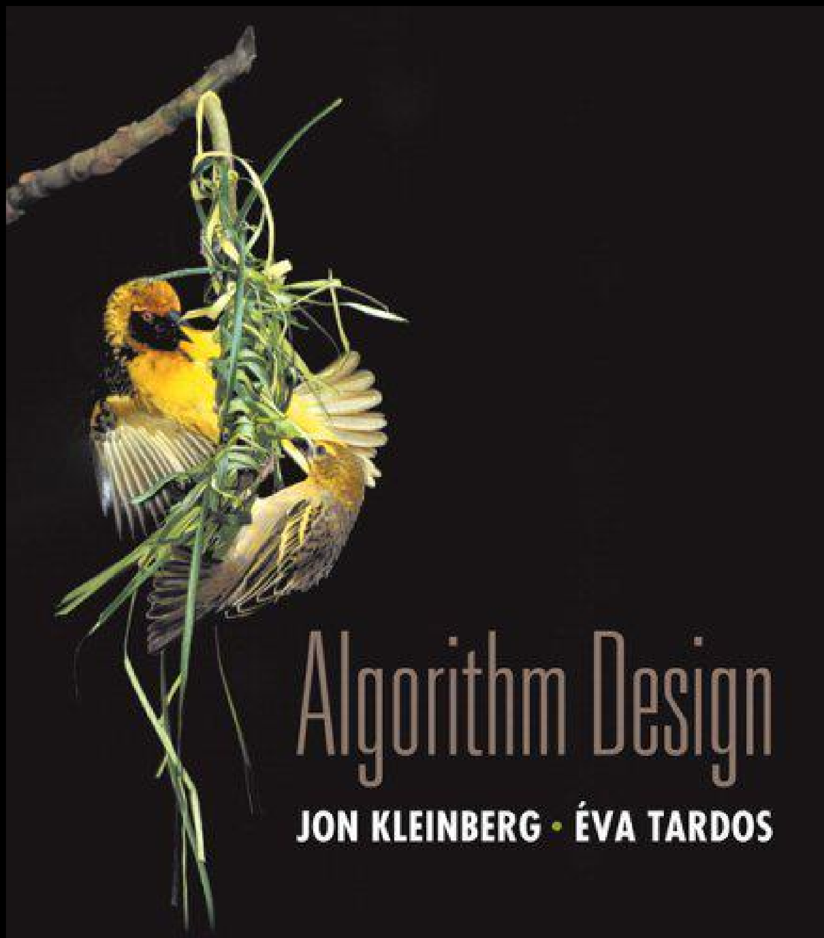
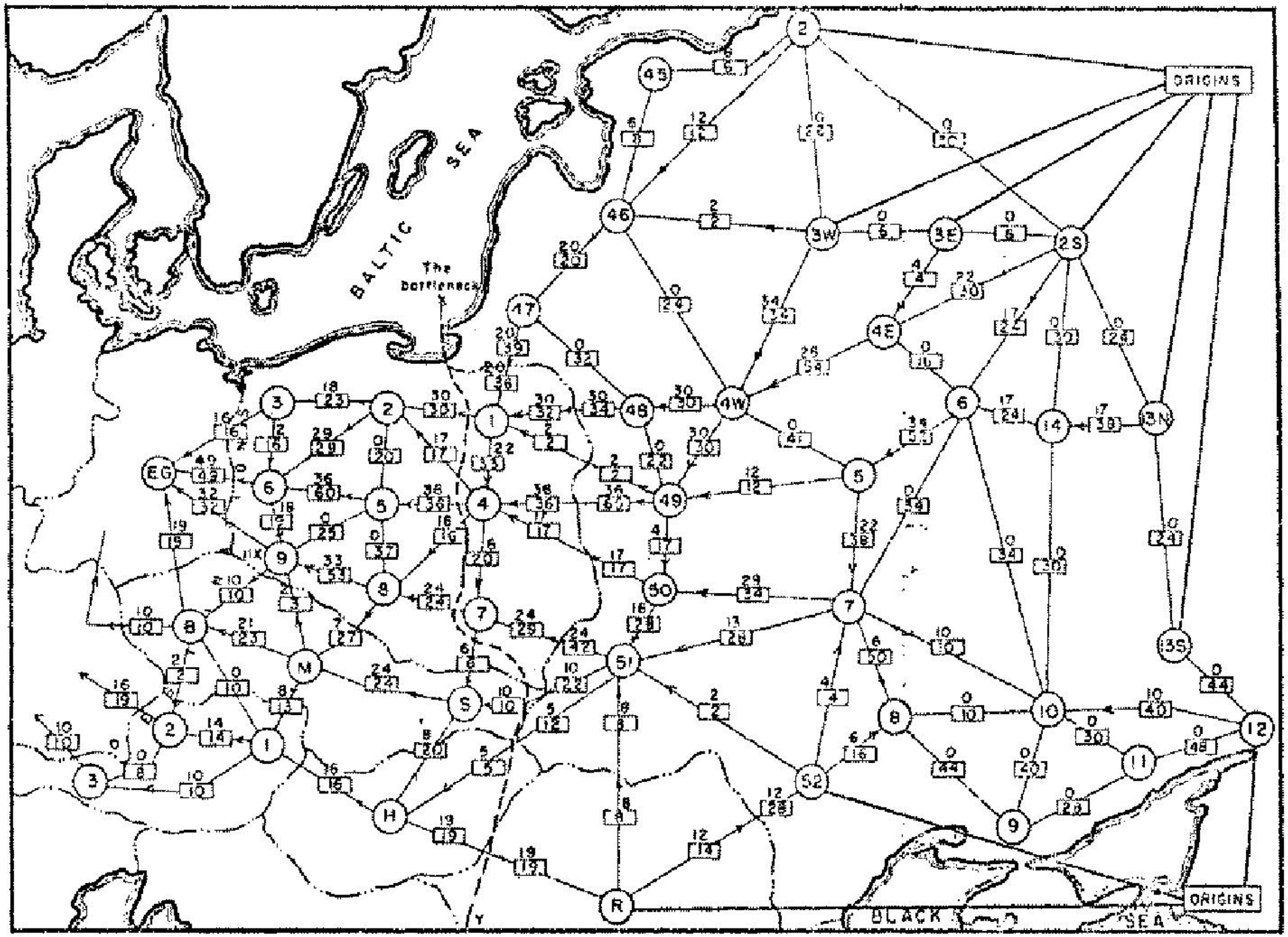


Chapter 7

Network Flow

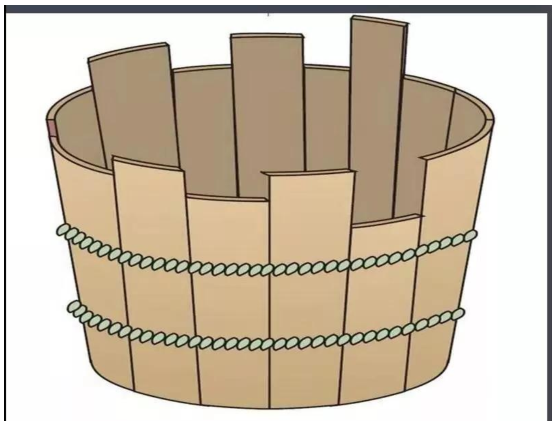
Slides by Kevin Wayne.

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



the barrel effect

the capacity of a barrel is determined not by the longest wooden bars, but by the shortest

Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

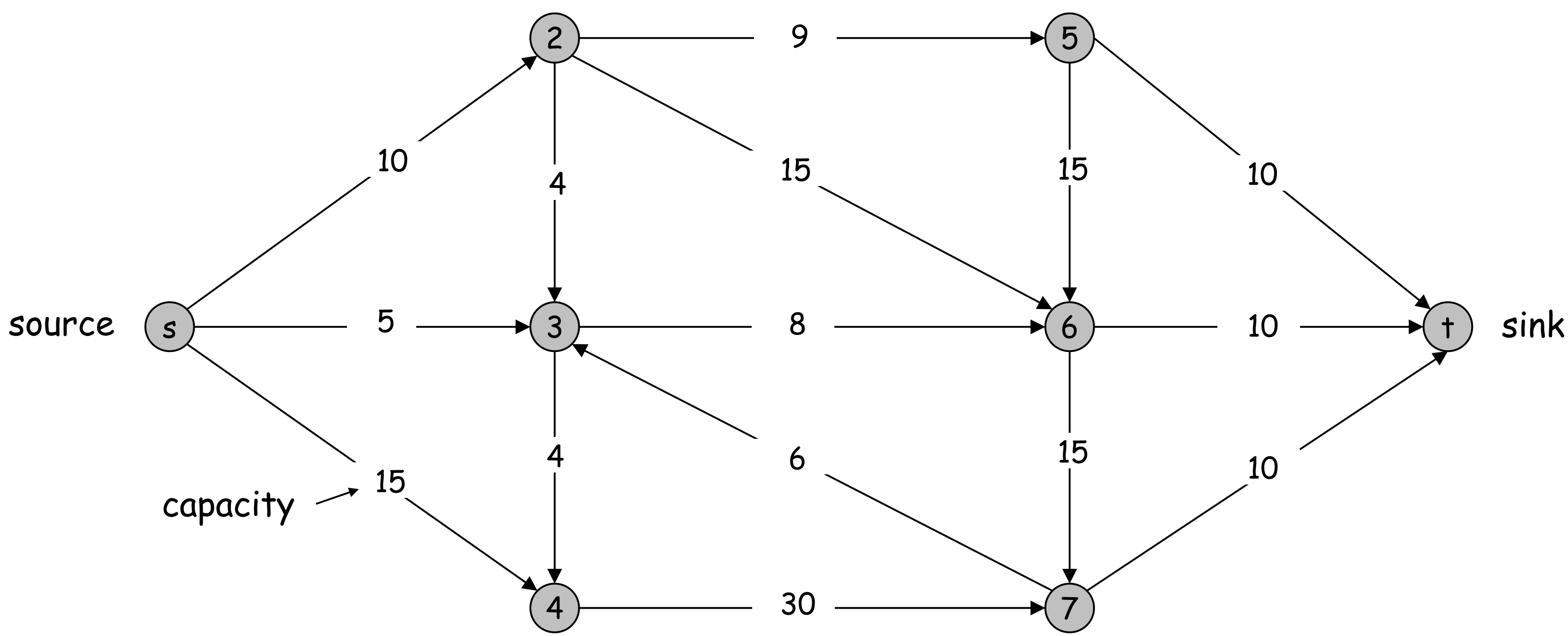
Nontrivial applications / reductions.

- 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.

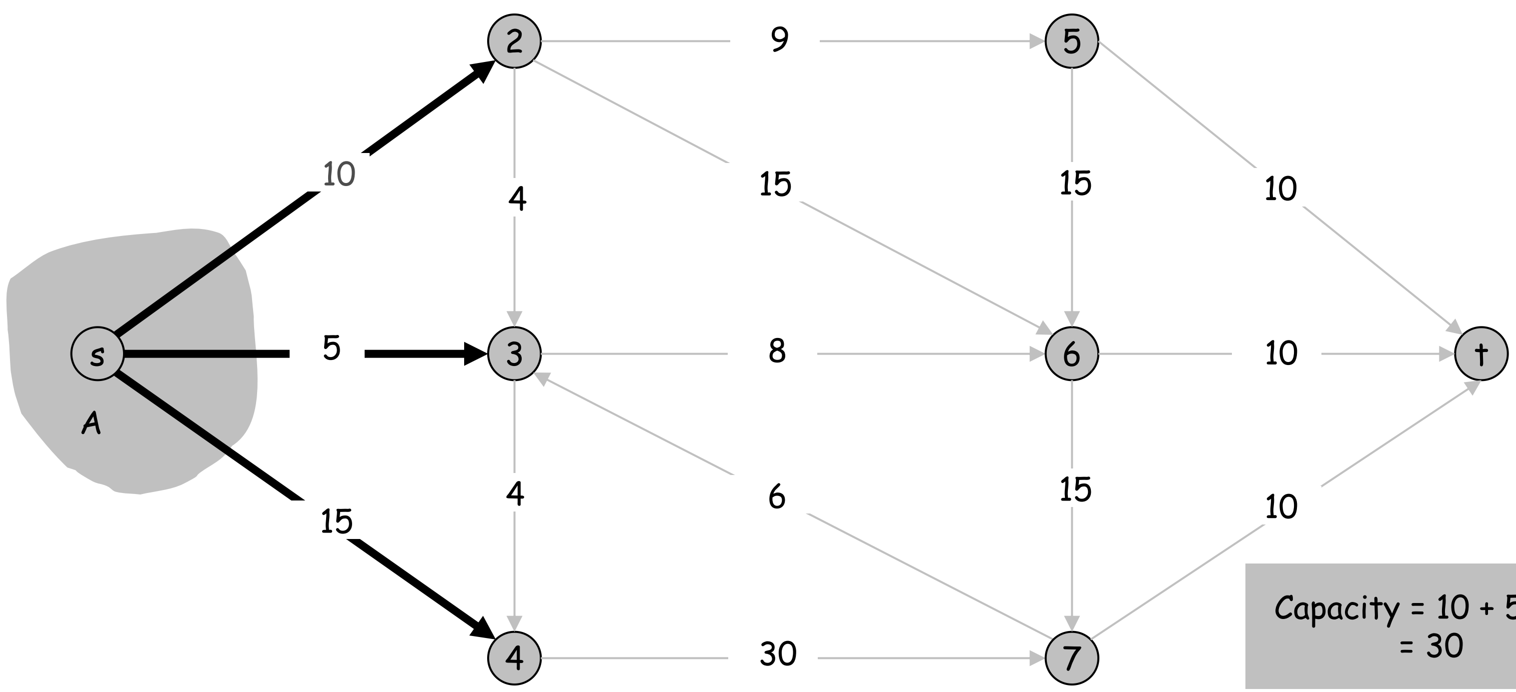
- Abstraction for material flowing through the edges.
- G = (V, E) = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- c(e) = capacity of edge e.



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

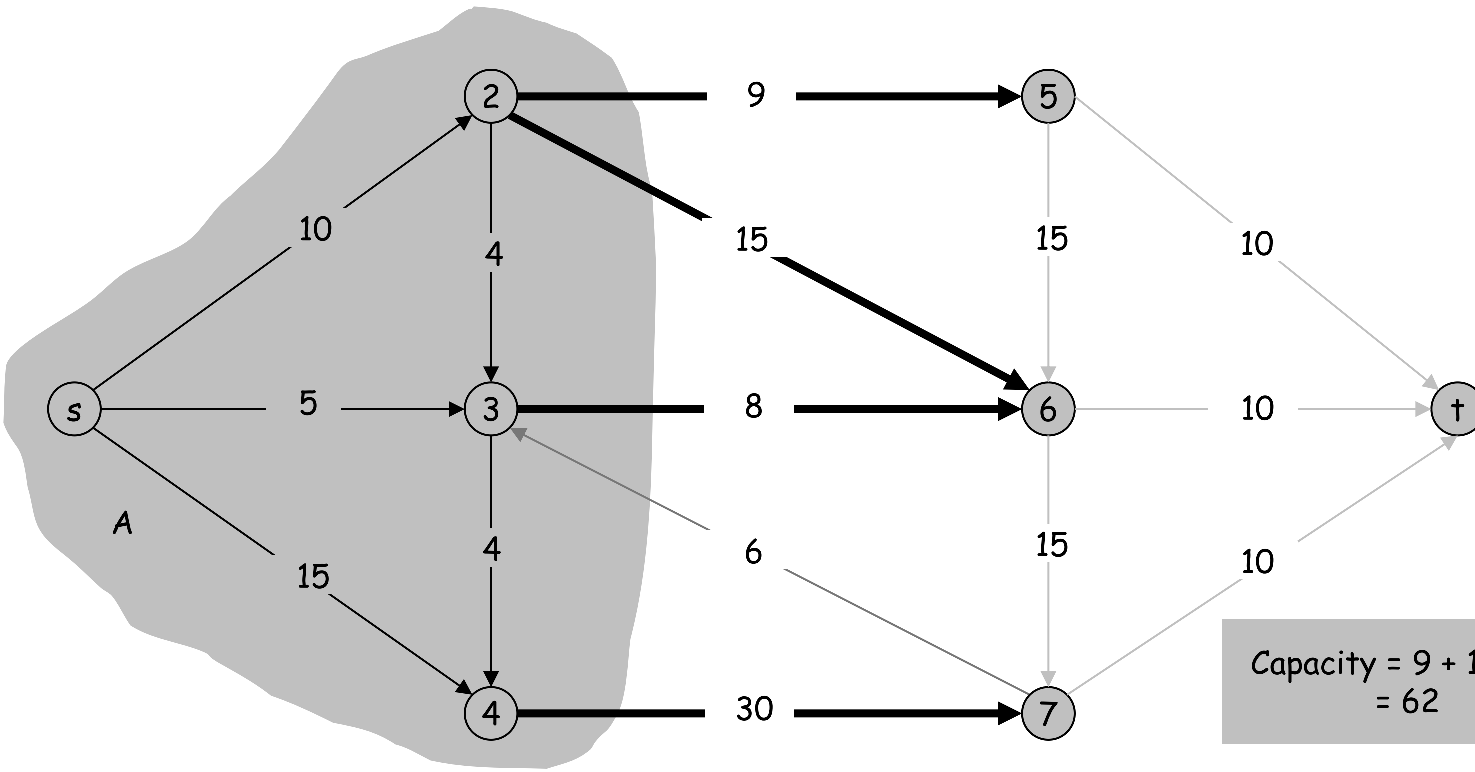


Capacity = 10 + 5 + 15
= 30

Cuts

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

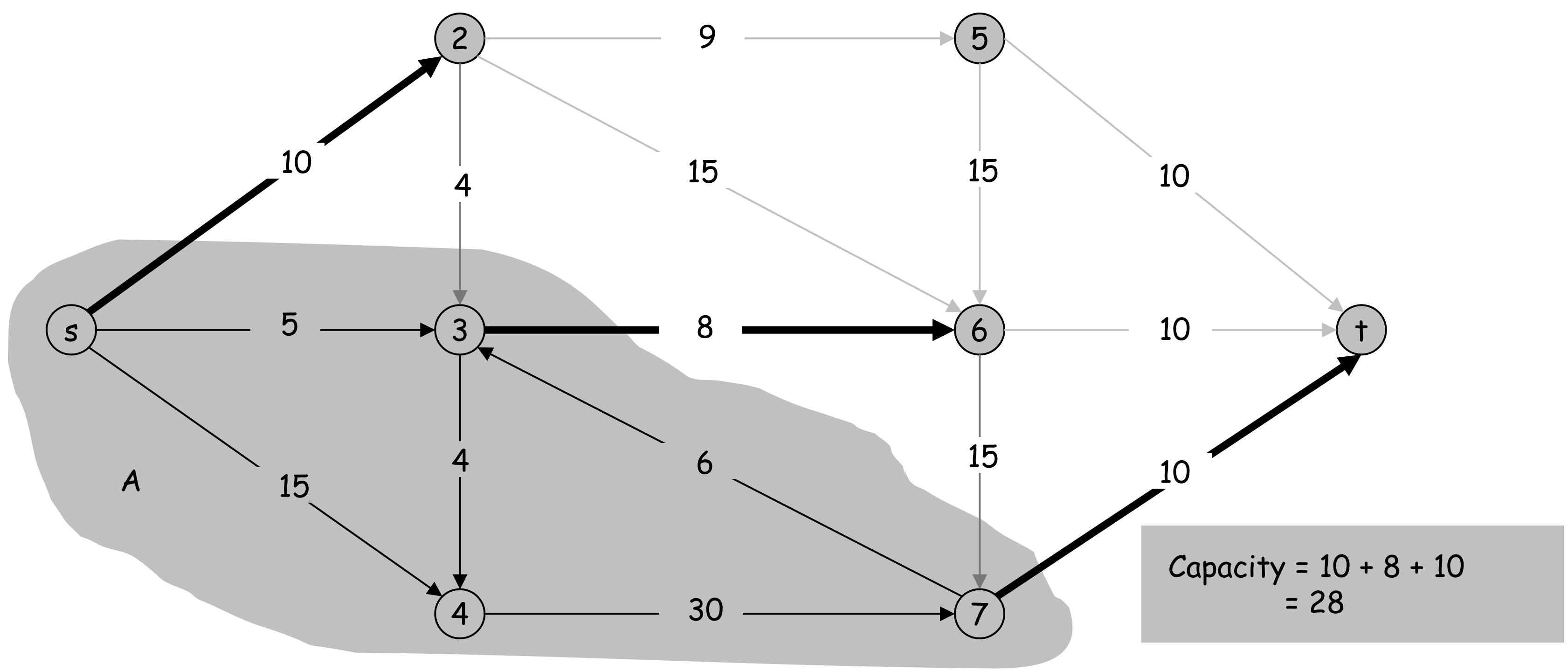
Def. The capacity of a cut (A, B) is: $$cap(A, B) = \sum_{e \text{ out of } A} c(e)$$



