

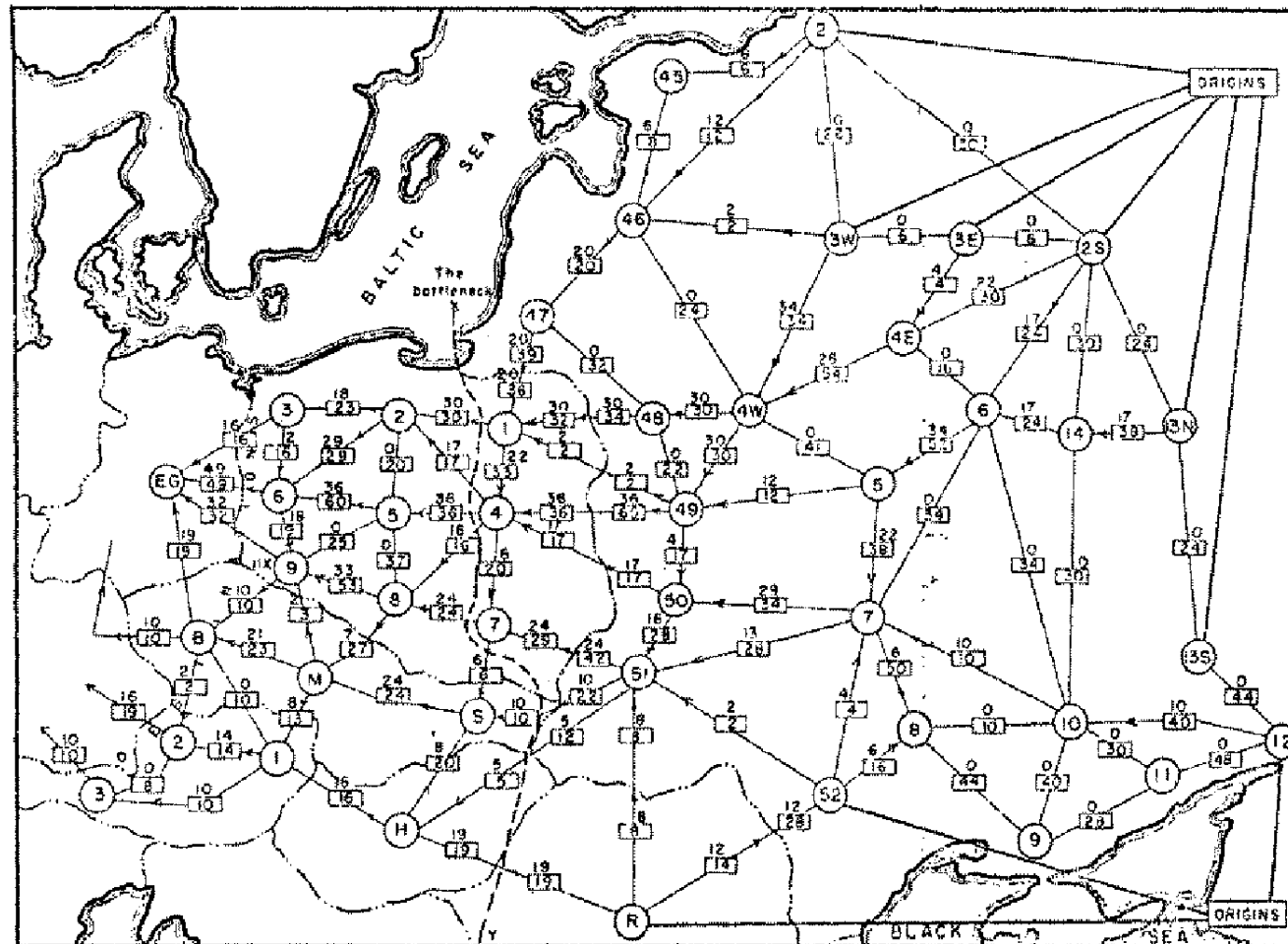
Chapter 7

Network Flow



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

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.

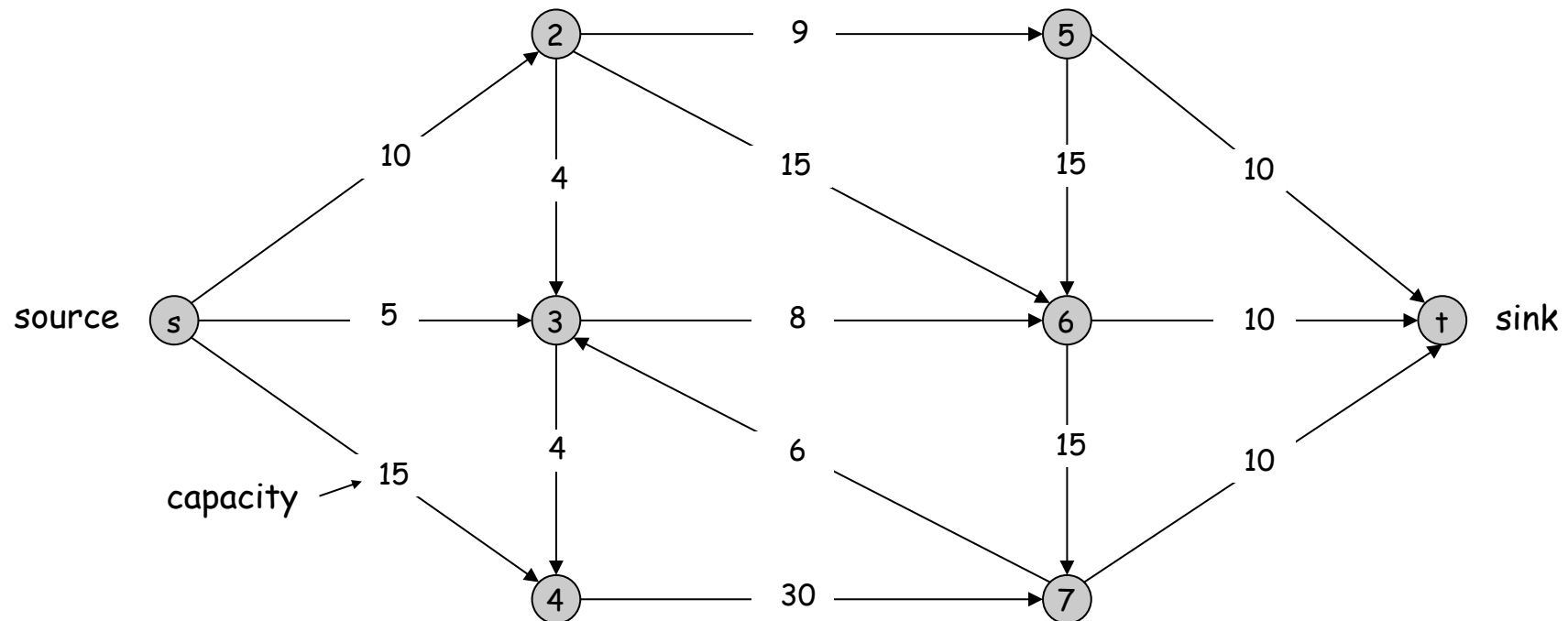
Nontrivial applications / reductions.

- 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

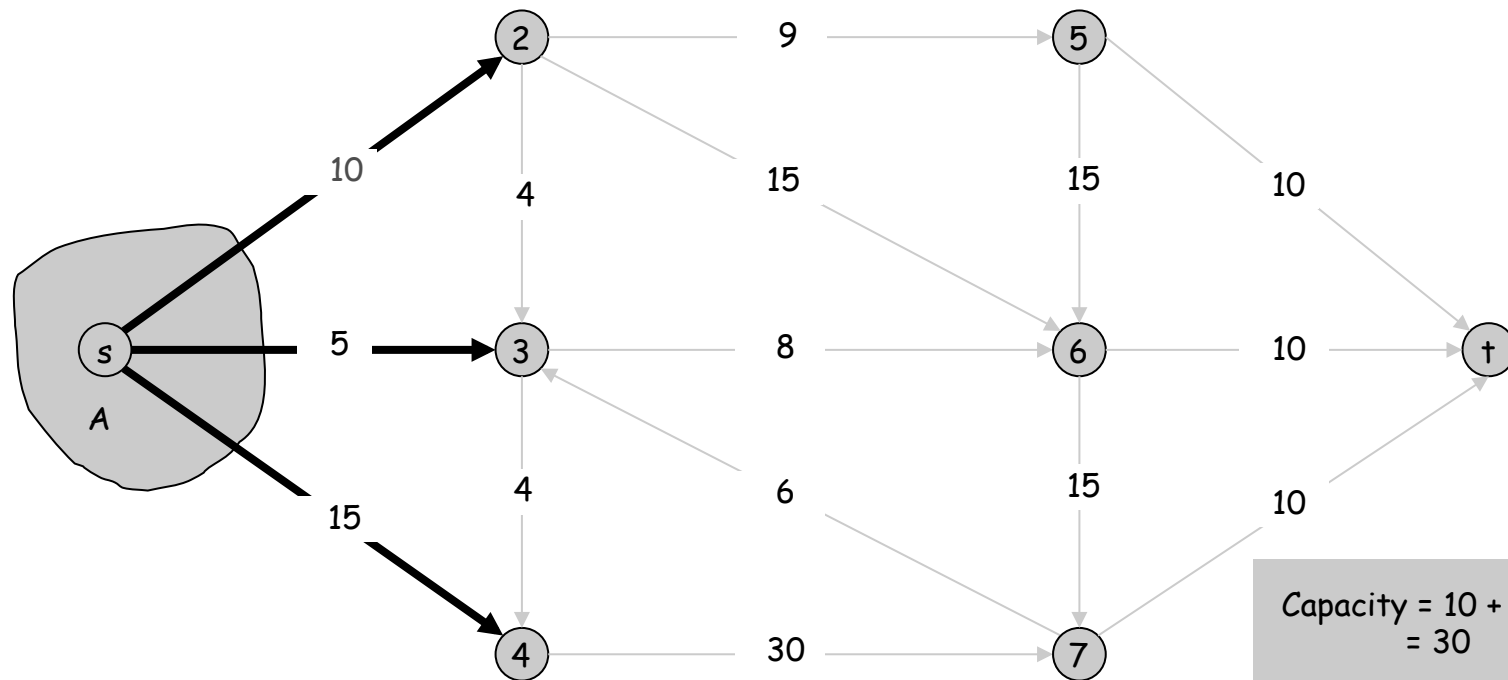
For the nodes other than s and t.

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 $t \in B$.

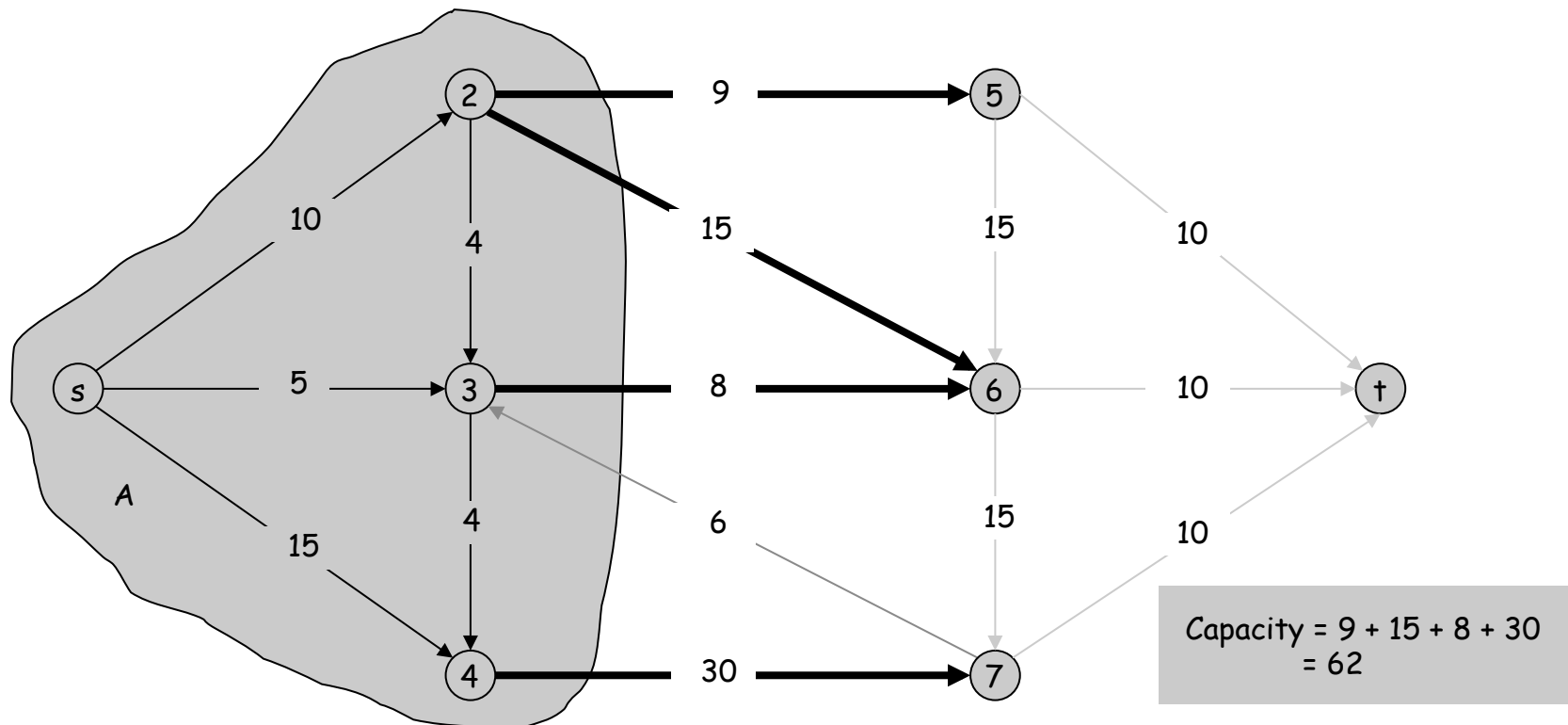
Def. The **capacity** of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Cuts

