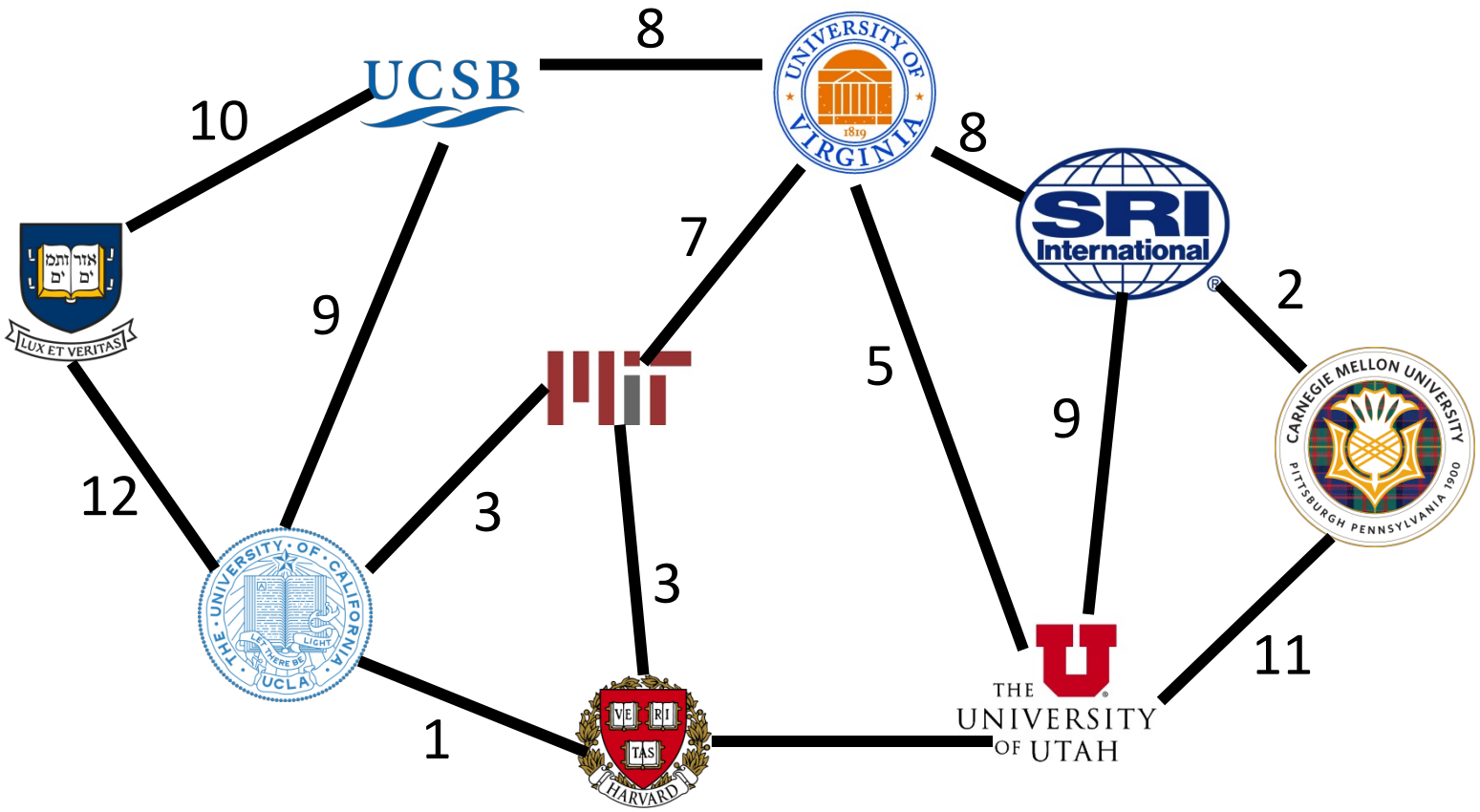
CS 4102: Algorithms

Lecture 22: Network Flow

David Wu

Fall 2019

Review: All-Pairs Shortest Path



Thus far: single-source shortest path algorithms (Dijkstra, Bellman-Ford)

All-pairs shortest-paths: find shortest path between every pair of nodes

Review: All-Pairs Shortest Path

Naïvely: Run single-source shortest paths algorithm for each node s (to compute shortest path from s to every other node in the graph)

- If edge weights are all non-negative, can use Dijkstra (running time $O(|V||E|\log|V|)$
- If edge weights can be negative, can use Bellman-Ford (running time $O(|V|^2|E|)$

When $|E| = \Omega(|V|^2)$, both of these algorithms are $O(|V|^3 \log|V|)$ or $O(|V|^4)$

Floyd-Warshall All-Pairs Shortest Paths

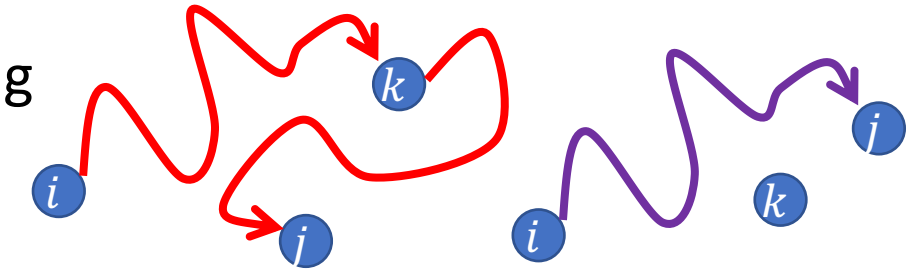
Finds all-pairs shortest paths in $\Theta(|V|^3)$ using dynamic programming

Also works if graph has negative-weight edges

Same observation as before: Every subpath of a shortest path is itself a shortest path (optimal substructure)

- Namely if shortest path from i to j goes through k, then the $i \to k$ and $k \to j$ subpaths must themselves be a shortest path

$\text{Short}(i, j, k) =$ weight of shortest path from $i \to j$ using nodes $1, \ldots, k$ as intermediate hops



Two possibilities for node k:

Shortest path from i to j includes k

$\text{Short}(i, k, k-1) + \text{Short}(k, j, k-1)$

OR

Shortest path from i to j excludes k

$\text{Short}(i, j, k-1)$

Floyd-Warshall All-Pairs Shortest Paths

Finds all-pairs shortest paths in $\Theta(|V|^3)$ using dynamic programming

Also works if graph has negative-weight edges

Same observation as before: Every subpath of a shortest path is itself a shortest path (optimal substructure)

- Namely if shortest path from i to j goes through k, then the $i \to k$ and $k \to j$ subpaths must themselves be a shortest path

$\text{Short}(i, j, k) =$ weight of shortest path from $i \to j$ using nodes $1, \ldots, k$ as intermediate hops

$$\text{Short}(i, j, k) = \min \begin{cases} \text{Short}(i, k, k-1) + \text{Short}(k, j, k-1) \\ \text{Short}(i, j, k-1) \end{cases}$$

Floyd-Warshall All-Pairs Shortest Paths

```
allocate short[n][n][n] (initialized to ∞)
for (i, j) in E:
  short[i][j][0] = w[i][j]
for i = 1,...,n:
  short[i][i][0] = 0
```

$\boldsymbol{k = 0}$**:** shortest path cannot use any intermediate nodes (must be direct path)

$\boldsymbol{k = 0}$**:** shortest path from node to itself is always 0

$$\text{Short}(i,j,k) = \min \begin{cases} \text{Short}(i,k,k-1) + \text{Short}(k,j,k-1) \\ \text{Short}(i,j,k-1) \end{cases}$$

```
for k = 1,...,n:
  for i = 1,...,n:
    for j = 1,...,n:
      short[i][j][k] = min(short[i][k][k-1] + short[k][j][k-1], short[i][j][k-1])
```

Floyd-Warshall All-Pairs Shortest Paths

```
allocate short[n][n] (initialized to ∞)
for (i, j) in E:
  short[i][j] = w[i][j]
for i = 1,...,n:
  short[i][i] = 0
```

In this case, the initialization step is constructing the adjacency matrix of the graph (this step is not needed if graph already represented in this form!)

```
for k = 1,...,n:
  for i = 1,...,n:
    for j = 1,...,n:
      short[i][j] = min(short[i][k] + short[k][j], short[i][j])
```

$$\text{Short}(i,j,k) = \min \begin{cases} \text{Short}(i,k,k-1) + \text{Short}(k,j,k-1) \\ \text{Short}(i,j,k-1) \end{cases}$$

Observation: `short[i][j][k]` only depends on values for `short[][][k - 1]`, so we can just use a single two-dimensional array

Floyd-Warshall All-Pairs Shortest Paths

```
allocate short[n][n] (initialized to ∞)
for (i, j) in E:
  short[i][j] = w[i][j]
for i = 1,...,n:
  short[i][i] = 0
```

$\boldsymbol{k = 0}$**:** shortest path cannot use any intermediate nodes (must be direct path)

$\boldsymbol{k = 0}$**:** shortest path from node to itself is always 0

```
for k = 1,...,n:
  for i = 1,...,n:
    for j = 1,...,n:
      short[i][j] = min(short[i][k] + short[k][j], short[i][j])
```

$$\text{Short}(i,j,k) = \min \begin{cases} \text{Short}(i,k,k-1) + \text{Short}(k,j,k-1) \\ \text{Short}(i,j,k-1) \end{cases}$$

Very simple implementation!

Running time: $O(n^3) = O(|V|^3)$

Today's Keywords

Graphs

Network flow

Ford-Fulkerson

Edmonds-Karp

Max-Flow Min-Cut Theorem

CLRS Readings: Chapter 25, 26

Homework

HW7 due **today, 11pm**

- Graph algorithms
- Written (use LaTeX!) – Submit both **zip** and **pdf** (two separate attachments)!

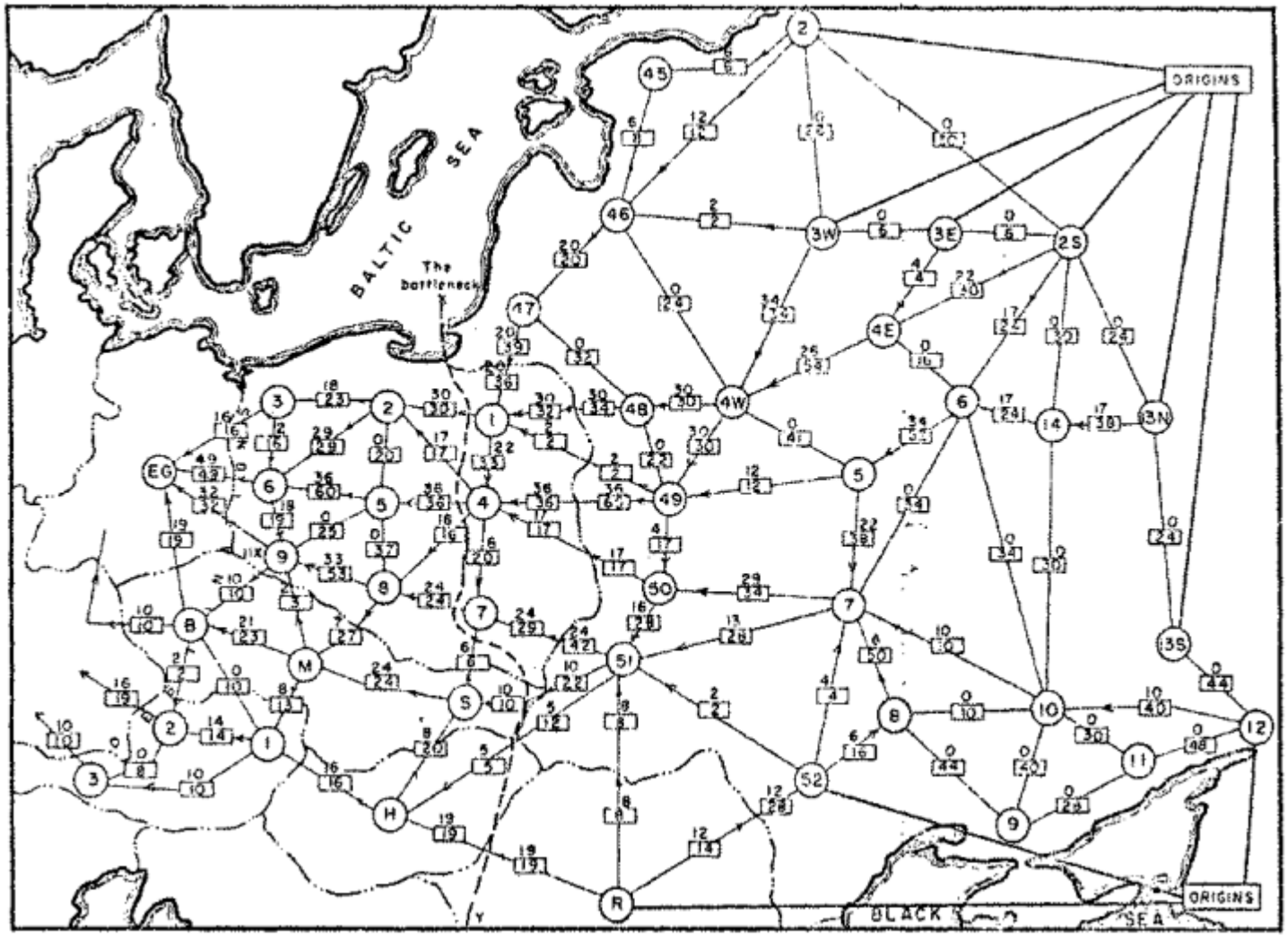
HW10B due **today, 11pm**

- No late submissions allowed (no exceptions)

HW8 out **today**, due **Thursday, November 21, 11pm**

- Programming assignment (Python or Java)
- Graph algorithms

Network Flow



Railway map of Western USSR, 1955

Question: What is the maximum throughput of the railroad network?

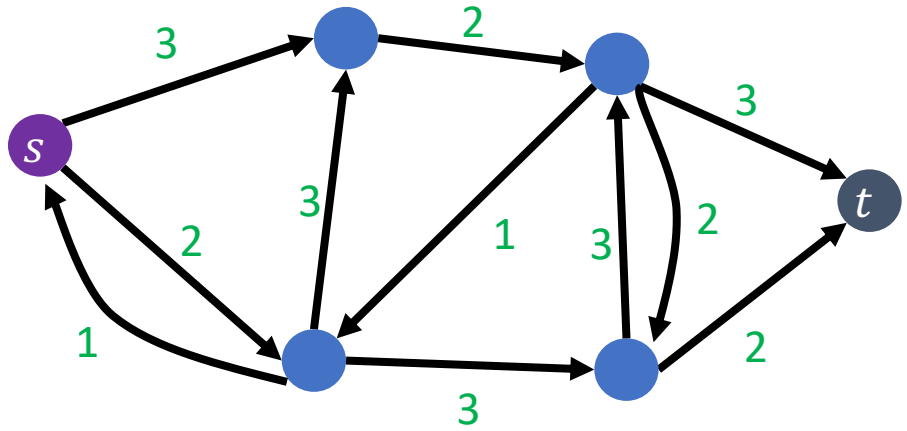
Flow Networks

Graph $G = (V, E)$

Source node $s \in V$

Sink node $t \in V$

Edge capacities $c(e) \in \mathbb{R}^+$



Max flow intuition: If s is a faucet, t is a drain, and s connects to t through a network of pipes E with capacities $c(e)$, what is the <u>maximum</u> amount of water which can flow from the faucet to the drain?

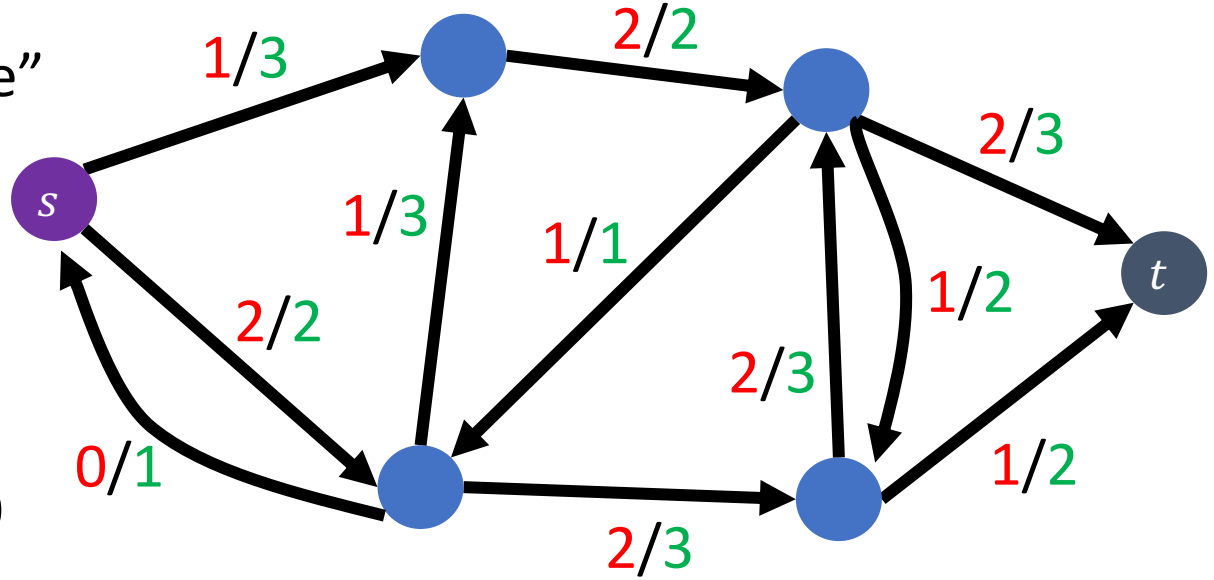
Network Flow

Assignment of values $f(e)$ to edges
- "Amount of water going through that pipe"

Capacity constraint
- $f(e) \leq c(e)$
- "Flow cannot exceed capacity"

Flow constraint
- $\forall v \in V - \{s, t\}, \text{inflow}(v) = \text{outflow}(v)$
- $\text{inflow}(v) = \sum_{x \in V} f(x, v)$
- $\text{outflow}(v) = \sum_{x \in V} f(v, x)$
- Water going in must match water coming out

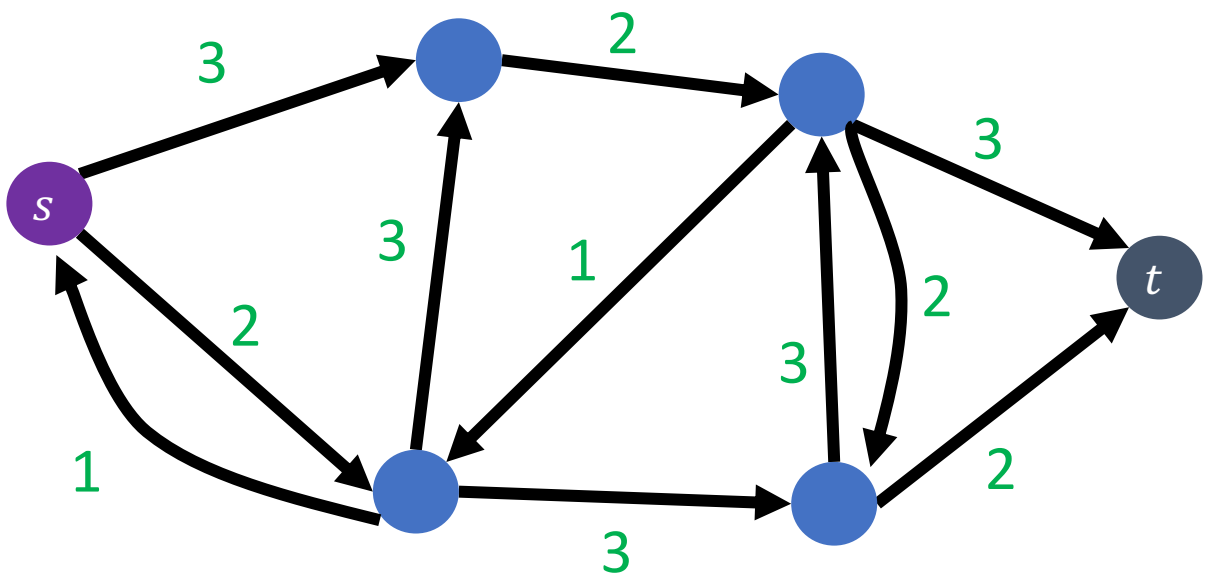
Flow of G: $|f| = \text{outflow}(s) - \text{inflow}(s)$
- Net outflow of s 3 in this example

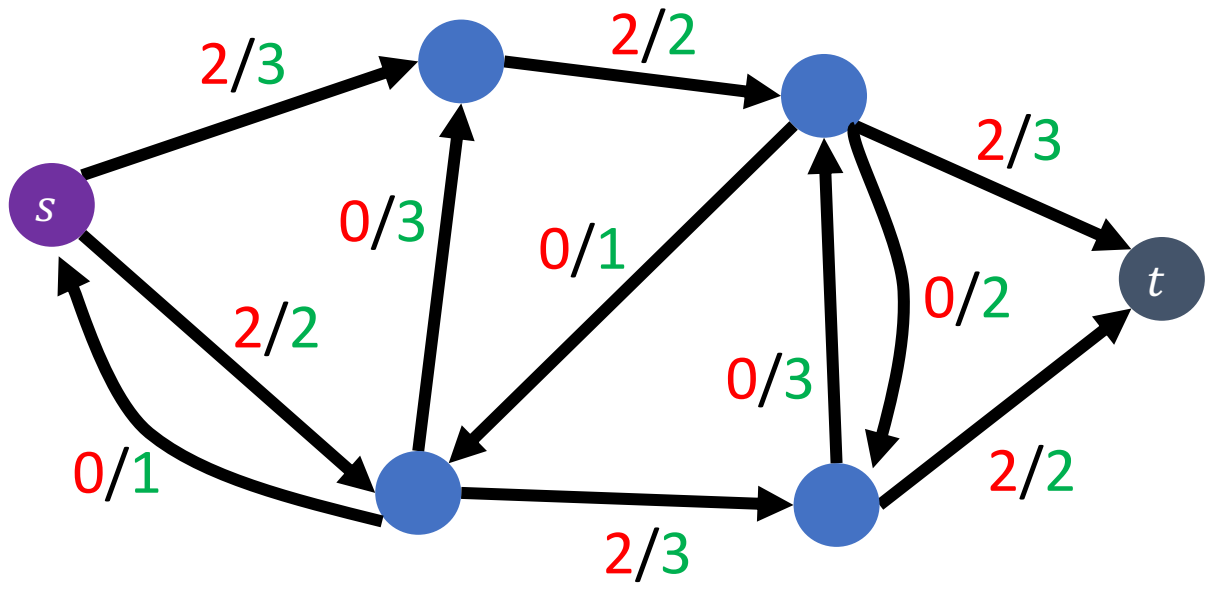


Maximum Flow Problem

Of all valid flows through the graph, find the one that maximizes:

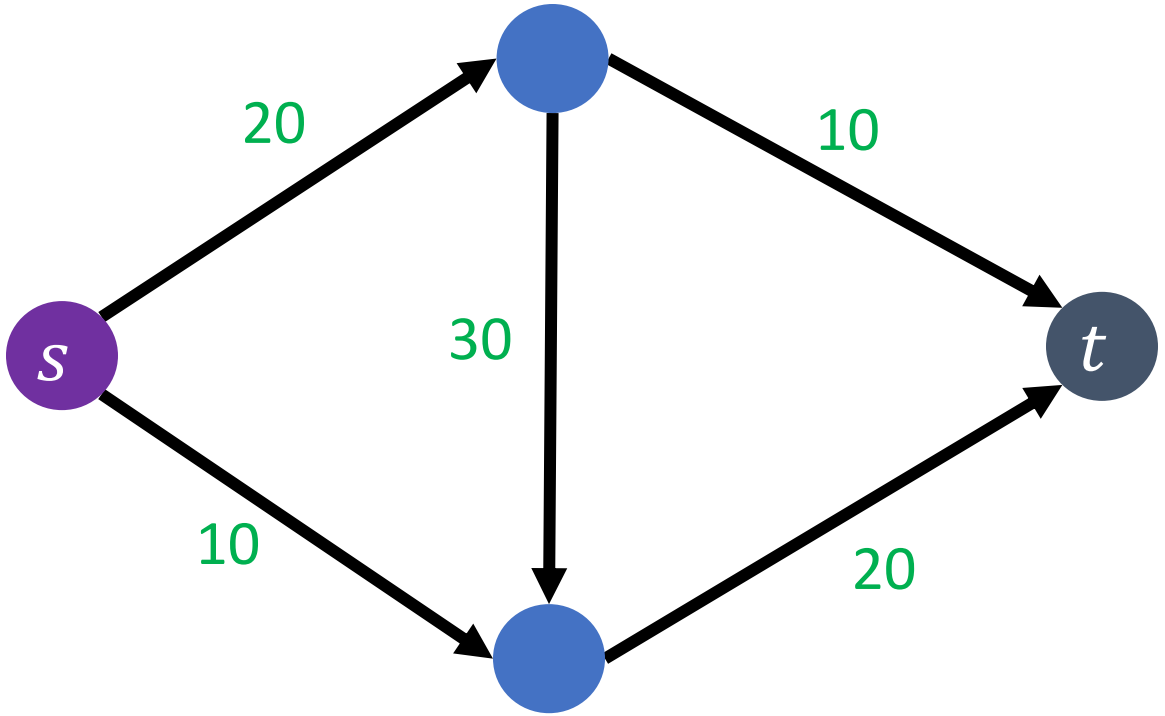
$$|f| = \text{outflow}(s) - \text{inflow}(s)$$





