CSE 417 Autumn 2025

Lecture 23: More Max Flow Applications

Nathan Brunelle

Flow Graph

Defn: An s-t flow in a flow network is a function $f: E \to \mathbb{R}$ that satisfies:

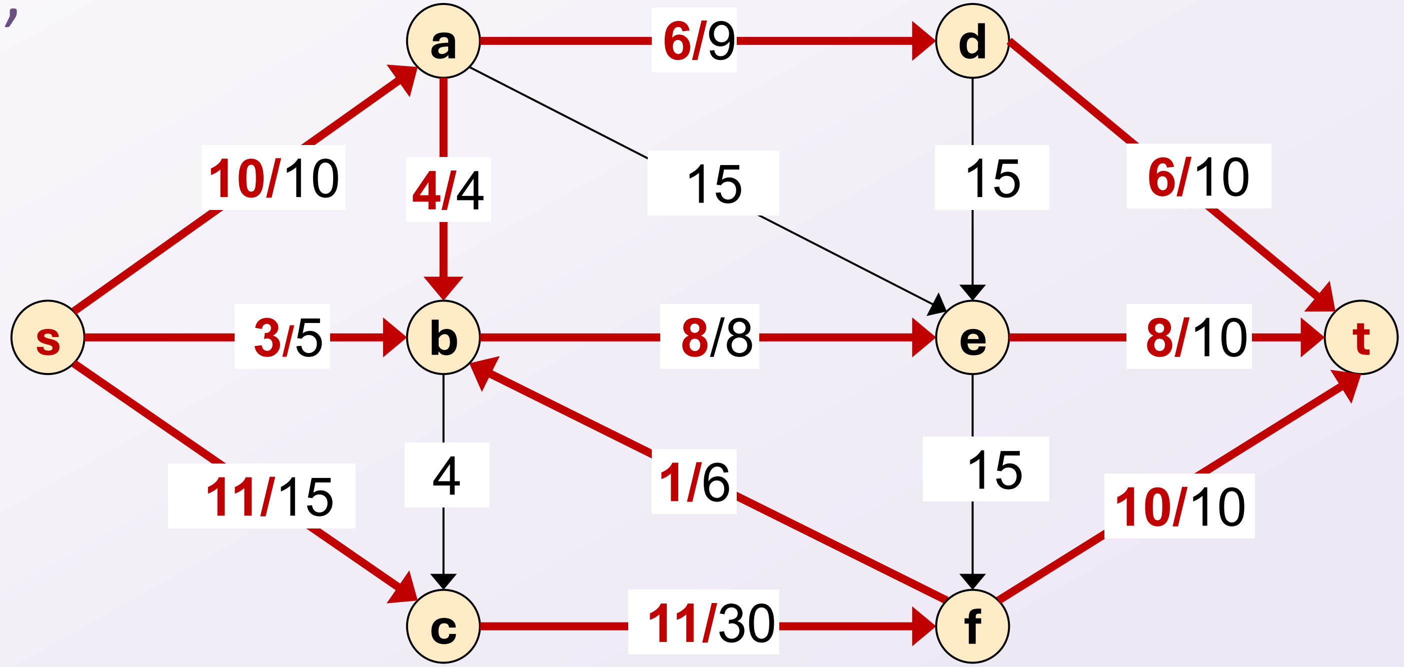
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity constraints]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [flow conservation]

Defn: The value of flow f,

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

Only show non-zero values of f

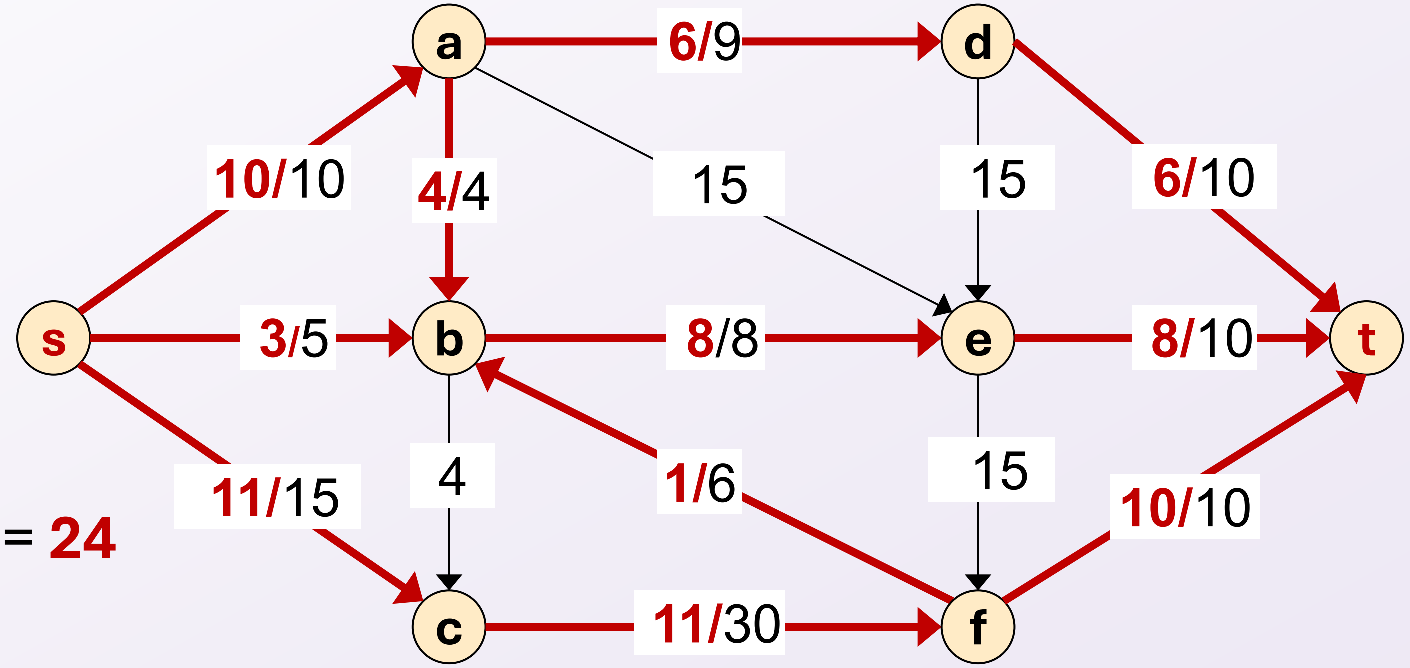
value = **24**



Maximum Flow Problem

Given: a flow network

Find: an s-t flow of maximum value



value = **24**

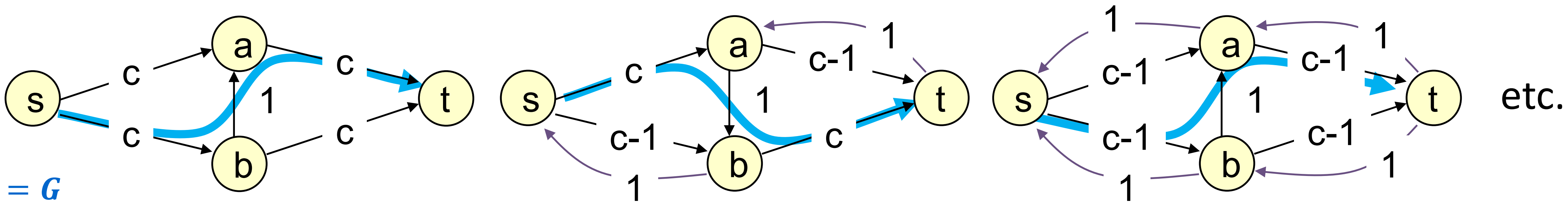
Ford-Fulkerson Running Time

Worst case runtime $O(mnC)$ with integer capacities $\leq C$.

- $O(m)$ time per iteration.
- At most nC iterations.
- This is "pseudo-polynomial" running time.

- May take exponential time, even with integer capacities:

