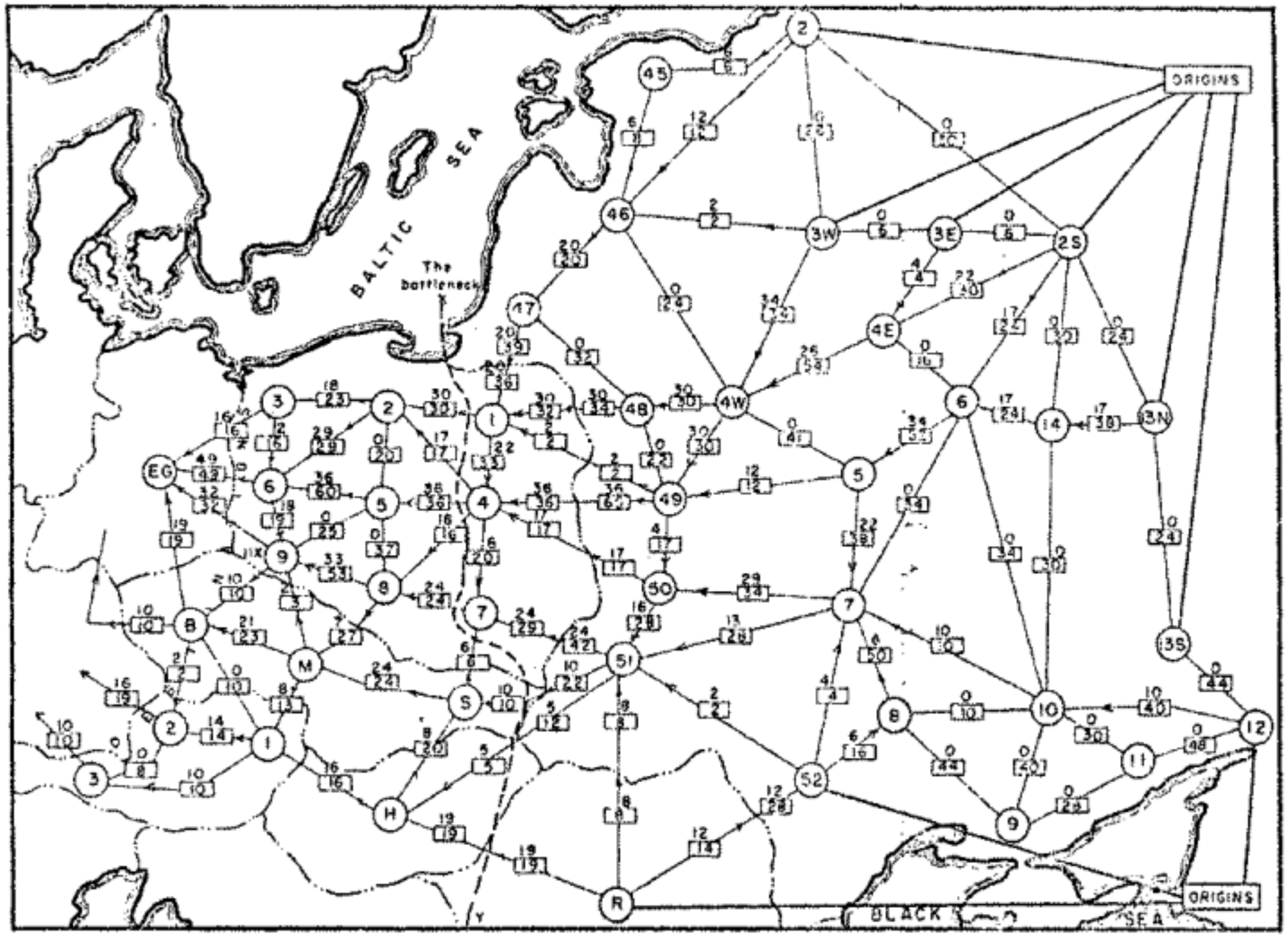
Network Flow



Railway map of Western USSR, 1955

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

Announcements

Unit B

- Programming due Friday, 4/15, 11:30pm

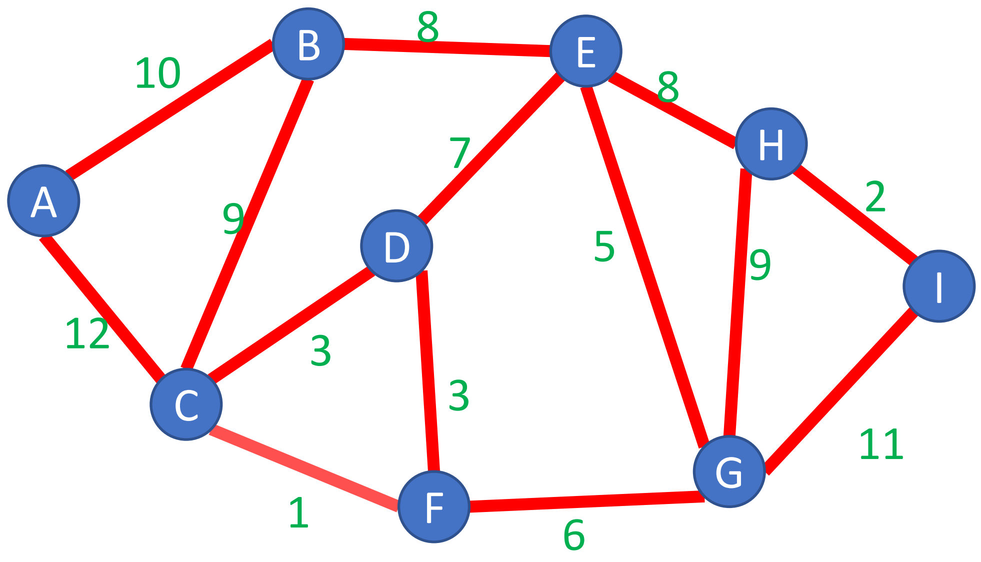
Unit C

- Basic 1 + 2 due Friday, 4/15, 11:30pm
- Advanced due Friday, 4/22
- Programming due Friday 4/22 – Seam carving!

Kruskal's Algorithm

Start with an empty tree A

Add to A the lowest-weight edge that does not create a cycle



Greedy Algorithms

Require Optimal Substructure

- Solution to larger problem contains the solution to a smaller one
- Only one subproblem to consider!

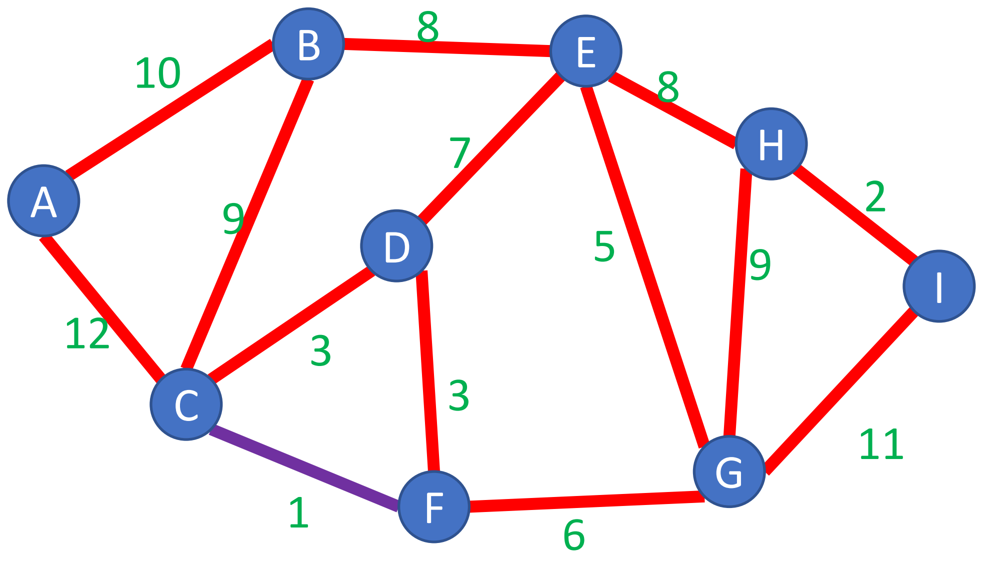
Idea:

1. Identify a greedy choice property
    - How to make a choice guaranteed to be included in some optimal solution
2. Repeatedly apply the choice property until no subproblems remain

Kruskal’s Algorithm

Start with an empty tree A

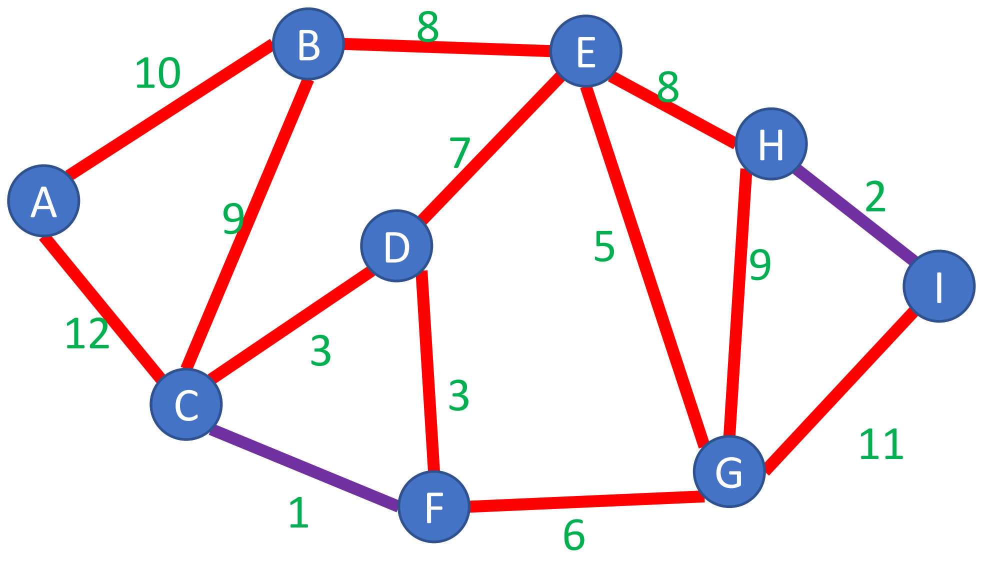
Add to A the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

Start with an empty tree A

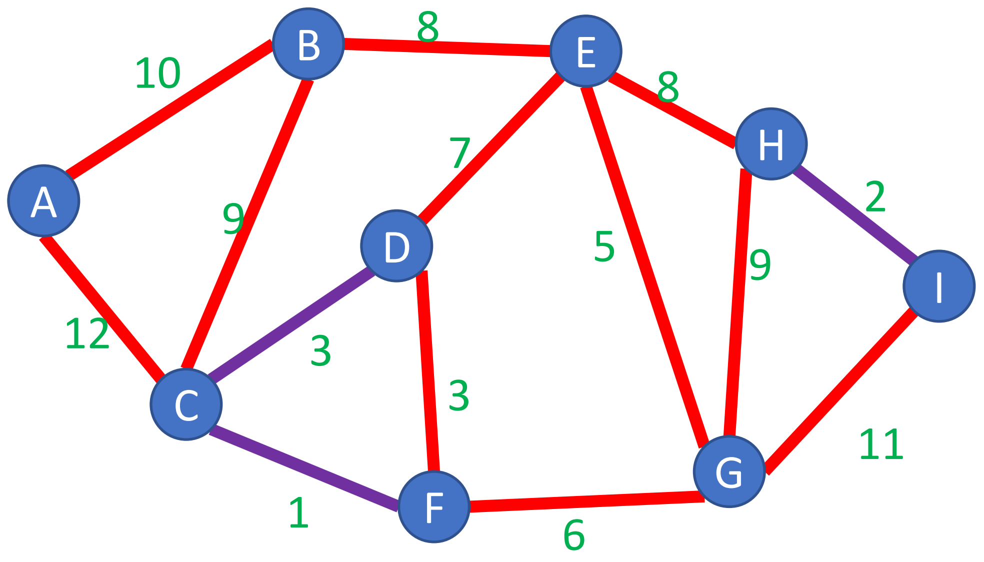
Add to A the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

Start with an empty tree A

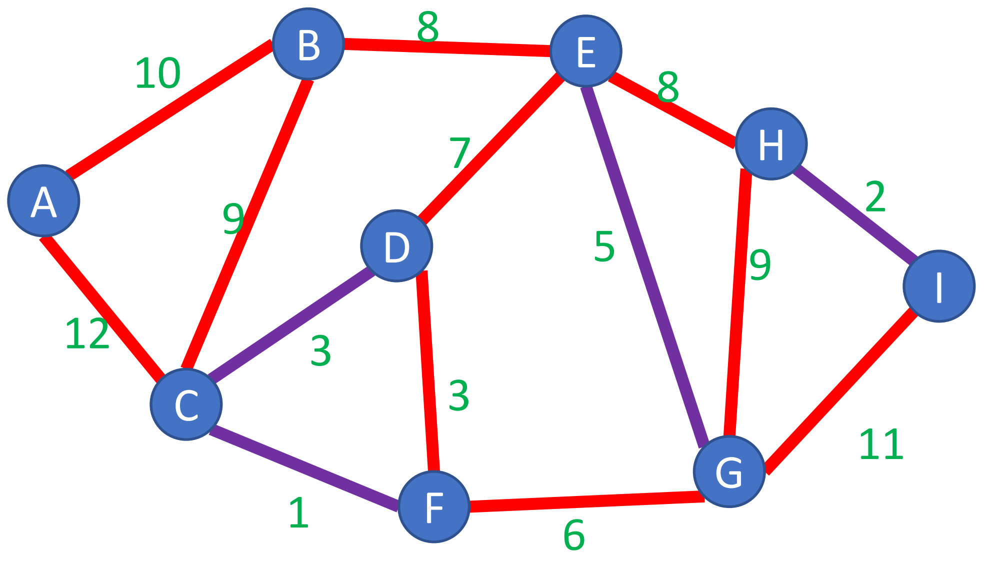
Add to A the lowest-weight edge that does not create a cycle



Kruskal’s Algorithm

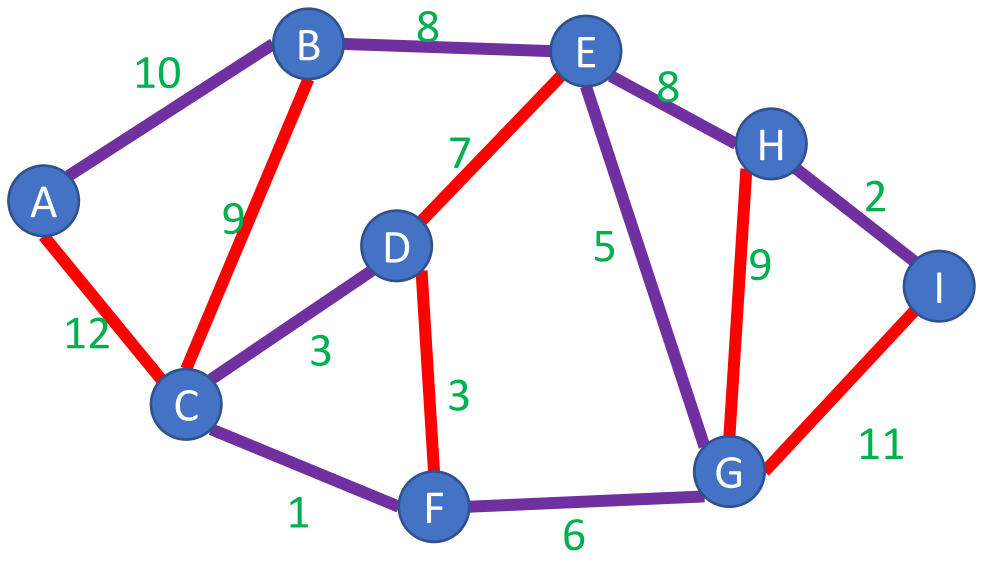
Start with an empty tree A

Add to A the lowest-weight edge that does not create a cycle



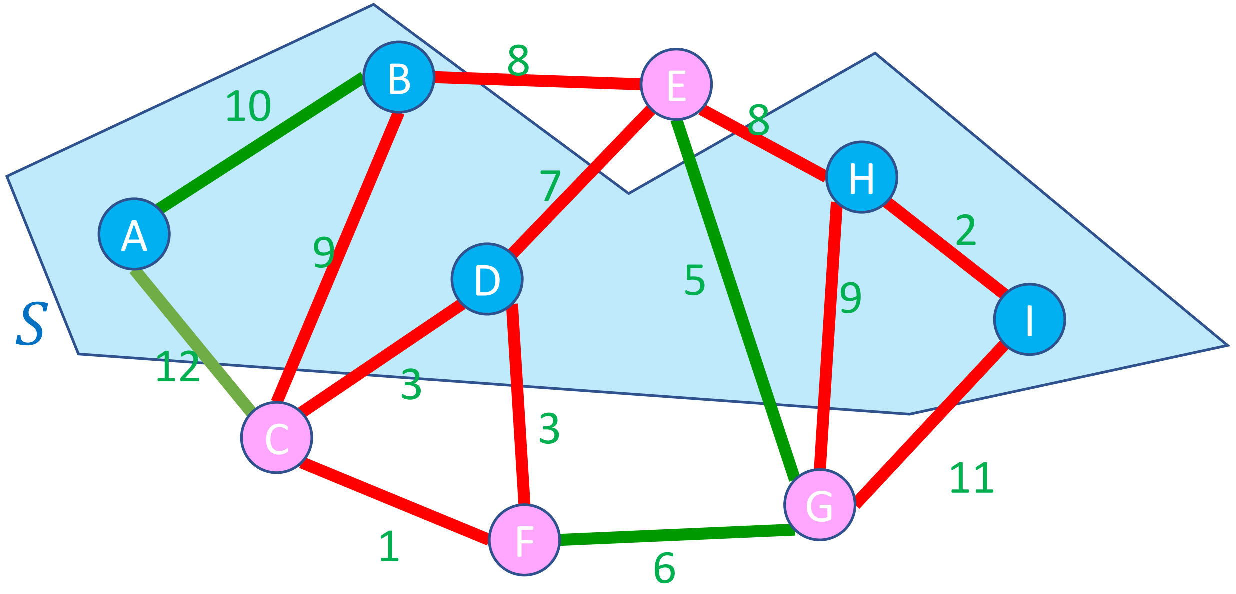
Kruskal's Algorithm

Start with an empty tree A
Add to A the lowest-weight edge that does not create a cycle



Definition: Cut

A Cut of graph $G = (V, E)$ is a partition of the nodes into two sets, S and $V - S$



Edge $(v_1, v_2) \in E$ crosses a cut if $v_1 \in S$ and $v_2 \in V - S$ (or opposite), e.g. (A, C)

A set of edges R Respects a cut if no edges cross the cut e.g. $R = \{(A, B), (E, G), (F, G)\}$

Exchange argument

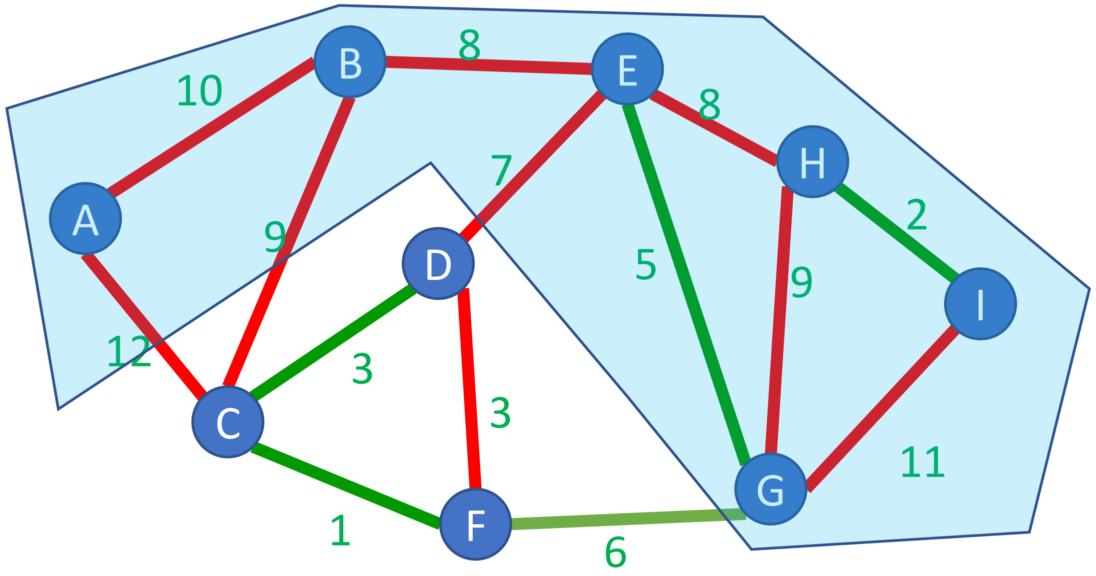
Shows correctness of a greedy algorithm

Idea:

- Show exchanging an item from an arbitrary optimal solution with your greedy choice makes the new solution no worse
- How to show my sandwich is at least as good as yours:
  - Show: “I can remove any item from your sandwich, and it would be no worse by replacing it with the same item from my sandwich”

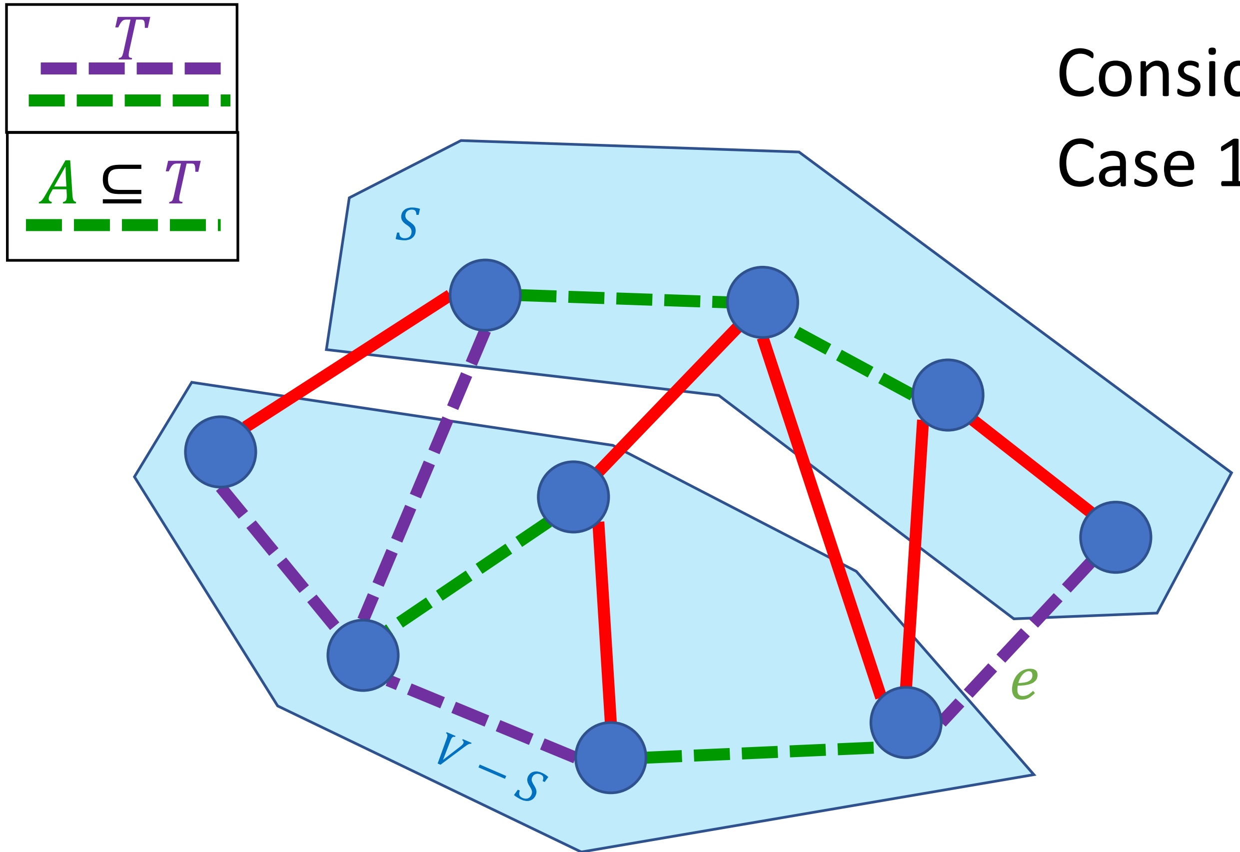
Cut Theorem

If a set of edges A is a subset of a minimum spanning tree T, let $(S, V - S)$ be any cut which A respects. Let e be the least-weight edge which crosses $(S, V - S)$. $A \cup \{e\}$ is also a subset of a minimum spanning tree.



Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.



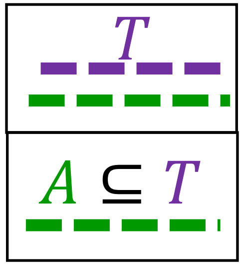
Consider some MST T,

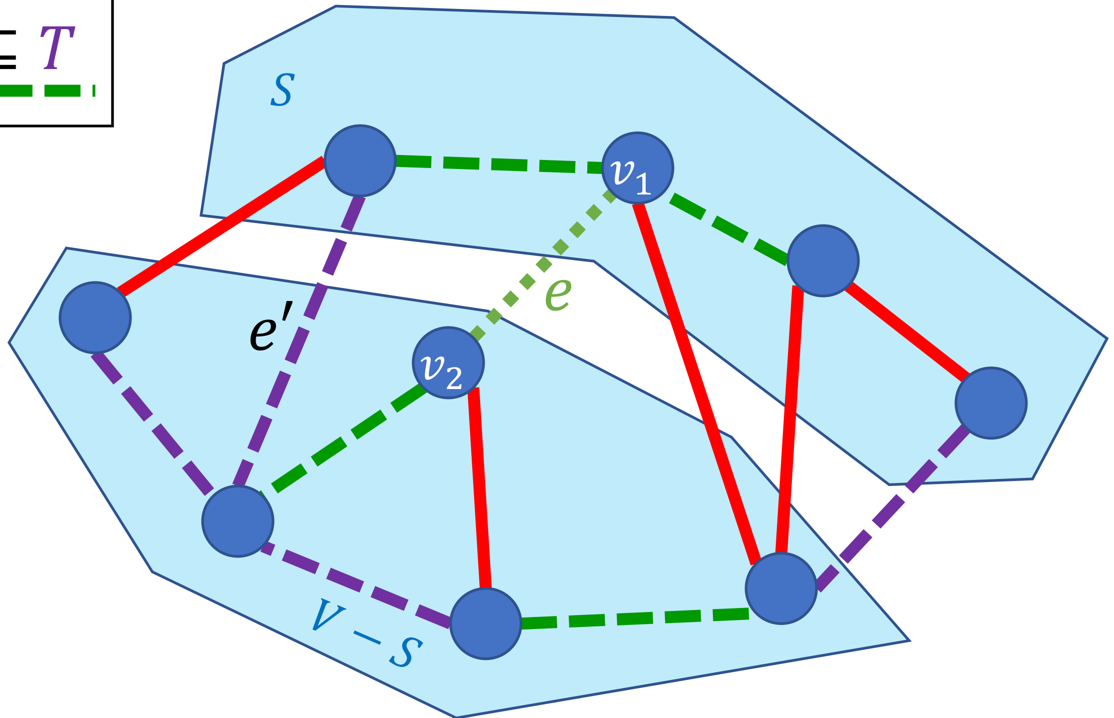
Case 1: (the easy case)

If $e \in T$ Then claim holds

Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.





