Algorithms - Spring '25

Flows:
Ford-Fulkerson

Recap:

- HW: due Wednesday
- Next HW: up Wed, due the following Friday (?)
- Next Monday, NO Class.
  office hours will be Wed. from 1-3pm (next week)

More formally: Flow

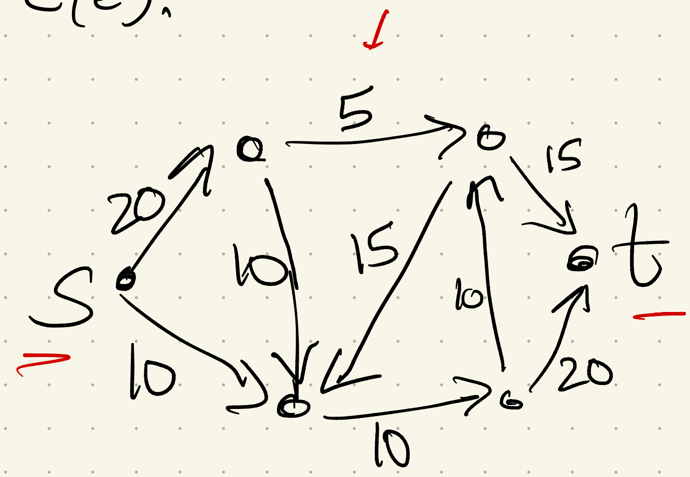
Given a directed graph with two designated vertices, s and t.

Each edge is given a capacity c(e).

Assume: - No edges enter s

- No edges leave t

- Every $c(e) \in \mathbb{Z}^+$.



Max flow: Find most I can send from s to t without exceeding edge capacities.

Min cut: find lightest set of edges separating s from t

Formalizing flow:

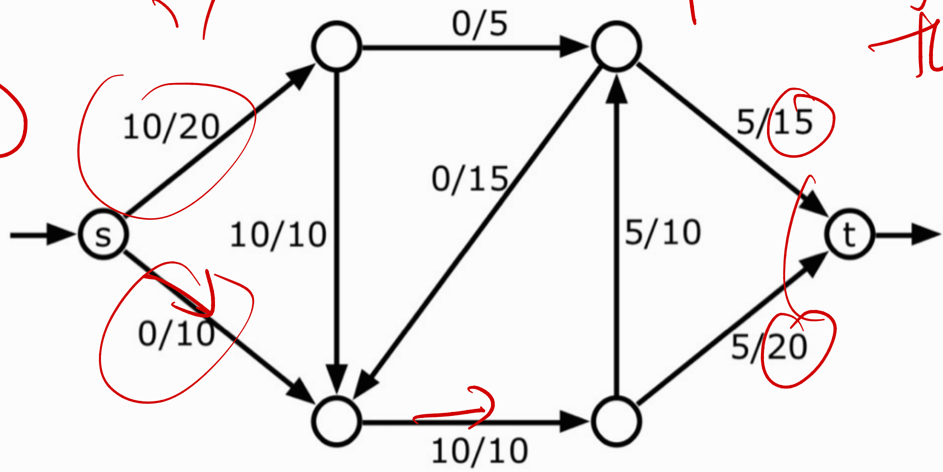
A flow is a function $f: E \to \mathbb{R}^+$, where $f(e)$ is the amount of flow going over edge e.

Must satisfy 2 things: → feasible

- Edge constraints: $0 \le f(e) \le c(e)$ (don't overflow edge)
- Vertex constraints: $\sum_{v \ne s \text{ or } t}$ flow into v $= \sum$ flow out

only s can ship out, & no other vertex than t can store

$\partial(v) = 0$



An (s,t)-flow with value 10. Each edge is labeled with its flow/capacity.

Value(f) =

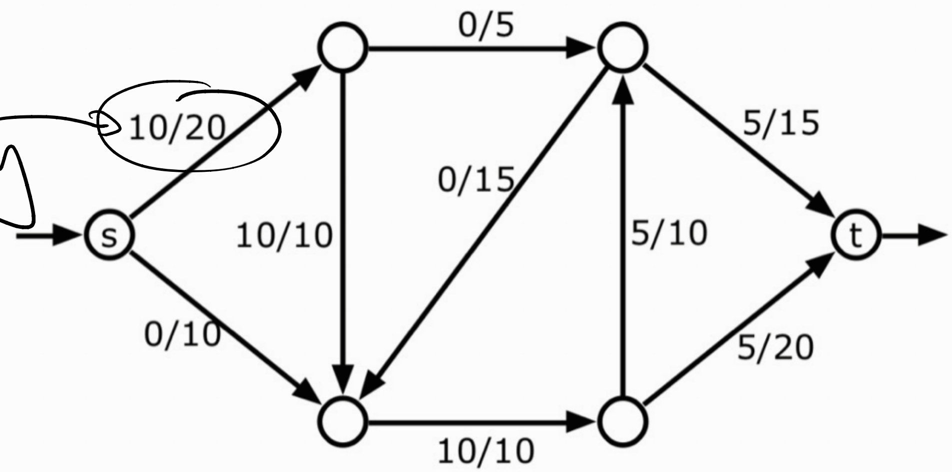
$$\partial(s) = \sum_{e \text{ out of } s} f(e)$$

$$= \sum_{e \text{ into } t} f(e)$$

$-\delta(t)$

Note on notation & conventions:

flow/capacity



An (s, t)-flow with value 10. Each edge is labeled with its flow/capacity.

