CS4102 Algorithms

Fall 2021 – Horton and Floryan

These are Horton's version of the slides, used in his lecture.

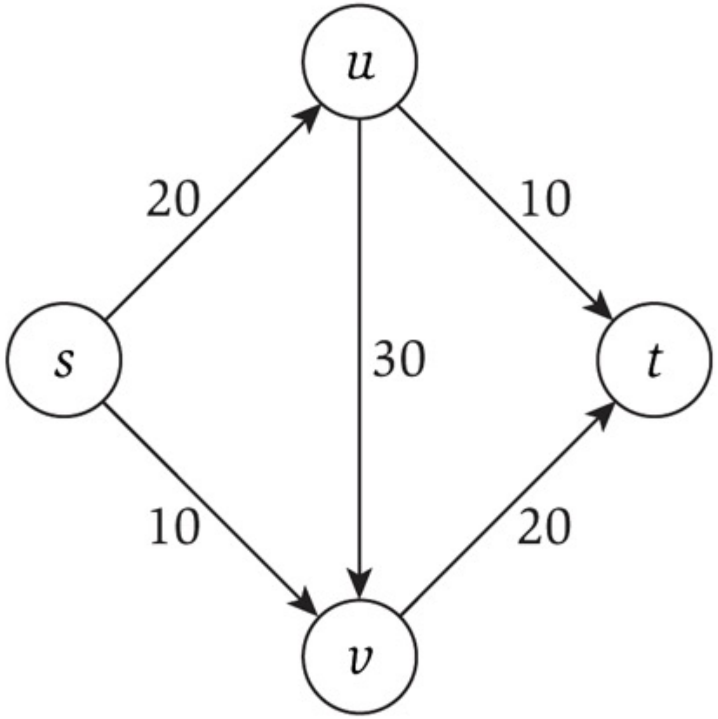
Network Flow, Ford-Fulkerson

In your textbook

- CLRS 26.1 and 26.2
- Includes simple solutions to the following “complications”
  - What if (u,v) and (v,u) are in the flow graph?
    - Called “Antiparallel” edges – easy to adjust for this, example later
  - What if we need >1 source? >1 sink?

Flow networks

- Consider a flow network, which is a specialized directed graph with:
  - A single source node s
  - A single terminus node t
  - Capacities on each edge
    - That must be integer!
- What is the maximum flow you can send from s to t?

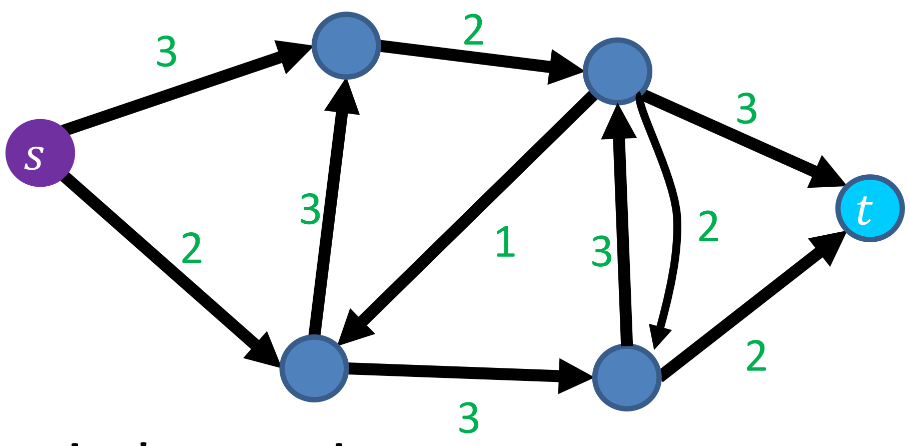


Applications

- Transportation networks
  - How many people can be routed?
- Computer networks
- Electrical distribution
- Water distribution

- Note that all these applications have multiple sources and multiple sinks!
  - Whereas the flow networks we study do not, yet

Flow Network

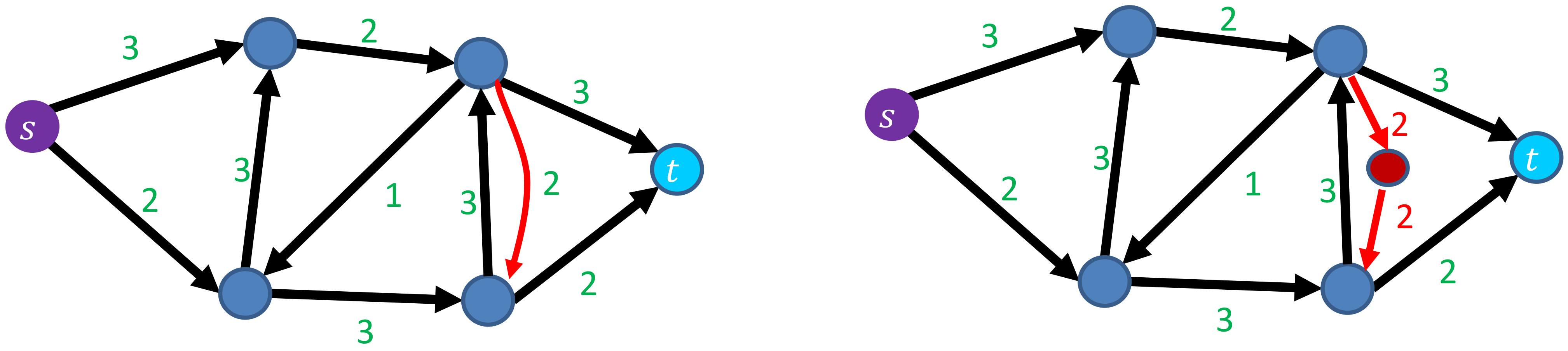


Graph $G = (V, E)$
Source node $s \in V$
Sink node $t \in V$
Edge Capacities $c(e) \in$ Positive whole* numbers
If $(u, v) \in E$ then $(v, u) \notin E$ (Note our example here violates this!)

Max flow intuition: If s is a faucet, t is a drain, and s connects to t through a network of pipes with given capacities, what is the maximum amount of water which can flow from the faucet to the drain?

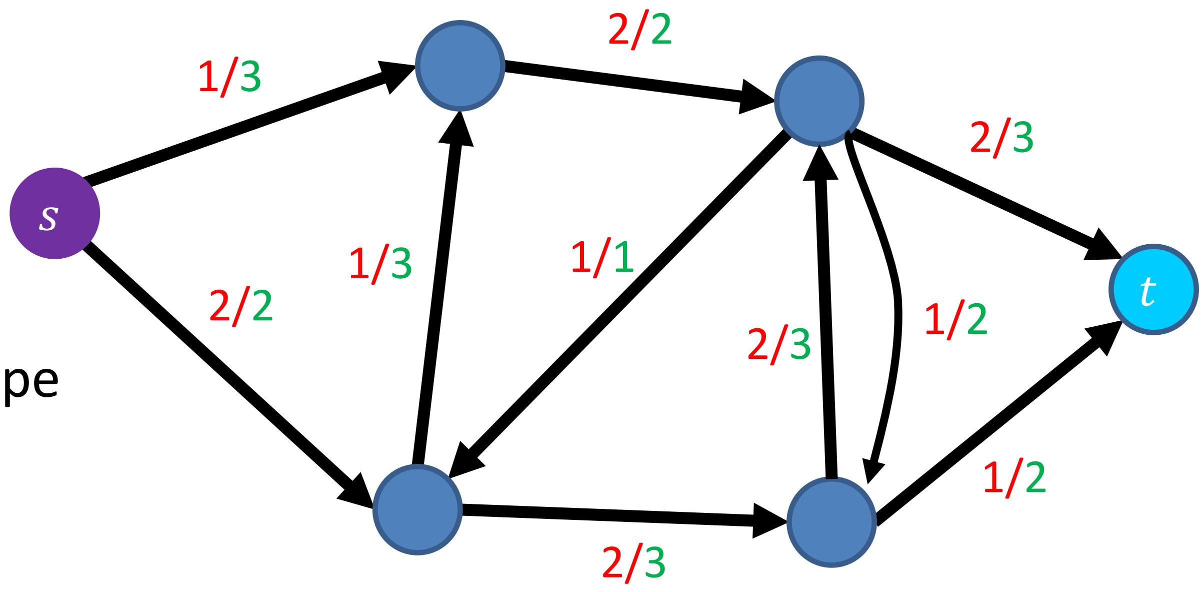
Flow Network: Antiparallel Edges

Easy adjustment to remove antiparallel edges and have equivalent flow graph: add intermediate node



(Note: our later examples use graph on the left without this adjustment.)

Flow



Flow/Capacity

- Assignment of values to edges
  - $f(e) = n$
  - E.g. n units of water going through that pipe
- Capacity constraint
  - $f(e) \leq c(e)$
  - Flow cannot exceed capacity
- Flow constraint
  - $\forall v \in V - \{s, t\}, inflow(v) = outflow(v)$
  - $inflow(v) = \sum_{x \in V} f(v, x)$
  - $outflow(v) = \sum_{x \in V} f(x, v)$
  - Water going in must match water coming out
- Flow of G: $|f| = outflow(s) - inflow(s)$
  - Net outflow of s

3 in example above

Let's Make Some Rules

- Source node has NO incoming flow
- Terminal (sink) node has NO outgoing flow

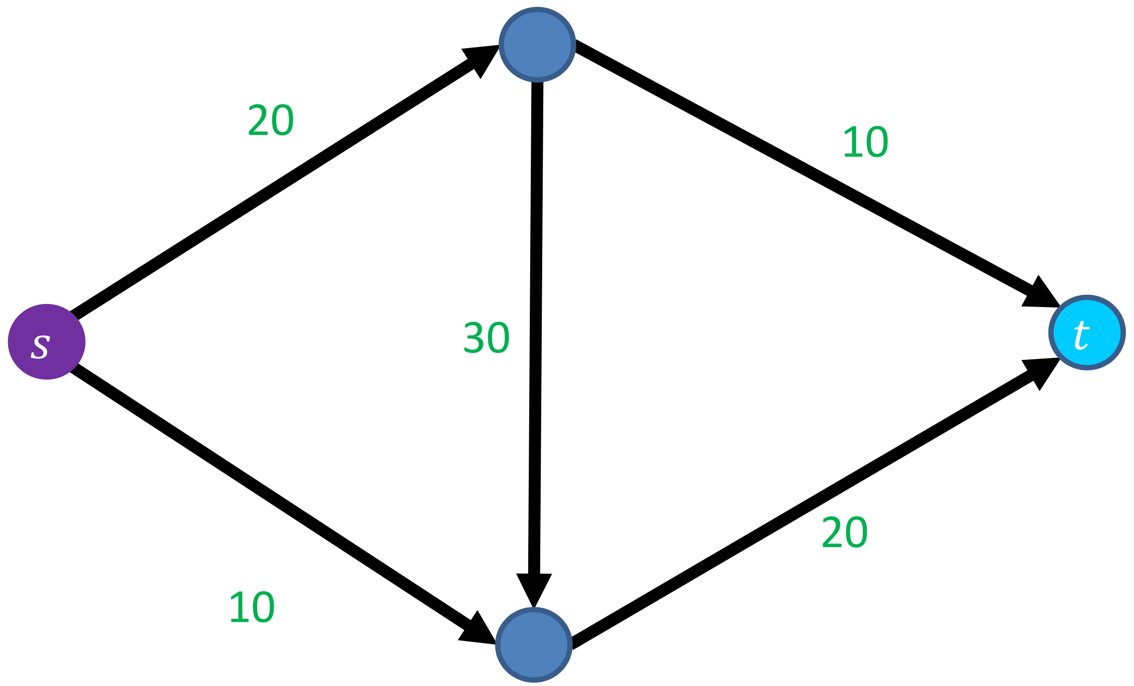
- Internal nodes has net zero flow
  - all units of flow going in must be going out as well

- No edge is over capacity

- GOAL: Find the maximum flow that can be "pushed" through the network
  - I.e. maximize: $|f| = outflow(s) - inflow(s)$

