CS240 Algorithm Design and Analysis

Lecture 7

Network Flow (Cont.)

Quan Li
Fall 2023
2023.10.20

Last Time – What you need to know

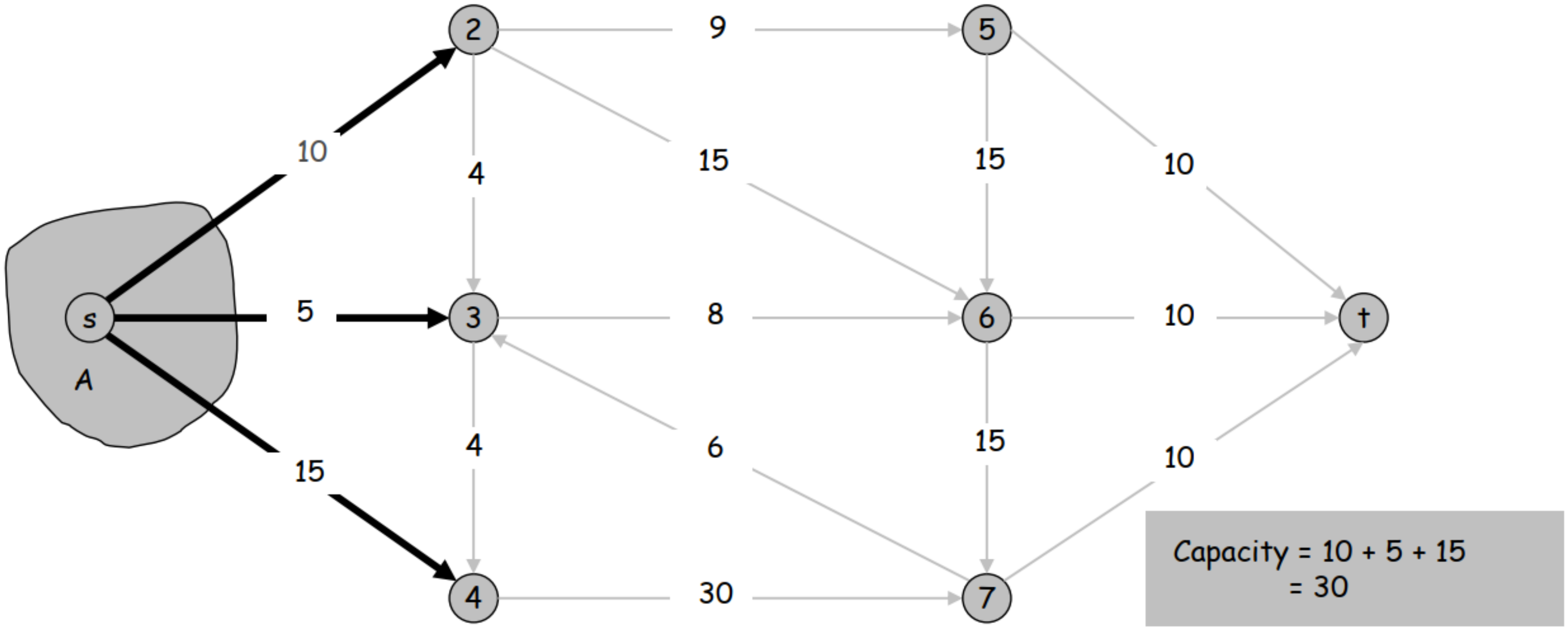
- **Dynamic Programming**
  - Shortest path
    - Dijkstra
    - Bellman-Ford
- **Max-flow**
  - Greedy
  - Ford-Fulkerson Algorithm based on Residual Graph

Max-flow and Min-Cut

Cuts

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

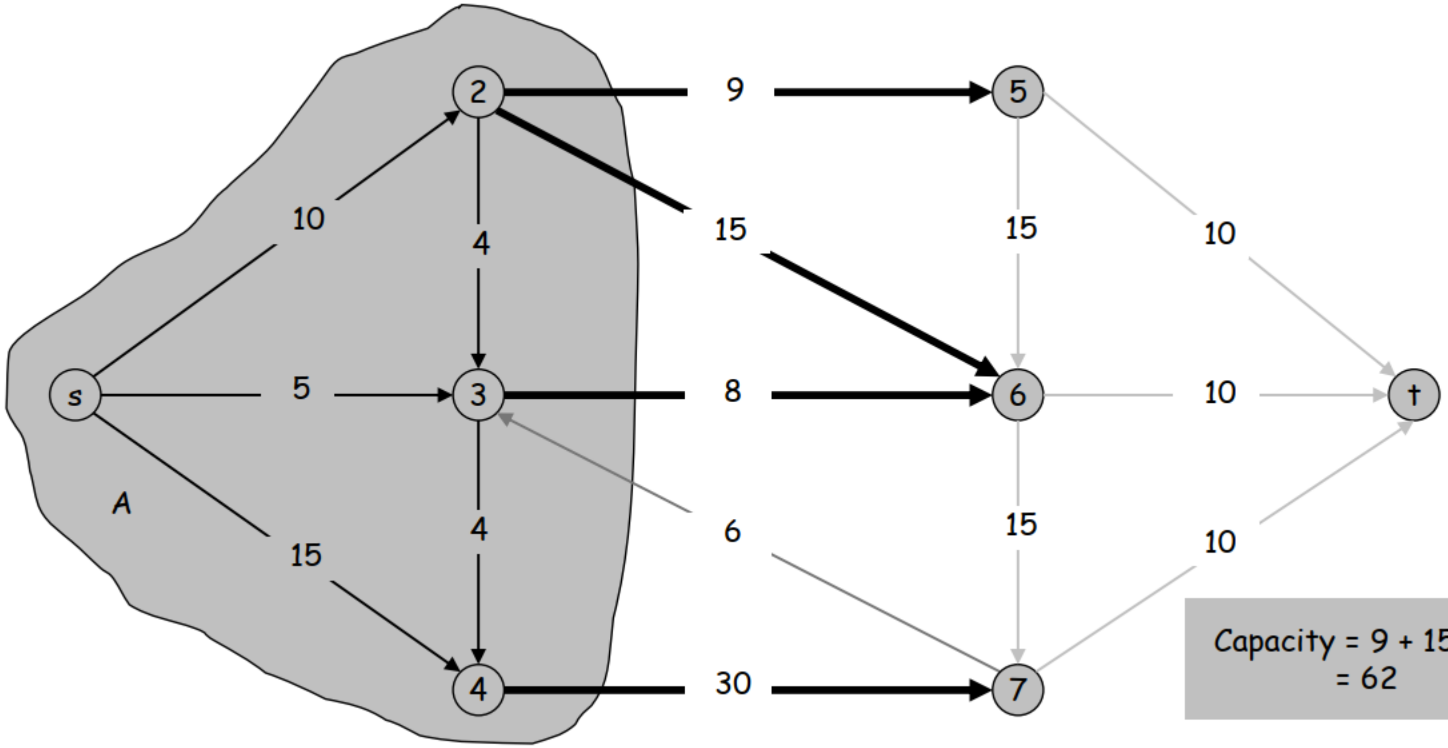
- **Def.** The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Cuts

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

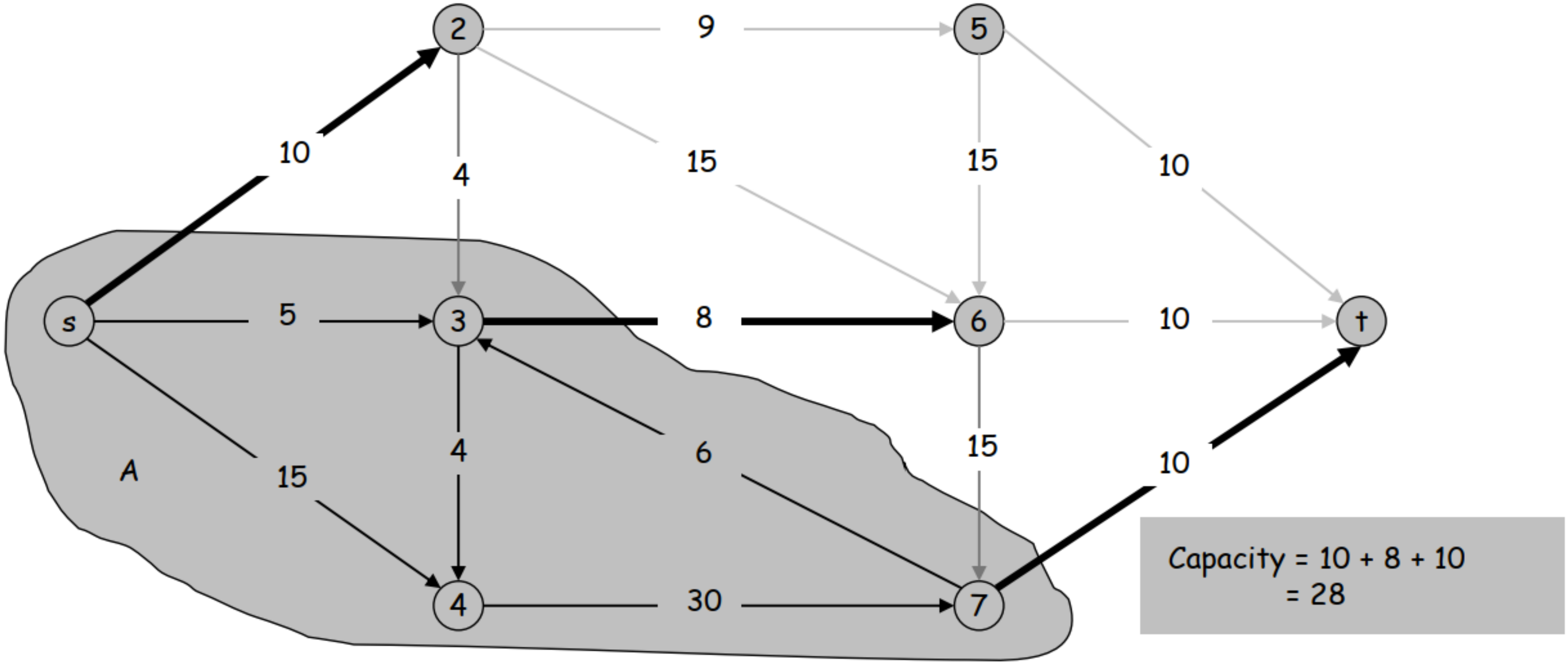
- **Def.** The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

