CSE 417 Autumn 2025

Lecture 22: Max Flow Applications

Nathan Brunelle

Origins of Max Flow and Min Cut Problems

Max Flow problem formulation:

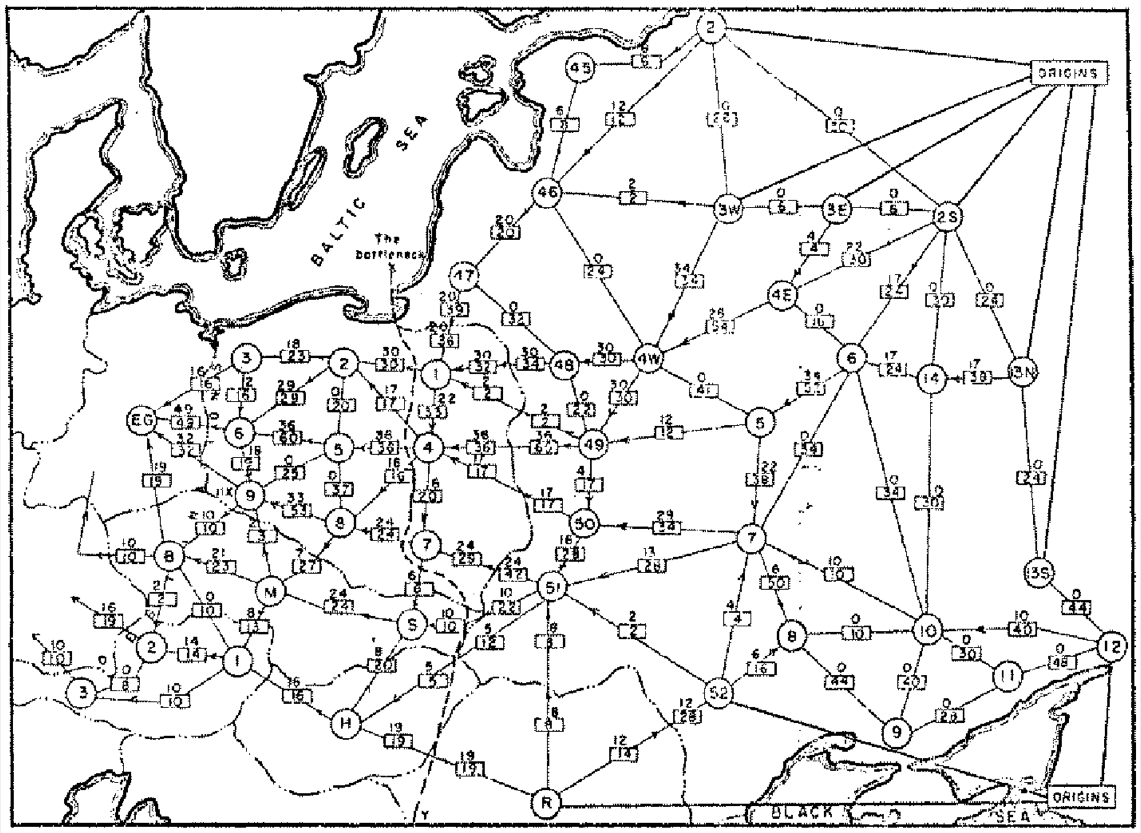
[Tolstoy 1930] Rail transportation planning for the Soviet Union

Min Cut problem formulation:

Cold War: US military planners want to find a way to cripple Soviet supply routes

[Harris 1954] Secret RAND corp report for US Air Force

[Ford-Fulkerson 1955] Problems are equivalent



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

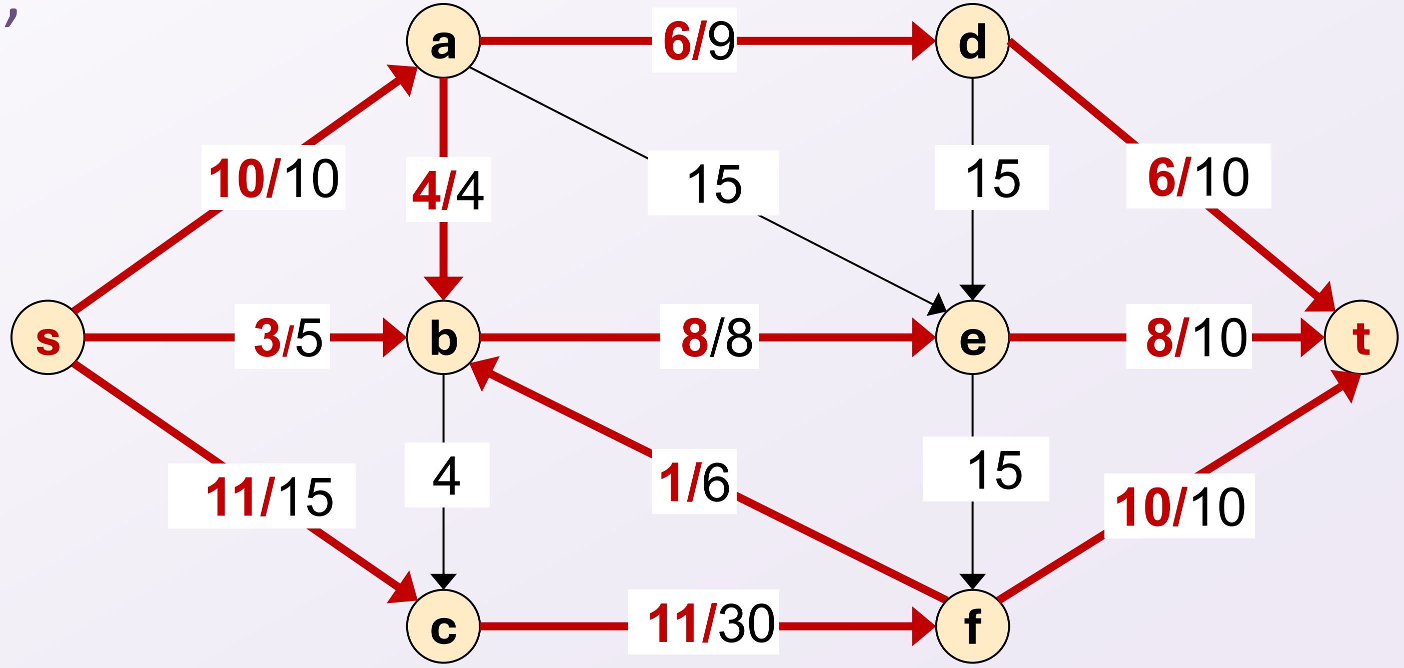
Flow Graph

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

- For each $e \in E$: $0 \le\ f(e) \le\ c(e)$ [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,

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



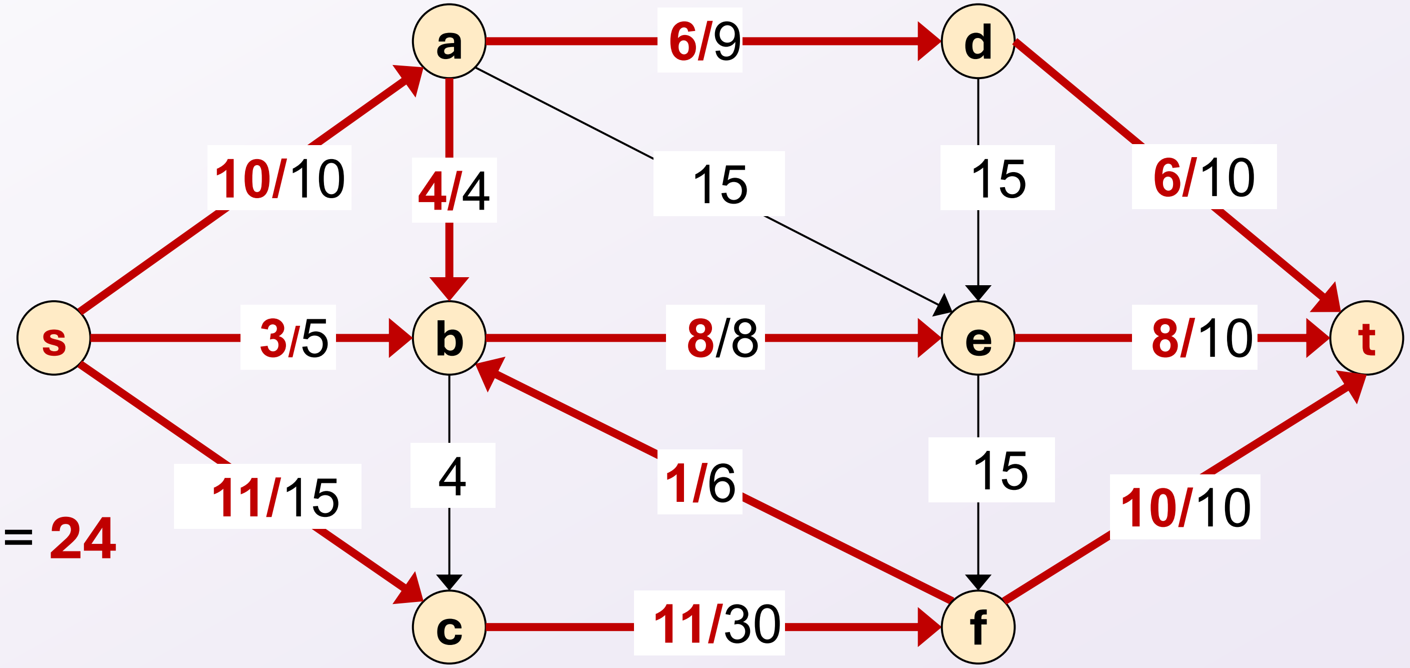
Only show non-zero values of f

value = **24**

Maximum Flow Problem

Given: a flow network

Find: an s-t flow of maximum value



value = **24**

Residual Graphs

An alternative way to represent a flow network

Represents the net available flow between two nodes

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

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

Residual edges of two kinds:

Forward: $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$

- Amount of extra flow we can add along e

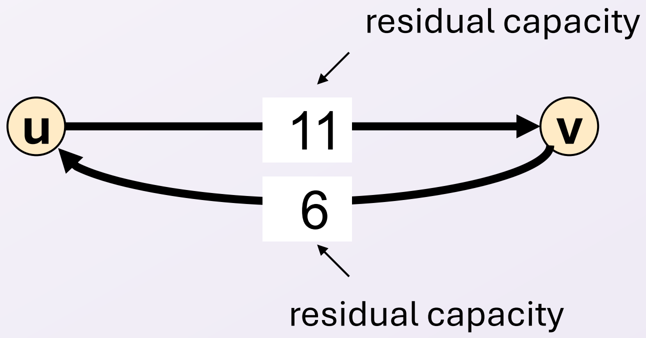
Backward: $e^{\mathrm{R}} = (v, u)$ with capacity $c_f(e) = f(e)$

- Amount we can reduce/undo flow along e

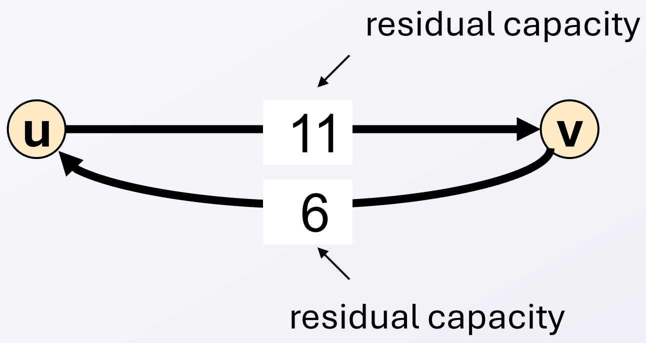
Residual graph: $G_f = (V, E_f)$.

Residual edges with residual capacity $c_f(e) > 0$.

