

Algorithm Design and Analysis

Network Flows Part I: Flows, Cuts, and Matchings

Roadmap for today

- Define the *maximum network flow* problem
- Learn the *Ford-Fulkerson* algorithm for maximum flow
- Define the related *minimum cut* problem
- See and prove the *min-cut max-flow theorem*
- Study an application of network flow (*bipartite matchings*)

Definition: Network flow

Motivation (Flow network): Consider a network of pipes, each able to handle a certain number of liters of water per minute. How much water can you send from s to t ?

Definition (Flow network): A directed graph with

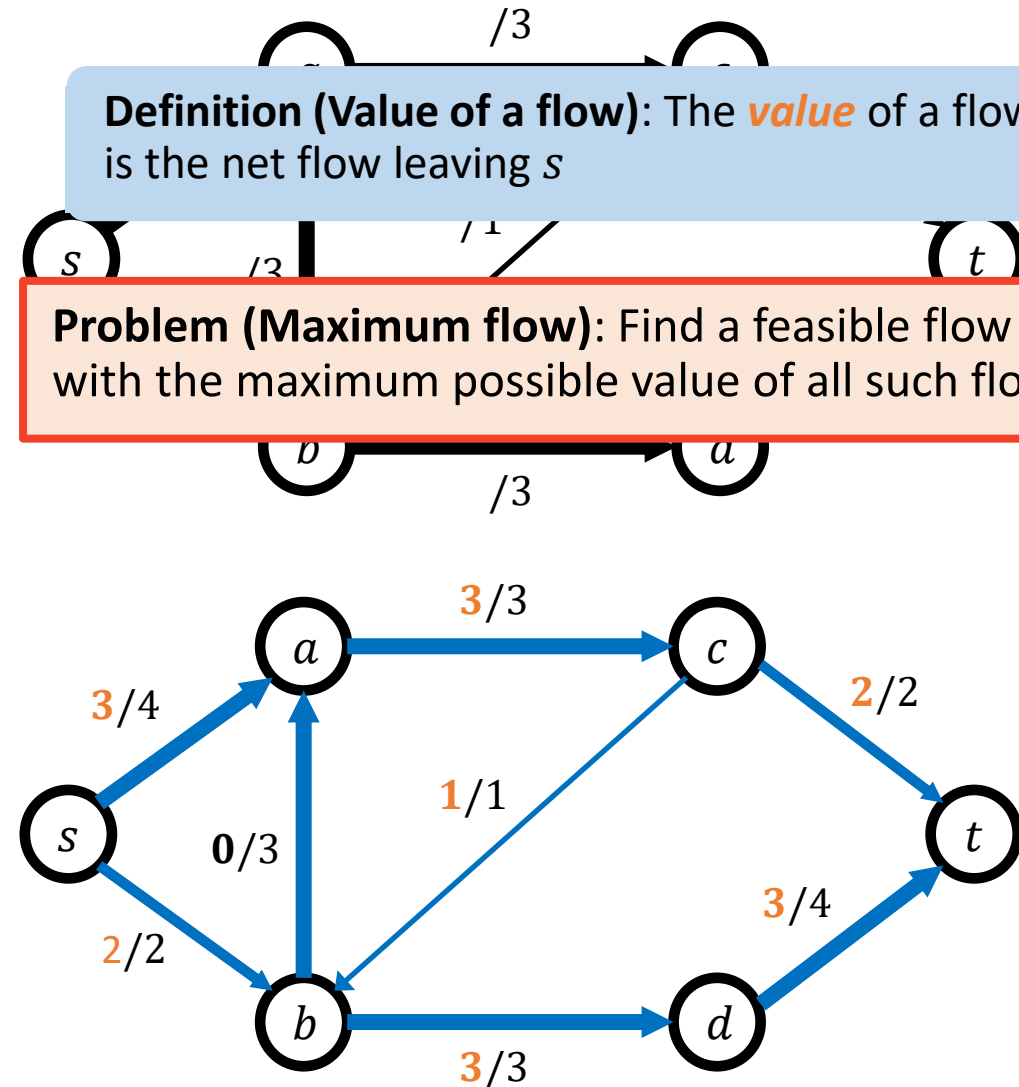
- Edge **capacities** $c(u, v)$
- A **source** vertex s
- A **sink** vertex t

Definition (A flow): A quantity of flow on each edge, $f: E \rightarrow \mathbb{R}$, called **feasible** if:

- **Conservation:** Flow in = Flow out $\forall v \notin \{s, t\}$
- **Capacity:** $0 \leq f(u, v) \leq c(u, v)$