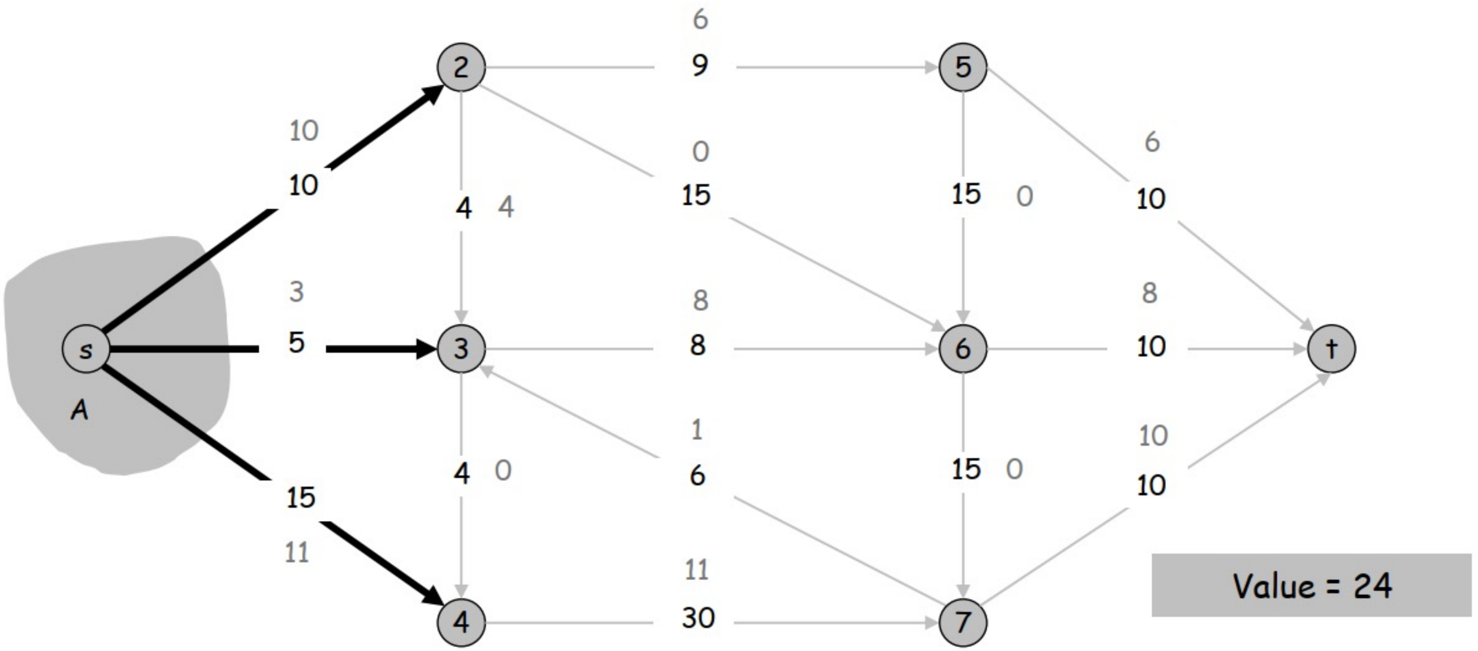
- **Min s-t cut problem.** Find an s-t cut of minimum capacity



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

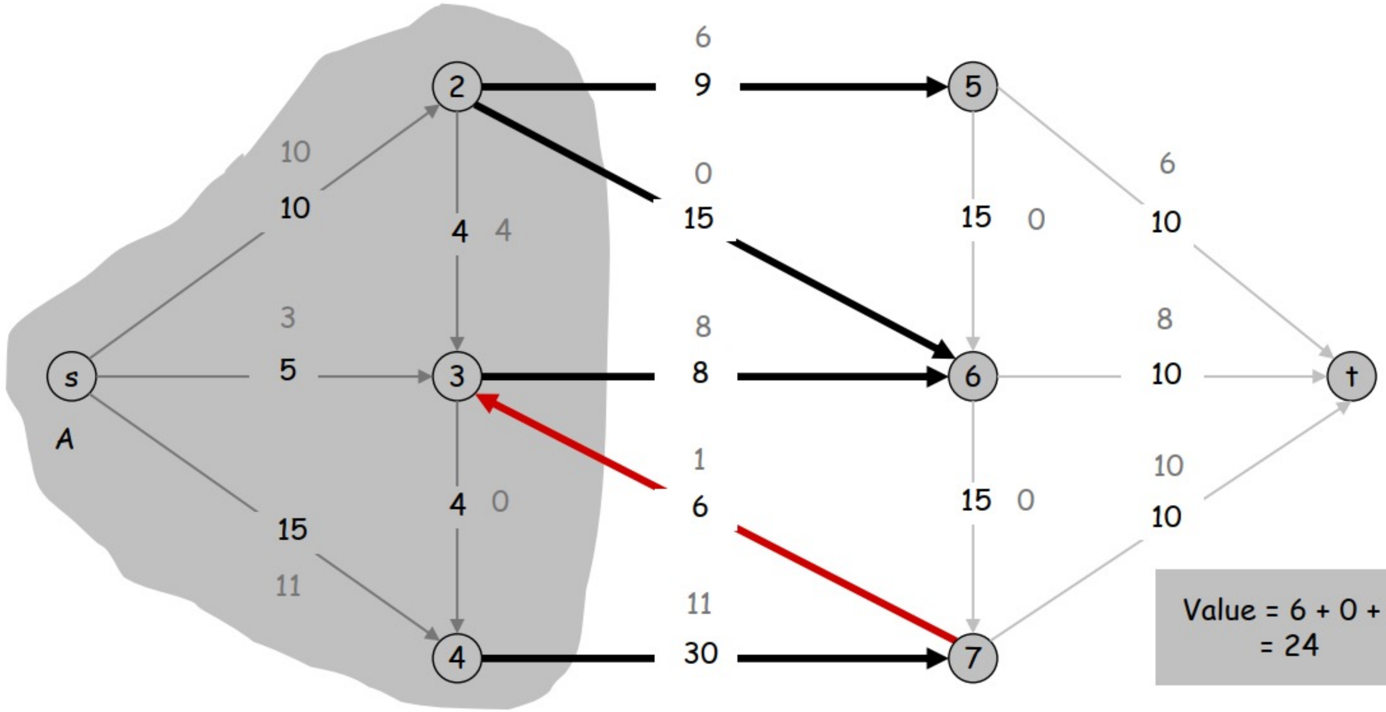
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



