CSE 421 Winter 2025
Lecture 18:
Max Flow Applications

Nathan Brunelle

http://www.cs.uw.edu/421

Max Flow Algorithm Summary

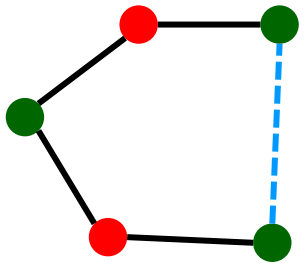
- Ford-Fulkerson:
  - "Find an augmenting path, identify a bottleneck edge, add flow to fill the bottleneck"
  - Worst case running time: $\Theta(|V| \cdot |E| \cdot C)$ where C is the maximum capacity of any edge
- Edmonds-Karp:
  - "Do Ford Fulkerson, but use the *shortest* augmenting path (with BFS)"
  - Worst case running time: $\Theta(|E|^2|V|)$
- Observation:
  - Since Edmonds-Karp is just a special case of Ford-Fulkerson the running time of Edmonds-Karp should be:
  - $\Theta(\min(|V| \cdot |E| \cdot C, |E|^2|V|))$
    - Use the FF bound when $C > |E|$, otherwise use EK.

Today – Reductions and Max Flow

- Max flow is primarily useful as the destination of a reduction
  - Given some problem that is not already a max flow problem
  - Use that to create a flow network
  - Run Edmonds Karp on that flow network
  - Use the flow assignment to solve your original problem
- Proving correctness:
  - Argue that the flow through your constructed network is maximal if and only if your final answer is correct
    - Valid flow assignment in the network ⇒ Valid answer to original problem
      - The flow we found is guaranteed to give us a feasible solution
    - Valid answer to original problem ⇒ Valid flow assignment in the network
      - We must have had the best feasible solution as a better one would have allowed more flow

Recall: Bipartite Graph

Definition: An undirected graph G is bipartite iff we can color its vertices **red** and **green** so each edge has different color endpoints



On a cycle the two colors must alternate, so

- **green** every 2nd vertex
- **red** every 2nd vertex

Can't have either if length is not divisible by 2.

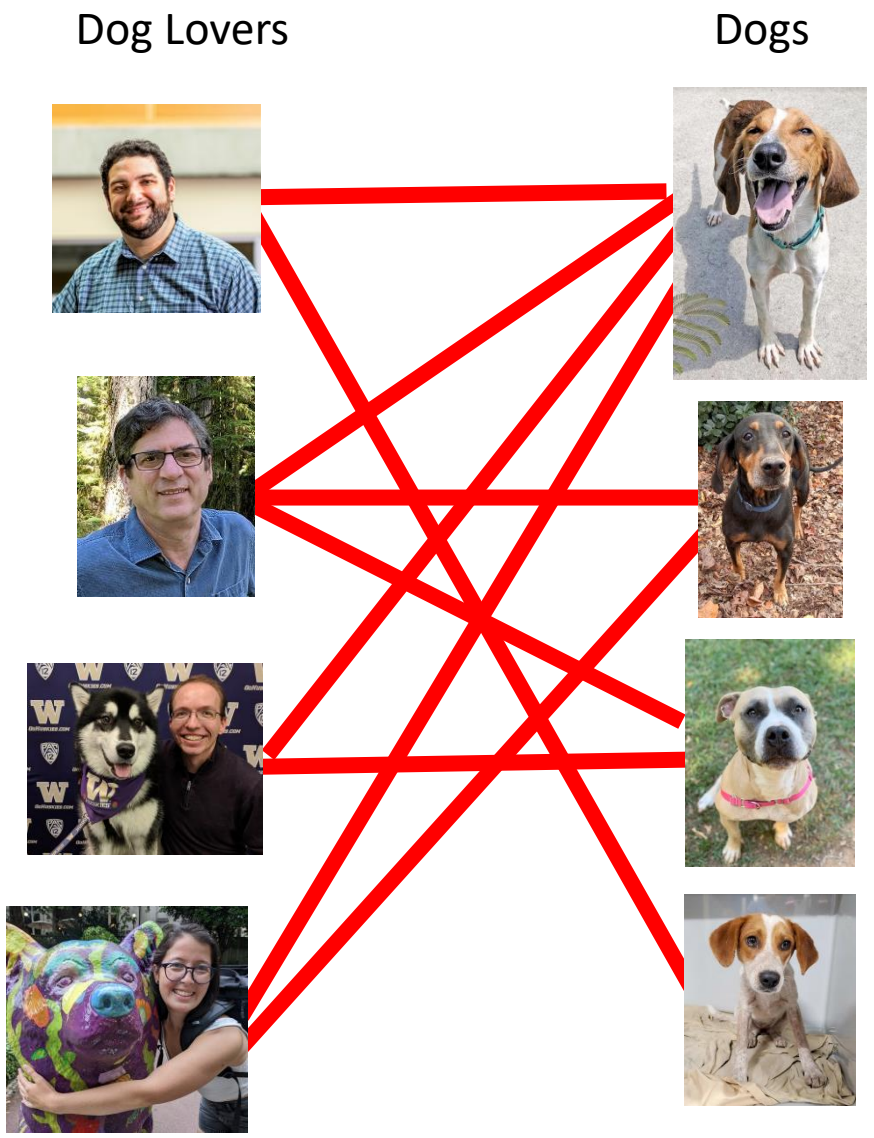
Alternative: A graph G is bipartite iff we can find a cut (L, R) such that every edge in the graph crosses the cut

Bipartite Matching

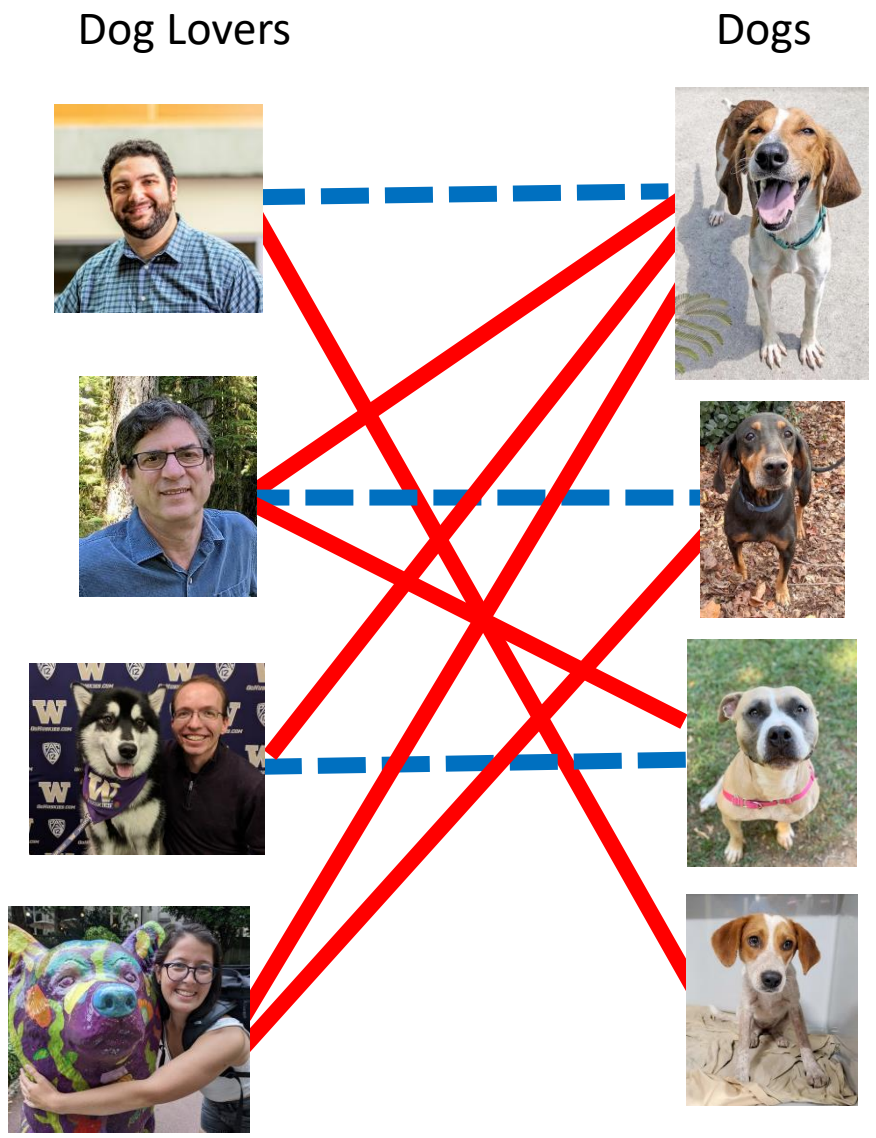
Input: Undirected Bipartite graph $G = (L \cup R, E)$

Goal: Find the largest subset of edges $M \subseteq E$ such that ever vertex is adjacent to at most one edge