c $= 10^9$, say

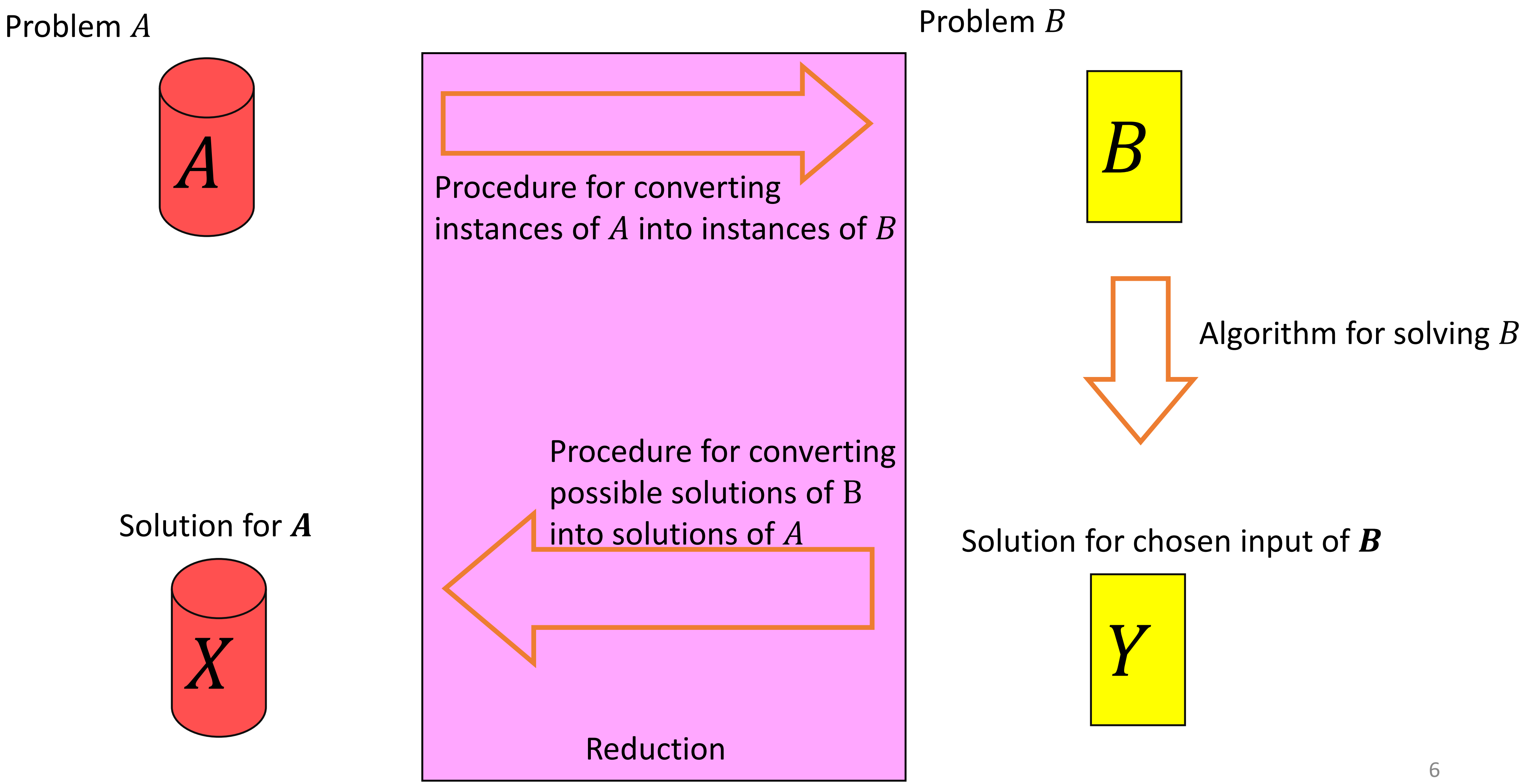


$G_f = G$

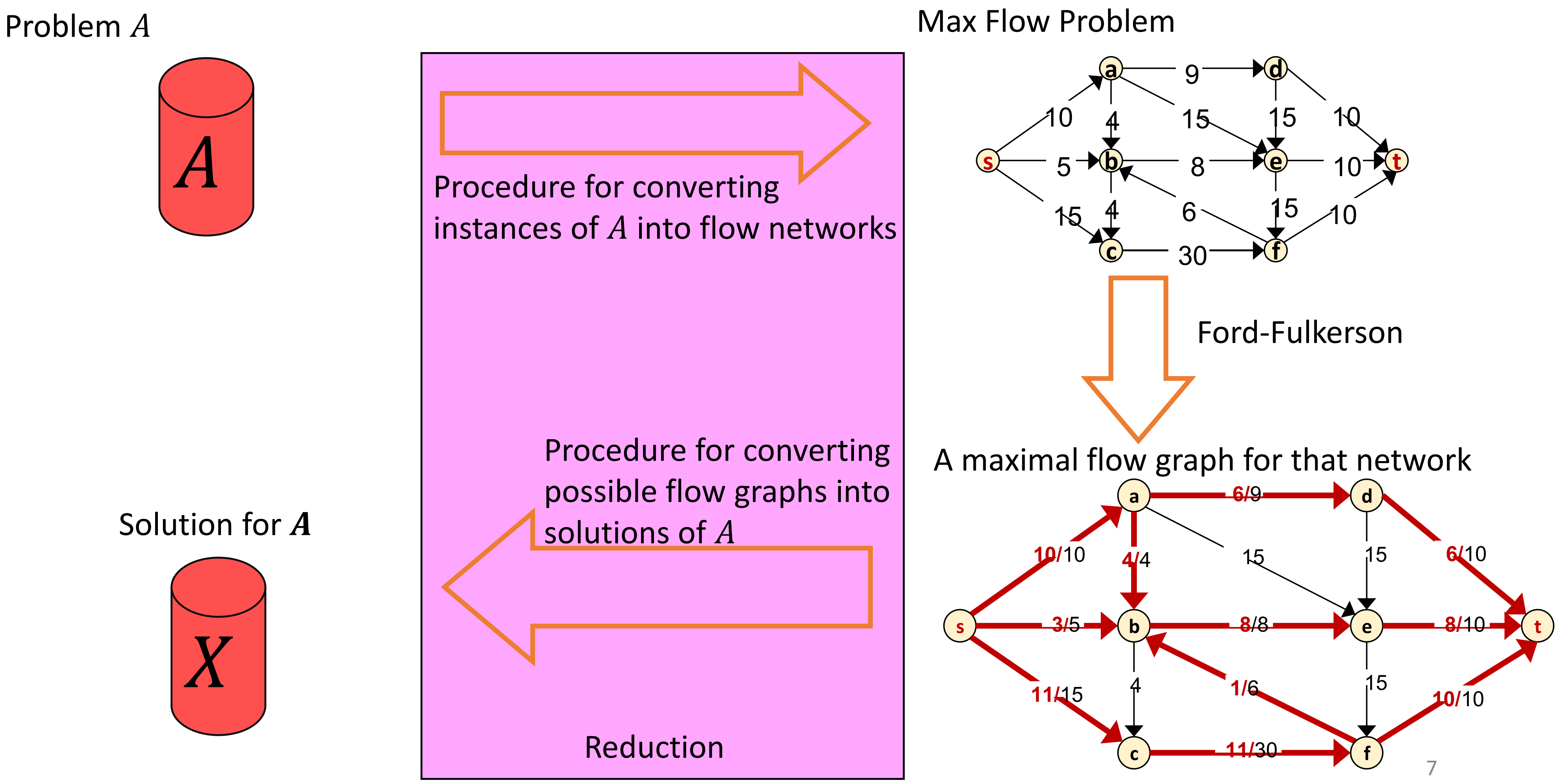
Applications of Max Flow

- Max flow is most useful when paired with reductions
- Reduction idea:
  - Create an algorithm for a new problem by transforming it into a different problem that can be solved by a preexisting algorithm
- Reduction Definition - A pair of procedures:
  - One that takes inputs for the new problem and transforms them into inputs for the old problem
  - One that takes solutions from the old problem and converts those into solutions for the new problem
    - Note: this second procedure only needs to apply to solutions to inputs that could possible come from the reduction (i.e. it does not have to work for every possible solution)
- The way we'll use max flow:
  - Start with a non max flow problem
  - **Write a procedure to convert its input to a flow network**
  - Use Ford-Fulkerson to find the max flow through the network
  - **Use that max flow to find the solution to our non max flow problem.**

Reductions


Problem A
A
Procedure for converting instances of A into instances of B
Problem B
B
Algorithm for solving B
Procedure for converting possible solutions of B into solutions of A
Solution for $\boldsymbol{A}$
X
Solution for chosen input of $\boldsymbol{B}$
Y
Reduction

Max Flow Reductions



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

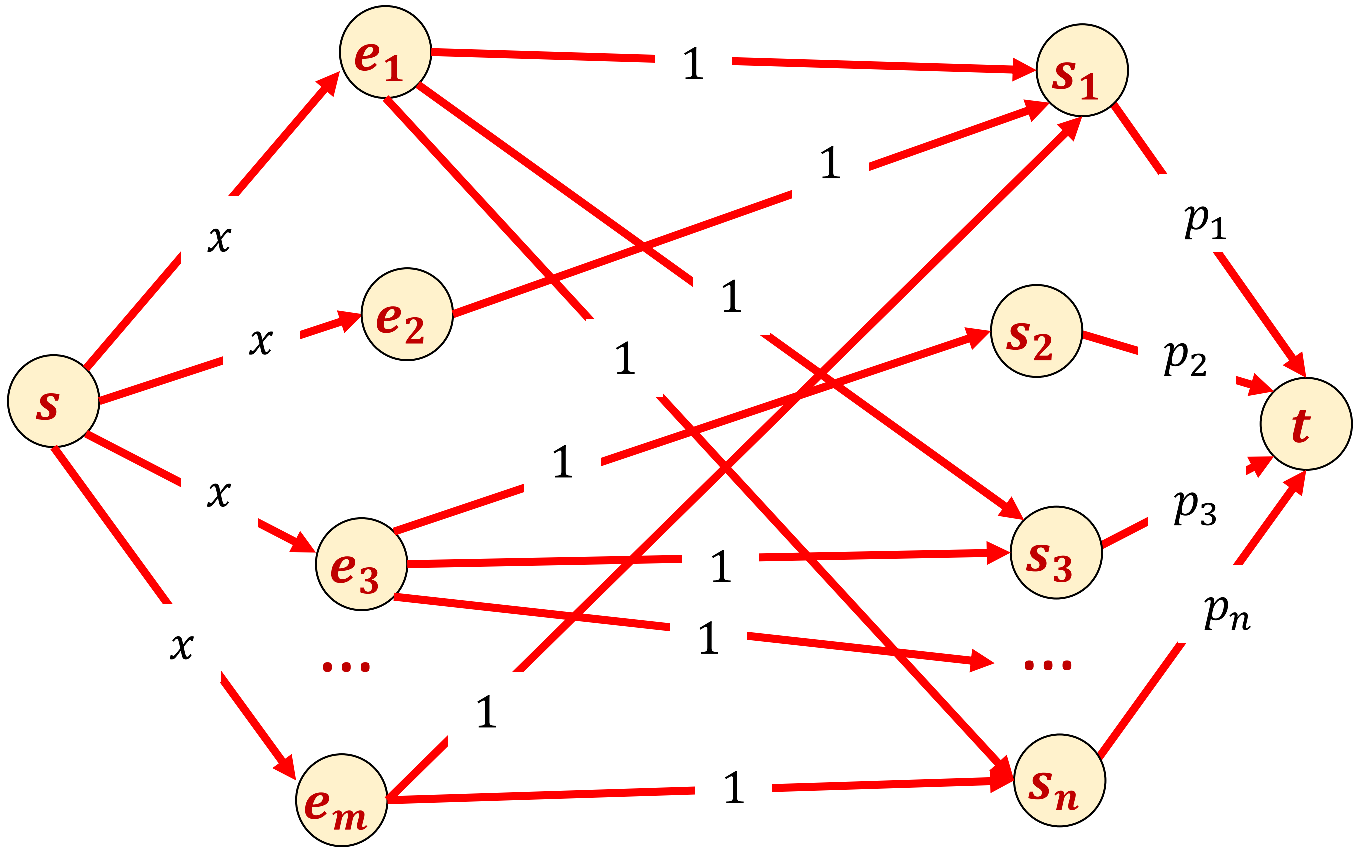
Employees:
1. 6am, 9am, 3pm
2. 6am, 9am, 12pm
3. 6am, 3pm

