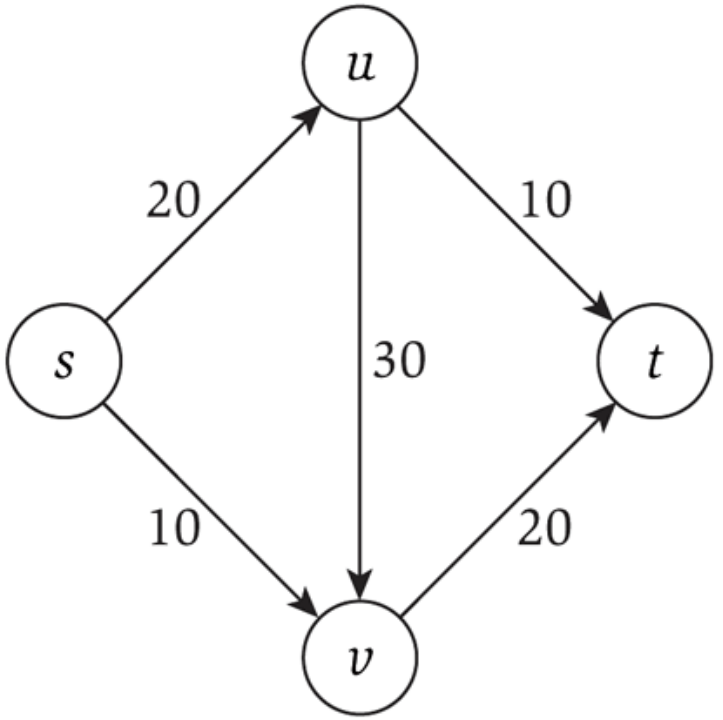
FLOW NETWORK

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

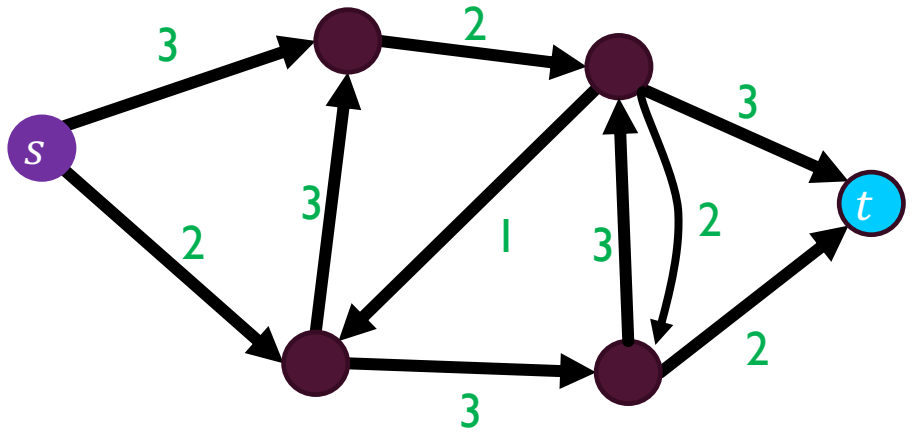
Graph $G = (V, E)$

Source node $s \in V$

Sink node $t \in V$

Edge Capacities $c(e)$ ∈ Positive whole numbers

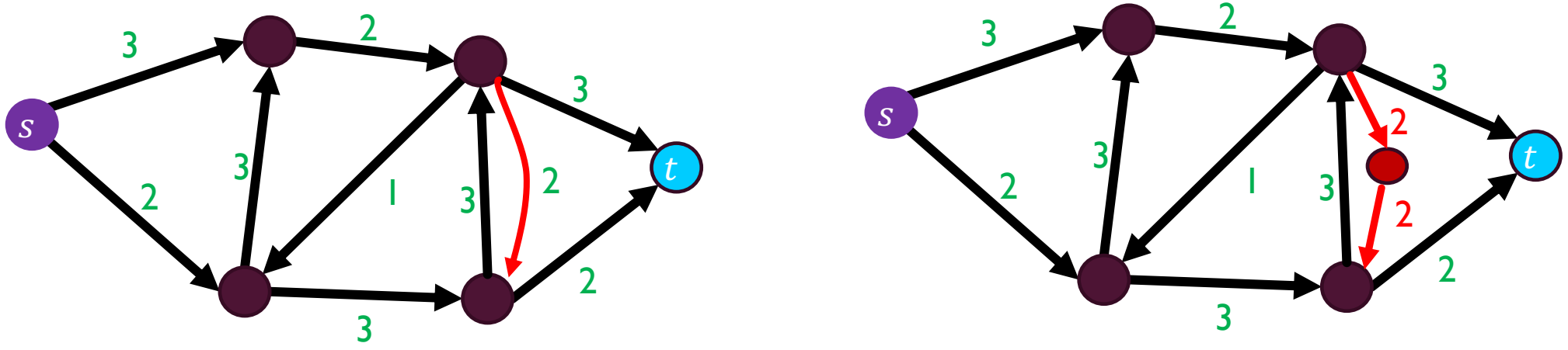
If $(u, v) \in E$ then $(v, u) \notin E$ (Note our example here violates this!)



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

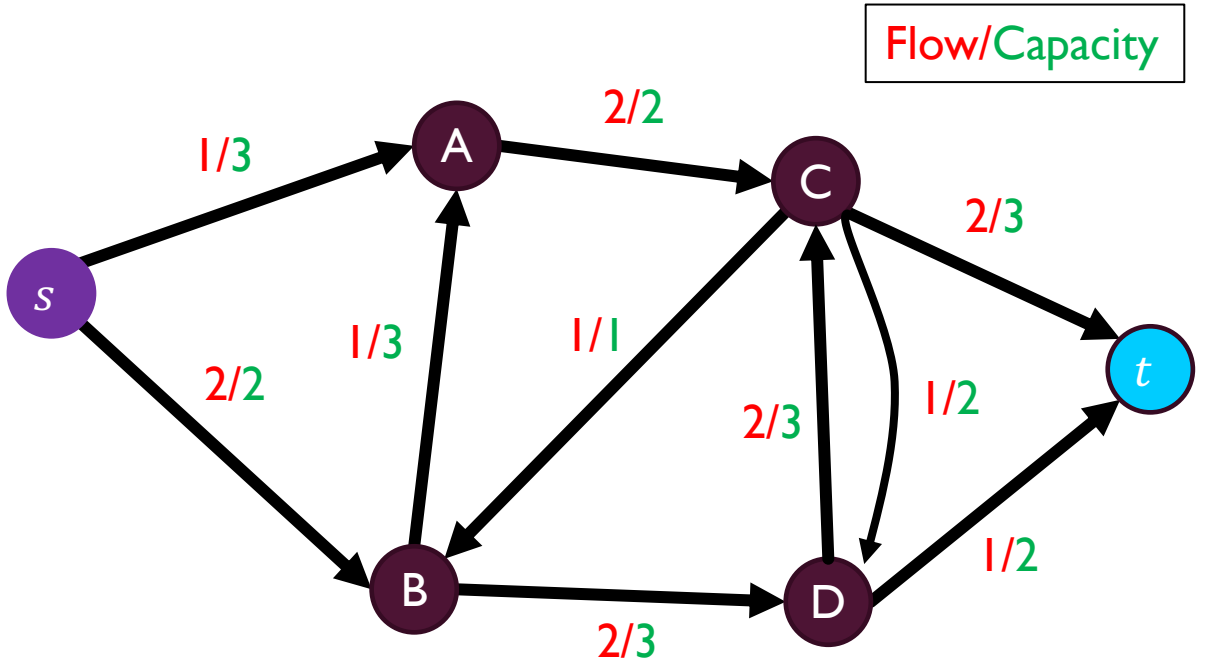
Easy adjustment to remove antiparallel edges and have equivalent flow graph: add intermediate node



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

FLOW

- 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
  - $\forall v \in V - \{s, t\}, inflow(v) = outflow(v)$
  - $inflow(v) = \sum_{x \in V} f(x, v)$
  - $outflow(v) = \sum_{x \in V} f(v, x)$
  - *Water going in must match water coming out*
- Flow of G: $|f| = outflow(s) - inflow(s)$
  - Net outflow of s



Flow of G=?

LET'S MAKE SOME RULES

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

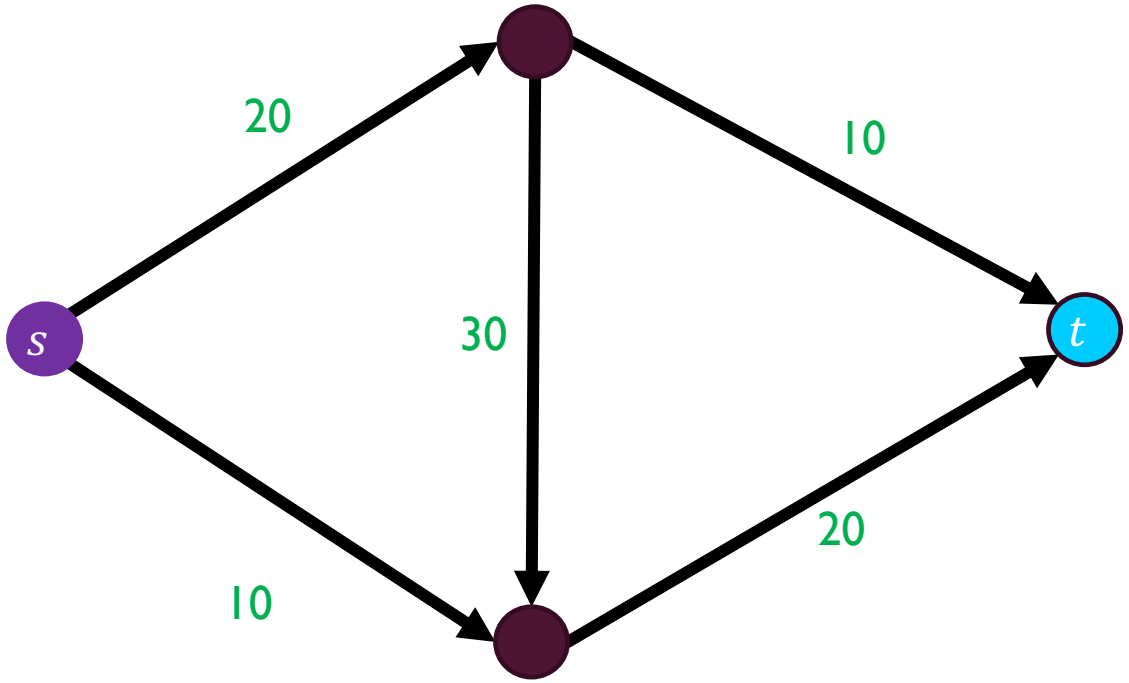
- Internal nodes has net zero flow
  - all units of flow going in must be going out as well

- No edge is over capacity

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

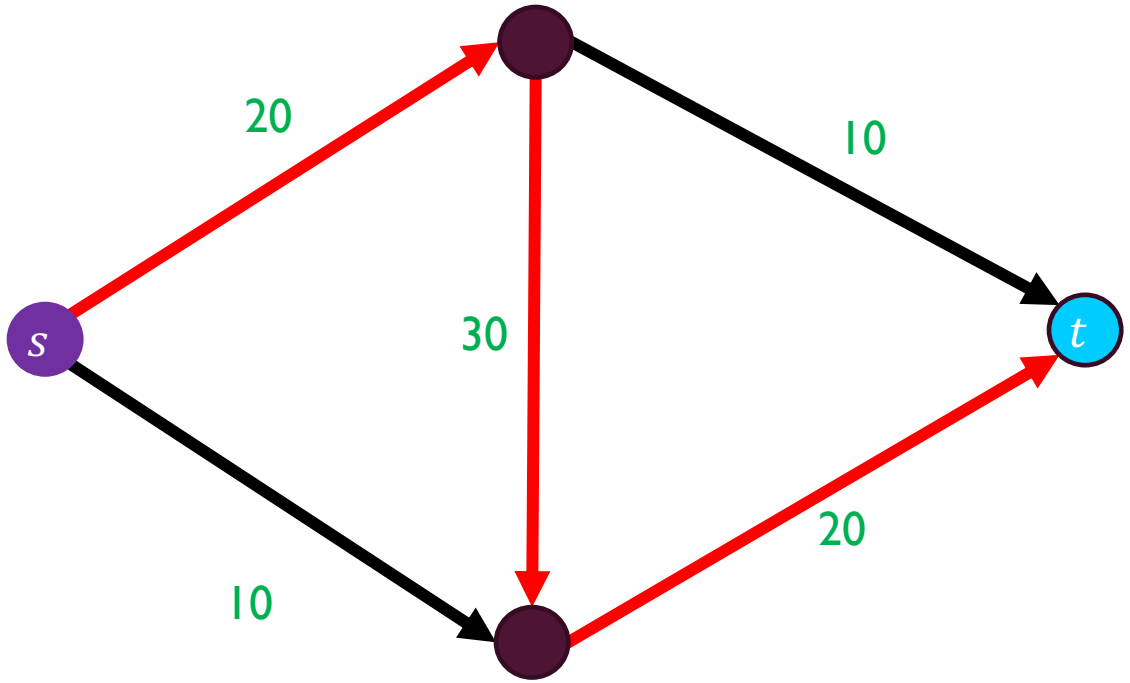
HOW TO SOLVE THIS? LET'S TRY GREEDY!