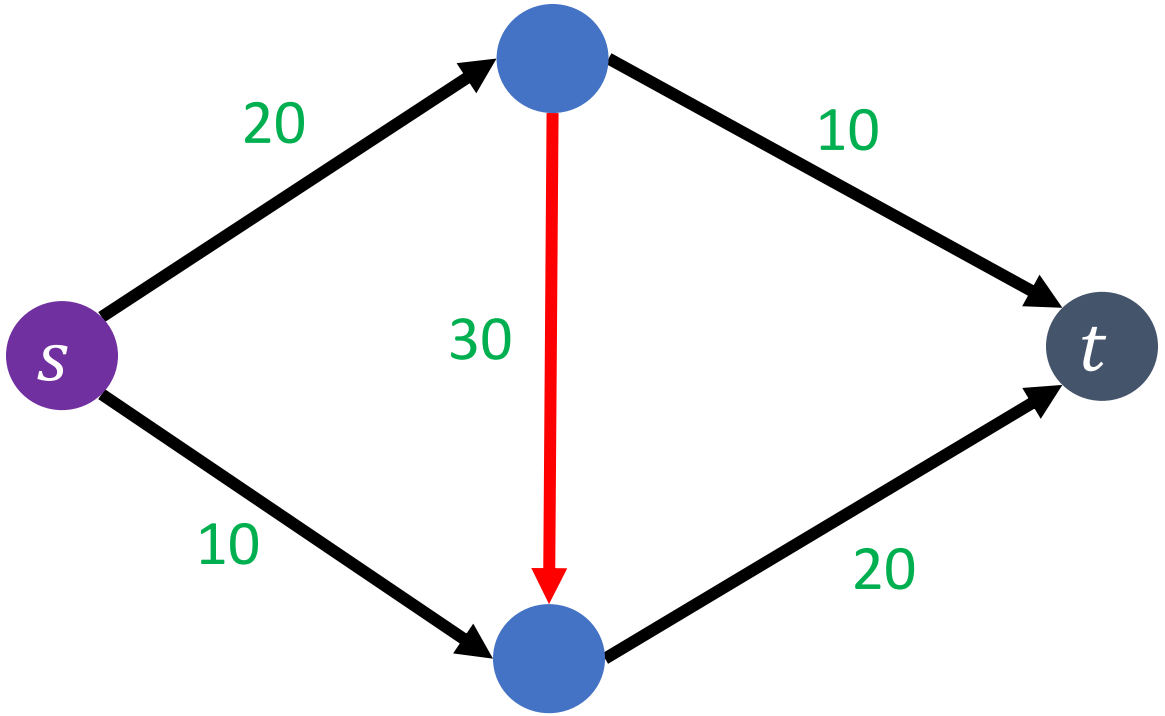
Greedy Approach

Greedy choice: saturate highest capacity path first



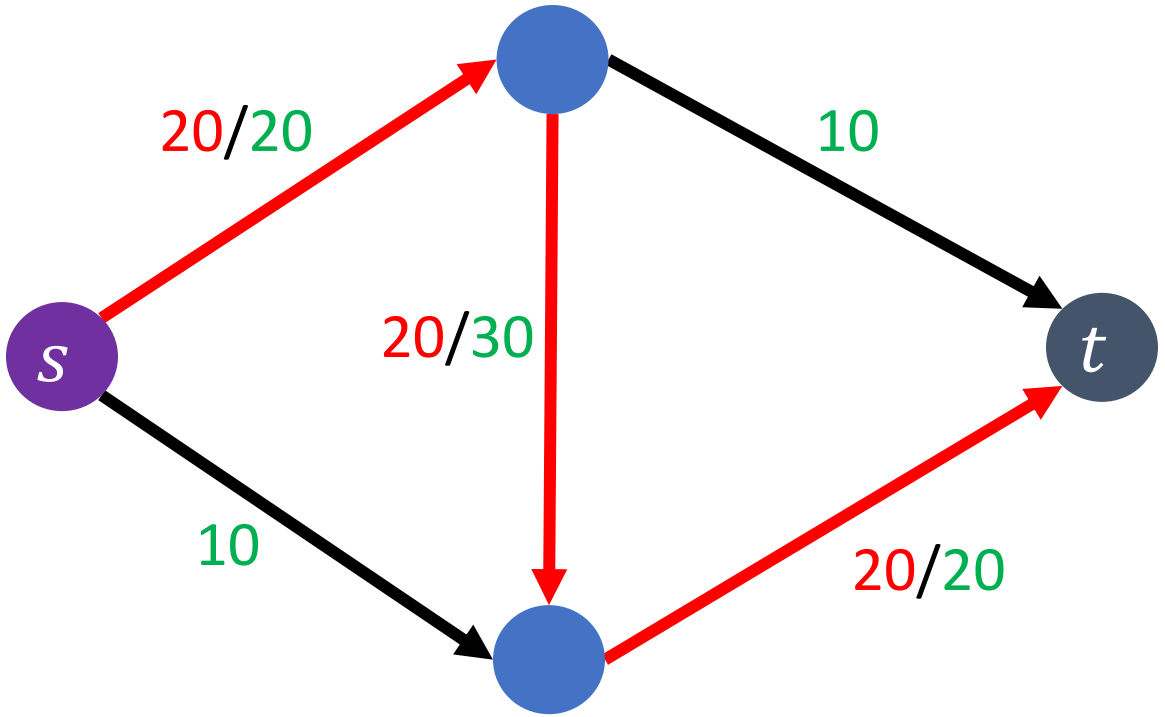
Greedy Approach

Greedy choice: saturate highest capacity path first



Greedy Approach

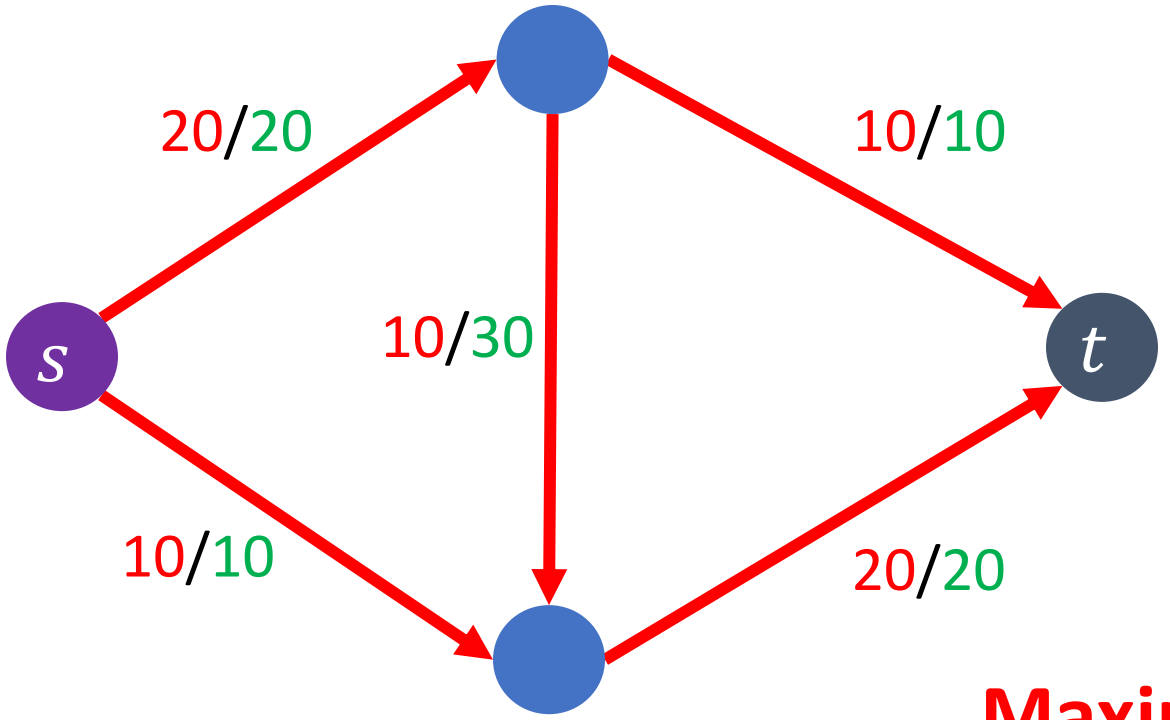
Greedy choice: saturate highest capacity path first



Flow: 20

Greedy Approach

Greedy choice: saturate highest capacity path first



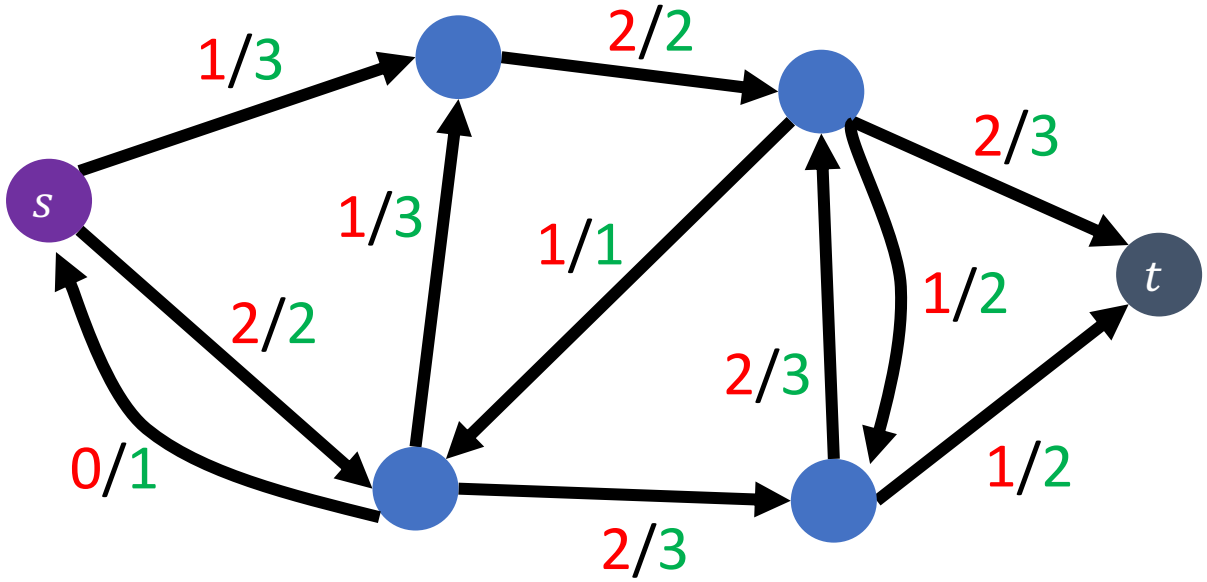
Maximum Flow: 30

Observe: highest capacity path is not saturated in optimal solution

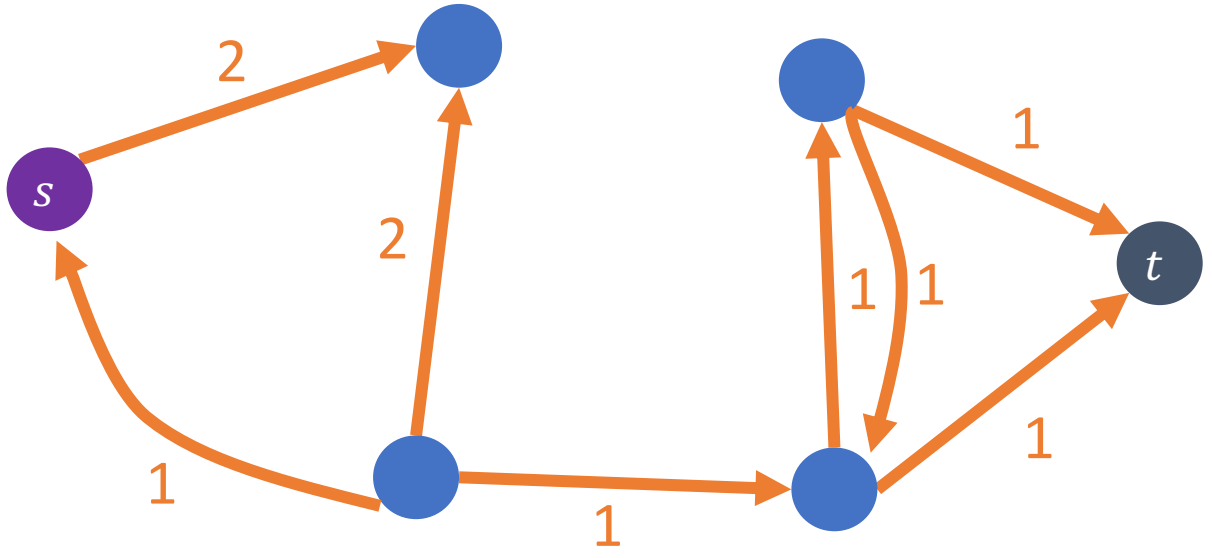
Residual Graphs

Given a flow f in graph G, the residual graph G_f models additional flow that is possible

- Forward edge for each edge in G with weight set to remaining capacity $c(e) - f(e)$
  - Models additional flow that can be sent along the edge



Flow f in G

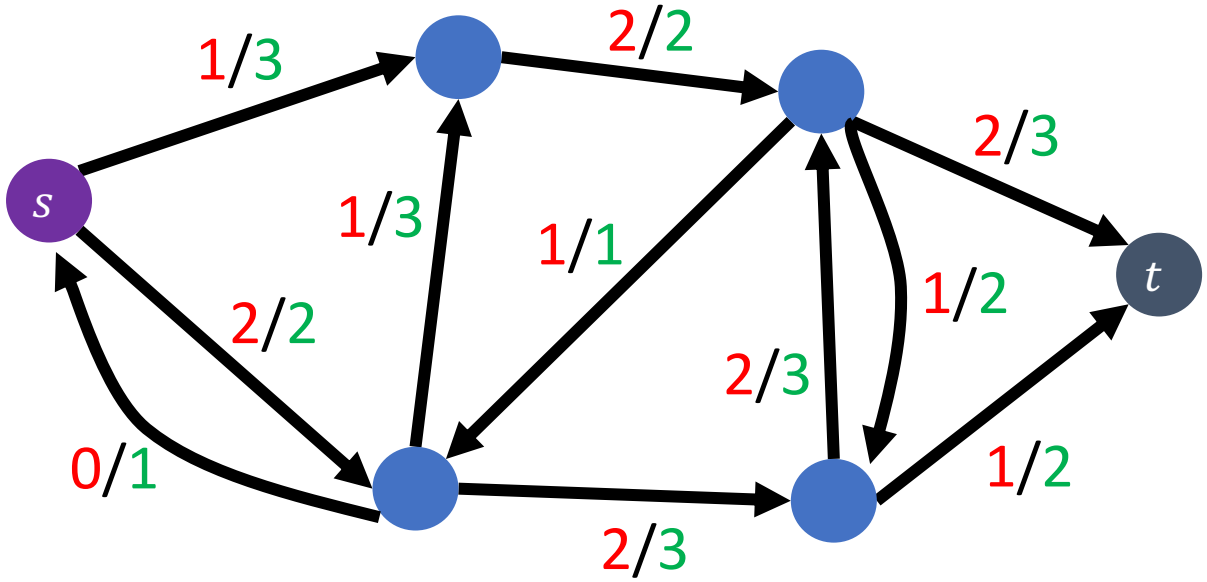


Residual graph G_f

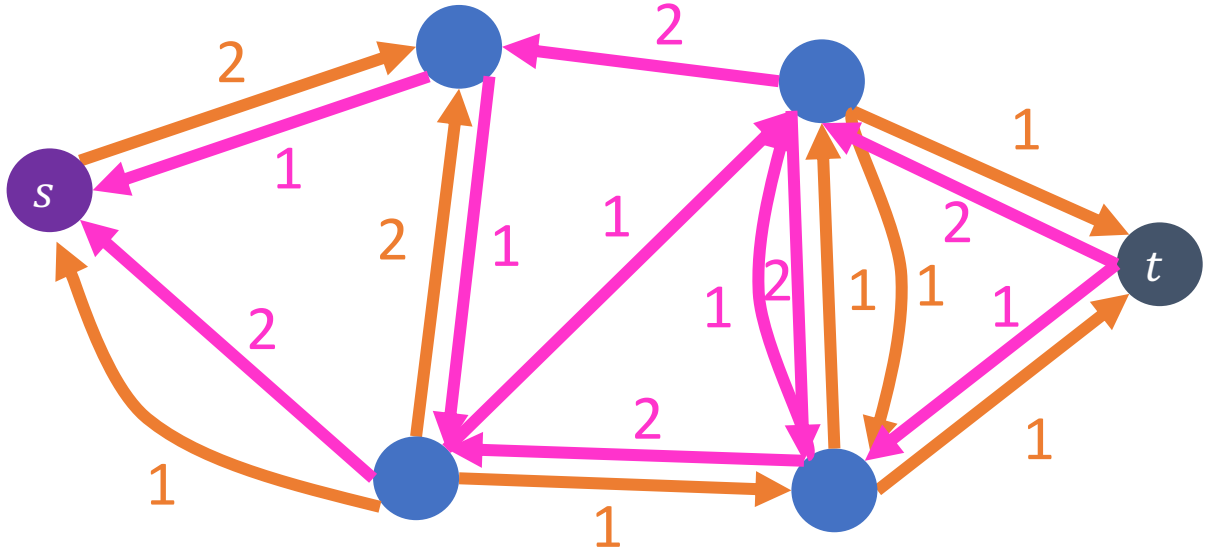
Residual Graphs

Given a flow f in graph G, the residual graph G_f models additional flow that is possible

- Forward edge for each edge in G with weight set to remaining capacity $c(e) - f(e)$
  - Models additional flow that can be sent along the edge
- Backward edge by flipping each edge e in G with weight set to flow $f(e)$
  - Models amount of flow that can be removed from the edge



Flow f in G



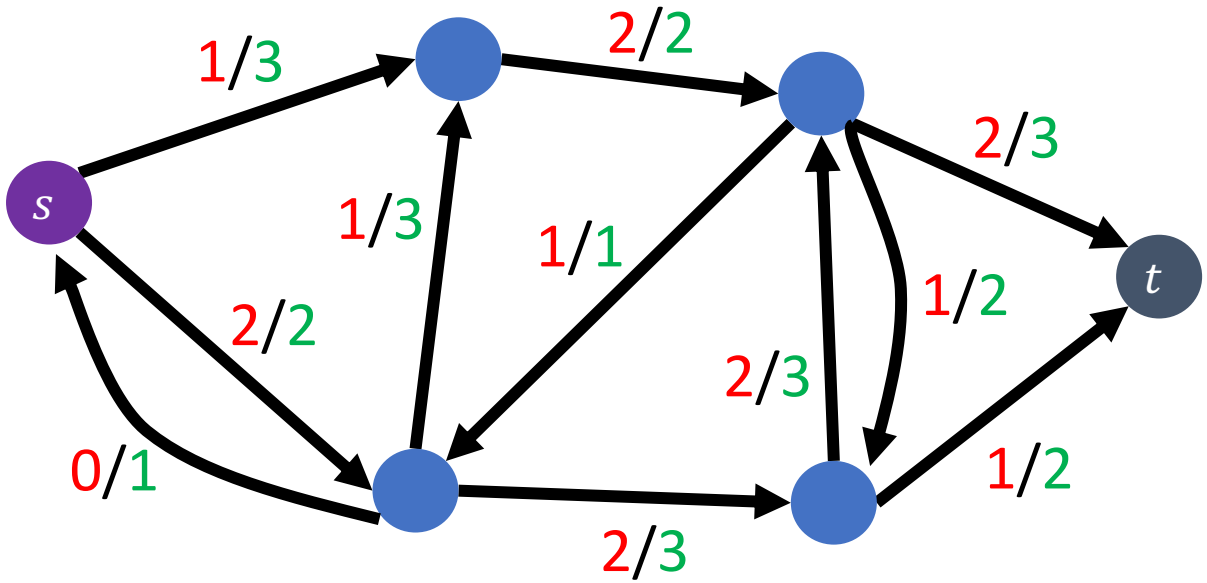
Residual graph G_f

Residual Graphs

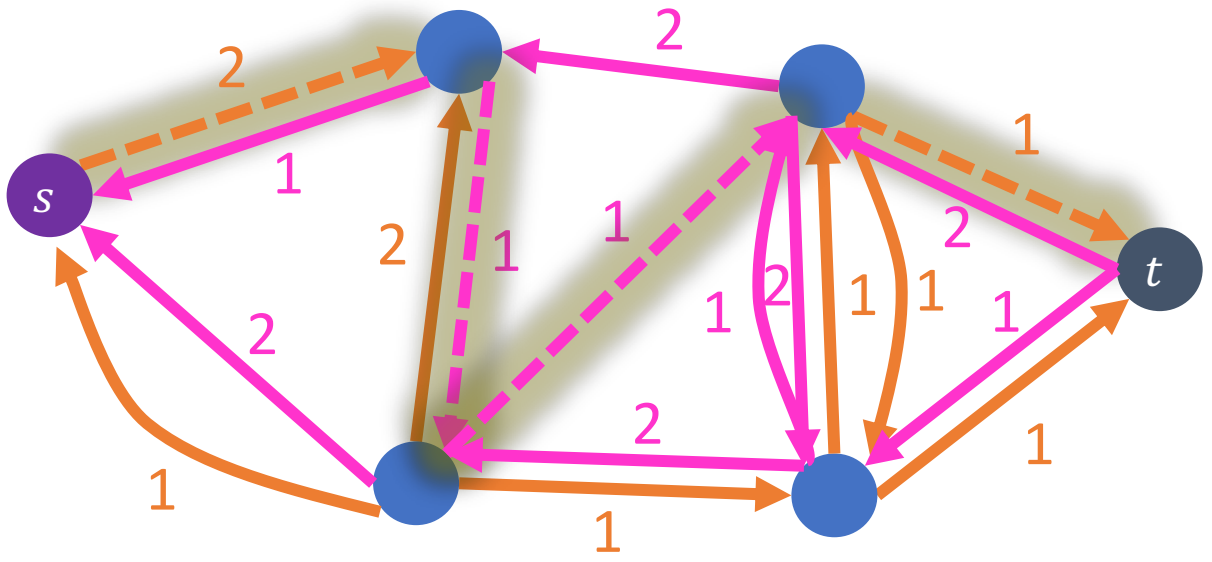
Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

- Send $w(e)$ flow along all forward edges (these have at least $w(e)$ capacity)



Flow f in G



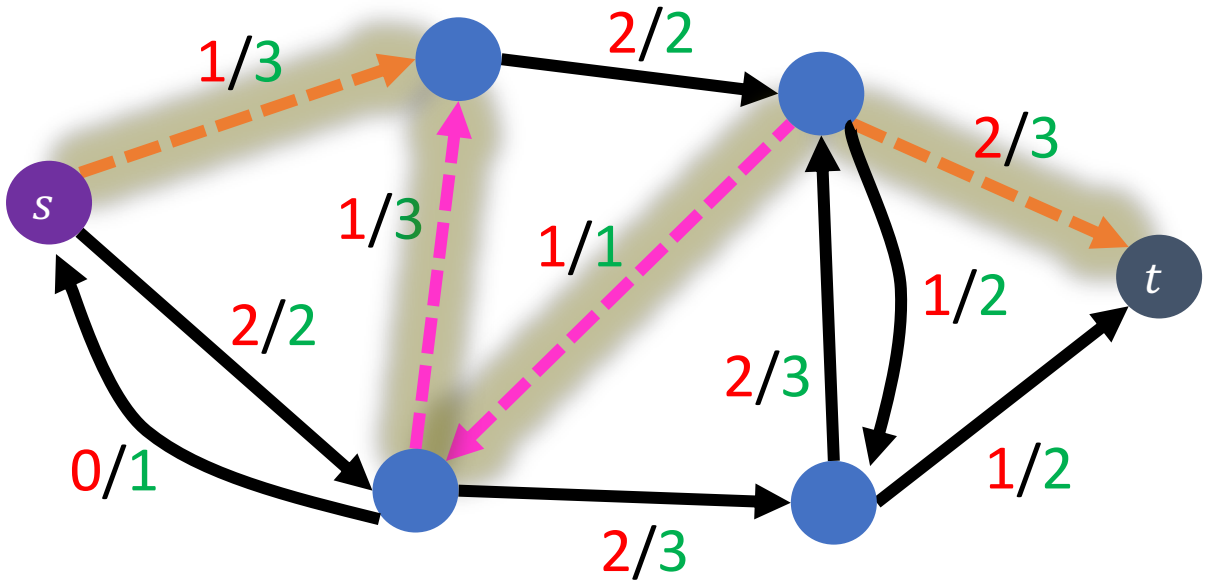
Residual graph G_f

Residual Graphs

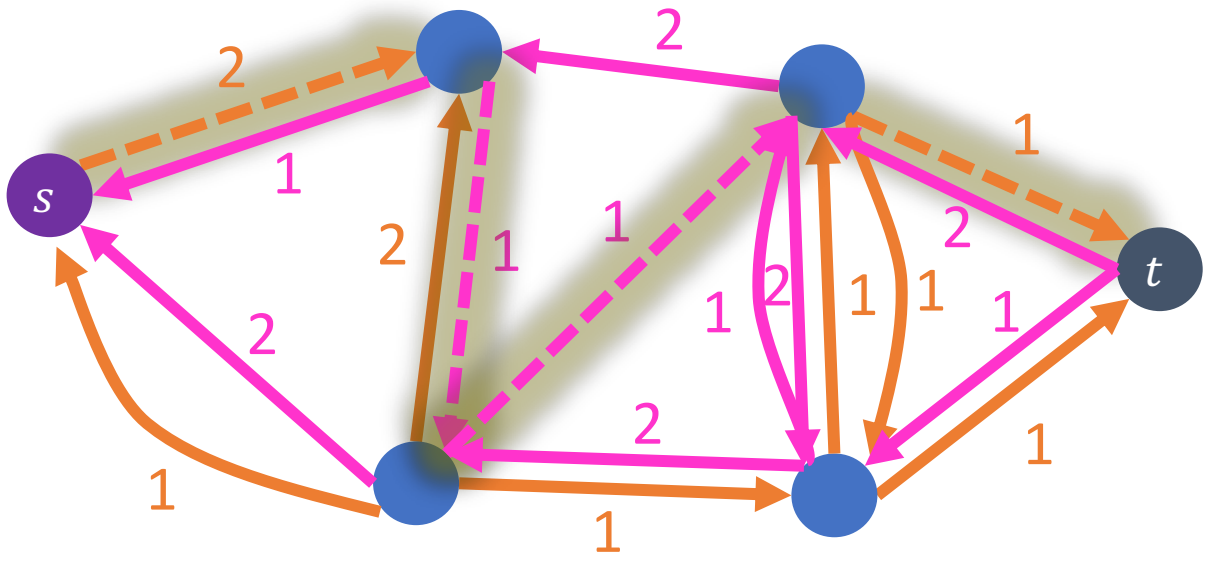
Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