Consider some MST T,

Case 2:

Consider if $e = (v_1, v_2) \notin T$

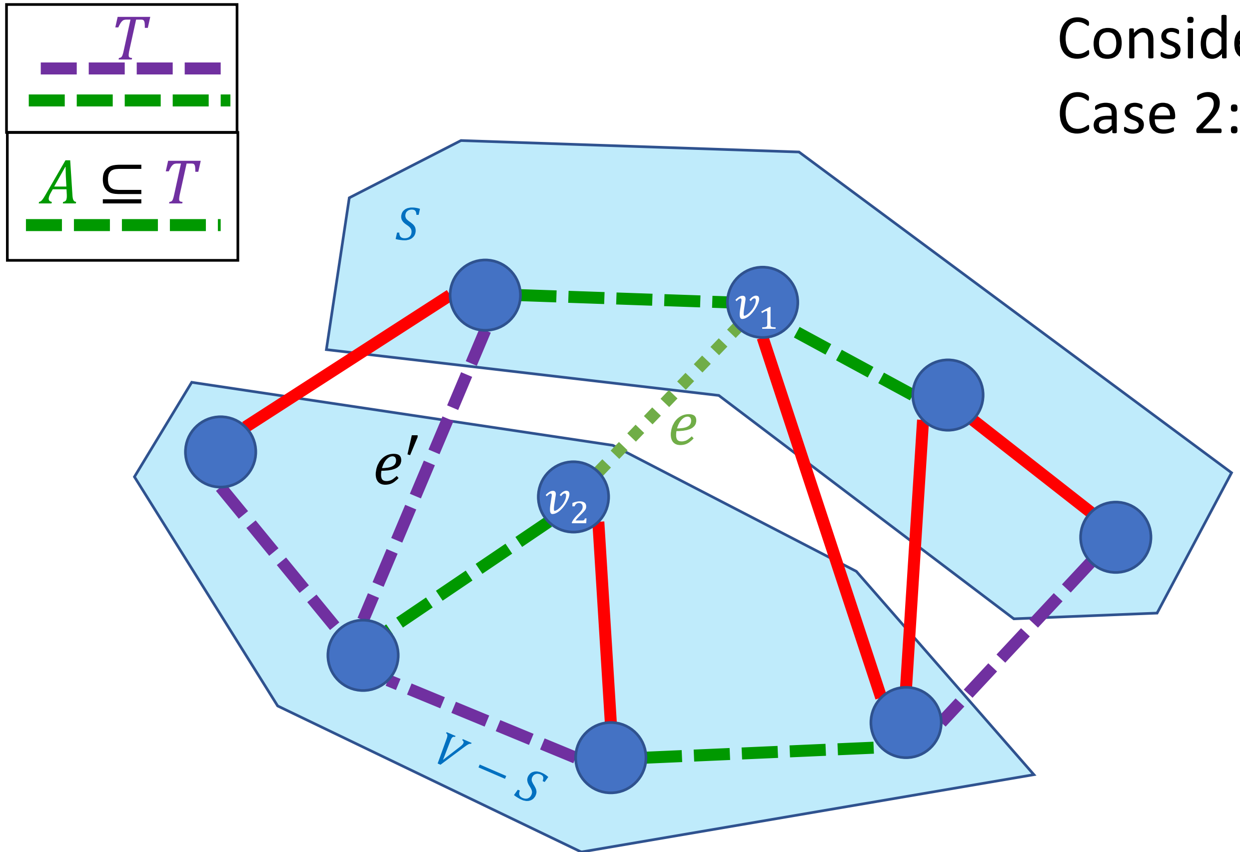
Since T is a MST, there is some path from v_1 to v_2.

Let e' be the first edge on this path which crosses the cut

Build tree T' by exchanging e' for e

Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V-S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.



Consider some MST T,

Case 2:

Consider if $e = (v_1, v_2) \notin T$

$T' = T$ with edge e instead of e'

We assumed $w(e) \leq w(e')$

$w(T') = w(T) - w(e') + w(e)$

$w(T') \leq w(T)$

So T' is also a MST!

Thus the claim holds

Kruskal's Algorithm

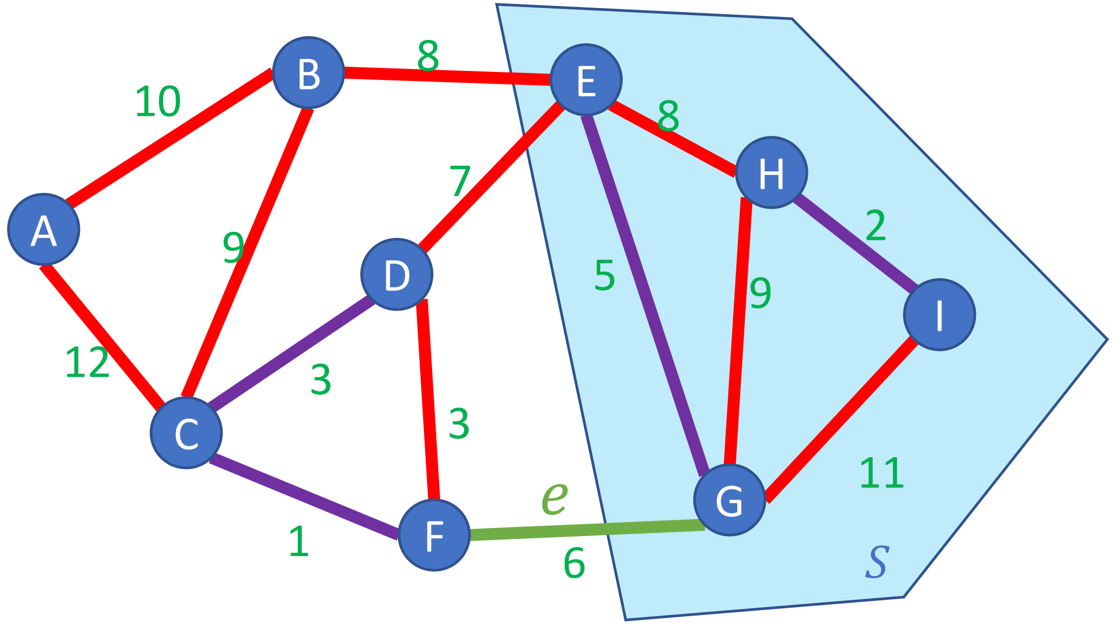
Start with an empty tree A

Repeat $V - 1$ times:

Add the min-weight edge that doesn't cause a cycle

Keep edges in a Disjoint-set data structure (very fancy)

$O(E \log V)$



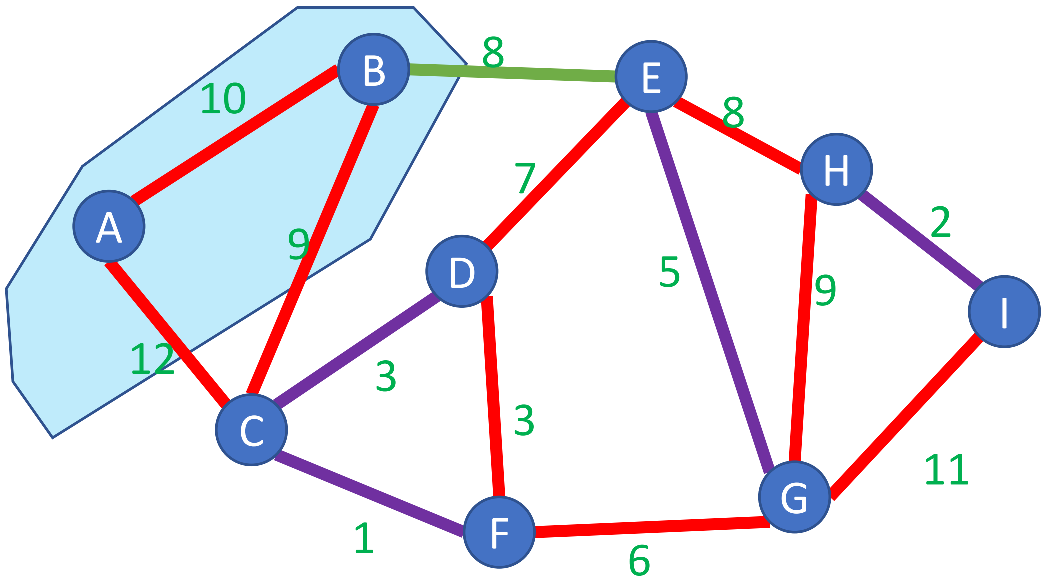
General MST Algorithm

Start with an empty tree A

Repeat $V - 1$ times:

Pick a cut $(S, V - S)$ which A respects

Add the min-weight edge which crosses $(S, V - S)$



Prim's Algorithm

Start with an empty tree A

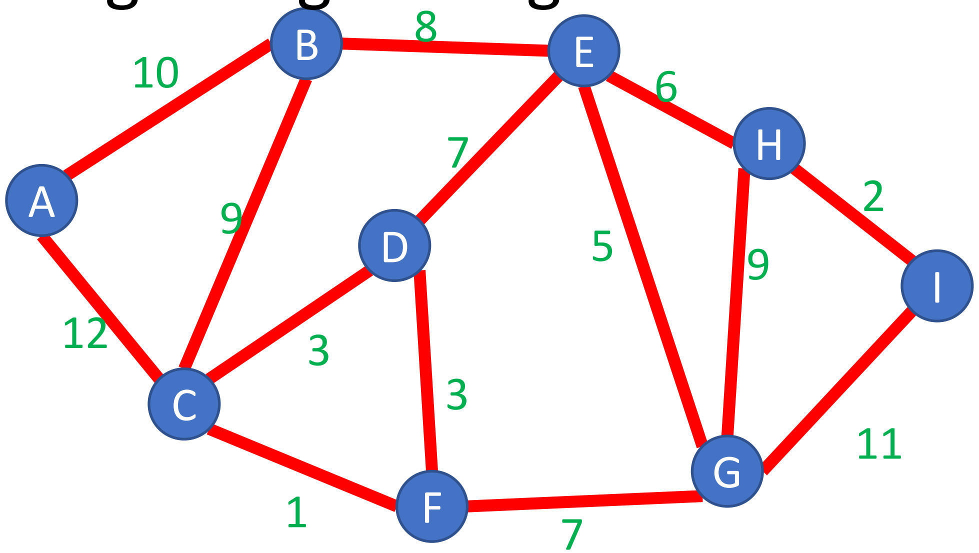
Repeat $V - 1$ times:

Pick a cut $(S, V - S)$ which A respects

Add the min-weight edge which crosses $(S, V - S)$

S is all endpoint of edges in A

e is the min-weight edge that grows the tree



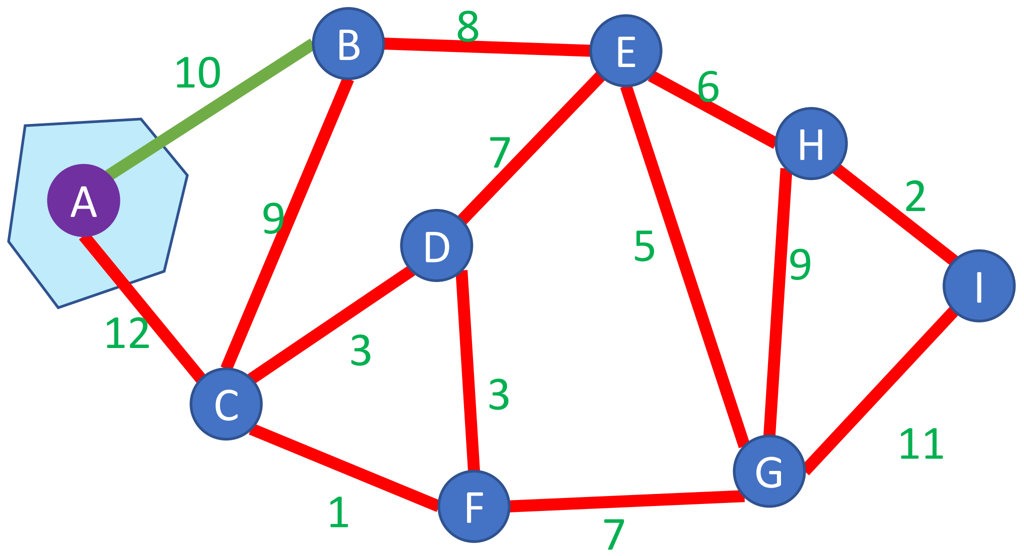
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



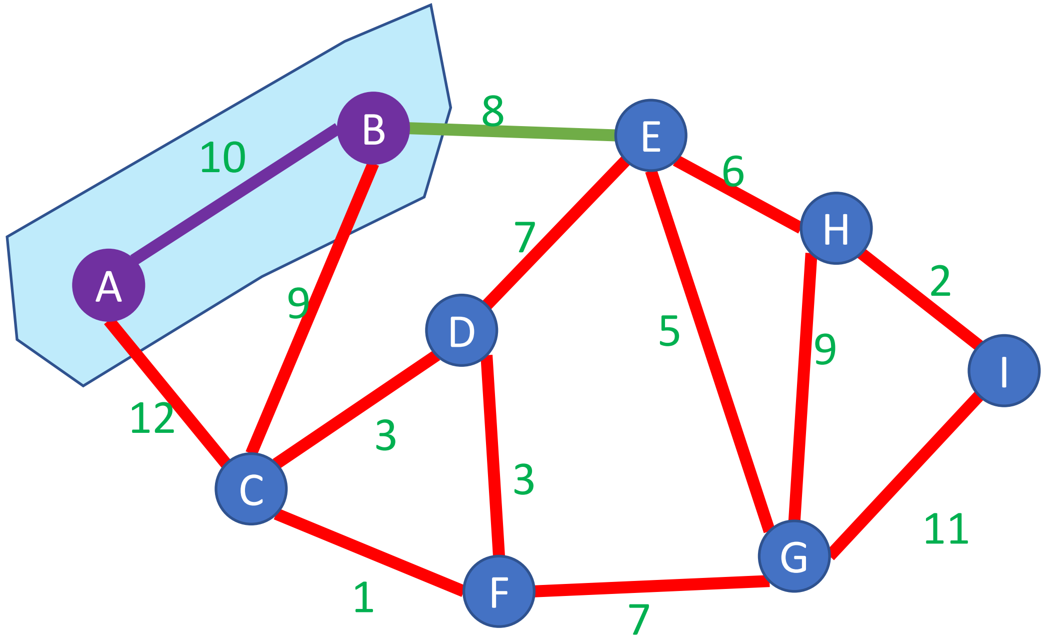
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



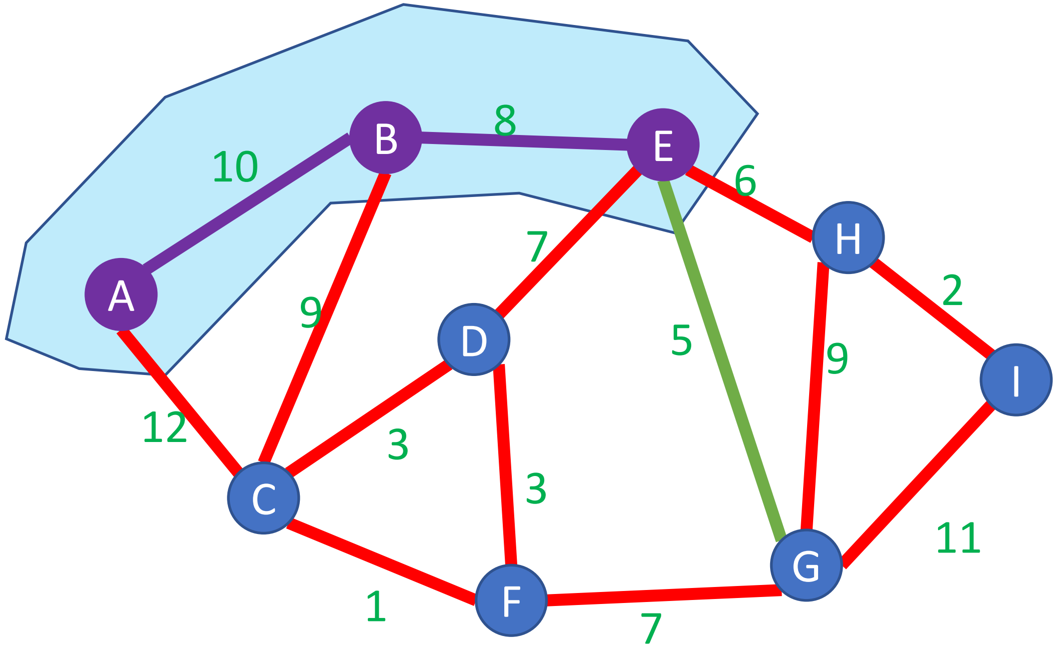
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



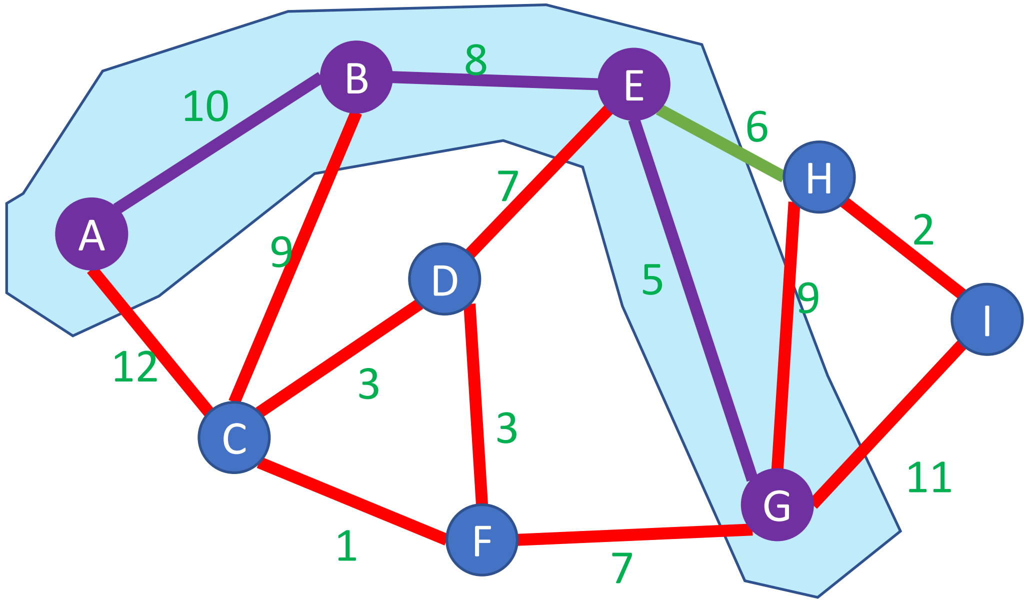
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

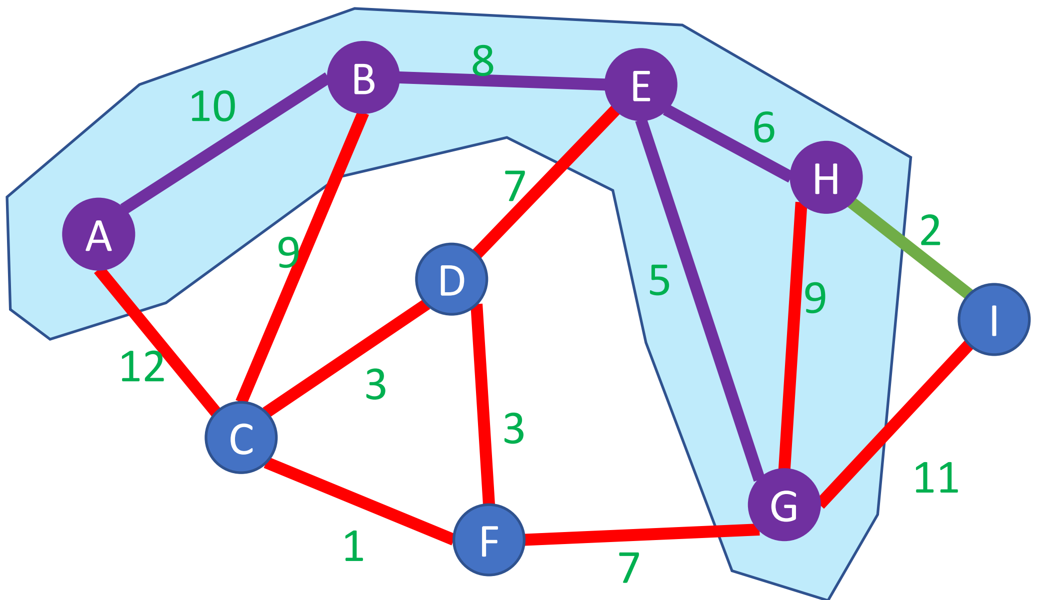
Add the min-weight edge which connects to node in A with a node not in A



Prim's Algorithm

Start with an empty tree A
Pick a start node
Repeat $V - 1$ times:
Add the min-weight edge which connects to node in A with a node not in A

Keep edges in a Heap
$O(E \log V)$



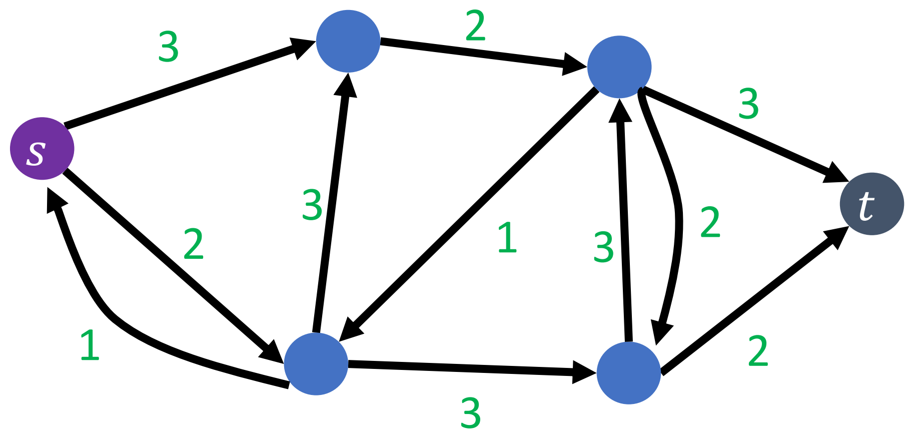
Flow Networks

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

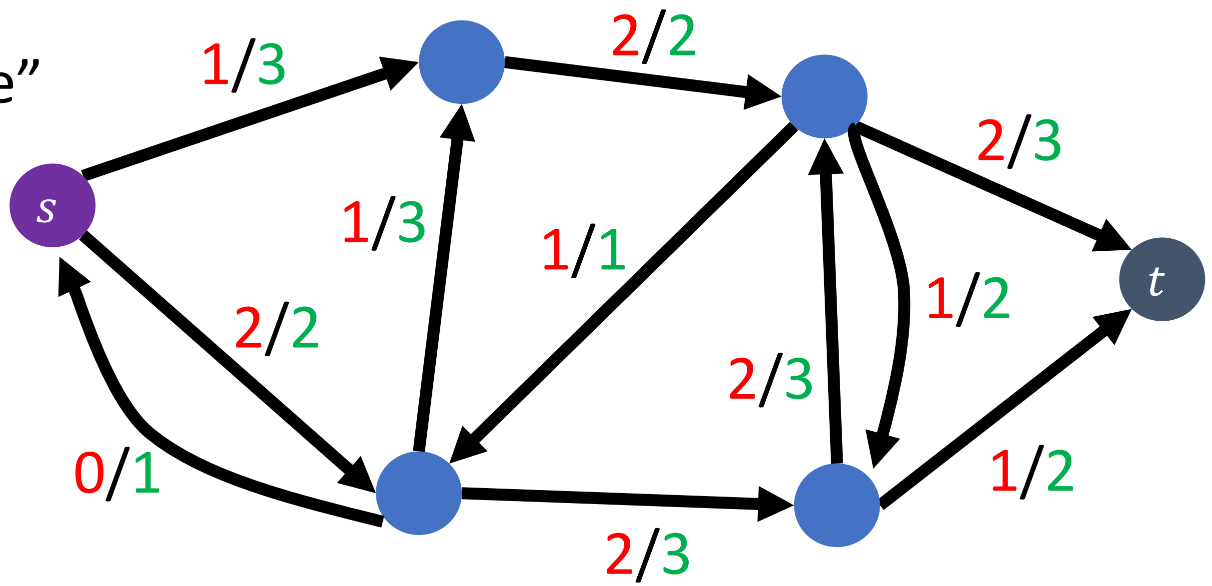
- $f(e) \leq c(e)$
- "Flow cannot exceed capacity"