$x = 2$

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



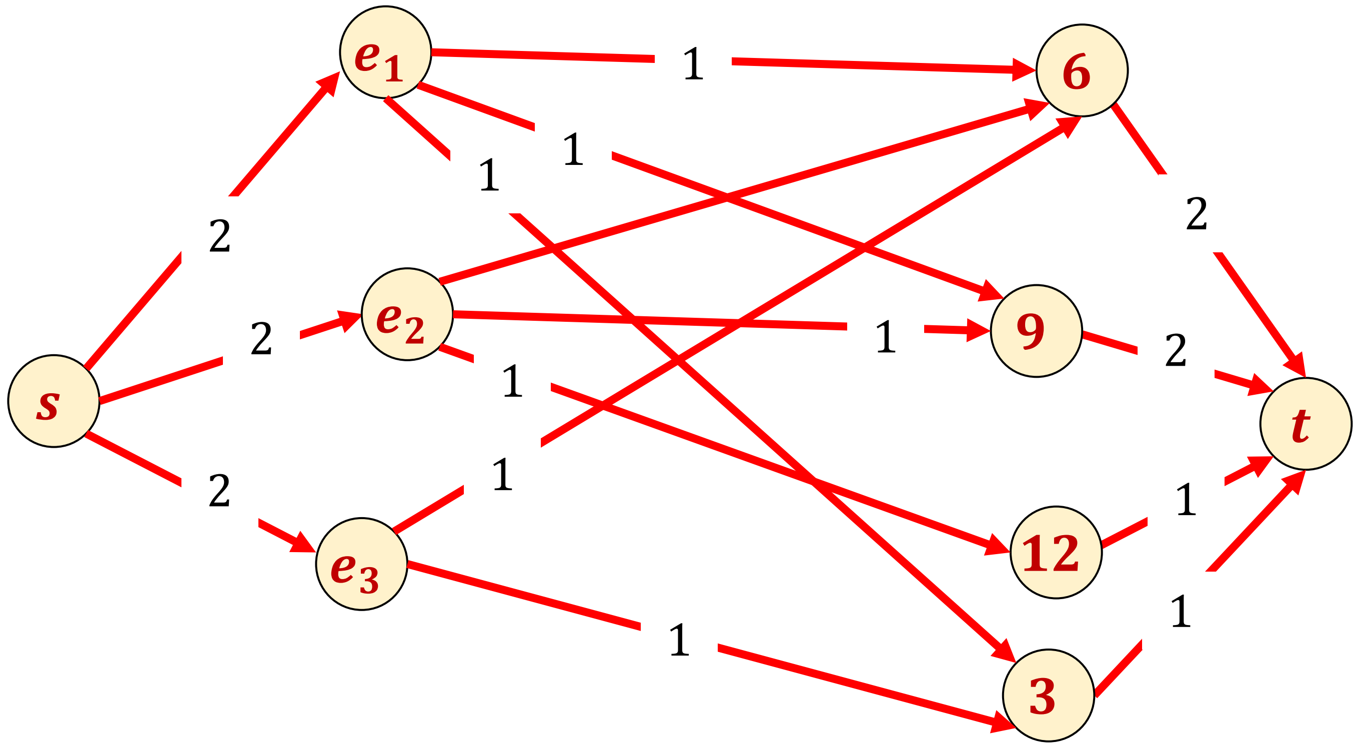
Reducing to Max Flow

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



Shift Scheduling Reduces to Max Flow

Shift Scheduling

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

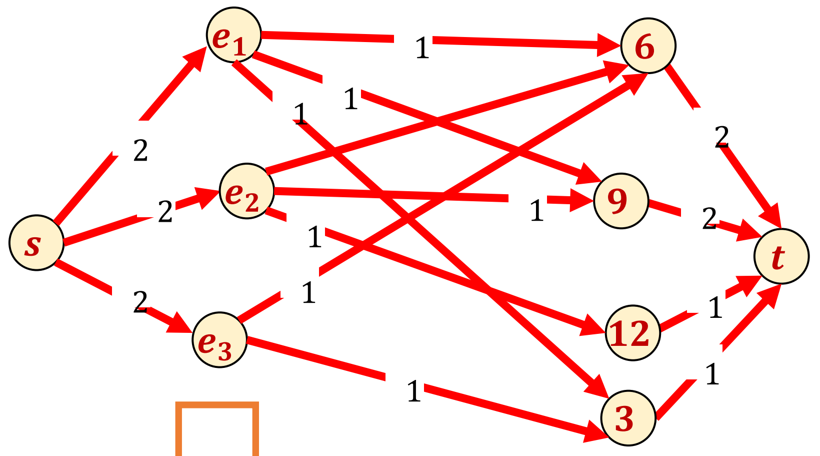
$x = 2$

Nodes: One per employee, One per shift, source, sink
Edges: s to employees with capacity x, employees to available shifts with capacity 1, shifts to sink with capacity p_i