- Send $w(e)$flow along all forward edges (these have at least $w(e)$ capacity)
- Remove $w(e)$flow along all backward edges (these contain at least $w(e)$ units of flow)



Flow f in G



Residual graph G_f

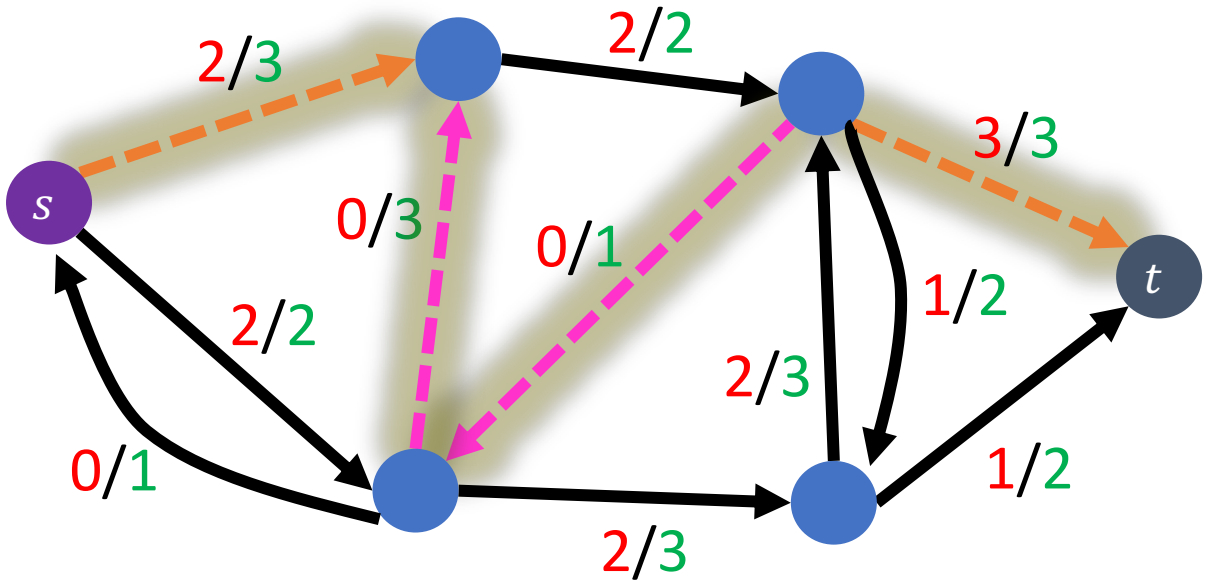
Residual Graphs

Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

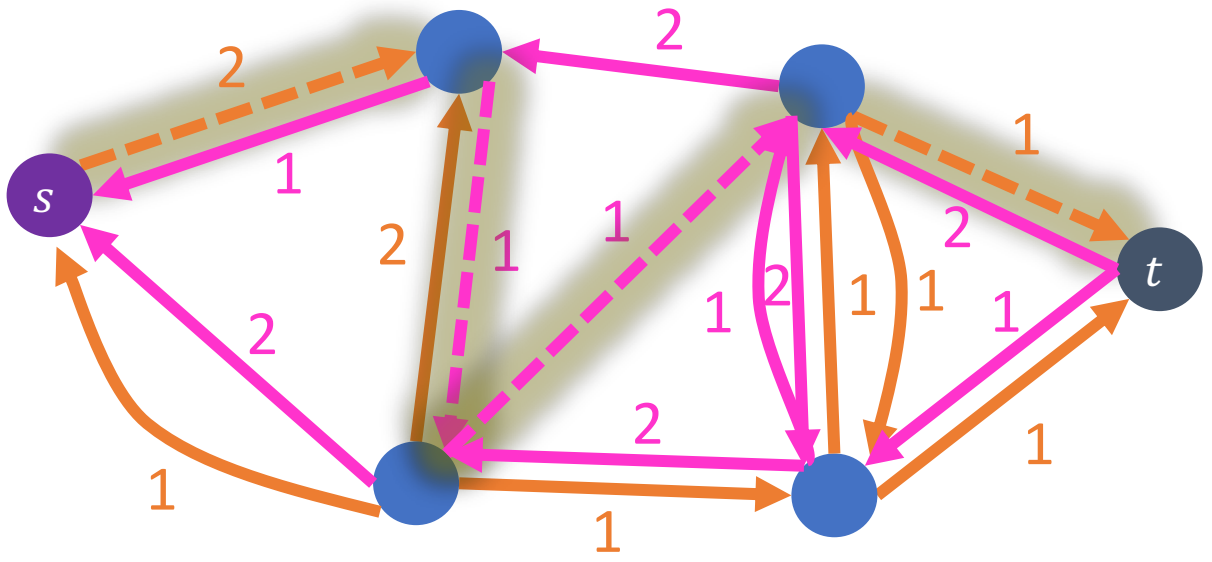
Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

- Send $w(e)$flow along all forward edges (these have at least $w(e)$ capacity)
- Remove $w(e)$flow along all backward edges (these contain at least $w(e)$ units of flow)

Observe: Flow has increased by $w(e)$



Flow f in G



Residual graph G_f

Residual Graphs

Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

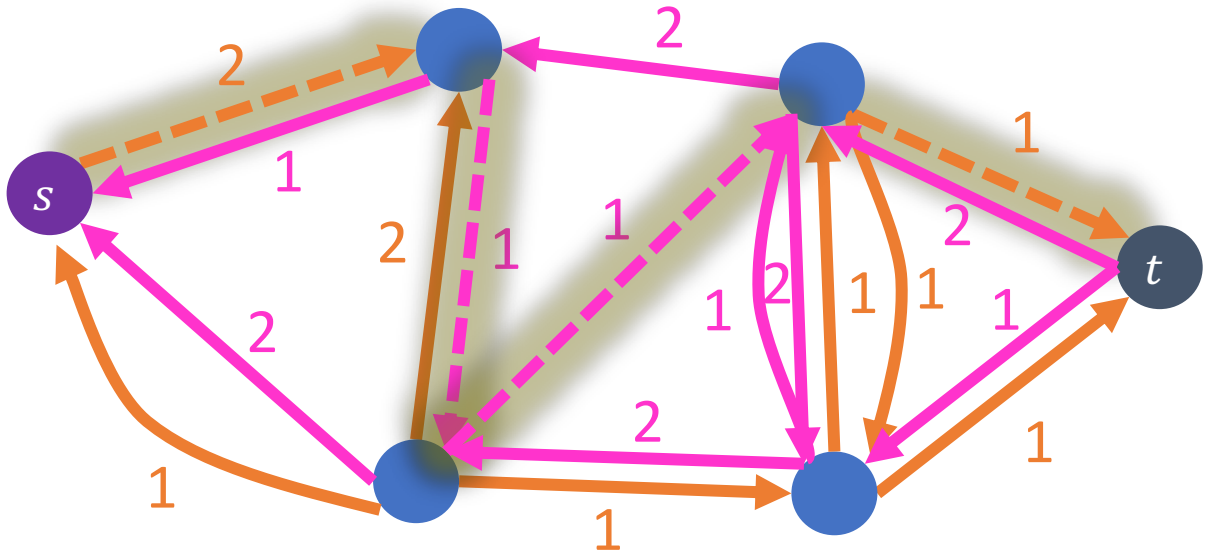
Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

- Send $w(e)$ flow along all forward edges (these have at least $w(e)$ capacity)
- Remove $w(e)$ flow along all backward edges (these contain at least $w(e)$ units of flow)

Observe: Flow has increased by $w(e)$

Why does this respect flow constraints?

- Incoming edge to a node always corresponds to increased flow to the node (more incoming flow from forward edge or less outgoing flow from backward edge)
- Outgoing edge to a node always corresponds to decreased flow to the node



Residual graph G_f

Residual Graphs

Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

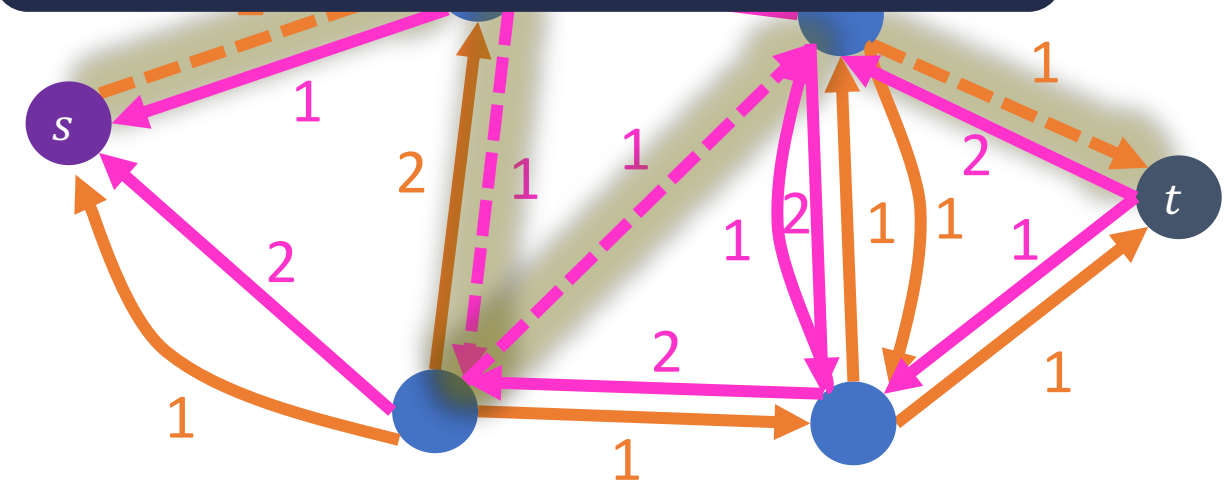
- Send $w(e)$ flow along all forward edges (these have at least $w(e)$ capacity)
- Remove $w(e)$ flow along all backward edges (these contain at least $w(e)$ units of flow)

Observe: Flow has increased by $w(e)$

Why does this respect flow constraints?

- Incoming edge to a node always corresponds to increased flow to the node (more incoming flow from forward edge or less outgoing flow from backward edge)
- Outgoing edge to a node always corresponds to decreased flow to the node

Capacity constraints satisfied by construction of the residual network



Residual graph G_f

Ford-Fulkerson Algorithm

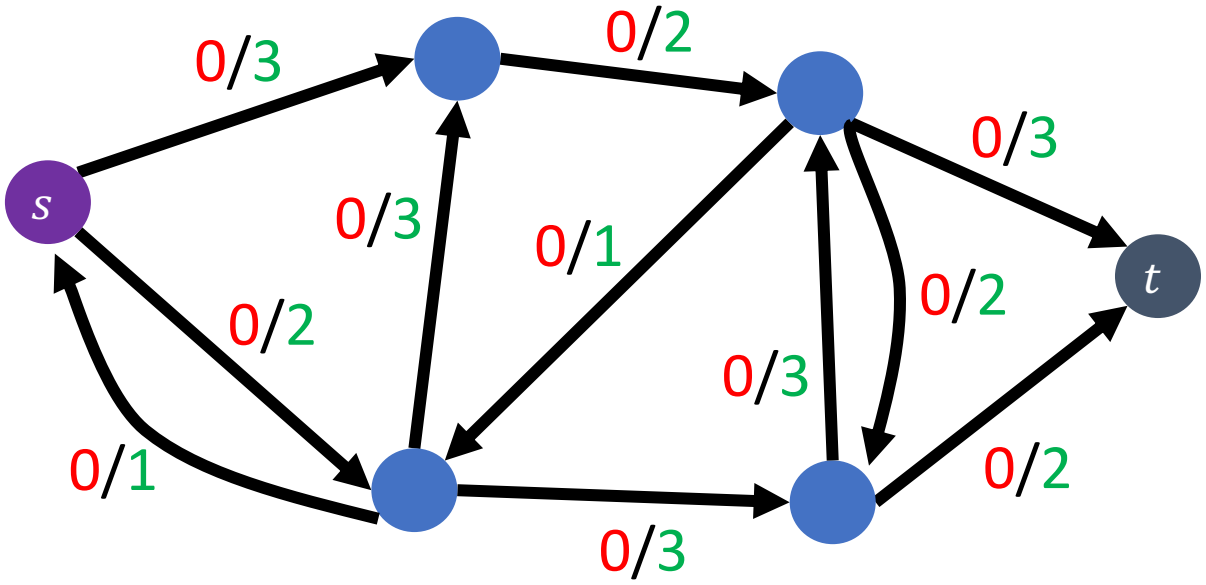
Define an augmenting path to be an $s \to t$ path in the residual graph G_f (using edges of non-zero weight)

Ford-Fulkerson max-flow algorithm:

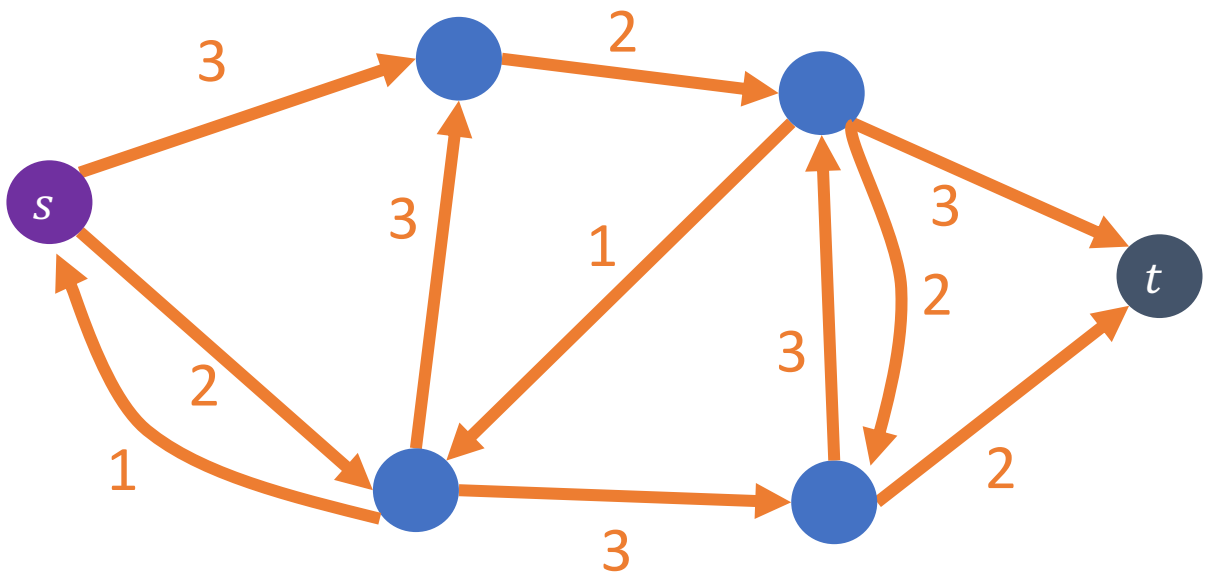
- Initialize $f(e) = 0$ for all $e \in E$
- Construct the residual network G_f
- While there is an augmenting path p in G_f:
  - Let $c = \min_{e \in E} c_f(e)$ ($c_f(e)$ is the weight of edge e in the residual network G_f)
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

Ford-Fulkerson approach: take any augmenting path (will revisit this later)