A flow is a function on edges!
(so are capacities)

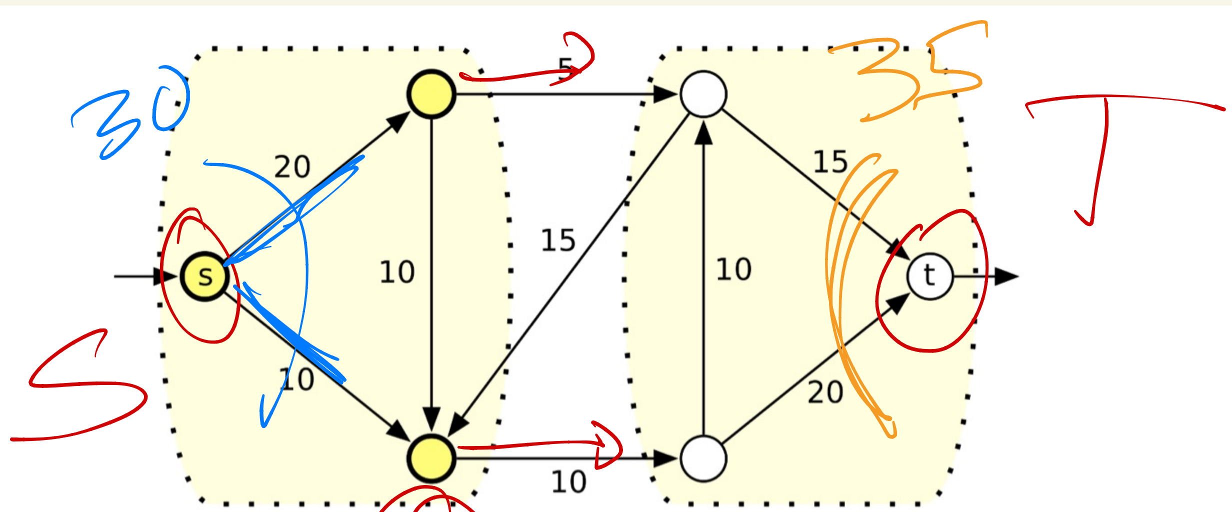
Assume both are positive!
(so vertex constraints make sense)

Formalizing Cuts

An s-t cut is a partition of the vertices into 2 sets, S and T, so that

- $s \in S$
- $t \in T$
- $S \cap T = \emptyset$,

$S \cup T = V$



An (s,t)-cut with capacity 15. Each edge is labeled with its capacity.

S, T is a partition of V

(S,T)

The capacity of a cut is $\sum_{\substack{\vec{uv} \in E \\ \text{with } u \in S,\, v \in T}} c(\vec{uv})$

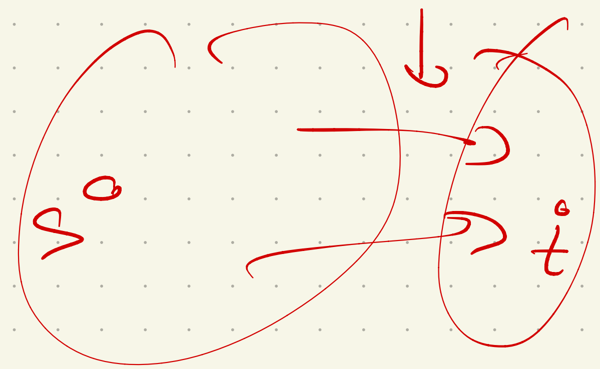
Thm: (Ford-Fulkerson '54, Elias-Feinstein-Shannon '56)

The max flow value = min cut value

Wow!

One way is easy:

Any flow $\leq$ any cut.



Why?

Can exceed edges out of S + into T

More formally:

any flow ≤ any cut

Proof: Choose your favorite flow f and your favorite cut (S, T), and then follow the bouncing inequalities:

$f(e)$

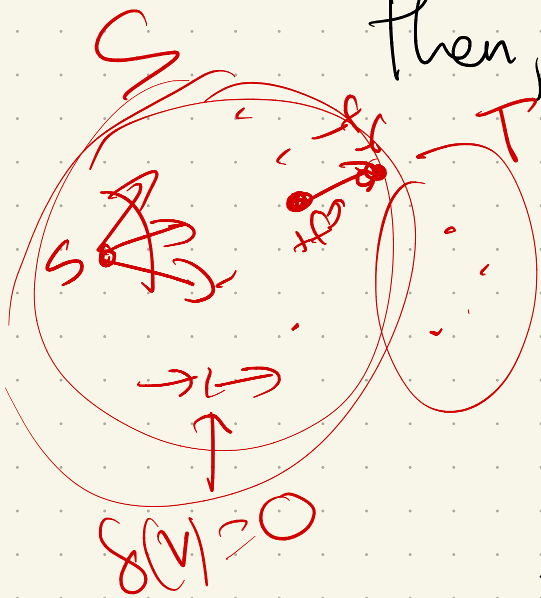
$$
\begin{aligned}
|f| &= \partial f(s) && \text{[by definition]}\\
&= \sum_{v\in S} \partial f(v) && \text{[conservation constraint]}\\
&= \sum_{v\in S}\sum_{w} f(v\to w) - \sum_{v\in S}\sum_{u} f(u\to v) && \text{[math, definition of } \partial\text{]}\\
&= \sum_{v\in S}\sum_{w\notin S} f(v\to w) - \sum_{v\in S}\sum_{u\notin S} f(u\to v) && \text{[removing edges from } S \text{ to } S\text{]}\\
&= \sum_{v\in S}\sum_{w\in T} f(v\to w) - \sum_{v\in S}\sum_{u\in T} f(u\to v) && \text{[definition of cut]}\\
&\le \sum_{v\in S}\sum_{w\in T} f(v\to w) && \text{[because } f(u\to v) \ge 0\text{]}\\
&\le \sum_{v\in S}\sum_{w\in T} c(v\to w) && \text{[because } f(v\to w) \le c(v\to w)\text{]}\\
&= \|S, T\| && \text{[by definition]}
\end{aligned}
$$

→ Slower...

More carefully:

Choose flow f + cut (S, T)

$$\text{value}(f) = \delta f(s) \underbrace{+0+0+0}_{v \neq s} = \sum_{s \to u} f(s) - \overbrace{\sum_{u \to s} f(u)}^{=0}$$



then, since flow in = flow out for all $v \neq s$ or t, $\delta(v) =$ flow out of v $-$ flow into v

$$= \sum_{v \in S} \partial(v) \quad \text{+ by dfn of } \partial$$

$$= \sum_{v \in S} \left[\sum_{w \in V} f(v \to w) - \sum_{w \in V} f(w \to v) \right]$$

+ any edge entirely on S side is counted twice

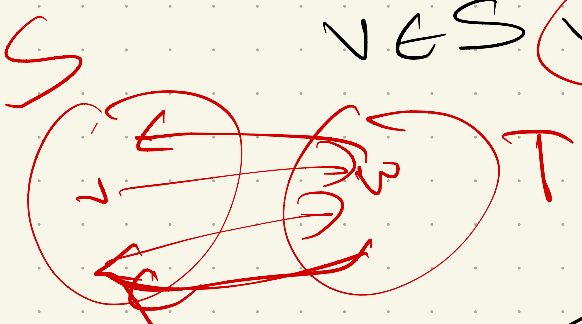
$\Longrightarrow$

$$= \sum_{v \in S} \sum_{w \notin S} f(v \to w) - \sum_{v \in S} \sum_{w \notin S} f(w \to v)$$

then if $w \notin S$, know $w \in T$

(because it's a cut!)