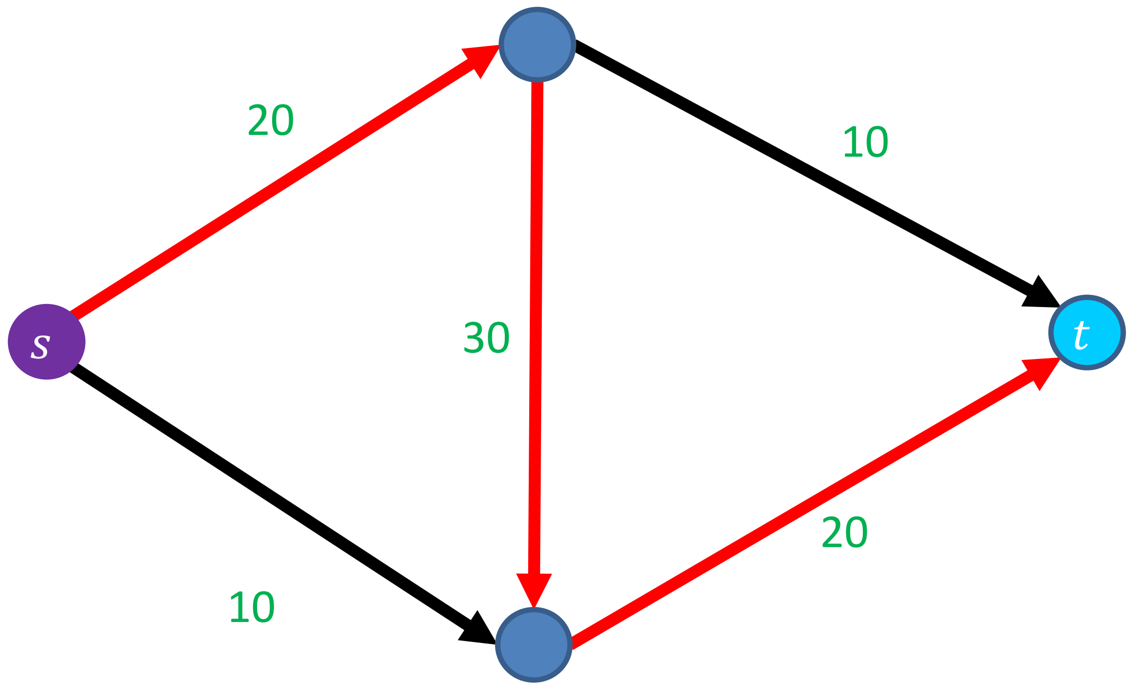
How to Solve This? This Greedy doesn't work

Saturate Highest Capacity Path First



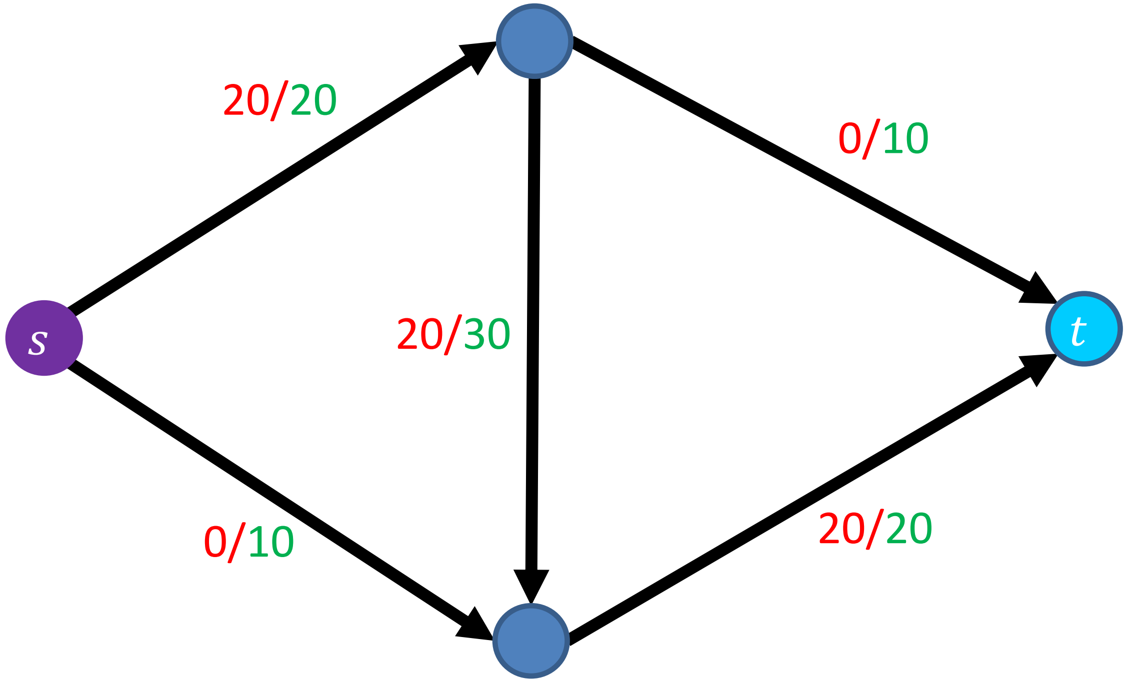
Greedy doesn’t work

Saturate Highest Capacity Path First



Greedy doesn't work

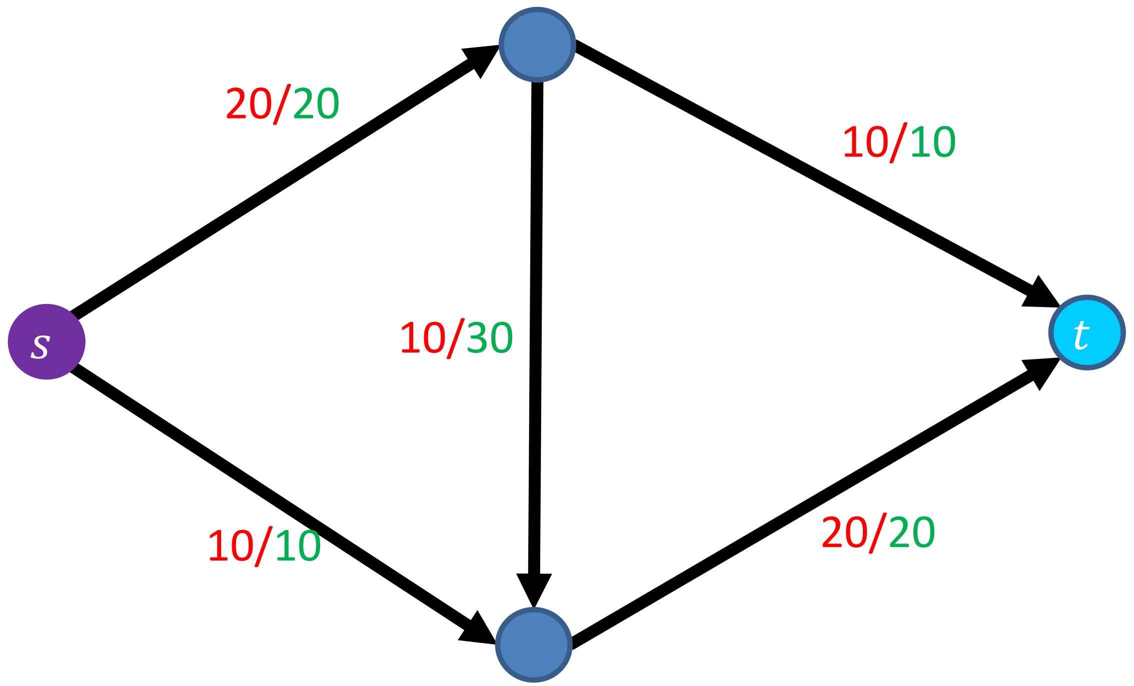
Saturate Highest Capacity Path First



Overall Flow: $|f| = 20$

Greedy doesn't work

Better Solution



Overall Flow: $|f| = 30$

Ford-Fulkerson: Algorithm overview

- Iterative algorithm: push some flow along some path at each step
- Model or record the *residual* capacities
  - how much capacity is left after taking into account how much flow is going through that edge at this time
- Find a path from *s* to *t* such that the minimum residual capacity of an edge on that path is greater than zero
  - Since each value is an integer, it must be 1 or more
- Update the residual capacities after taking into account this new flow
- Repeat until no more such paths are found

Algorithm notation

- $f(u,v)$: the flow on the edge from u to v
- $f(v,u)$: the backflow on the edge from v to u
- $c(u,v)$: the capacity on the edge from u to v
- $c_f(u,v)$: the *residual* capacity on the edge from u to v
- G_f is the graph where the edges weights are the residual capacities
  - THIS is usually the graph we actually use when running the algorithm we are about to see.

Backflow

- Each edge has forward flow and backflow
  - The two must always be “inverses” of each other!
  - I.e. they sum to the total capacity for that edge
- This allows for modeling of flow “returning” along a given edge

- One way to think about this:
  - How much of the forward flow we could “un-do”

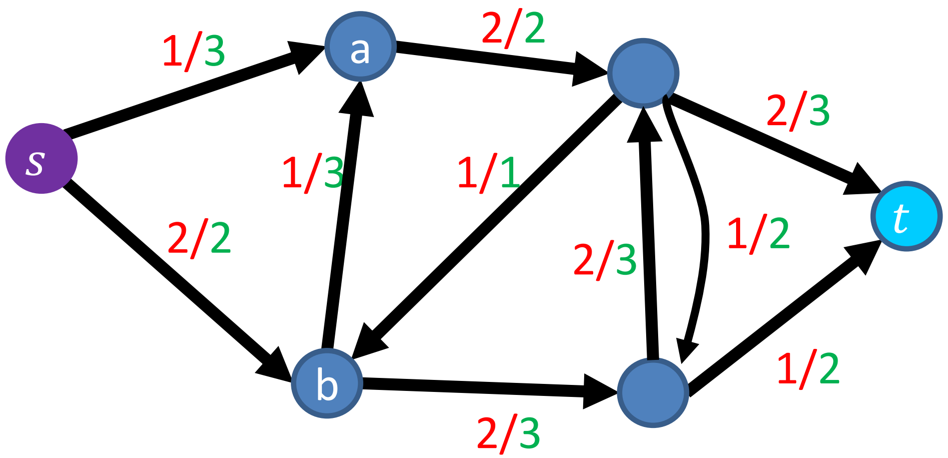
Residual Graph G_f

- Keep track of net available flow along each edge
- "Forward edges": weight is equal to available flow along that edge in the flow graph
  - $w(e) = c(e) - f(e)$
- "Back edges": weight is equal to backflow along that edge in the flow graph
  - $w(e) = f(e)$