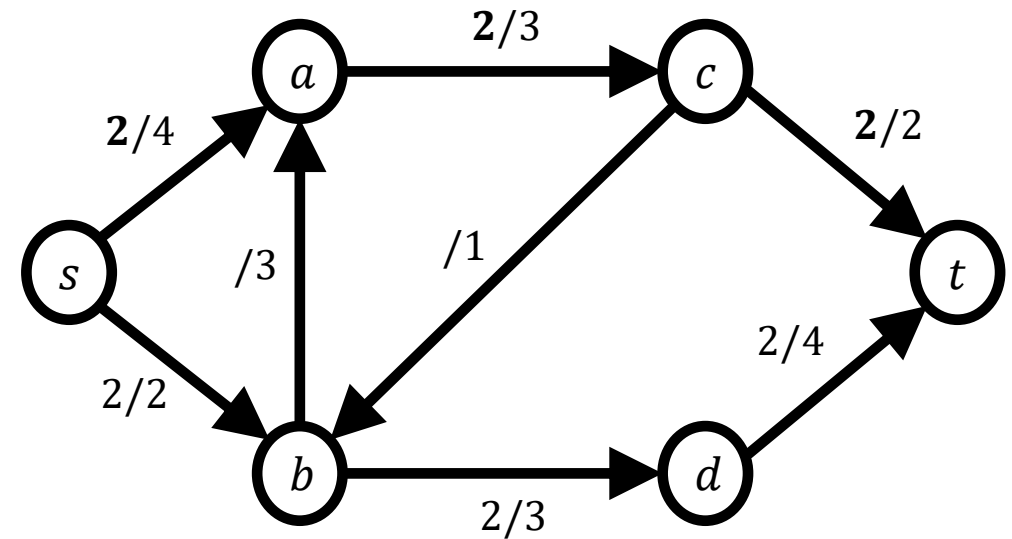
Definition (Value of a flow): The **value** of a flow is the net flow leaving s

Problem (Maximum flow): Find a feasible flow with the maximum possible value of all such flows.



Improving a flow: s - t paths

- Is the flow on the right optimal?



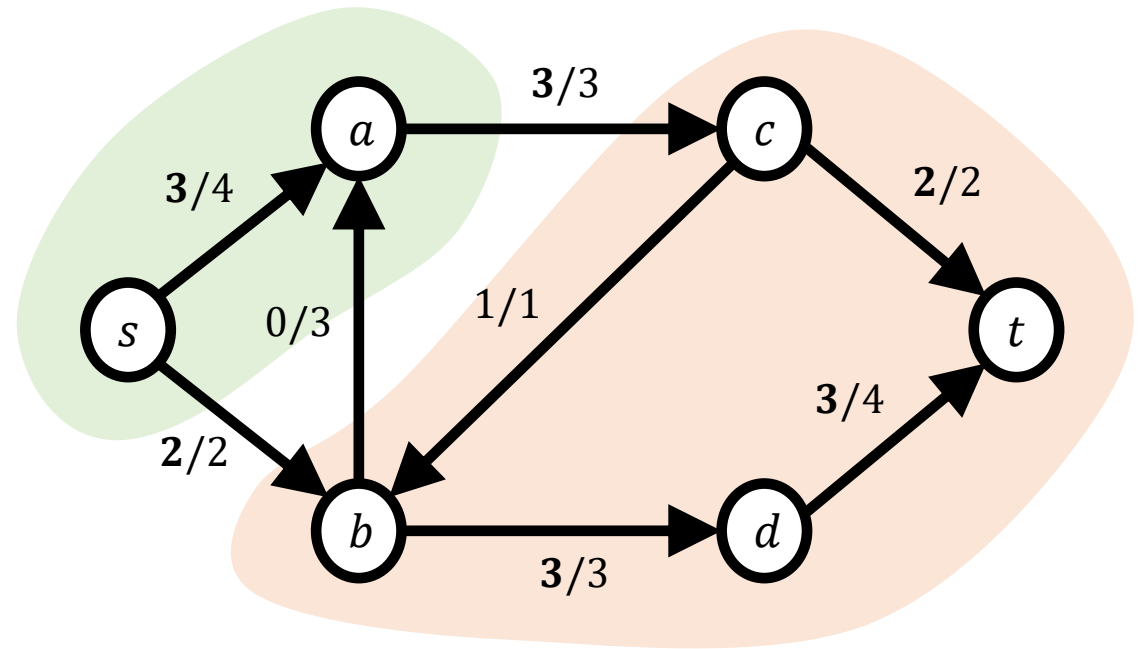
Certifying optimality: s - t cuts

- Is the flow on the right optimal?