Saturate Highest Capacity Path First



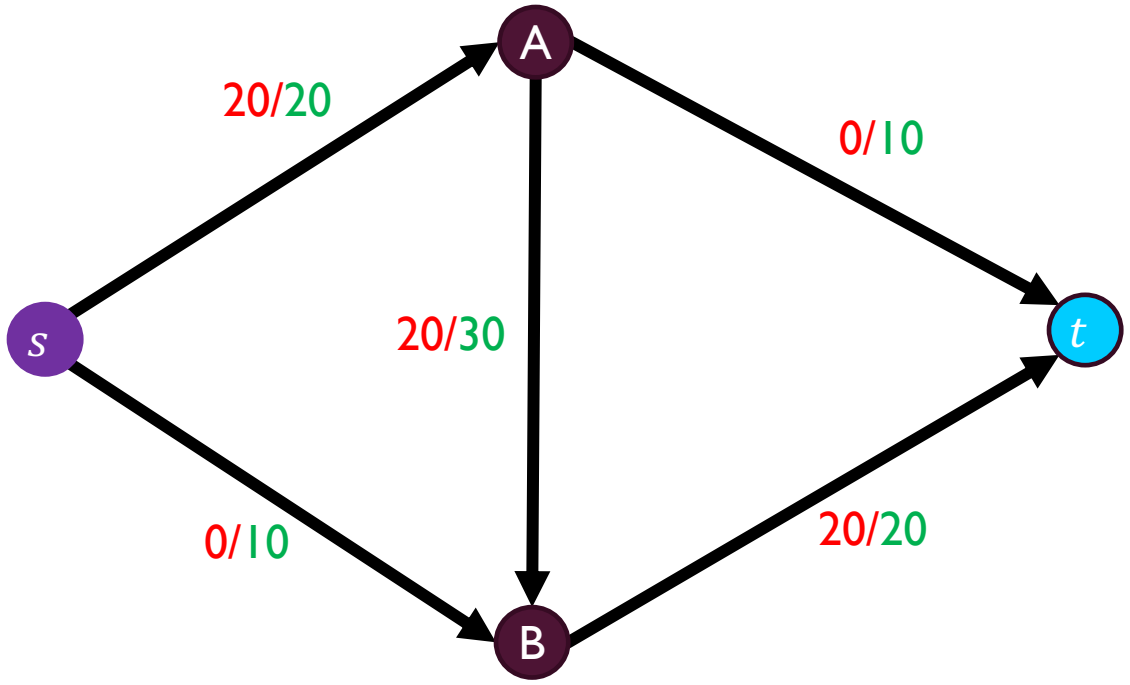
GREEDY?

Saturate Highest Capacity Path First



GREEDY?

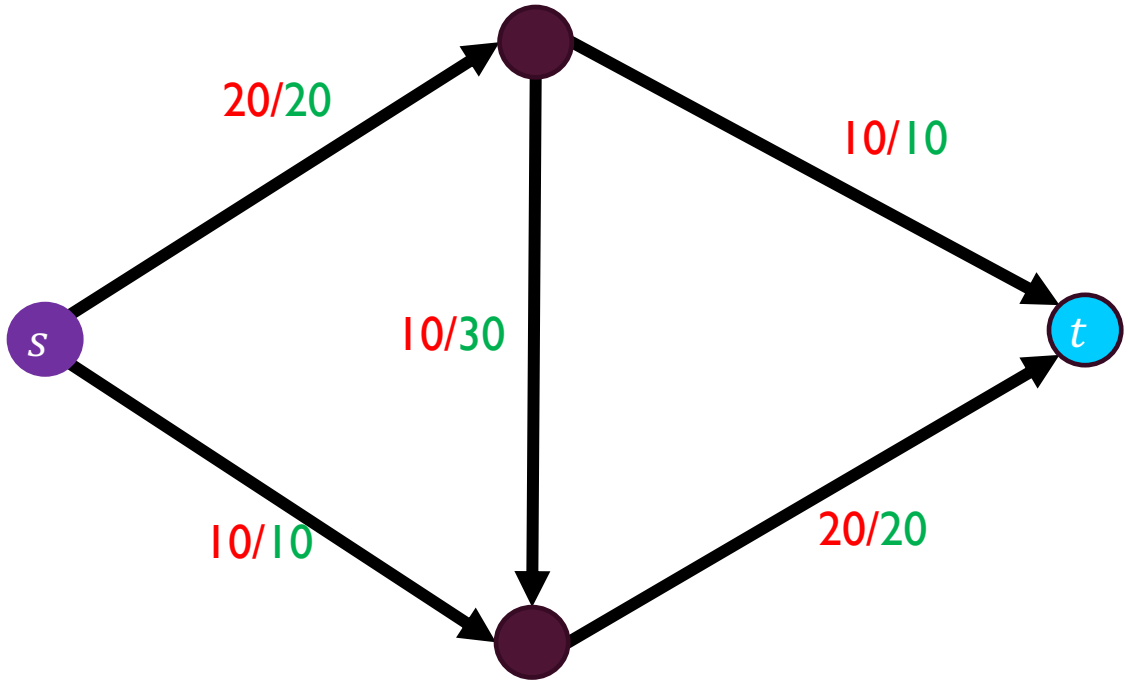
Saturate Highest Capacity Path First



Overall Flow: $|f| = 20$

GREEDY DOESN'T WORK

We have 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"*

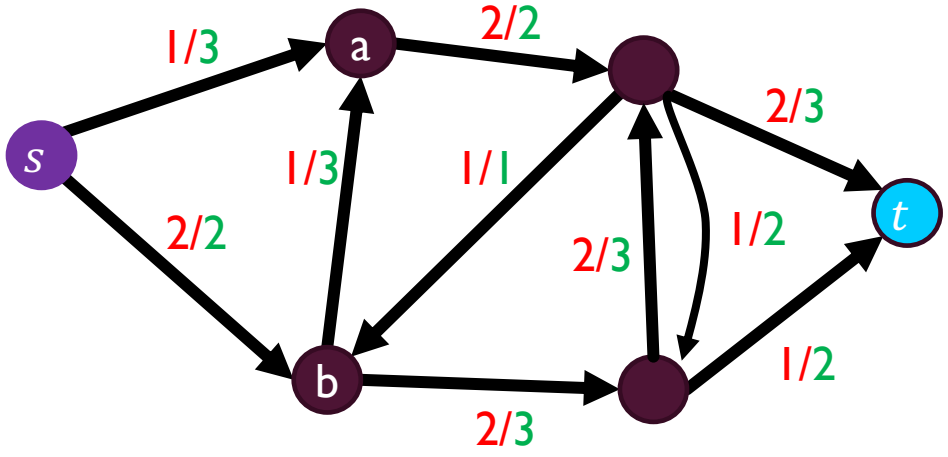
RESIDUAL GRAPH G_f

- 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
  - $w(e) = f(e)$

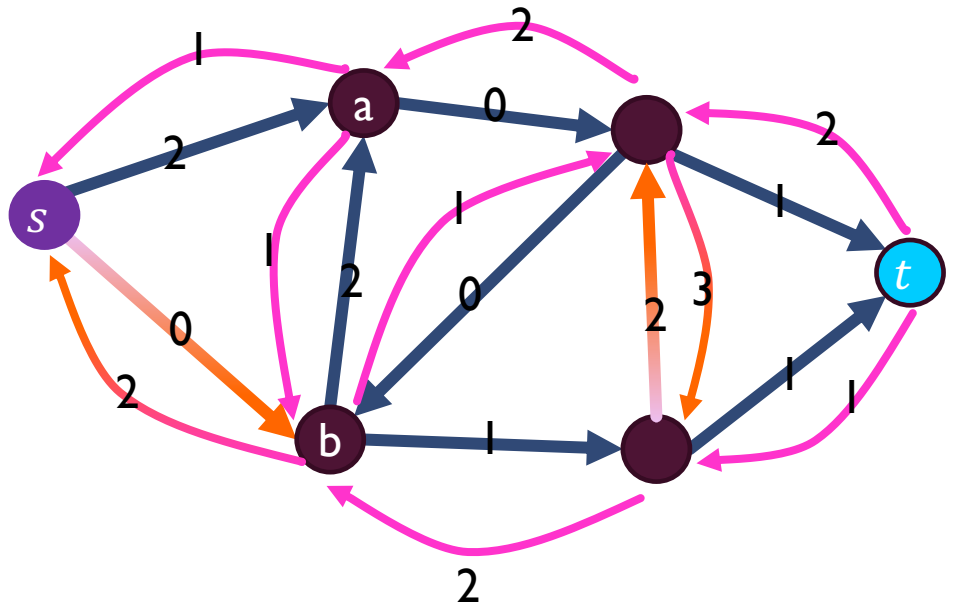
Flow I *could* add

Flow I *could* remove

Flow Graph G

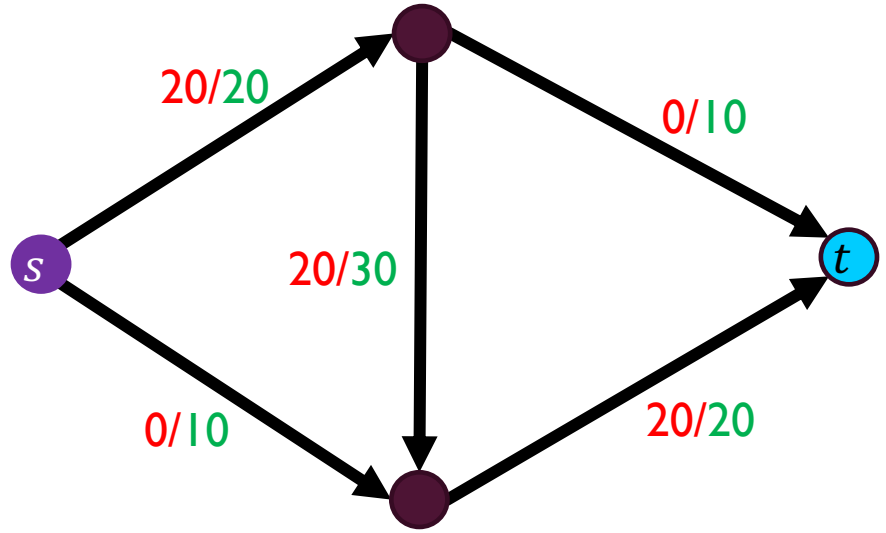


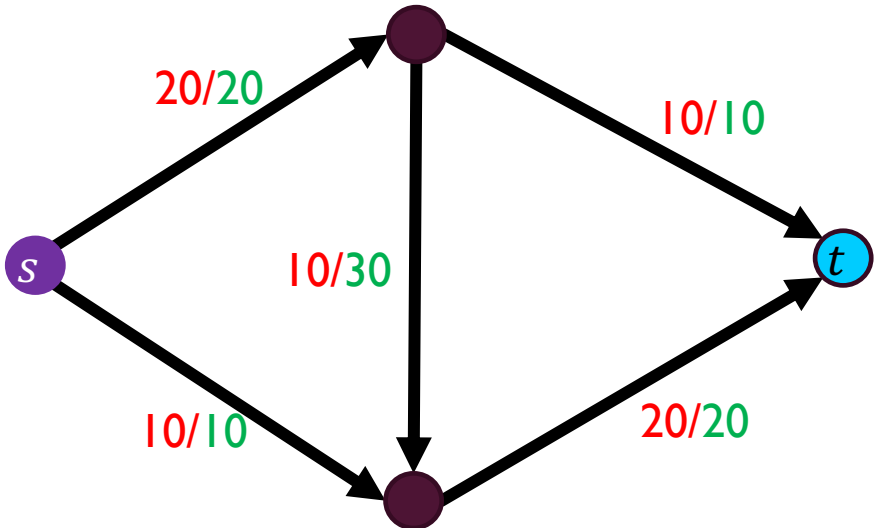
Residual Graph G_f



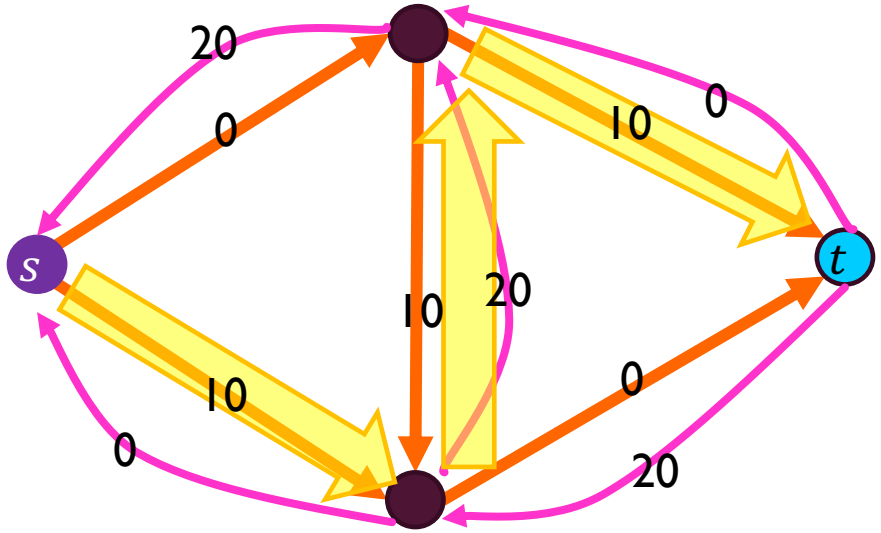
RESIDUAL GRAPHS EXAMPLE

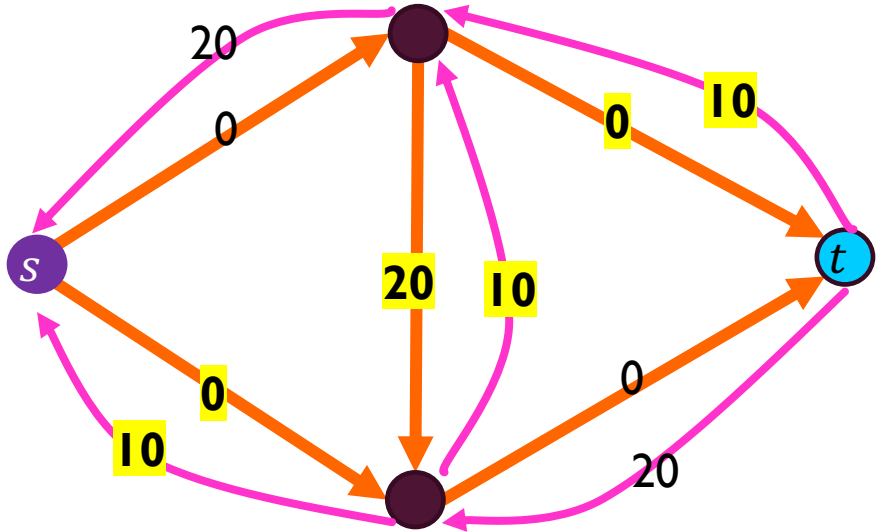
Flow Graph





Residual Graph





FORD-FULKERSON ALGORITHM

Define an **augmenting path** to be a path from $s \rightarrow t$ 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 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 any 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

