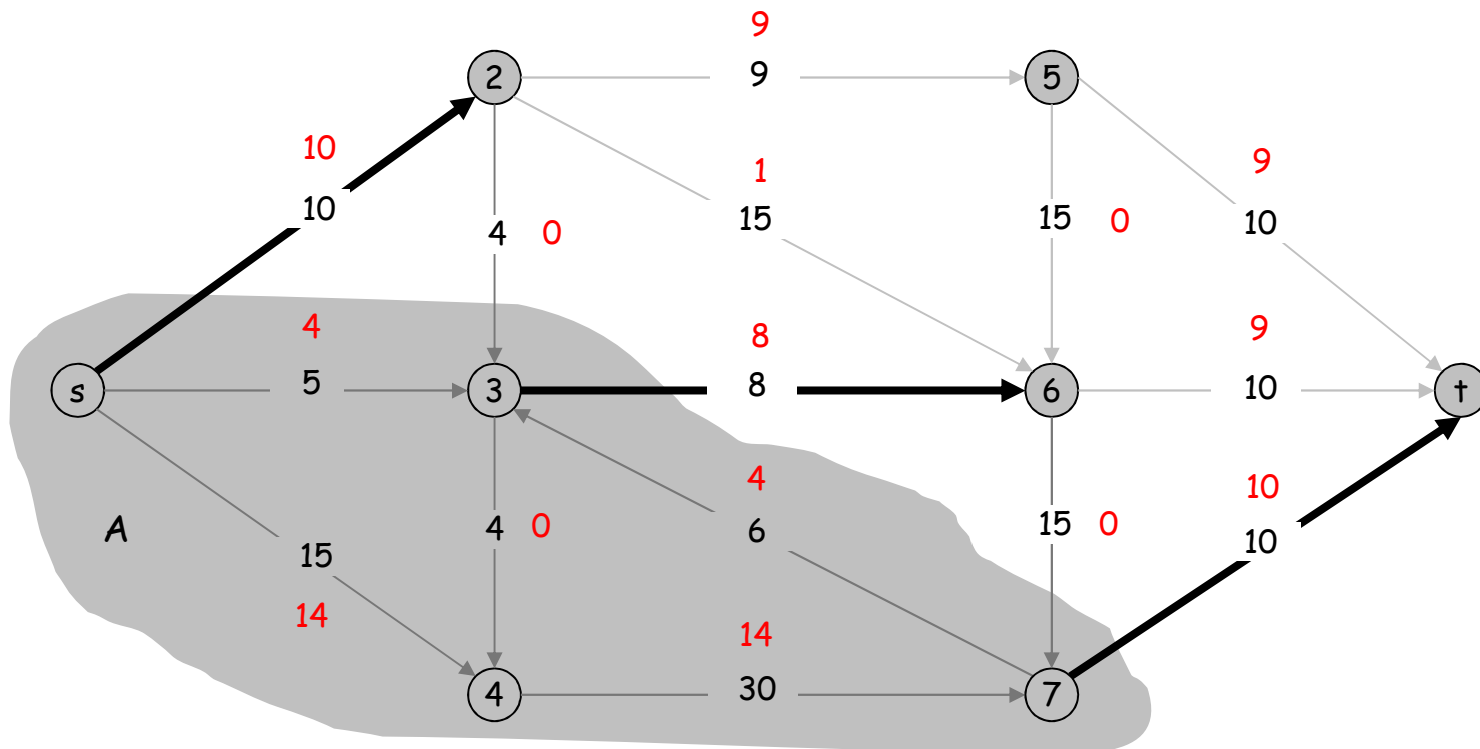


Certificate of Optimality

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

Value of flow = 28

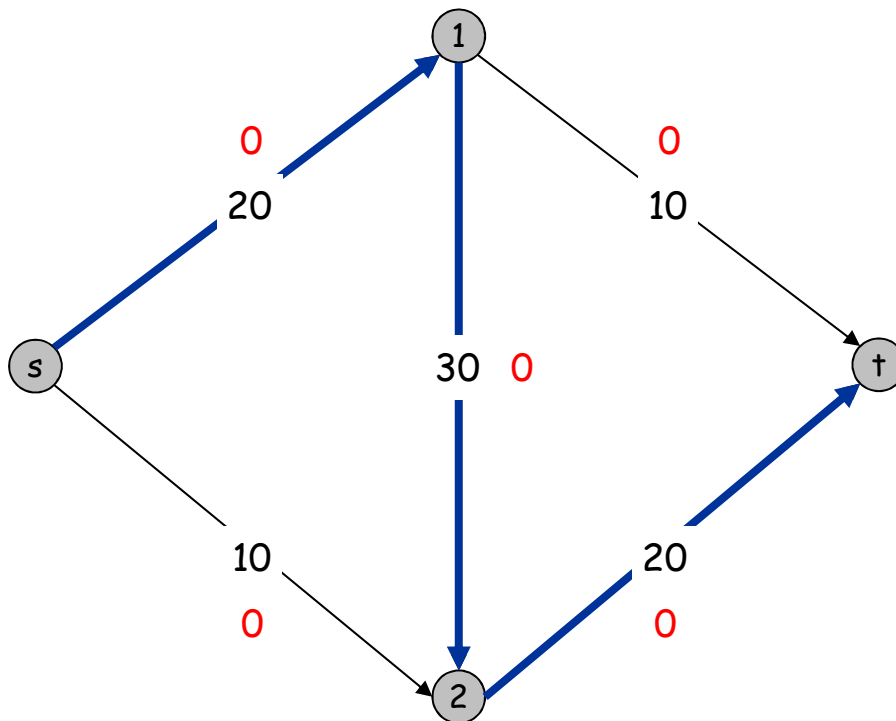
Cut capacity = 28 $\Rightarrow$ Flow value ≤ 28



Towards a Max Flow Algorithm : “Realization”

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.

