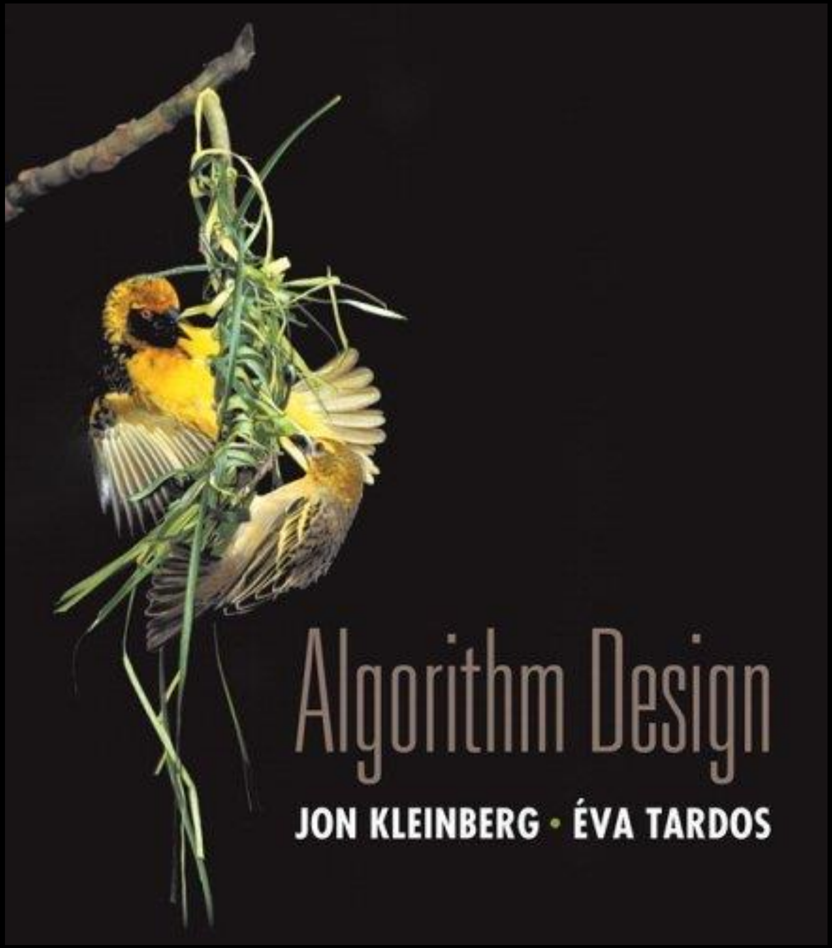


Chapter 7

Network Flow

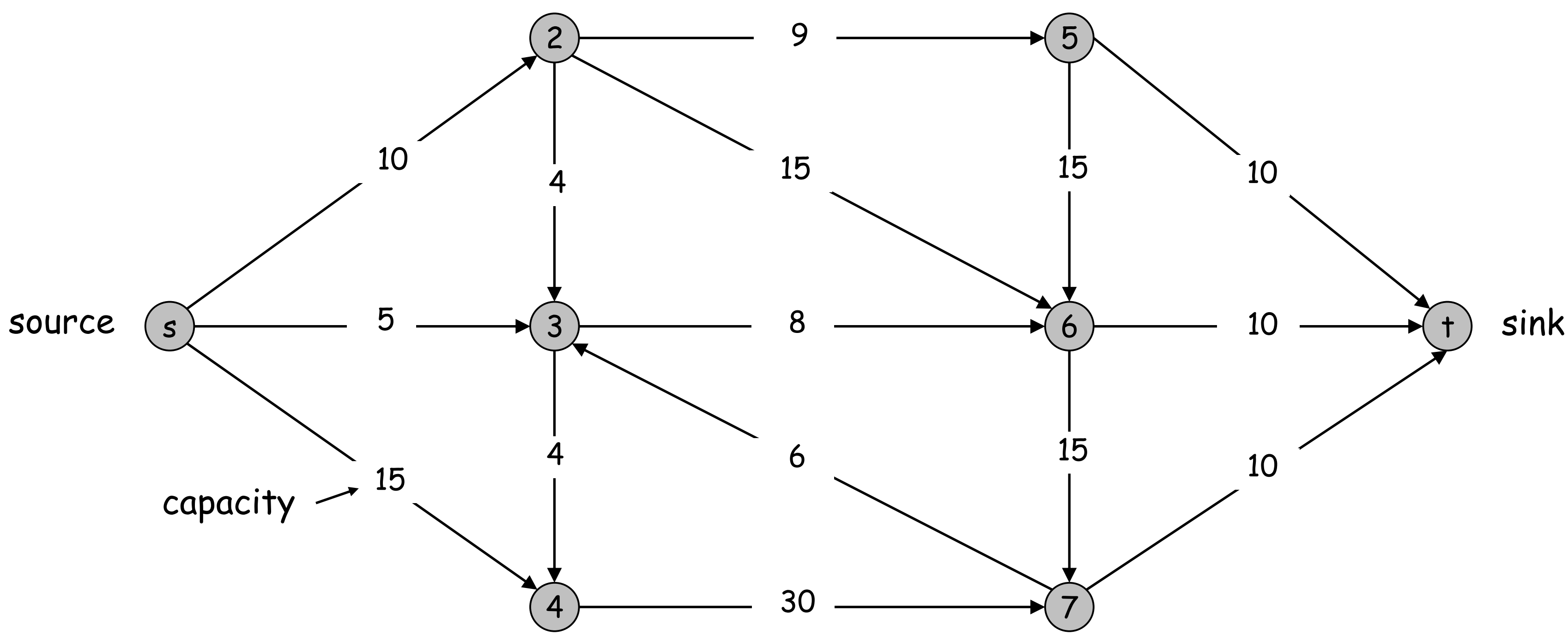
Slides by Kevin Wayne.

7.1 Max-flow and Ford-Fulkerson Algorithm

Flows

Flow network.

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

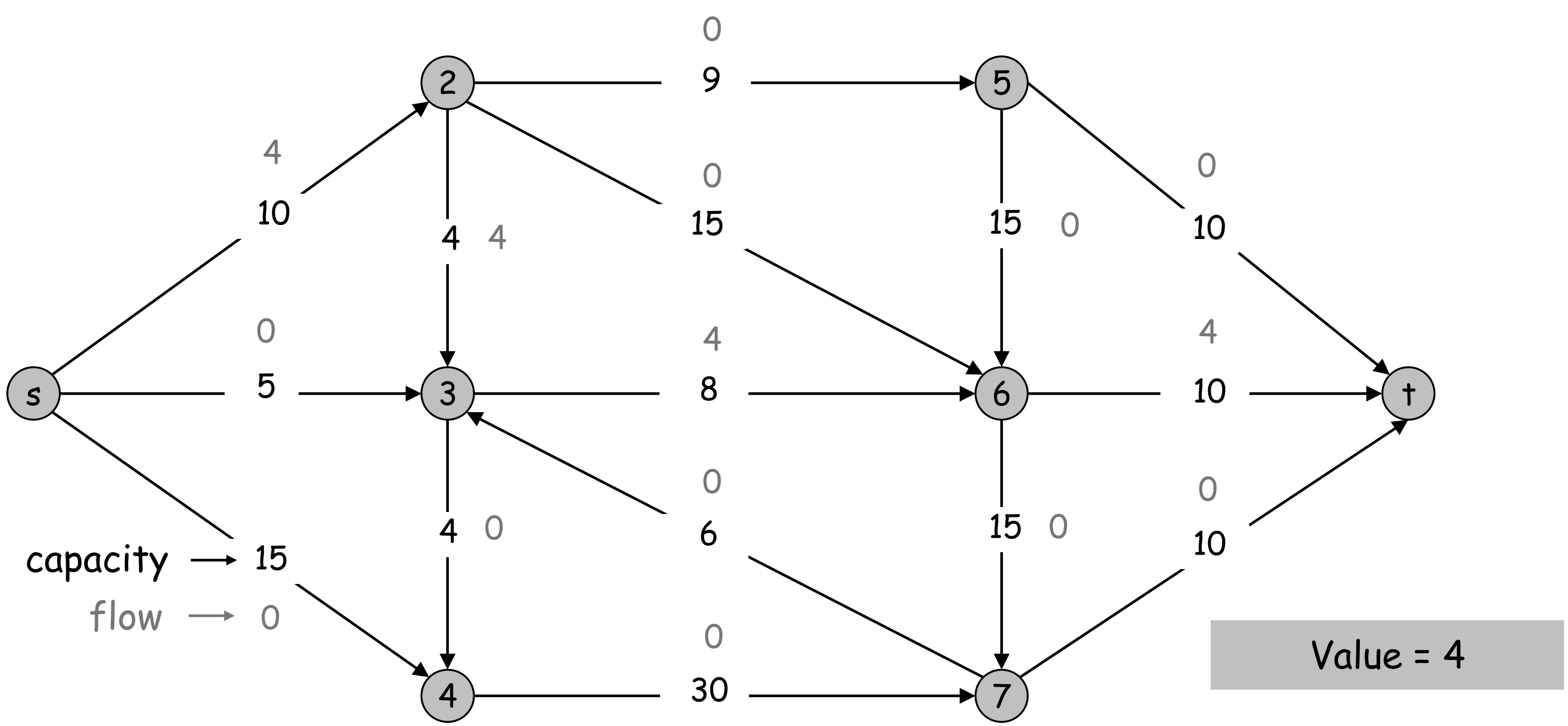


Flows

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

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

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

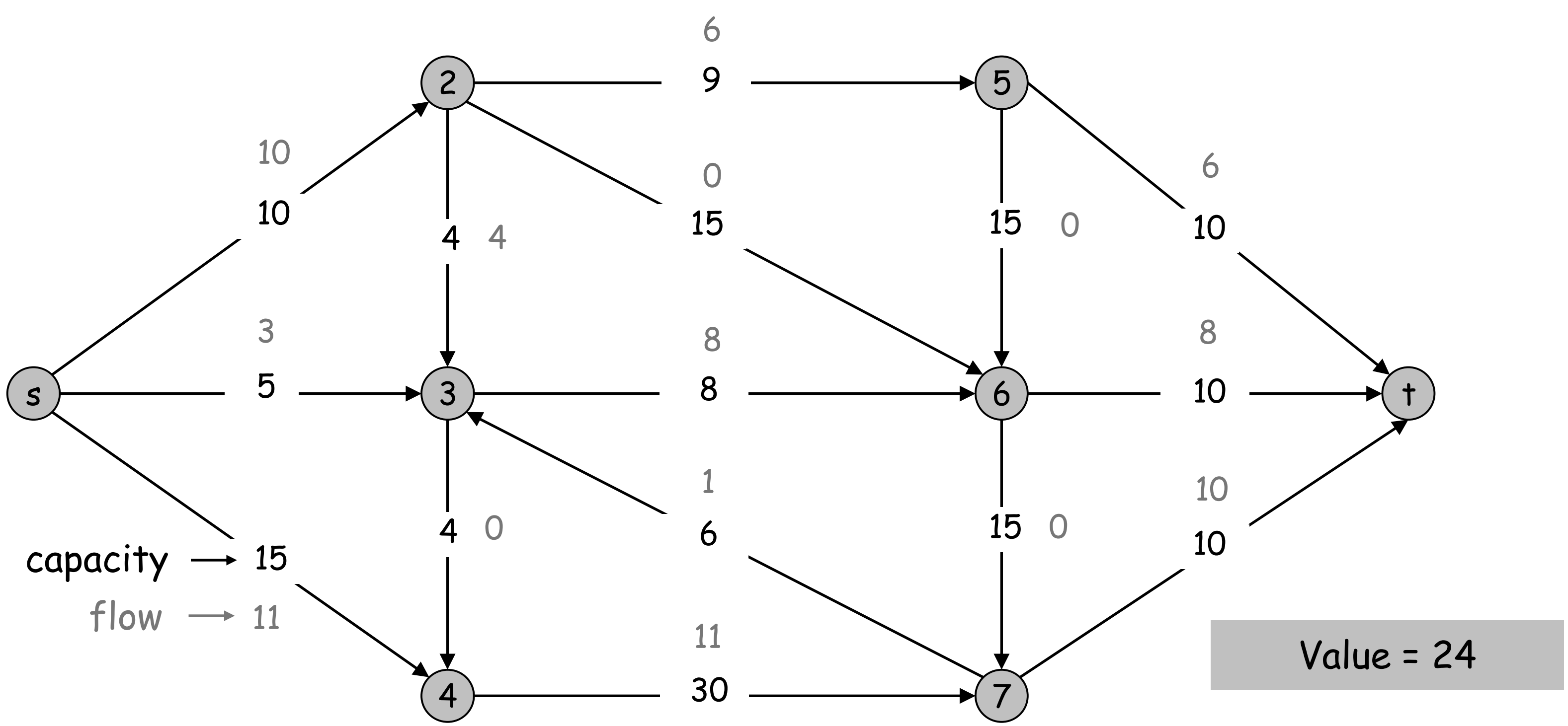


Flows

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

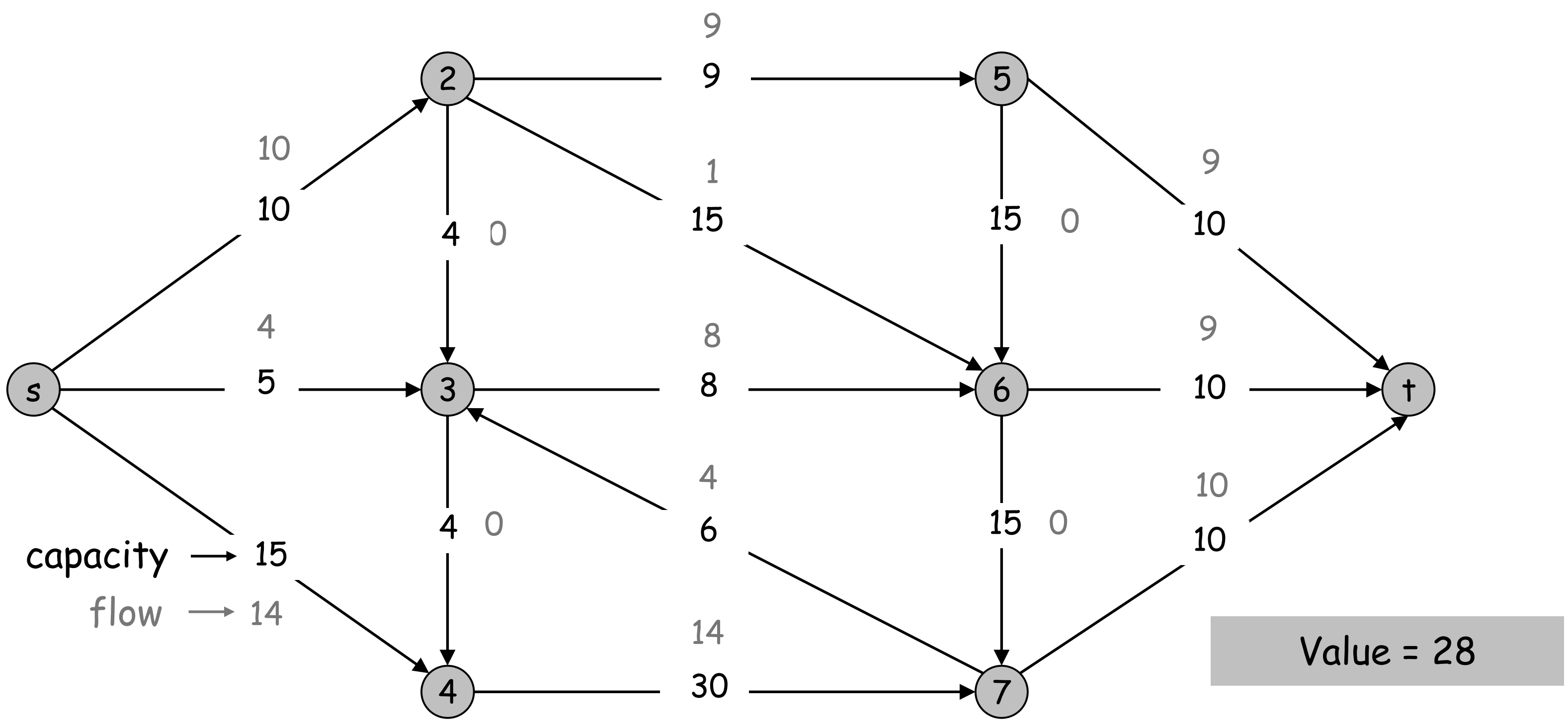
- For each $e \in E$: $\quad 0 \leq f(e) \leq c(e) \quad$ (capacity)
- For each $v \in V - \{s, t\}$: $\quad \sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e) \quad$ (conservation)

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



Maximum Flow Problem

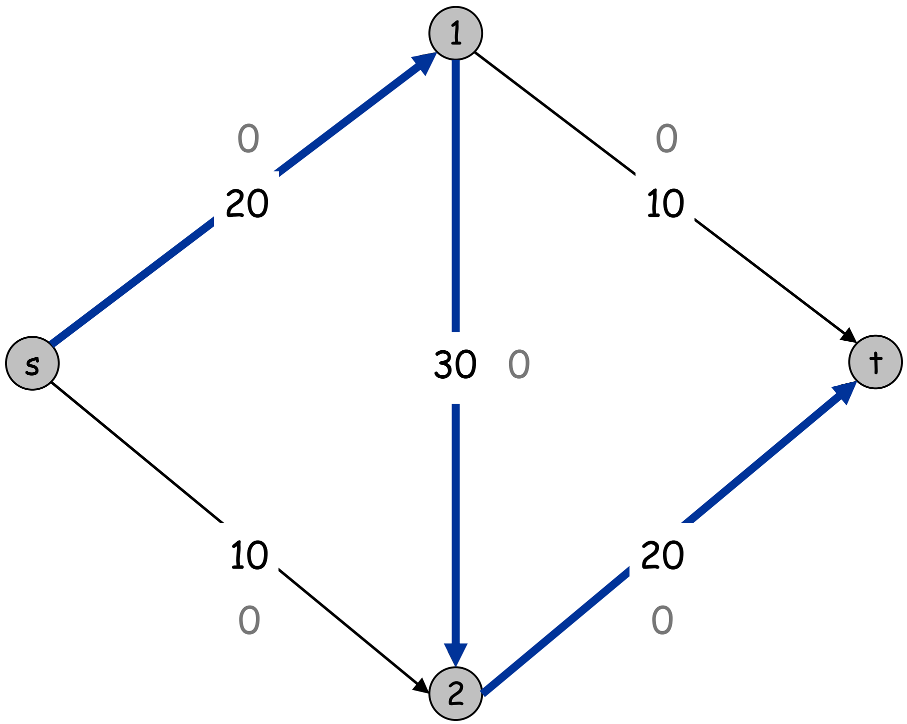
Max flow problem. Find s-t flow of maximum value.



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.

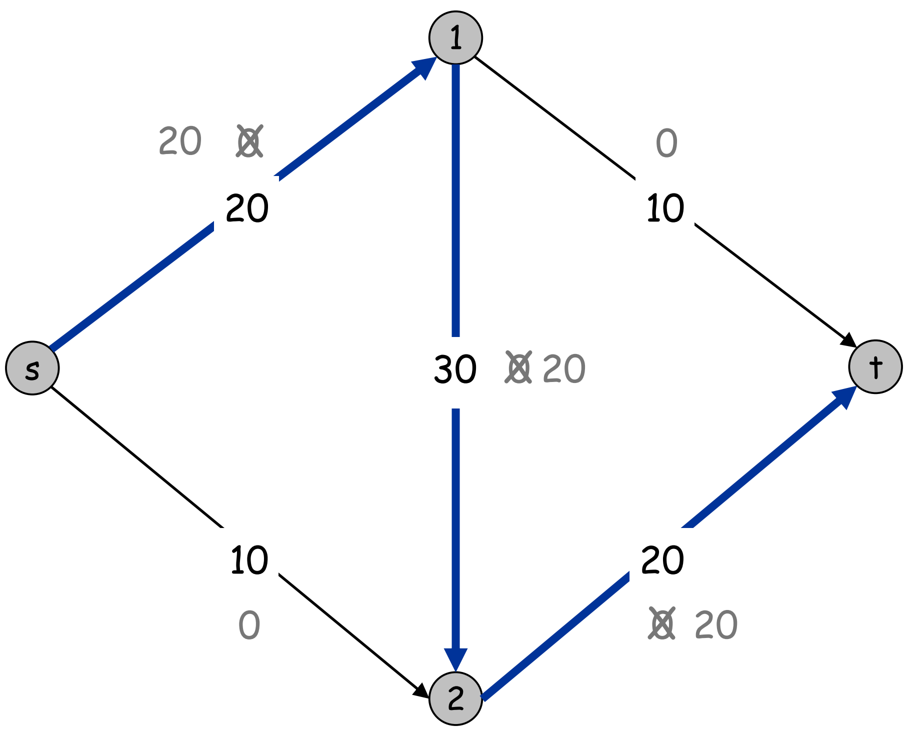


Flow value = 0

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.

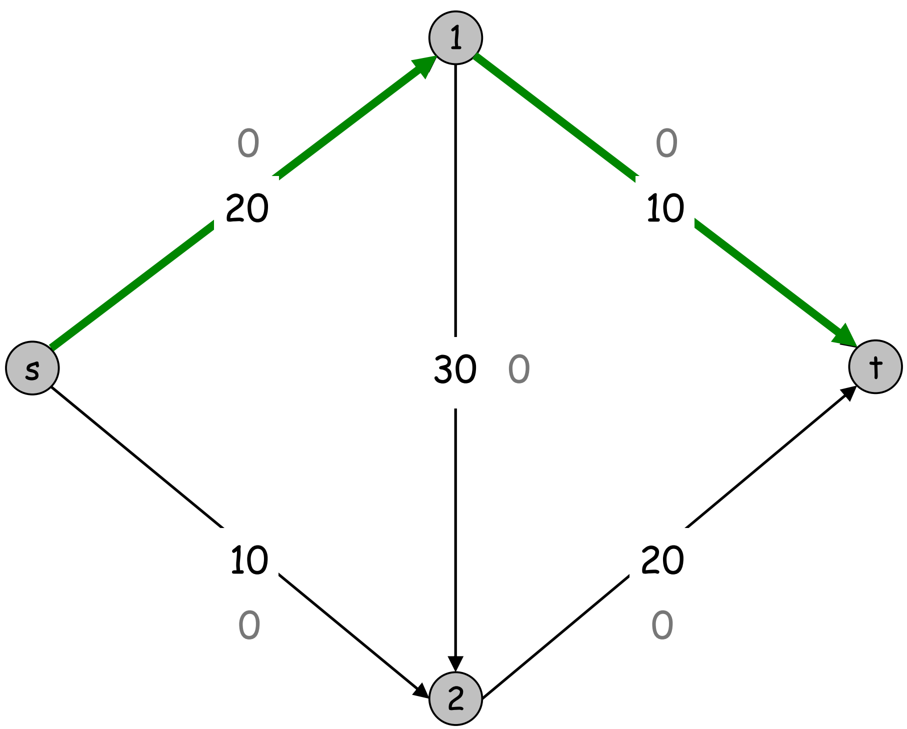


Flow value = 20

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.

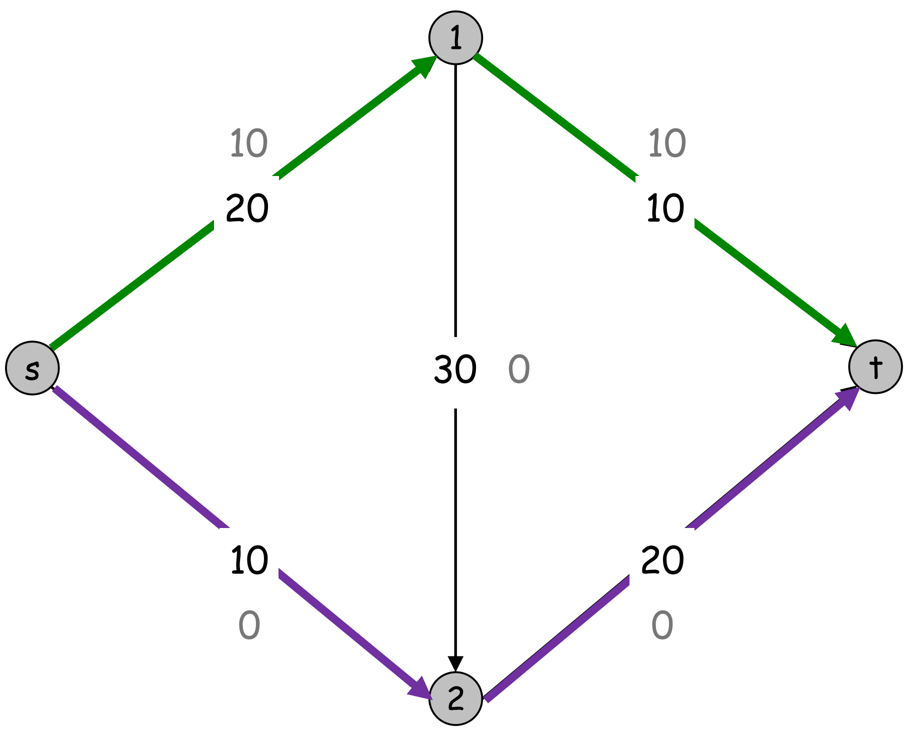


Flow value = 0

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.