Capacity = 9 + 15 + 8 + 30
= 62

Minimum Cut Problem

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

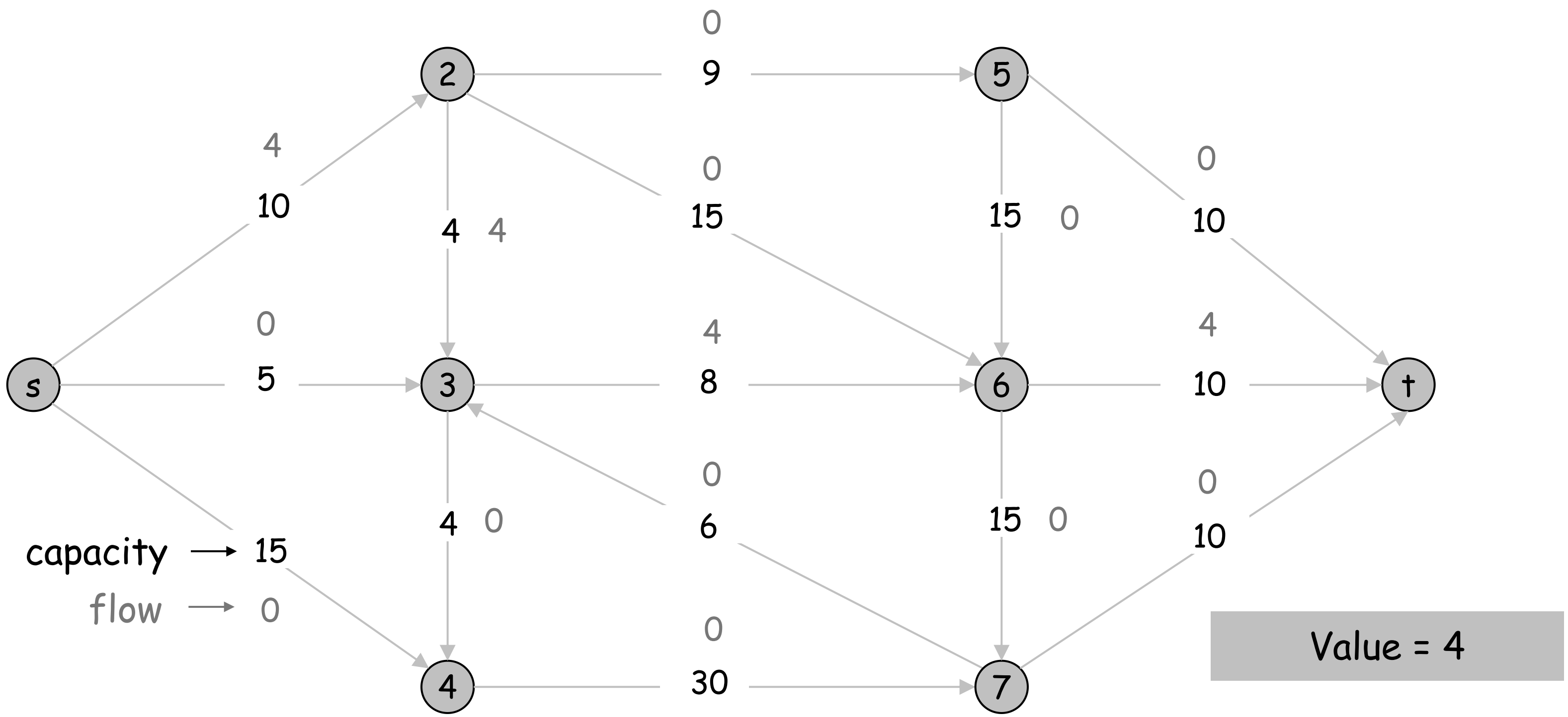


Flows

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

- For each $e \in E$: $0 \le f(e) \le c(e)$ [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: $v(f) = \sum_{e \text{ out of } s} f(e)$.

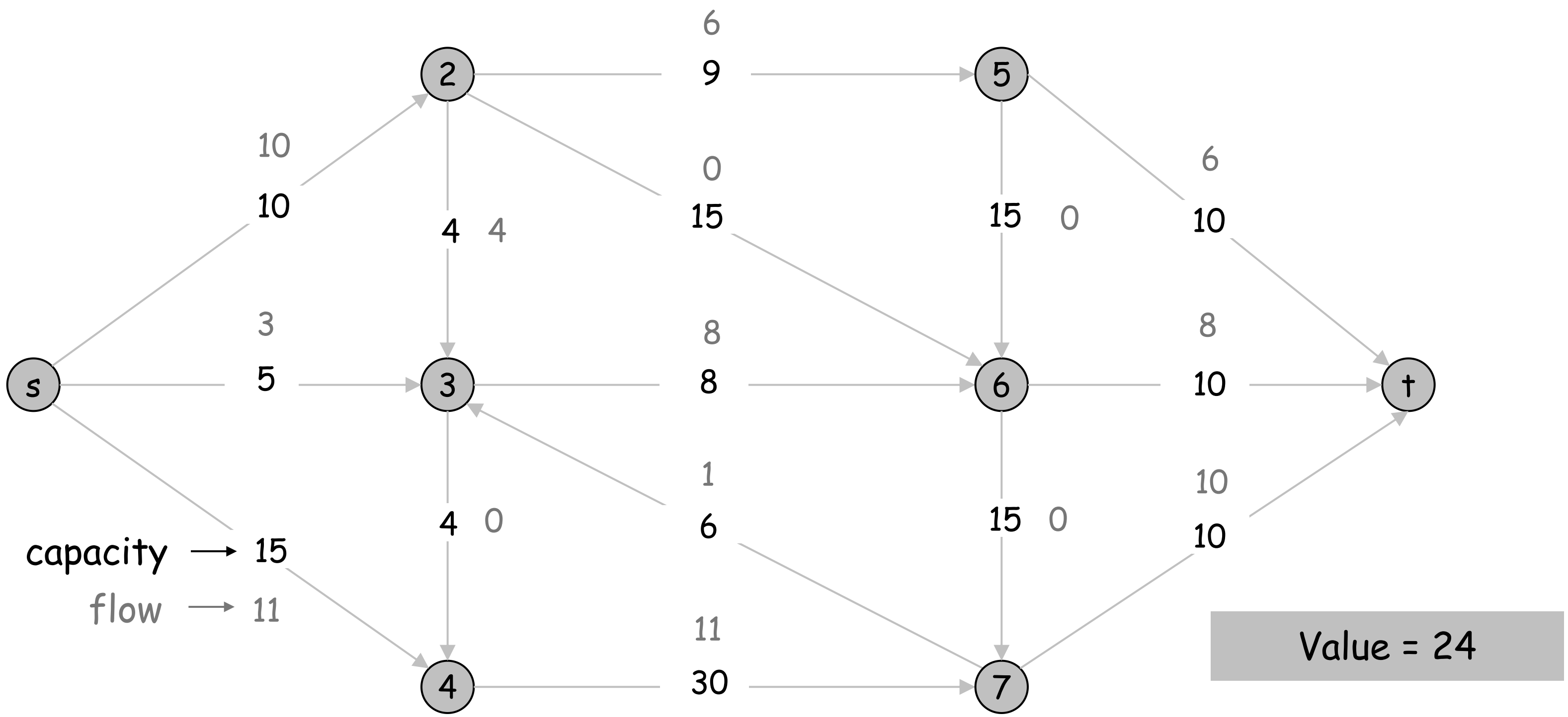


Flows

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

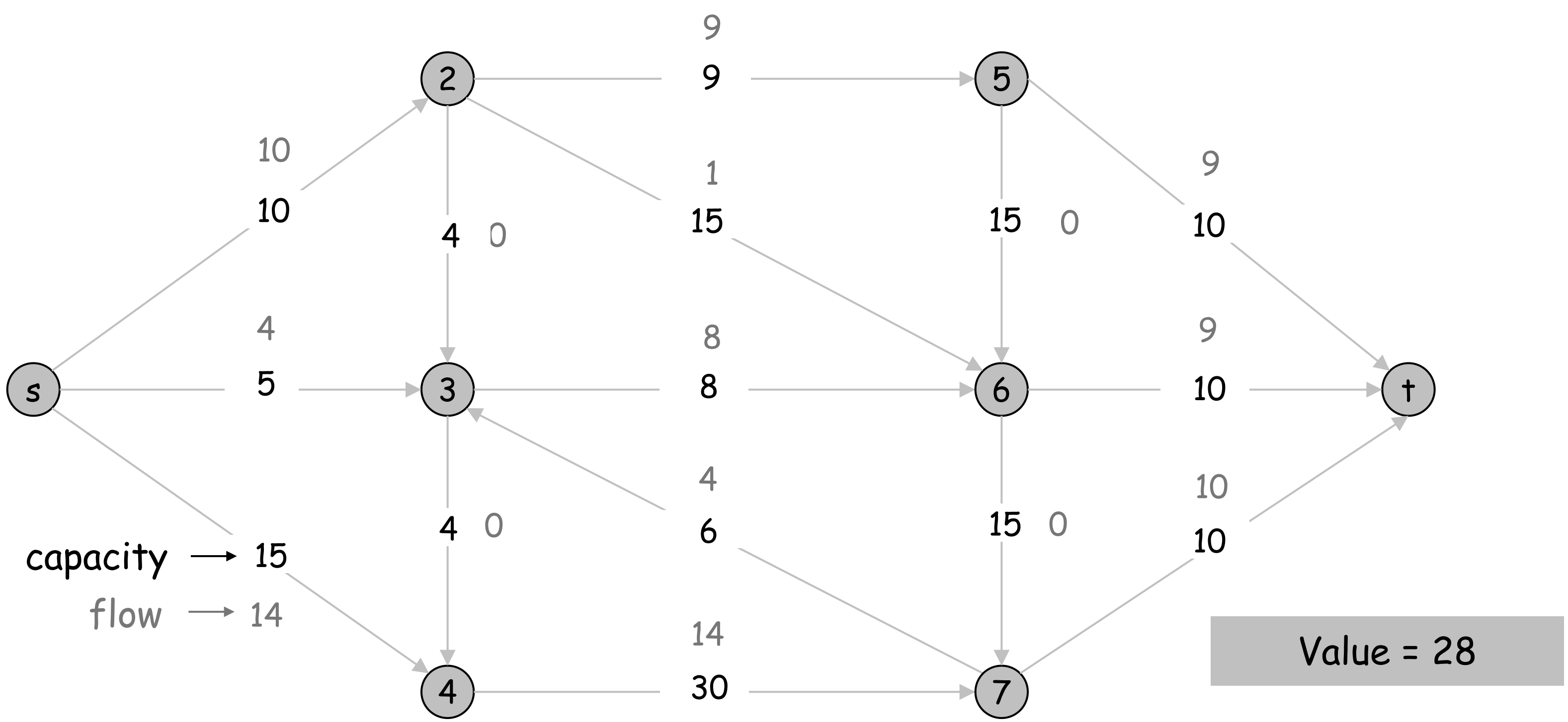
- For each e ∈ E: $0 \le f(e) \le c(e)$ [capacity]
- For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

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



Maximum Flow Problem

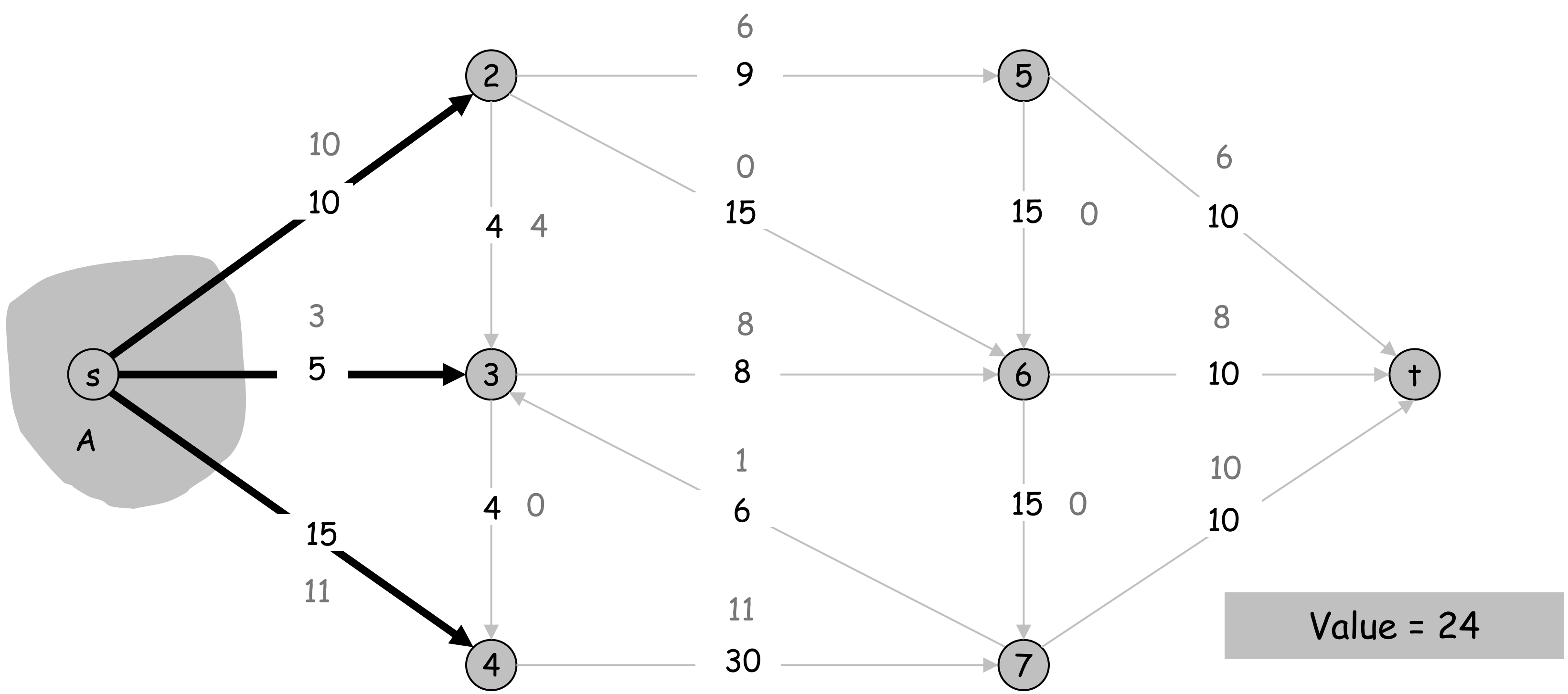
Max flow problem. Find s-t flow of maximum value.



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.

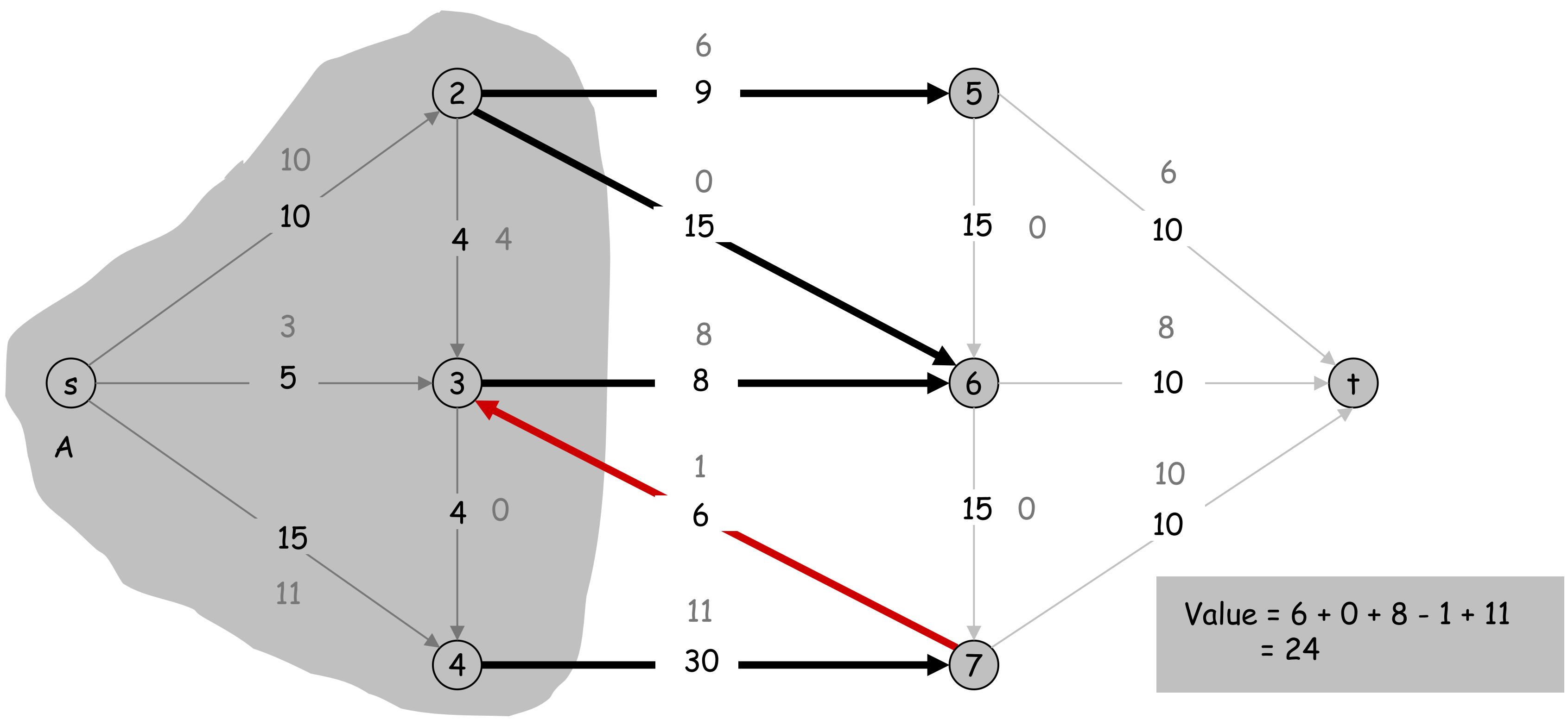
$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$



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.

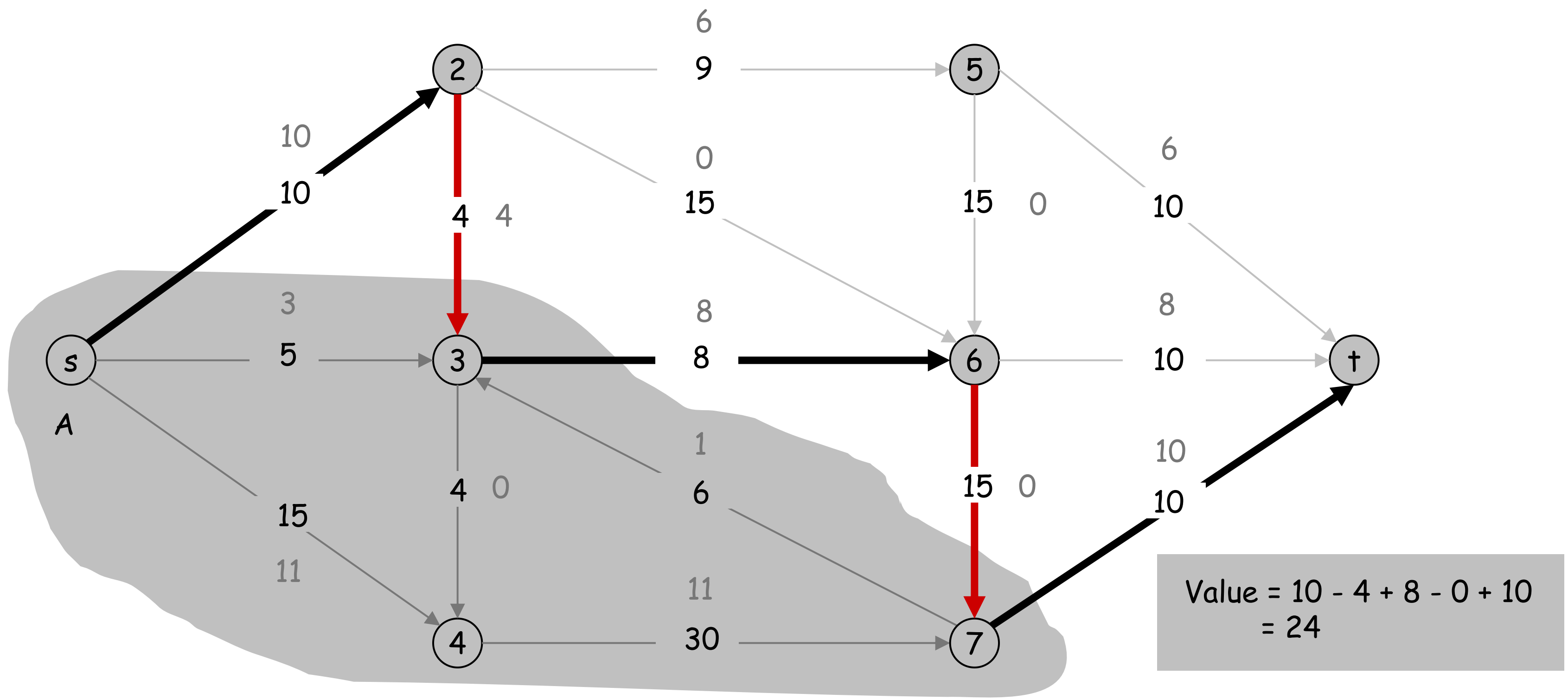
$$\sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;=\; v(f)$$



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.