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



Residual Graphs and Augmenting Paths



Residual edges of two kinds:

Forward: $e = (u, v)$ with capacity $c_f(e) = c(e) - f(e)$

- Amount of extra flow we can add along e

Backward: $e^{\mathrm{R}} = (v, u)$ with capacity $c_f(e) = f(e)$

- Amount we can reduce/undo flow along e

Residual graph: $G_f = (V, E_f)$.

Residual edges with residual capacity $c_f(e) > 0$.

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

Augmenting Path: Any s-t path P in G_f. Let bottleneck$(P)= \min_{e \in P} c_f(e)$.

Ford-Fulkerson idea: Repeat “find an augmenting path P and increase flow by bottleneck(P)” until none left.

6

Fork Fulkerson Algorithm

```
FordFulkerson(G, s, t, c){
    for each e ∈ E{
        set f(e) = 0
    }
    calculate residual graph G_f
    while G_f has an s − t path P{
        augment(f, c, P)
        update G_f
    }
    return f
}
```

```
augment(f, c, P){
    b = bottleneck(P)
    for each e ∈ P{
        f(e) += b
        f(e^R) −= b
    }
    return f
}
```

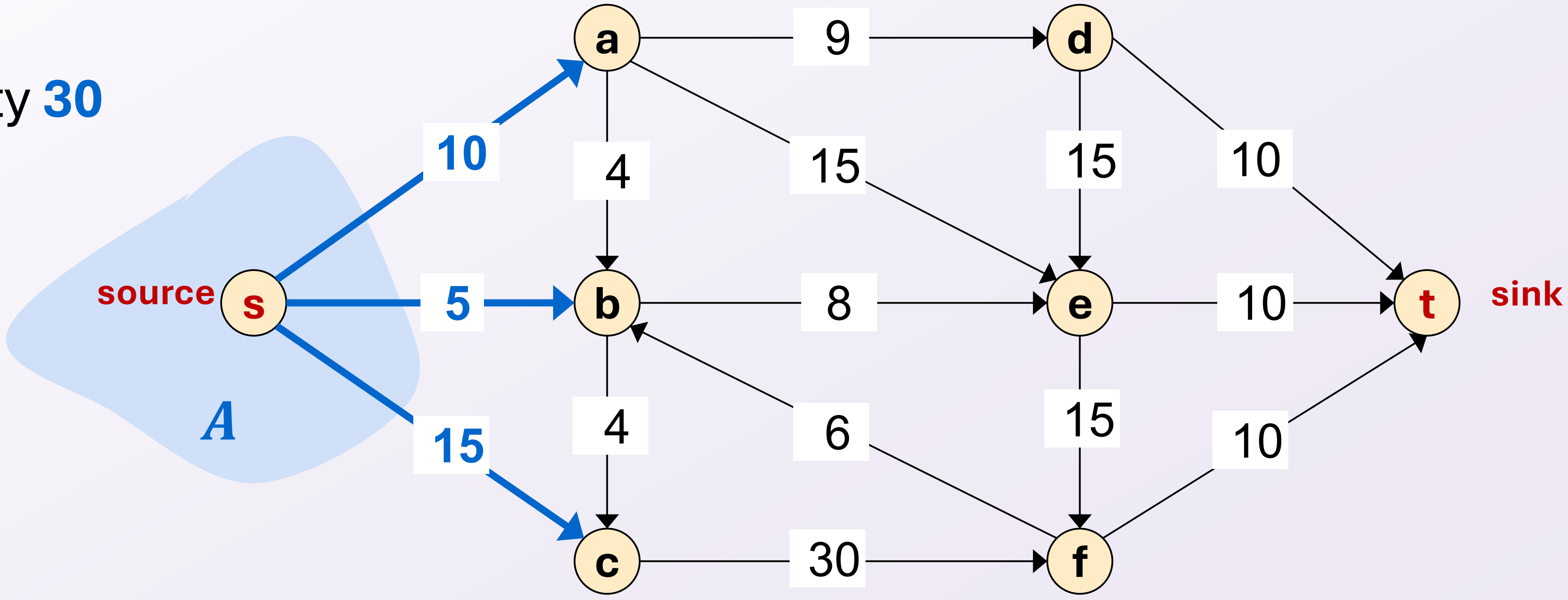
Cuts

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

The capacity of cut (A, B) is

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

capacity **30**



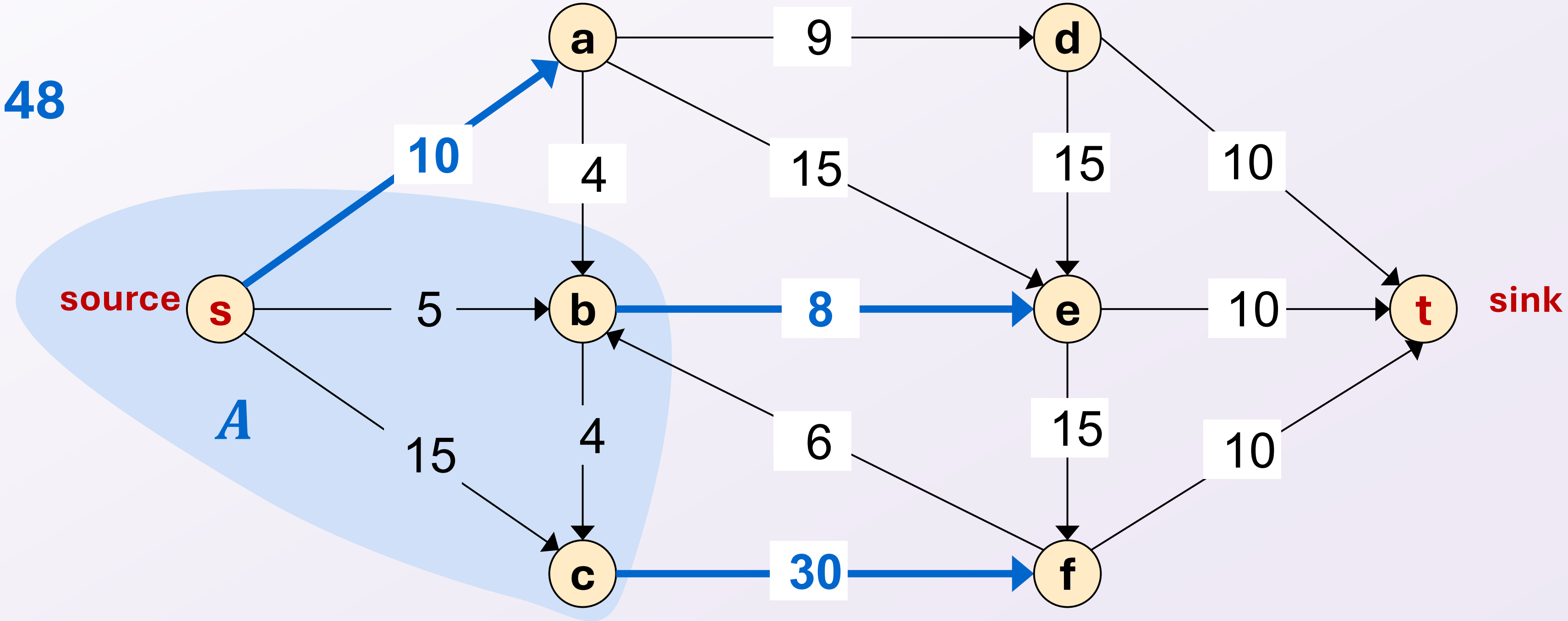
Cuts

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

The capacity of cut (A, B) is

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

capacity **48**



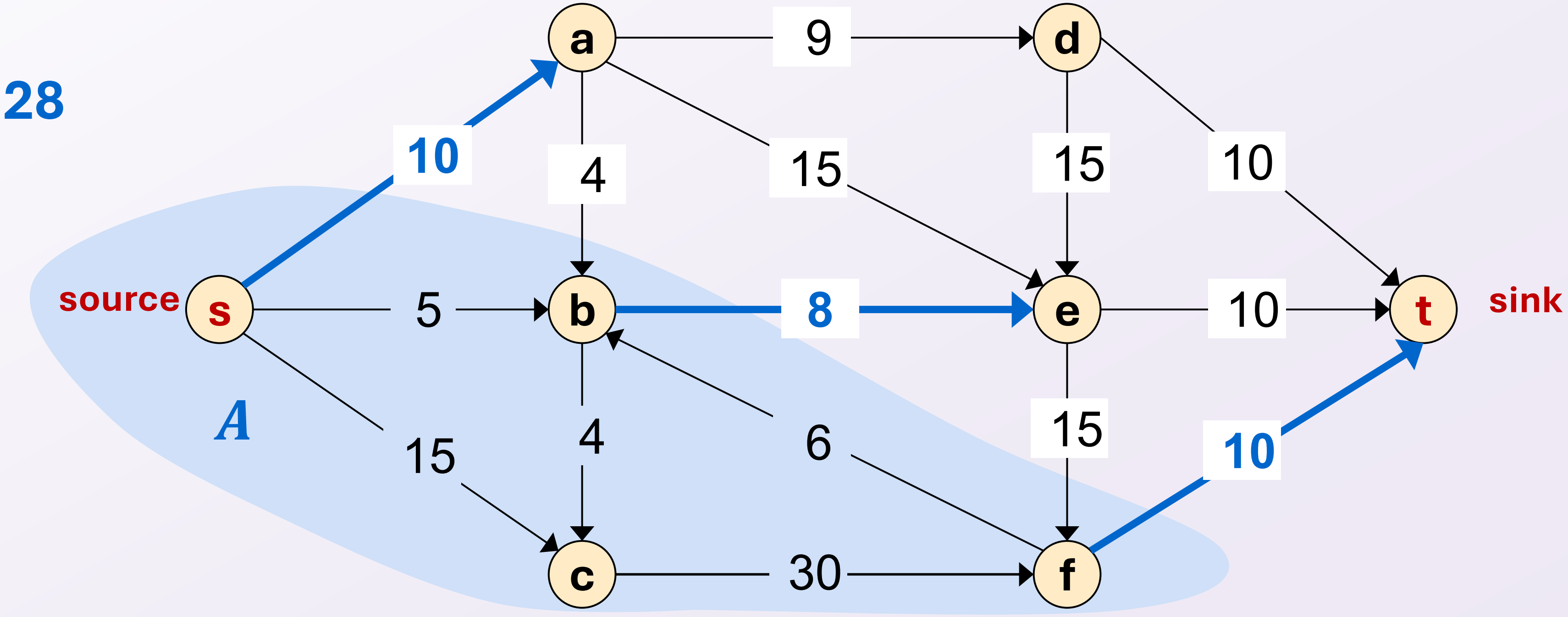
Minimum Cut Problem

Minimum s-t cut problem:

Given: a flow network

Find: an s-t cut of minimum capacity

capacity **28**



Flows and Cuts

Let $\boldsymbol{f}$ be any $\boldsymbol{s}$-$\boldsymbol{t}$ flow and $(\boldsymbol{A}, \boldsymbol{B})$ be any $\boldsymbol{s}$-$\boldsymbol{t}$ cut:

Flow Value Lemma: The net value of the flow sent across $(\boldsymbol{A}, \boldsymbol{B})$ equals $\boldsymbol{v}(\boldsymbol{f})$.

Intuition: All flow coming from s must eventually reach t, and so must cross that cut

Weak Duality: The value of the flow is at most the capacity of the cut; i.e., $\boldsymbol{v}(\boldsymbol{f}) \leq \boldsymbol{c}(\boldsymbol{A}, \boldsymbol{B})$.

Intuition: Since all flow must cross any cut, any cut's capacity is an upper bound on the flow

Corollary: If $\boldsymbol{v}(\boldsymbol{f}) = \boldsymbol{c}(\boldsymbol{A}, \boldsymbol{B})$ then $\boldsymbol{f}$ is a maximum flow and $(\boldsymbol{A}, \boldsymbol{B})$ is a minimum cut.