Definition (s - t Cut): An **s - t cut** is a partition of the vertices into two disjoint sets (S, T) such that $s \in S$ and $t \in T$

Definition (Capacity): The **capacity** of an s - t cut (S, T) is the total capacity on edges (u, v) where $u \in S$ and $v \in T$:

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



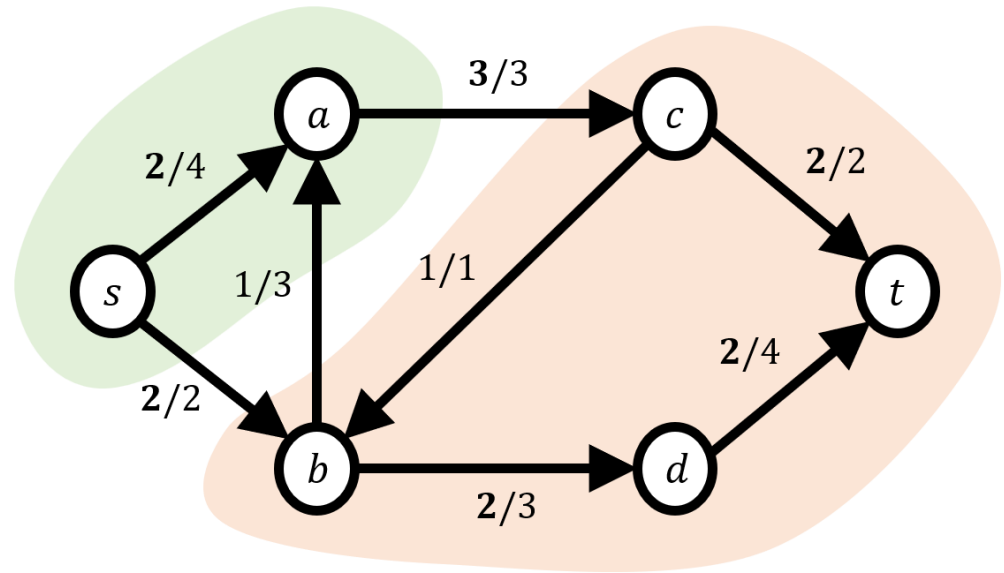
Net flow across a cut

Definition (Net flow): The *net flow* across an s - t cut (S, T) is the amount of flow moving from S to T :

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

Observe: The value of a flow (which we defined as the net flow out of s) is the net flow across the cut $(\{s\}, V \setminus \{s\})$

Theorem: For any s - t cut (S, T) , the net flow across the cut equals the value of the flow!



Proof: Algebra using the definitions.

Net flow theorem

Theorem: For any s - t cut (S, T) :

$$f(S, T) \leq \text{cap}(S, T)$$

Proof:

Corollary: $\text{max-flow} \leq \text{min-cut}$

Proof:

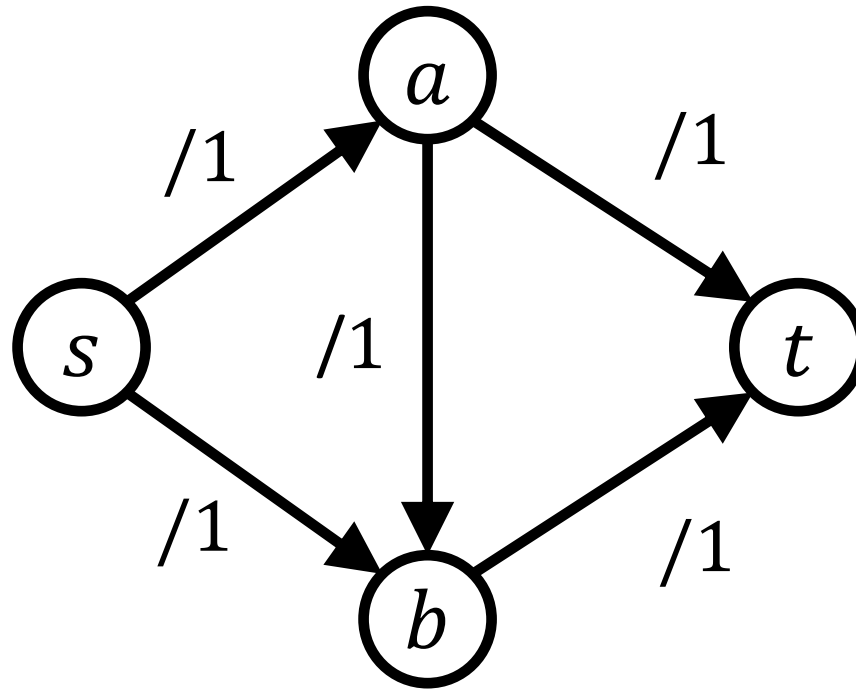
Definition (Capacity): The **capacity** of an s - t cut (S, T) is the total capacity on edges (u, v) where $u \in S$ and $v \in T$:

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

Definition (Net flow): The **net flow** across an s - t cut (S, T) is the amount of flow moving from S to T :

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

Is greedy optimal?