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

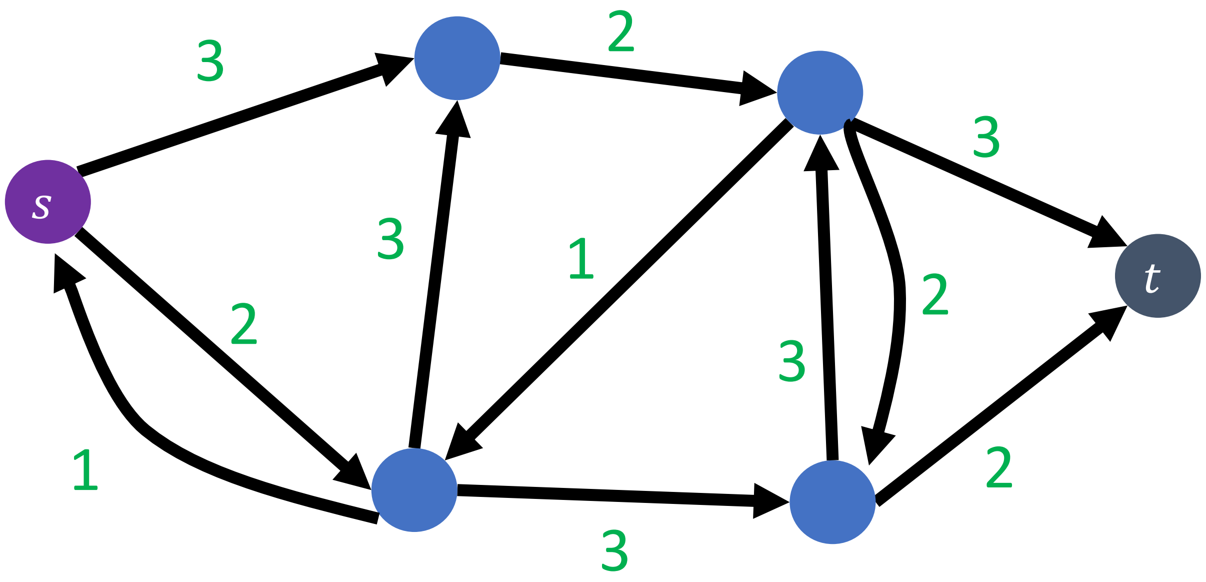


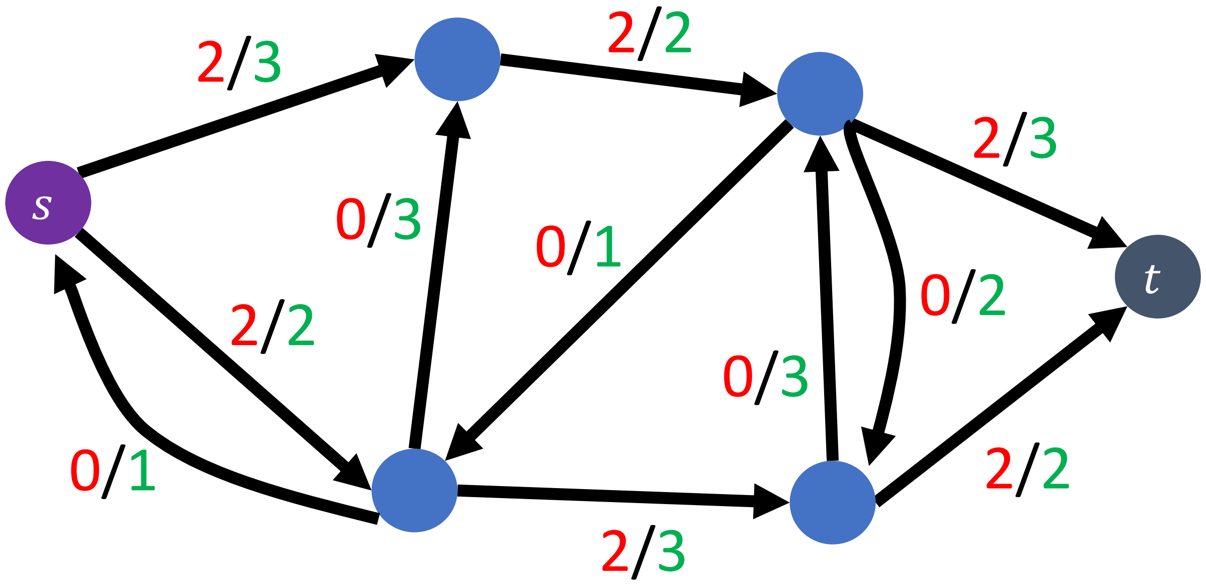
flow / capacity

Maximum Flow Problem

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

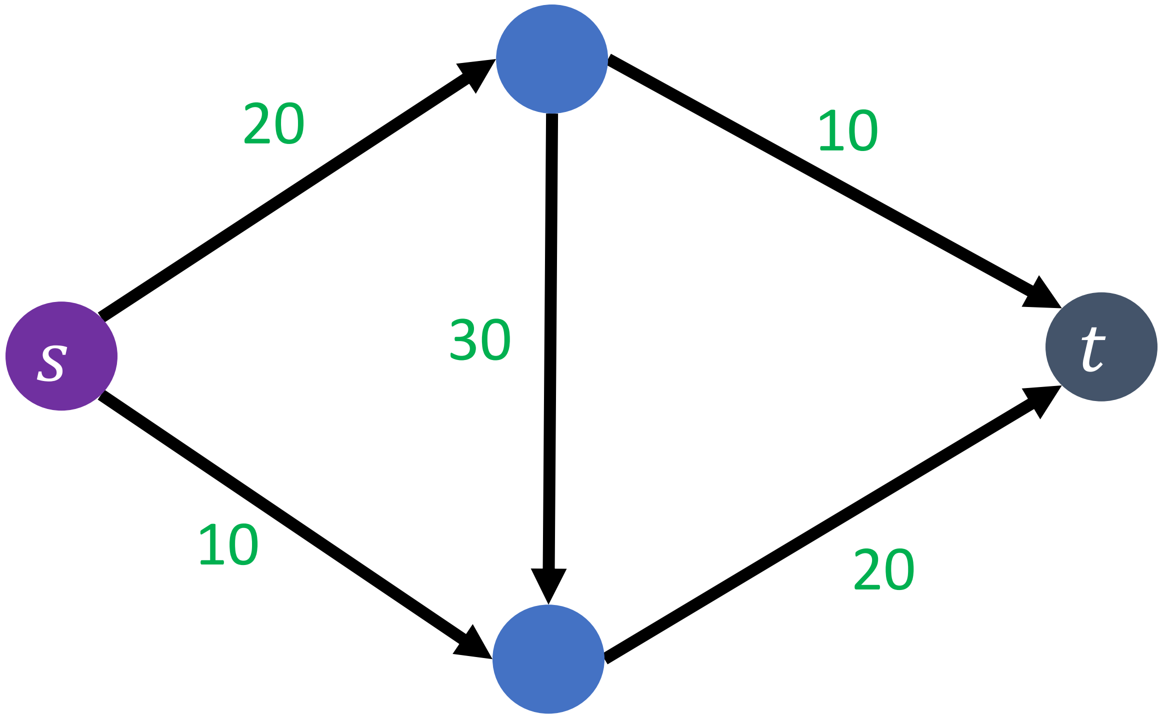
$$|f| = \text{outflow}(s) - \text{inflow}(s)$$





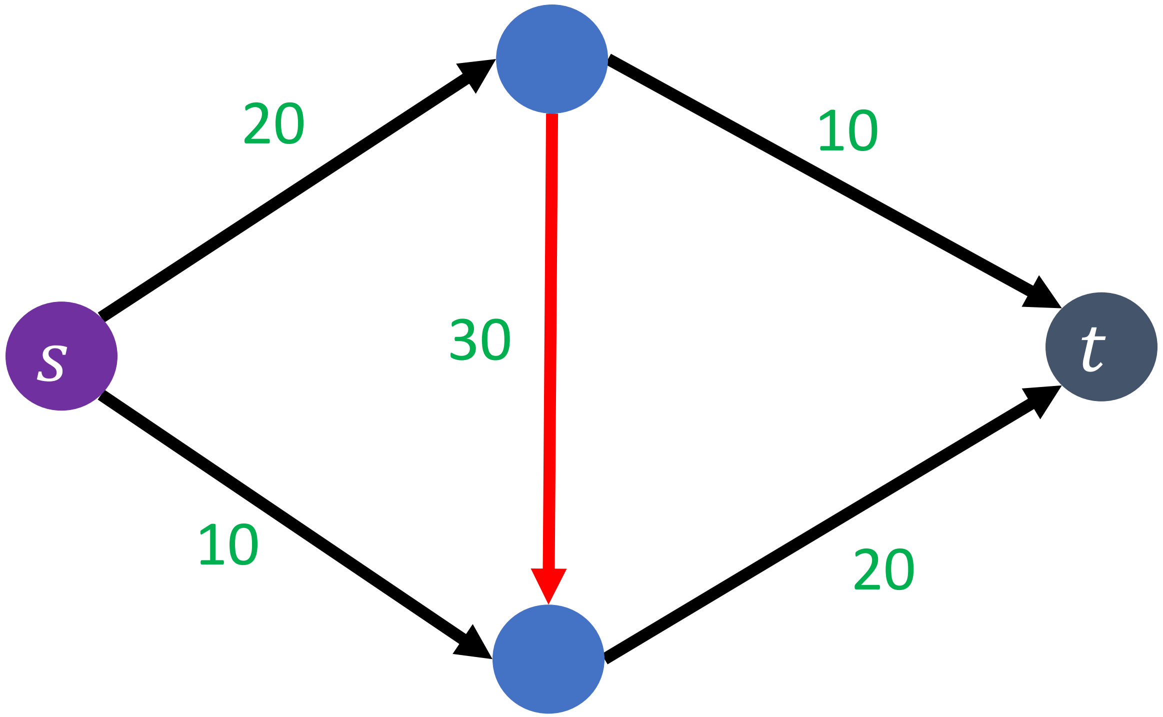
Greedy Approach

Greedy choice: saturate highest capacity path first



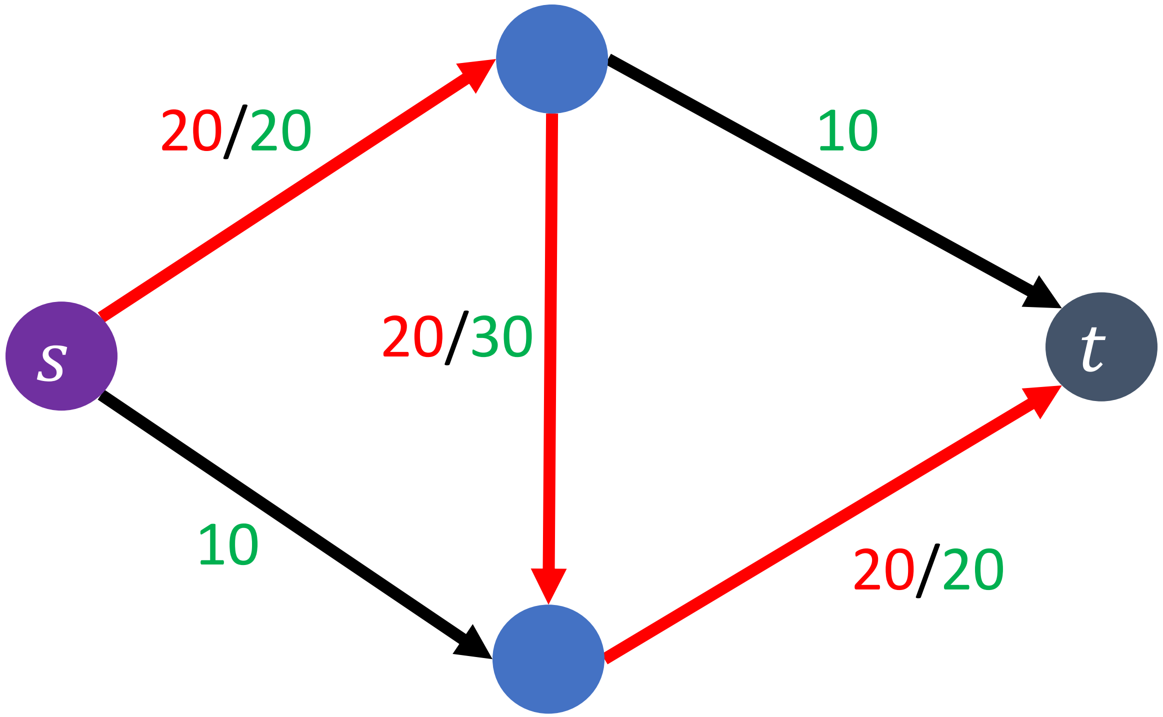
Greedy Approach

Greedy choice: saturate <u>highest</u> capacity path first



Greedy Approach

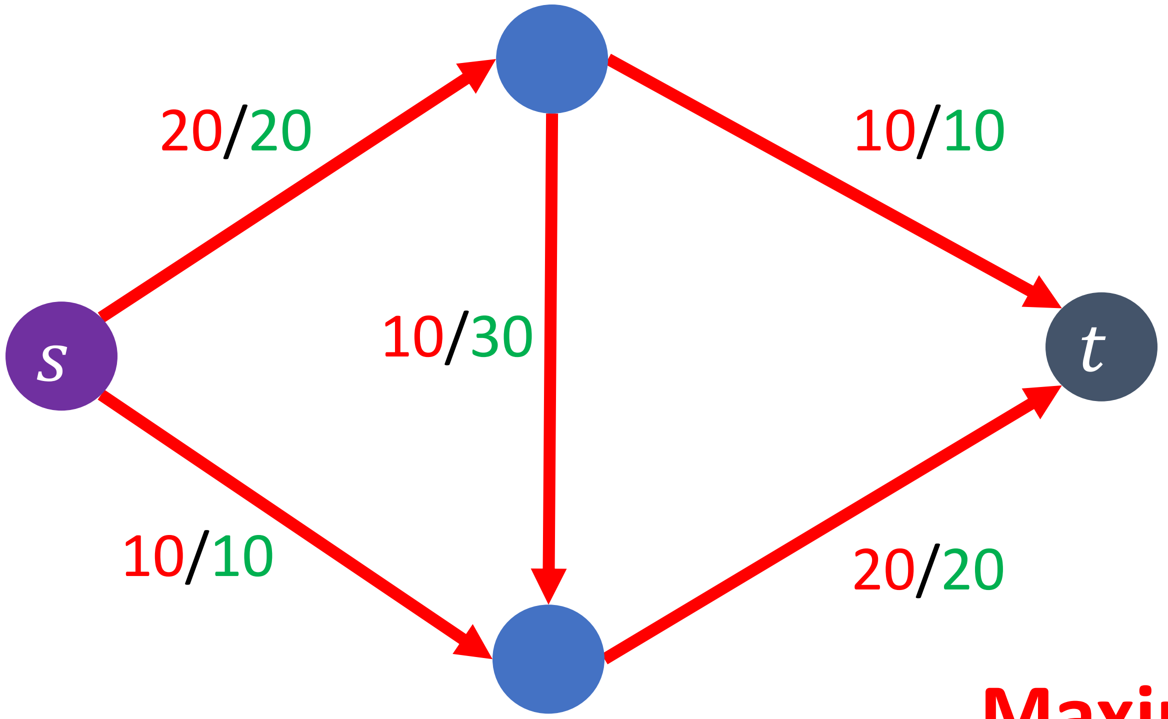
Greedy choice: saturate <u>highest</u> capacity path first



Flow: 20

Greedy Approach

Greedy choice: saturate <u>highest</u> capacity path first



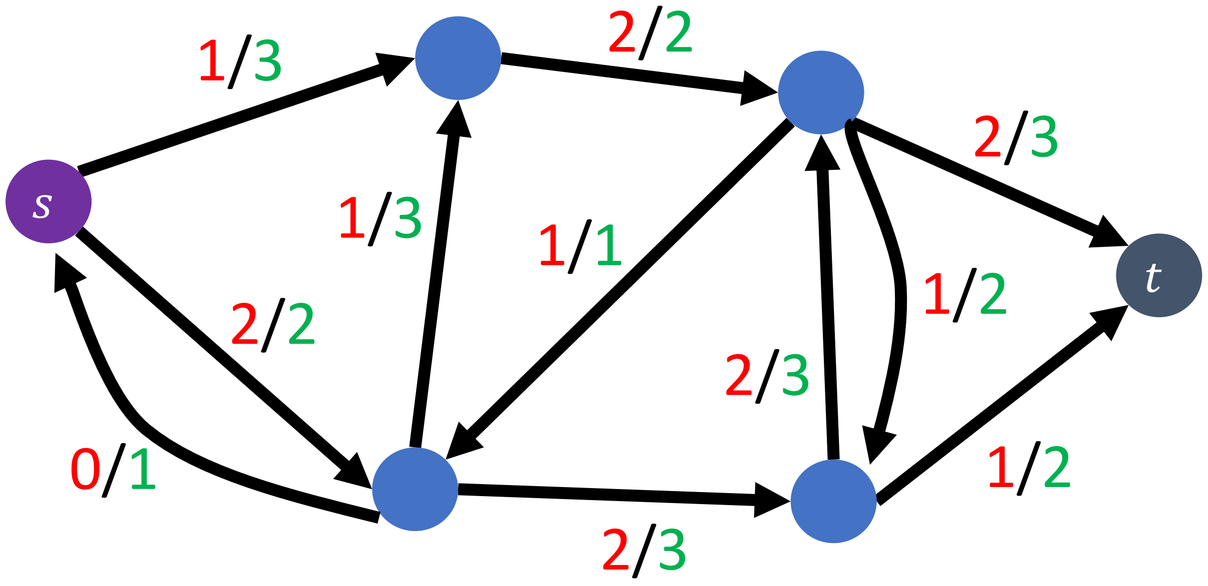
Maximum Flow: 30

Observe: highest capacity path is not <u>saturated</u> in optimal solution

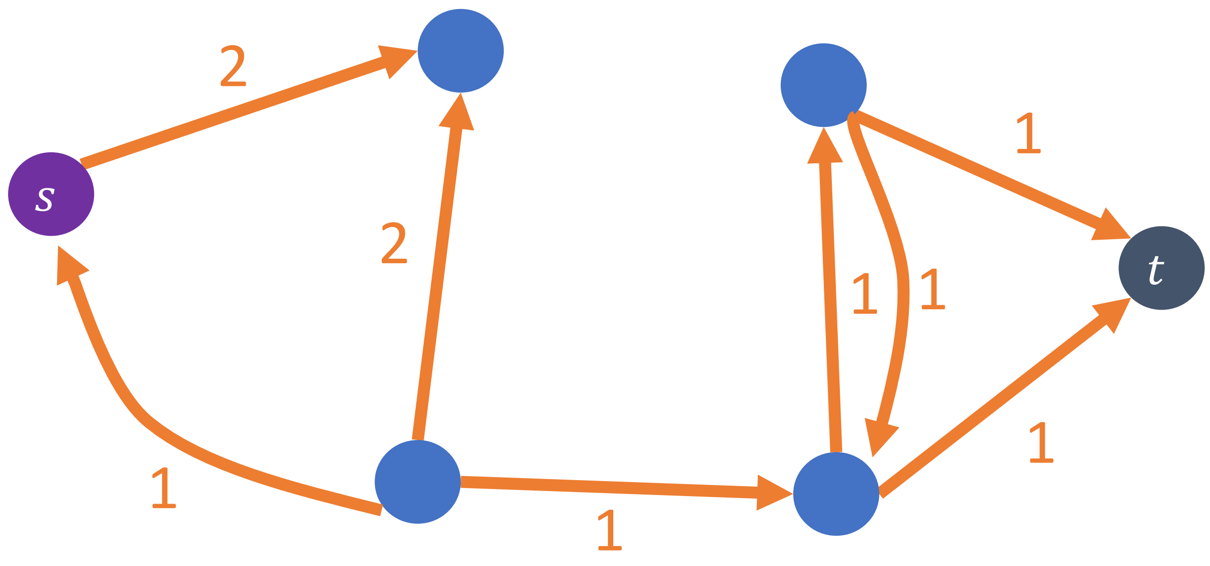
Residual Graphs

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

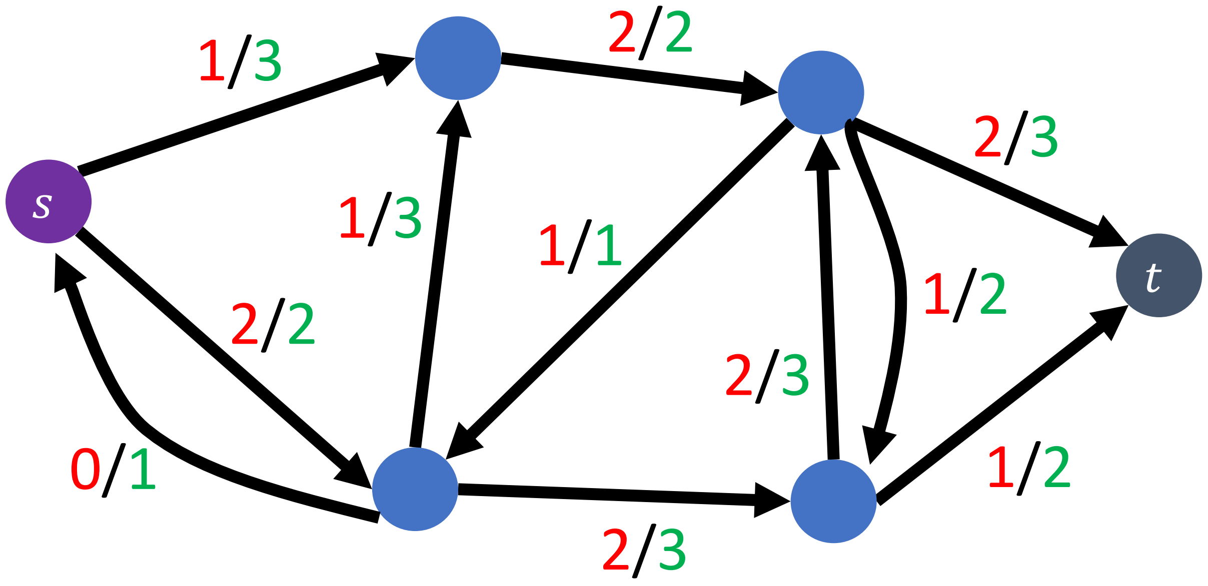


Residual graph G_f

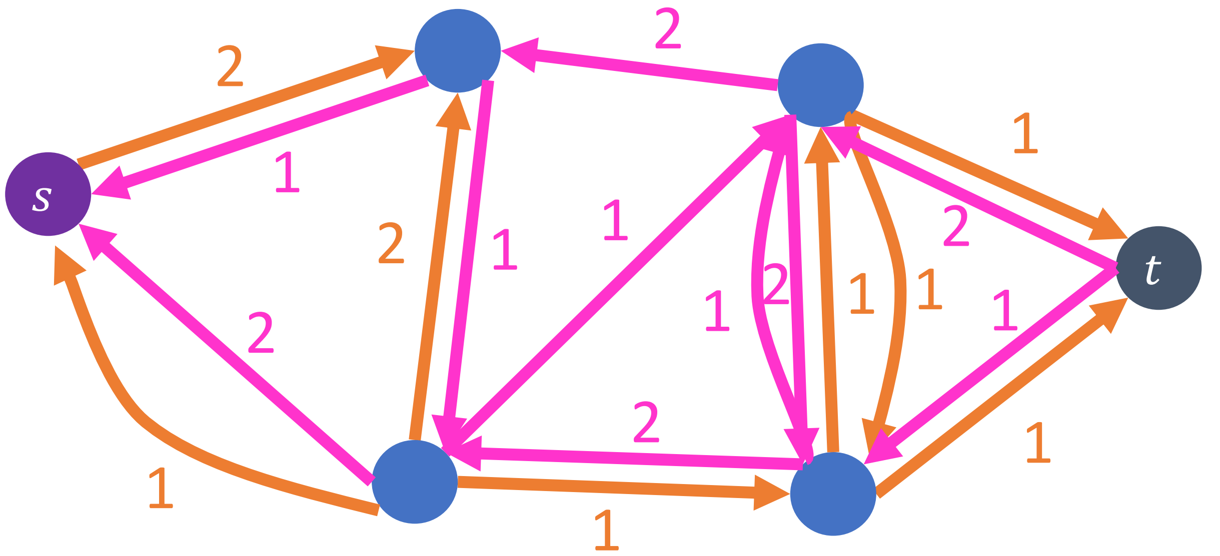
Residual Graphs

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



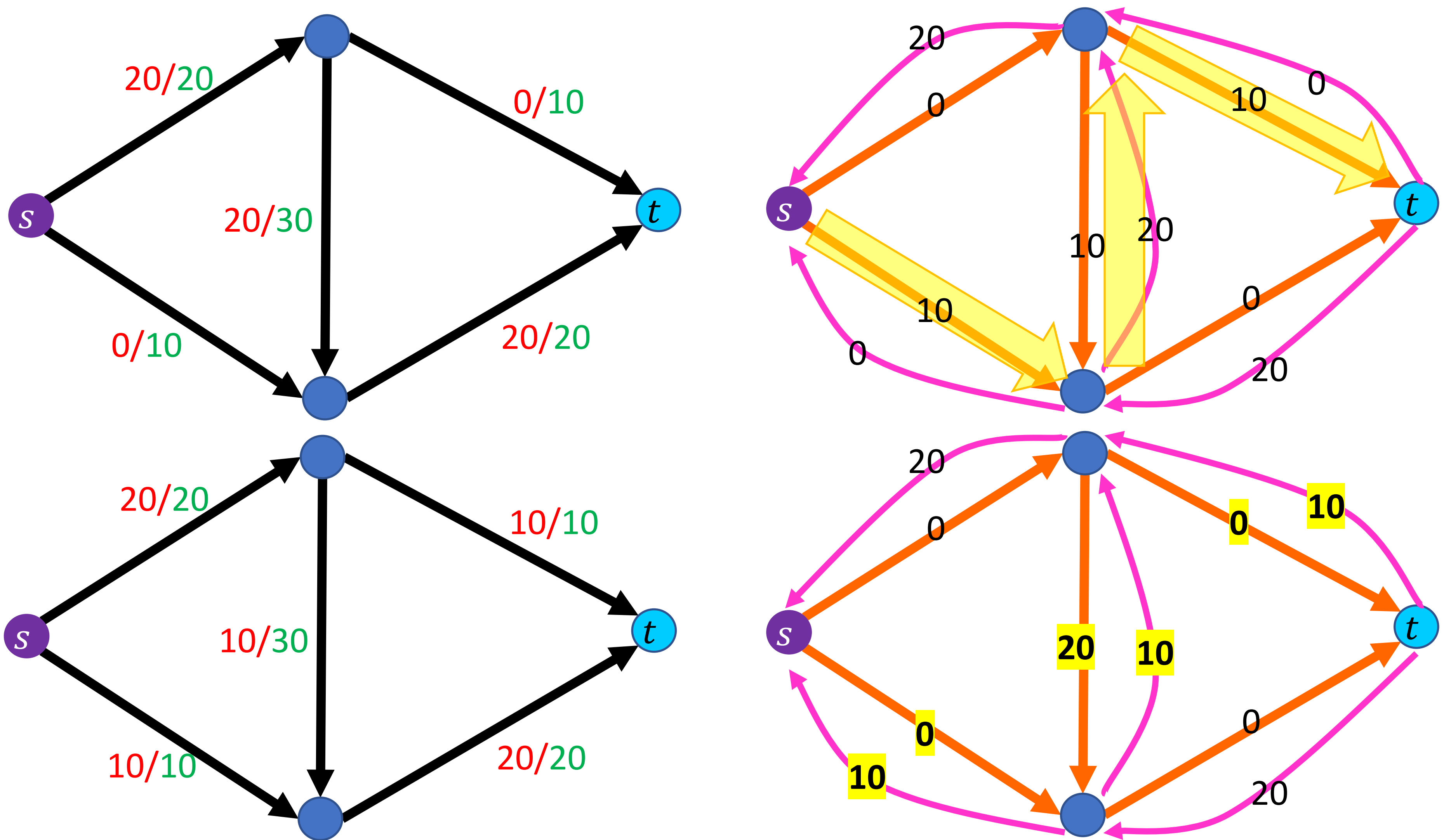
Flow f in G



Residual graph G_f

Residual Graphs Example

Flow Graph | **Residual Graph**

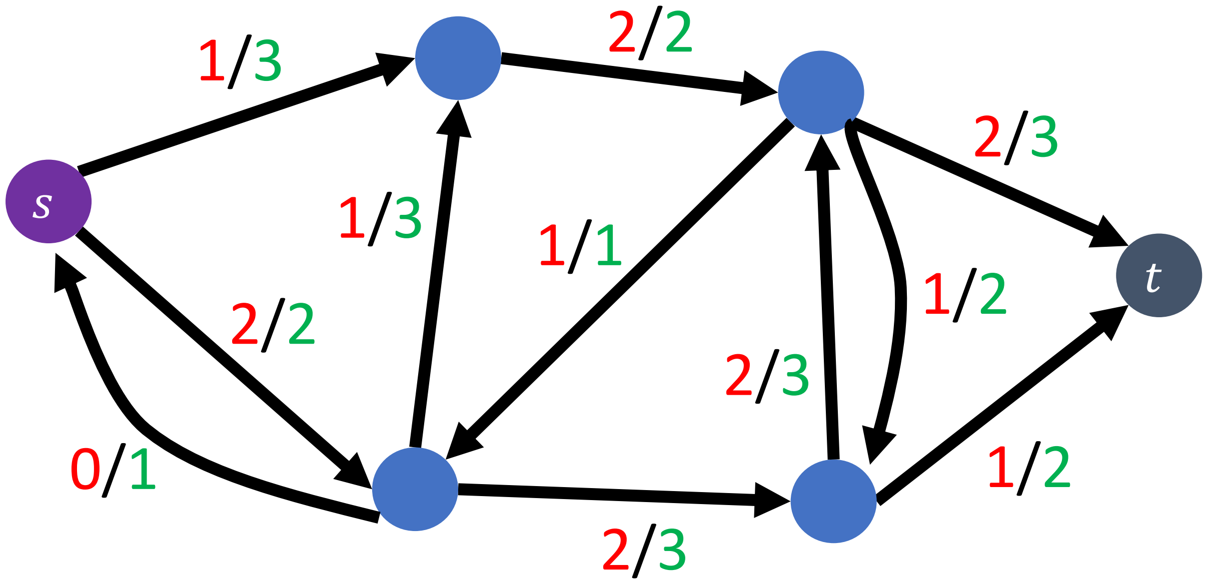


Residual Graphs

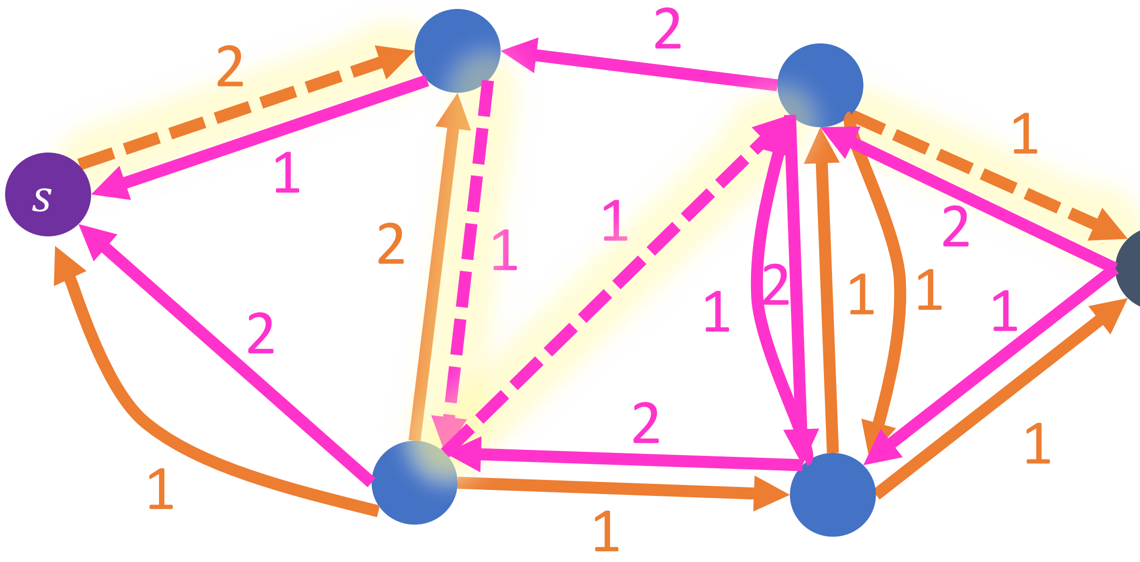
Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