Ford-Fulkerson Example

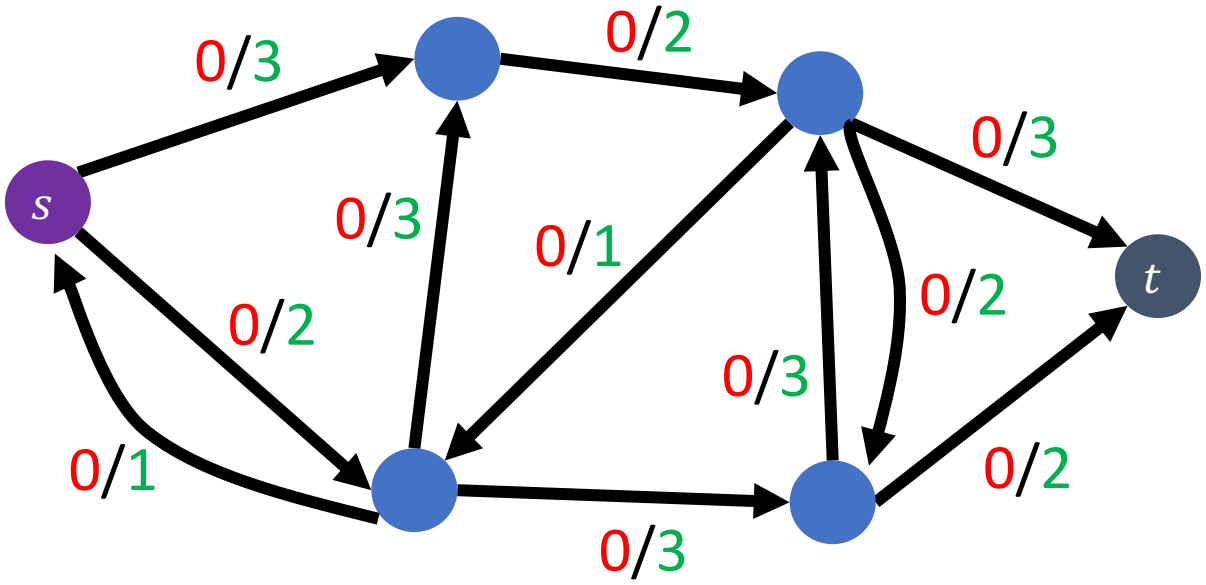


Initially: $f(e) = 0$ for all $e \in E$

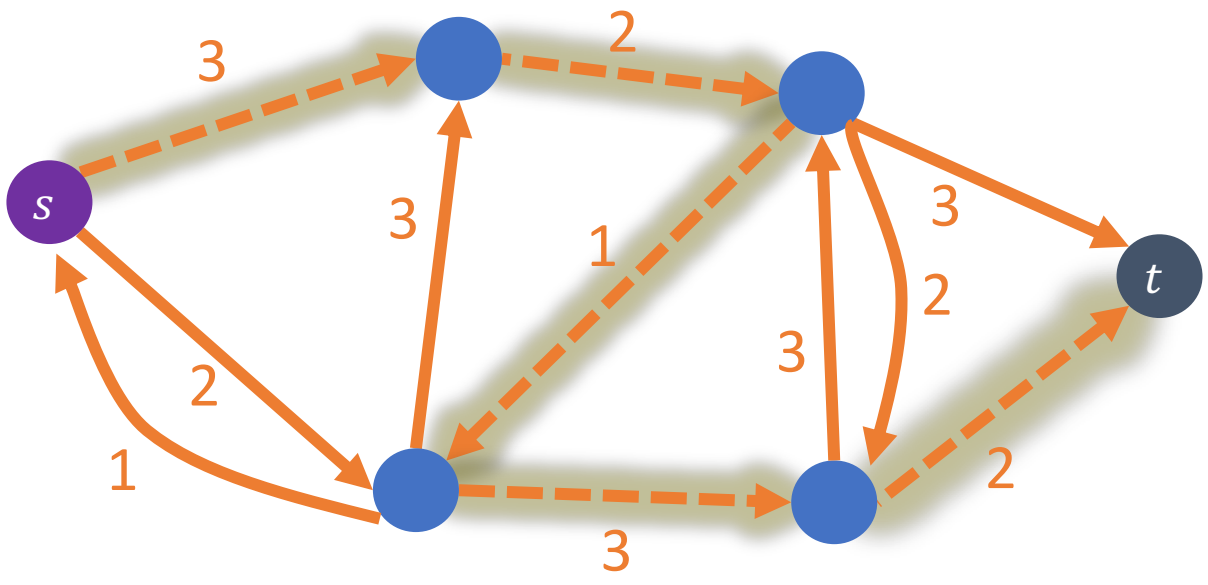


Residual graph G_f

Ford-Fulkerson Example

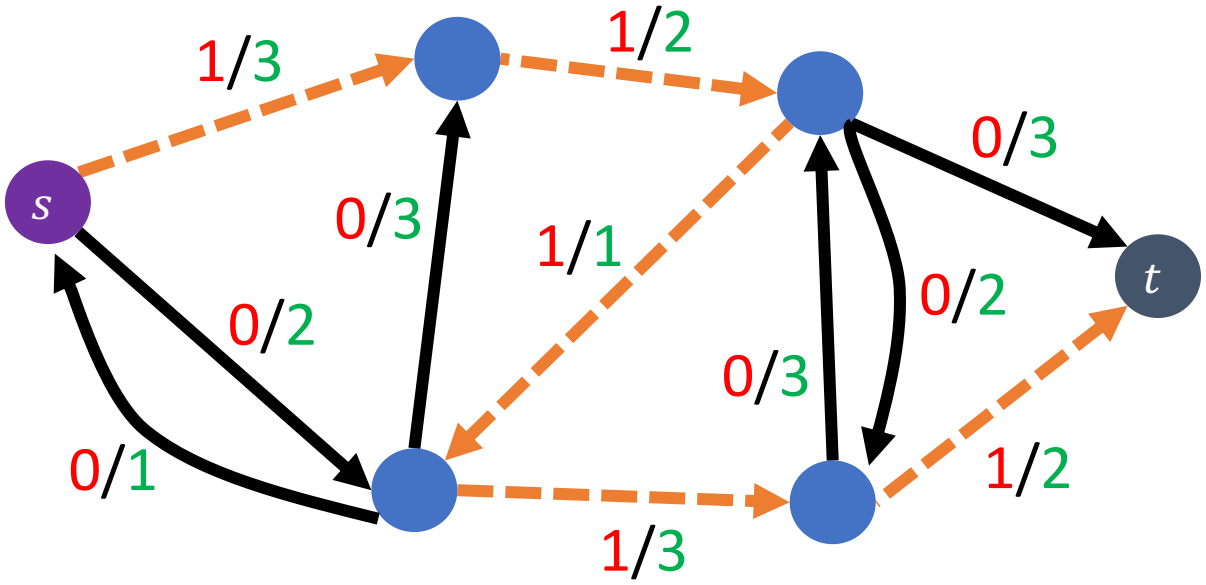


Increase flow by 1 unit

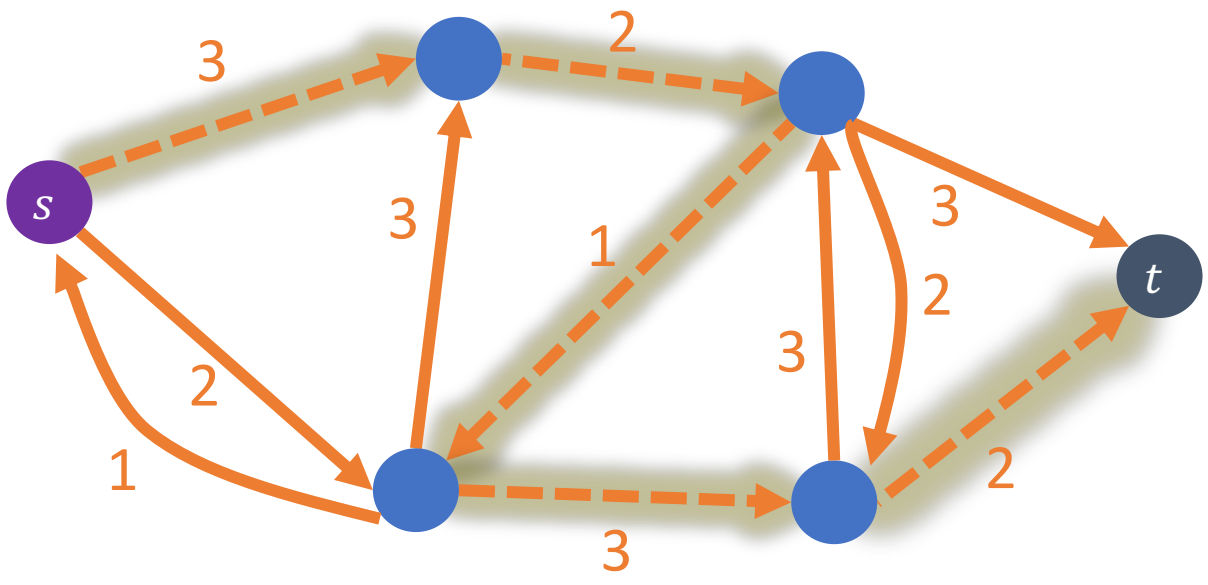


Residual graph G_f

Ford-Fulkerson Example

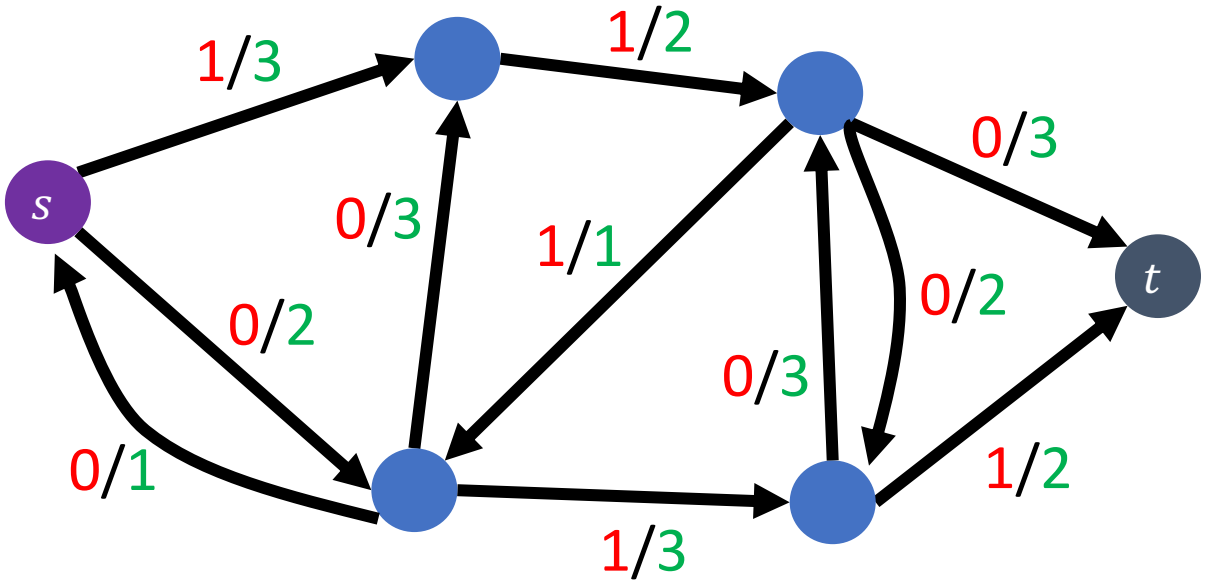


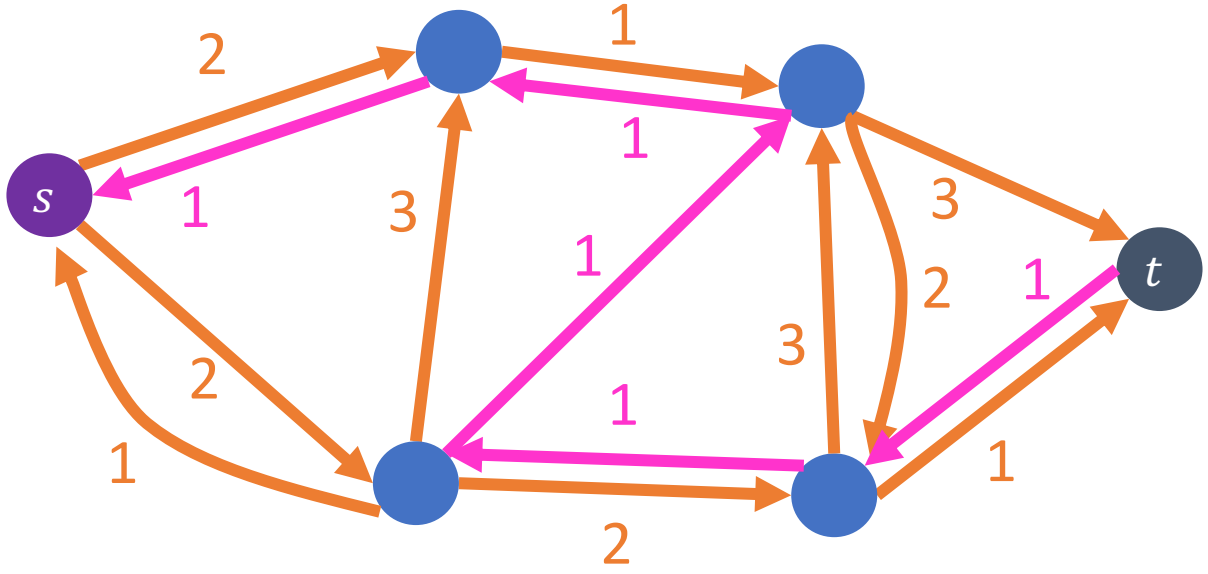
Increase flow by 1 unit



Residual graph G_f

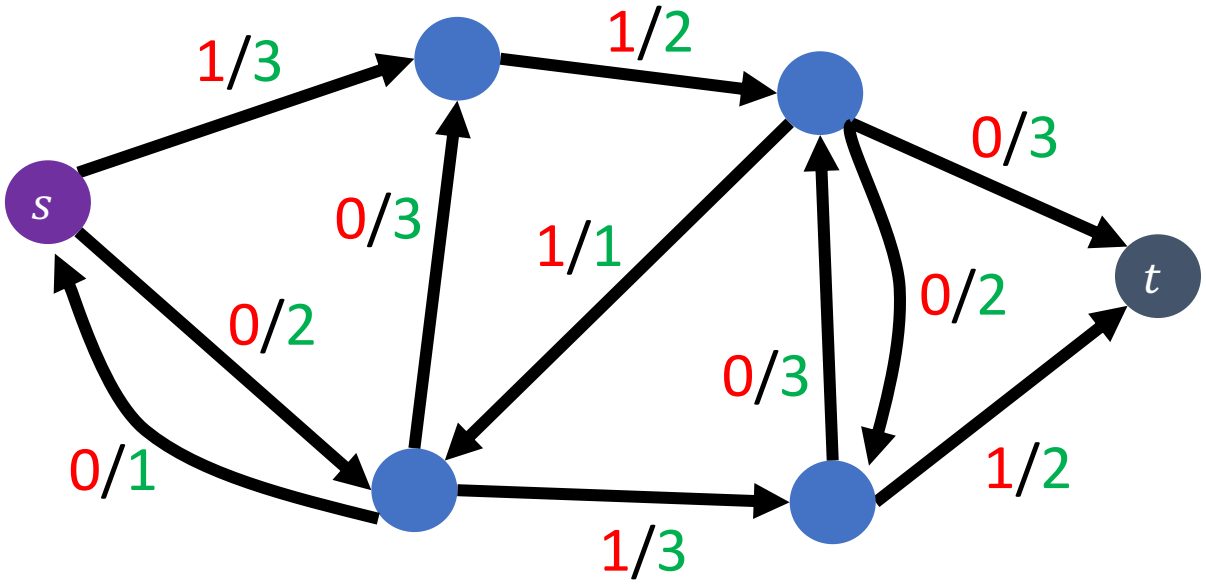
Ford-Fulkerson Example



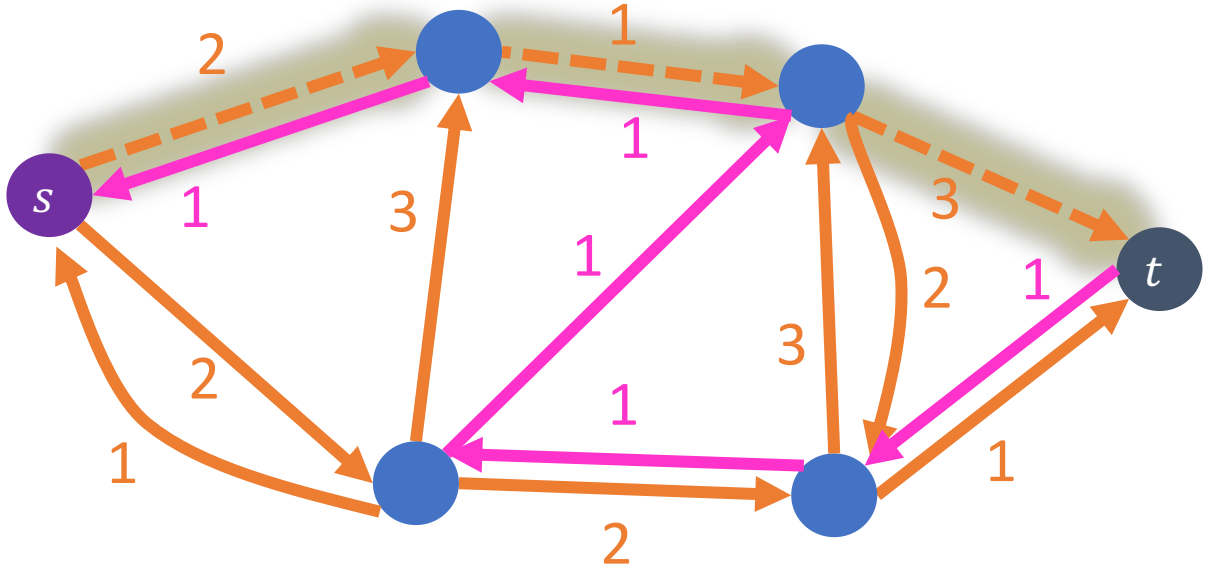


Residual graph G_f

Ford-Fulkerson Example

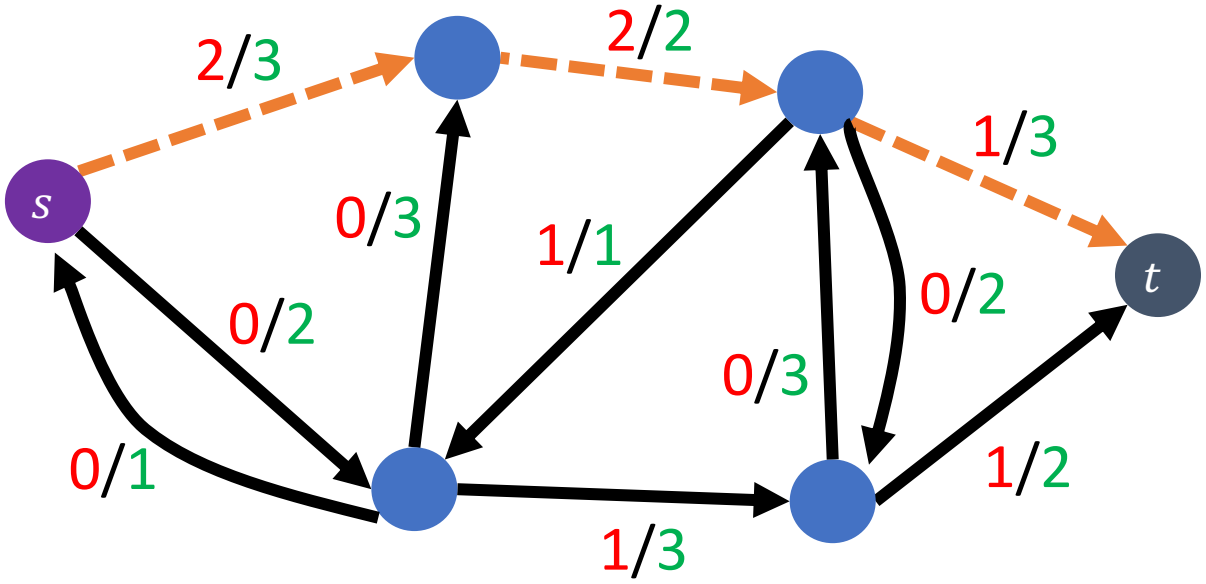


Increase flow by 1 unit

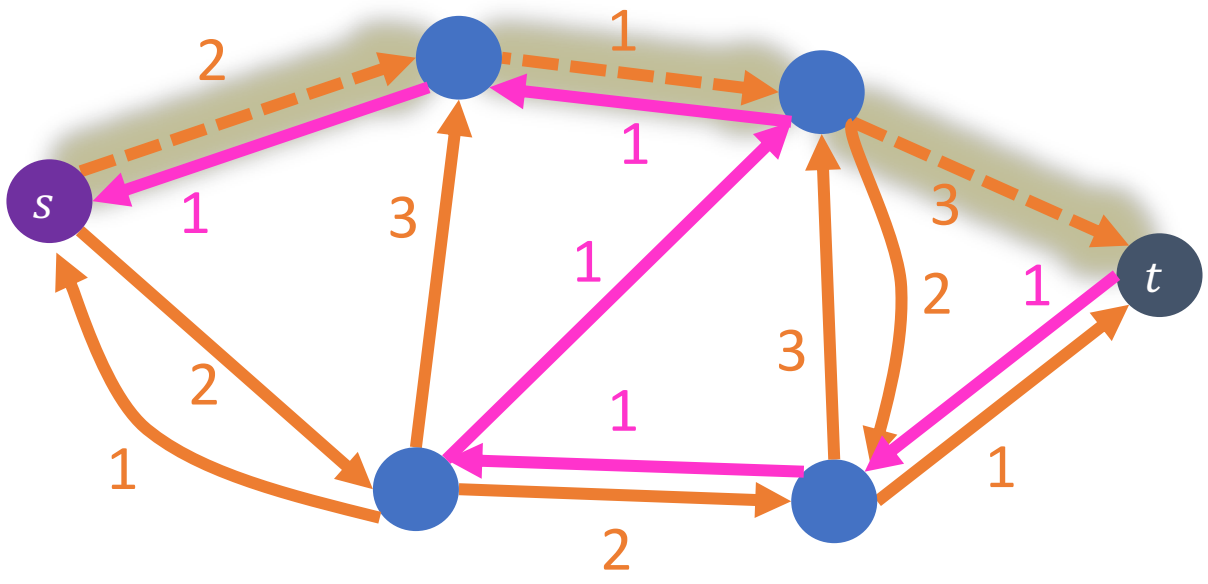


Residual graph G_f

Ford-Fulkerson Example

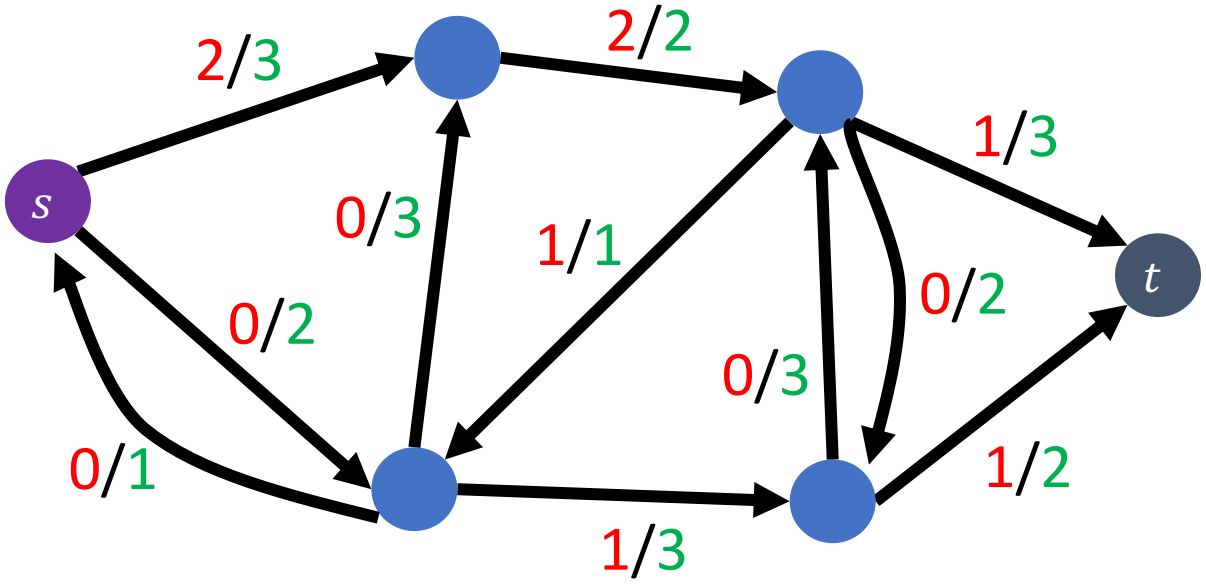


Increase flow by 1 unit

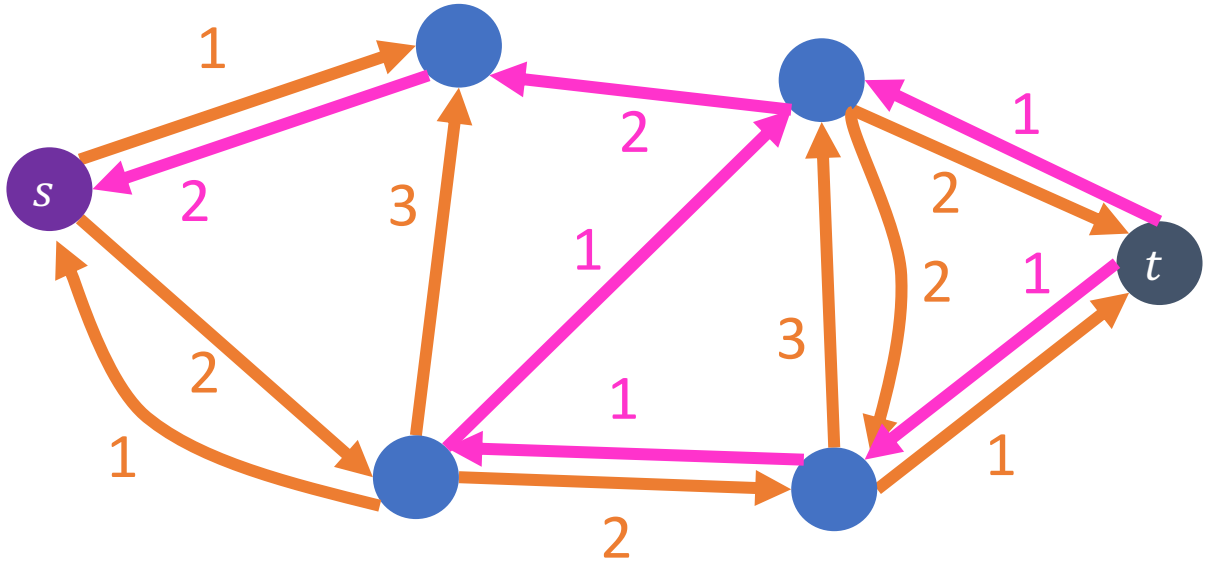


Residual graph G_f

Ford-Fulkerson Example

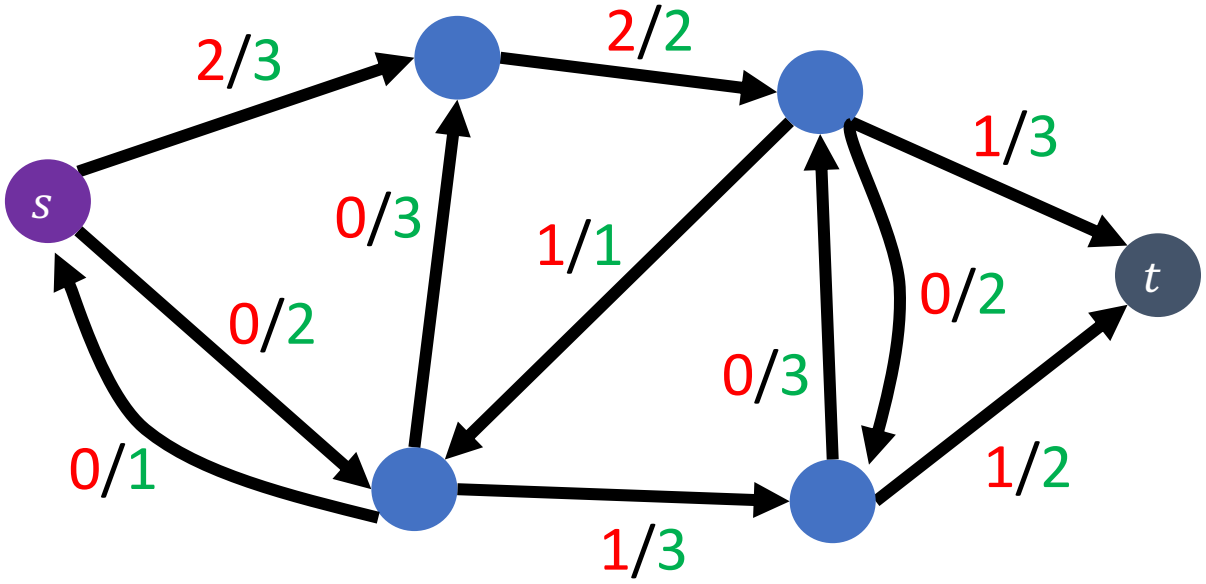


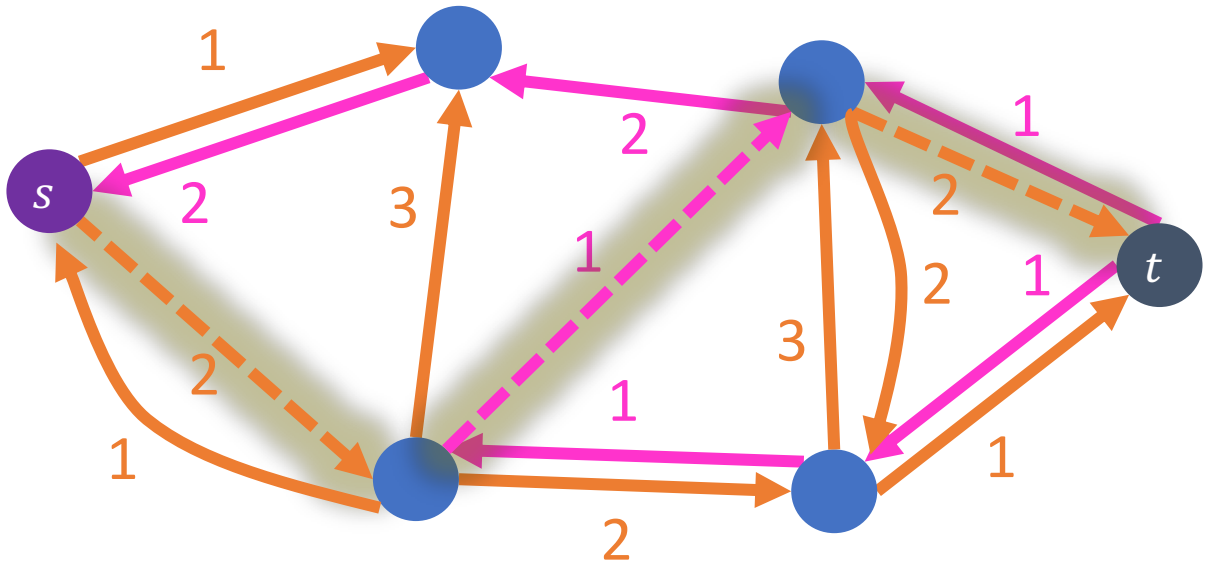
Increase flow by 1 unit



Residual graph G_f

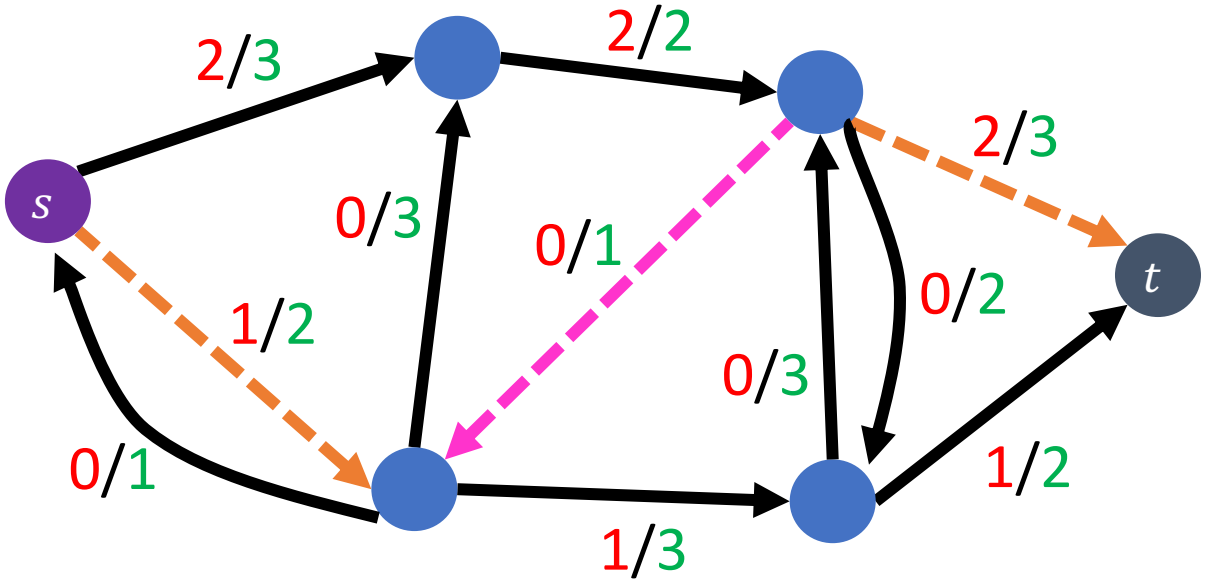
Ford-Fulkerson Example



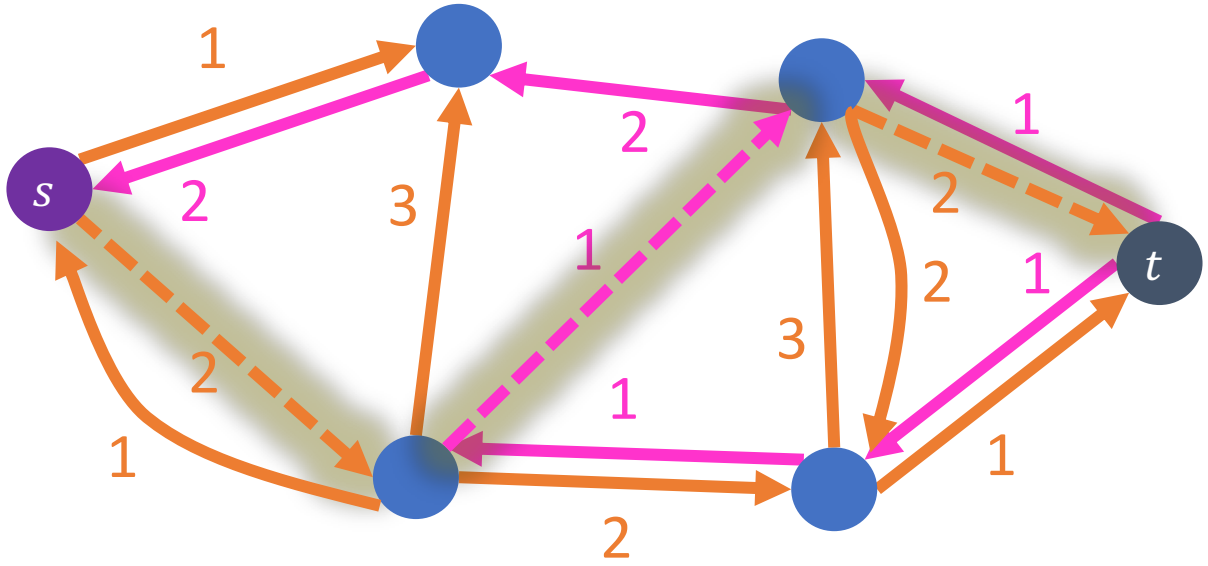


Residual graph G_f

Ford-Fulkerson Example

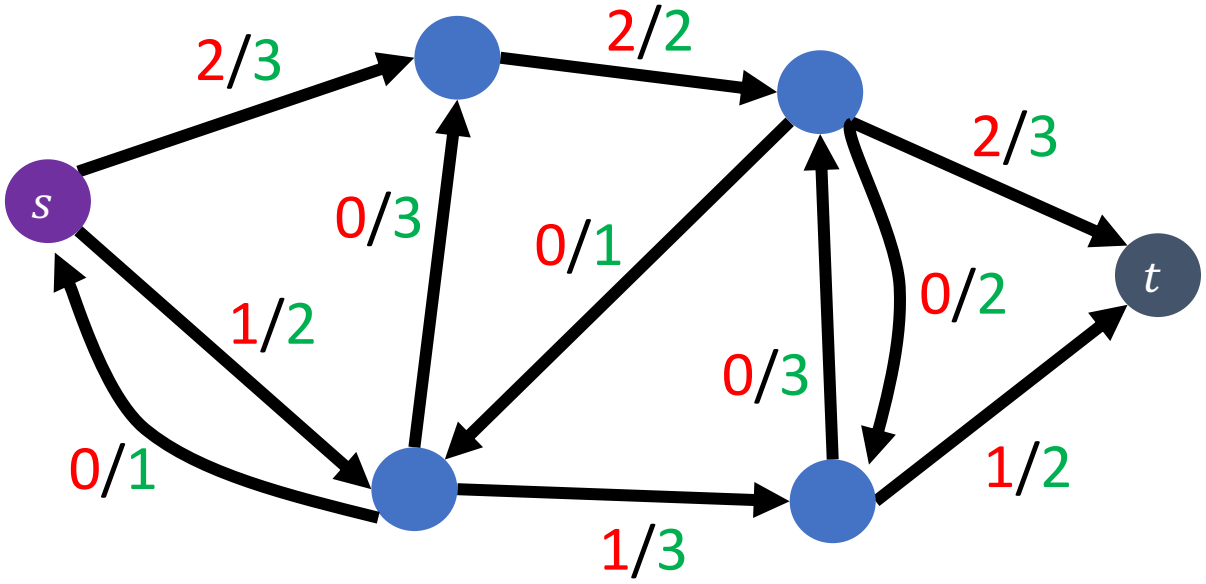


Increase flow by 1 unit

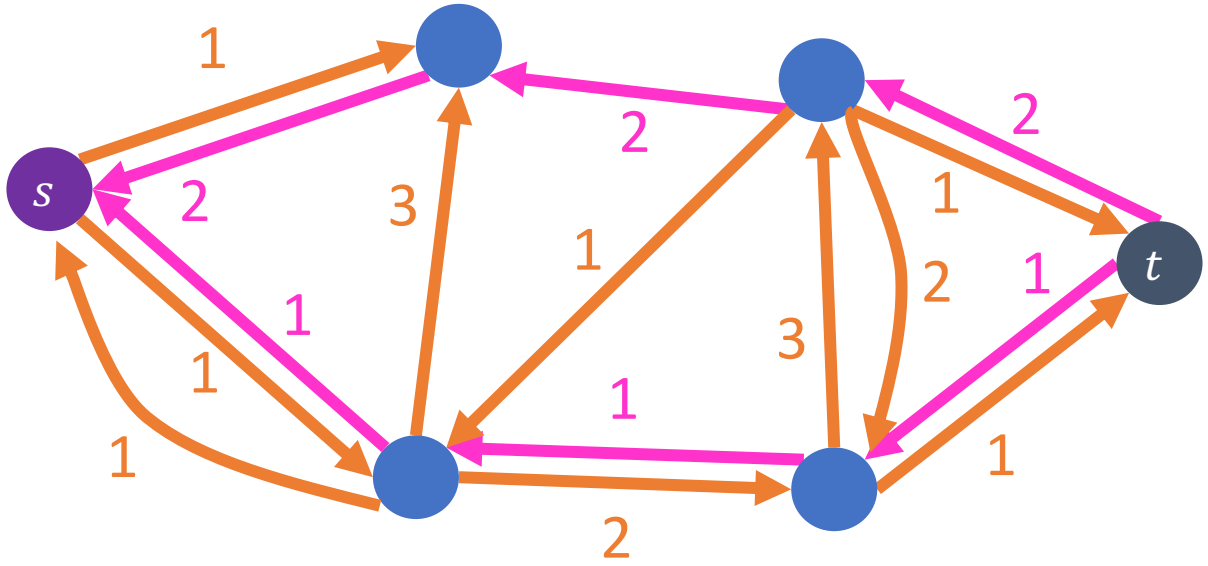


Residual graph G_f

Ford-Fulkerson Example

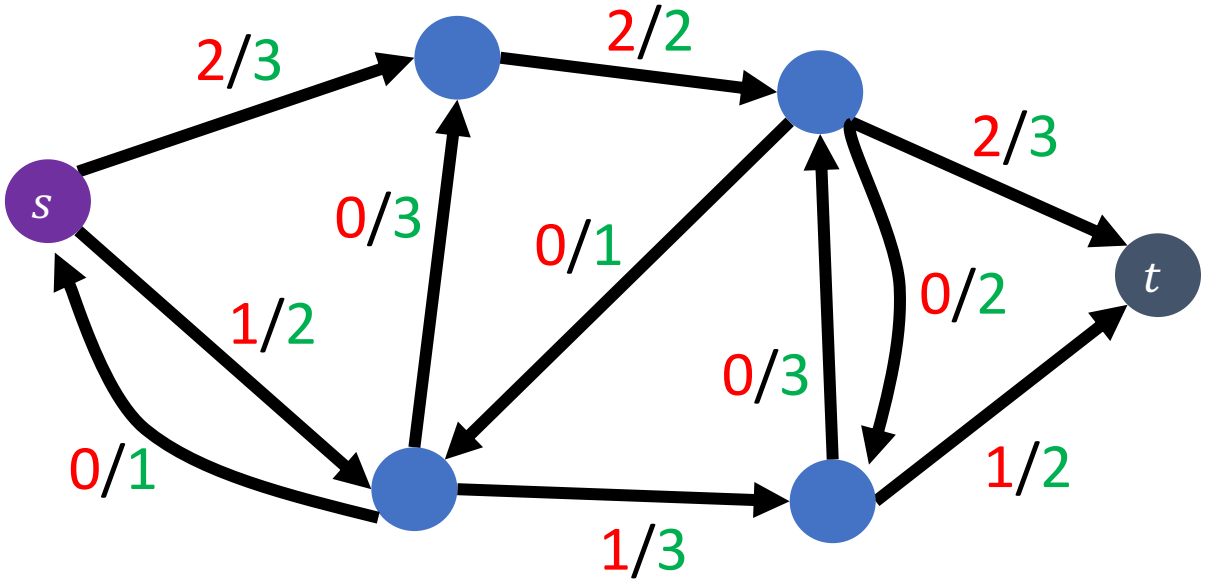


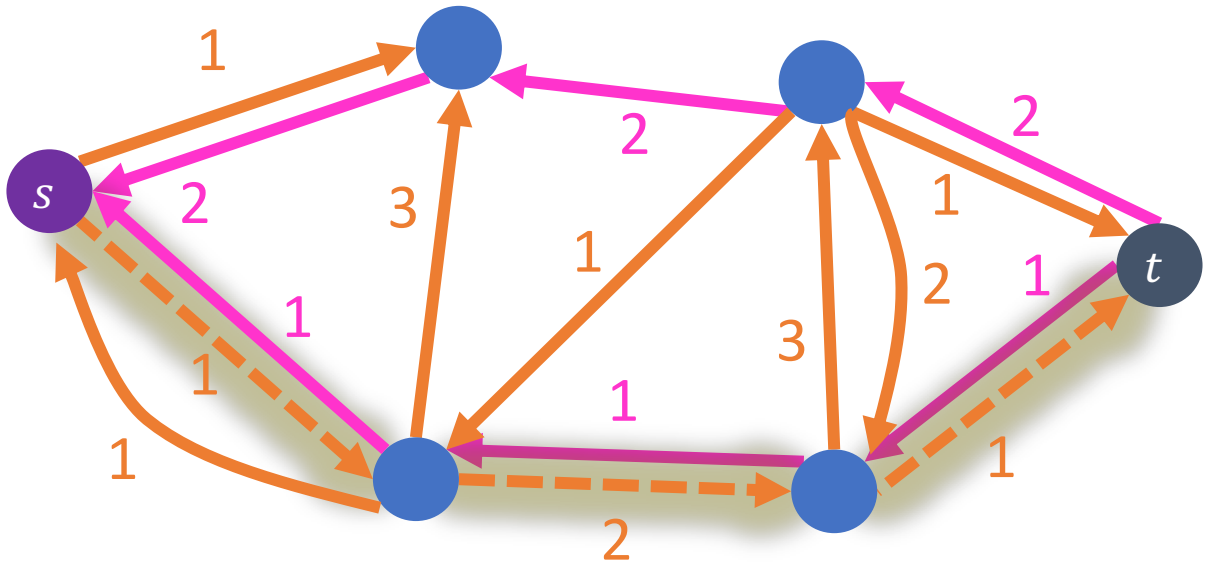
Increase flow by 1 unit



Residual graph G_f

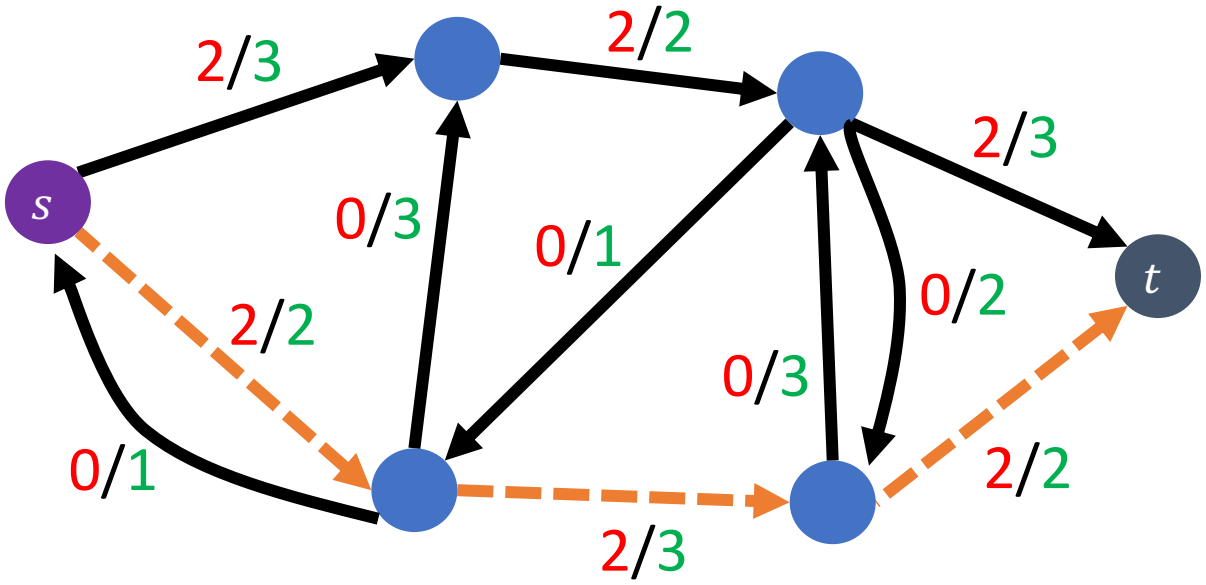
Ford-Fulkerson Example



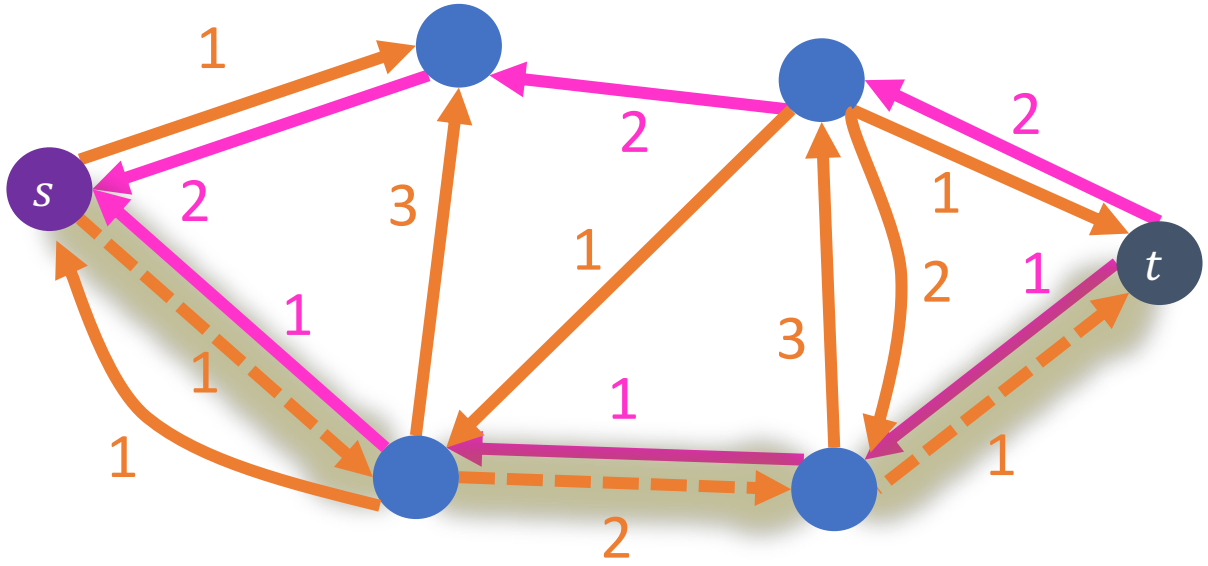


Residual graph G_f

Ford-Fulkerson Example

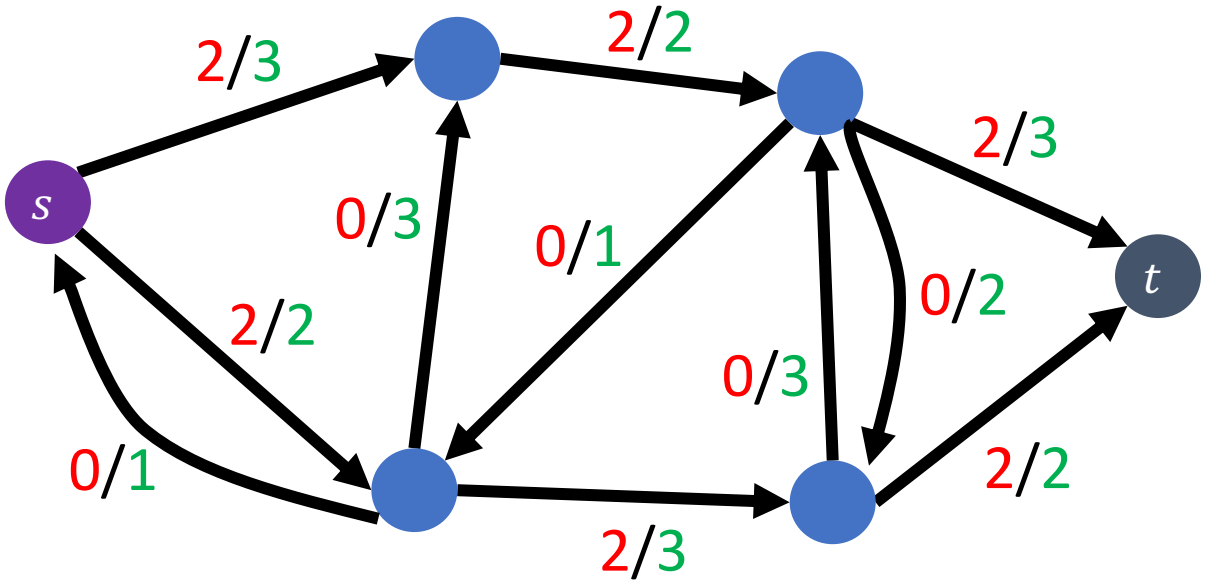


Increase flow by 1 unit



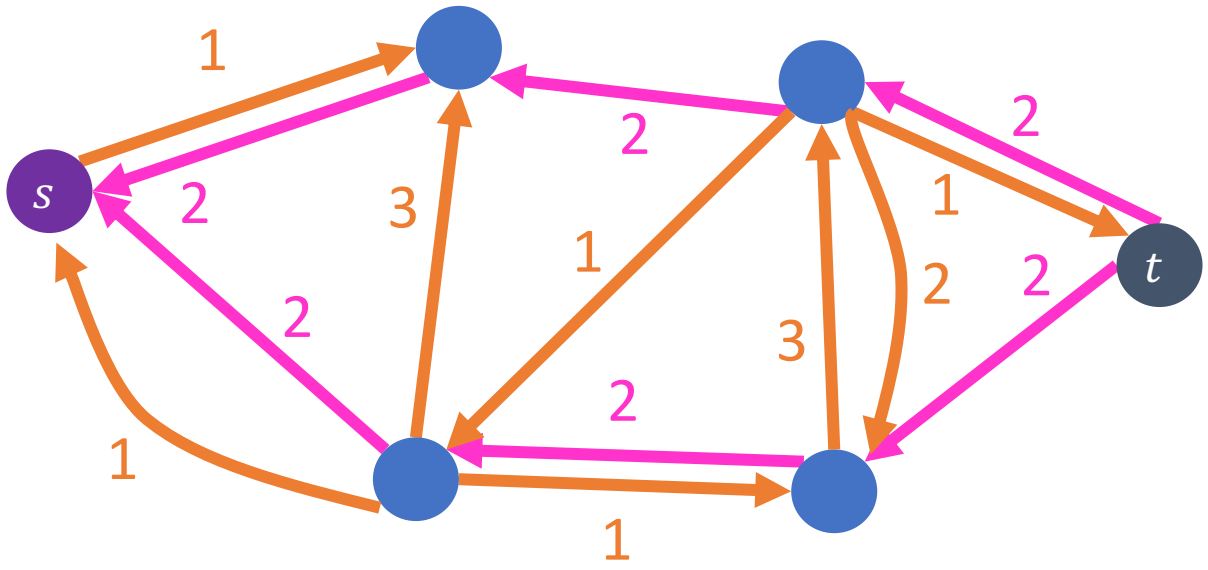
Residual graph G_f

Ford-Fulkerson Example



Maximum flow: 4

No more augmenting paths



Residual graph G_f

Ford-Fulkerson Running Time

Define an augmenting path to be an $s \rightarrow t$ path in the residual graph G_f (using edges of non-zero weight)

Ford-Fulkerson max-flow algorithm:

- Initialize $f(e) = 0$ for all $e \in E$
- Construct the residual network G_f
- While there is an augmenting path p in G_f:
  - Let $c = \min_{e \in E} c_f(e)$ ($c_f(e)$ is the weight of edge e in the residual network G_f)
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

Initialization: $O(|E|)$

Ford-Fulkerson Running Time

Define an augmenting path to be an $s \rightarrow t$ path in the residual graph G_f (using edges of non-zero weight)

Ford-Fulkerson max-flow algorithm:

- Initialize $f(e) = 0$ for all $e \in E$