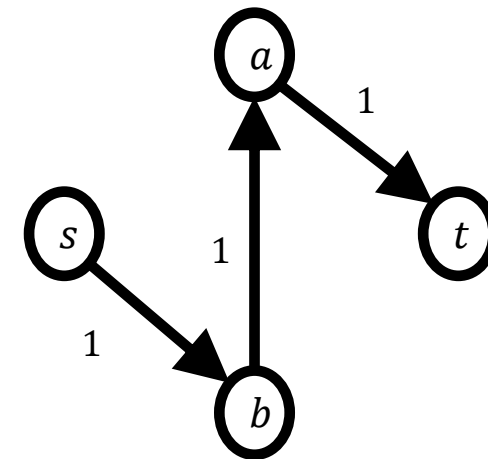
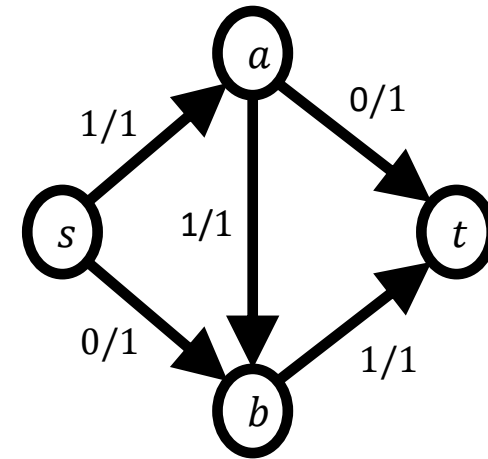


The residual graph

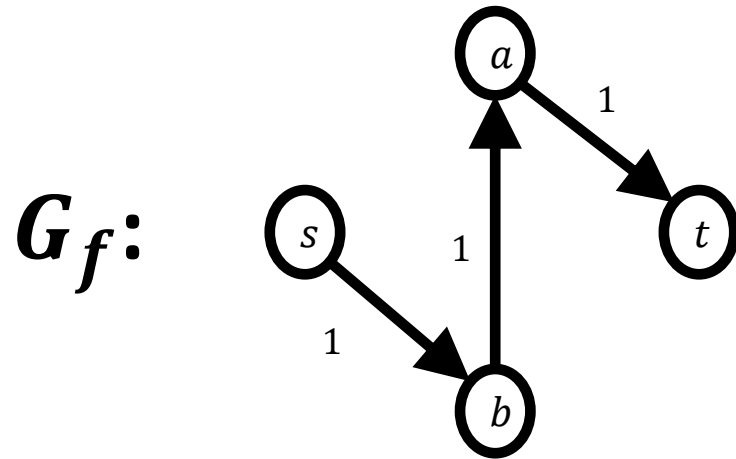
Definition (residual capacity): An edge (u, v) with capacity $c(u, v)$ and current flow $f(u, v)$ has **residual capacity**

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v), & (u, v) \in E \\ f(v, u), & (v, u) \in E \end{cases}$$

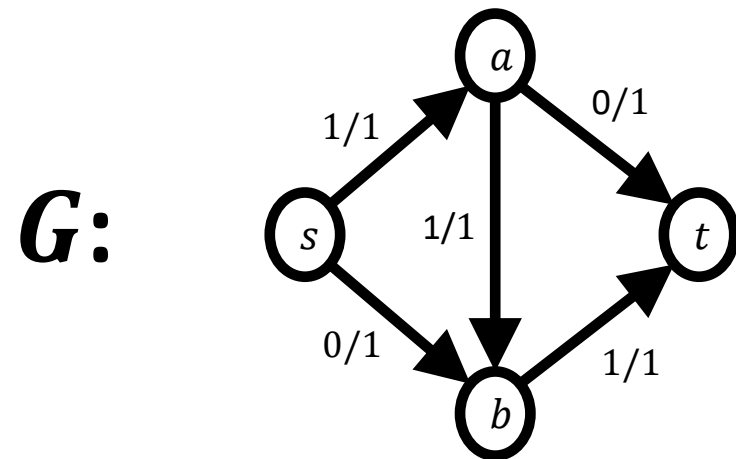
Definition (residual network): Given a flow network G and a current flow f , the **residual network** G_f is a flow network whose capacities are the residual capacities $c_f(u, v)$



Augmenting paths



Definition (augmenting path): An *augmenting path* is a path from s to t of non-zero capacity in the residual network.