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



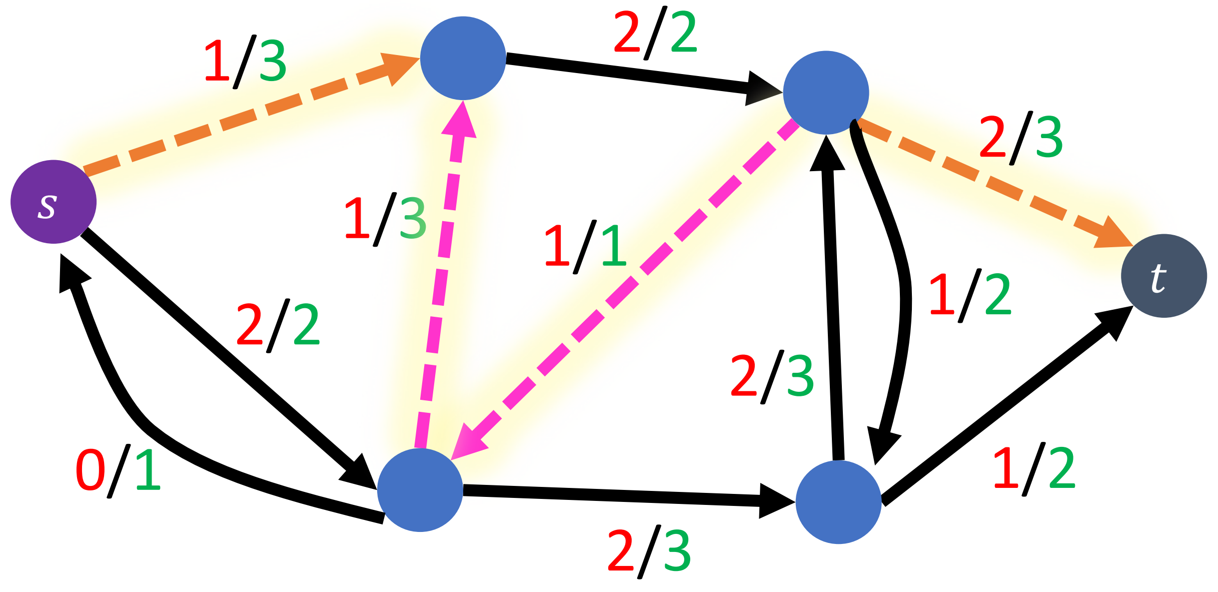
Residual graph G_f

Residual Graphs

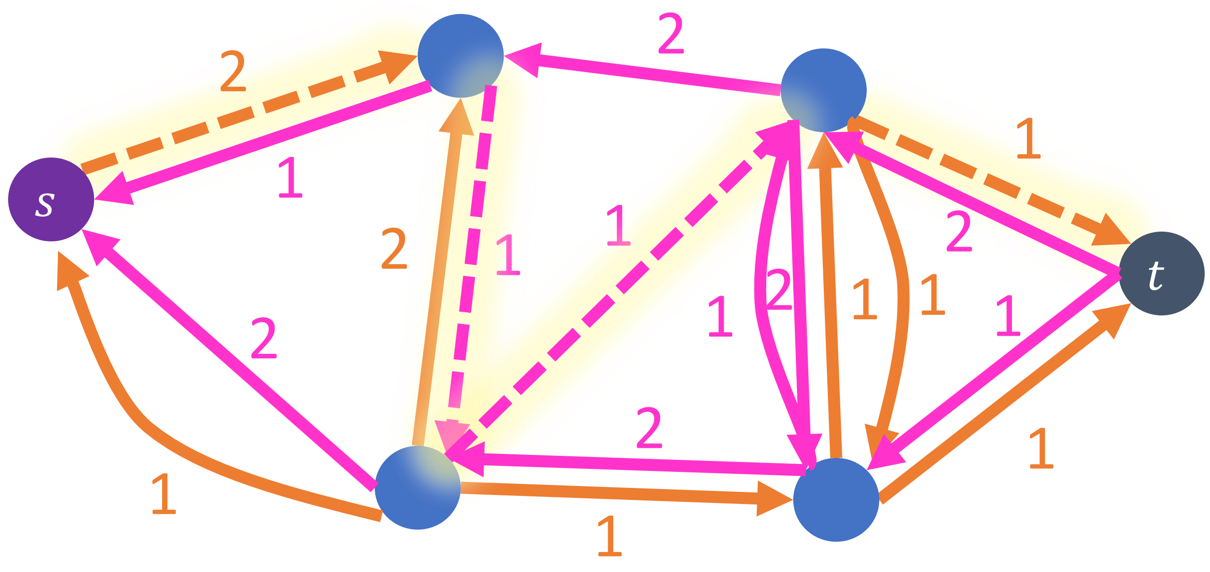
Consider a path from $s \to t$ in G_f using only edges with positive (non-zero) weight

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

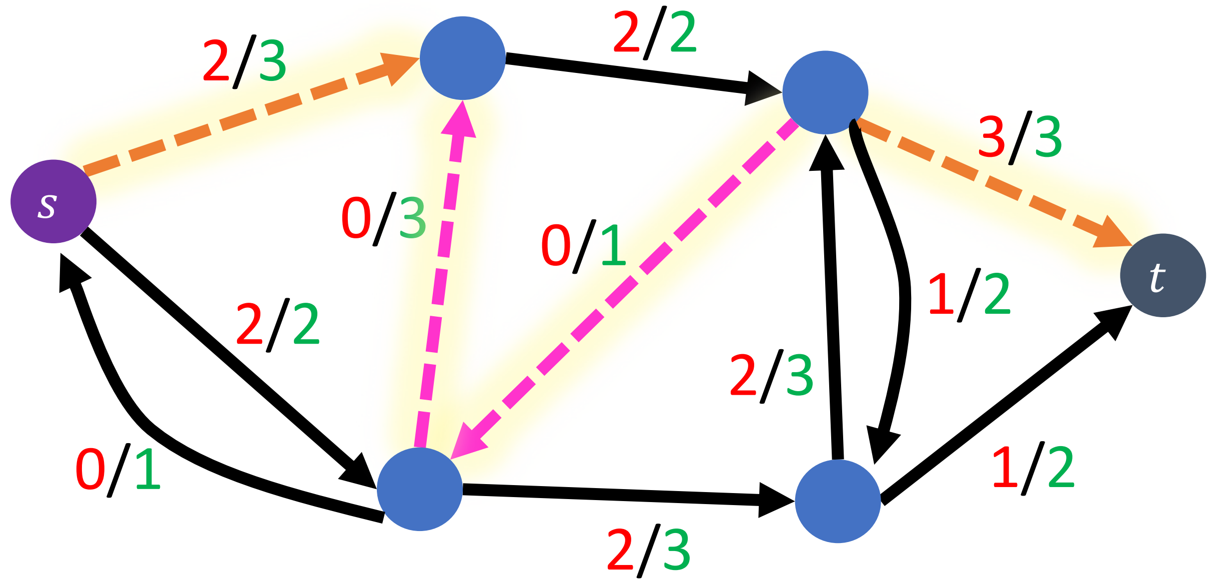
Residual Graphs

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

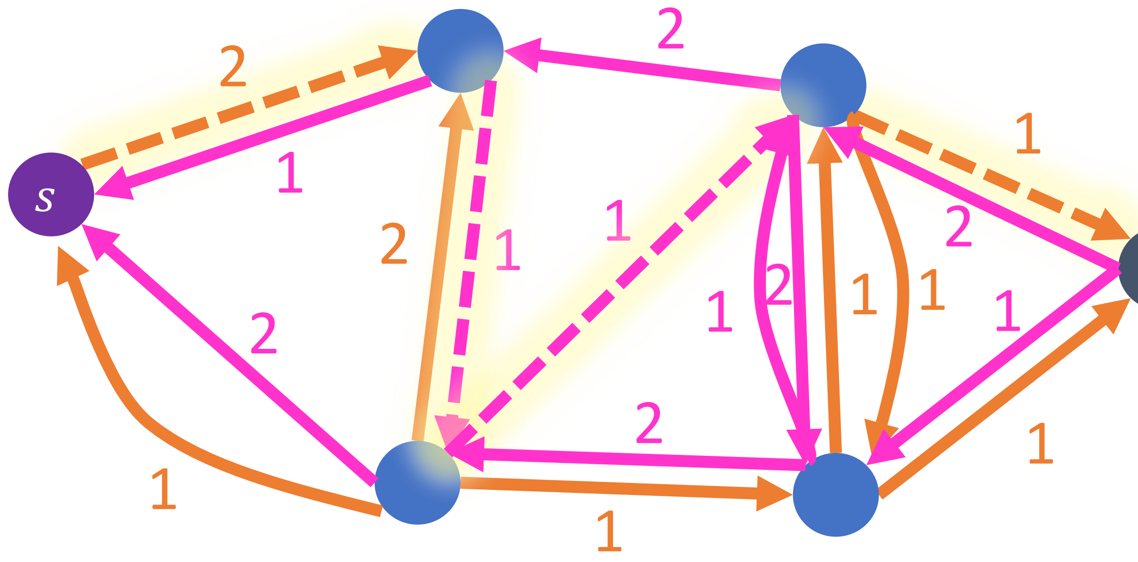
Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

- 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

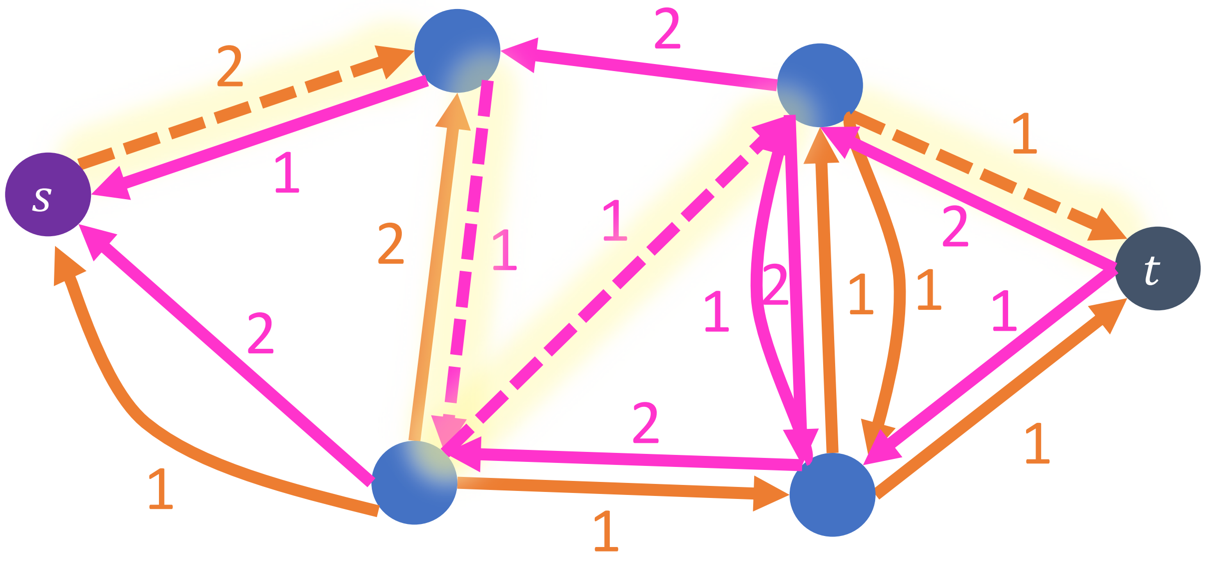
Consider the minimum-weight edge e along the path: we can increase the flow by $w(e)$

- 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 $w(e)$

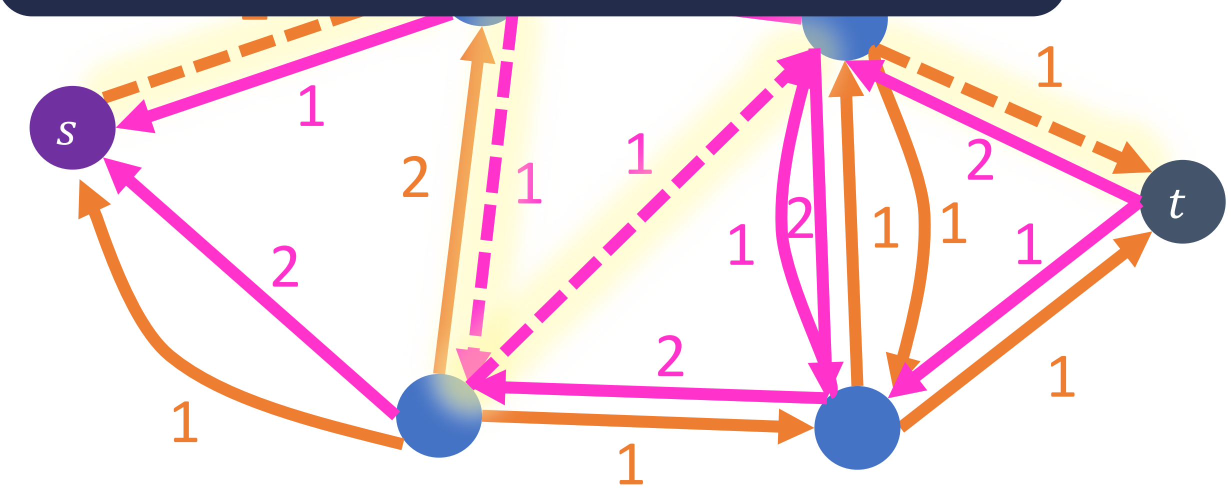
- Send $w(e)$ flow along all forward edges (these have at least $w(e)$ capacity)
- Remove $w(e)$ flow along all backward edges (these contain at least $w(e)$ units of flow)

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

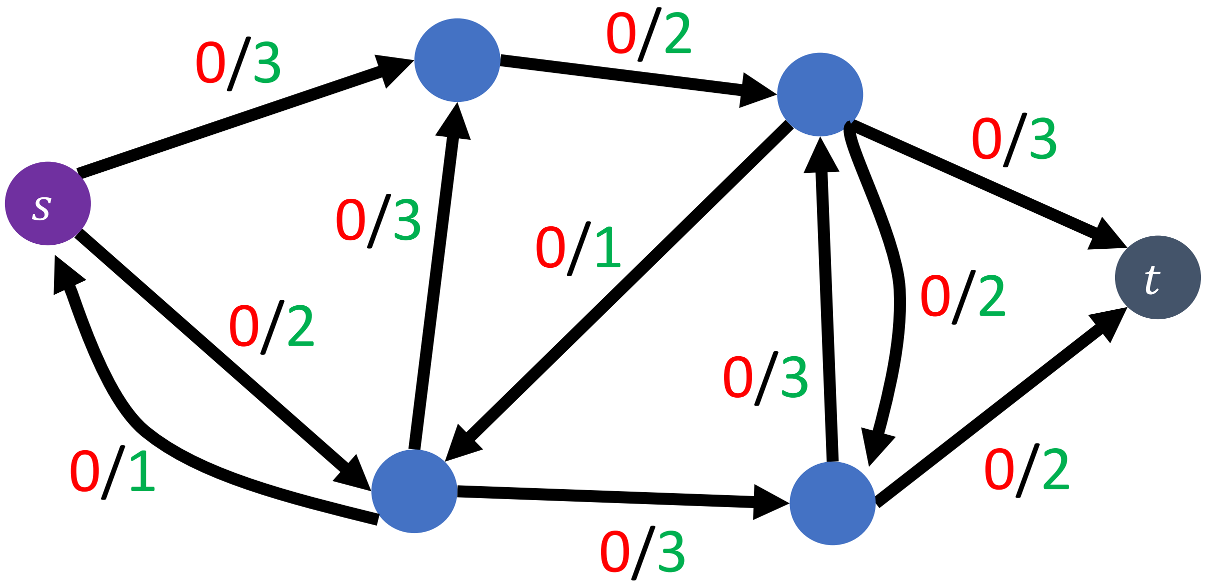
Define an <u>augmenting path</u> to be an $s \to t$ path in the residual graph G_f (using edges of non-zero weight)

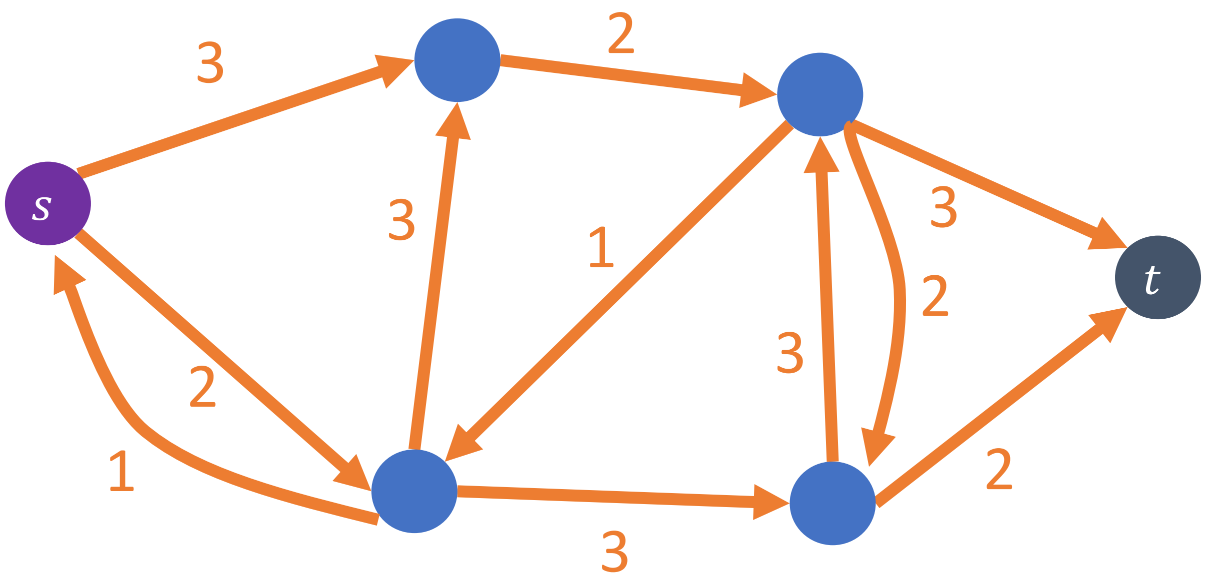
Ford-Fulkerson max-flow algorithm:

- 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 <u>any</u> augmenting path (will revisit this later)

