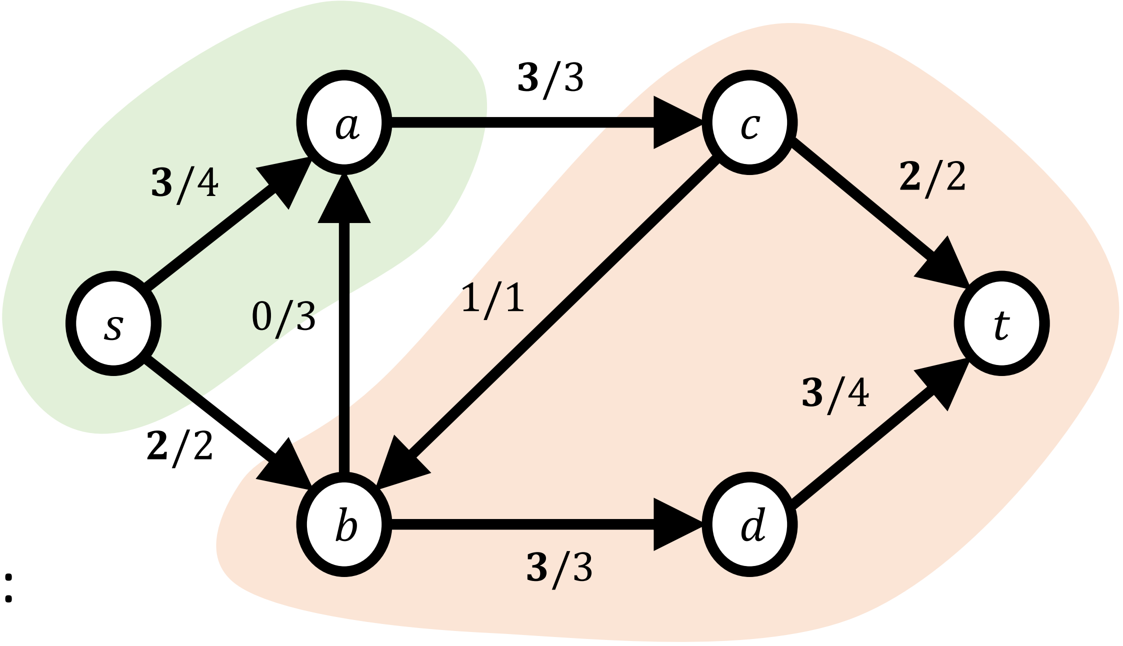
Algorithm Design and Analysis

Network Flow Part II: Advanced Flow Algorithms

Roadmap for today

- Review network flow and the ***Ford-Fulkerson*** algorithm
- Applications of network flow: **Bipartite matching**
- Make the Ford-Fulkerson algorithm faster! (***Edmonds-Karp*** algorithm)
- Another flow problem, ***minimum-cost flows***
- The ***cheapest augmenting paths*** algorithm

Network Flow recap

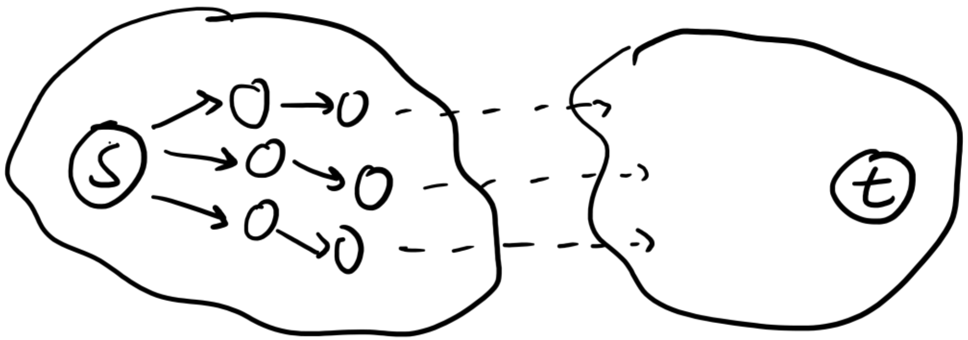


- A ***flow network*** is a directed graph with:
  - ***capacities*** $c(u,v)$
  - A ***source vertex*** s and ***sink vertex*** t
- A ***flow*** is an assignment of values to edges:
  - ***Capacity constraint:*** $0 \leq f(u,v) \leq c(u,v)$
  - ***Conservation constraint:*** "flow in = flow out" for all vertices except s, t