$$= \sum_{v \in S} \sum_{w \in T} f(v \to w) - \underbrace{\sum_{v \in S} \sum_{w \in T} f(w \to v)}_{\geq 0 \text{ b/c flow } \geq 0}$$



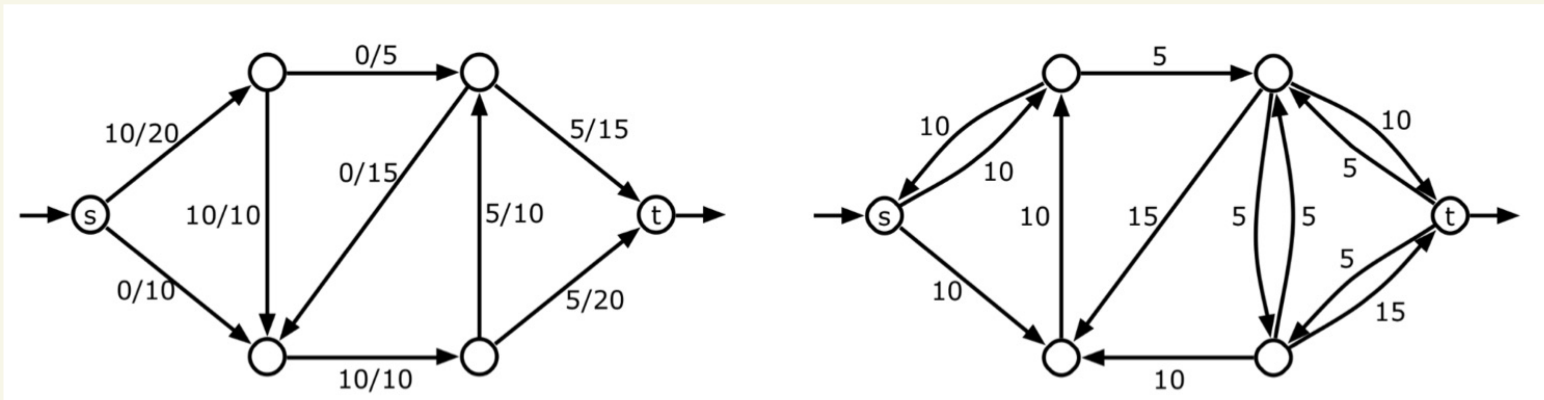
so $\leq \displaystyle\sum_{v \in S} \sum_{w \in T} f(v \to w)$ & flow $\leq$ cap

$$\Rightarrow \leq \sum_{v \in S} \sum_{w \in T} c(v \to w) = \text{cap of cut}$$

Key tool in proof:

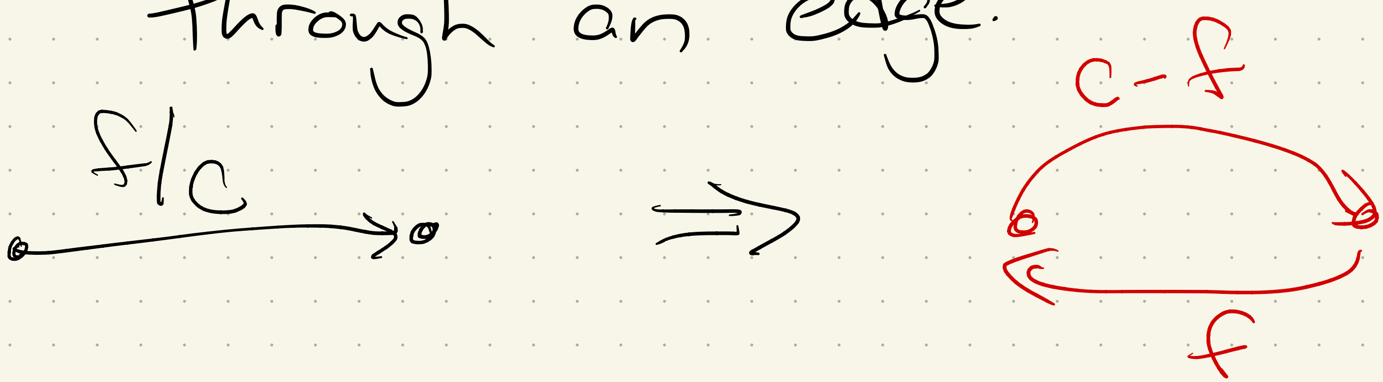
Residual network G_f:

↙ E in G_f is ≤ 2(# edges in G)

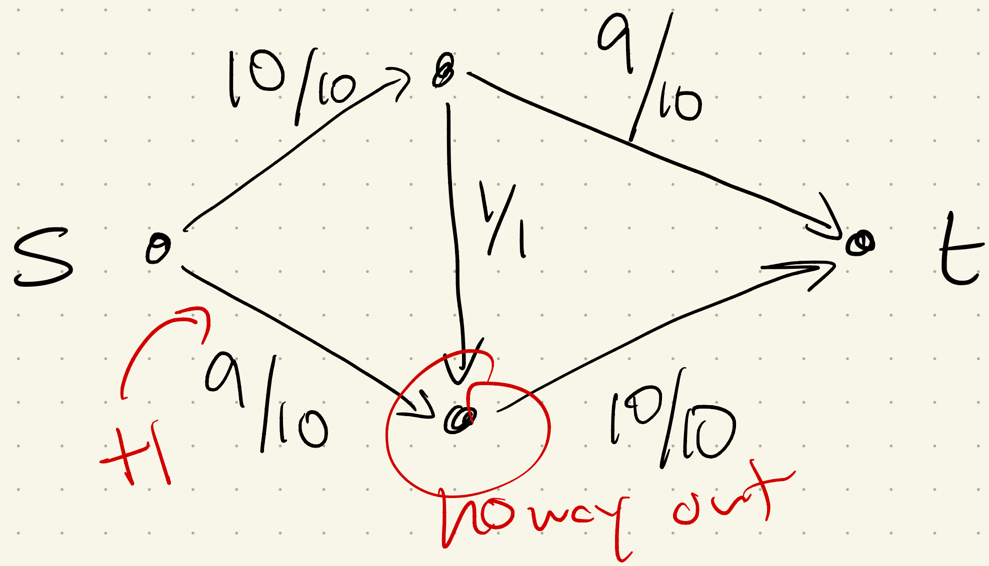


A flow f in a weighted graph G and the corresponding residual graph G_f.