Intuition: If we find a cut whose capacity matches the flow, we can't push more flow through that cut because it's already at capacity. We additionally can't find a smaller cut, since that flow was achievable.

Flows and Cuts (Simplified)

1. The net flow crossing any cut equals the flow value.
    - Why? Everything must cross the cut eventually
2. The capacity of any cut therefore is an upper bound on the max flow
    - Why? No flow can exceed that capacity due to statement 1
3. If we found a flow whose value matches the capacity of some cut, then we know that the flow must be maximum, and the cut must be minimum
    - Why? If there was a smaller cut or larger flow, we've broken statement 2

What we need for correctness:

When Ford-Fulkerson terminates, there exists a cut whose capacity matches the current flow value.

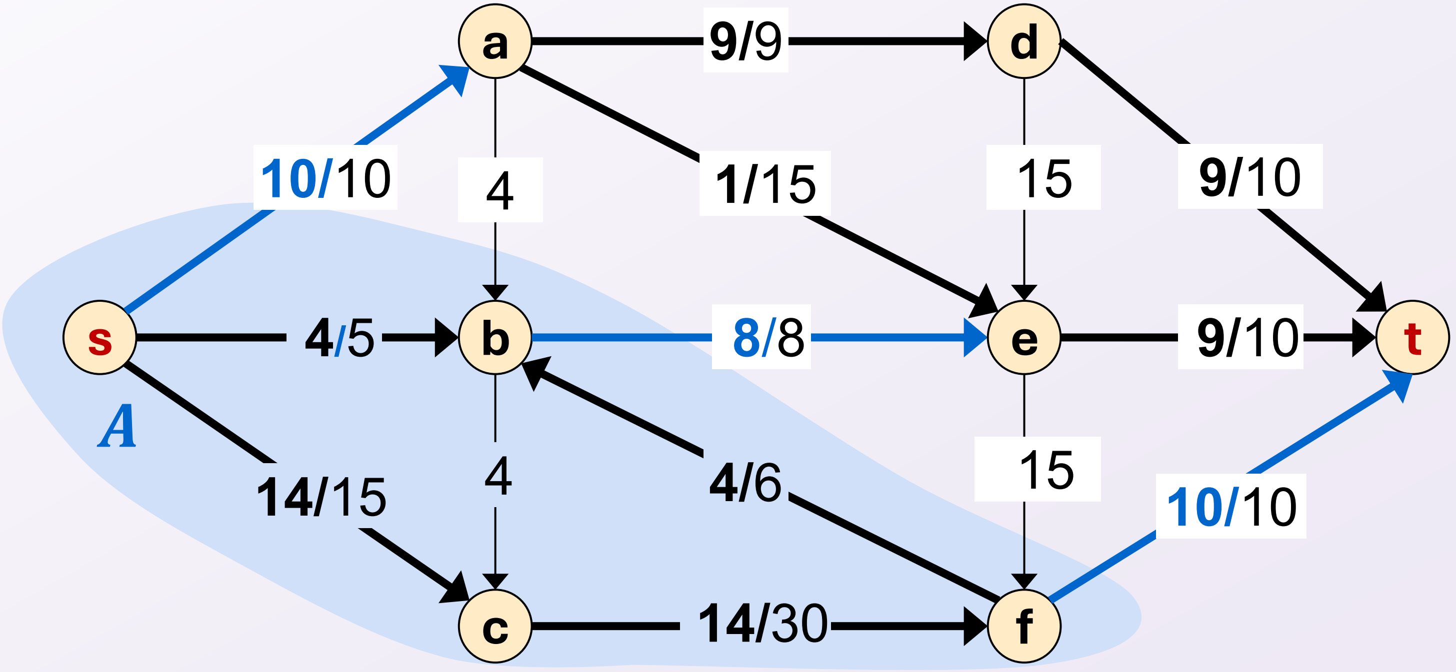
Certificate of Optimality

Let $\boldsymbol{f}$ be any $\boldsymbol{s}$-$\boldsymbol{t}$ flow and $(\boldsymbol{A}, \boldsymbol{B})$ be any $\boldsymbol{s}$-$\boldsymbol{t}$ cut.

If $\boldsymbol{v}(\boldsymbol{f}) = \boldsymbol{c}(\boldsymbol{A}, \boldsymbol{B})$ then $\boldsymbol{f}$ is a max flow and $(\boldsymbol{A}, \boldsymbol{B})$ is a min cut.

Value of flow = **28**

Capacity of cut = **28**



Both are optimal!
Each "certified" correctness of the other!

13

Identifying the cut

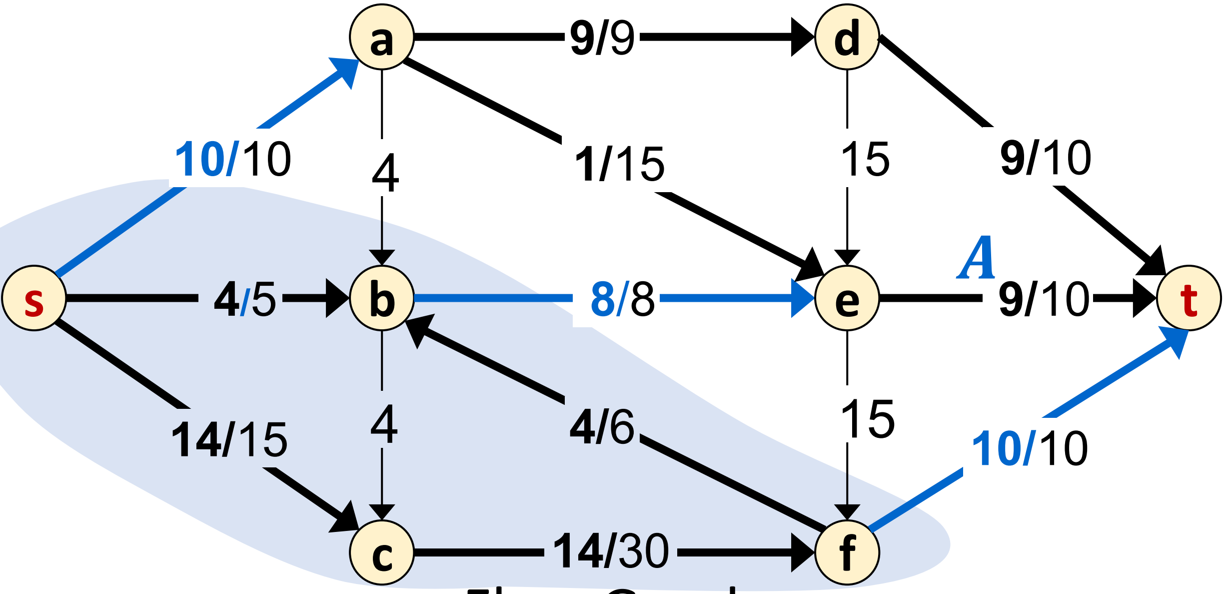
To Show: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

Selecting a cut: Let f be a flow with no augmenting paths.

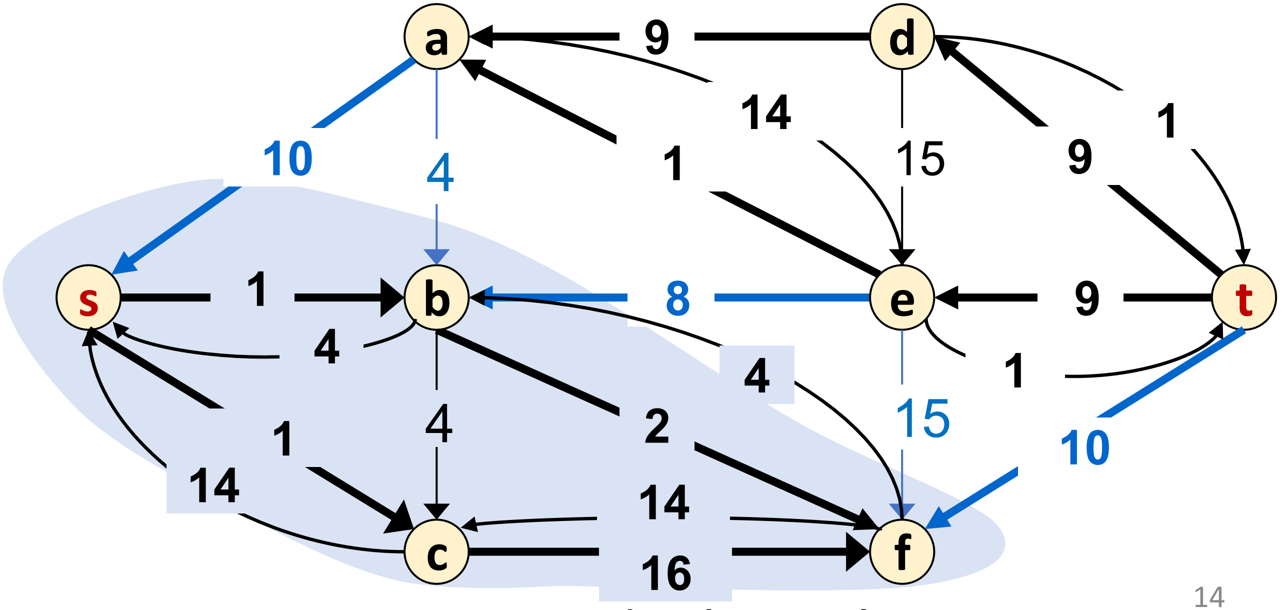
Let A be the set of vertices reachable from s in residual graph G_f.

- By definition of A, $s \in A$.
- Since no augmenting path (s-t path in G_f), $t \notin A$.

Notice:
- all edges out of the cut are saturated (flow=capacity)
- all edges into the cut have no flow



Flow Graph



Residual Graph

Flow Value = Cut Capacity

To Show: If there is no augmenting path w.r.t. f, there is a cut (A, B) s.t. $v(f) = c(A, B)$.

The cut: A is the set of all nodes reachable from s in the residual graph

B is the set of all the other nodes in the graph

Showing Flow value = Cut Capacity:

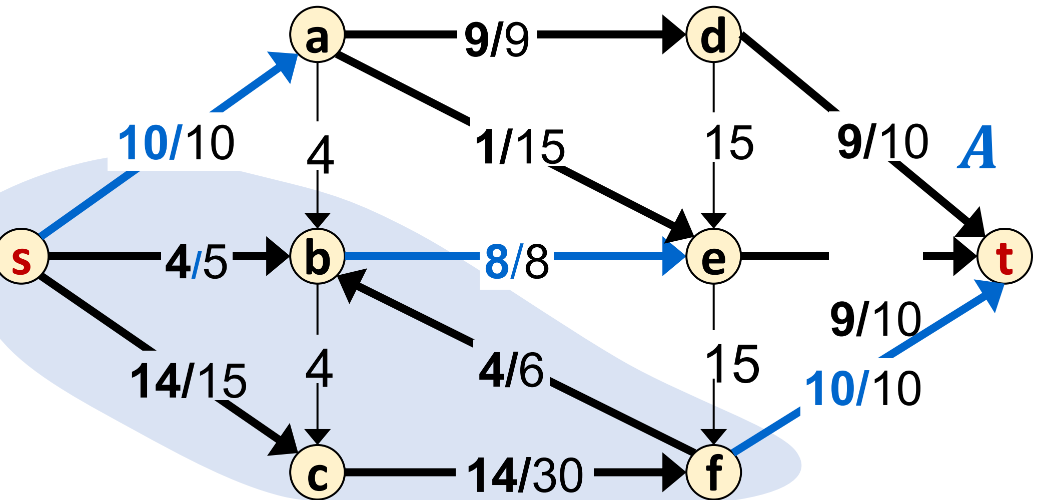
For any edge outgoing from A to B, that edge is saturated (flow = capacity)

Otherwise there would be an edge in the residual graph for the remaining capacity. Contradiction!

