Lecture 4: Network Flows

- Review of Max Flow/Min Cut and Ford Fulkerson
- Dinitz-Edmonds-Karp Algorithm
- Application to Bipartite Matching

Resources

- CLRS, *Introduction to Algorithms*
- Erikson, *Algorithms*
- CMU 15-451, Introduction to Algorithms, *Network Flows 1 and 2*

PRINCETON COMPUTER SCIENCE

Network Flow

Motivation (Flow network): Consider a network of pipes, each able to handle a certain number of liters of water per minute.

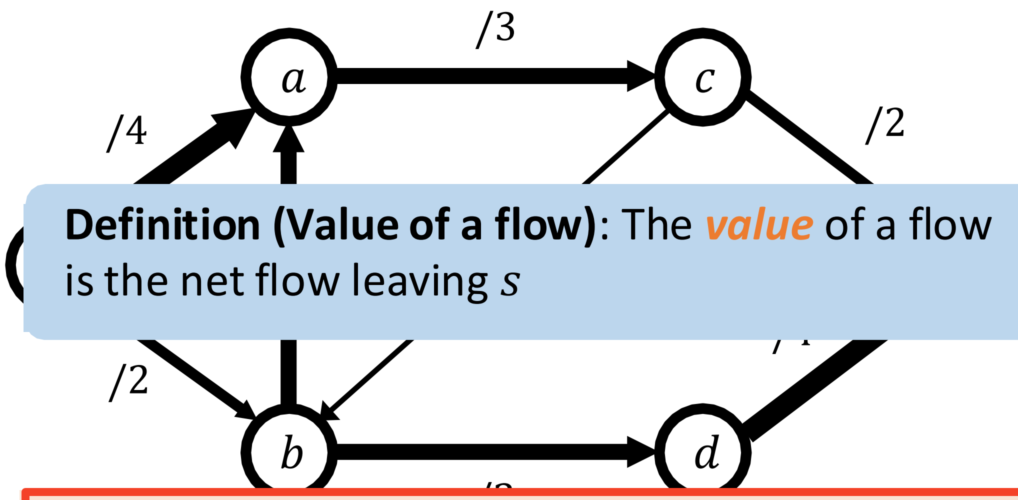
How much water can you send from s to t?

Definition (Flow network): A directed graph with

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

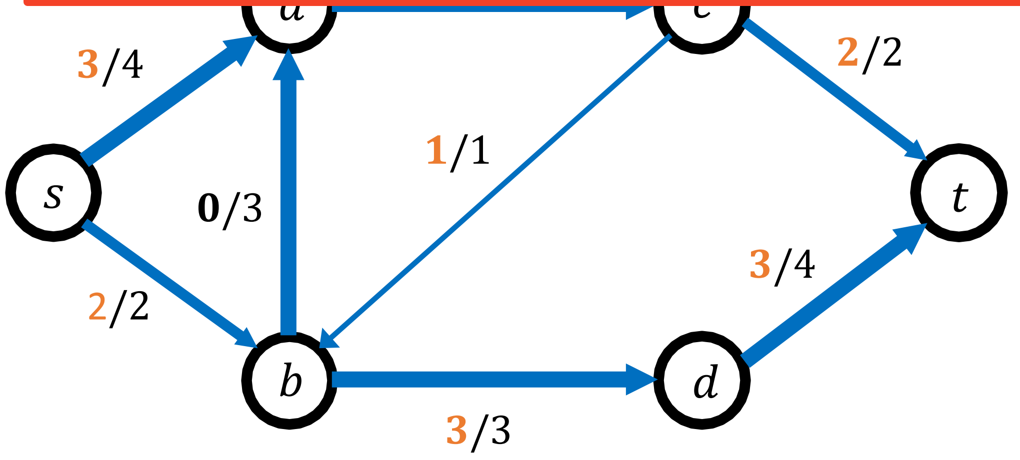
Definition (A flow): A quantity of flow on each edge, $f: E \to \mathbb{R}$, called ***feasible*** if:

- **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

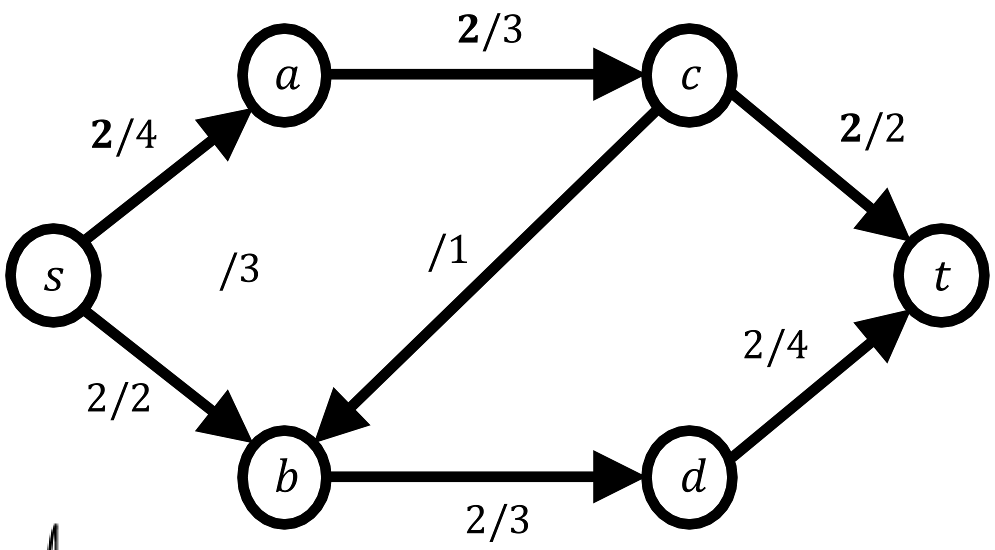
- Is the flow on the right optimal?

Consider the path
$s \to a \to c \to b \to d \to t$

Can bump up flow along the path by +1.

Flow conservation always satisfied.

Each edge on the path has residual capacity ≥ 1.