Intuitively: Shows how much more (or less) flow can be pushed through an edge.



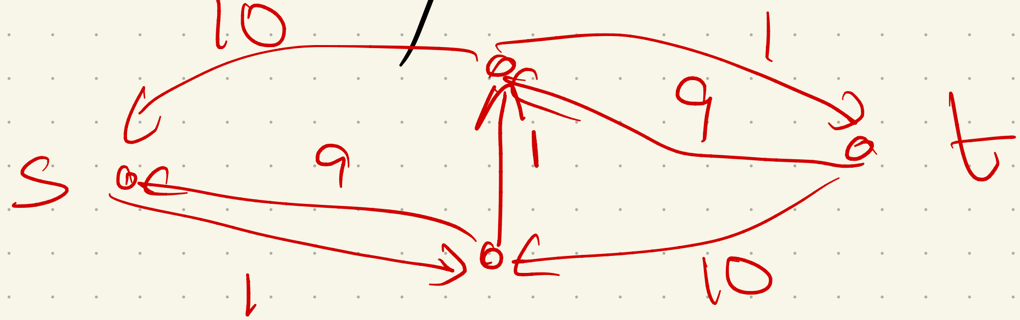
Why can't we just be greedy & push?



19 units

Can get "stuck" if we choose wrong initially:

Are there any more flow paths?



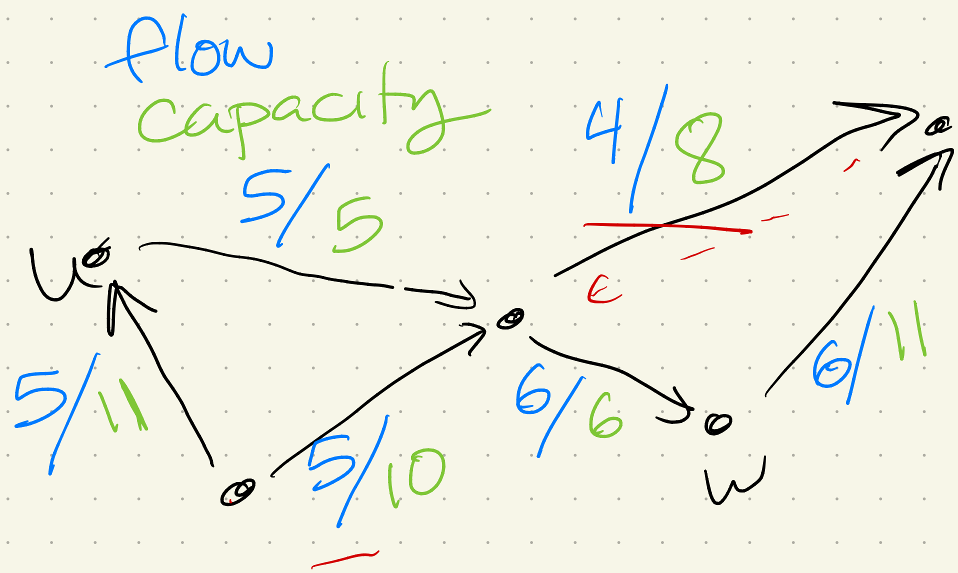
More formally: Residual network G_f:

Given G & f:

$$c_f(u \to v) = \begin{cases} c(u\to v) - f(u\to v) & \text{if } u \to v \text{ is in } E \\ f(u \to v) & \text{if } v \to u \text{ is in } E \\ \text{no edge (or 0)} & \text{otherwise} \end{cases}$$

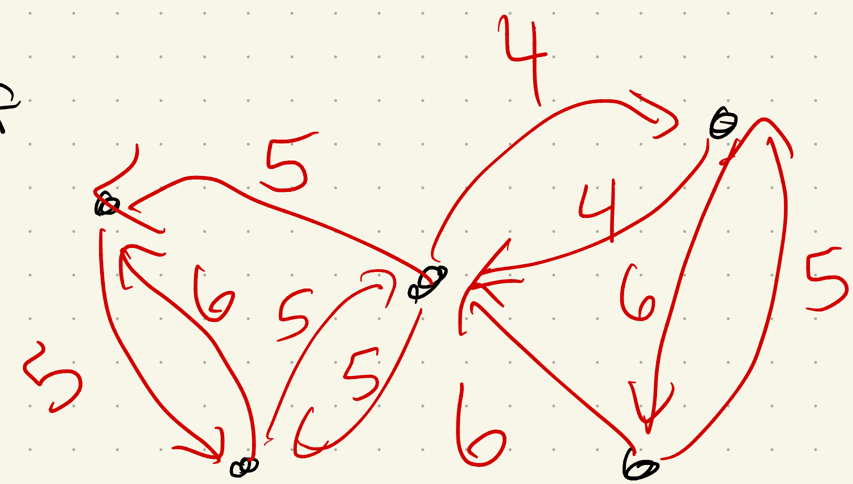
↑ if reverse edge

Ex: G, f

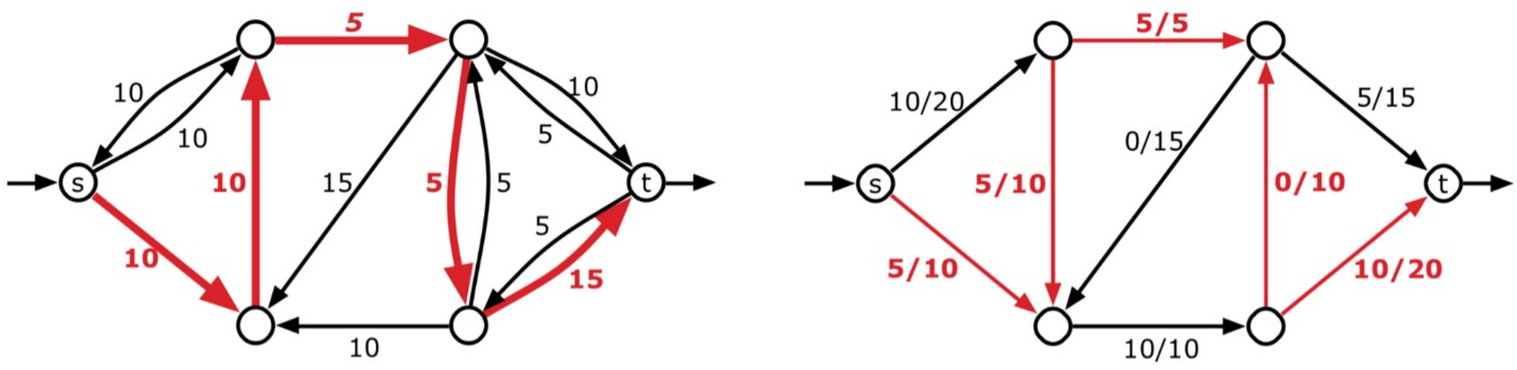


$\Rightarrow$

G_f



Augmenting a path: send more!