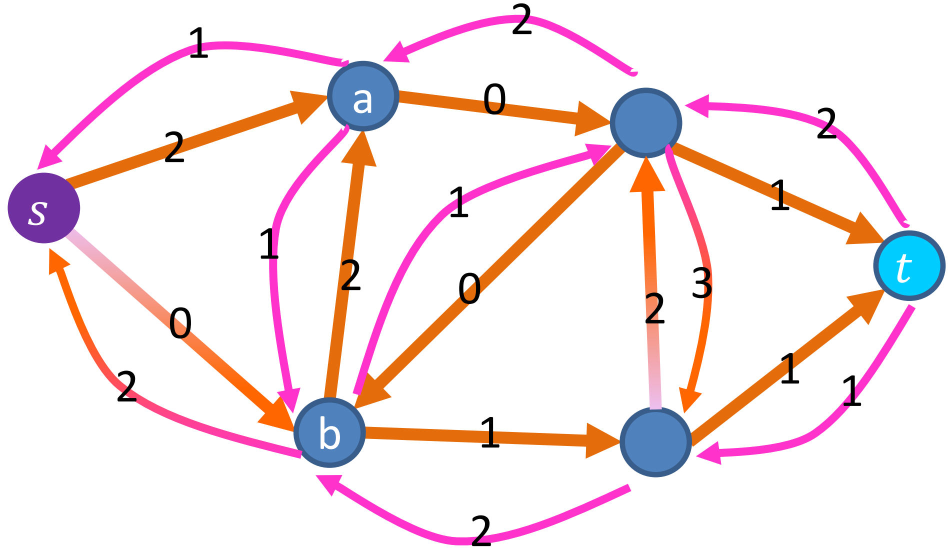
Flow I *could* add

Flow I *could* remove

Flow Graph G

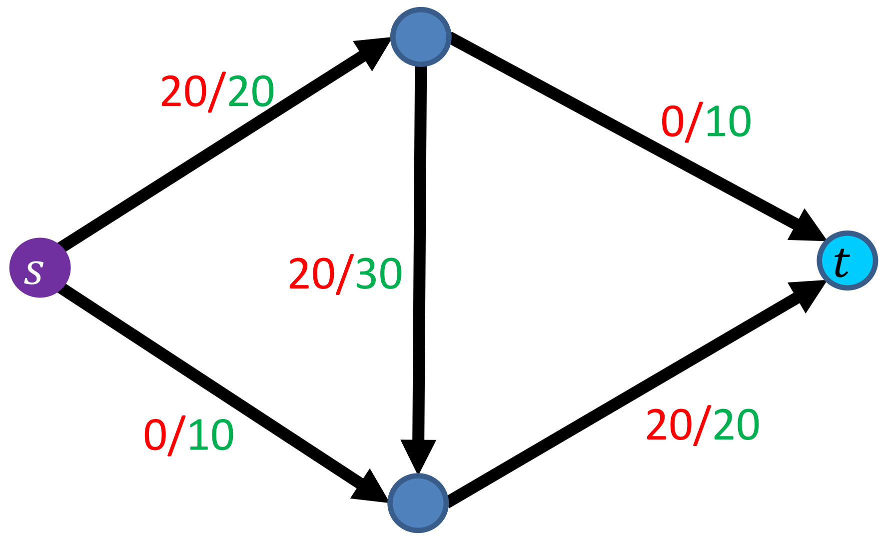


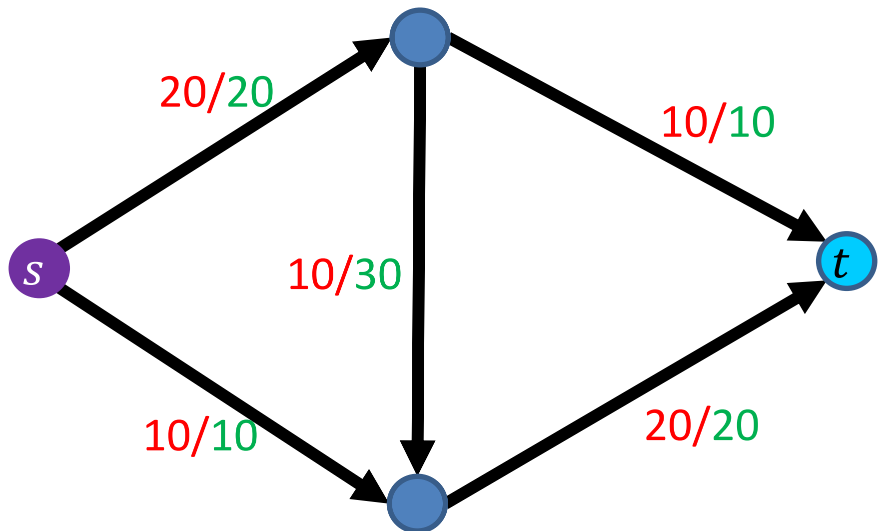
Residual Graph G_f



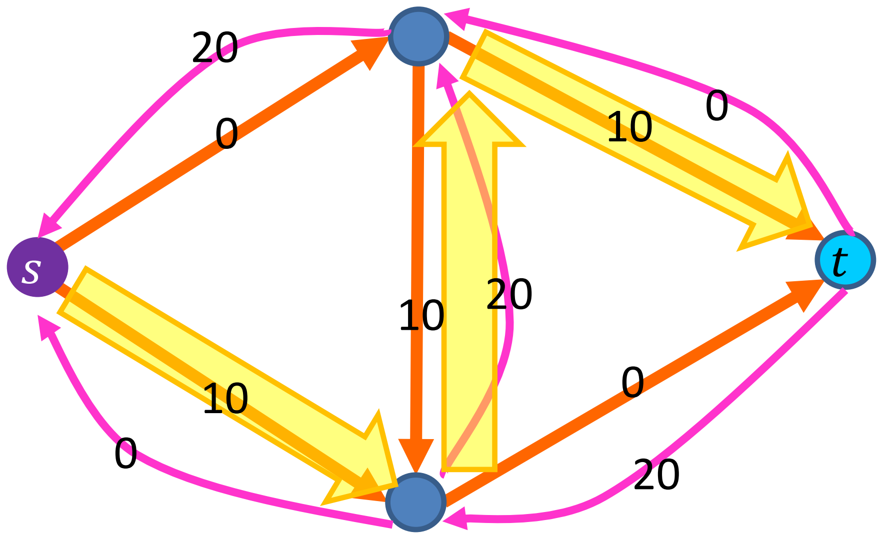
Residual Graphs Example

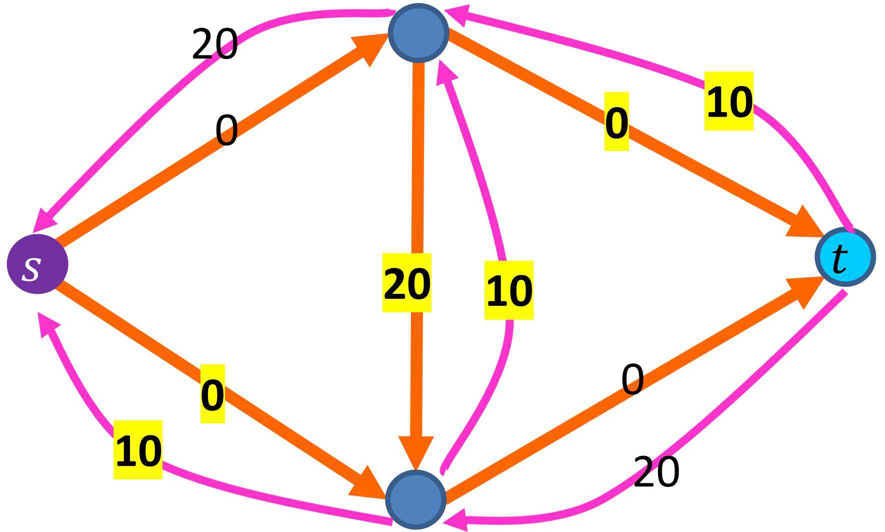
Flow Graph





Residual Graph





Ford-Fulkerson Algorithm

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

Overview: Repeatedly add the flow of any 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 p in G_f:
  - Let $c = \min_{u,v \in p} c_f(u, v)$
  - 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 Algorithm

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

Overview: Repeatedly add the flow of any 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 p in G_f:
  - Let $c = \min_{u,v \in p} c_f(u, v)$
  - 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 Algorithm

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

Overview: Repeatedly add the flow of any 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 p in G_f:
  - Let $c = \min_{u,v \in p} c_f(u, v)$
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

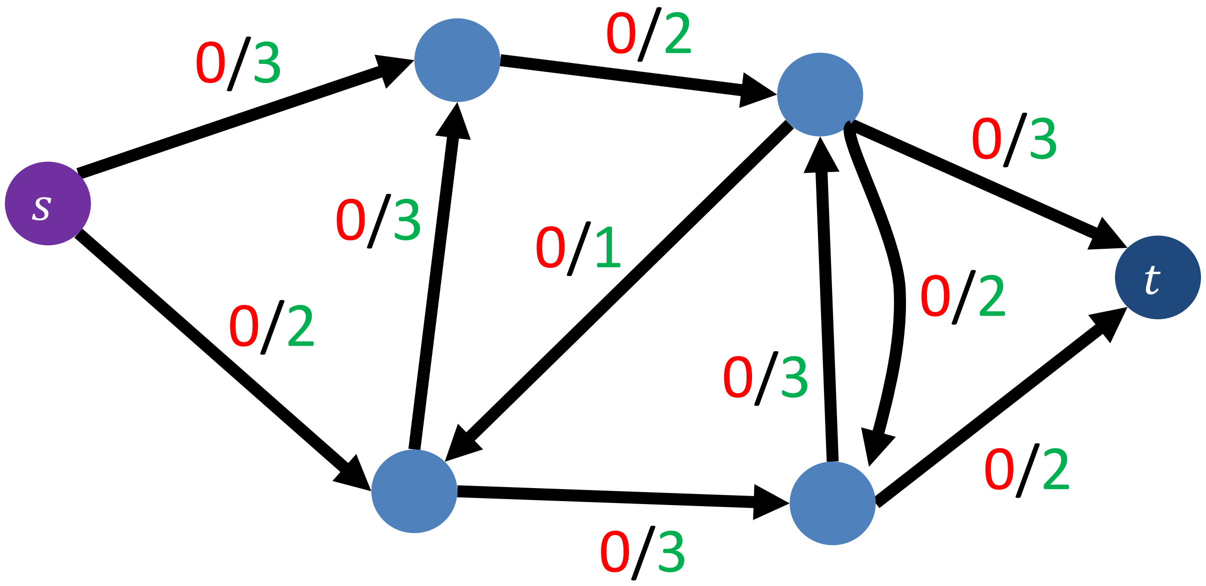
Ford-Fulkerson approach: take <u>any</u> augmenting path (will revisit this later)

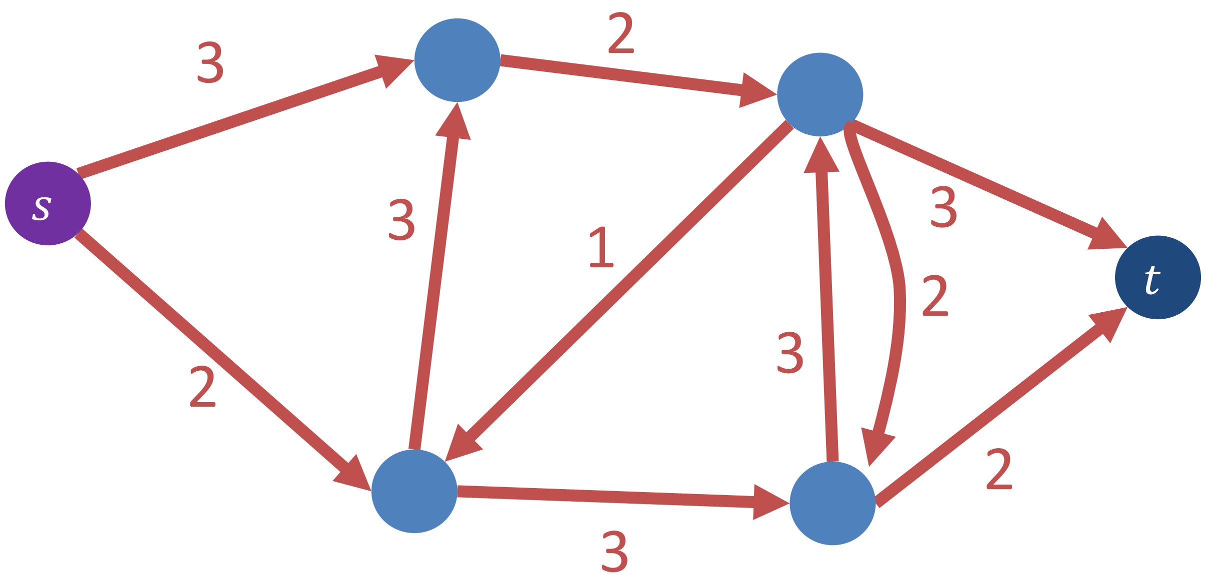
($c_f(u, v)$ is the weight of edge (u, v) in the residual network G_f)