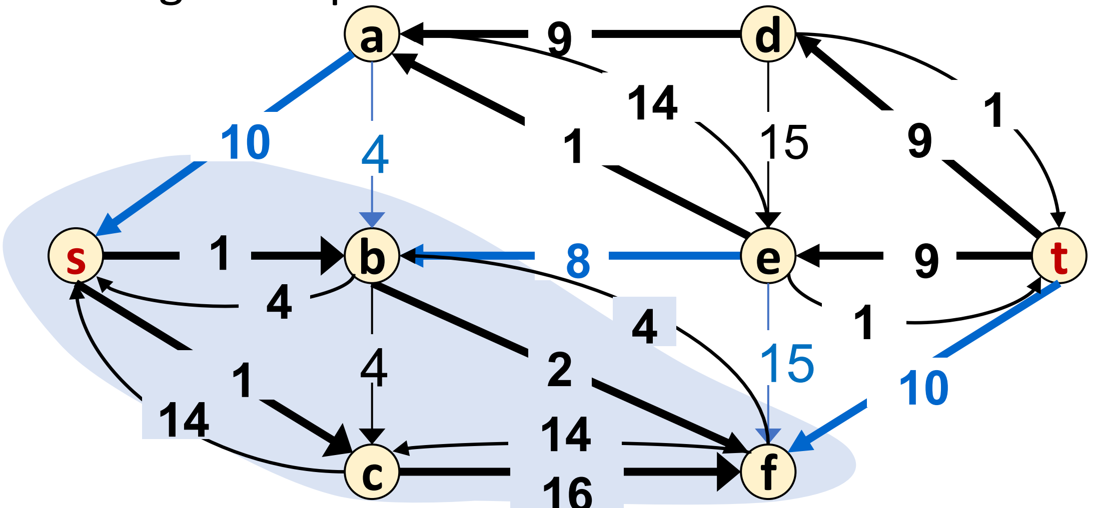
For any edge incoming from B to A, that edge has no flow (flow = 0)

Otherwise there would be an edge in the residual graph to undo the flow. Contradiction!

So summing the flows of all A to B edges is the same as summing the capacities!



Flow Graph



Residual Graph

Flows and Cuts (Complete)

1. The net flow crossing any cut equals the flow value.
    - Why? Everything must cross the cut eventually
2. The capacity of any cut therefore is an upper bound on the max flow
    - Why? No flow can exceed that capacity due to statement 1
3. If we found a flow whose value matches the capacity of some cut, then we know that the flow must be maximum, and the cut must be minimum
    - Why? If there was a smaller cut or larger flow, we've broken statement 2
4. When Ford Fulkerson terminates, there is a cut whose capacity matches the flow
    - Why? Select one side of the cut to be nodes reaching from s in the residual graph, the other side to be the rest of the nodes. That cut's capacity matches the flow value.
    - Thus the cut is minimum, and the flow is maximum!

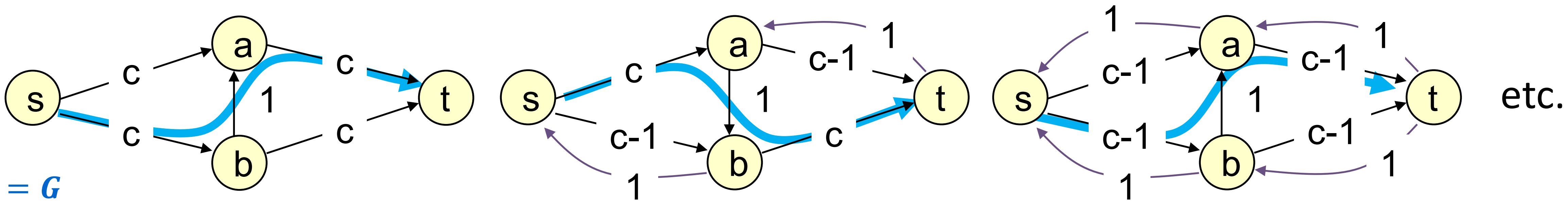
Ford-Fulkerson Running Time

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

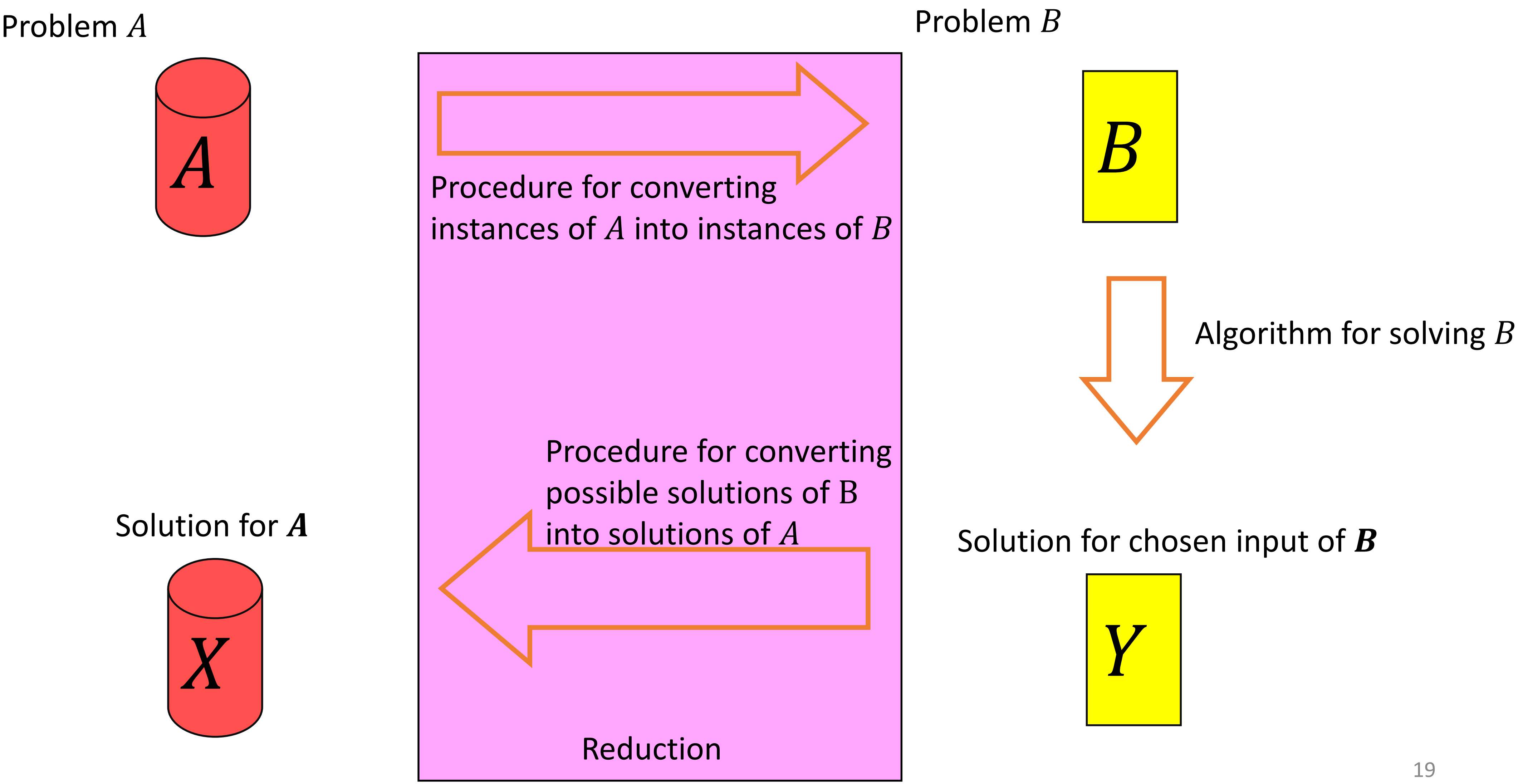


$G_f = G$

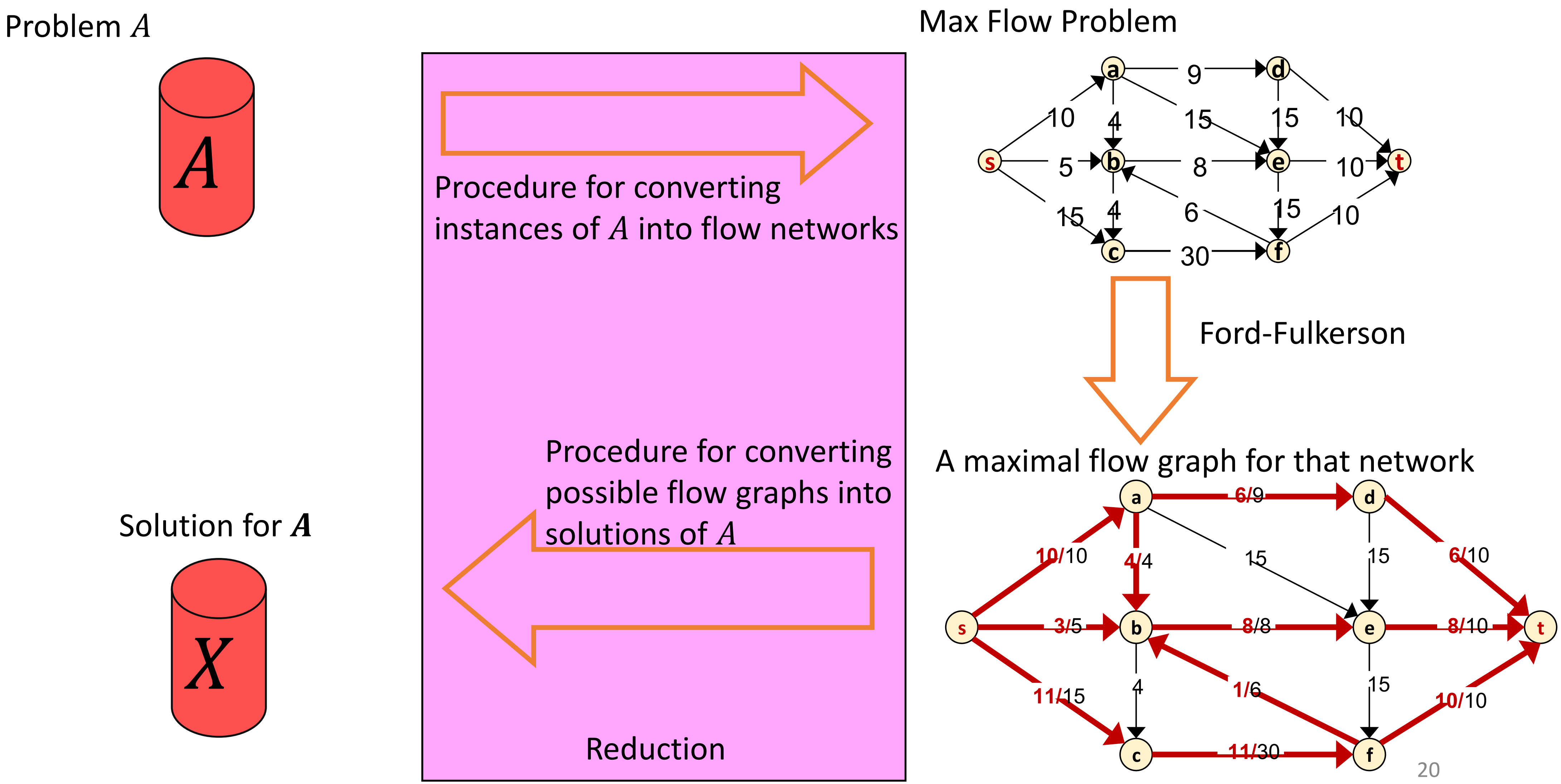
Applications of Max Flow

- 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 $s_1, \dots s_n$, the number of employees needed for each shift $p_1, \dots p_n$, the availability of m employees $e_1, \dots, e_m$, 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