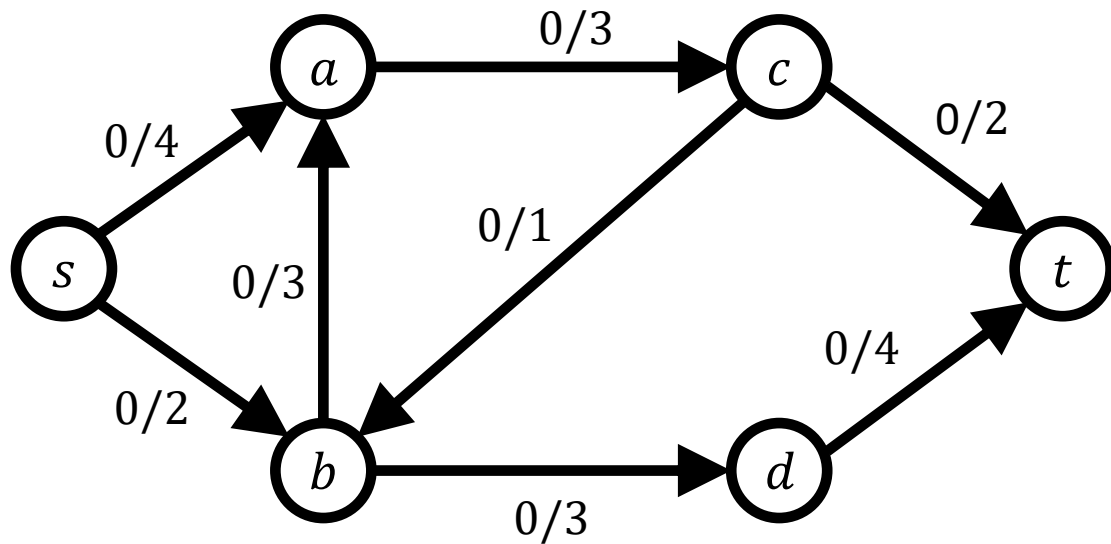
Key idea (reverse edges): Augmenting along a *reverse edge* removes that amount of flow from the edge.

The Ford-Fulkerson Algorithm

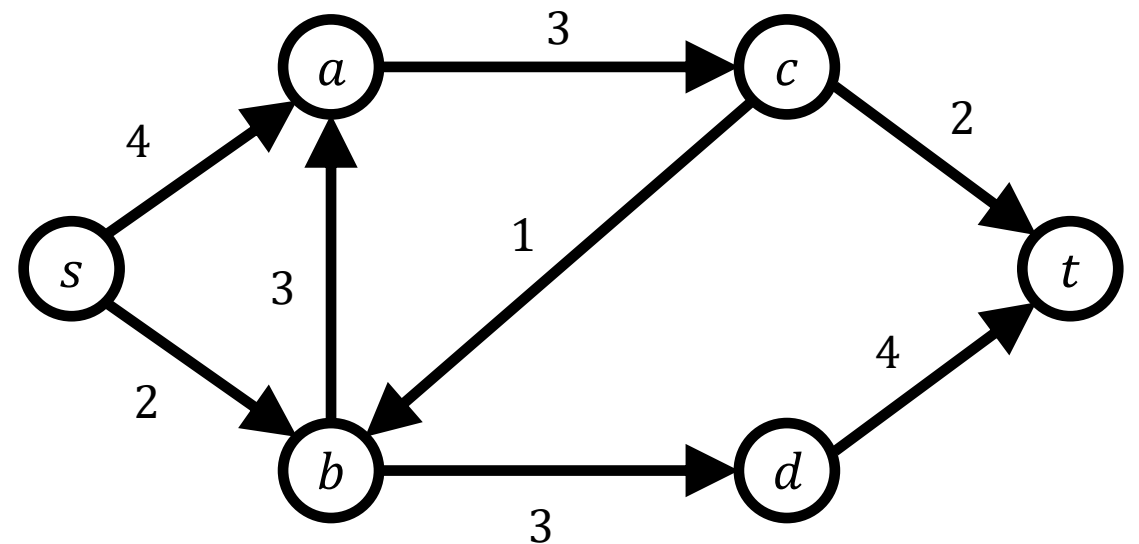
Algorithm (Ford-Fulkerson):

Example

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E, \\ f(v, u) & \text{if } (v, u) \in E. \end{cases}$$