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



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.

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

Pf.

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

by flow conservation, all terms except v = s are 0 $\longrightarrow$

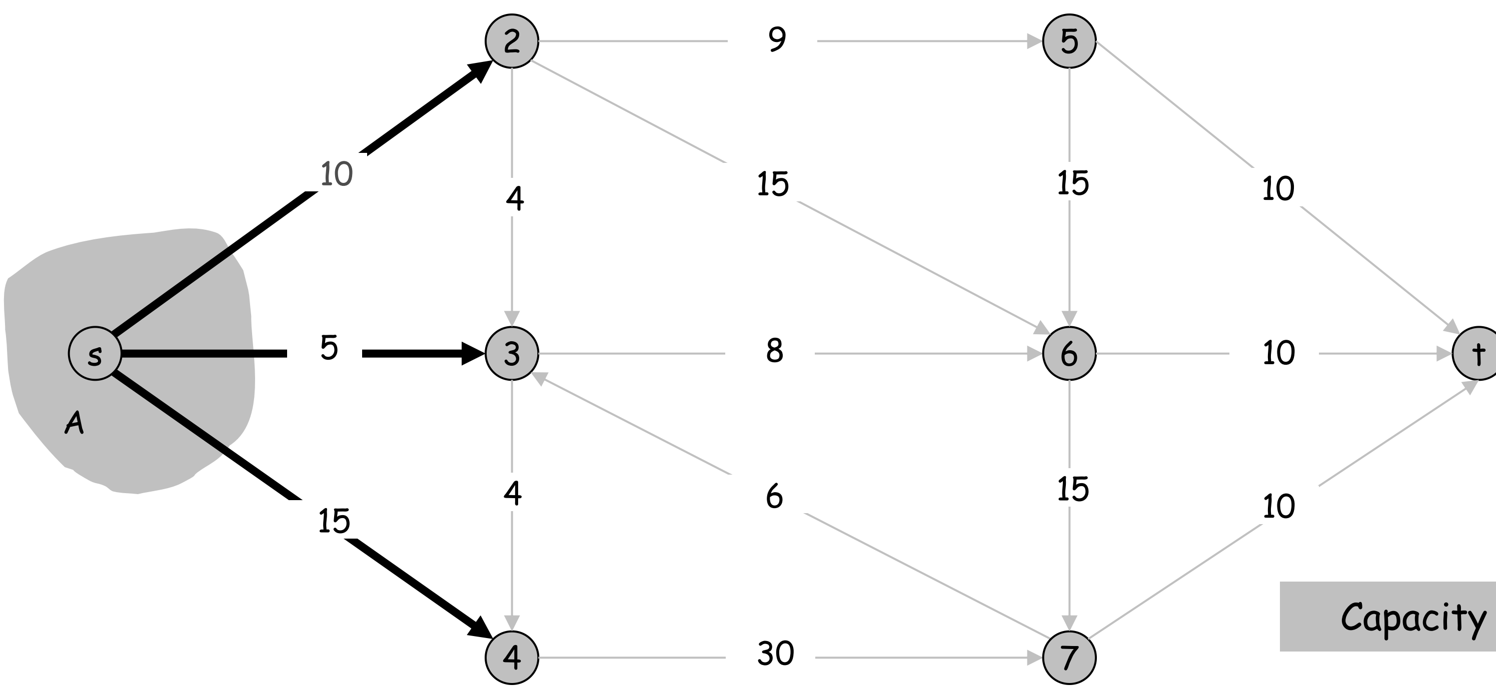
$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \right)$$

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

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

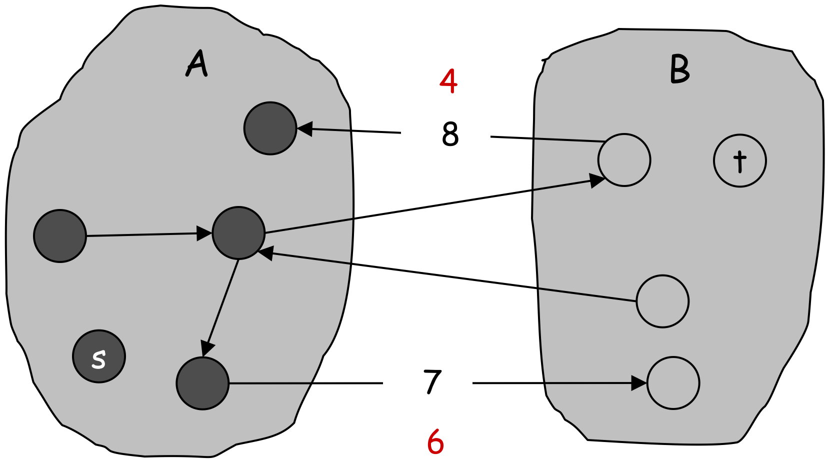


Capacity = 30

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

Pf.

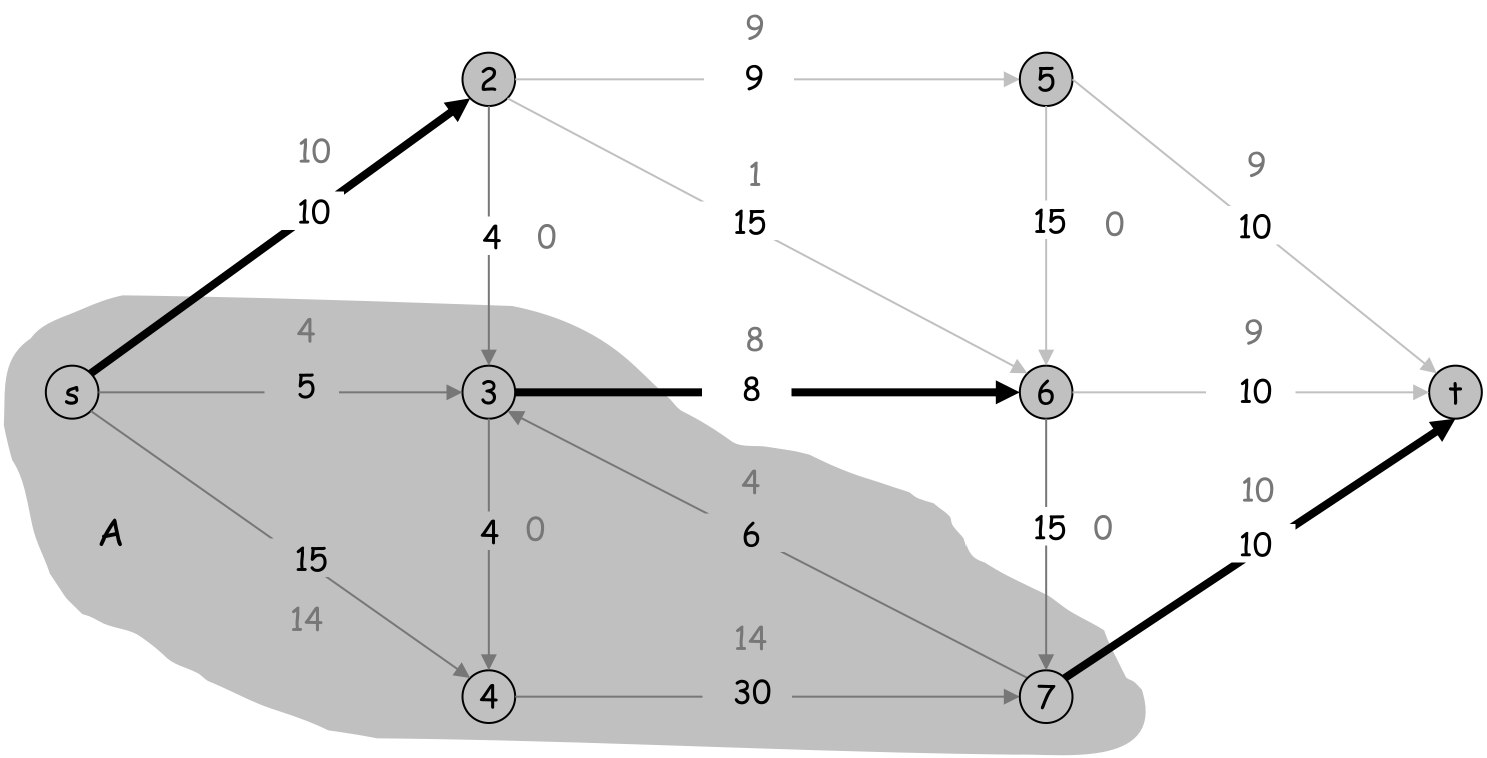
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\le \sum_{e \text{ out of } A} f(e) \\ &\le \sum_{e \text{ out of } A} c(e) \\ &= \text{cap}(A, B) \quad \blacksquare \end{aligned}$$



Certificate of Optimality

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

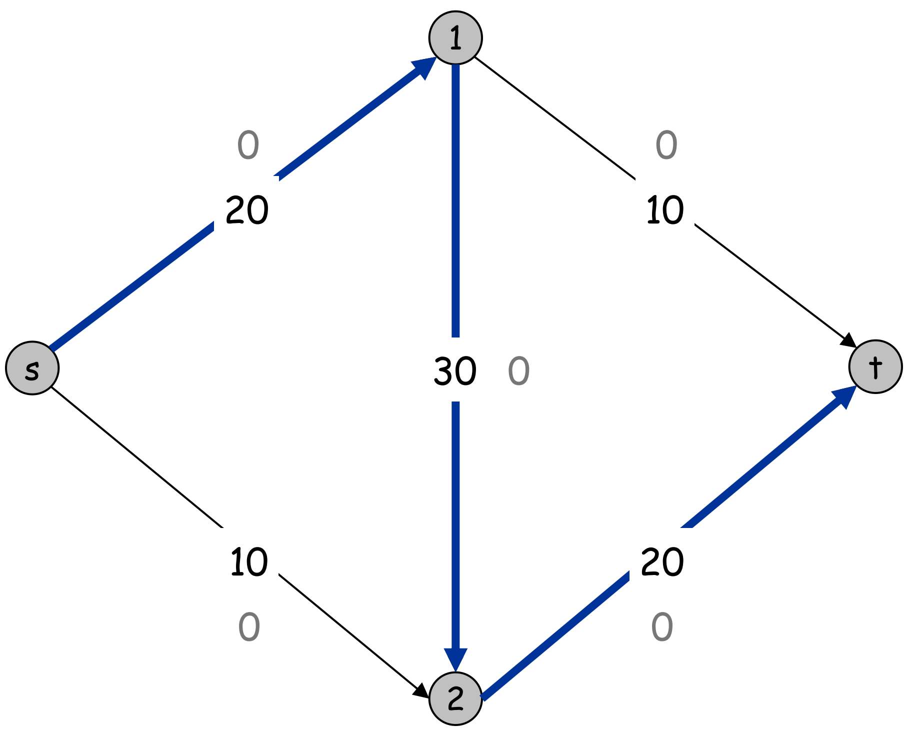
Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value $\leq$ 28



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.

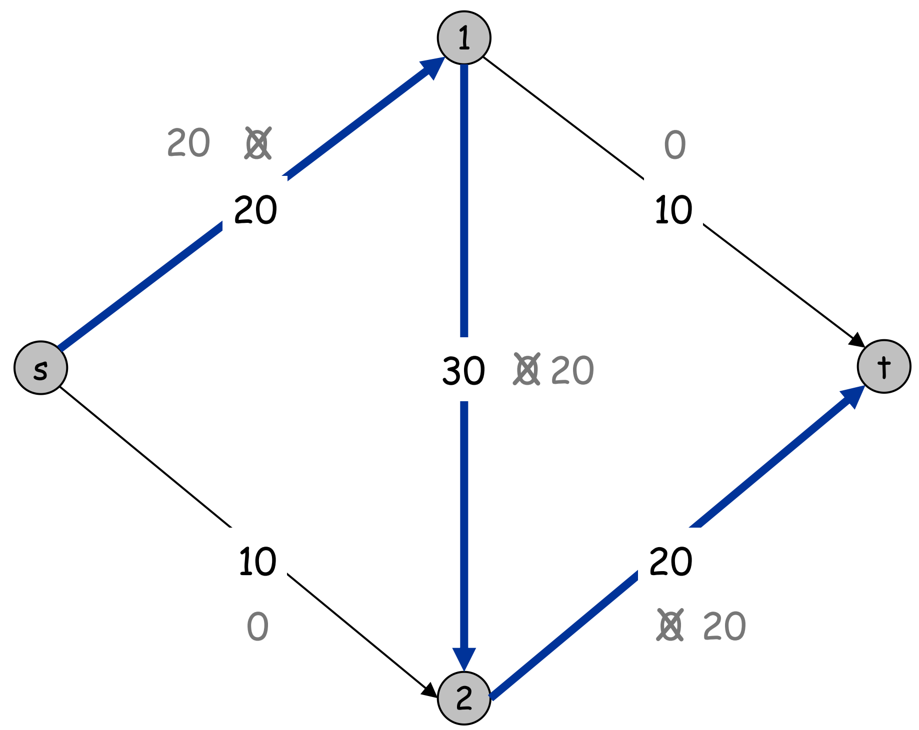


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

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



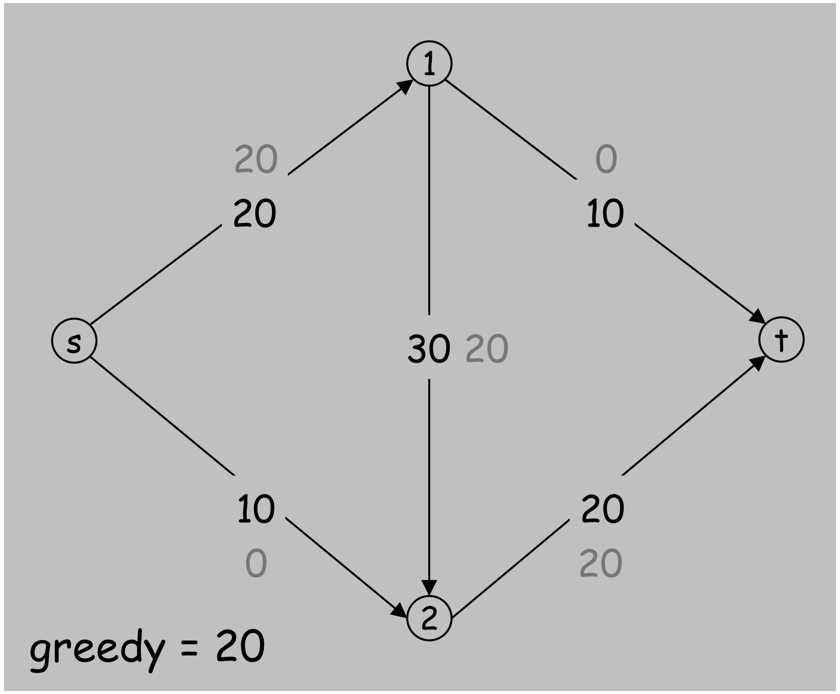
Flow value = 20

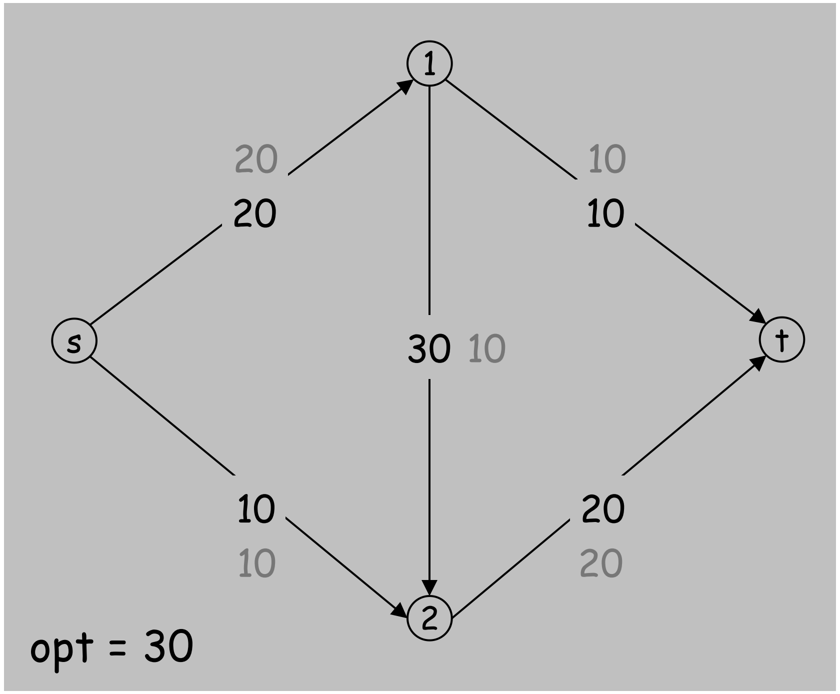
Towards a Max Flow Algorithm

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 $\not\Rightarrow$ global optimality

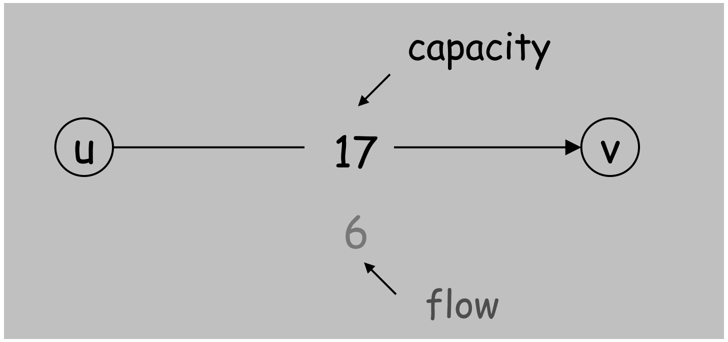




Residual Graph

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

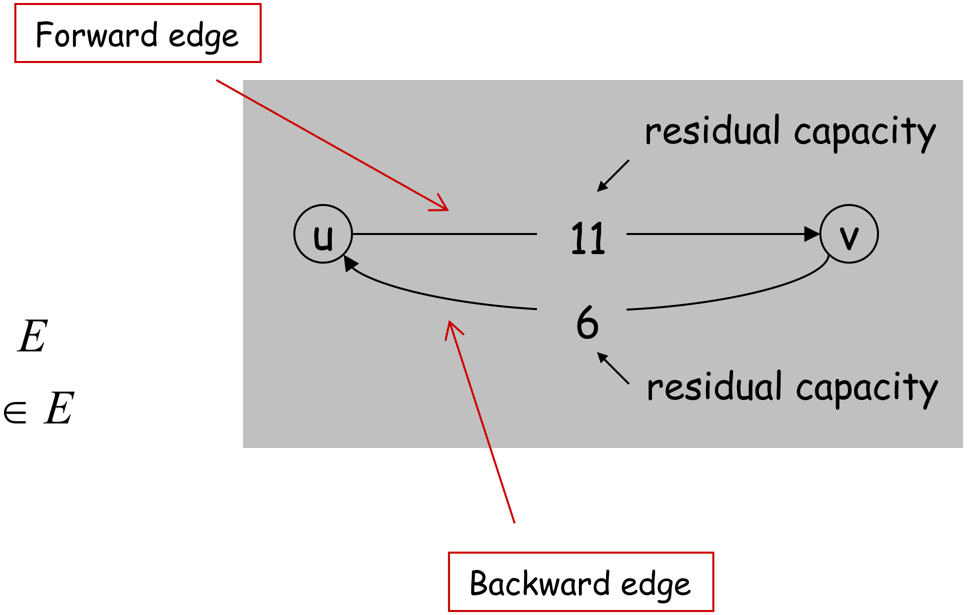
- Flow $f(e)$, capacity $c(e)$.