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

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

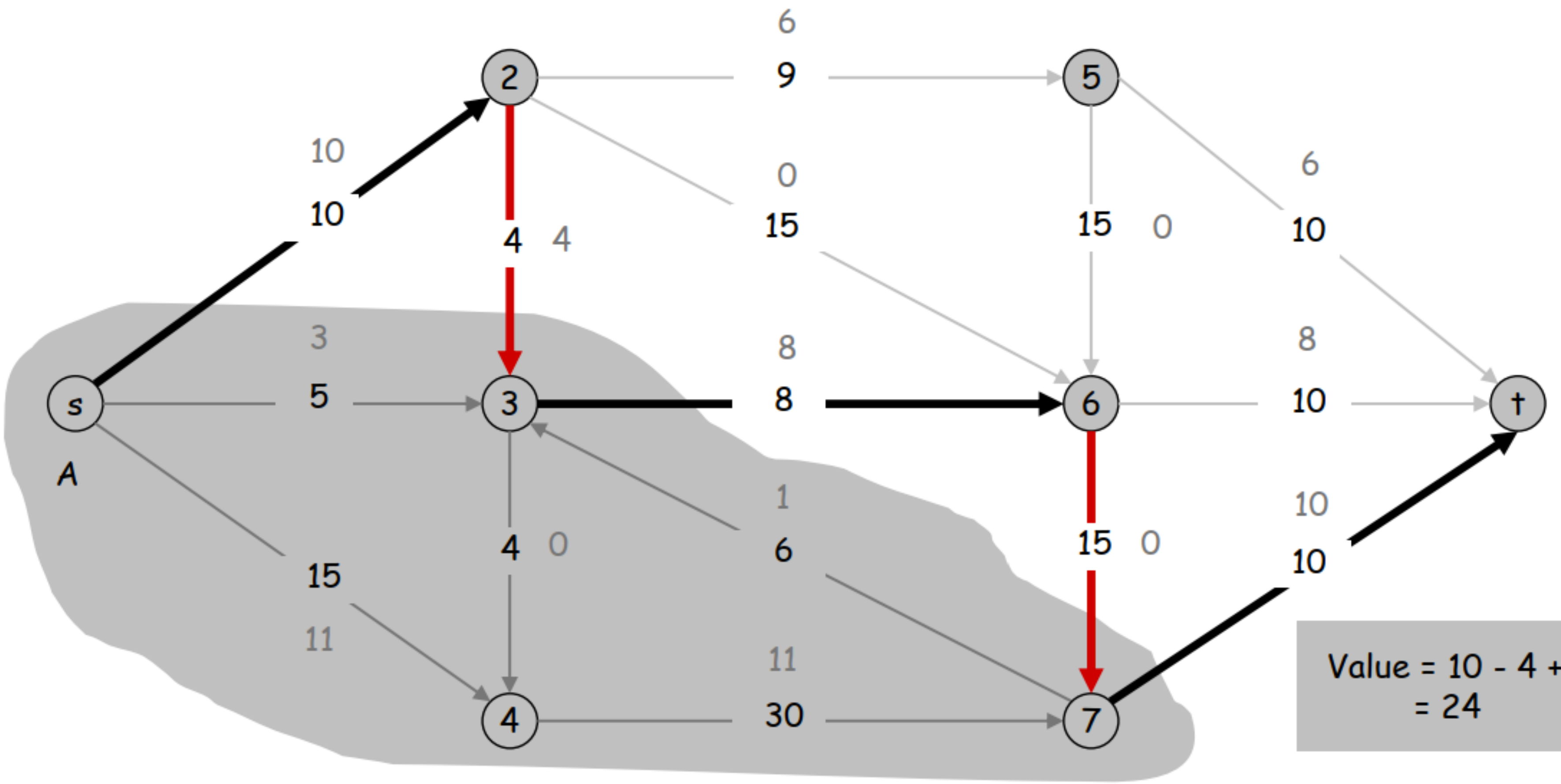


Value = 6 + 0 + 8 - 1 + 11
= 24

Flows and Cuts

上海科技大学 ShanghaiTech University

- **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



Value = 10 - 4 + 8 - 0 + 10
= 24

立志成才报国裕民

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 $\longrightarrow$

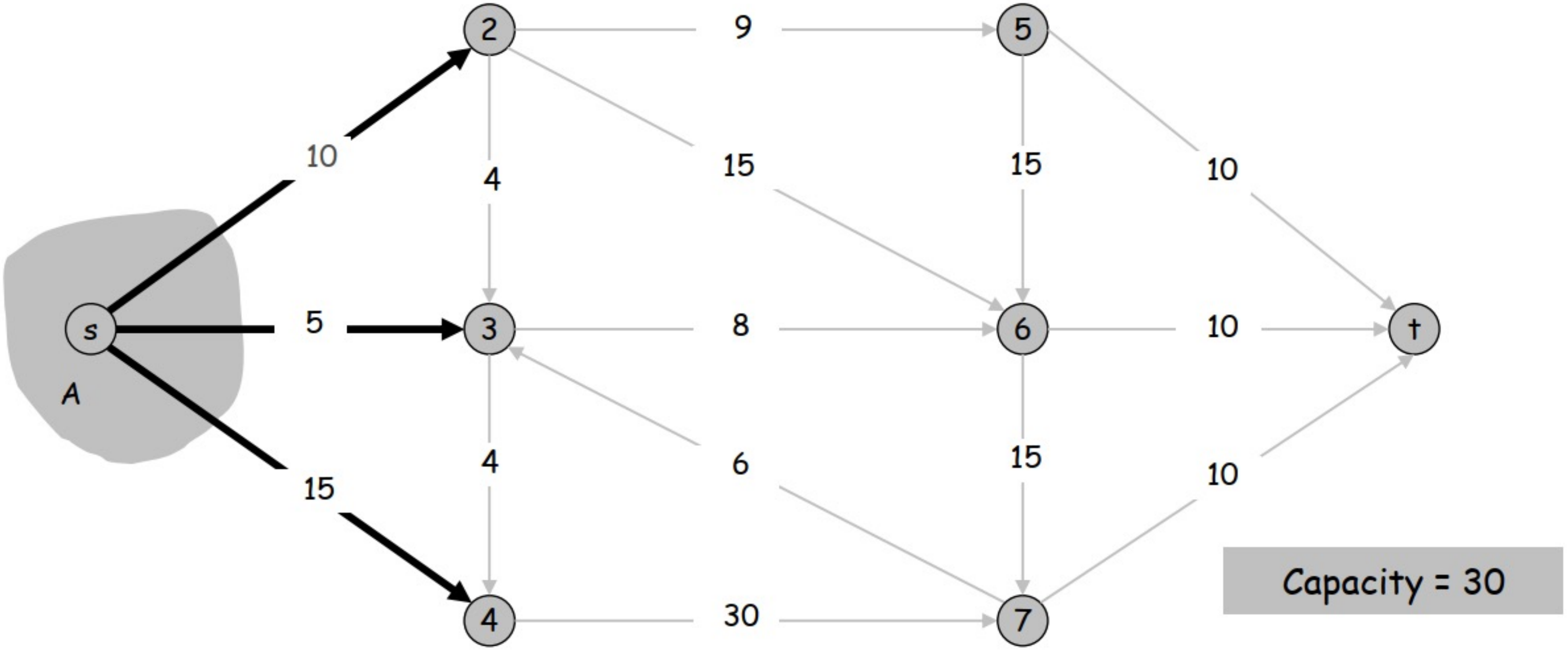
$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \right)$$

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

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



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.

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

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

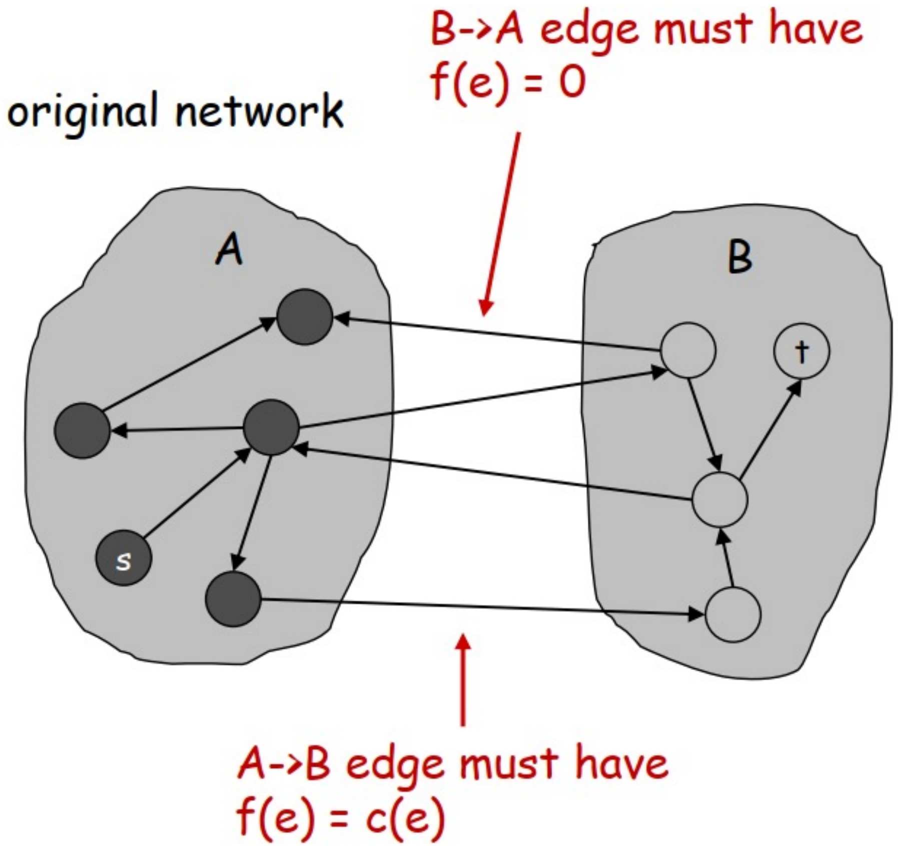
- (i) → (ii) This was the corollary to weak duality lemma

- (ii) → (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) → (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 ∈ A
  - By definition of f, t ∉ A

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



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

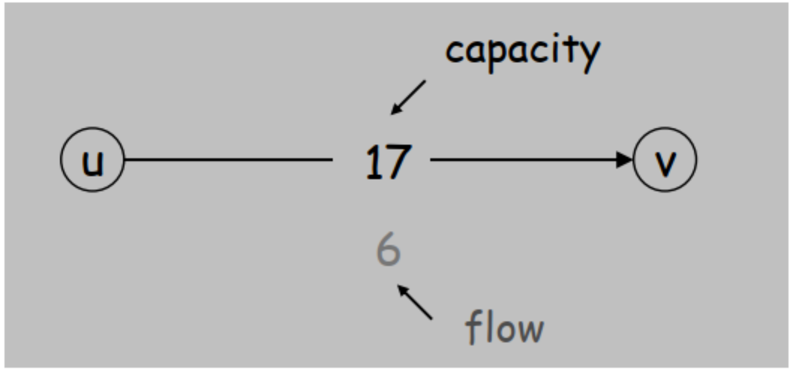
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   G_f ← residual graph

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

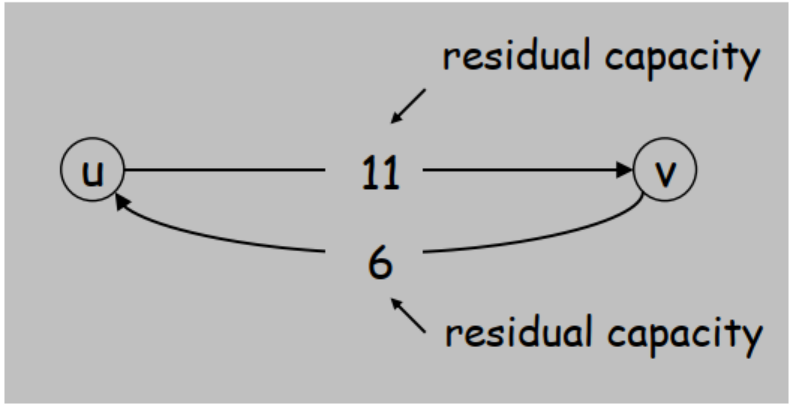
Residual Graph

- **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
  - $E_f = \{e: f(e) < c(e)\} \cup \{e^R: f(e) > 0\}$





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 $v(f^*) <= nC$ iterations.

