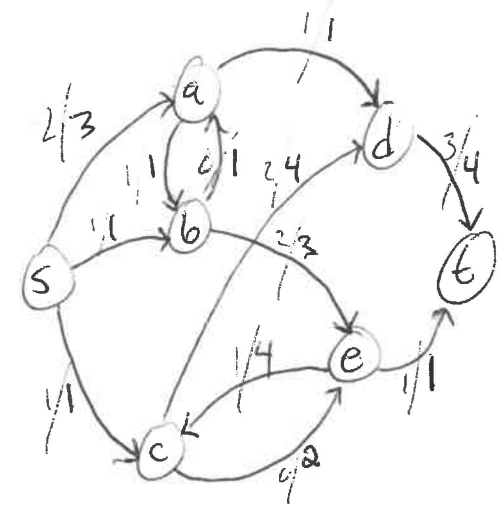
Lecture 12: Max-st-Flow

[Max-Flow is a very general algorithms problem that can be nevertheless solved efficiently.
The definition of the problem involves shipping stuff around a network, but it can be used to model & solve so many other unrelated-looking problems...]

a paradigm for ~~solving~~ finding optimal
- matchings in bip. graphs
- shortest paths in graphs with negative edge lens
- schedules with precedence constraints
- sports playoff scenarios...

[so much more...]



Input:
- Directed graph $G = (V, E)$
- "source" $s \in V$ (no in-edges)
  "target" $t \in V$ (no out-edges)
  (sink)
- "positive capacities" $c_e \in \cancel{\mathbb{R}} \to \mathbb{Z}$ on each edge $e \in E$

[~~You could use~~ If you have fractional caps, scale them all up to integers. There's something special about ints for this problem...]

You are shipping "stuff" (e.g., tons of cement) along edges (e.g. railroads). Capacities indicate max amount you can ship (e.g., per day). Fractional amts OK.

<u>Goal:</u> ship as much from s to t as possible,

subject to "<u>Flow constraint</u>": $\forall v \in V \setminus \{s,t\}$, incoming flow = outgoing flow

Key Facts: (1) Problem is efficiently solvable! :)
[[we'll see how]]

(2) If all caps $c(e)$ are integers,
$\exists$ an optimal solution where all flow amounts are integers

[[This is really nice; if "stuff" is tons of cement, you don't mind shipping fractional amounts, but if you're shipping people on flights you don't want to ship fractional people!!]]

(1) How efficient? Say capacities in $\{1, 2, \ldots, U\}$, n verts, m edges

'50s: "Ford-Fulkerson", $O(mnU)$, in fact $O(m \cdot F^*)$, F^* = max flow val. [[think $U = \text{poly}(n)$]]
[[we'll study]]
[[Kinda terrible. Even if $U=1$, it's quadratic, no good if $n \gg 10{,}000$ say. Also, if U is 2^{64} (why not? fits in a word), it's insane. Not even considered "polynomial time". Still, good to study, you learn a lot about flows. And okay for small instances.]]

'70s: $\tilde{O}(m^{1.5} \log U)$
[[much better. $m = 1$ mill is OK]]

'10s: $\tilde{O}(m^{4/3})$ if $U=1$

2022: CMU Prof. Richard Peng, +5
$\tilde{O}(m \log U)$!!!! [[Nearly linear time, arguably first progress in almost 50 years]]

[[We'll just do F.F. Alg.]]

Formally: seek $f : E \to \mathbb{Q}^{\geq 0}$ [[allowing fractional flow]] s.t.

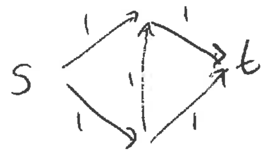
(i) $0 \leq f(e) \leq c(e)$ for all $e \in E$ [["cap. constraints"]]

(ii). $\underbrace{\sum_{e \text{ into } u} f(e)}_{\text{"}f^{in}(u)\text{"}} = \underbrace{\sum_{e \text{ out of } u} f(e)}_{\text{"}f^{out}(u)\text{"}}$ for all "internal" vtcs $u \in V \setminus \{s,t\}$

Maximize "$\underline{\text{value}}(f)$" $= f^{out}(s)$ $\quad$ $\underline{\text{exercise}}$: equals $f^{in}(t)$

Alg. idea 1: "Greedy"

Find a path $s \rightsquigarrow t$. Push as much flow as you can. Repeat.