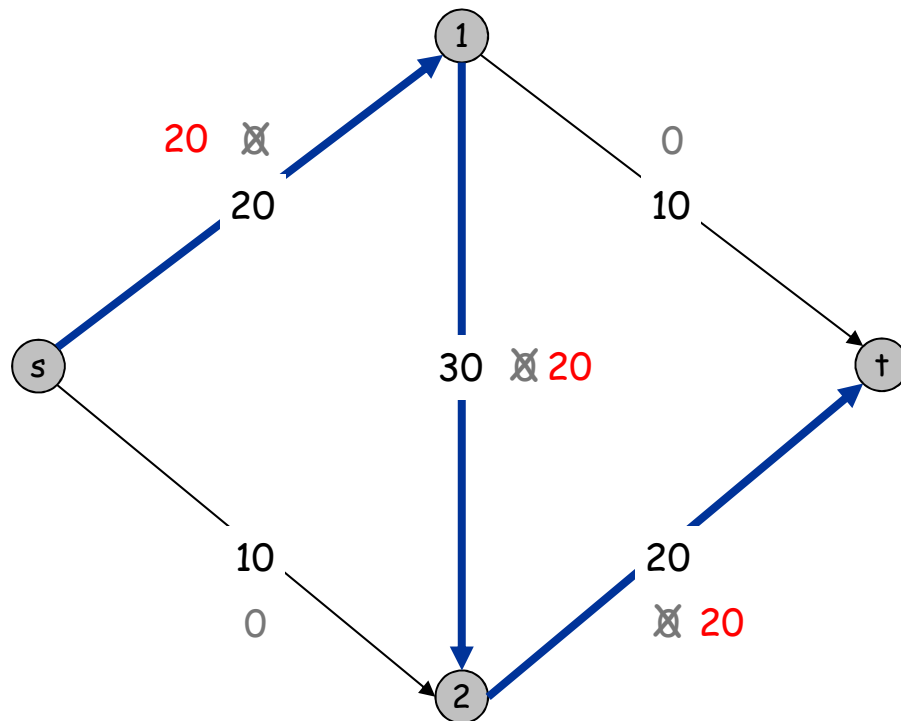


Flow value = 0

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.



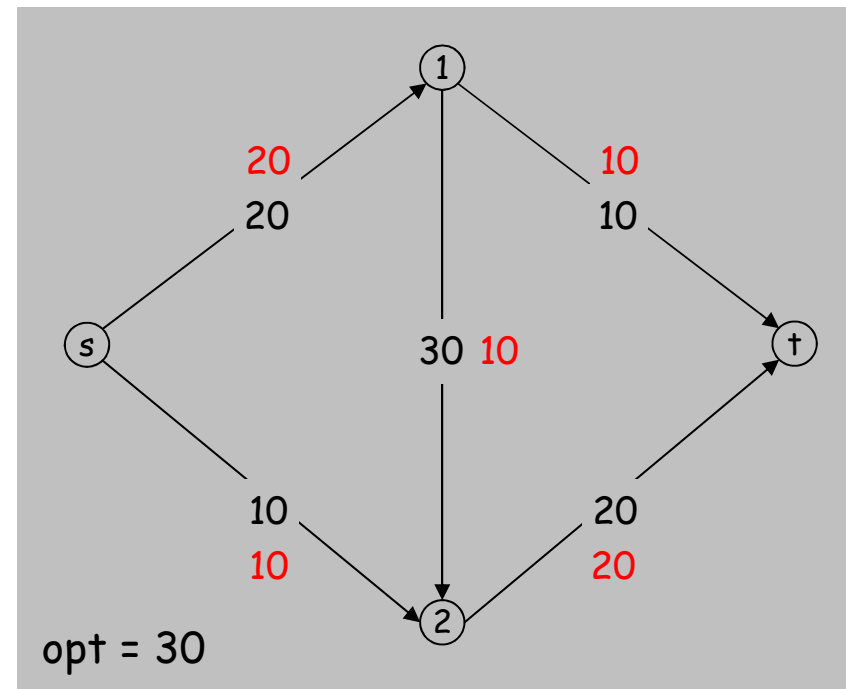
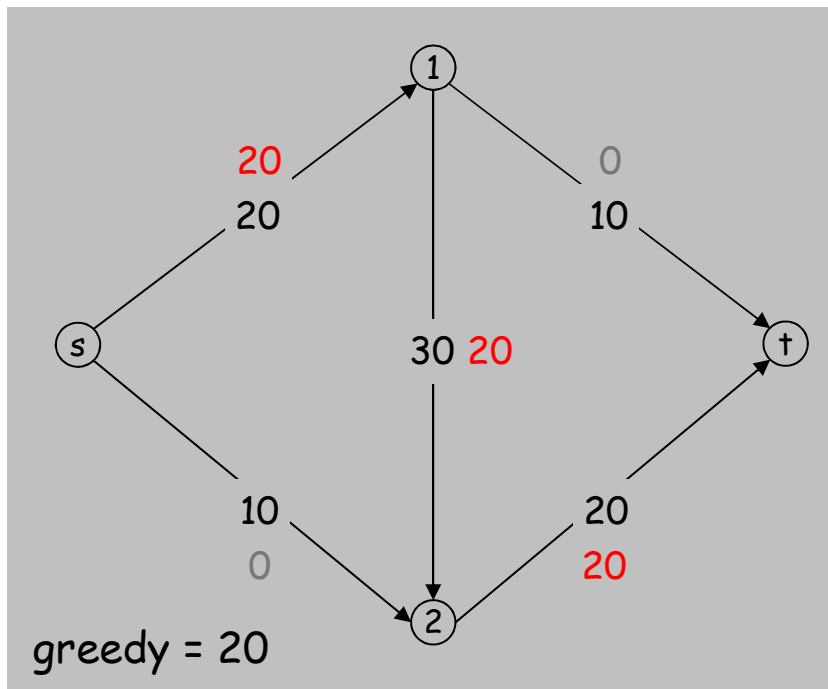
Flow value = 20

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**.