Maximum Bipartite Matching

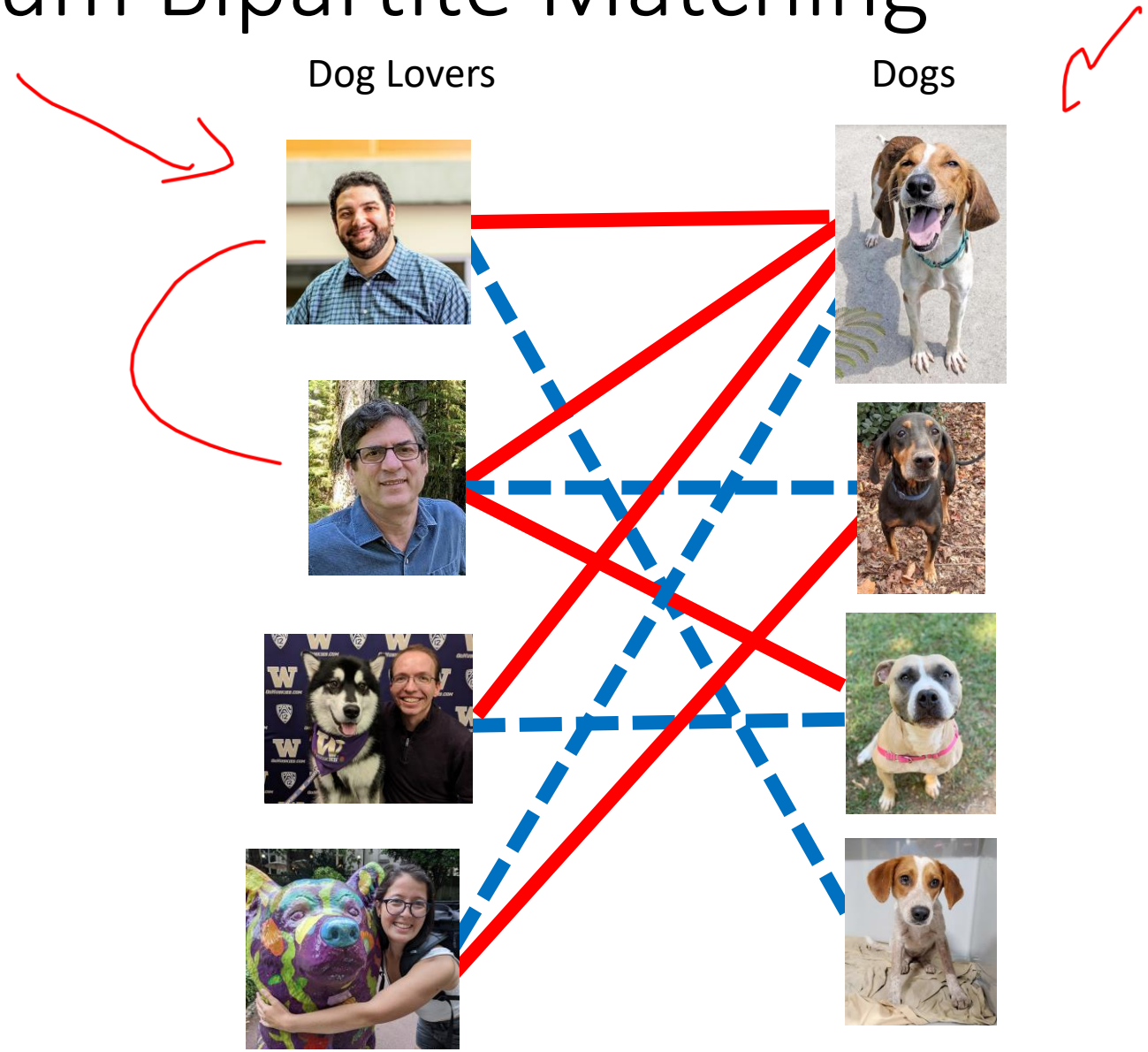


Maximum Bipartite Matching



A (non-maximum) bipartite matching

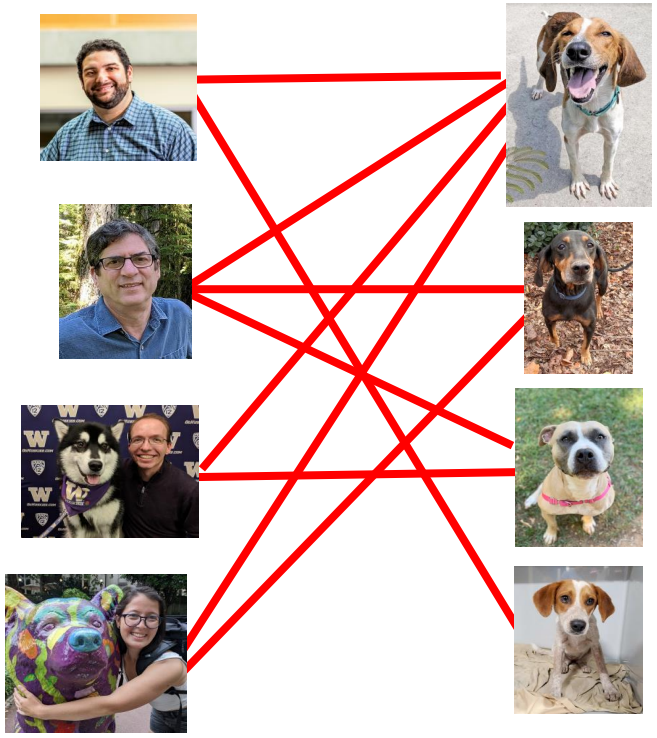
Maximum Bipartite Matching



A maximum bipartite matching!

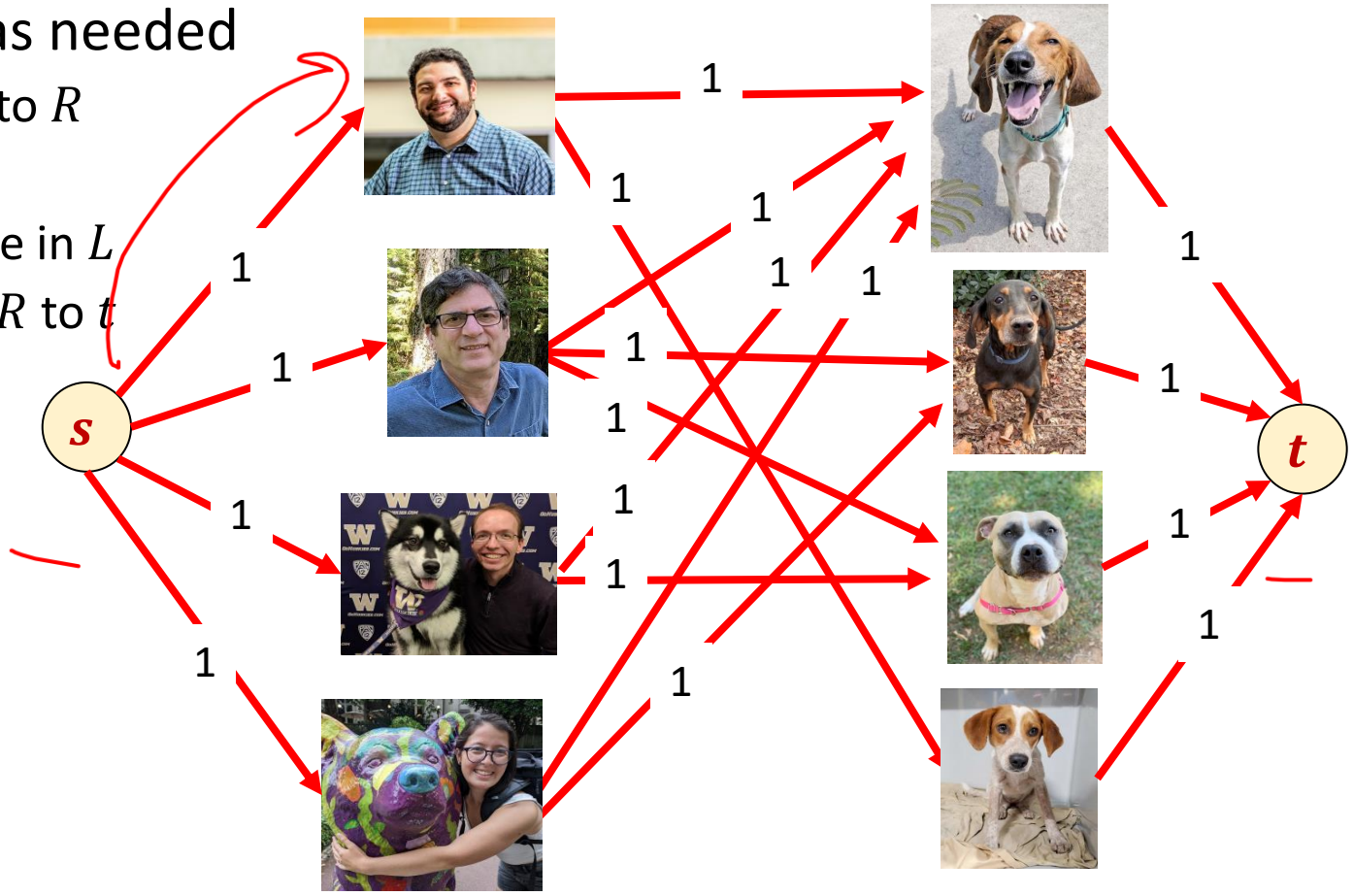
Reducing to Max Flow

- We need to create a flow network
  - Add/remove any edges/nodes as needed (must be directed)
  - Identify a source/sink
  - Give capacities to each edge



Reducing to Max Flow

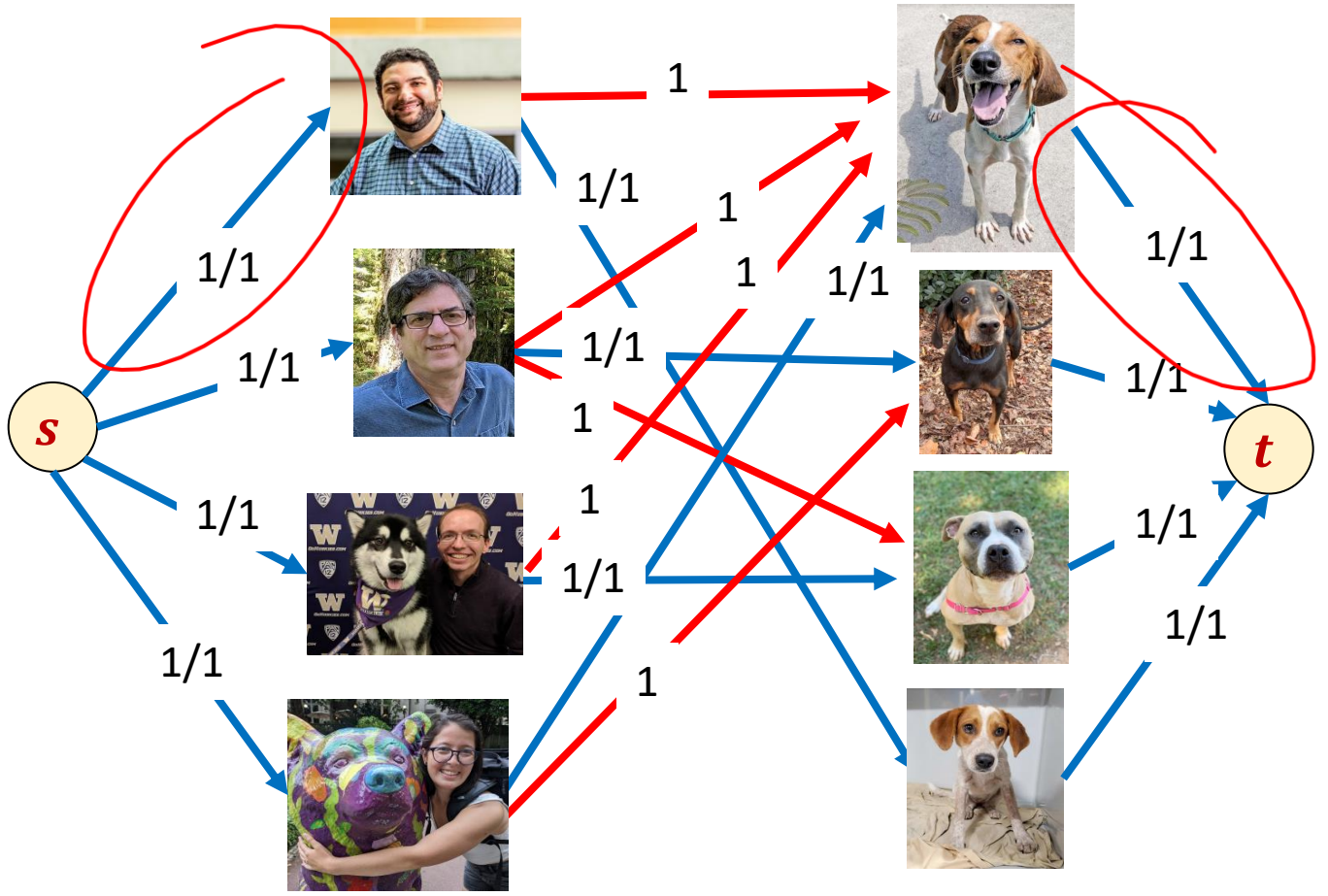
- We need to create a flow network
  - Add/remove any edges/nodes as needed
    - Make each edge directed from L to R
    - Add nodes s and t
    - Draw an edge from s to each node in L
    - Draw an edge from each node in R to t
  - Identify a source/sink
    - Make s the source and t the sink
  - Give capacities to each edge
    - Give each edge capacity 1