  Pf. Each augmentation increase value by at least 1

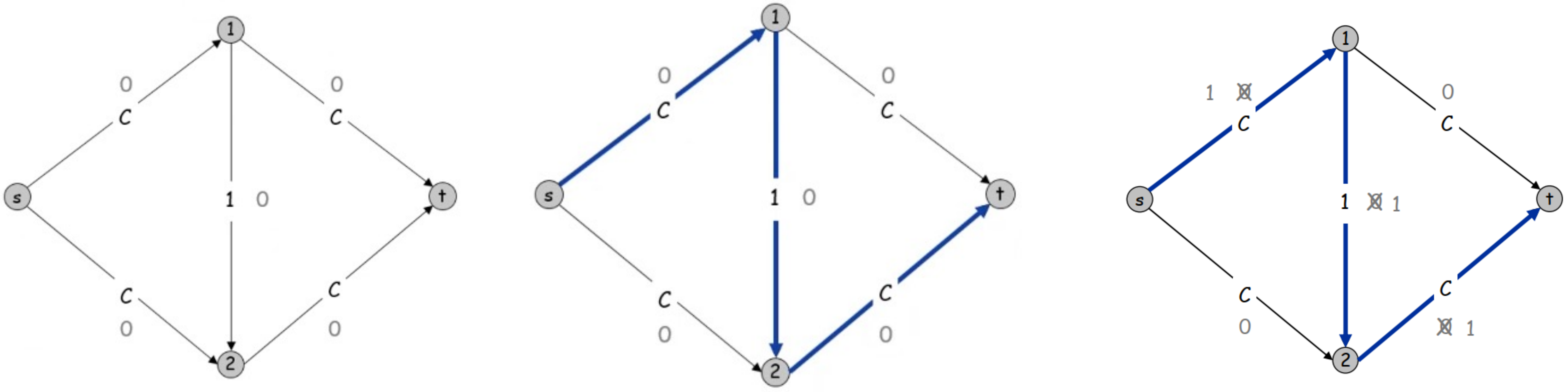
- **Corollary.** Running time of Ford-Fulkerson is O(mnC) ← Polynomial?

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

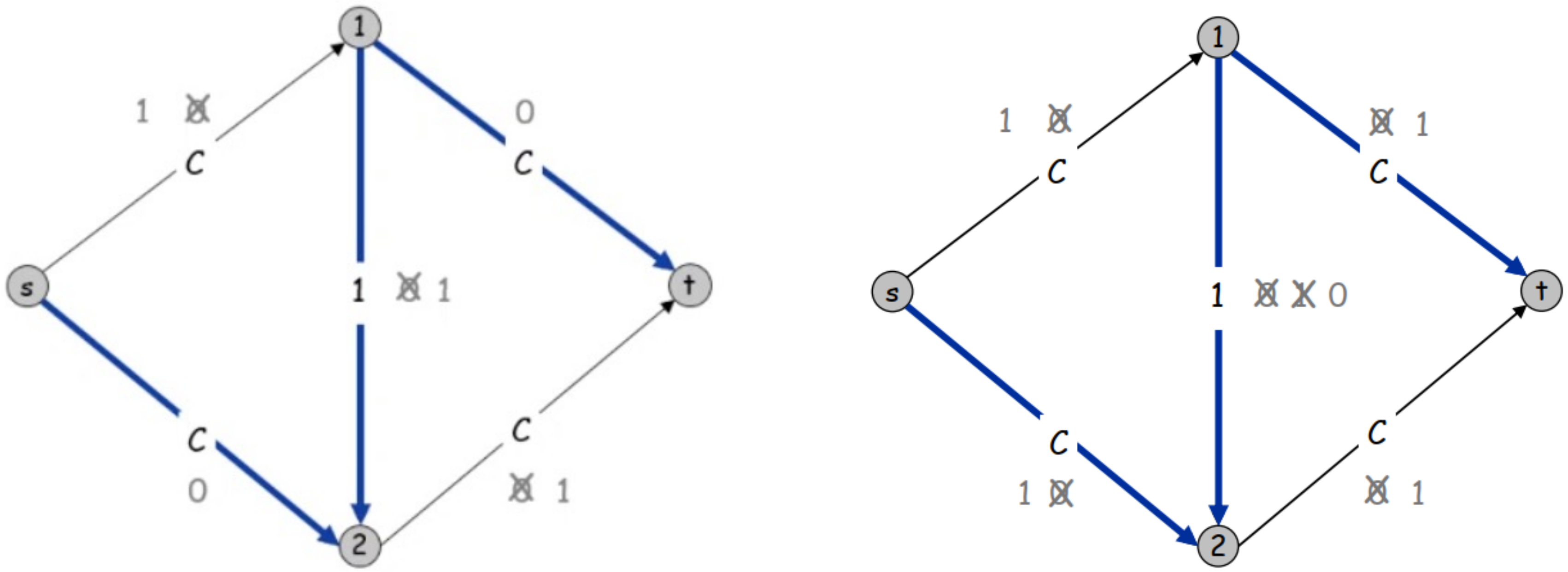


Ford-Fulkerson: Exponential Number of Augmentations

- **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 >= C iterations

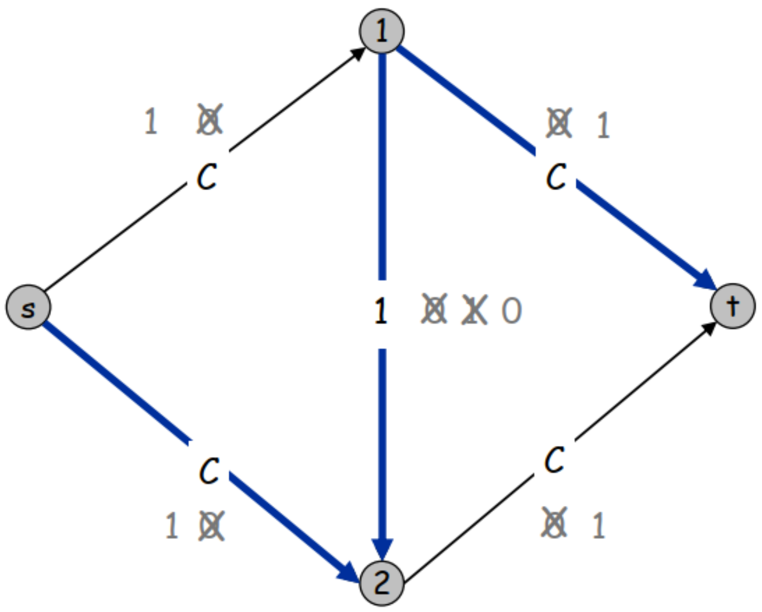


Ford-Fulkerson: Exponential Number of Augmentations

- **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 >= C iterations



Each augmenting path sends only 1 unit of flow (#augmenting paths = 2C)

Choosing Good Augmenting Paths

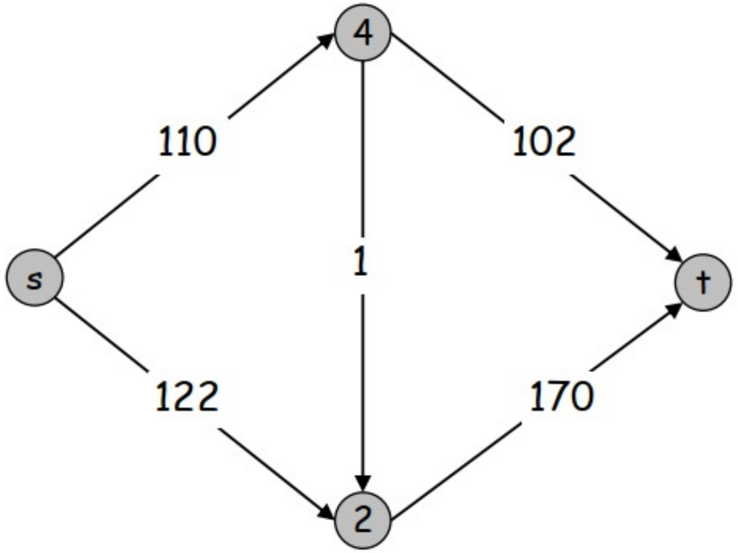
- **Use care when selecting augmenting paths**
  - Some choices led to exponential algorithms
  - Clever choices lead to polynomial algorithms
  - (If capacities are irrational, algorithm not guaranteed to terminate!)