An augmenting path in G_f with value $F = 5$ and the augmented flow f'.

This is just an s-t path in G_f.

Then, find min capacity edge on that path

Claim: I can build a new flow whose value is bigger than f's

<u>Next</u>: Show that can get them equal.

How?

Well, take some flow. Either:

1. f is maximum.

   If so, find a cut of the same value.

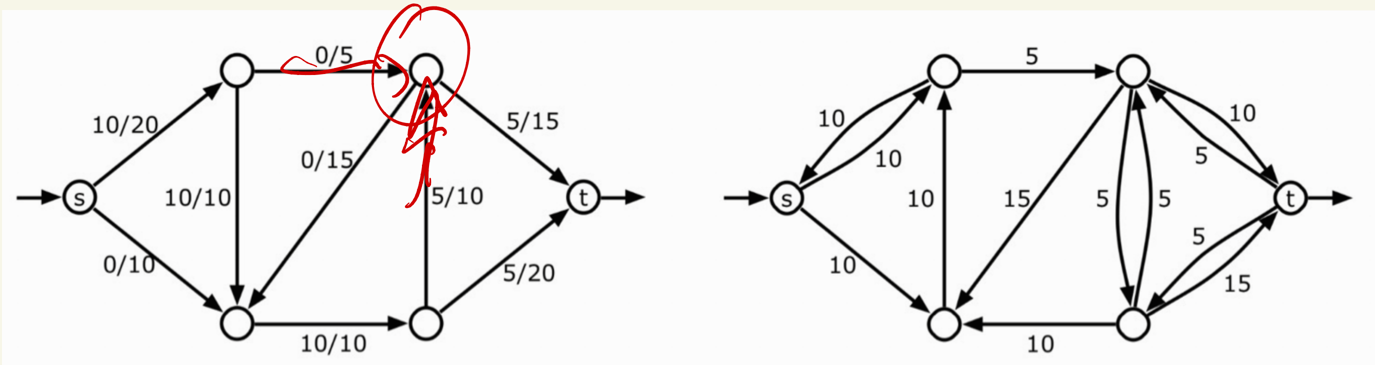
2. Or, it isn't!

   ↳ Find a bigger flow.

   use aug. path & G_f

Augmenting a path:

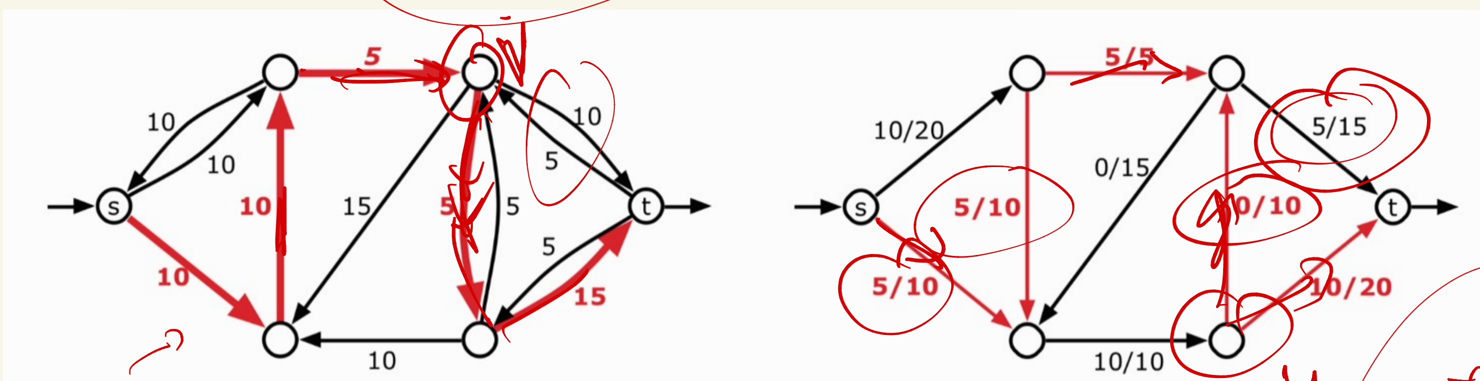
Suppose there is a path in G_f from s to t:



A flow f in a weighted graph G and the corresponding residual graph G_f.

Build f':

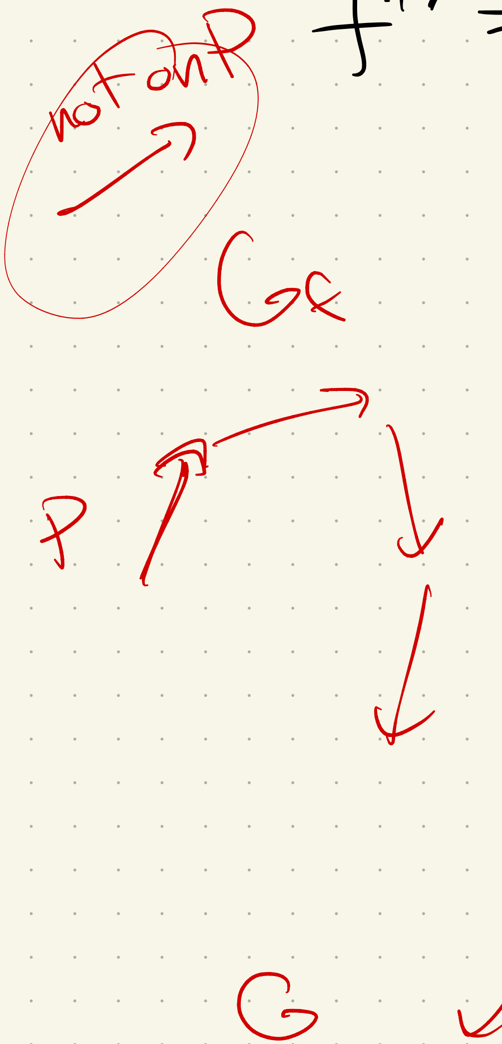
$\xrightarrow{+c} \circ \xleftarrow{-c}$



An augmenting path in G_f with value $F = 5$ and the augmented flow f'.