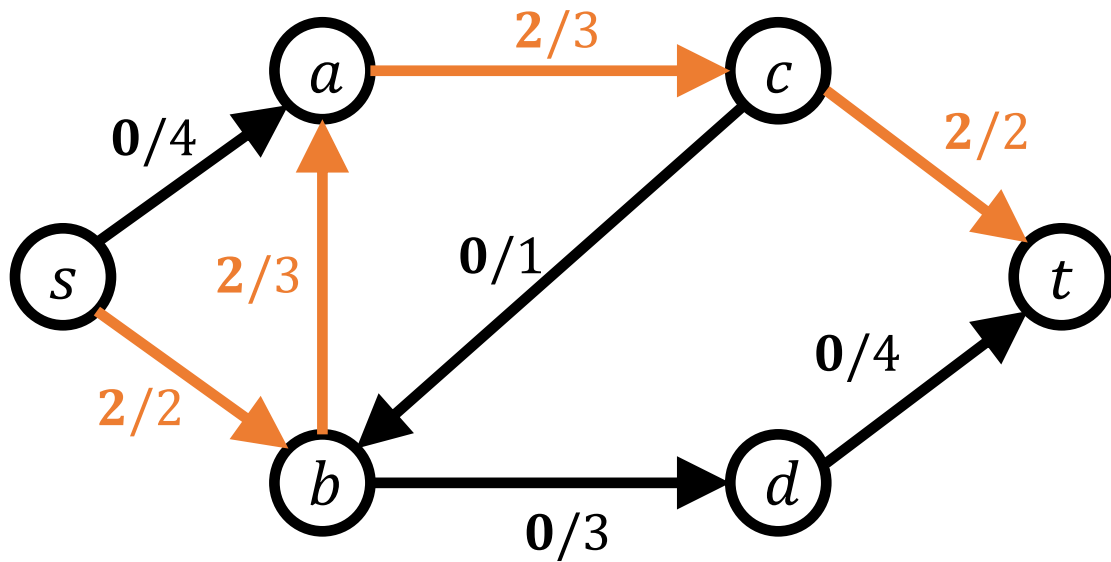
Flow network G



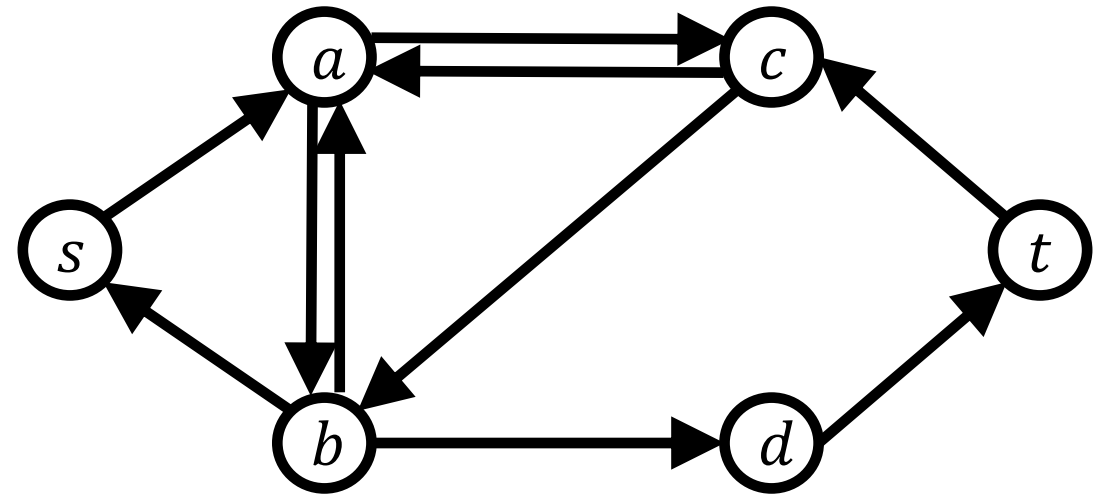
Residual network G_f

Example

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E, \\ f(v, u) & \text{if } (v, u) \in E. \end{cases}$$