↖ local optimality $\Rightarrow$ global optimality

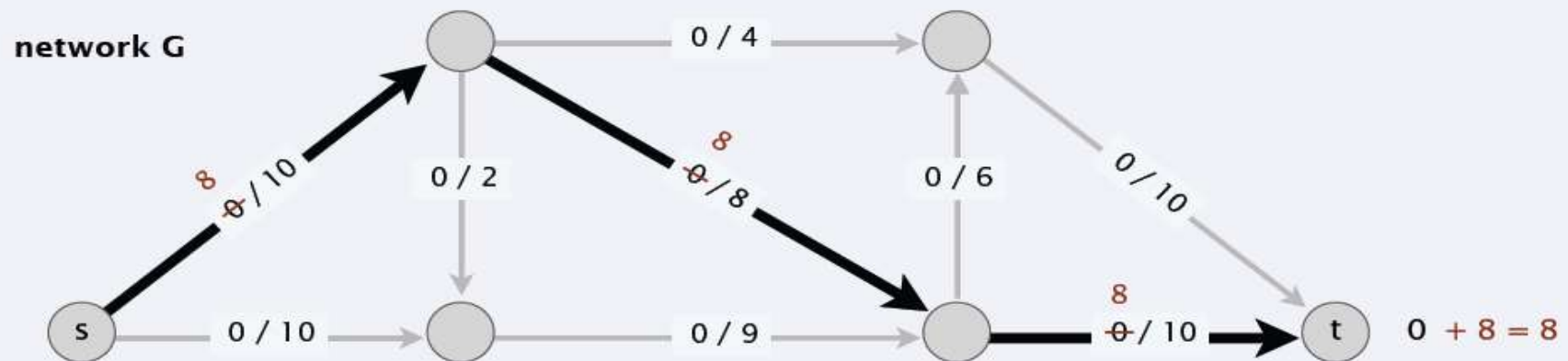


Another Example

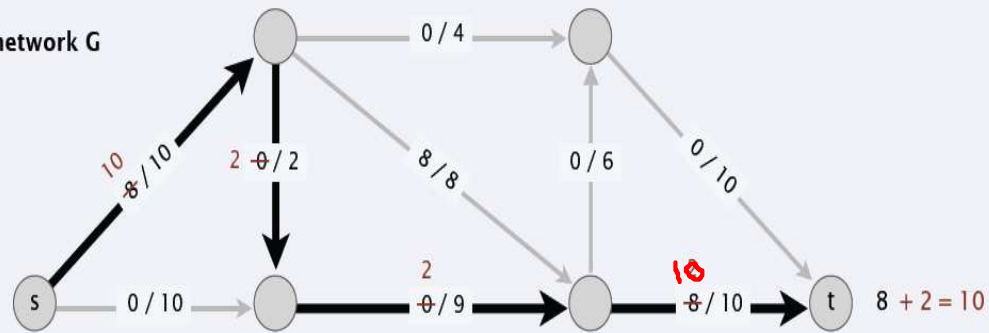
Towards a max-flow algorithm

Greedy algorithm.

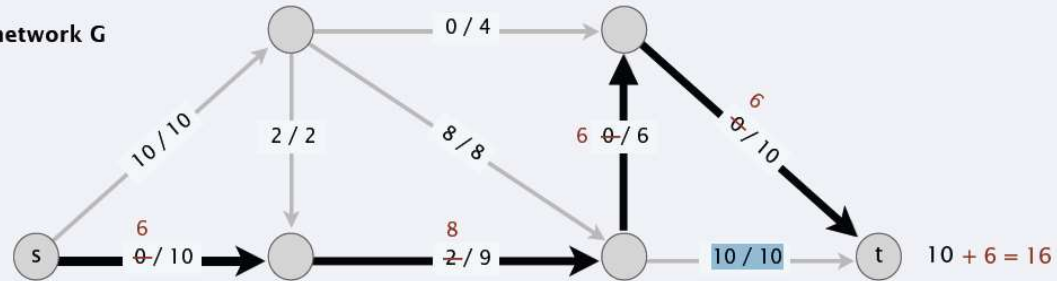
- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an $s \rightarrow t$ path P where each edge has $f(e) < c(e)$.
- Augment flow along path P .
- Repeat until you get stuck.



