



CS240 Algorithm Design and Analysis

Lecture 7

Network Flow (Cont.)

Quan Li
Fall 2022
2022.09.27



Last Time – What you need to know



- **Dynamic Programming**
 - Shortest path
 - Dijkstra
 - Bellman-Ford
- **Max-flow**
 - Greedy
 - Ford-Fulkerson Algorithm based on Residual Graph
- **Min-cut**
 - Max-Flow Min-Cut Theorem
 - Proof





Ford-Fulkerson Algorithm

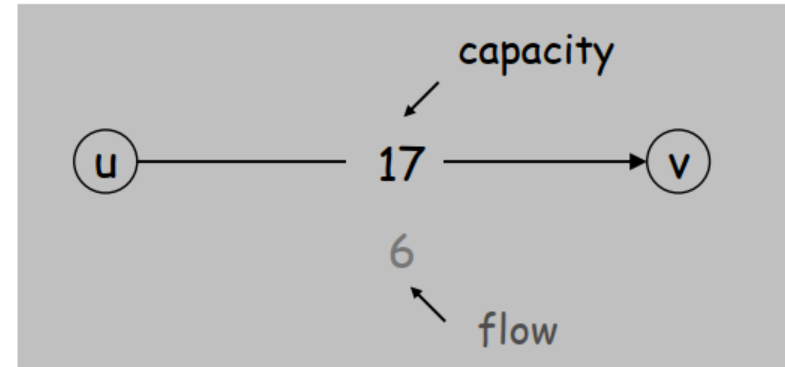
- **Ford-Fulkerson Algorithm**

- 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  $e \in E$   $f(e) \leftarrow 0$   
     $G_f \leftarrow$  residual graph  
  
    while (there exists augmenting path  $P$ ) {  
         $f \leftarrow$  Augment( $f, c, P$ )  
        update  $G_f$   
    }  
    return  $f$   
}
```

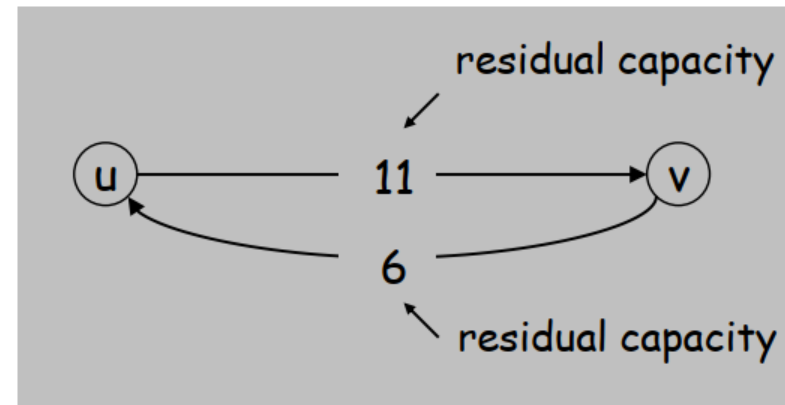
Residual Graph

- **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 $v(f^*) \leq nC$ iterations.
 - Pf.** Each augmentation increase value by at least 1
- **Corollary.** Running time of Ford-Fulkerson is $O(mnC)$ ← Polynomial?

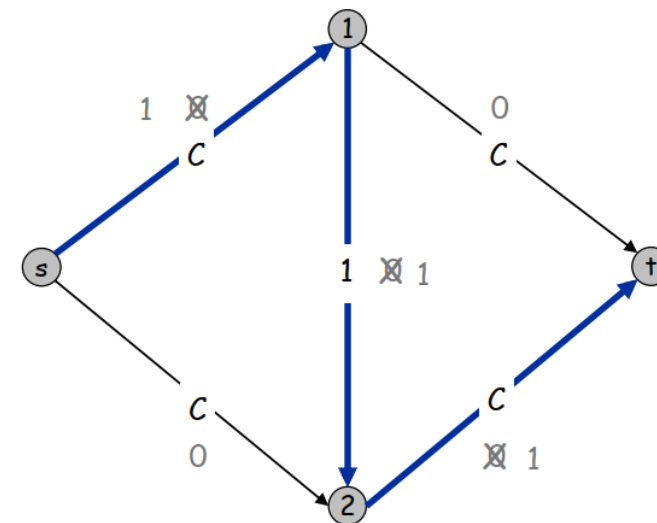
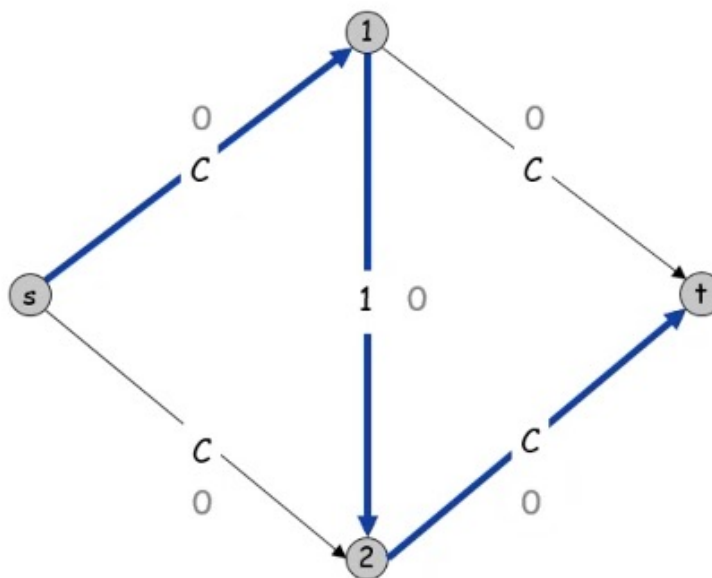
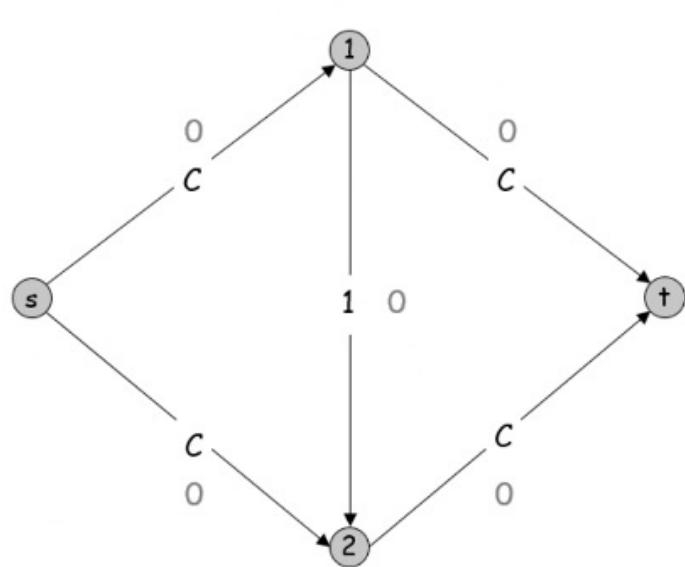


Ford-Fulkerson: Exponential Number of Augmentations

- 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 $\geq C$ iterations



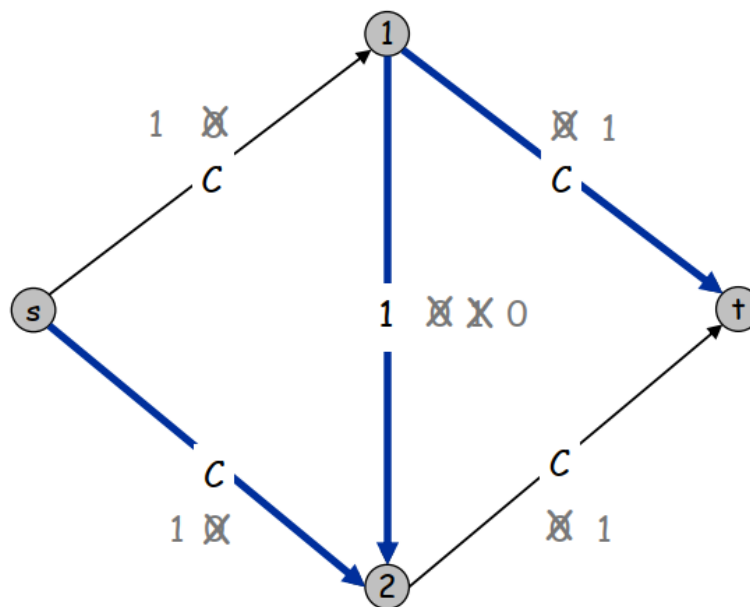
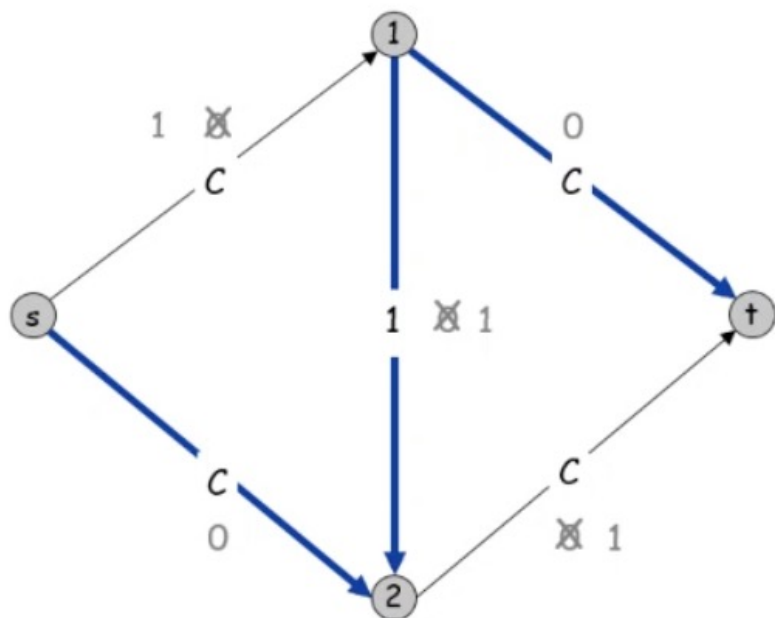


Ford-Fulkerson: Exponential Number of Augmentations

- 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 $\geq C$ iterations



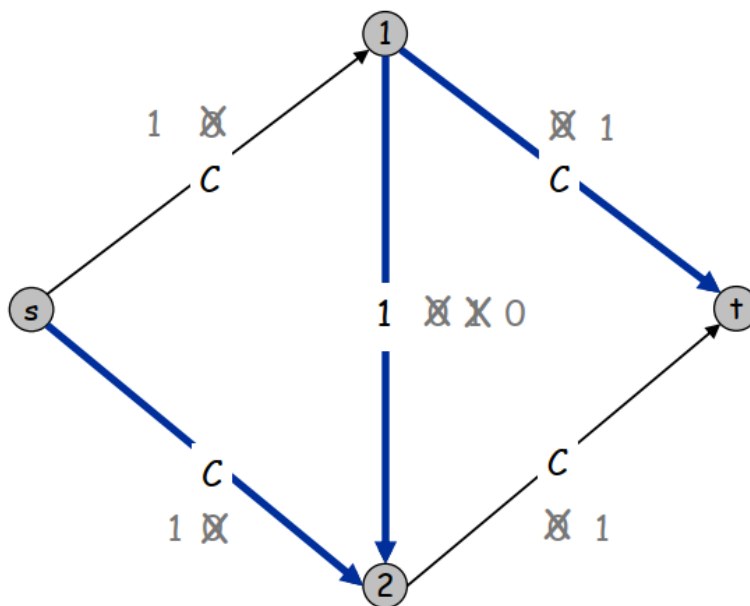


Ford-Fulkerson: Exponential Number of Augmentations

- 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 $\geq C$ iterations



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

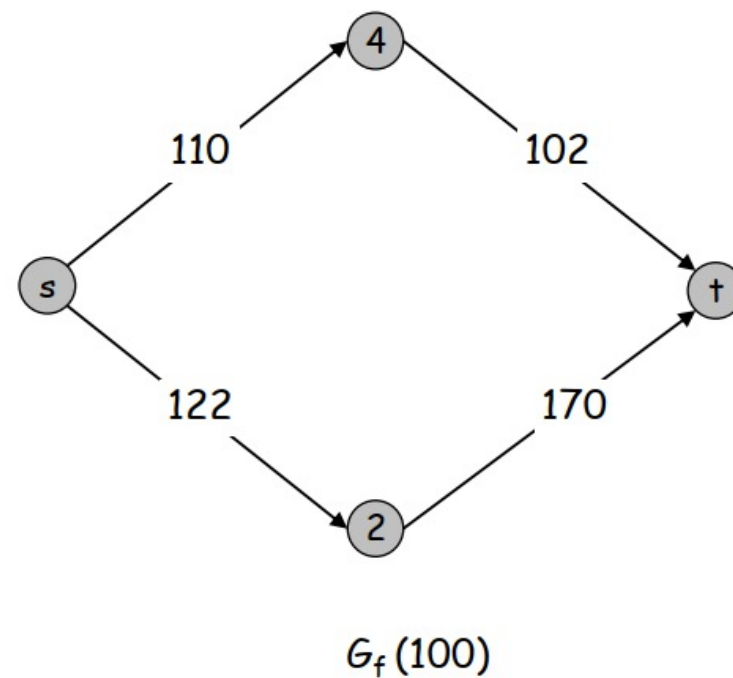
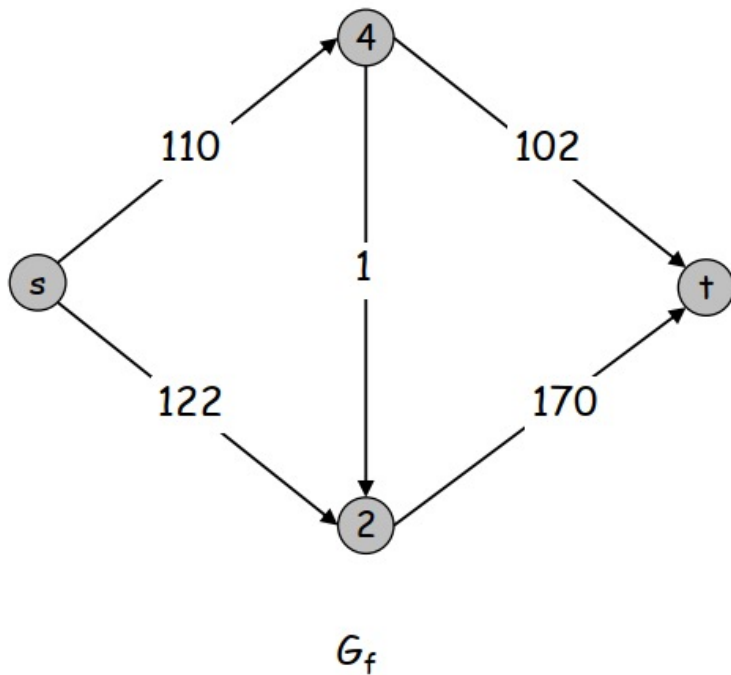


Choosing Good Augmenting Paths

- **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



Capacity Scaling