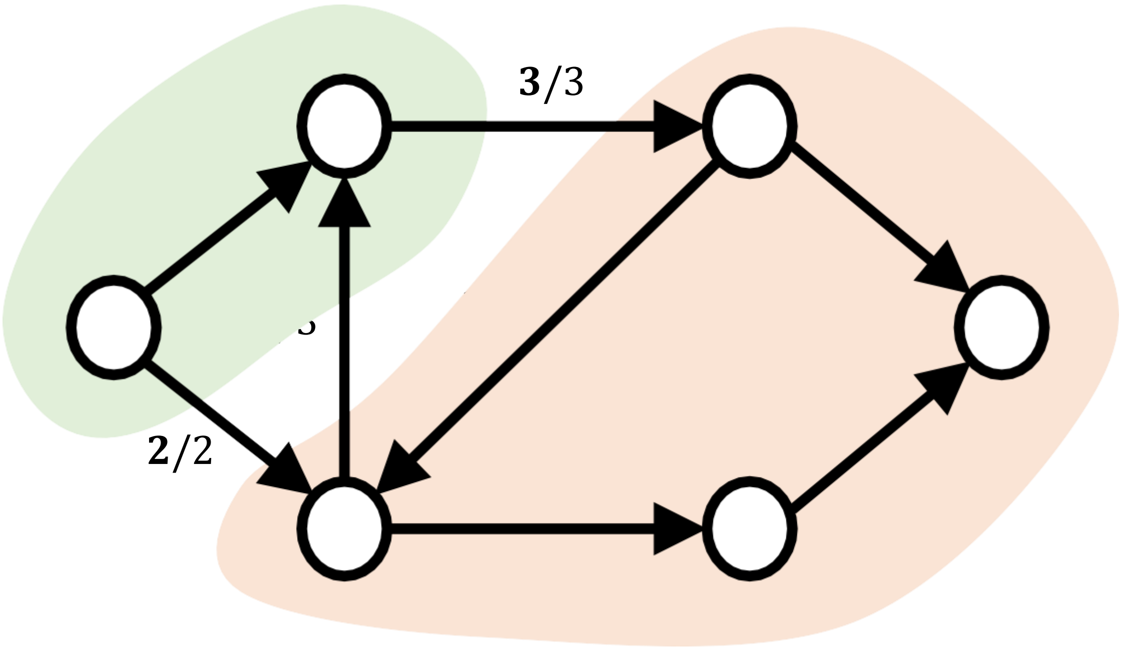
Certifying Optimality: s-t cuts

- 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 $v \in T$:

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



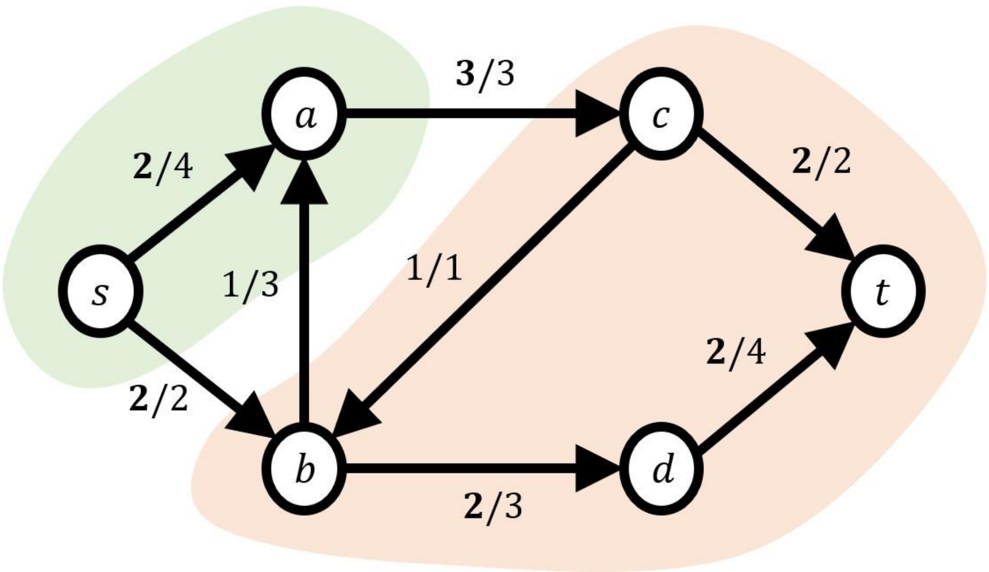
Net Flow Across a Cut

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:

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

Corollary: max-flow $\leq$ min-cut

Proof:

any flow $\leq$ max flow $\leq$ mincut $\leq$ any cut

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

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

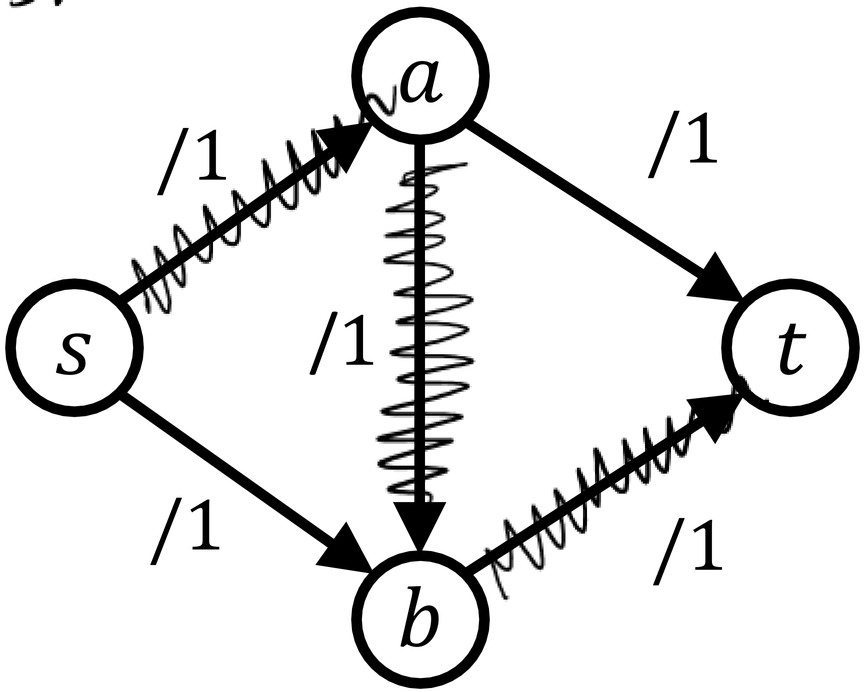
Definition (Net flow): The ***net flow*** across an s-t cut (S, T) is the amount of flow moving from S to T:

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

How does greedy do?

If flow along jagged edges, then there's no path with capacity.
Yet this flow isn't optimal.

Sending a flow along (a,b) is a mistake



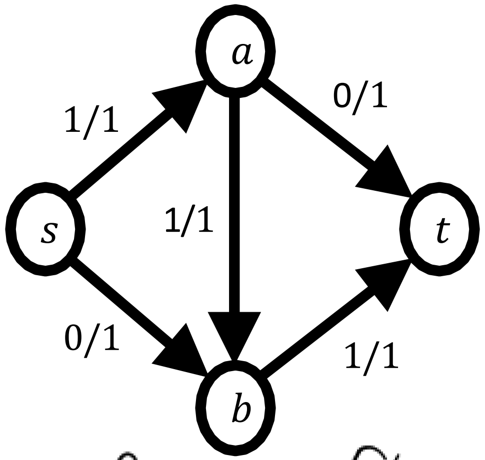
Obs^n: If we do greedy, need a way to undo our mistakes.