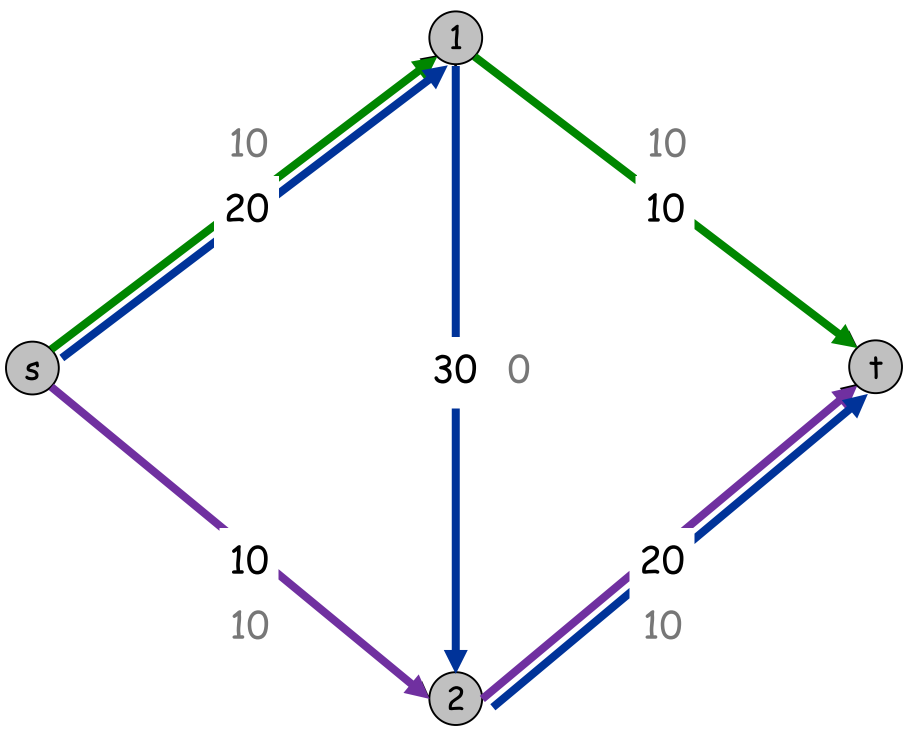


Flow value = 10

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.

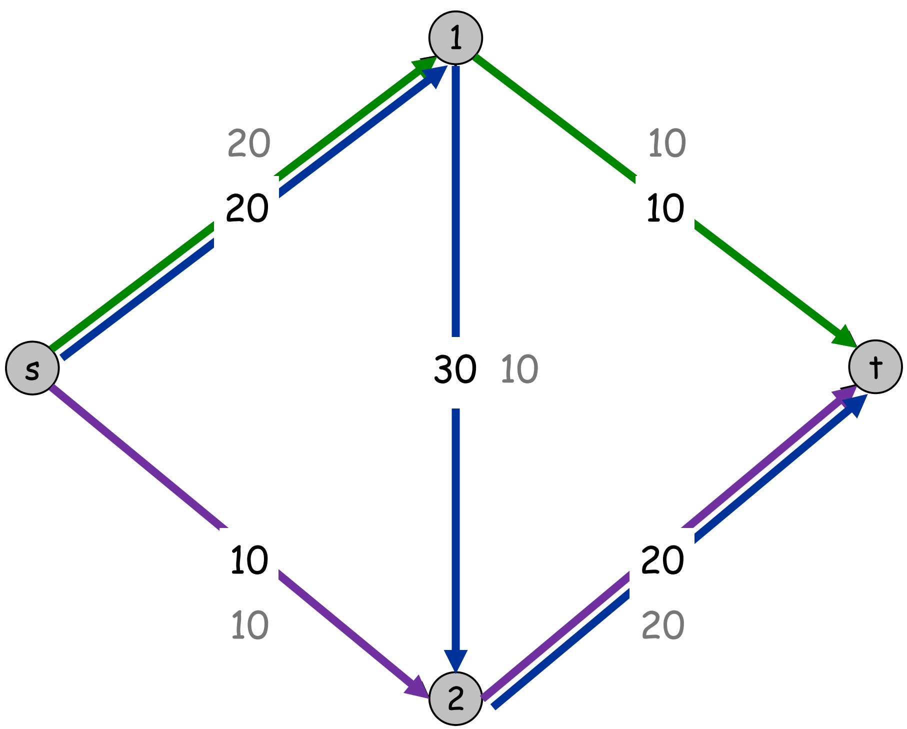


Flow value = 20

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.



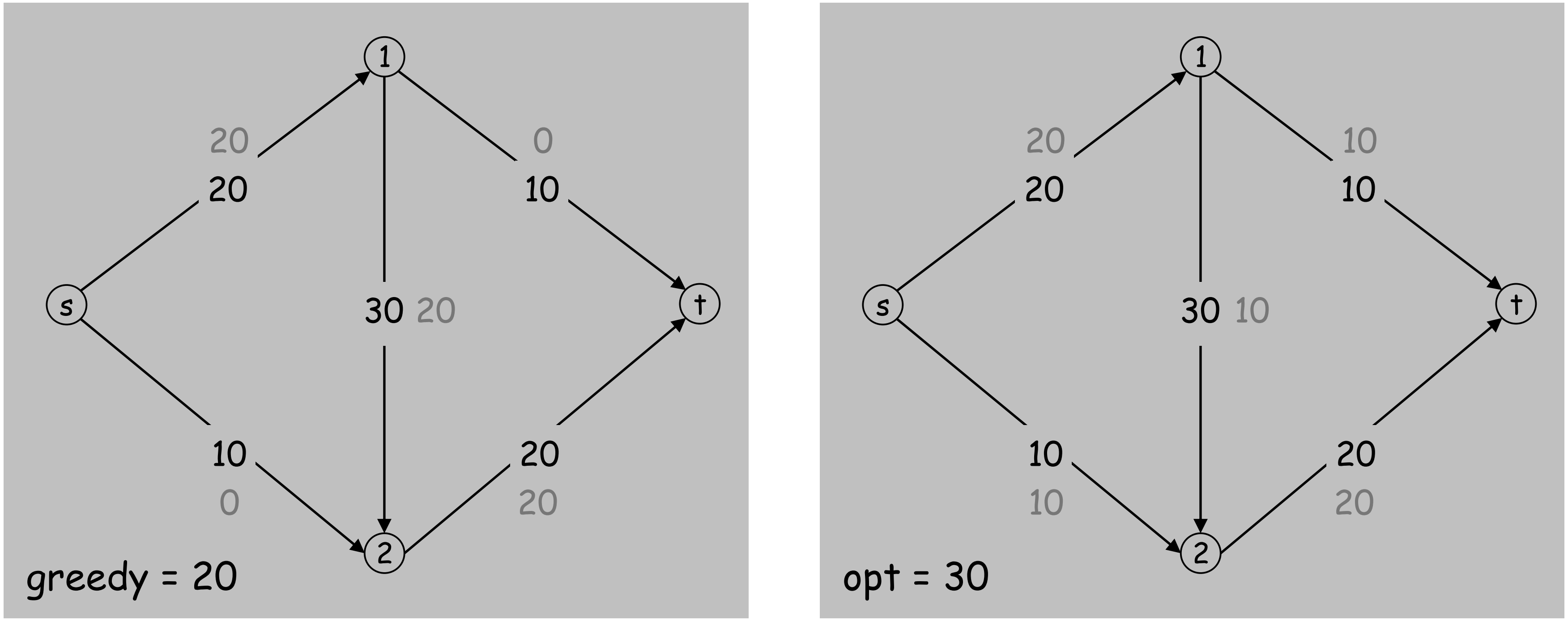
Flow value = 30

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.

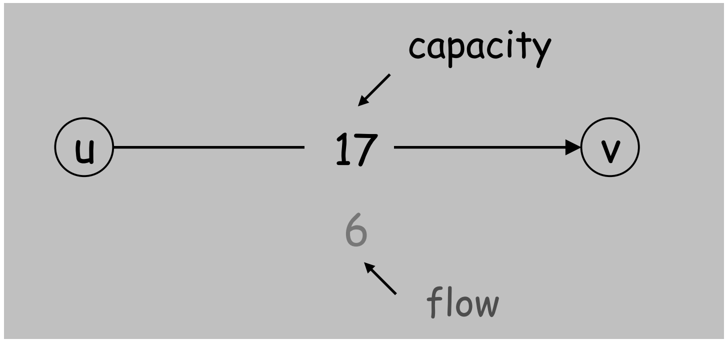
locally optimality $\not\Rightarrow$ global optimality



Residual Graph

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

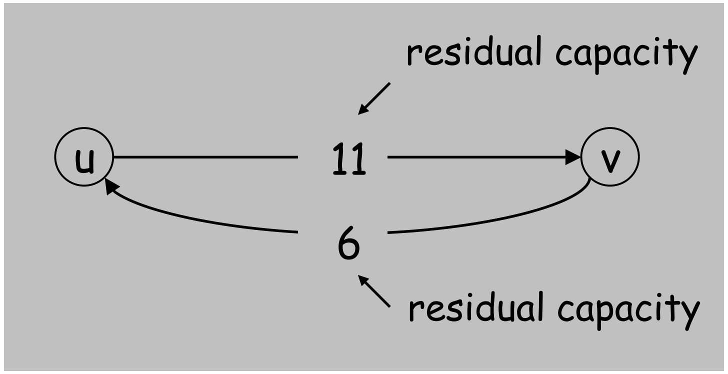
- Flow $f(e)$, capacity $c(e)$.



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

Augmenting path: a simple s-t path P in the residual graph G_f

Bottleneck capacity of an augmenting path P is the minimum residual capacity of any edge in P

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

Claim: After augmentation, f is still a flow.

Ford-Fulkerson Algorithm

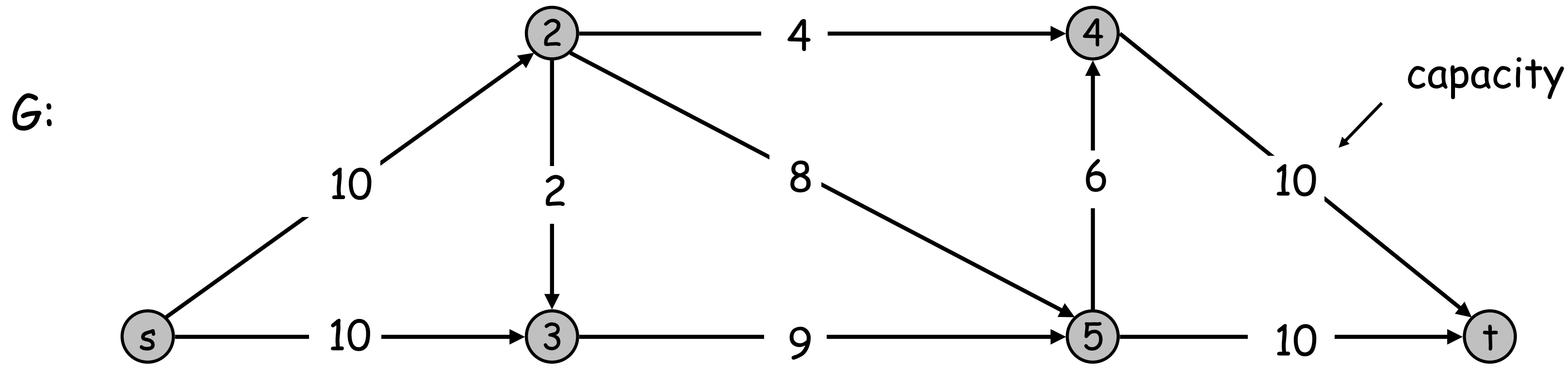
Ford-Fulkerson Algorithm

- 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.

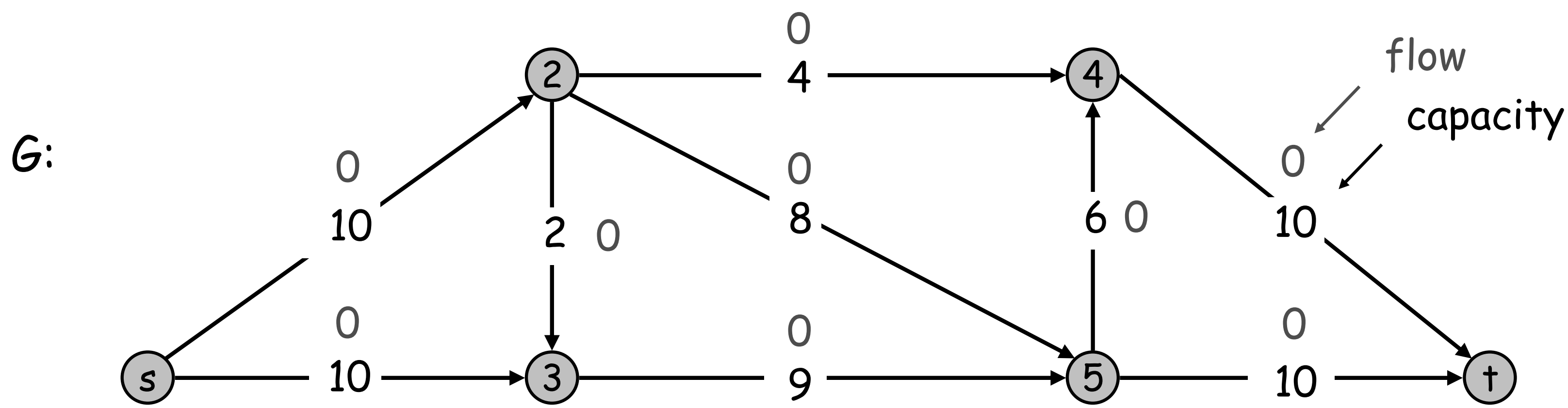
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Gf ← residual graph

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

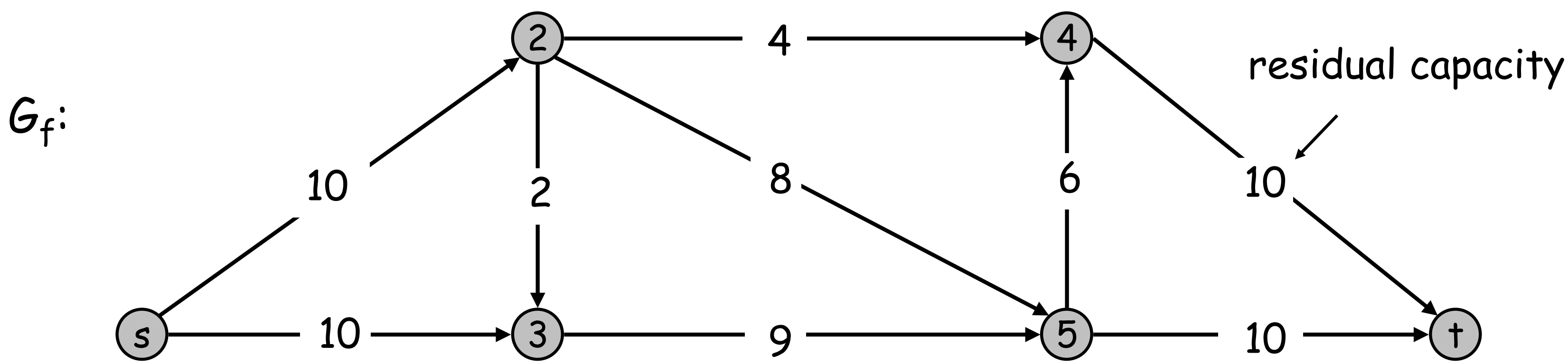
Ford-Fulkerson Algorithm



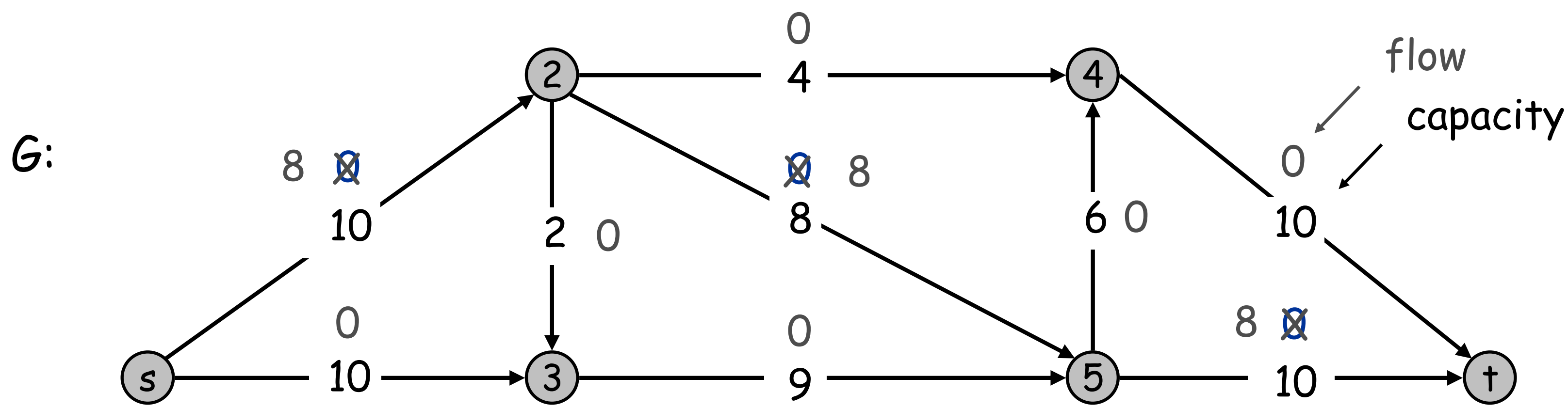
Ford-Fulkerson Algorithm



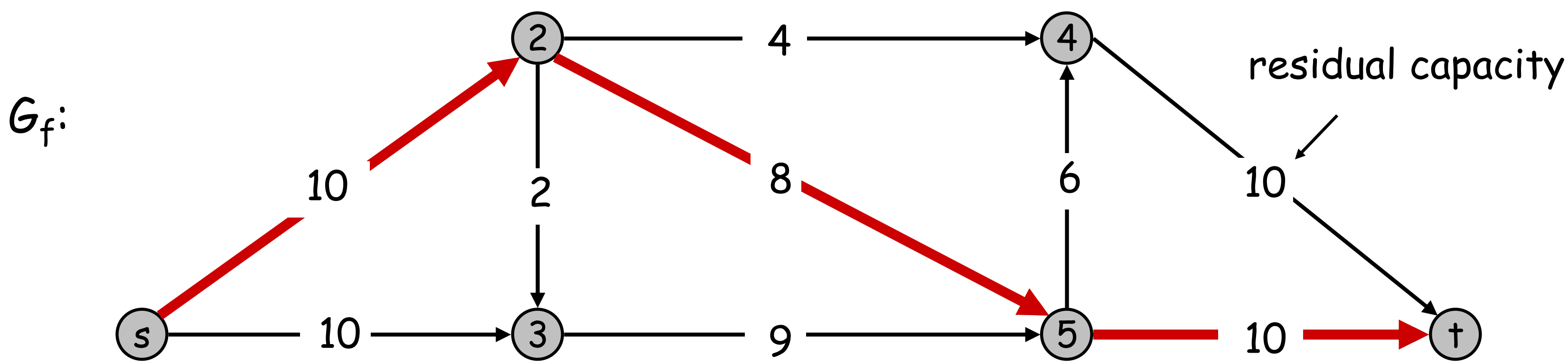
Flow value = 0



Ford-Fulkerson Algorithm

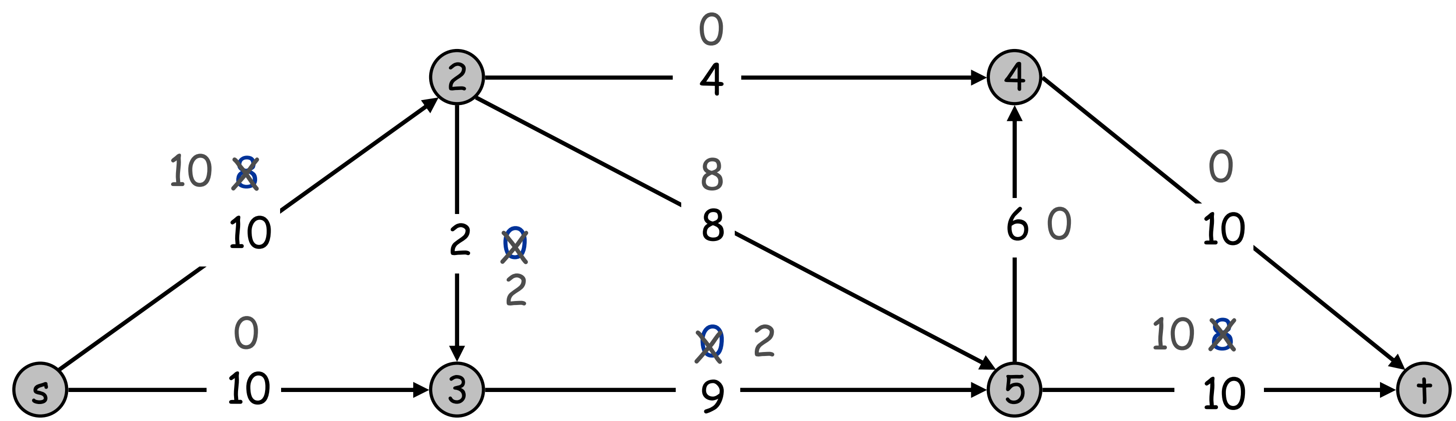


Flow value = 0



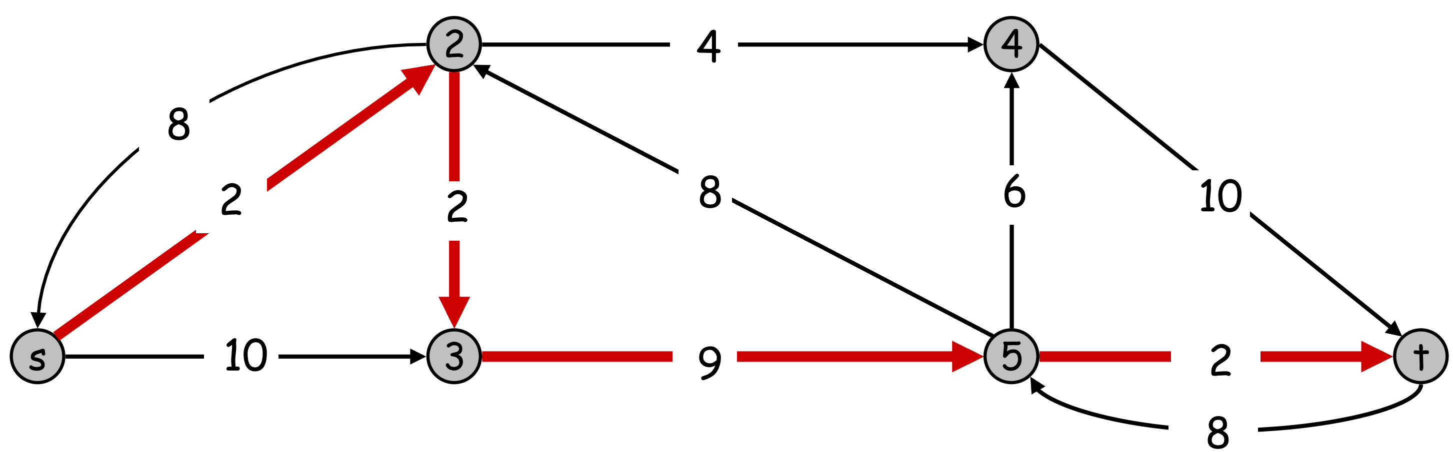
Ford-Fulkerson Algorithm

G:



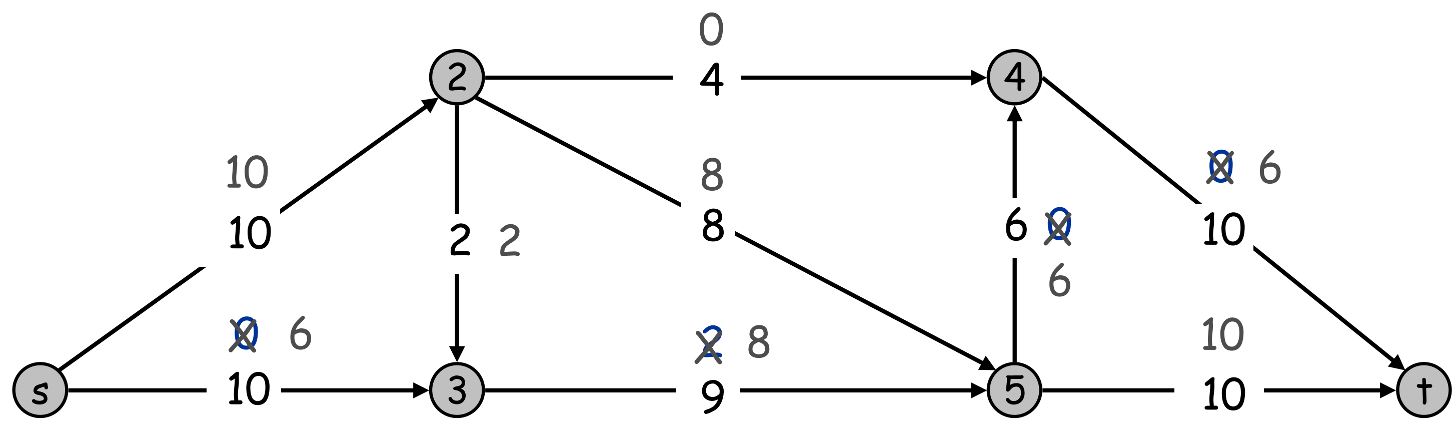
Flow value = 8

G_f:



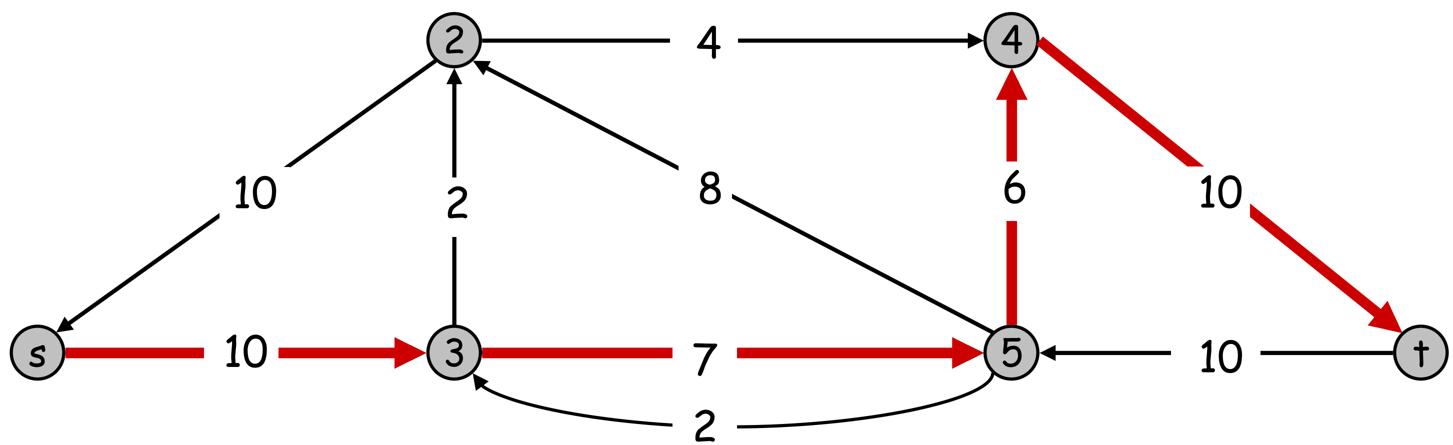
Ford-Fulkerson Algorithm

G:

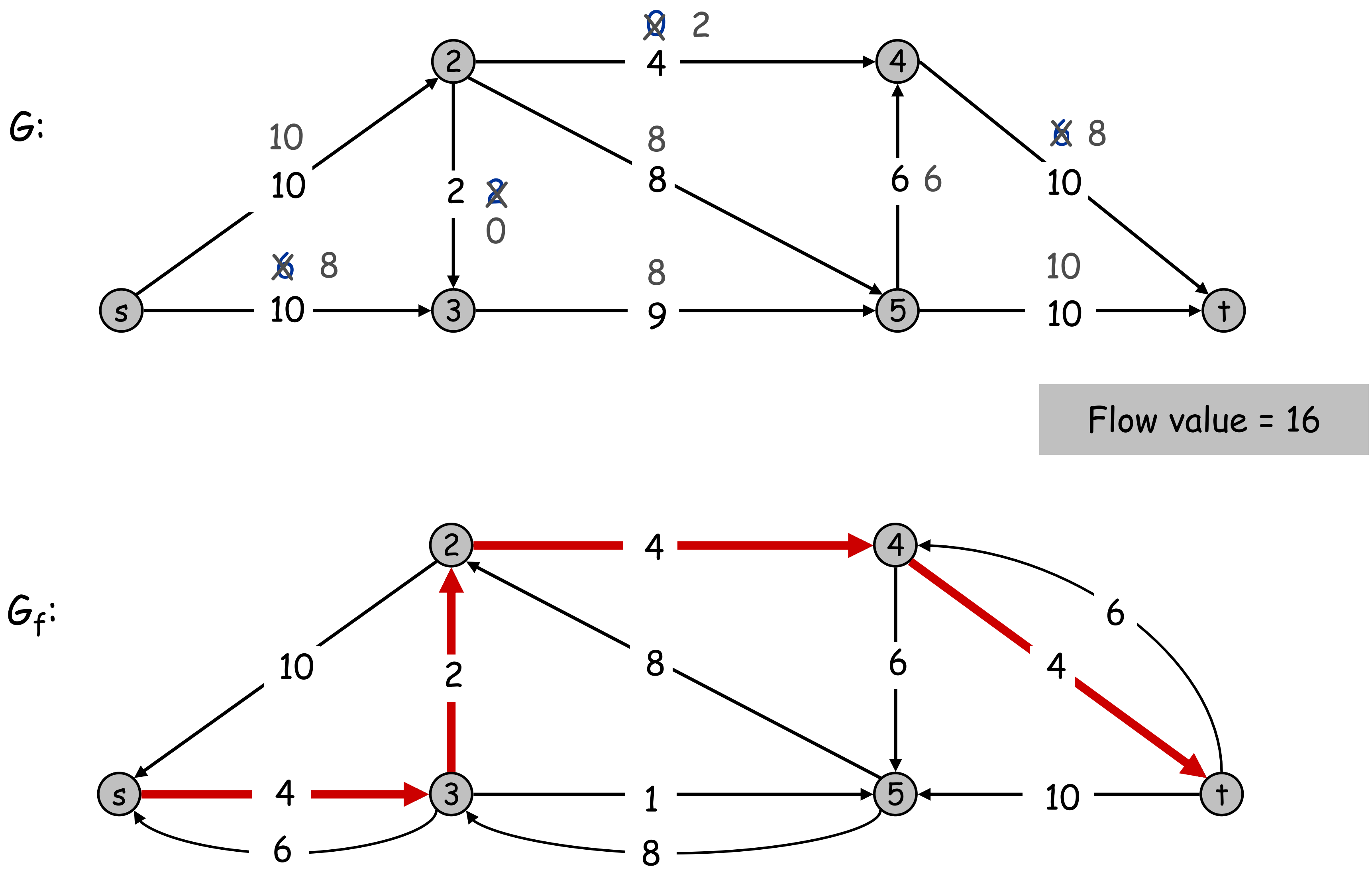


Flow value = 10

G_f:

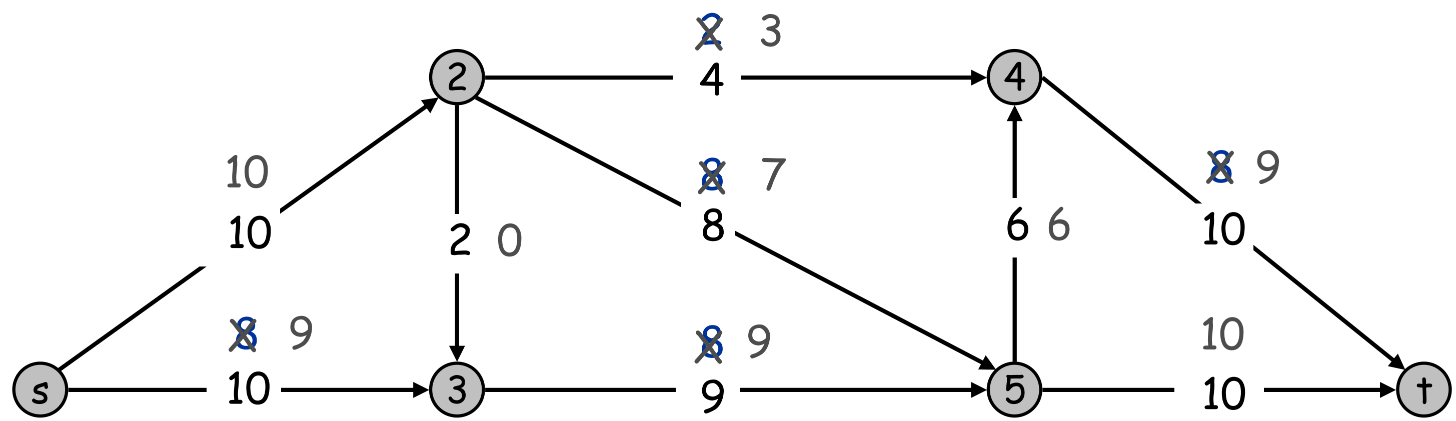


Ford-Fulkerson Algorithm

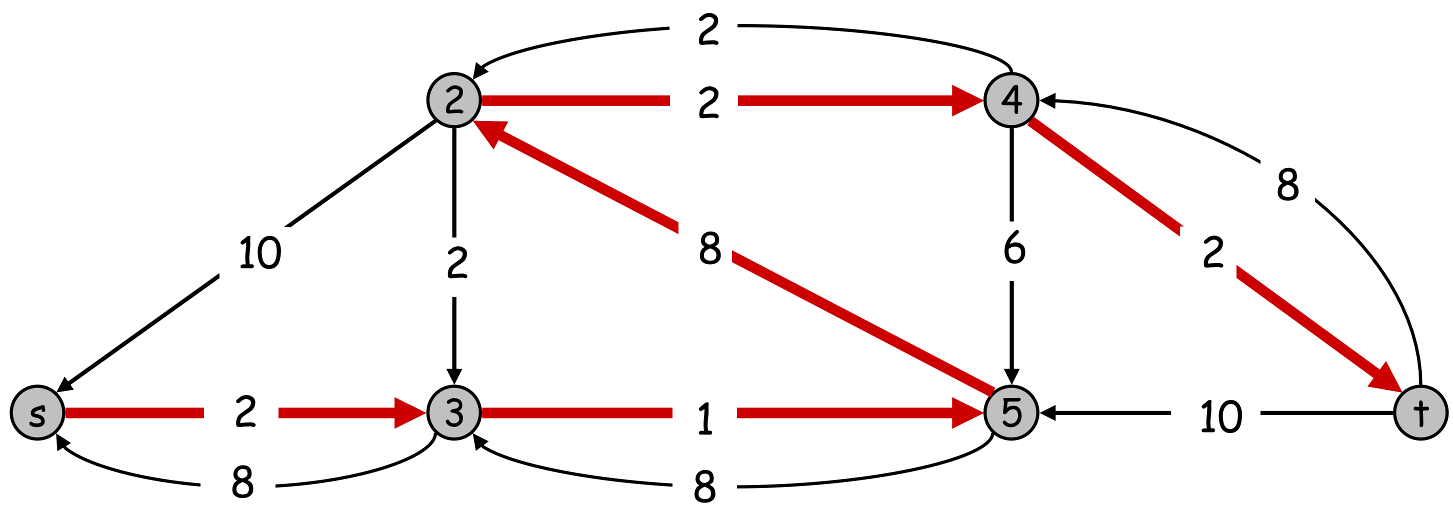


Ford-Fulkerson Algorithm

G:

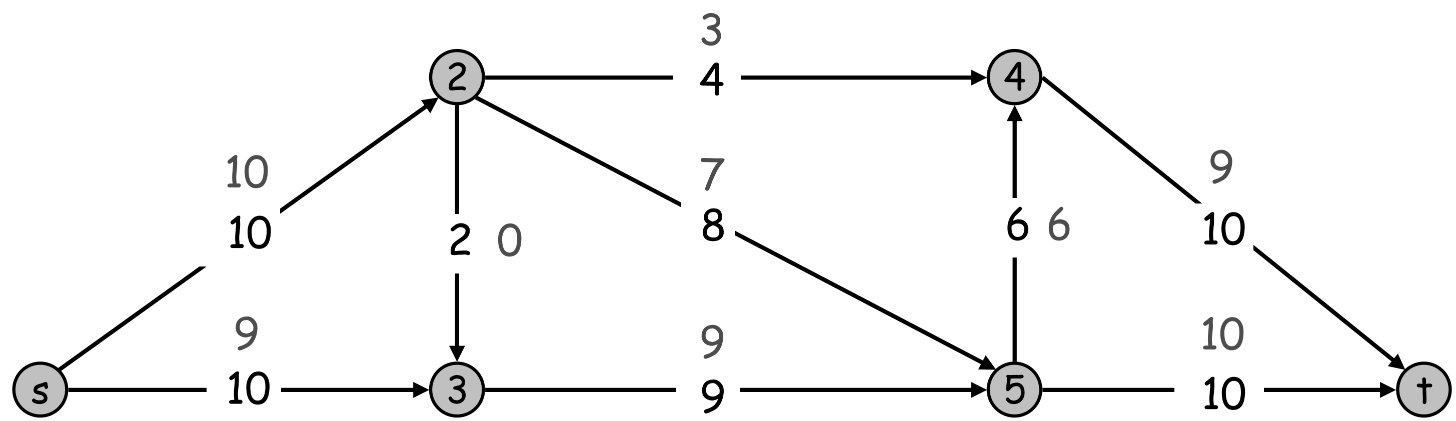


G_f:



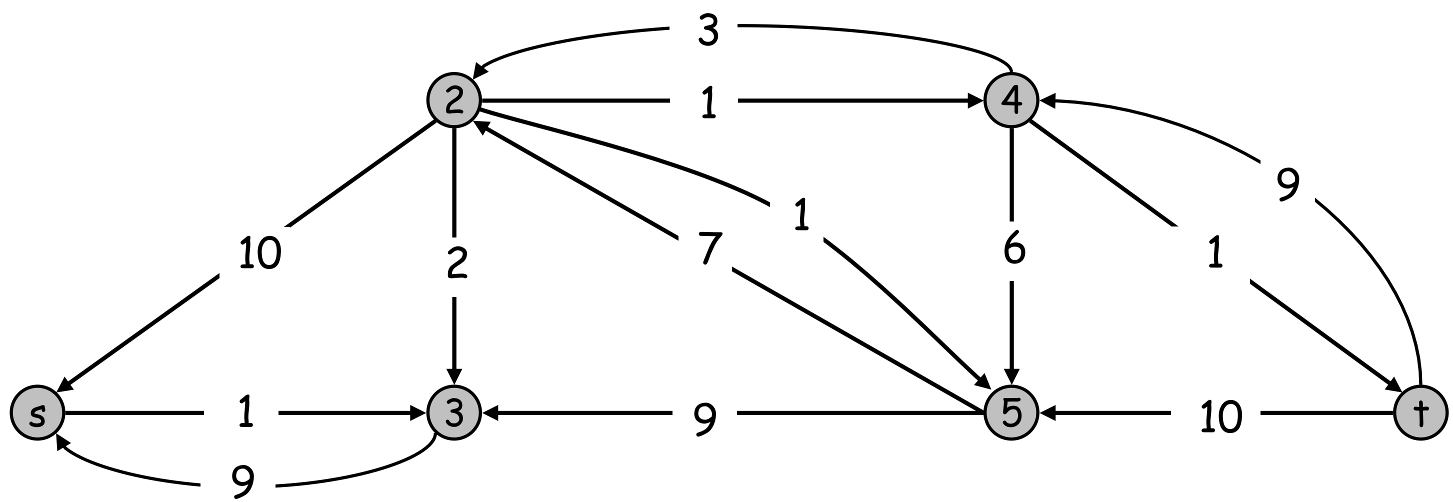
Ford-Fulkerson Algorithm

G:



Flow value = 19

G_f:

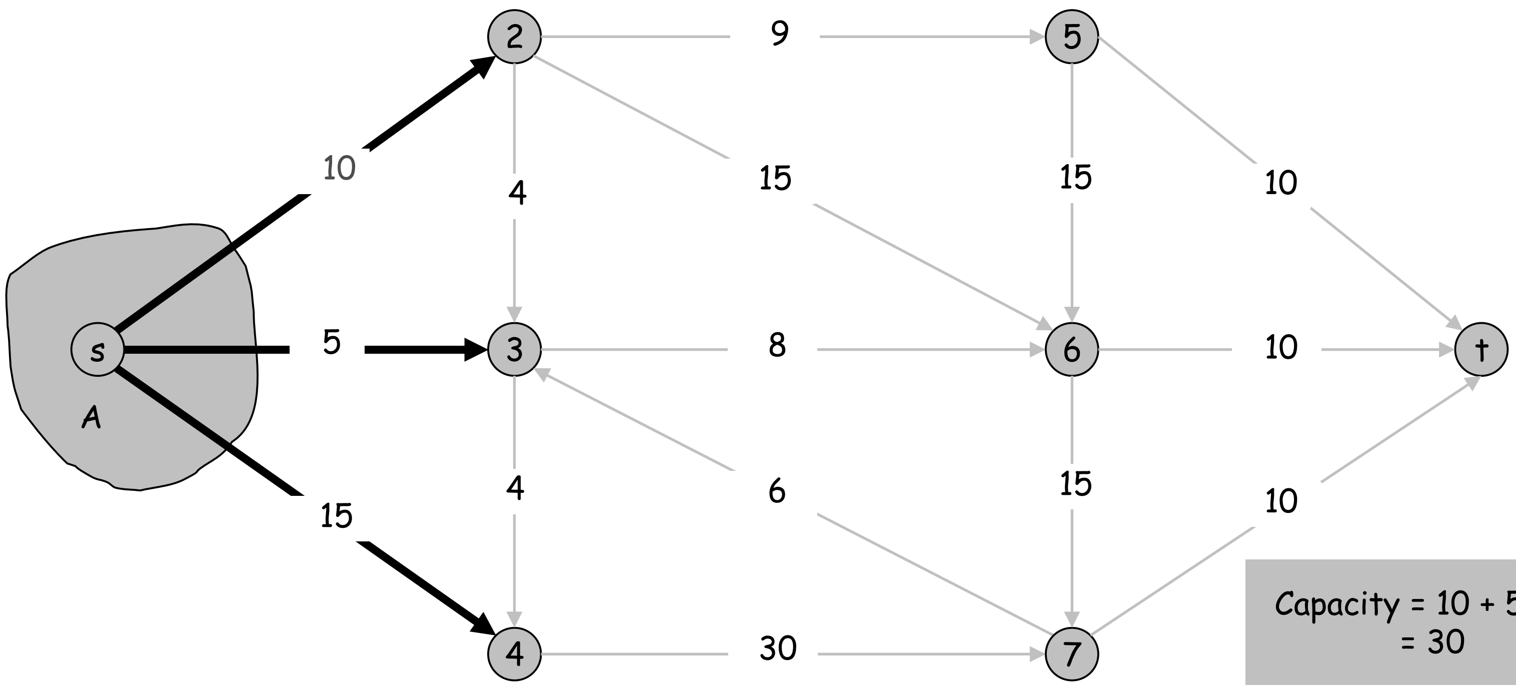


7.2 Max-flow and Min-cut

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

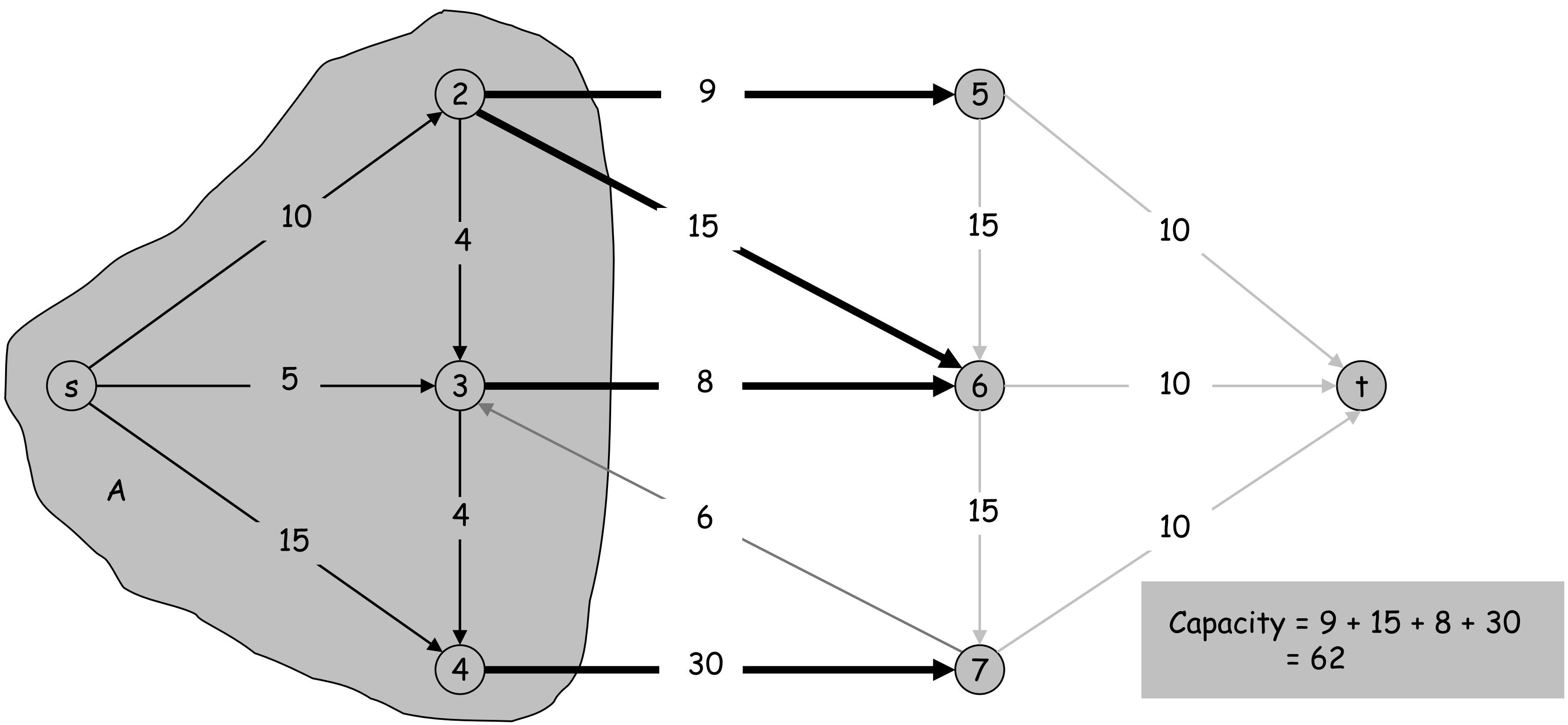


Capacity = 10 + 5 + 15
= 30

Cuts

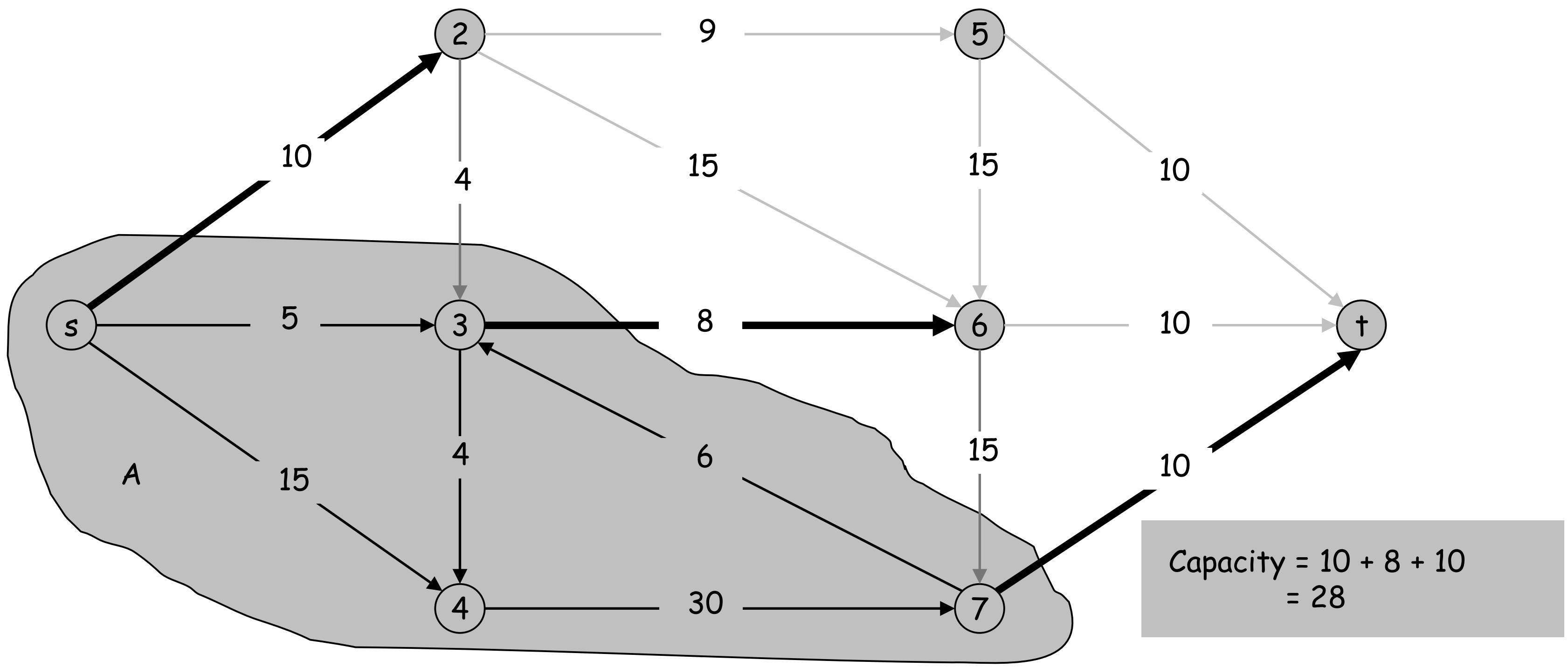
Def. An s-t cut is a partition (A, B) of V with s $\in$ A and t $\in$ B.