If an employee-shift edge has flow, assign the employee to that shift

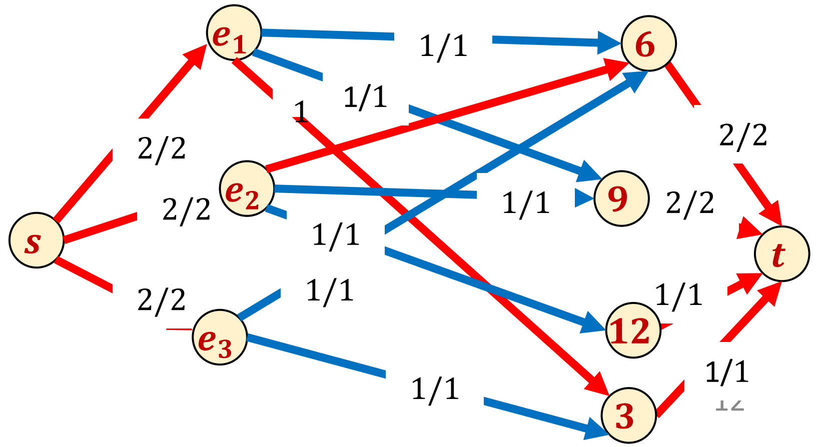
Reduction

Max Flow Problem



Ford-Fulkerson

A maximal flow graph for that network

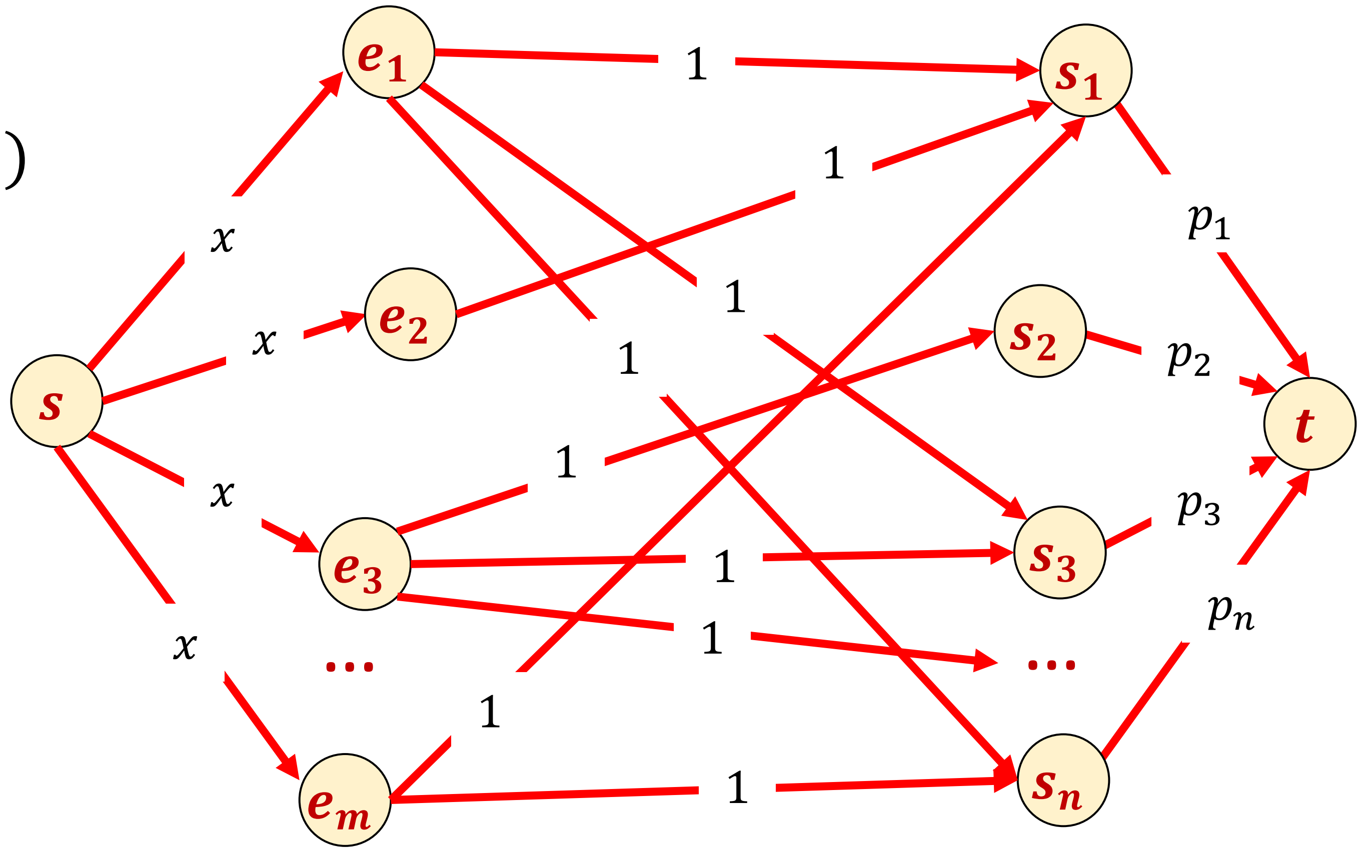


Schedule

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

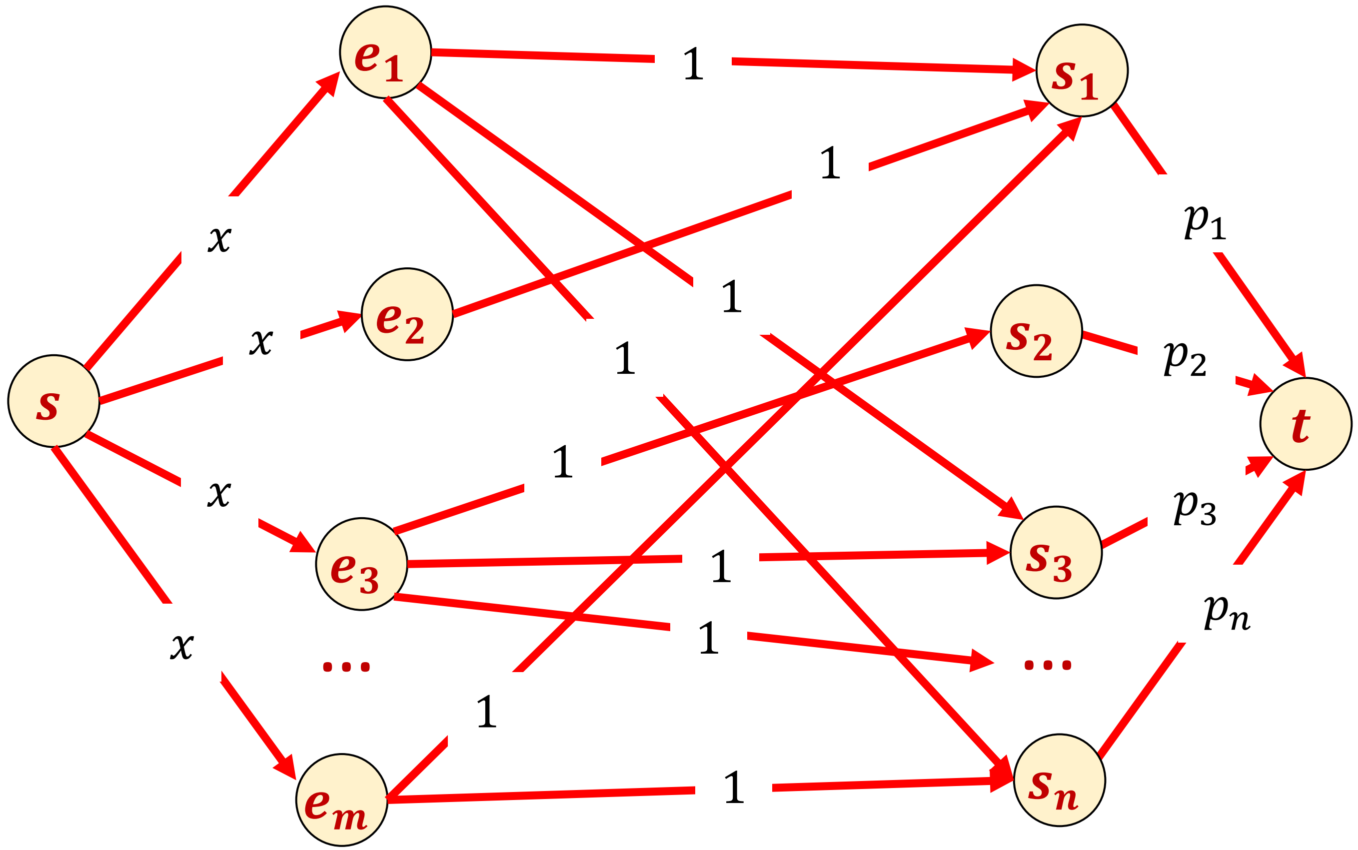
Running Time

- Constructing the graph
  - Nodes: $n + m + 2$
  - Edges: not more than $n \cdot m$
  - Largest capacity: $C = \max(x, p_1, \ldots, p_n)$
- Running Max Flow
  - $\Theta(Cn^2m + Cnm^2)$



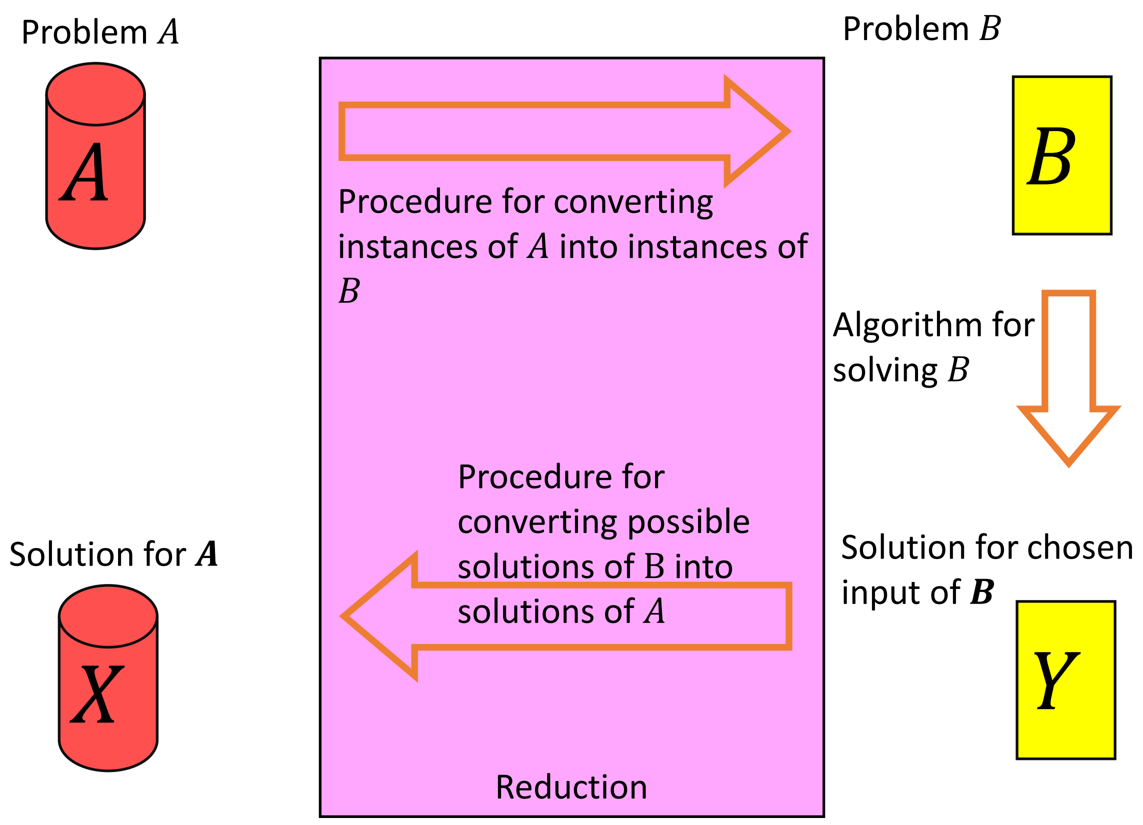
Correctness

- 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



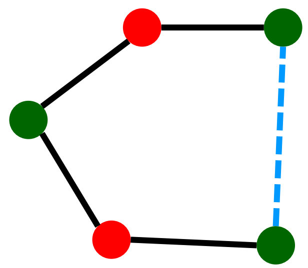
Reductions and Correctness

- A valid answer to the chosen problem B input produces a valid answer to the original problem A
  - Our reduction produces a meaningful result
- A valid answer to the original problem A results in a valid answer to the chosen problem B input
  - If there was a better answer for A, then the algorithm for B would have found it



Recall: Bipartite Graph

Definition: An undirected graph $\boldsymbol{G}$ 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 $\boldsymbol{G}$ 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 $\boldsymbol{G = (L \cup R, E)}$