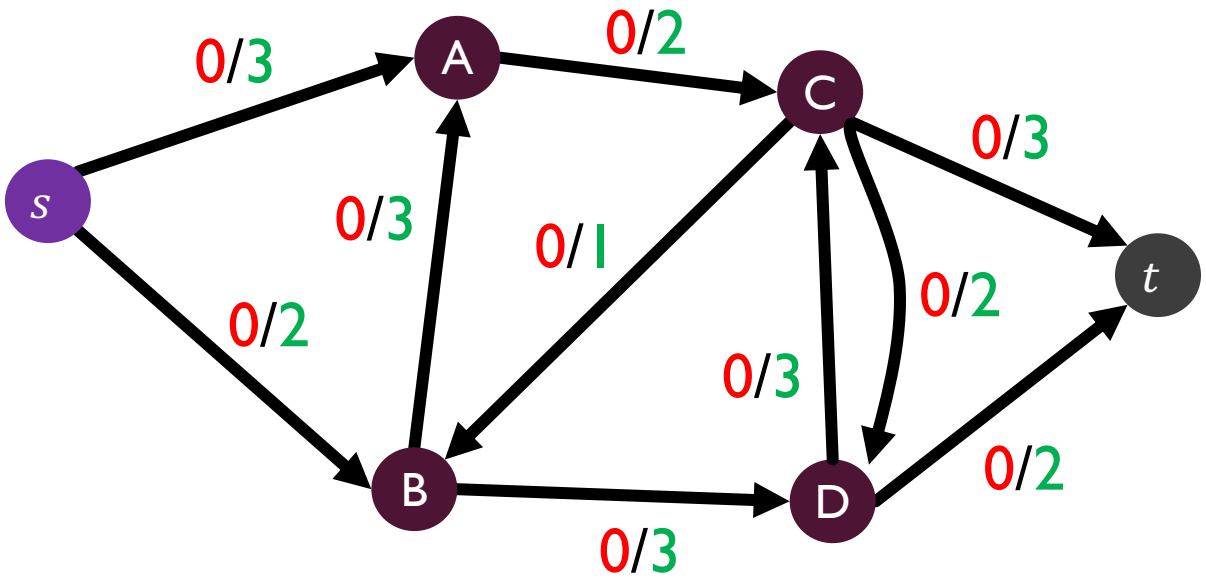
Ford-Fulkerson approach:
take any augmenting path
(will revisit this later)

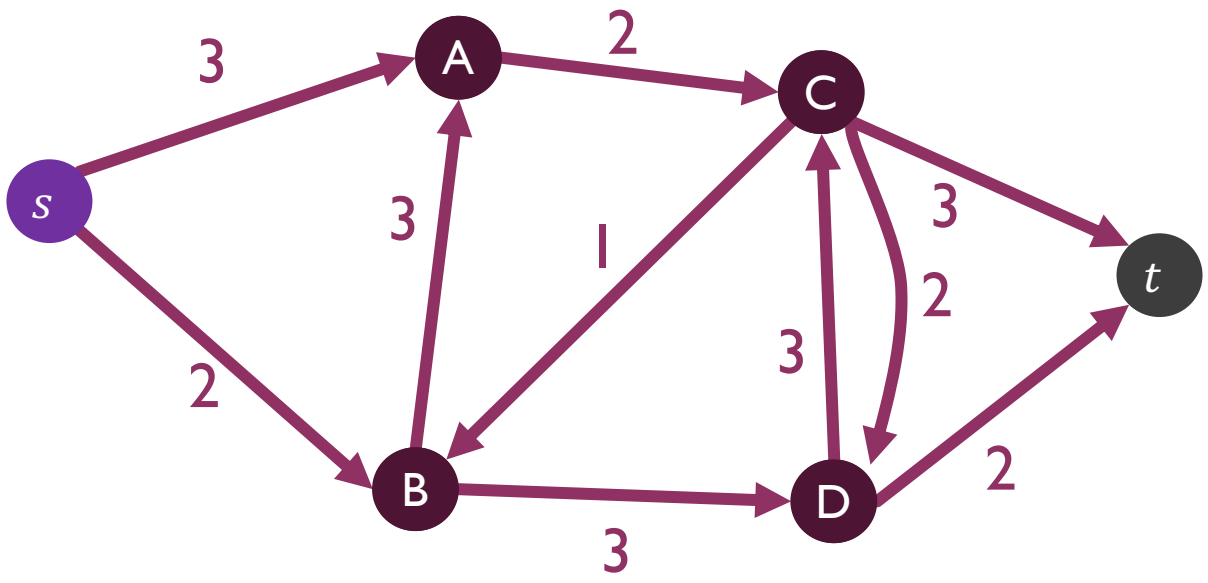
($c_f(u, v)$ is the weight of edge (u, v) in the residual network G_f)