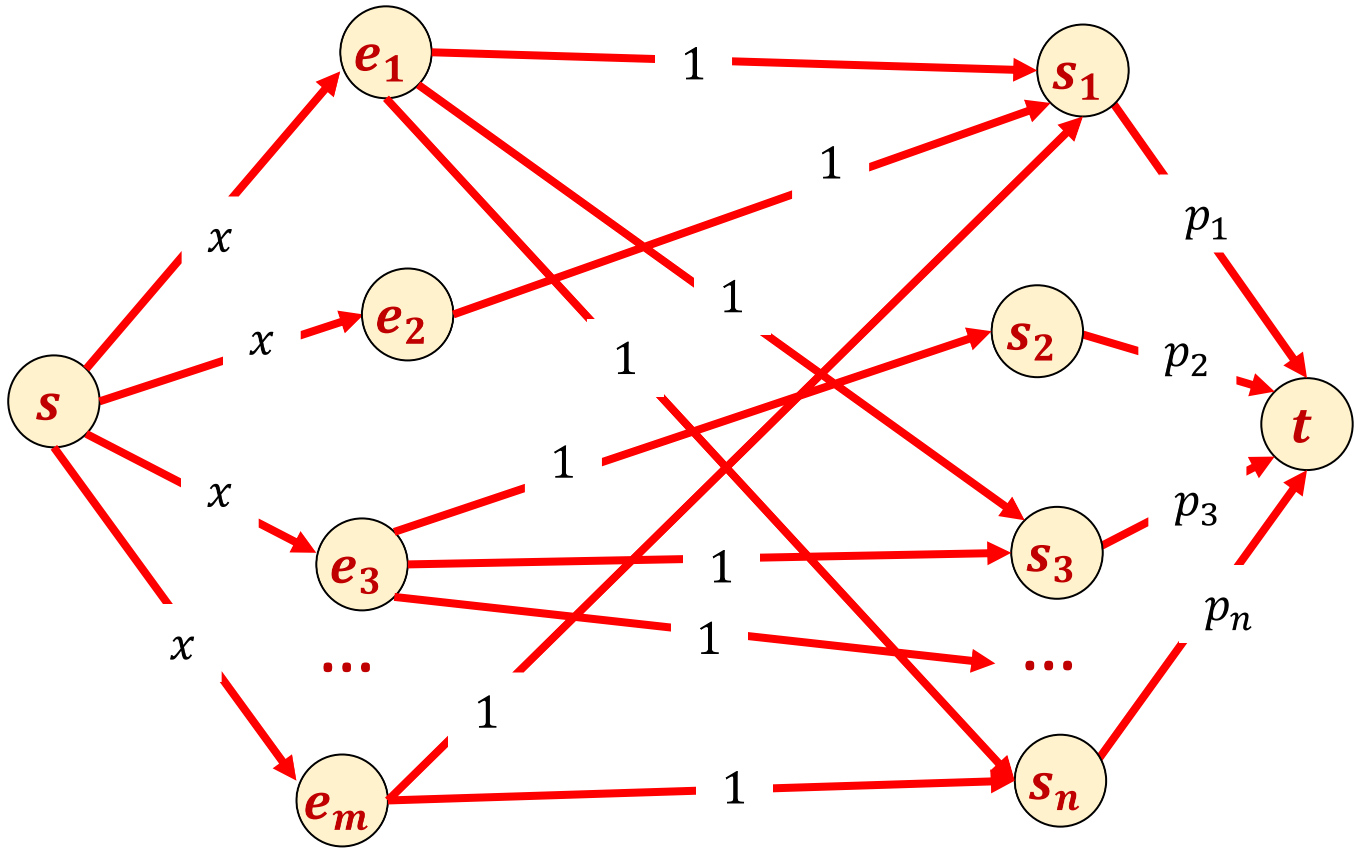
Employees:
1. 6am, 9am, 3pm
2. 6am, 9am, 12pm
3. 6am, 3pm

$x = 2$

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



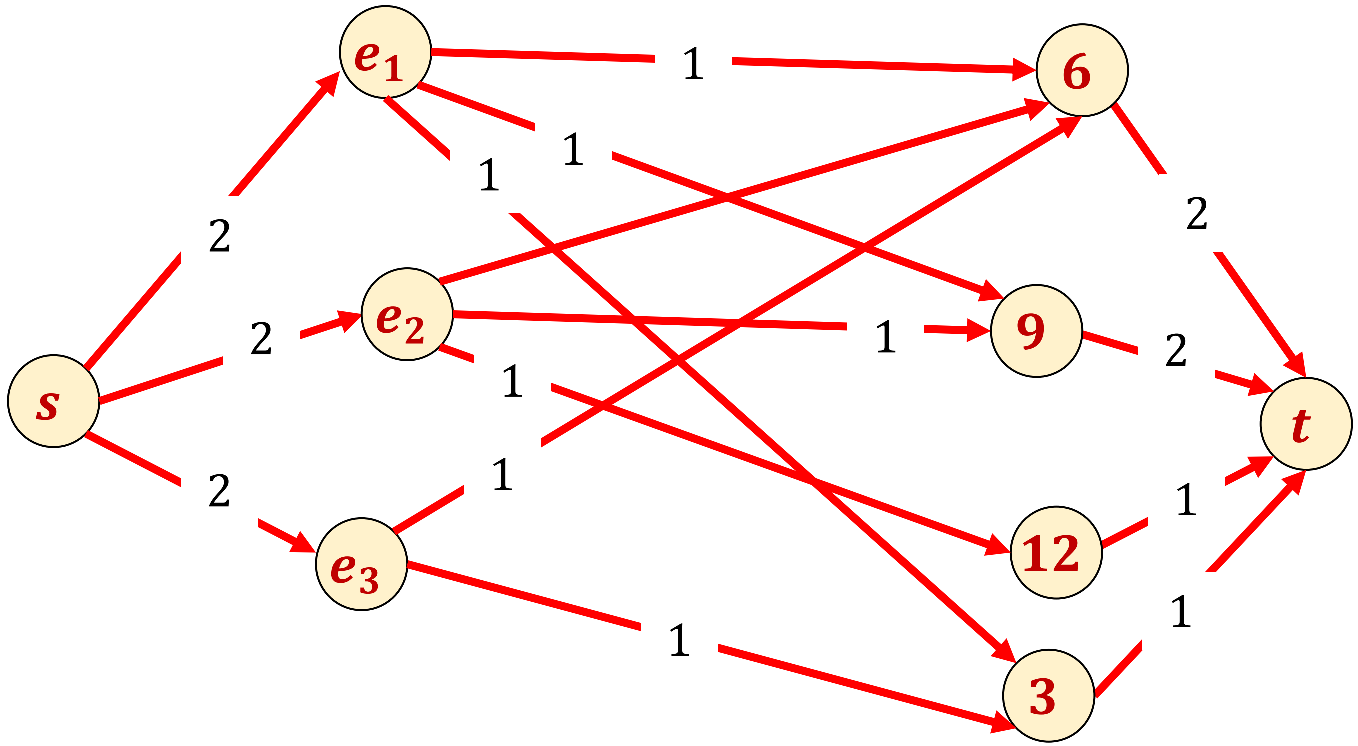
Reducing to Max Flow

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

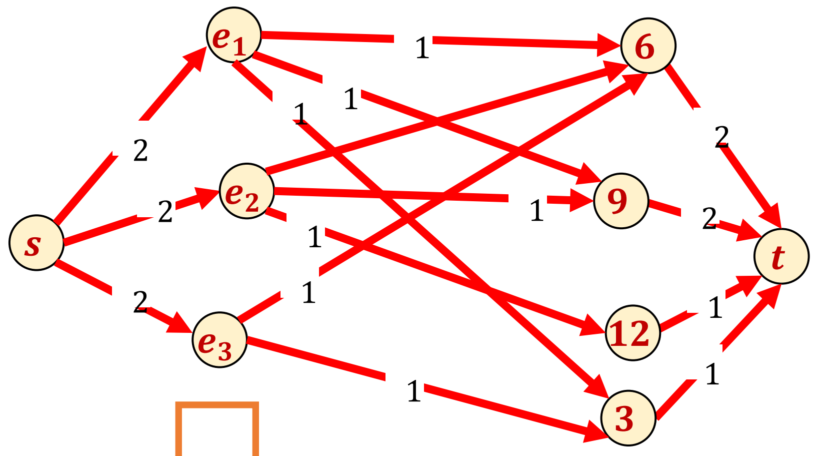
Shifts: 1. 6am, 2 2. 9am, 2 3. 12pm, 1 4. 3pm, 1

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

$x = 2$

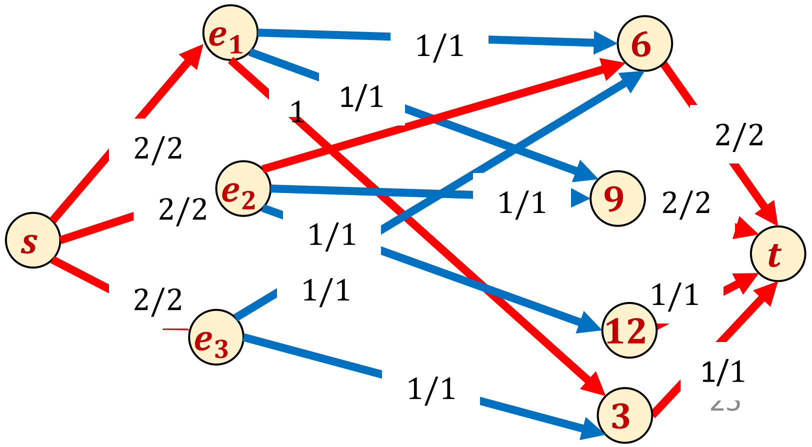
Nodes: One per employee, One per shift, source, sink
Edges: s to employees with capacity x, employees to available shifts with capacity 1, shifts to sink with capacity p_i