$$\sum_{v \in V} f(u,v) = \sum_{v \in V} f(v,u)$$

- The **value** of a flow is the net flow out of the source (can prove via conservation that is = net flow into sink)

Network Flow recap

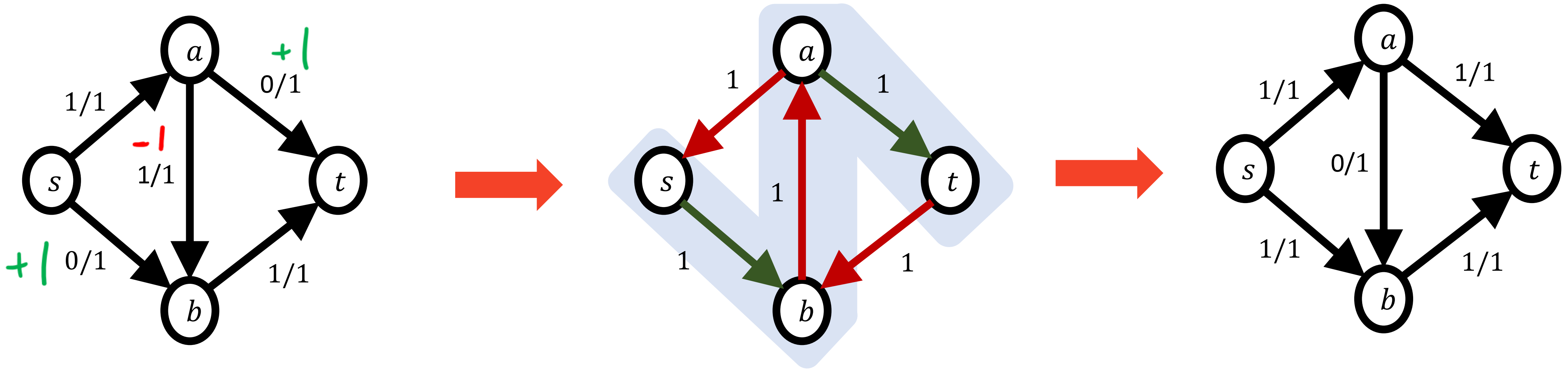


- The ***maximum flow*** problem is to find a flow of maximum value
- We learned the ***Ford-Fulkerson*** algorithm:
  - Define the ***residual capacities:***