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

Maximum Bipartite Matching

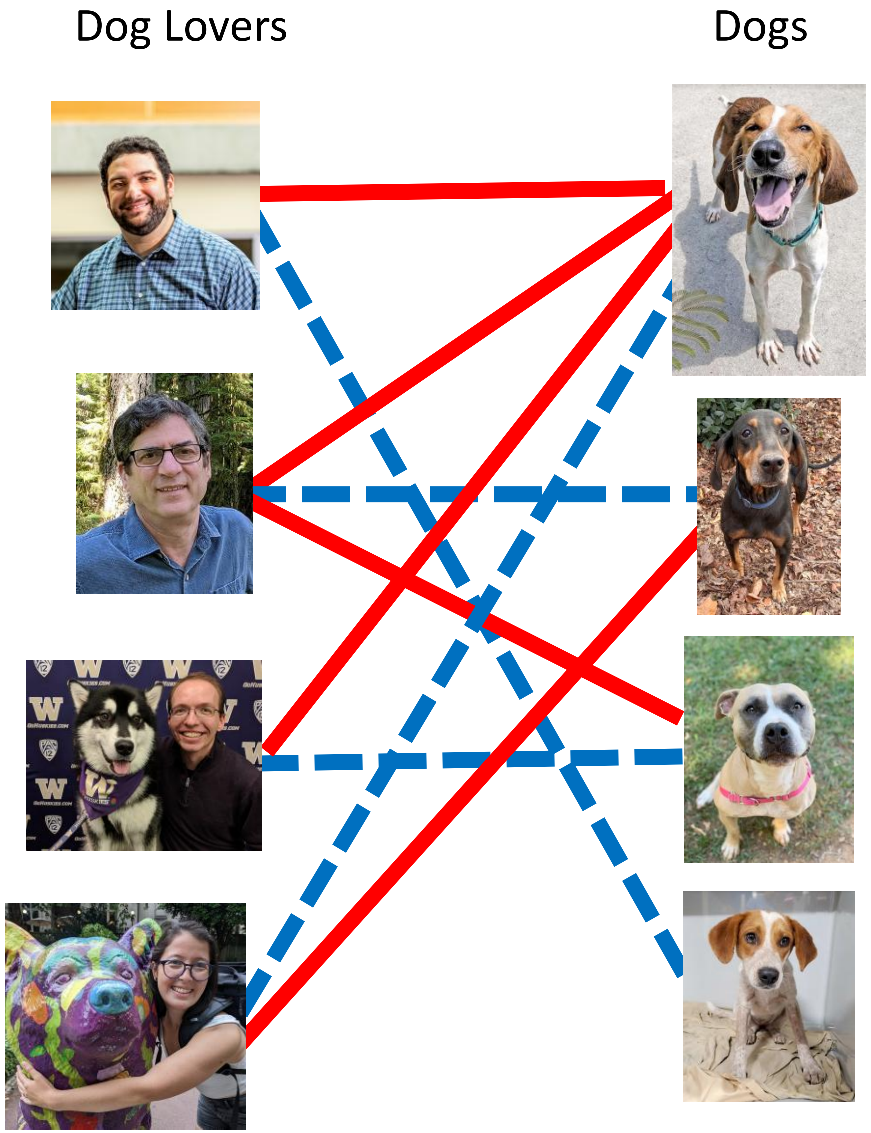
Dog Lovers

Dogs

Maximum Bipartite Matching

A (non-maximum) bipartite matching

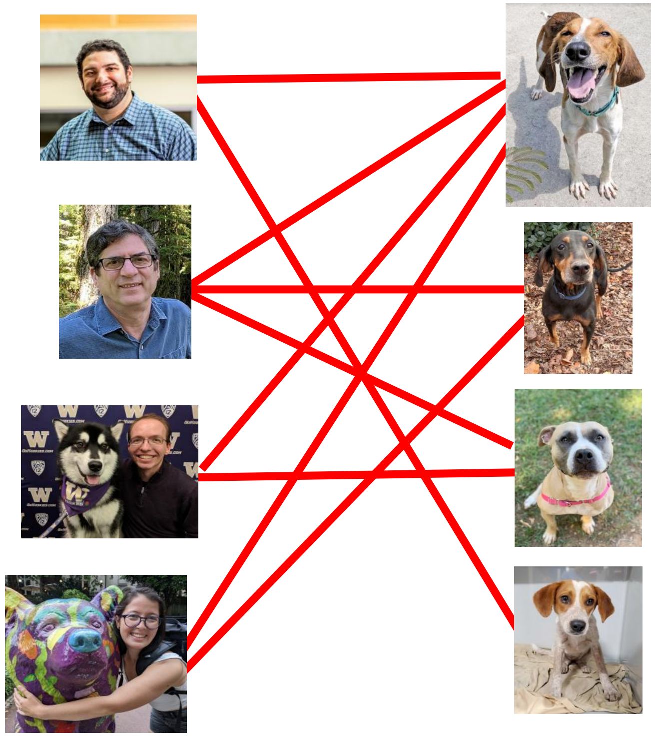
Maximum Bipartite Matching



A maximum bipartite matching!

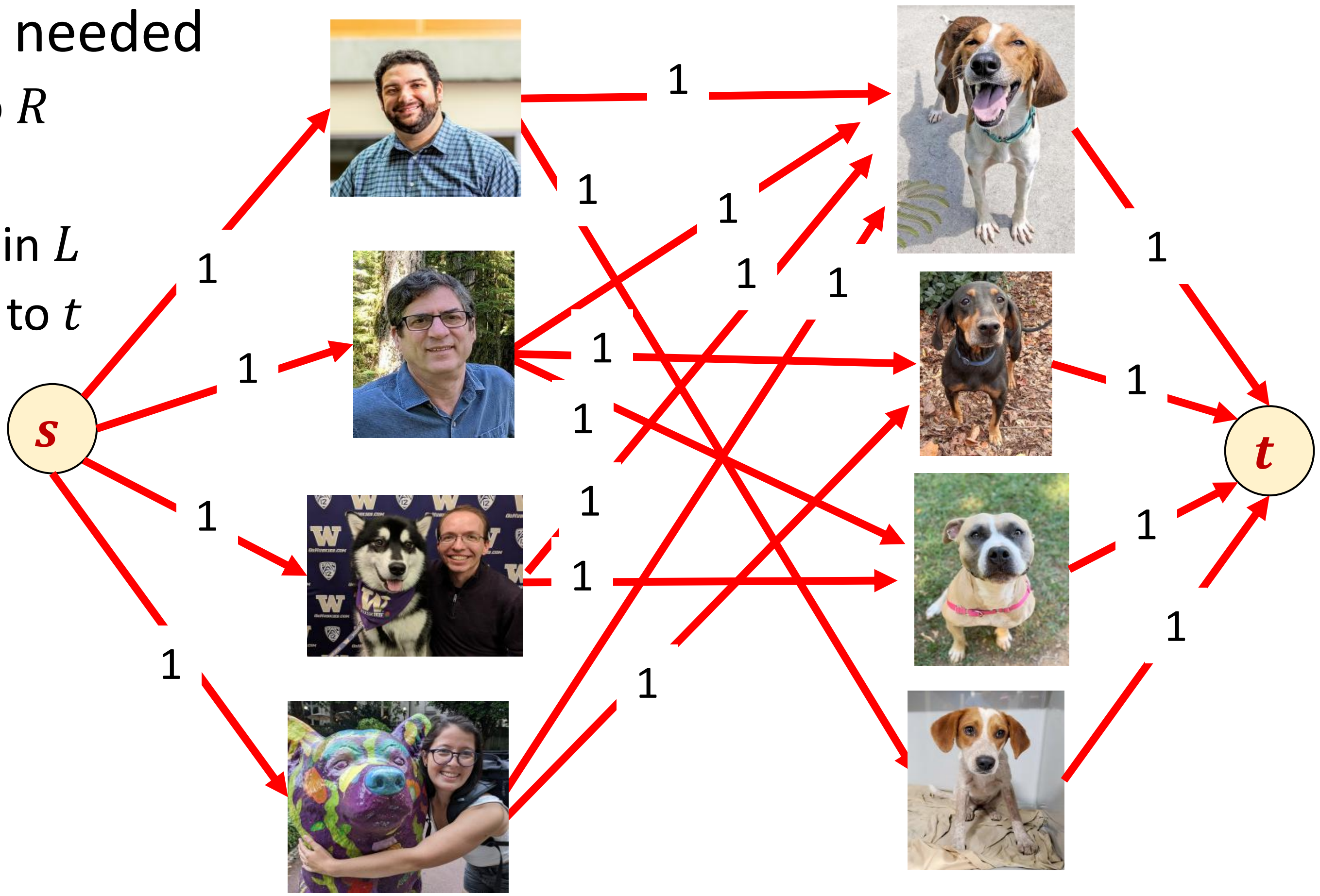
Reducing to Max Flow

- 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

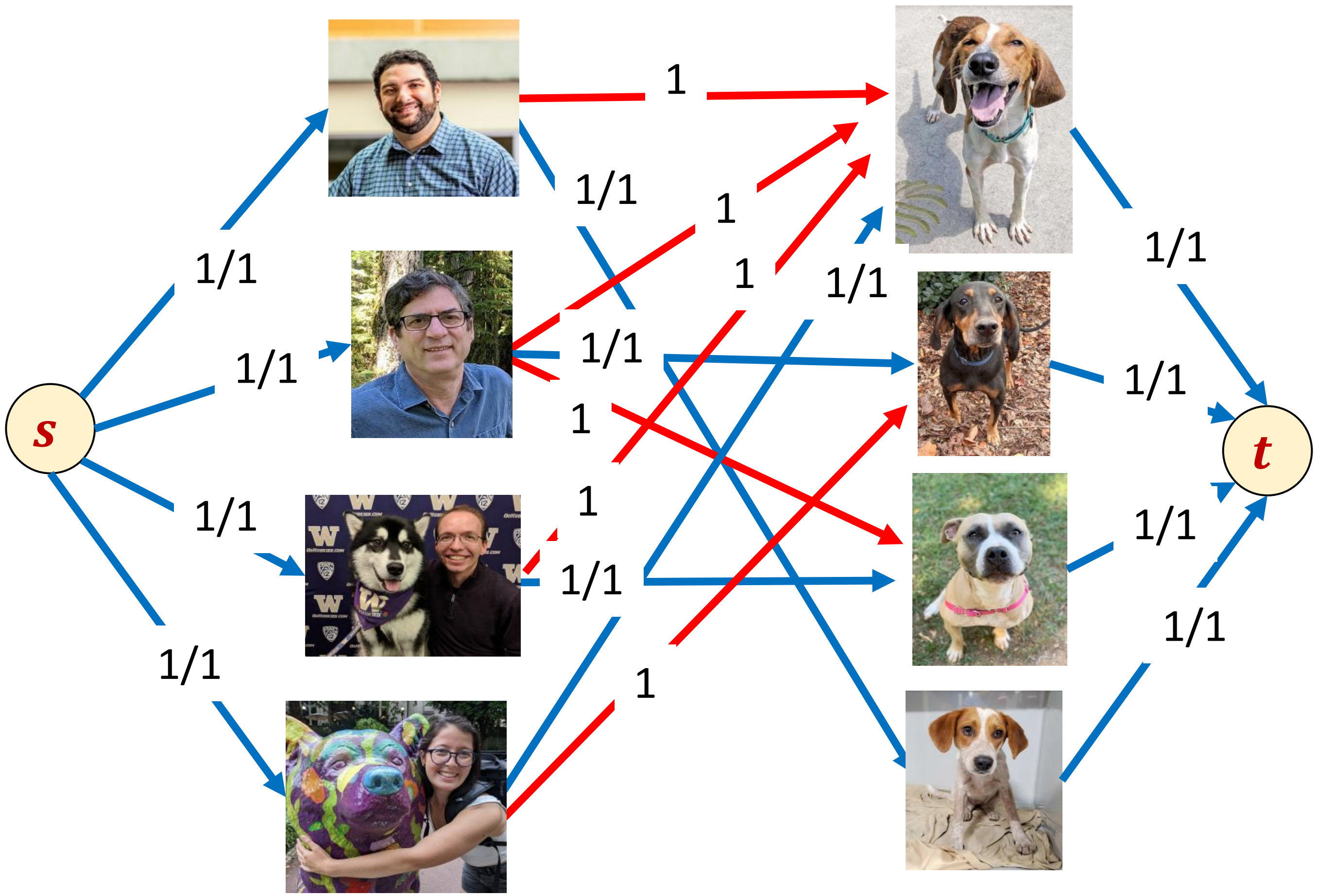
- We need to create a flow network
  - Add/remove any edges/nodes as needed
    - Make each edge directed from L to R
    - Add nodes s and t
    - Draw an edge from s to each node in L
    - Draw an edge from each node in R to t
  - Identify a source/sink
    - Make s the source and t the sink
  - Give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