network G

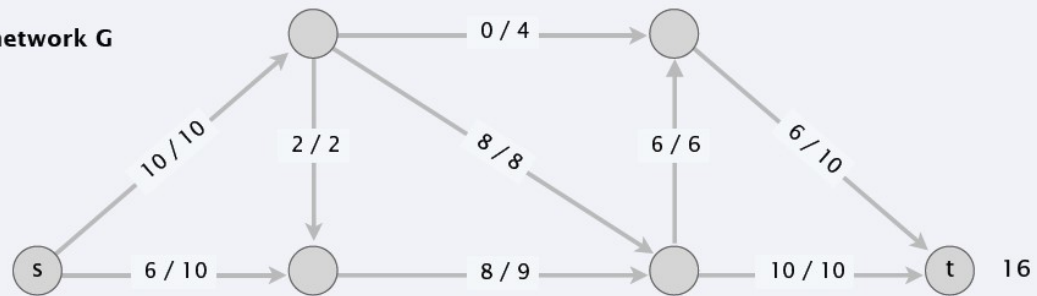


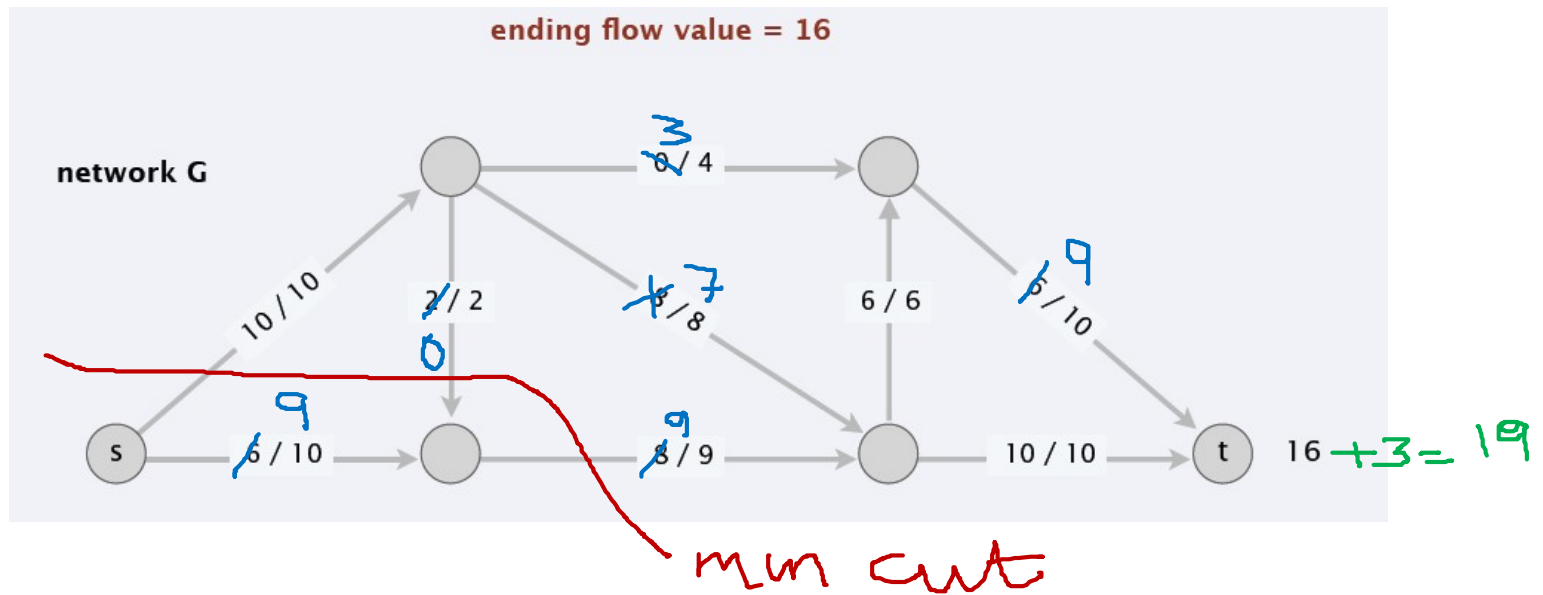
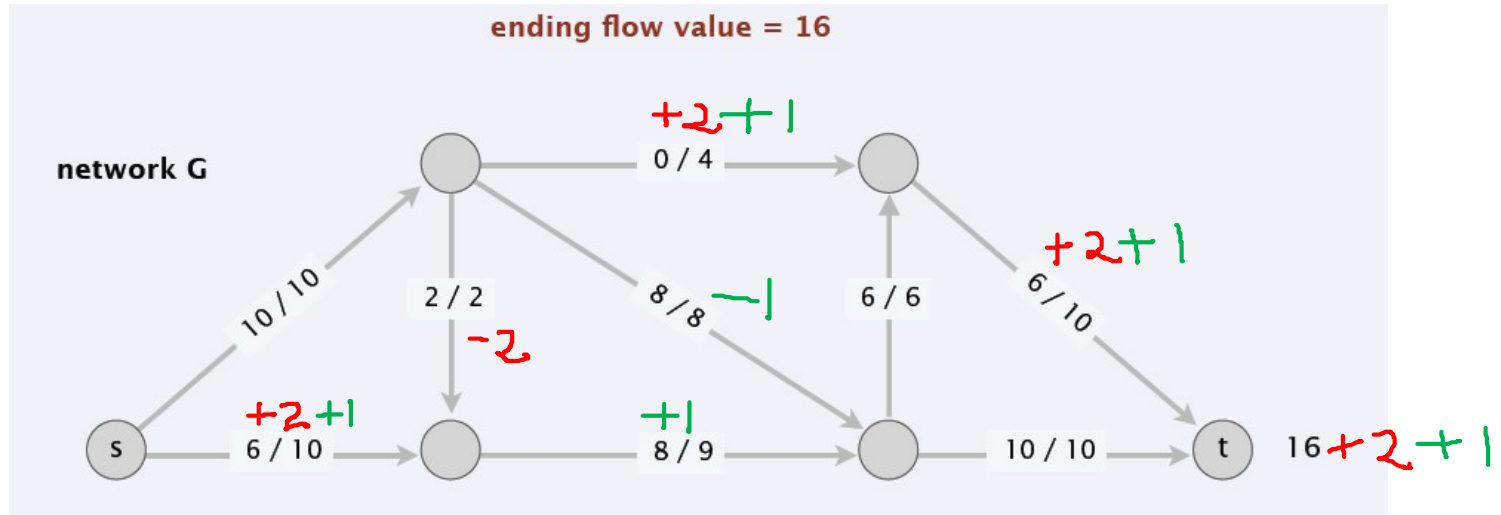
network G