Reducing to Max Flow

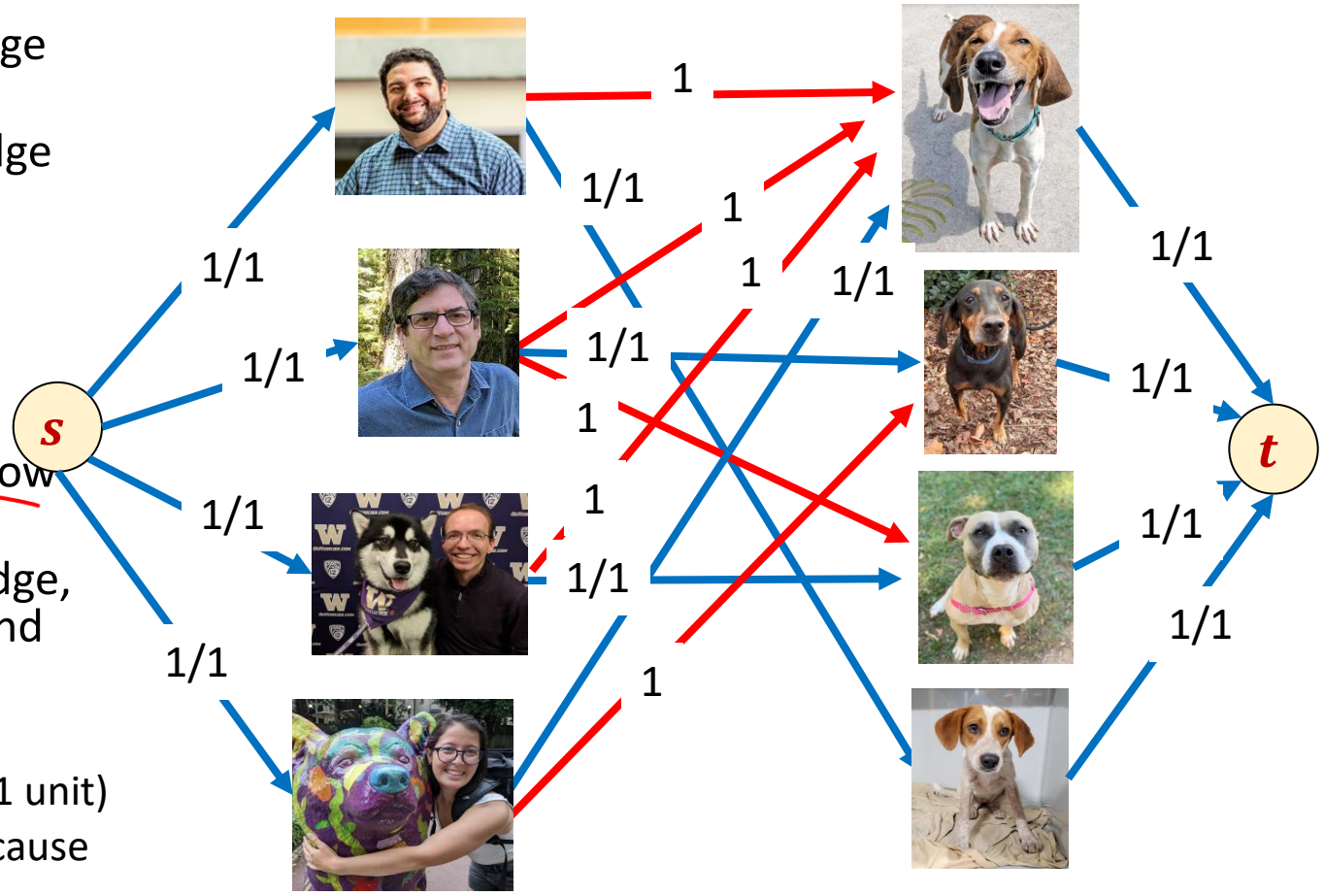
- Next, run Edmonds-Karp
- Use the answer to select a Matching
  - M = the set of L-to-R edges with flow
  - $M = \{e : e \in L \times R, f(e) = 1\}$

- Running Time:
  - Constructing the graph
    - $|V| + |E|$
  - Running Edmonds-Karp
    - $|V| \cdot |E|$
  - Finding M
    - $|V|$
  - Overall
    - $\Theta(|V| \cdot |E|)$



Correctness

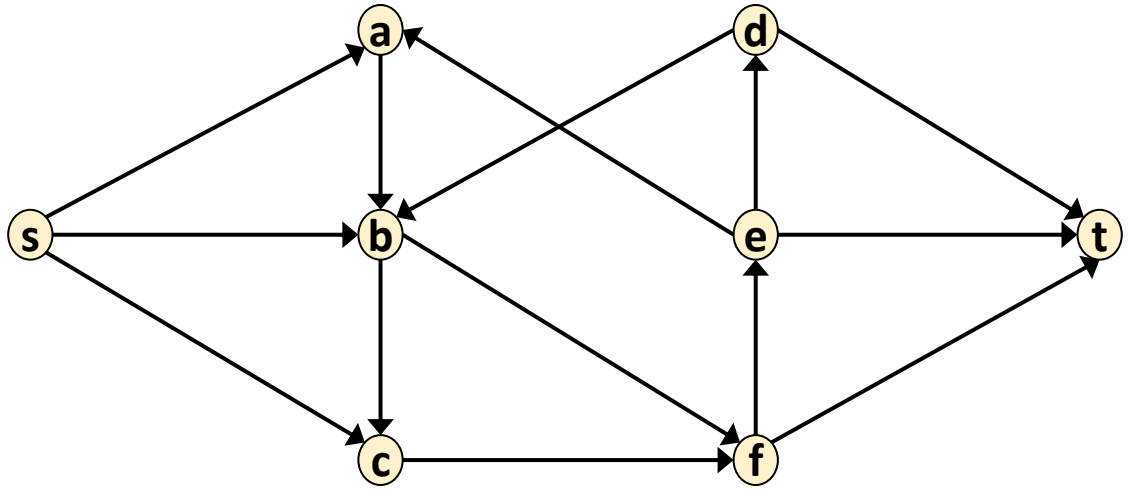
- Valid flow ⇒ Valid answer
  - Need to show: M is a valid match
  - Requirement 1: $M \subseteq E$
  - Requirement 2: each node is part of at most one match
    - No node in L is adjacent to more than one edge because the capacity of the edge from s is 1
    - No node in R is adjacent to more than one edge because the capacity of the edge to t is 1
- Valid answer ⇒ Valid flow
  - Any matching M enables a flow with value equal to $|M|$
    - Idea: "reverse" the construction to derive a flow from the match
    - For each edge in M, assign flow across that edge, then add flow incoming to each L endpoint and outgoing form each t endpoint
    - This must be a valid flow because
      - No edge exceeds its capacity (we only assign 1 unit)
      - All nodes (except s and t) have 0 net flow because each node is incident at most one edge in M



Edge-Disjoint Paths

Defn: Two paths in a graph are edge-disjoint iff they have no edge in common.

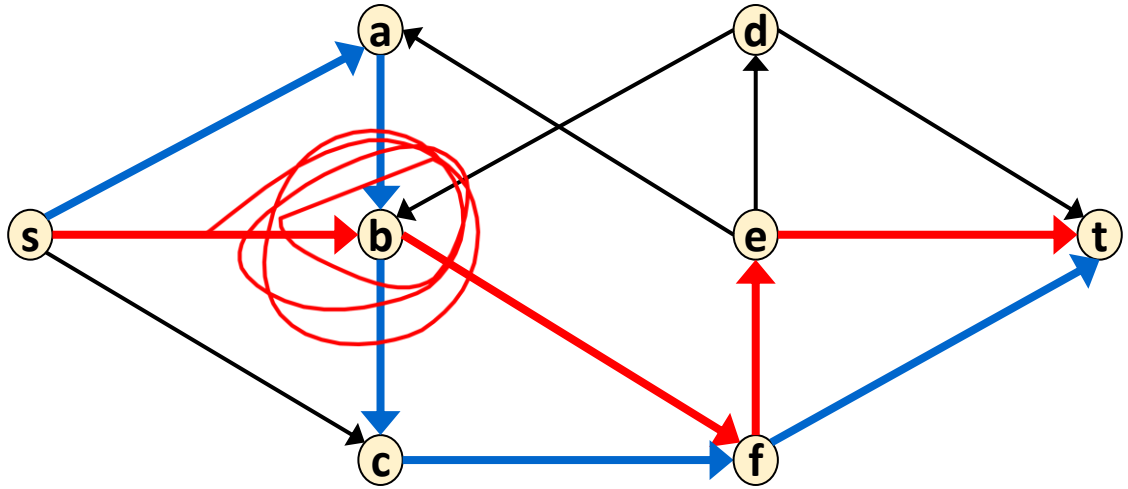
Edge disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.
Find: the maximum # of edge-disjoint simple s-t paths in G.



Edge-Disjoint Paths – Example of size 2

Defn: Two paths in a graph are edge-disjoint iff they have no edge in common.

Edge disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.
Find: the maximum # of edge-disjoint simple s-t paths in G.



Edge-Disjoint Paths

MaxFlow for edge-disjoint paths

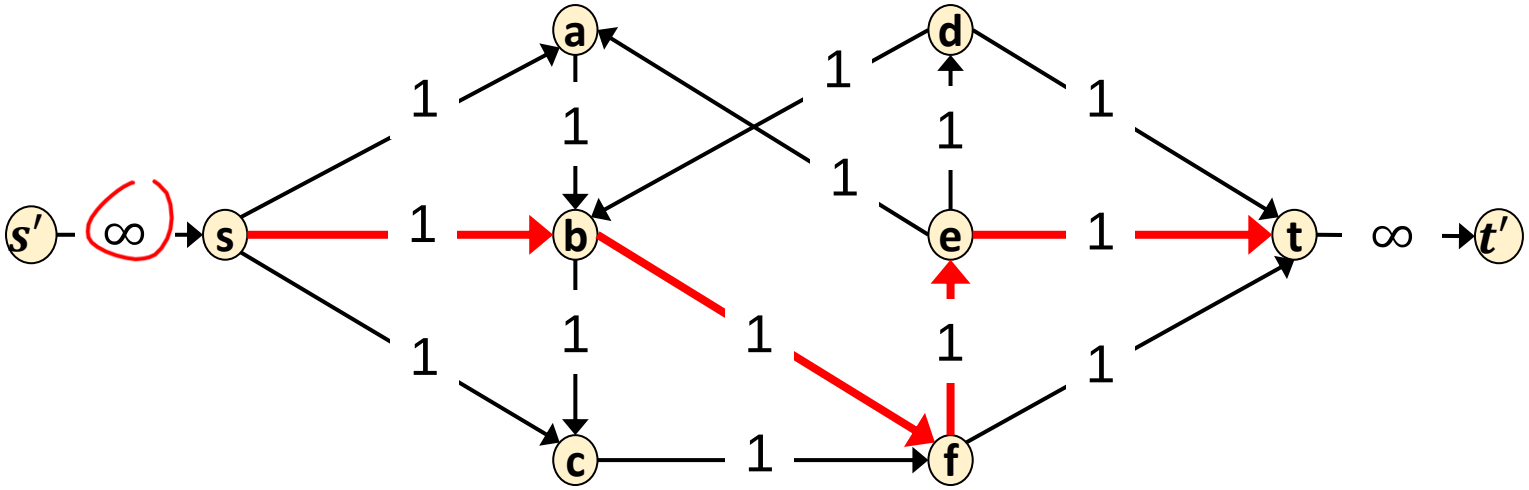
- Assign capacity $\mathbf{1}$ to every edge
- Add a source $\boldsymbol{s'}$ and a sink $\boldsymbol{t'}$
- Connect $\boldsymbol{s'}$ to $\boldsymbol{s}$ and $\boldsymbol{t'}$ to $\boldsymbol{t}$ with capacity ∞
- Compute max flow