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

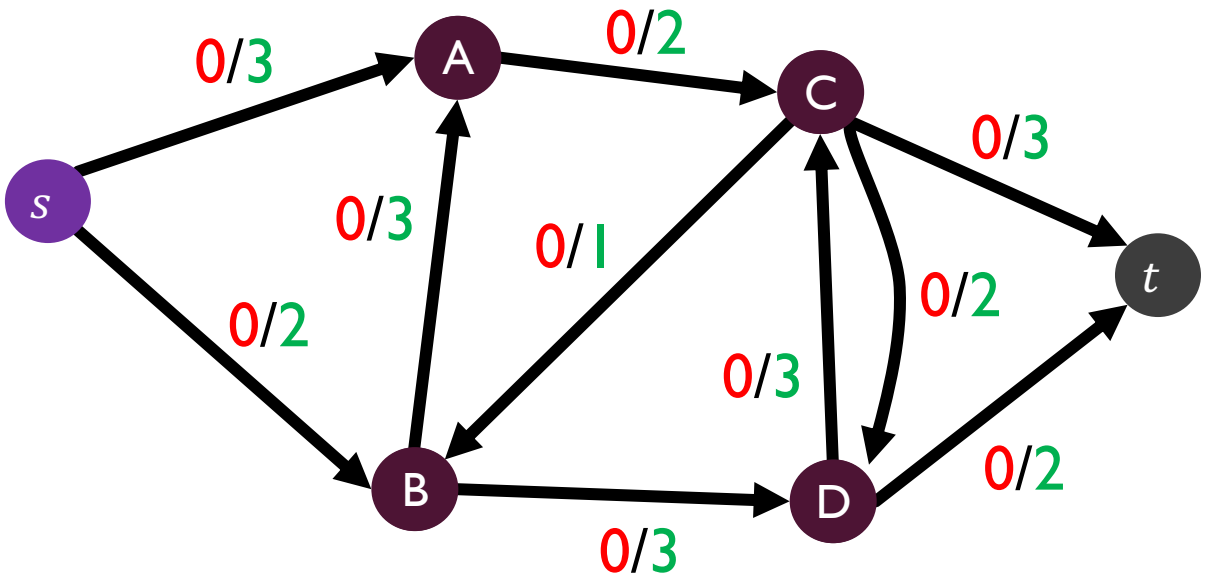




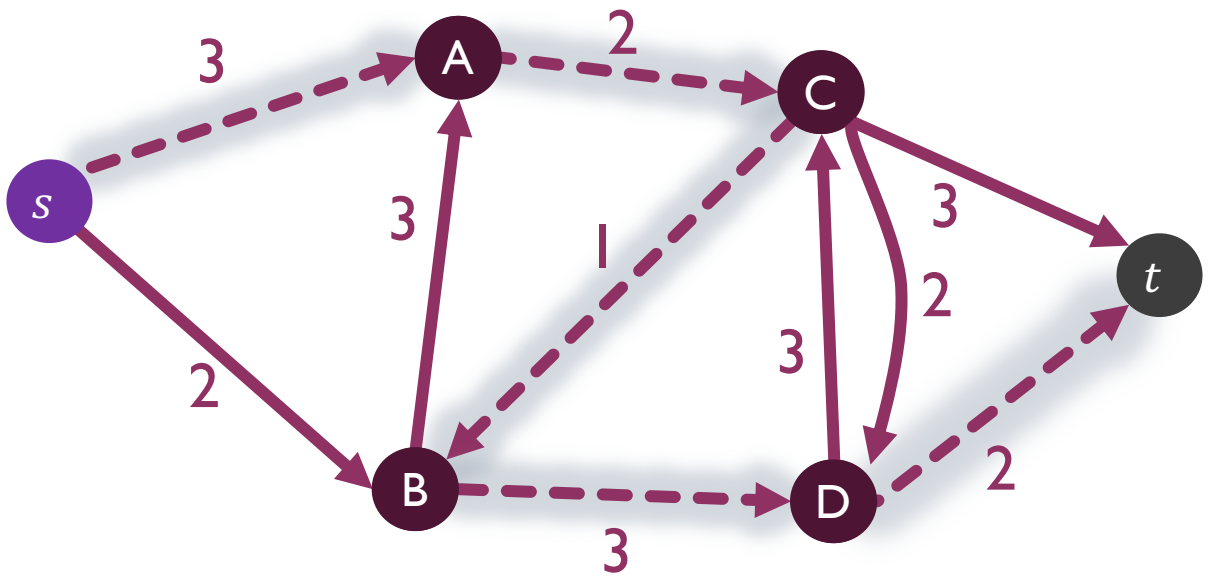
Initially: $f(e) = 0$ for all $e \in E$

Residual graph G_f

FORD-FULKERSON EXAMPLE



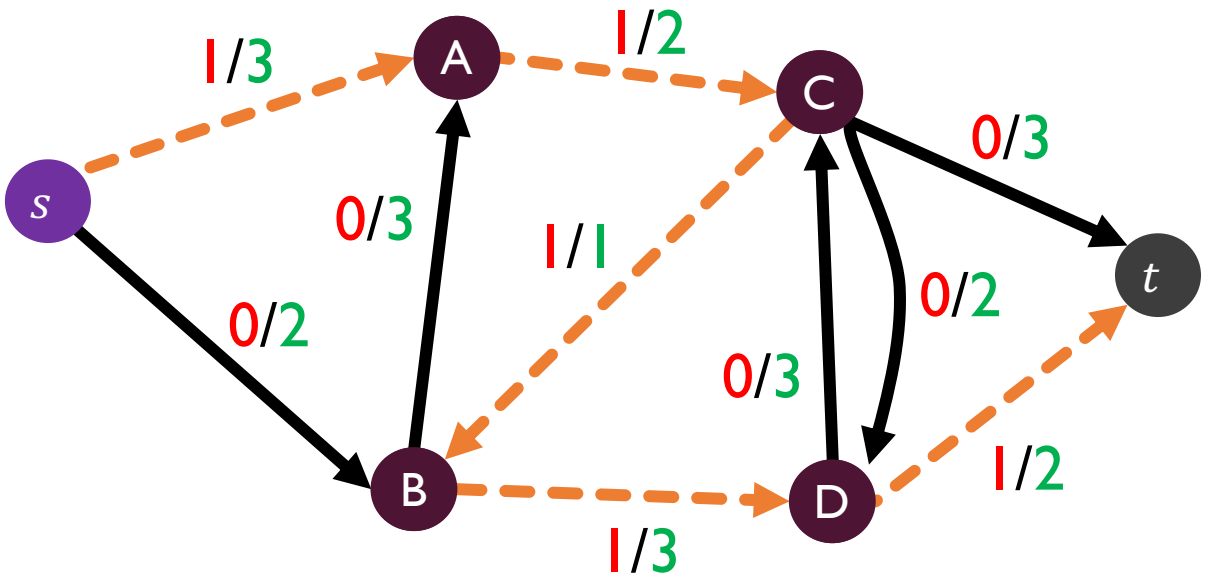
Increase flow by 1 unit

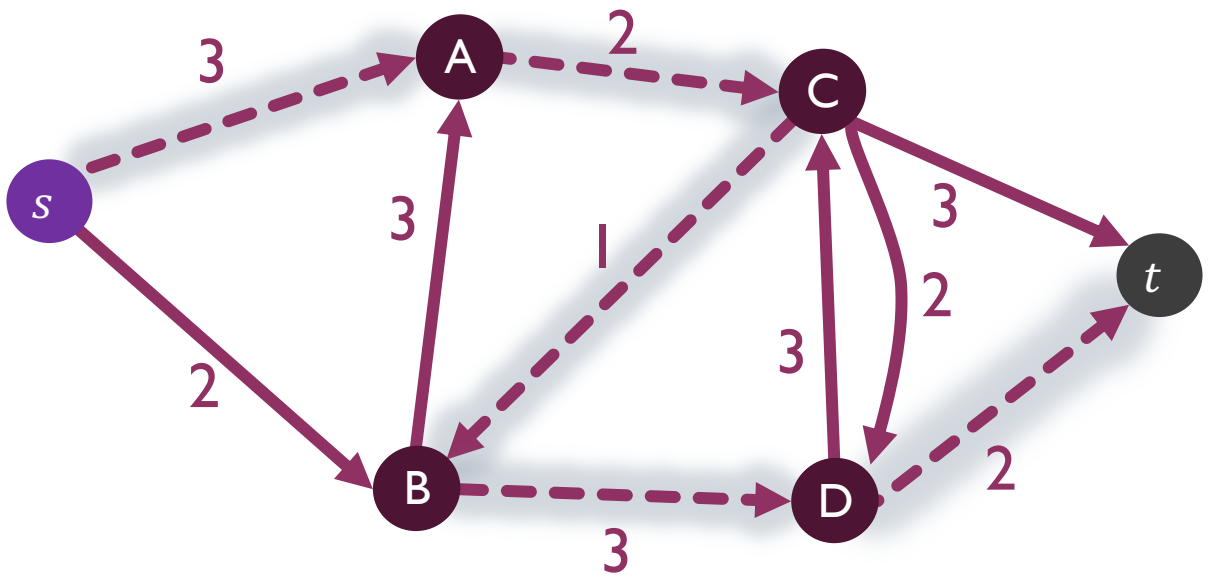


Residual graph G_f

FORD-FULKERSON EXAMPLE

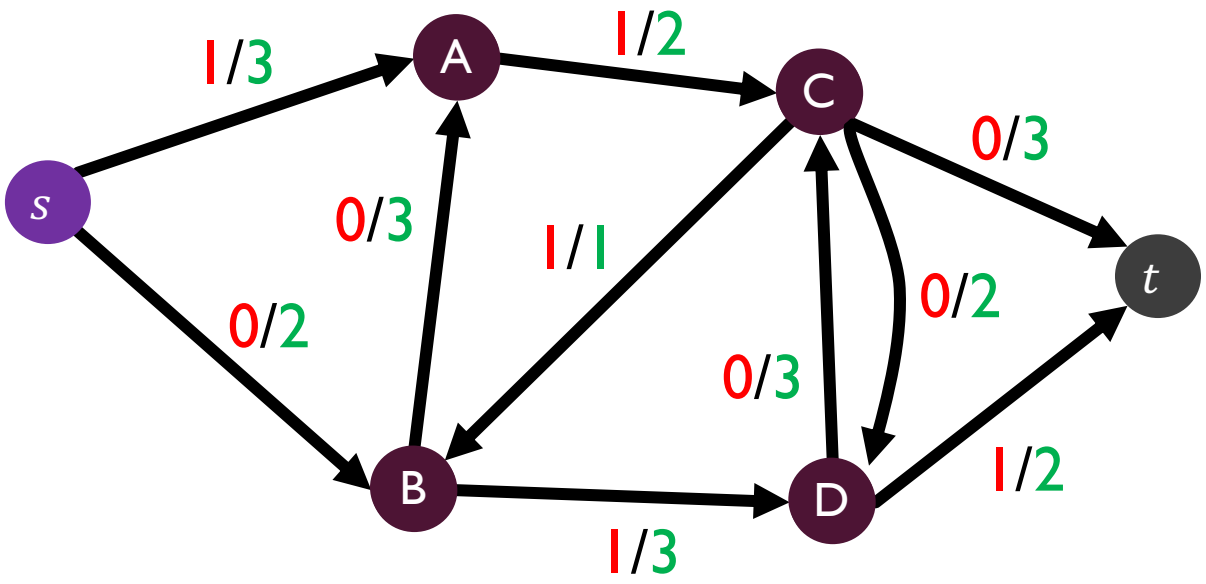
Increase flow by 1 unit

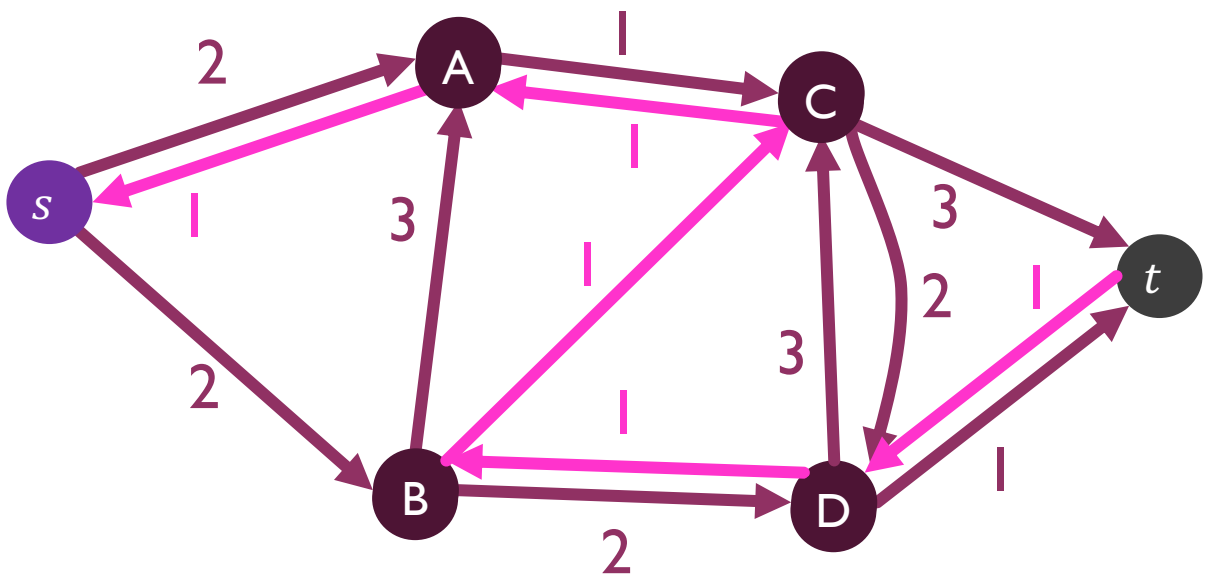




Residual graph G_f

FORD-FULKERSON EXAMPLE

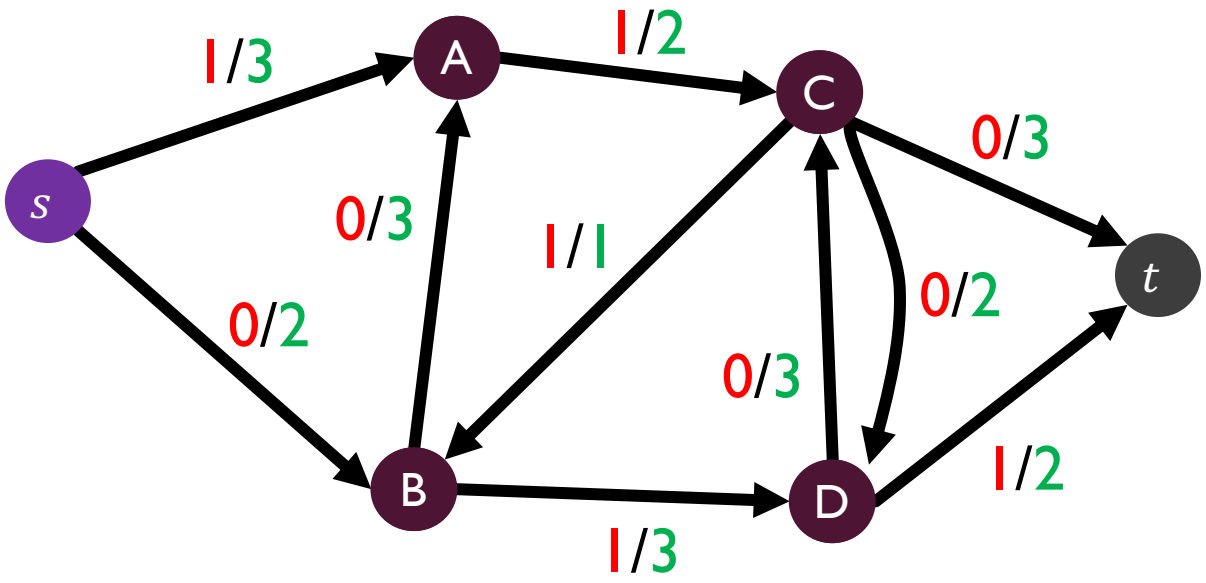


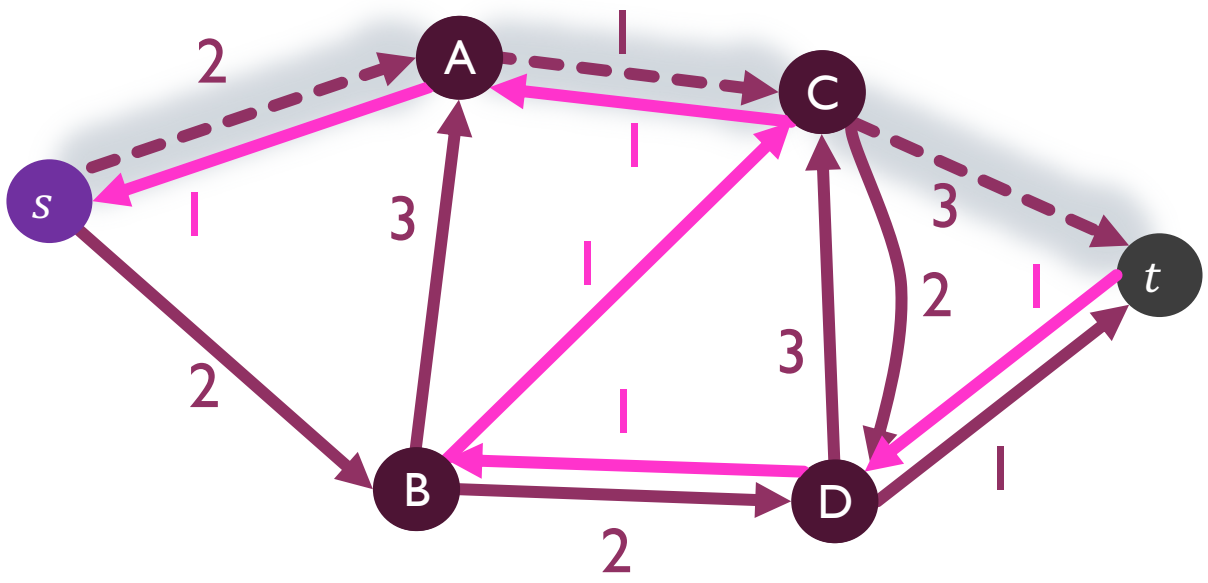


Residual graph G_f

FORD-FULKERSON EXAMPLE

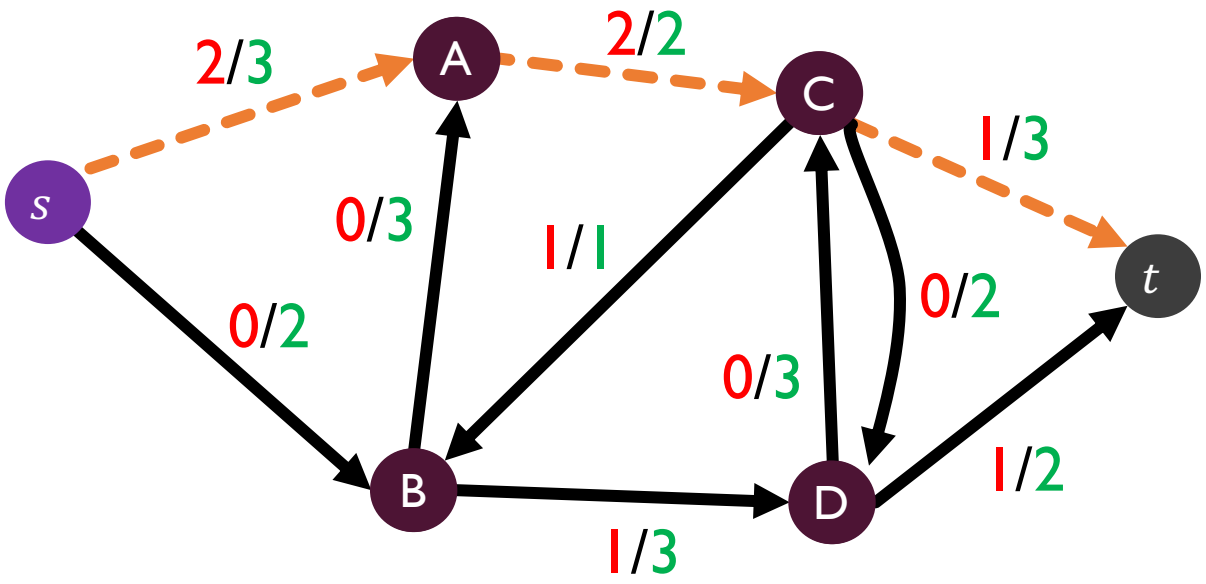
Increase flow by 1 unit



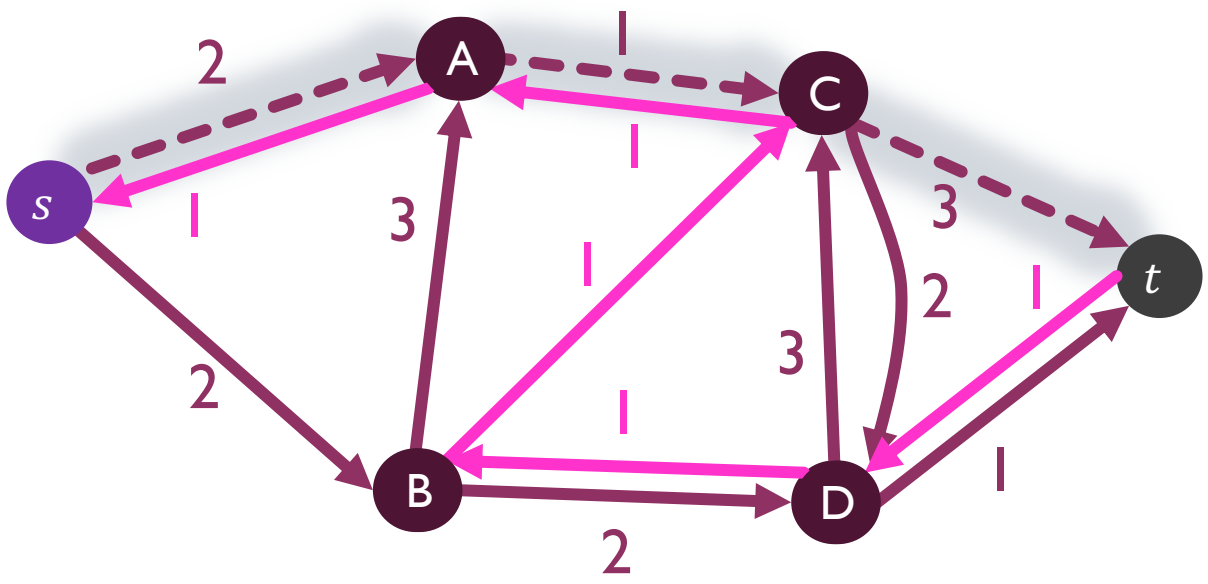


Residual graph G_f

FORD-FULKERSON EXAMPLE



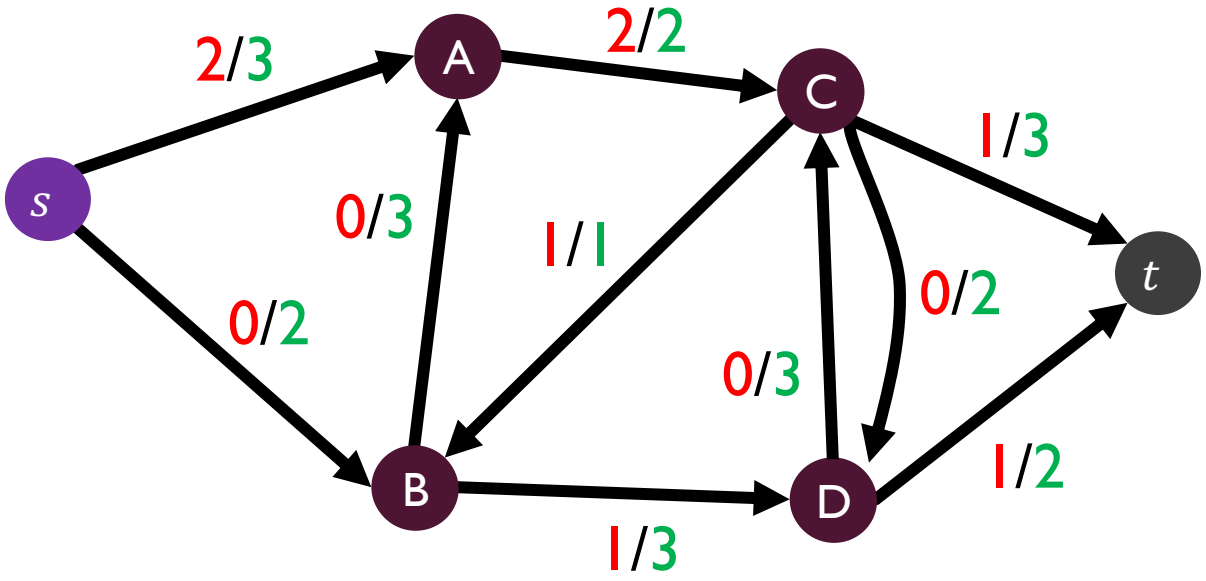