- **Goal: choose augmenting paths so that:**
  - Can find augmenting paths efficiently
  - Few iterations

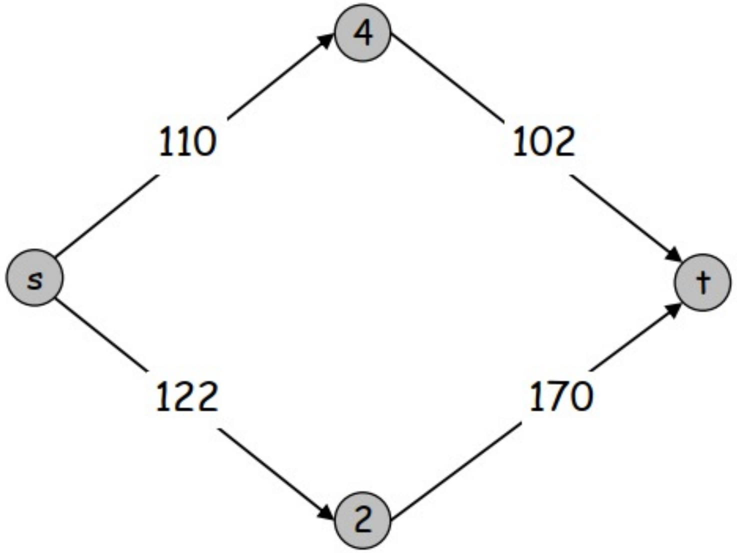
- **Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]**
  - Max bottleneck capacity
  - Fewest number of edges
  - **Sufficiently large bottleneck capacity**

Capacity Scaling

- **Intuition.** Choosing path with high bottleneck capacity
  - Maintain scaling parameter Δ
  - Let the **Δ-residual graph $G_f(\Delta)$** 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 $C/2 < \Delta <= C$. 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

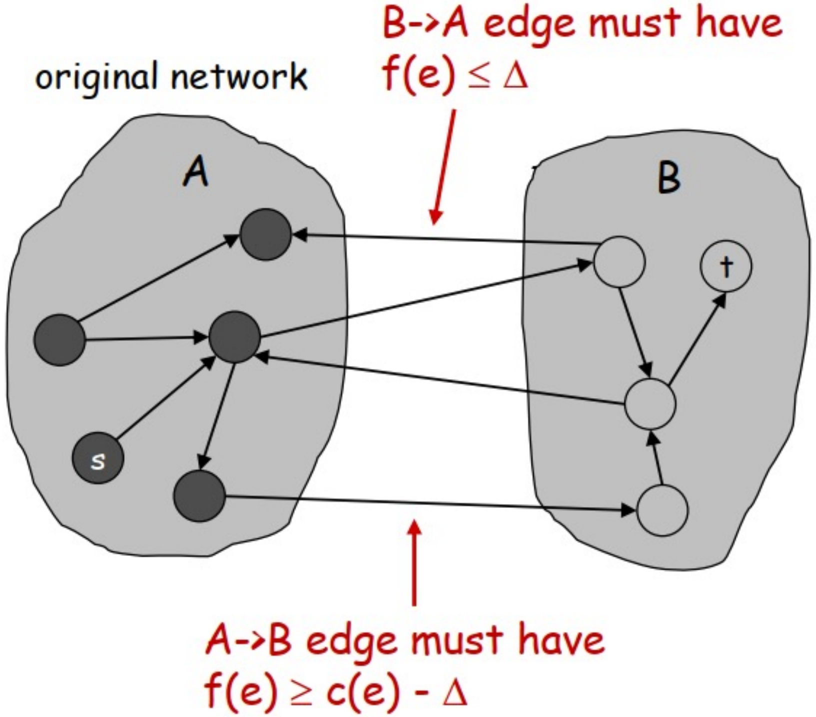
- **Lemma 3.** There are at most 2m augmentation per scaling phase.
  - Let f be the flow at then end of the previous scaling phase
  - L2 → $V(f^*) <= v(f) + m(2\Delta)$
  - Each augmentation in a Δ-phase increases v(f) by at least Δ

- **Theorem.** The scaling max-flow algorithm finds a max flow in O(mlogC) augmentations. It can be implemented to run in $O(m^2 \log C)$ time

Capacity Scaling: Running Time

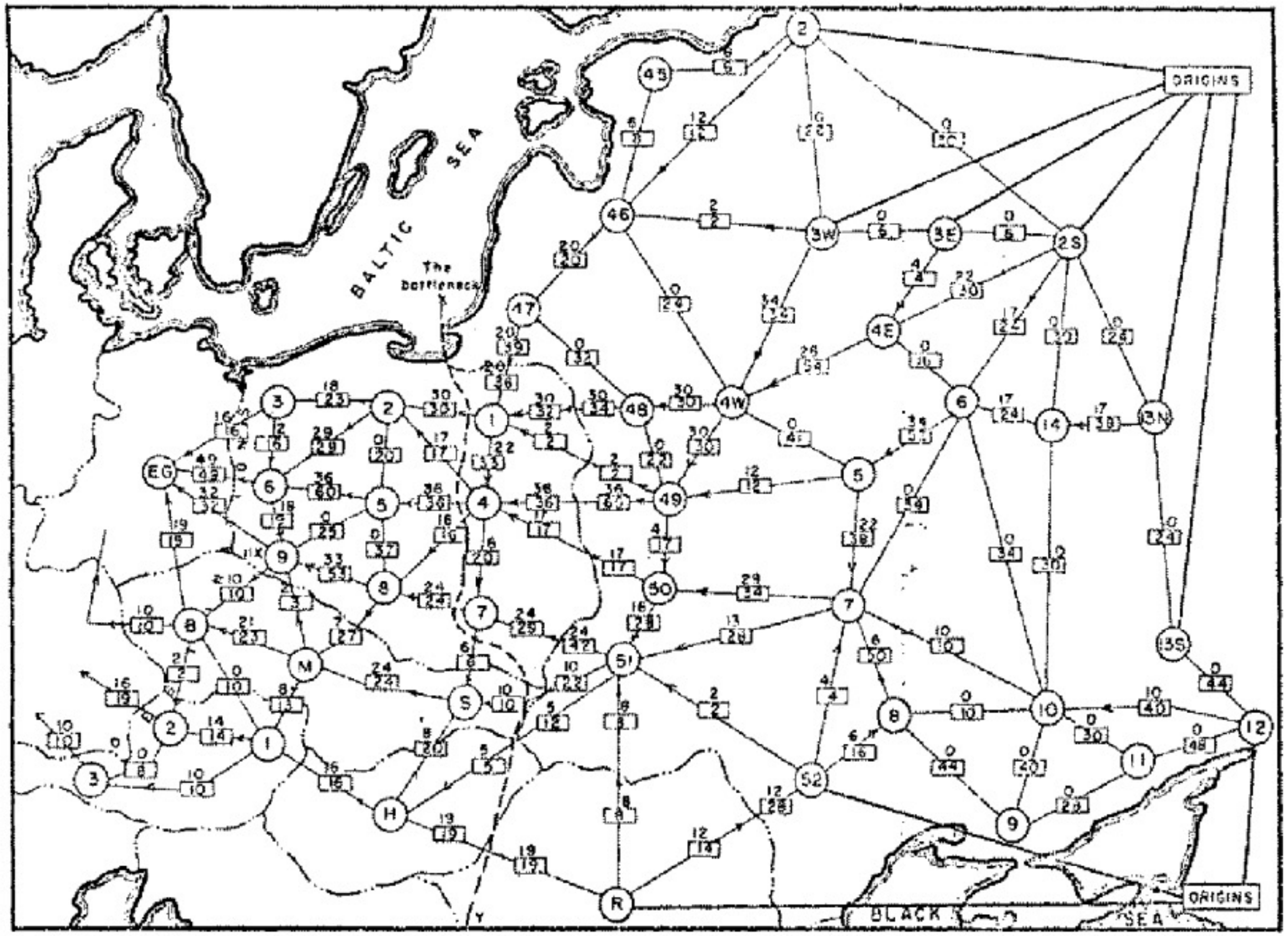
- **Lemma 2.** Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow is at most $v(f) + m\Delta$
- **Pf.** (almost identical to proof of max-flow min-cut theorem)
  - We show that at the end of a Δ-phase, there exists a cut (A, B) such that cap(A, B) <= $v(f) + m\Delta$
  - 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, $t \notin A$