Reducing to Max Flow

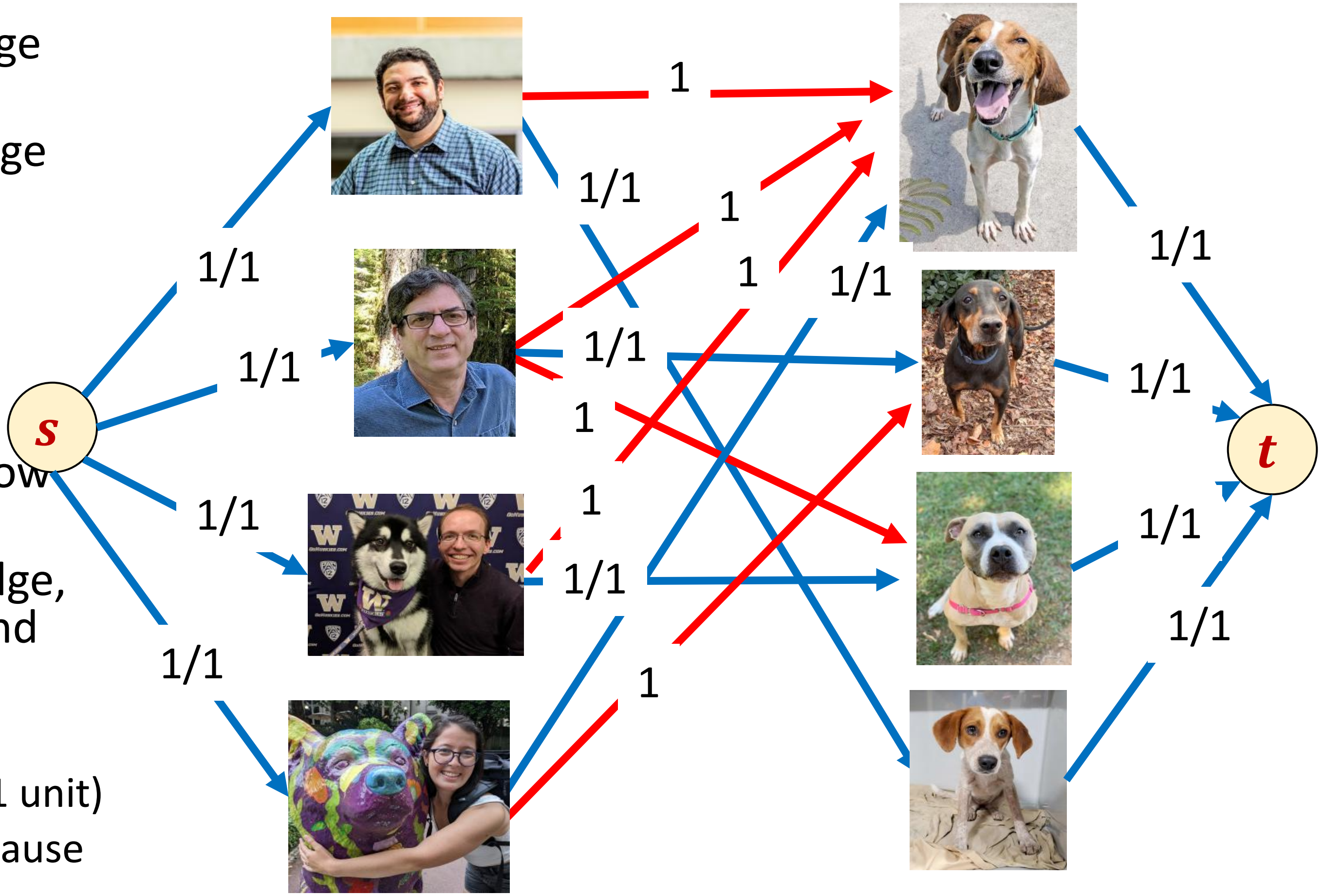
- Next, run Edmonds-Karp
- Use the answer to select a Matching
  - $M =$ the set of L-to-R edges with flow
  - $M = \{e : e \in L \times R, f(e) = 1\}$

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



Correctness

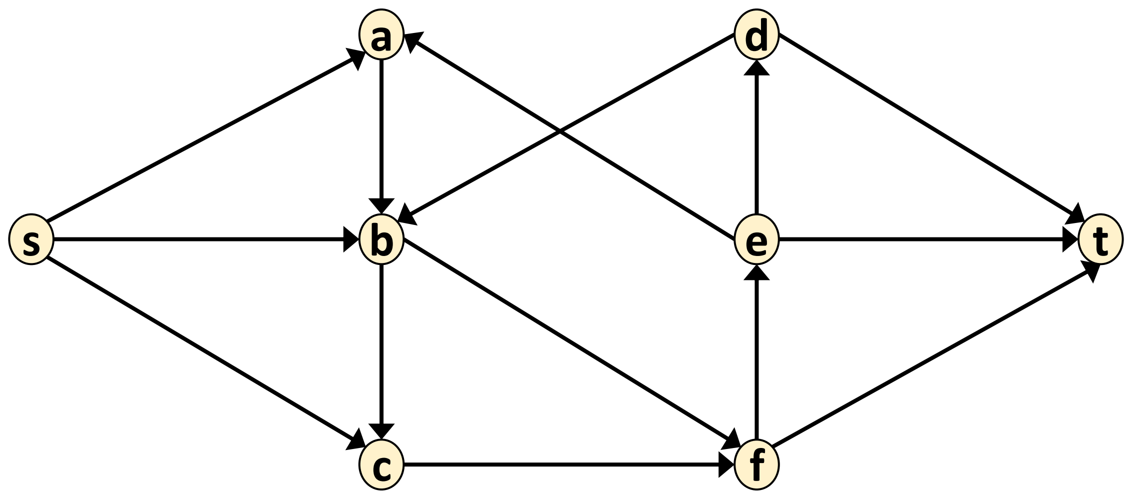
- Valid flow ⇒ Valid answer
  - Need to show: M is a valid match
  - Requirement 1: $M \subseteq E$
  - Requirement 2: each node is part of at most one match
    - No node in L is adjacent to more than one edge because the capacity of the edge from s is 1
    - No node in R is adjacent to more than one 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.

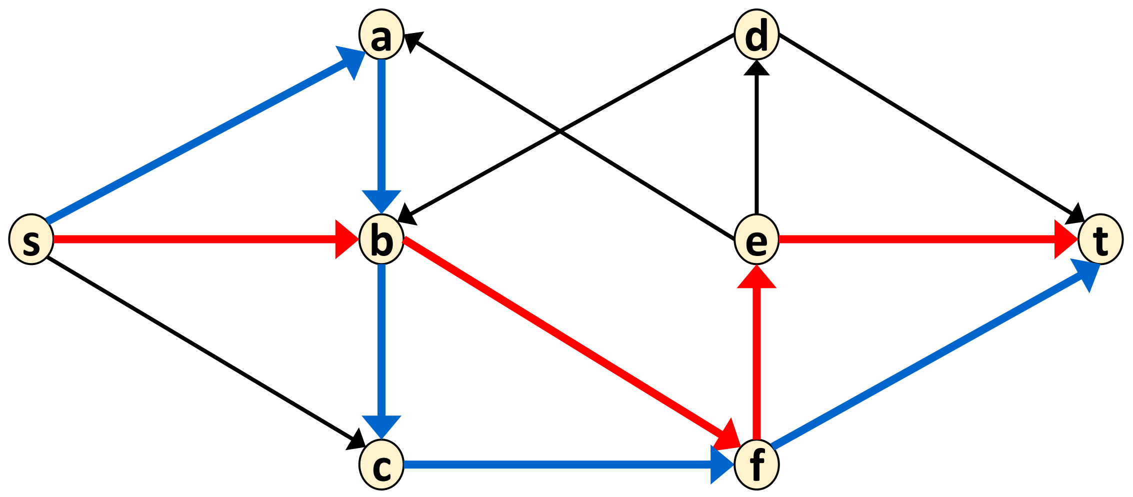
Edge disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.
Find: the maximum # of edge-disjoint simple s-t paths in G.



Edge-Disjoint Paths – Example of size 2

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

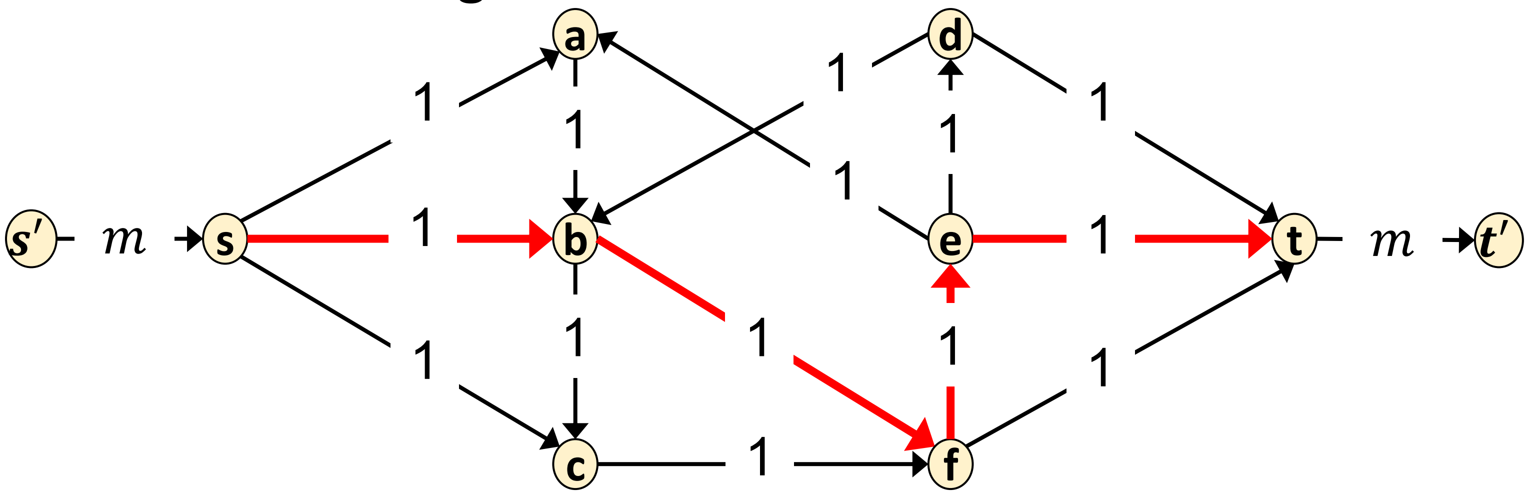
Edge disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.
Find: the maximum # of edge-disjoint simple s-t paths in G.