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

Ford Fulkerson Algorithm

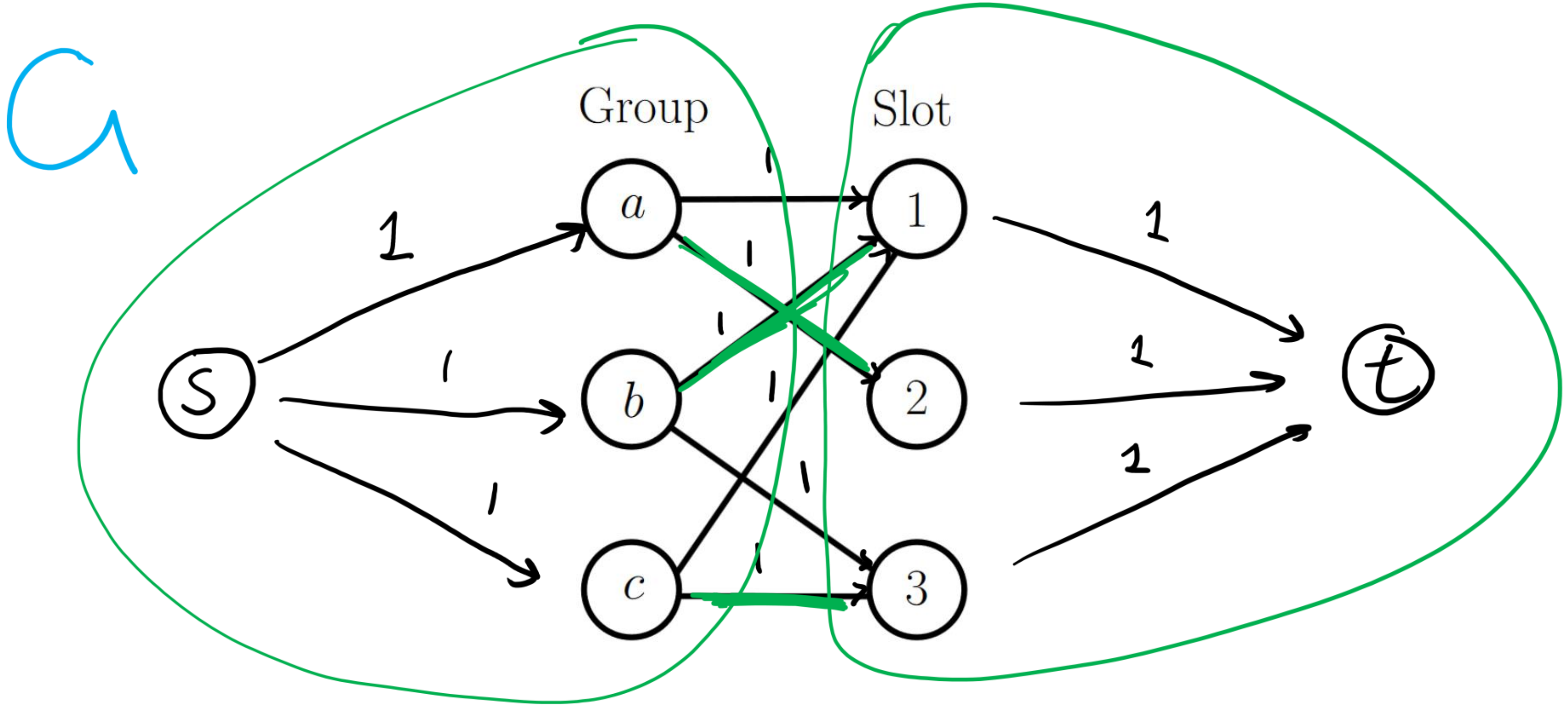
while there exists an s-t path in the residual network:
  add maximal flow to that path.



Applications

Bipartite Matching

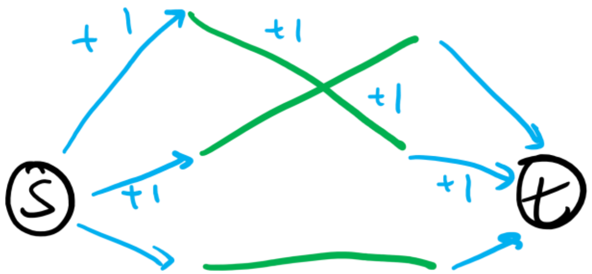
Problem (Bipartite matching): Given a bipartite graph G, find a largest possible set of edges with no endpoints in common.