- Construct the residual network G_f
- While there is an augmenting path p in G_f:
  - Let $c = \min_{e \in E} c_f(e)$ ($c_f(e)$ is the weight of edge e in the residual network G_f)
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

Initialization: $O(|E|)$

Construct residual network: $O(|E|)$

Ford-Fulkerson Running Time

Define an augmenting path to be an $s \to t$ path in the residual graph G_f (using edges of non-zero weight)

Ford-Fulkerson max-flow algorithm:

- Initialize $f(e) = 0$ for all $e \in E$
- Construct the residual network G_f
- While there is an augmenting path p in G_f:
  - Let $c = \min_{e \in E} c_f(e)$ ($c_f(e)$ is the weight of edge e in the residual network G_f)
  - Add c units of flow to G based on the augme
  - Update the residual network G_f for the upda

We only care about nodes reachable from the source s (so the number of nodes that are "relevant" is at most $|E|$)

Initialization: $O(|E|)$

Construct residual network: $O(|E|)$

Finding augmenting path in residual network: $O(|E|)$ using BFS/DFS

Ford-Fulkerson Running Time

Define an augmenting path to be an $s \to t$ path in the residual graph G_f (using edges of non-zero weight)

How many iterations are needed?

- For integer-valued capacities, min-weight of each augmenting path is 1, so number of iterations is bounded by $|f^*|$, where $|f^*|$ is max-flow in G
- For rational-valued capacities, can scale to make capacities integer
- For irrational-valued capacities, algorithm may never terminate!
  - **Extra credit:** Construct a graph where this happens

Initialization: $O(|E|)$

Construct residual network: $O(|E|)$

Finding augmenting path in residual network: $O(|E|)$ using BFS/DFS

Ford-Fulkerson Running Time

Define an augmenting path to be an $s \to t$ path in the residual graph G_f (using edges of non-zero weight)

Ford-Fulkerson max-flow algorith

- Initialize $f(e) = 0$ for all e
- Construct the residual net
- While there is an augmen
  - Let $c = \min_{e \in E} c_f(e)$ (c_f(
  - Add c units of flow to
  - Update the residual n

Initialization: $O(|E|)$

Construct residual network: $O(|E|)$

Finding augmenting path in residual network: $O(|E|)$ using BFS/DFS

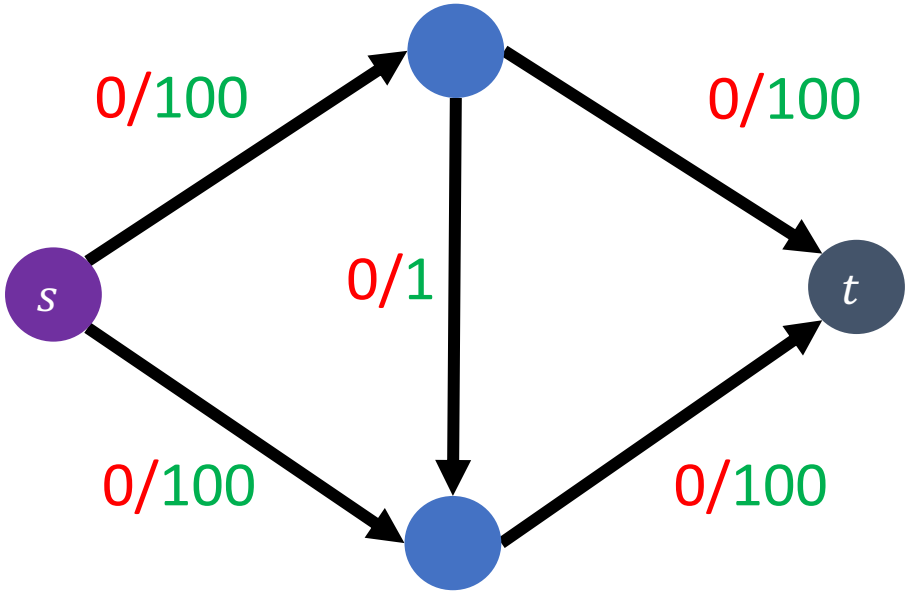
For graphs with integer capacities, running time of Ford-Fulkerson is

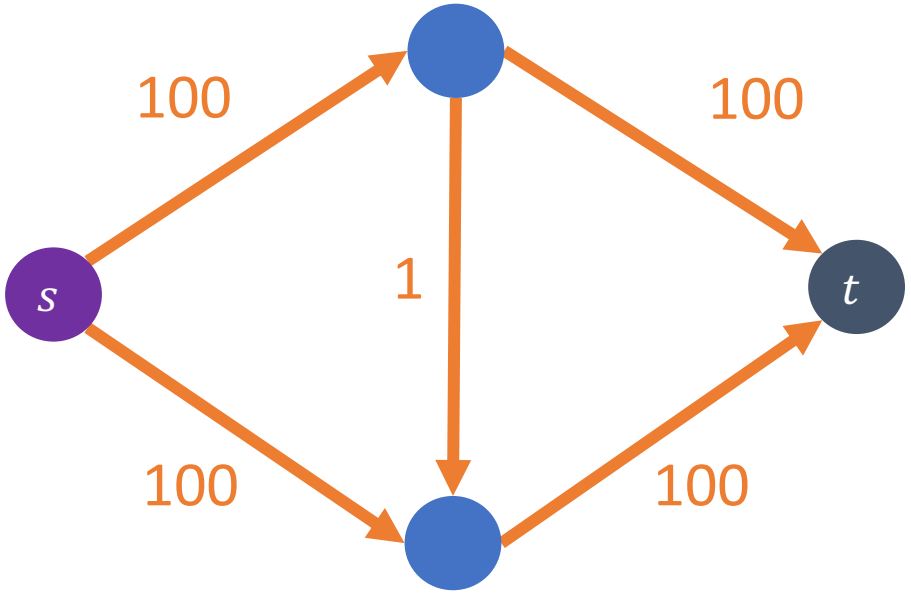
$$O(|f^*| \cdot |E|)$$

Highly undesirable if $|f^*| \gg |E|$ (e.g., graph is small, but capacities are $\approx 2^{32}$)

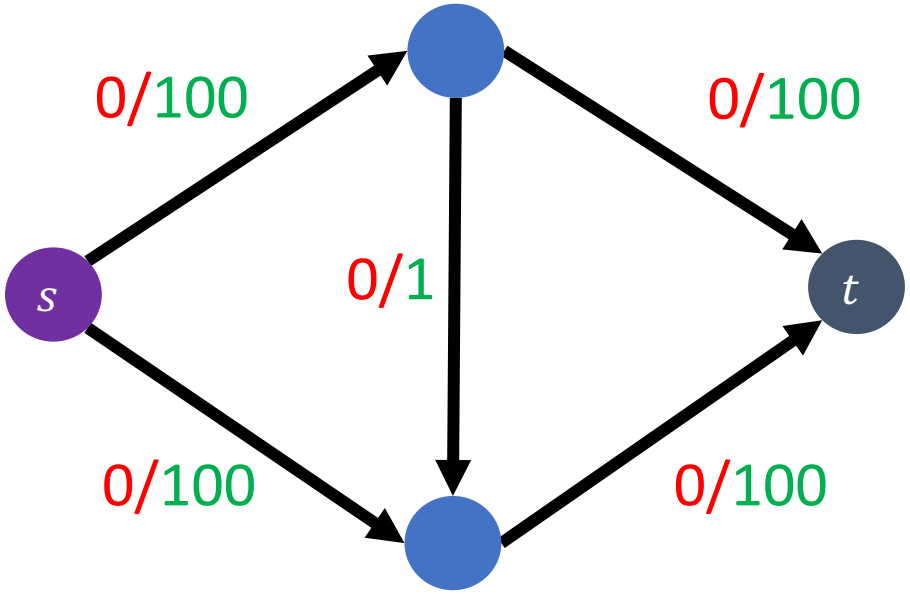
As described, algorithm is <u>not</u> polynomial-time!