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

Augmenting path

Def. An augmenting path is a simple s->t path in the residual graph G_f

Def. The bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P.

Key property. Let f be a flow and let P be an augmenting path in G_f, then after calling $f' \leftarrow$ Augment(f,c,P), the resulting f' is flow and

$$v(f') = v(f) + bottleneck(G_f, P)$$

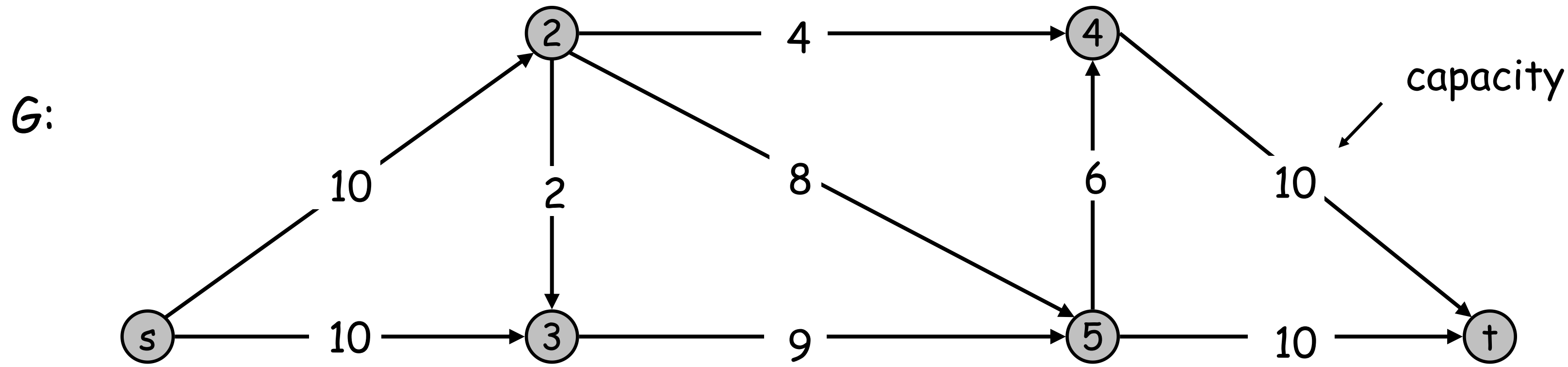
Augmenting Path Algorithm

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

```
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
}
```

Ford-Fulkerson Algorithm



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. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956]
The value of the max flow is equal to the value of the min cut.

Pf. We prove both simultaneously by showing TFAE (the following are equivalent) :

(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) $\Rightarrow$ (ii) This was the corollary to weak duality lemma. (Slide 17)

(ii) $\Rightarrow$ (iii) We show contrapositive.

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

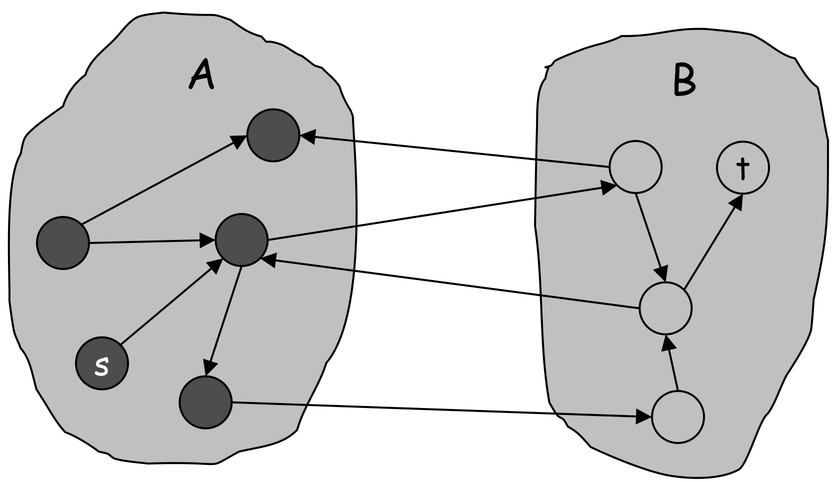
(iii) $\Rightarrow$ (i)

- 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, $t \notin A$.

f(e) = 0, if not, there will be a backward edge in G_f ,
Violate no augmenting paths in G_f

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

If not, there will be a forward edge in G_f ,
Violate no augmenting paths in G_f



original network

Running Time

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

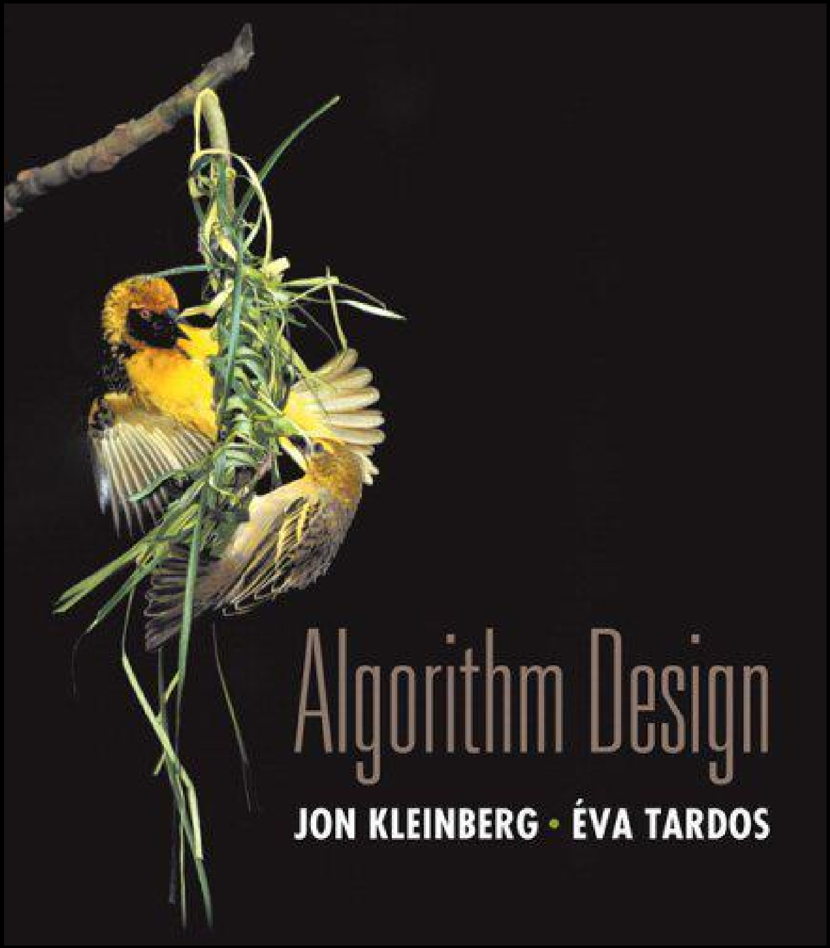
Invariant. Every flow value f(e) and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.

Theorem. The algorithm terminates in at most $v(f^*) \leq C$ iterations.
Pf. Each augmentation increase value by at least 1. ▪

To find an *s-t* path in G_f ,say by BFS,O(m+n) with $m \geq n/2$,
Procedure augment(f,P) takes O(n), as the path has at most n-1 edges

Corollary. If C = 1, Ford-Fulkerson runs in O(mn) 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. ▪



Chapter 7

Network Flow

7.3 Choosing Good Augmenting Paths

Choosing good augmenting paths

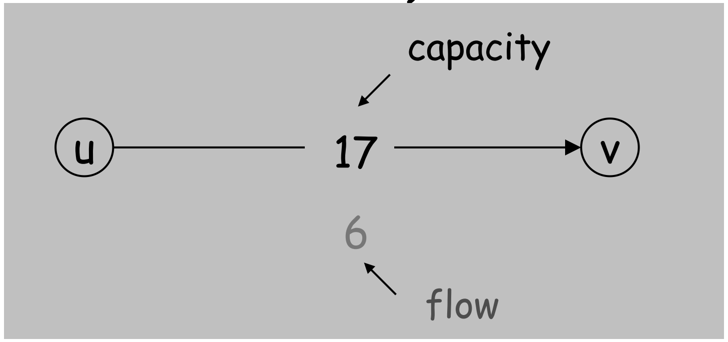
Use care when selecting augmenting paths

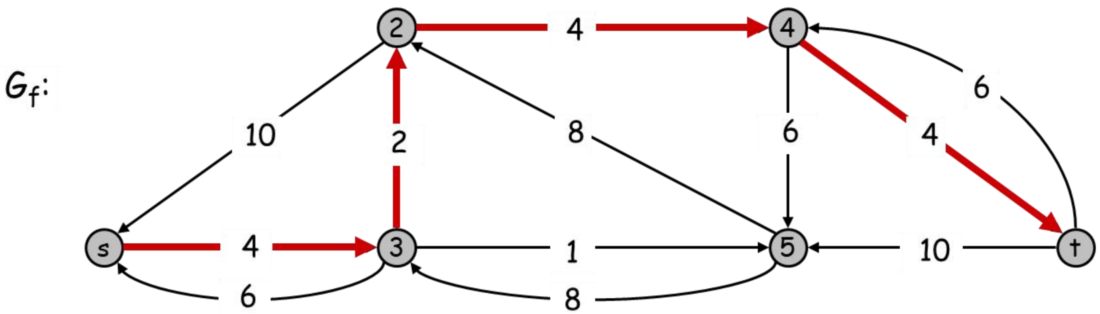
- Some choices lead to exponential algorithms
- Clever choice lead to polynomial algorithms

Pathology. When edge capacities can be irrational, no guarantee that Ford-Fulkerson terminates (or converges to a maximum flow)!

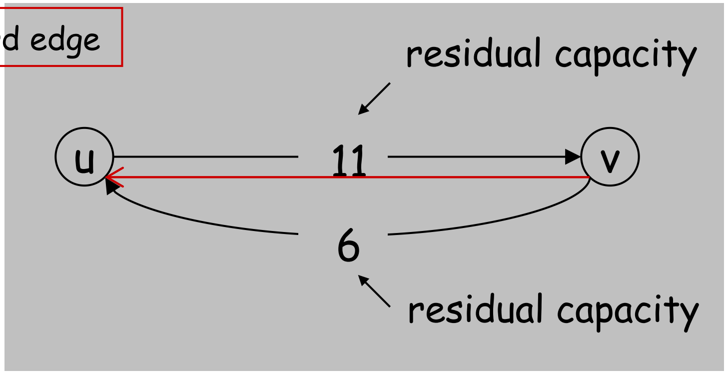
Goal. Choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations





Forward edge



Backward edge

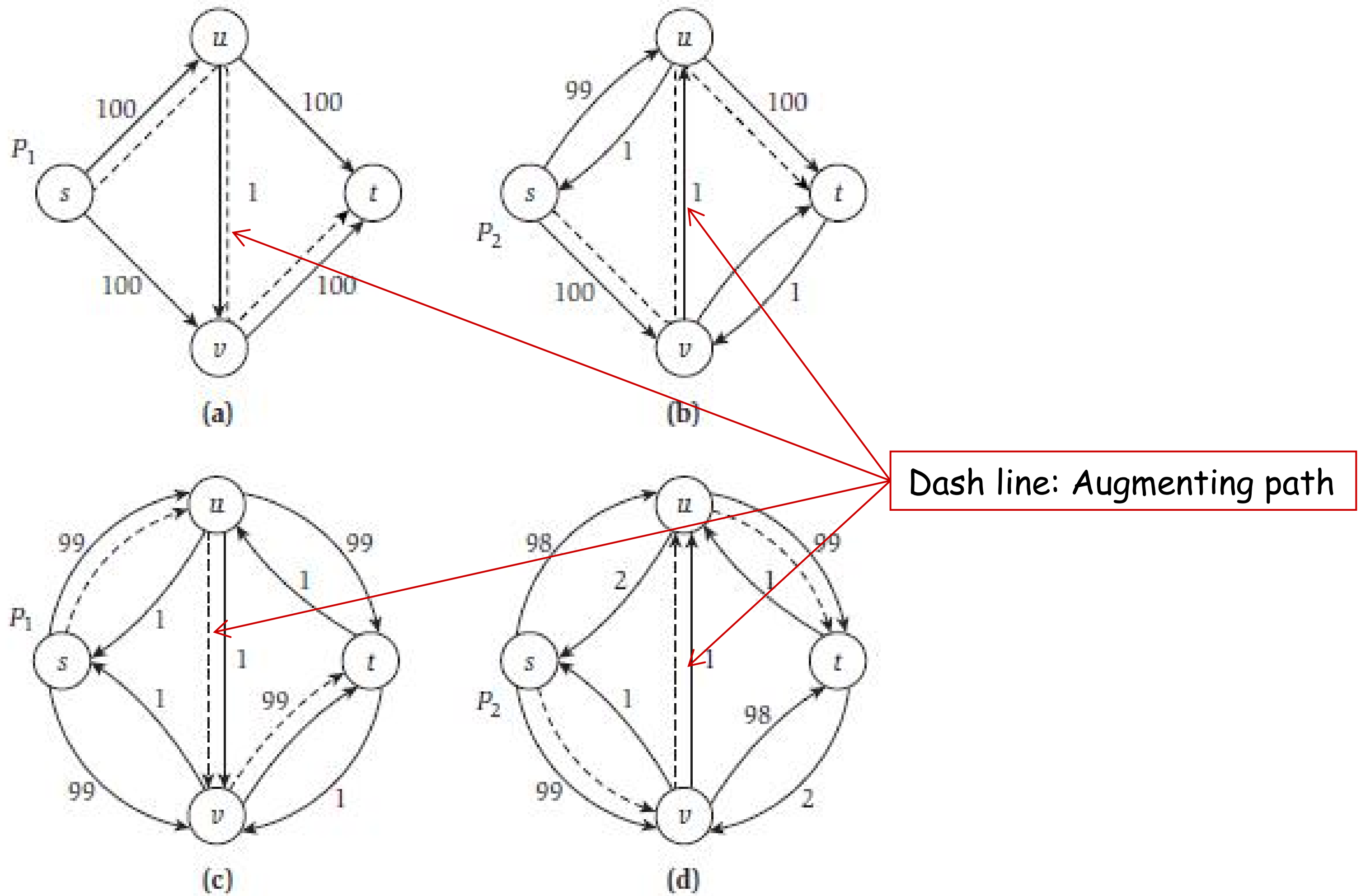


Figure Parts (a) through (d) depict four iterations of the Ford-Fulkerson Algorithm using a bad choice of augmenting paths: The augmentations alternate between the path P_1 through the nodes s, u, v, t in order and the path P_2 through the nodes s, v, u, t in order.

Choosing good augmenting paths

Choose augmenting paths with:

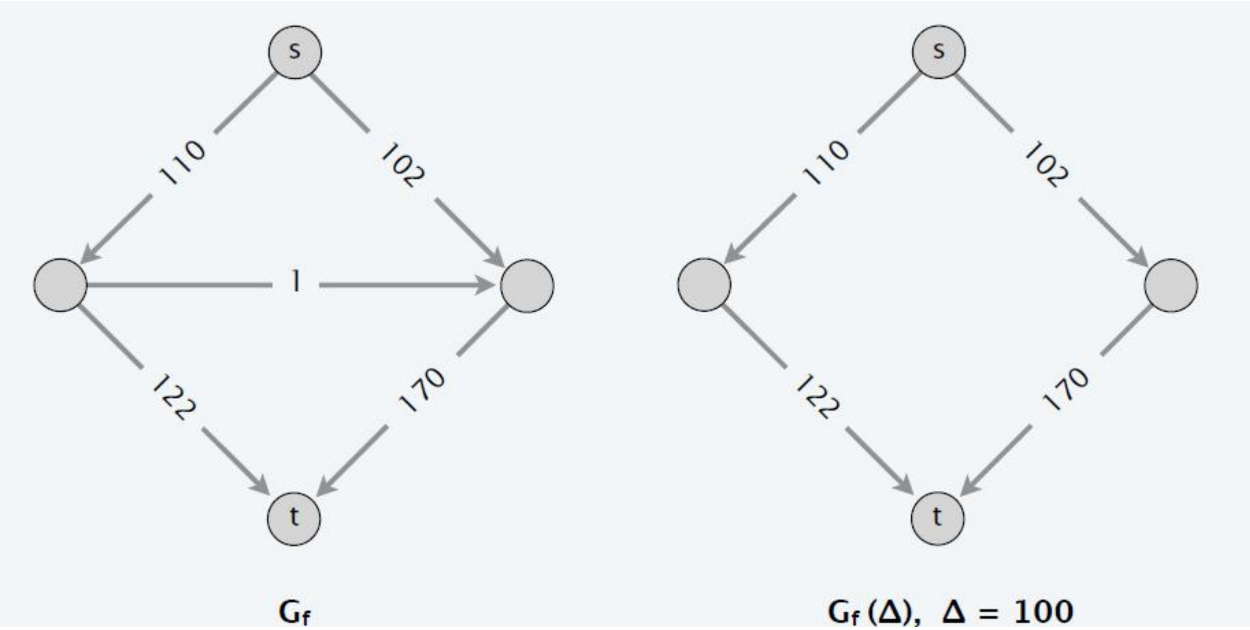
- Max bottleneck capacity ("fattest"). ← how to find?
- Sufficiently large bottleneck capacity. ← next
- Fewest edges. ← ahead

Capacity-scaling algorithm

Overview. Choosing augmenting paths with "large" bottleneck capacity.

though not necessarily largest

- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the part of the residual graph containing only those edges with capacity $\geq \Delta$.
- Any augmenting path in $G_f(\Delta)$ has bottleneck capacity $\geq \Delta$.



```
Scaling Max-Flow
  Initially f(e) = 0 for all e in G
  Initially set Δ to be the largest power of 2 that is no larger
       than the maximum capacity out of s: Δ ≤ max_{e out of s} c_e
    While Δ ≥ 1
       While there is an s-t path in the graph G_f(Δ)
          Let P be a simple s-t path in G_f(Δ)
          f' = augment(f, P)
          Update f to be f' and update G_f(Δ)
       Endwhile
       Δ = Δ/2
    Endwhile
Return f
```

Capacity-scaling algorithm: proof of correctness

Assumption: All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.
Pf. Initially a power of 2 (largest power of 2 $\leq C$); each phase divides Δ by exactly 2.

Integrality invariant. Throughout the algorithm, every edge flow f(e) and residual capacity $c_f(e)$ is an integer.
Pf. Same as for genetic Ford-Fulkerson.

Theorem. If capacity-scaling 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.
- Result follows augmenting path theorem.

Capacity-scaling algorithm: analysis of running time

Lemma 1. There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.

Pf. Initial $C/2 < \Delta \le C$; Δ decreases by a factor of 2 in each iteration.

Lemma 2. Let f be the flow at the end of a Δ-scaling phase, then the max-flow value $\le v(f) + m\Delta$.

Pf. Next slide.

Lemma 3. There are $\le 2m$ augmentations per scaling phase.

or equivalently, at the end of a 2 Δ-scaling phase

Pf.

- Let f be the flow at the beginning of a Δ-scaling phase.
- Lemma 2 ➔ max-flow value $\le v(f) + m(2\Delta)$.
- Each augmentation in a Δ-scaling phase increases $v(f)$ by at least Δ.

Theorem. The capacity-scaling algorithm takes $O(m^2 \log C)$ time.

Pf.

- Lemma 1 + Lemma 3 ➔ $O(m \log C)$ augmentations.
- Finding an augmenting path takes $O(m)$ time.