The residual graph

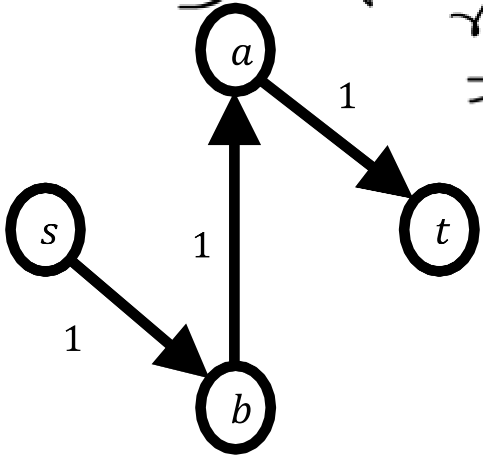
Definition (residual capacity): An edge (u, v) with capacity $c(u, v)$ and current flow $f(u, v)$ has ***residual capacity***

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

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)

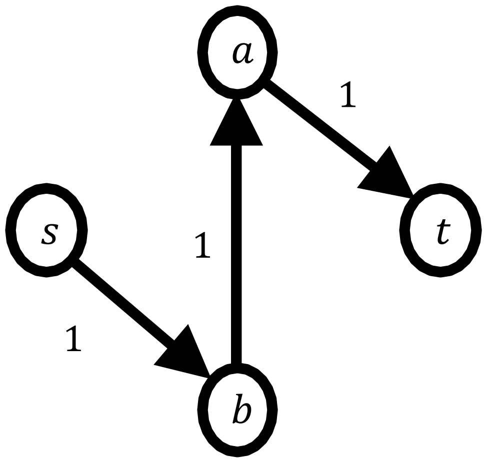


sending a flow in reverse = undoing sending flow forward.

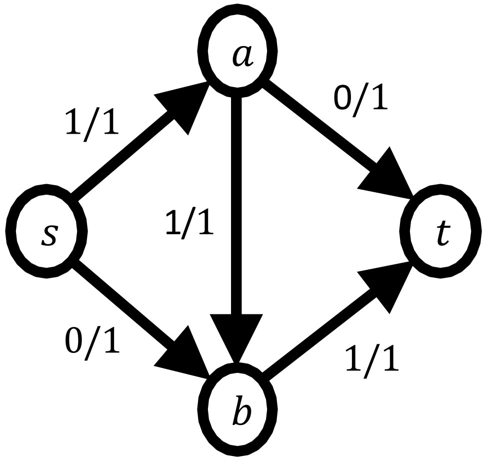


Augmenting Paths

G_f:



G:



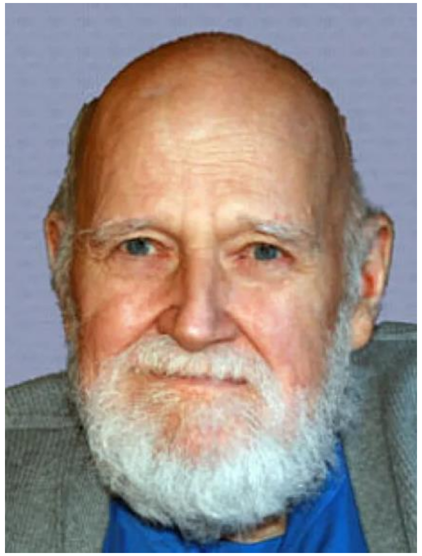
Definition (augmenting path): An *augmenting path* is a path from s to t of non-zero capacity in the residual network.

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

The Ford-Fulkerson Algorithm

Algorithm (Ford-Fulkerson):

While
∃ an augmenting path
(find using DFS or BFS)
add +1 flow to it.