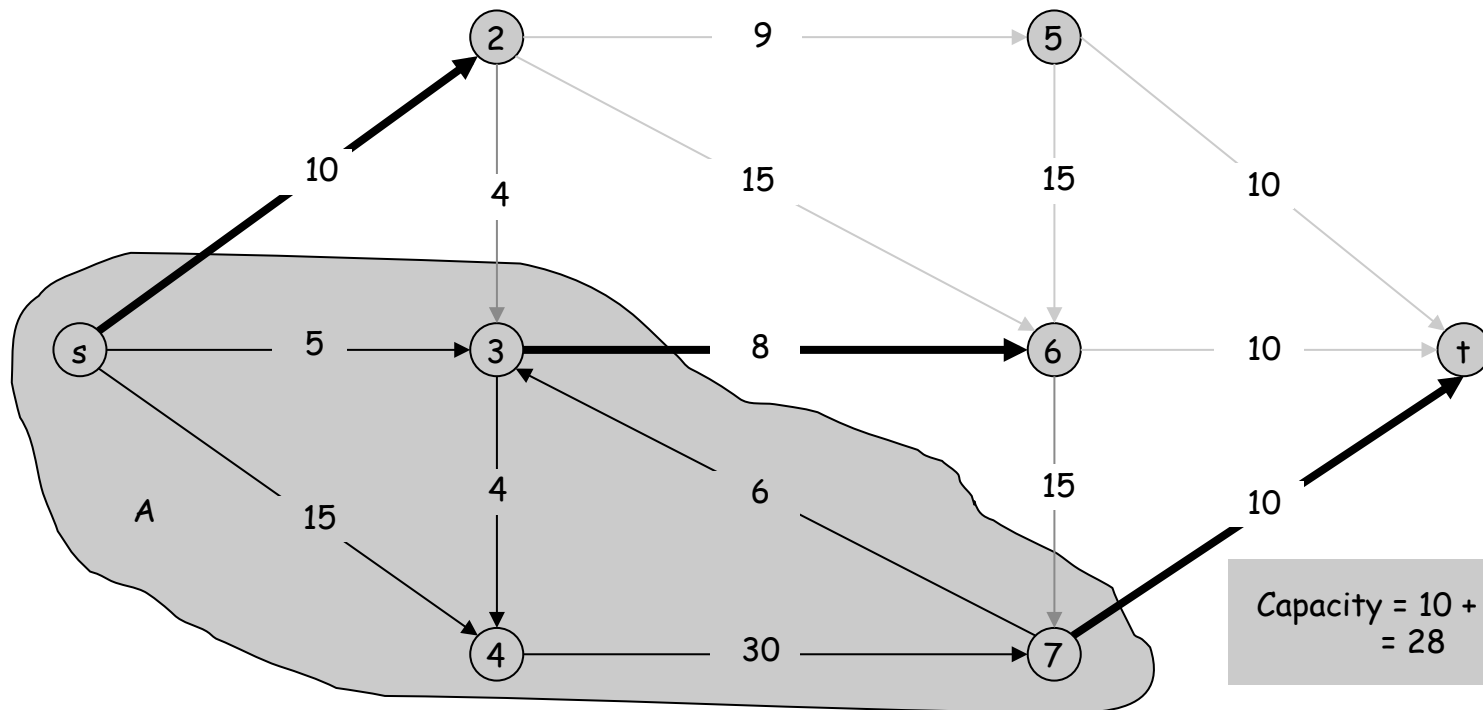
Def. An **s-t cut** is a partition (A, B) of V with $s \in A$ and $t \in B$.

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.