Max Flow Problem



Ford-Fulkerson

A maximal flow graph for that network



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

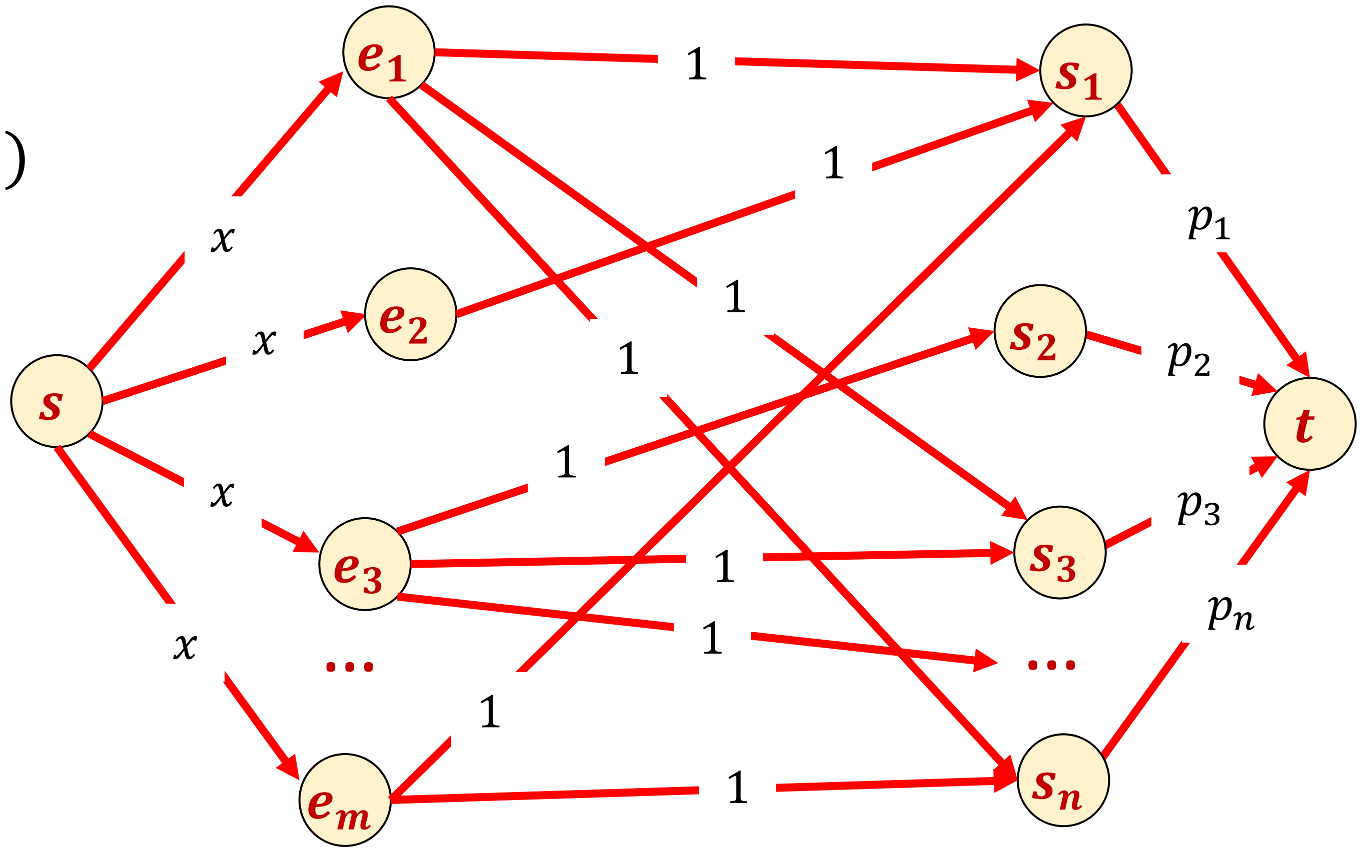
Reduction

Schedule

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

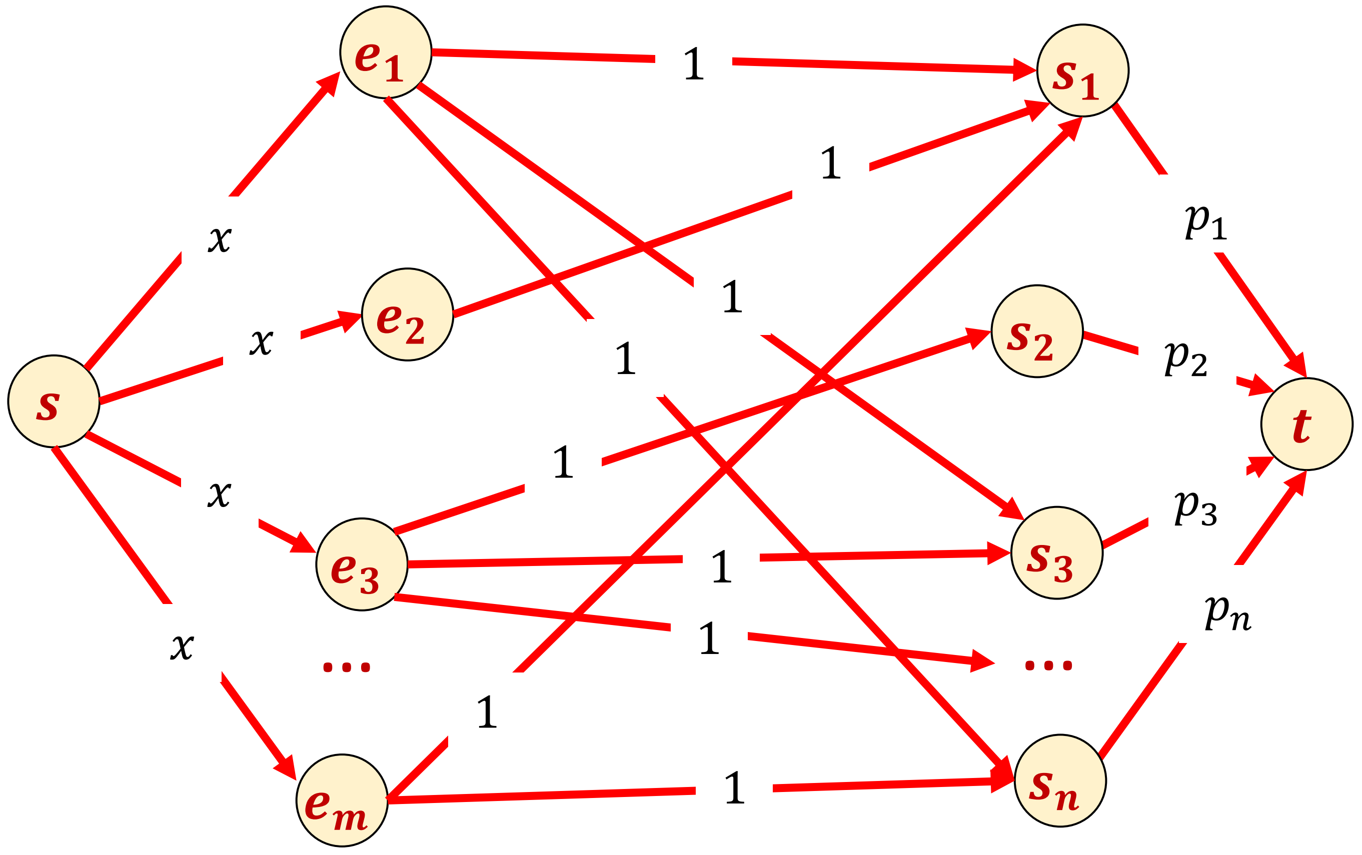
Running Time

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



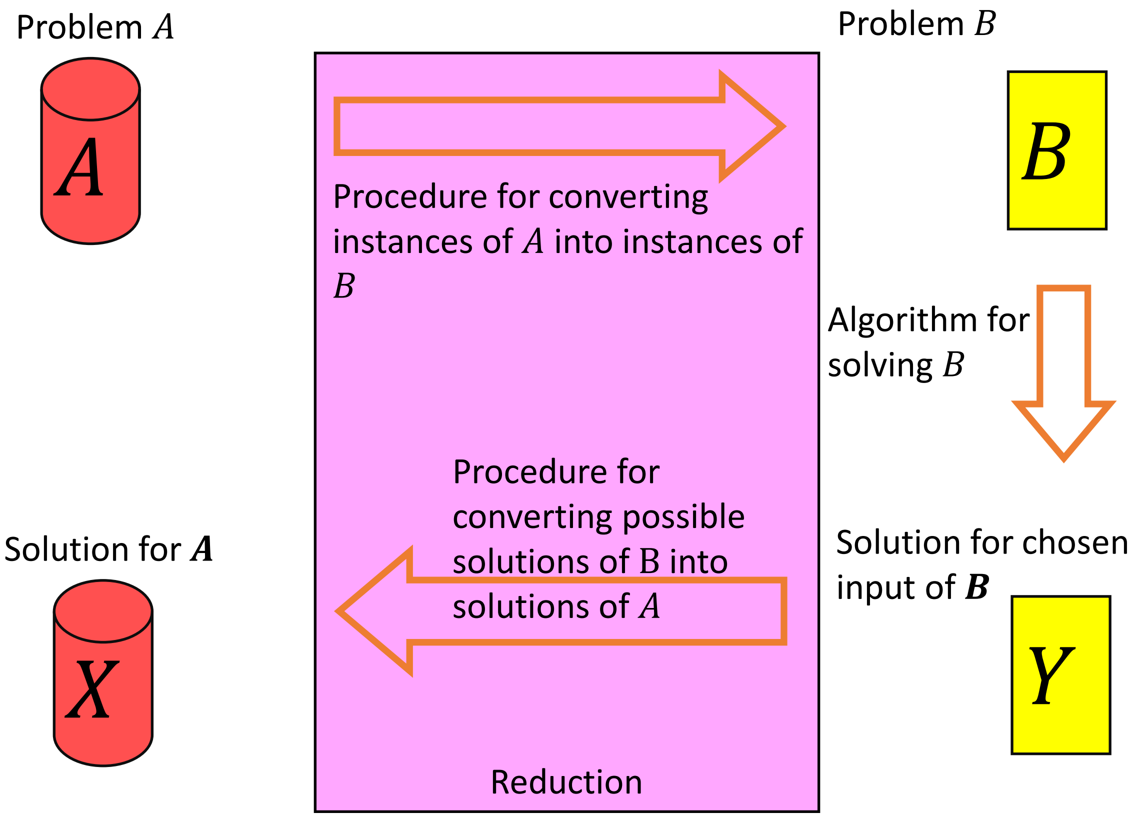
Correctness

- Valid flow $\Rightarrow$ 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 $\Rightarrow$ 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

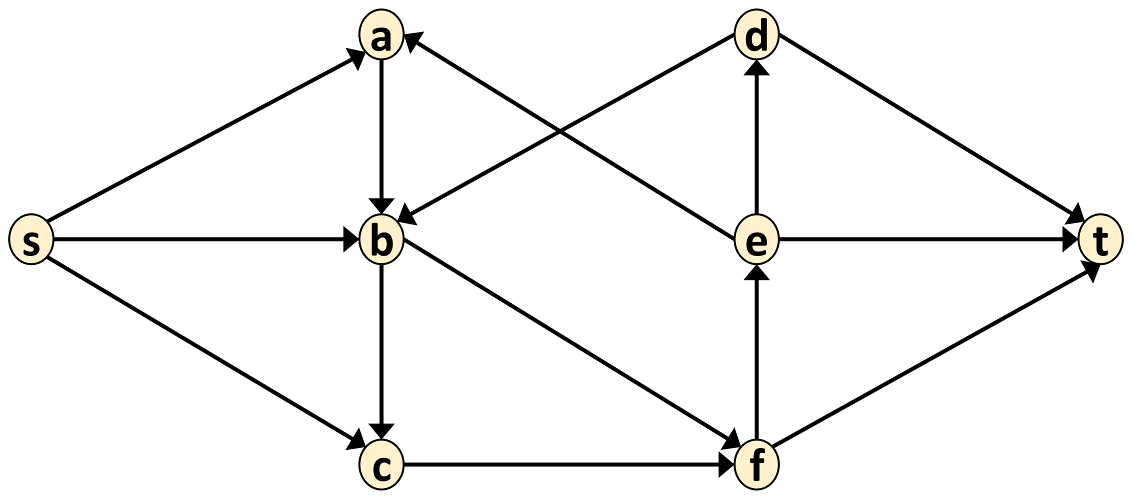
- A valid answer to the chosen problem B input produces a valid answer to the original problem A
  - Our reduction produces a meaningful result
- A valid answer to the original problem A results in a valid answer to the chosen problem B input
  - If there was a better answer for A, then the algorithm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



Edge-Disjoint Paths

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

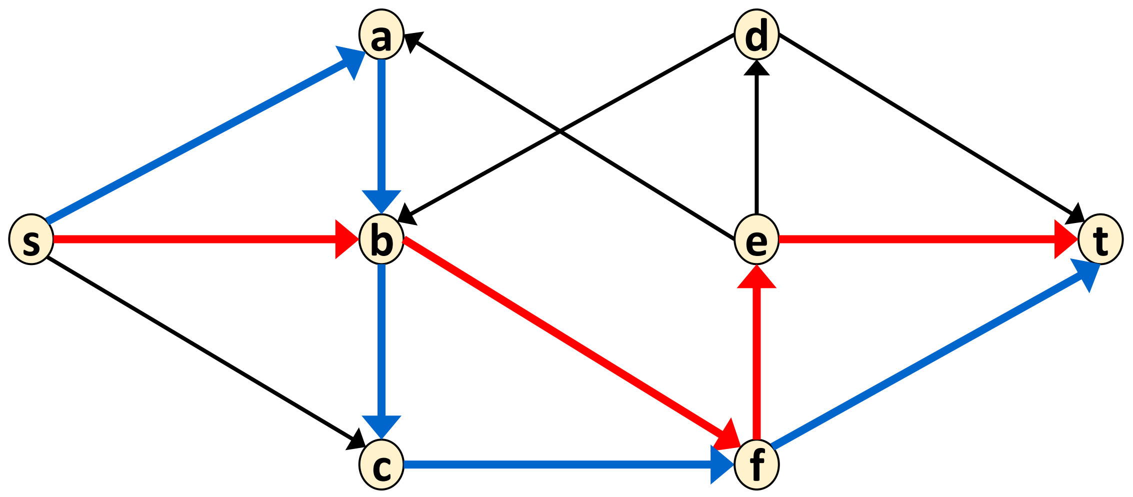
Edge disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.
Find: the maximum # of edge-disjoint simple s-t paths in G.