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



Minimum Cut Problem

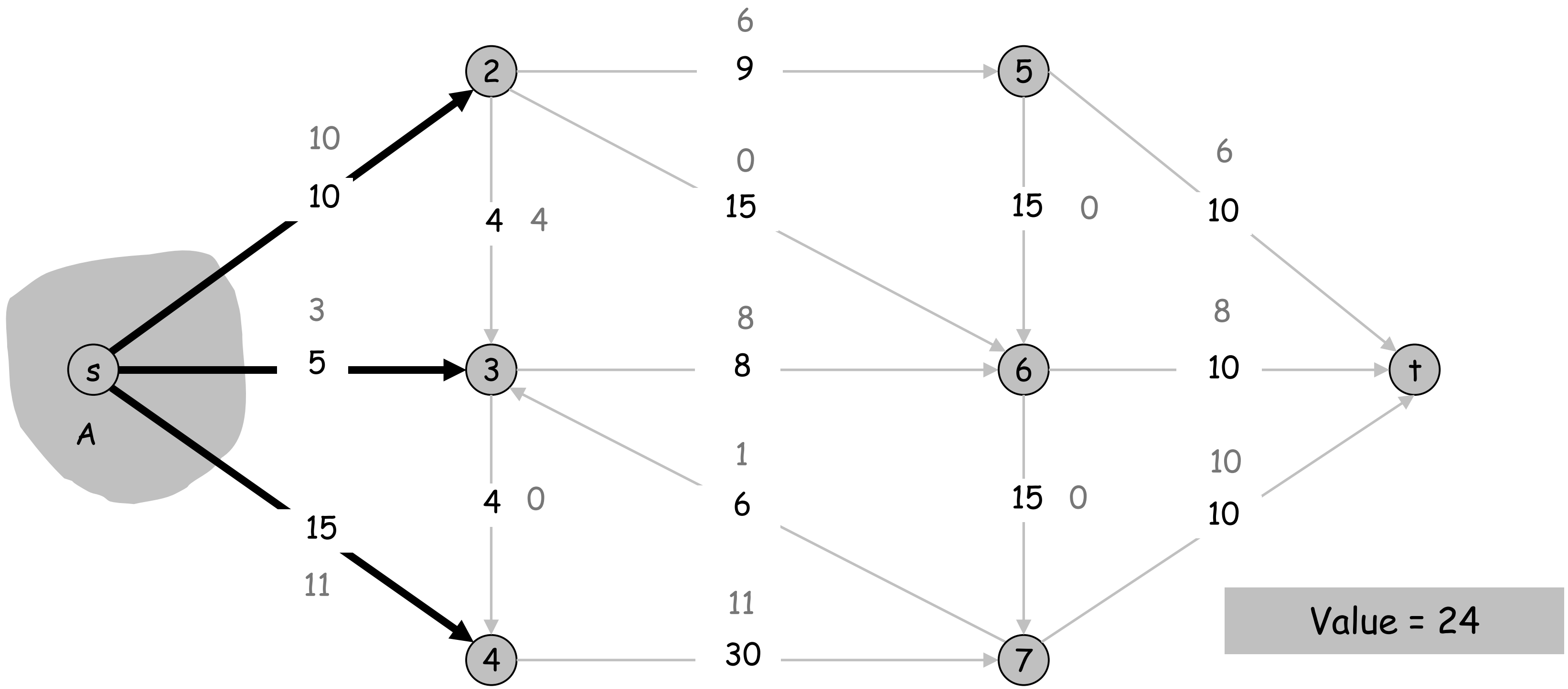
Min s-t cut problem. Find an s-t cut of minimum capacity.



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.

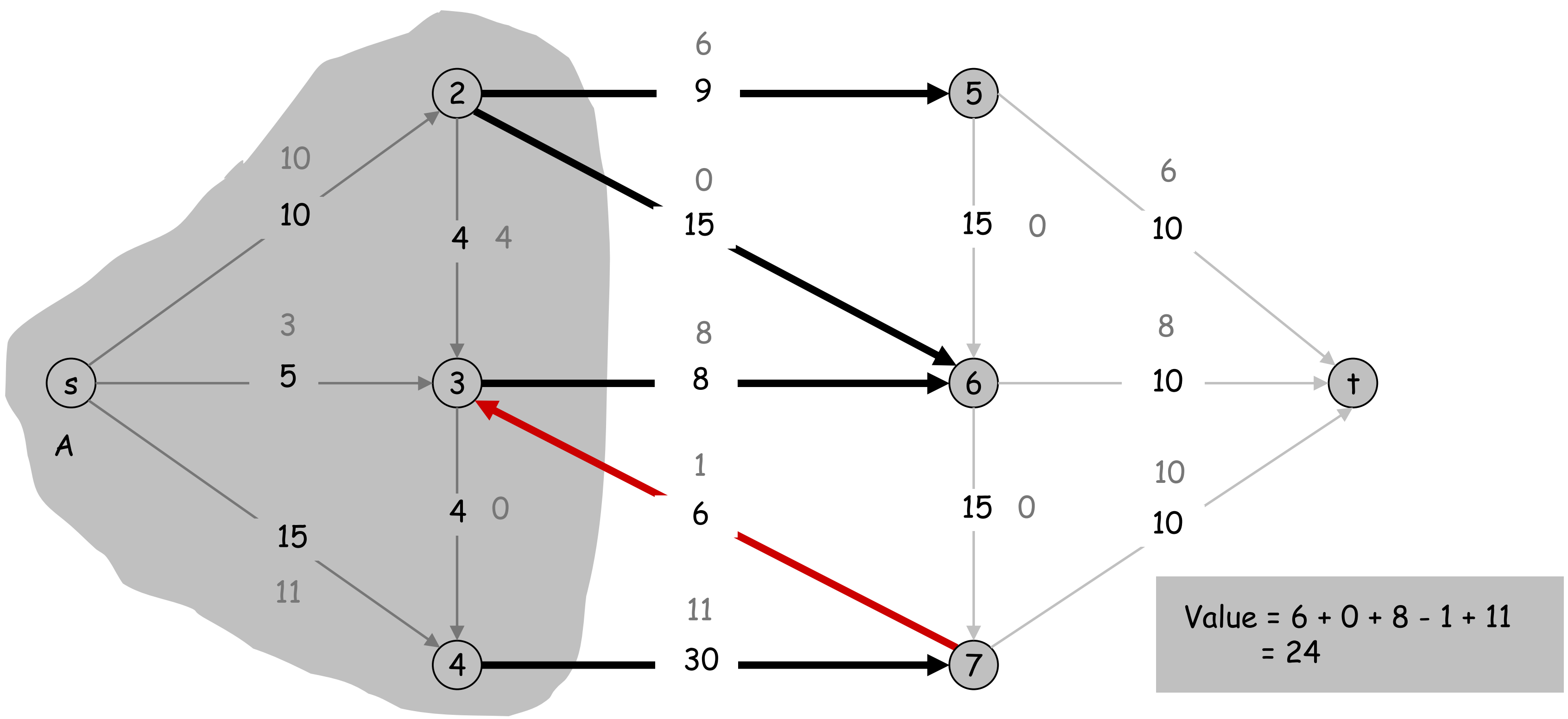
$$\sum_{e \text{ out of } A} f(e) \; - \sum_{e \text{ in to } A} f(e) \; = \; v(f)$$



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.

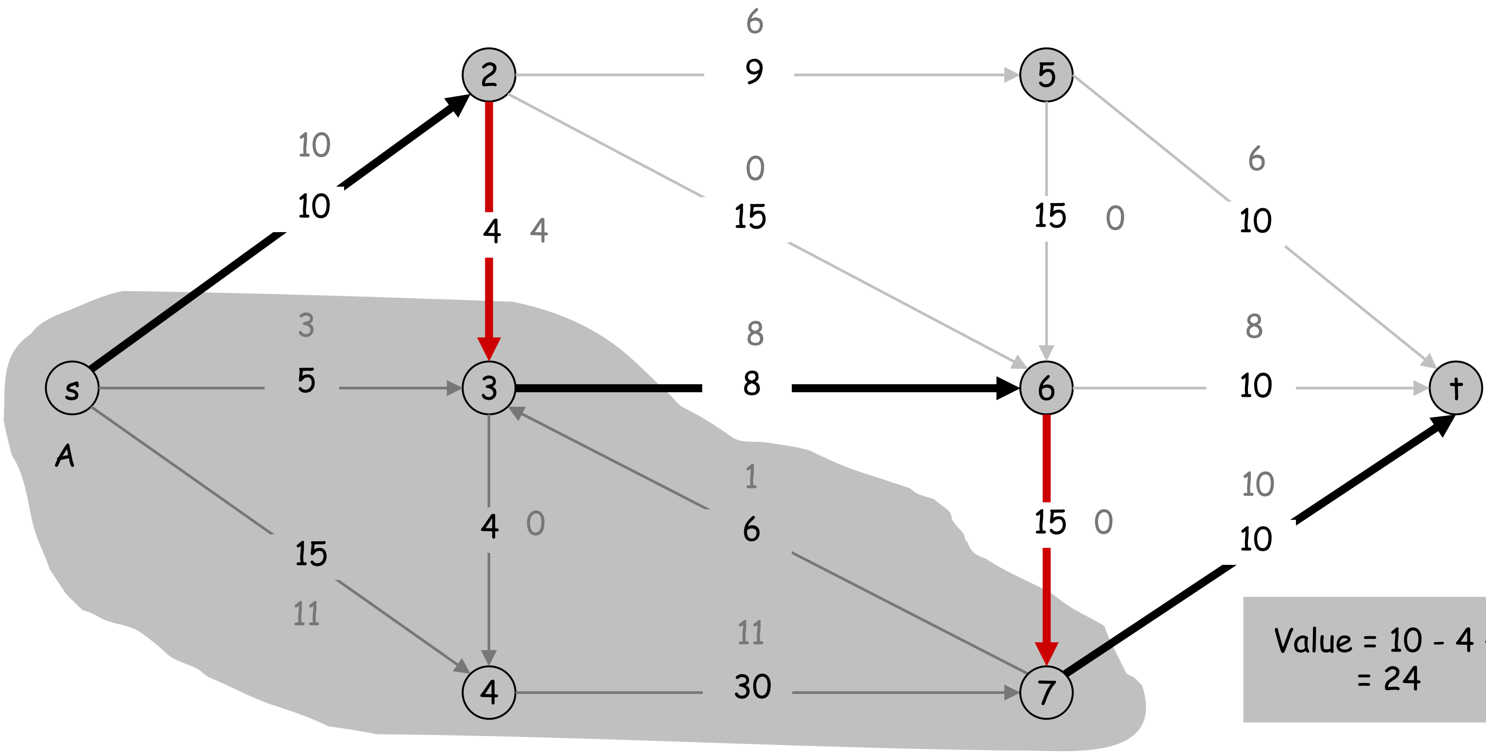
$$\sum_{e \text{ out of } A} f(e) \;-\; \sum_{e \text{ in to } A} f(e) \;=\; v(f)$$



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

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 $\longrightarrow$

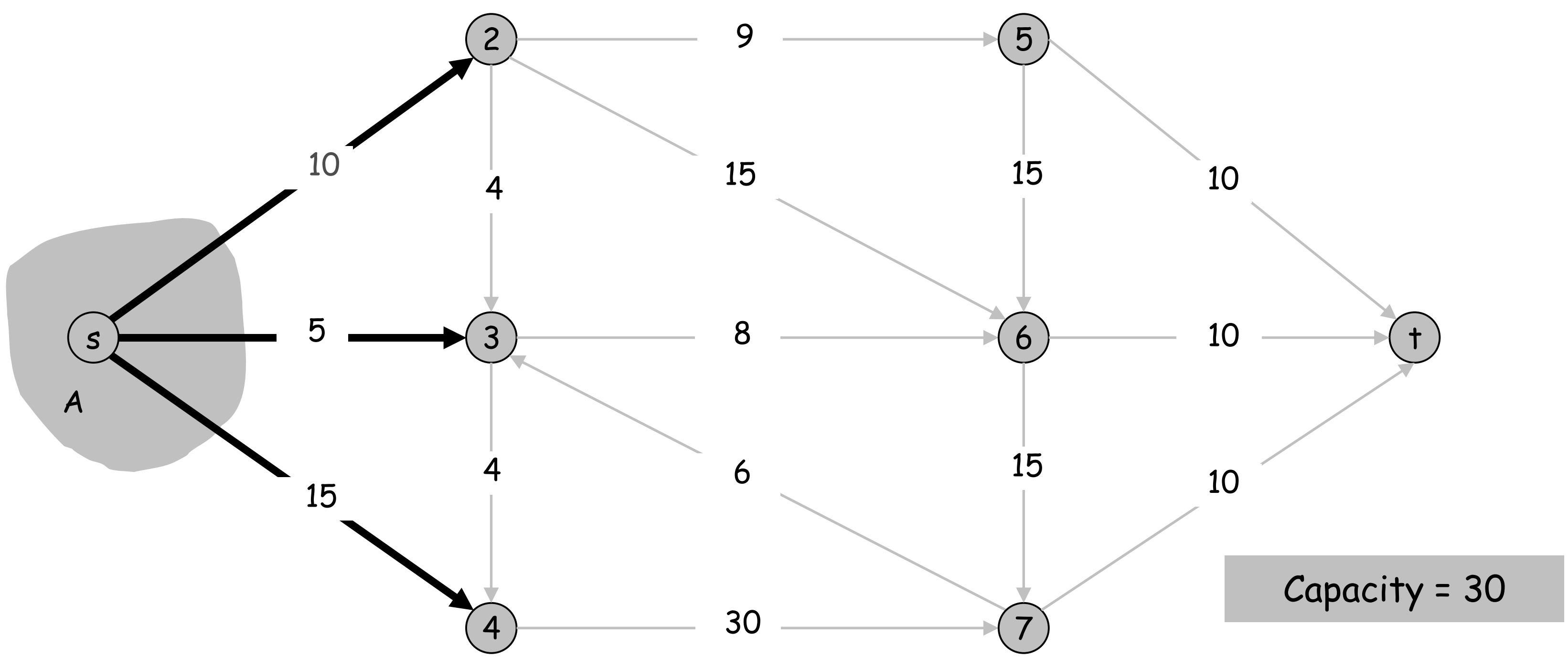
$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to v}} f(e) \right)$$

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

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) $\leq$ cap(A, B).

Pf.

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

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

(ii) $\Rightarrow$ (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

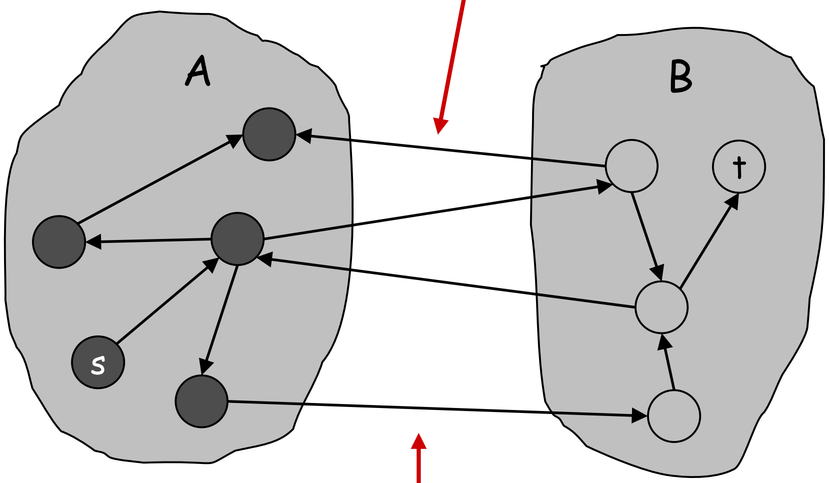
(iii) $\Rightarrow$ (i)

- 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.

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

original network

B->A edge must have
f(e) = 0



A->B edge must have
f(e) = c(e)

7.3 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. ▪

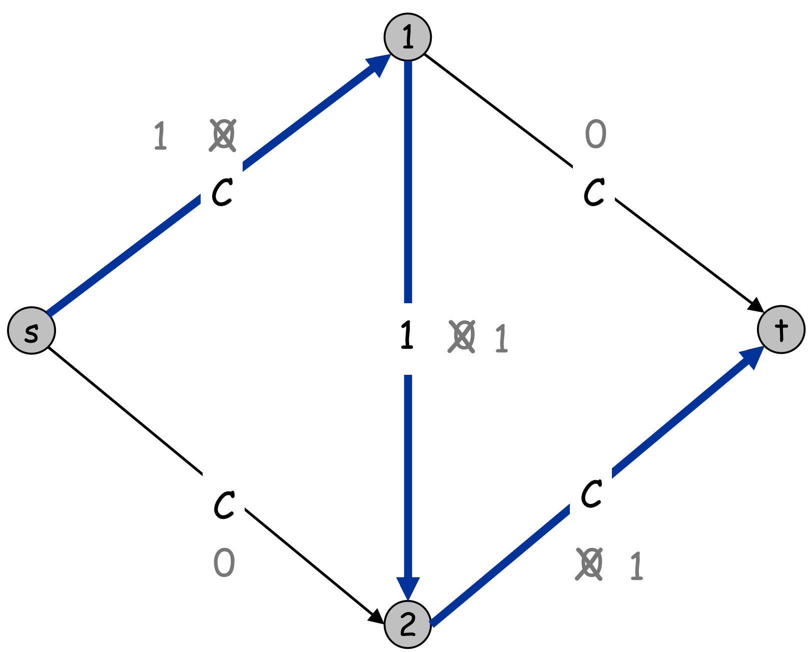
Corollary. Running time of Ford-Fulkerson is $O(mnC)$ $\longleftarrow$ Polynomial?

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.

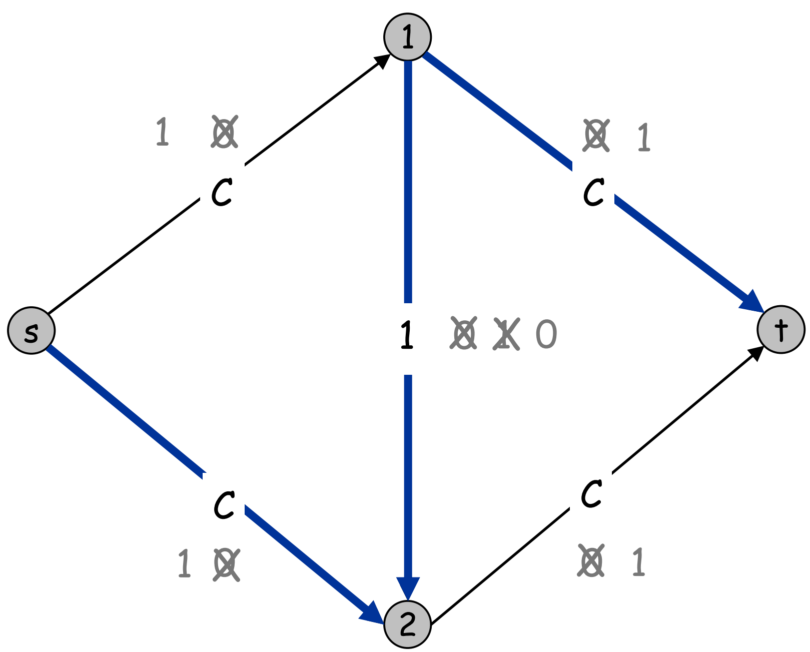


Ford-Fulkerson: Exponential Number of Augmentations

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.

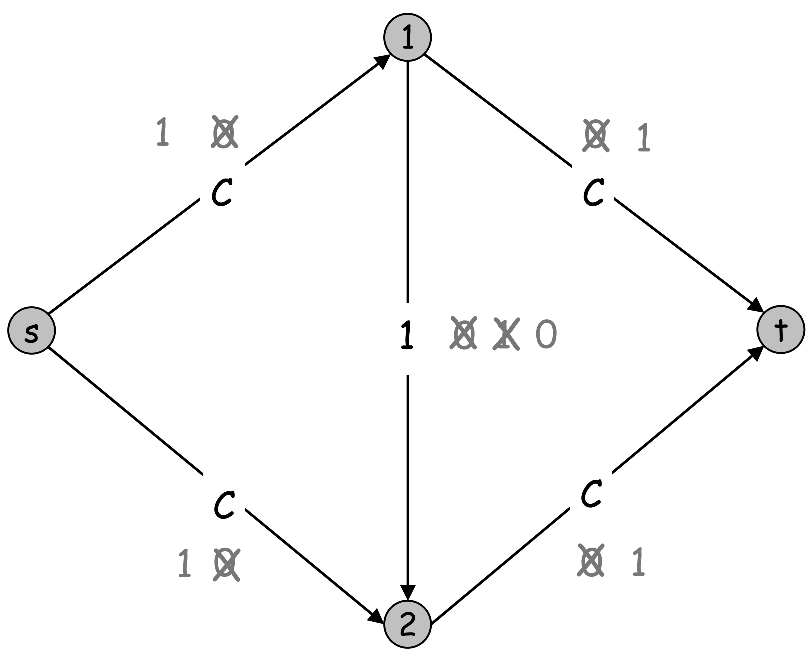


Ford-Fulkerson: Exponential Number of Augmentations

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 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.