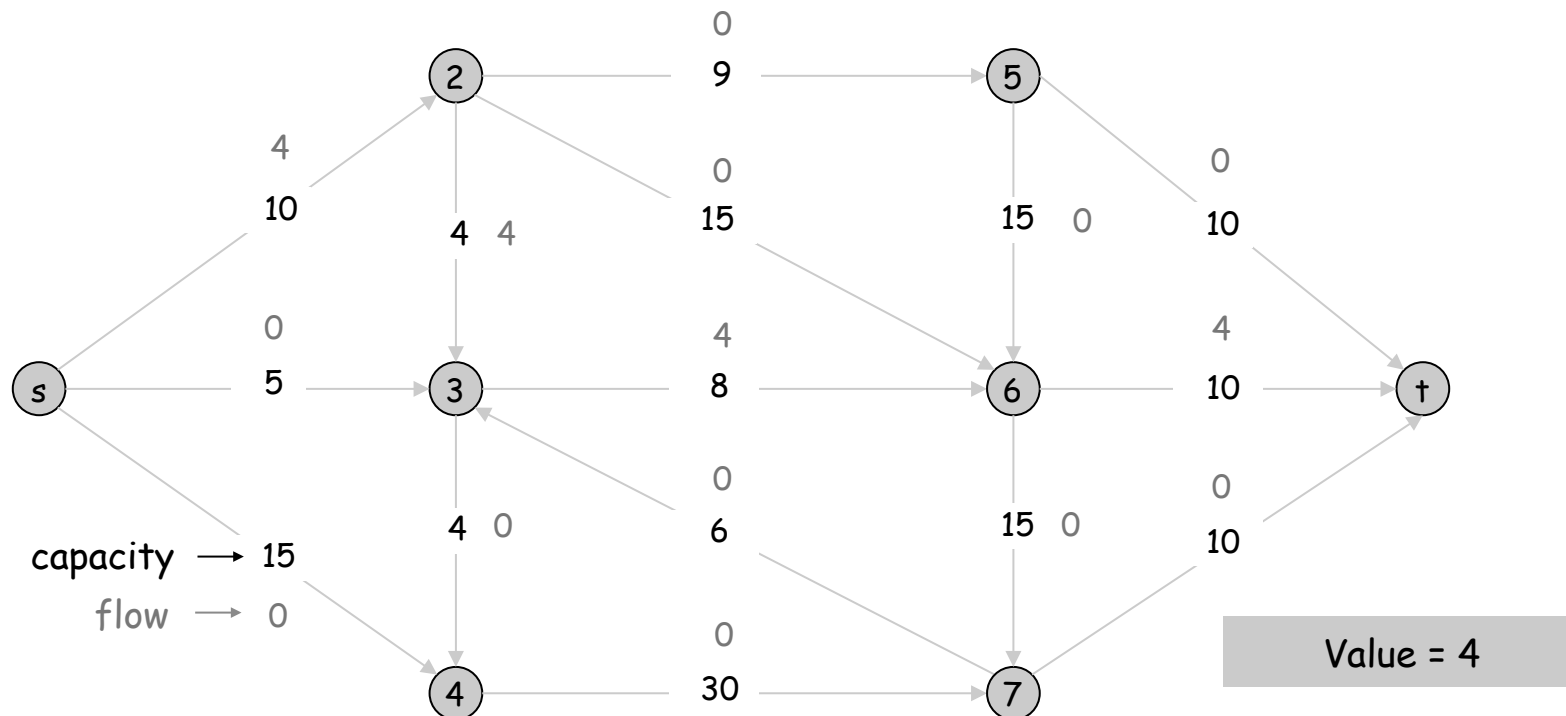


Flows

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

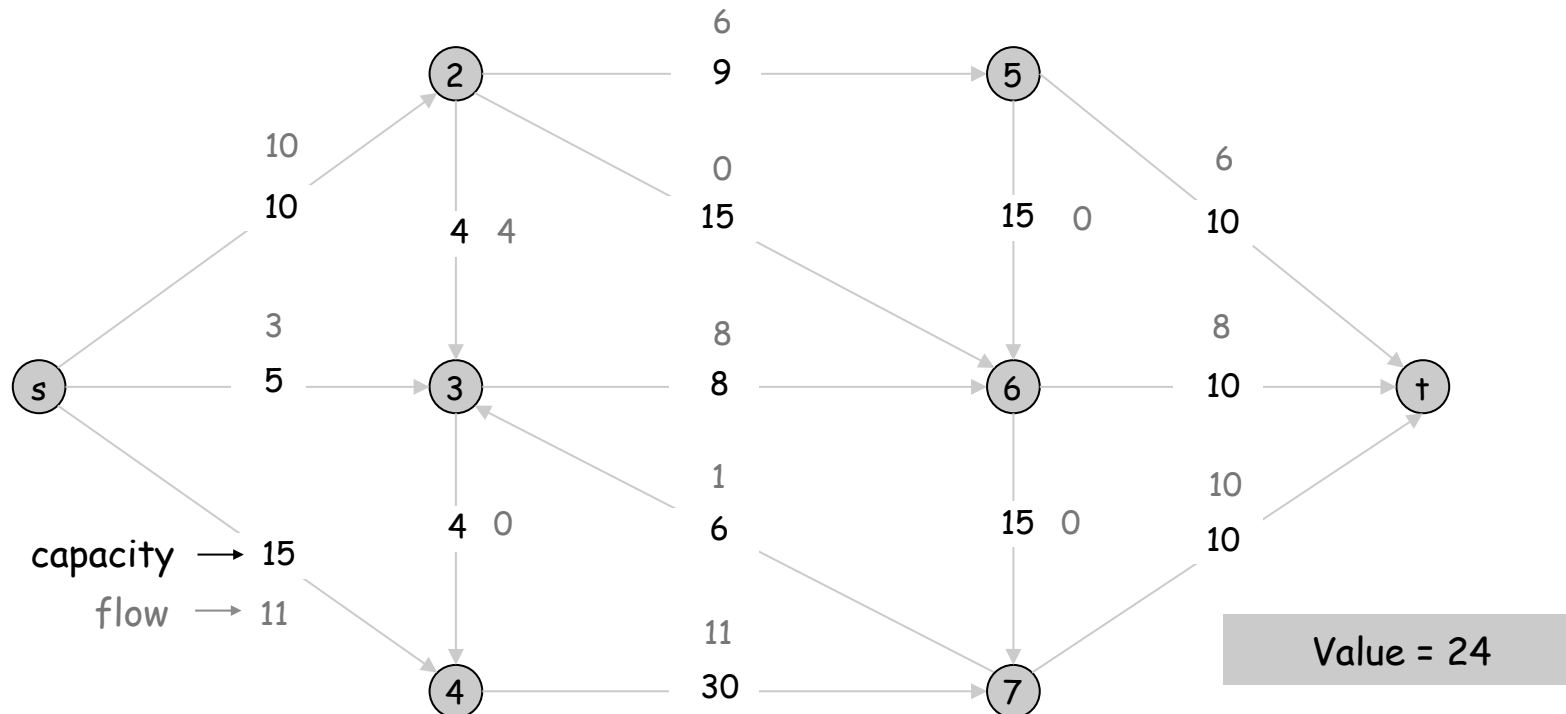


Flows

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

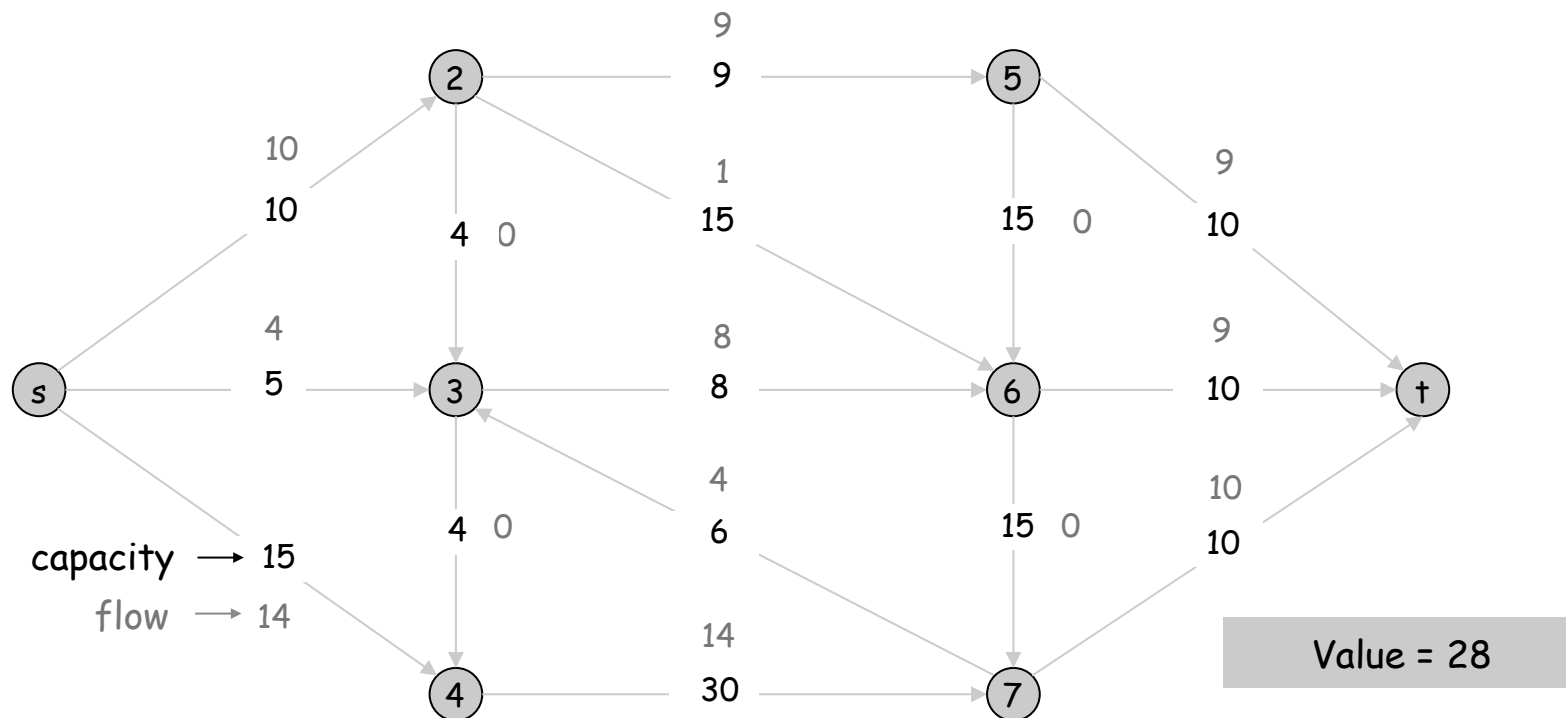
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

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



Maximum Flow Problem

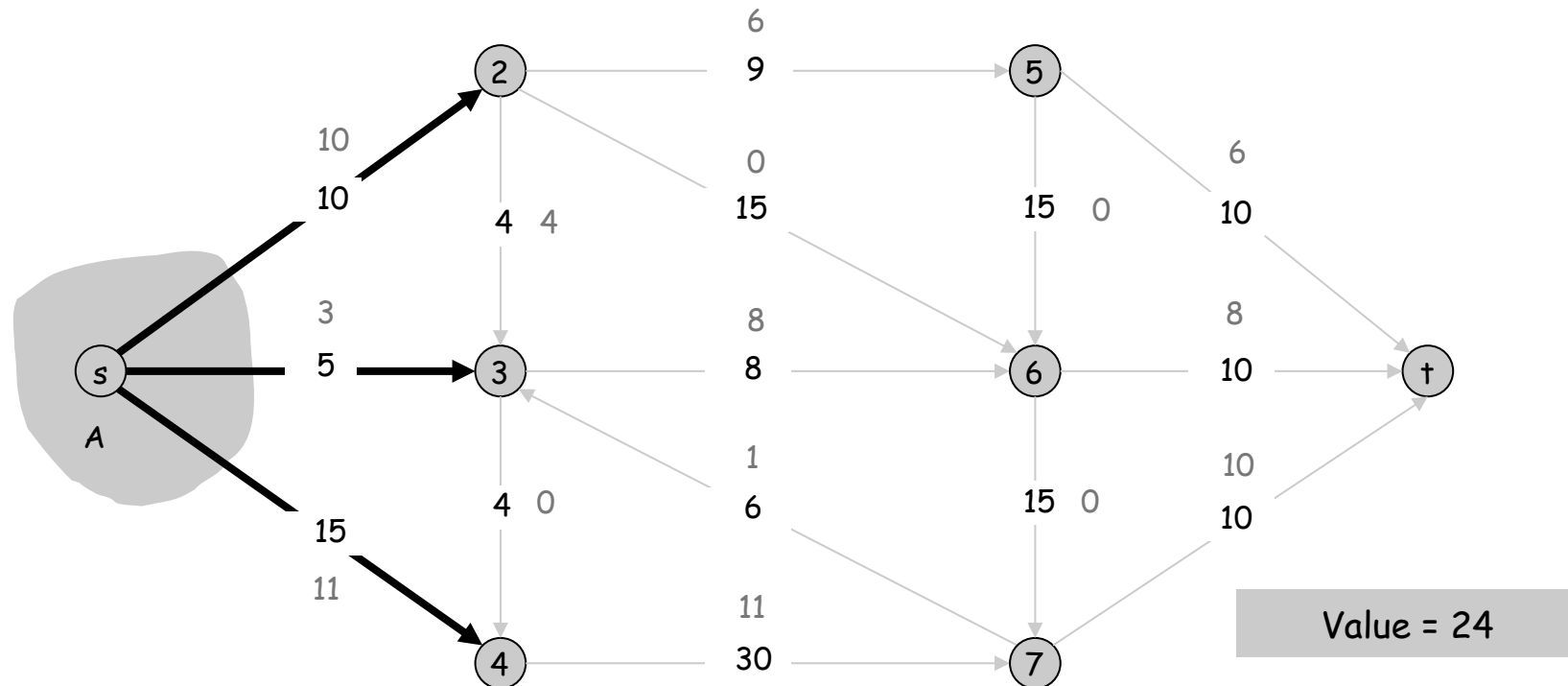
Max flow problem. Find s-t flow of maximum value.



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 .

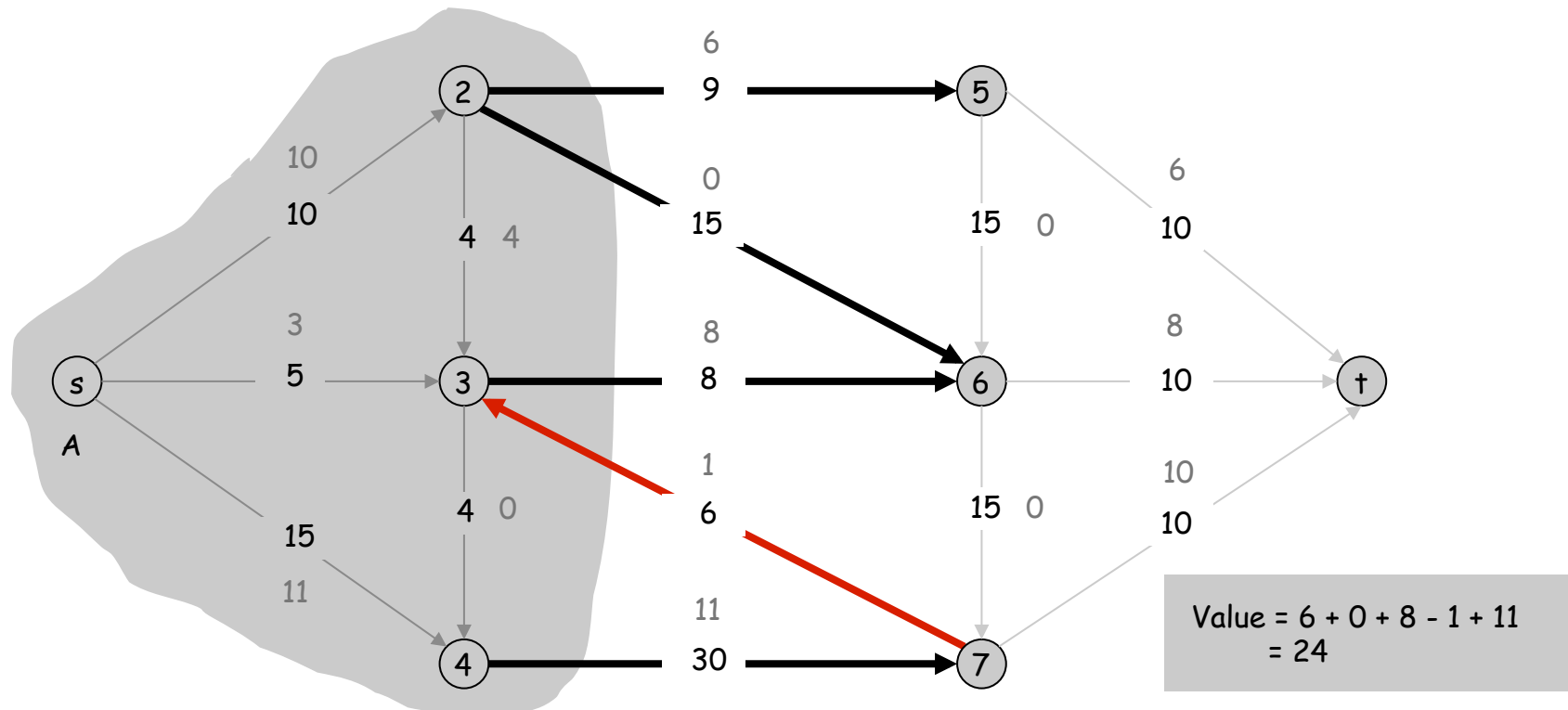
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



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 .