ending flow value = 16

network G

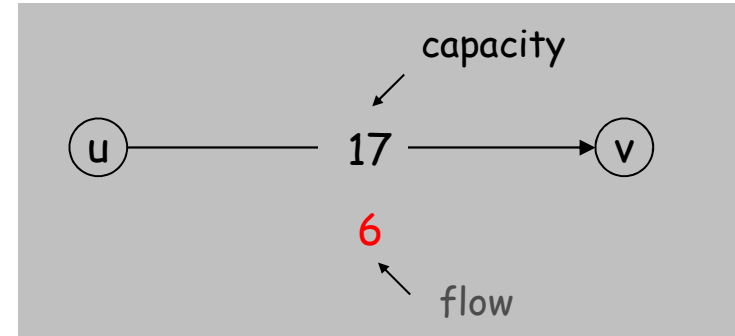




Residual Graph

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

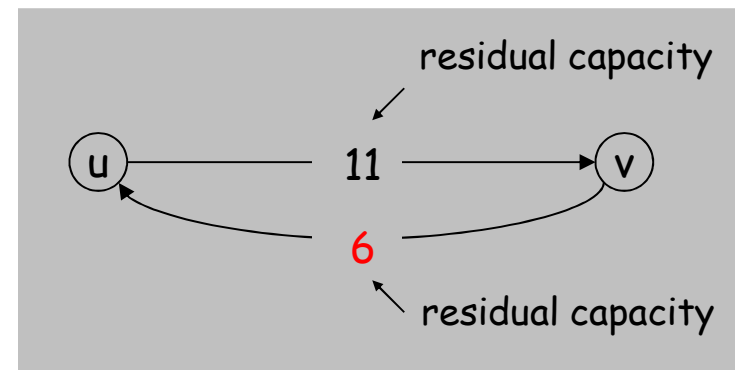
- Flow $f(e)$, capacity $c(e)$.



Residual edge.

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

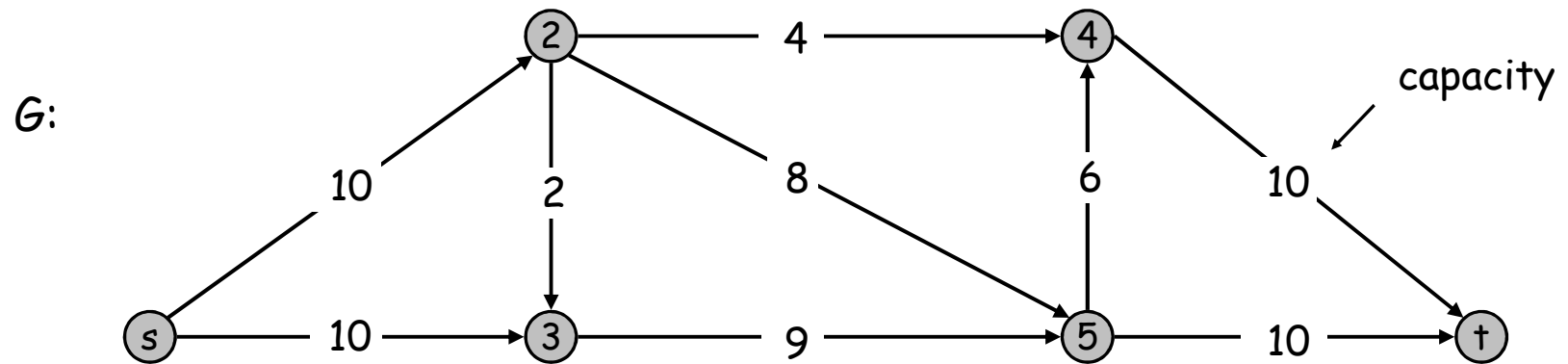
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



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

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

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 capacity of the min cut.

Proof strategy. We prove both simultaneously by showing the TFAE:

- (i) There exists a cut (A, B) such that $v(f) = \text{cap}(A, B)$.
- (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.

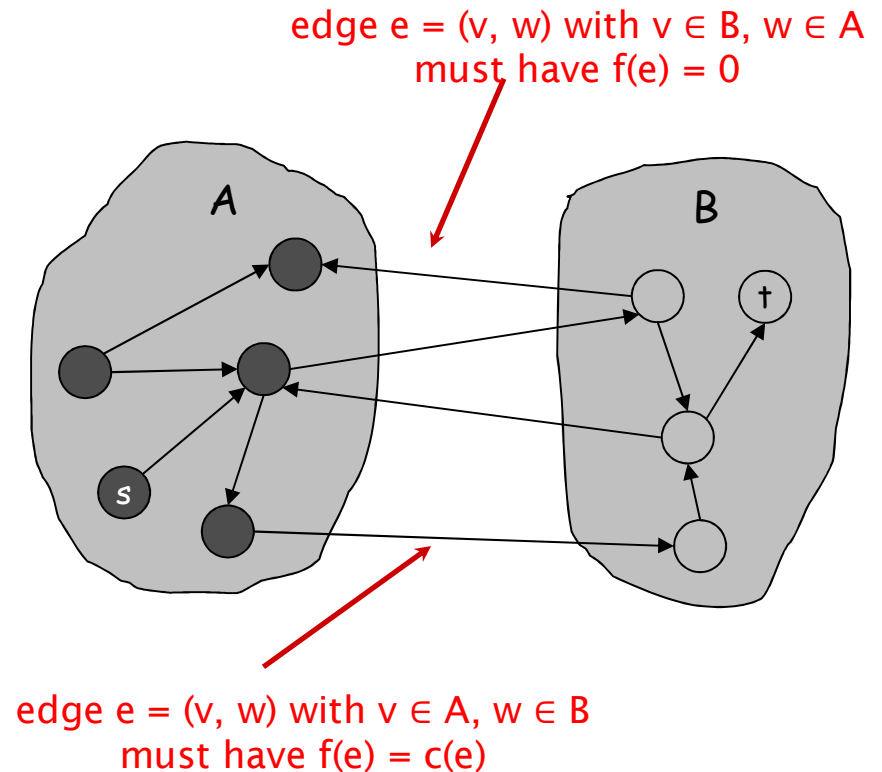
- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along that path.