Ford-Fulkerson Example



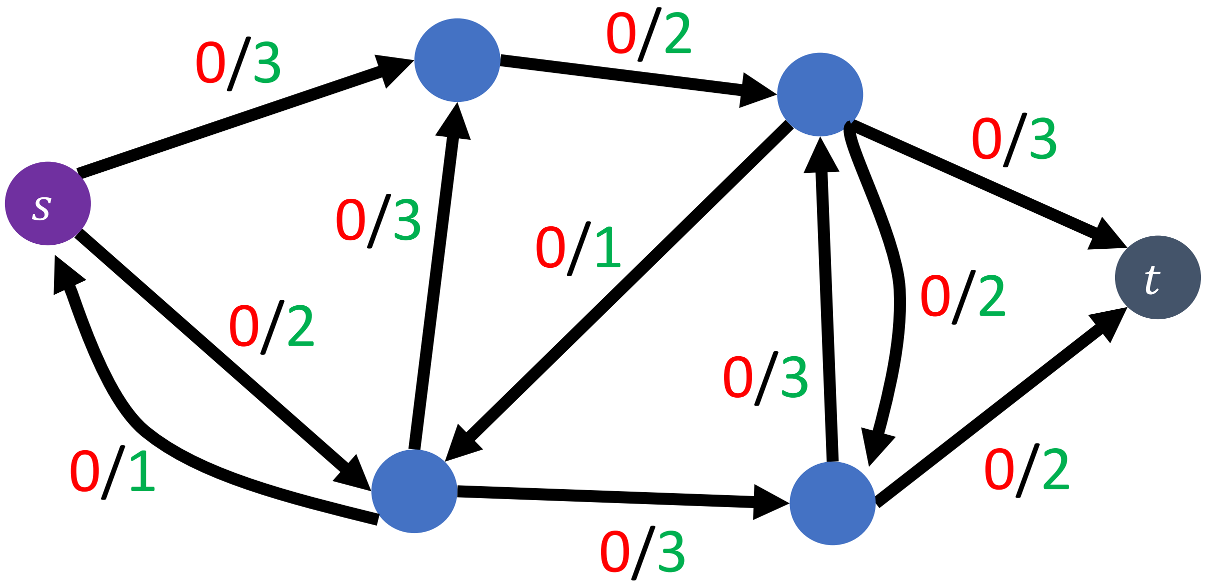


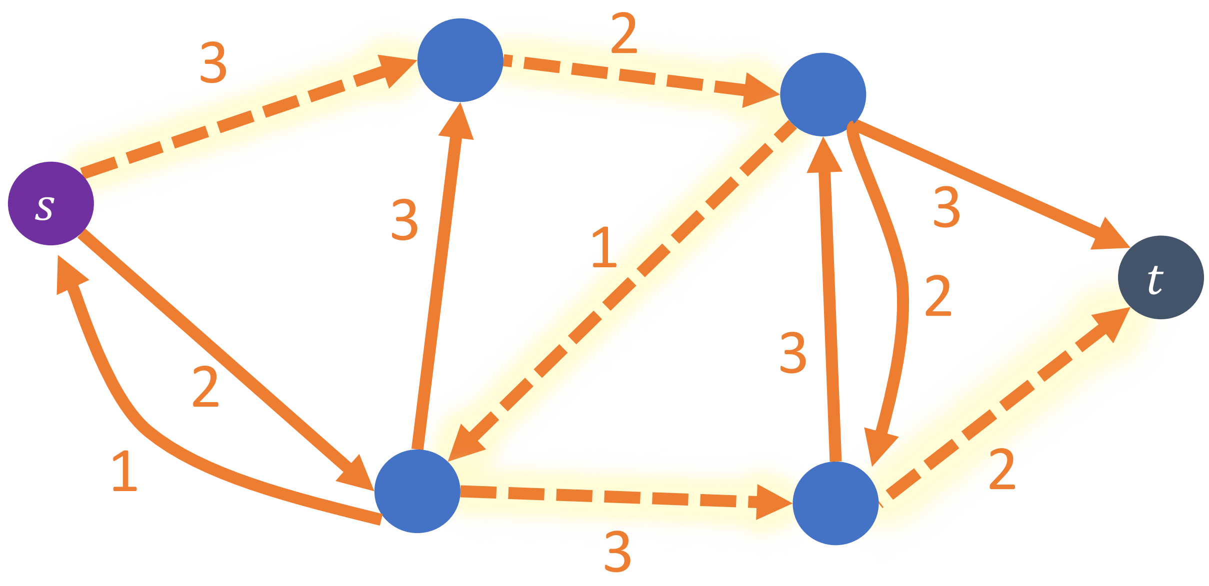
Initially: $f(e) = 0$ for all $e \in E$

Residual graph G_f

Ford-Fulkerson Example

Increase flow by 1 unit

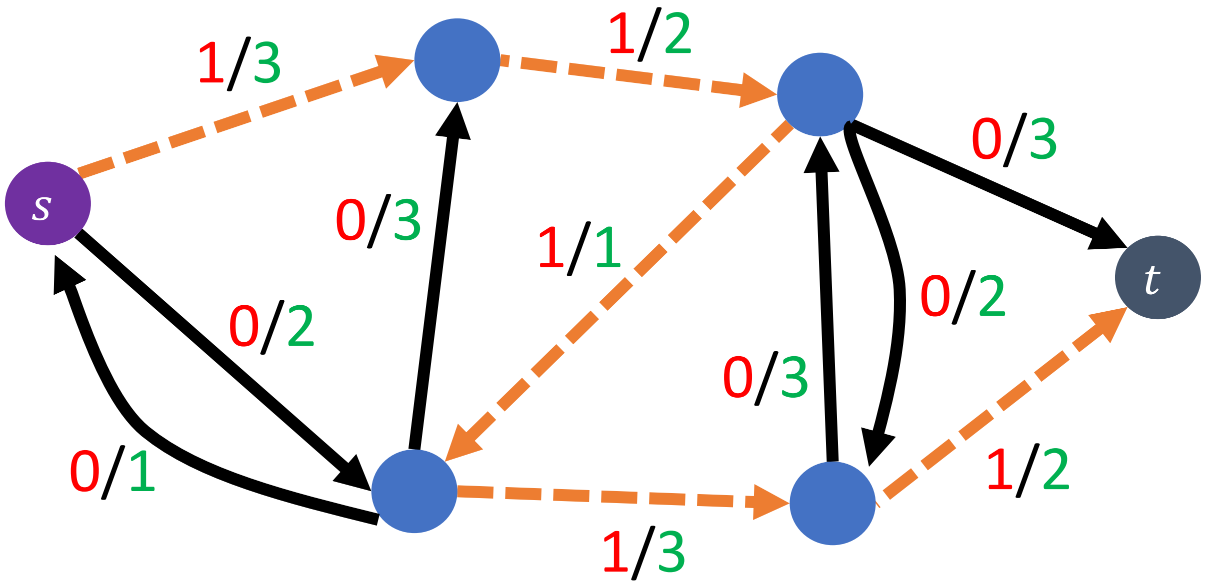


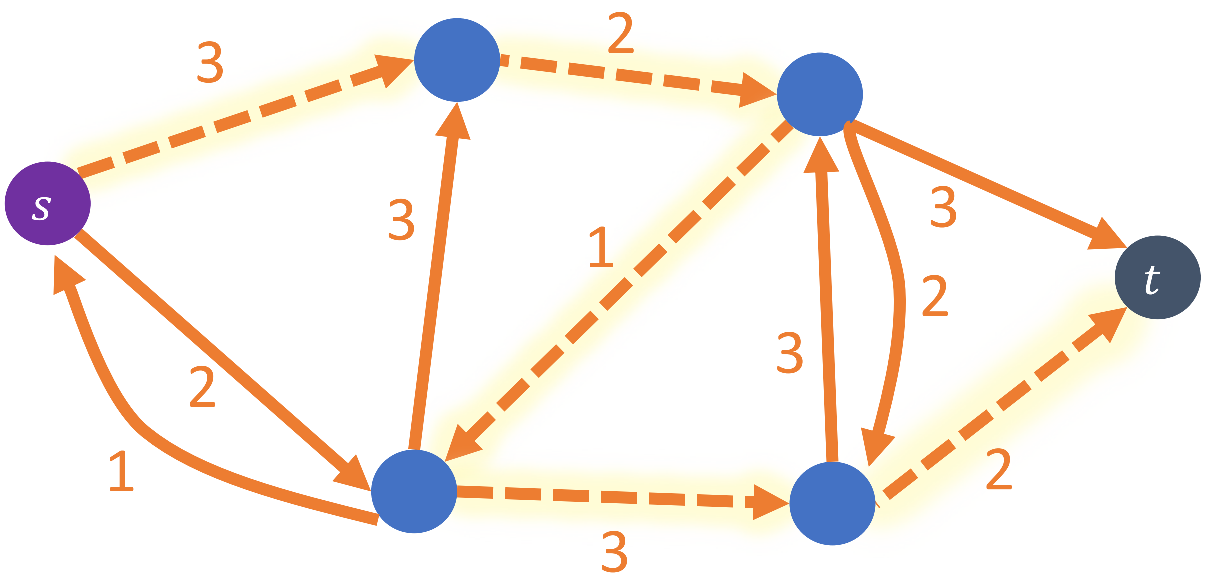


Residual graph G_f

Ford-Fulkerson Example

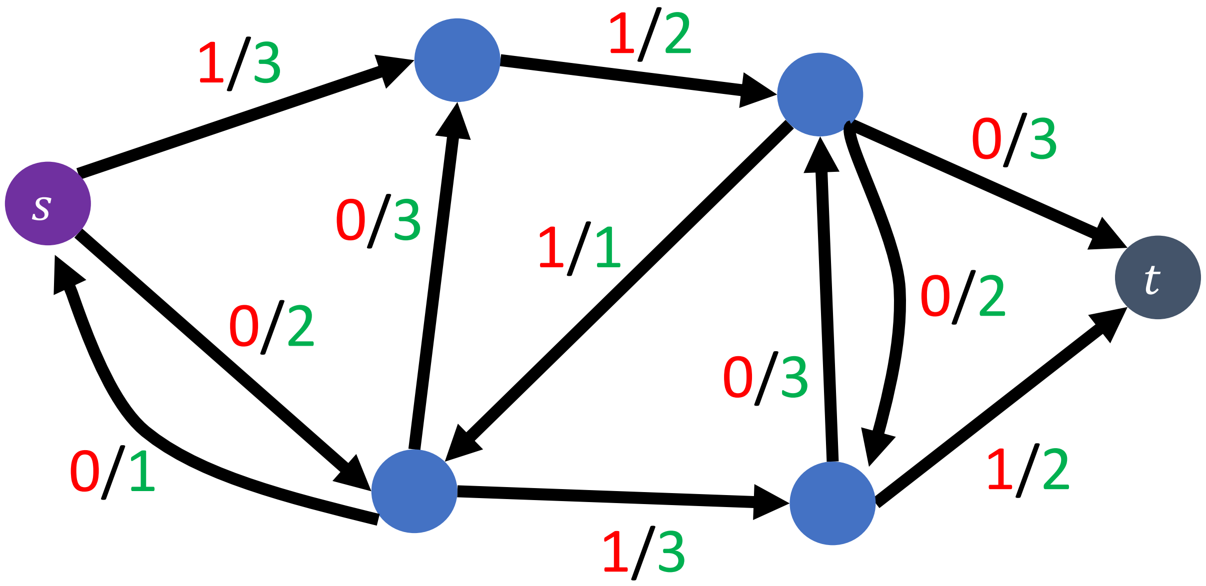
Increase flow by 1 unit

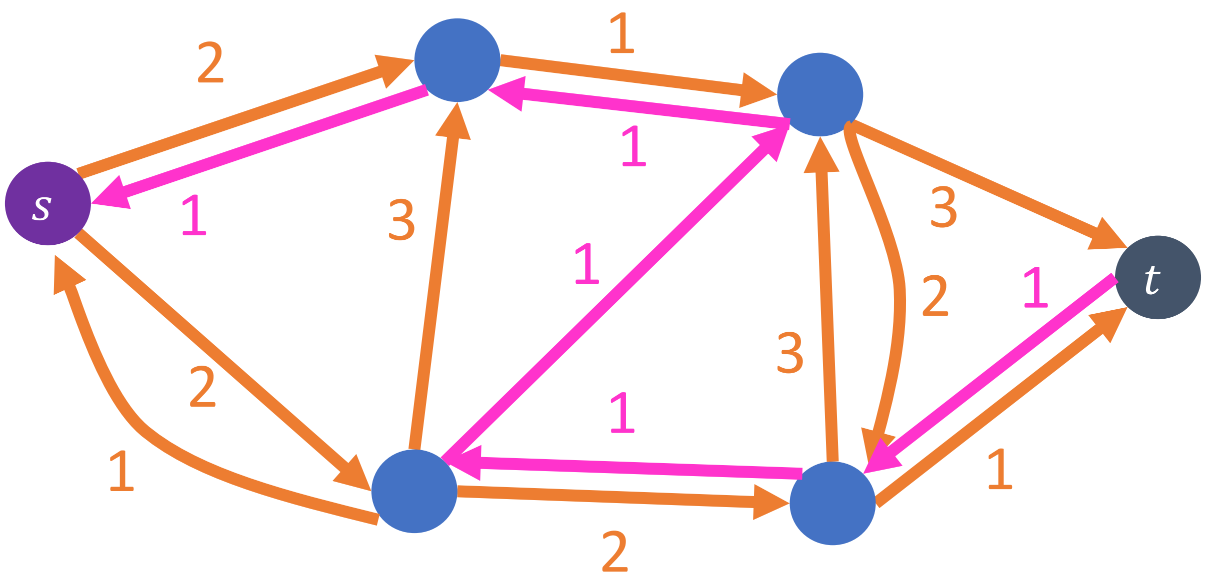




Residual graph G_f

Ford-Fulkerson Example

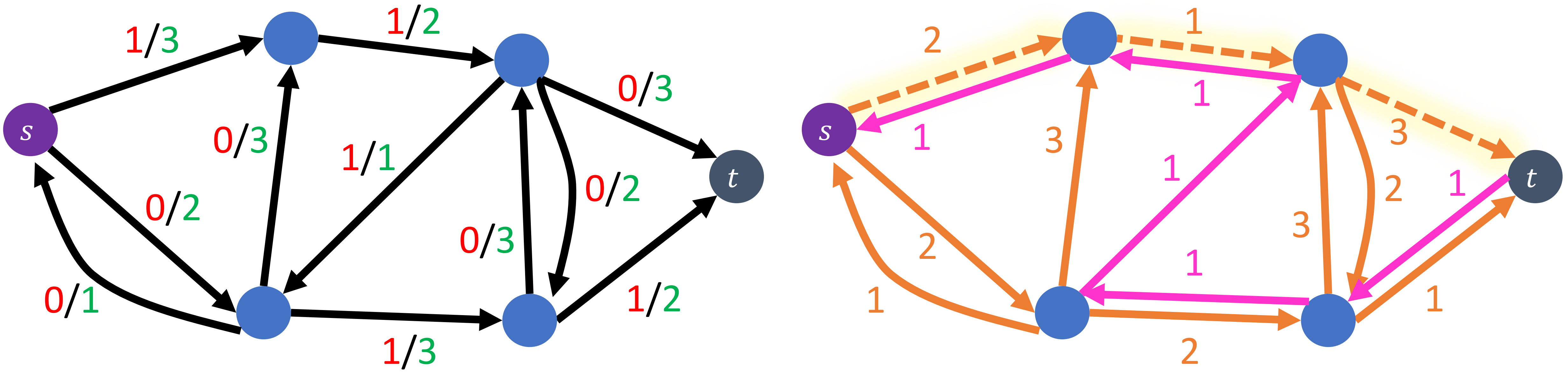




Residual graph G_f

Ford-Fulkerson Example

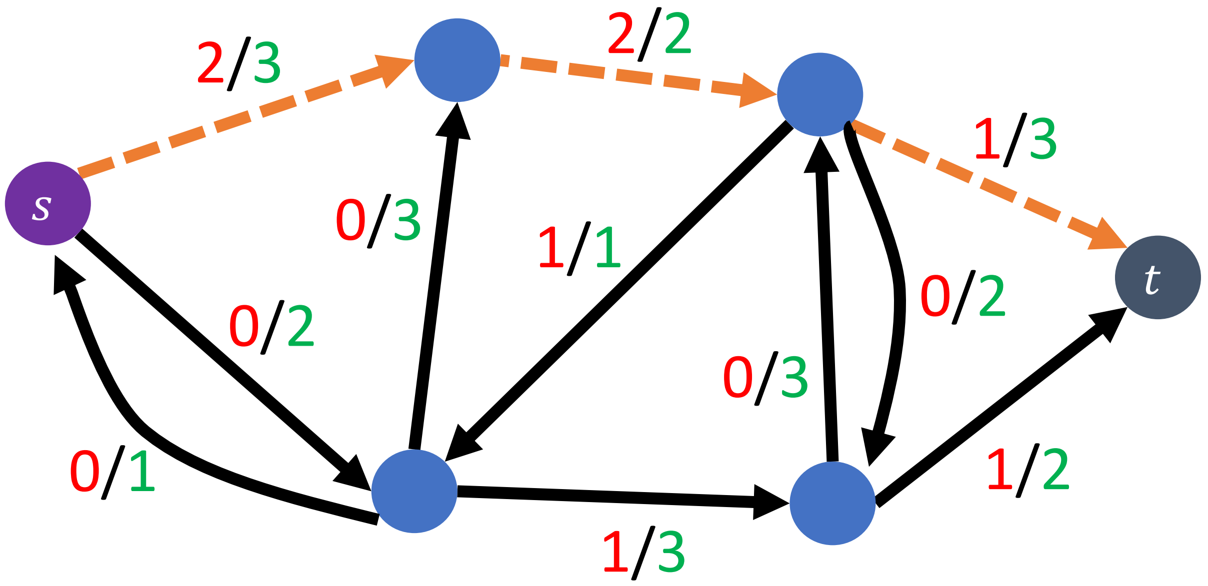
Increase flow by 1 unit

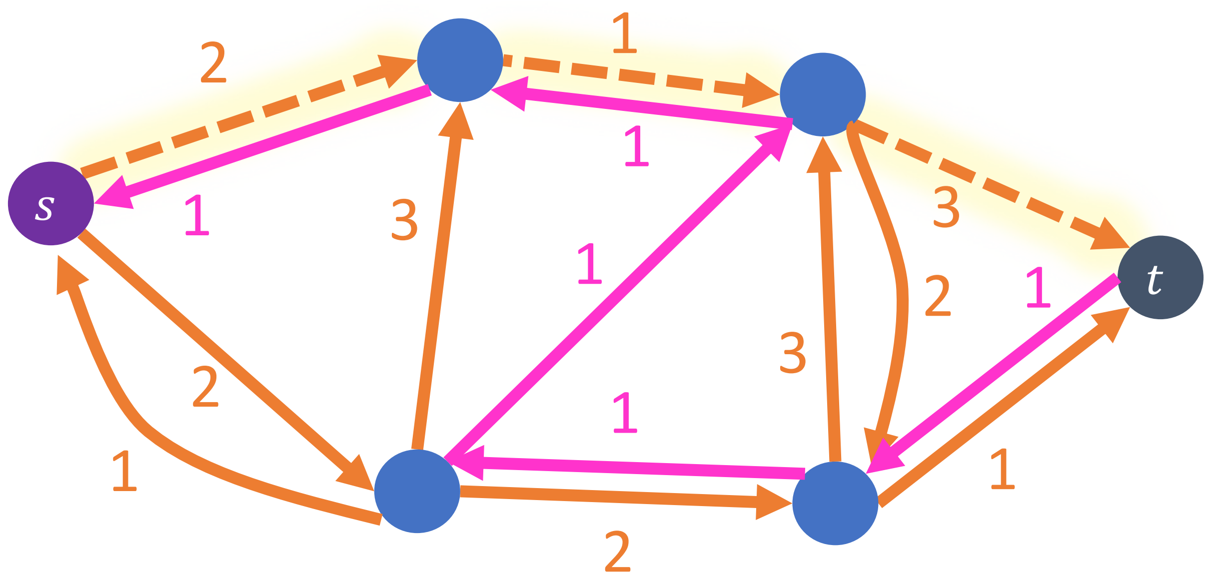


Residual graph G_f

Ford-Fulkerson Example

Increase flow by 1 unit

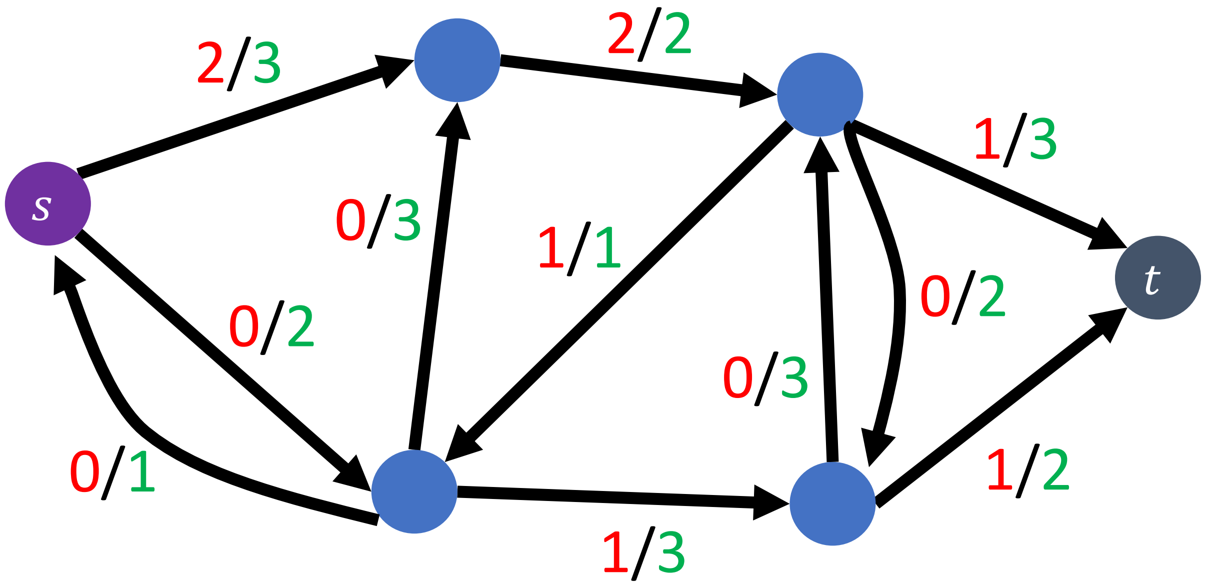


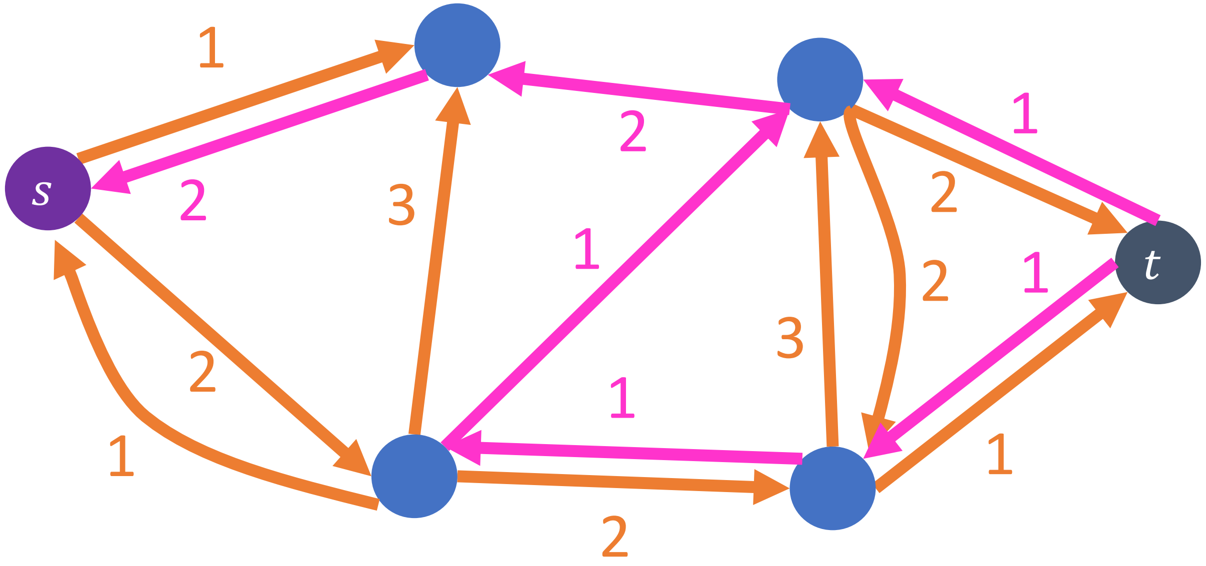


Residual graph G_f

Ford-Fulkerson Example

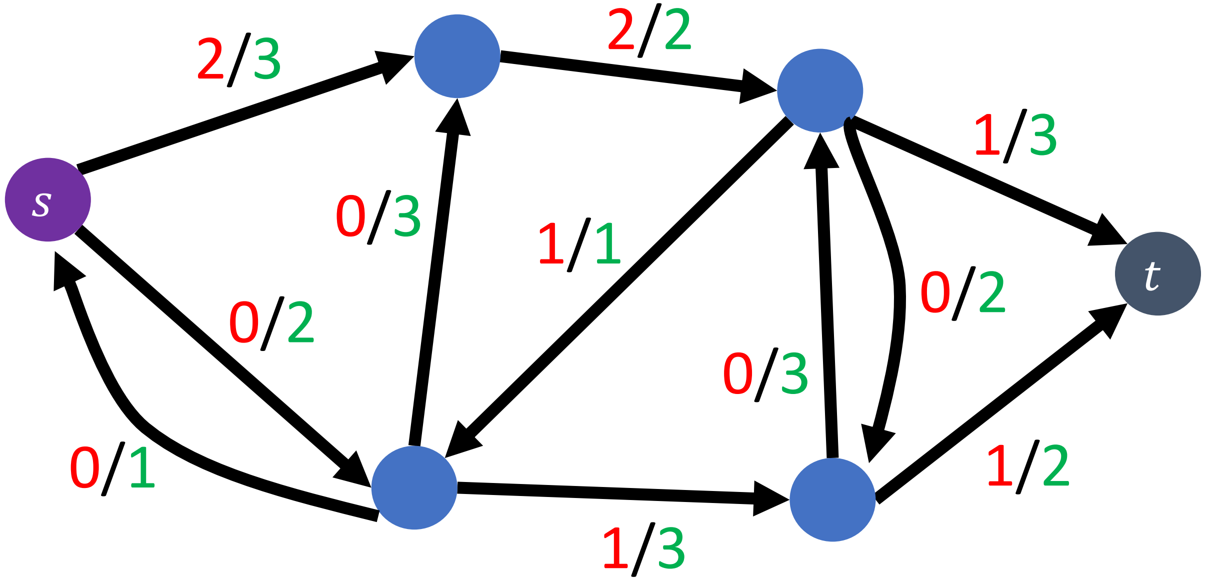
Increase flow by 1 unit

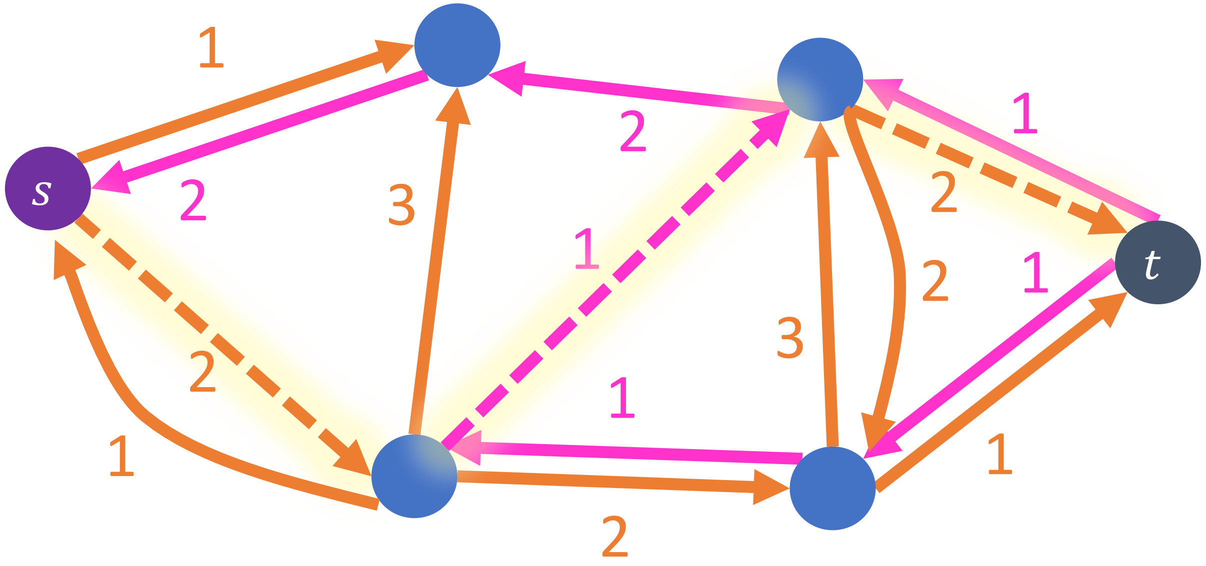




Residual graph G_f

Ford-Fulkerson Example

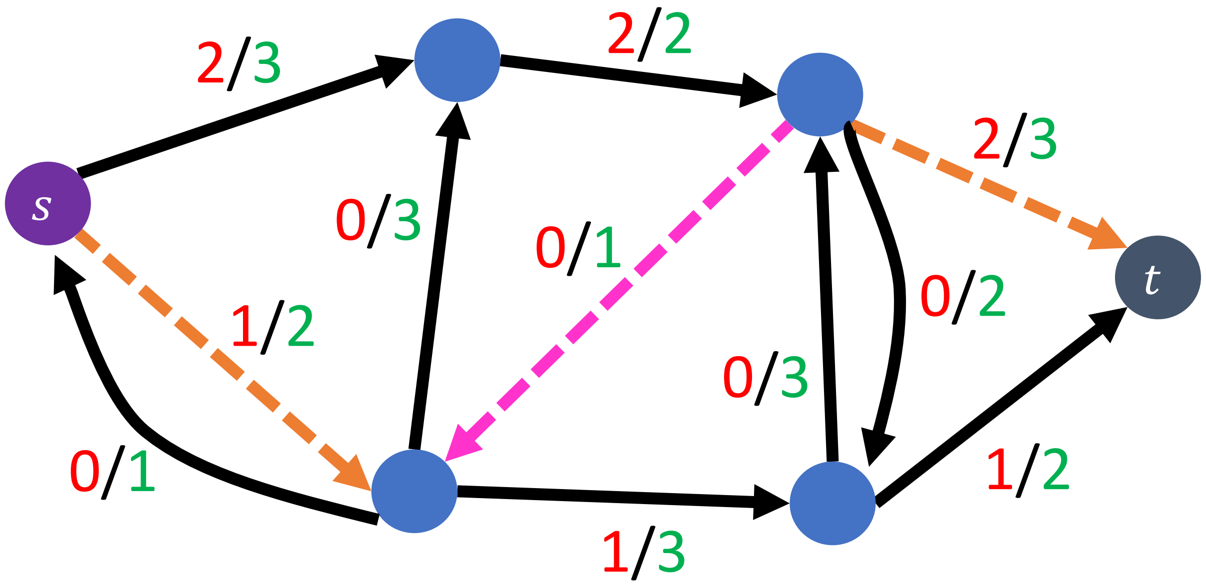


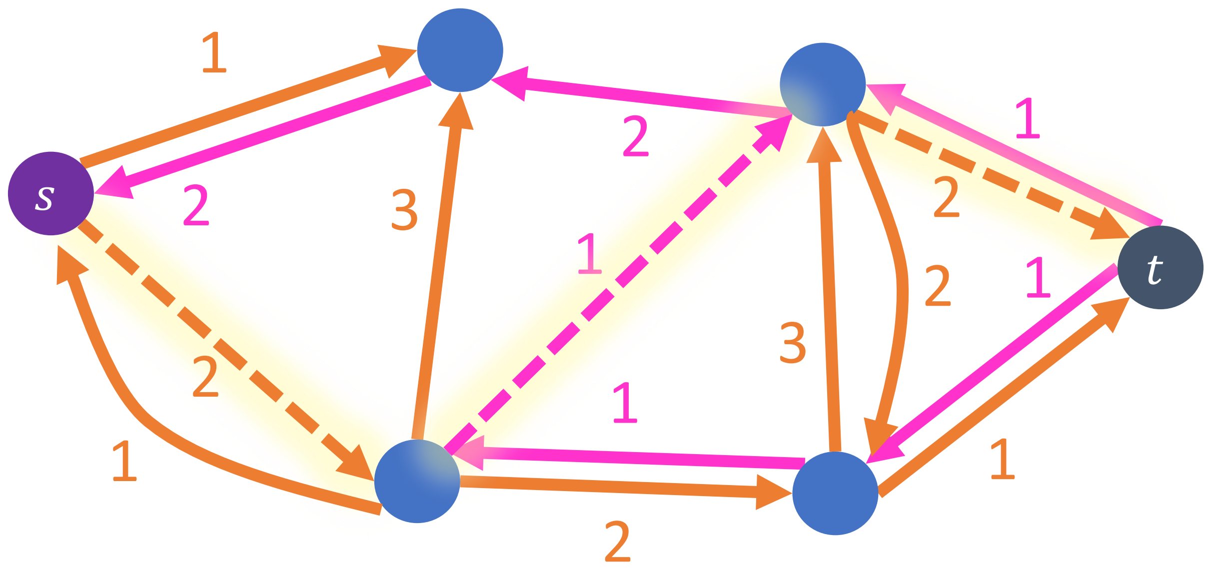


Residual graph G_f

Ford-Fulkerson Example

Increase flow by 1 unit