Increase flow by 1 unit

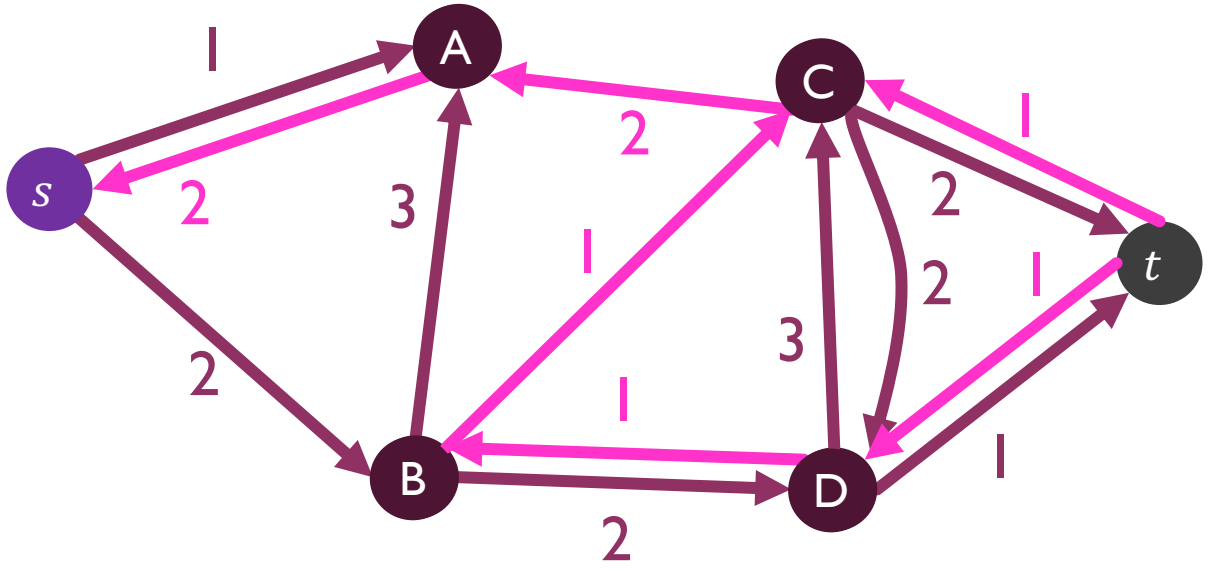


Residual graph G_f

FORD-FULKERSON EXAMPLE

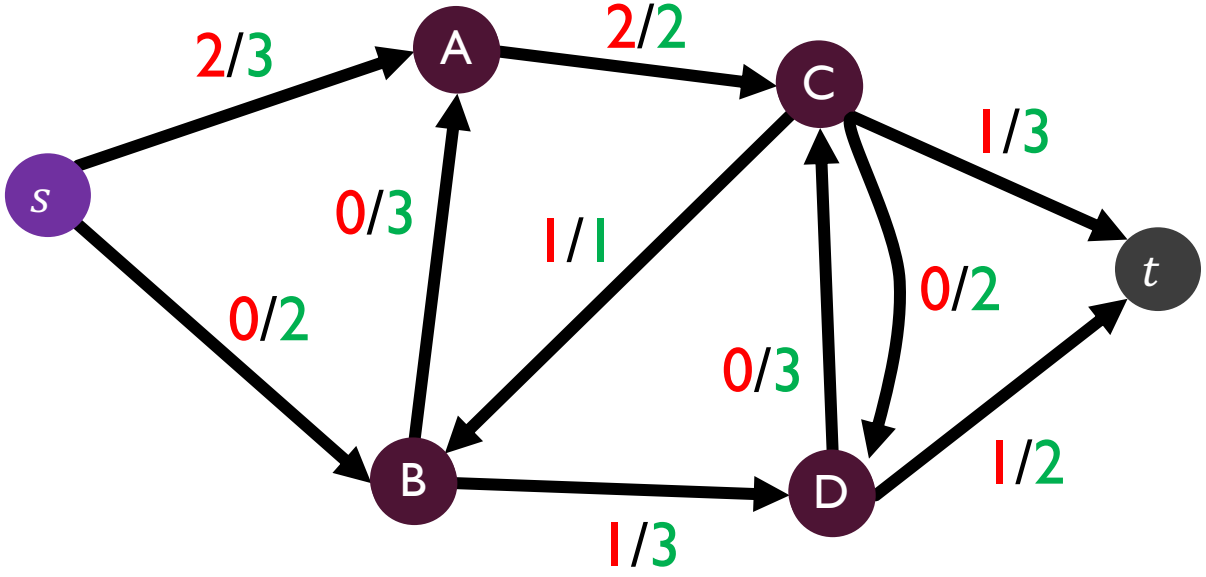
Increase flow by 1 unit

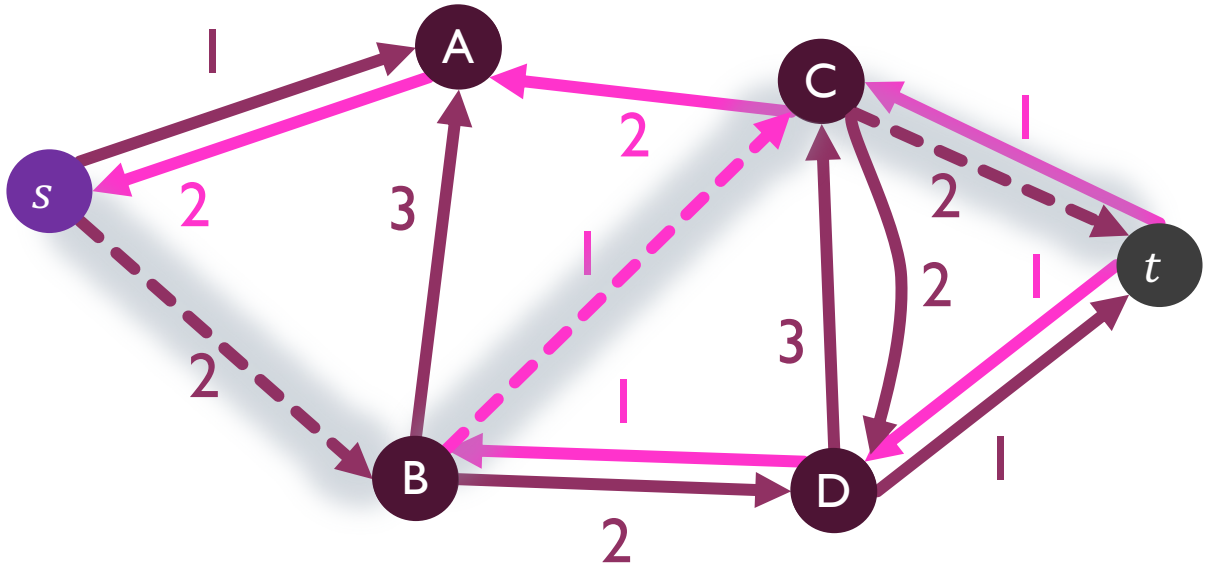




Residual graph G_f

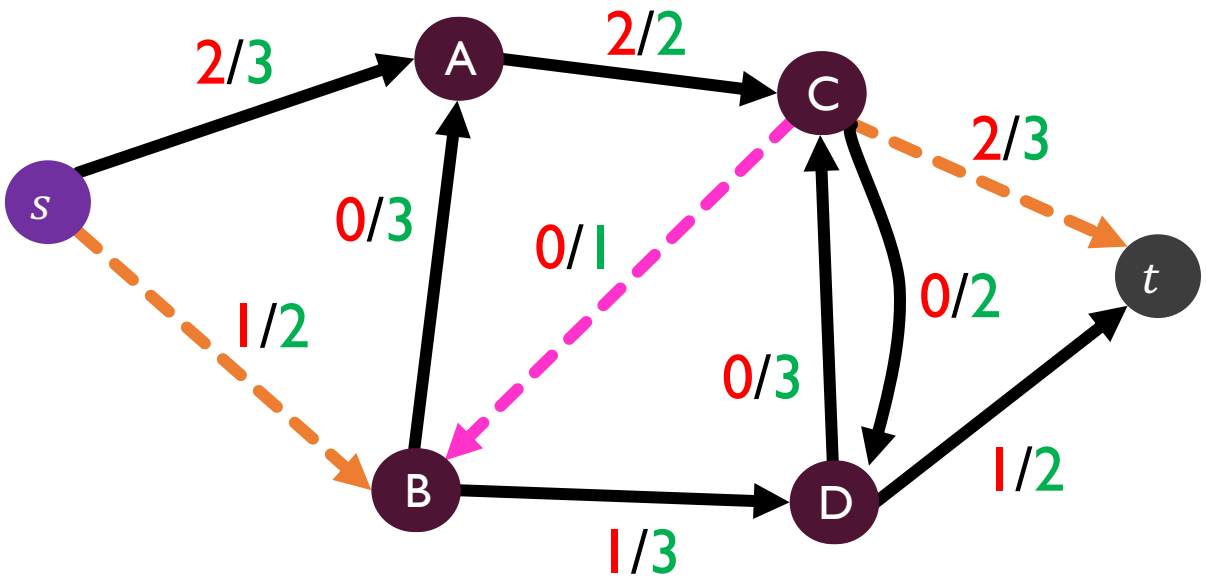
FORD-FULKERSON EXAMPLE



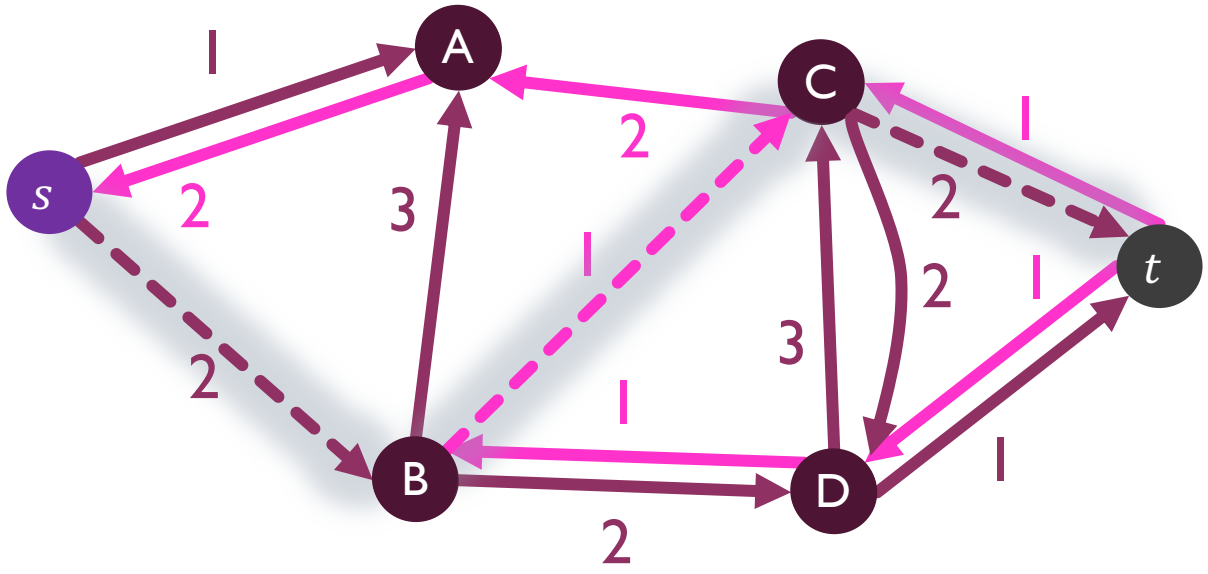


Residual graph G_f

FORD-FULKERSON EXAMPLE

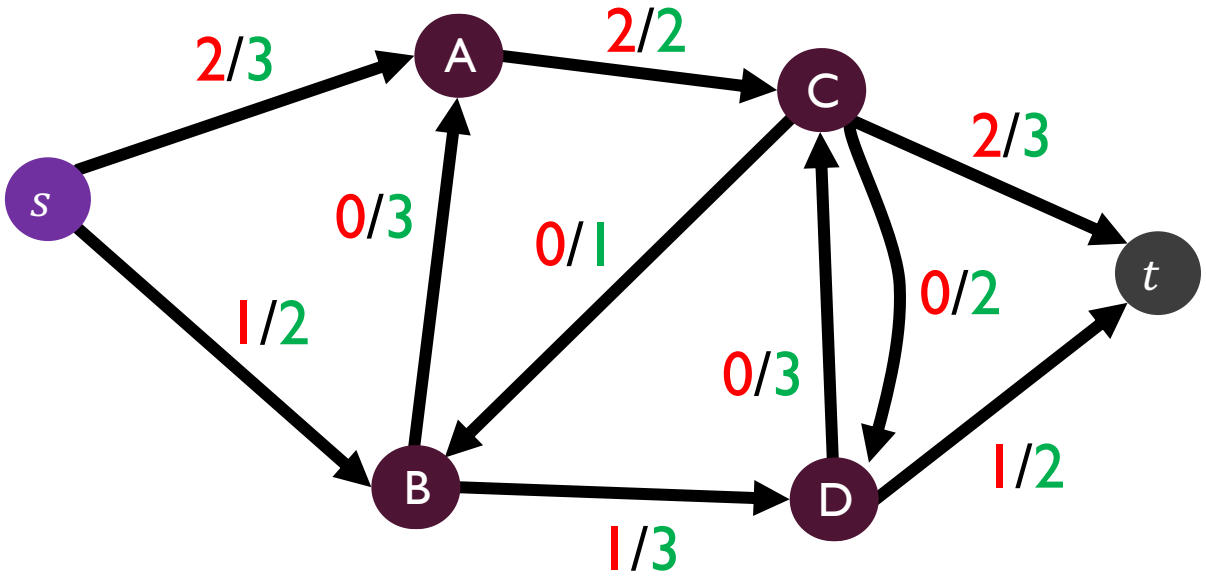


Increase flow by 1 unit

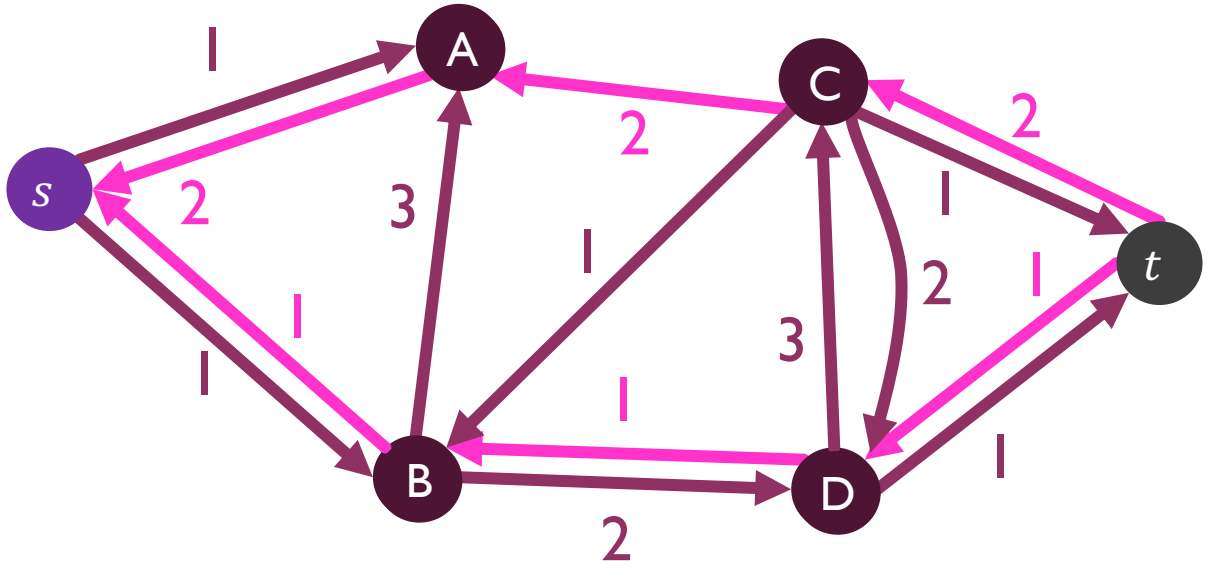


Residual graph G_f

FORD-FULKERSON EXAMPLE

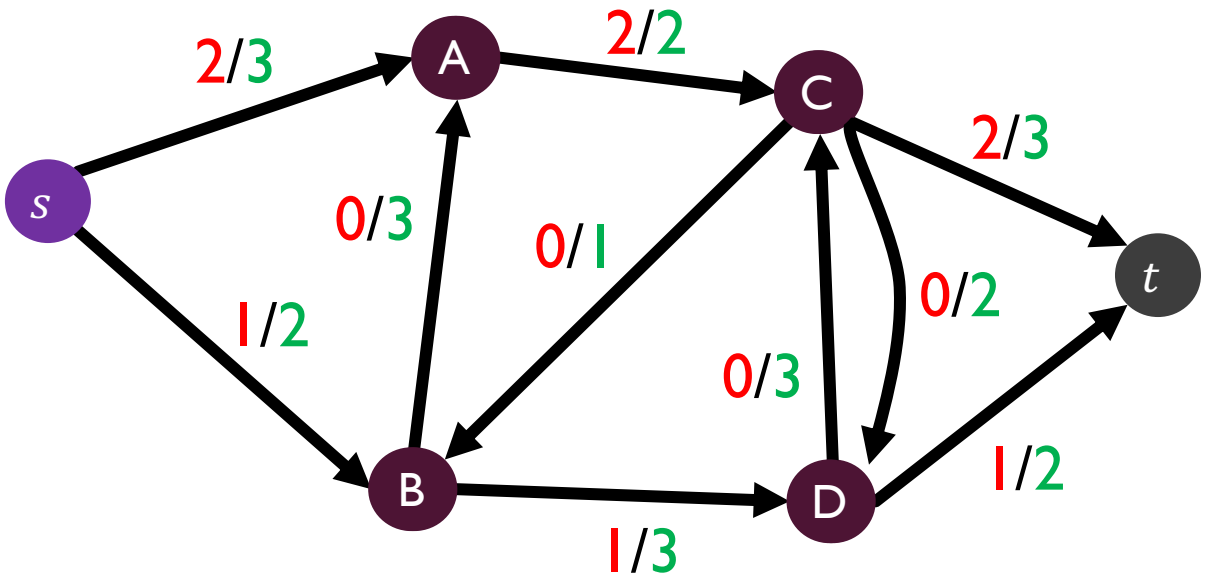


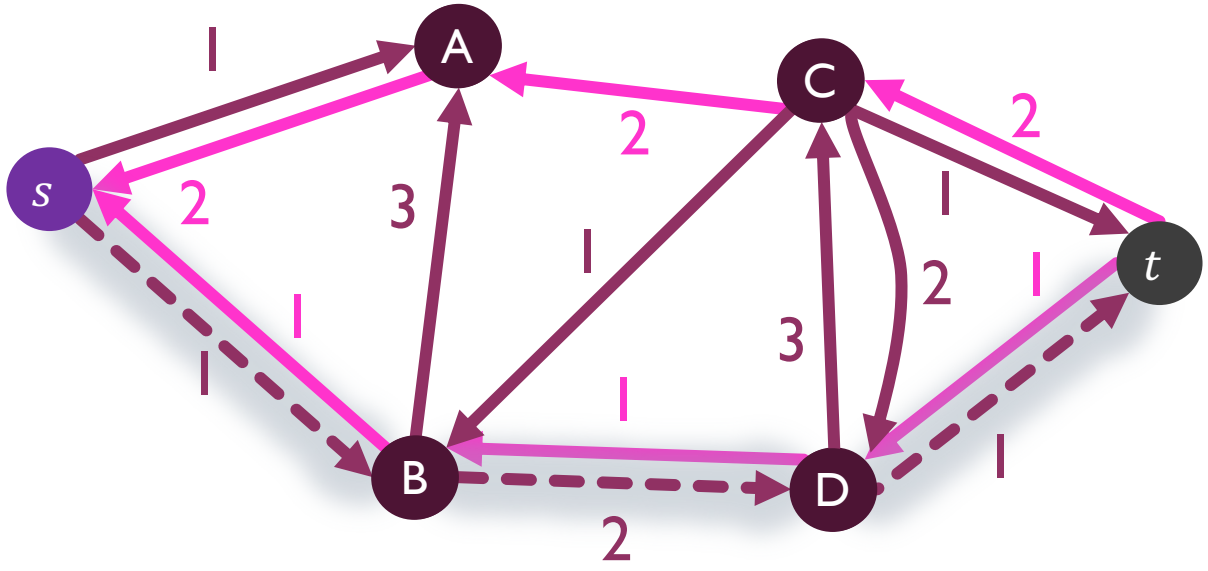
Increase flow by 1 unit



Residual graph G_f

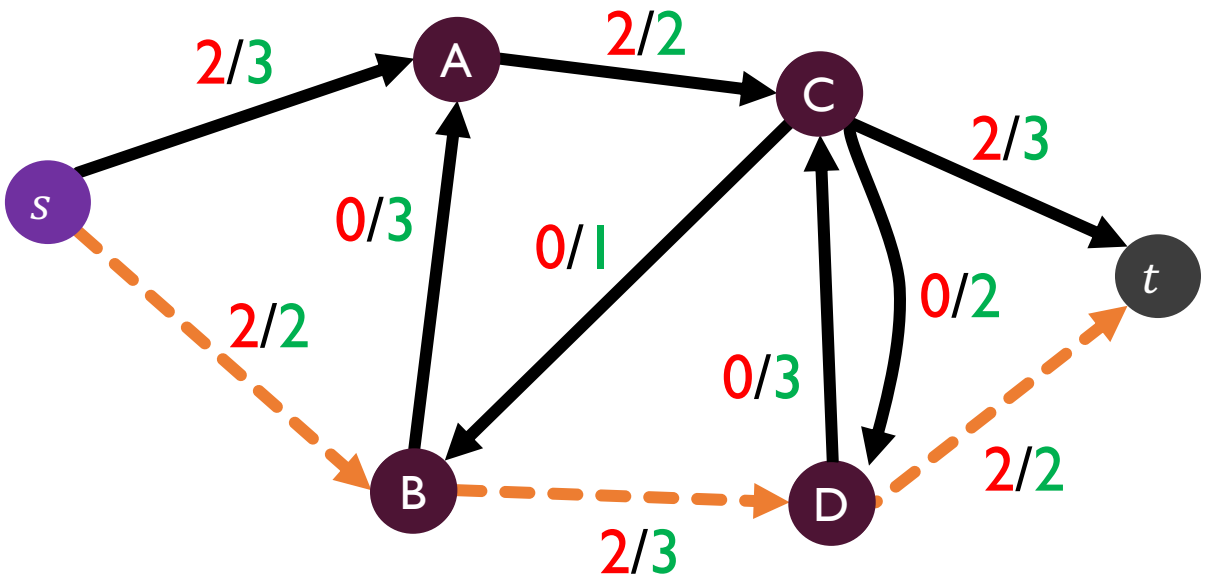
FORD-FULKERSON EXAMPLE



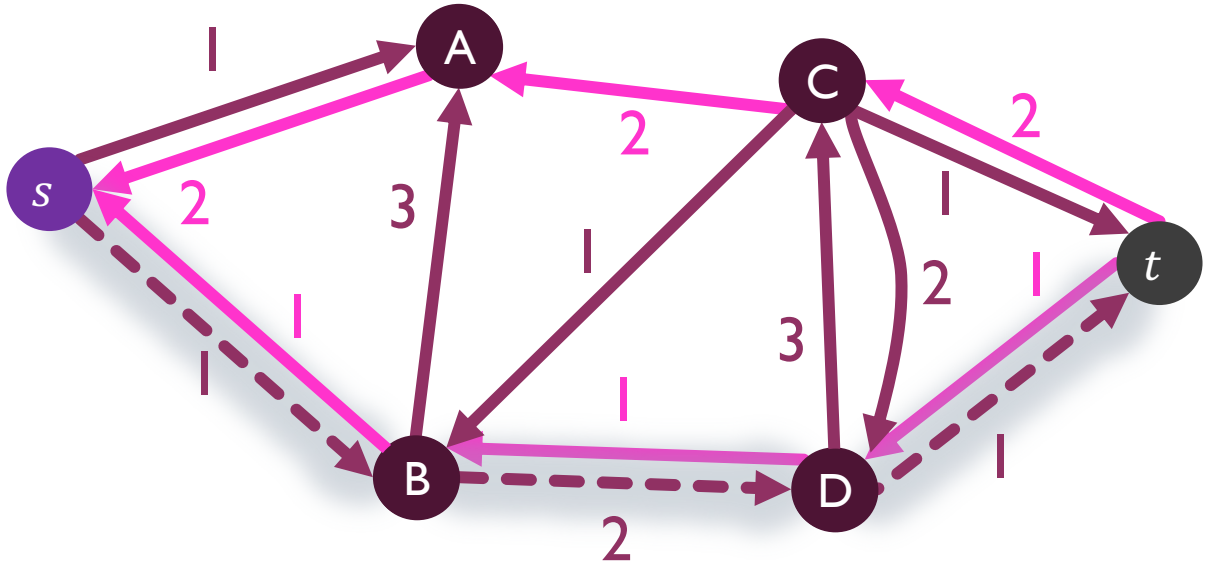


Residual graph G_f

FORD-FULKERSON EXAMPLE

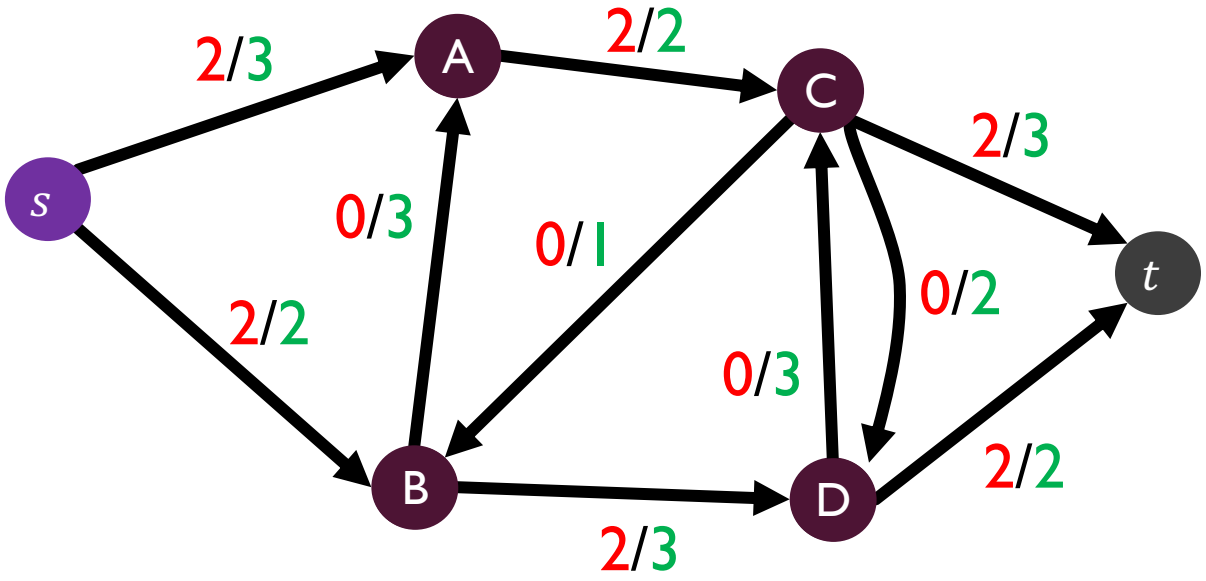