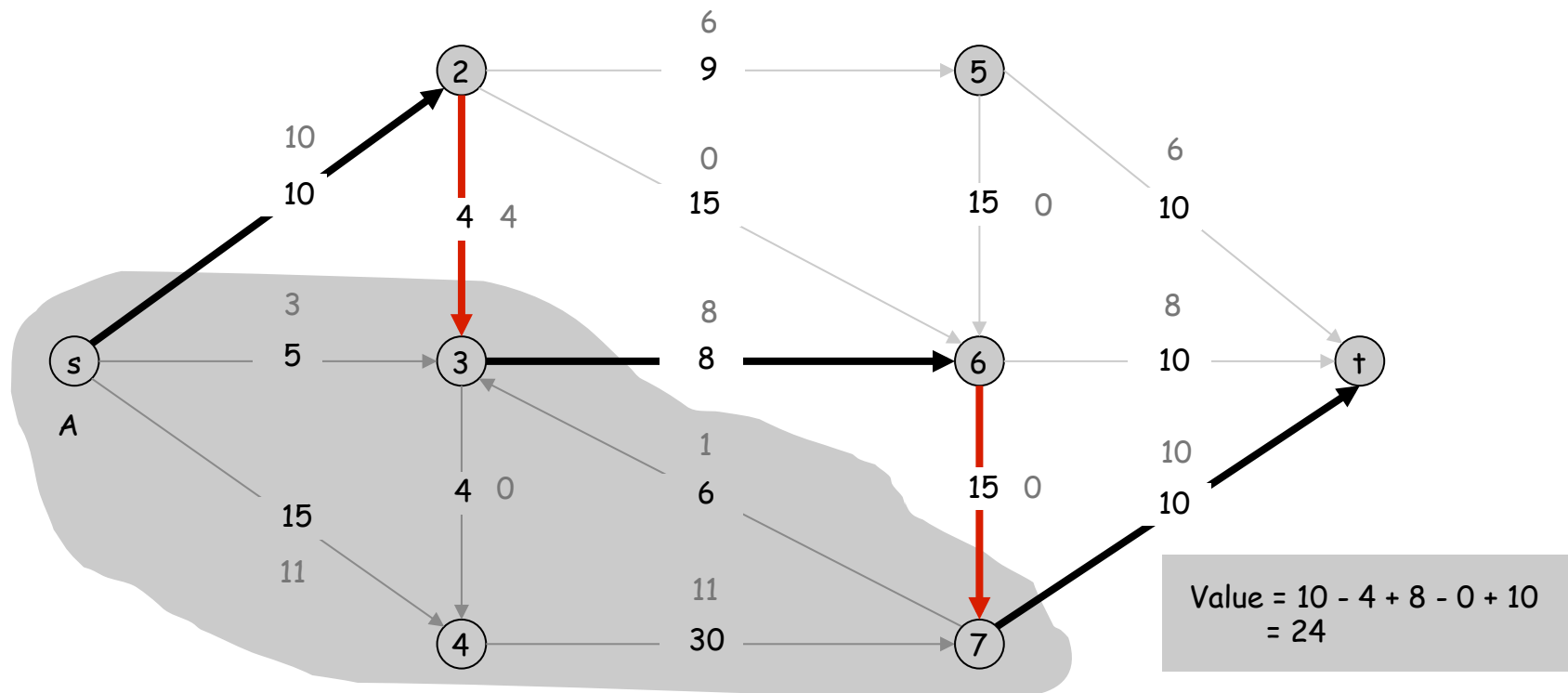
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



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

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

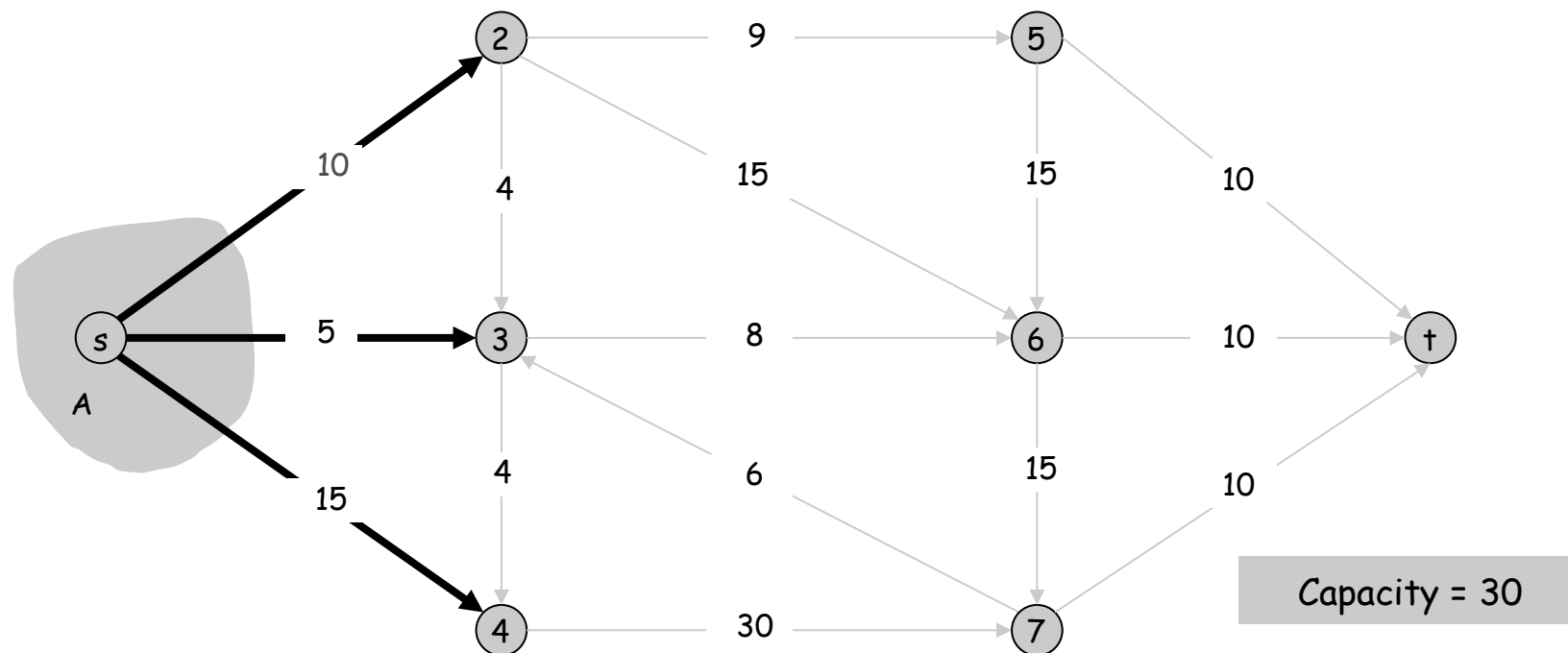
Pf.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } s} f(e) \\ \text{by flow conservation, all terms} &\longrightarrow = \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right) \\ \text{except } v = s \text{ are } 0 & \\ &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e). \end{aligned}$$

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 ≤ 30

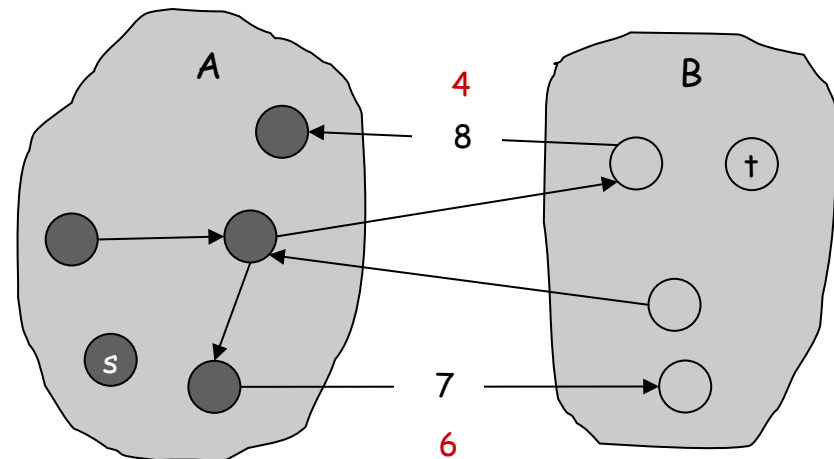


Flows and Cuts

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.

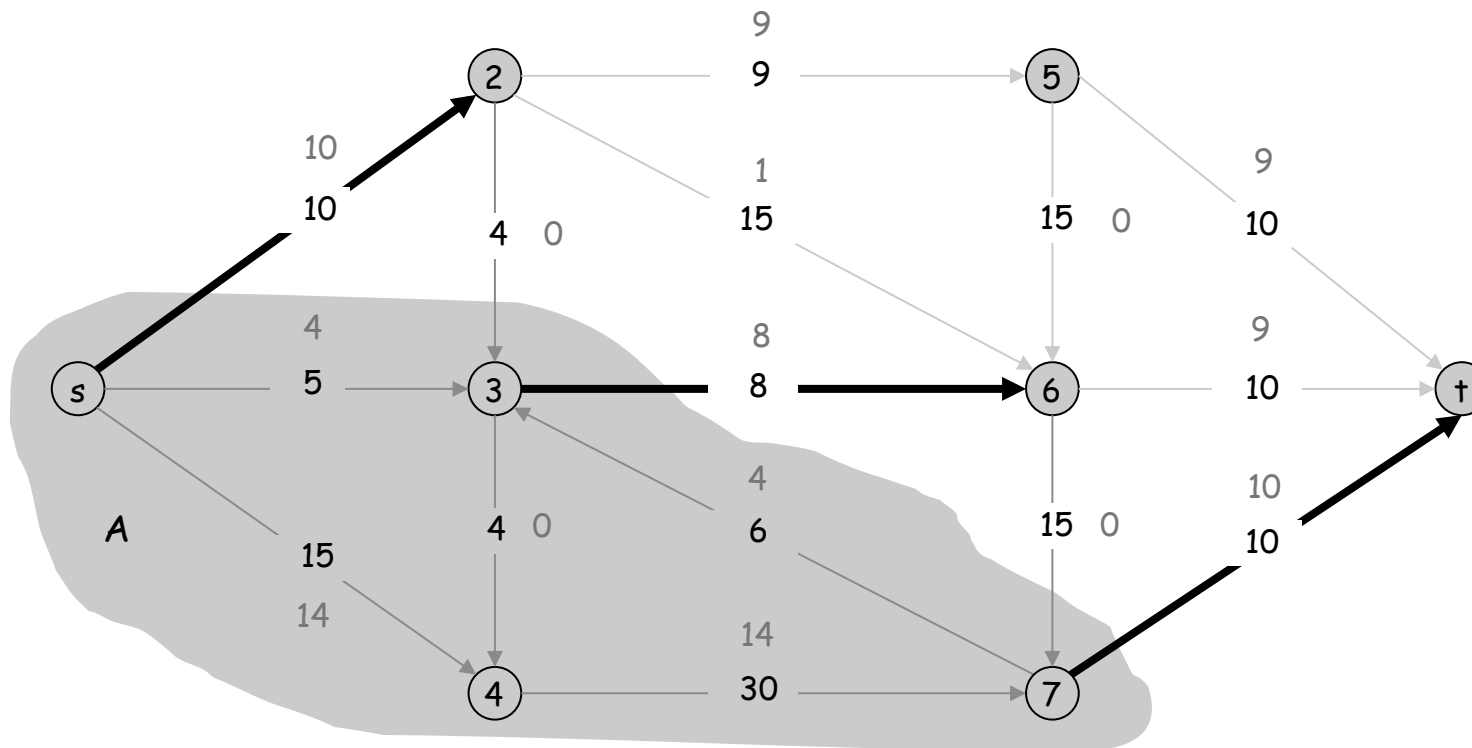
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &= \text{cap}(A, B) \quad \blacksquare \end{aligned}$$



Certificate of Optimality

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

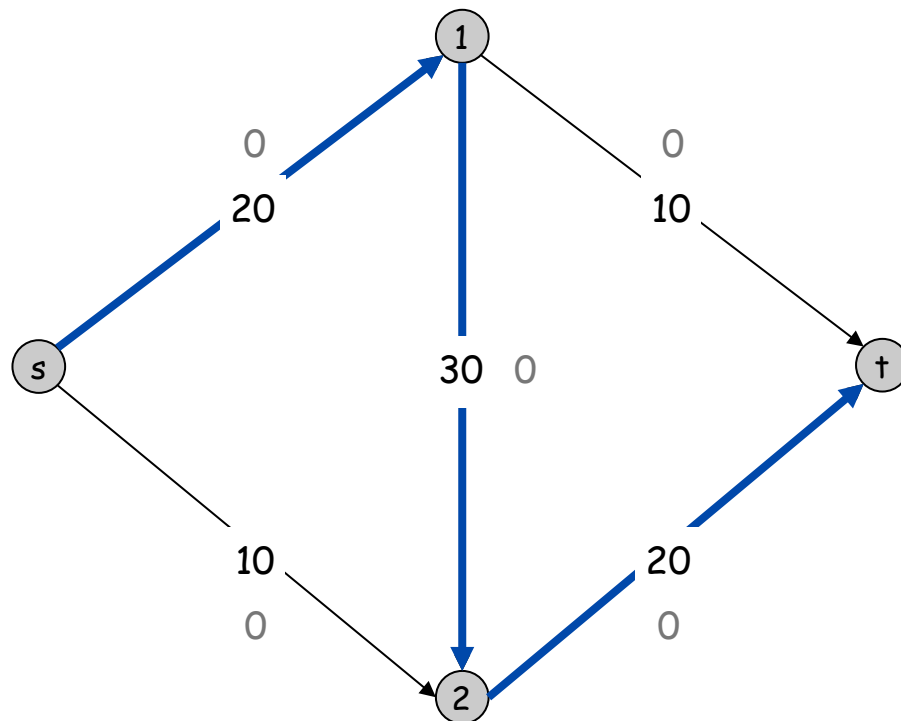
Value of flow = 28
Cut capacity = 28 \Rightarrow Flow value \leq 28



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.