Ford-Fulkerson Algorithm: Updating G_f

1. $f(u,v) = 0$ for all edges (u,v)
2. While there is an "augmenting" path p from s to t in G_f such that $c_f(u,v) > 0$ for all edges $(u,v) \in p$
   a. Find $c_f(p) = \min\{c_f(u,v) \mid (u,v) \in p\}$
   b. For each edge $(u,v) \in p$
      i. $f(u,v) = f(u,v) + c_f(p)$ send flow along the path
      ii. $f(v,u) = f(v,u) - c_f(p)$ send backflow the other way

Ford-Fulkerson Example

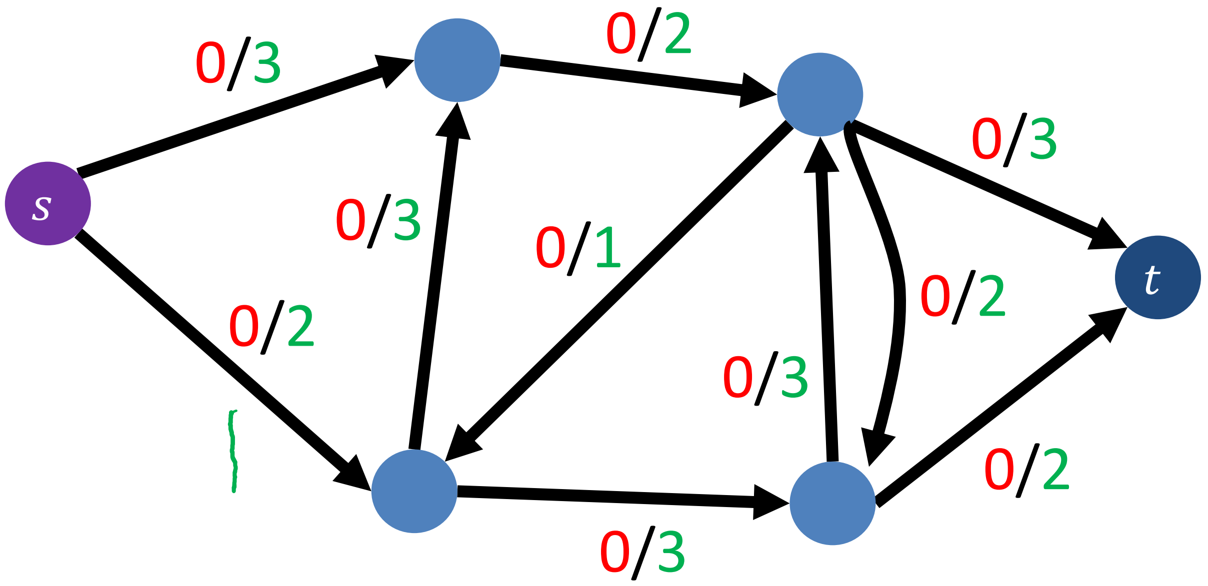




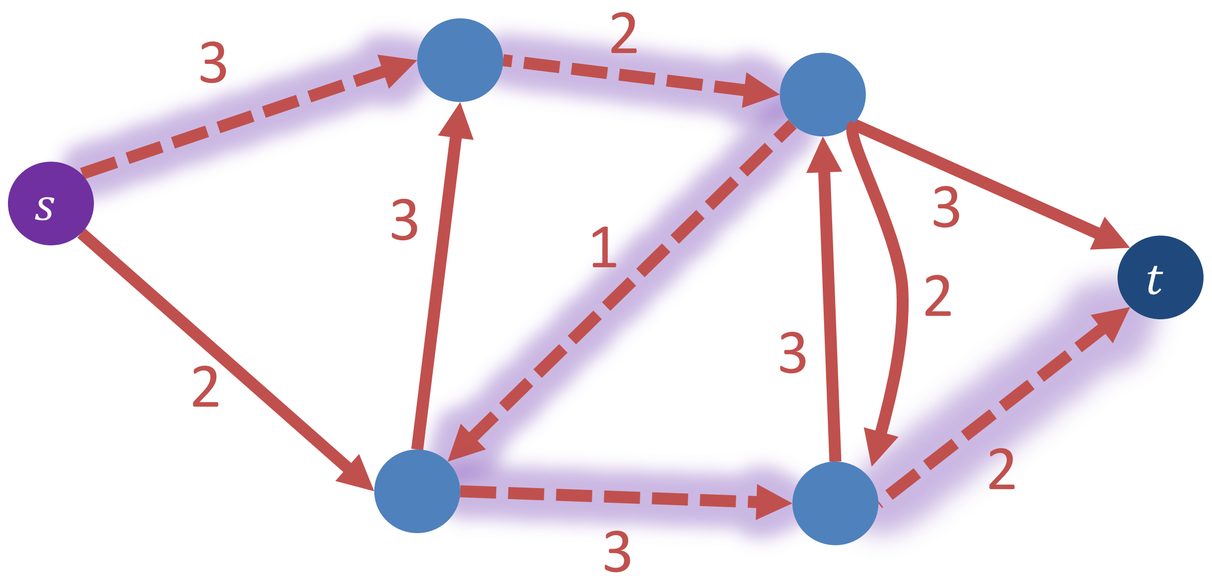
Initially: $f(e) = 0$ for all $e \in E$

Residual graph G_f

Ford-Fulkerson Example

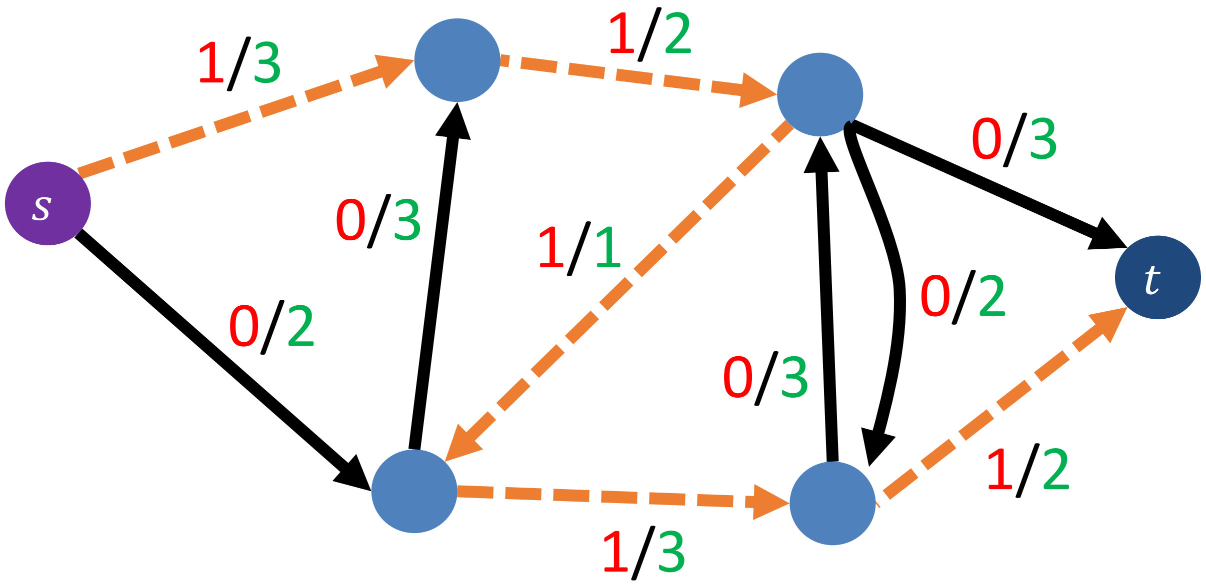


Increase flow by 1 unit

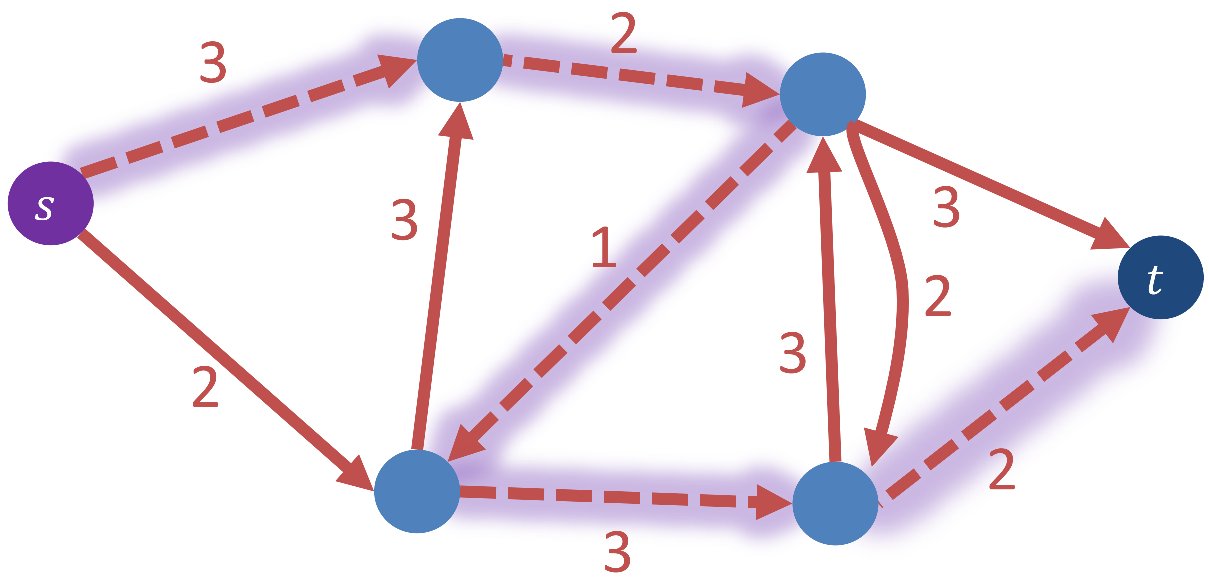


Residual graph G_f

Ford-Fulkerson Example

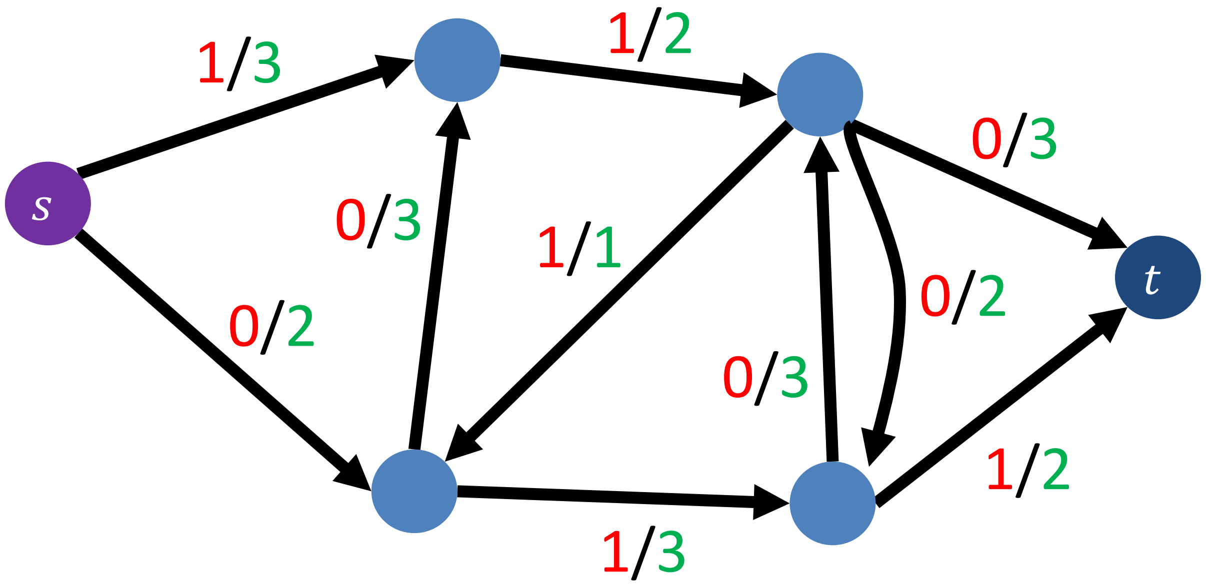