```
Scaling-Max-Flow( $G, s, t, c$ ) {  
  foreach  $e \in E$   $f(e) \leftarrow 0$   
   $\Delta \leftarrow$  largest power of 2  $\leq C$   
  
  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

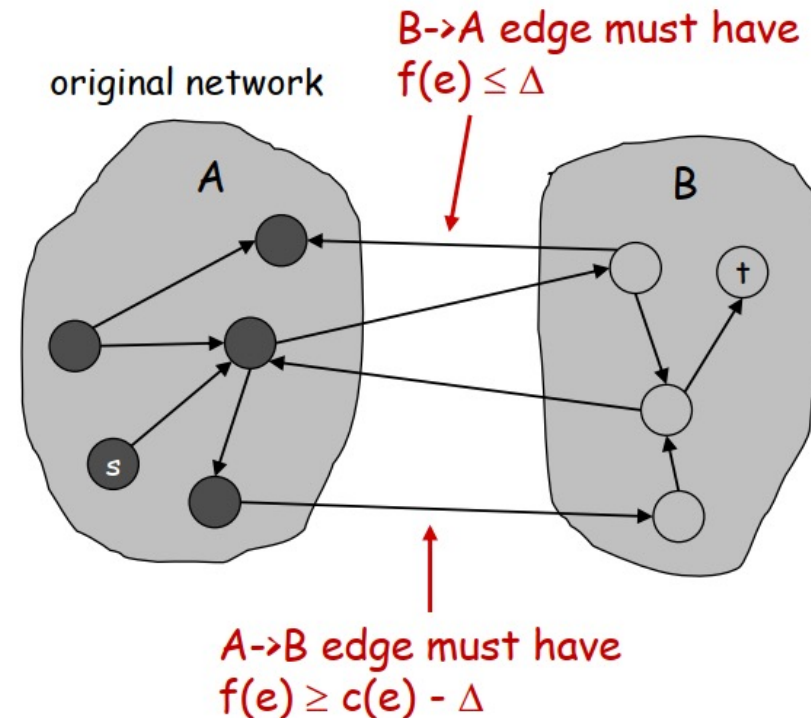
- **Lemma 1.** The outer while loop repeats $1 + \lfloor \log_2 C \rfloor$ times
- Pf. Initially $C/2 < \Delta \leq C$. Δ decreases by a factor of 2 each iteration
- **Lemma 2.** Let f be the flow at the end of a Δ -scaling phase. Then the value of the maximum flow is at most $v(f) + m\Delta \leftarrow$ proof on next slide
- **Lemma 3.** There are at most $2m$ augmentation 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 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 $\text{cap}(A, B) \leq v(f) + m\Delta$
 - 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 , $t \notin A$