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 an integral flow f in our flow network of value $|M|$ (⇒ ***max flow*** *≥* ***max-matching****)*



For each $(u,v) \in M$

send 1 flow $s \to u \to v \to t$

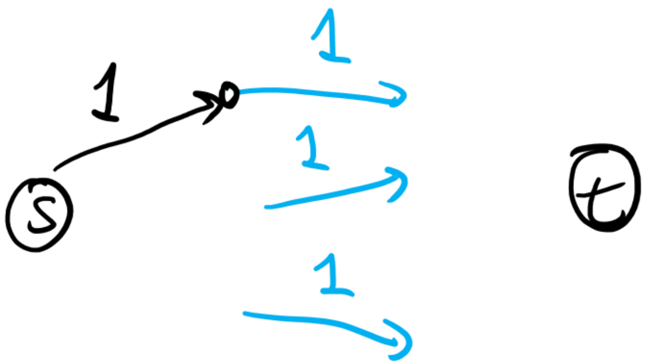
Valid by def'n of matching

value of f is M and its integral

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 an integral flow f in our flow network, there exists a matching M of size $|f|$ in the original graph (⇒ ***max-flow*** $\leq$ ***max-matching****)*



cap $(s,u)=1$ + conservation

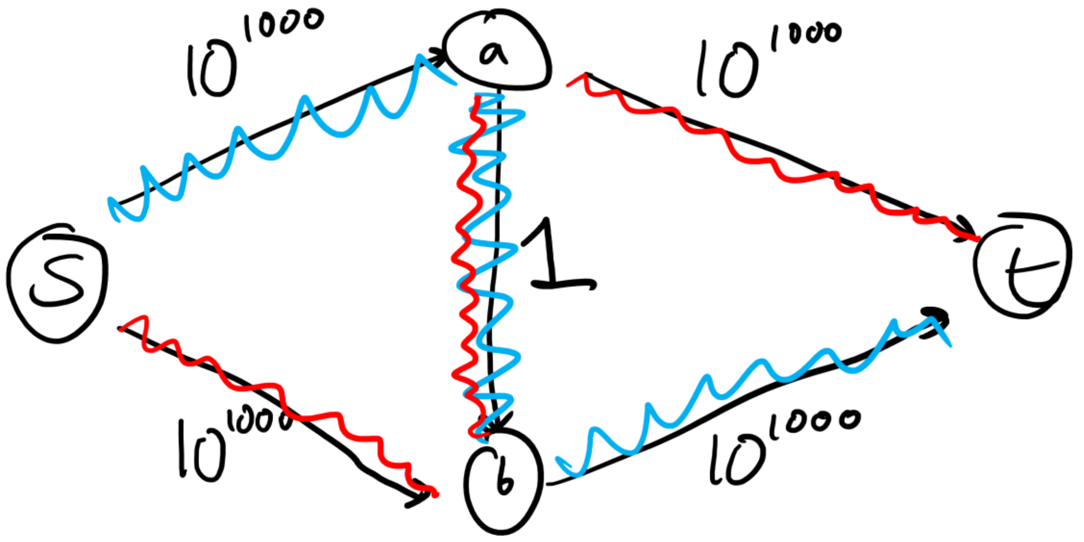
⇒ at most one edge matches u

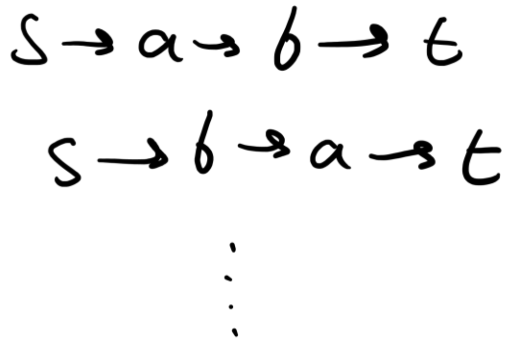
Back to Network Flow Part II

Worst-case runtime

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

Also Theorem: This bound is tight





How to 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**: Maximum bottleneck path
- **Idea #2**: Shortest augmenting paths

Edmonds-Karp (Shortest Augmenting Paths)