Worst-Case Ford-Fulkerson

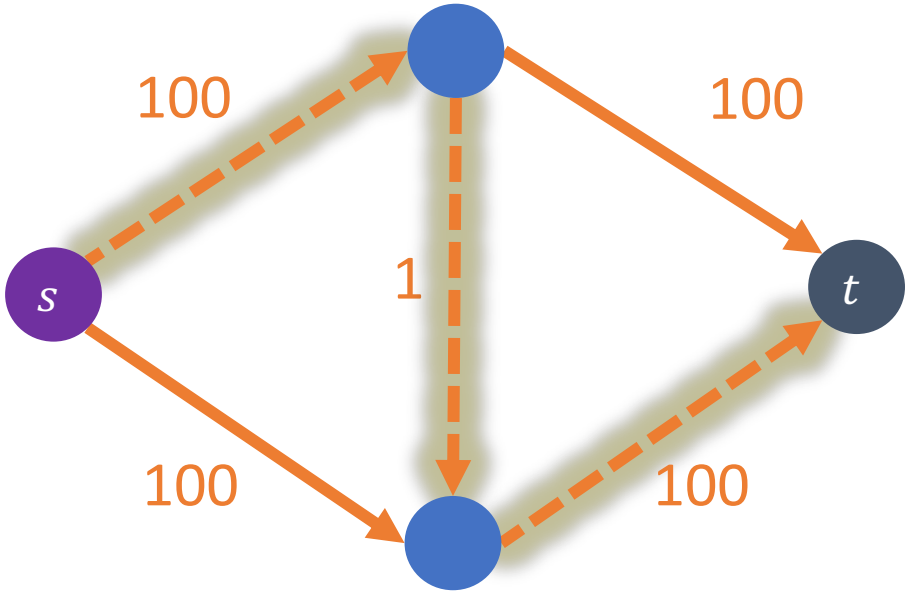




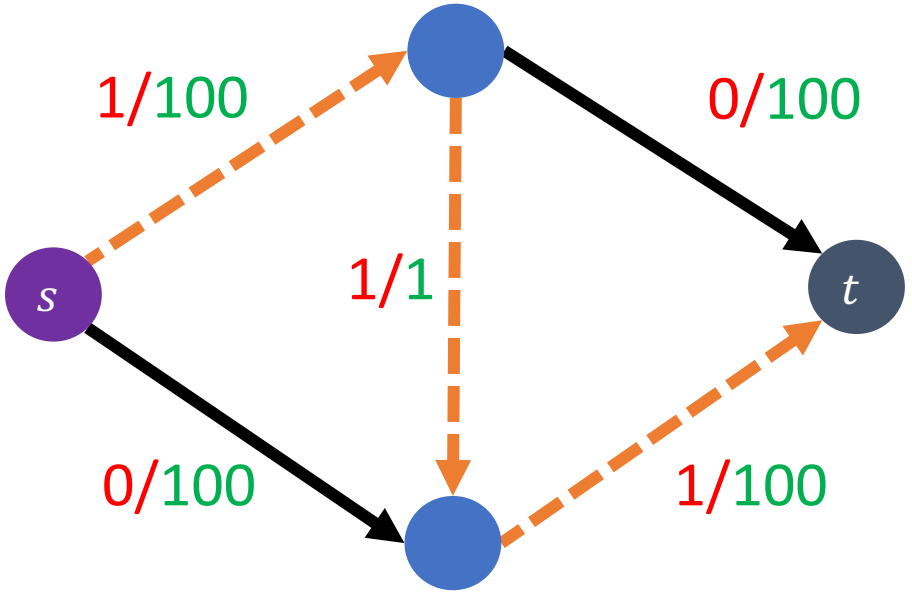
Worst-Case Ford-Fulkerson



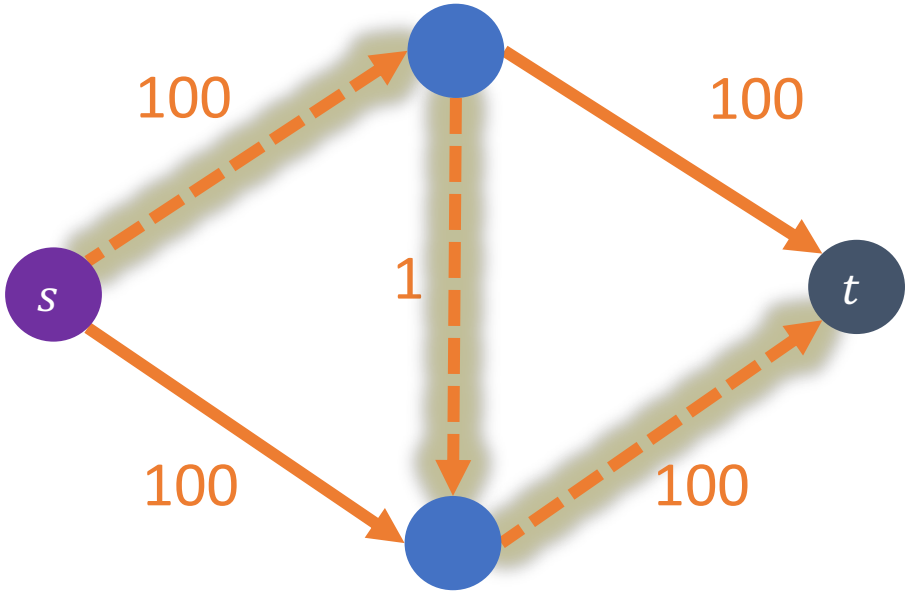
Increase flow by 1 unit



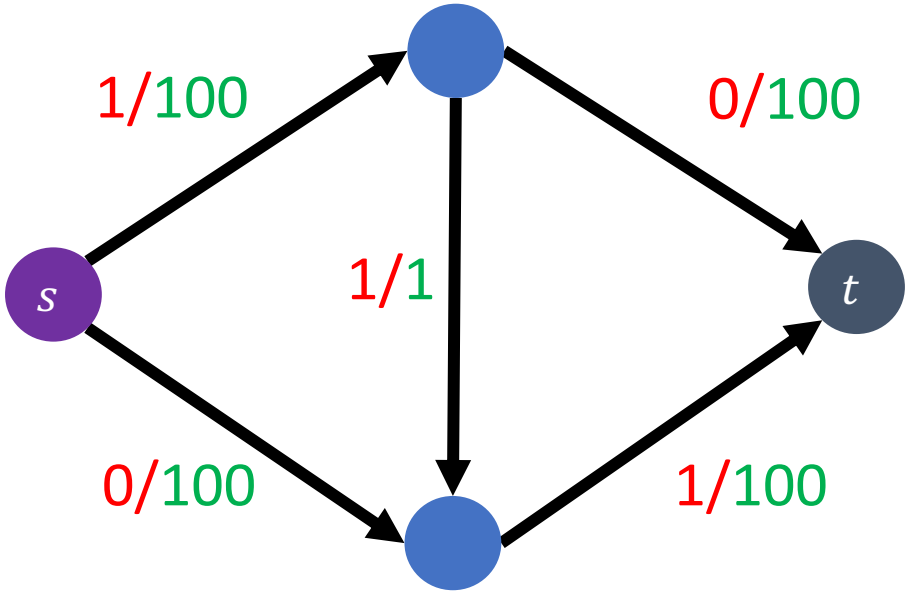
Worst-Case Ford-Fulkerson

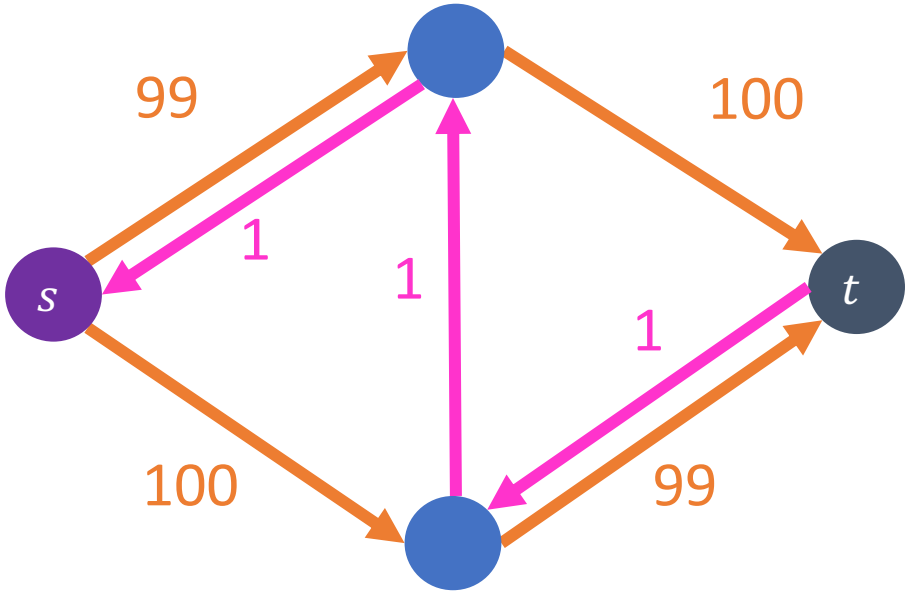


Increase flow by 1 unit

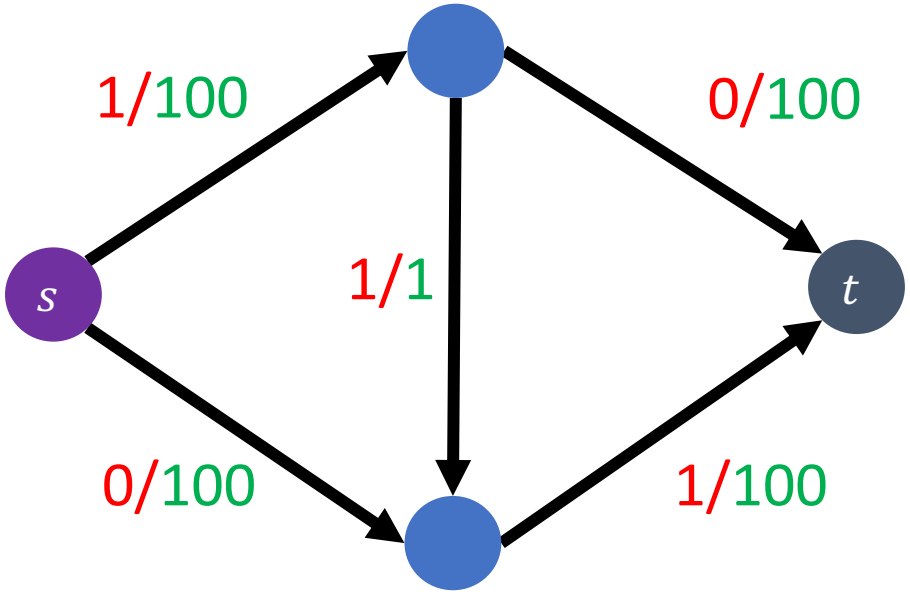


Worst-Case Ford-Fulkerson

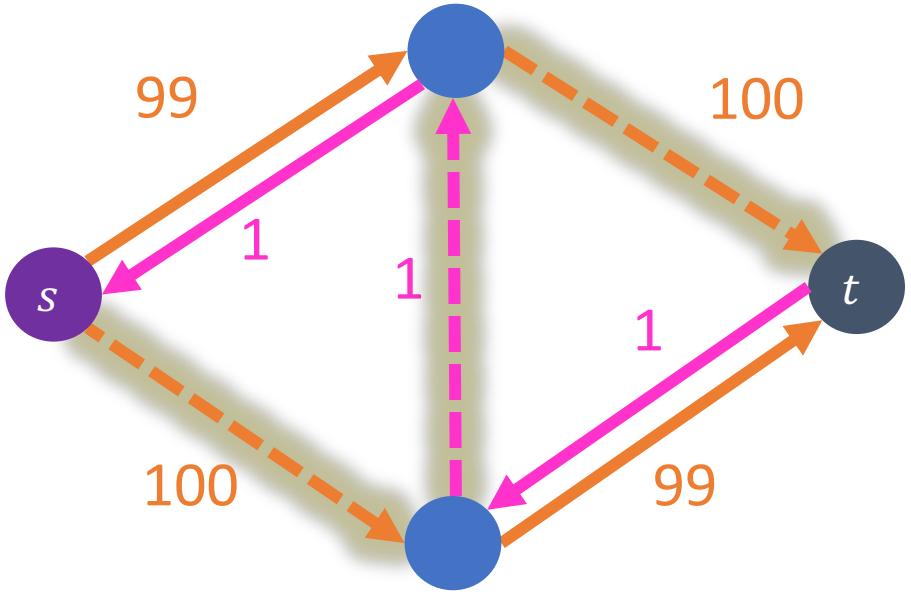




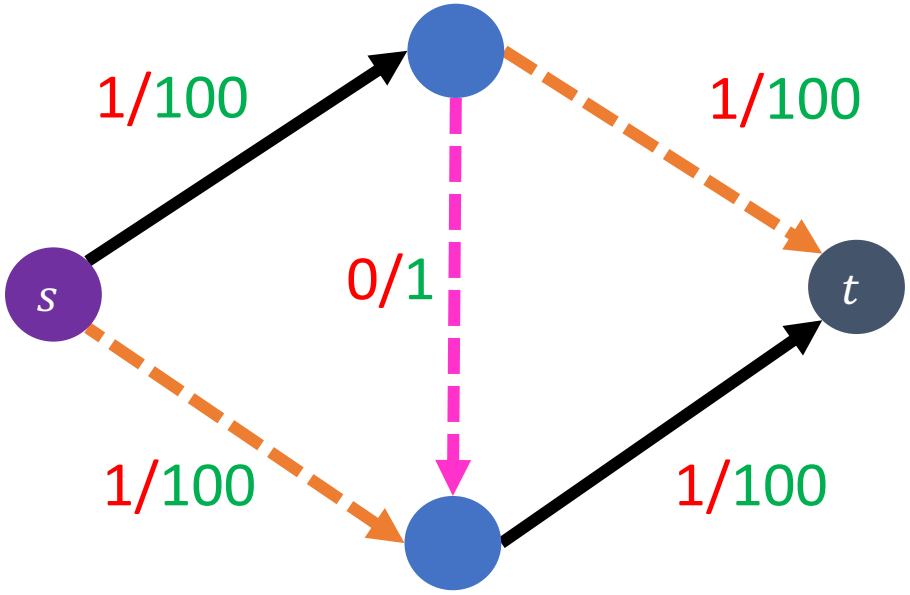
Worst-Case Ford-Fulkerson



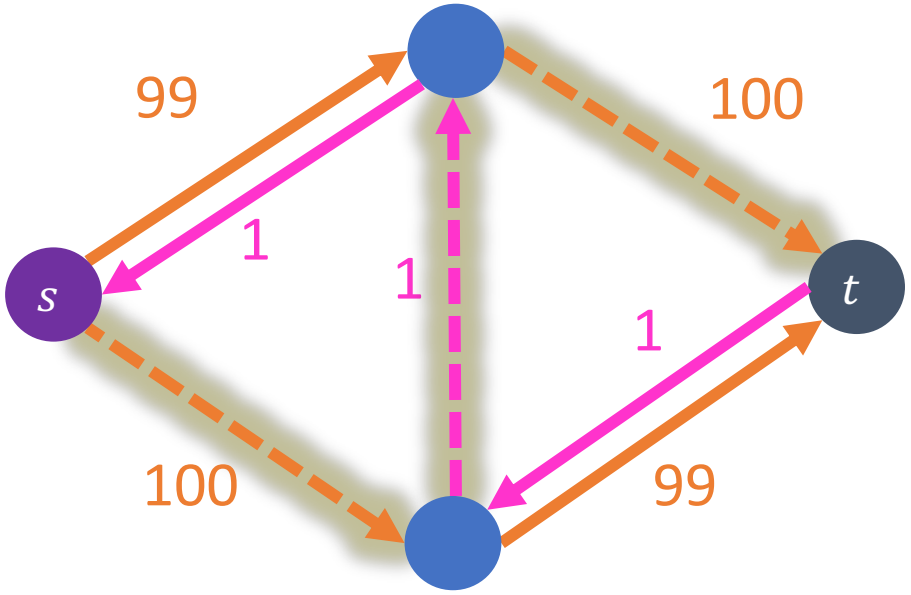
Increase flow by 1 unit



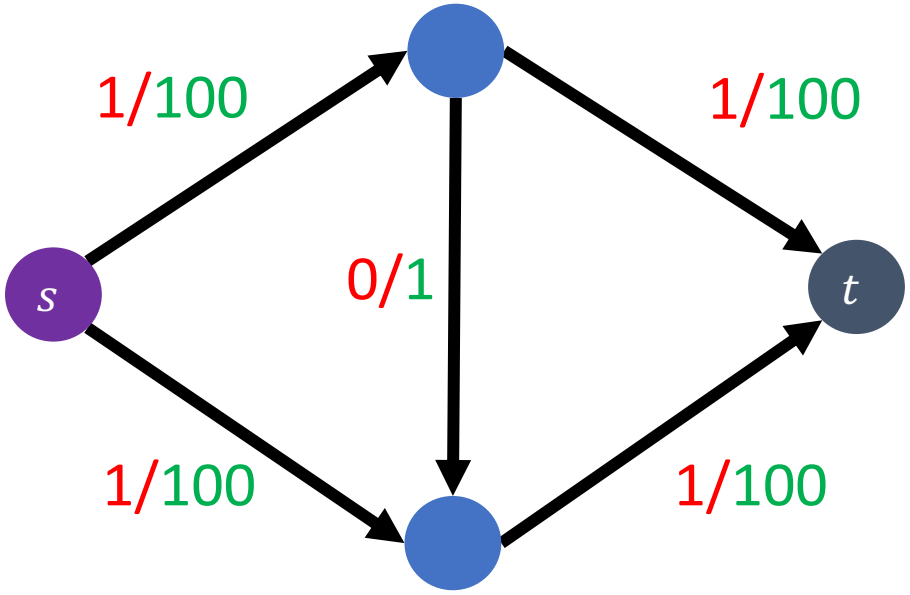
Worst-Case Ford-Fulkerson

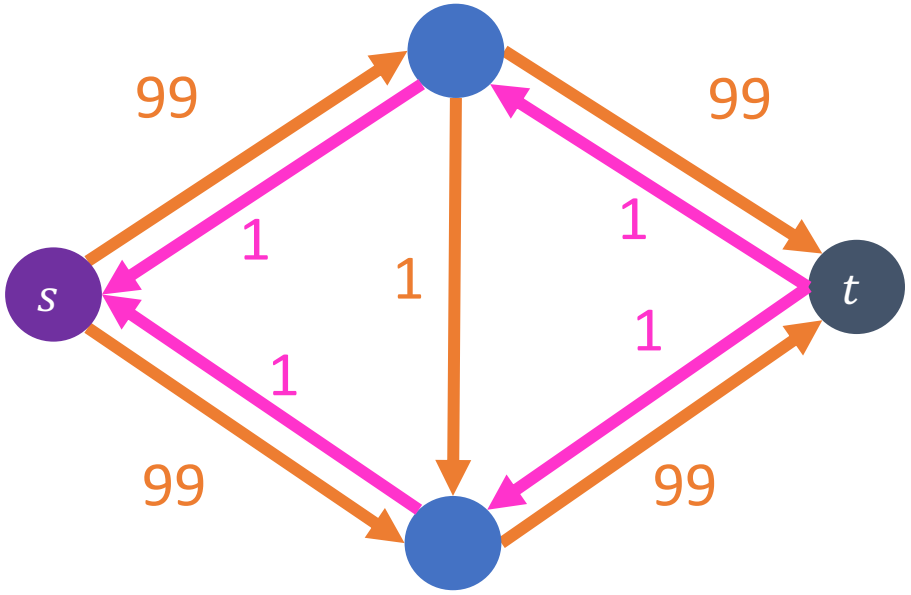


Increase flow by 1 unit

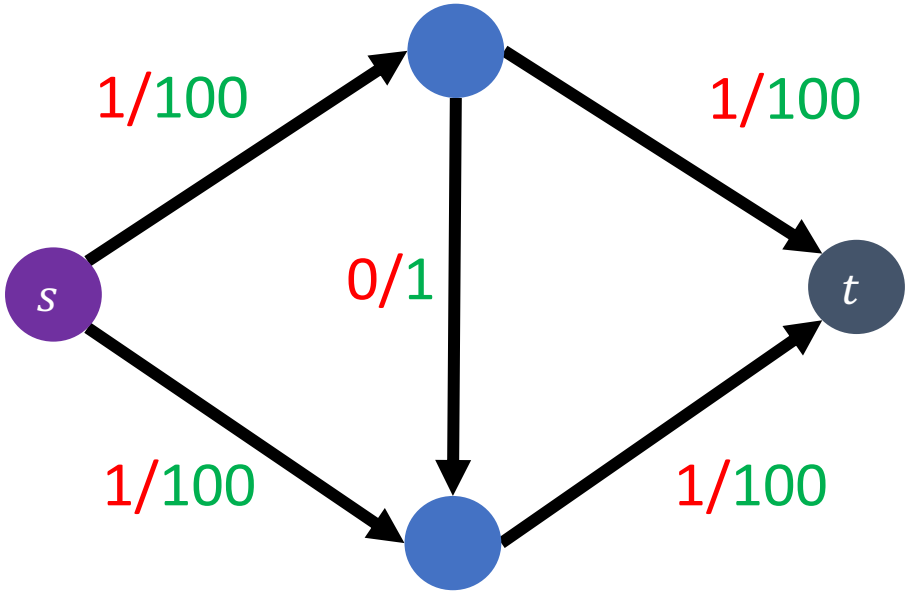


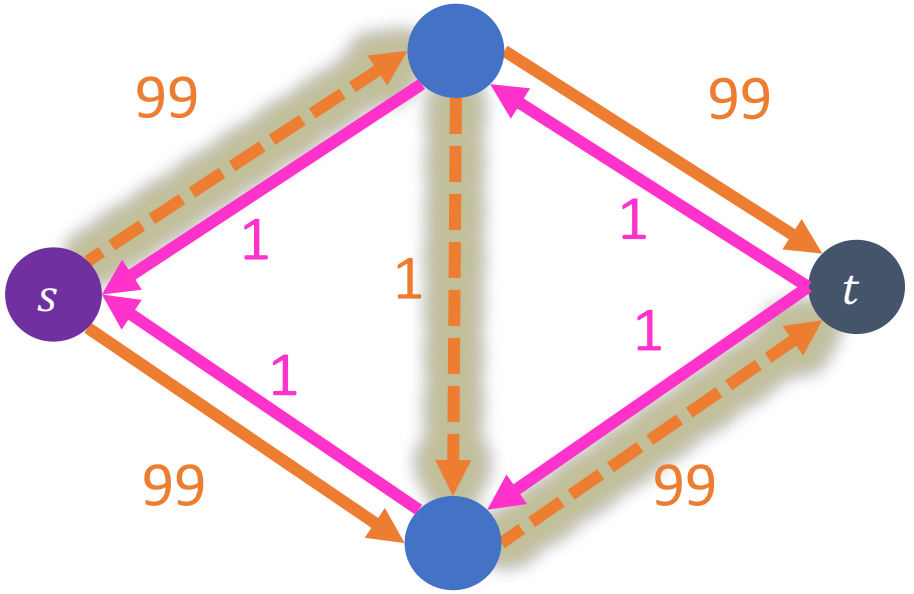
Worst-Case Ford-Fulkerson





Worst-Case Ford-Fulkerson





Observation: each iteration increases flow by 1 unit

Total number of iterations: $|f^*| = 200$

Can We Avoid this?

Edmonds-Karp Algorithm: choose augmenting path with fewest hops

Running time: $\Theta(\min(|E||f^*|, |V||E|^2)) = O(|V||E|^2)$

How to find this?
Use breadth-first search (BFS)!