Running Time:

Constructing the flow network: $|V| + |E|$

Computing Max Flow: $|V| \cdot |E|^2$

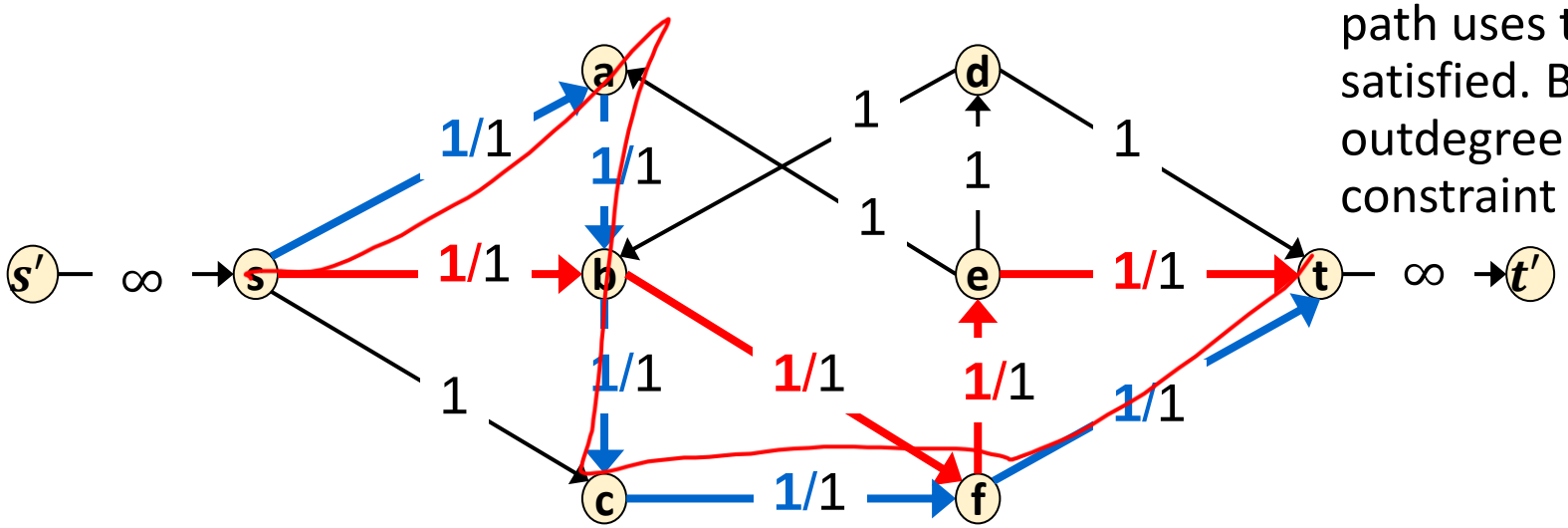
Overall: $\Theta(|V| \cdot |E|^2)$



Edge-Disjoint Paths

MaxFlow for edge-disjoint paths

- Assign capacity $\mathbf{1}$ to every edge
- Add a source s' and a sink t'
- Connect s' to s and t' to t with capacity ∞
- Compute max flow



Theorem: MaxFlow = # edge-disjoint paths

Valid flow ⇒ Valid answer:

Need to show: no edge is used more than once all paths go from s to t

Each edge has capacity 1, so its used once. To get from s' to t' we must go from s to t along the way

Valid answer ⇒ Valid flow:

Need to show: Any set of edge-disjoint paths could be used to produce flow of the same amount.

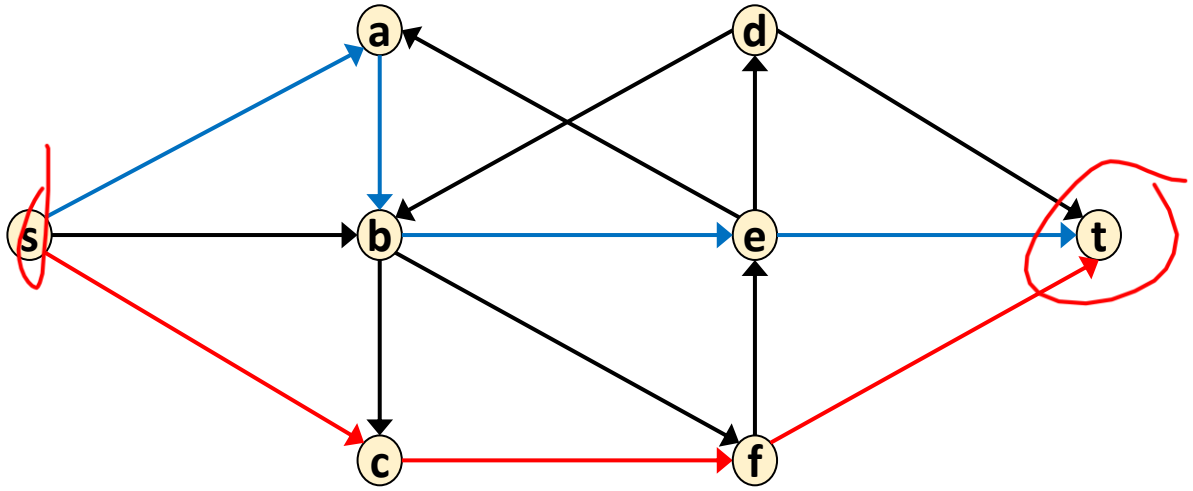
Add 1 unit of flow along each path. Since no path uses the same edge twice, capacity constraint is satisfied. Because the indegree matches the outdegree for each node (except s' and t'), the flow constraint is satisfied.

Vertex-Disjoint Paths

Defn: Two paths in a graph are vertex-disjoint iff they have no vertices in common, except their end points.

Vertex disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.

Find: the maximum # of vertex-disjoint simple s-t paths in G.

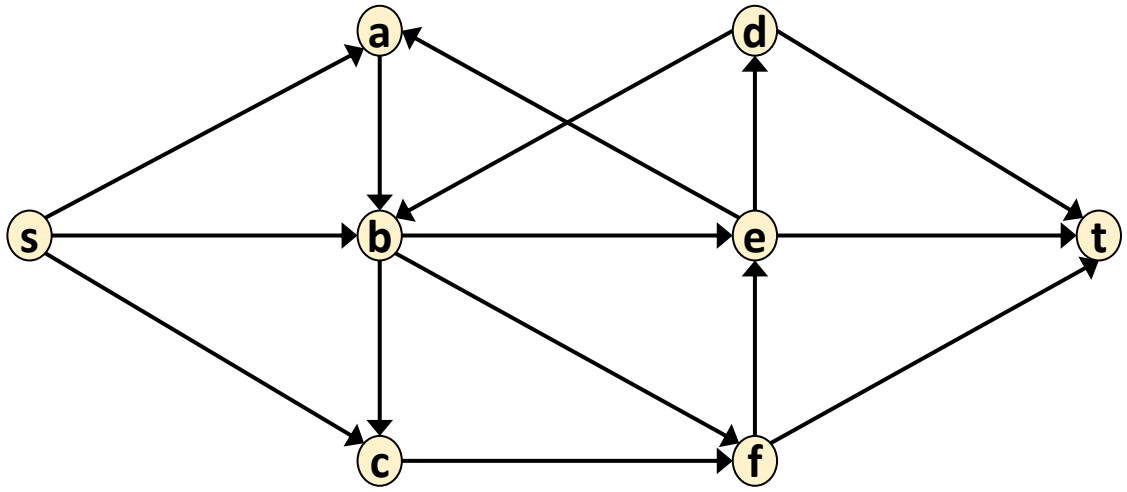


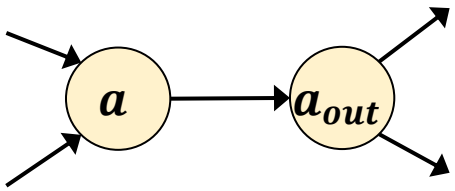
Vertex-Disjoint Paths

Defn: Two paths in a graph are vertex-disjoint iff they have no vertices in common.

Vertex disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.

Find: the maximum # of vertex-disjoint simple s-t paths in G.





Observation: Every vertex-disjoint path is also edge-disjoint.
(Two paths which share an edge also share that edge's endpoints)

Idea: Modify the graph so that all edge-disjoint paths are also vertex disjoint

Vertex-Disjoint Paths

MaxFlow for vertex-disjoint paths

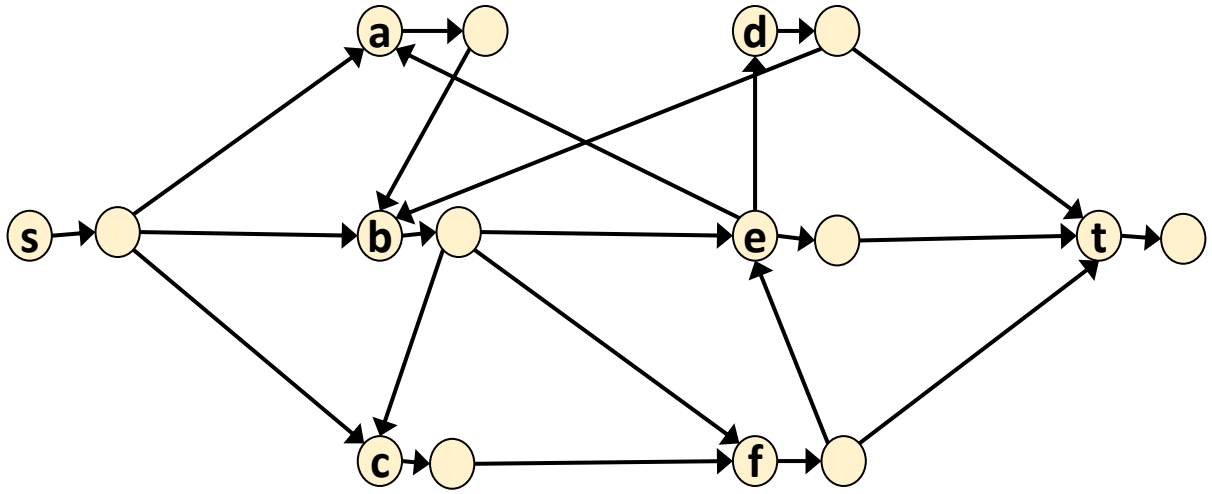
- For each node v, add in v_{out}
- For every outgoing edge from v, instead make it an outgoing edge from v_{out}
- Add edge (v, v_{out})
- Compute edge-disjoint paths

Running Time:

Constructing the new graph: $|V| + |E|$

Computing edge disjoint paths: $|V| \cdot |E|^2$

Overall: $\Theta(|V| \cdot |E|^2)$



Shift Scheduling

- The manager at a bagel shop needs to staff all shifts during the day.
- We have the following constraints:
  - Shift s_i must have at least p_i people assigned to it
  - Each employee e_i has a list of shifts that they are able to work
  - No employee is able to work more than x shifts

Shifts:

1. 6am, 2
2. 9am, 2
3. 12pm, 1
4. 3pm, 1

Employees:

1. 6am, 9am, 3pm
2. 6am, 9am, 12pm
3. 6am, 3pm

$x = 2$

Solution:

- Employee 1 assigned to 6am, 9am
- Employee 2 assigned to 9am, 12pm
- Employee 3 assigned to 6am, 3pm

Shift Scheduling problem

Given: A list of n shifts $s_1, \ldots s_n$, the number of employees needed for each shift $p_1, \ldots p_n$, the availability of m employees $e_1, \ldots, e_m$, and a number x

Find: whether it is possible to assign employees to their available shifts such that all shifts are full-staffed and no employee is assigned to more than x shifts

Shifts:
1. 6am, 2
2. 9am, 2
3. 12pm, 1
4. 3pm, 1

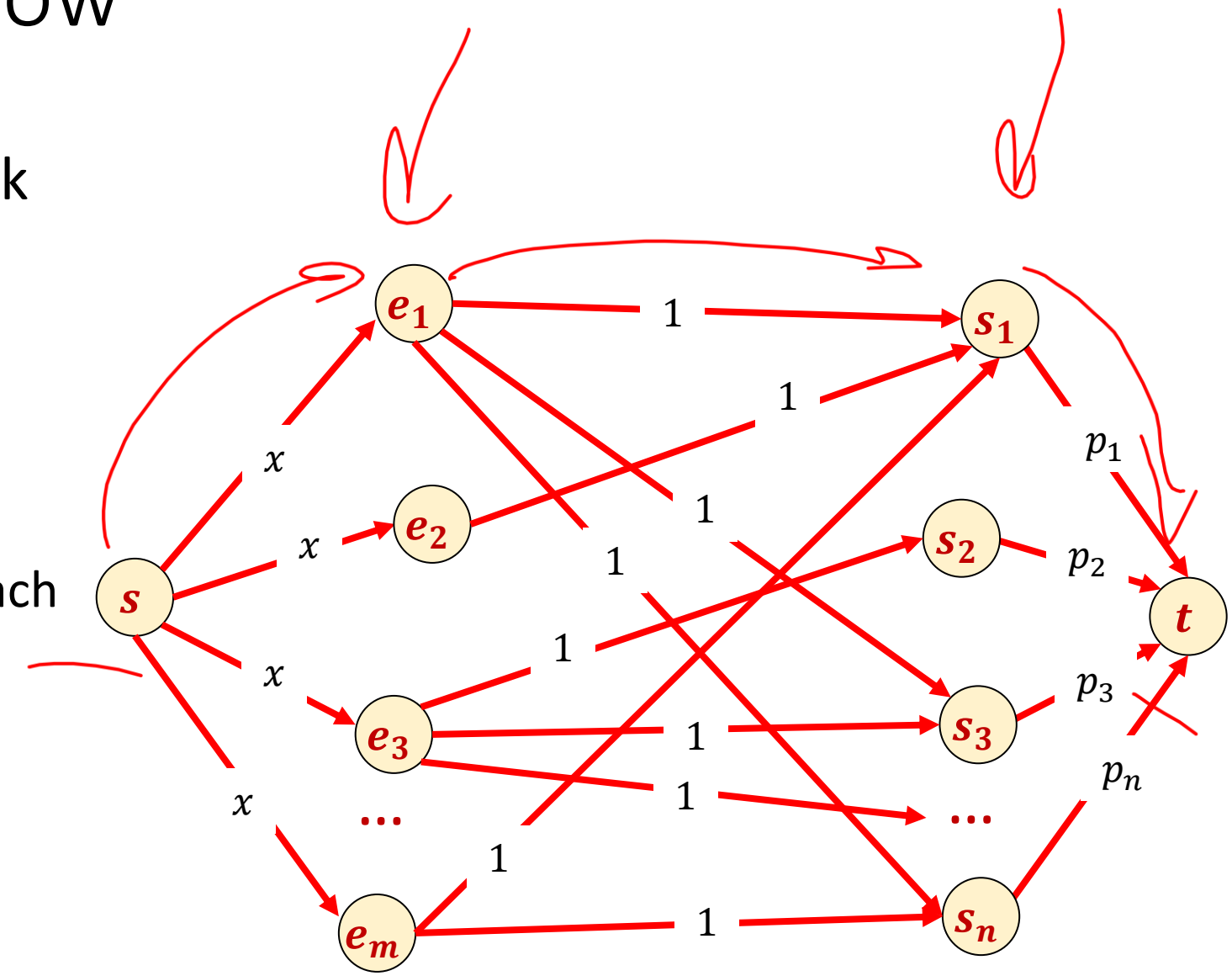
Employees:
1. 6am, 9am, 3pm
2. 6am, 9am, 12pm
3. 6am, 3pm

$x = 2$

Solution:
- Employee 1 assigned to 6am, 9am
- Employee 2 assigned to 9am, 12pm
- Employee 3 assigned to 6am, 3pm

Reducing to Max Flow

- We need to create a flow network
  - One node per shift
  - One node per employee
  - A source node and a sink node
  - An edge from the source to each employee node with capacity x
  - An edge from each employee to each available shift with capacity 1
  - An edge from each shift node s_i to the sink with capacity p_i