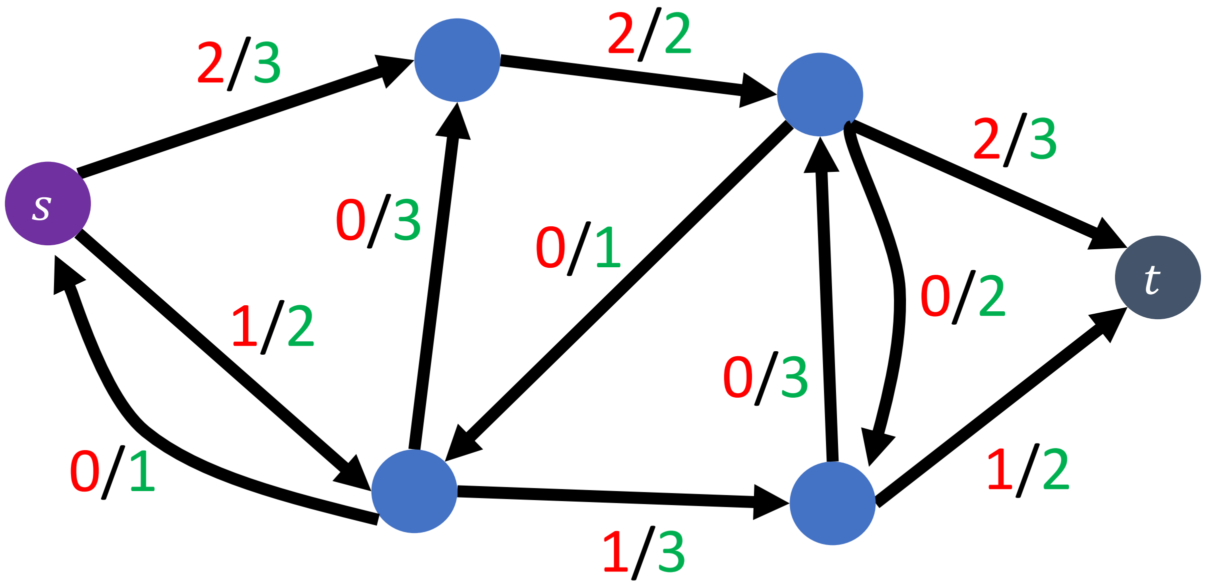


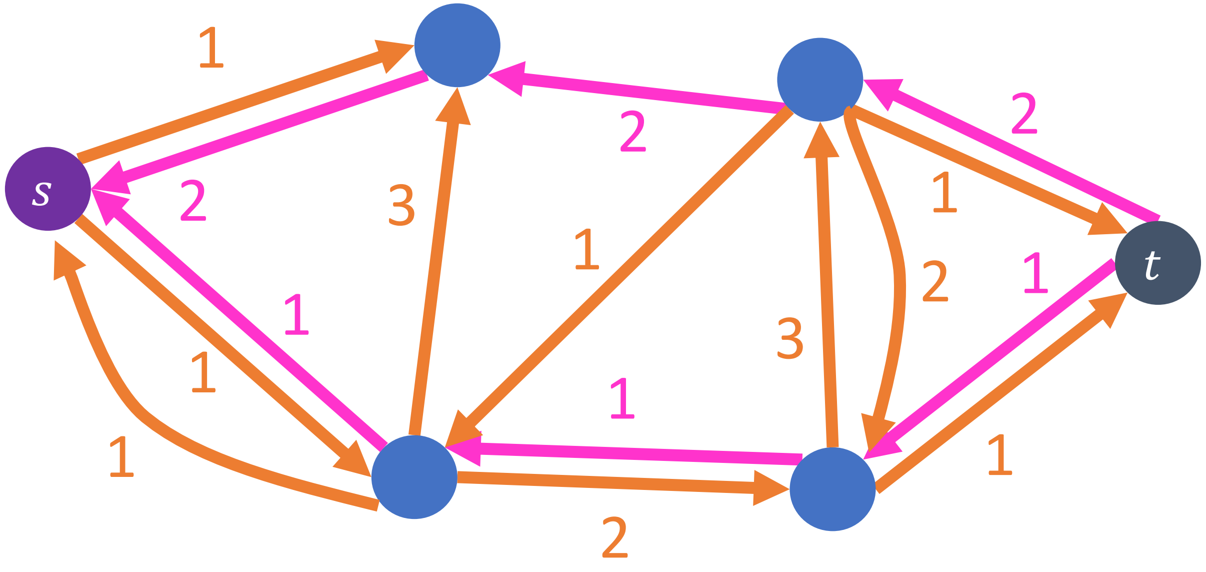


Residual graph G_f

Ford-Fulkerson Example

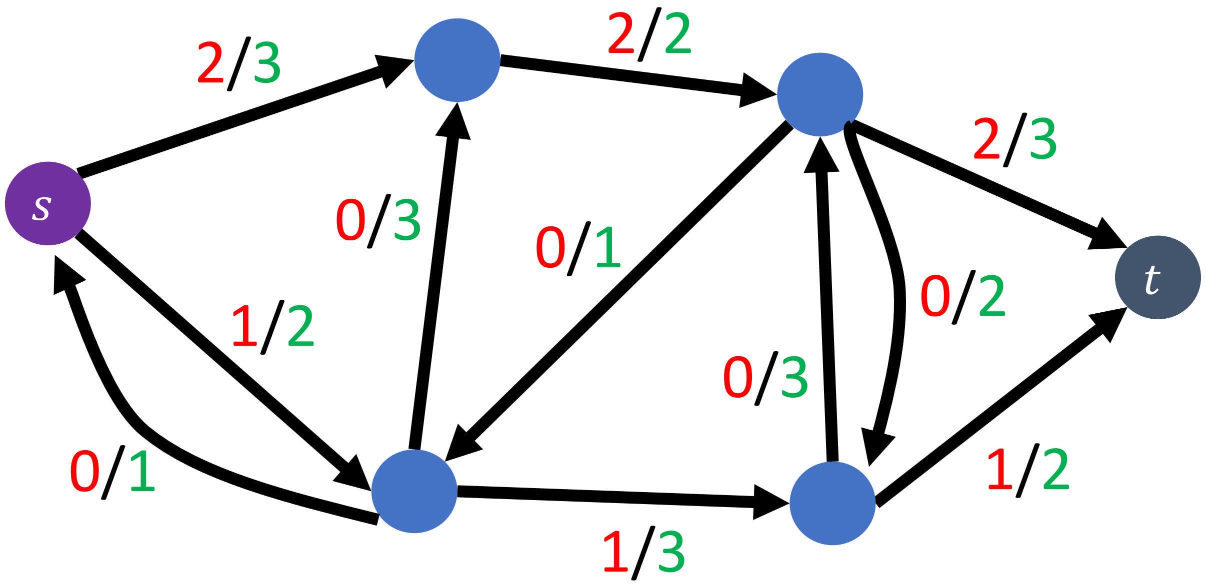
Increase flow by 1 unit

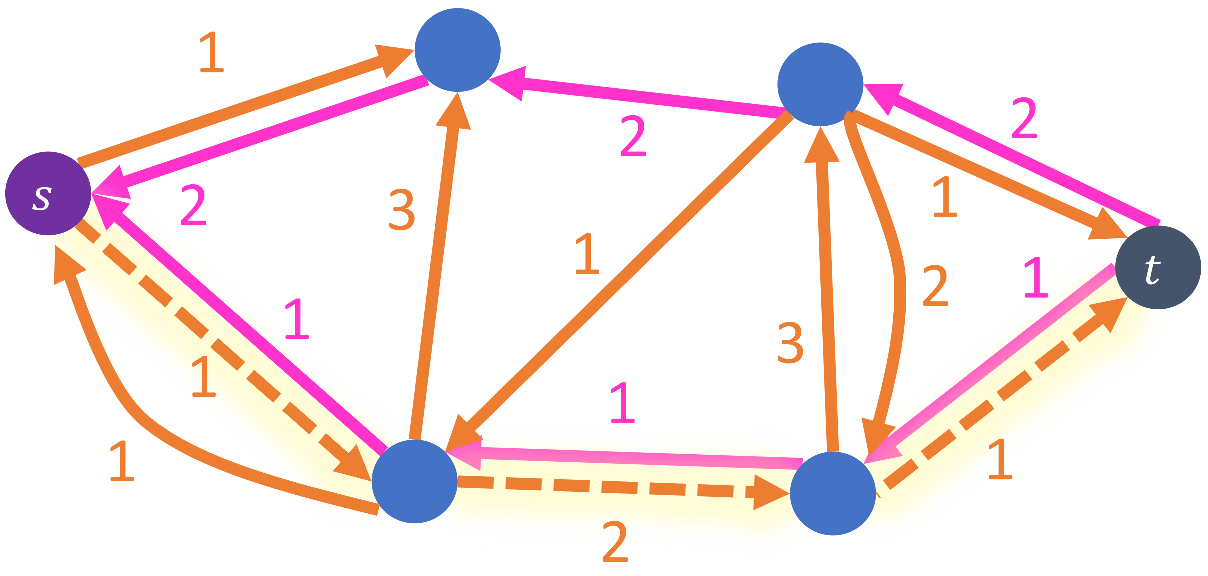




Residual graph G_f

Ford-Fulkerson Example

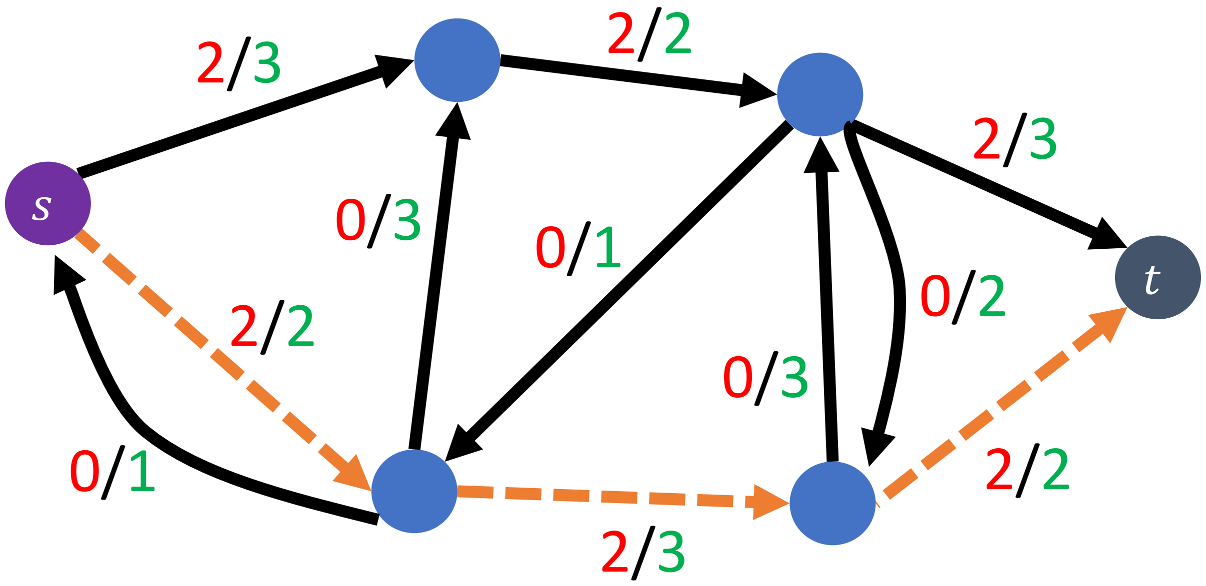


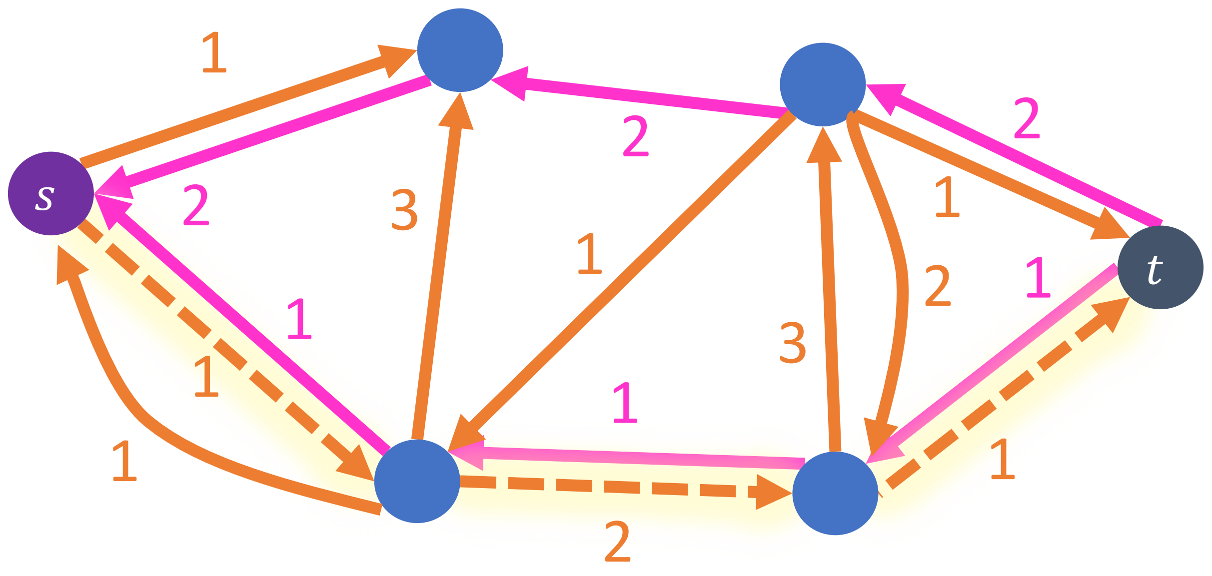


Residual graph G_f

Ford-Fulkerson Example

Increase flow by 1 unit

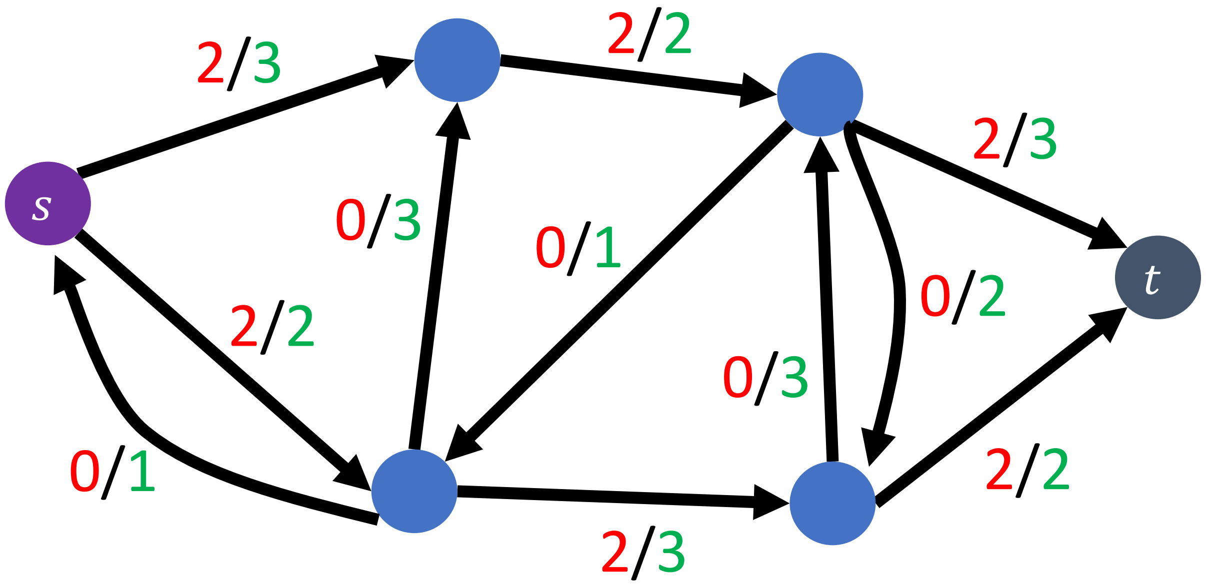


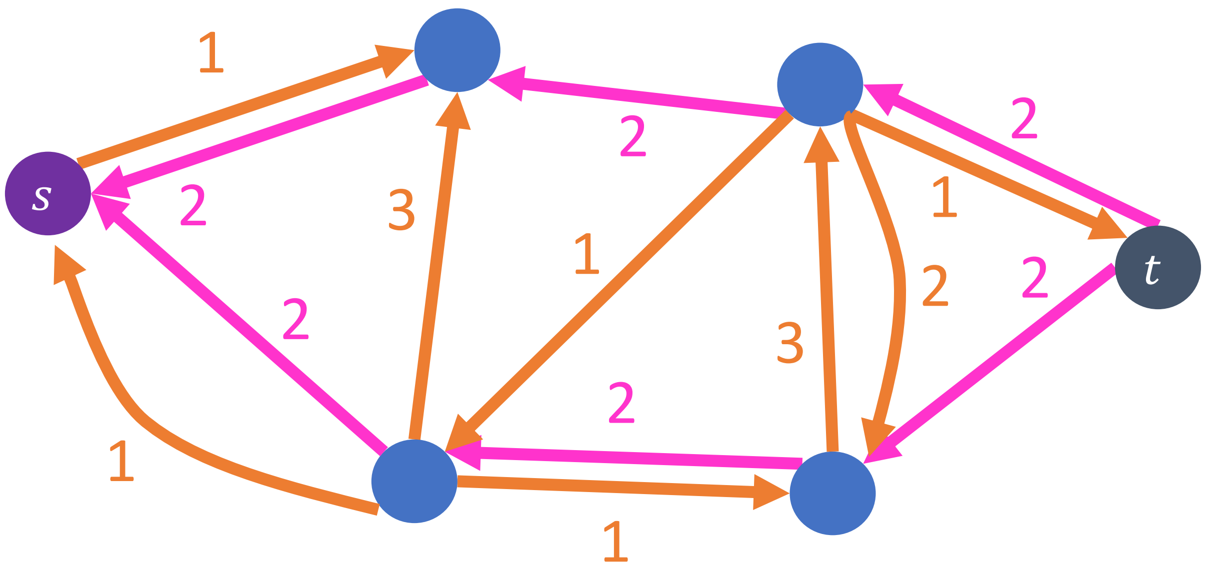


Residual graph G_f

Ford-Fulkerson Example

No more augmenting paths





Residual graph G_f

Maximum flow: 4

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 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 $s \to t$ 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

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!

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

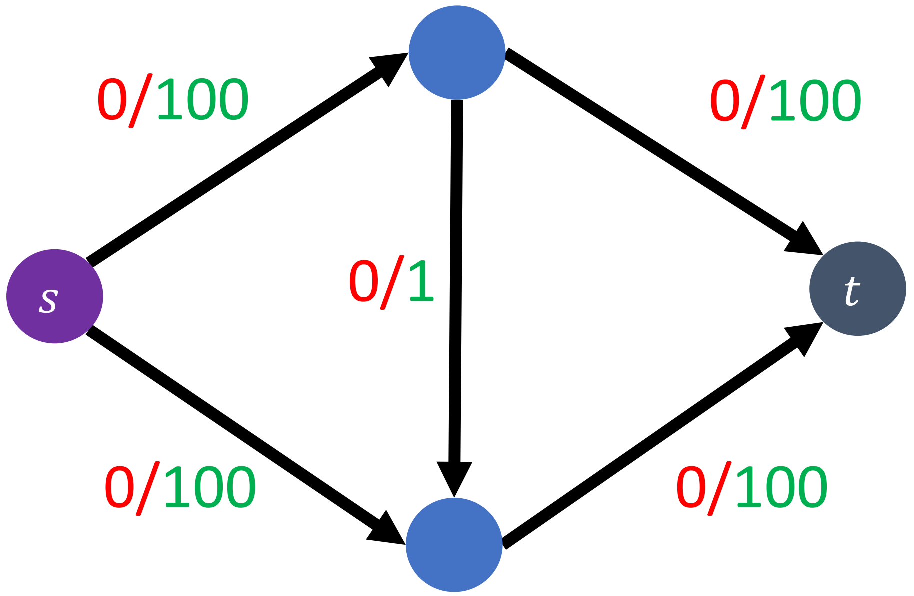
For graphs with integer capacities, running time of Ford-Fulkerson is

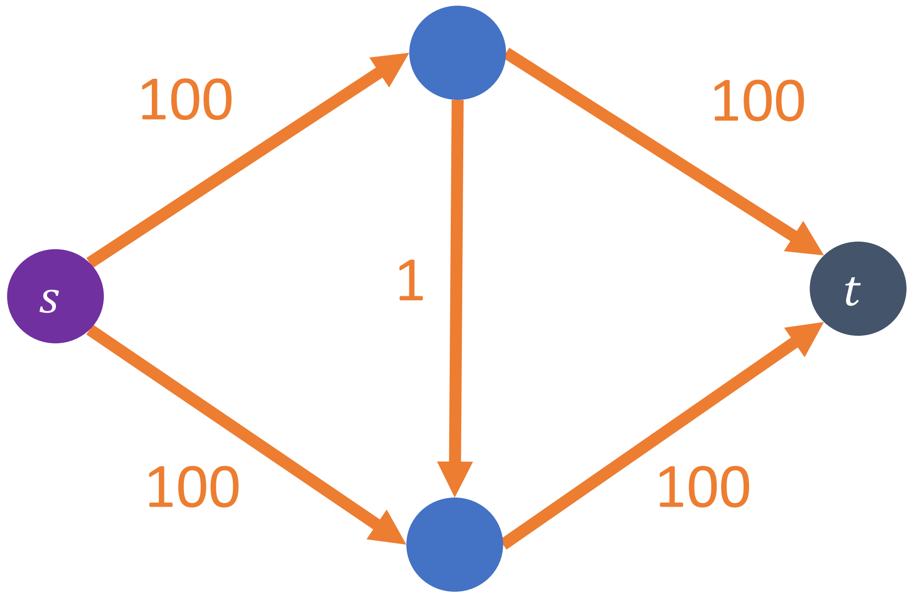
$$O(|f^*| \cdot |E|)$$

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

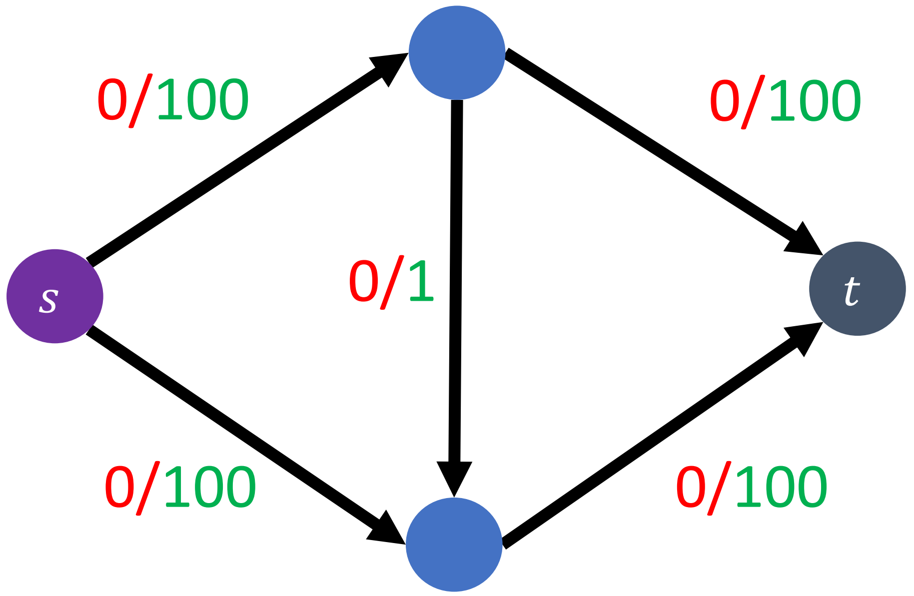
As described, algorithm is <u>not</u> polynomial-time!