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 m (number of edges)
  - At most every edge is its own path
- Compute max flow
- Use all edges with flow



Running Time:

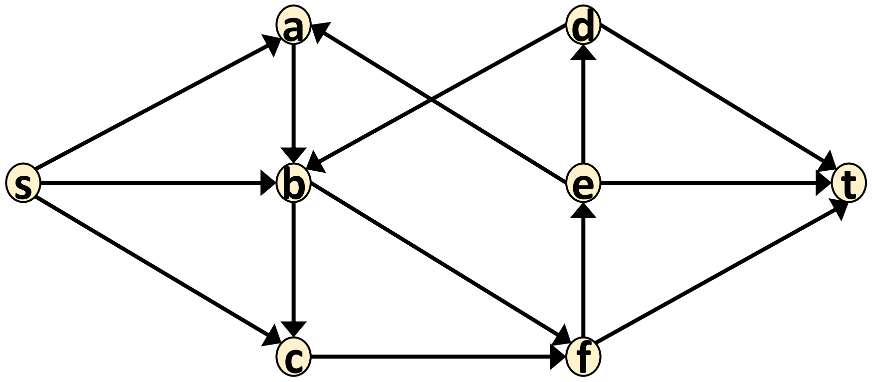
Constructing the flow network: $O(n+m)$

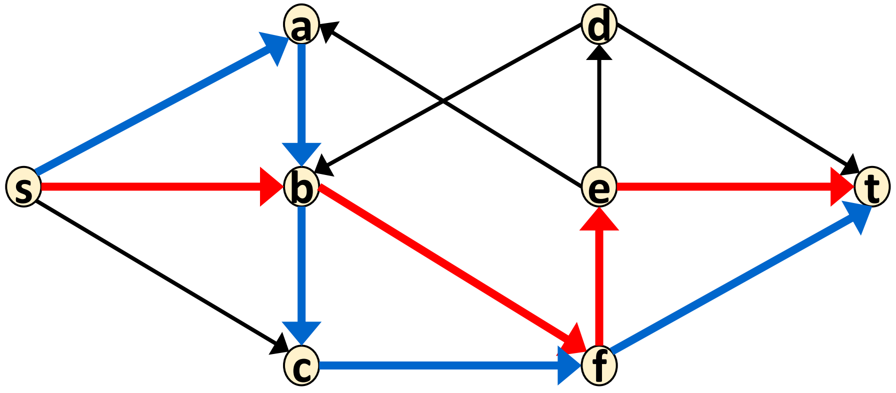
Computing Max Flow: $O(nm^2)$

Overall: $\Theta(nm^2)$

Edge Disjoint Paths Reduction to Max Flow

Edge Disjoint Paths



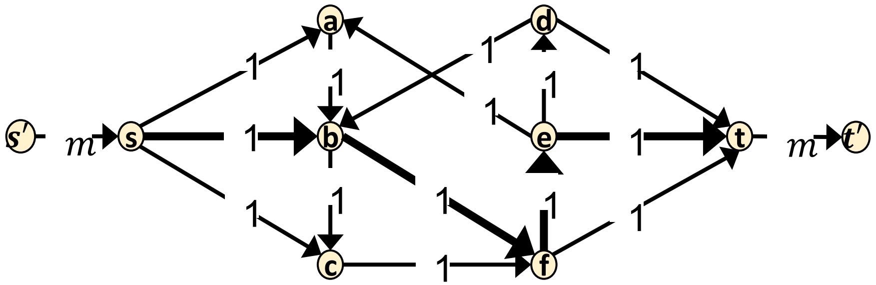


Add a new source and sink, add edge from new source to S with capacity m, edge from t to new sink with capacity m, add capacity 1 to all original edges

If an employee-shift edge has flow, assign the employee to that shift

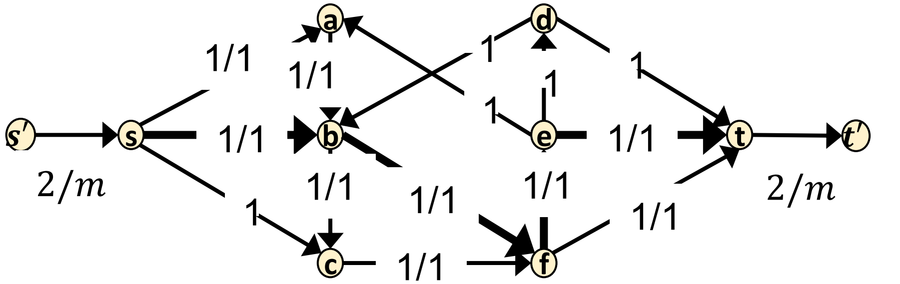
Reduction

Max Flow Problem



Ford-Fulkerson

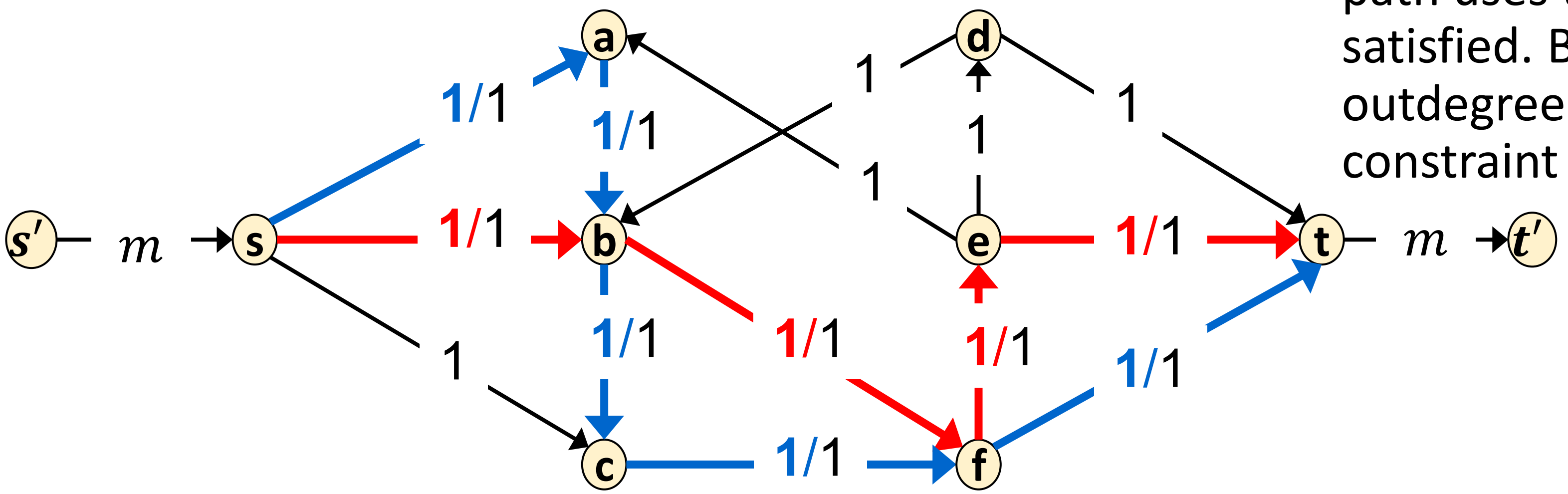
A maximal flow graph for that network



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 m
- Compute max flow
- Use all edges with flow



Theorem: MaxFlow = # edge-disjoint paths

Valid flow ⇒ Valid answer:

Need to show: no edge is used more than once, all paths go from $\boldsymbol{s}$ to $\boldsymbol{t}$

Each edge has capacity 1, so it's 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.

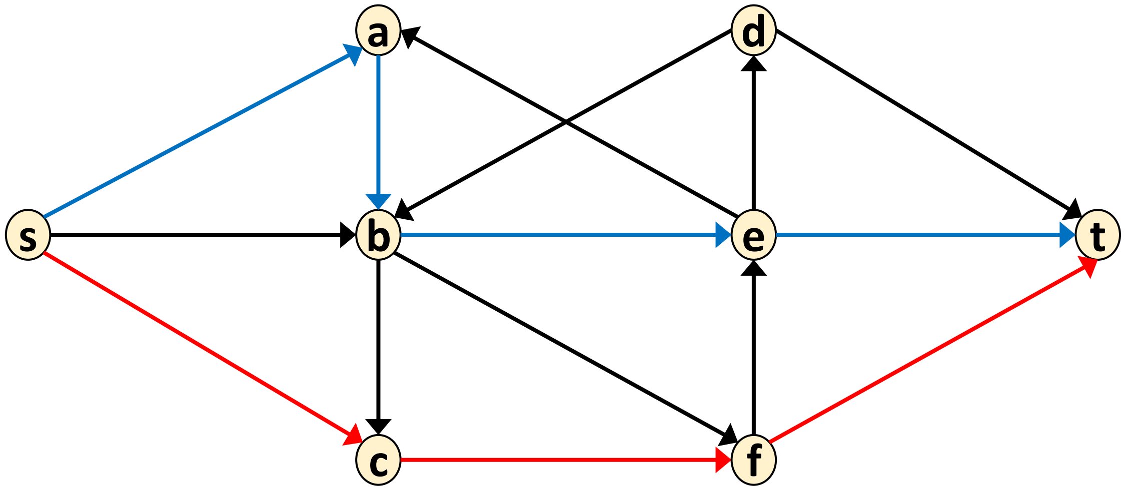
Add 1 unit of flow along each path. Since no path uses the same edge twice, capacity constraint is satisfied. Because the indegree matches the outdegree for each node (except s' and t'), the flow constraint is satisfied.

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.

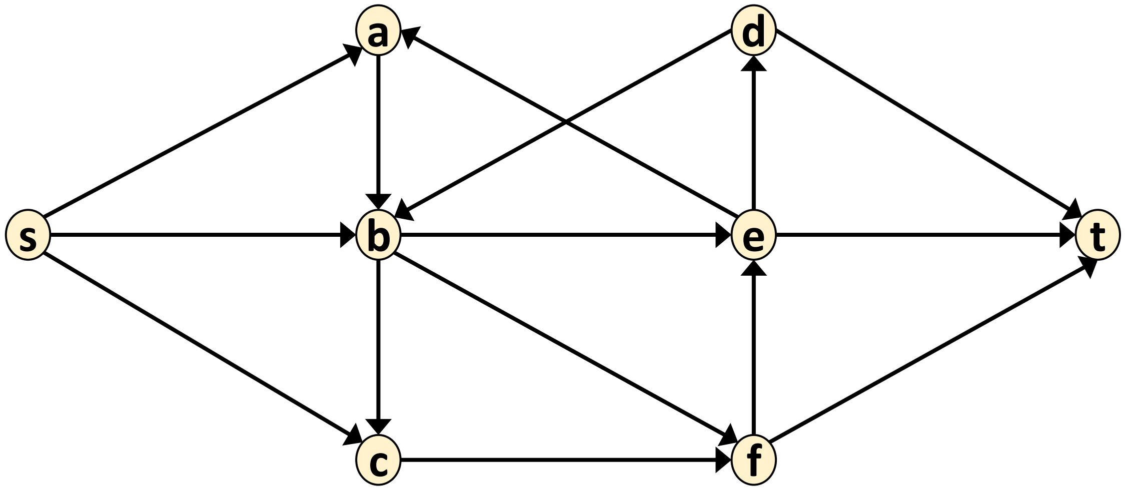


Vertex-Disjoint Paths

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

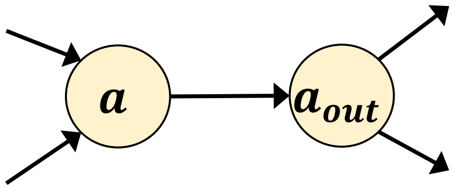
Vertex disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.