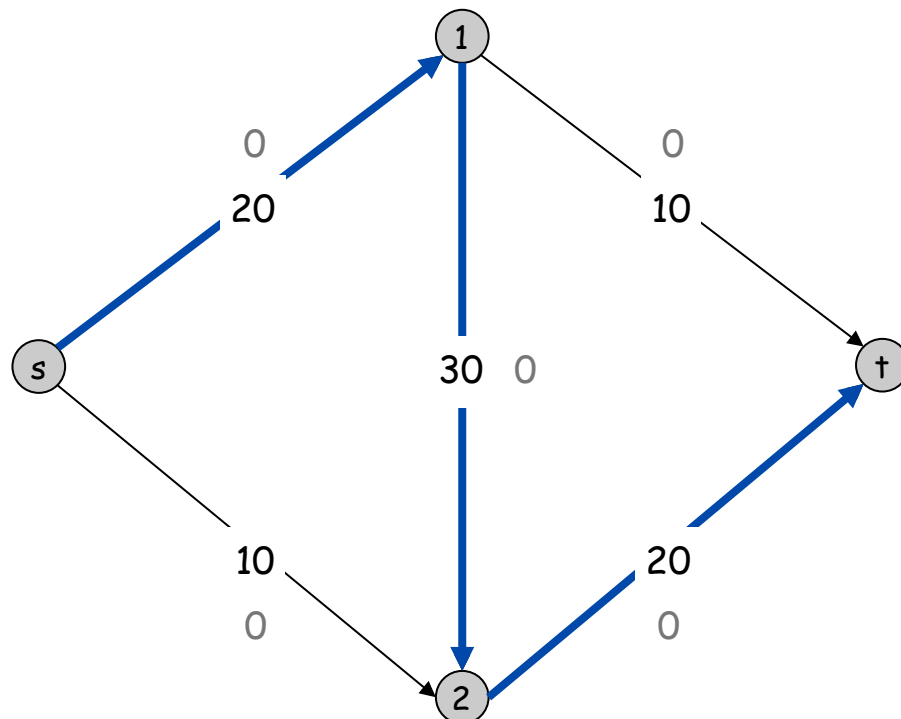


Flow value = 0

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.
- Q:How do you find an s-t path?

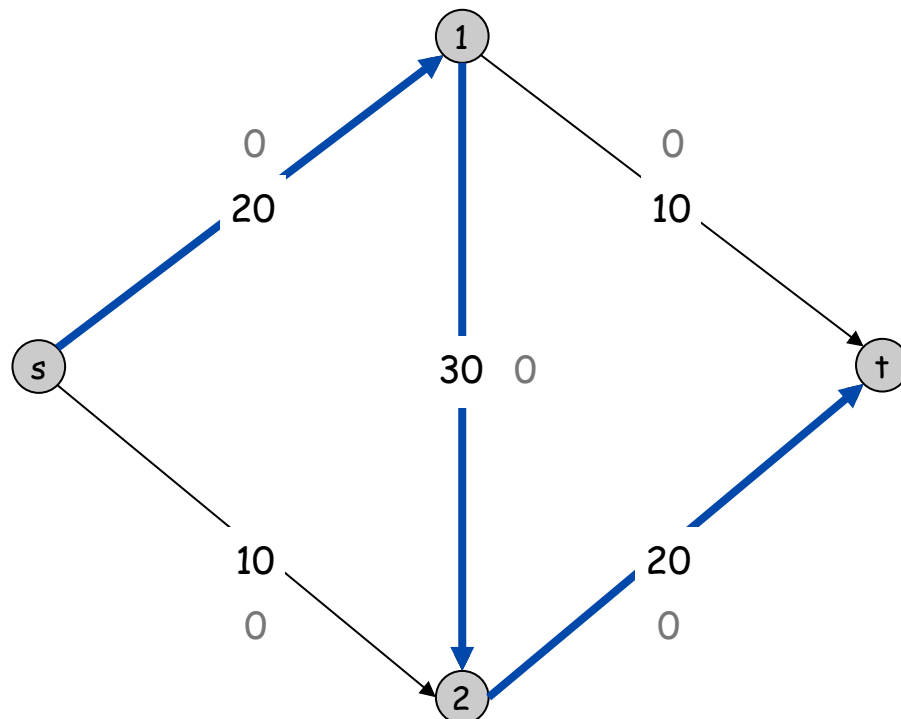


Flow value = 0

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.
- Q: How do you find an s-t path?
Do BFS or DFS.

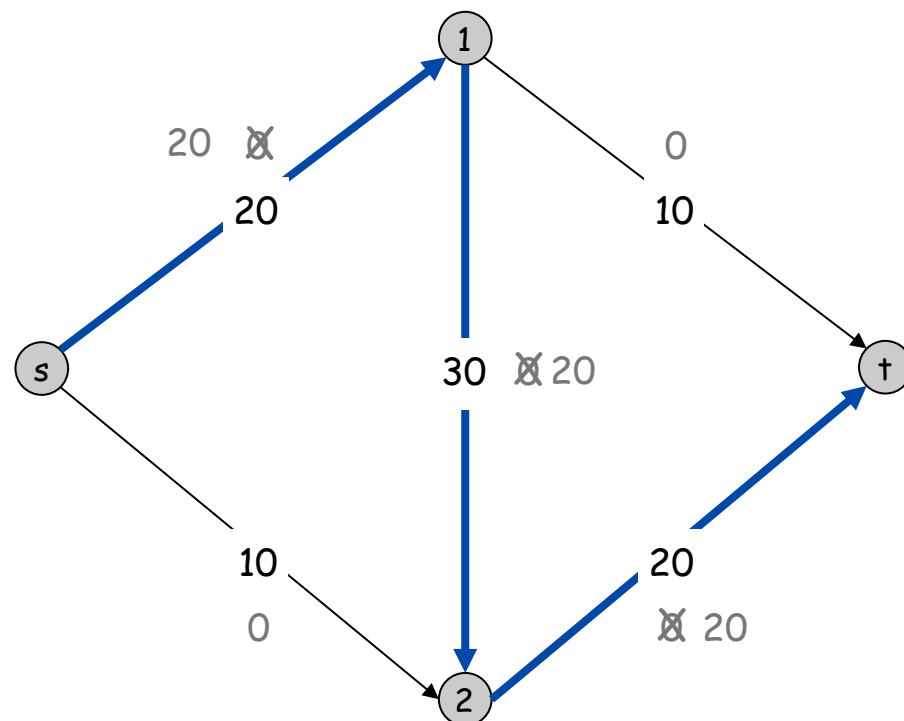


Flow value = 0

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.



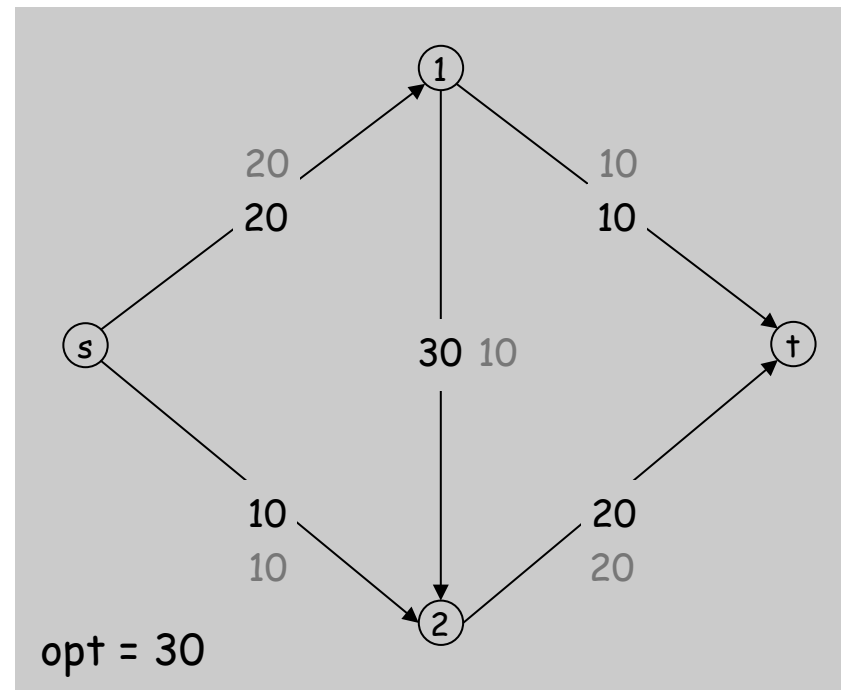
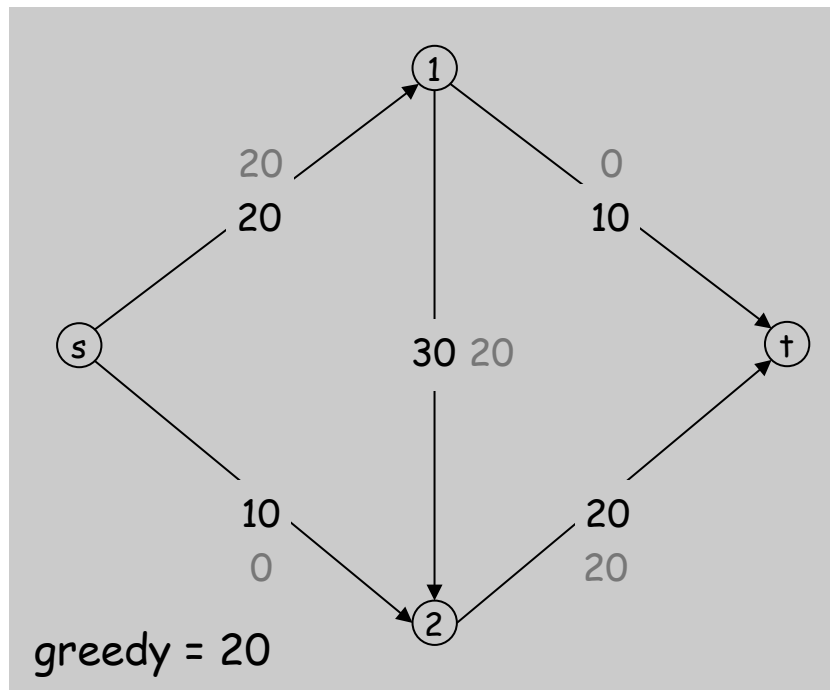
Flow value = 20

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**.

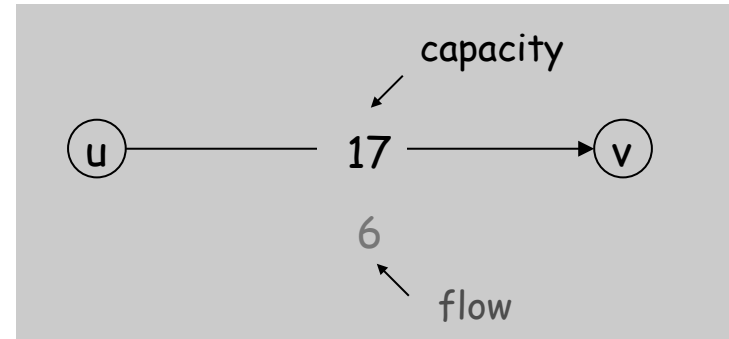
← locally optimality \nRightarrow global optimality



Residual Graph

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

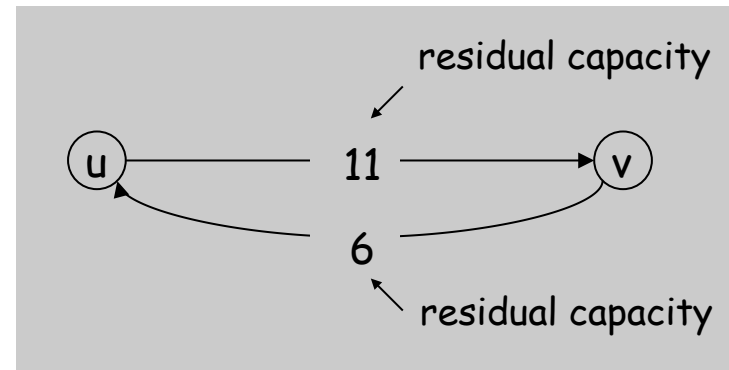
- Flow $f(e)$, capacity $c(e)$.