Worst-Case Ford-Fulkerson

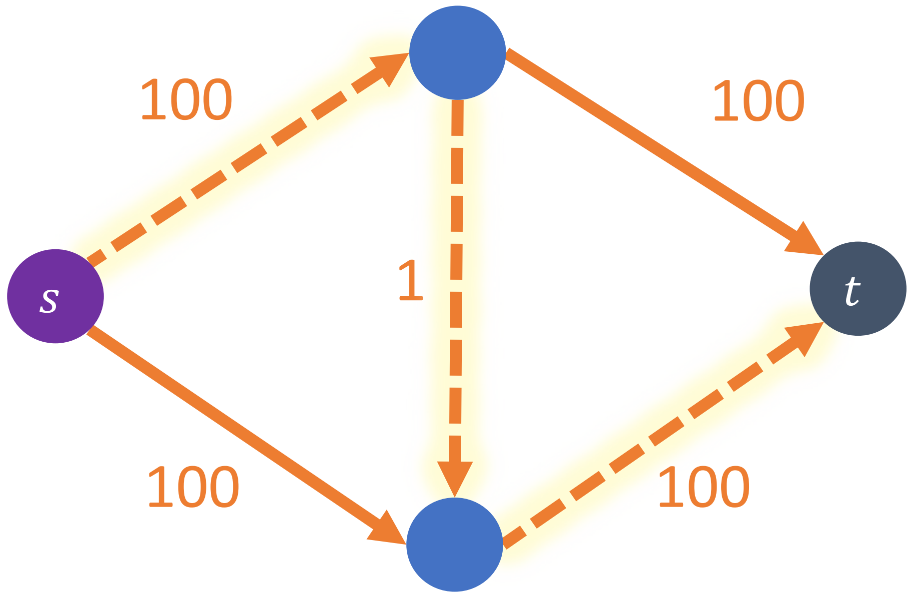




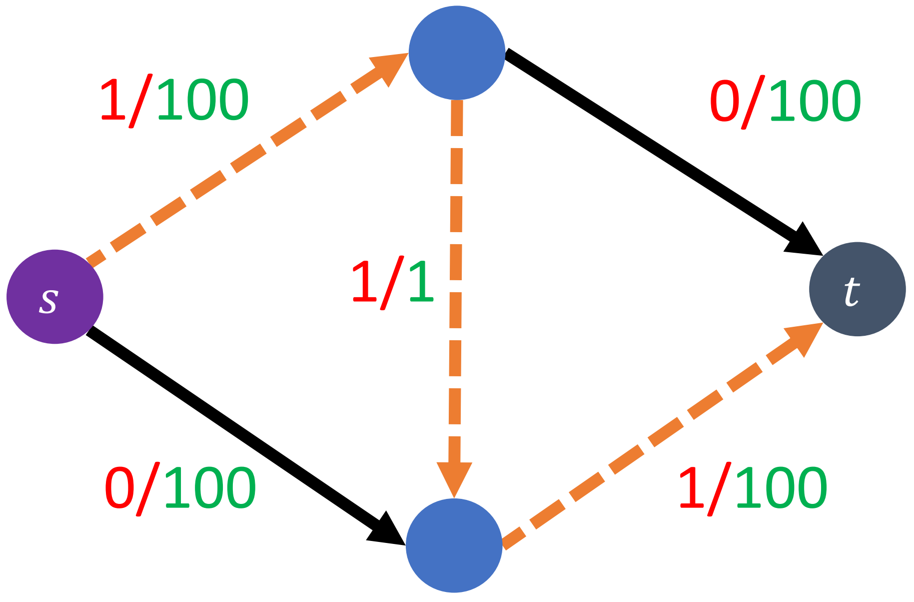
Worst-Case Ford-Fulkerson



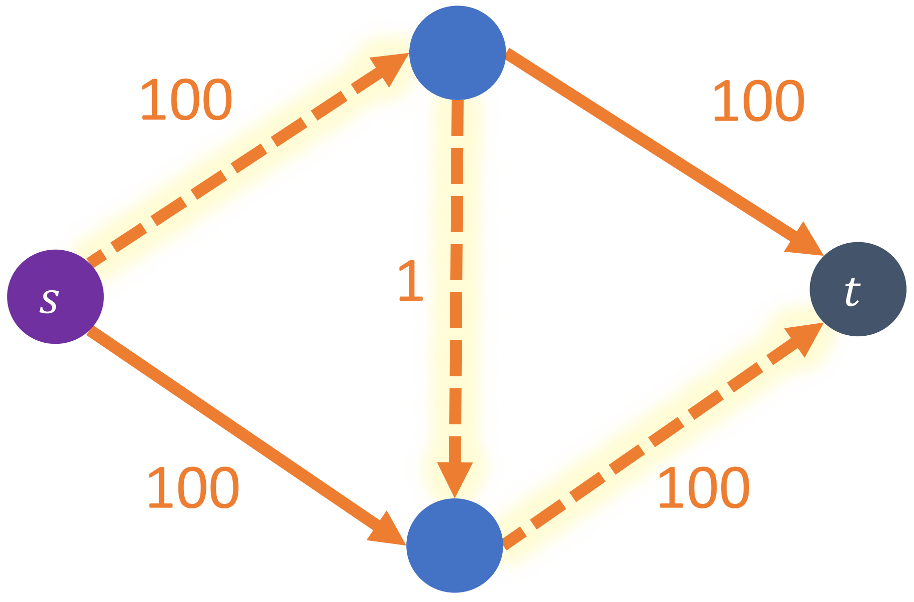
Increase flow by 1 unit



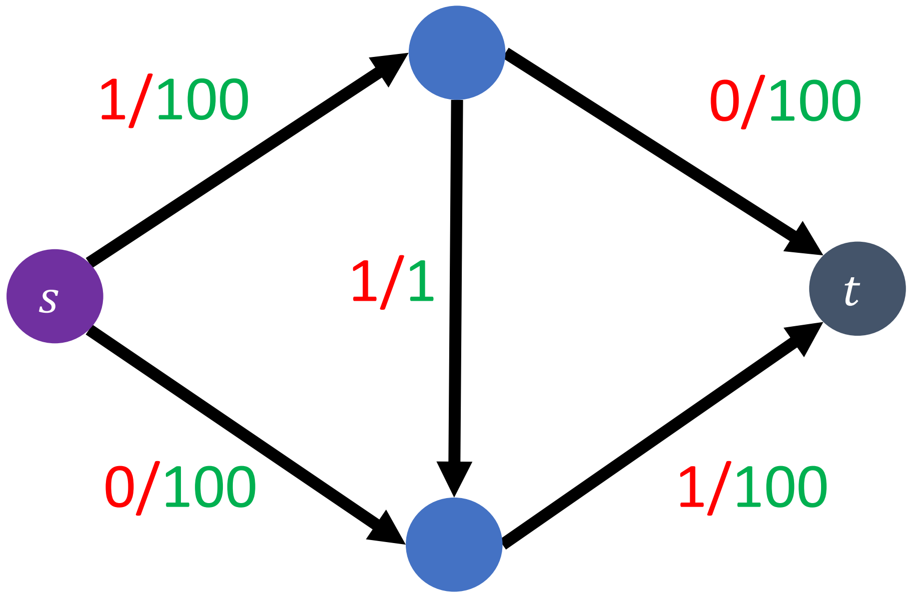
Worst-Case Ford-Fulkerson

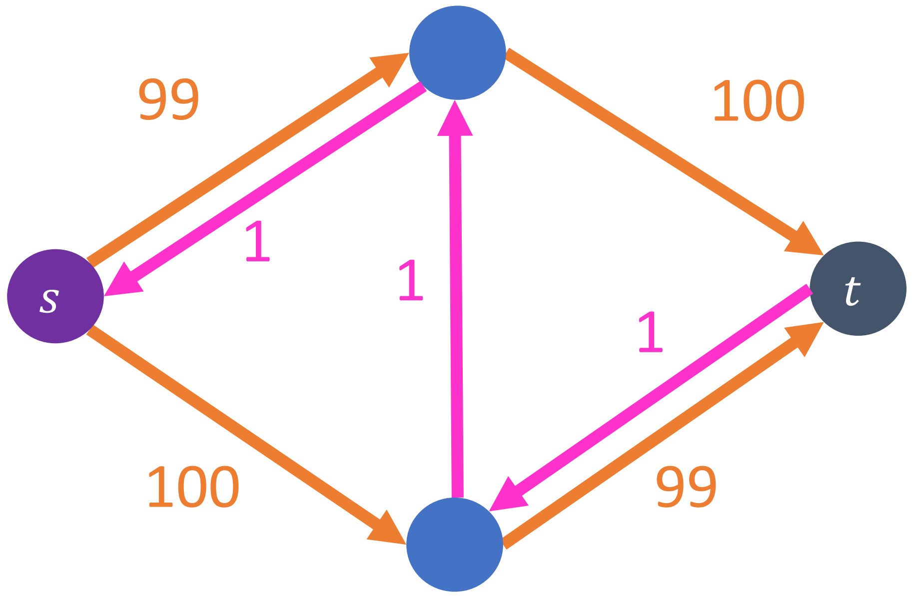


Increase flow by 1 unit

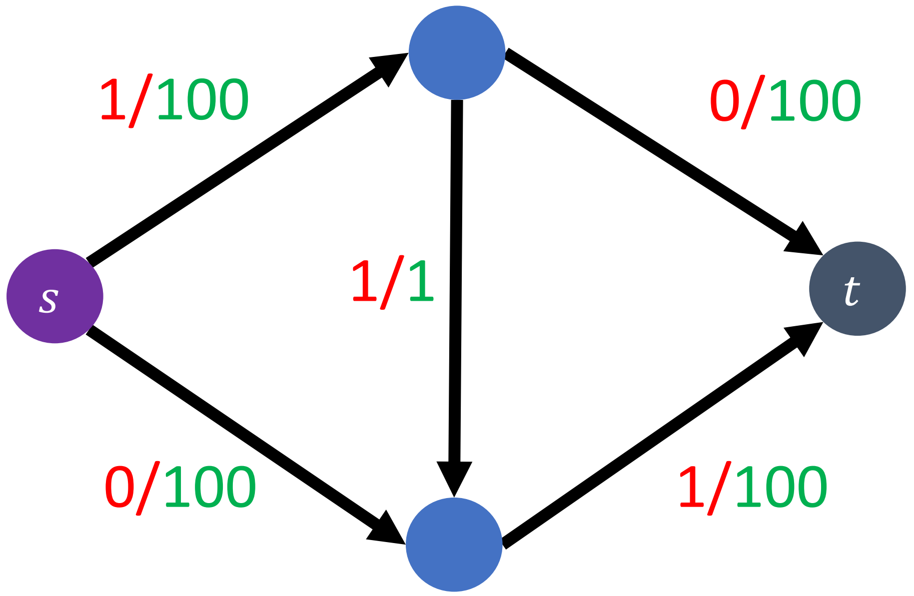


Worst-Case Ford-Fulkerson

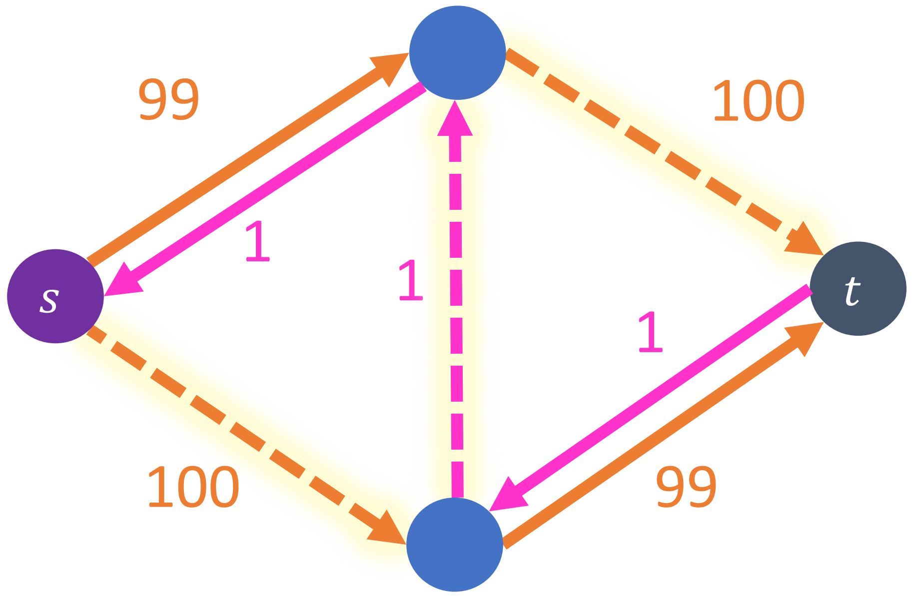




Worst-Case Ford-Fulkerson

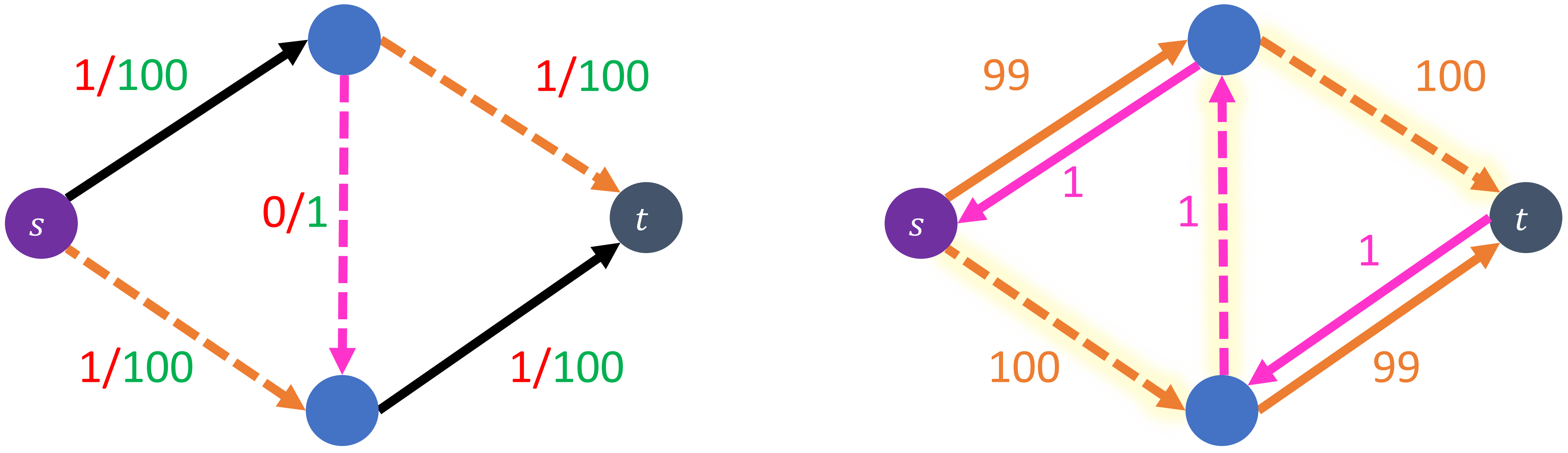


Increase flow by 1 unit

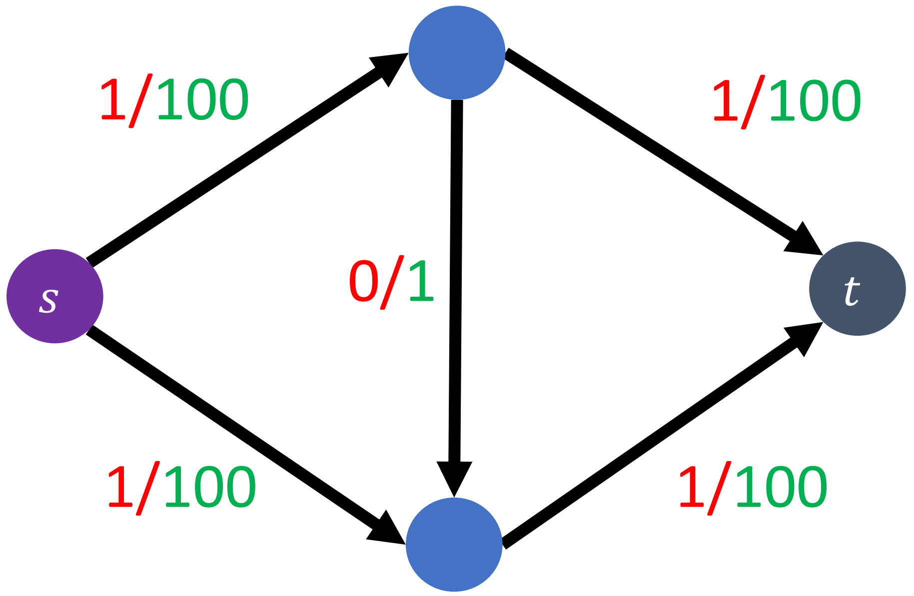


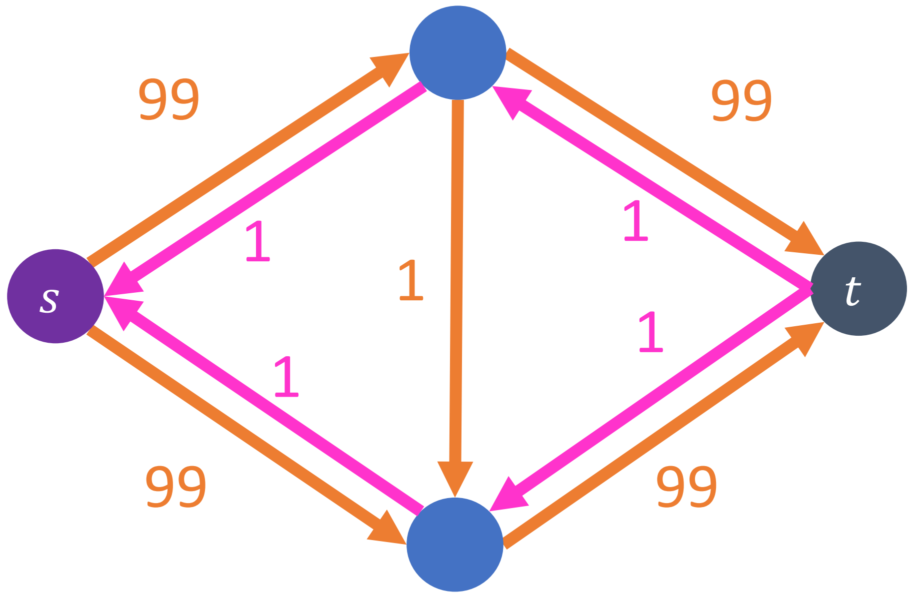
Worst-Case Ford-Fulkerson

Increase flow by 1 unit

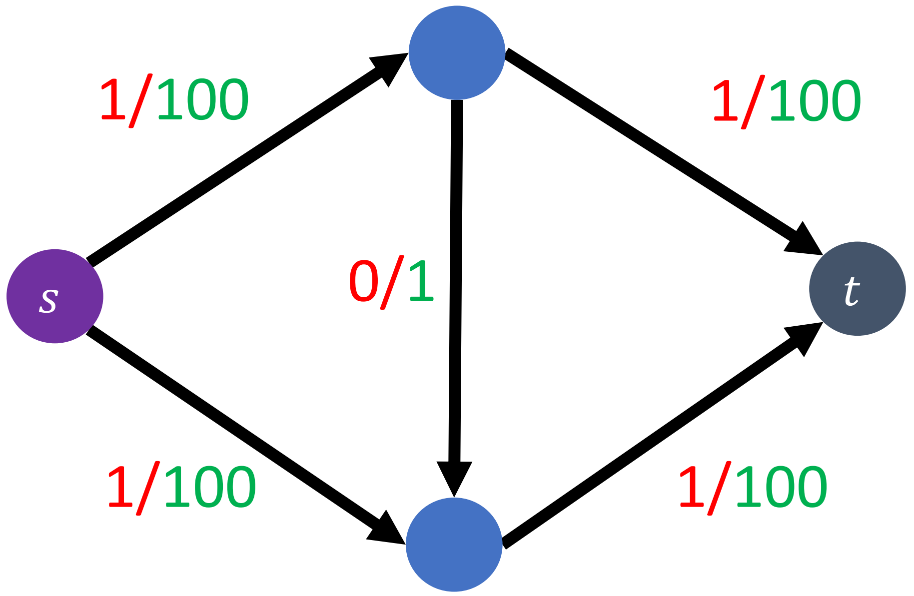


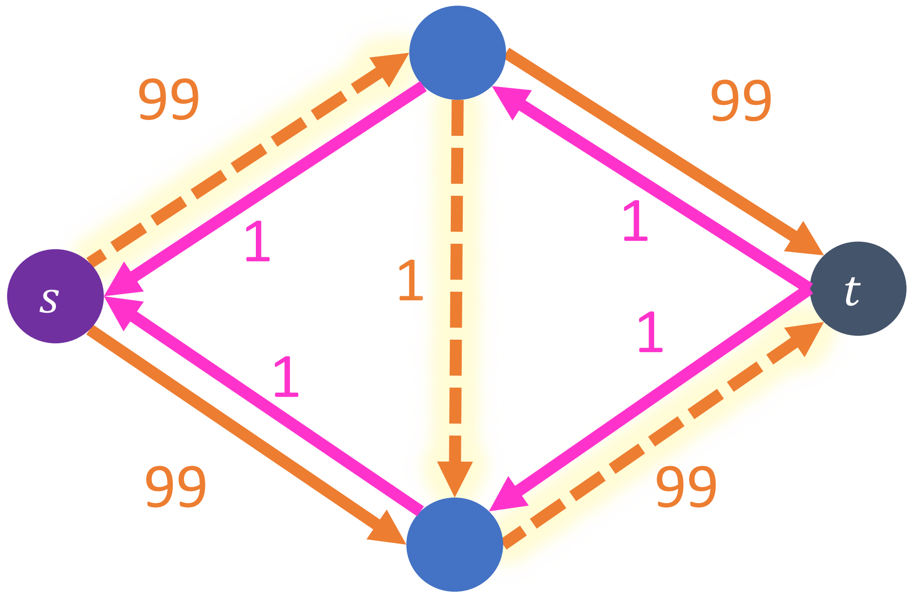
Worst-Case Ford-Fulkerson





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

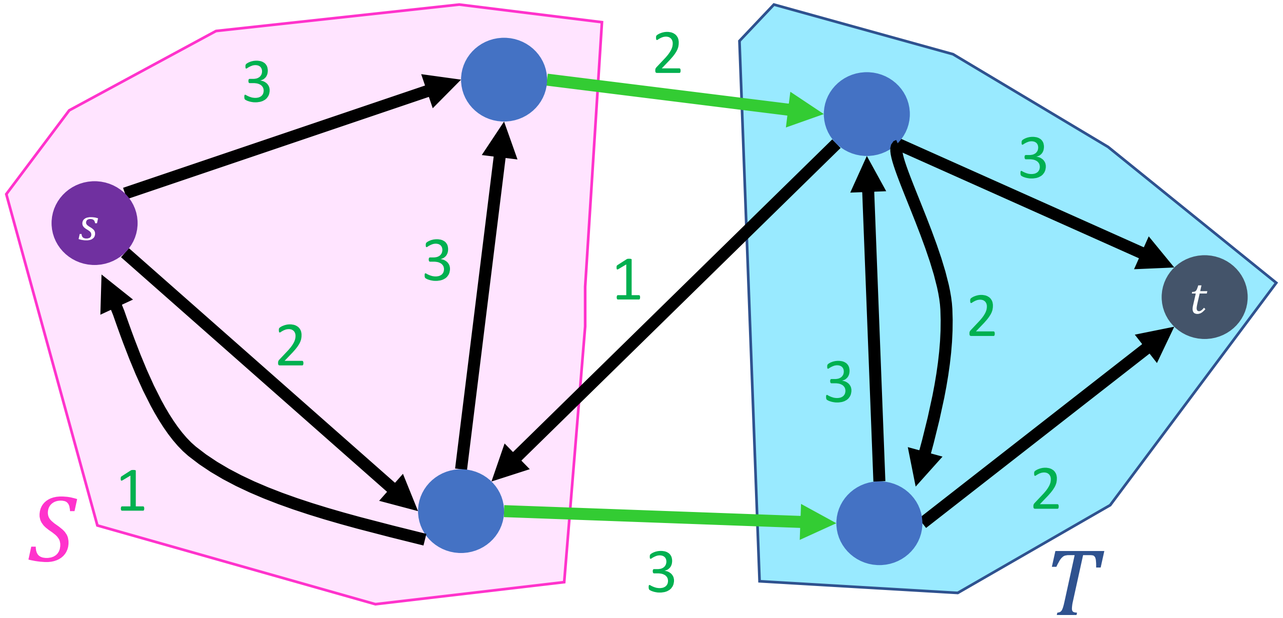
Proof: See CLRS (Chapter 26.2)

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

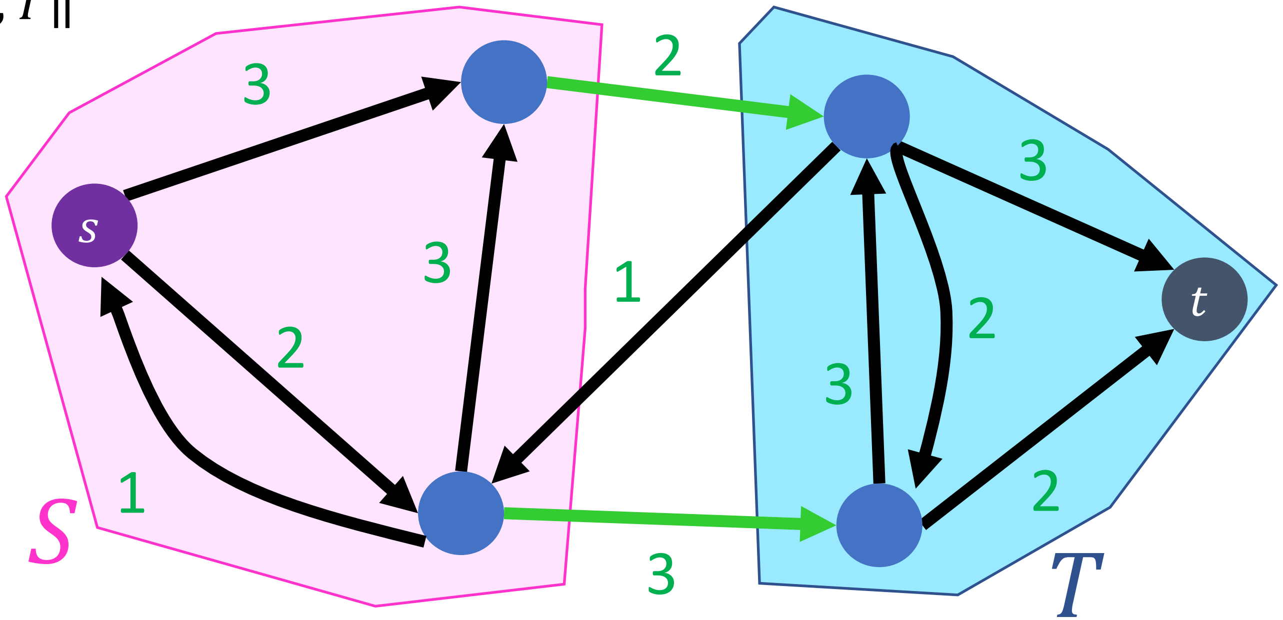
Claim: Maximum flow in a flow network G always upper-bounded by the cost any cut that separates s and t

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

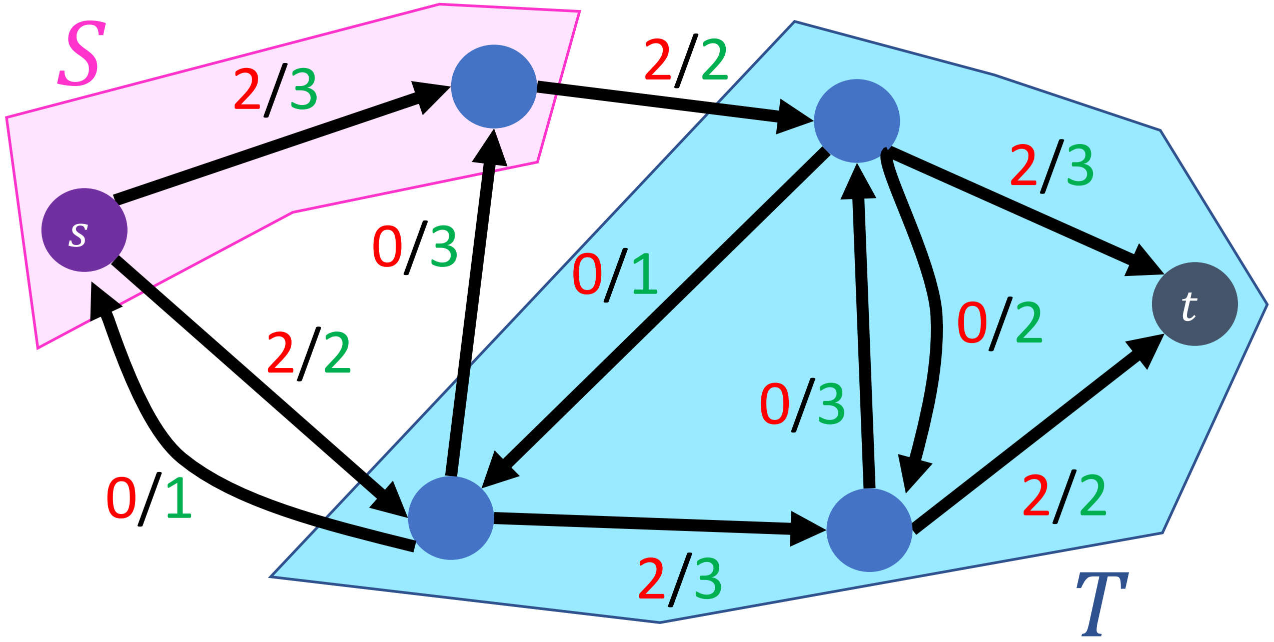
f is a maximum flow in G $\Rightarrow$ there are no augmenting paths in the residual graph G_f $\Rightarrow$ there exists a cut (S, T) of G where $|f| = \|S, T\|$

Statements are equivalent!