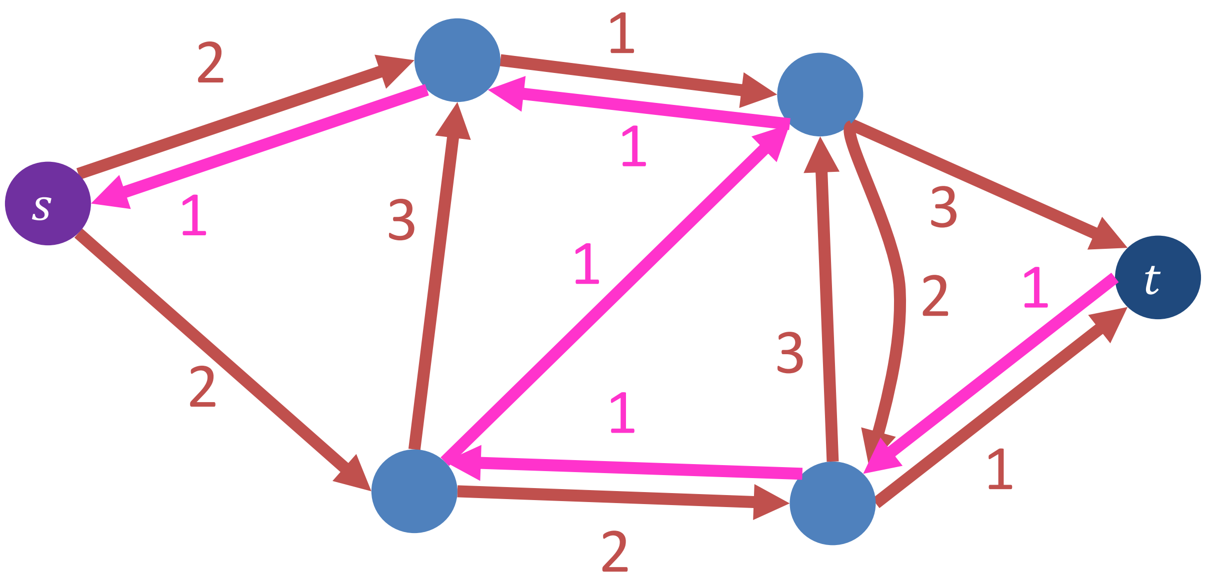
Increase flow by 1 unit



Residual graph G_f

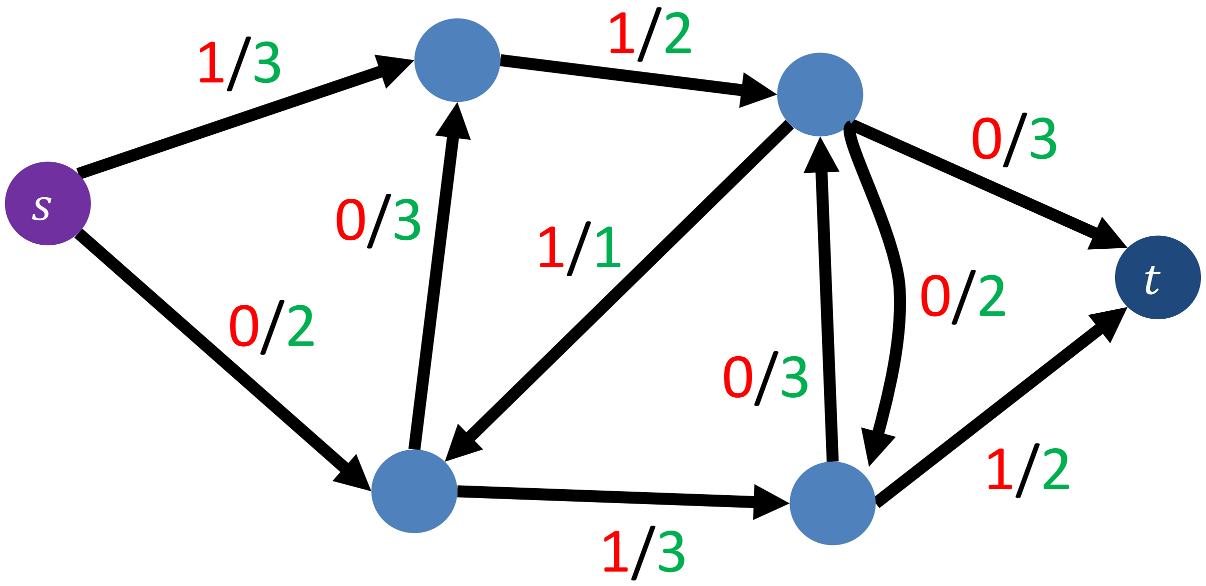
Ford-Fulkerson Example



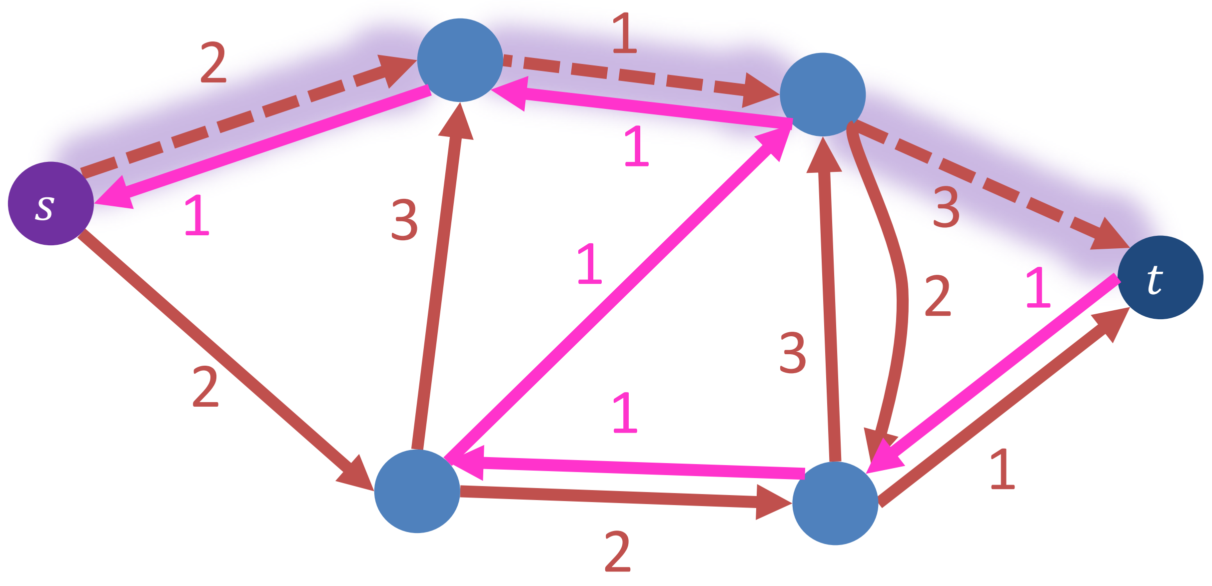


Residual graph G_f

Ford-Fulkerson Example

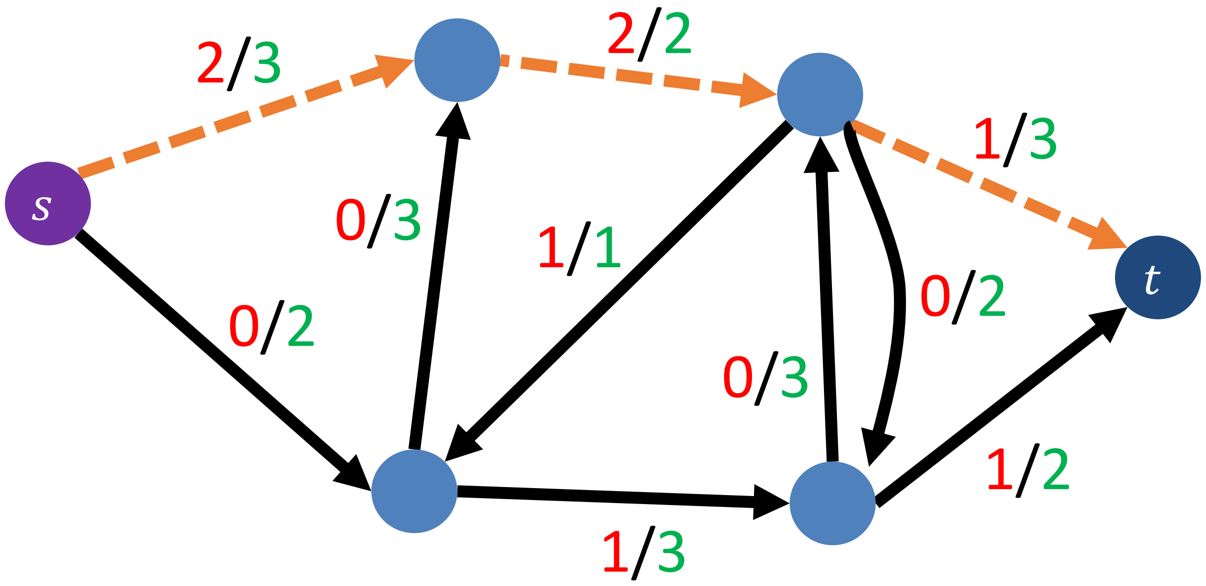


Increase flow by 1 unit

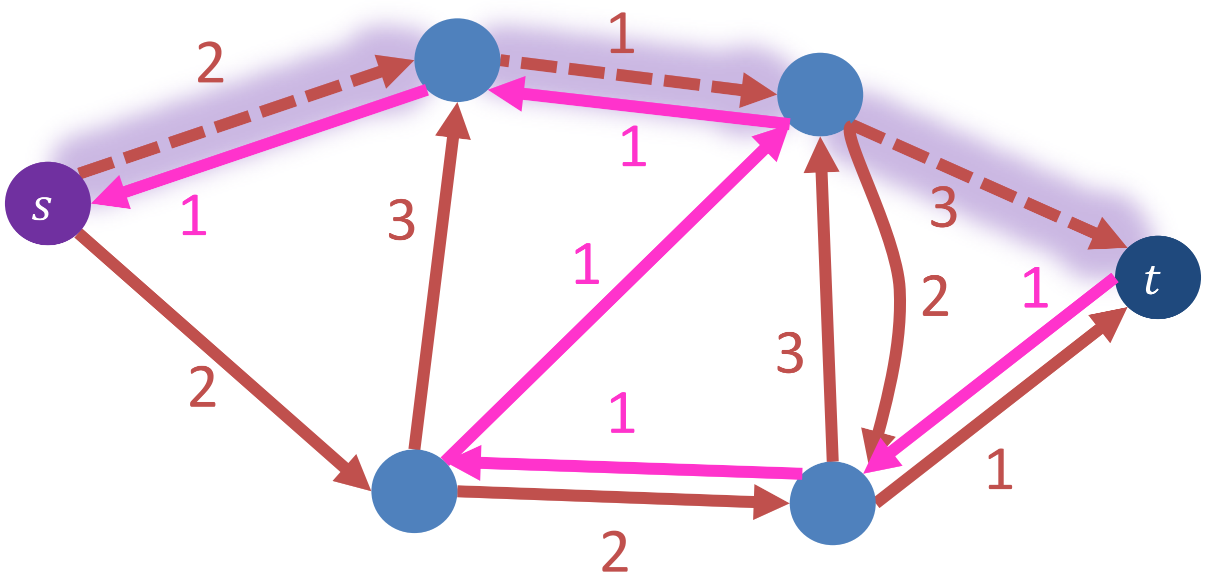


Residual graph G_f

Ford-Fulkerson Example



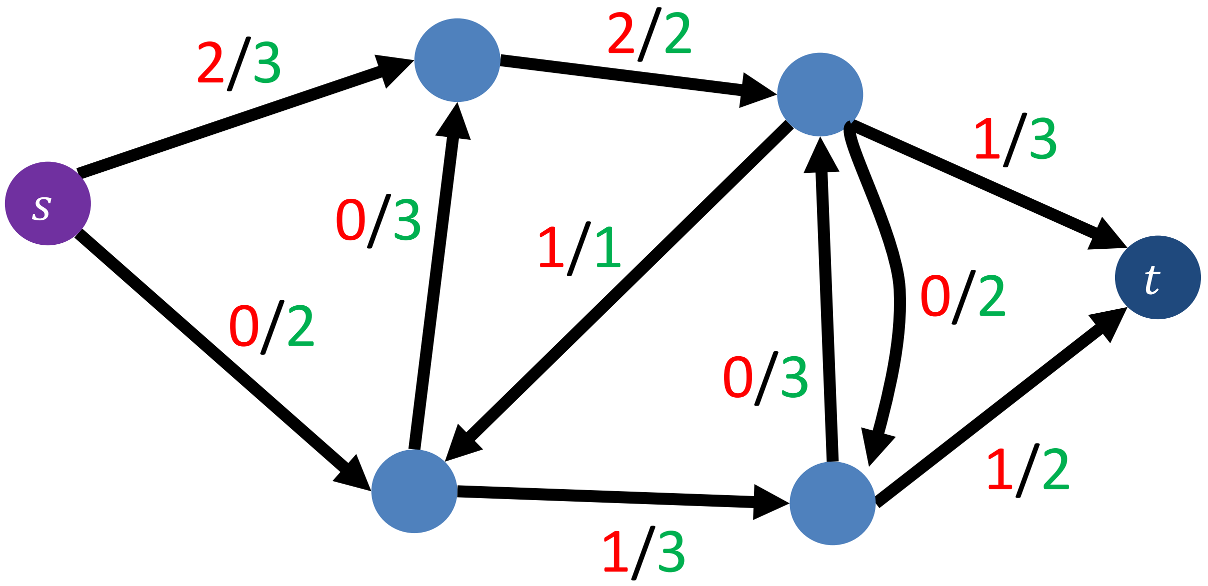
Increase flow by 1 unit

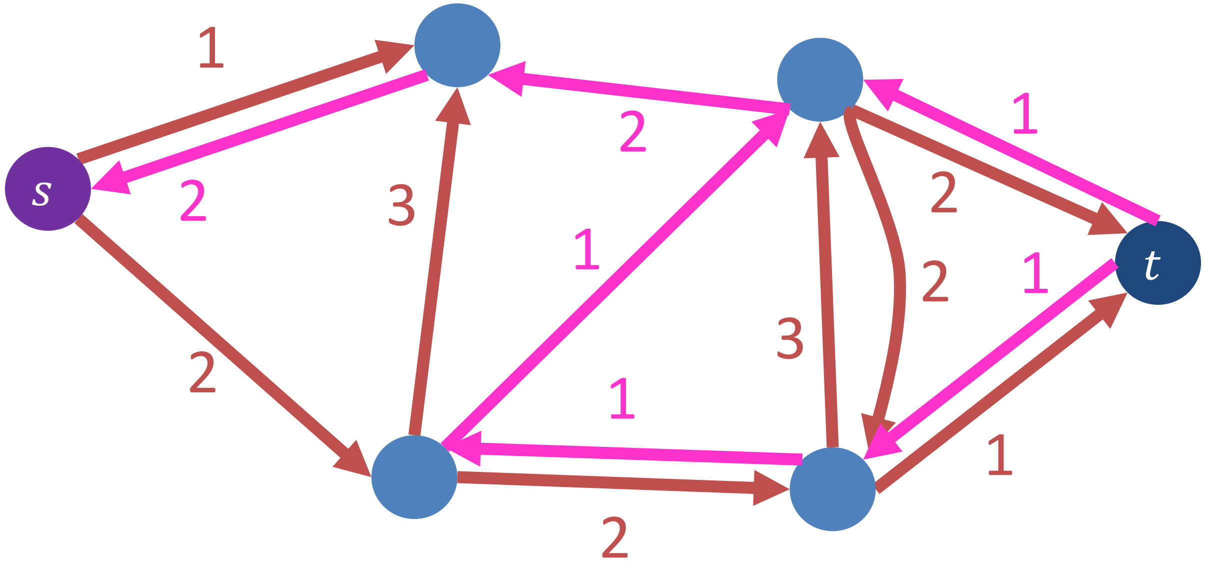


Residual graph G_f

Ford-Fulkerson Example

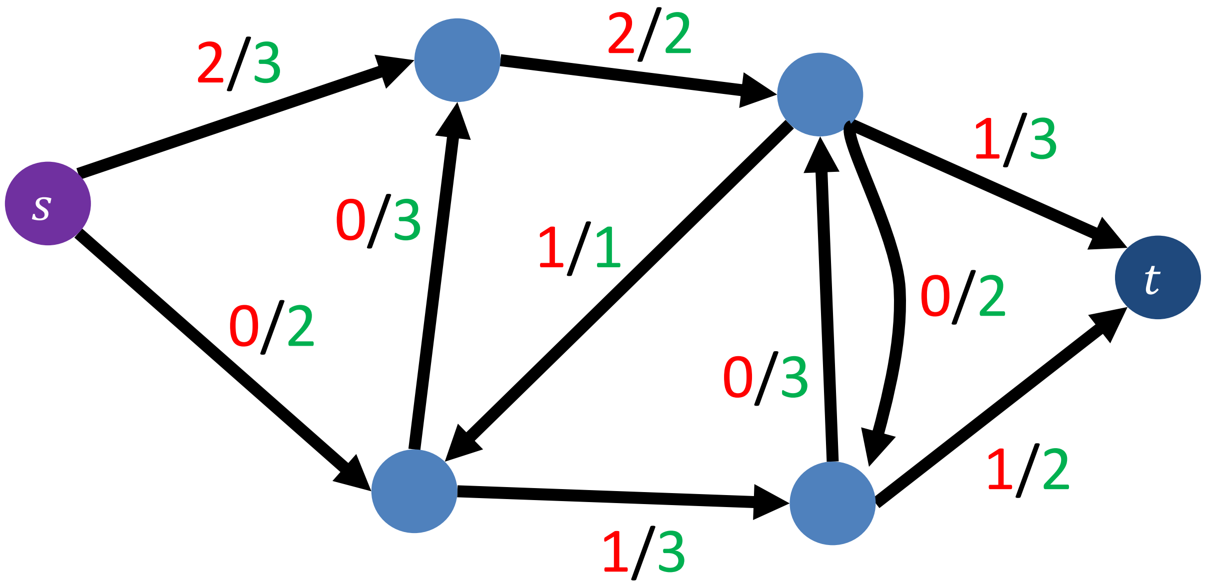
Increase flow by 1 unit

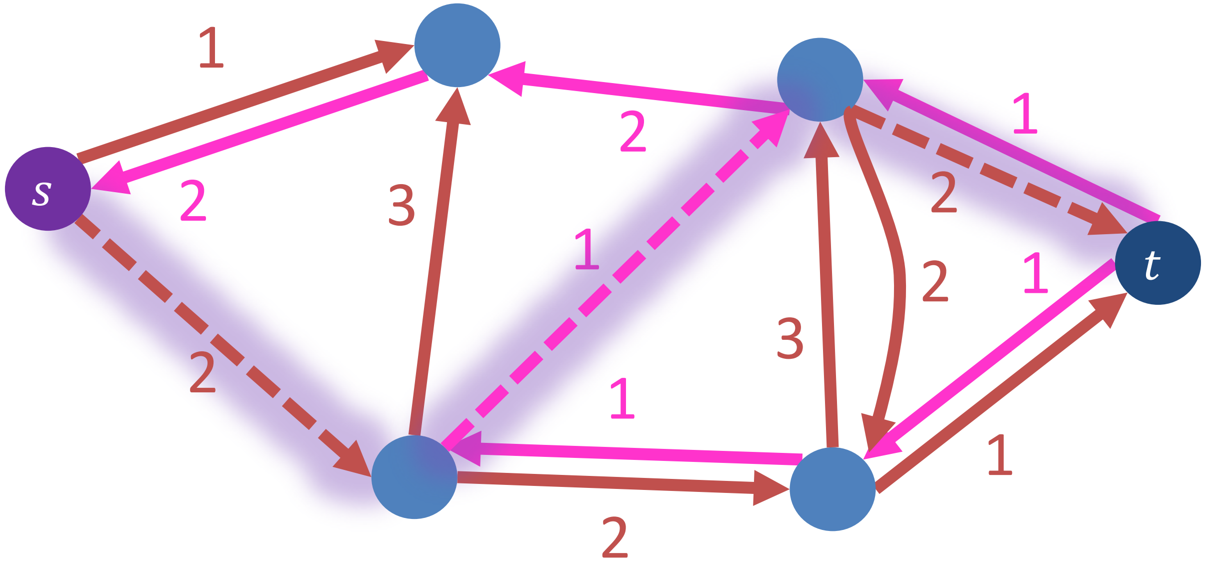




Residual graph G_f

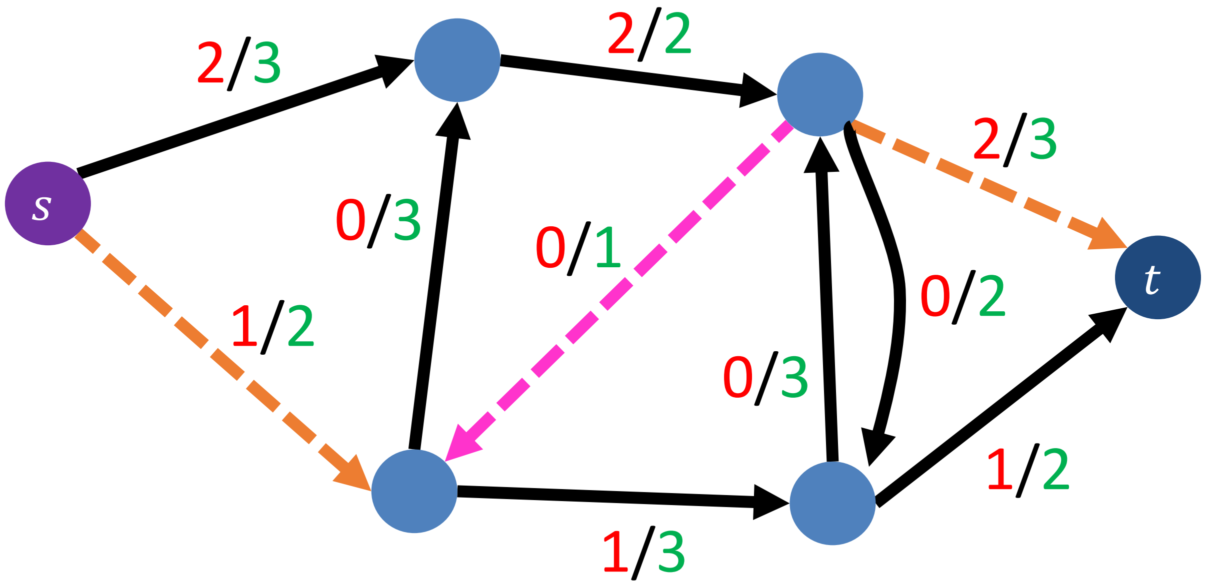
Ford-Fulkerson Example