Edge-Disjoint Paths – Example of size 2

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

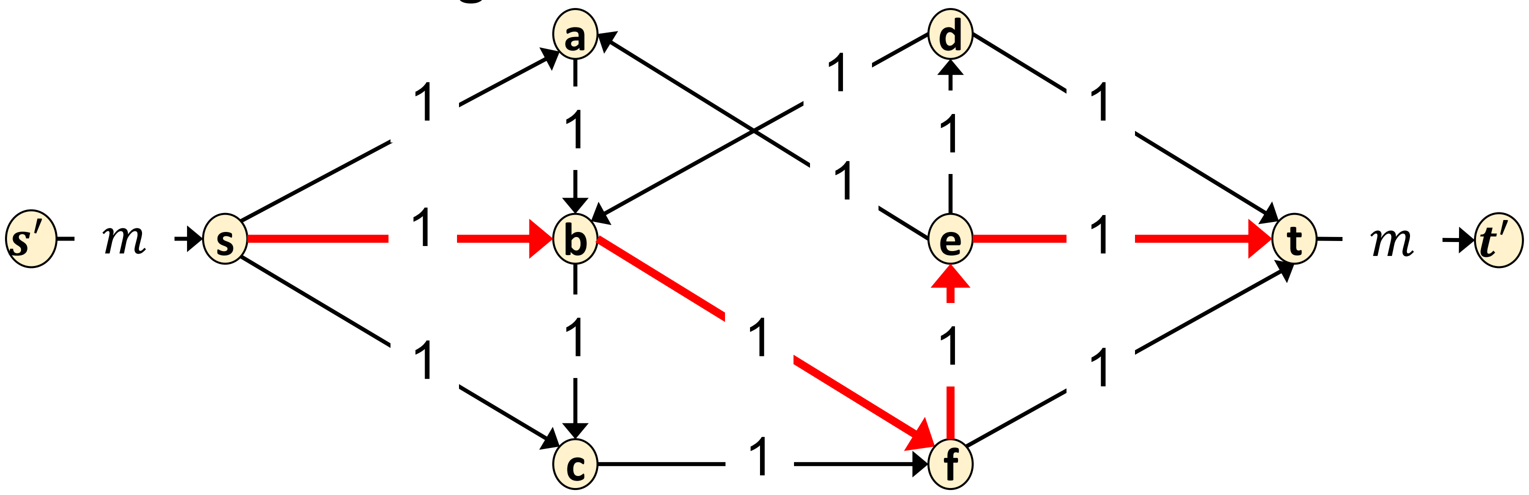
Edge disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.
Find: the maximum # of edge-disjoint simple s-t paths in G.



Edge-Disjoint Paths

MaxFlow for edge-disjoint paths

- Assign capacity **1** to every edge
- Add a source s' and a sink t'
- Connect s' to s and t' to 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:

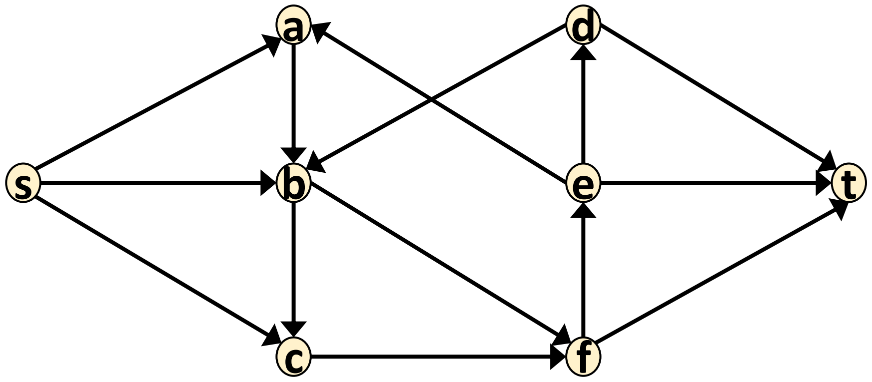
Constructing the flow network: $O(n+m)$

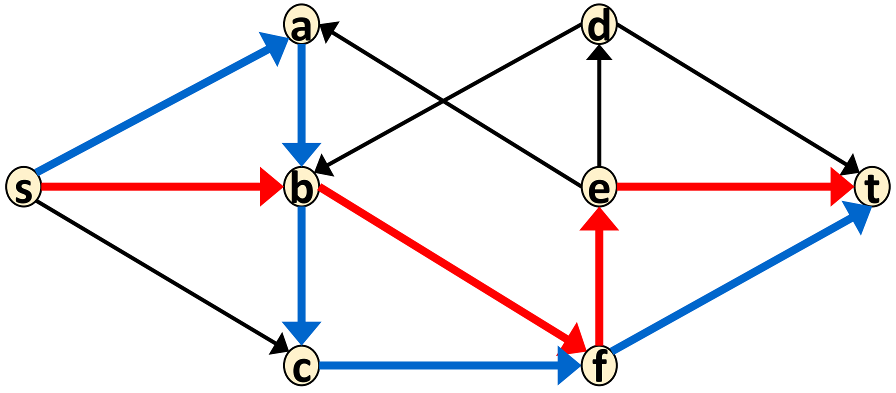
Computing Max Flow: $O(nm^2)$

Overall: $\Theta(nm^2)$

Edge Disjoint Paths Reduction to Max Flow

Edge Disjoint Paths



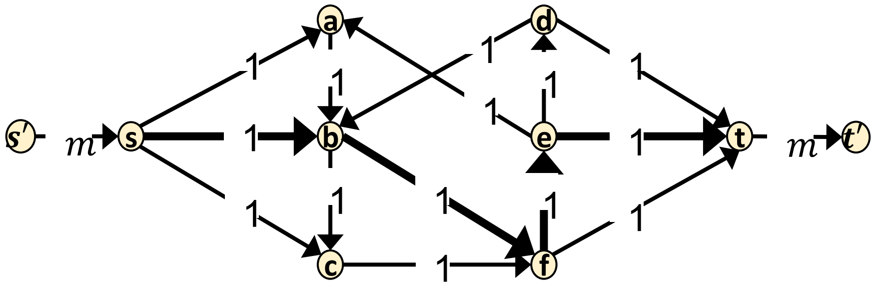


Add a new source and sink, add edge from new source to S with capacity m, edge from t to new sink with capacity m, add capacity 1 to all original edges

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

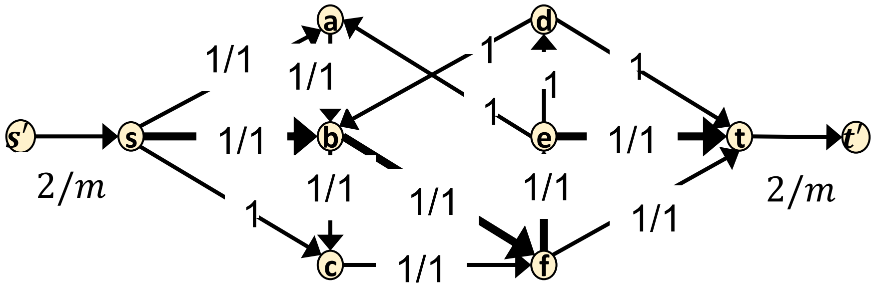
Reduction

Max Flow Problem



Ford-Fulkerson

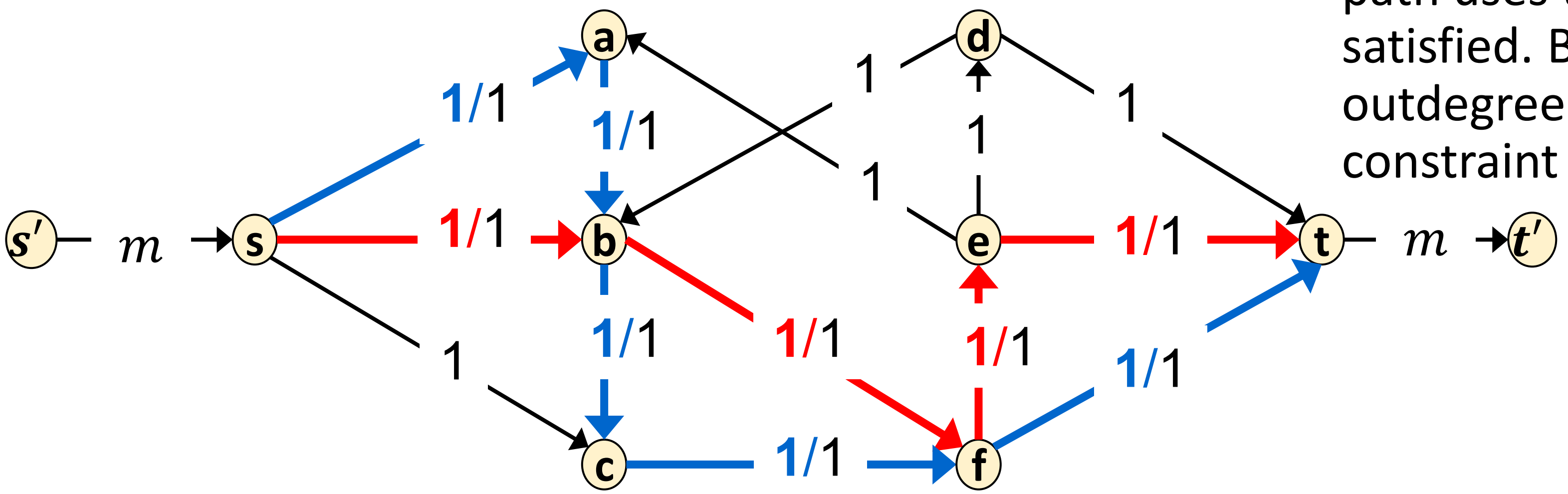
A maximal flow graph for that network



Edge-Disjoint Paths

MaxFlow for edge-disjoint paths

- Assign capacity **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.