Implications:

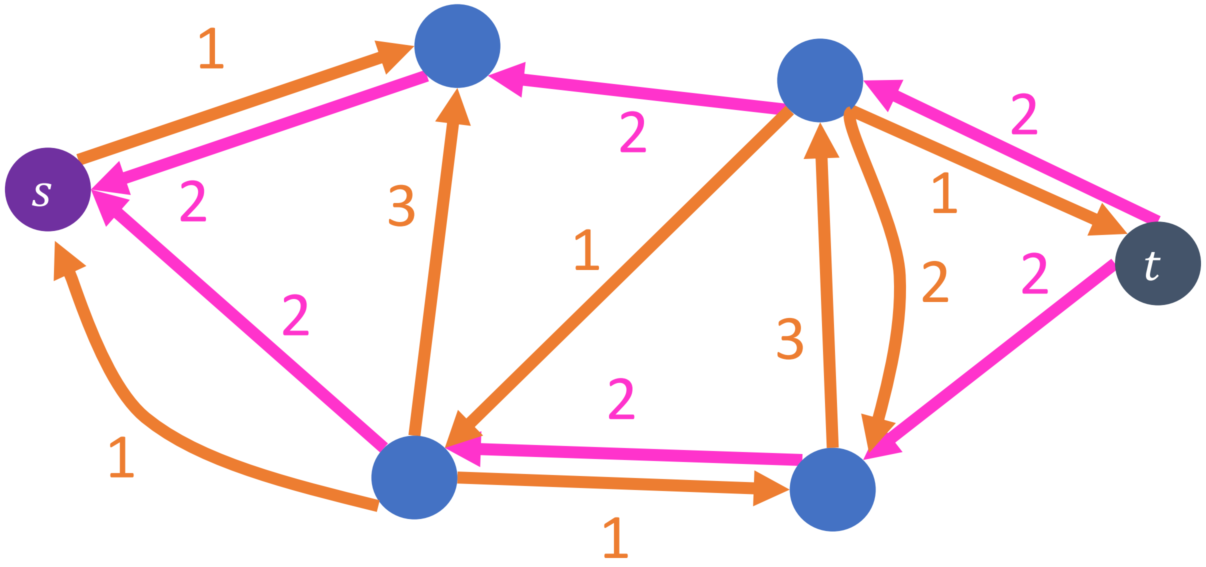
- **Correctness of Ford-Fulkerson:** Ford-Fulkerson terminates when there are no more augmenting paths in the residual graph G_f, which means that f is a maximum flow
- **Max-flow min-cut duality:** the maximum flow in a network coincides with the minimum cut of the graph ($\max_f |f| = \min_{S,T} \|S,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

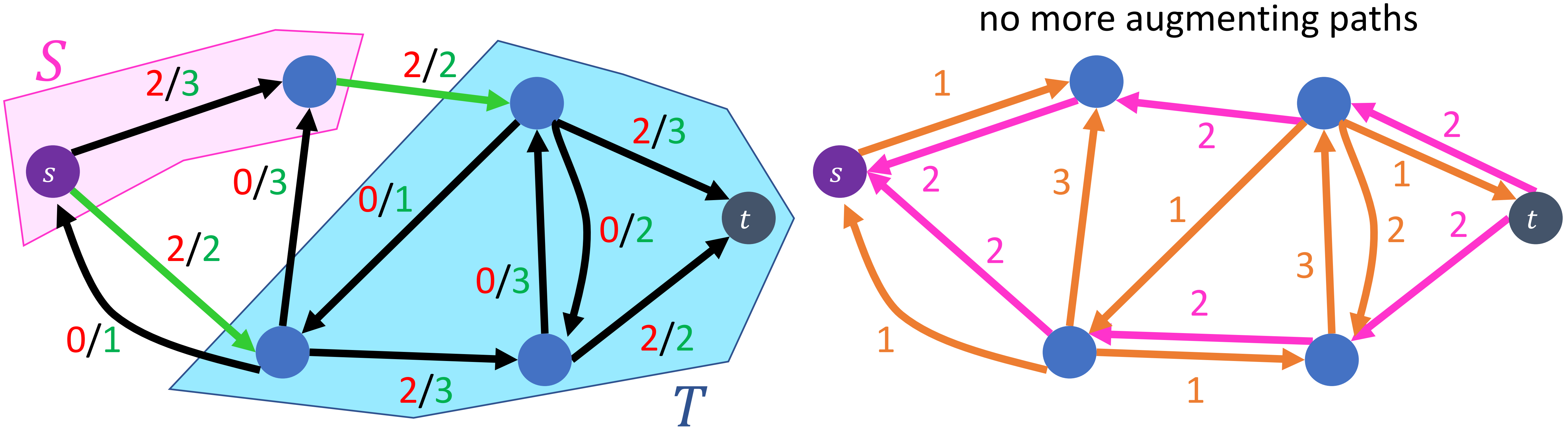
no more augmenting paths



Residual graph

Max flow: 4

Max-Flow Min-Cut Duality Example



Flow graph

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

f is a maximum flow in G $\Rightarrow$ there are no augmenting paths in the residual graph G_f $\Rightarrow$ there exists a cut (S, T) of G where $|f| = \|S, T\|$

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

f is a maximum flow in G

there are no augmenting paths in the residual graph G_f

there exists a cut (S, T) of G where $|f| = \|S, T\|$

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

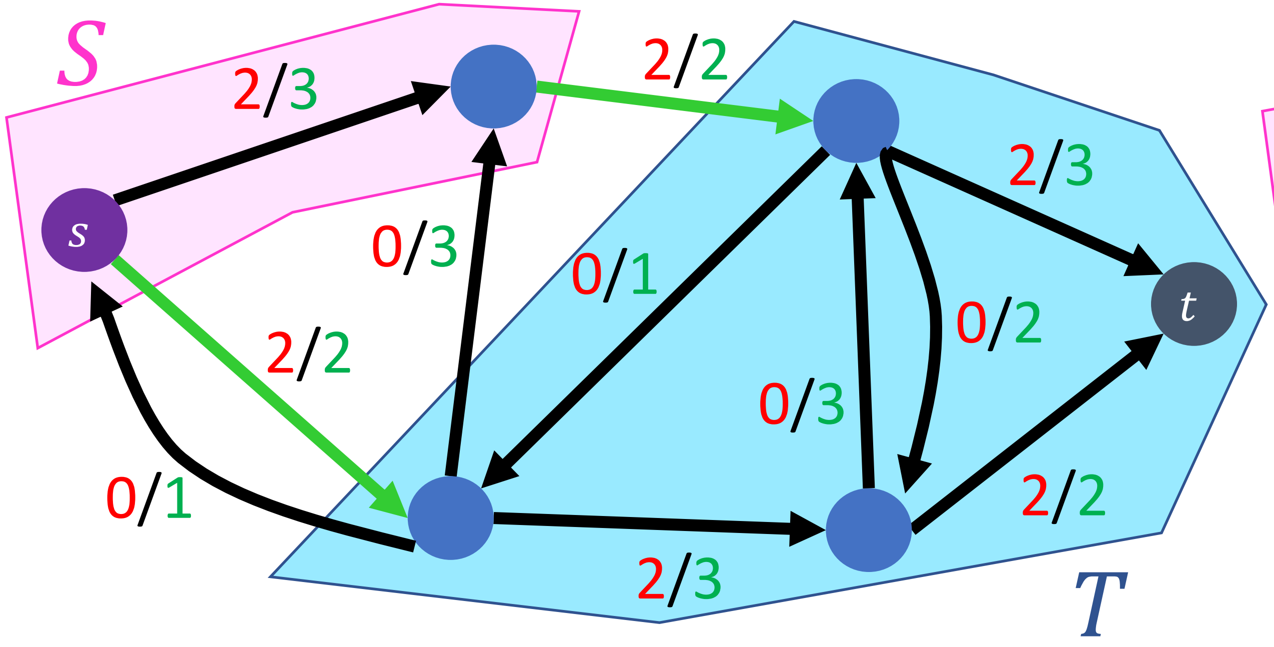
Let f be a flow in a graph G

f is a maximum flow in G

there are no augmenting paths in the residual graph G_f

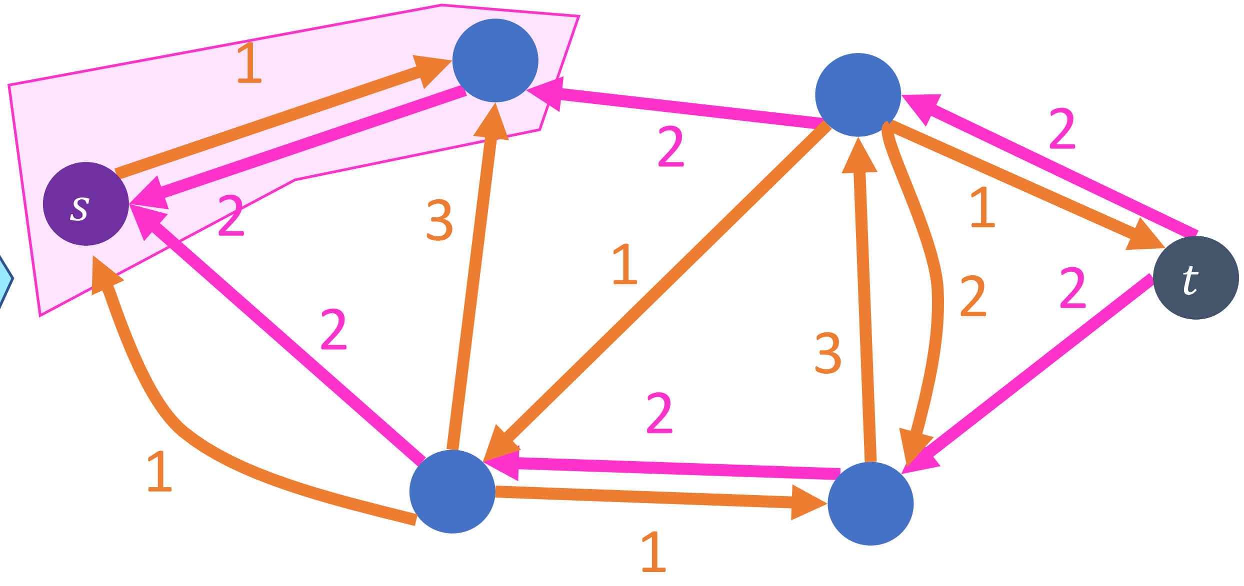
there exists a cut (S, T) of G where $|f| = \|S, T\|$

Max-Flow Min-Cut Theorem Proof



Flow graph

no more augmenting paths

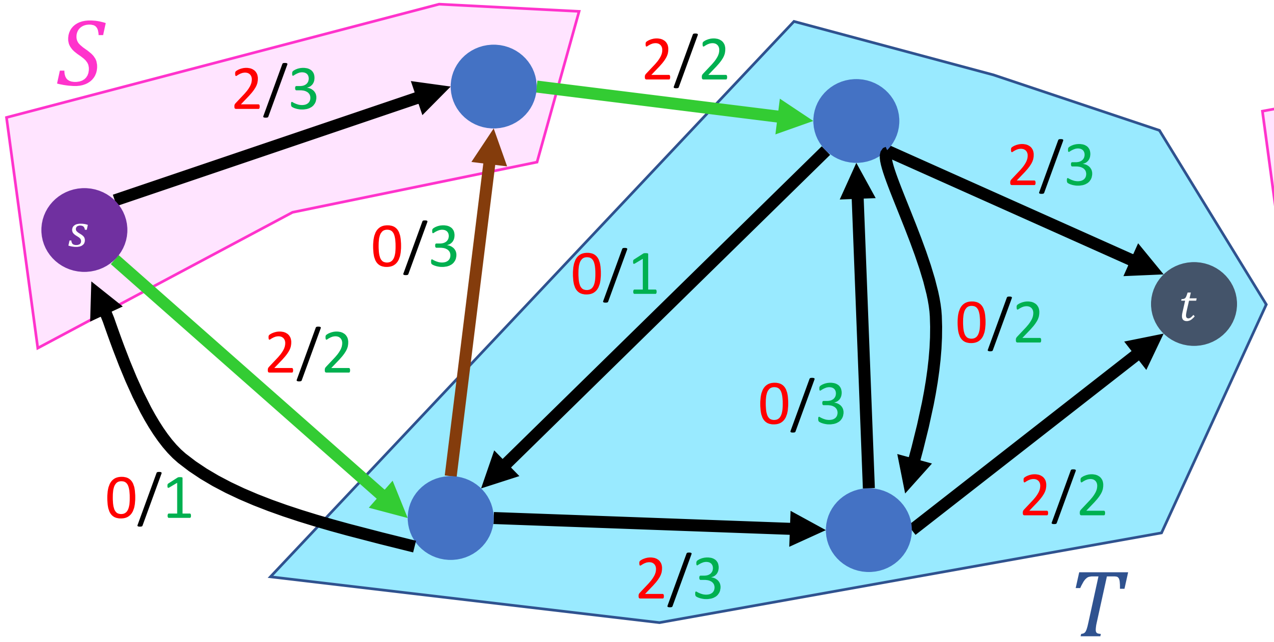


Residual graph

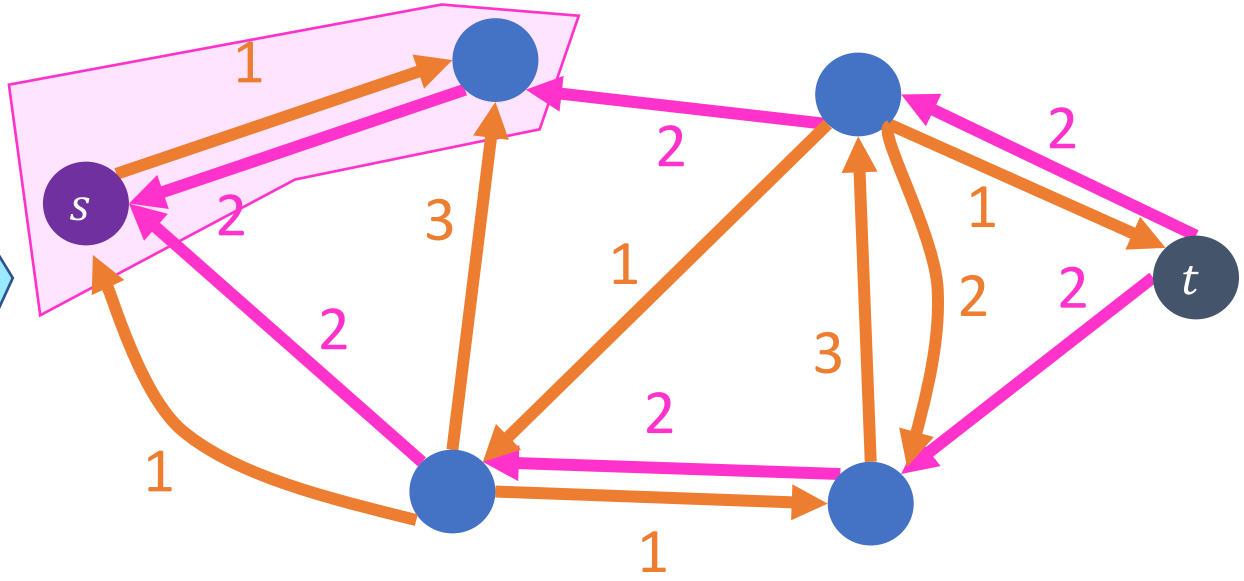
No augmenting paths means there is <u>no</u> path from s to t in G_f

- 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

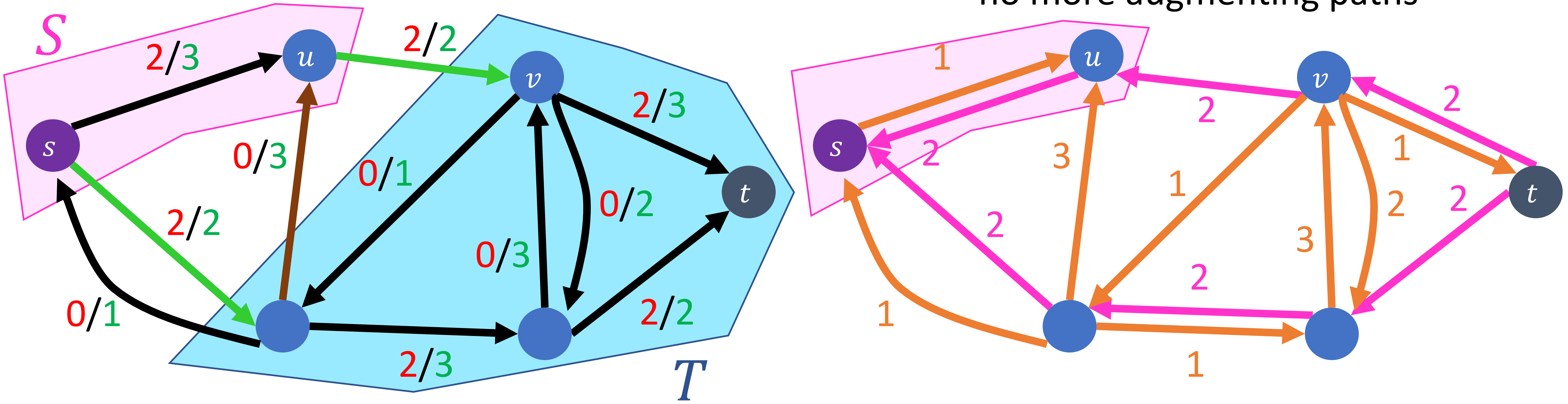


Claim: $\|S,T\| = |f|$

- 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

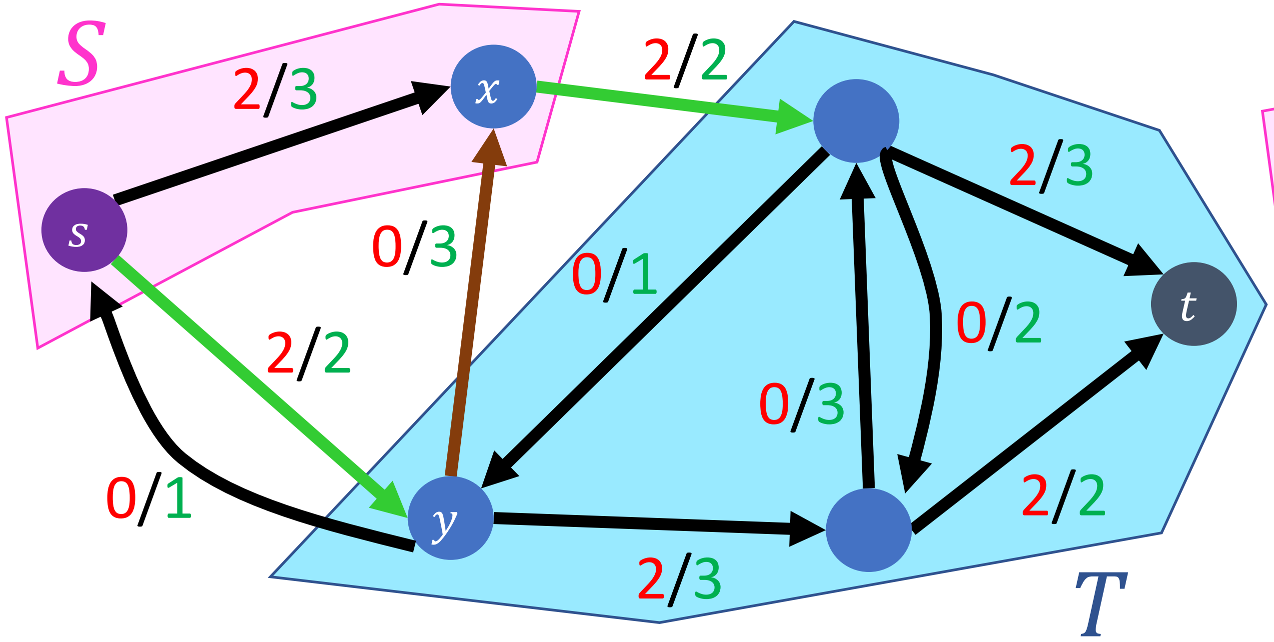
no more augmenting paths



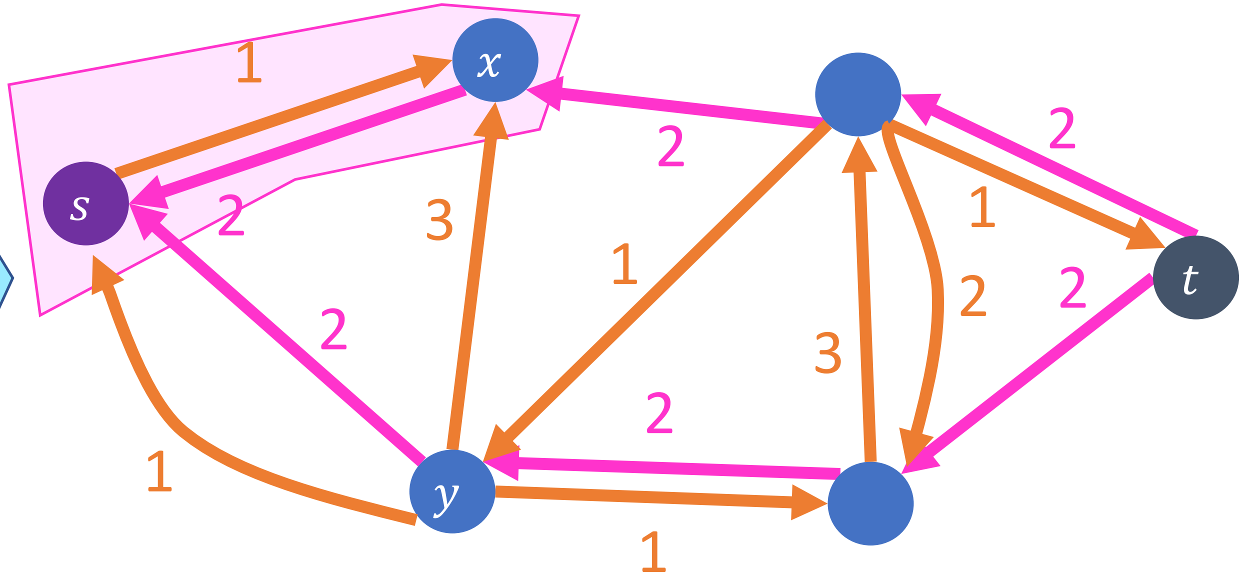
Claim: $\|S, T\| = |f|$

- 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



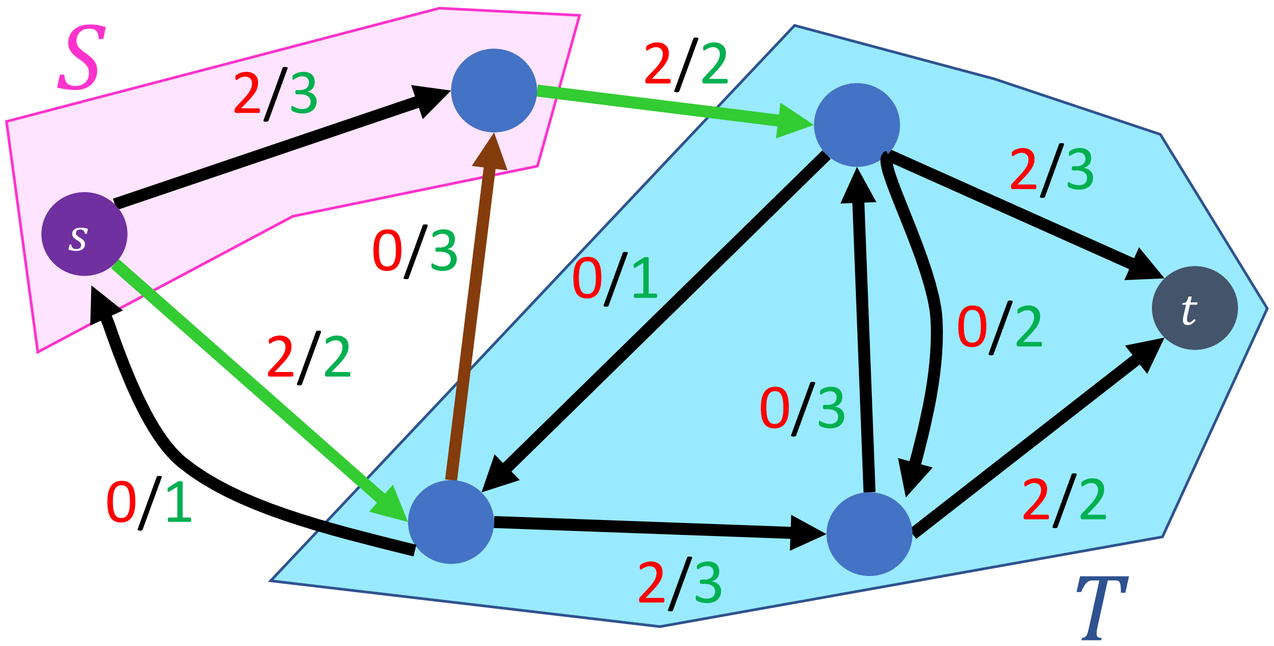
no more augmenting paths



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

f is a maximum flow in G → there are no augmenting paths in the residual graph G_f → there exists a cut (S, T) of G where $|f| = \|S, T\|$

Statements are equivalent!

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)

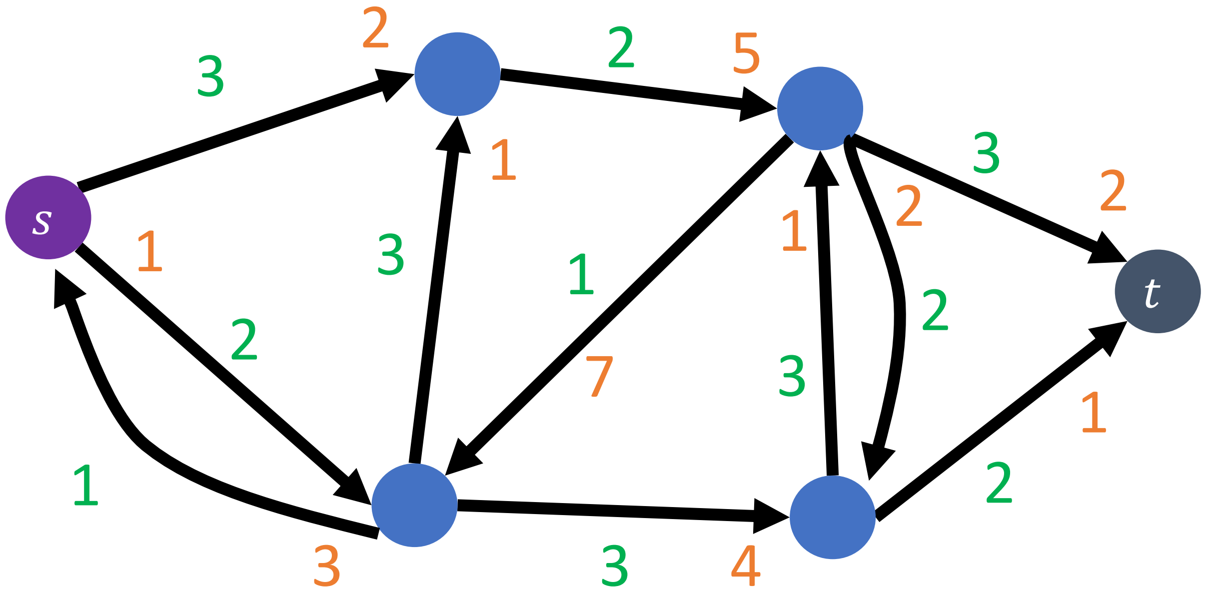
- $\Theta(|E||V|^2)$

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