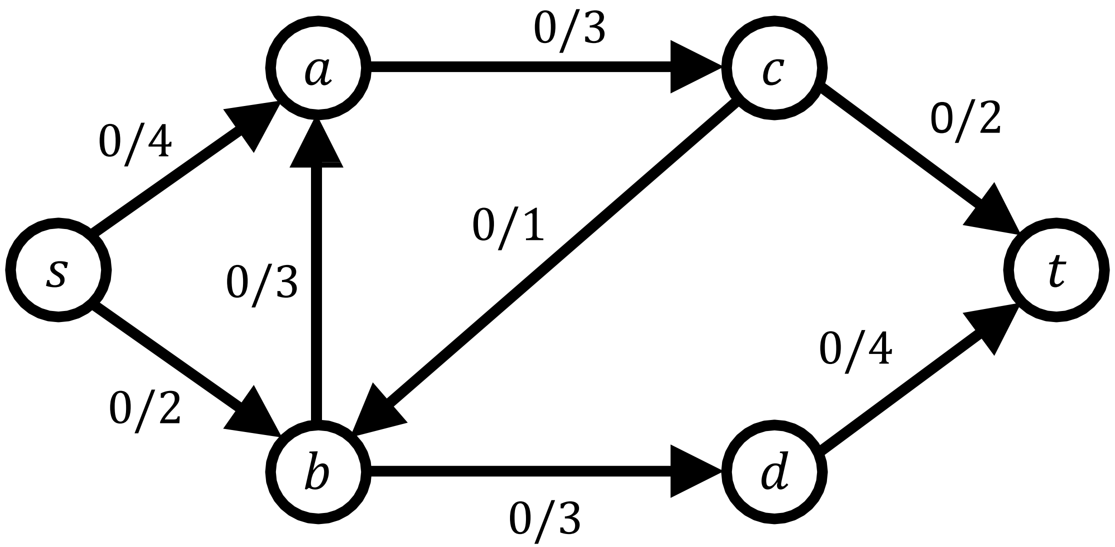


Les Ford

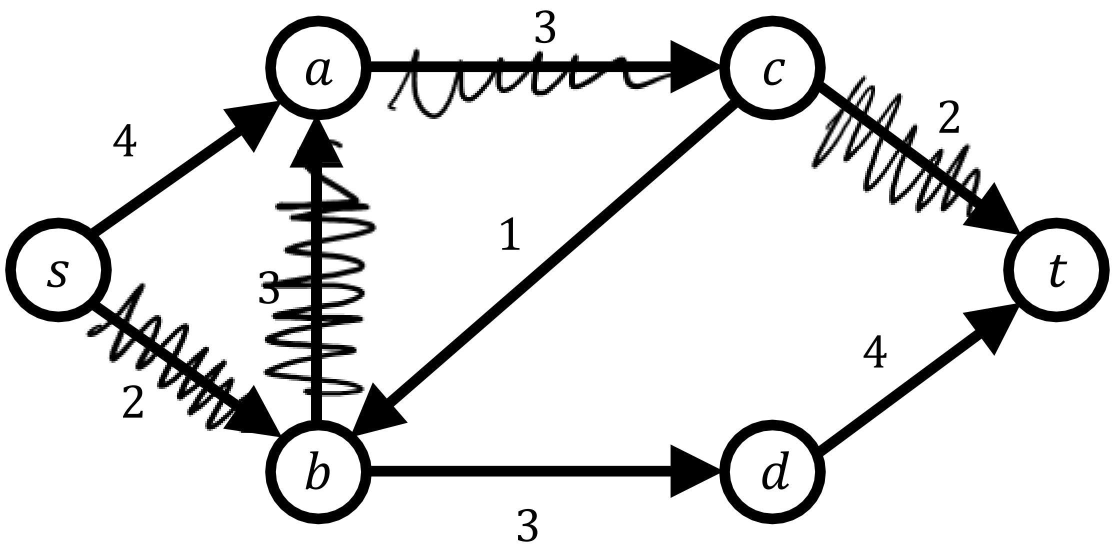
Delbert Fulkerson

Example

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



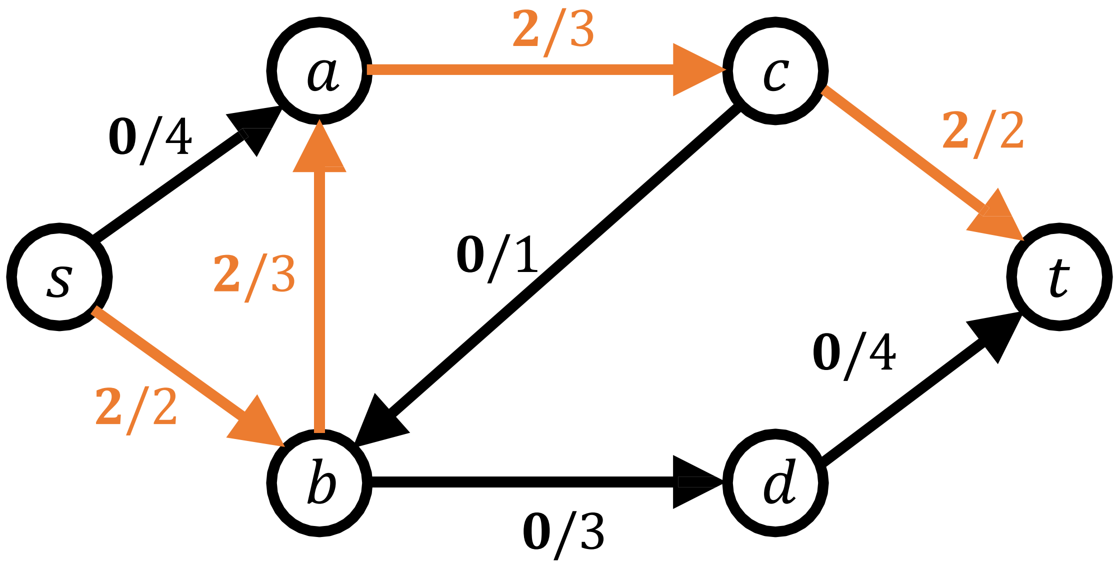
Flow network ***G***



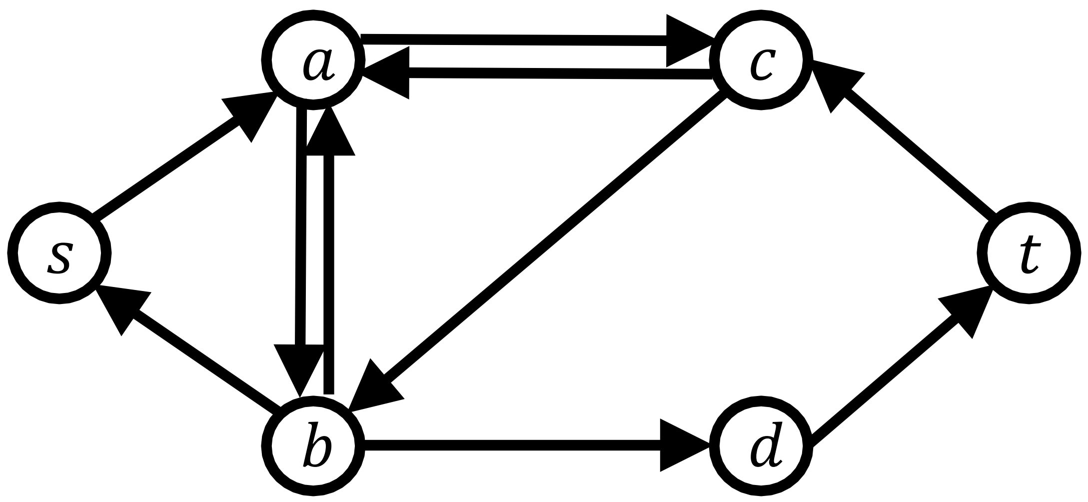
Residual network ***G**$_f$*

Example

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



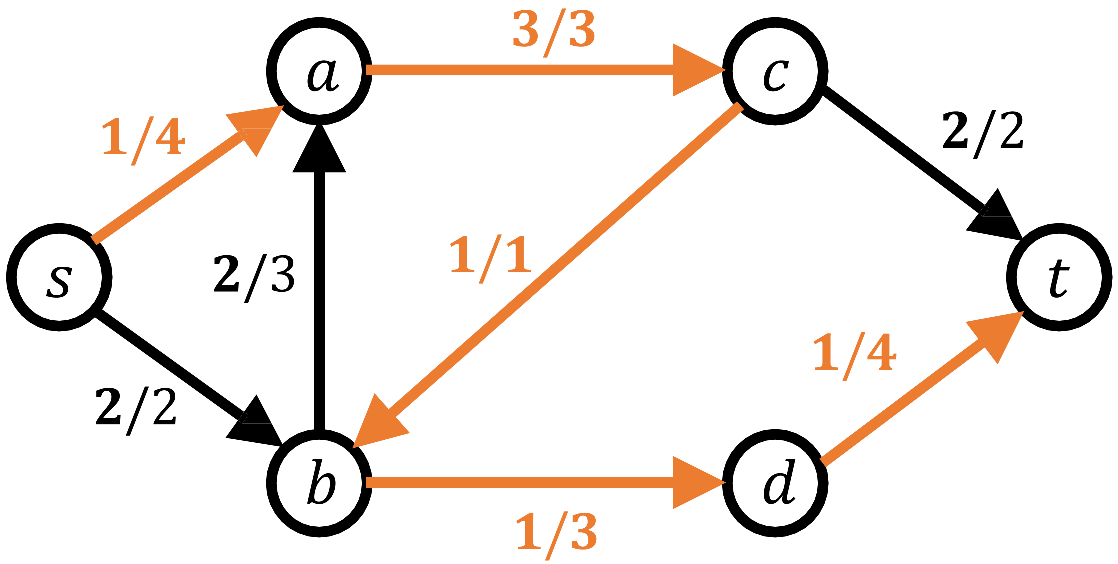
Flow network G



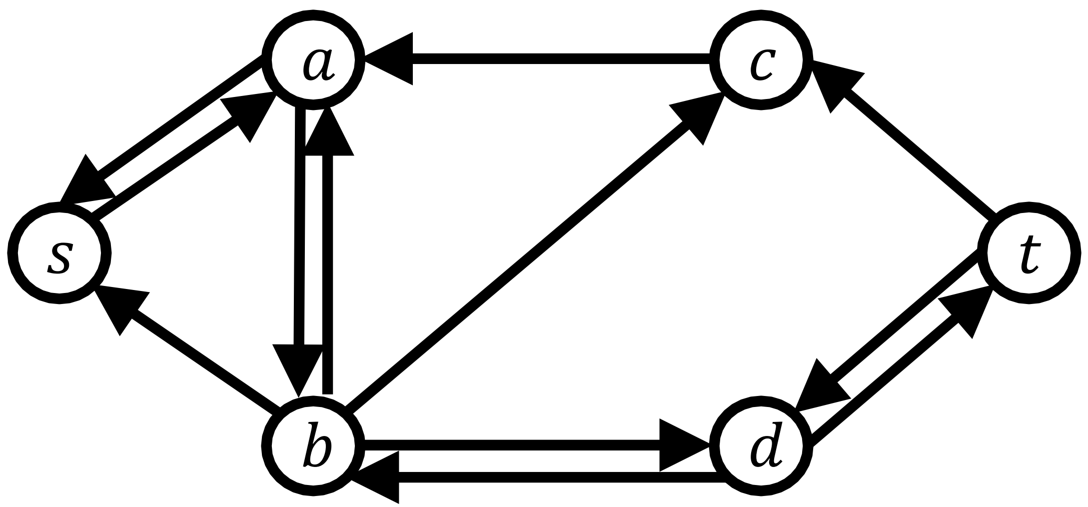
Residual network G_f

Example

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



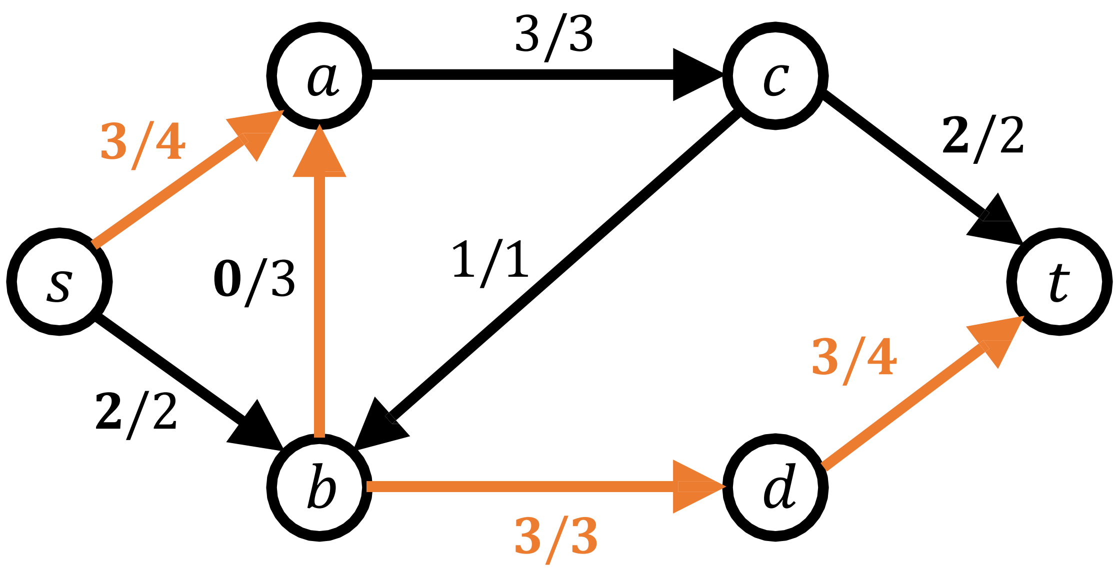
Flow network G



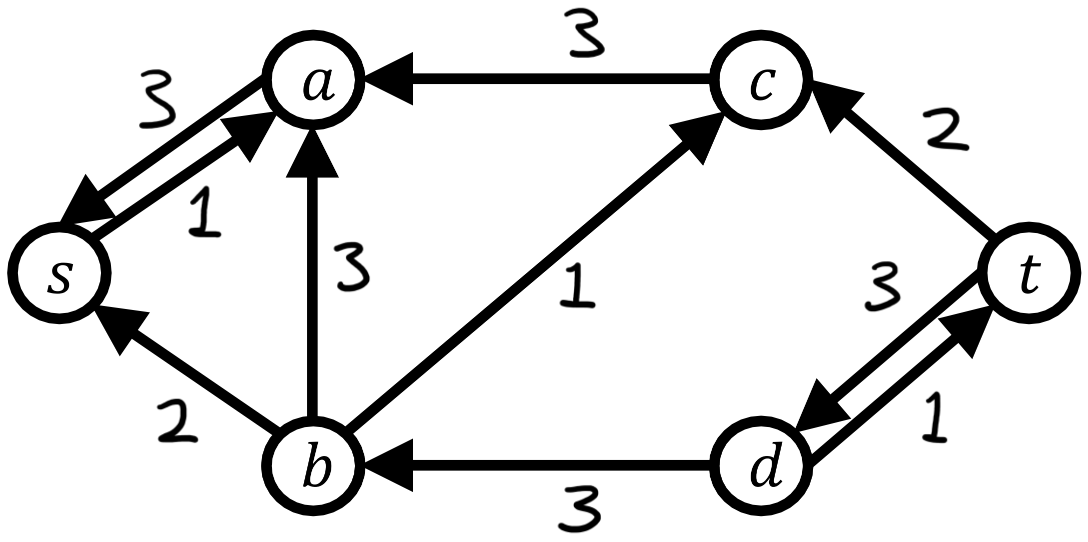
Residual network G_f

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

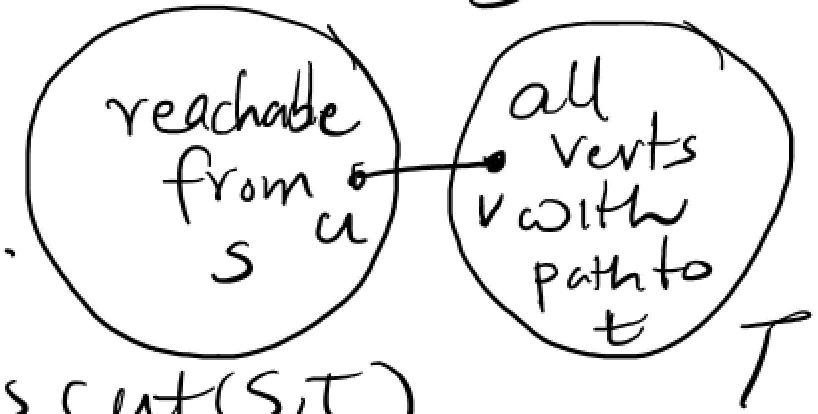