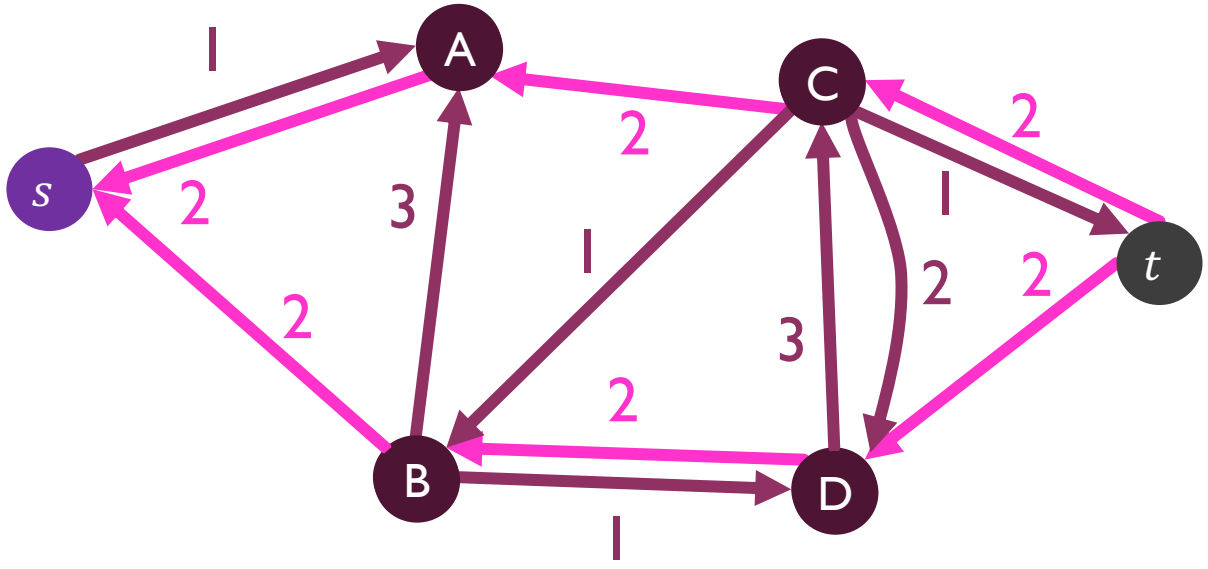
Increase flow by 1 unit



Residual graph G_f

FORD-FULKERSON EXAMPLE

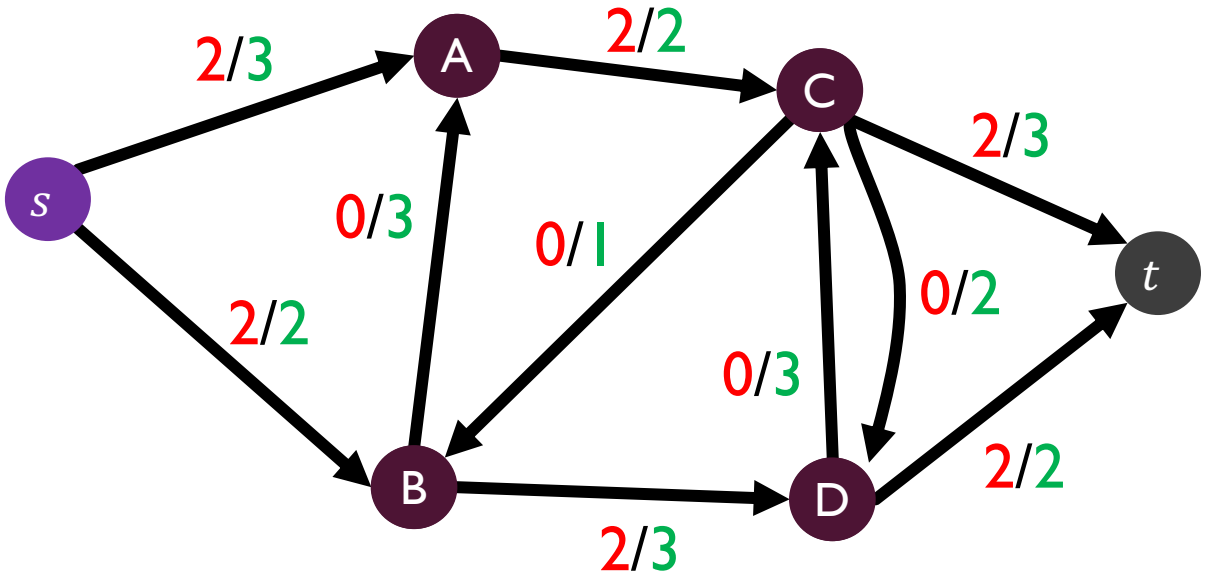


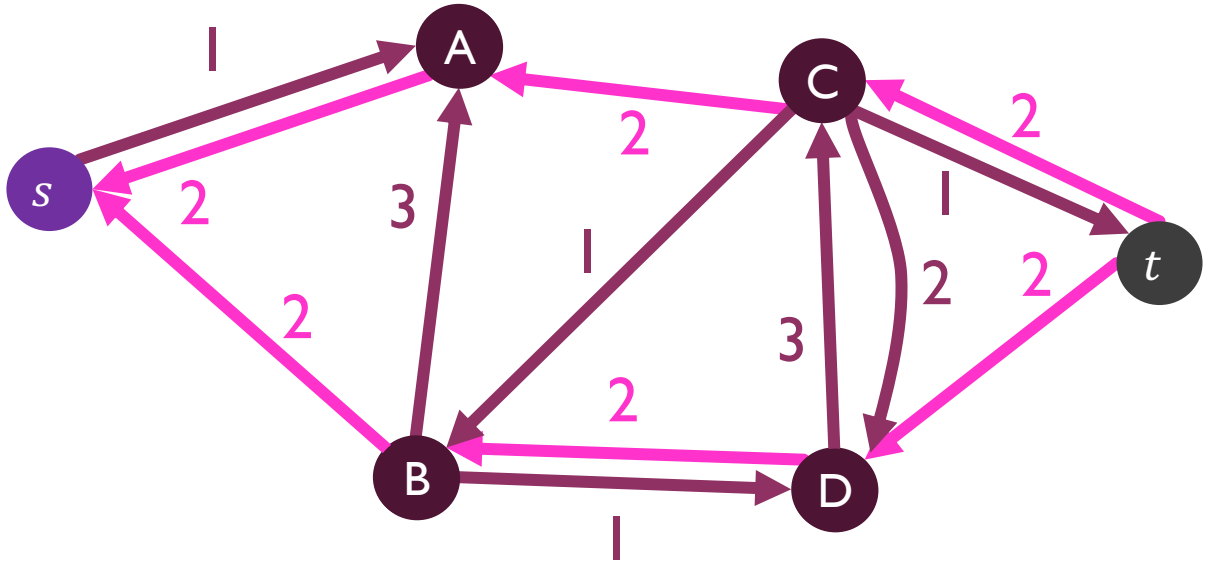


Residual graph G_f

FORD-FULKERSON EXAMPLE

No more augmenting paths





Maximum flow: 4

Residual graph G_f

FORD-FULKERSON ALGORITHM - RUNTIME

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

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 $s \to t$ 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

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

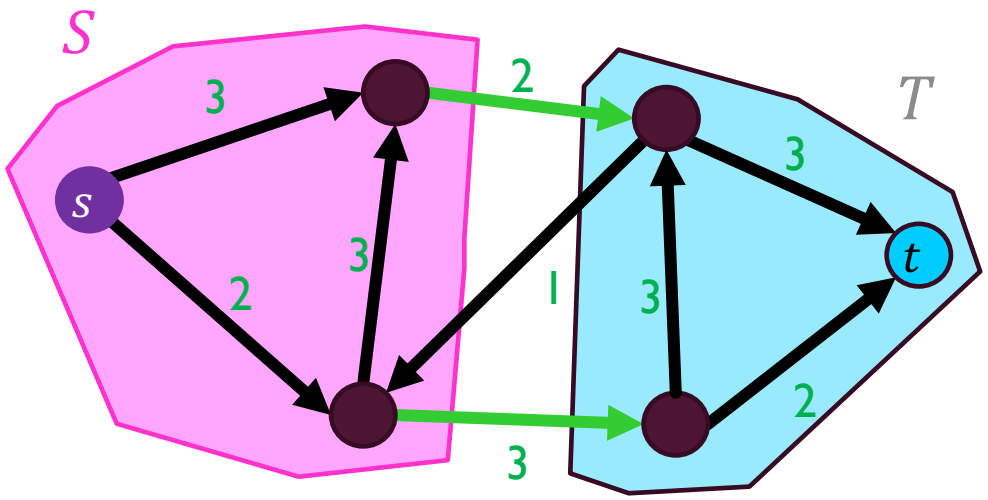
SHOWING CORRECTNESS OF FORD-FULKERSON

- To prove Ford-Fulkerson correct
  - We'll use the idea of a **cut** in a network flow graph
  - We'll prove properties related to cuts and flows

CUTS IN NETWORK FLOW GRAPHS

- Given a flow network, we want to *cut* edges…