Capacity-scaling algorithm: analysis of running time

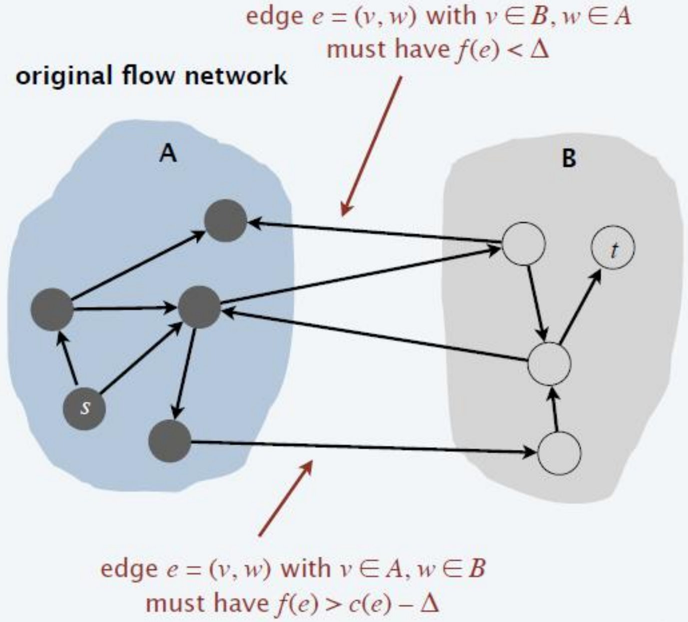
Lemma 2. Let f be the flow at the end of a Δ-scaling phase, then the max-flow value ≤ v(f) + mΔ.

Pf.

- We show there exists a cut(A,B) such that cap(A,B) ≤ v(f) + m Δ.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of flow f: $t \notin A$.

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

flow value lemma



Residual capacity:

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

Shortest augmenting path

Q. How to choose next augmenting path in Ford-Fulkerson?
A. Pick one that uses the fewest edges.

can find via BFS

SHORTEST-AUGMENTING-PATH(G)

FOREACH $e \in E$: $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of G with respect to flow f.

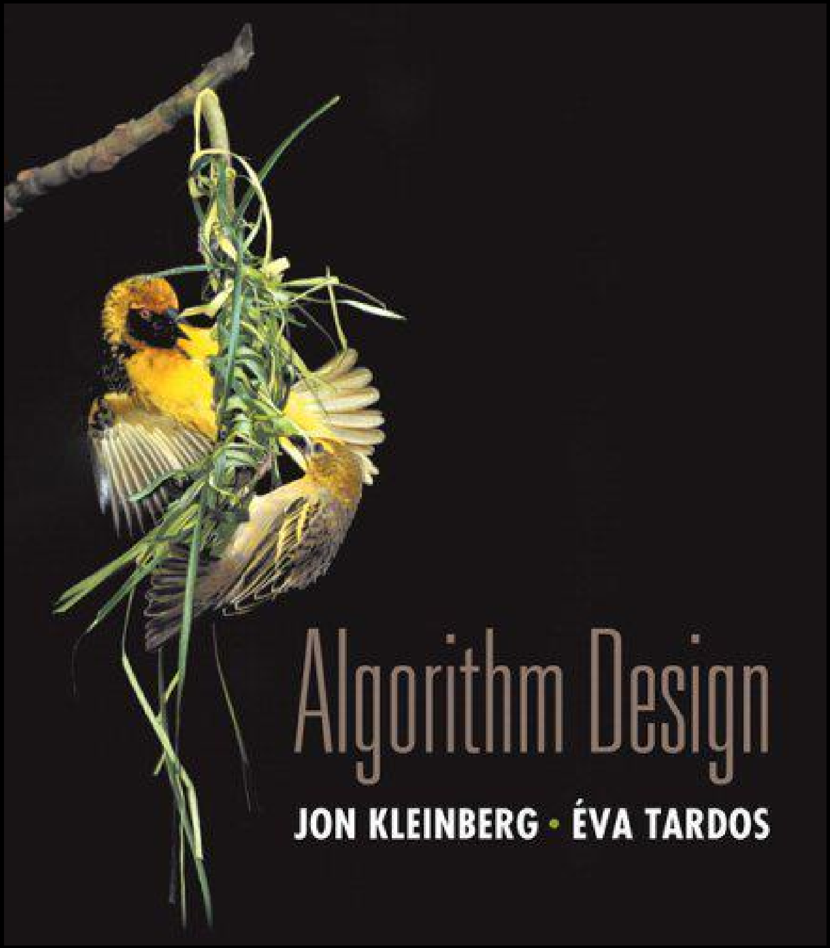
WHILE (there exists an $s \to t$ path in G_f)

$P \leftarrow$ BREADTH-FIRST-SEARCH(G_f).

$f \leftarrow$ AUGMENT(f, c, P).

Update G_f.

RETURN f.



Chapter 7

Network Flow

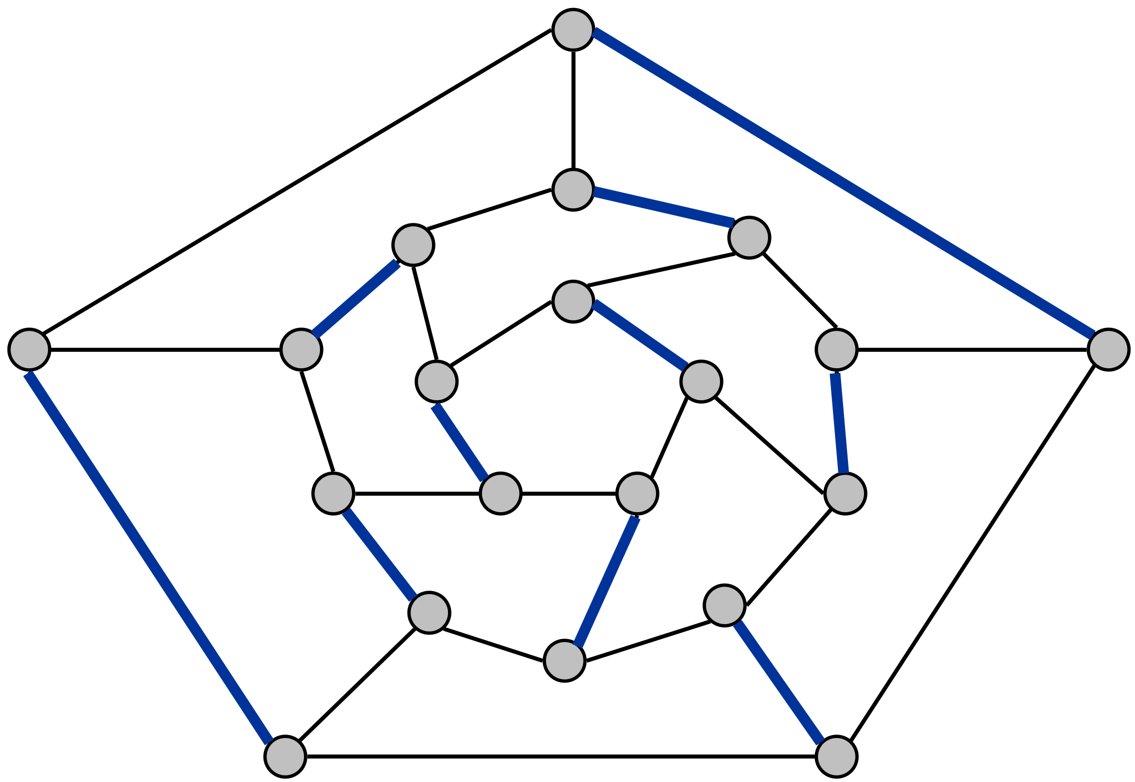
Slides by Kevin Wayne.

7.5 Bipartite Matching

Matching

Matching.

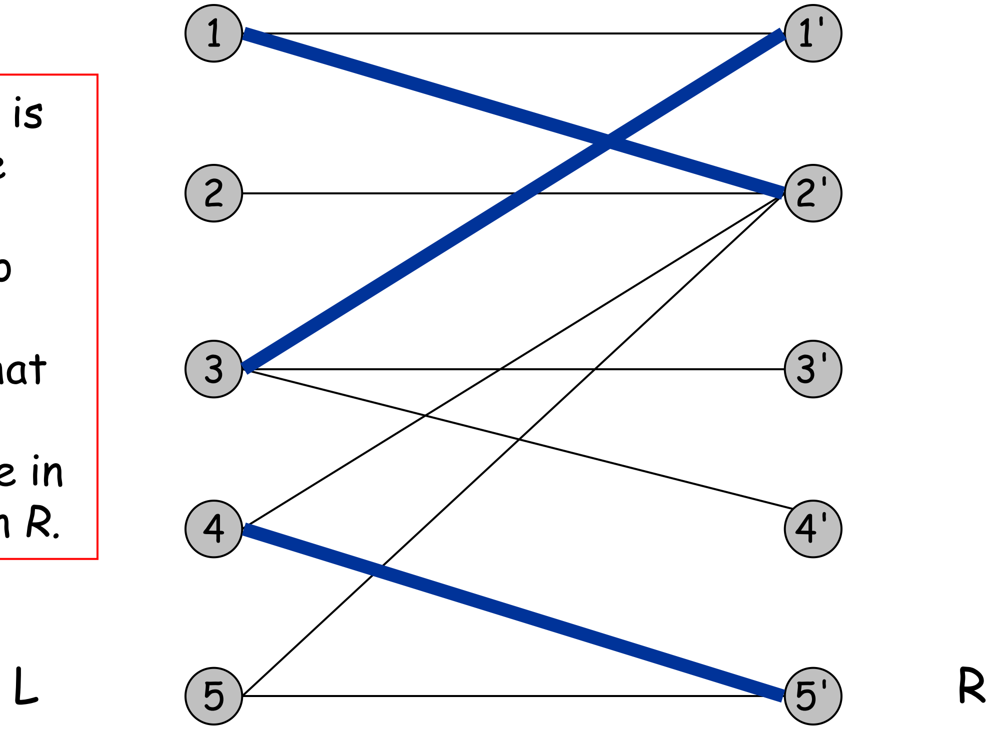
- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in M.
- Max matching: find a max cardinality matching.



Bipartite matching.

- 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.

Def. A graph G is bipartite if the nodes can be partitioned into two subsets L and R such that every edge connects a node in L with a node in R.

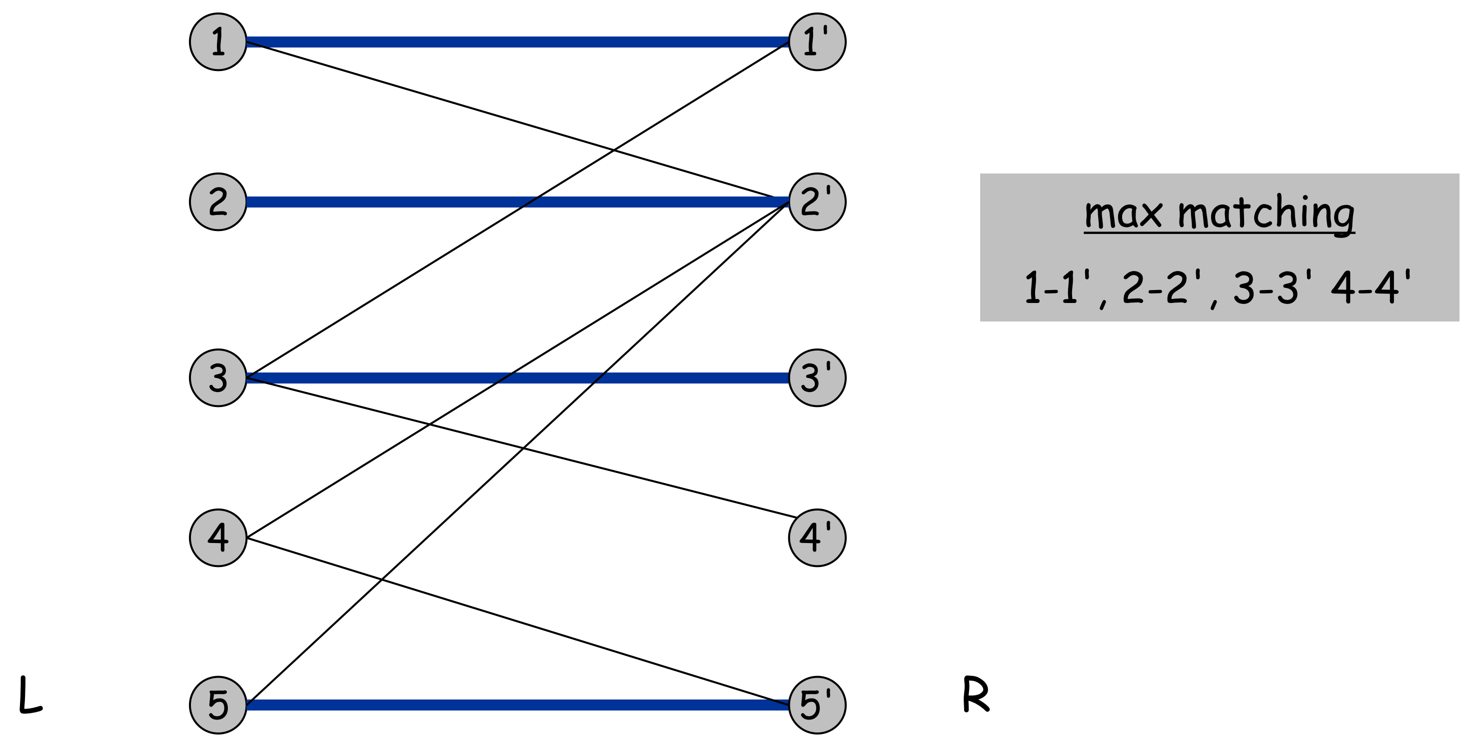


matching

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

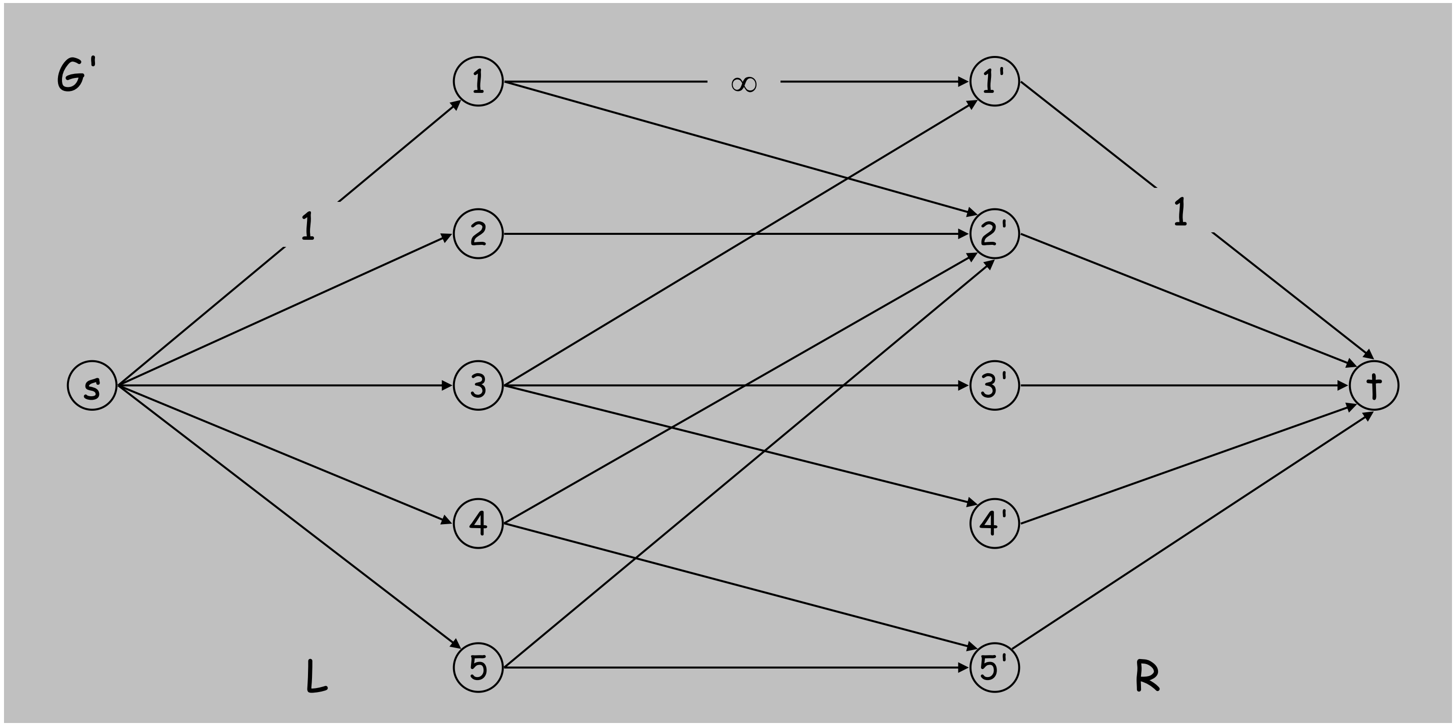
Bipartite matching.

- 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 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.

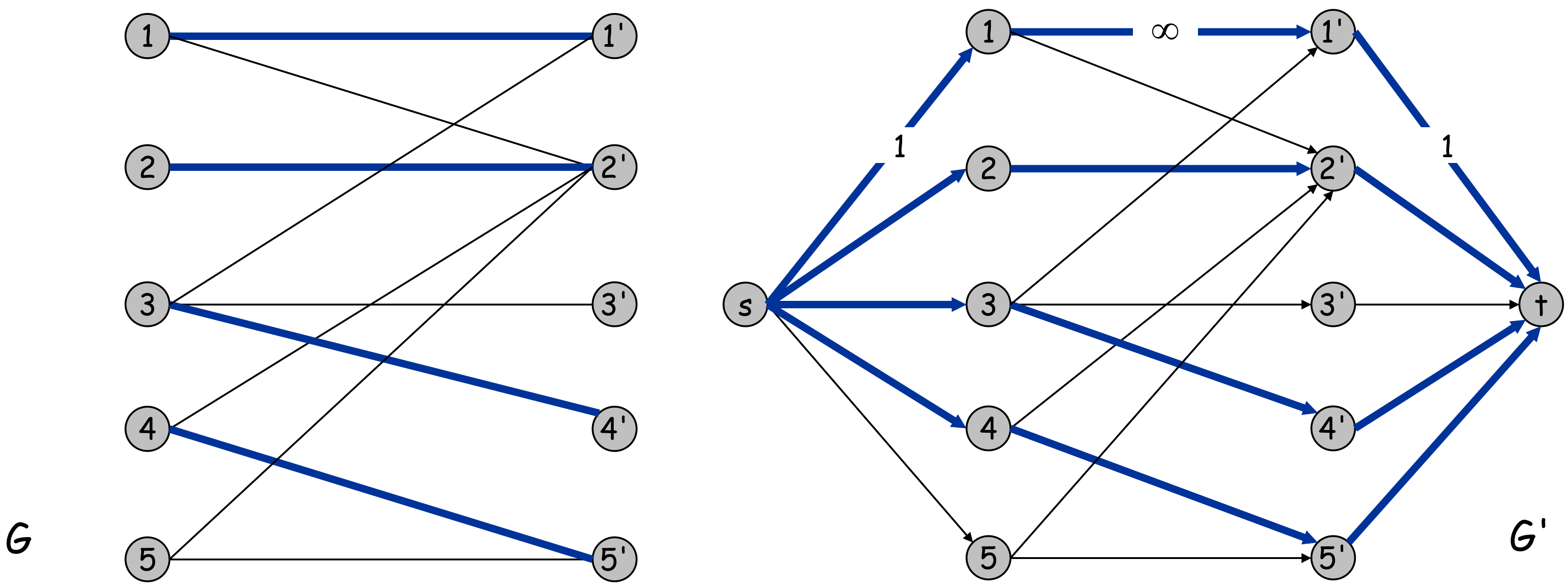


Bipartite Matching: Proof of Correctness

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



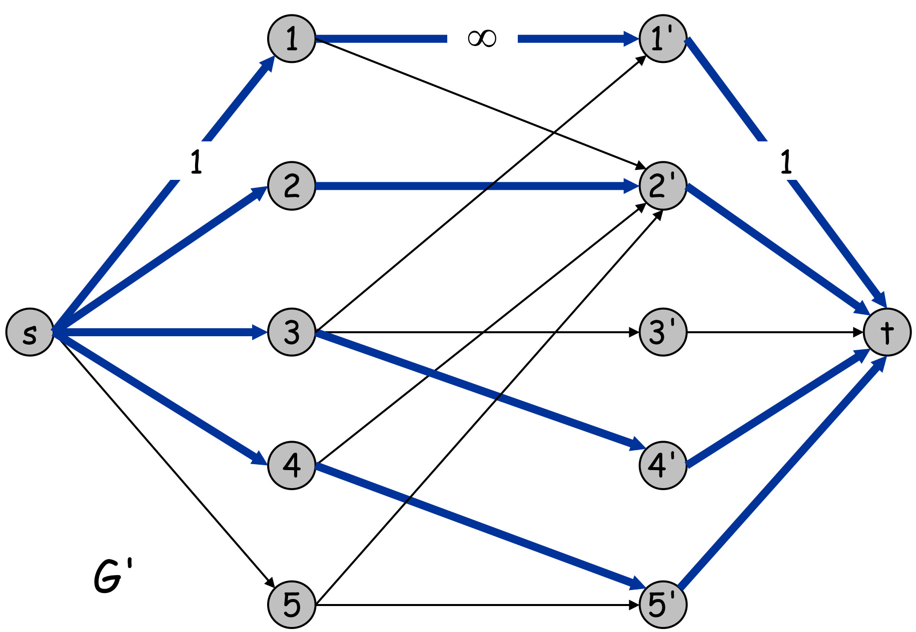
Bipartite Matching: Proof of Correctness

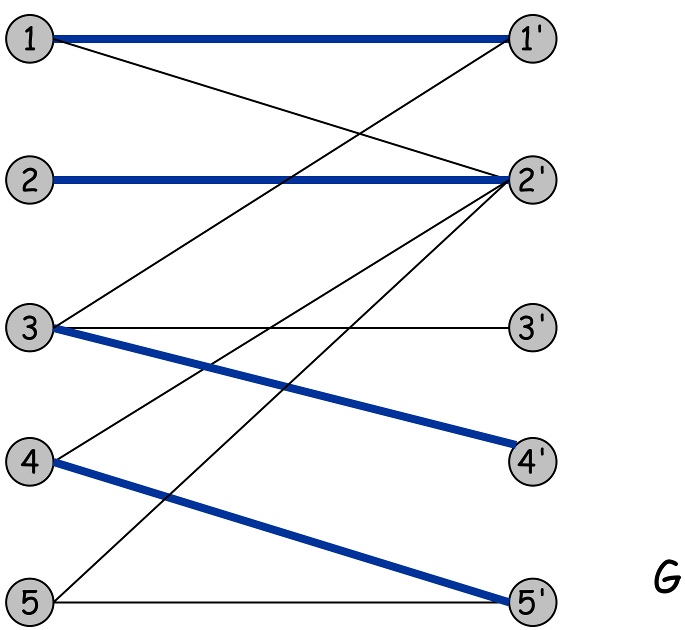
Theorem. value of max flow in G' = Max cardinality matching in G .