Theorem (maximality): ***If all capacities are integers,*** Ford-Fulkerson finds a flow whose value is equal to the capacity of the minimum cut.

Proof: Ford Fulkerson stops $\Leftrightarrow$ residual network has no $s \to t$ path

$S = \{u \mid s \rightsquigarrow u\}$, $T = \{v \mid v \rightsquigarrow t\}$

Let (u,v) be any edge, u S, $t \in V$ in the original network.

Since (u,v) has 0 capacity in the residual network, $(u,v) = c(u,v)$.



So: value of flow = Net flow across cut(S,T)

$$= \sum_{(u,v)\in cut(S,T)} f(u,v) = \sum_{(u,v)\in cut(S,T)} c(u,v) = Cap(S,T).$$

Analysis

Theorem (runtime): ***If all capacities are integers,*** Ford-Fulkerson runs in $O(mF)$ time, where F is the value of the maximum s-t flow.

Proof: +1 flow each iteration

$\Rightarrow \leq F$ iterations

Each iteration takes $O(n+m) = O(m)$ time if G is connected (WLOG).

Analysis: Max-Flow = Min-Cut

Corollary (*Min-cut Max-flow theorem*): If the capacities are all integers, for any flow network, the value of the maximum s-t flow is equal to the capacity of the minimum s-t cut

Proof:

We already proved: val(flow) = cap(S,T)
for the bottleneck cut.

We also know: val(flow) ≤ min-cut

So (S,T) must be a min-cut!

Analysis

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

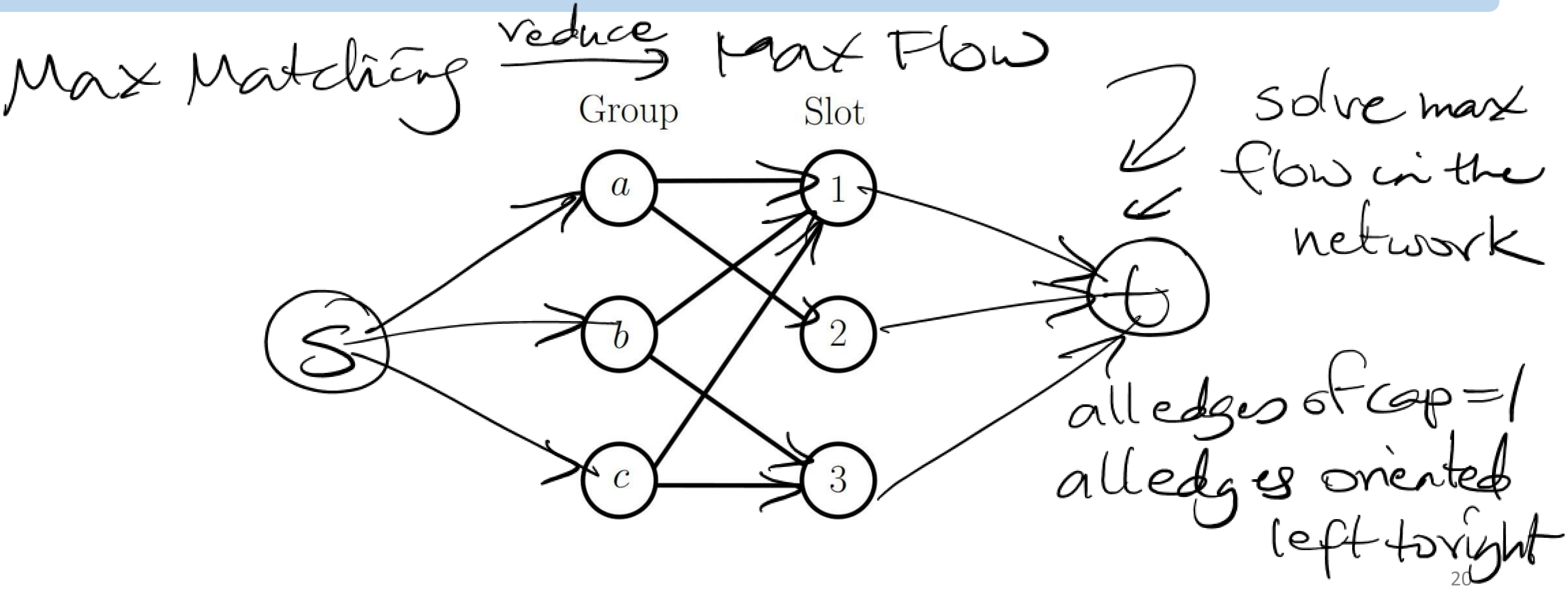
Proof:

A priori, there was no reason to expect integer flows. But Ford-Fulkerson shows that there's always an optimal flow that is integral.

Applications

Bipartite Matching

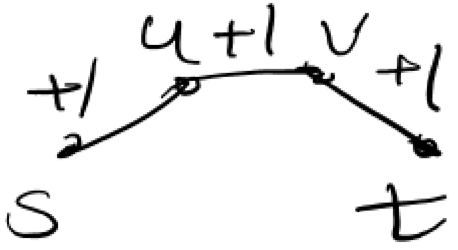
Problem (Bipartite matching): Given a bipartite graph G, find a largest possible set of edges with no endpoints in common.