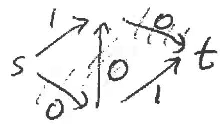
e.g.: 

Say you find path $P =$ 

Can push 1 unit of flow.

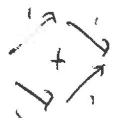
[[Granted, not the most obvious one to find, but it's an example…]]

Left with these capacities:



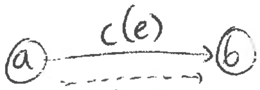
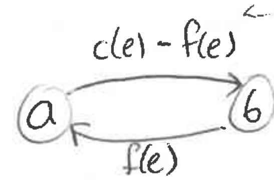
Stuck: no more paths.

But max-flow was 2! ☹



Idea 2: ~~[illegible]~~ Greedy, but allow "undoing".

Def^{n}: Given instance (G, c) & a flow f, the residual instance (G_f, c_f) formed as follows…

for each edge $e \in E$ "(a,b)"  replace with 

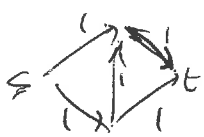
[[how much you can "undo"]]

[[new cap is how much more you could push]]

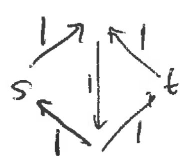
[[& merge parallel edges, summing capacities]]

[[# of edges at most doubles, so no big deal for running time analysis; won't worry about m vs. $2m$]]

Previous example:

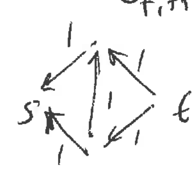
G 

$f =$ 

G_{f_1} 

[[Now there are still paths]]

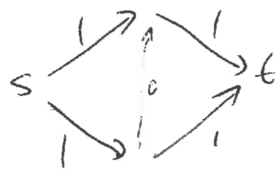
Find another path, f_2 ← "augmenting"

$G_{f_1+f_2}$ 

[[No more paths]].

Stop.

In this e.g., $f_1 + f_2 =$



is ~~max~~ flow.

[[Flow in one dir. cancels flow in opposite dir.]]

[[Not a coincidence, this alg works!]]

F.F. Alg on (G, c)

- Initialize $f(e) = 0 \;\; \forall e$.
- While $\exists$ $s \rightsquigarrow t$ path in residual graph G_f,

  Let P be any such path [[simple, no vtx repeated]] [[along edges with $c_f > 0$]]

  Let $b = \min\{c_f(e) : e \in P\}$ [["bottleneck"]]

  For each $e \in P$:

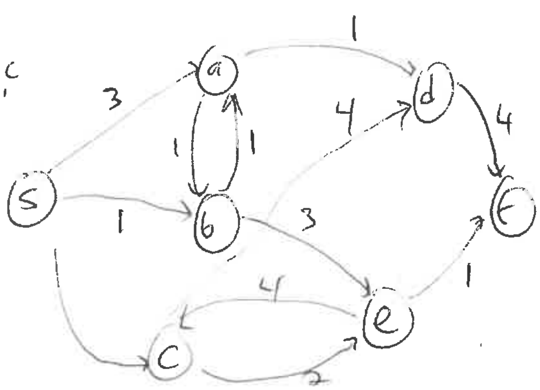
  $f(e)$ += b

  // this creates a new residual inst. G_f, c_f.