$\xleftarrow{-c} \overset{u}{\circ} \xrightarrow{+c}$

More formally, given path $P \in G_f$ with bottleneck $F = c$:



$$f' = \begin{cases} \text{if } \vec{uv} \notin P: & f'(\vec{uv}) = f(\vec{uv}) \\ \text{if } \vec{uv} \in P \text{ and } \vec{uv} \in G: & f'(\vec{uv}) = f(\vec{uv}) + c \\ \text{if } \vec{uv} \in P \text{ and } \vec{vu} \in G: & f'(\vec{uv}) = f(\vec{uv}) - c \end{cases}$$

Claim: f' is also a feasible flow!

Why? Need edge & vertex constraints:

& bigger by $+c$

- For any $u \to v$ not on augmenting path, $f'(u \to v) = f(u \to v)$

  so unchanged & still $\le cap(u \to v)$

- For $u \to v$ on augmenting path,

  if $u \to v \in G$: choose lowest bottleneck so $+c$ will not exceed cap.

  If $v \to u \in G$: know can subtract $f(u \to v)$ because that was part of bottleneck also

And vertex constraints:

Consider v:

if not on path: unchanged so if f had $\partial(v)=0$, still true

If on path in G_f: $\longrightarrow \overset{v}{\circ} \longrightarrow$

in G: $\xrightarrow{+c} \circ \xrightarrow{+c}$ net $=0$ $\quad \xleftarrow{-c} \circ \xleftarrow{-c}$ $\partial(v)=0$

$\longrightarrow \circ \longleftarrow \qquad \longleftarrow \circ \longrightarrow$

Claim: If f is a maximum flow, then G_f has no augmenting path

Proof: by contradiction

Assume f is maximum.

Build G_f + find path.

Use this path a bigger flow f'.

+ bottle neck edge

$\Rightarrow$ then f was not maximum

<u>So:</u> f wasn't a max flow, since f' is larger.

On other hand:

if G_f has no $s \to t$ path, find

$|S|$ = set of vertices that s can reach

<u>Claim:</u> $(S, V-S)$ is a cut.

(& f uses every $S \to V-S$ edge to its max capacity)

Why?

Immediate Algorithm:

Start with $f = 0$.

Build G_f

WFS(G_f, s)

While t & s in same component:

find $s \to t$ path via WFS

Augement along the path to get f'

$f \leftarrow f'$

Build G_f

WFS(G_f, s)

$\boxed{\text{Runtime?}}$

Why all this integrality stuff?

We are assuming each path pushes at least 1 more unit of flow!

Can it be that bad?

Yes:

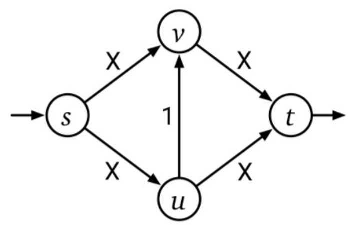


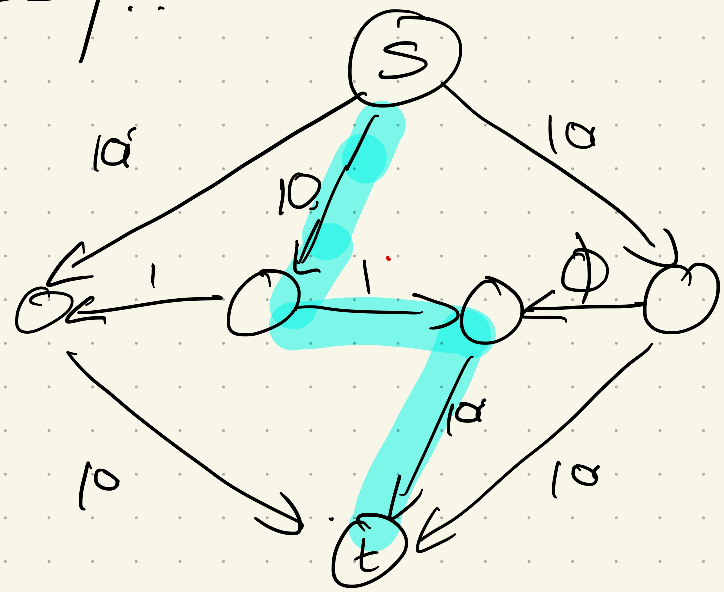
Figure 10.7. Edmonds and Karp's bad example for the Ford-Fulkerson algorithm.

How "big" is f?

(Remember, not part of input!)

What if it's <u>not</u> integers?

Messy!!



The key:

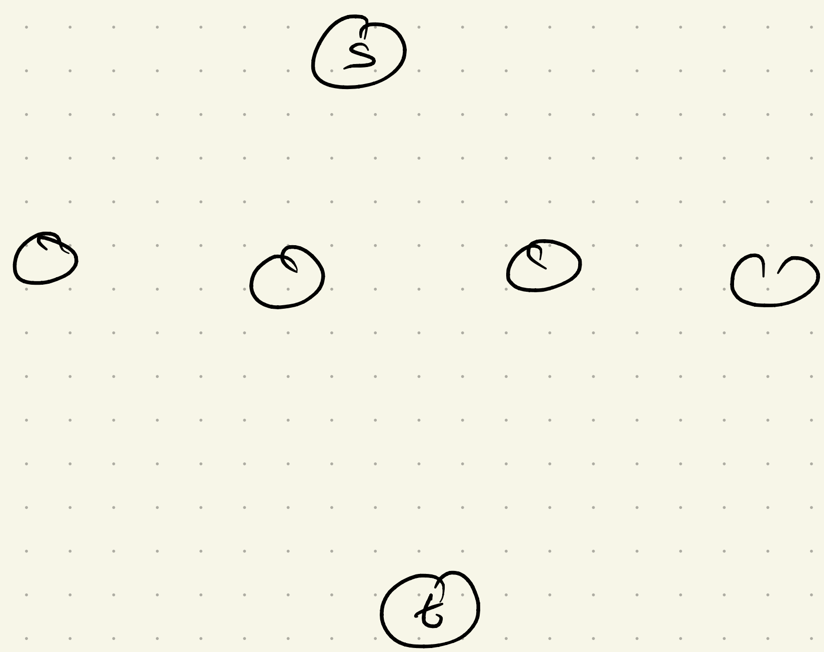
$$\phi = \frac{1+\sqrt{5}}{2}$$

WHY??