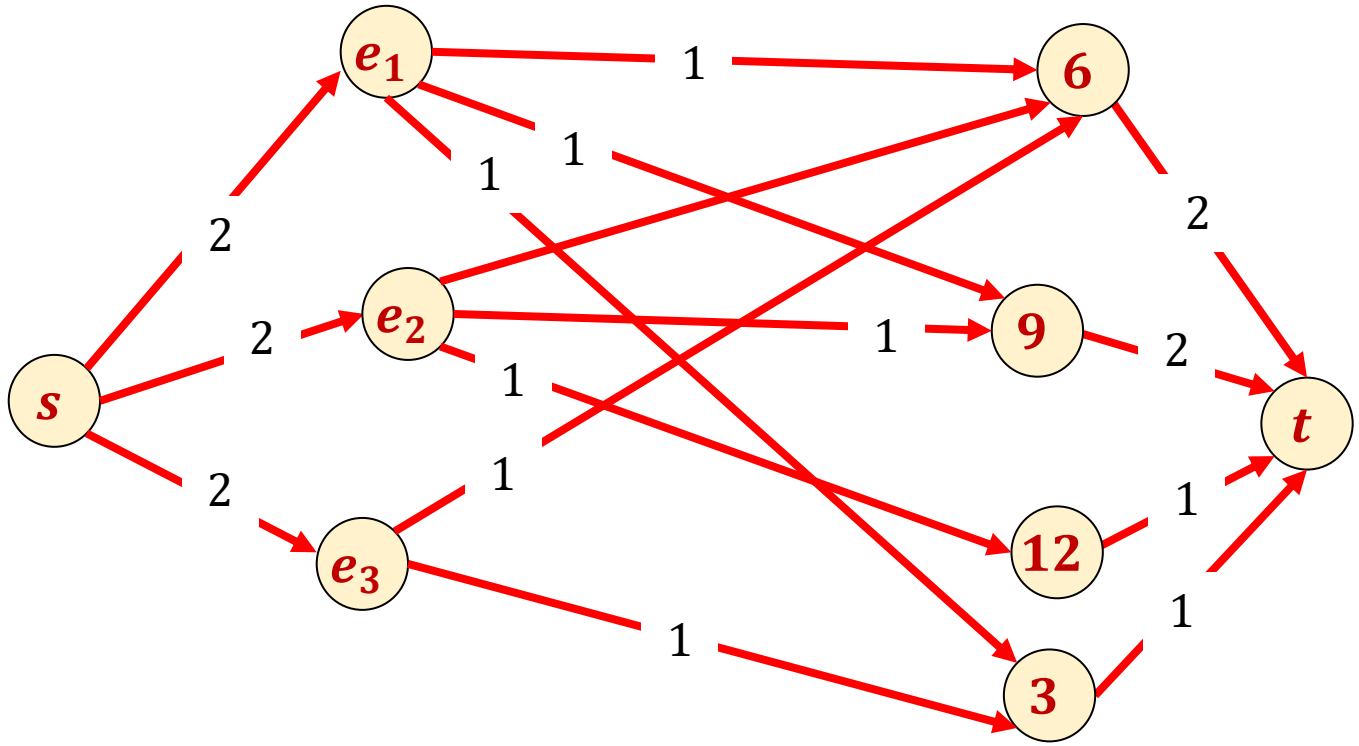
Reducing to Max Flow

Shifts:
1. 6am, 2
2. 9am, 2
3. 12pm, 1
4. 3pm, 1

Employees:
1. 6am, 9am, 3pm
2. 6am, 9am, 12pm
3. 6am, 3pm

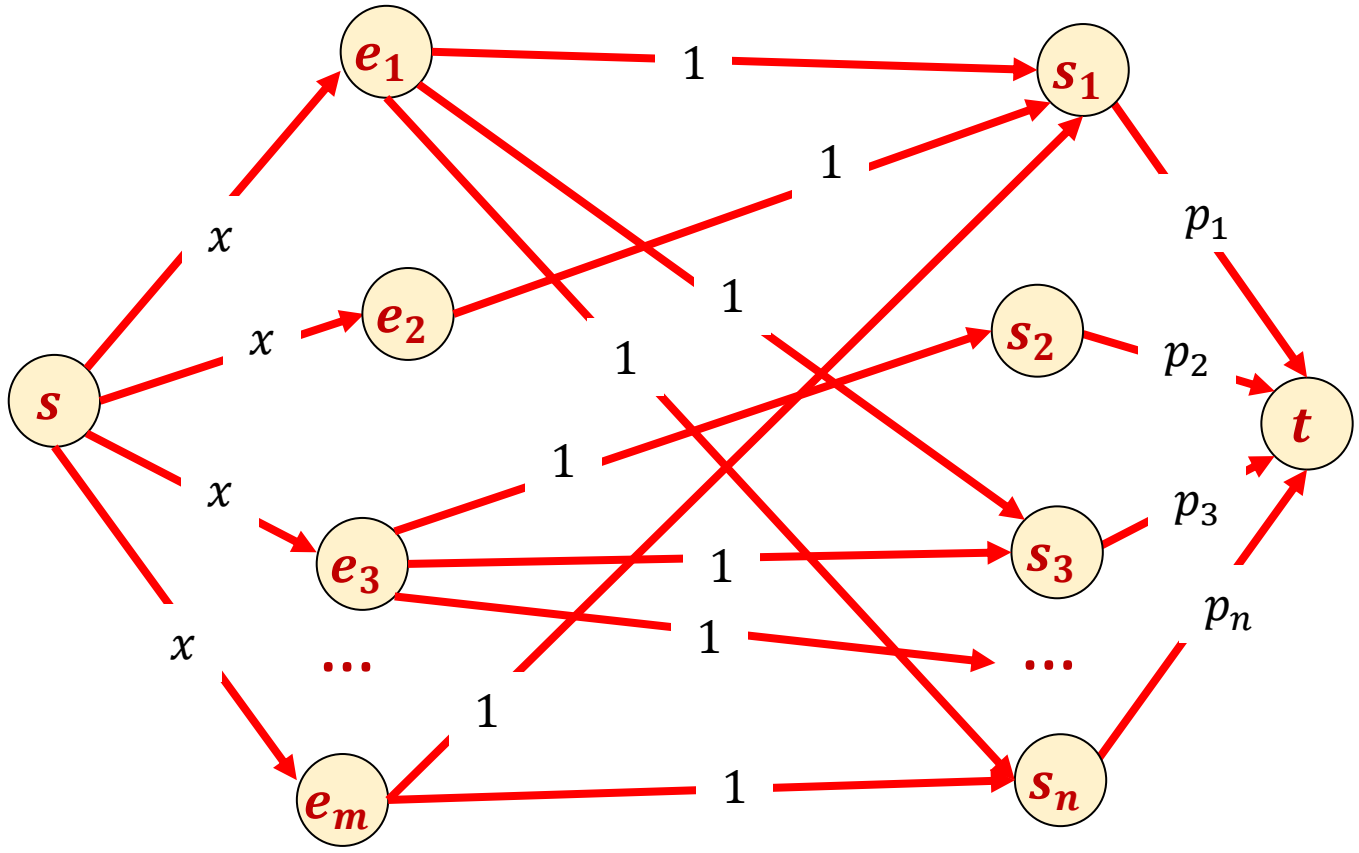
$x = 2$

- We need to create a flow network
  - One node per shift
  - One node per employee
  - A source node and a sink node
  - An edge from the source to each employee node with capacity x
  - An edge from each employee to each available shift with capacity 1
  - An edge from each shift node s_i to the sink with capacity p_i



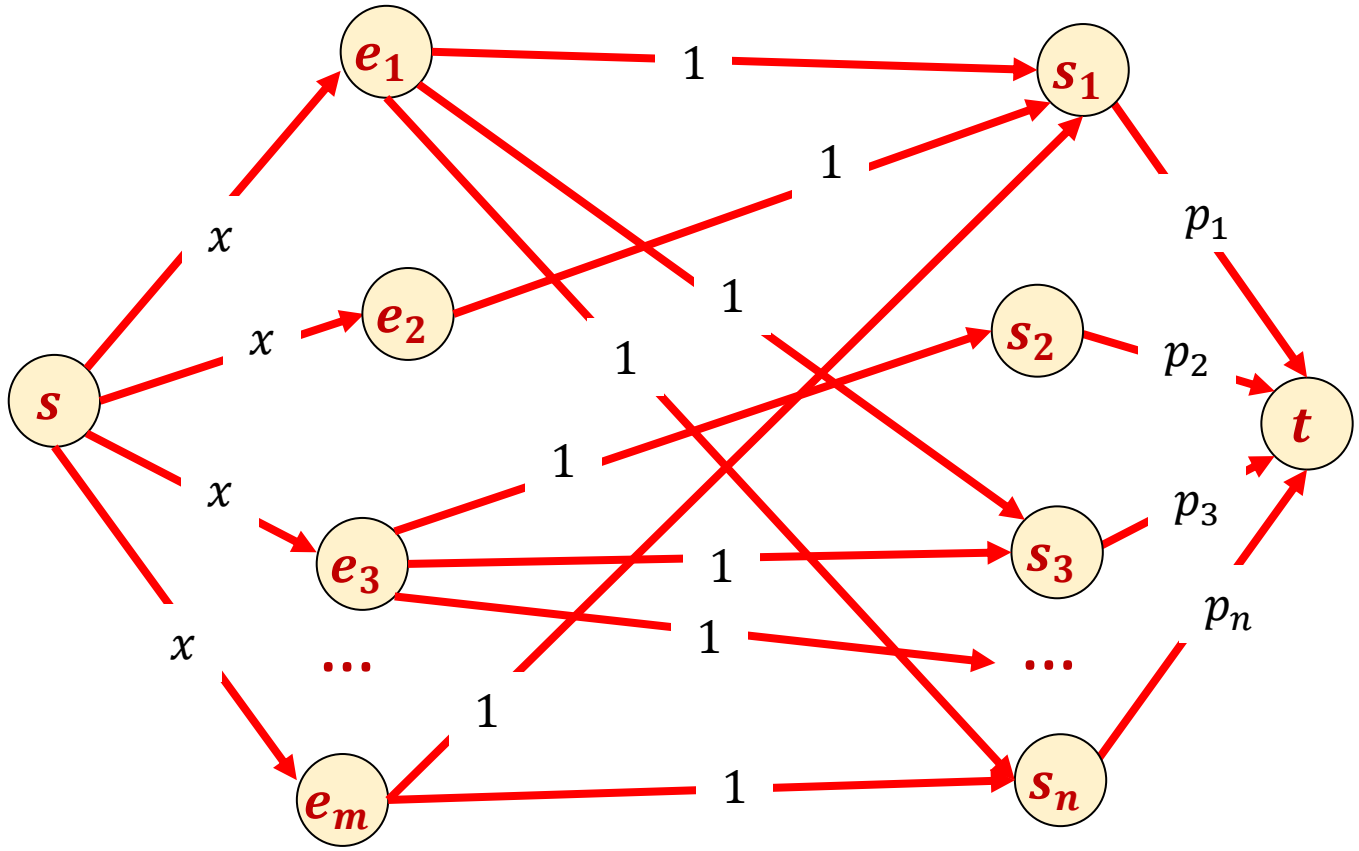
Running Time

- Constructing the graph
  - $n \cdot m$
- Running Max Flow
  - Which bound to use depends on the value of x and each p_i, so probably best to just use the Edmonds-Karp bound
  - $\Theta\big((n \cdot m)^2(n+m)\big)$



Correctness

- Valid flow ⇒ Valid answer
  - No employee is assigned to more than x shifts (capacity on s to e_i)
  - No employee is assigned to the same shift more than once (capacity of e_i to s_j)
  - No employee is assigned to an unavailable shift (by selection of edges to draw)
  - All shifts staffed if flow value is $\sum p_i$
- Valid answer ⇒ Valid flow
  - Suppose we had a way of staffing the shifts, we will show that there must be flow through the graph whose value matches $\sum p_i$
    - All capacity constraints will be observed
    - It will only use edges we drew
    - It will assign flow across $\sum p_i$ e_i-to-s_j shifts



Baseball Elimination

- The champion will be the one with the most wins at the end of the season
- A team is “eliminated” if it has become impossible for them to become the champion
- Given a current win/loss record for each team, and the remaining games in the season, determine whether a team has been eliminated

Input: The win/loss record for each team (w_i, ℓ_i), the number of games remaining between each pair of teams $r_{i,j}$, and the team x that we’re checking

Goal: Determine whether x can be the champion (i.e. there is a way to assign winners/losers to the remaining games such that x has the most wins)

Baseball Elmination

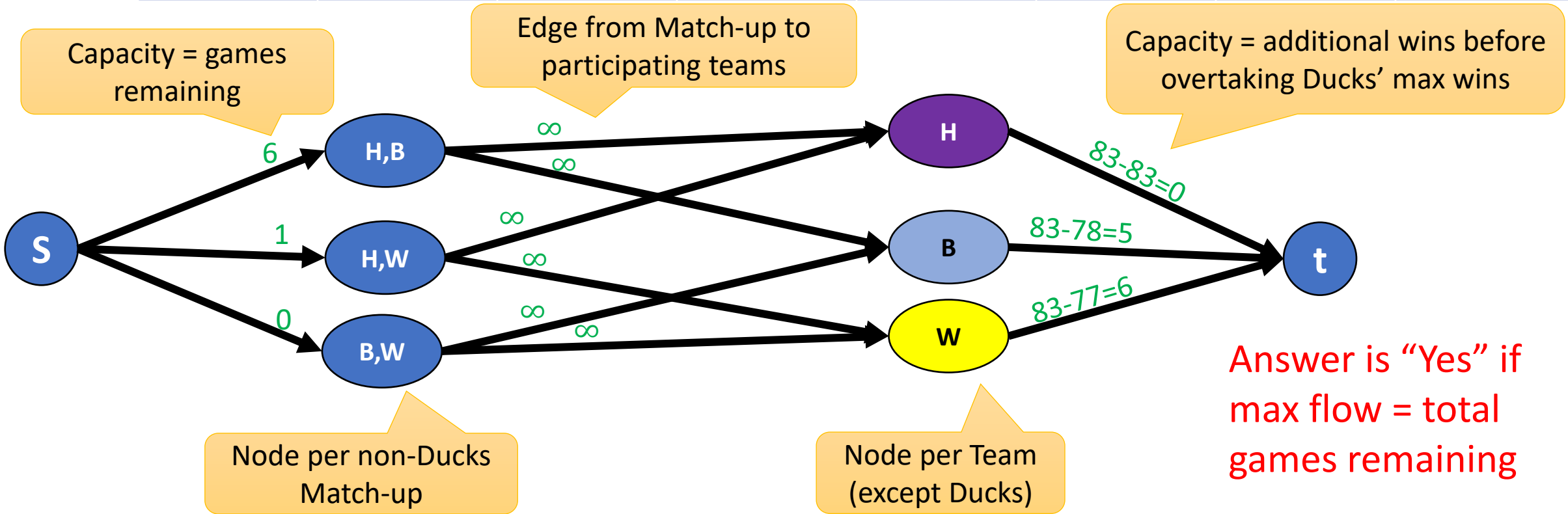
If the Huskies lose all of these, then the Ducks win them all

				Remaining Opponents			
Team	Wins	Losses	Games Left	Huskies	Ducks	Bruins	Wolverines
Huskies	83	71	8	--	1	6	1
Ducks	80	79	3	1	--	0	2
Bruins	78	78	6	6	0	--	0
Wolverines	77	82	3	1	2	0	--

Idea: Can we “distribute” wins so that all are less than the Duck’s best possible record?

Can The Ducks Win the Season?