Residual edge.

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

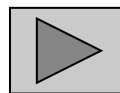
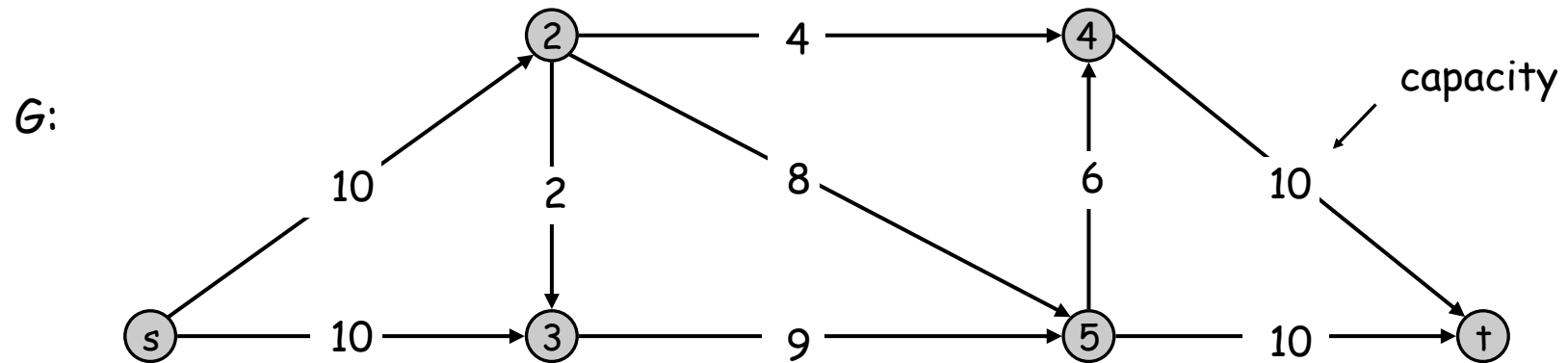
$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



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 $v(f) = \text{cap}(A, B)$.
- (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.

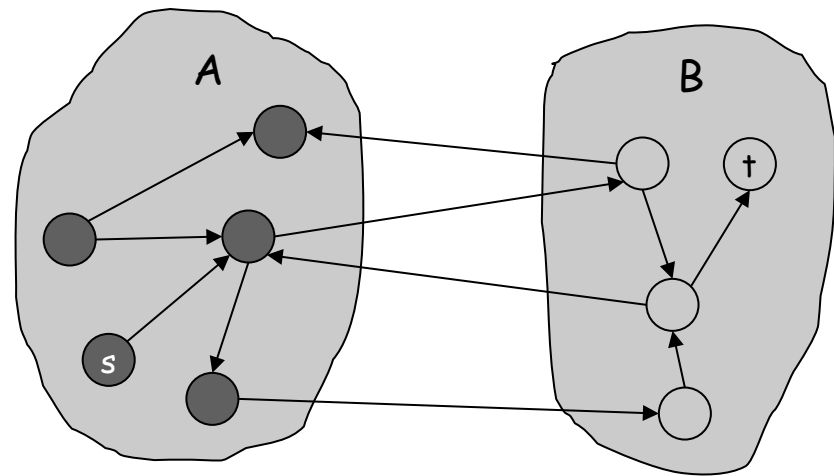
- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

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 , $t \notin A$.

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



original network

Running Time

Assumption. All capacities are integers between 1 and C where $C = \sum_{e \text{ out of } s} c(e)$

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. ■

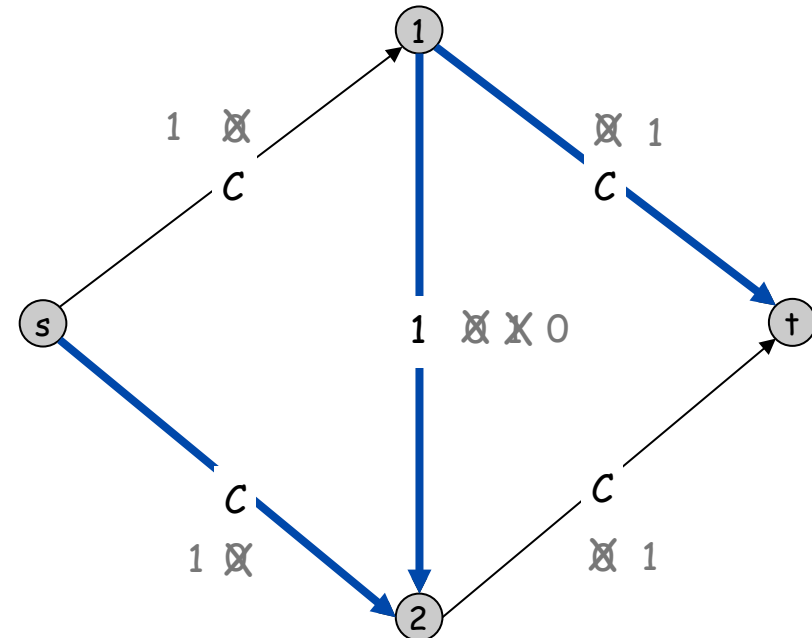
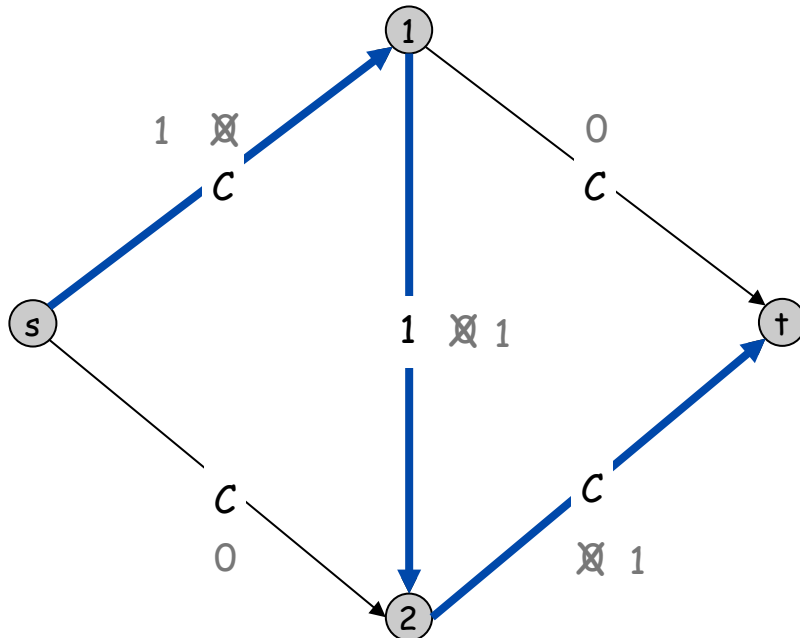
7.3 Choosing Good Augmenting Paths

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.

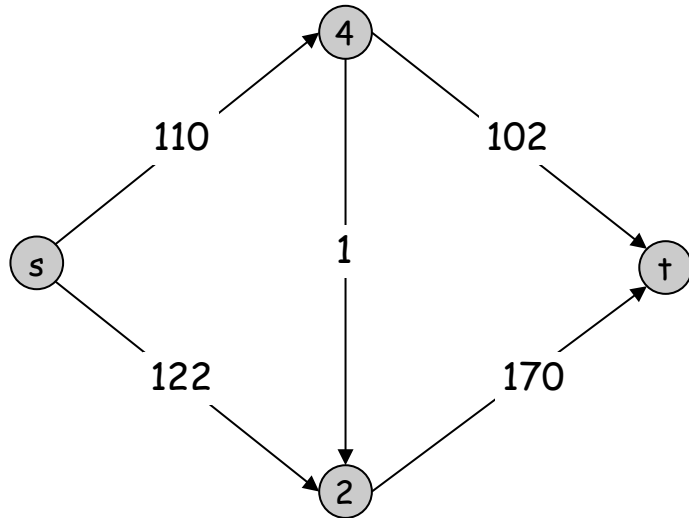
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

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

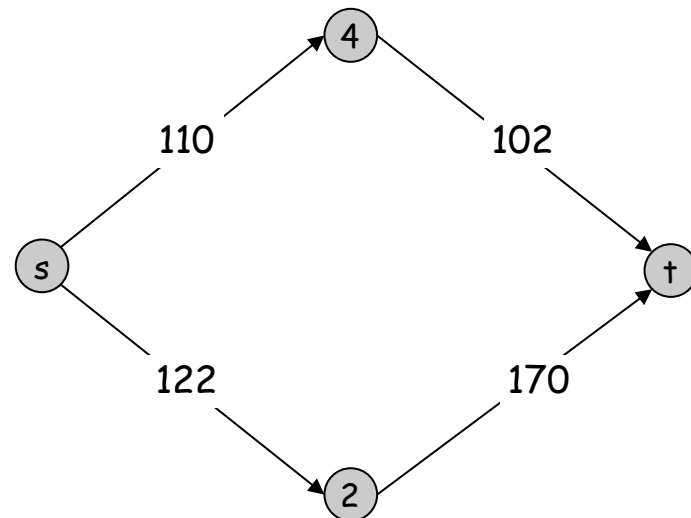
Capacity Scaling

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  $e \in E$   $f(e) \leftarrow 0$   
   $\Delta \leftarrow$  smallest power of 2 greater than or equal to  $C$   
   $G_f \leftarrow$  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 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 $C \leq \Delta < 2C$. 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.
- $L2 \Rightarrow v(f^*) \leq v(f) + m (2\Delta)$.
- 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. ■

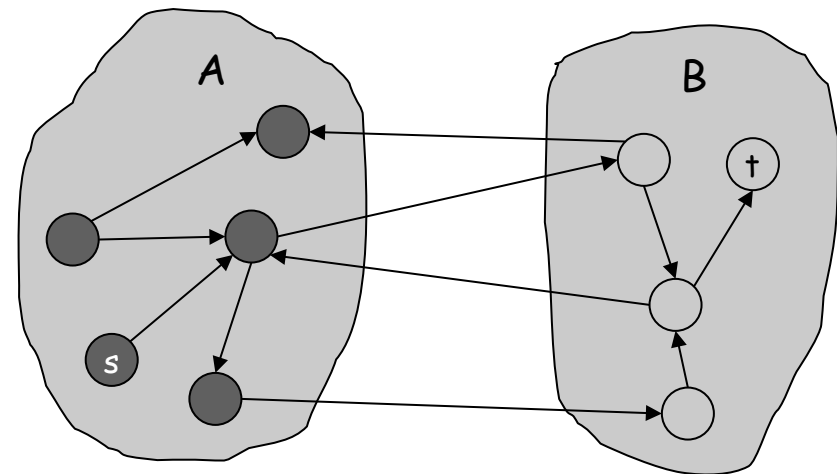
Capacity Scaling: Running Time

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

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



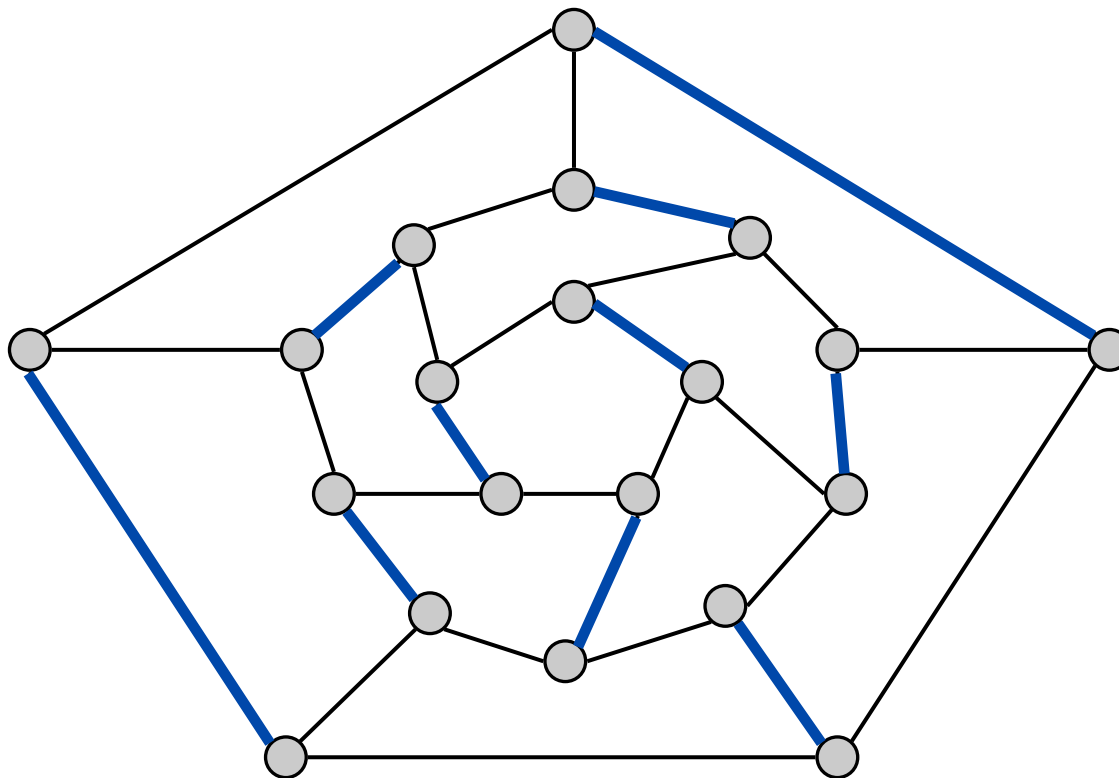
original network

7.5 Bipartite Matching

Matching

Matching.