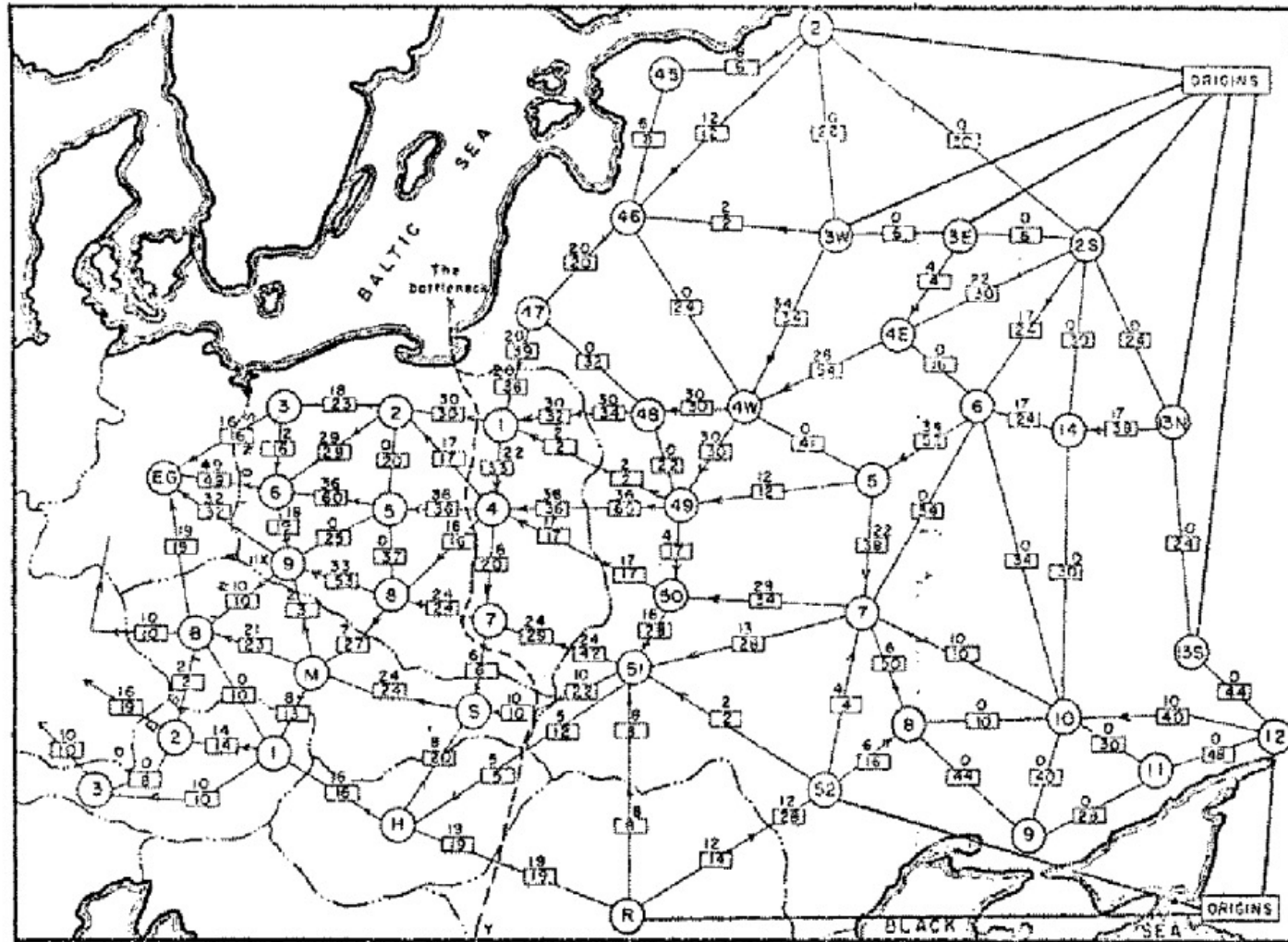
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\ &= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\ &\geq \text{cap}(A,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