				Remaining Opponents			
Team	Wins	Losses	Games Left	Huskies	Ducks	Bruins	Wolverines
Huskies	83	71	8	--	1	6	1
Ducks	80	79	3	1	--	0	2
Bruins	78	78	6	6	0	--	0
Wolverines	77	82	3	1	2	0	--



Constructing the flow network

- **Suppose we have n total teams**
  - Make a node for each team except for x
    - $n-1$ nodes
  - Make a node of each remaining matchup (t_i, t_j) that does not involve x
    - $(n-1)^2$ nodes
  - Add a source node s and a sink node t
    - 2 nodes
  - Draw an edge from s to each matchup node (t_i, t_j) with capacity $r_{i,j}$
    - $(n-1)^2$ edges
  - Draw an edge from each matchup node (t_i, t_j) to each team node t_i and t_j with capacity ∞
    - $2(n-1)^2$ edges
  - Draw an edge from each team node t_i to the sink t with capacity representing the number of wins team x has if they win all remaining games minus the number of wins t_i currently has
    - $n-1$ edges
- $|V| = \Theta(n^2), |E| = \Theta(n^2)$
- Running time: $\Theta(n^6)$

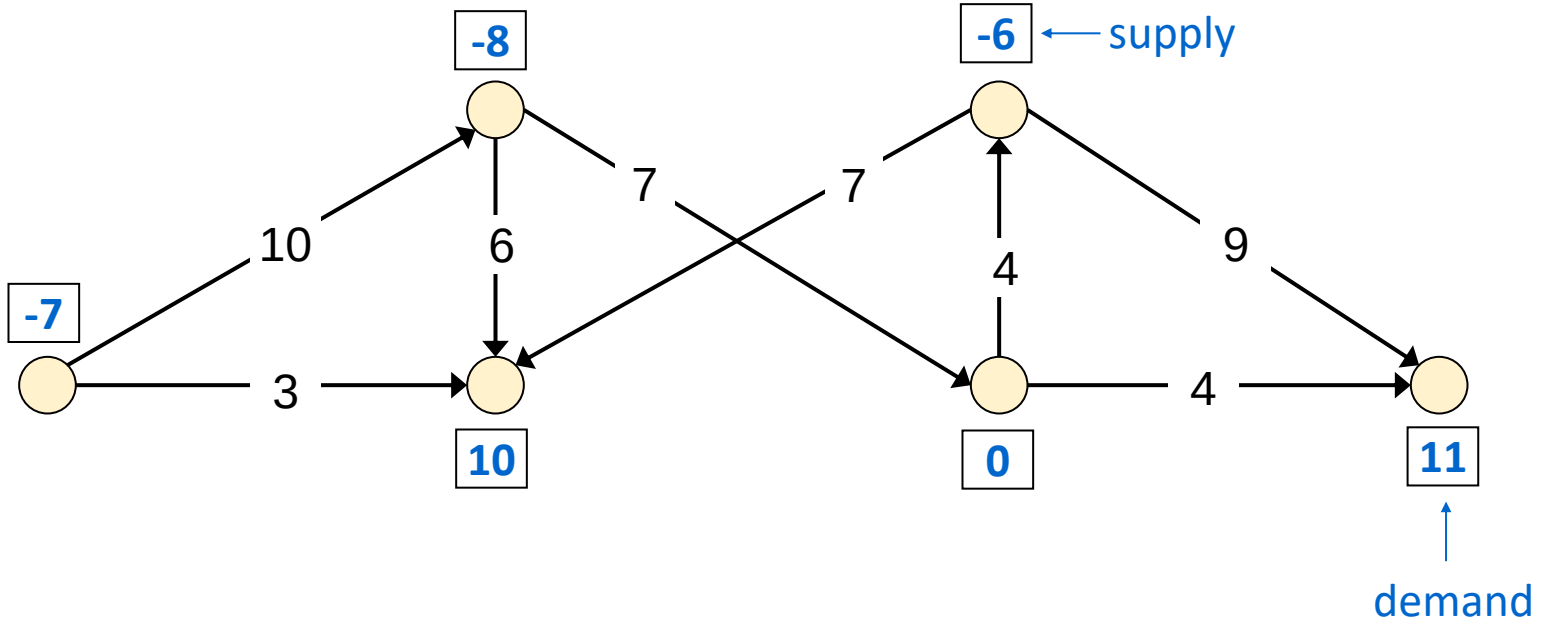
Correctness

- Valid flow ⇒ Valid answer
  - Claim: If we have flow matching the number of games remaining then x can win the season
  - The amount of flow going from each matchup edge to each team edge represents the number of times that team won in that matchup
  - Max flow equaling the number of games means we could assign winners to each remaining game
  - The capacity on the team-to-sink edges guarantees none of them won more games than x
- Valid answer ⇒ Valid flow
  - Claim: If we had a way for x to be champion, we could find flow through the graph to match the number of remaining games
  - Consider the collection of game outcomes which would cause this.
  - For each game, assign 1 unit of flow along the path $s, (t_i, t_j), w, t$ where w is the winner of the game
  - After doing this for all games we will not have violated any capacity constraints because we required that x would be the champion (and so no other team could have more wins)

Circulation with Demands

Nodes have either a “supply” (negative value) or “demand” (positive value)

We want to transport from our supply to our demand through a transportation network



Circulation with Demands

- Single commodity, directed graph $G = (V, E)$
- Each node v has an associated demand $d(v)$
  - Needs to receive an amount of the commodity: demand $d(v) > 0$
  - Supplies some amount of the commodity: "demand" $d(v) < 0$ (amount = $|d(v)|$)
- Each edge e has a capacity $c(e) \geq 0$.
- Nothing lost: $\sum_v d(v) = 0$.

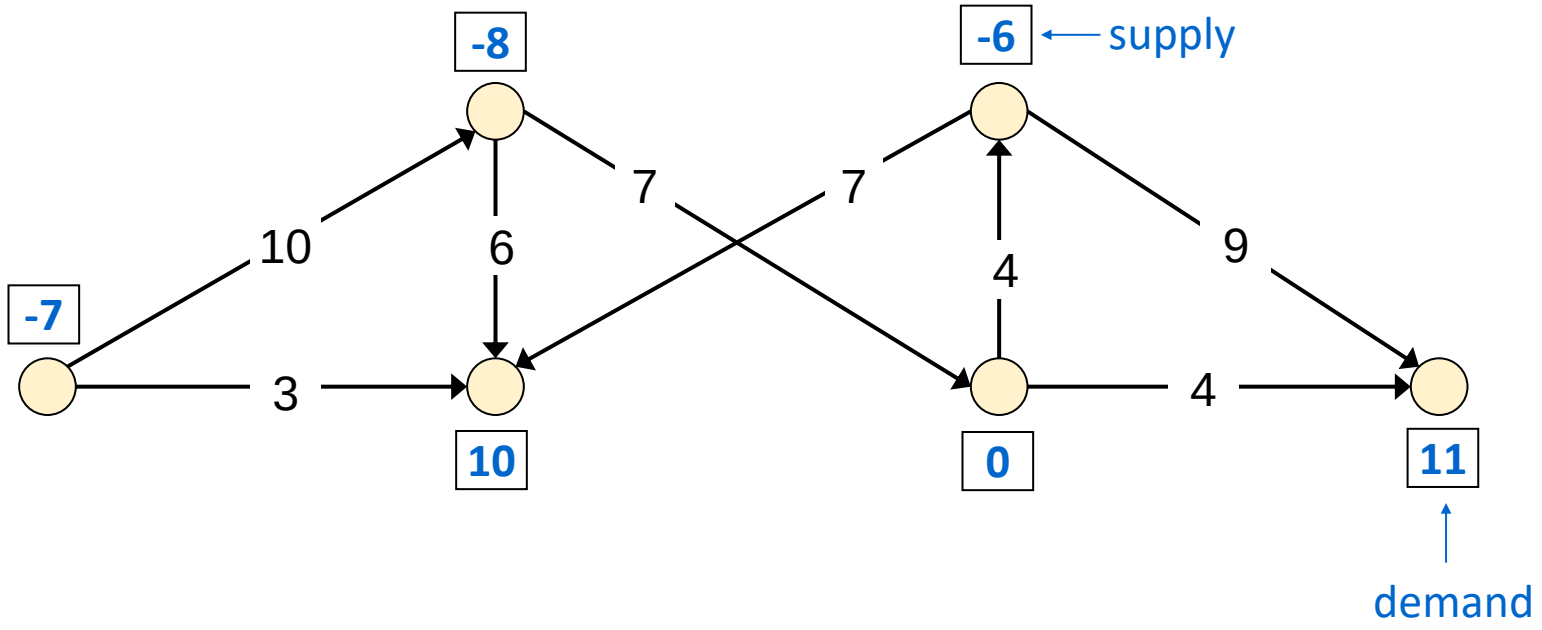
Defn: A circulation for (G, c, d) is a flow function $f: E \to \mathbb{R}$ meeting all the capacities, $0 \leq f(e) \leq c(e)$, and demands:
$\sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$.

Circulation with Demands: Given (G, c, d), does it have a circulation? If so, find it.

Circulation with Demands

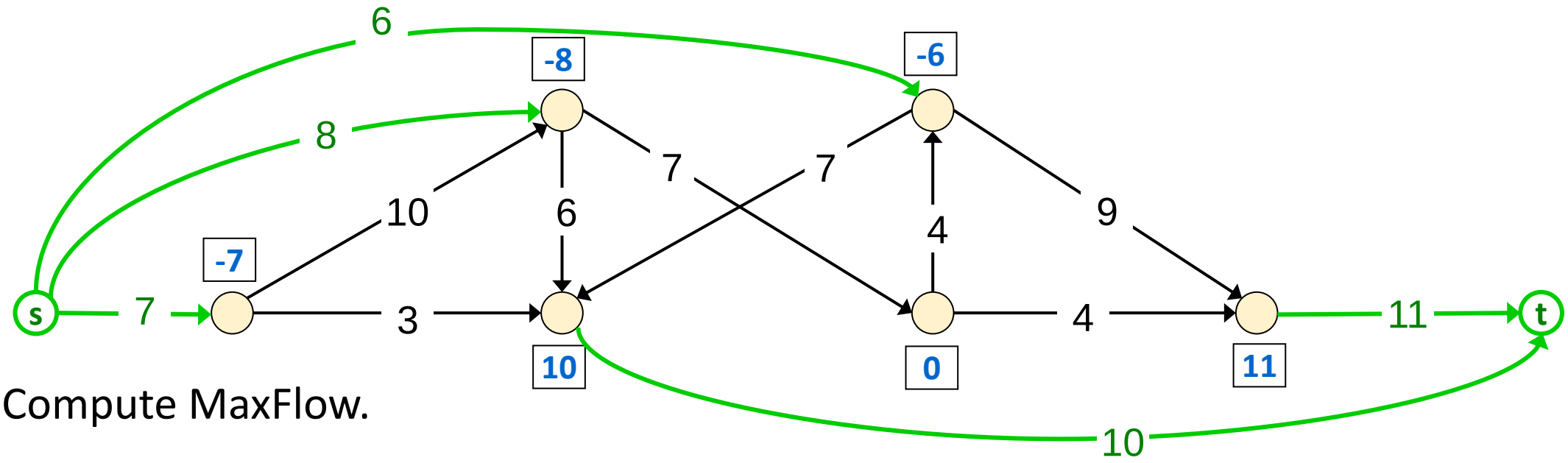
Defn: Total supply $\boldsymbol{D = \sum_{v:\, d(v)<0} |d(v)| = -\sum_{v:\, d(v)<0} d(v)}$.