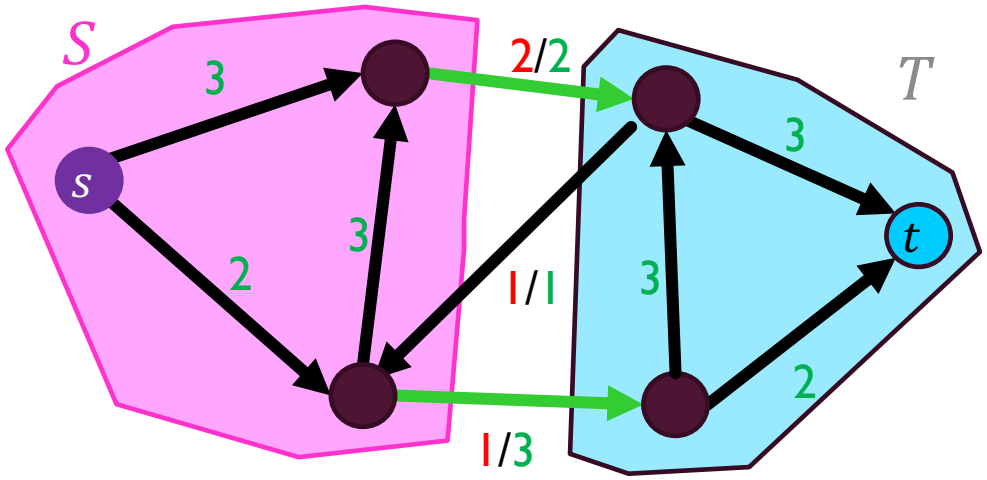
- A cut C = (S, T) where:
  - S is a set of vertices (S is a subset of V)
  - T is a set of vertices (T also a subset of V)
  - S intersect T = null set (no shared vertices)
  - S union T = V (all vertices in either S or T)
  - s in S, t in T



- What do we care about?
  - Well, we care about the edges that go across this cut.
  - Either from node in S to a node in T or vice versa.

TWO PROPERTIES OF A CUT

- Consider a **cut** that separates s and t
  - Let $s \in S, t \in T$, s.t. $V = S \cup T$



- **Capacity** of the cut $C = (S, T)$
  - Sum the **capacities** of all edges which go from S to T
  - This example: 2 + 3 = 5

- **Net Flow** of the cut $C = (S, T)$
  - Sum of the **flows** on its edges from S to T minus the sum of the **flows** on its edges from T to S
  - This example: 2 + 1 – 1 = 2