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

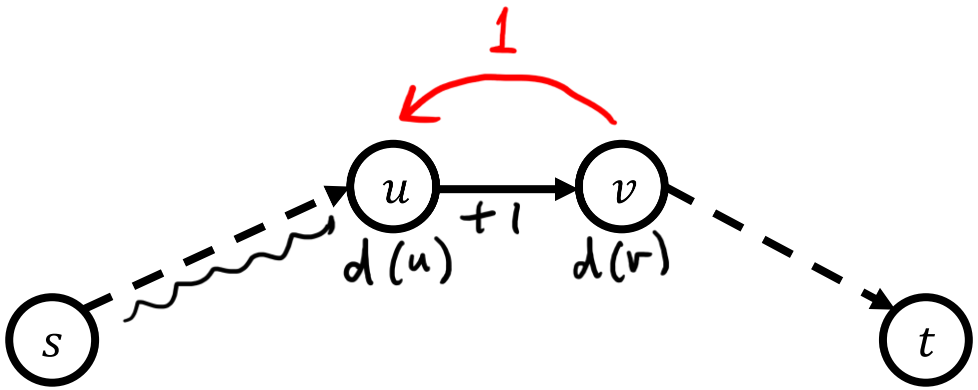
Algorithm (Edmonds-Karp): Implement Ford-Fulkerson by finding ***shortest augmenting paths*** (e.g., using BFS) at each iteration.

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

$O(mF)$

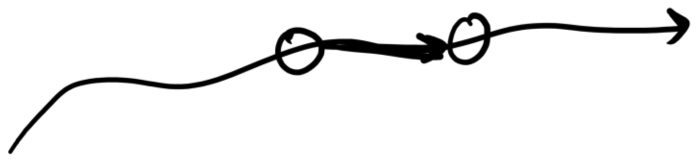
Analysis

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



Analysis

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



An edge only unsaturate after d increases.

Conclusion:

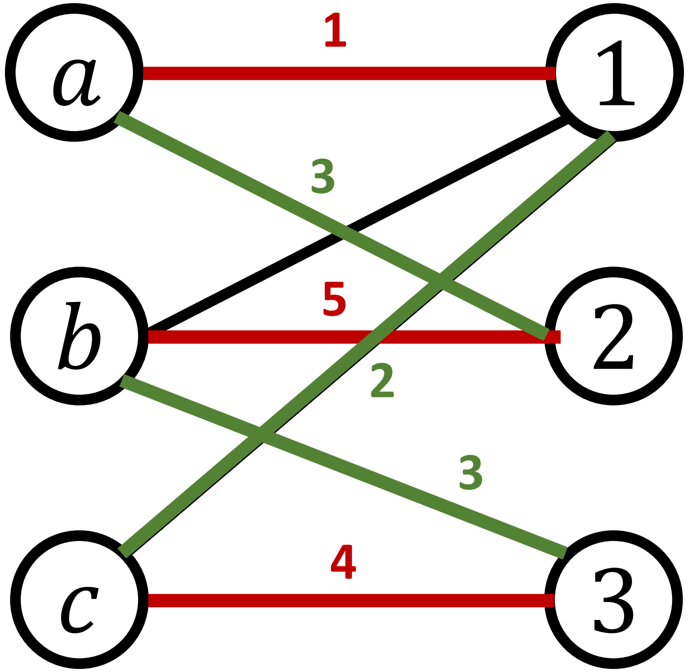
- Each iteration takes: $O(m)$
- Iterations per value of d: $O(m) \Rightarrow O(nm^2)$
- d can increase: $n-1$ times

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

Minimum-cost Flows

Motivation

- 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?



Minimum-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

$$\text{cost}(f) = \sum_{e \in E} \$(e) f(e)$$

- ***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***

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

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

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

Does it work?

- 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.