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: apply flow-value lemma to cut ($L \cup s$, $R \cup t$) ▪

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.





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?

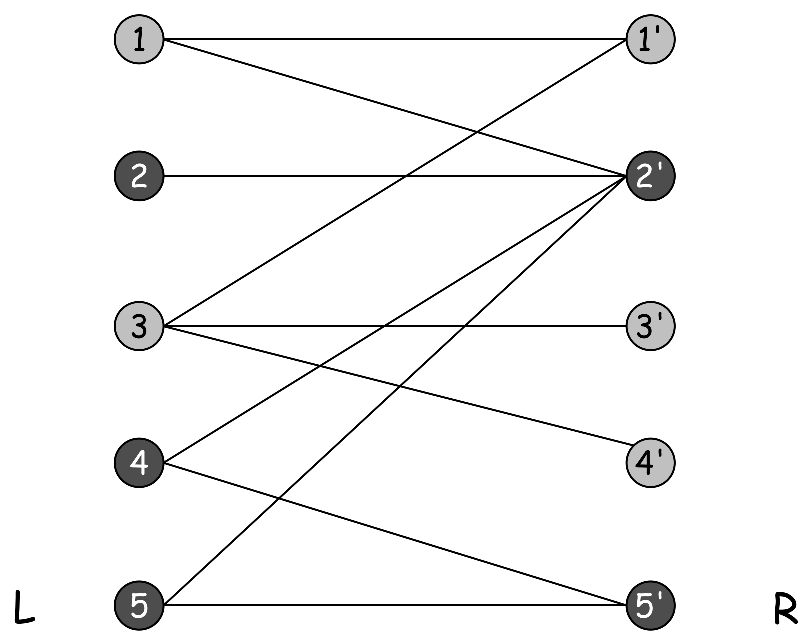
Structure of bipartite graphs with perfect matchings.

- 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).

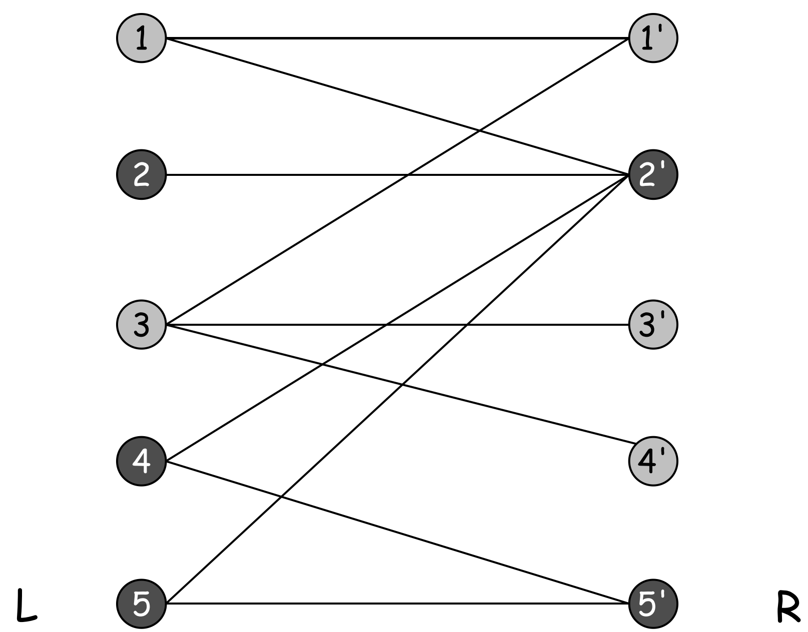


<u>No perfect matching:</u>
$S = \{ 2, 4, 5 \}$
$N(S) = \{ 2', 5' \}$.

Hall's marriage theorem

Theorem. [Frobenius 1917, Hall 1935] Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then, graph 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:

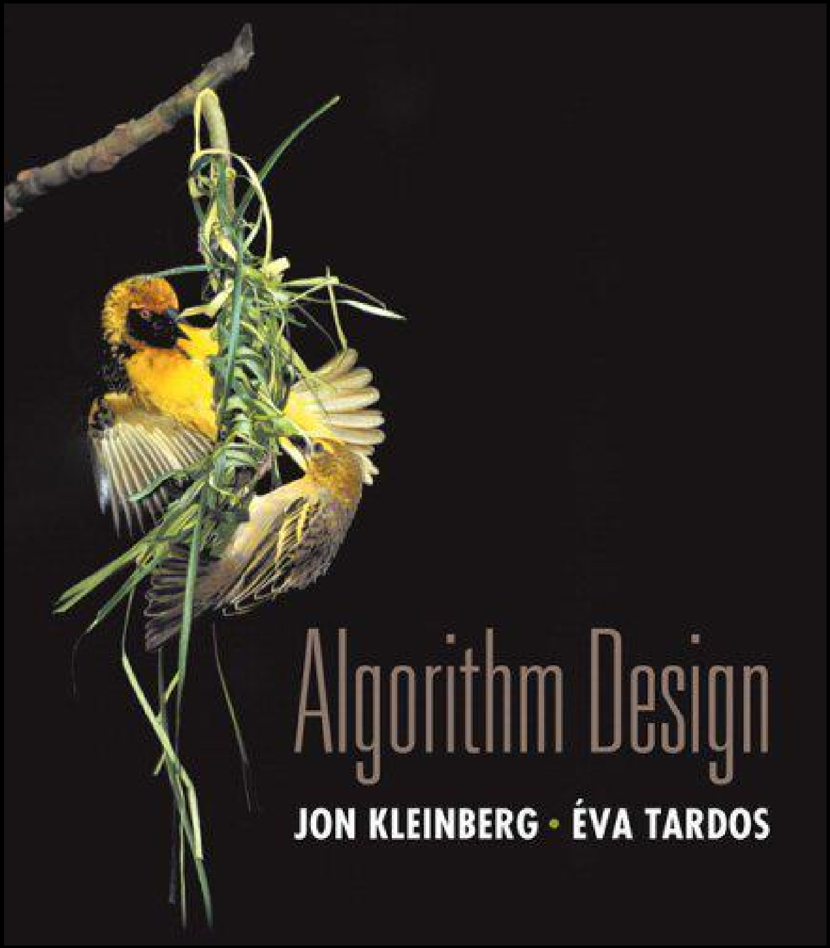
$S = \{ 2, 4, 5 \}$

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

Bipartite Matching: Running Time

Which max flow algorithm to use for bipartite matching?
- Generic augmenting path: $O(mn \text{ val}(f^*)) = O(mnC)$.
- Capacity scaling: $O(m^2 \log C)$.

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]



Chapter 7

Network Flow

Slides by Kevin Wayne.

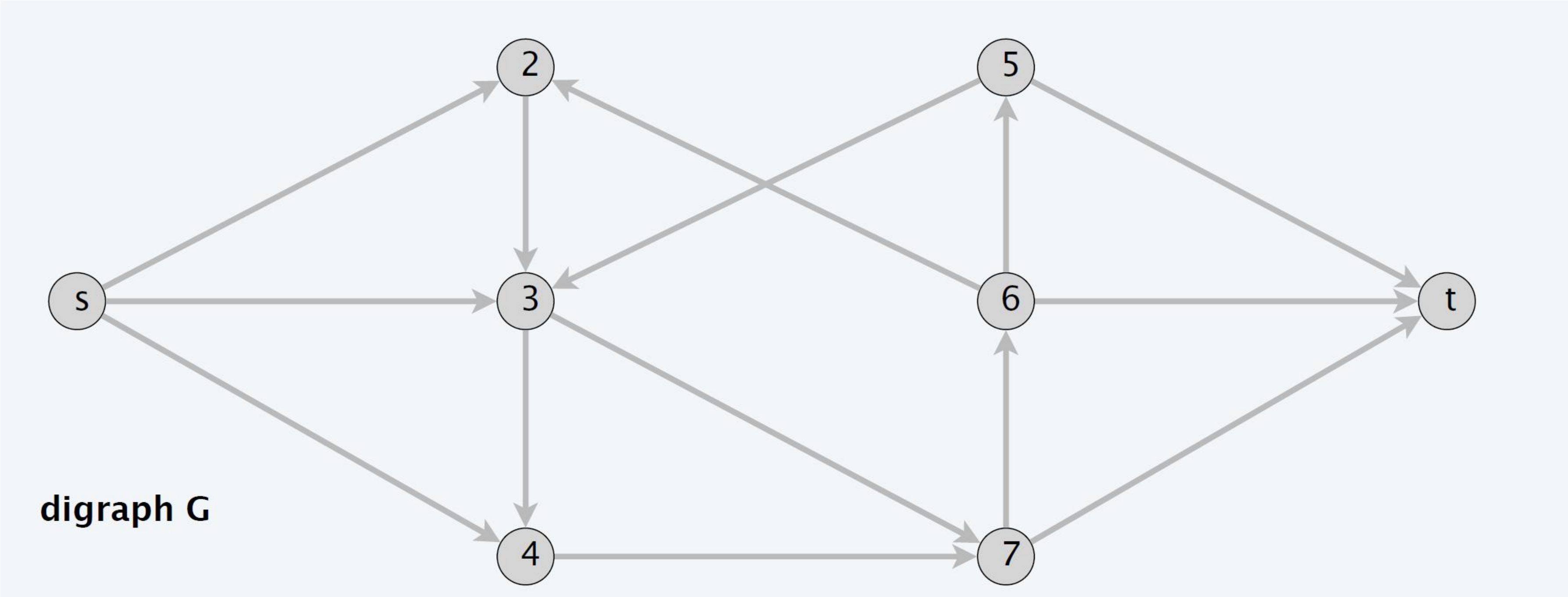
7.6 Disjoint Paths in Directed and Undirected Graphs

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightsquigarrow t$ paths.

Ex. Communication networks.

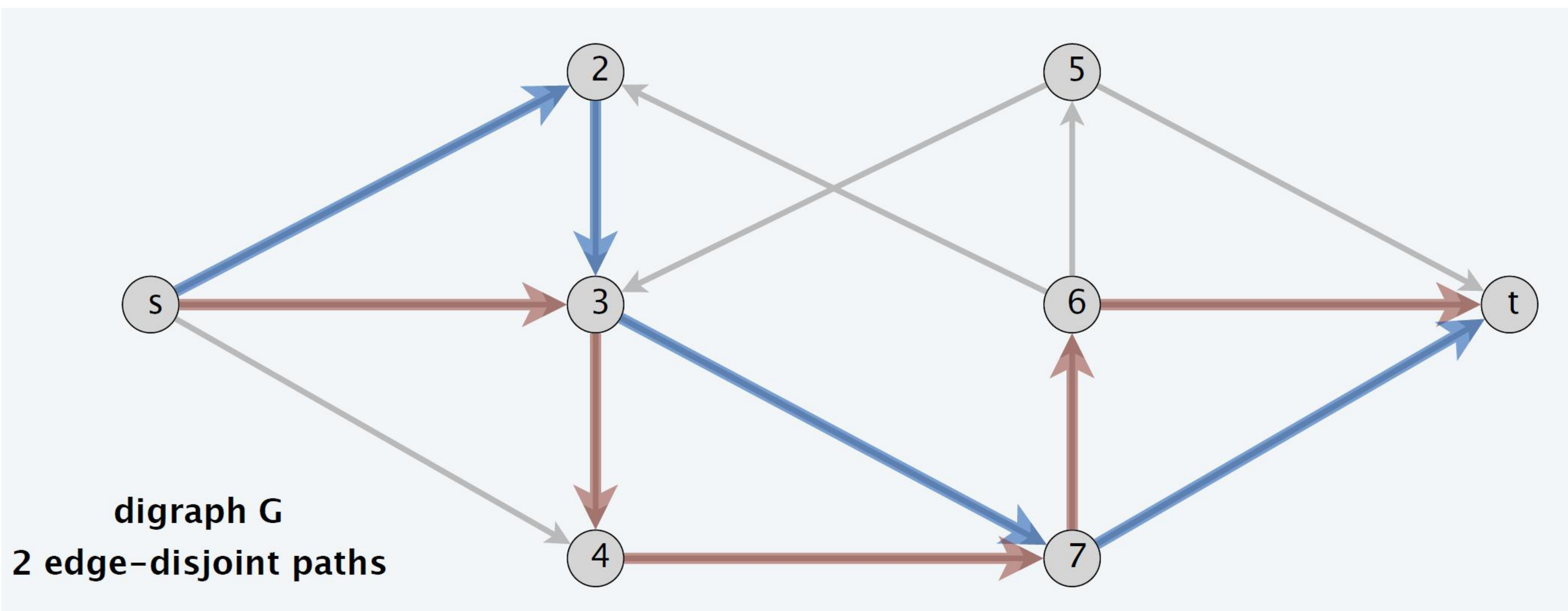


Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightsquigarrow t$ paths.

Ex. Communication networks.



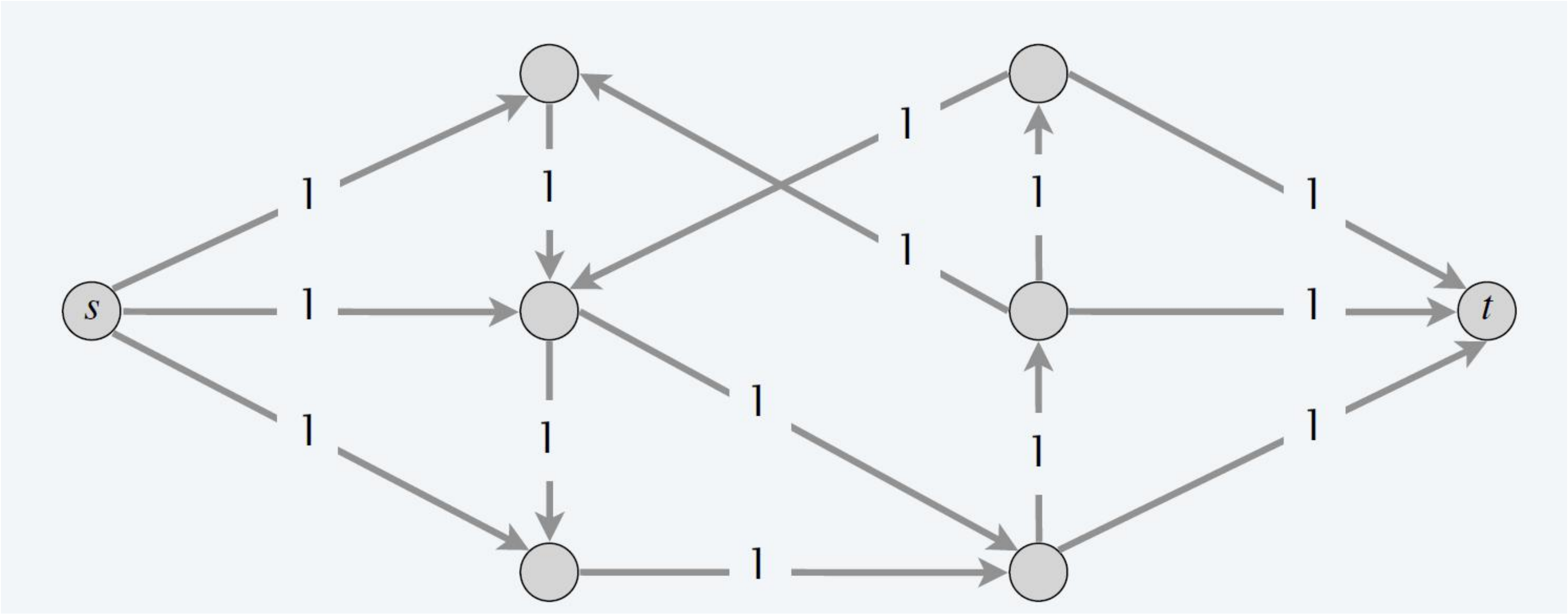
Edge-disjoint paths

Max-flow formulation. Assign unit capacity to every edge.

Theorem. Max number of edge-disjoint $s \rightsquigarrow t$ paths = value of max flow.

Pf. $\geq$

- Suppose there are k edge-disjoint $s \rightsquigarrow t$ paths $P1, \ldots, Pk$.
- Set $f(e) = 1$ if e participates in some path Pj ; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k. ▪

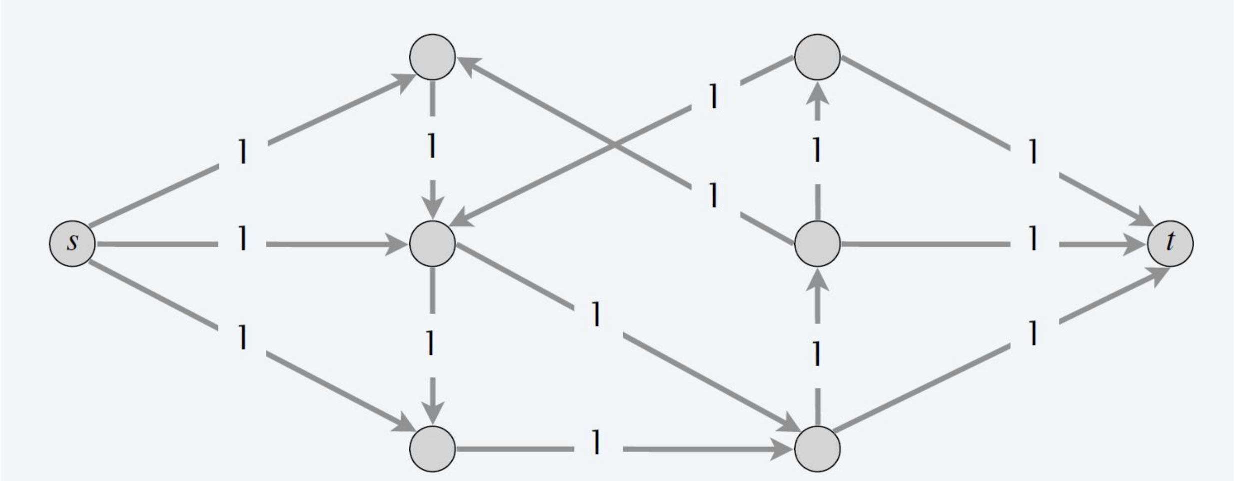


Edge-disjoint paths

Max-flow formulation. Assign unit capacity to every edge.