Applications of max-flow

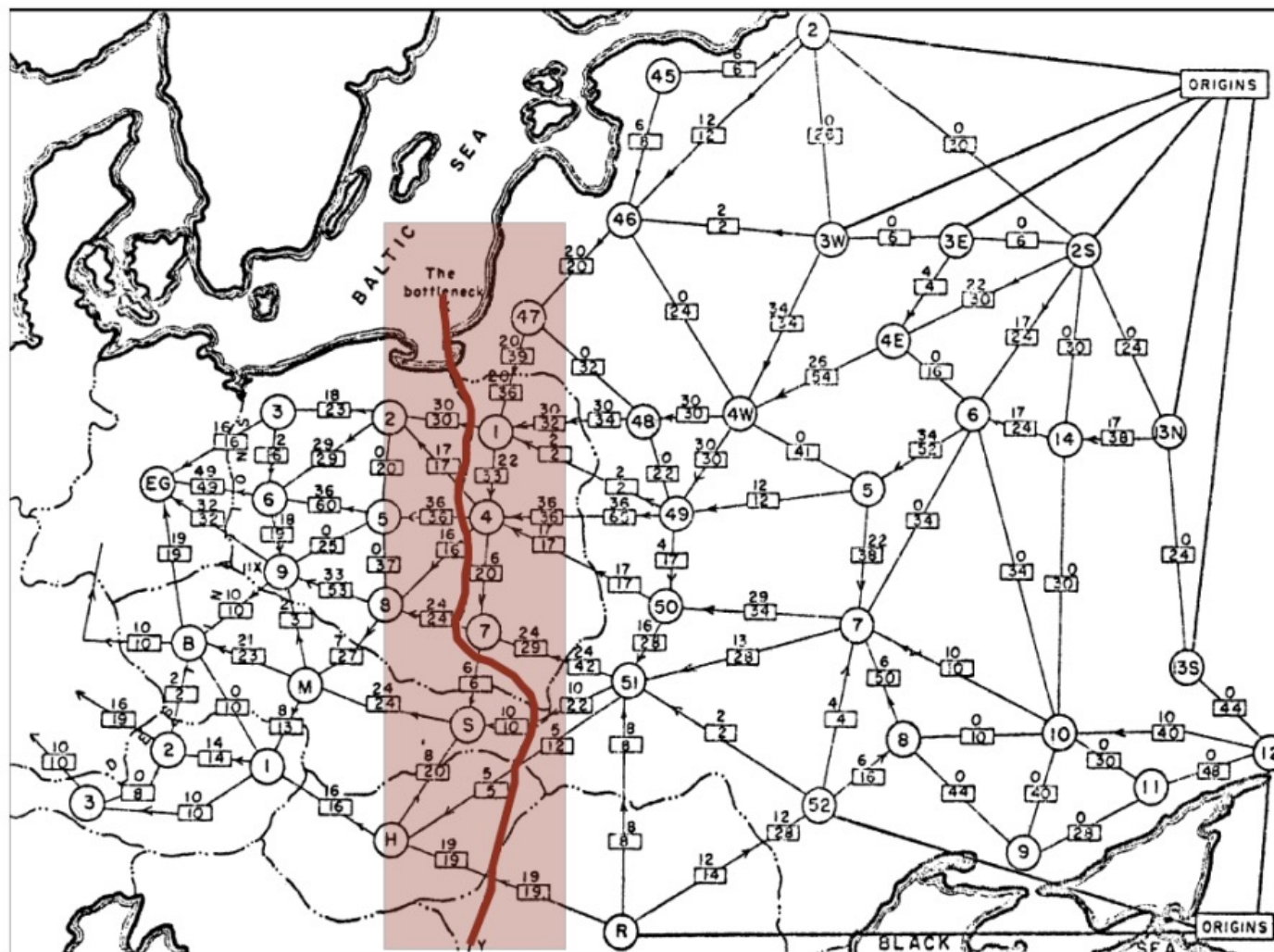
Soviet Rail Network, 1955



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



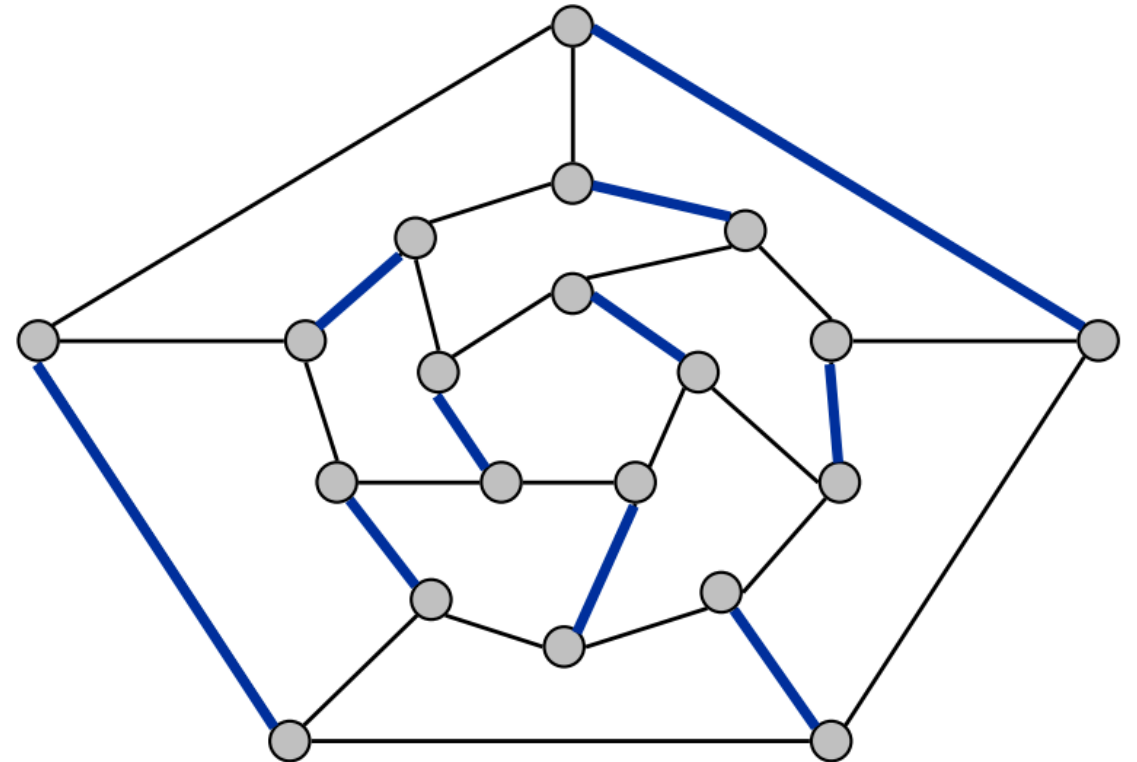
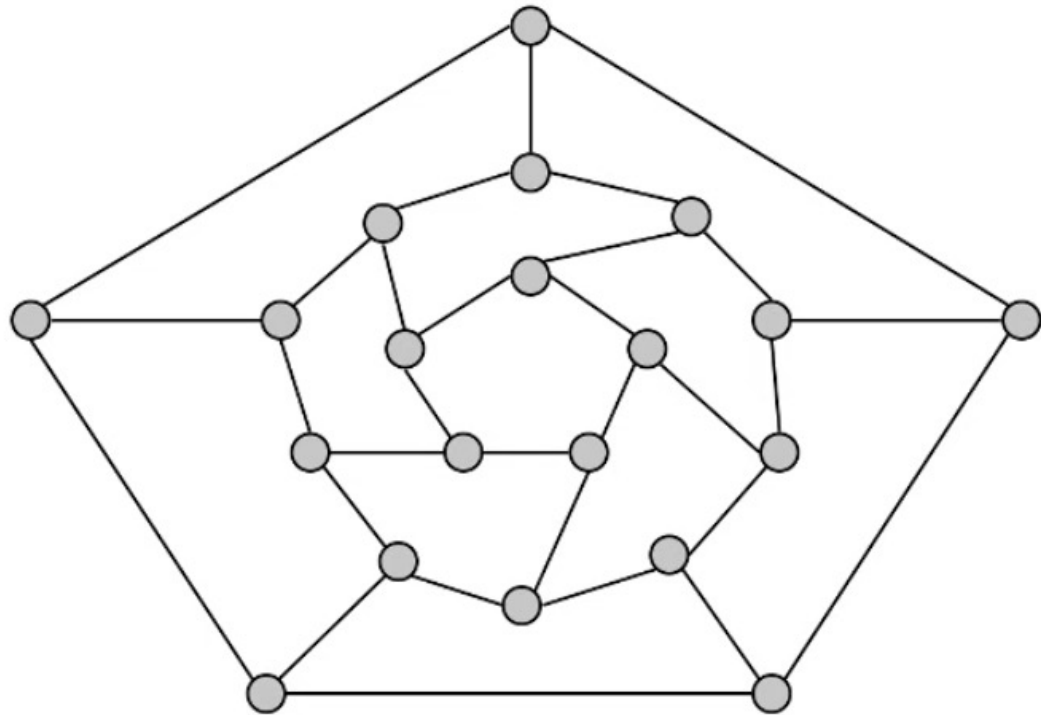
Bipartite Matching