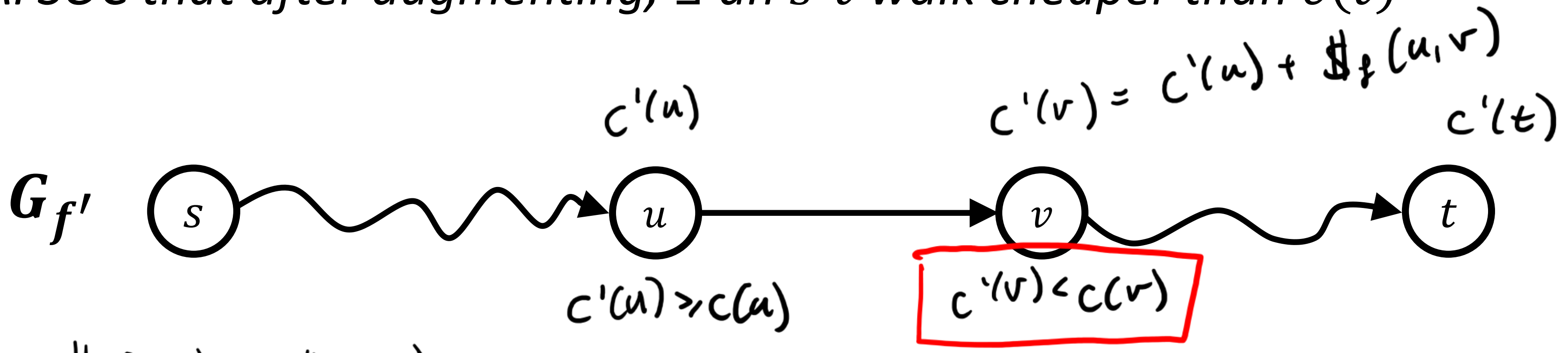
Lemma: Augmenting a cheapest path does not **decrease** the cost of the cheapest s-t path in the residual network.

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* $s \to v$ *path in* G_f ***(before augmenting)***

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



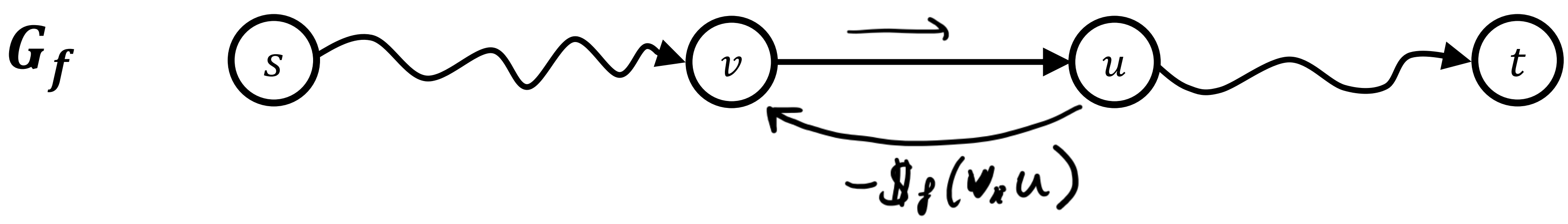
Assume $\$_f(u,v) = \$(u,v)$

$c'(v) \geq c(u) + \$_f(u,v) = c(u) + \$(u,v) = c(v)$

A powerful lemma, continued…

So, $\$_{f'}(u, v)$ *must have changed! What is it?* $\$_f(u,v) = -\$_f(v,u)$

$c(v)$ $\quad$ $c(u) = c(v) + \$_f(v,u)$



$c'(v) \geq c(u) - \$_f(v,u)$

$= c(v)$

$\Leftarrow \quad c(u) = c(v) + \$_f(v,u)$

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: 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 $O(nmF)$

Take-home messages

- Maximum flow can be solved in polynomial time!
- ***Edmonds-Karp*** (shortest augmenting paths) runs in $O(nm^2)$ time
- The ***minimum-cost flow problem***, and an algorithm
- ***Cheapest augmenting paths***
  - Ford-Fulkerson but always use cheapest cost augmenting path
  - Works for integer-capacity, negative-cycle-free networks
  - Runs in $O(nmF)$ time