Theorem. Max number of edge-disjoint $s \rightsquigarrow t$ paths = value of max flow.
Pf. $\leq$

- 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 flow conservation, there exists an edge (u, v) with $f(u, v) = 1$
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths

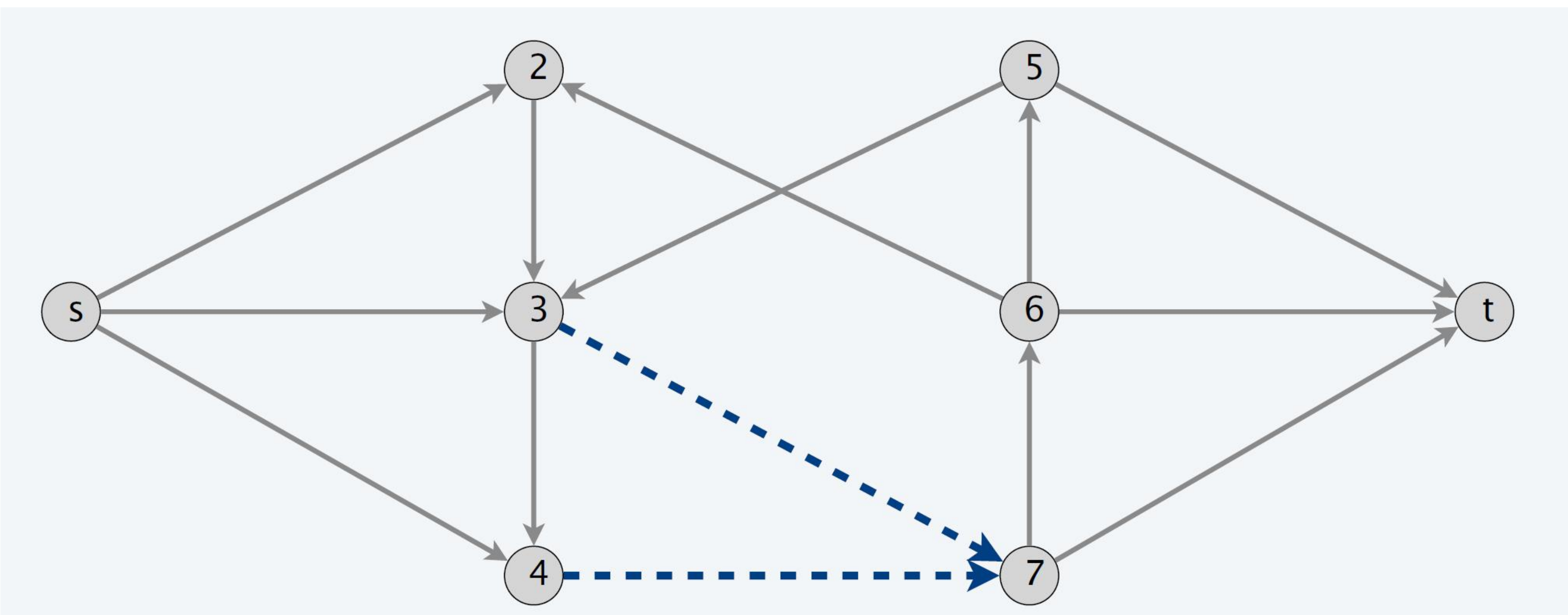


can eliminate cycles
to get simple paths
in $O(mn)$ time if desired
(flow decomposition)

Network connectivity

Def. A set of edges $F \subseteq E$ disconnects t from s if every $s \rightsquigarrow t$ path uses at least one edge in F.

Network connectivity. Given a digraph $G = (V, E)$ and two nodes s and t, find min number of edges whose removal disconnects t from s.

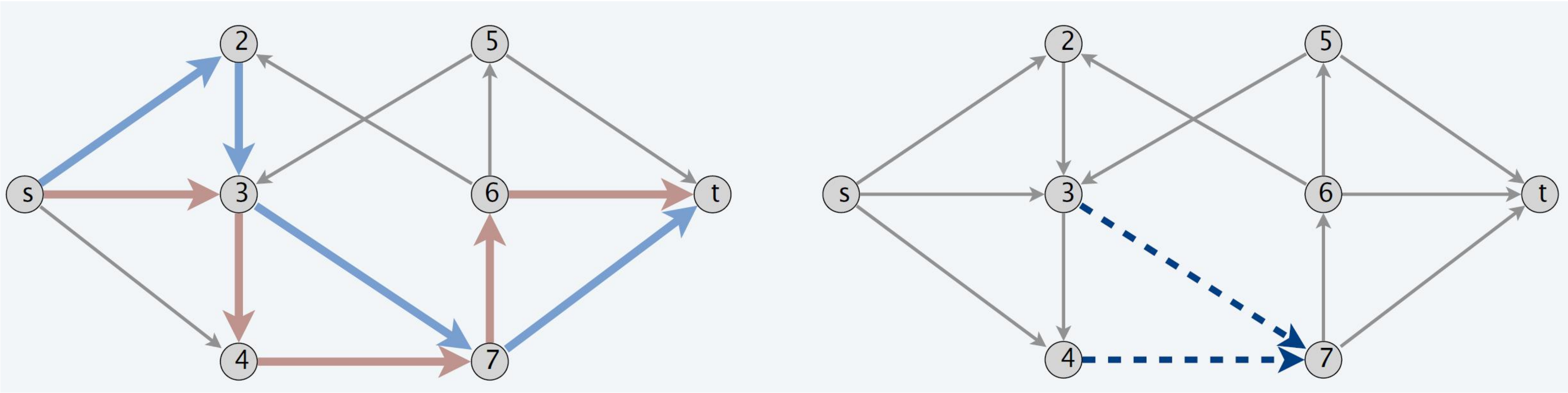


Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \leadsto t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\leq$

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- Every $s \leadsto t$ path uses at least one edge in F.
- Hence, the number of edge-disjoint paths is $\leq k$.

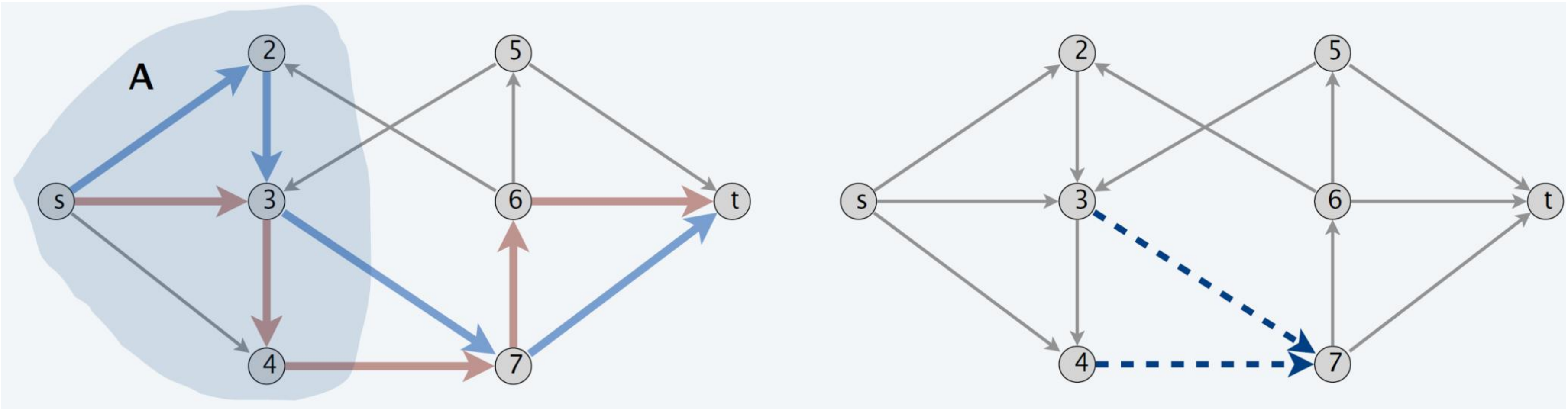


Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \rightsquigarrow t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\geq$

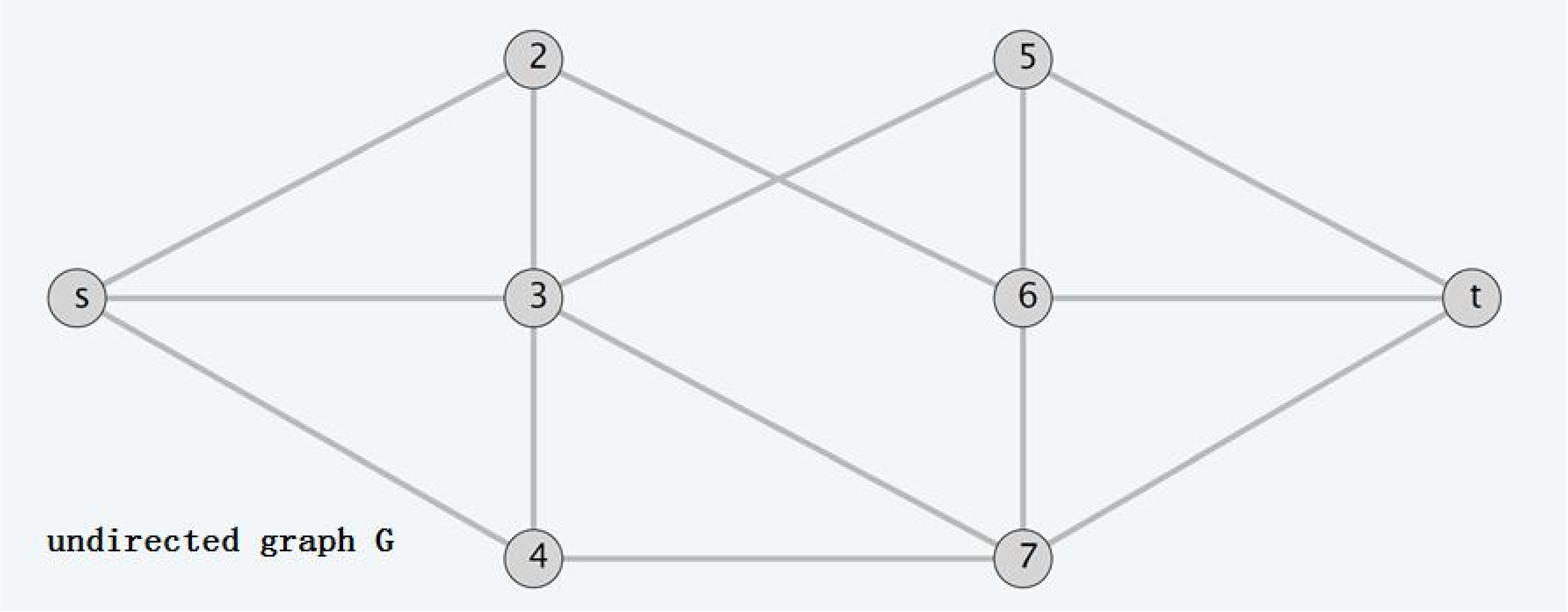
- Suppose max number of edge-disjoint paths is k.
- Then value of max flow = k.
- Max-flow min-cut theorem $\Rightarrow$ there exists a cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- $|F| = k$ and disconnects t from s.



Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

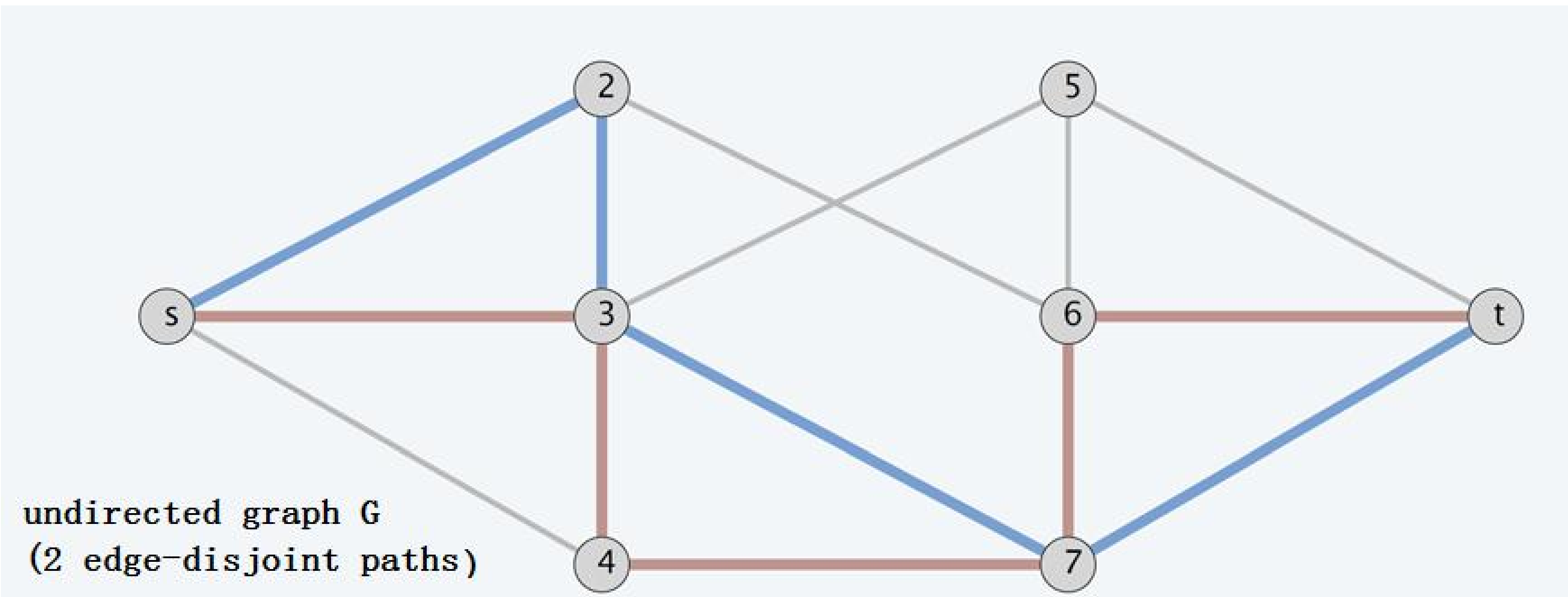
Edge-disjoint paths problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.



Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

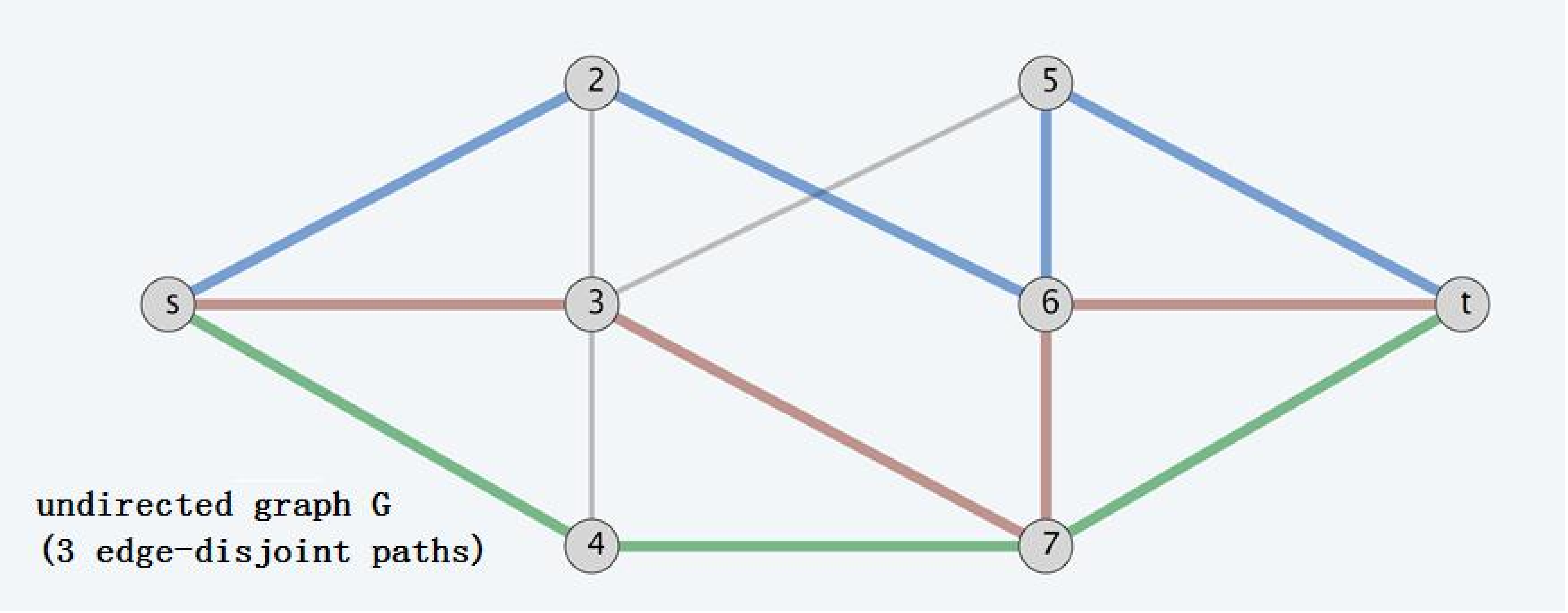
Edge-disjoint paths problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s–t paths.



Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Edge-disjoint paths problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.

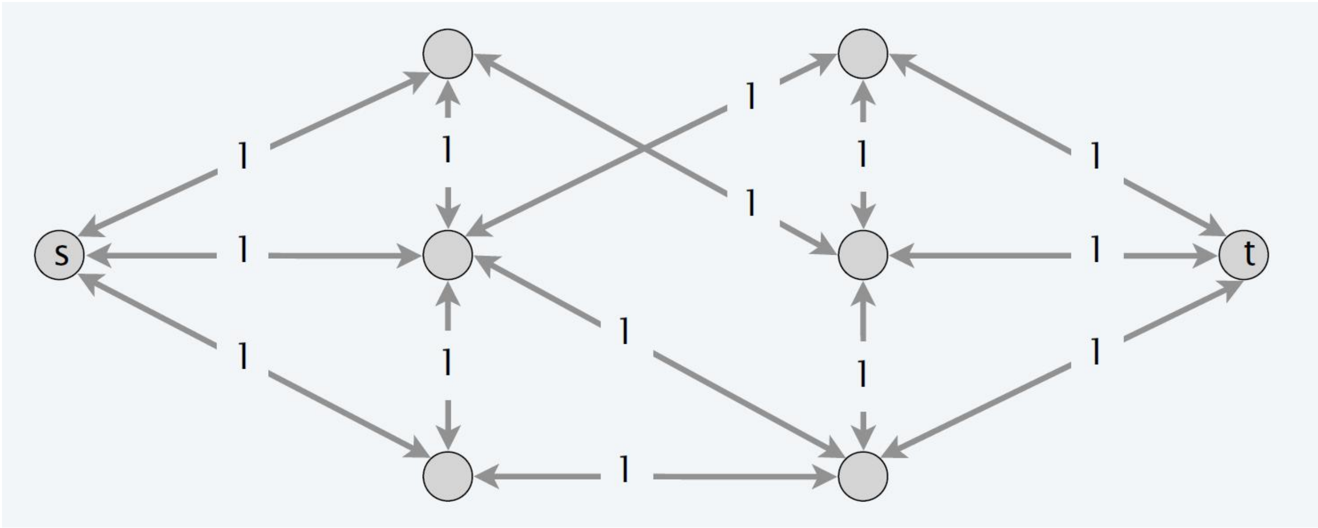


Edge-disjoint paths in undirected graphs

Max-flow formulation. Replace each edge with two antiparallel edges and assign unit capacity to every edge.