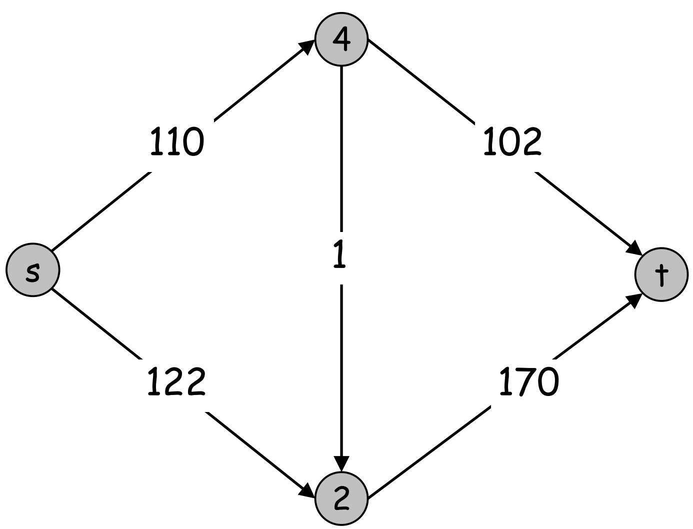
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

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

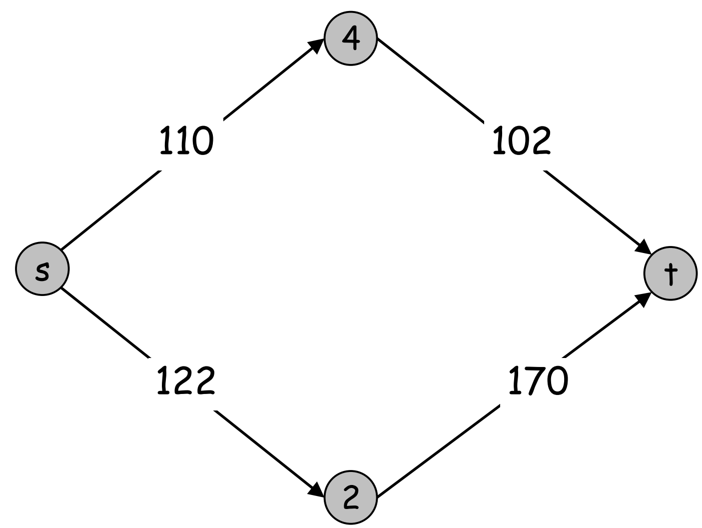
Capacity Scaling

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) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(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 $C/2 < \Delta \leq C$. Δ decreases by a factor of 2 each iteration. ▪

Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then the value of the maximum flow is at most $v(f) + m\,\Delta$. ← proof on next slide

Lemma 3. There are at most 2m augmentations per scaling phase.
- Let f be the flow at the end of the previous scaling phase.
- 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. ▪

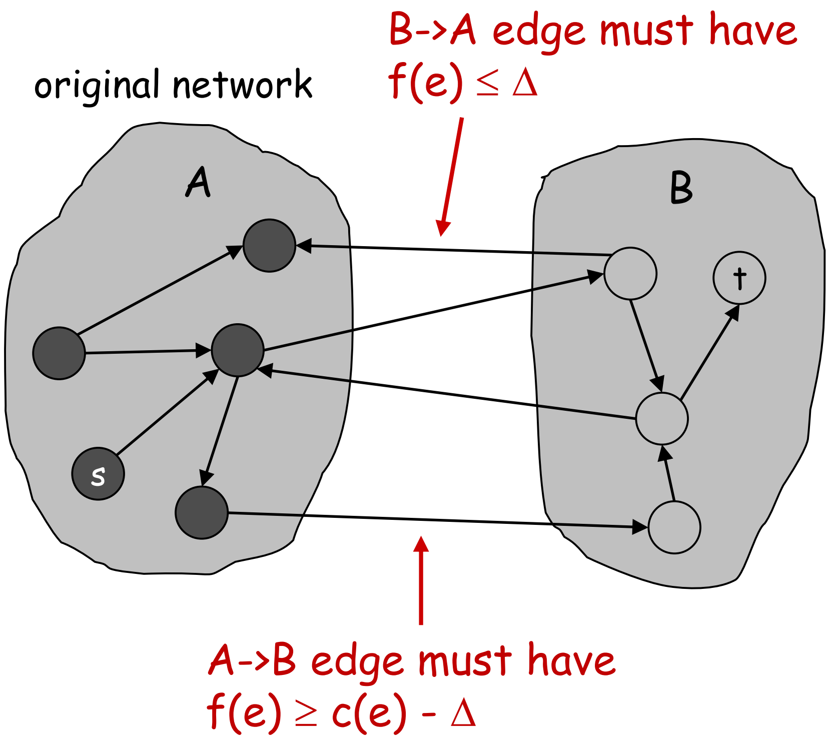
Capacity Scaling: Running Time

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)

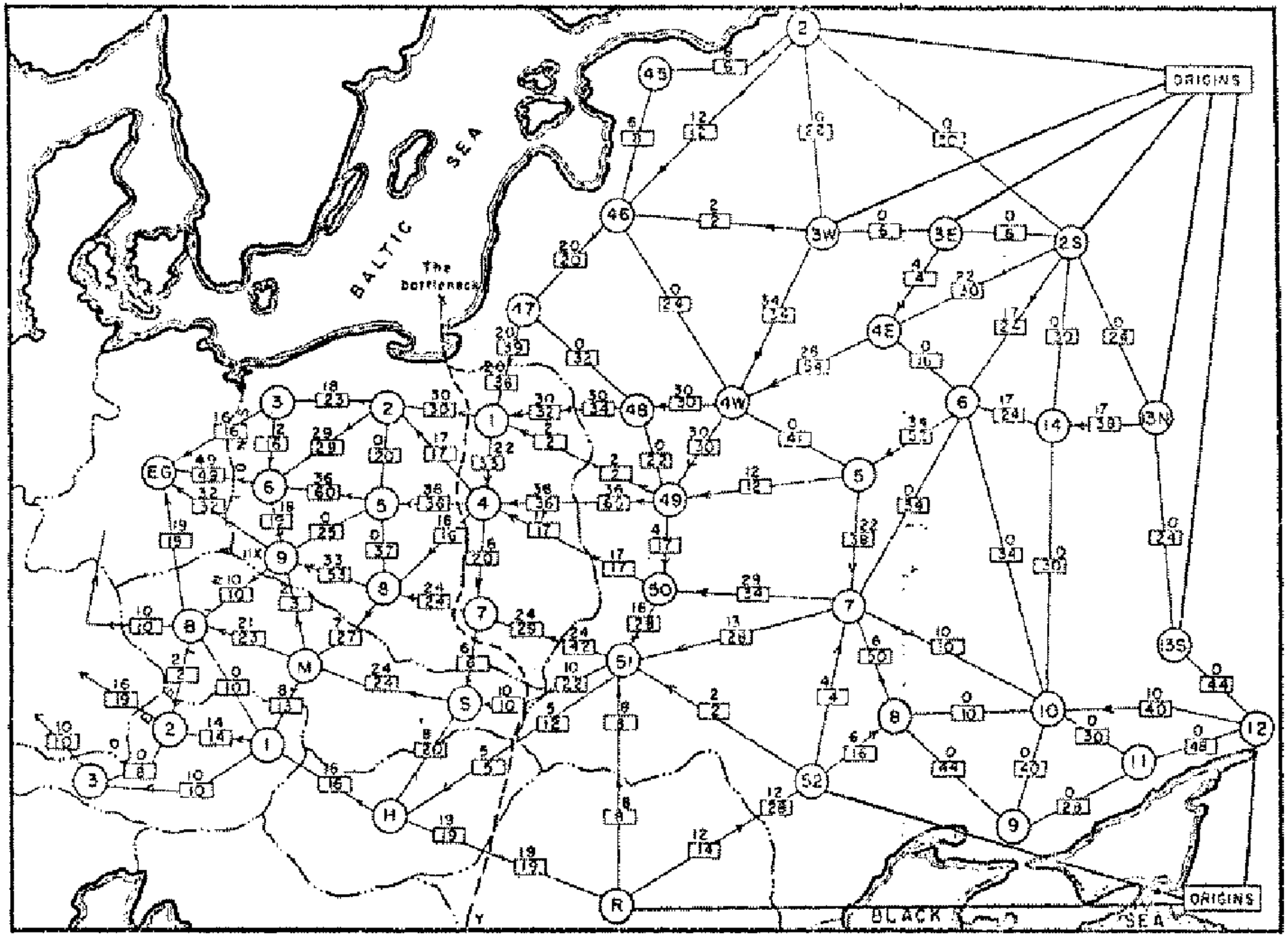
- We show that at the end of a Δ-phase, there exists a cut (A, B) such that $\text{cap}(A, B) \leq v(f) + m\,\Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.

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



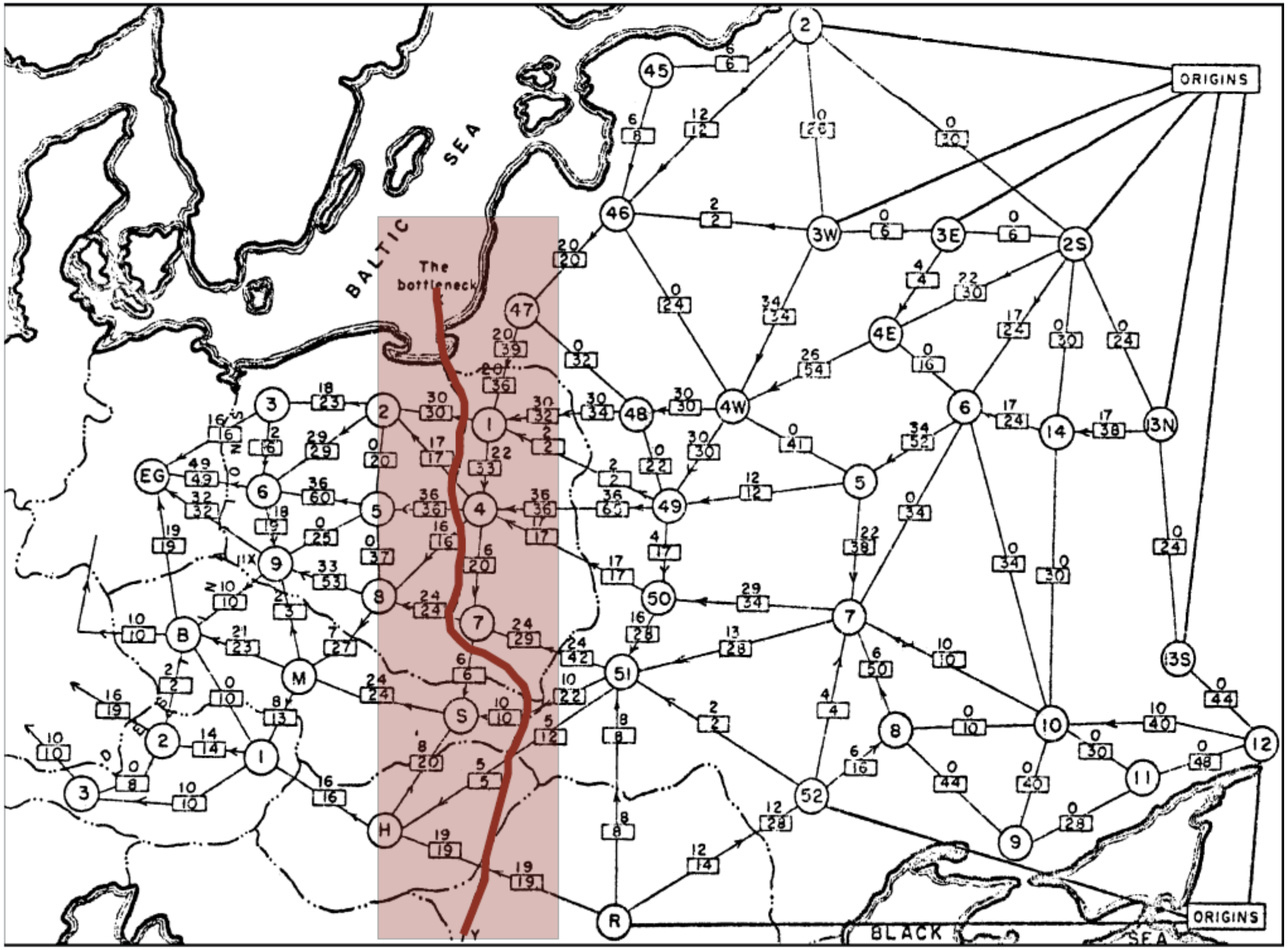
Applications of max-flow

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



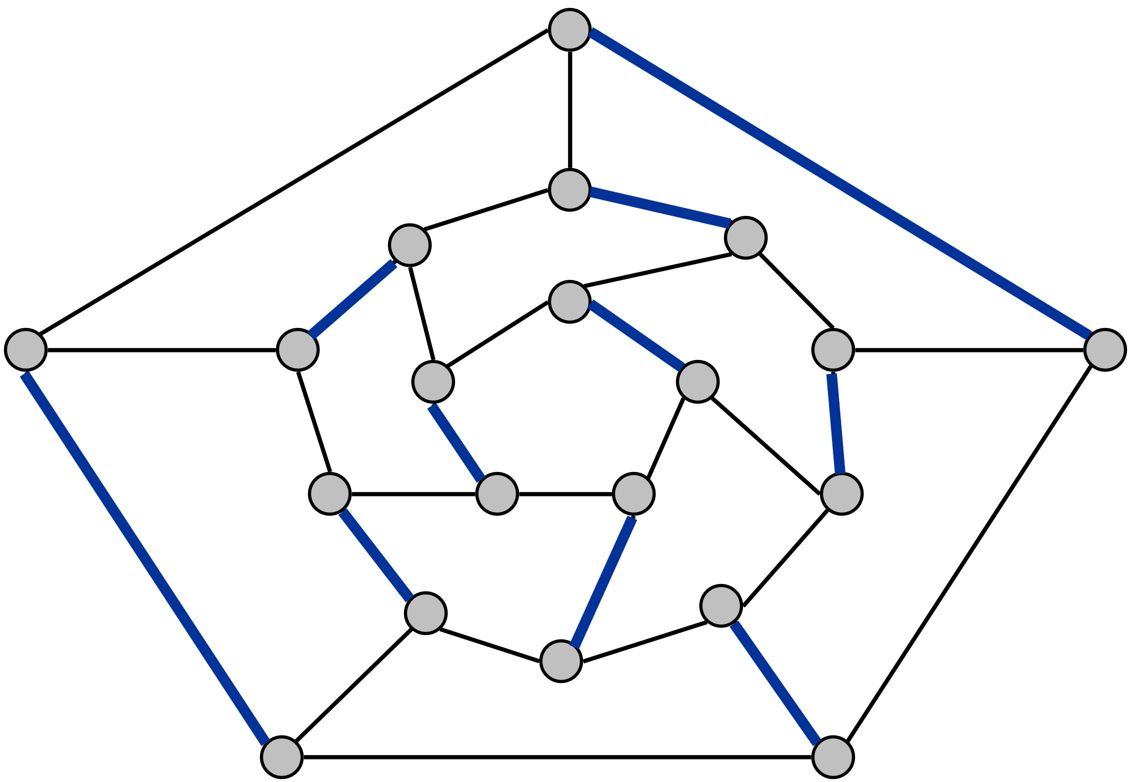
Reference: *On the history of the transportation and maximum flow problems.* Alexander Schrijver in Math Programming, 91: 3, 2002.

7.5 Bipartite Matching

Matching

Matching.

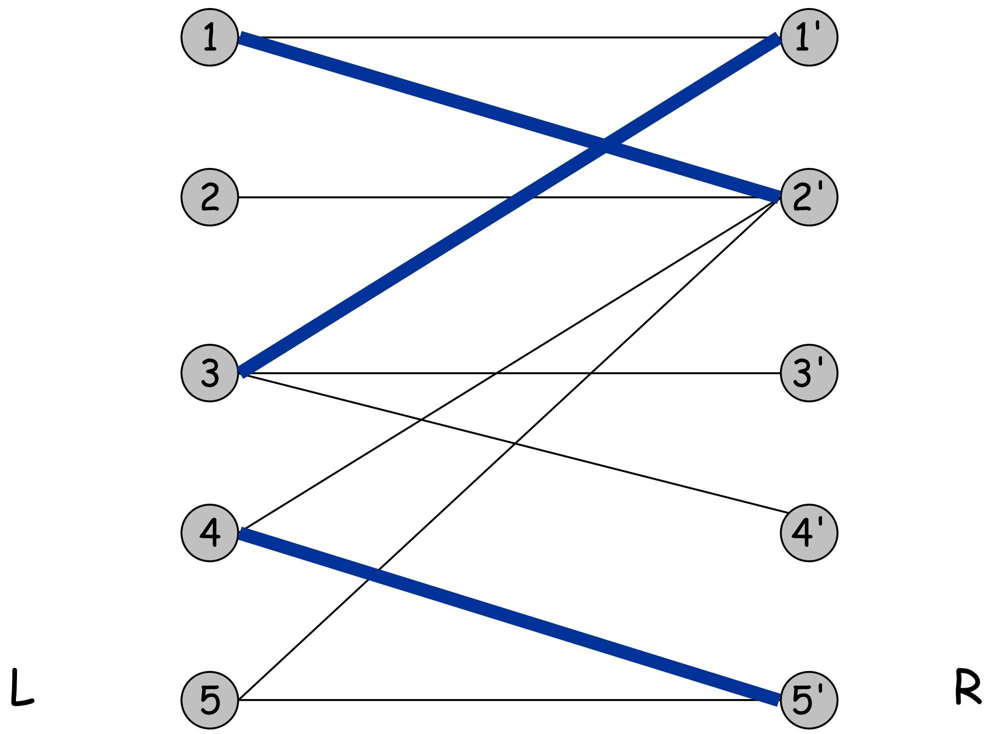
- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a matching if each node appears in at most one edge in M.
- Max matching: find a max cardinality matching.



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.



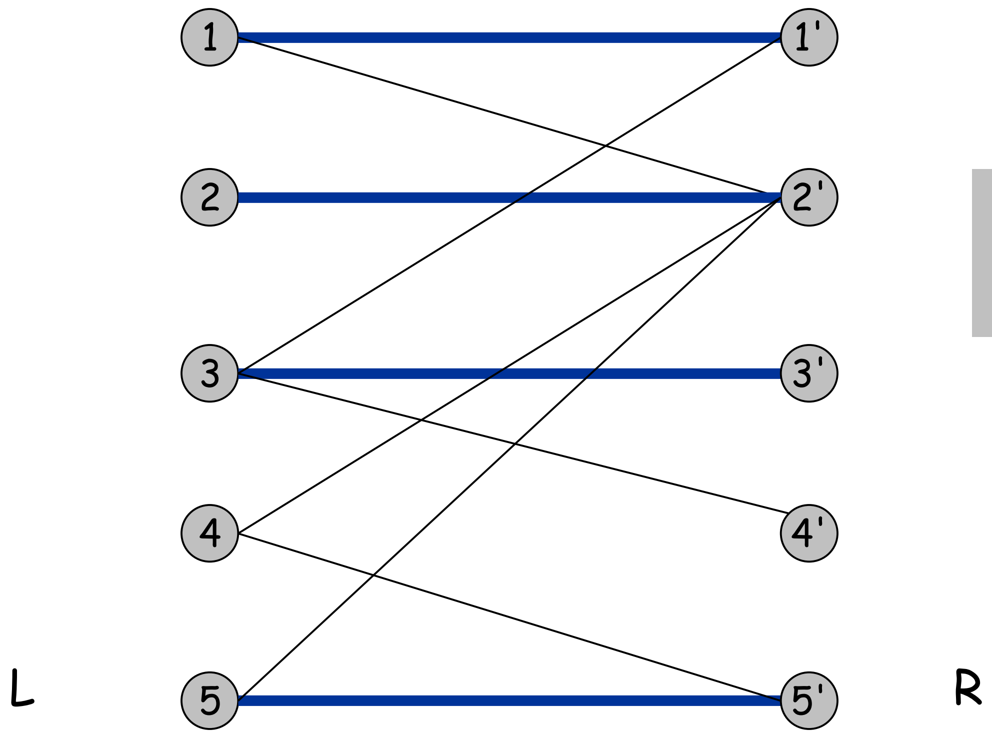
matching

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.



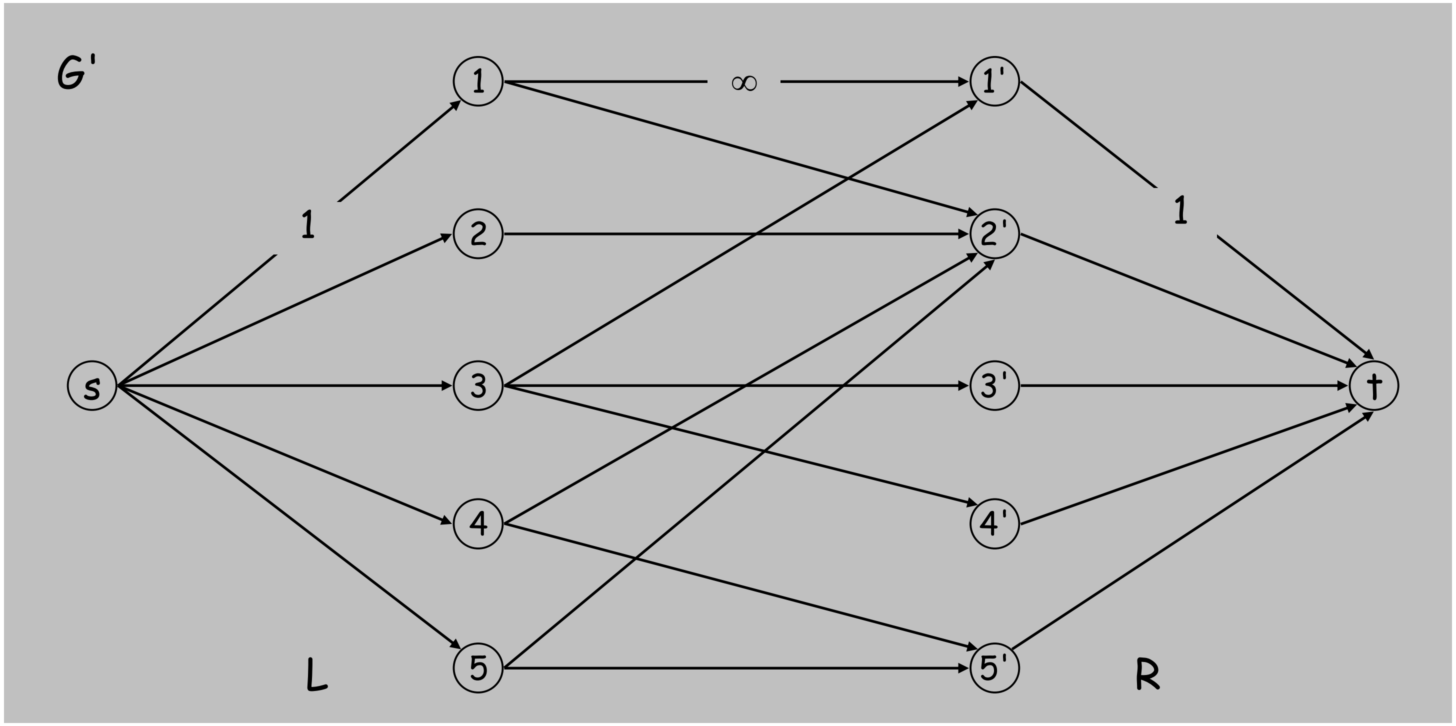
max matching

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.

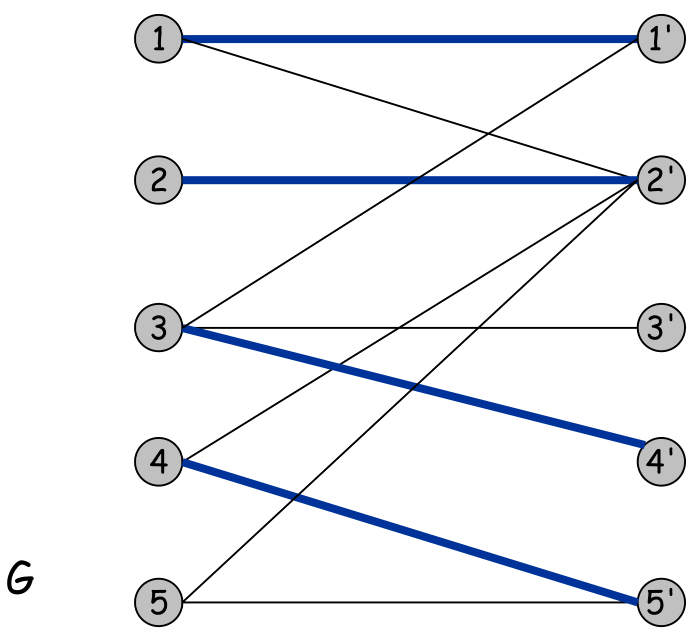


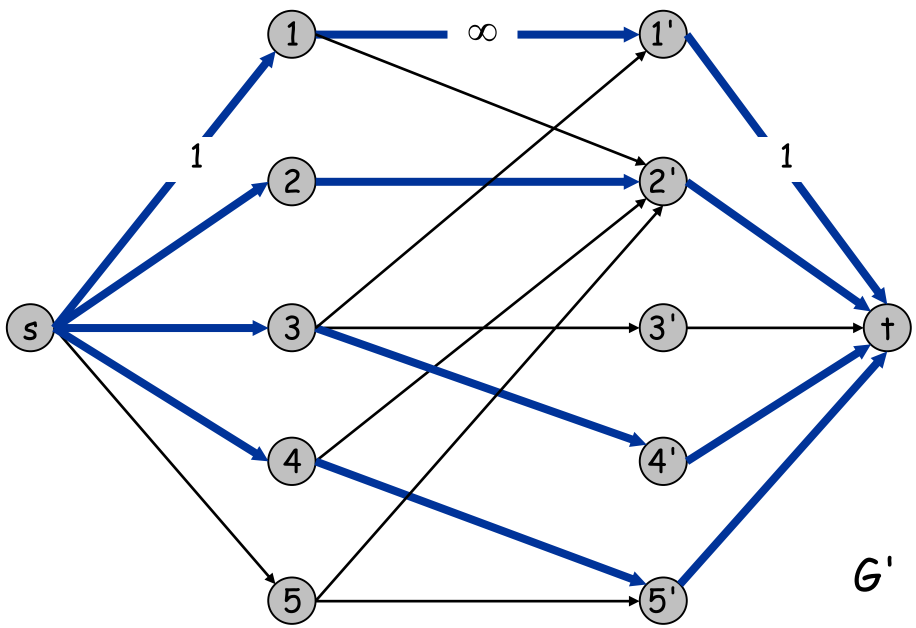
Bipartite Matching: Proof of Correctness

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. ▪



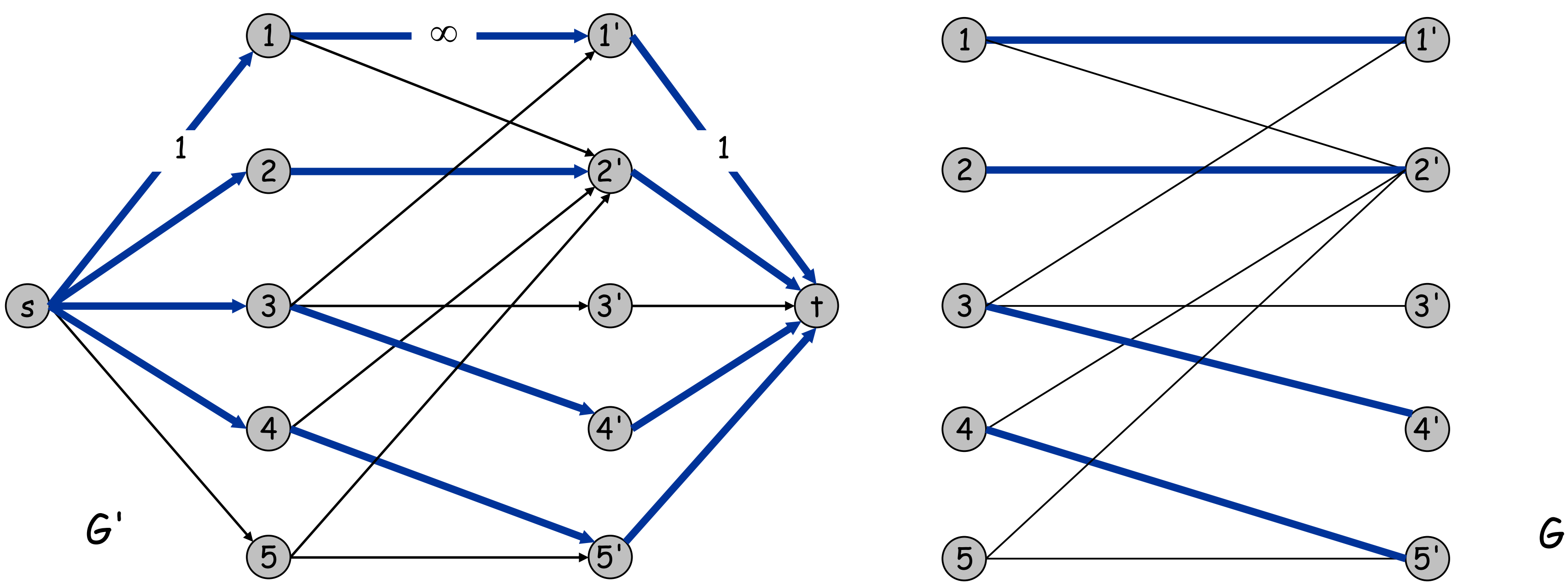


Bipartite Matching: Proof of Correctness

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.