Matching

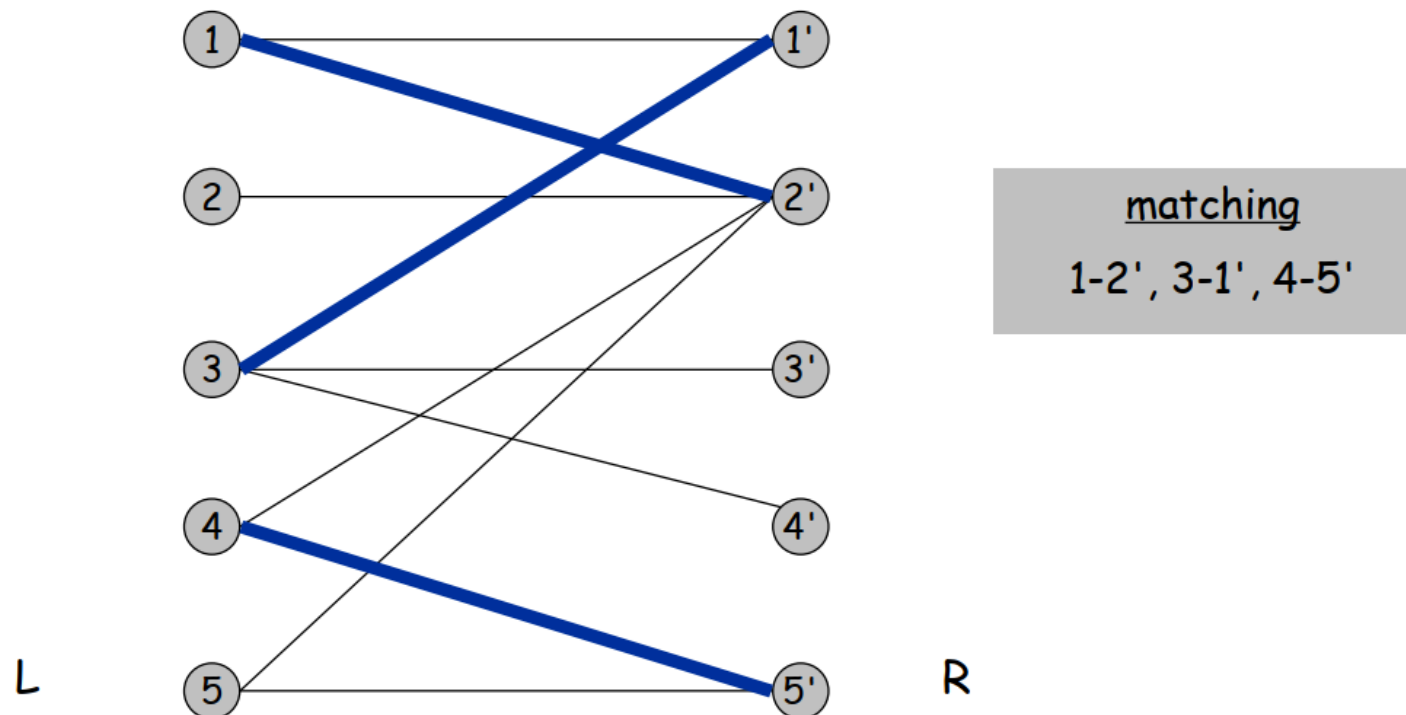
- **Matching**

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



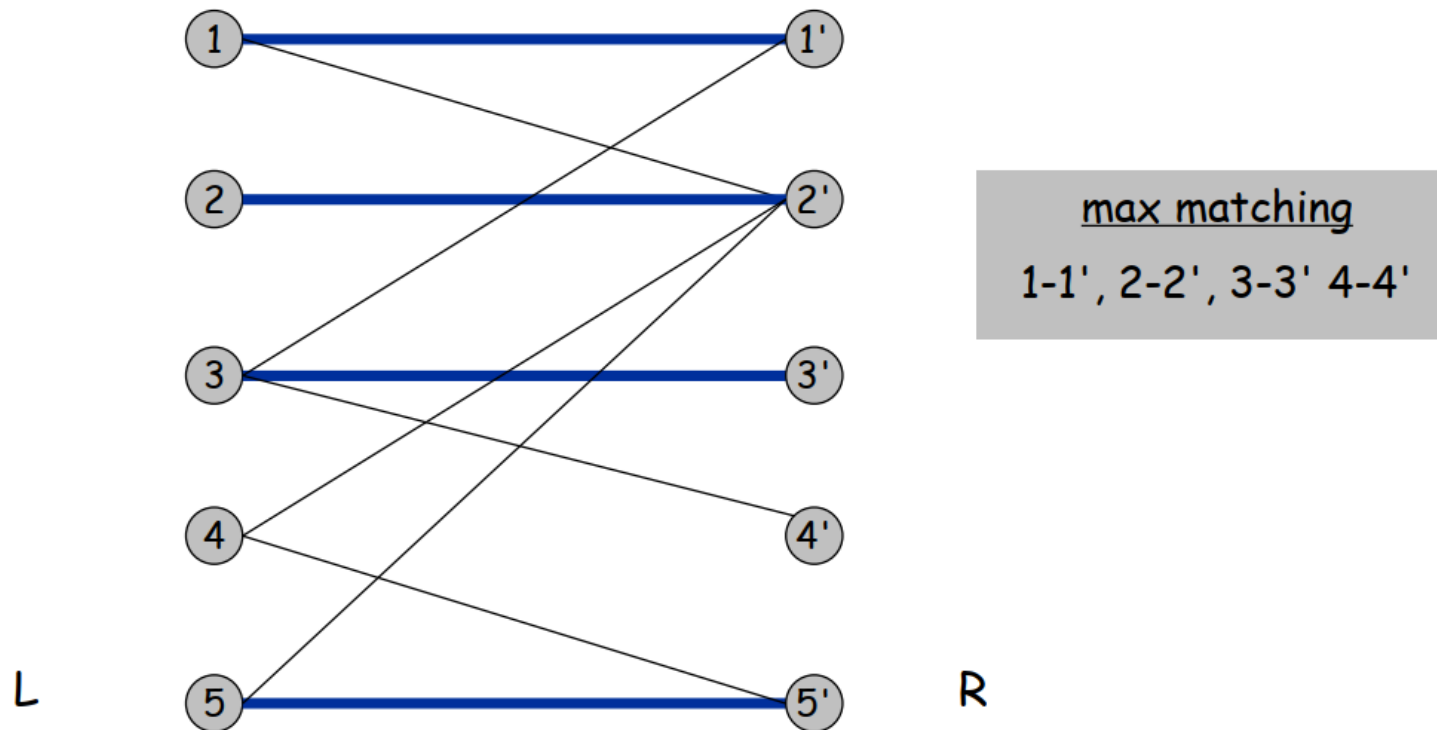
Bipartite Matching

- 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



Bipartite Matching

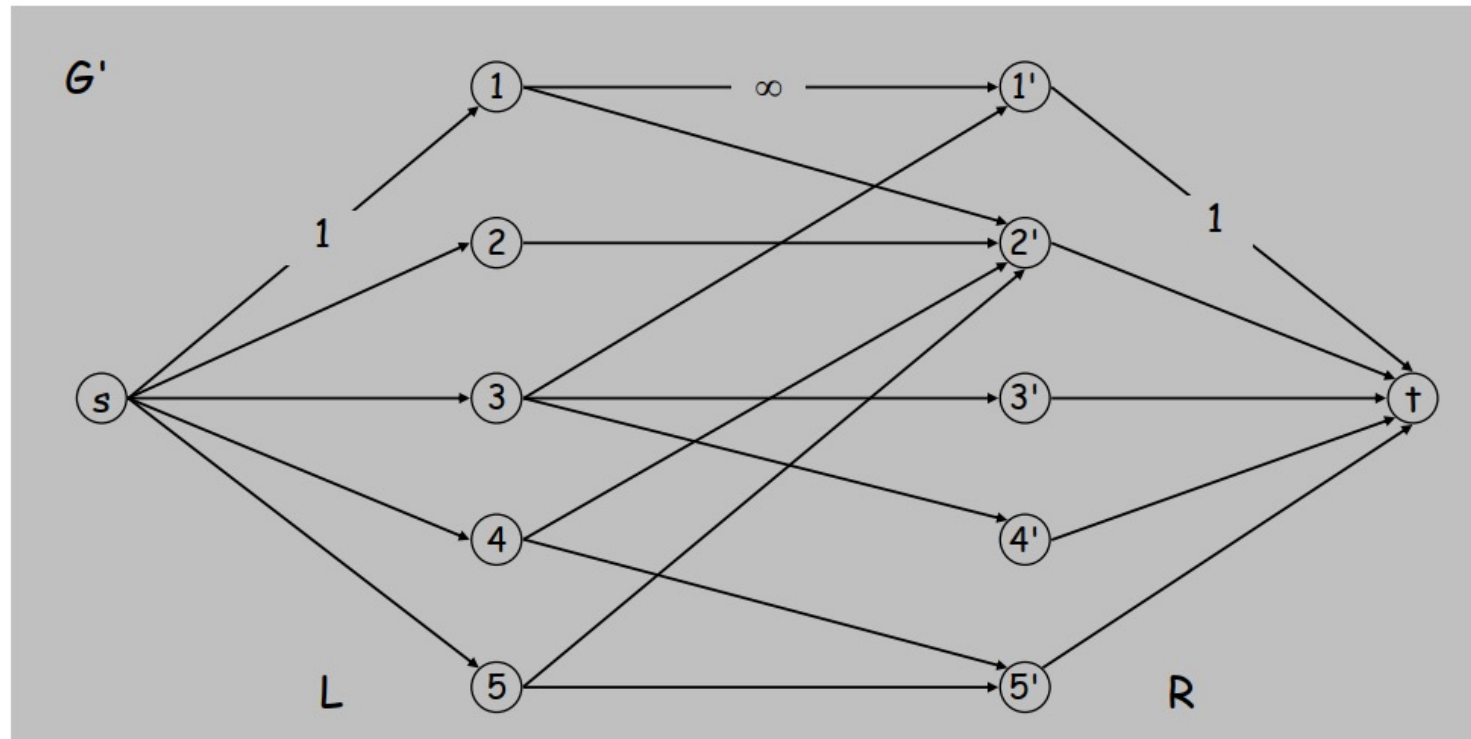
- Bipartite matching
 - Input: undirected, **bipartite** graph $G = (L \cup R, E)$
 - $M \subseteq E$ is a **matching** if each node appears in at most one edge in M
 - Max matching: find a max cardinality matching



Bipartite Matching

- **Max flow formulation**

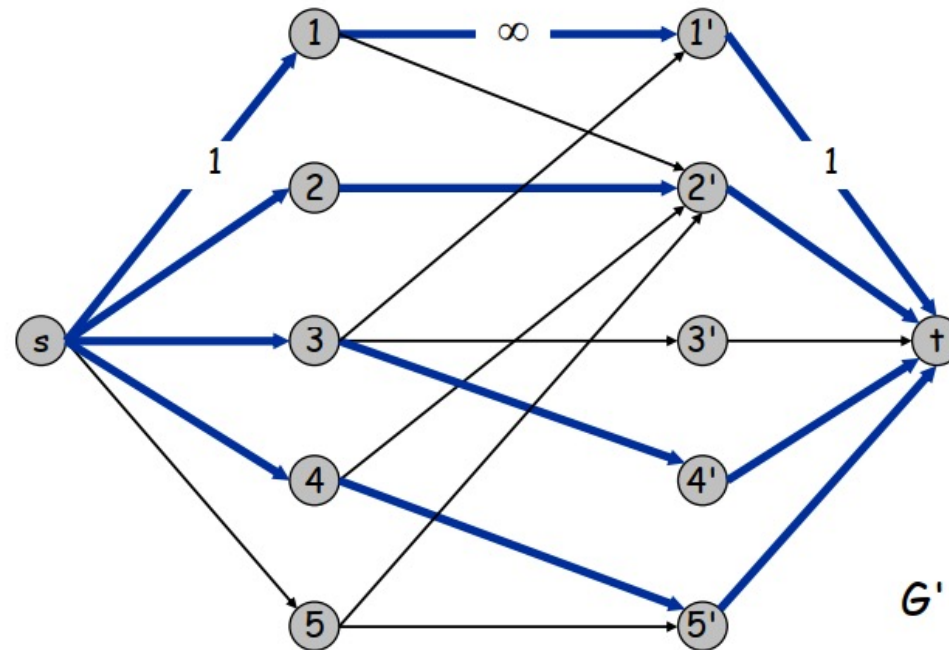
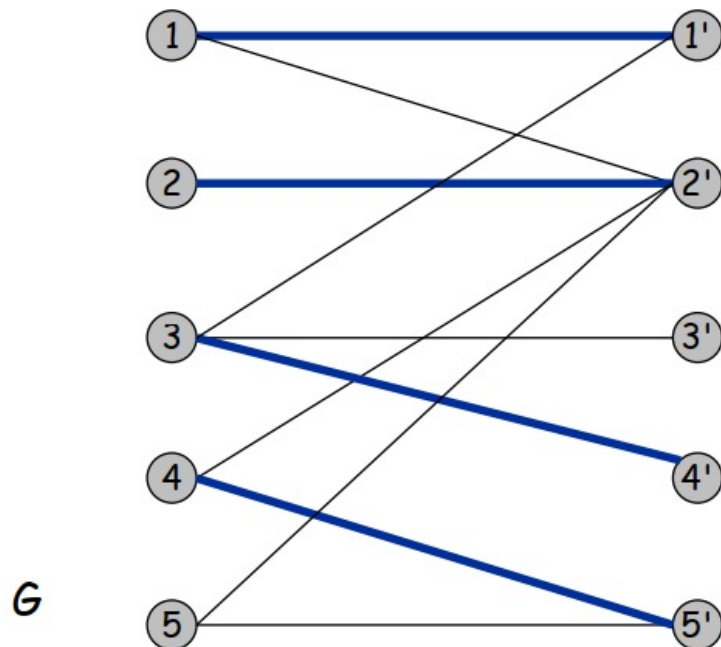
- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$
- Direct all edges from L to R, and assign infinite (or unit) capacity
- Add source s , and unit capacity edges from s to each node in L
- Add sink t , and unit capacity edges from each node in R to t





Bipartite Matching: Proof of Correctness

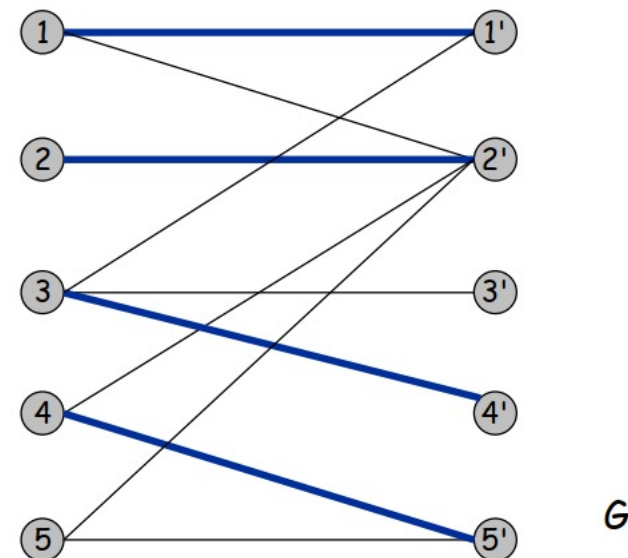
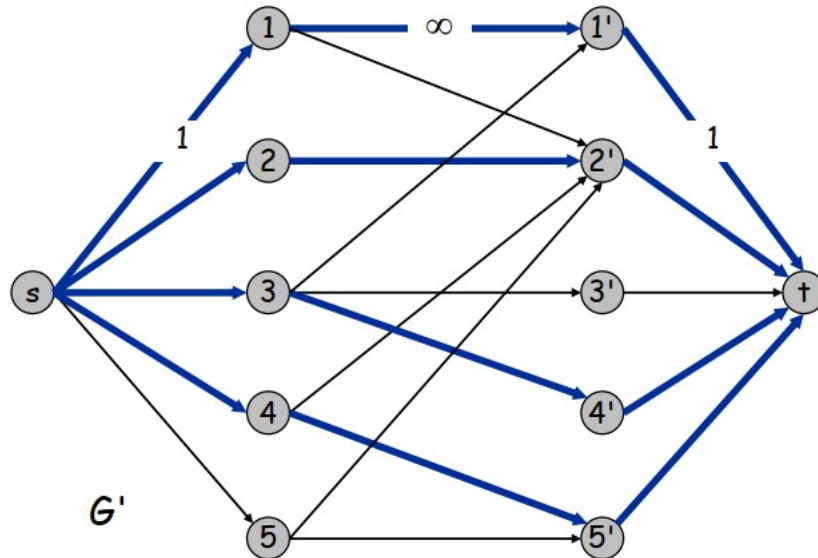
- **Theorem.** Max cardinality matching in G = value of max flow in G'
- **Pf. \Leftarrow**
 - Given max matching M of cardinality k
 - Consider flow f that sends 1 unit along each of k paths
 - f is a flow, and has cardinality k