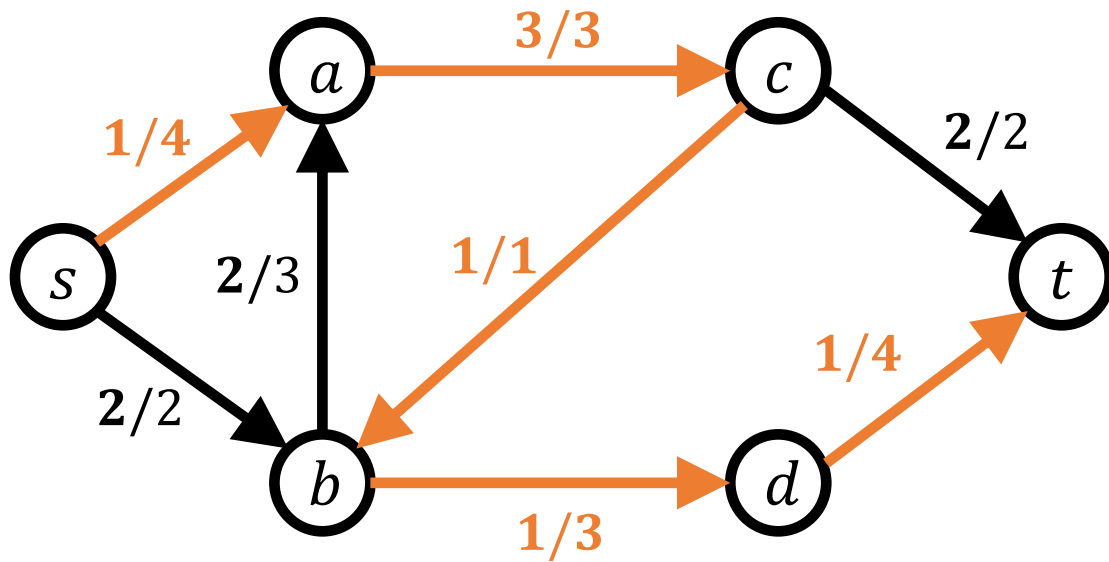
Flow network G



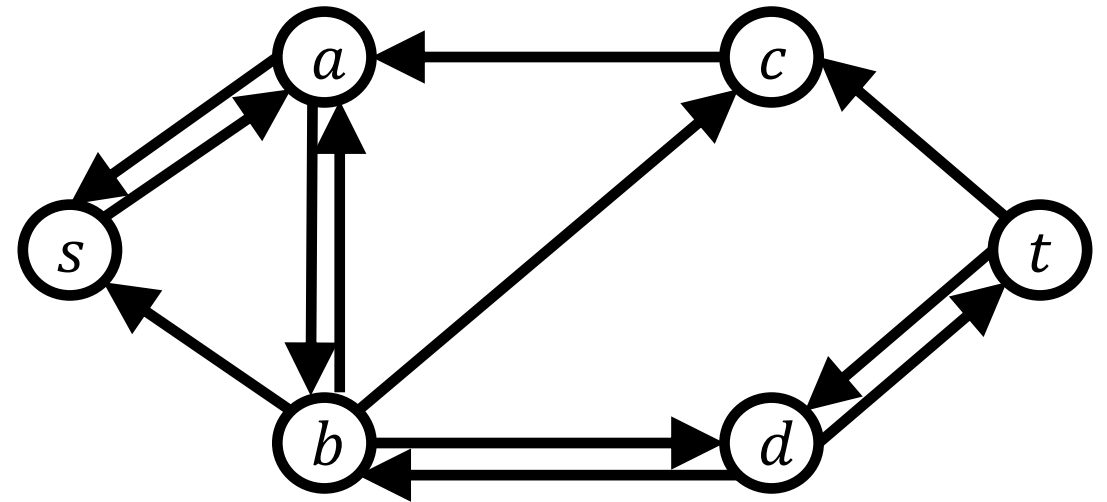
Residual network G_f

Example

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E, \\ f(v, u) & \text{if } (v, u) \in E. \end{cases}$$