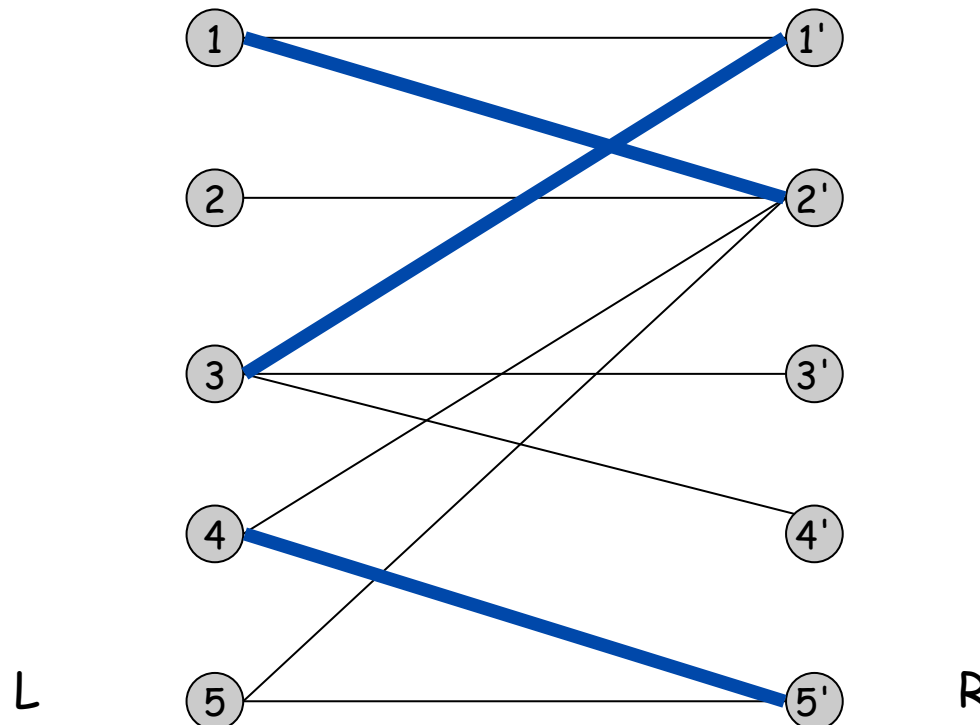
- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .
- Max matching: find a max cardinality matching.



Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .
- Max matching: find a max cardinality matching.

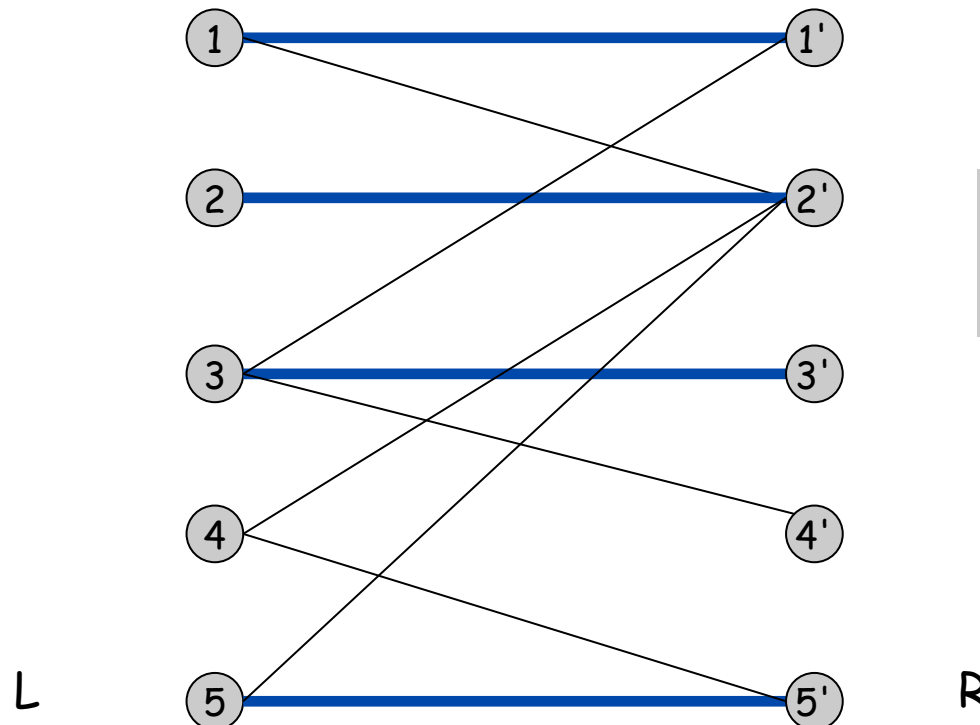


matching
1-2', 3-1', 4-5'

Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most one edge in M .
- Max matching: find a max cardinality matching.

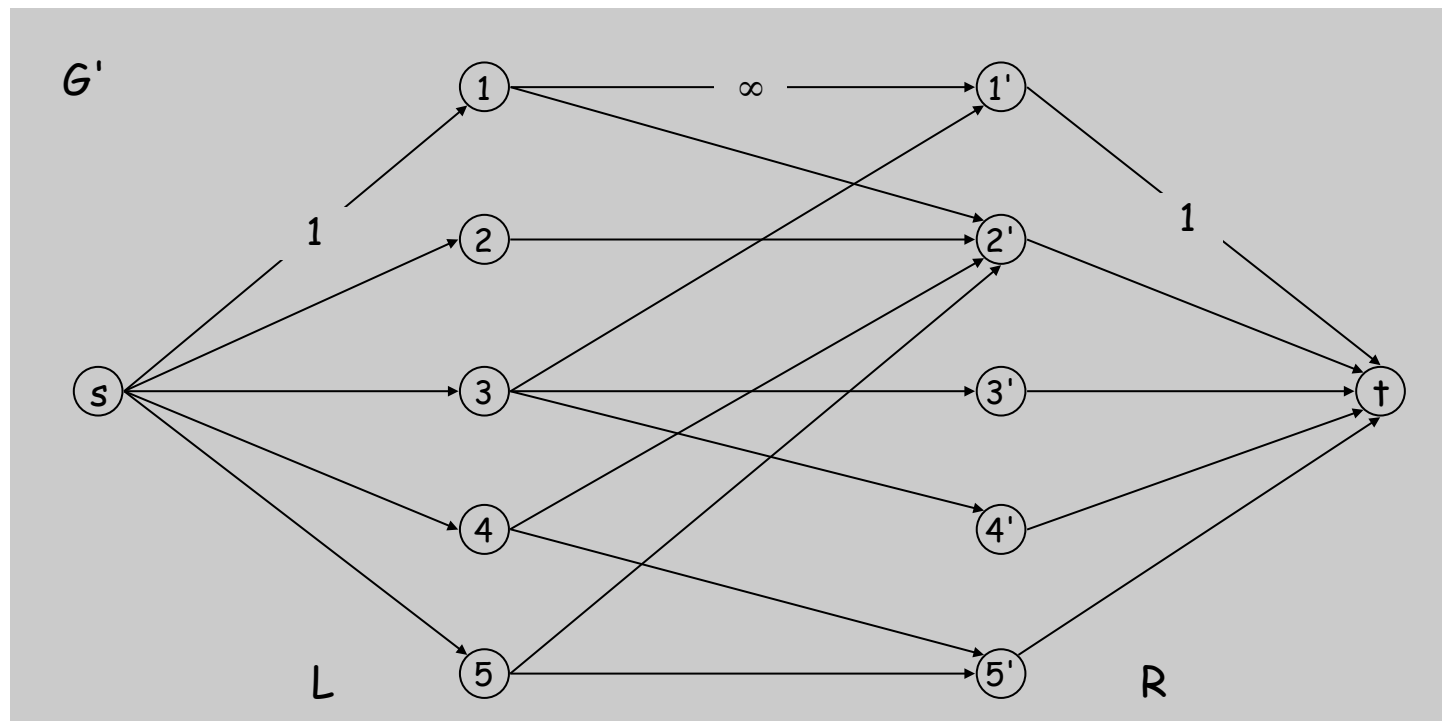


max matching
1-1', 2-2', 3-3' 4-4'

Bipartite Matching

Max flow formulation.

- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
- Direct all edges from L to R , and assign infinite (or unit) capacity.
- Add source s , and unit capacity edges from s to each node in L .
- Add sink t , and unit capacity edges from each node in R to t .

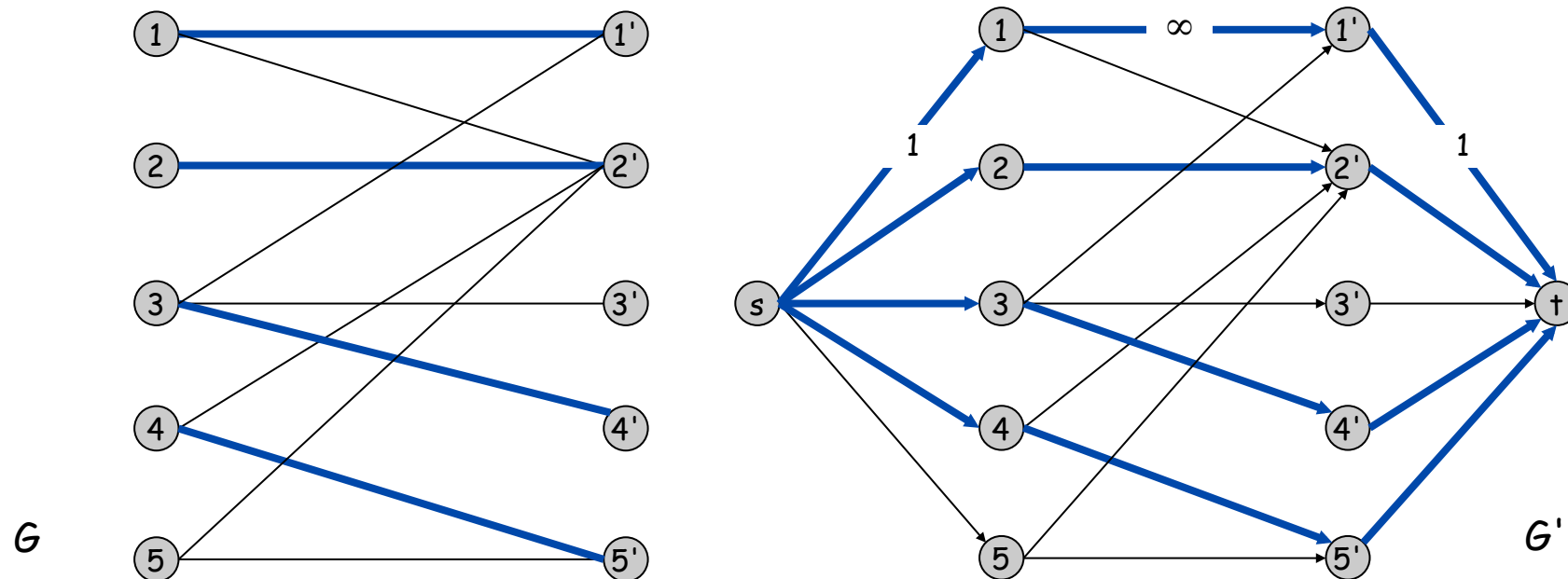


Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in G = value of max flow in G' .