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 $(L \cup s, R \cup t)$

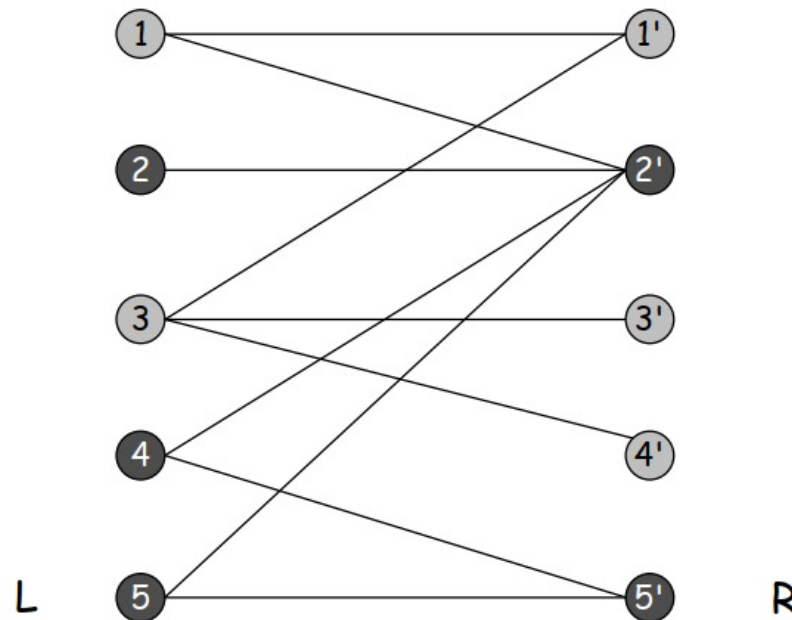


Perfect Matching

- **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

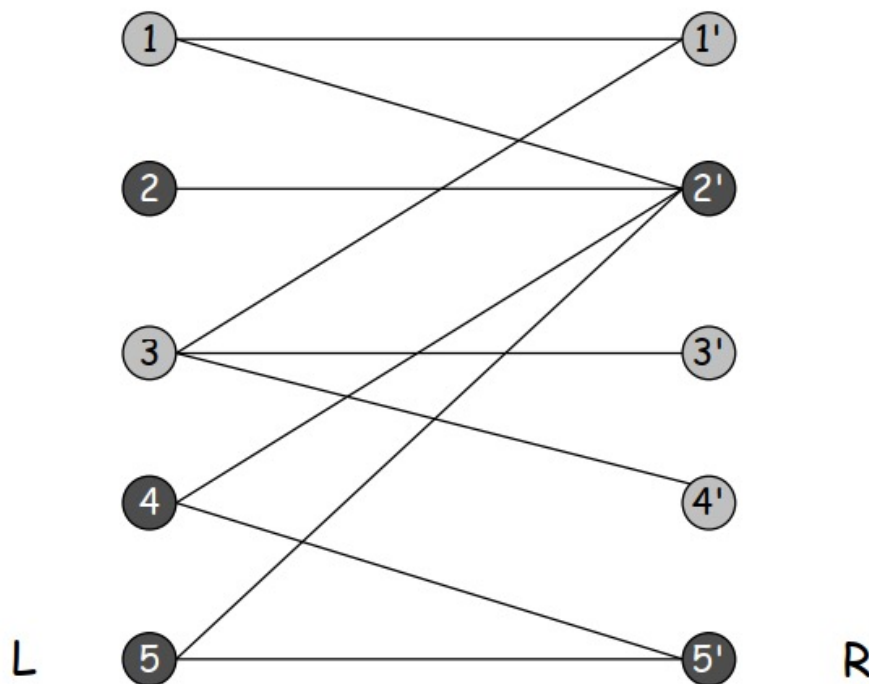
- **Notation.** Let S be a subset of nodes, and let $N(S)$ be the set of nodes adjacent to nodes in S
- **Observation.** If a bipartite graph $G = (L \cup R, E)$, has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$
- **Pf.** Each node in S has to be matched to a different node 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:

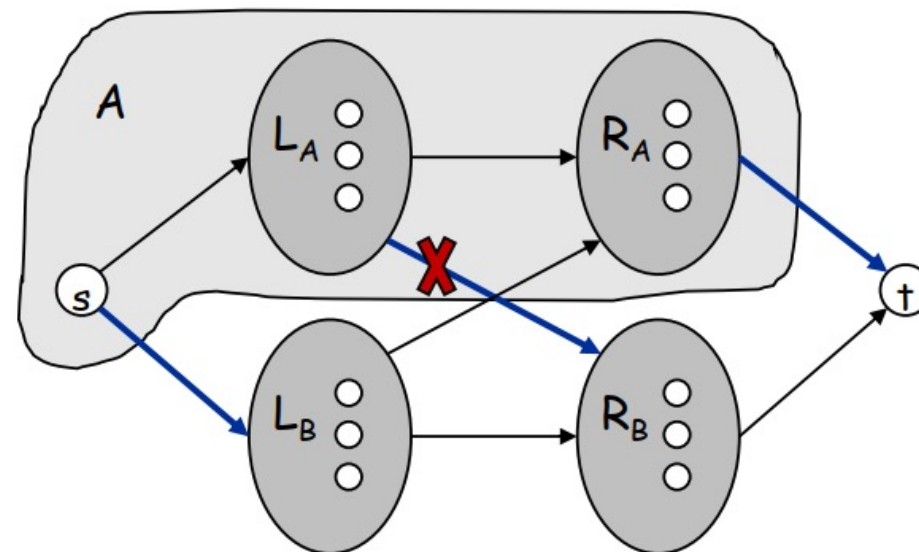
$S = \{ 2, 4, 5 \}$

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



Proof of Marriage Theorem

- **Marriage Theorem.** G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$
- **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'
 - Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$, $R_B = R \cap B$
 - $\text{Cap}(A, B) = v(f^*) = |M| < |L|$ (" $<$ ": because no perfect matching)
 - Since min cut can't use ∞ edges, no edge between L_A and R_B
 - $\text{Cap}(A, B) = |L_B| + |R_A|$
 - $N(L_A) \subseteq R_A$
 - $|N(L_A)| \leq |R_A|$
 $= \text{cap}(A, B) - |L_B|$
 $< |L| - |L_B|$
 $= |L_A|$
 - This contradicts the condition