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 , $t \notin A$.

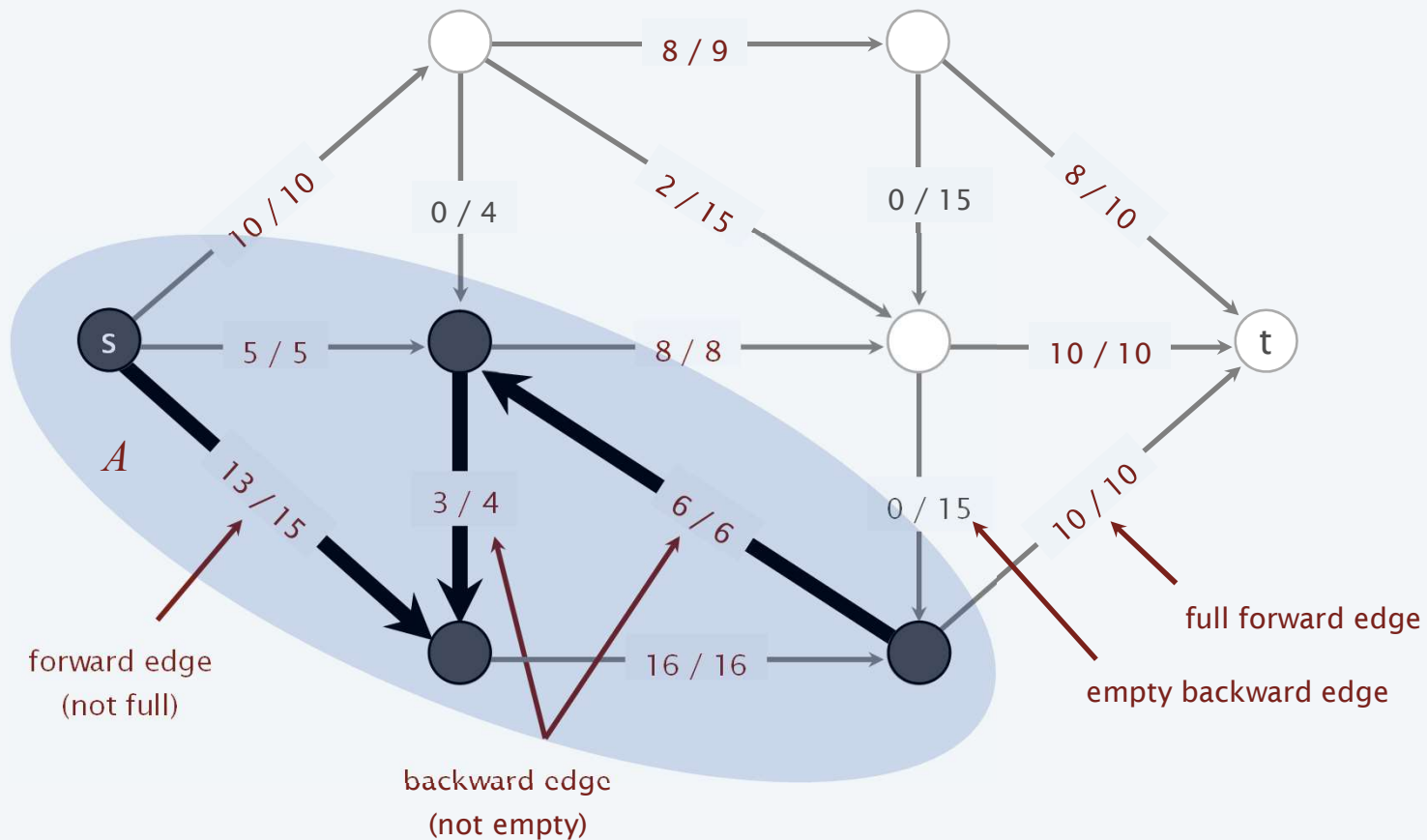
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &= \sum_{e \text{ out of } A} c(e) \quad \blacksquare \\ &= \text{cap}(A, B) \end{aligned}$$



Computing a min-cut from a max-flow

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

- By augmenting path theorem, no augmenting paths in G_f^* (the residual graph).
- Compute A = set of nodes reachable from s in G_f^* .



Running Time

Assumption. All capacities are integers between 1 and C .

Invariant. Flow values $f(e)$ and residual capacities $c_f(e)$ remain integers throughout the algorithm.

Theorem. The algorithm terminates in at most $v(f^*) \leq nC$ iterations, and hence, takes $O(mnC)$ time.