Pf. \leq

- Given max matching M of cardinality k .
- Consider flow f that sends 1 unit along each of k paths.
- f is a flow, and has cardinality k . ▪

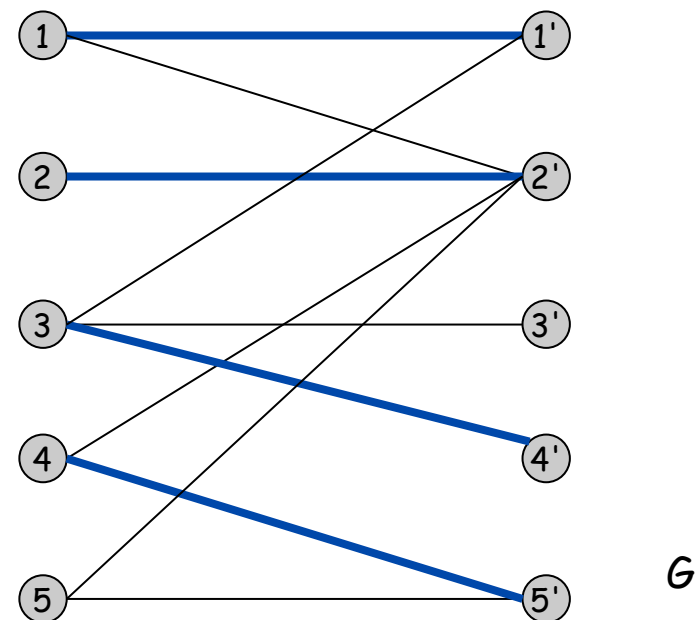
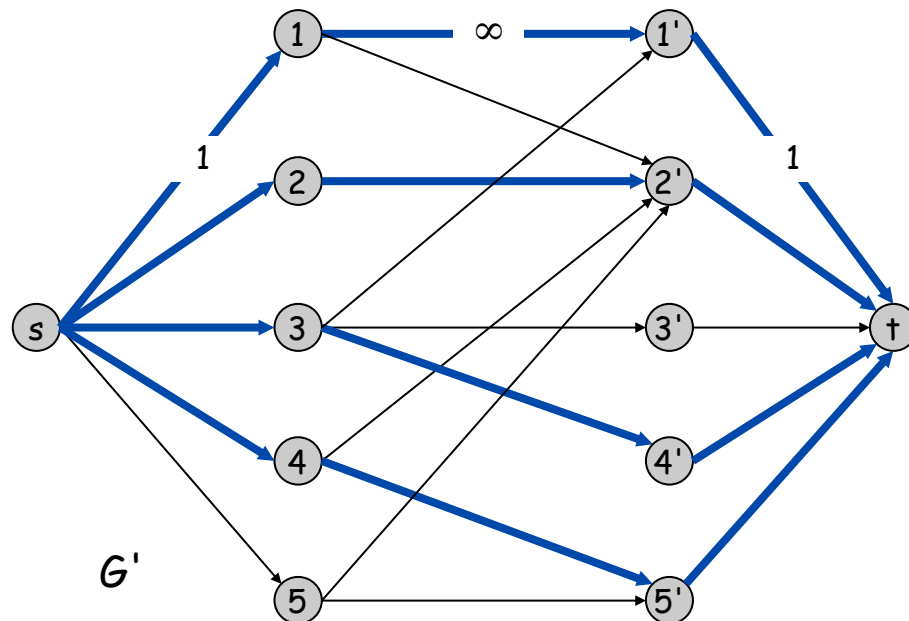


Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in G = value of max flow in G' .

Pf. \geq

- Let f be a max flow in G' of value k .
- Integrality theorem \Rightarrow k is integral and can assume f is 0-1.
- Consider M = set of edges from L to R with $f(e) = 1$.
 - each node in L and R participates in at most one edge in M
 - $|M| = k$: consider cut $(L \cup s, R \cup t)$ ■



Perfect Matching

Def. A matching $M \subseteq E$ is **perfect** if each node appears in exactly one edge in M .

Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.

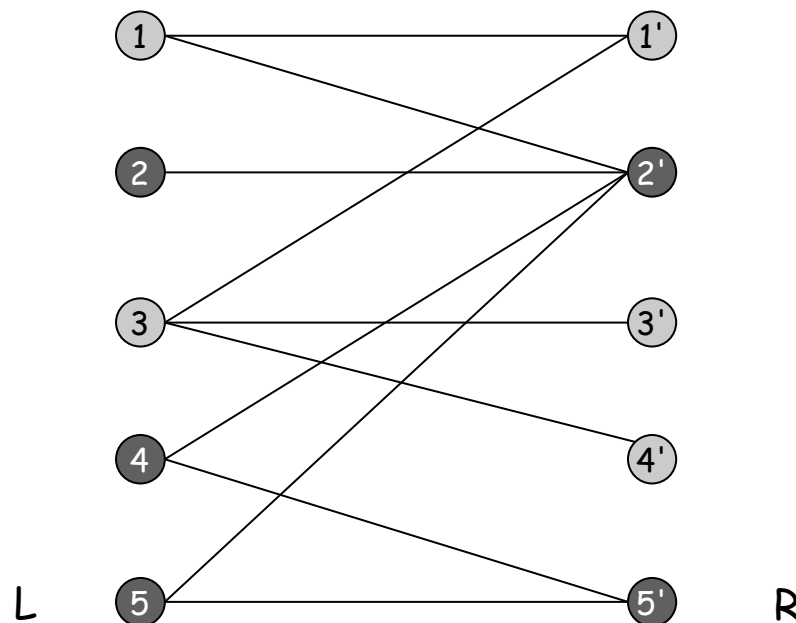
- Clearly we must have $|L| = |R|$.
- What other conditions are necessary?
- What conditions are sufficient?

Perfect Matching

Notation. Let S be a subset of nodes, and let $N(S)$ be the set of nodes adjacent to nodes in S .

Observation. If a bipartite graph $G = (L \cup R, E)$, has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. Each node in S has to be matched to a different node in $N(S)$.



No perfect matching:

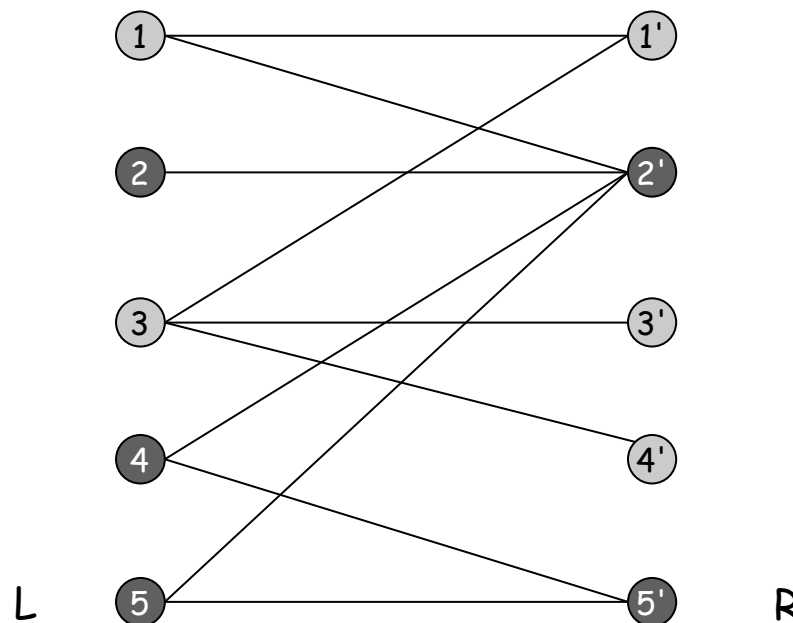
$S = \{ 2, 4, 5 \}$

$N(S) = \{ 2', 5' \}.$

Marriage Theorem

Marriage Theorem. [Frobenius 1917, Hall 1935] Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then, G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. \Rightarrow This was the previous observation.



No perfect matching:

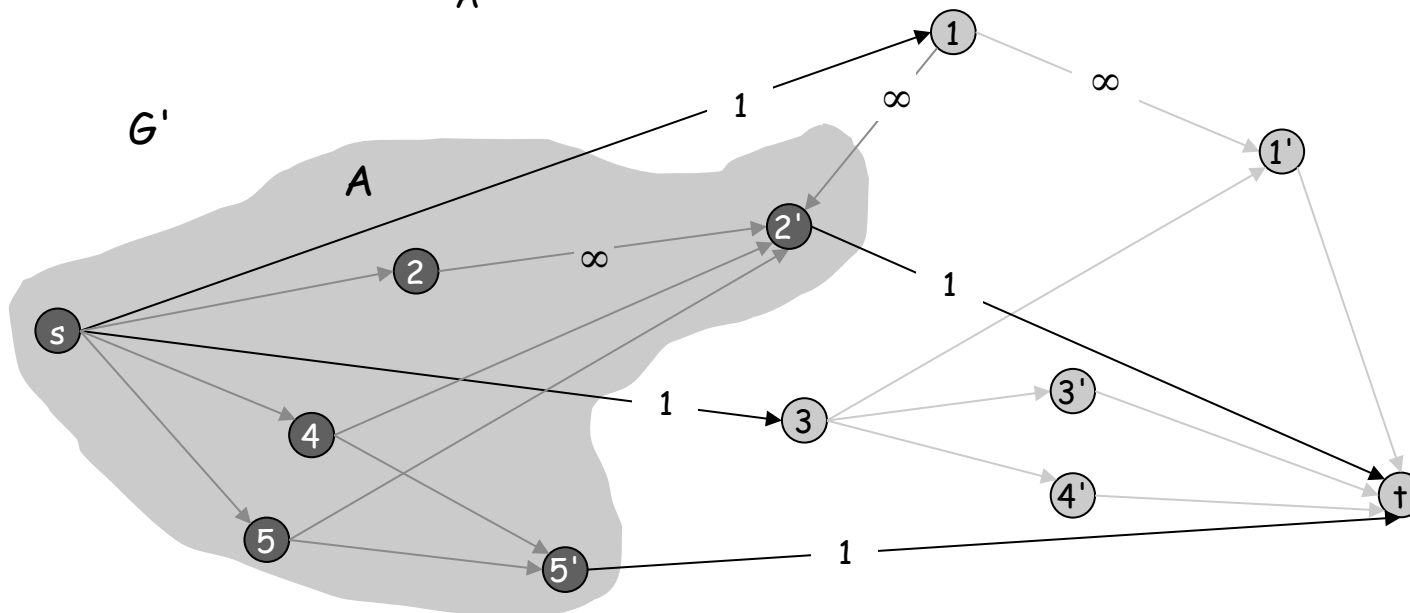
$S = \{ 2, 4, 5 \}$

$N(S) = \{ 2', 5' \}.$

Proof of Marriage Theorem

Pf. \Leftarrow Suppose G does not have a perfect matching.

- Formulate as a max flow problem and let (A, B) be min cut in G' .
- By max-flow min-cut, $\text{cap}(A, B) < |L|$.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$.
- $\text{cap}(A, B) = |L_B| + |R_A|$.
- Since min cut can't use ∞ edges: $N(L_A) \subseteq R_A$.
- $|N(L_A)| \leq |R_A| = \text{cap}(A, B) - |L_B| < |L| - |L_B| = |L_A|$.
- Choose $S = L_A$. ■



$$\begin{aligned} L_A &= \{2, 4, 5\} \\ L_B &= \{1, 3\} \\ R_A &= \{2', 5'\} \\ N(L_A) &= \{2', 5'\} \end{aligned}$$

Complexity of Finding a Bipartite Match

Complexity of Ford Fulkerson = $O(mC)$ where $C = \sum_{e \text{ out of } s} c(e)$

In graph G' , $C = n$

Therefore, in $O(mC) = O(mn)$ time we can find the bipartite matching.

Bipartite Matching: Running Time

Which max flow algorithm to use for bipartite matching?

- Generic augmenting path: $O(m \text{ val}(f^*)) = O(mn)$.
- Capacity scaling: $O(m^2 \log C) = O(m^2)$.
- Shortest augmenting path: $O(m n^{1/2})$.

Non-bipartite matching.

- Structure of non-bipartite graphs is more complicated, but well-understood. [Tutte-Berge, Edmonds-Galai]
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m n^{1/2})$. [Micali-Vazirani 1980]