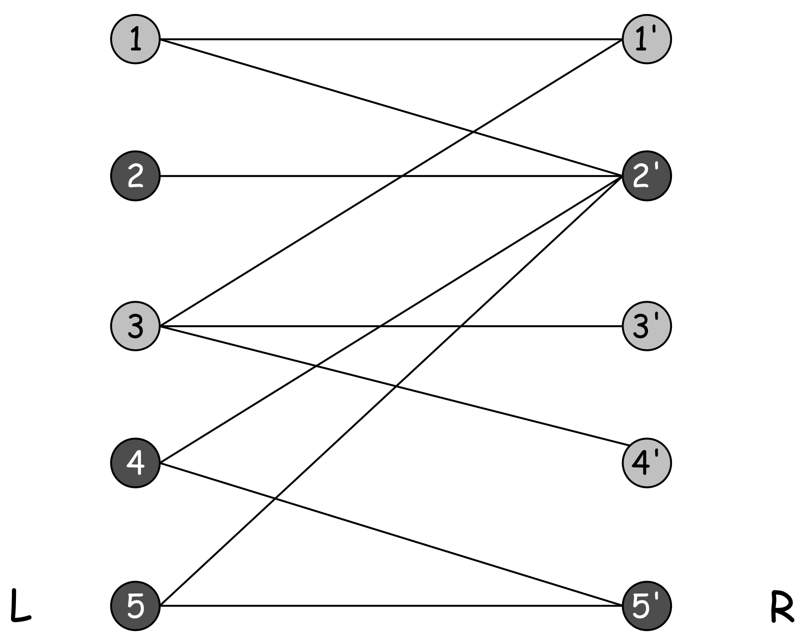
- Clearly we must have $|L| = |R|$.
- What other conditions are necessary?
- What conditions are sufficient?

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:

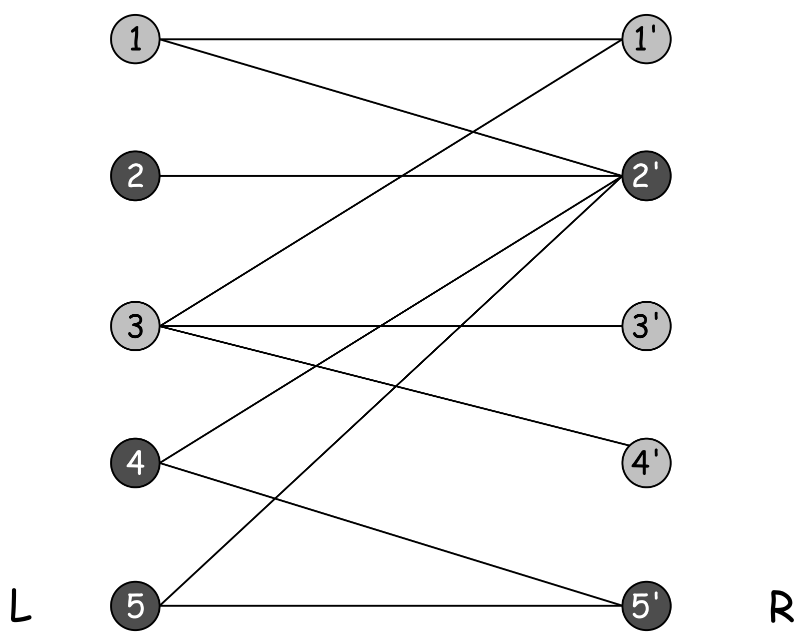
S = { 2, 4, 5 }

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

Marriage Theorem. G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

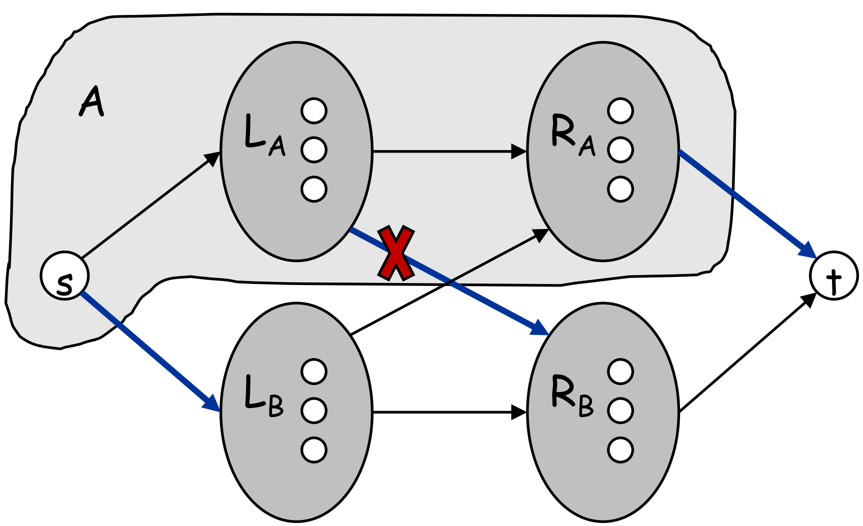
Pf. $\Leftarrow$ Suppose G does not have a perfect matching.

- Formulate as a max flow problem and let (A, B) be min cut in G'.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$, $R_B = R \cap B$.
- $cap(A, B) = v(f^*) = |M| < |L|$ ("<": because no perfect matching)
- Since min cut can't use ∞ edges, no edge between L_A and R_B
  - $cap(A, B) = |L_B| + |R_A|$
  - $N(L_A) \subseteq R_A$.
- $|N(L_A)| \leq |R_A|$

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

  $< |L| - |L_B|$

  $= |L_A|$.
- This contradicts the condition ▪



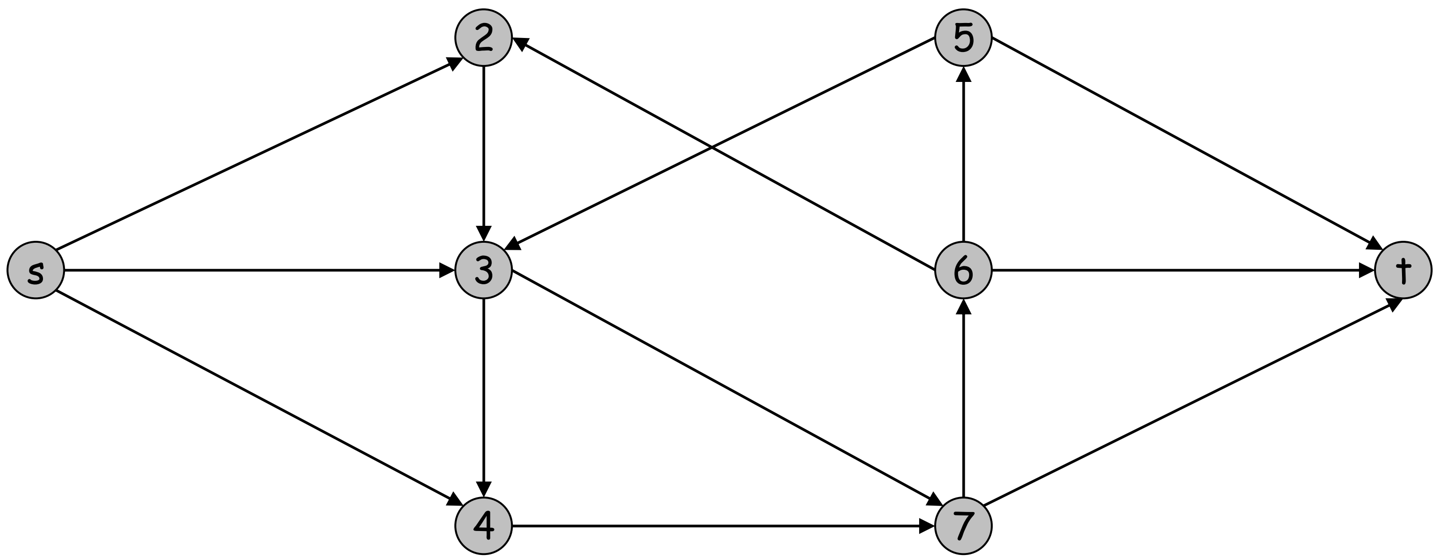
7.6 Disjoint Paths

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.

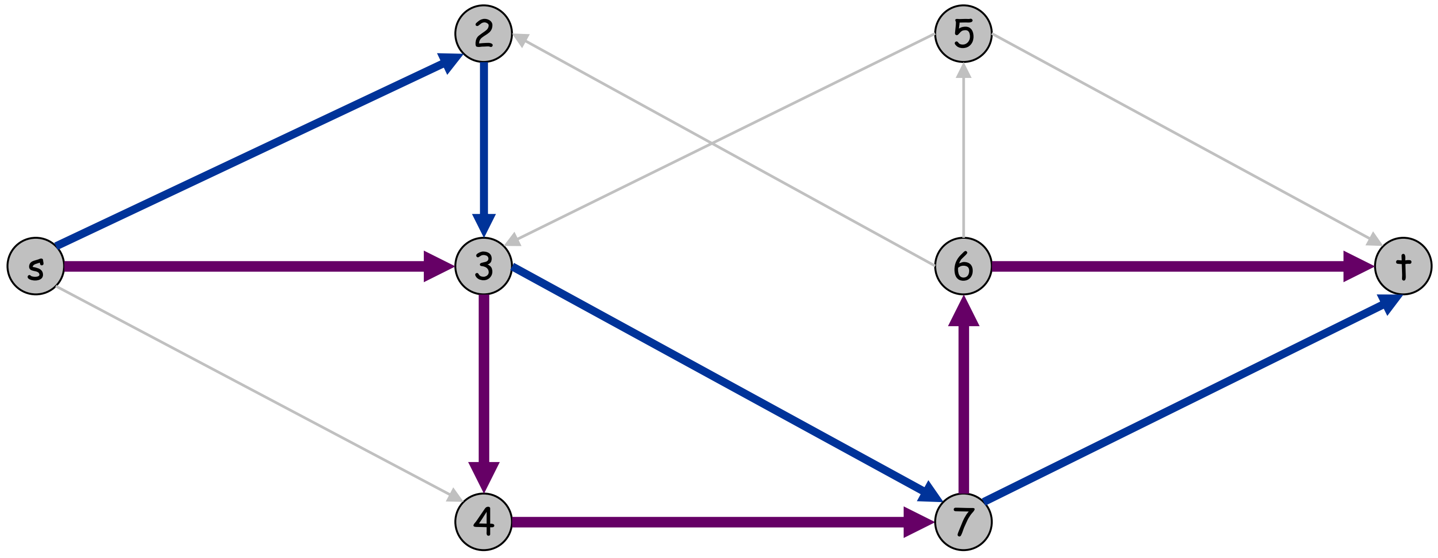


Edge Disjoint Paths

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.

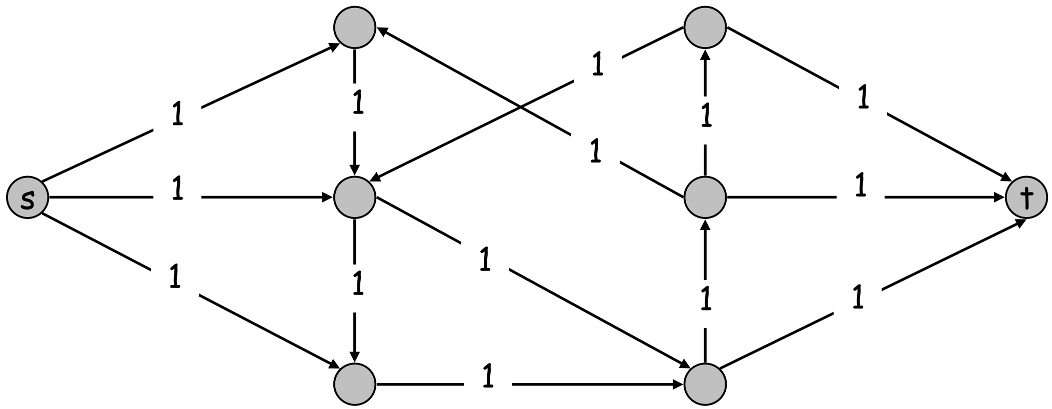
Def. Two paths are edge-disjoint if they have no edge in common.

Ex: communication networks.



Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



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

Edge Disjoint Paths

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

Pf. $\leq$

- Suppose there are k edge-disjoint paths $P_1, \ldots, P_k$.
- 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

7.7 Extensions to Max Flow

Circulation with Demands

Circulation with demands.

- Directed graph G = (V, E).
- Edge capacities c(e), e ∈ E.
- Node supply and demands d(v), v ∈ 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 ∈ E: $0 \le f(e) \le c(e)$ (capacity)
- For each v ∈ V: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

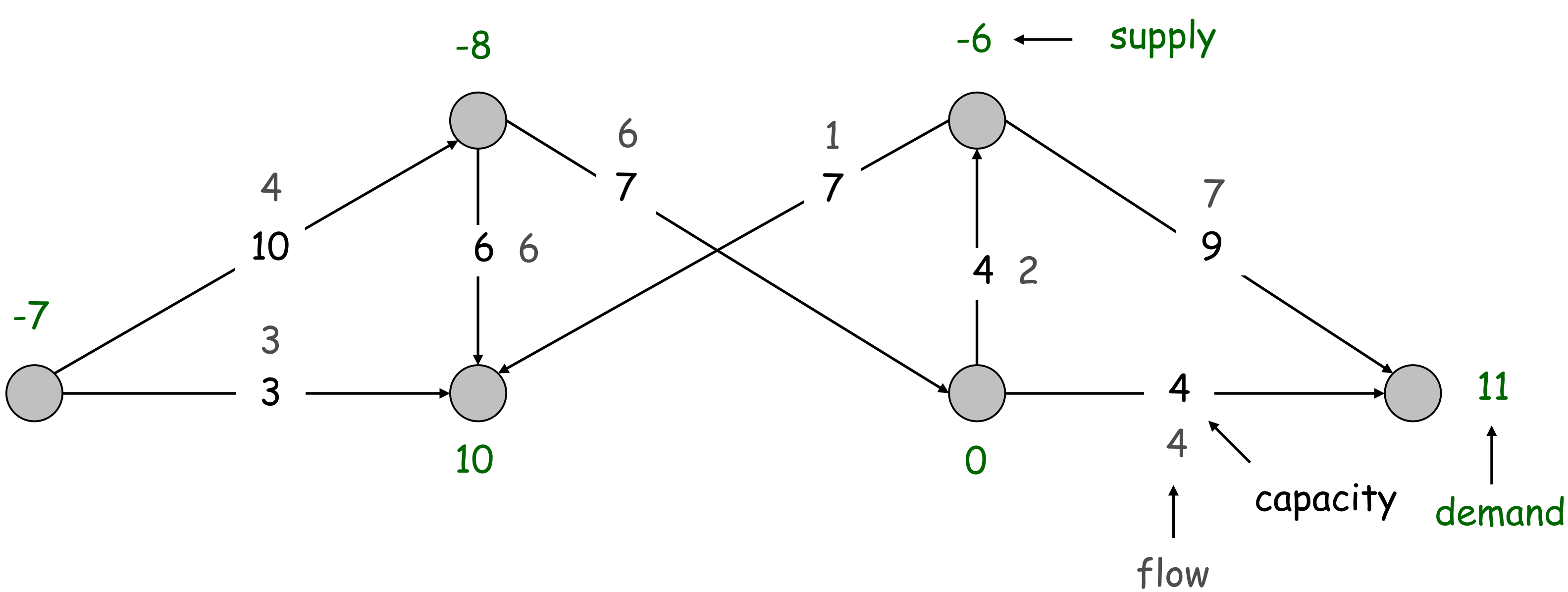
Circulation problem: given (V, E, c, d), does there exist a circulation?

Circulation with Demands

Necessary condition: sum of supplies = sum of demands.

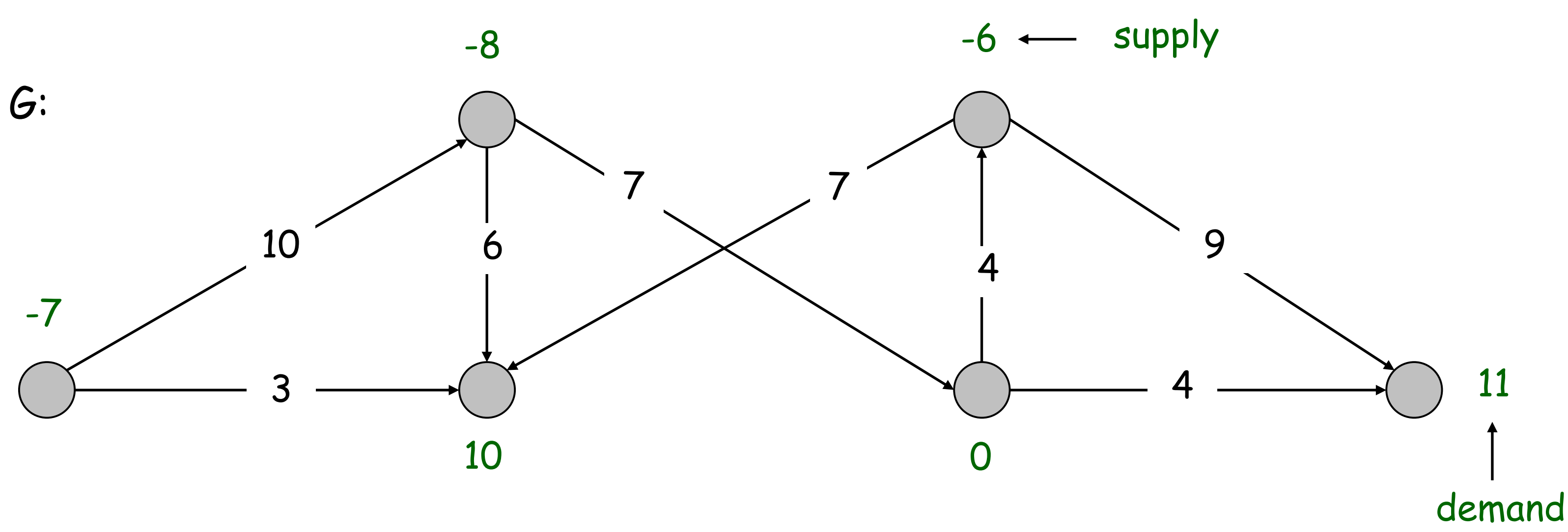
$$\sum_{v\,:\,d(v)>0} d(v) \;=\; \sum_{v\,:\,d(v)<\,0} -d(v) \;=:\; D$$

Pf. Sum conservation constraints for every demand node v.



Circulation with Demands

Max flow formulation.



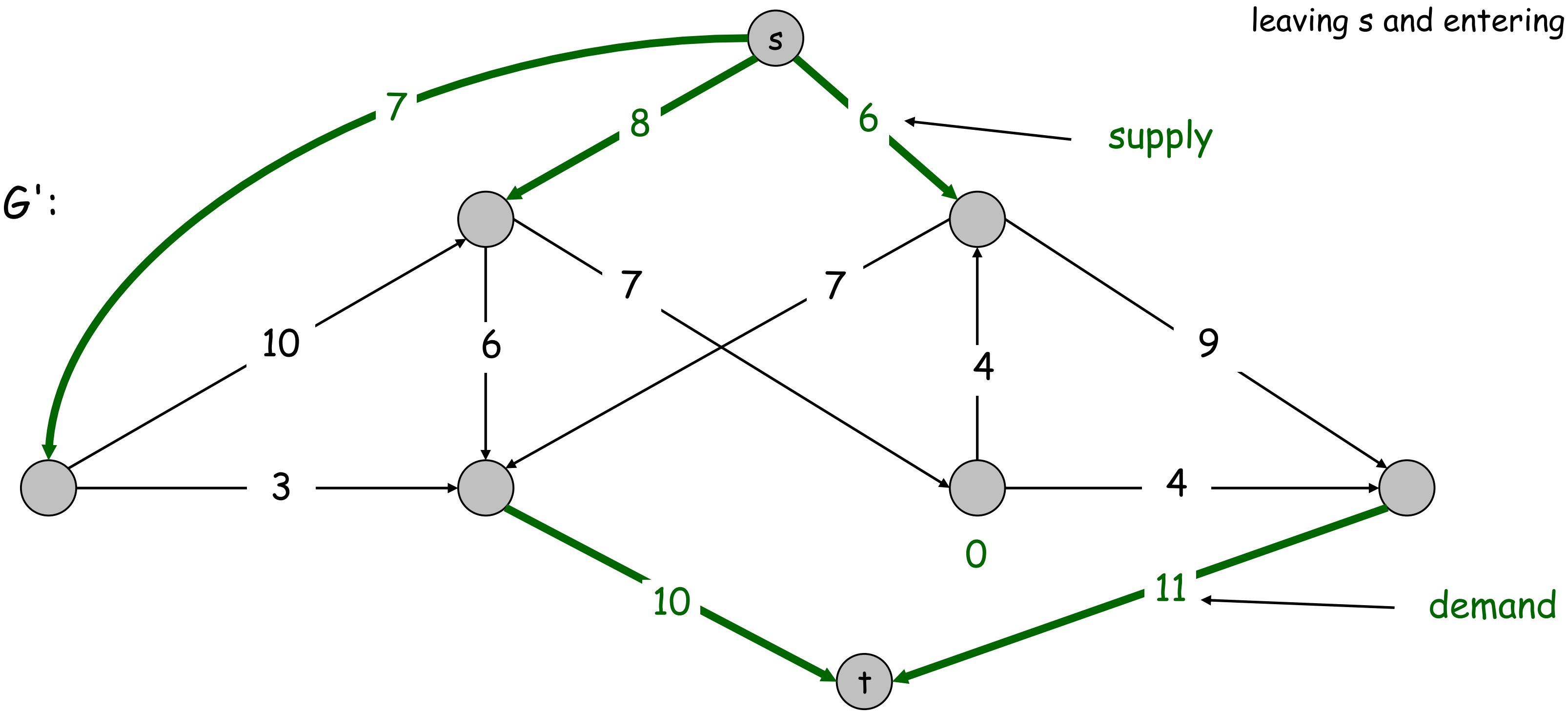
Circulation with Demands

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.

saturates all edges
leaving s and entering t

G':



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 exists a circulation iff there exists a node partition (A, B) such that $\Sigma_{v \in B} d_v > \text{cap}(A, B)$

Pf idea. Look at max flow and 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

Circulation with Demands and Lower Bounds

Feasible circulation.