Pf. Each augmentation increases flow value by at least 1, and maximum outflow from s is at most nC . ▀

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, it follows from invariant. ▀

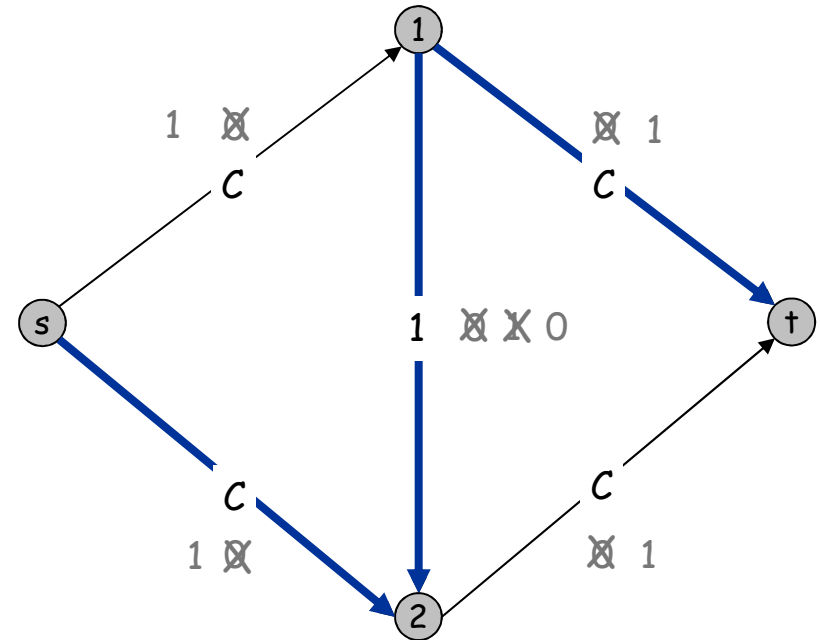
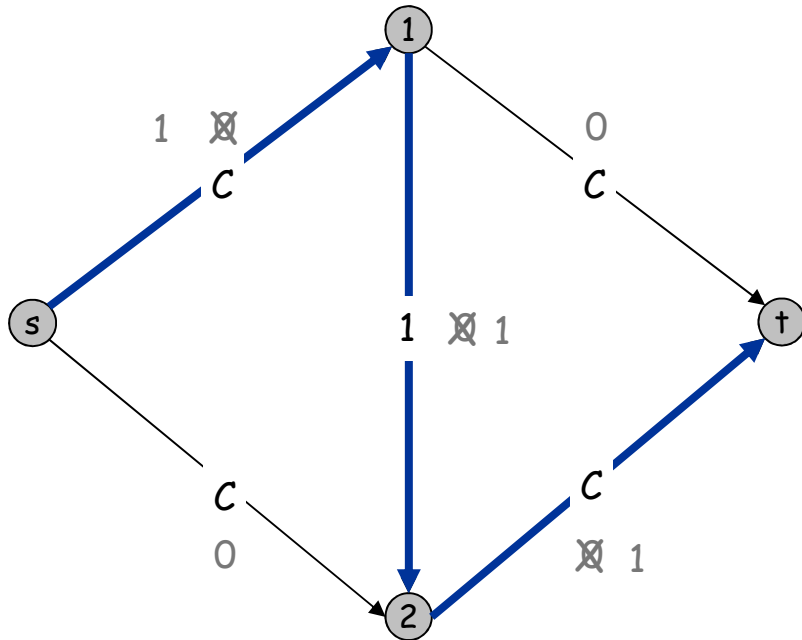
7.3 Choosing Good Augmenting Paths

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 edge capacity is C , then it can take $\geq C$ iterations.



Worse case for Ford-Fulkerson algorithm

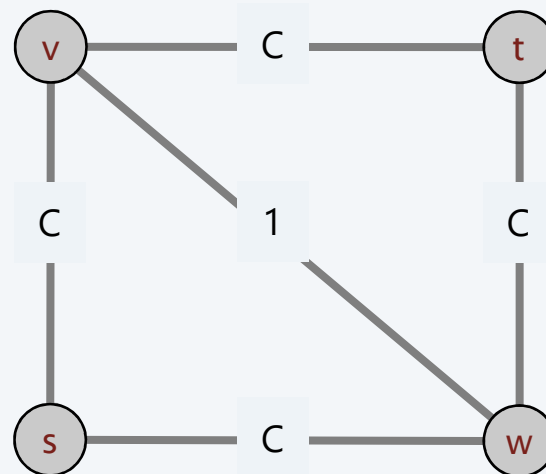
Q. Is generic Ford-Fulkerson algorithm poly-time in input size?

← $m, n,$ and $\log C$

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

- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$
- ...
- $s \rightarrow v \rightarrow w \rightarrow t$
- $s \rightarrow w \rightarrow v \rightarrow t$

← each augmenting path
sends only 1 unit of flow
augmenting paths
= $\text{degree}(s) * C$
= $2 * C$



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!
 - **See Demo of the Pathological case**

Goal: choose augmenting paths that:

- Can be found efficiently.
- Require fewer iterations (by choosing larger increments).