Edmonds-Karp = Ford-Fulkerson using BFS to find augmenting path

Ford-Fulkerson max-flow algorithm:

- Initialize $f(e) = 0$ for all $e \in E$
- Construct the residual network G_f
- While there is an augmenting path in G_f, let p be the path with fewest hops:
  - Let $c = \min_{e \in E} c_f(e)$ ($c_f(e)$ is the weight of edge e in the residual network G_f)
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

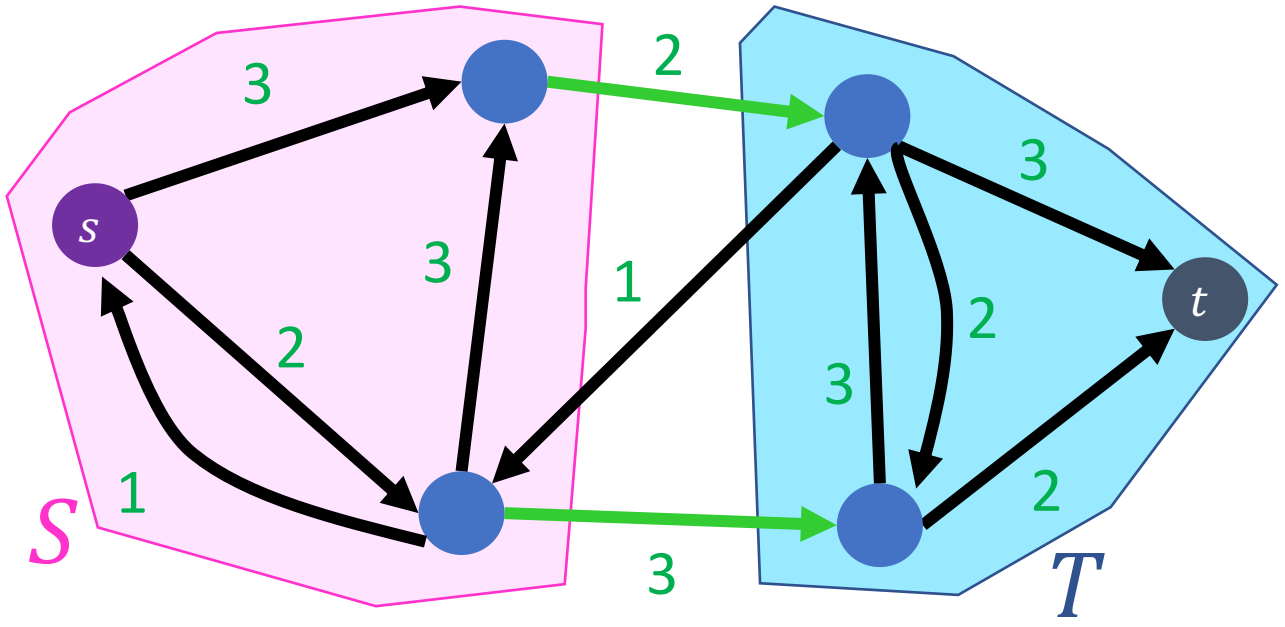
Proof: See CLRS (Chapter 26.2)

Correctness of Ford-Fulkerson

Consider cuts which separate s and t

- Let $s \in S$, $t \in T$, such that $V = S \cup T$

Cost $\|S, T\|$ of cut (S, T): sum of the **capacities** of edges from S to T



$\|S, T\| = 5$

Max-Flow / Min-Cut

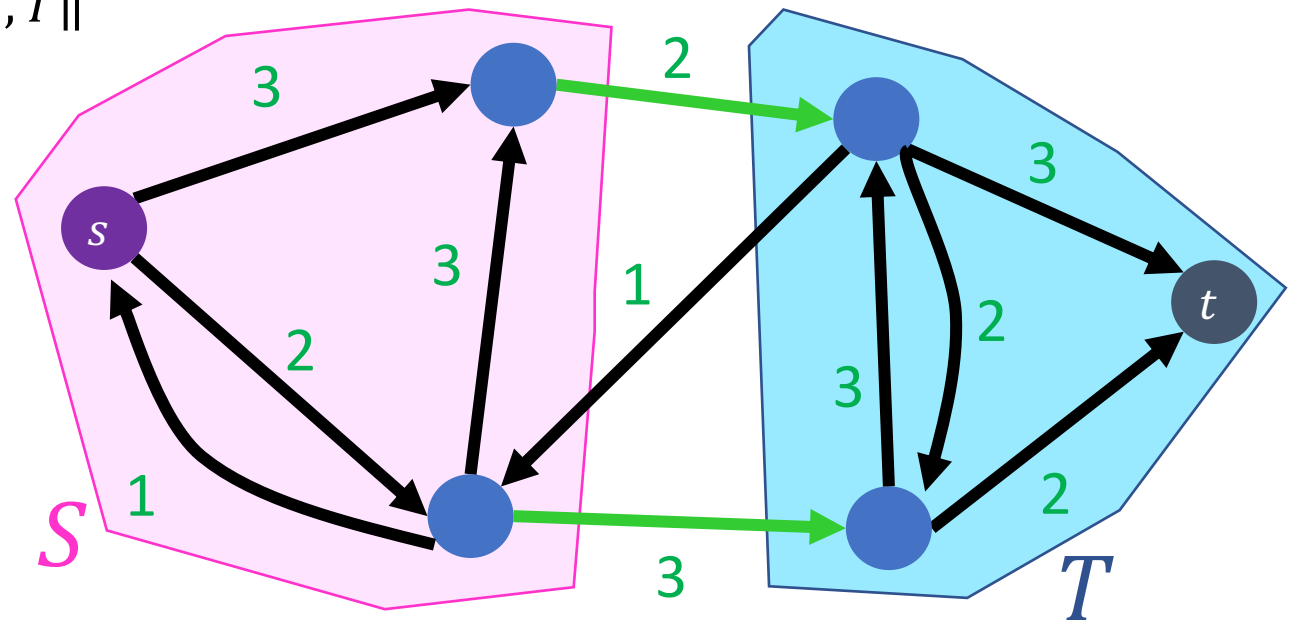
Claim: Maximum flow in a flow network G always upper-bounded by the cost any cut that separates s and t

Proof: "Conservation of flow"

- All flow from s must eventually get to t
- To get from s to t, all flow must cross the cut somewhere

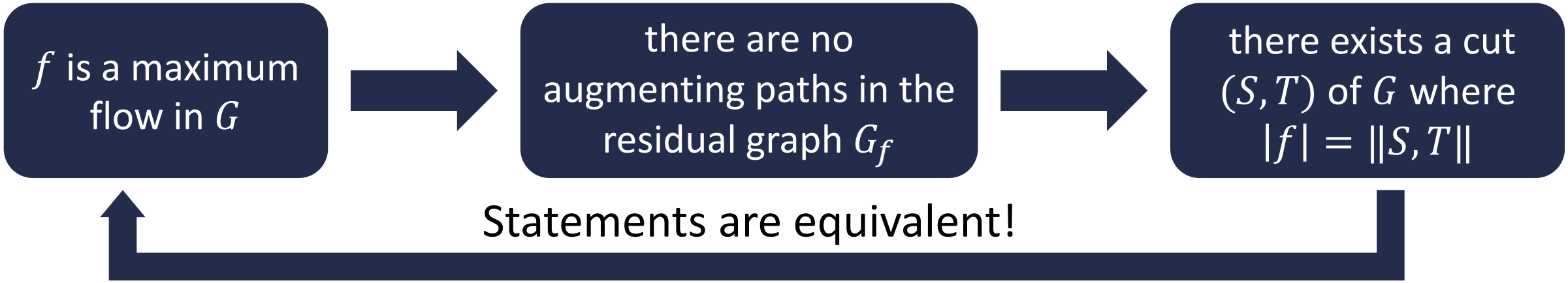
Conclusion: Max-flow in G is <u>at most</u> the cost of the min-cut in G

- $\max_f |f| \leq \min_{S,T} \|S,T\|$



Max-Flow Min-Cut Theorem

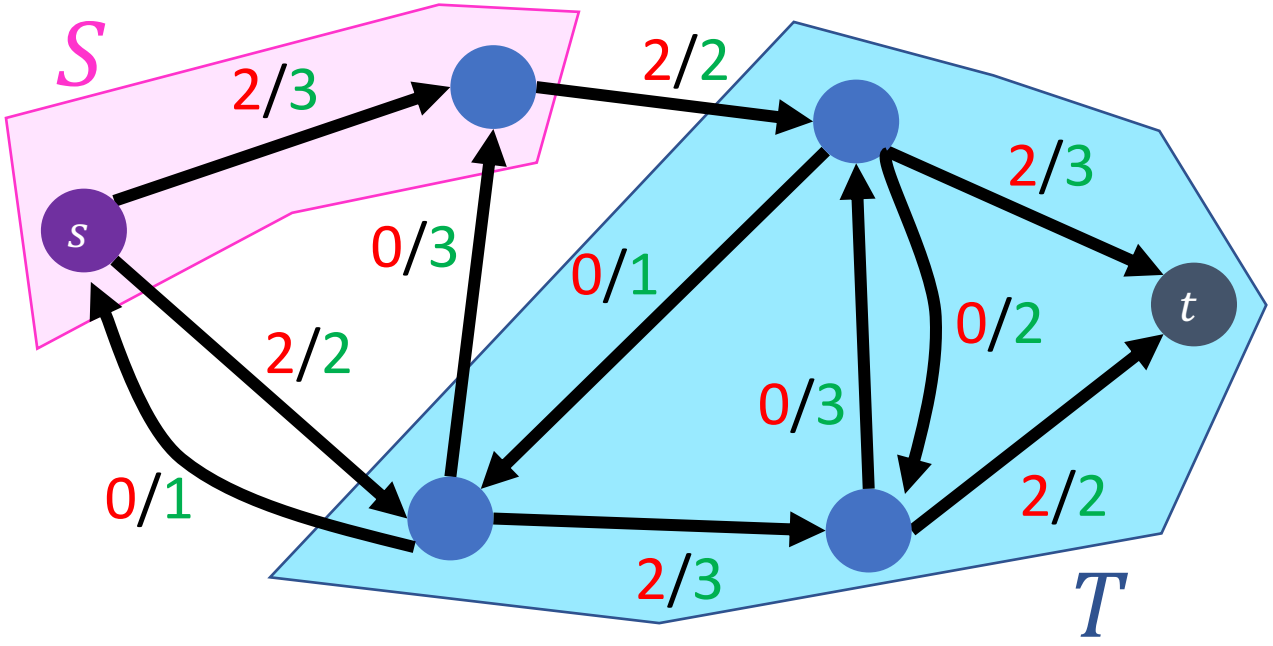
Let f be a flow in a graph G



Implications:

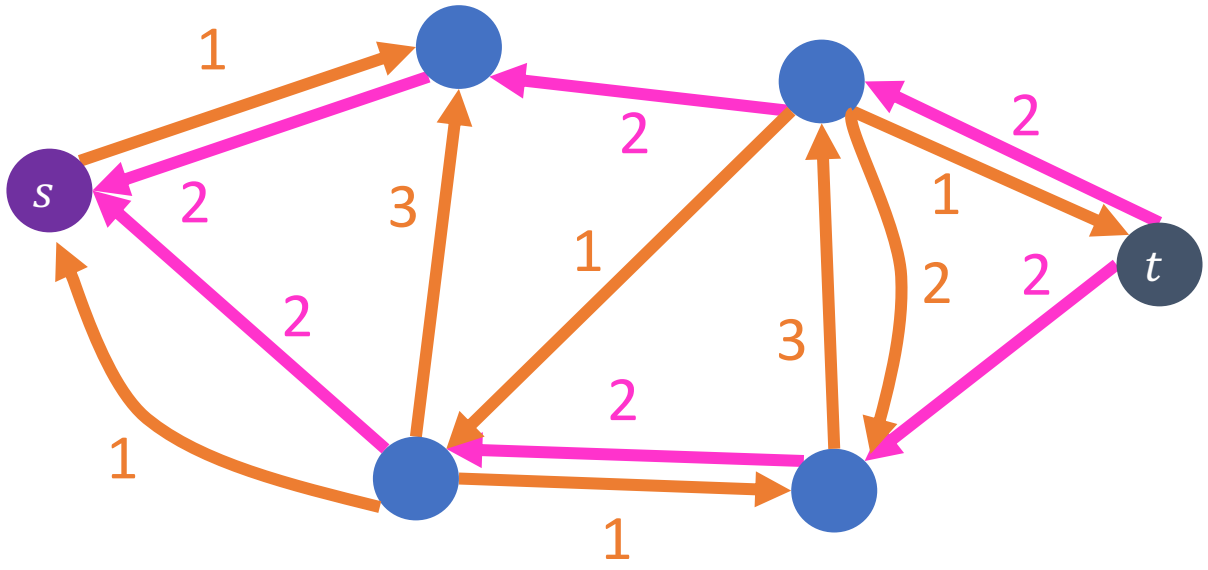
- **Correctness of Ford-Fulkerson:** Ford-Fulkerson terminates when there are no more augmenting paths in the residual graph G_f, which means that f is a maximum flow
- **Max-flow min-cut duality:** the maximum flow in a network coincides with the minimum cut of the graph ($\max_f |f| = \min_{S,T} \|S, T\|$)
  - Finding either the minimum cut or the maximum flow yields solution to the other
  - Special case of more general principle (duality in linear programming)

Max-Flow Min-Cut Duality Example



Flow graph

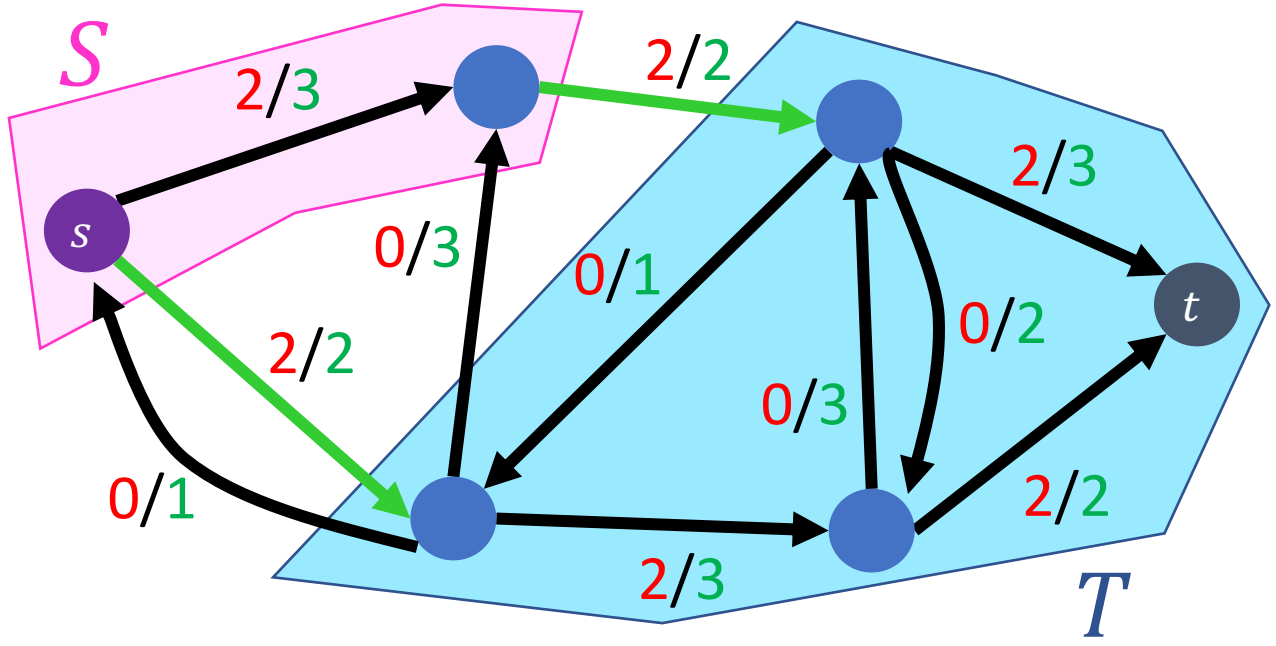
no more augmenting paths



Residual graph

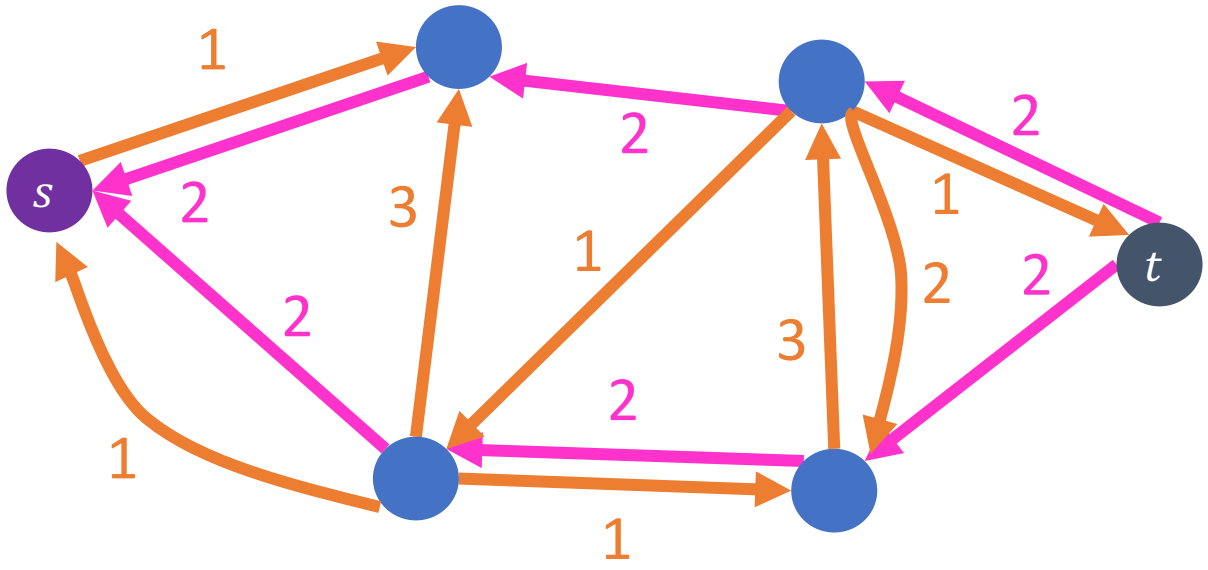
Max flow: 4

Max-Flow Min-Cut Duality Example



Flow graph

no more augmenting paths



Residual graph

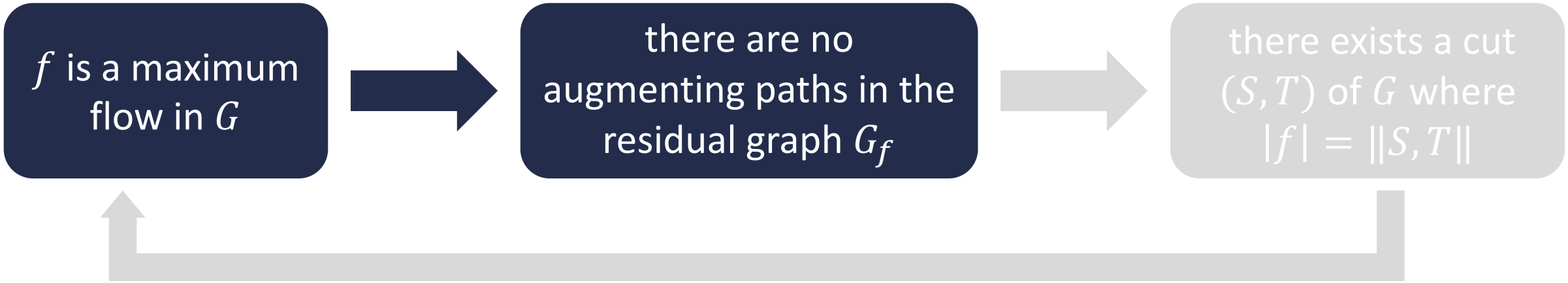
Max flow: 4

Min cut: 4

When there are no more augmenting paths in the graph, there is a cut whose cost matches the flow

Max-Flow Min-Cut Theorem Proof

Let f be a flow in a graph G

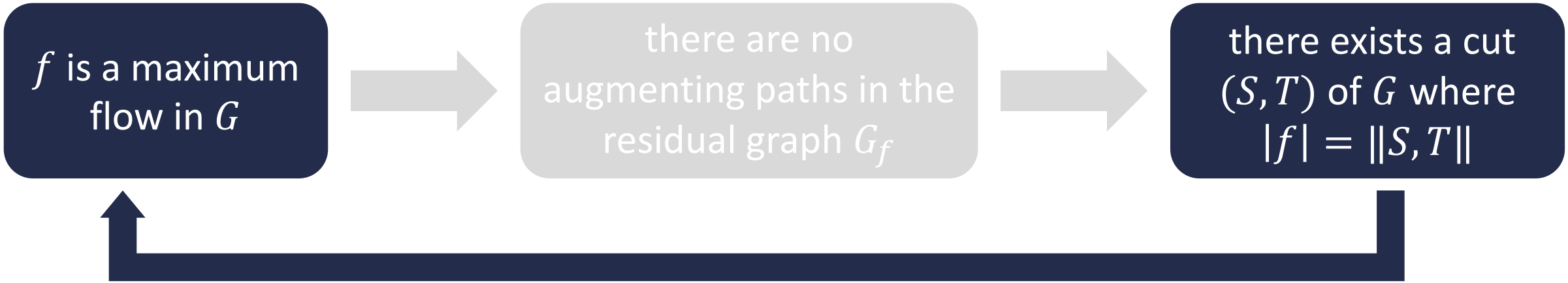


Proof:

- Suppose f is a max flow in G and there is an augmenting path in G_f
- If there is an augmenting path in G_f, then we can send additional units of flow though the network along the augmenting path
- This contradicts optimality of f

Max-Flow Min-Cut Theorem Proof

Let f be a flow in a graph G

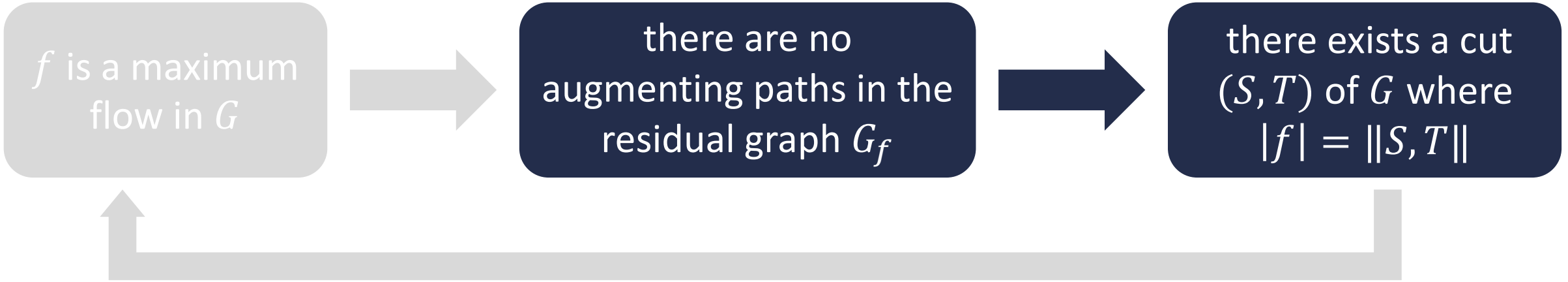


Proof:

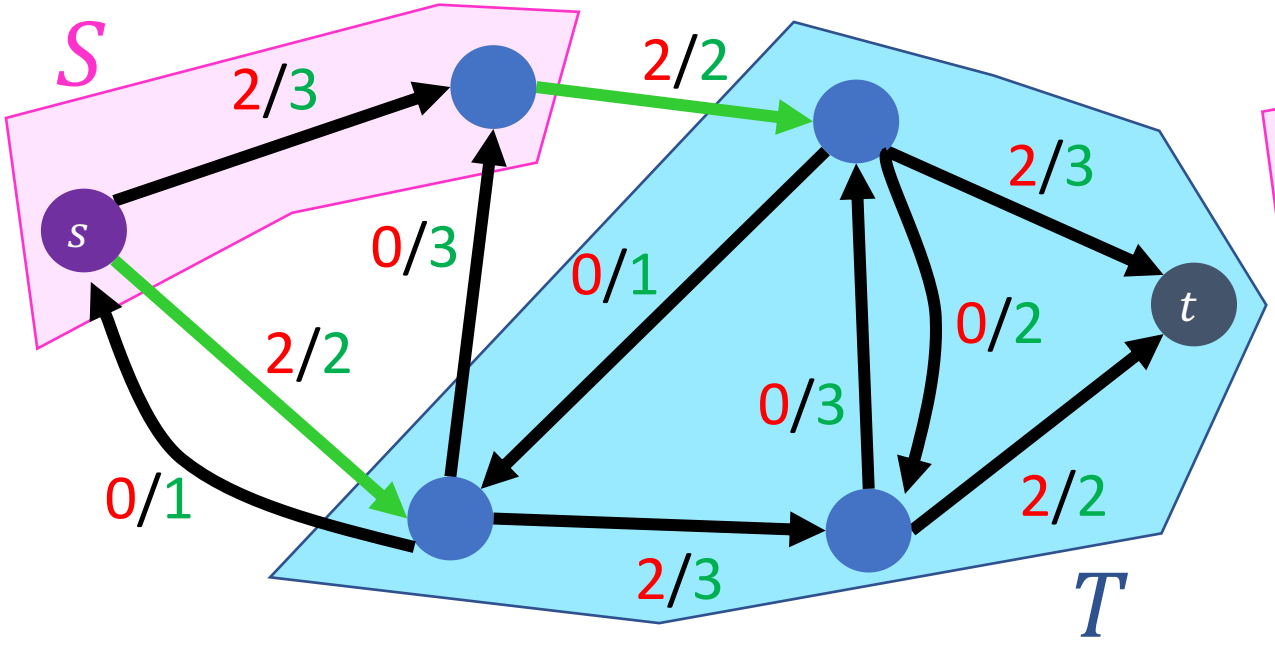
- Take any flow f'
- Consider the cut (S, T) of G; then, $|f'| \leq \|S, T\| = |f|$
- Thus, $|f'| \leq |f|$, so f must be a maximum flow