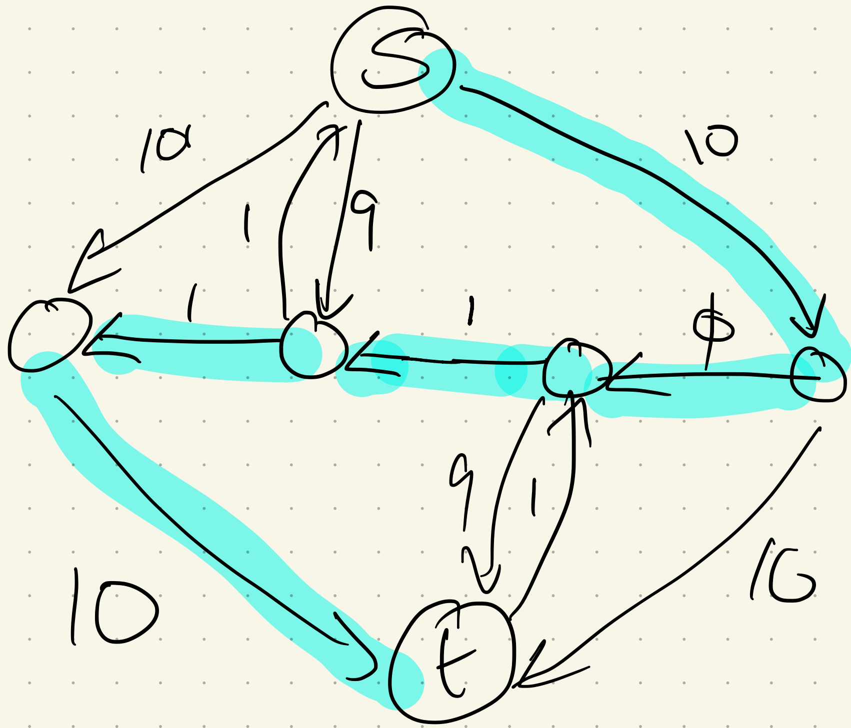
Simple:

$$1-\phi = \phi^2$$

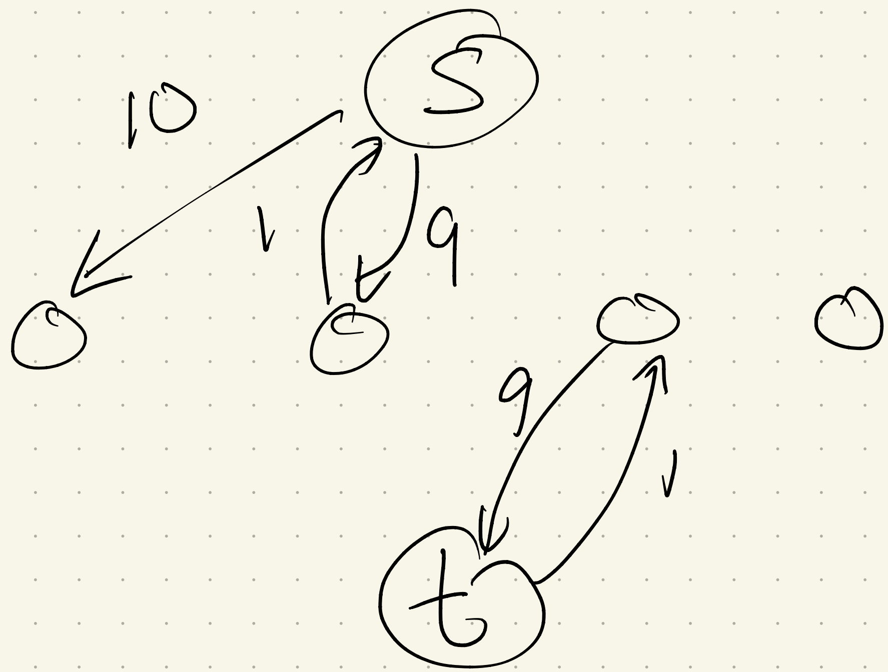
G_f:



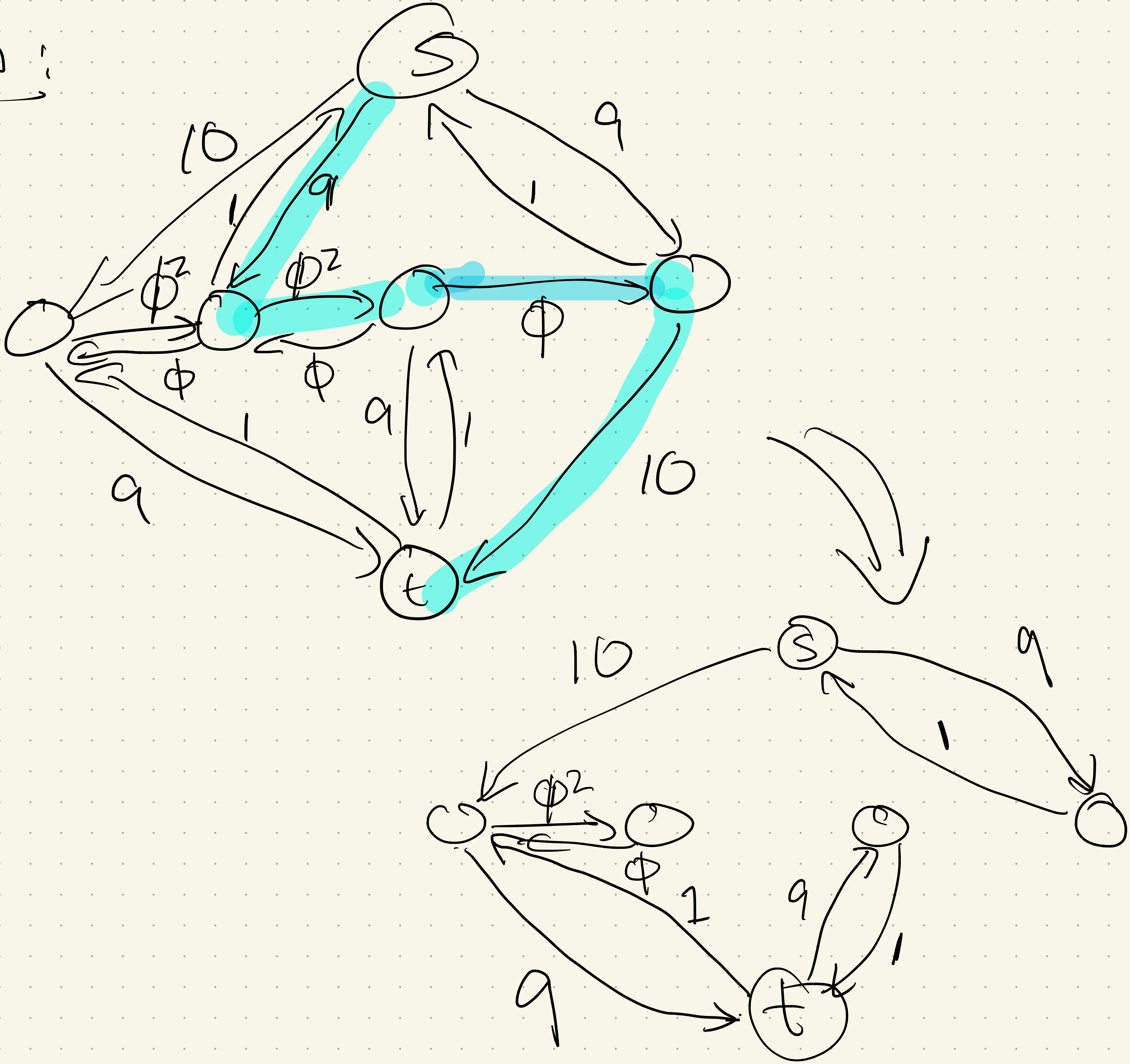
Next path:



New G_f:



Then:



Continue to push:

Ends with:

ϕ, 0, and $1-\phi=\phi^2$

Repeat:

- ϕ^2, 0, ϕ^3

then

-

etc $\vdots$

But: max flow $= 21$

Faster versions

This is an active area of research!

We'll see two faster examples, both (relatively) simple variations on the Ford-Fulkerson algorithm:

1. Edmonds-Karp: choose largest bottleneck edge
   $\hookrightarrow O(E^2 \log E \log |f^*|)$

2. shortest augmenting path
   $\hookrightarrow O(VE^2)$

And... not done!

Technique	Direct	With dynamic trees	Source(s)
Blocking flow	$O(V^2E)$	$O(VE\log V)$	[Dinitz; Karzanov; Even and Itai; Sleator and Tarjan]
Network simplex	$O(V^2E)$	$O(VE\log V)$	[Dantzig; Goldfarb and Hao; Goldberg, Grigoriadis, and Tarjan]
Push-relabel (generic)	$O(V^2E)$	—	[Goldberg and Tarjan]
Push-relabel (FIFO)	$O(V^3)$	$O(VE\log(V^2/E))$	[Goldberg and Tarjan]
Push-relabel (highest label)	$O(V^2\sqrt{E})$	—	[Cheriyan and Maheshwari; Tunçel]
Push-relabel-add games	—	$O(VE\log_{E/(V\log V)} V)$	[Cheriyan and Hagerup; King, Rao, and Tarjan]
Pseudoflow	$O(V^2E)$	$O(VE\log V)$	[Hochbaum]
Pseudoflow (highest label)	$O(V^3)$	$O(VE\log(V^2/E))$	[Hochbaum and Orlin]
Incremental BFS	$O(V^2E)$	$O(VE\log(V^2/E))$	[Goldberg, Held, Kaplan, Tarjan, and Werneck]
Compact networks	—	$O(VE)$	[Orlin]

Figure 10.10. Several purely combinatorial maximum-flow algorithms and their running times.

Many use <u>very</u> different techniques

- linear programming
- complex data structures
- <u>not</u> residual graphs

Still active:

Showing 1–50 of 368 results for all: maximum flow in graphs

Search v0.5.6 released 2020-02-24

maximum flow in graphs | All fields | Search

Show abstracts Hide abstracts

Advanced Search

50 results per page. Sort results by Announcement date (newest first) Go

1 2 3 4 5 … Next

1. arXiv:2503.20985 [pdf, ps, other] cs.DS

Deterministic Vertex Connectivity via Common-Neighborhood Clustering and Pseudorandomness

Authors: Yonggang Jiang, Chaitanya Nalam, Thatchaphol Saranurak, Sorrachai Yingchareonthawornchai

Abstract: We give a deterministic algorithm for computing a global minimum vertex cut in a vertex-weighted graph n vertices and m edges in $\widehat{O}(mn)$ time. This breaks the long-standing $\widehat{\Omega}(n^4)$-time barrier in dense… ▽ More

Submitted 26 March, 2025; originally announced March 2025.

2. arXiv:2503.13274 [pdf, ps, other] cs.DS

Parallel Minimum Cost Flow in Near-Linear Work and Square Root Depth for Dense Instances

Authors: Jan van den Brand, Hossein Gholizadeh, Yonggang Jiang, Tijn de Vos

Abstract: …-edge graphs with integer polynomially-bounded costs and capacities, we provide a randomized parallel algorithm for the minimum cost flow problem with $\tilde{O}(m+n^{1.5})$ work and $\tilde{O}(\sqrt{n})$ depth. On moderately dense graphs $(m>n^{1.5})$, our algorithm is the fir… ▽ More

Submitted 17 March, 2025; originally announced March 2025.

3. arXiv:2502.09105 [pdf, other] cs.DS

Incremental Approximate Maximum Flow via Residual Graph Sparsification

Authors: Gramoz Goranci, Monika Henzinger, Harald Räcke, A. R. Sricharan

Abstract: …maximum flow in undirected, uncapacitated n-vertex graphs undergoing m edge insertions in $\tilde{O}(m+nF^*/\varepsilon)$ total update time, where F^* is the… ▽ More

Submitted 13 February, 2025; originally announced February 2025.

Plus work for special classes of graphs:
planar, sparse, etc.