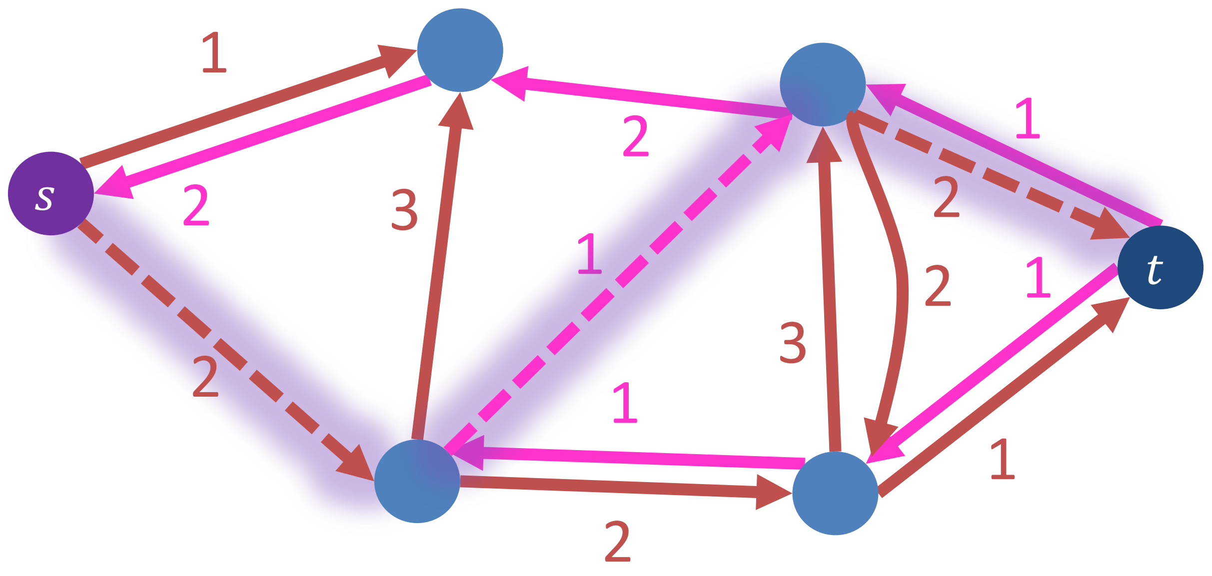


Residual graph G_f

Ford-Fulkerson Example

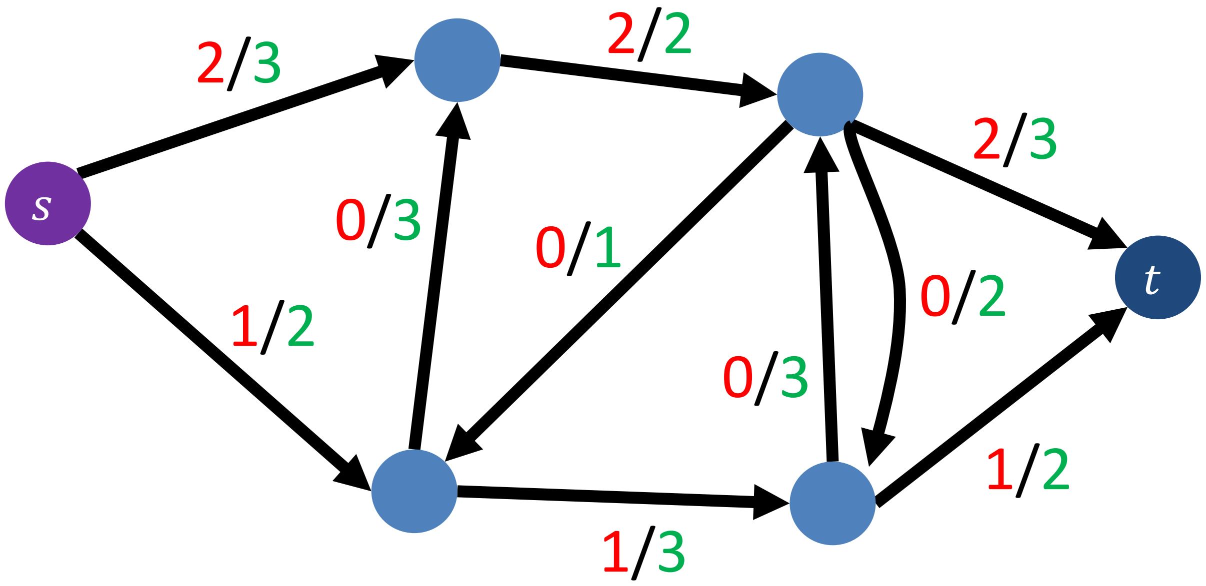


Increase flow by 1 unit

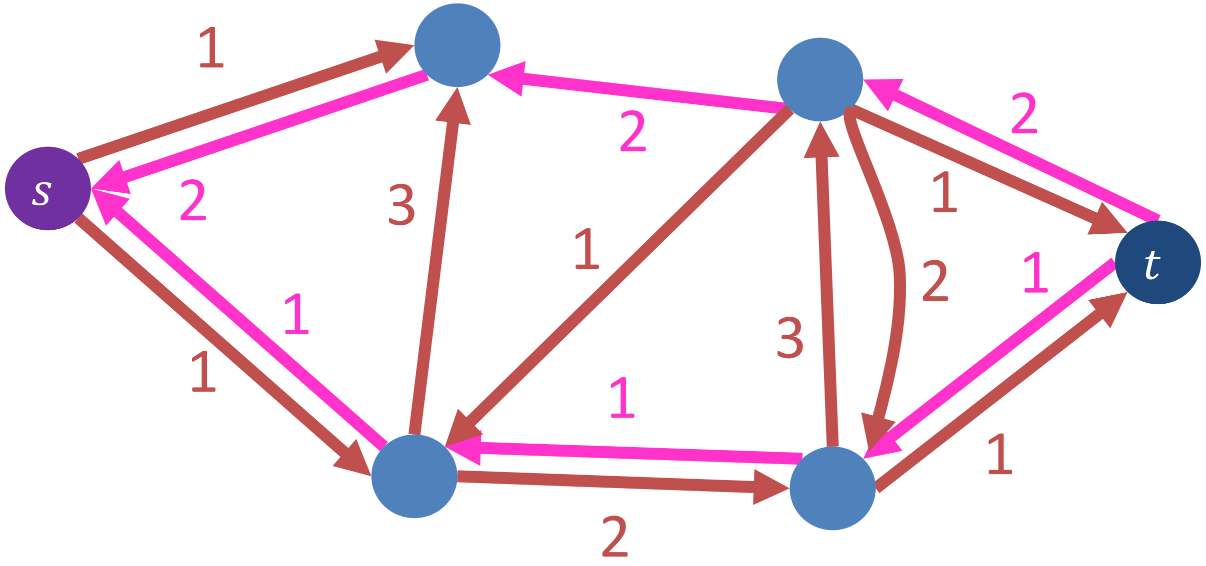


Residual graph G_f

Ford-Fulkerson Example

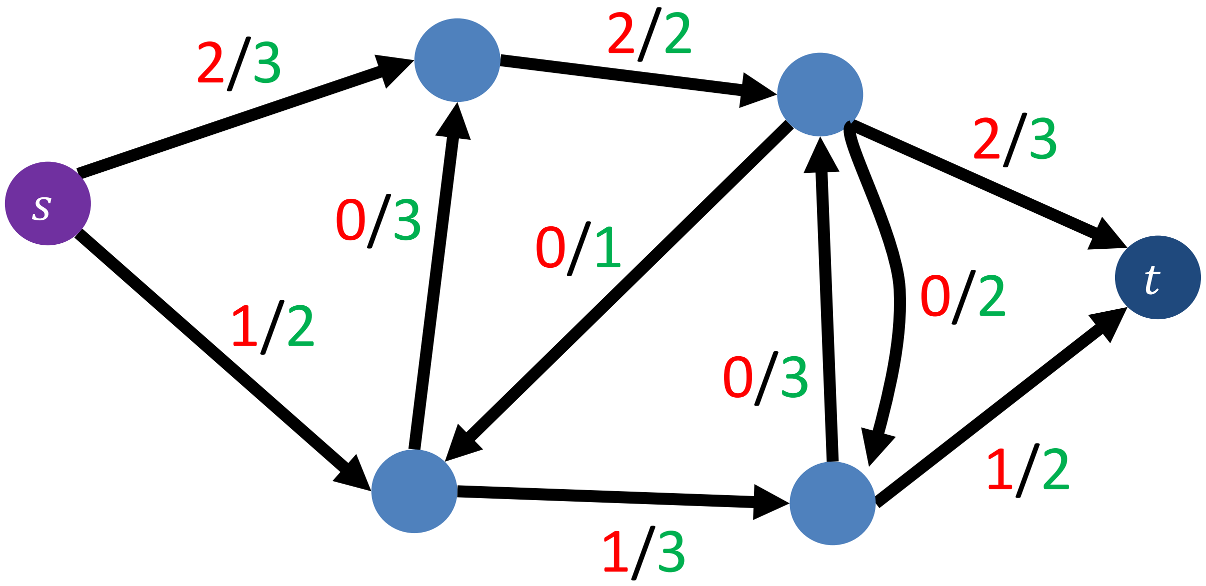


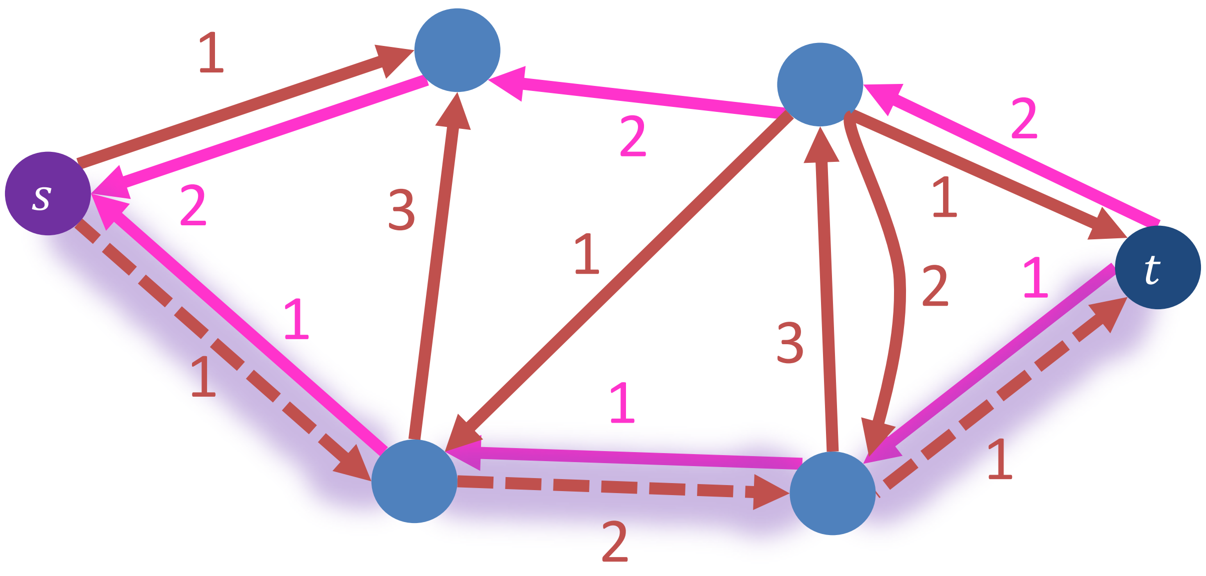
Increase flow by 1 unit



Residual graph G_f

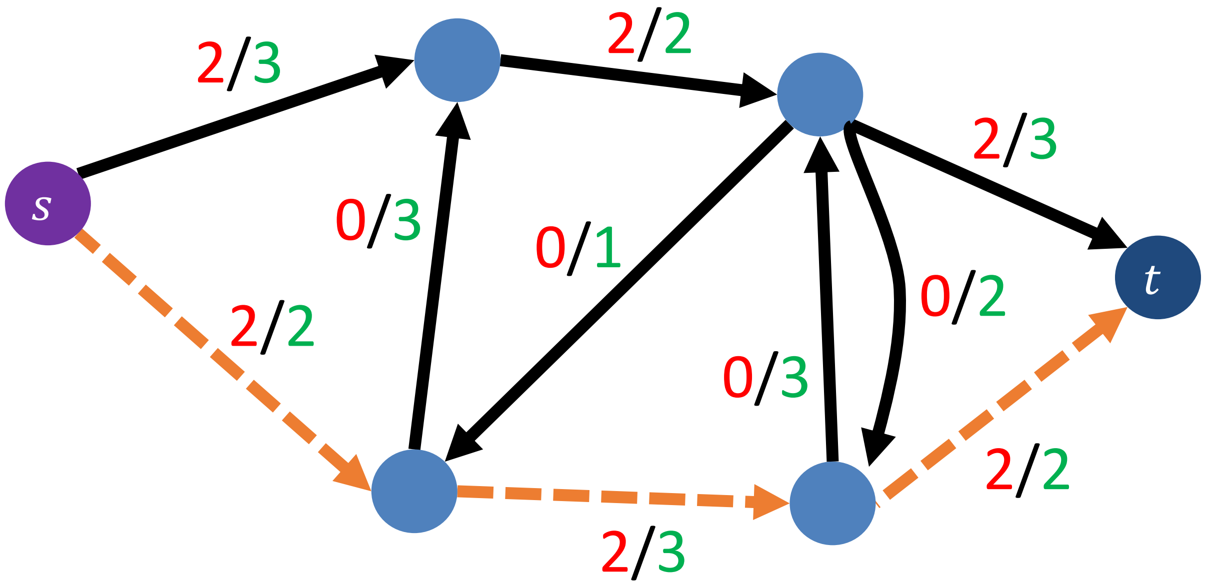
Ford-Fulkerson Example



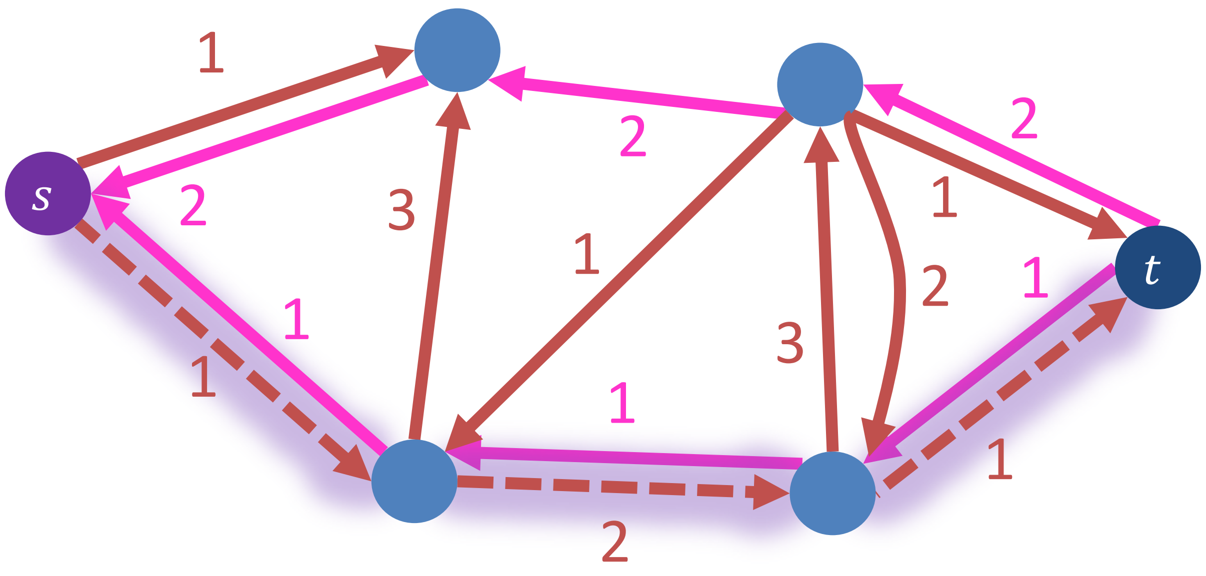


Residual graph G_f

Ford-Fulkerson Example

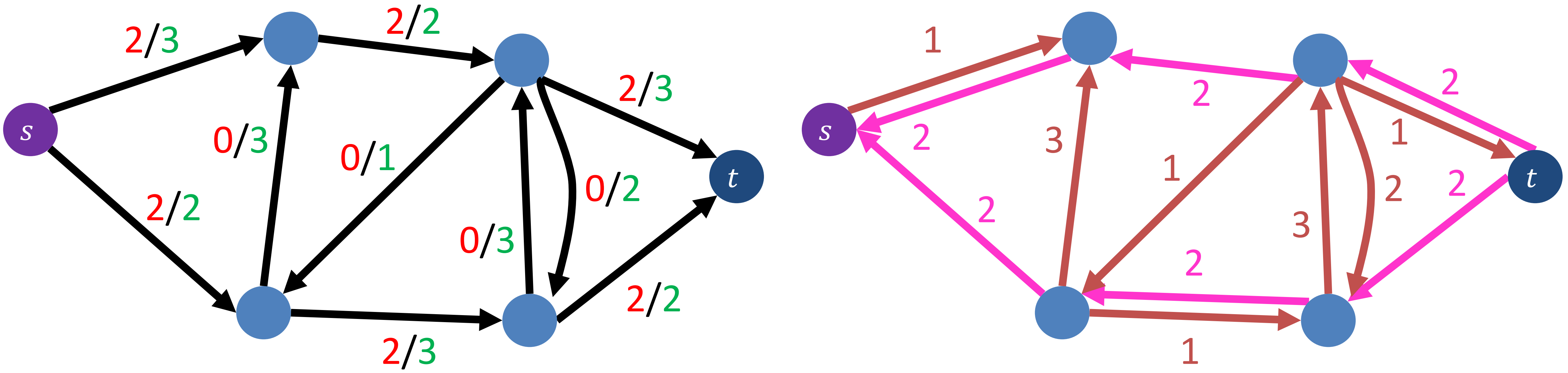


Increase flow by 1 unit



Residual graph G_f

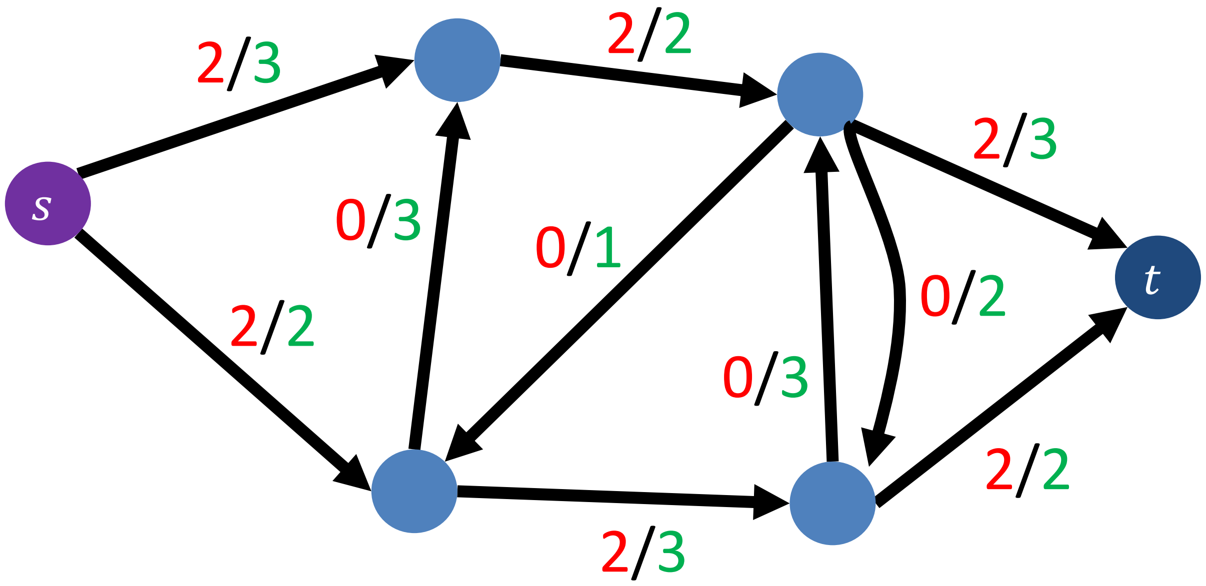
Ford-Fulkerson Example

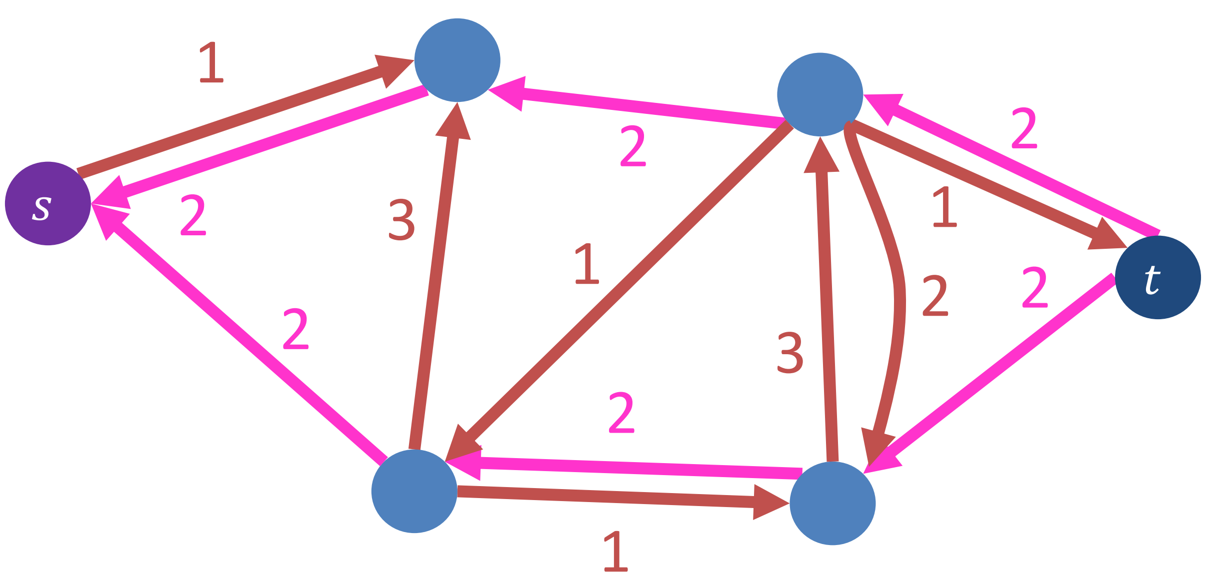


Residual graph G_f

Ford-Fulkerson Example

No more augmenting paths