Max-Flow Min-Cut Theorem Proof

Let f be a flow in a graph G

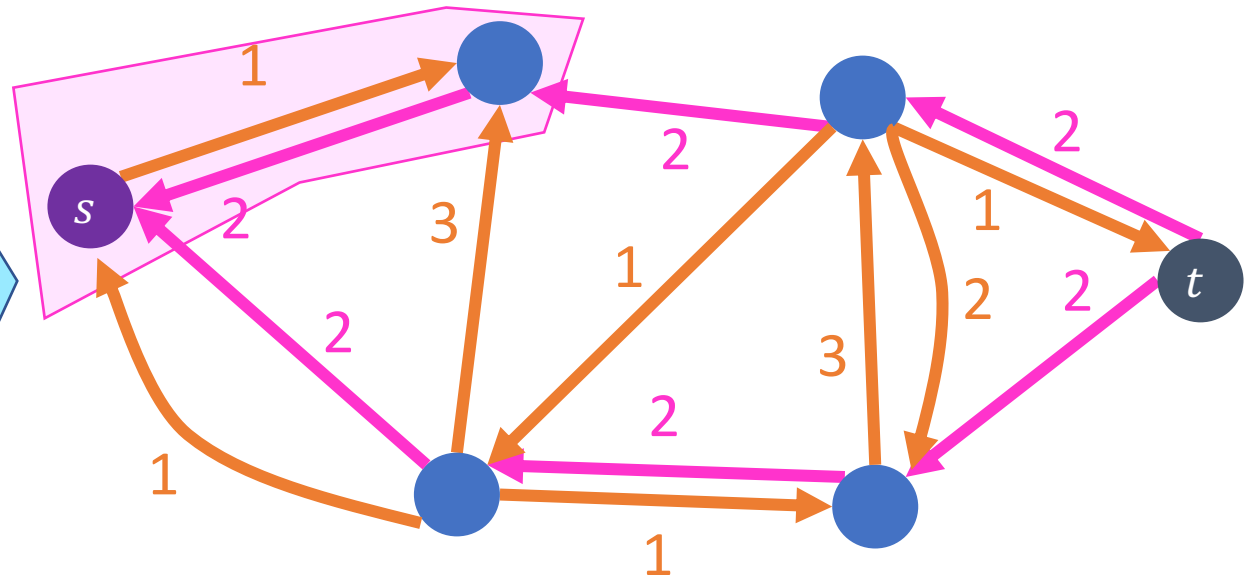


Max-Flow Min-Cut Theorem Proof



Flow graph

no more augmenting paths

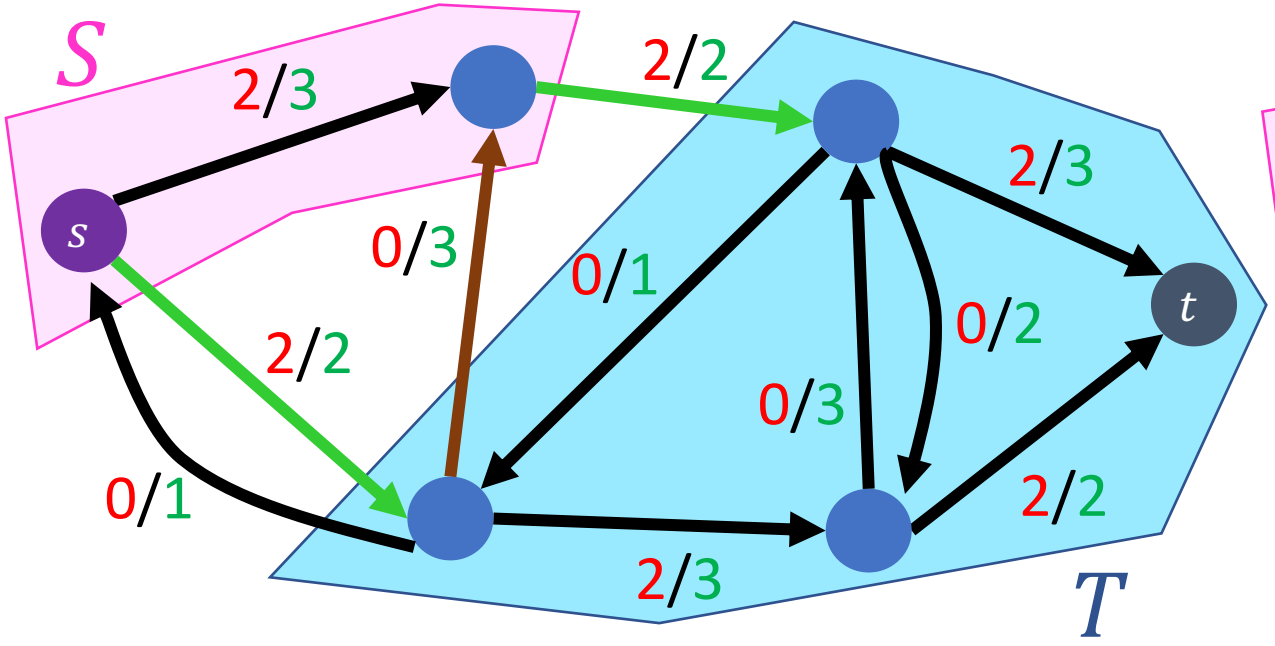


Residual graph

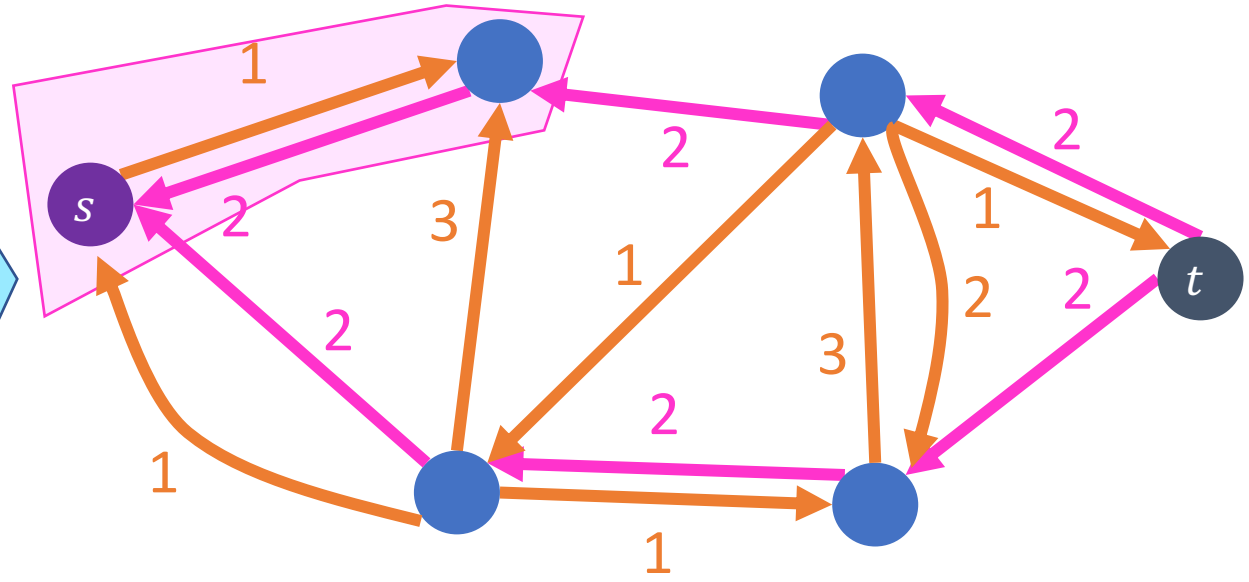
No augmenting paths means there is no path from s to t in G_f

- Let S be set of nodes reachable from s in G_f
- Let $T = V - S$

Max-Flow Min-Cut Theorem Proof



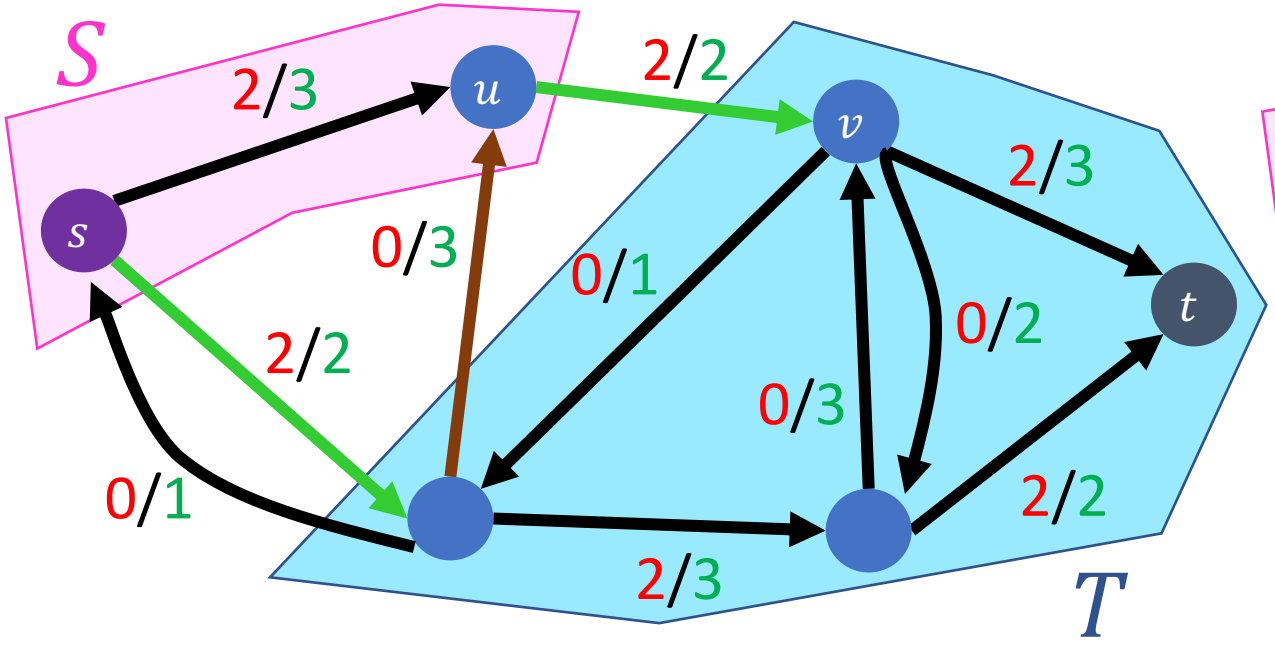
no more augmenting paths



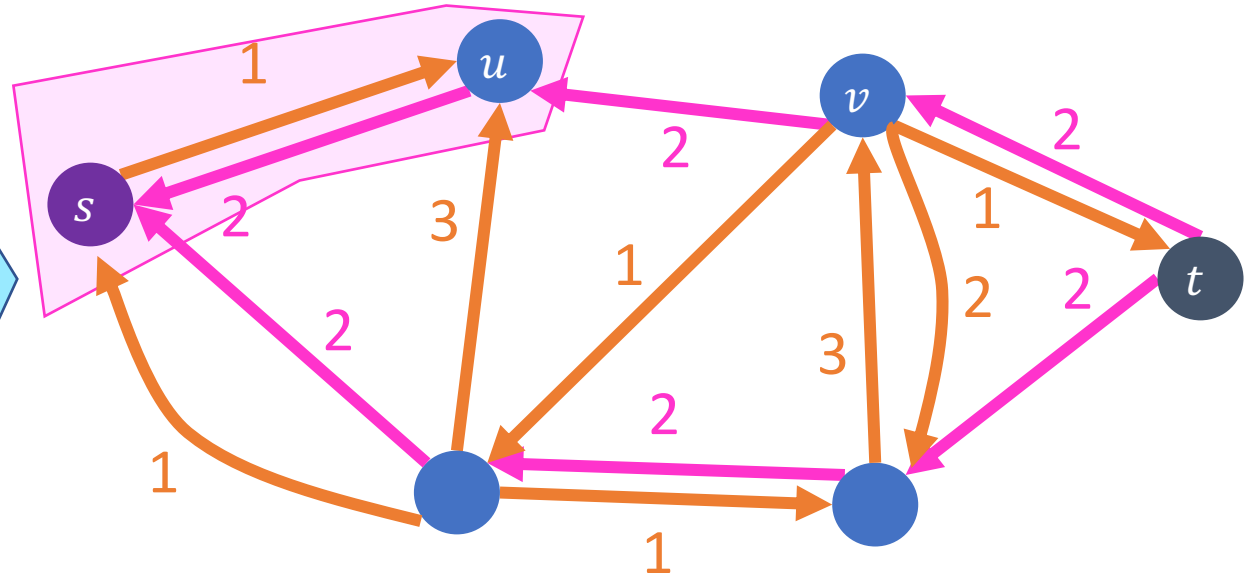
Claim: $\|S, T\| = |f|$

- Total flow $|f|$ is amount of outgoing flow from S to T minus the amount of incoming flow from T to S

Max-Flow Min-Cut Theorem Proof



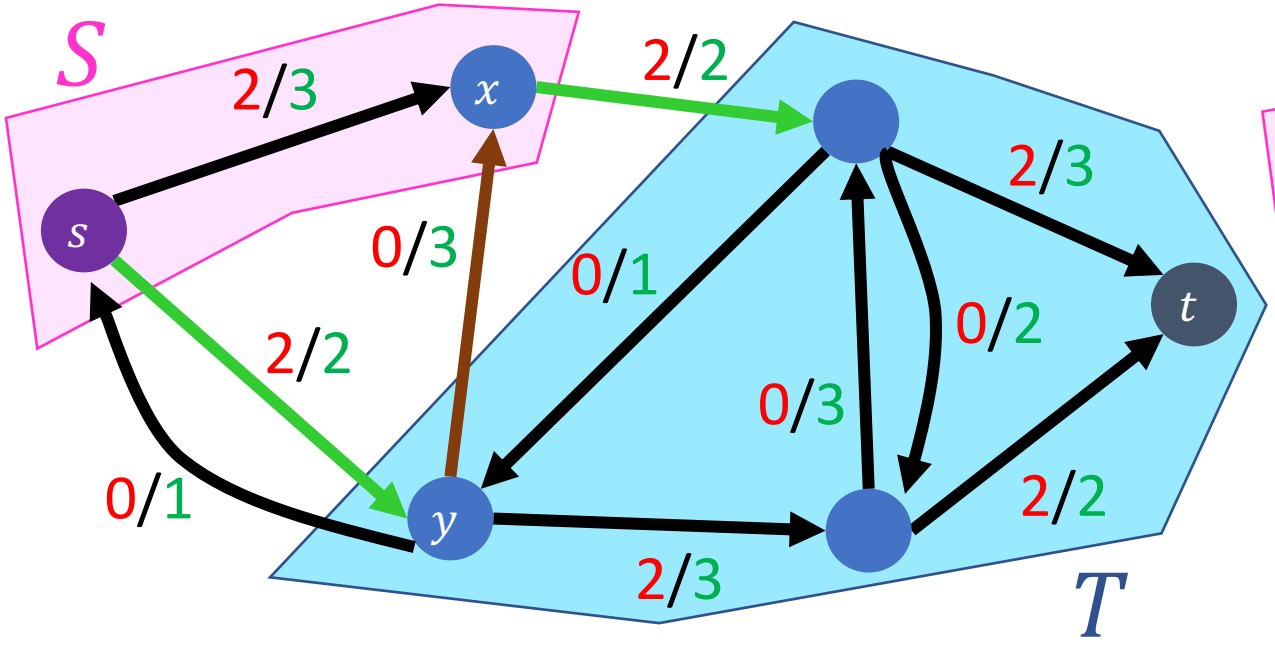
no more augmenting paths



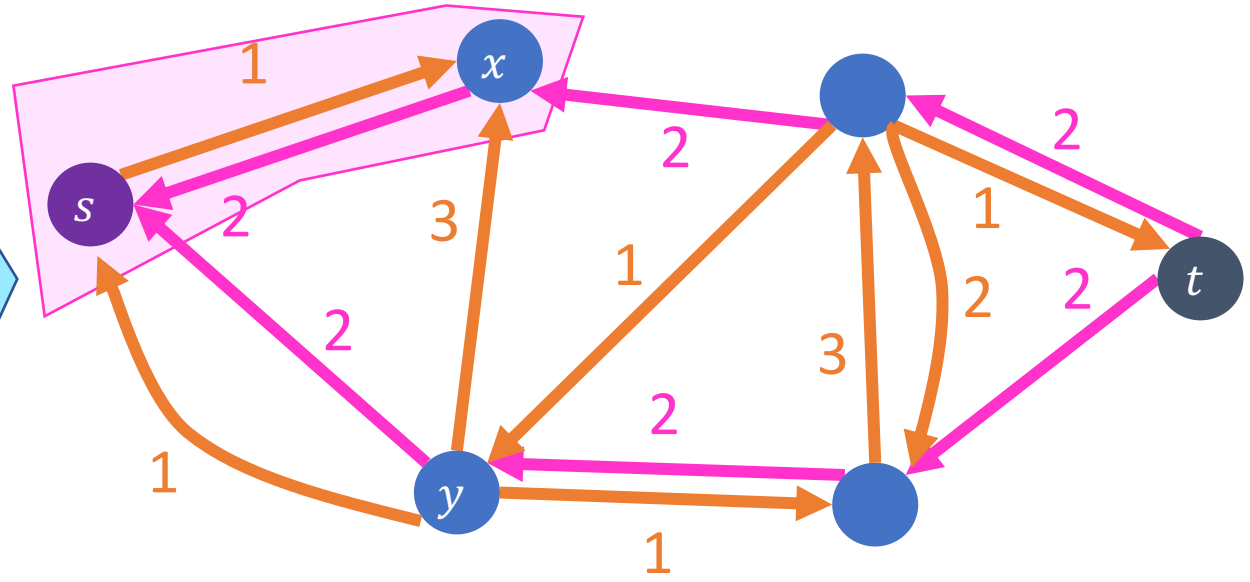
Claim: $\|S, T\| = |f|$

- Total flow $|f|$ is amount of outgoing flow from S to T minus the amount of incoming flow from T to S
- **Outgoing flow:** Consider edge (u, v) where $u \in S$ and $v \in T$
  - Then, $f(u, v) = c(u, v)$. Otherwise, there is a forward edge (u, v) with positive weight in G_f and $v \in S$

Max-Flow Min-Cut Theorem Proof



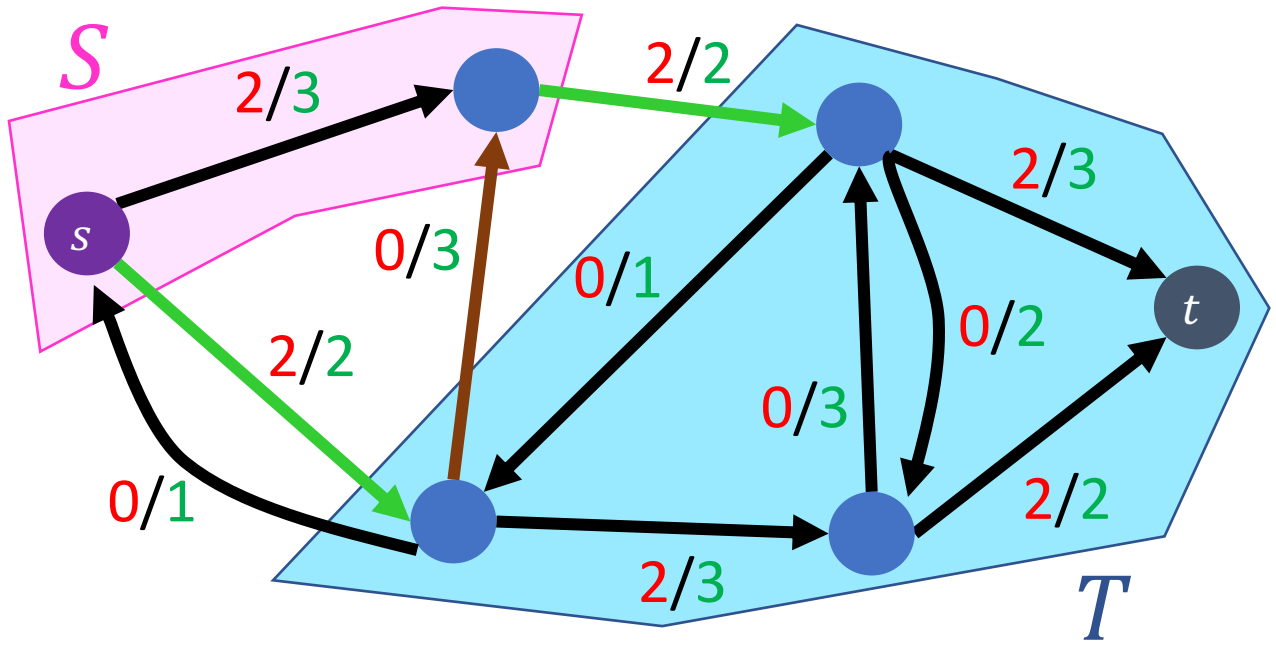
no more augmenting paths



Claim: $\|S, T\| = |f|$

- Total flow $|f|$ is amount of outgoing flow from S to T minus the amount of incoming flow from T to S
- **Outgoing flow:** Consider edge (u, v) where $u \in S$ and $v \in T$
  - Then, $f(u, v) = c(u, v)$. Otherwise, there is a forward edge (u, v) with positive weight in G_f and $v \in S$
- **Incoming flow:** Consider edge (y, x) where $y \in T$ and $x \in S$
  - Then, $f(y, x) = 0$. Otherwise, there is a backward edge (x, y) with positive weight in G_f and $y \in S$

Max-Flow Min-Cut Theorem Proof



$$|f| = \sum_{u \in S, v \in T} f(u,v) - \sum_{y \in T, x \in S} f(y,x)$$

$$= \sum_{u \in S, v \in T} c(u,v)$$

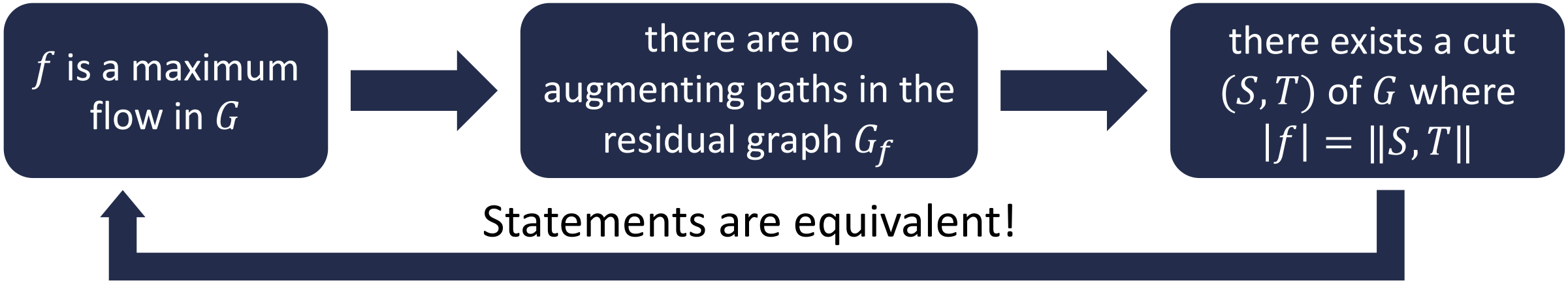
$$= \|S,T\|$$

Claim: $\|S,T\| = |f|$

- Total flow $|f|$ is amount of outgoing flow from S to T minus the amount of incoming flow from T to S
- **Outgoing flow:** Consider edge (u,v) where $u \in S$ and $v \in T$
  - Then, $f(u,v) = c(u,v)$. Otherwise, there is a forward edge (u,v) with positive weight in G_f and $v \in S$
- **Incoming flow:** Consider edge (y,x) where $y \in T$ and $x \in S$
  - Then, $f(y,x) = 0$. Otherwise, there is a backward edge (x,y) with positive weight in G_f and $y \in S$

Max-Flow Min-Cut Theorem

Let f be a flow in a graph G



Implications:

- **Correctness of Ford-Fulkerson:** Ford-Fulkerson terminates when there are no more augmenting paths in the residual graph G_f, which means that f is a maximum flow
- **Max-flow min-cut duality:** the maximum flow in a network coincides with the minimum cut of the graph ($\max_f |f| = \min_{S,T} \|S, T\|$)

Other Max Flow Algorithms

Ford-Fulkerson

- $\Theta(|E||f^*|)$

Edmonds-Karp (Ford-Fulkerson using BFS to choose augmenting path)

- $\Theta(|E|^2|V|)$

Push-Relabel (Tarjan)

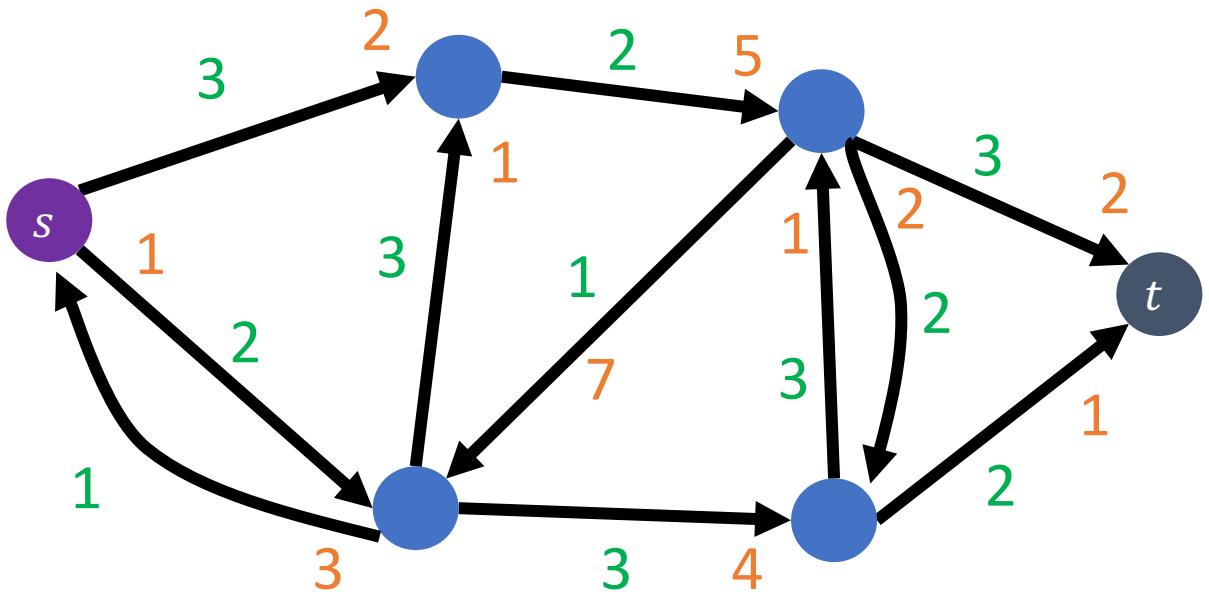
- $\Theta(|E||V|^2)$

Faster Push-Relabel (also Tarjan)

- $\Theta(|V|^3)$

Minimum-Cost Maximum-Flow Problem

Not all paths are created equal!



A cost is associated with each unit of flow sent along an edge

Goal: Maximize flow while minimizing cost

Much harder problem!
Can solve using linear programming