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



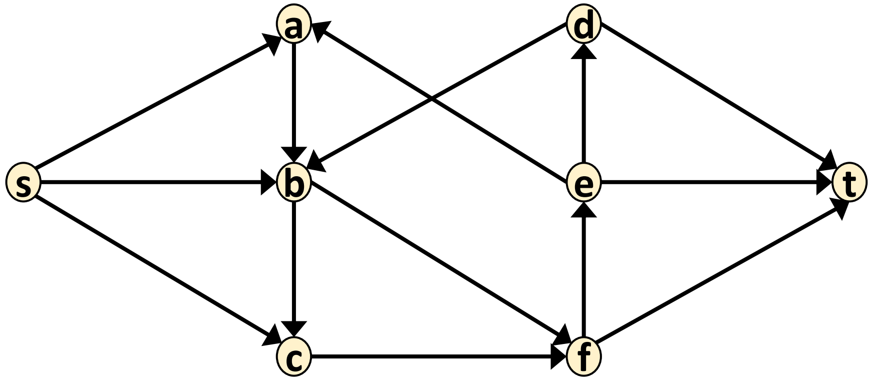
Observation: Every vertex-disjoint path is also edge-disjoint.
(Two paths which share an edge also share that edge's endpoints)

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



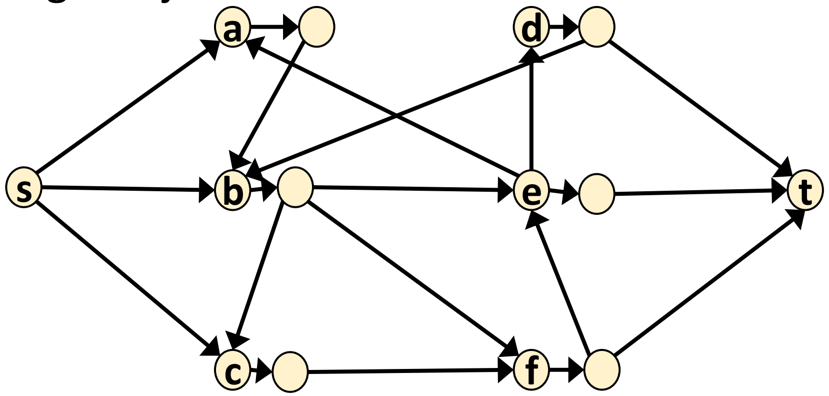
Vertex Disjoint Paths Reduction to Edge-Disjoint Paths

Vertex Disjoint Paths



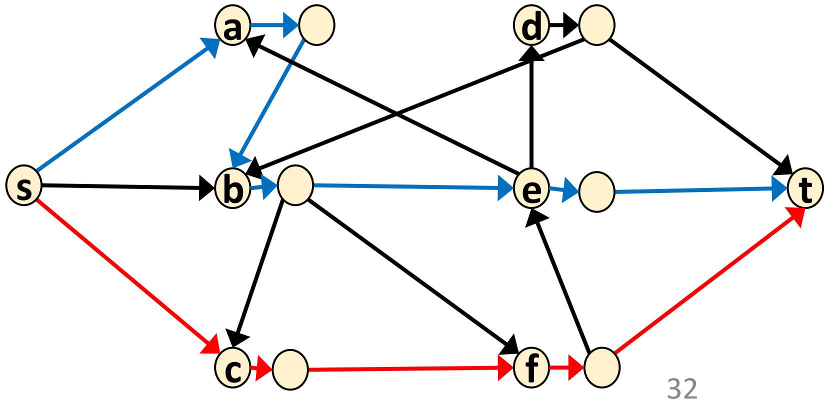
Split each node into a “in” and “out” node. Add an edge from in to out

Edge Disjoint Paths



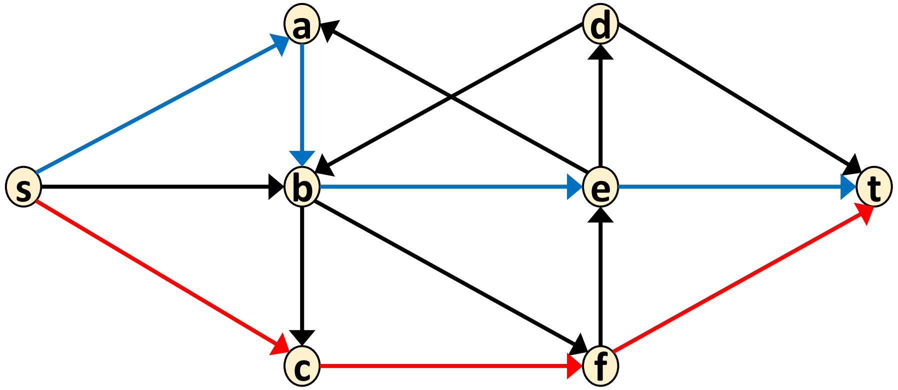
Reduction to Max Flow

Maximal set of Edge-disjoint paths



Recombine the split nodes

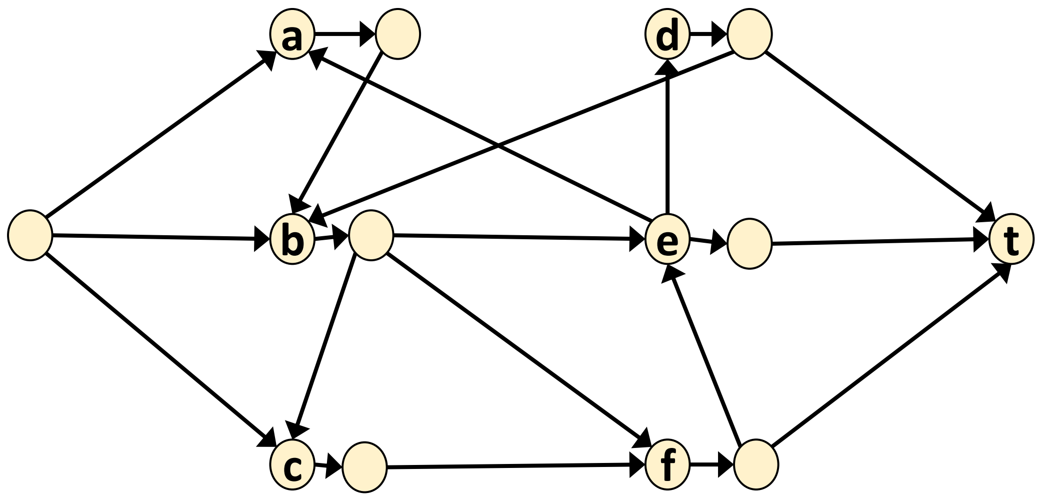
Reduction



Vertex-Disjoint Paths Running time

Reduction 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: $O(n + m)$

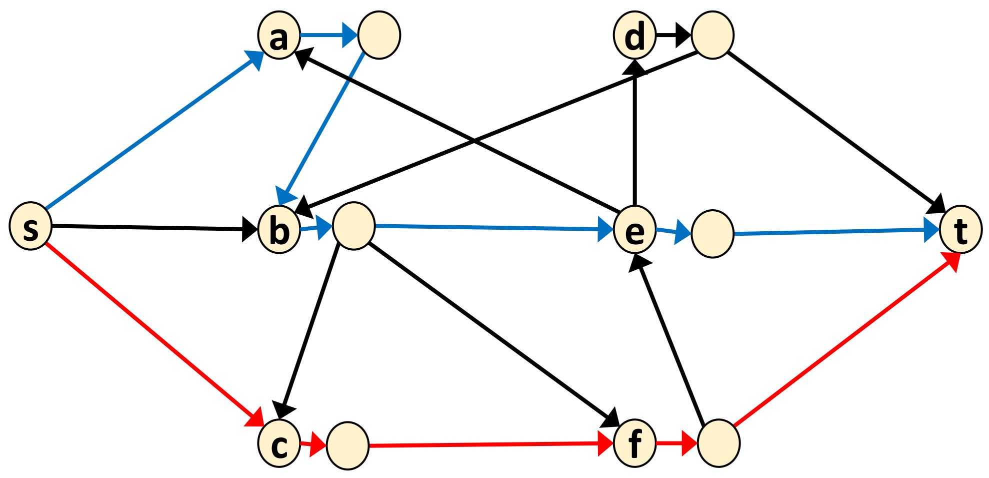
Computing edge disjoint paths: $O(nm^2)$

Overall: $O(nm^2)$

Vertex-Disjoint Paths

Reduction 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



Theorem: # vertex-disjoint paths = # edge-disjoint paths

Valid set of edge-disjoint paths ⇒ Valid set of vertex-disjoint paths:

Need to show: if no edge is used more than once then no vertex is used more than once

Any path that passes through a node v must use the edge (v, v_{out}), so if no edge is used more than once, then that includes these new edges we added, so no vertex can be used more than once either.

Valid set of vertex-disjoint paths ⇒ Valid set of edge-disjoint paths :

Need to show: Any set of vertex-disjoint paths could be a set of edge-disjoint paths of the same size

All vertex-disjoint paths are edge disjoint

Final reminders

HW7 due Wednesday 11/26 @ 11:59pm

Quiz 2 on Friday 11/21 in class

Practice Quiz is available

Review in Wednesday’s class

I have OH now-12:30pm:

- Meet at front of classroom, we’ll walk over together
- CSE (Allen) 434 if you’re coming later

Glenn has online OH 12–1pm