Observation. Two paths *P1* and *P2* may be edge-disjoint in the digraph but not edge-disjoint in the undirected graph.

if *P1* uses edge (*u*, *v*)
and *P2* uses its antiparallel edge (*v*, *u*)



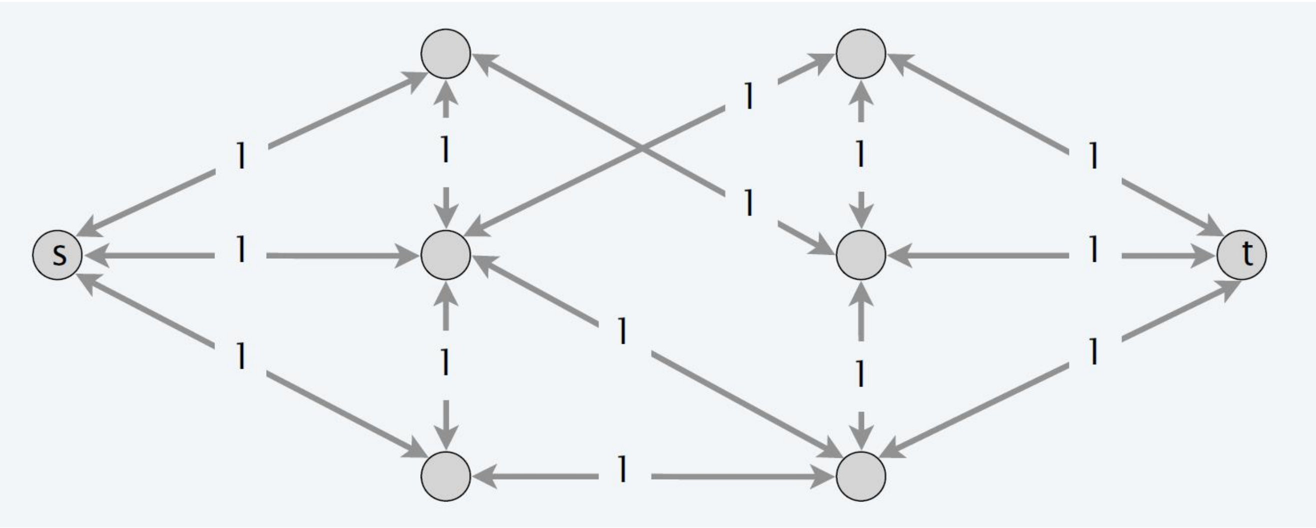
Edge-disjoint paths in undirected graphs

Max-flow formulation. Replace each edge with two antiparallel edges and assign unit capacity to every edge.

Lemma. In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e' : either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.

Pf. [by induction on number of such pairs]

- Suppose $f(e) > 0$ and $f(e') > 0$ for a pair of antiparallel edges e and e'.
- Set $f(e) = f(e) - \delta$ and $f(e') = f(e') - \delta$, where $\delta = \min \{ f(e), f(e') \}$.
- f is still a flow of the same value but has one fewer such pair

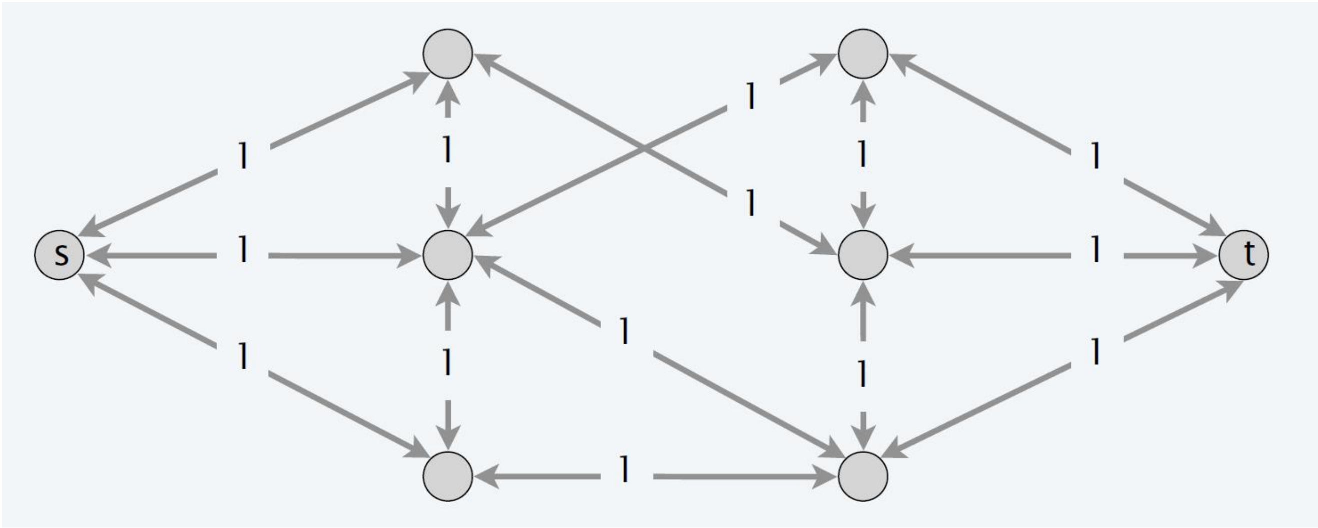


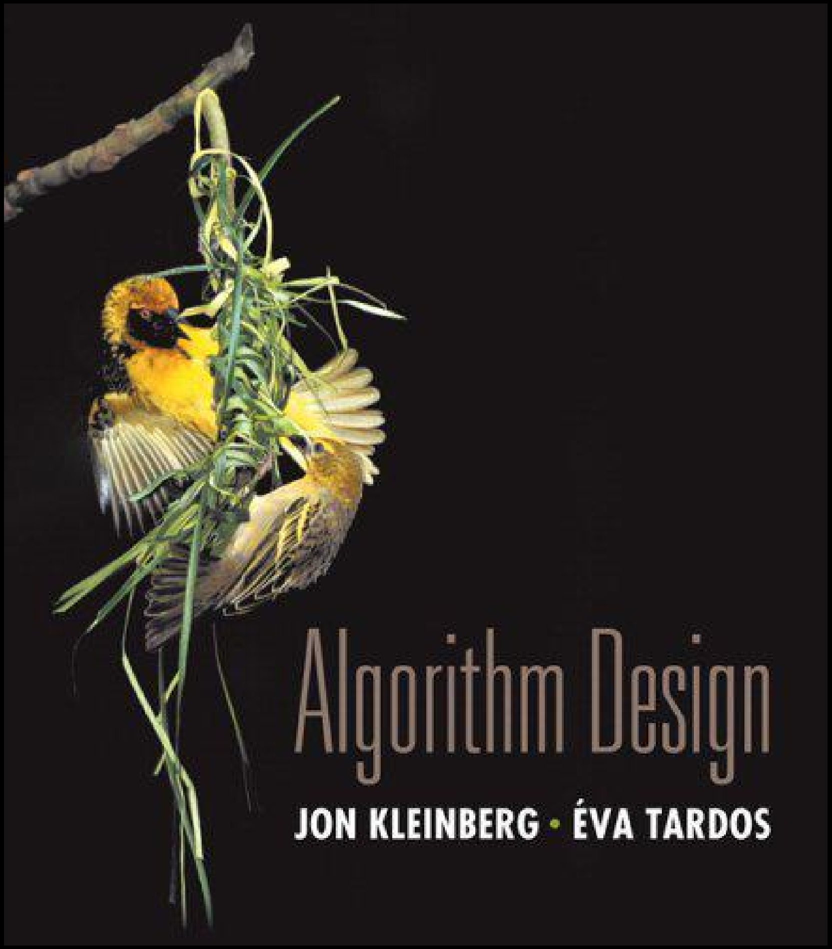
Edge-disjoint paths in undirected graphs

Max-flow formulation. Replace each edge with two antiparallel edges and assign unit capacity to every edge.

Lemma. In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e' : either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.

Theorem. Max number of edge-disjoint $s \rightsquigarrow t$ paths = value of max flow.
Pf. Similar to proof in digraphs; use lemma.





Chapter 7

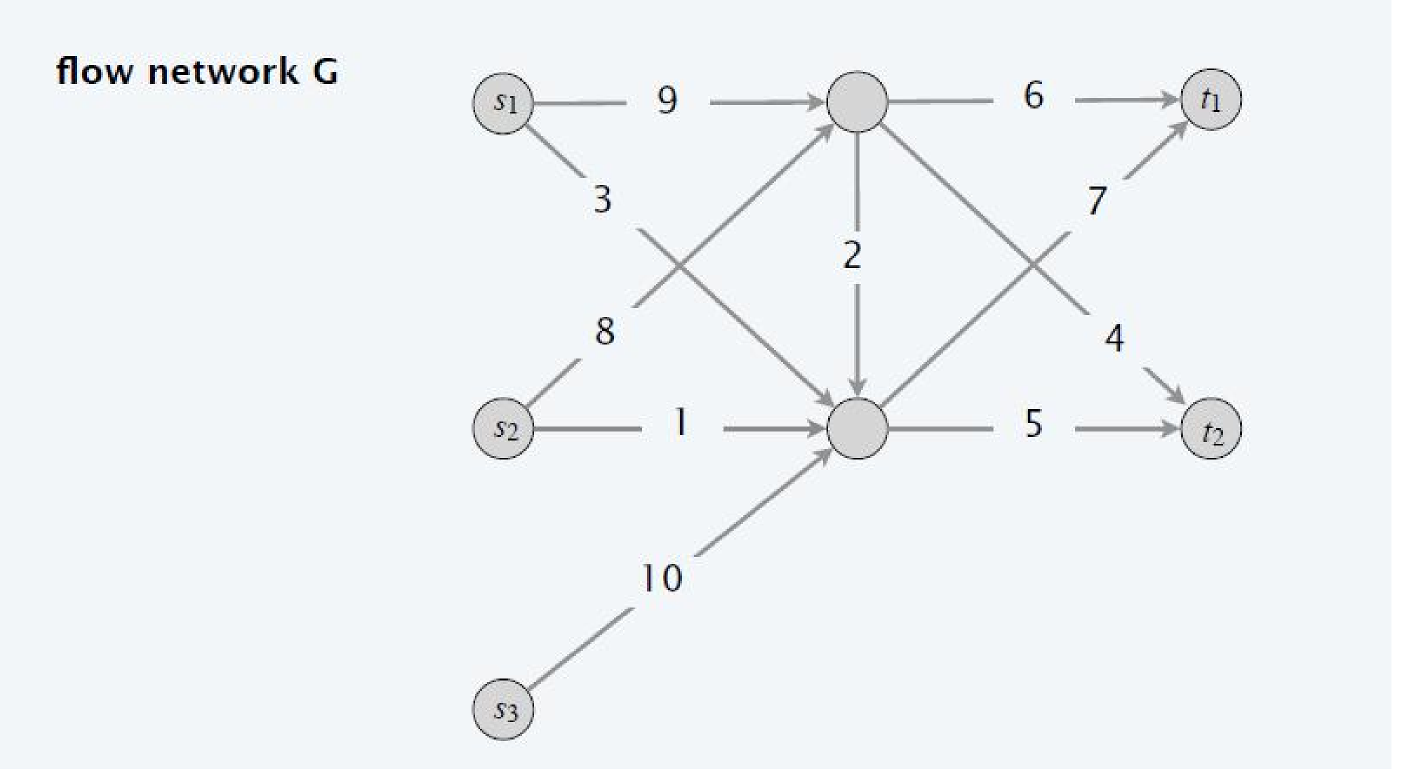
Network Flow

Slides by Kevin Wayne.

7.7 Extensions to Maximum-Flow Problem

Multiple sources and sinks

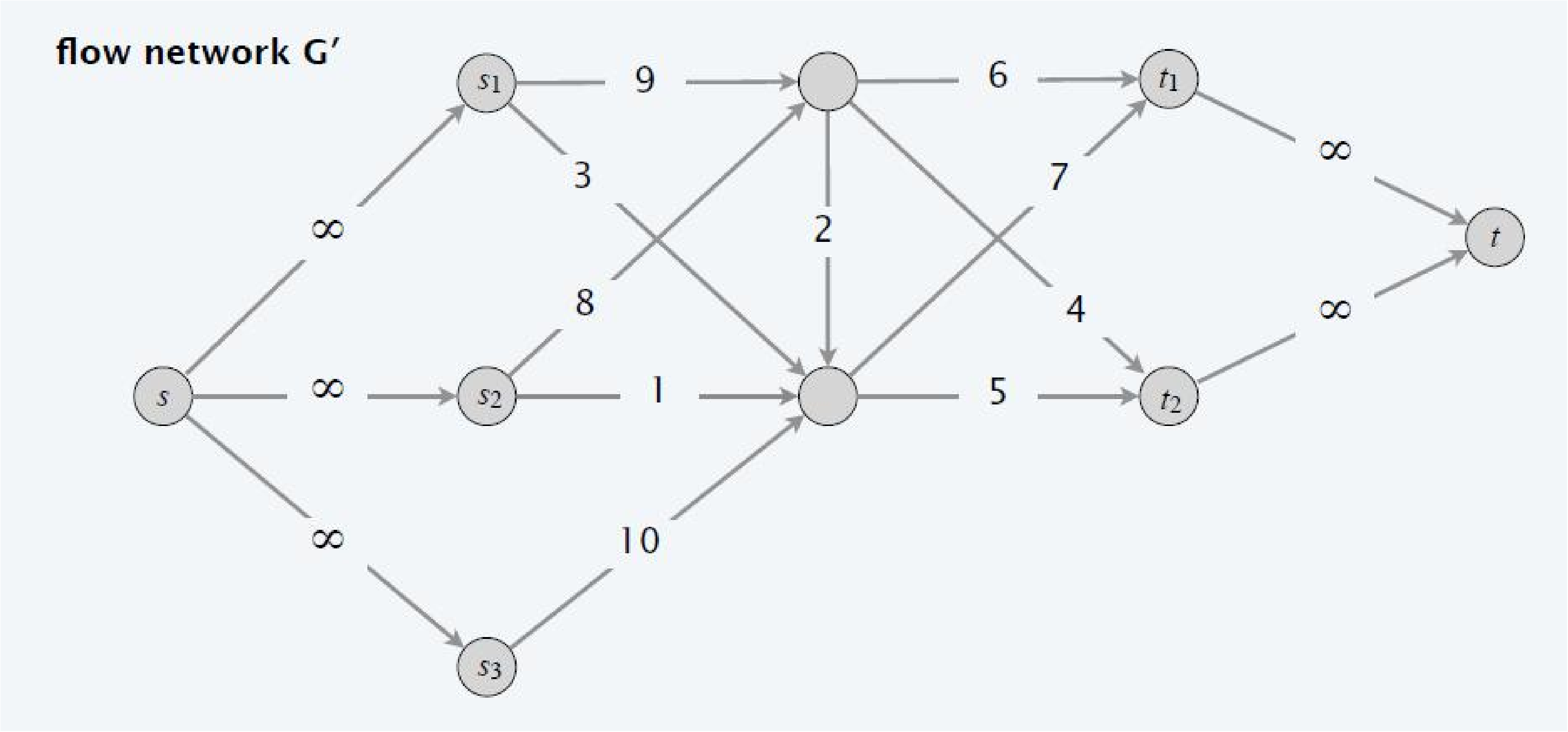
Def. Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and multiple source nodes and multiple sink nodes, find max flow that can be sent from the source nodes to the sink nodes.



Multiple sources and sinks: max-flow formulation

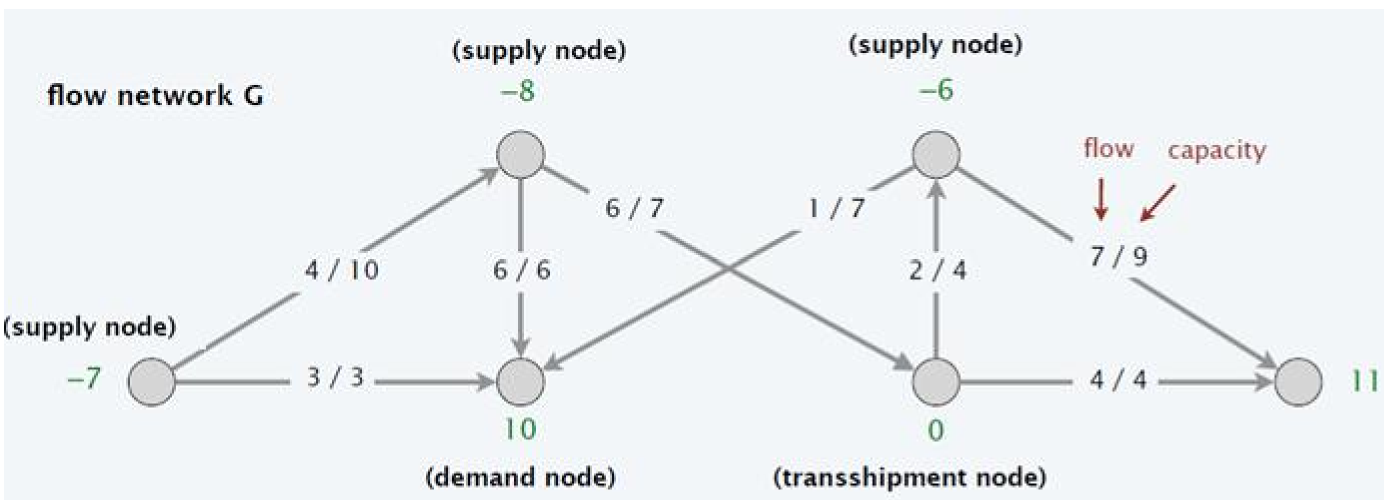
- Add a new source node s and sink node t.
- For each original source node s_i add edge (s, s_i) with capacity ∞.
- For each original sink node t_j, add edge (t_j, t) with capacity ∞.

Claim. 1–1 correspondence betweens flows in G and G'.



Circulation with supplies and demands

Def. Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and node demands $d(v)$, a circulation is a function $f(e)$ that satisfies:
For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (flow conservation)

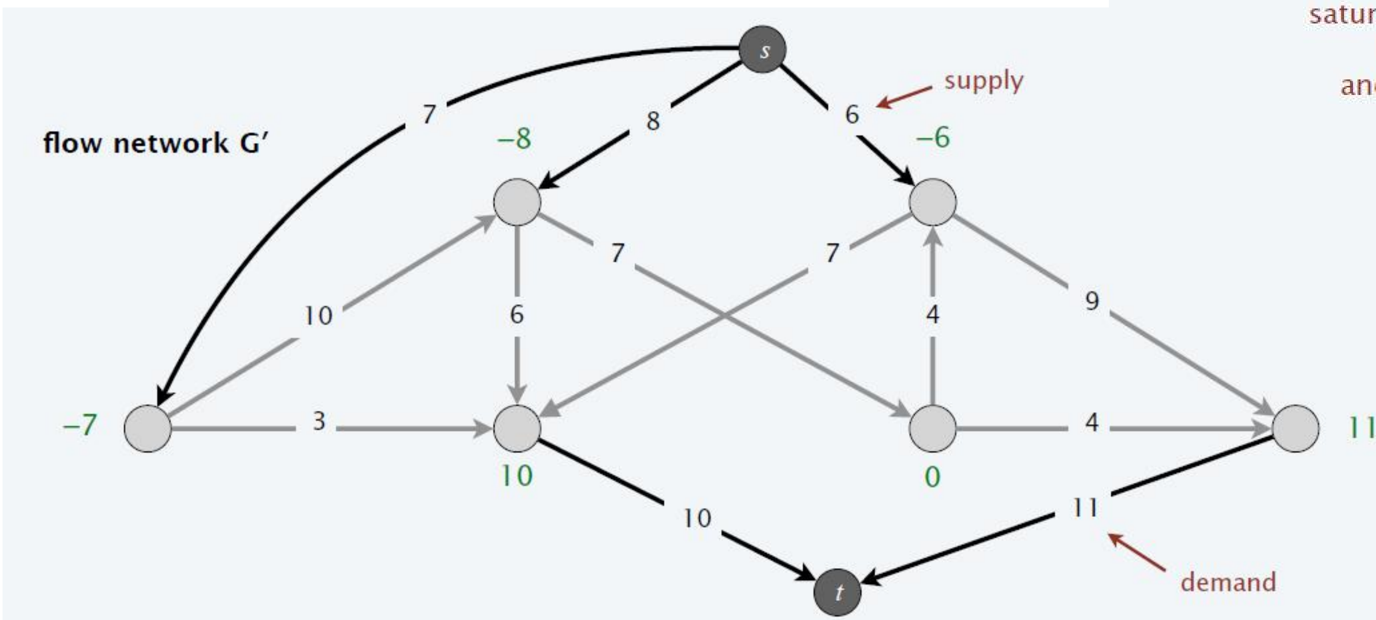


Circulation with supplies and demands: max-flow formulation

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

saturates all edges leaving s and entering t



Circulation with supplies and 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 + integrality theorem for max flow.

Theorem. Given (V, E, c, d), there does not exist a circulation iff there exists a node partition (A, B) such that $\Sigma_{v \in B}\, d(v) > cap(A, B)$.

Pf sketch. Look at min cut in G '.

demand by nodes in B exceeds
supply of nodes in B plus
max capacity of edges going from A to B

$\boxed{\text{Previous slide: } G \text{ has circulation iff } G\text{' has max flow of value} ==》 \text{ max flow == min cut}}$