Necessary condition: $\boldsymbol{\sum_{v:\, d(v)>0} d(v) = D}$ (no supply is lost)



Circulation with Demands using Network Flow

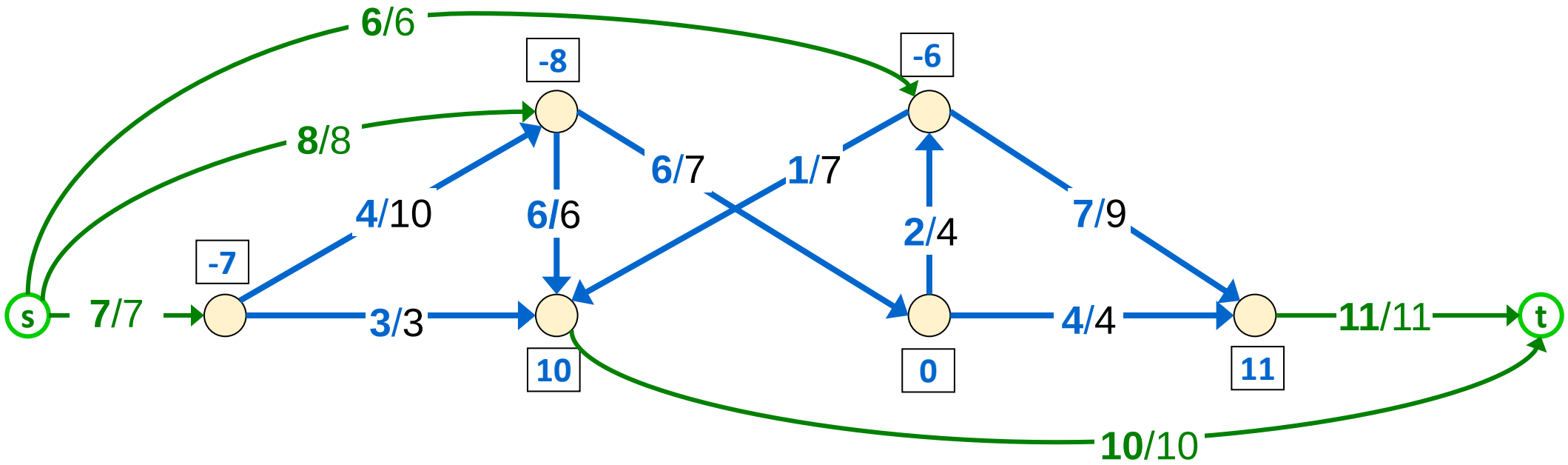
- Add new source s and sink t.
- Add edge (s, v) for all supply nodes v with capacity $|d(v)|$.
- Add edge (v, t) for all demand nodes v with capacity $d(v)$.



- Compute MaxFlow.

Circulation with Demands using Network Flow

- MaxFlow $\leq D$ based on cuts out of s or into t.
- If MaxFlow $= D$ then all supply/demands satisfied.

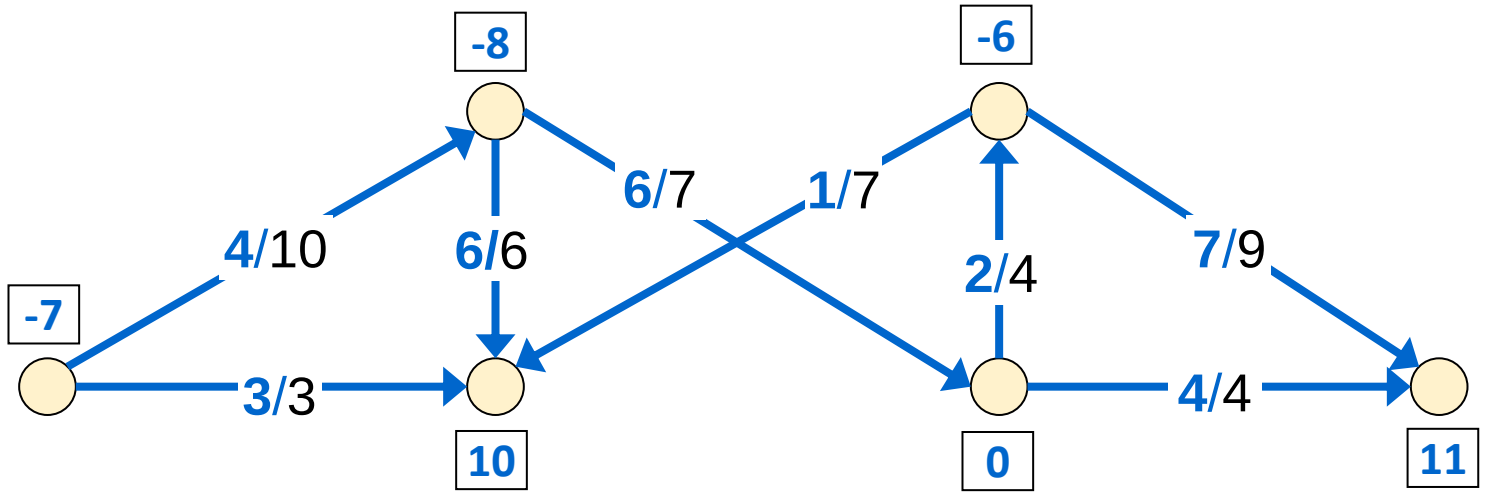


- Compute MaxFlow. Circulation iff value = D

Circulation with Demands using Network Flow

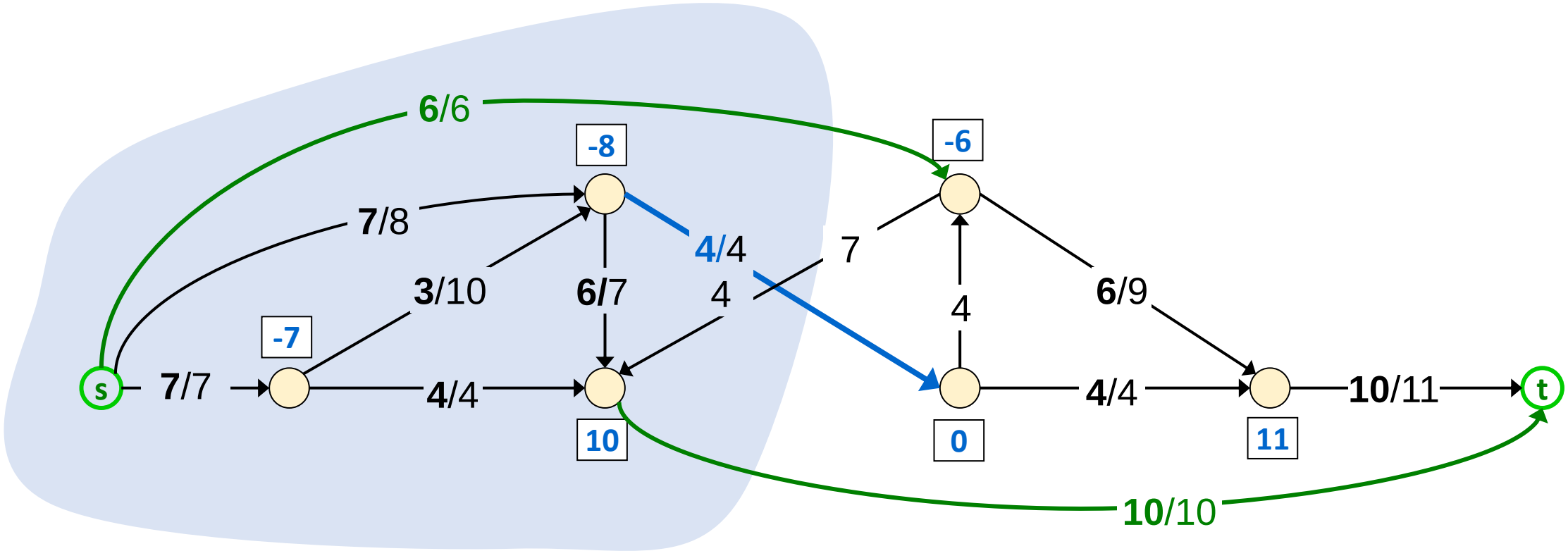
Circulation = flow on original edges

Circulations only need integer flows



Circulation with Demands using Network Flow

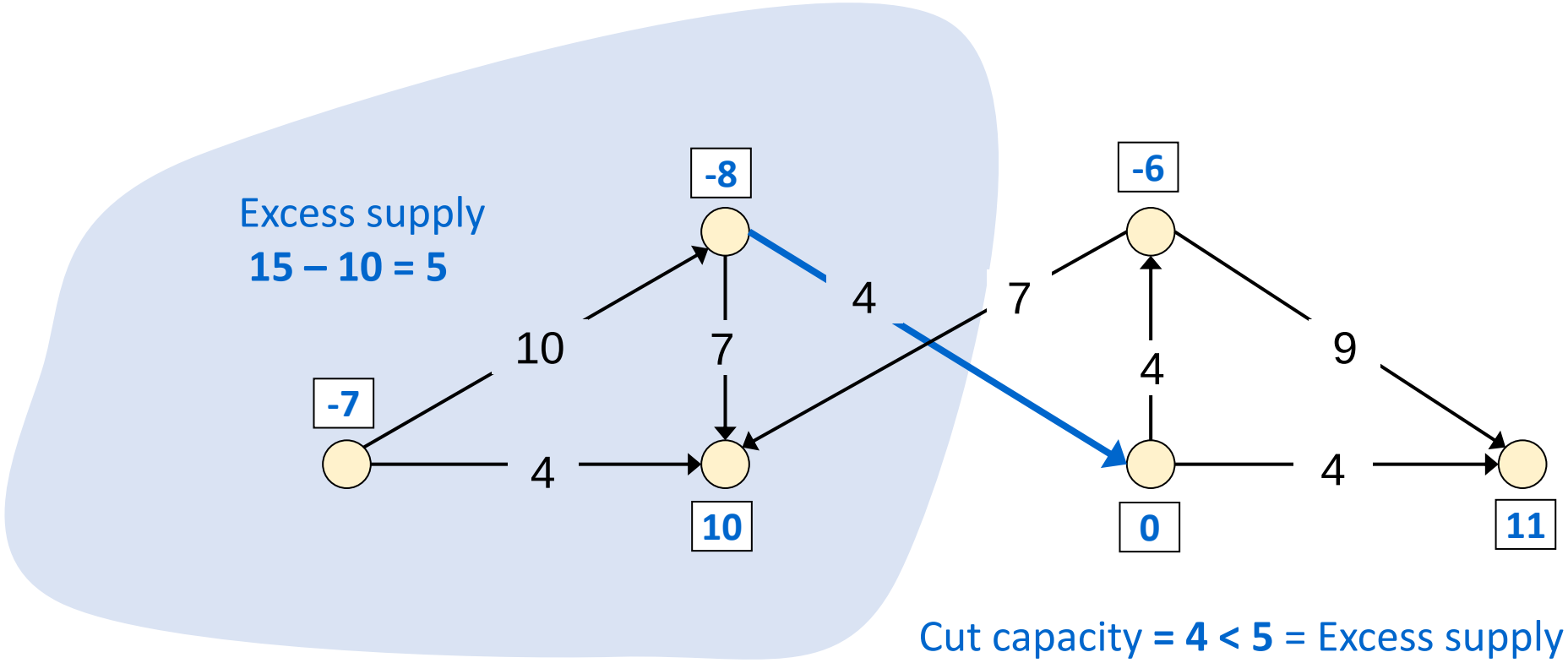
When does a circulation not exist? MaxFlow $< \boldsymbol{D}$ iff MinCut $< \boldsymbol{D}$.



Circulation with Demands using Network Flow

When does a circulation not exist? MaxFlow $< \boldsymbol{D}$ iff MinCut $< \boldsymbol{D}$.

Equivalent to excess supply on "source" side of cut smaller than cut capacity.



Some general ideas for using MaxFlow/MinCut

- If no source/sink, add them with appropriate capacity depending on application
- Sometimes can have edges with no capacity limits
  - Infinite capacity (or, equivalently, very large integer capacity)
- Convert undirected graphs to directed ones
- Can remove unnecessary flow cycles in answers
- Another idea:
  - To use them for vertex capacities c_v
    - Make two copies of each vertex v named v_{in}, v_{out}