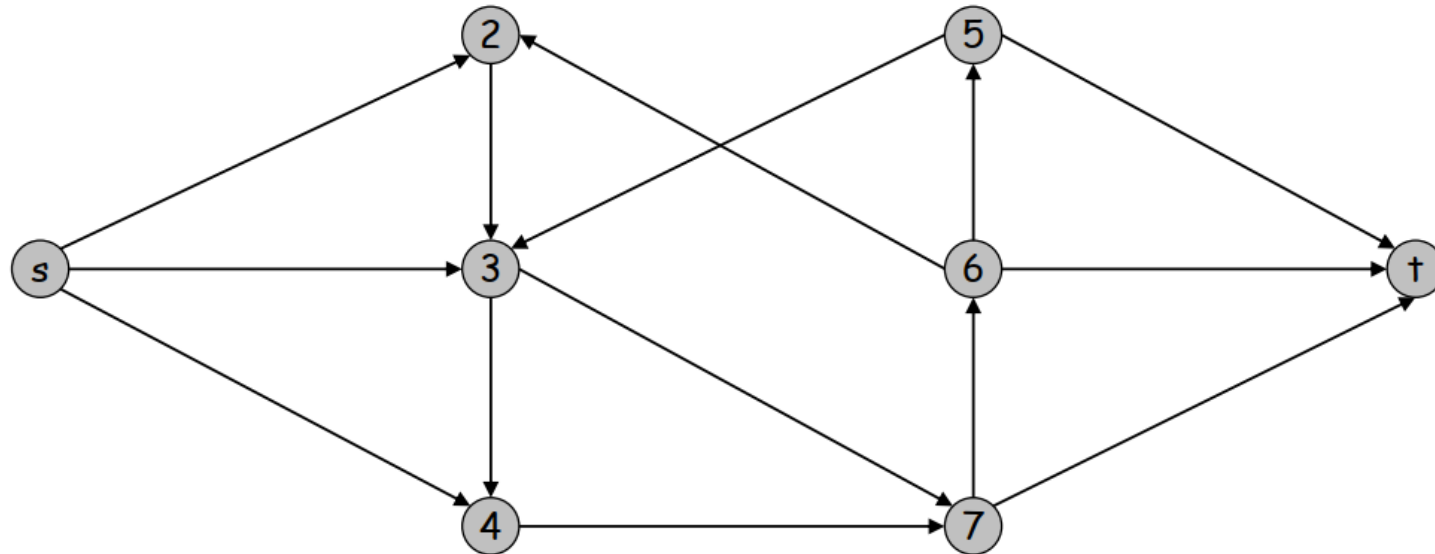




Disjoint Paths

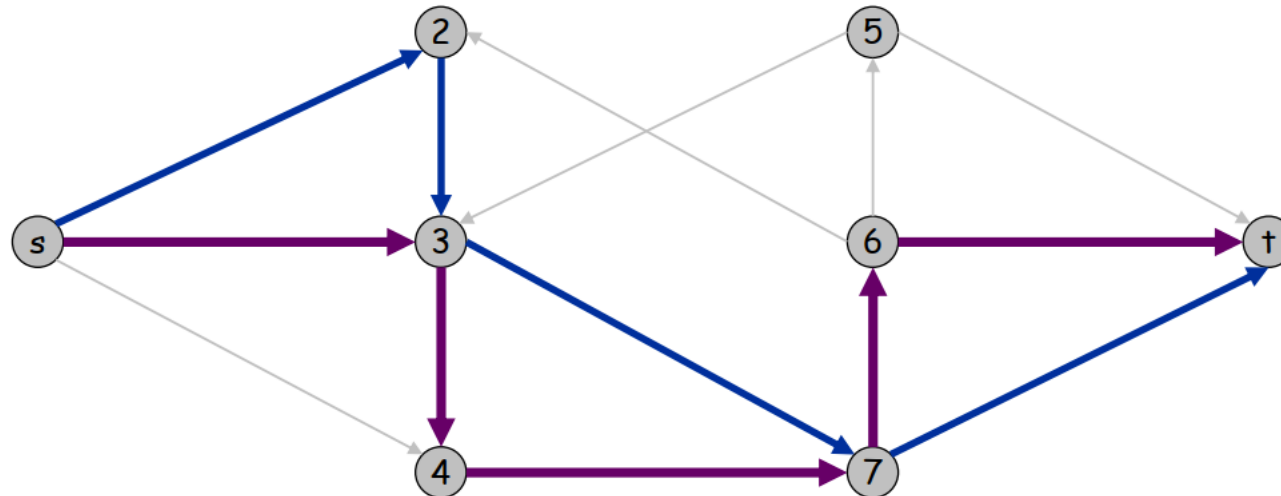
Edge Disjoint Paths

- **Disjoint path problem.** Given a digraph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths
- **Def.** Two paths are **edge-disjoint** if they have no edge in common
- **Ex:** communication networks



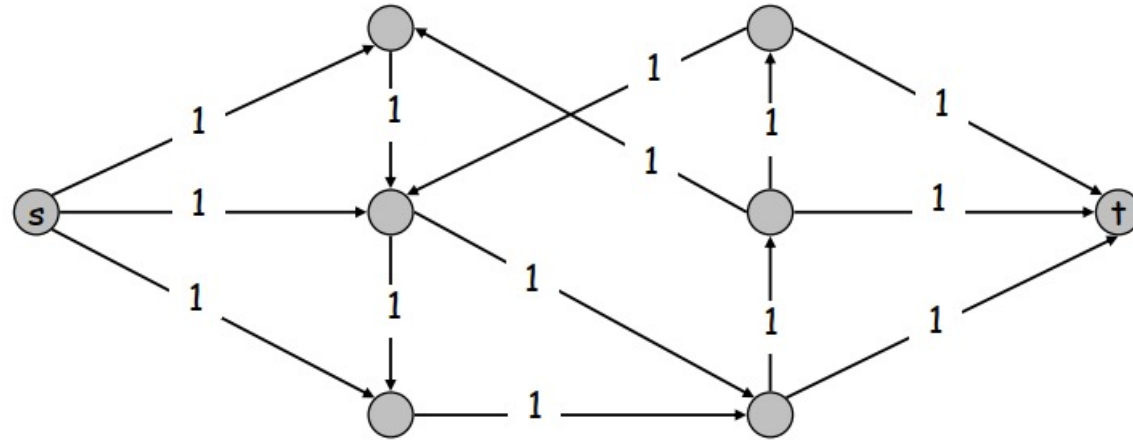
Edge Disjoint Paths

- **Disjoint path problem.** Given a digraph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths
- **Def.** Two paths are **edge-disjoint** if they have no edge in common
- **Ex:** communication networks



Edge Disjoint Paths

- **Max flow formulation:** assign unit capacity to every edge



- **Theorem.** Max number edge-disjoint s-t paths equals to max flow value

Edge Disjoint Paths

- **Theorem.** Max number edge-disjoint s-t paths equals max flow value
- **Pf. \Leftarrow**
 - Suppose there are k edge-disjoint paths P_1, \dots, P_k
 - Set $f(e) = 1$ if e participates in some path P_i ; else set $f(e) = 0$
 - Since paths are edge-disjoint, f is a flow of value k

Edge Disjoint Paths

- **Theorem.** Max number edge-disjoint s - t paths equals max flow value
- **Pf. \geq**
- Suppose max flow value is k
- Integrality theorem \rightarrow there exists 0-1 flow f of value k
- Consider edge (s, u) with $f(s, u) = 1$
 - By conservation, there exists an edge (u, v) with $f(u, v) = 1$
 - Continue until reach t , always choosing a new edge
 - So we get a s - t path
- Reduce the flow to 0 along the path, so we get a flow of value $k-1$
- Repeat the process for k times, then we get k (not necessarily simple) edge-disjoint paths

Can eliminate cycles to get simple paths if desired





Extensions to Max Flow



Circulation with Demands

- **Circulation with demands**

- Directed graph $G = (V, E)$
- Edge capacities $c(e)$, $e \in E$
- Node supply and demands $d(v)$, $v \in V$



demand if $d(v) > 0$; supply if $d(v) < 0$; transshipment if $d(v) = 0$

- **Def.** A **circulation** is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)

- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

- **Circulation problem:** given (V, E, c, d) , does there exist a circulation?