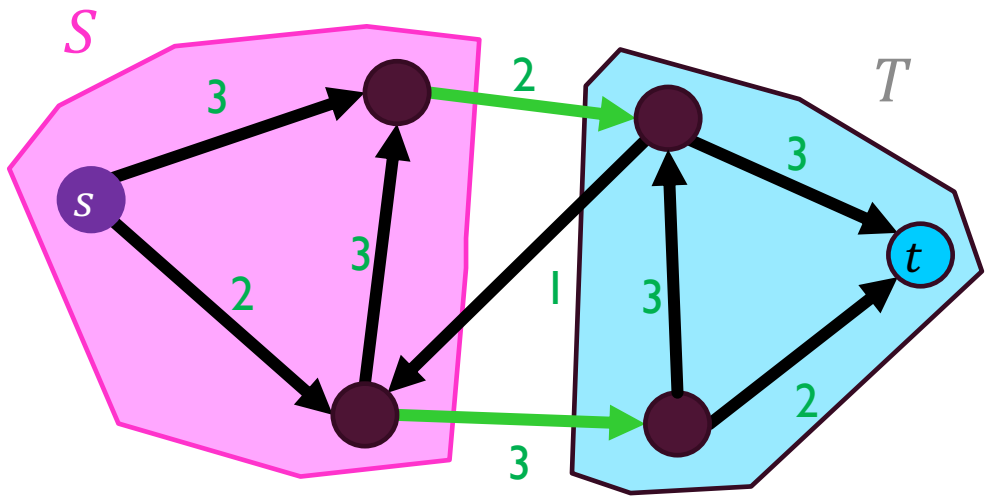
WE'LL SHOW THESE 3 THINGS

- Weak duality property
- Flow-value lemma
- Max-flow Min-Cut theorem

- …and how we can then argue Ford-Fulkerson is correct

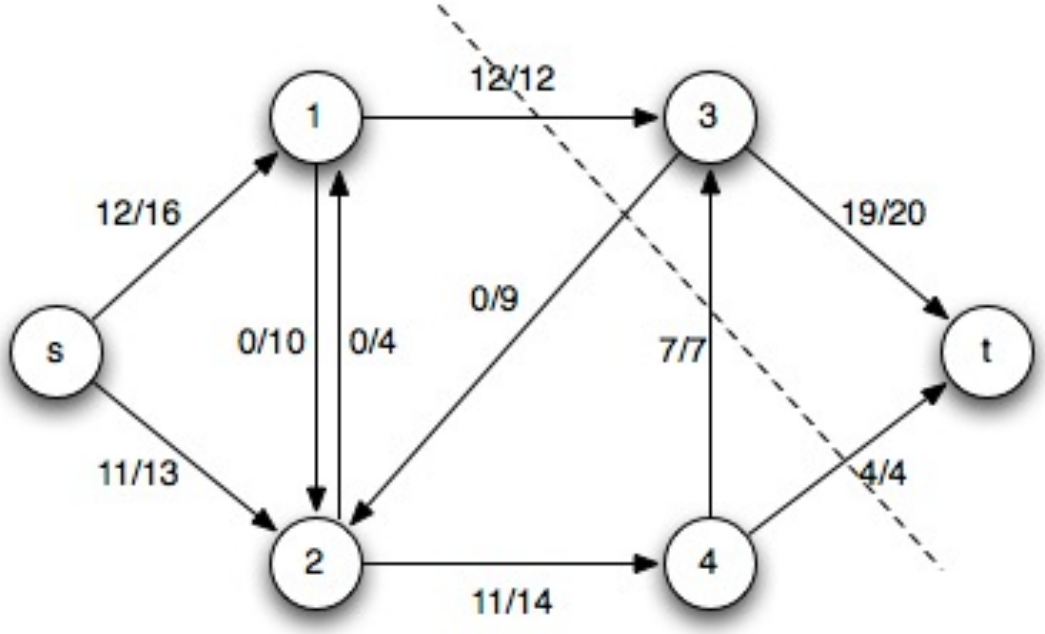
FIRST DEFINITION: WEAK DUALITY

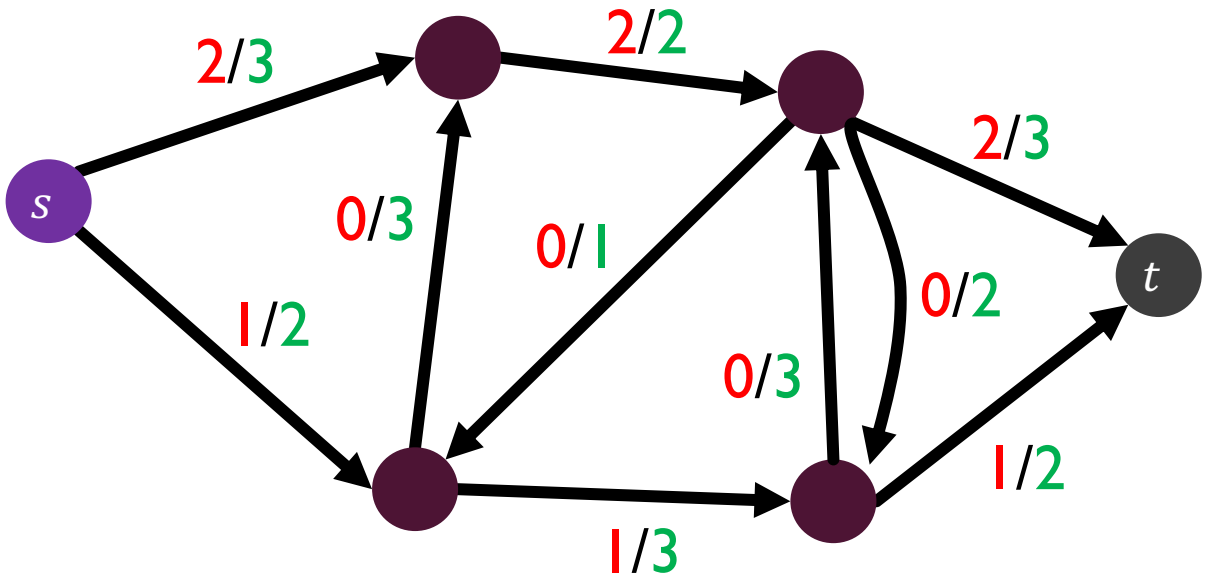
- Let ***f*** be any flow and C = (S, T) be any cut
- Then the value of ***f*** <= capacity of C
  - We'll refer to this as the weak duality property
- Note: We are talking about the **capacity** of C, not the value of the flow across C (i.e. not the net flow of C)



DEFINITION: FLOW-VALUE LEMMA

- Let ***f*** be any flow and C = (S, T) be any cut in G
  - The **net flow** across (S, T) equals the value of the flow ***f***





PROOF: FLOW-VALUE LEMMA

- Let ***f*** be any flow and C = (S, T) be any cut
  - The net flow across (S, T) equals the value of the flow ***f***

- Proof by induction on the size of T
  - Base C. T = {t} (T is only the sink)
    - Clearly this is true as the flow across the cut is everyone sinking into t, which is the definition of the flow f
  - I.H. Assume true for some cut C = (S, T)
  - I.S. Move one node from S to T
    - Choose a node p to move that has at least one edge to a node in T
    - We know the flow f never changes
    - How does the value of the new cut C' = (S', T') change?
    - It doesn't! Why?

PROOF: FLOW-VALUE LEMMA CONT.

- Let ***f*** be any flow and C = (S, T) be any cut
  - The net flow across (S, T) equals the value of the flow ***f***

- Why does C' = (S', T') have the same net flow?
  - Local equilibrium: net flow coming into node p from nodes in S only must equal the flow going out across the cut to nodes in T
  - After node p is moved:
    - everything going across the cut now goes to something (node) in T from T, everything going to or from a node in S now goes across the cut.
  - Thus, by local equilibrium the value of the cut C' is equivalent to the value of the cut C
  - Induction done!

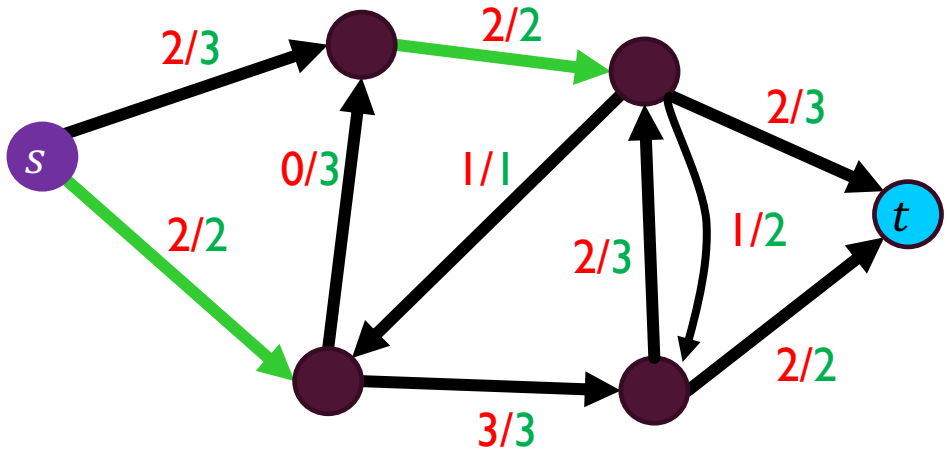
MAX-FLOW MIN-CUT THEOREM

- Reminder about *weak duality*:
  - Let ***f*** be any flow and C = (S, T) be any cut
  - Then the value of ***f*** <= capacity of C
- The max flow ***f*** <= capacity of any cut C
  - So the best ***f*** can be is the capacity of the smallest-capacity cut
  - Can we guarantee that a flow with that value exists? (Yes, in later slides.)

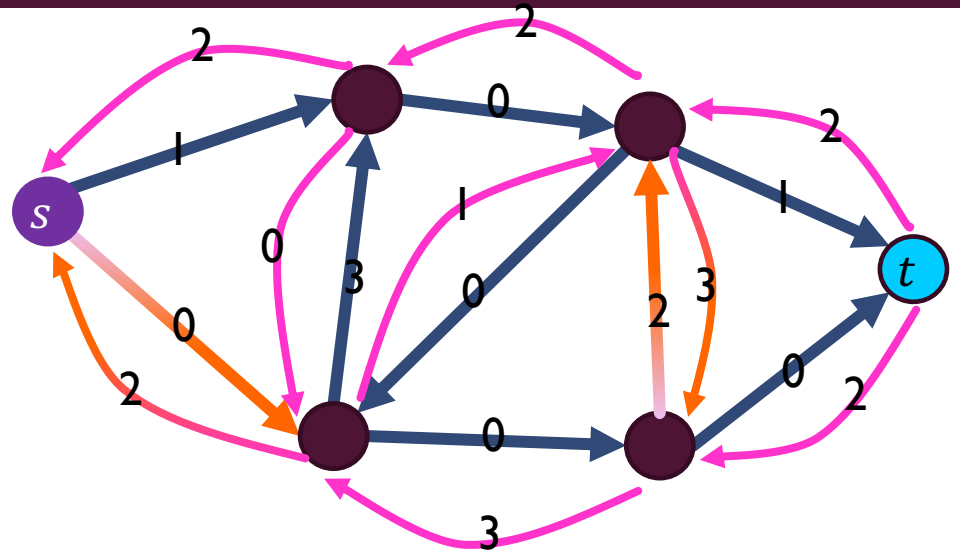
- **The Max-flow Min-cut theorem:** the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut

- In other words, if you look at all the possible cuts in the graph, and find the smallest capacity of those cuts, then that value is the value of the maximum flow for that network.

EXAMPLE: MAX-FLOW/MIN-CUT



Flow Graph G



Residual Graph G_f

$|f| = 4$

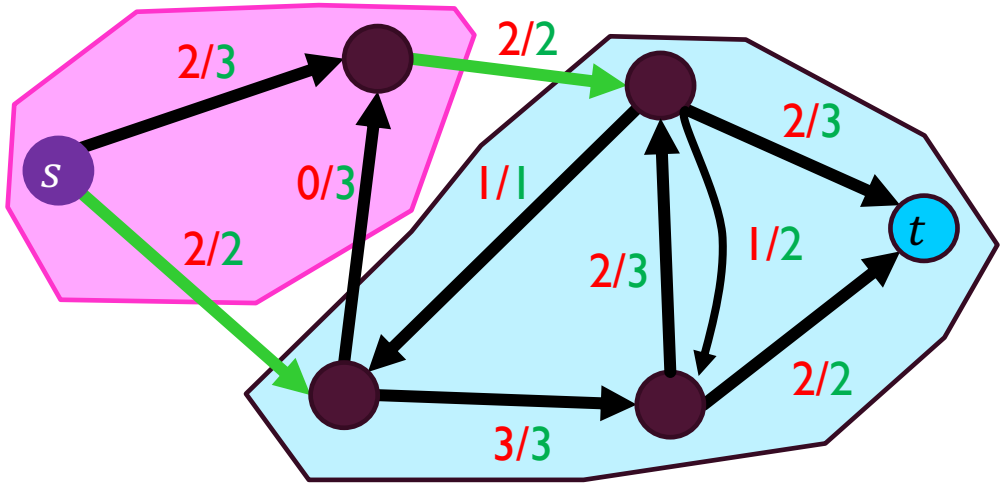
Min Capacity cut $= 4$

Right? Look at other cuts in G.

No Augmenting Paths

MAX-FLOW MIN-CUT AND FORD-FULKERSON

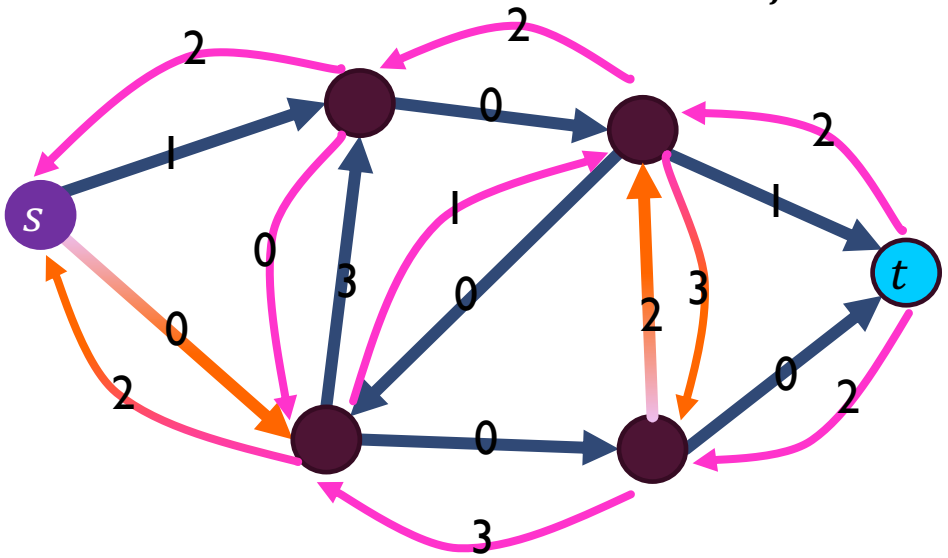
Flow Graph G



$|f| = 4$

Min Capacity cut $= 4$

Residual Graph G_f



No Augmenting Paths

Idea: When there are no more augmenting paths, there exists a cut in the graph with cost matching the flow (We're going to use the flow-value lemma here.)

SUMMARY

HOW DO WE REASON ABOUT ALGORITHMS?

- Mathematical Foundations (basics of complexity analysis)
- Recursion Tree/ Master's Theorem/ Substitution Methods

  - *Explain the Big O Notation (implications, interpretation of parameters…)*
  - *Master Theorem (a? b? f(n)? Why Regularity Condition is required? Why we need this? Meaning of each cases?)*

DIVIDE AND CONQUER

- Understanding Recursion in Algorithms
- When D&C can be used?
- Merge Sort, Quick Sort, Binary Search and other algorithms where D&C can be directly used.

GRAPH ALGORITHMS

- Special graphs and their usages – how to model real-world problems using Graph. How to represent graphs.
- Shortest Path
- MST
- Graph traversal – which one is used for other tasks like detecting cycles, finding connected components
- Topological sorting
- When Greedy can be used and when not? Advantages/ Disadvantages.

DP

- Two key properties: Optimal substructure and Overlapping subproblems. Good understanding of these properties and how to use those.
- DP examples covered in the class
- Techniques to solve and print answers.

NP, NPC….

- Definitions and understanding of each complexity class.
- Reductions covered in class.

HASHING + NETWORK FLOW

- Different types of Hashing
- Collision resolution techniques
- What is Flow network? And all properties and algorithms
- How flow network can be used.

QS PATTERN

- Concepts covered in class (50) + How well you can use that on a given problem (30)

- Brief and concise answer!

THANK YOU!