- Directed graph G = (V, E).
- Edge capacities c(e) and lower bounds ℓ (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 exists 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) - ℓ(e) is a circulation in G'.

7.8 Survey Design

Survey Design

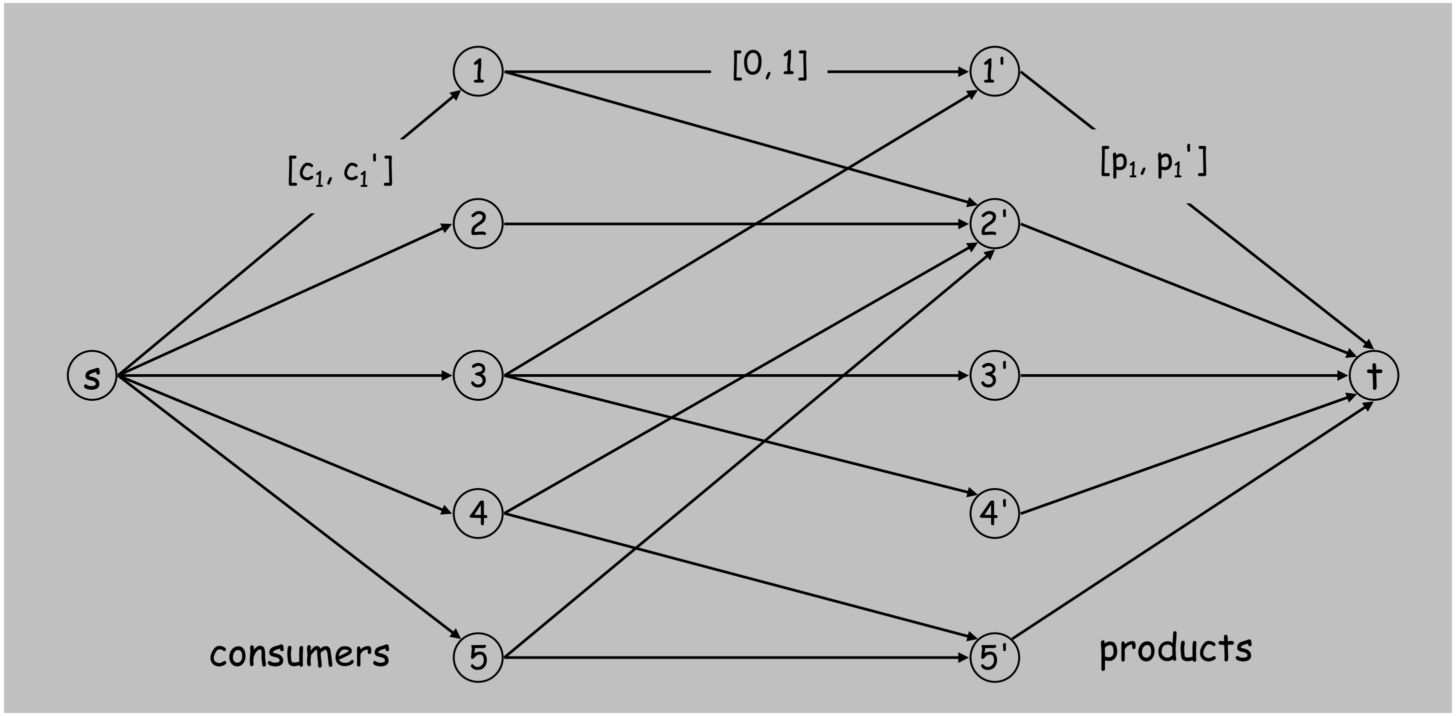
Survey design.

- Design survey asking n_1 consumers about n_2 products.
- Can only survey consumer i about a product j if they own it.
- Ask consumer i between c_i and c_i' questions.
- Ask between p_j and p_j' consumers about product j.

Goal. Design a survey that meets these specs, if possible.

Survey Design

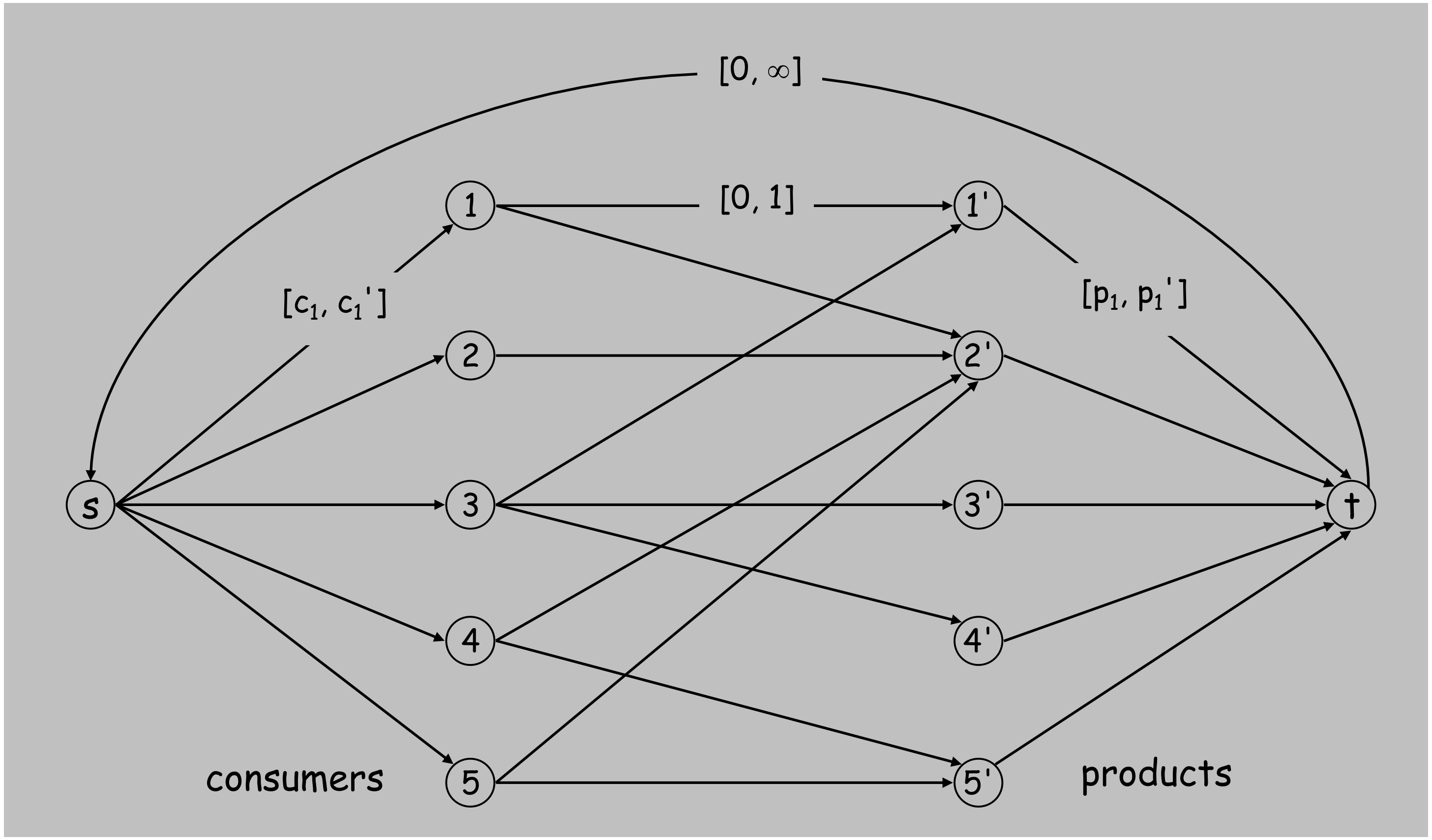
Algorithm. Formulate as a flow-network?
- Include an edge (i, j) if customer own product i.
- Goal: find a flow that satisfies edge upper&lower bounds. How?



Survey Design

Algorithm. Formulate as a circulation problem with lower bounds.

- Include an edge (i, j) if customer own product i.
- Integer circulation $\Leftrightarrow$ feasible survey design.



7.10 Image Segmentation

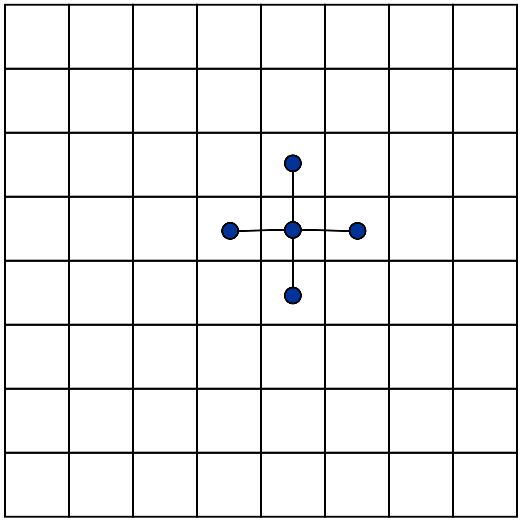
Image Segmentation

Image segmentation.

- Central problem in image processing.
- Divide image into coherent regions.

Ex: Two people standing in front of complex background scene. Identify each person as a coherent object.

Image Segmentation



Foreground / background segmentation.

- Label each pixel in picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighboring pixels.
- $a_i \geq 0$ is likelihood pixel i in foreground.
- $b_i \geq 0$ is likelihood pixel i in background.
- $p_{ij} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.

Goals.

- Accuracy: if $a_i > b_i$ in isolation, prefer to label i in foreground.
- Smoothness: if many neighbors of i are labeled foreground, we should be inclined to label i as foreground.
- Find partition (A, B) that maximizes: $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

foreground background

Image Segmentation

Formulate as min cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

Turn into minimization problem.

- Maximizing $\displaystyle \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

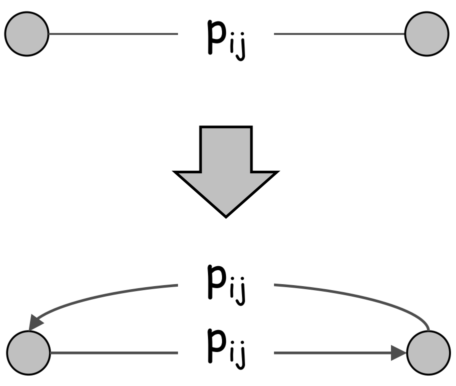
  is equivalent to minimizing $\displaystyle \underbrace{\left(\sum_{i \in V} a_i + \sum_{j \in V} b_j\right)}_{\text{a constant}} - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

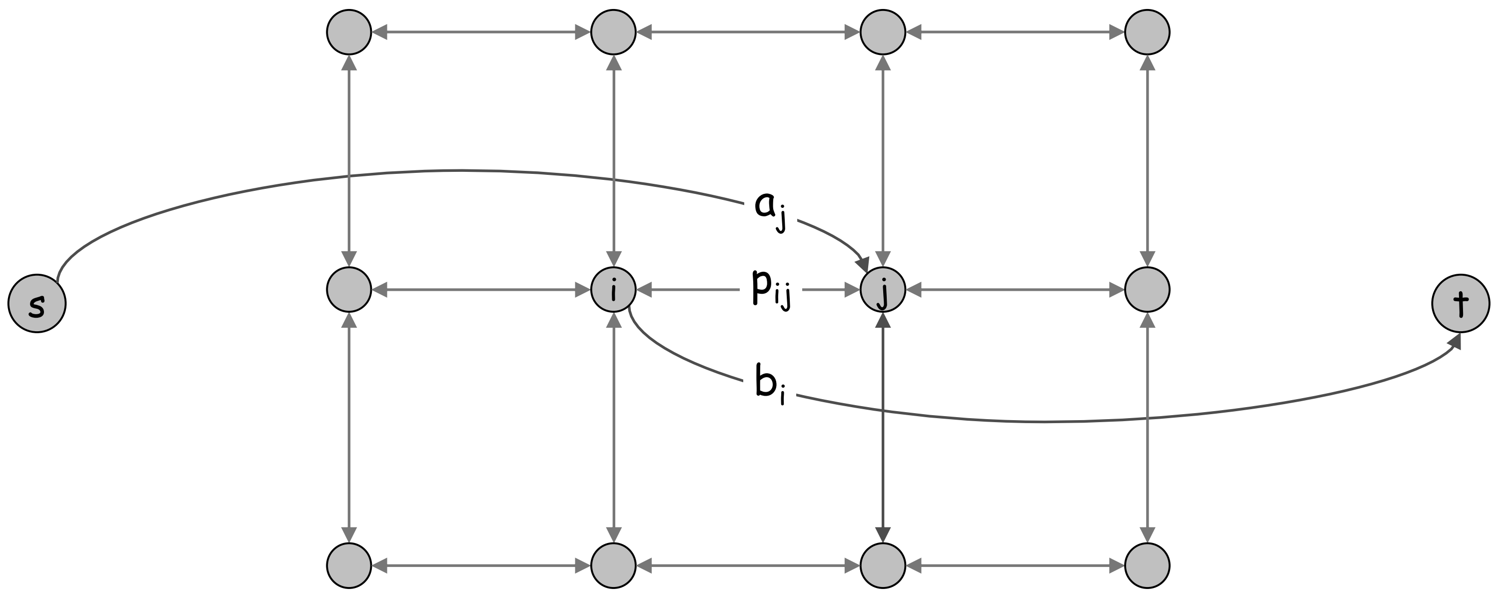
- or alternatively $\displaystyle \sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$

Image Segmentation

Formulate as min cut problem.

- $G' = (V', E')$.
- Add source to correspond to foreground; add sink to correspond to background
- Use two anti-parallel edges instead of undirected edge.





G'

Image Segmentation

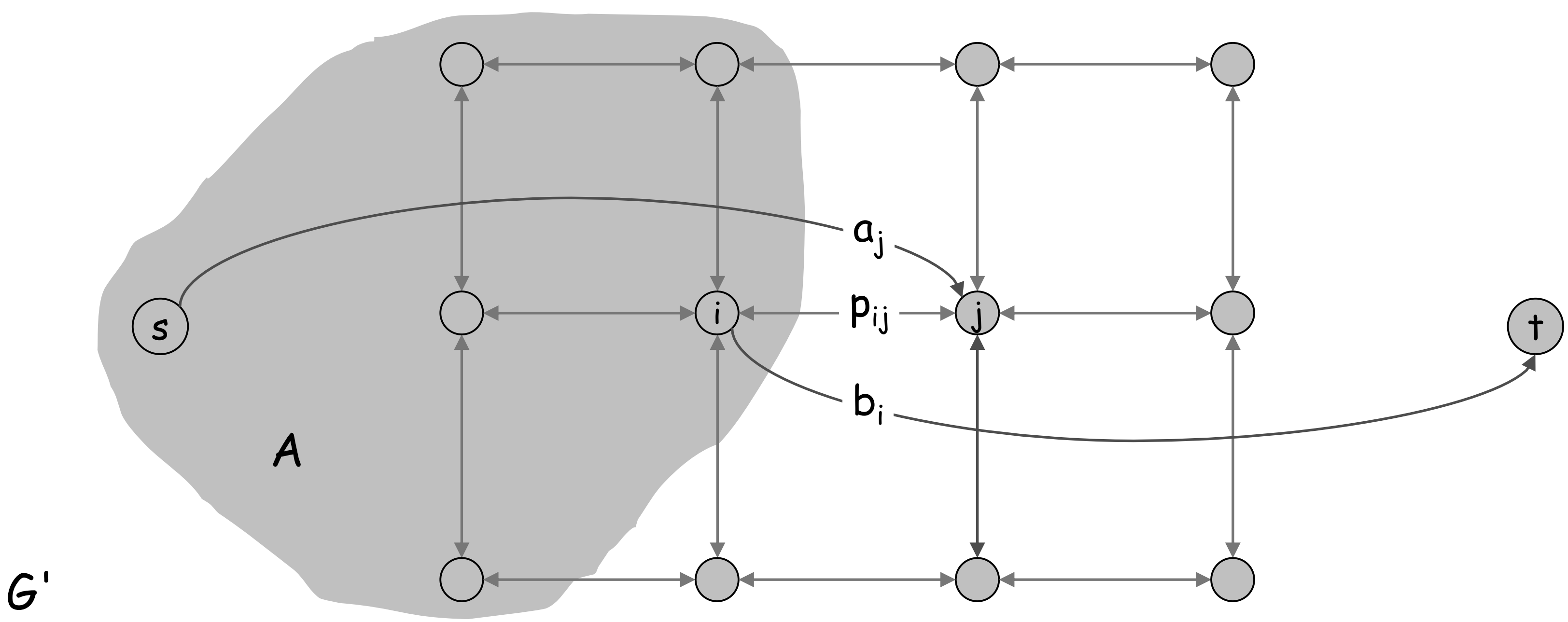
Consider min cut (A, B) in G'.

- A = foreground.

$$cap(A, B) \;=\; \sum_{j \in B} a_j + \sum_{i \in A} b_i \;+ \sum_{\substack{(i,j) \,\in\, E \\ i \in A,\; j \in B}} p_{ij}$$

if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize.



7.11 Project Selection

Project Selection

Projects with prerequisites.

can be positive or negative

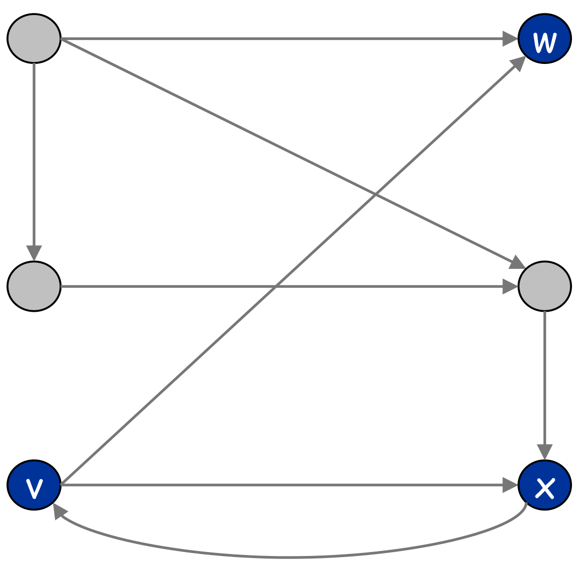
- Set P of possible projects. Project v has associated revenue p_v.
  - some projects generate money: create e-commerce interface, design web page
  - others cost money: upgrade computers, get site license
- Set of prerequisites E. If $(v, w) \in E$, can't do project v unless also do project w.
- A subset of projects $A \subseteq P$ is feasible if the prerequisite of every project in A also belongs to A.

Project selection. Choose a feasible subset of projects to maximize revenue.

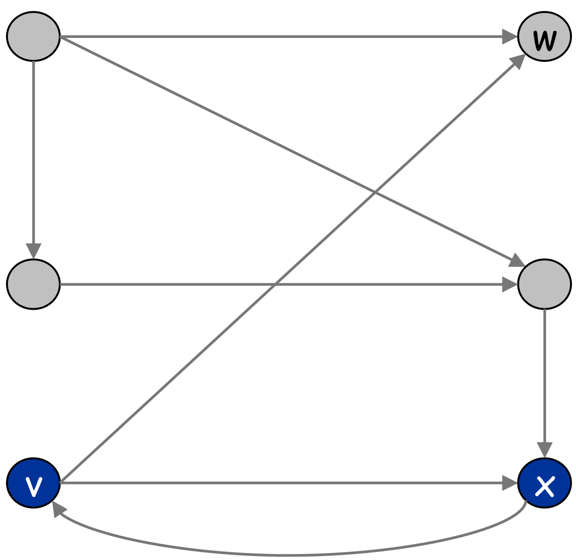
Project Selection: Prerequisite Graph

Prerequisite graph.

- Include an edge from v to w if can't do v without also doing w.
- {v, w, x} is feasible subset of projects.
- {v, x} is infeasible subset of projects.



feasible

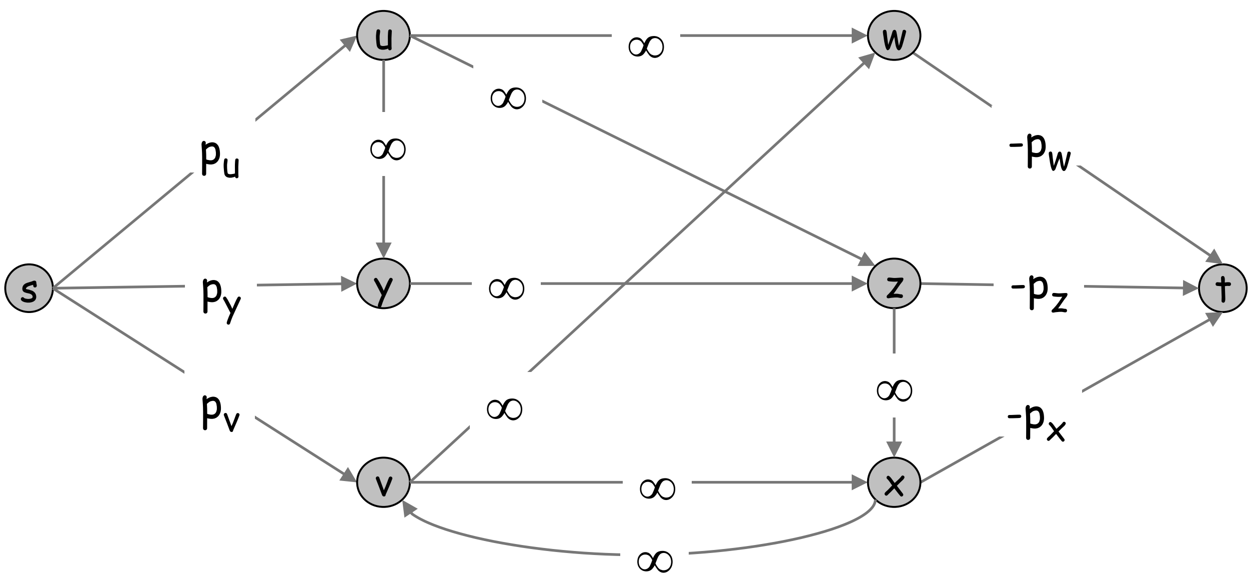


infeasible

Project Selection: Min Cut Formulation

Min cut formulation.

- Assign capacity ∞ to all prerequisite edges.
- Add edge (s, v) with capacity p_v if $p_v > 0$.
- Add edge (v, t) with capacity $-p_v$ if $p_v < 0$.
- For notational convenience, define $p_s = p_t = 0$.

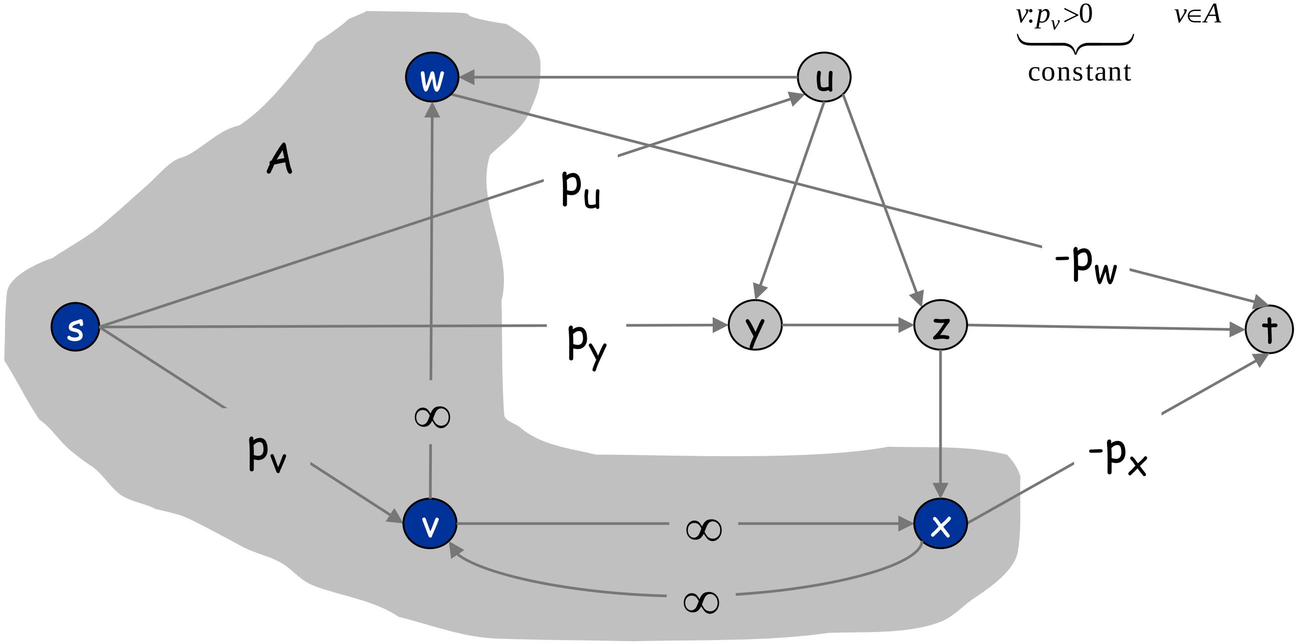


Project Selection: Min Cut Formulation

Claim. (A, B) is min cut iff A – { s } is optimal set of projects.