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



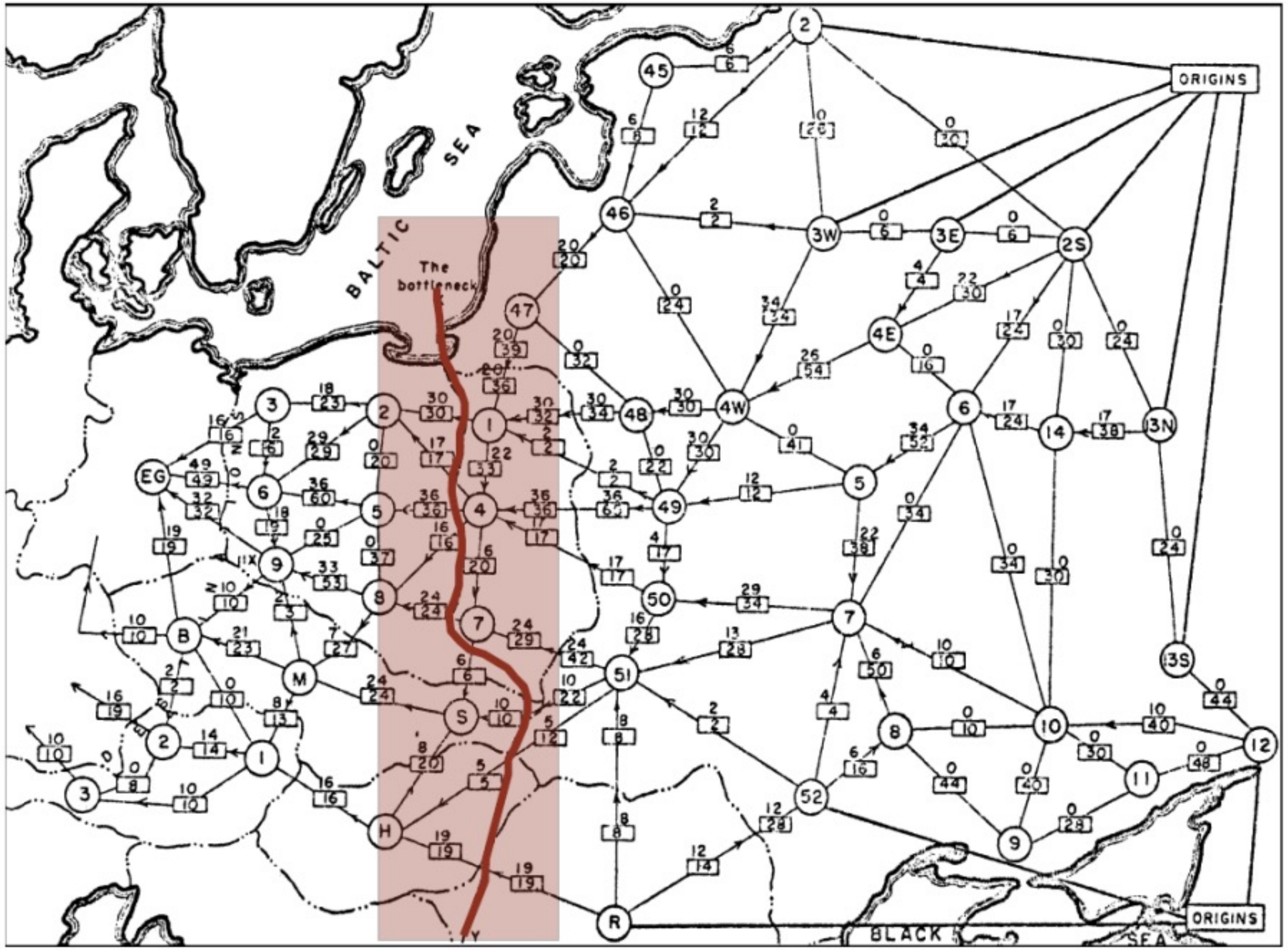
Applications of max-flow

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

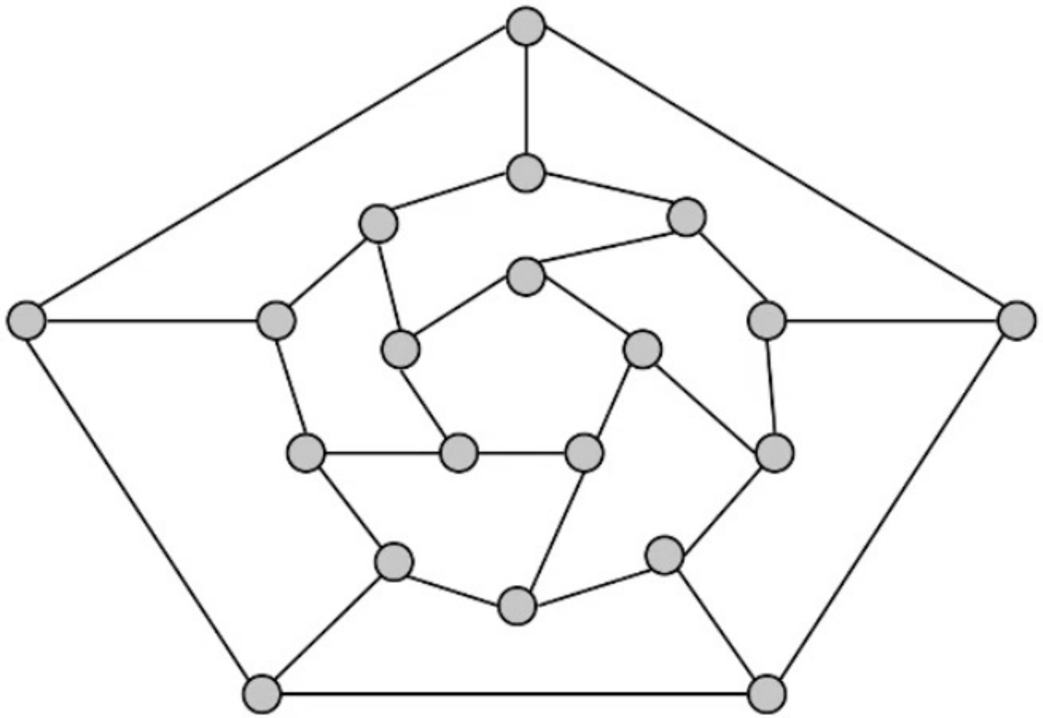


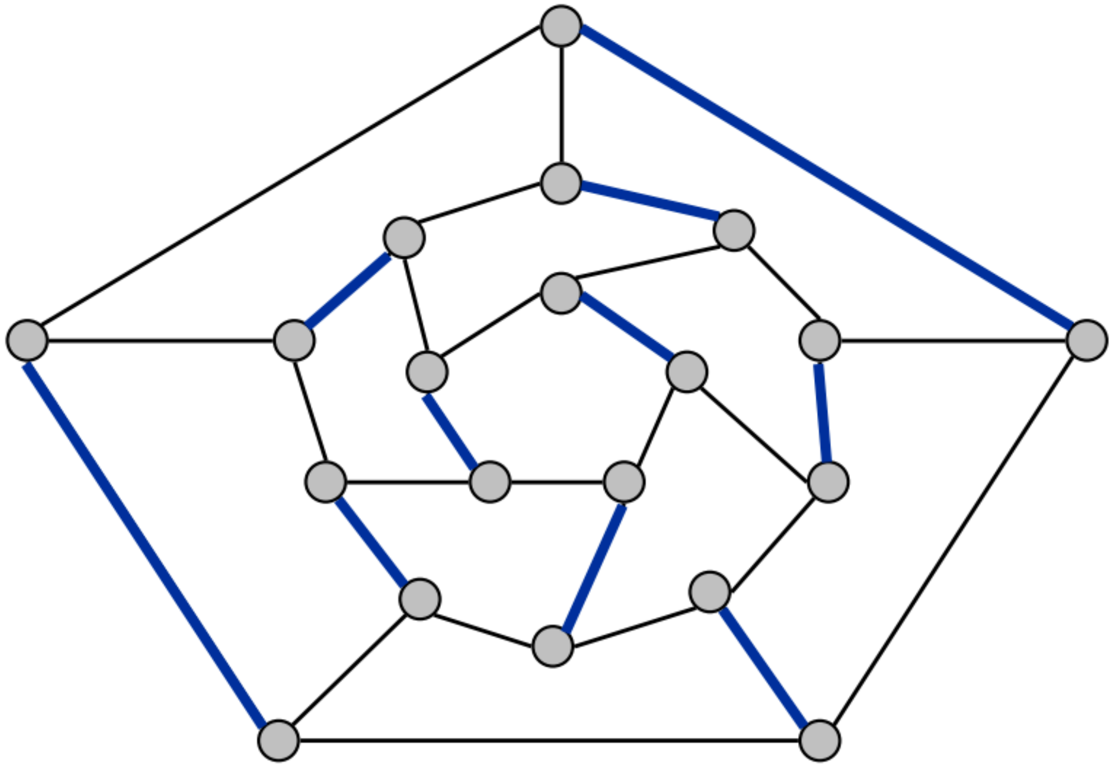
Reference: On the history of the transportation and maximum flow problems. Alexander Schrijver in Math Programming, 91: 3, 2002

Bipartite Matching

Matching

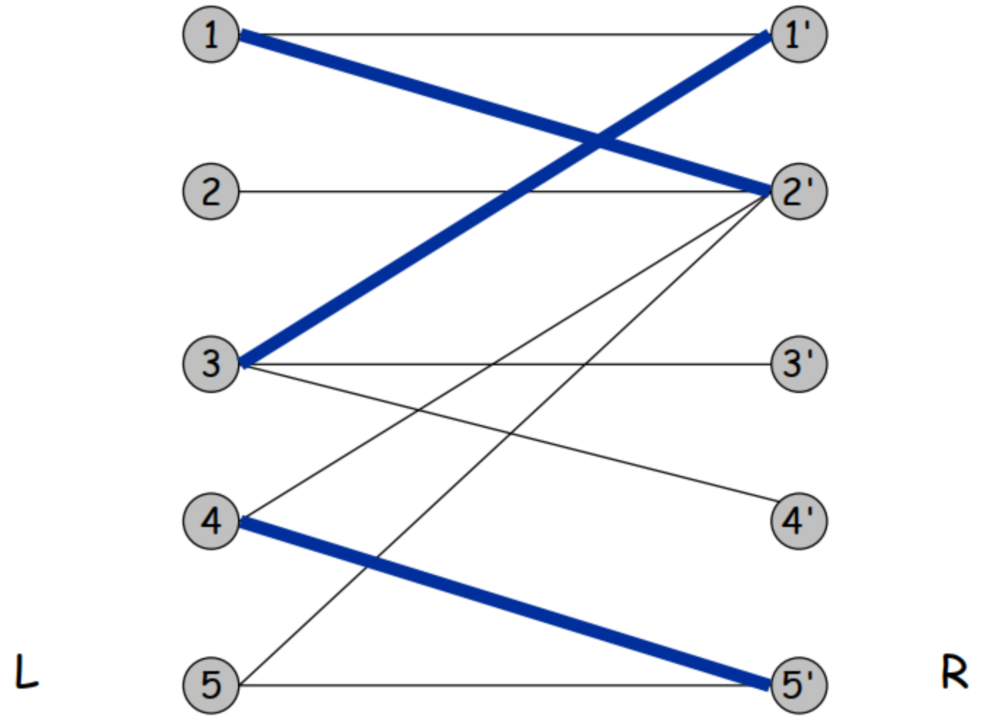
- **Matching**
  - Input: undirected graph G = (V, E)
  - M ⊆ E is a **matching** if each node appears in at most one edge in M
  - Max matching: find a max cardinality matching