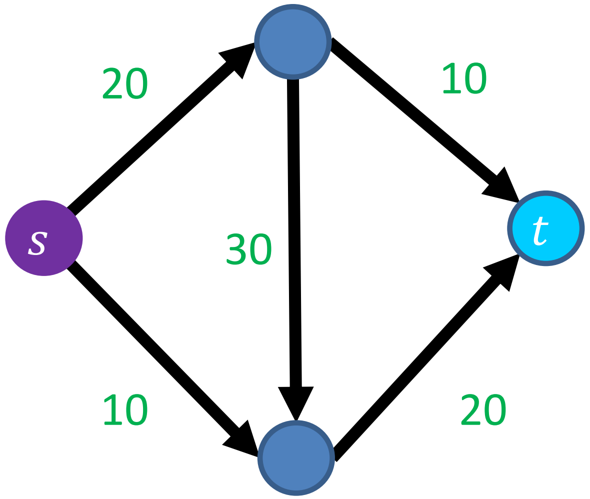




Residual graph G_f

Maximum flow: 4

Our example



Ford-Fulkerson Algorithm - Runtime

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

Overview: Repeatedly add the flow of any 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 p in G_f:
  - Let $c = \min_{u,v \in p} c_f(u, v)$
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

Time to find an augmenting path:

Number of iterations of While loop:

Ford-Fulkerson Algorithm - Runtime

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

Overview: Repeatedly add the flow of any 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 p in G_f:
  - Let $c = \min_{u,v \in p} c_f(u, v)$
  - Add c units of flow to G based on the augmenting path p
  - Update the residual network G_f for the updated flow

Time to find an augmenting path: DFS: $\Theta(V + E)$

Number of iterations of While loop: $|f|$

$\Theta(E \cdot |f|)$

What type of search?

- "While there is an augmenting path p in G_f"
  - Using a depth-first search is the Ford-Fulkerson algorithm
    - Each augmenting path can be found in O(E) time
    - And there can be $|f|$ paths
    - So the running time is O($E \cdot |f|$)
    - Will not terminate with irrational edge values
  - Using a breadth-first search is the Edmonds-Karp algorithm
    - Runs in $O(V \cdot E^2)$
      - Total number of augmentations is O(V·E)
      - And finding each augmentation takes O(E)
    - Guaranteed termination with irrational edge values
    - Run-time is independent of the maximum flow of the graph