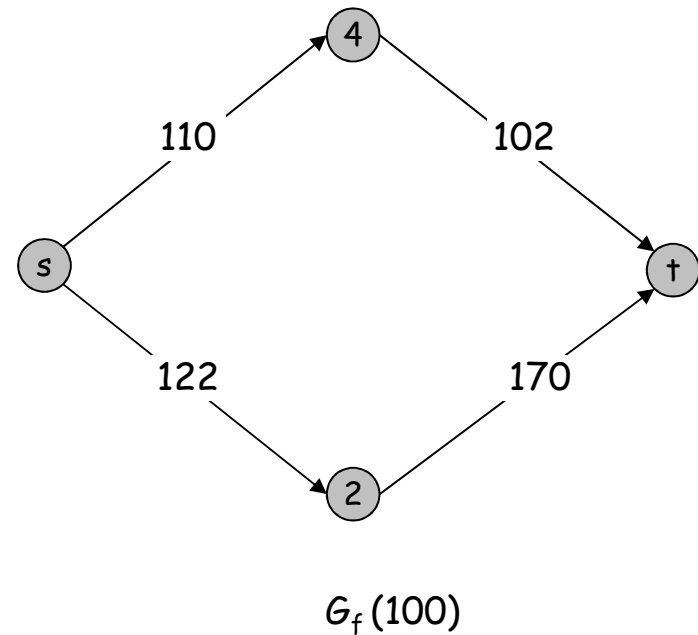
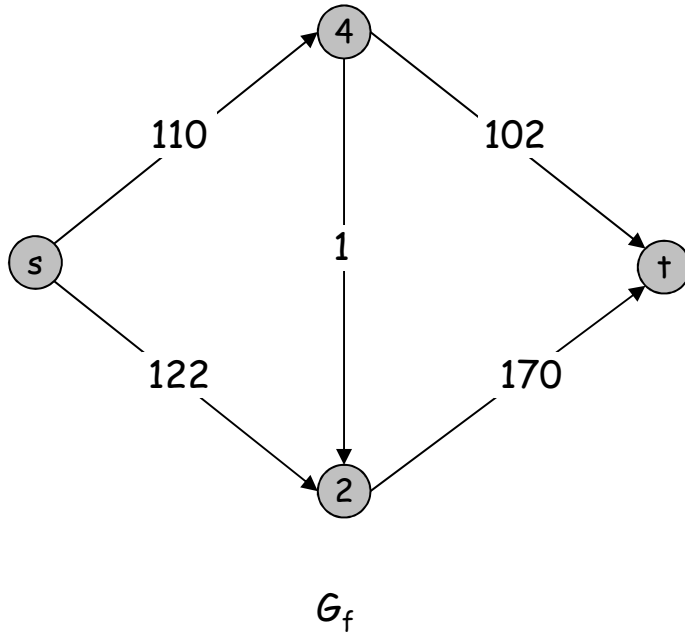
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- 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 edges with capacity at least Δ .



Capacity Scaling

```
Scaling-Max-Flow( $G, s, t, c$ ) {  
  foreach  $e \in E$   $f(e) \leftarrow 0$   
   $\Delta \leftarrow$  largest power of 2 smaller than or equal to max  
    capacity edge  $C$  (out of  $s$  is sufficient)  
   $G_f \leftarrow$  residual graph  
  
  while ( $\Delta \geq 1$ ) {  
     $G_f(\Delta) \leftarrow \Delta$ -residual graph  
    while (there exists augmenting 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 in $[1, \dots, C]$.

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

Correctness. Upon termination, 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 is no augmenting path. ▫

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.
- Lemma 2 $\Rightarrow v(f^*) \leq v(f) + m (2\Delta)$.
- 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.** ▀

Pf. Follows from Lemma 1 and Lemma 3. ▀

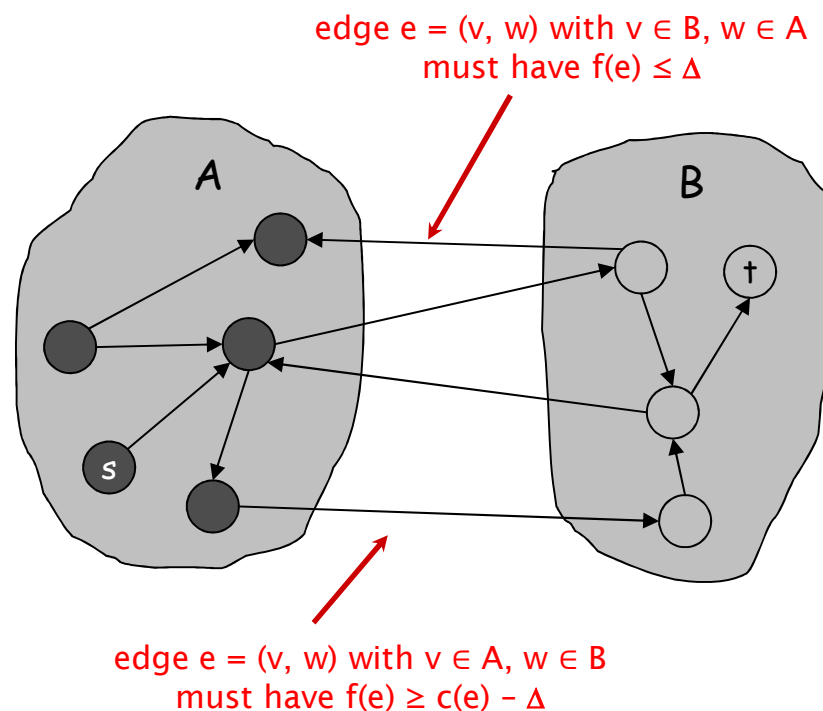
Capacity Scaling: Running Time

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 $\text{cap}(A, B) \leq v(f) + m \Delta$.
- 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 , $t \notin A$.

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



Maximum flow algorithms: theory

year	method	worst case	discovered by
1951	simplex	$O(m^3 C)$	Dantzig
1955	augmenting path	$O(m^2 C)$	Ford-Fulkerson
1970	shortest augmenting path	$O(m^3)$	Dinic, Edmonds-Karp
1970	fattest augmenting path	$O(m^2 \log m \log(m C))$	Dinic, Edmonds-Karp
1977	blocking flow	$O(m^{5/2})$	Cherkasky
1978	blocking flow	$O(m^{7/3})$	Galil
1983	dynamic trees	$O(m^2 \log m)$	Sleator-Tarjan
1985	capacity scaling	$O(m^2 \log C)$	Gabow
1997	length function	$O(m^{3/2} \log m \log C)$	Goldberg-Rao
2012	compact network	$O(m^2 / \log m)$	Orlin
?	?	$O(m)$	?

max-flow algorithms for sparse digraphs with m edges, integer capacities between 1 and C