Bipartite Matching

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

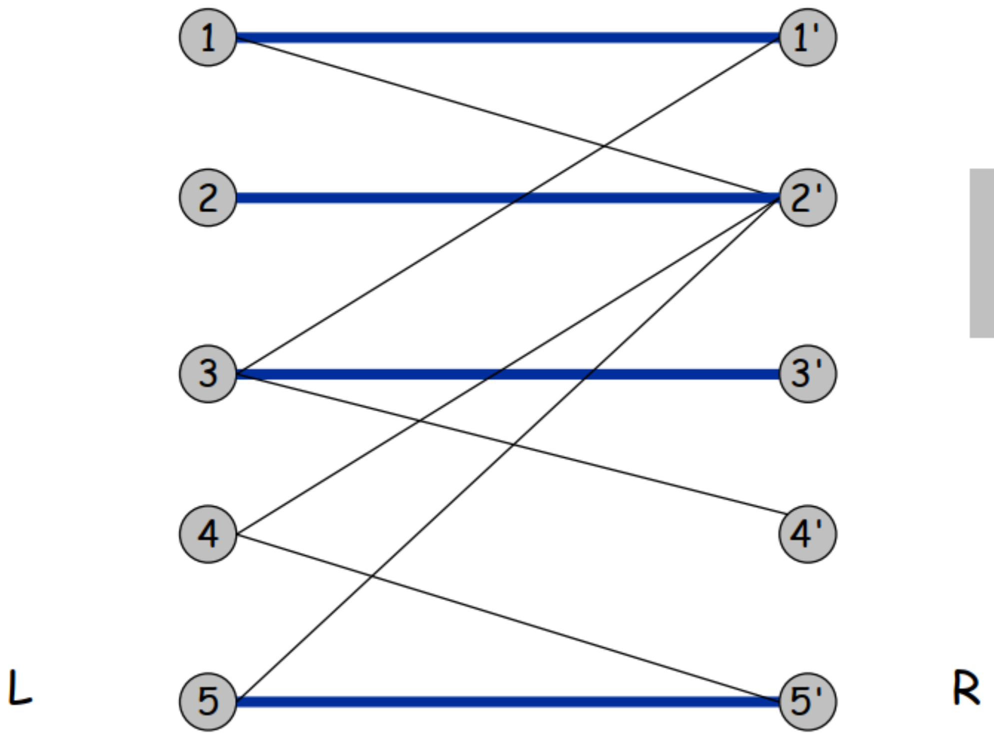


matching

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

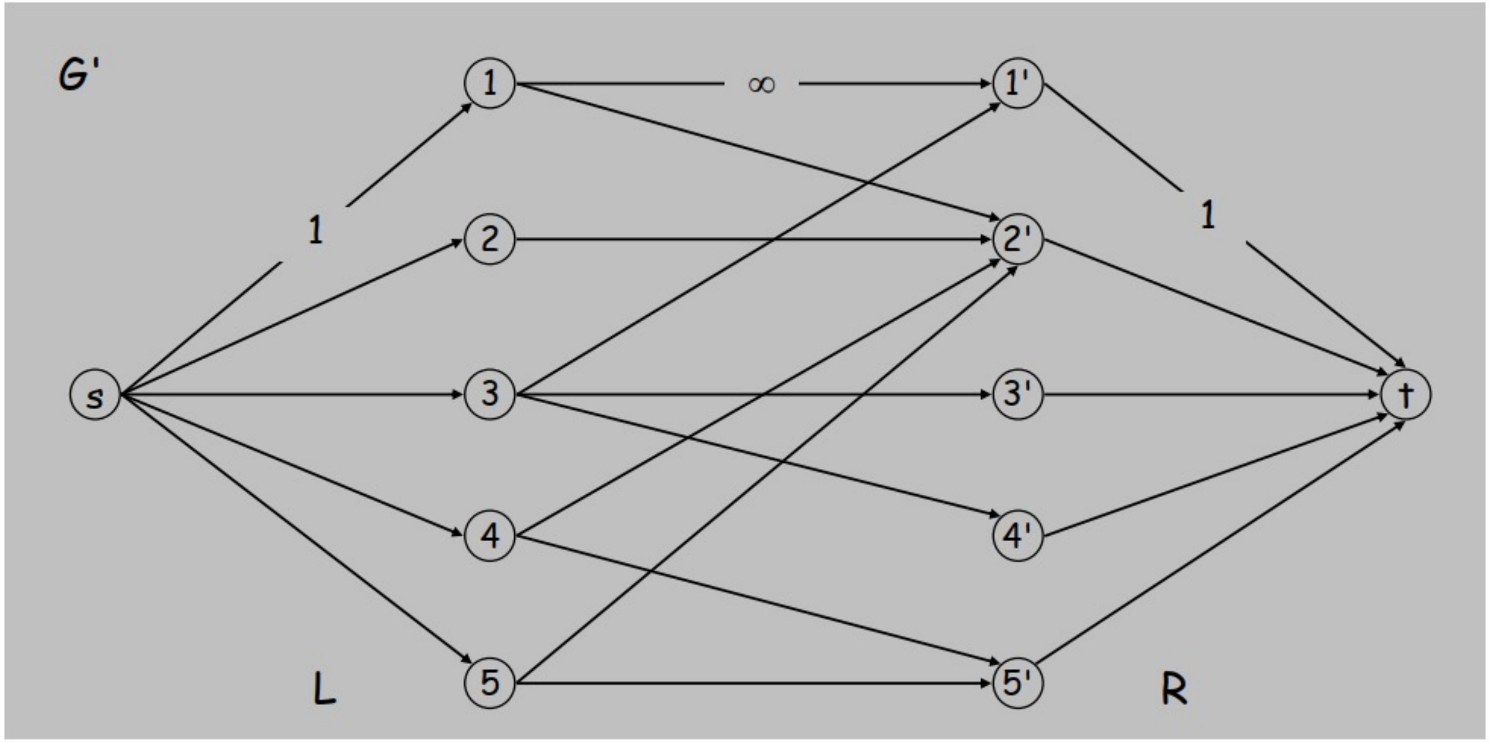


max matching

1-1', 2-2', 3-3' 4-4'

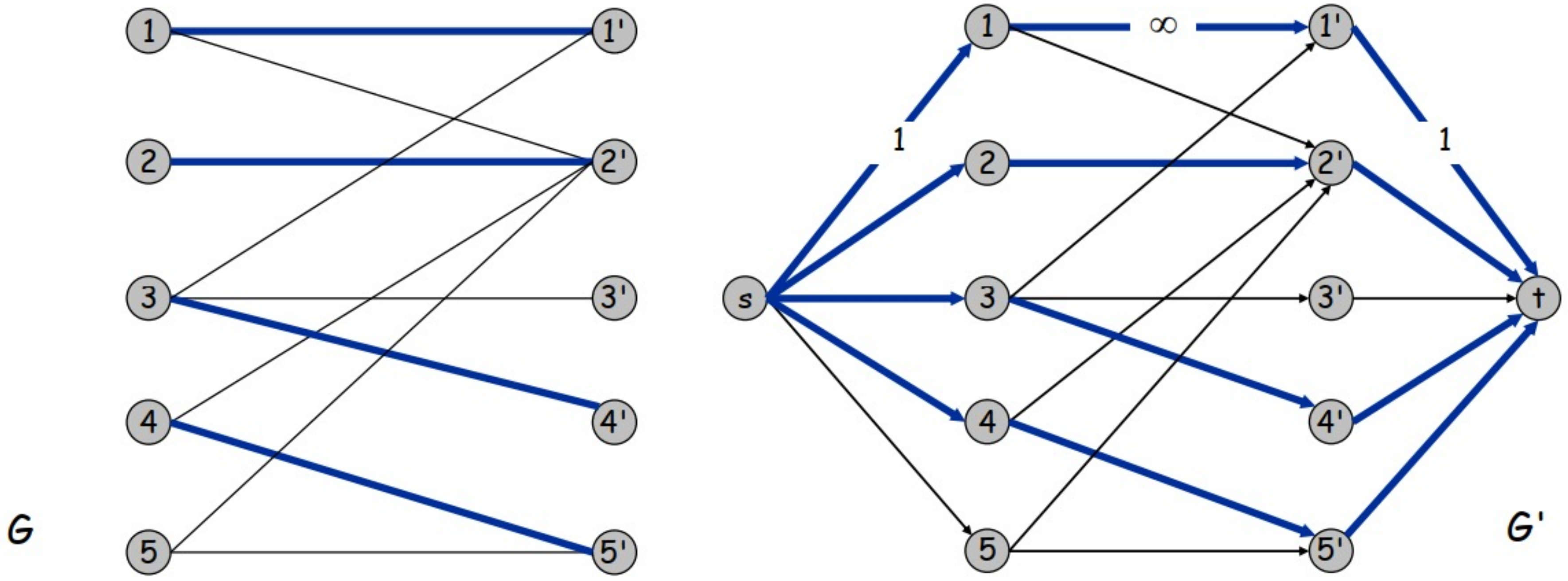
Bipartite Matching

- **Max flow formulation**
  - Create digraph G′ = (L∪R∪{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