Reducing bipartite matching to max-flow

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* $|M|$ (***max flow*** $\geq$ ***max-matching***)



$f(u,v) = 1 \ \forall \ (u,v) \in M.$

$f(v,t), f(s,u)$ are 1 if u,v are matched in M

Every vertex has at most one edge in M

So: capacity constraints sat.

Reducing bipartite matching to max-flow

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.

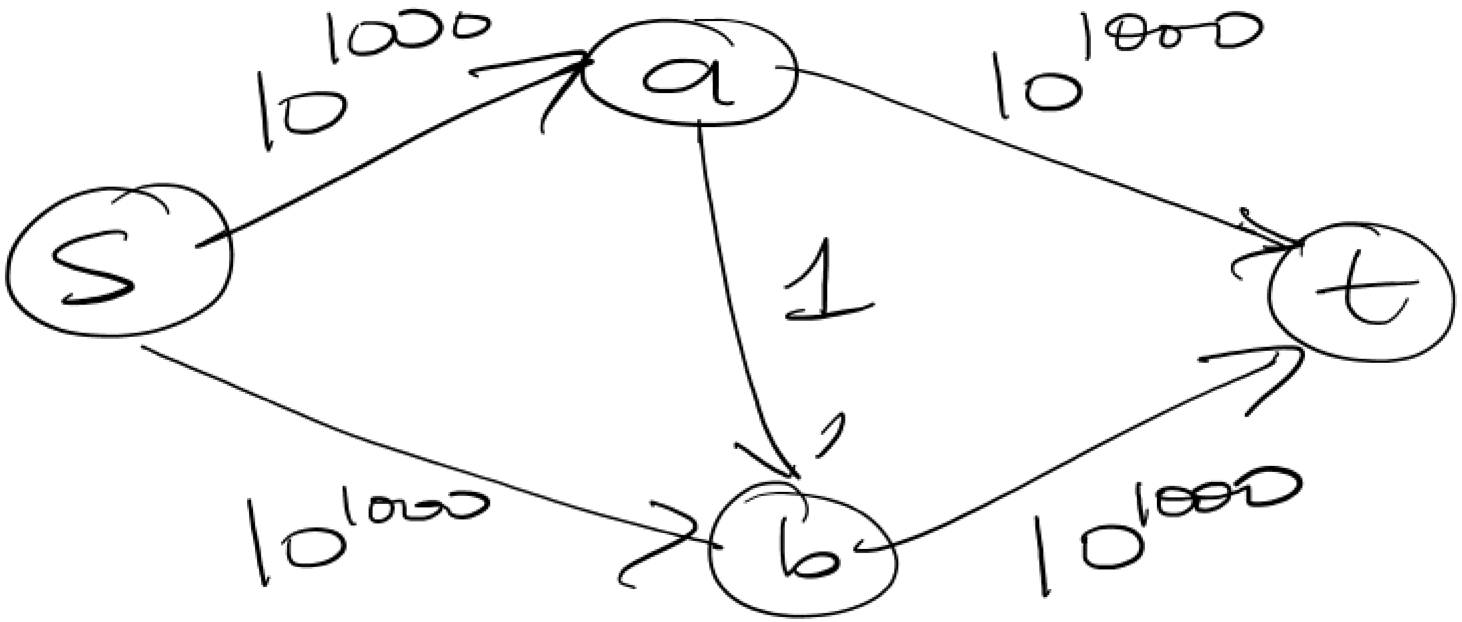
Claim #2: Given a flow f in our flow network, there exists a matching **M** *of size $|f|$ in the original graph* ($\Rightarrow$ ***max-flow*** $\leq$ ***max-matching***)

Let $M = \{\{u,v\} \mid f(u,v) = +1\}$. Then this set is a matching (why?). Further net flow $= |M|$ (why?)

Back to running time analysis for Ford-Fulkerson

Theorem: Ford-Fulkerson runs in $O(mF)$ time (with integer capacities)

Also Theorem: This bound is tight



how many iterations?

what's the size of the input?

Can we make it faster?

- Ford-Fulkerson finds *any* augmenting path until there are none left
- ***Idea***: Can we find "good" augmenting paths that guarantee a better running time? Yes!
- **Idea #1**: shortest augmenting paths
- **Idea #2**: "max bottleneck" paths

Dinitz-Edmonds-Karp: Shortest Augmenting Paths

- When we described Ford-Fulkerson, we found *any* augmenting path, (usually DFS is the simplest possible implementation)

Algorithm (Dinitz-Edmonds-Karp): Implement Ford-Fulkerson by finding ***shortest augmenting paths*** (e.g., using BFS) at each iteration.

Dinitz

Edmonds

Karp

Dinitz-Edmonds-Karp: Shortest Augmenting Paths

- When we described Ford-Fulkerson, we found *any* augmenting path, (usually DFS is the simplest possible implementation)

Algorithm (Dinitz-Edmonds-Karp): Implement Ford-Fulkerson by finding ***shortest augmenting paths*** (e.g., using BFS) at each iteration.

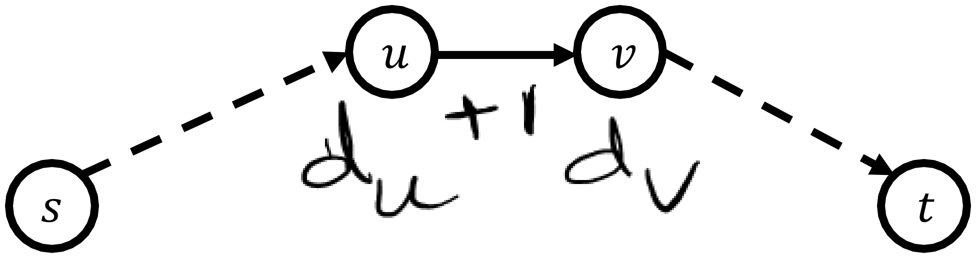
Theorem: Dinitz-Edmonds-Karp runs in $O(nm^2)$ time (poly time!)

No dependence on F!*

* See Wed lecture

Analysis of Dinitz-Edmonds-Karp

Lemma: Let d be the distance from s to t in the residual graph G_f. During Dinitz-Edmonds-Karp, d never decreases.



$d_v = 1 + d_u$

(since $s \to t$ is a shortest path)

Analysis of Dinitz-Edmonds-Karp

Lemma: After m iterations, d **must** increase.

An edge can only unsaturate after d increases.

Conclusion:

- Each iteration takes: $O(m)$
- Iterations per value of d: $O(m)$
- d can increase: n

$O(nm^2)$

#arithmetic operations do not depend on input numbers

Corollary: Maximum flow can be solved in strongly polynomial time!