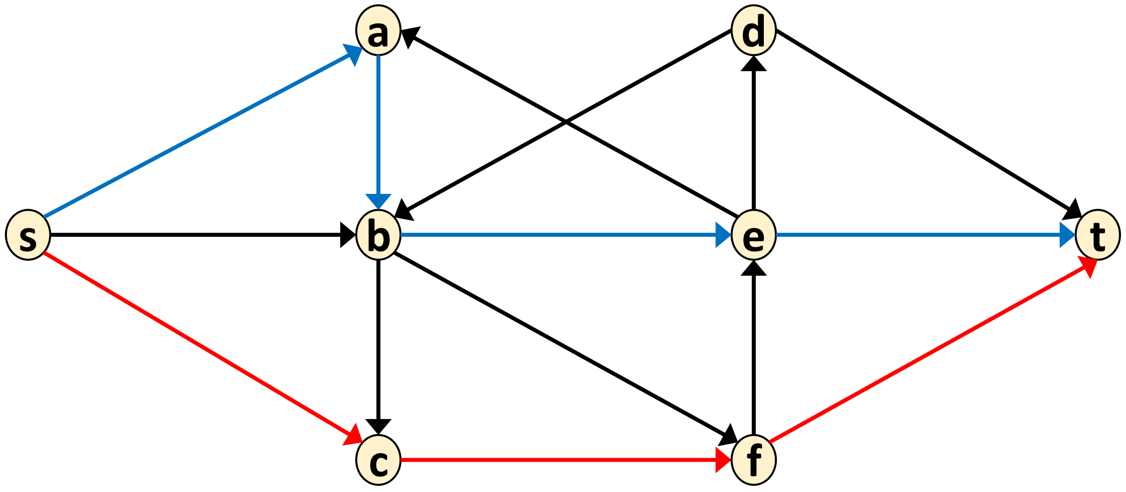
Add 1 unit of flow along each path. Since no path uses the same edge twice, capacity constraint is satisfied. Because the indegree matches the outdegree for each node (except s' and t'), the flow constraint is satisfied.

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.

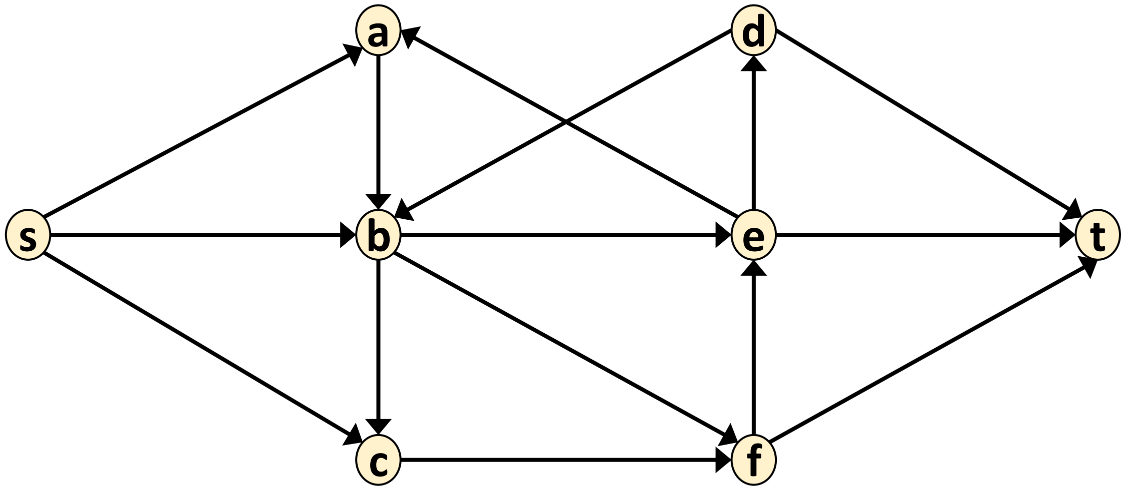


Vertex-Disjoint Paths

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

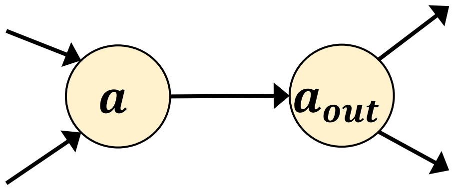
Vertex disjoint path problem: **Given:** a directed graph $G = (V, E)$ and two vertices s and t.

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



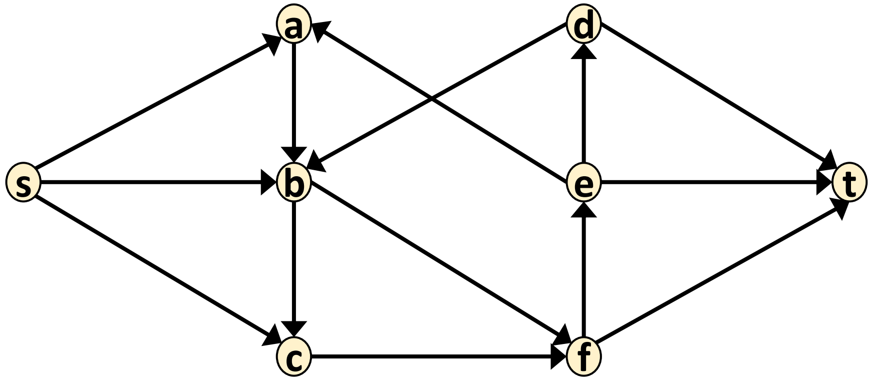
Observation: Every vertex-disjoint path is also edge-disjoint.
(Two paths which share an edge also share that edge's endpoints)

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



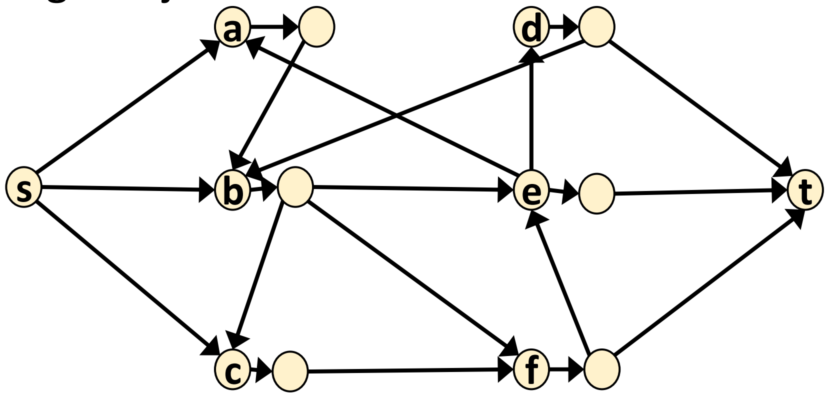
Vertex Disjoint Paths Reduction to Edge-Disjoint Paths

Vertex Disjoint Paths



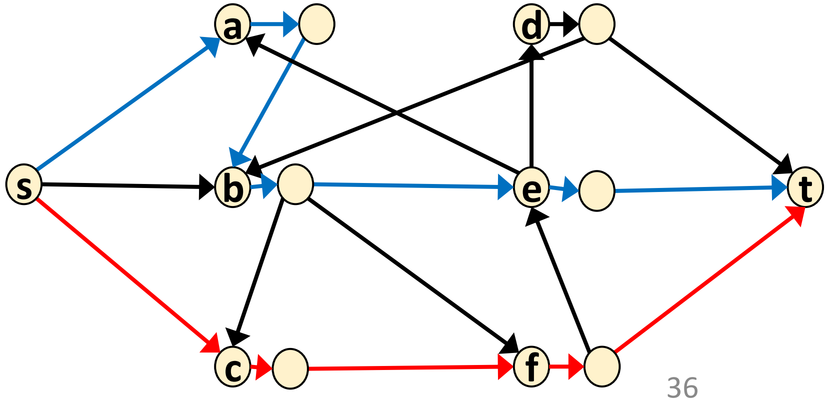
Split each node into a “in” and “out” node. Add an edge from in to out

Edge Disjoint Paths



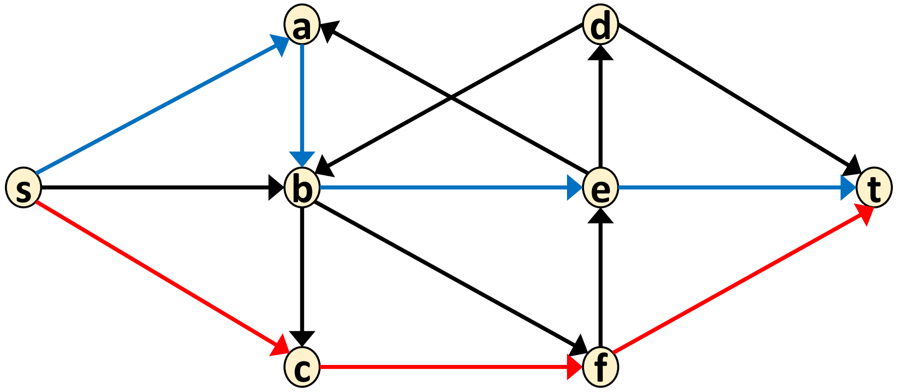
Reduction to Max Flow

Maximal set of Edge-disjoint paths



Recombine the split nodes

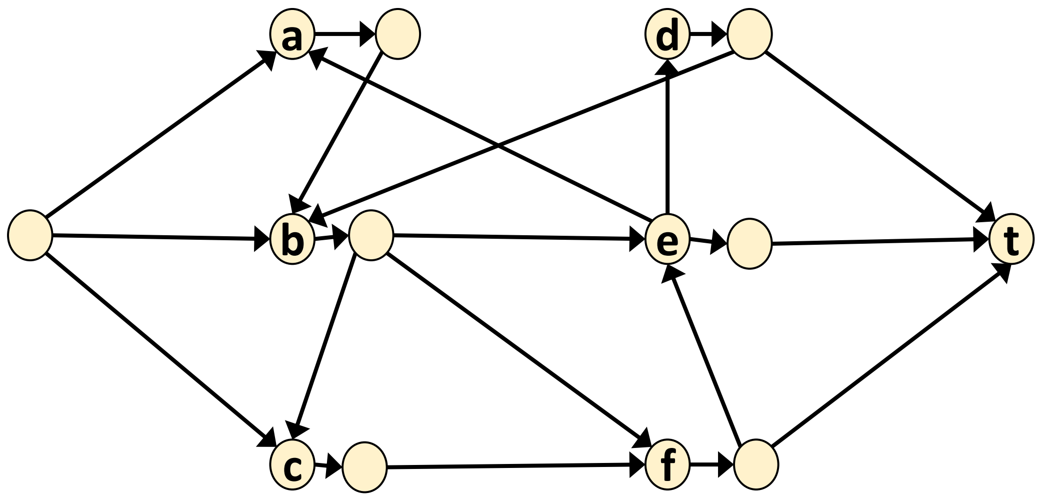
Reduction



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: $O(n+m)$

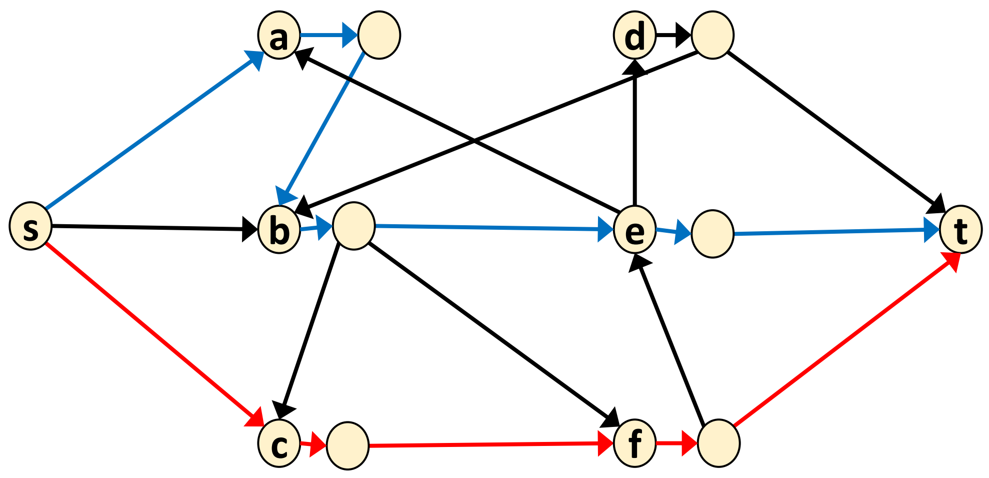
Computing edge disjoint paths: $O(nm^2)$

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

HW6 due today @ 11:59pm.

HW7 released today, due Wednesday 11/26 @ 11:59pm

Quiz 2 on Friday 11/21 in class

We'll release a practice quiz this evening

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