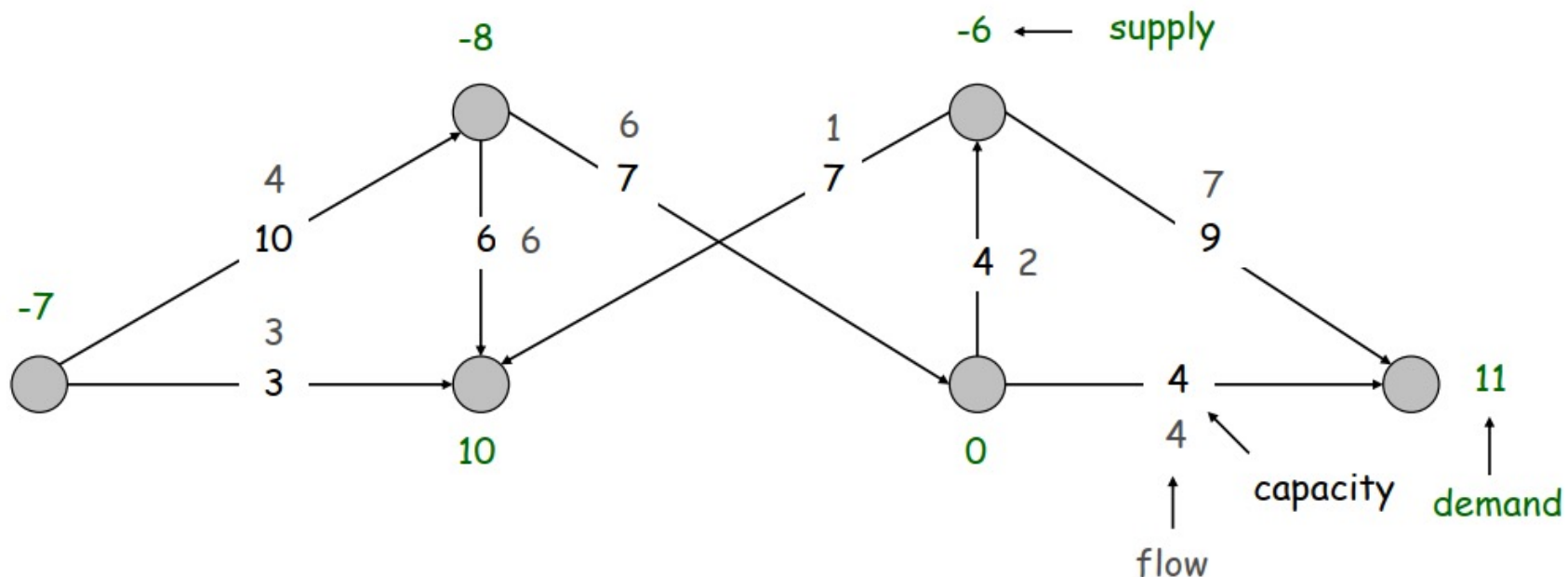


Circulation with Demands

- **Necessary condition:** sum of supplies = sum of demands

$$\sum_{v: d(v) > 0} d(v) = \sum_{v: d(v) < 0} -d(v) =: D$$

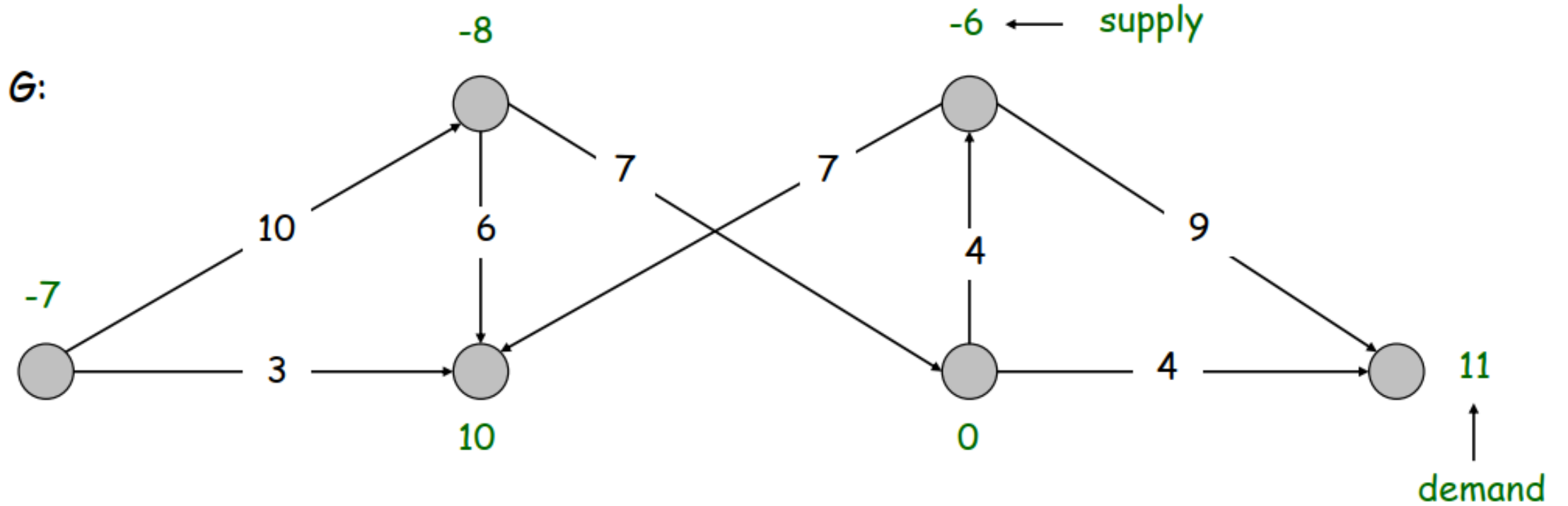
- **Pf.** Sum conservation constraints for every demand node v





Circulation with Demands

- Max flow formulation

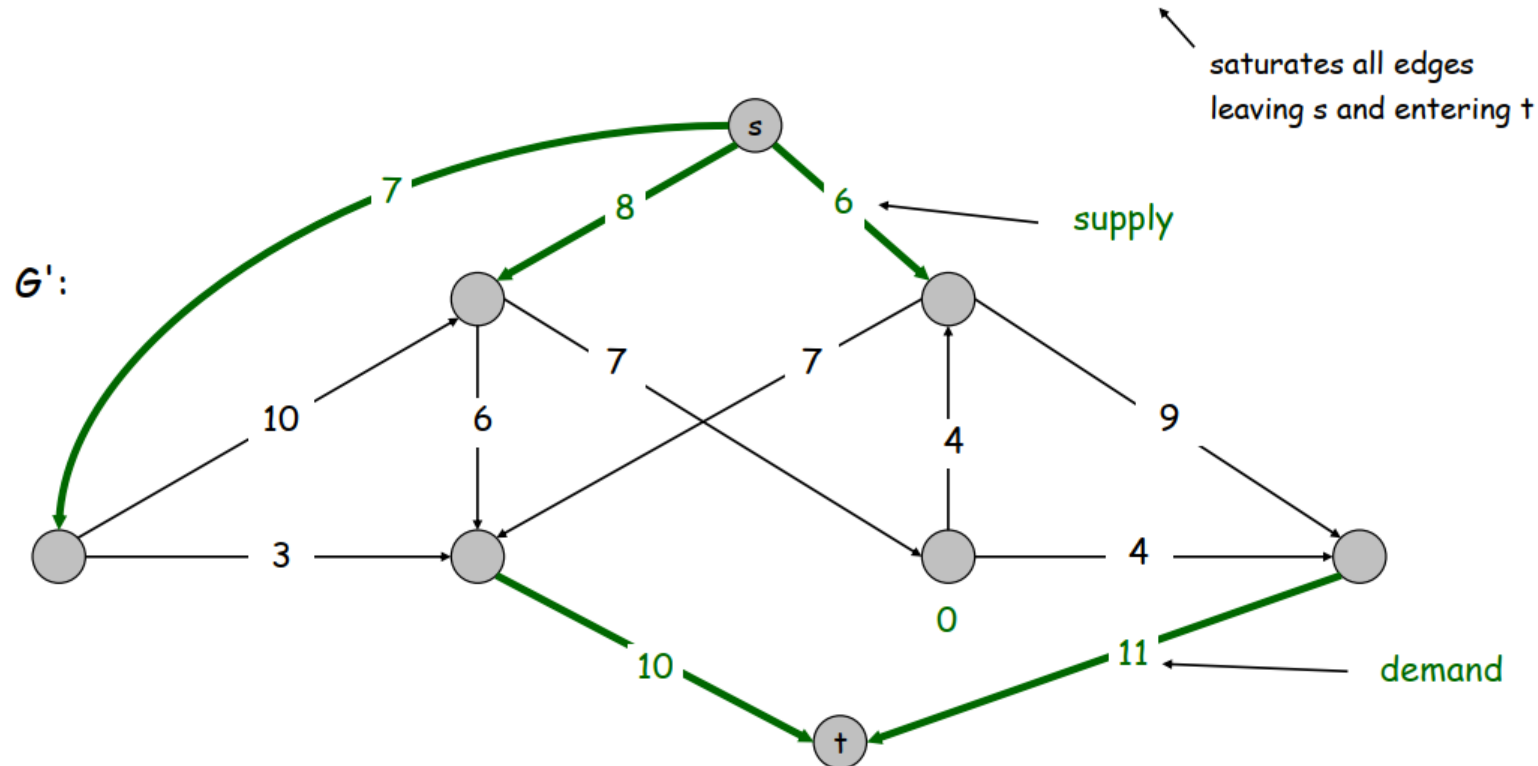




Circulation with Demands

- **Max flow formulation**

- Add new source s and sink t
- For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$
- For each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$
- Claim. G has circulation iff G' has max flow of value D





Circulation with Demands

- **Integrality theorem.** If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued
- **Pf.** Follows from max flow formulation and integrality theorem for max flow
- **Characterization.** Given (V, E, c, d) , there does **not** exist a circulation iff there exists a node partition (A, B) such that $\sum_{v \in B} d_v > \text{cap}(A, B)$
 - ↑
demand by nodes in B exceeds supply of nodes in B plus max capacity of edges going from A to B
- **Pf idea.** Look at max flow and min cut in G'



Circulation with Demands and Lower Bounds

- **Feasible circulation**

- Directed graph $G = (V, E)$
- Edge capacities $c(e)$ and **lower bounds** $\ell(e)$, $e \in E$
- Node supply and demands $d(v)$, $v \in V$

- **Def.** A **circulation** is a function that satisfies:

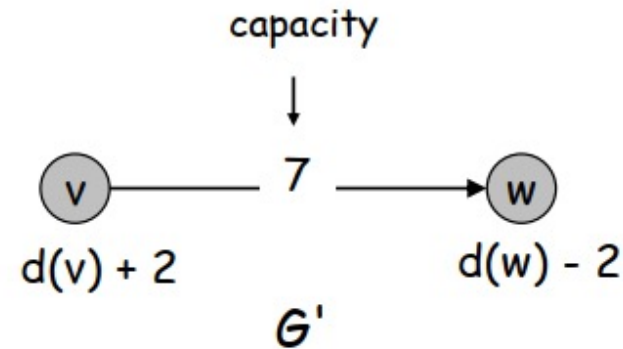
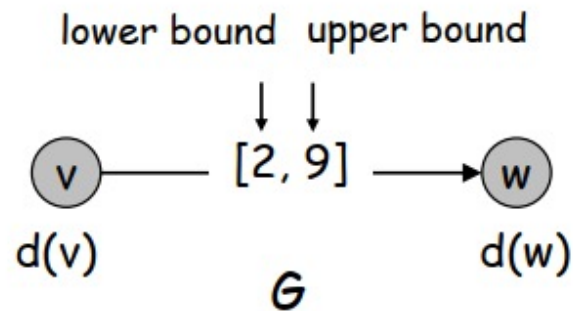
- For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

- **Circulation problem with lower bounds.** Given (V, E, ℓ, c, d) , does there exist a circulation?



Circulation with Demands and Lower Bounds

- **Idea.** Model lower bounds with demands
 - Send $\ell(e)$ units of flow along edge e
 - Update demands of both endpoints



- **Theorem.** There exists a circulation in G iff there exists a circulation in G' . If all demands, capacities, and lower bounds in G are integers, then there is a circulation in G that is integer-valued
- **Pf sketch.** $f(e)$ is a circulation in G iff $f'(e) = f(e) - \ell(e)$ is a circulation in G'



**Next Time:
Network Flow (Cont.)**