Modern Approach to Maximum Flow


"push relabel" approach to max flow

$O(n^2m)$ time algorithm

Enter continuous optimization [Christiano-Kelner-Madry-Spielman-Teng…]

View the problem as a problem of finding a point in the intersection of two convex sets.

[Chen-Kyng-Liu-Peng-ProbstGutenberg-Sachdeva'22]

$< O\left(m^{1=\epsilon)}\right)$ for every $\epsilon > 0$! A near-linear time algorithm!

Minimum-cost Flows

Not covered in the lecture

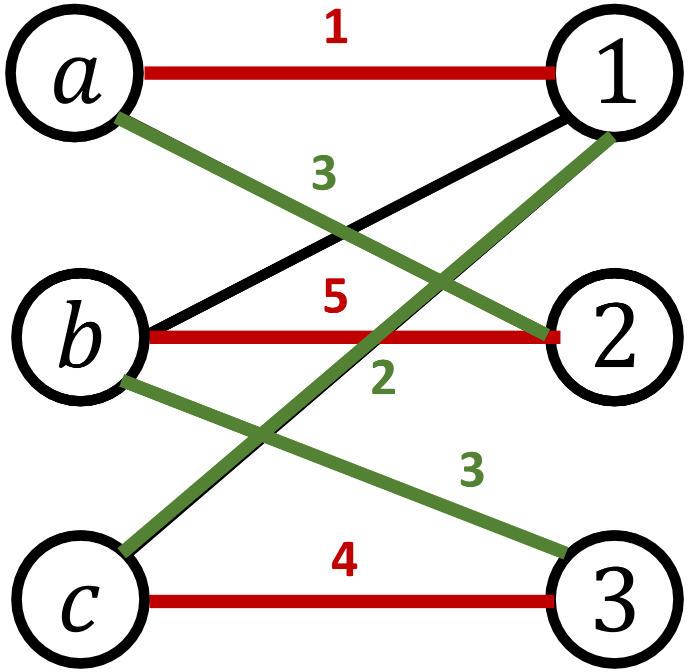
Won't be on the tests.

Min-Cost Flow

- There can be multiple maximum flows in a particular network

- What if we want to preference some over others?

- Example: Bipartite matching allows us to find whether a matching is possible. If there are multiple, can we also have preferences so that we get the “best” matching?



Min-Cost Flows

- We consider the same setting as before: A directed graph with capacities.
- Edges now also have ***costs***. Edge e costs $\$(e)$
- The cost of an edge is **per unit of flow**. The total cost is
- ***Goal***: Find maximum flow of minimum cost
- ***Note***: Other variants of the problem exist. E.g., you might want the minimum possible cost, regardless of the flow value (not maximum)

Assumptions

- Negative costs are allowed!
- Negative cycles are also allowed!!
  - However, some algorithms don't work.
  - Assume that there is no infinite capacity negative cycle (or the cost is $-\infty$)

The residual network

- The residual network is a powerful tool. Let's keep using it
- We define the *residual capacities and* ***residual costs***

$$c_f(u, v) =$$

$$\mathrm{S}_f(\boldsymbol{u}, \boldsymbol{v}) =$$

An Augmenting Path Algorithm

- Ford-Fulkerson finds a maximum flow (ignoring costs completely)
- What is a natural way to choose the augmenting paths?
- Find a ***cheapest augmenting path***.
- Use Bellman-Ford to find the augmenting paths (why not Dijkstra?)
- Requires no negative cycles in the input network!
- Assume integer capacities as well for termination

An Augmenting Path Algorithm

- We need two things:
  - ***Question 1:*** Does the algorithm terminate?
  - ***Question 2:*** Does it give a minimum-cost flow?

To answer Question 1, we need to prove that G_f never contains a negative-cost cycle! (Or the cheapest path would be undefined).

A Powerful Lemma

Theorem: Given a network G and flow f such that G_f contains no negative-cost cycles, if we augment a cheapest path, then the result still has no negative-cost cycles.

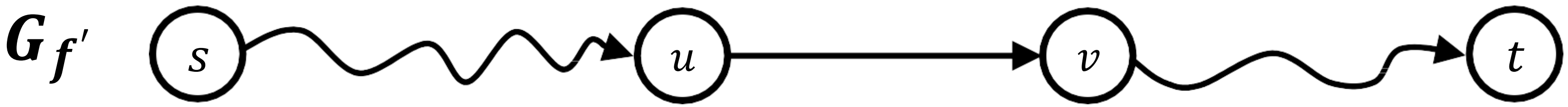
Lemma: Augmenting a cheapest path does not **decrease** the cost of the cheapest s-t path in the residual network.

A Powerful Lemma

Lemma: Augmenting a cheapest path does not **decrease** the cost of the cheapest $s - t$ path in the residual network.

Let $c(v)$ = *cost of cheapest* $s \rightarrow v$ *path in* G_f ***(before augmenting)***

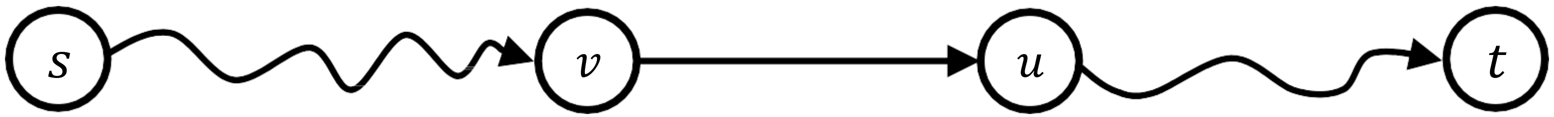
AFSOC that after augmenting, $\exists$ *an* s-t *walk cheaper than* $c(t)$



A Powerful Lemma

So, $S_{f'}(u, v)$ *must have changed! What is it?*

G_f



A Powerful Lemma

Lemma: Augmenting a cheapest path does not **decrease** the cost of the cheapest s-t path in the residual network.

⬇

Theorem: Given a network G and flow f such that G_f contains no negative-cost cycles, if we augment a cheapest path, then the result still has no negative-cost cycles.

⬇

Corollary: The cheapest augmenting path algorithm terminates!

Cheapest Augmenting Paths: Total Cost

- Similar analysis to Ford-Fulkerson

Theorem: Cheapest augmenting paths runs in $O(nmF)$ time

- Its just Ford-Fulkerson using Bellman-Ford at each iteration.
- Bellman-Ford costs $O(nm)$ and each iteration adds at least 1 flow
- So, the algorithm runs in $O(nmF)$

Takeaways

- Maximum flow can be solved in polynomial time (near-linear time as of 2022)!
- ***Dinitz-Edmonds-Karp*** (shortest augmenting paths) runs in $O(m^2 n)$ time.
- ***Powerful modeling tool***.