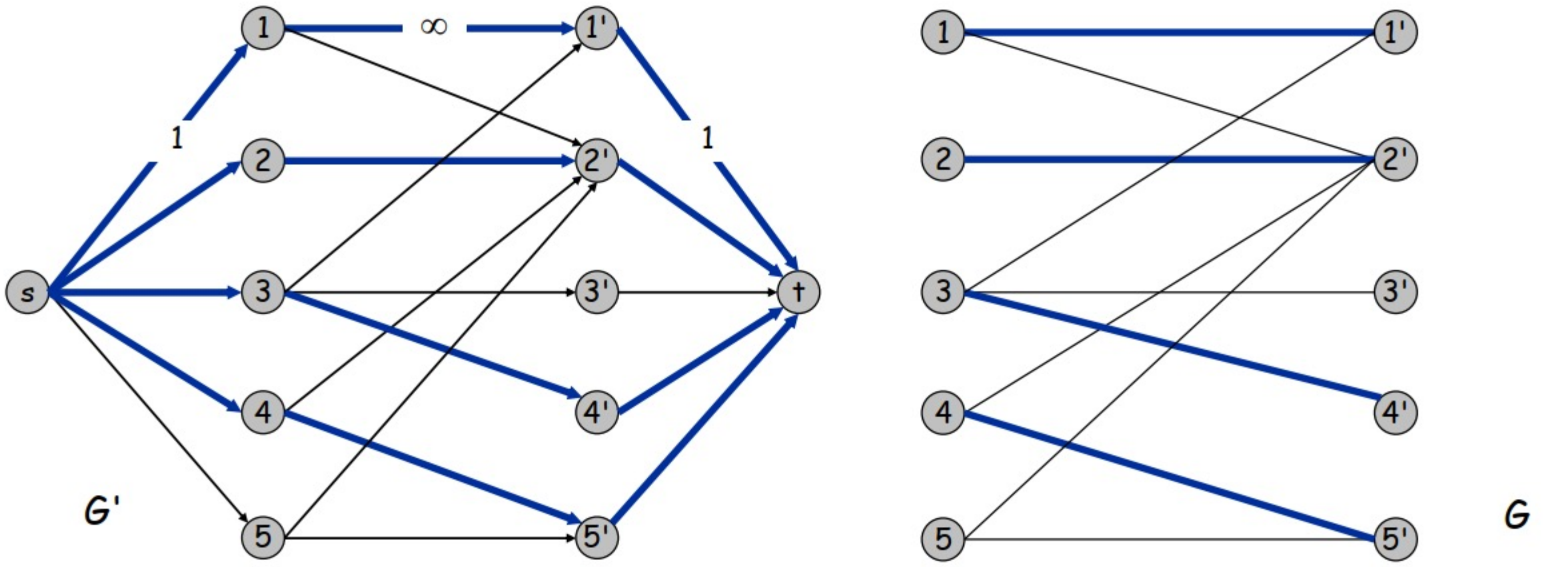
Bipartite Matching: Proof of Correctness

- **Theorem.** Max cardinality matching in G = value of max flow in G′
- **Pf.** <=
  - 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. >=**
  - Let f be a max flow in G′ of value k
  - Integrality theorem → 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∪s, R∪t)



Perfect Matching

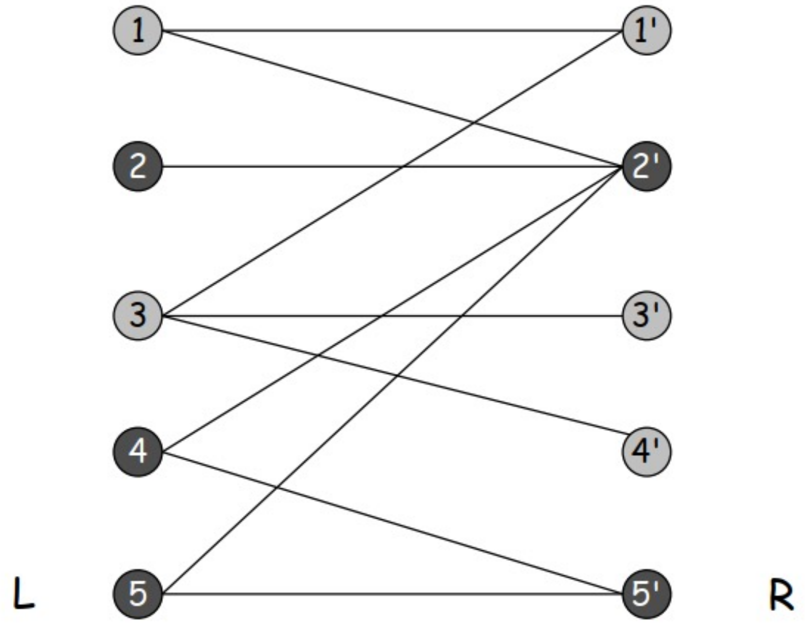
- **Def.** A matching $M \subseteq E$ is **perfect** if each node appears in exactly one edge in M

- **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∪R, E), has a perfect matching, then |N(S)| >= |S| for all subsets S ⊆ L
- **Pf.** Each node in S has to be matched to a different node in N(S)

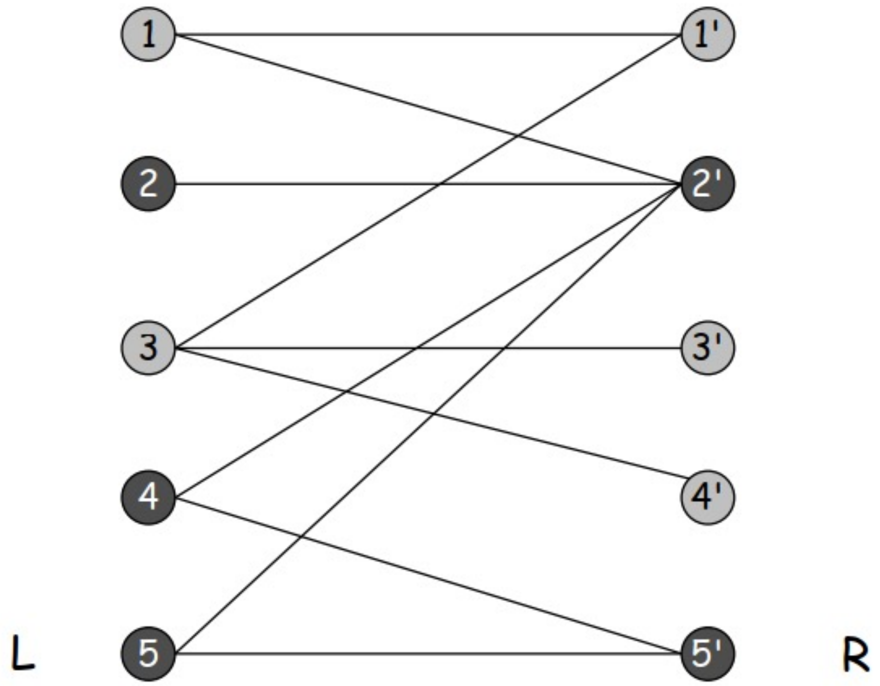


No perfect matching:
S = { 2, 4, 5 }
N(S) = { 2', 5' }.

Marriage Theorem

- **Marriage Theorem.** [Frobenius 1917, Hall 1935] Let G = (L∪R, E) be a bipartite graph with |L| = |R|. Then, G has a perfect matching iff |N(S)| >= |S| for all subsets S ⊆ L

- **Pf.** → This was the previous observation



No perfect matching:

S = { 2, 4, 5 }

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

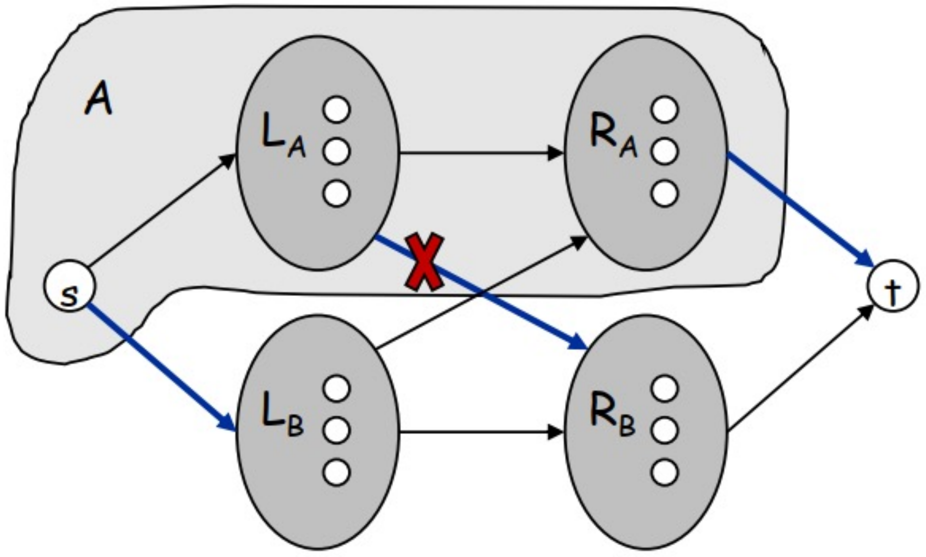
Proof of Marriage Theorem

- **Marriage Theorem.** G has a perfect matching iff $|N(S)| >= |S|$ for all subsets $S \subseteq L$
- **Pf.** ← Suppose G does not have a perfect matching
  - Formulate as a max flow problem and let (A, B) be min cut in G′
  - Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$, $R_B = R \cap B$
  - $Cap(A, B) = v(f^*) = |M| < |L|$ ("<": because no perfect matching)
  - Since min cut can't use ∞ edges, no edge between L_A and R_B
    - $Cap(A, B) = |L_B| + |R_A|$
    - $N(L_A) \subseteq R_A$
  - $|N(L_A)| <= |R_A|$

    $= cap(A, B) - |L_B|$

    $< |L| - |L_B|$

    $= |L_A|$
  - This contradicts the condition



Next Time:
Network Flow (Cont.)