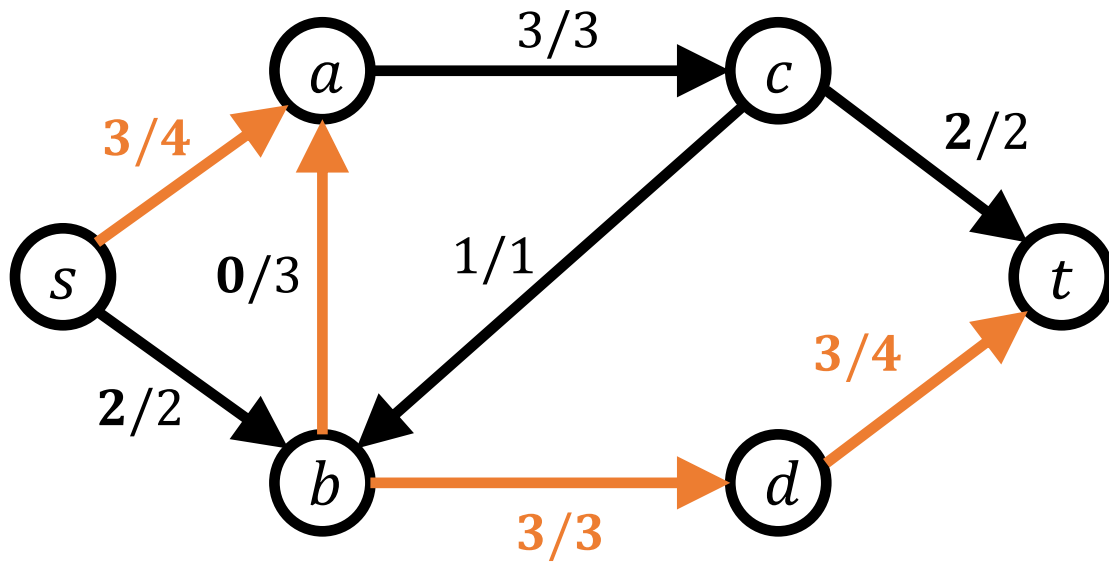
Flow network G



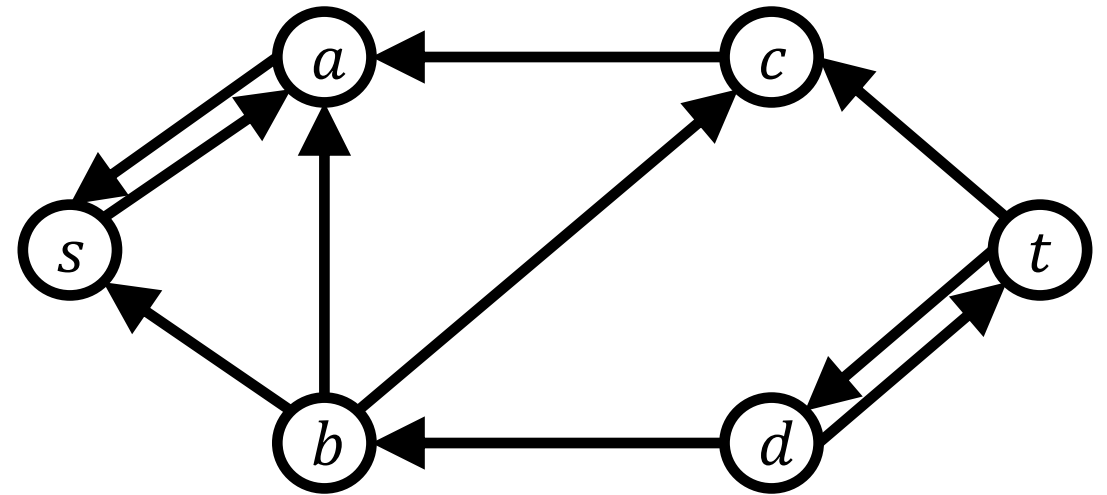
Residual network G_f

Example

$$c_f(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{if } (u, v) \in E, \\ f(v, u) & \text{if } (v, u) \in E. \end{cases}$$



Flow network G



Residual network G_f

Analysis

Analysis

Theorem (runtime): *Assuming integer capacities*, Ford-Fulkerson runs in $O(mF)$ time, where F is the value of the maximum s - t flow

Proof:

Analysis

Theorem (maximality): Ford-Fulkerson finds a flow whose value is equal to the capacity of the minimum cut.

Proof:

Analysis

Corollary (*Min-cut Max-flow theorem*): For any flow network, the value of the maximum s - t flow is equal to the capacity of the minimum s - t cut

Proof:

Analysis

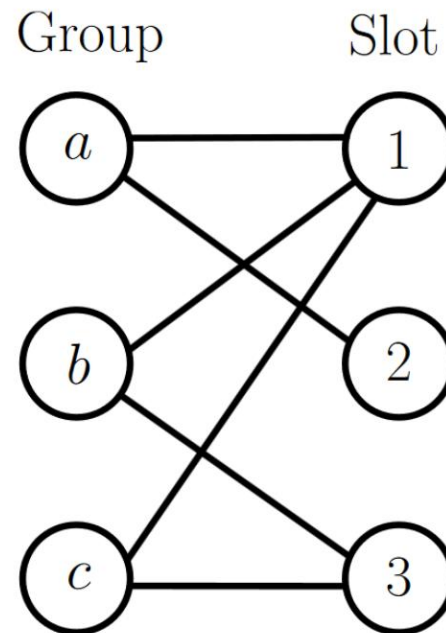
Theorem (Integral flows): For any flow network with integer capacities, there exists a maximum flow in which the flow on every edge is an integer

Proof:

Applications

Bipartite Matching

Problem (Bipartite matching): Given a bipartite graph G , find a largest possible set of edges with no endpoints in common.



Analysis of matching

Important (flow model proofs): When modeling problems with flow, you need to prove that the reduction is correct! This usually consists of a bidirectional proof.

*Claim #1 Given a matching M in the original graph, there exists a flow f in our flow network of value $|M|$ ($\Rightarrow$ **max flow** $\geq$ **max-matching**)*

Analysis of matching

Important (flow model proofs): When modeling problems with flow, you need to prove that the reduction is correct! This usually consists of a bidirectional proof.

*Claim #2: Given a flow f in our flow network, there exists a matching M of size $|f|$ in the original graph ($\Rightarrow$ **$\text{max-flow} \leq \text{max-matching}$**)*

Summary of flow fundamentals

- Lots of definitions! Take time to review them.
 - Flow networks, flows, s - t cuts, residual networks, augmenting paths
- The Ford-Fulkerson algorithm
 - Finds a maximum flow (in an integer-capacity network) in $O(mF)$ time
- Modeling with flows
 - Bipartite matching
 - Remember that the proofs must show both directions of correspondence!