- Infinite capacity edges ensure A – { s } is feasible.
- Max revenue because:

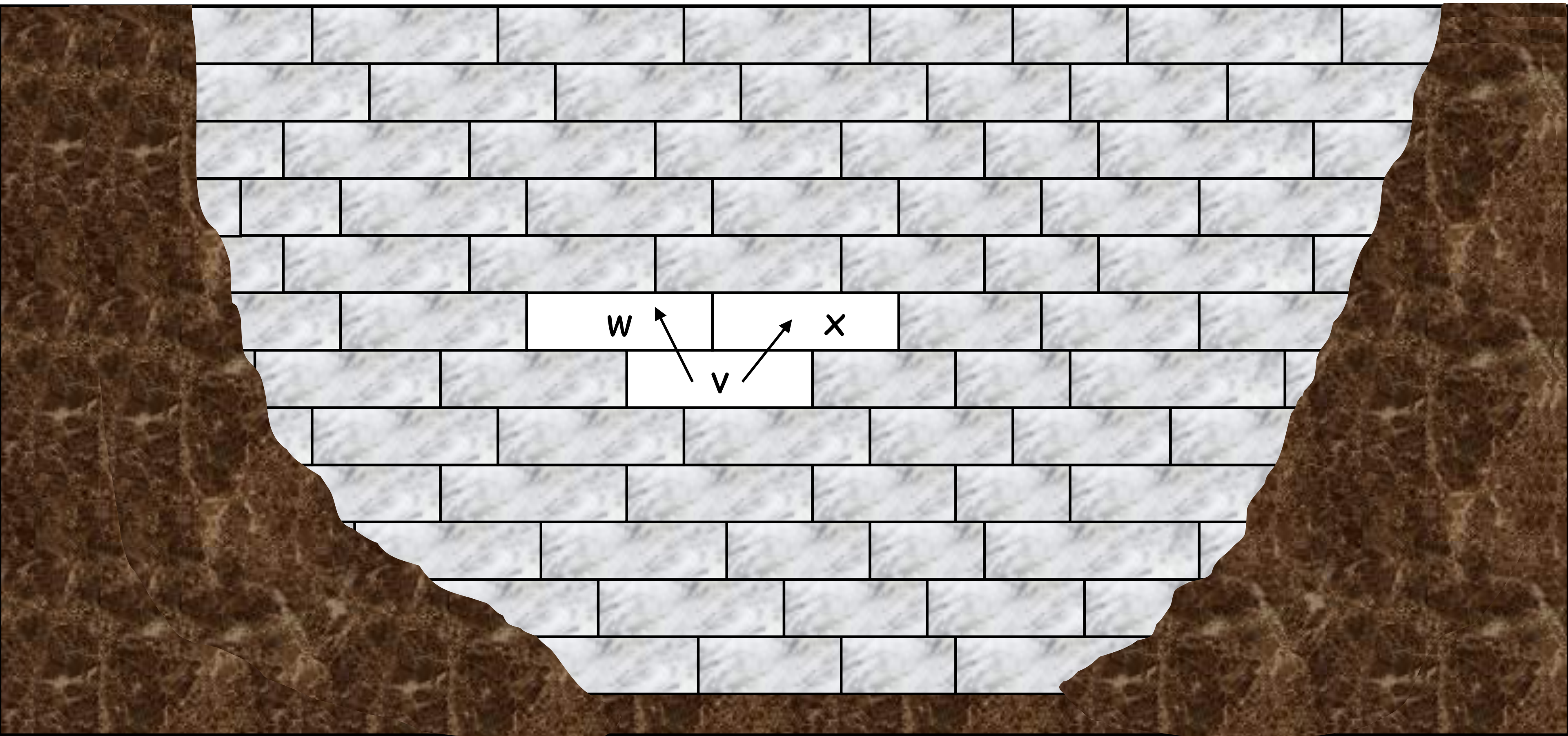
$$
\begin{aligned}
cap(A,B) &= \sum_{v \in B:\, p_v > 0} p_v + \sum_{v \in A:\, p_v < 0} (-p_v) \\
&= \sum_{v \in B:\, p_v > 0} p_v + \sum_{v \in A:\, p_v > 0} p_v - \sum_{v \in A:\, p_v > 0} p_v + \sum_{v \in A:\, p_v < 0} (-p_v) \\
&= \underbrace{\sum_{v:\, p_v > 0} p_v}_{\text{constant}} - \sum_{v \in A} p_v
\end{aligned}
$$



Open Pit Mining

Open-pit mining. (studied since early 1960s)

- Blocks of earth are extracted from surface to retrieve ore.
- Each block v has net value p_v = value of ore - processing cost.
- Can't remove block v before w or x.



7.12 Baseball Elimination

Baseball Elimination

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

Which teams have a chance of finishing the season with most wins?

- Montreal eliminated since it can finish with at most 80 wins, but Atlanta already has 83.
- $w_i + r_i < w_j \Rightarrow$ team i eliminated.
- Sufficient, but not necessary!

Baseball Elimination

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

Which teams have a chance of finishing the season with most wins?

- Philly can win 83, but still eliminated . . .
  - If Atlanta loses all games, then New York wins 84 . . .

Baseball Elimination

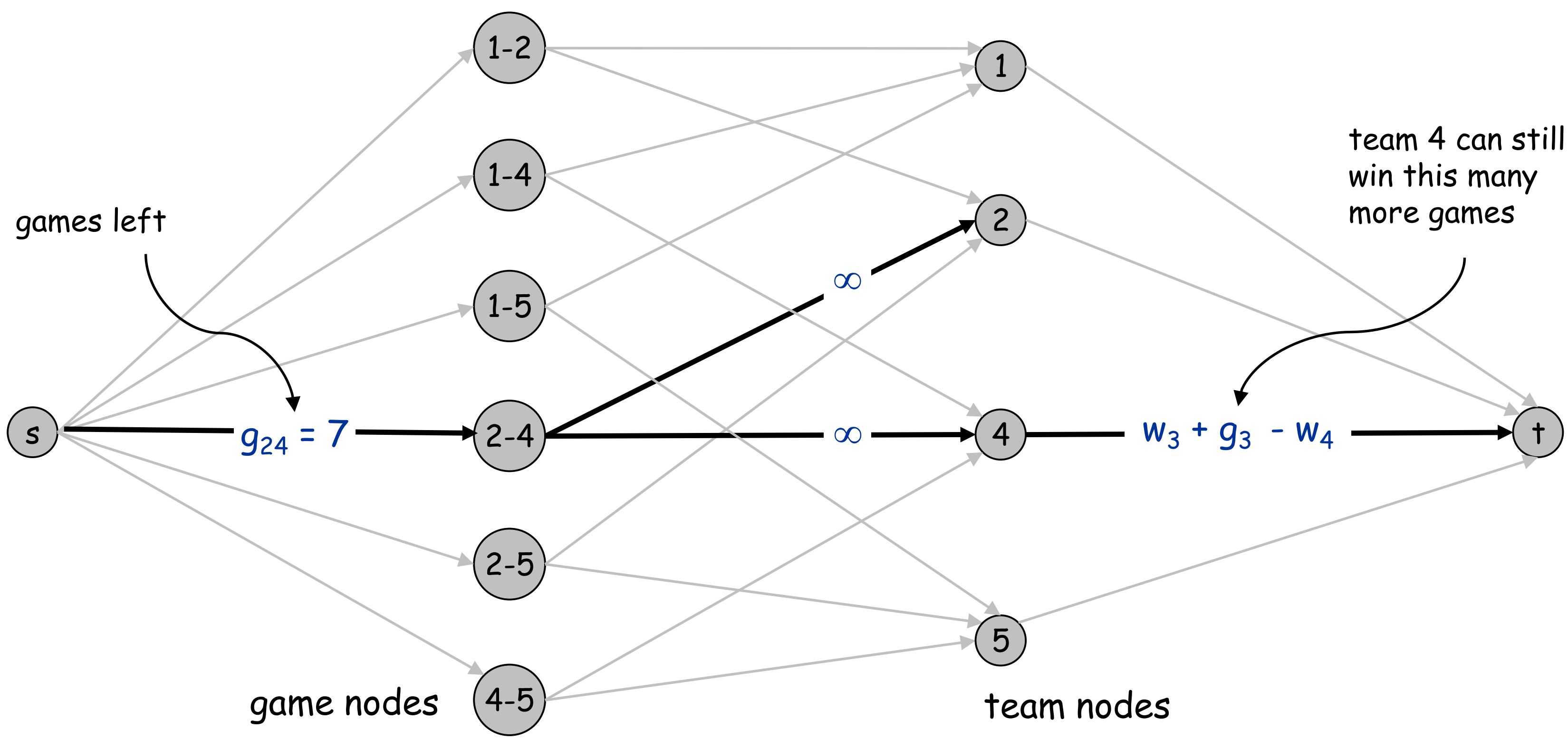
Baseball elimination problem.

- Set of teams S.
- Distinguished team $z \in S$.
- Team x has won w_x games already.
- Teams x and y play each other g_{xy} additional times.
- Is there any outcome of the remaining games in which team z finishes with the most (or tied for the most) wins?

Baseball Elimination: Max Flow Formulation

Can team 3 finish with most wins?

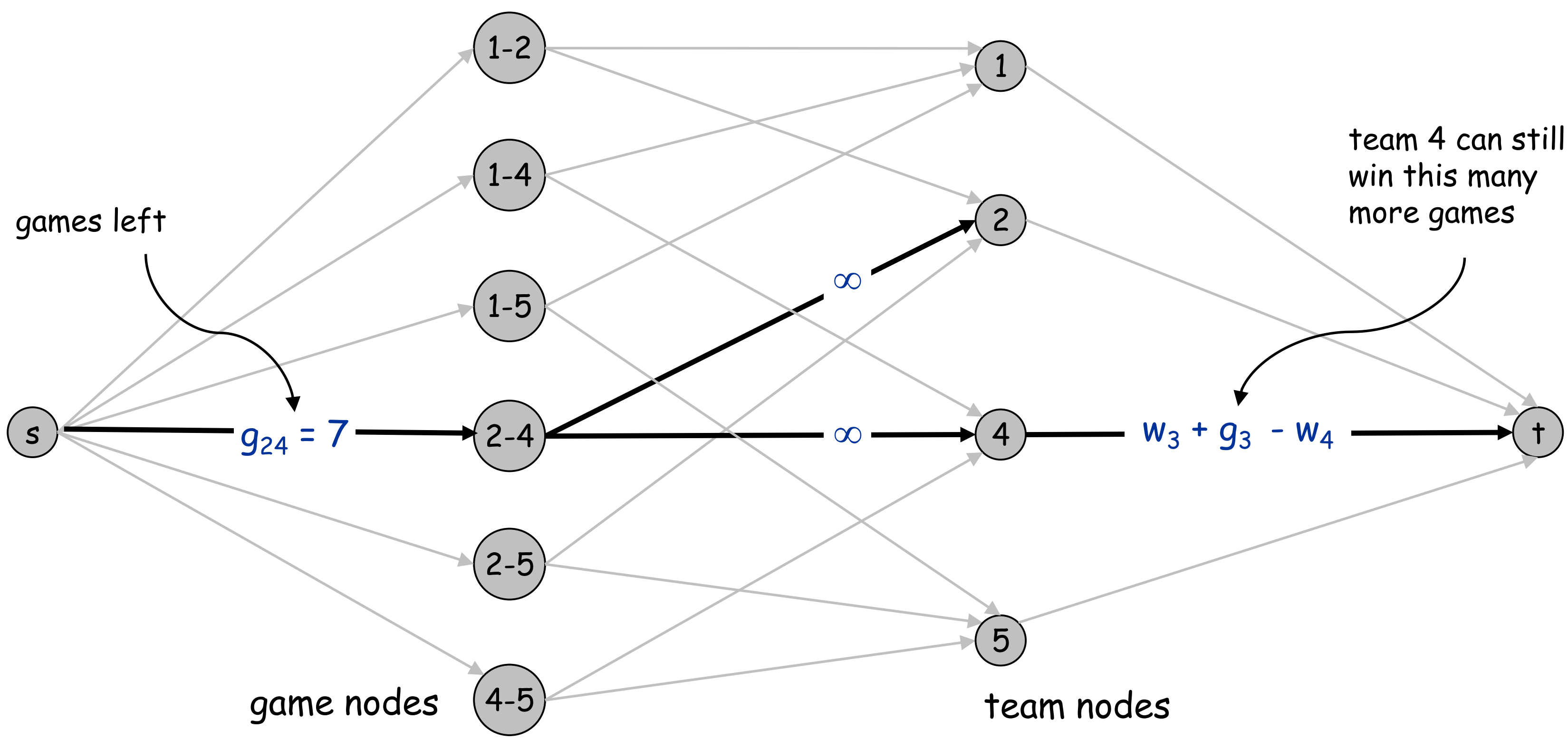
- Assume team 3 wins all remaining games $\Rightarrow$ $w_3 + g_3$ wins.
- Divvy remaining games so that all teams have $\leq$ $w_3 + g_3$ wins.



Baseball Elimination: Max Flow Formulation

Theorem. Team 3 is not eliminated iff max flow saturates all edges leaving source.

- Integrality theorem $\Rightarrow$ each remaining game between x and y added to number of wins for team x or team y.
- Capacity on (x, t) edges ensure no team wins too many games.



Baseball Elimination: Explanation for Sports Writers

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}				
				NY	Bal	Bos	Tor	Det
NY	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3	-	2	7	4
Boston	69	66	27	8	2	-	0	0
Toronto	63	72	27	7	7	0	-	-
Detroit	49	86	27	3	4	0	0	-

AL East: August 30, 1996

Which teams have a chance of finishing the season with most wins?

- Detroit could finish season with 49 + 27 = 76 wins.

Baseball Elimination: Explanation for Sports Writers

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}				
				NY	Bal	Bos	Tor	Det
NY	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3	-	2	7	4
Boston	69	66	27	8	2	-	0	0
Toronto	63	72	27	7	7	0	-	-
Detroit	49	86	27	3	4	0	0	-

AL East: August 30, 1996

Which teams have a chance of finishing the season with most wins?

- Detroit could finish season with 49 + 27 = 76 wins.

Certificate of elimination. R = {NY, Bal, Bos, Tor}

- Have already won w(R) = 278 games.
- Remaining games among R is r(R) = 3+8+7+2+7 = 27
- Average team in R wins at least (278+27)/4 > 76 games.

Baseball Elimination: Explanation for Sports Writers

Certificate of elimination.

$$T \subseteq S, \quad w(T) \coloneqq \overbrace{\sum_{i \in T} w_i}^{\#\text{ wins}}, \quad g(T) \coloneqq \overbrace{\sum_{\{x,y\} \subseteq T} g_{xy}}^{\#\text{ remaining games}},$$

Theorem. [Hoffman-Rivlin 1967] Team z is eliminated iff there exists a subset T* such that

$$\frac{\overbrace{w(T^*) + g(T^*)}^{\text{LB on avg \# games won}}}{|T^*|} > w_z + g_z$$

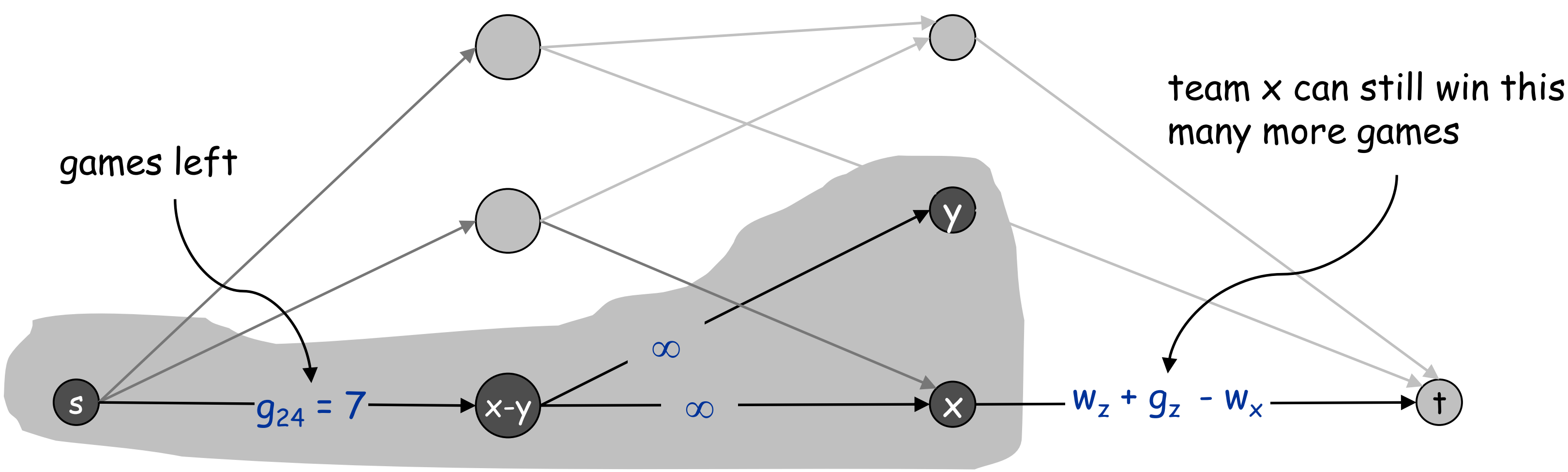
Proof. $\Leftarrow$

- The average number of wins of teams in T* is larger than the maximum number of wins of z

Baseball Elimination: Explanation for Sports Writers

Proof. $\Rightarrow$

- Use max flow formulation, and consider min cut (A, B).
- Define T^* = team nodes on source side of min cut.
- Observe $x\text{-}y \in A$ iff both $x \in T^*$ and $y \in T^*$.
  - infinite capacity edges ensure if $x\text{-}y \in A$ then $x \in A$ and $y \in A$
  - if $x \in A$ and $y \in A$ but $x\text{-}y \in B$, then adding x-y to A decreases capacity of cut



Baseball Elimination: Explanation for Sports Writers

Proof. $\Rightarrow$

- Use max flow formulation, and consider min cut (A, B).
- Define T* = team nodes on source side of min cut.
- Observe x-y $\in$ A iff both x $\in$ T* and y $\in$ T*.
- Since z is eliminated, by max-flow min-cut theorem:

$$
\begin{aligned}
g(S-\{z\}) &> cap(A,B) \\
&= \overbrace{g(S-\{z\}) - g(T^*)}^{\text{capacity of game edges leaving s}} + \overbrace{\sum_{x \in T^*} (w_z + g_z - w_x)}^{\text{capacity of team edges entering t}} \\
&= g(S-\{z\}) - g(T^*) - w(T^*) + |T^*| \, (w_z + g_z)
\end{aligned}
$$

- Rearranging terms: $w_z + g_z < \dfrac{w(T^*) + g(T^*)}{|T^*|}$ ▪

Chapter Summary

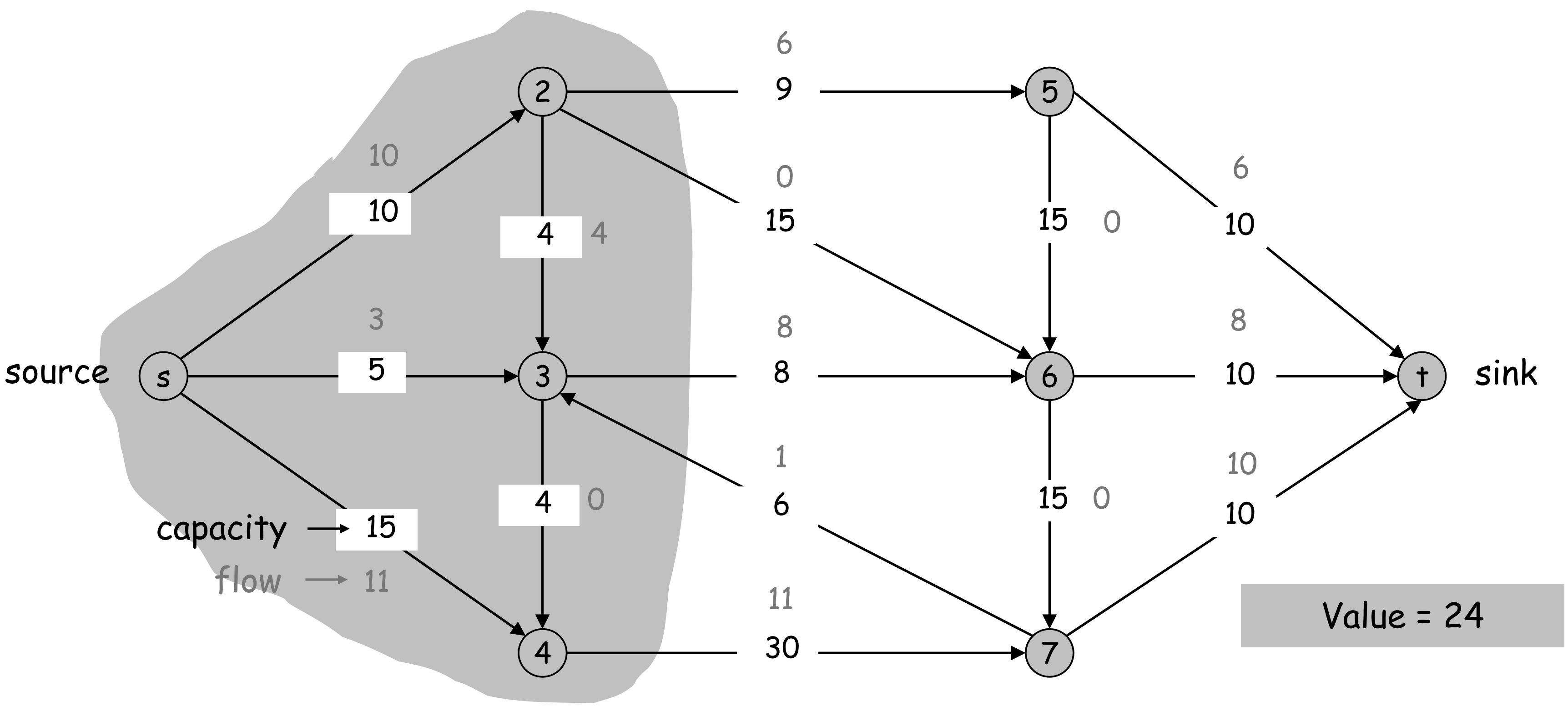
Flows

Concepts

- s-t flow
- Max-flow
- s-t cut
- Min-cut

Max-flow min-cut theorem.

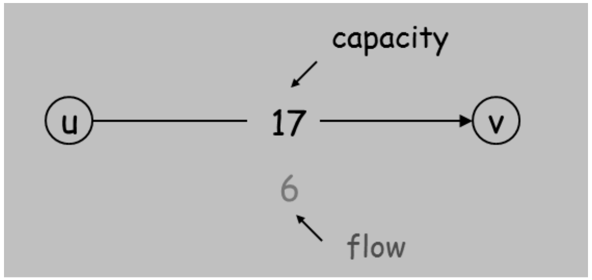
The value of the max flow is equal to the value of the min cut.

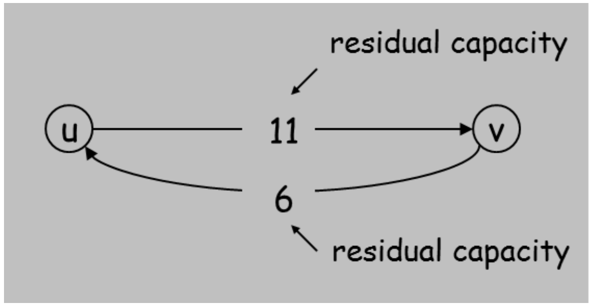


Ford-Fulkerson Algorithm

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.
  - Can be chosen using capacity scaling
- Augment flow along path P.
- Repeat until you get stuck.






```
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
}
```

Applications

Problems covered in class

- Bipartite Matching
- Disjoint Paths
- Circulation with Demands (+ edge lower bounds)
- Survey Design
- Image Segmentation
- Project Selection
- Baseball Elimination