Running time:
$O(m)$ per iter (BFS)
iters $\leq$ value of max flow, F^*,
because flow goes up +1 each iter.
Time: $O(F^* m)$

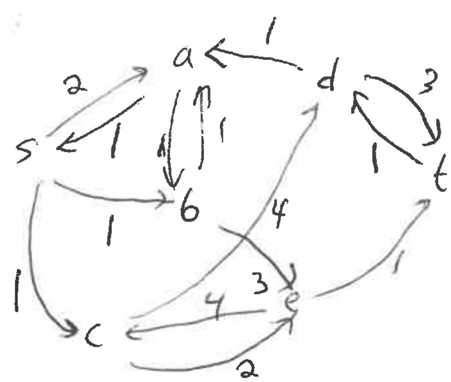
e.g.

G, c



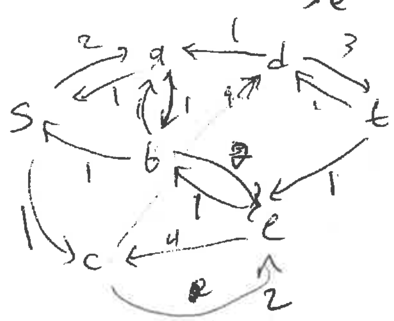
Say $f_1 = s \to a \to d \to t$. Bottleneck is 1, push 1 unit. (+1)

G_f, c_f

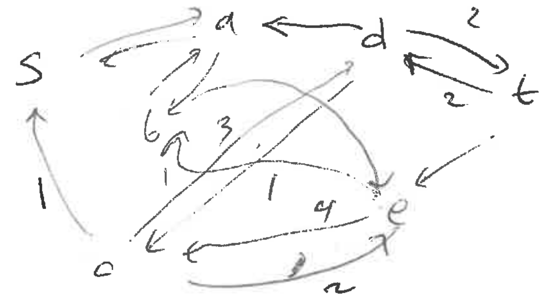


Say $f_2 = s \to b \to e \to t$. Bottleneck is 1. (+1)

New $G_{f_1 + f_2}$:



$f_3 = s \to c \to d \to t$ Bottleneck 1 (+1)



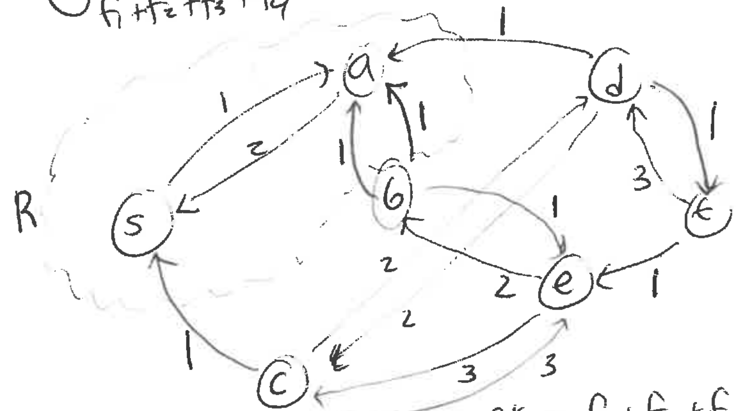
$f_4 = s \to a \to b \to e \to c \to d \to t$

bottleneck 1

total 4 so far

[we know that's max, I told you so]

$G_{f_1+f_2+f_3+f_4}$



Termination: No more $s \rightsquigarrow t$ paths. Final flow $f^* = f_1+f_2+f_3+f_4$.

Say, let $R = \{ v : \text{reachable from } s \} = \{s, a\} \not\ni t$.

Claim: in <u>original</u> G, each edge (u,v) <u>out of R</u> $(u \in R, v \notin R)$ must be <u>saturated</u> by f^* (flow = cap).

Else G_{f^*} would have some capacity on (u,v), and so v would be reachable from s too.

Claim: in <u>original</u> G, each edge $\underset{\notin R}{(v},\underset{\in R}{w)}$ <u>into R</u> must be <u>unused</u> by f^* (flow = 0).

Else G_{f^*} would have some capacity on (w,v).

$$\therefore\ f^*_{out}(R) - f^*_{in}(R) = \sum_{(u,v) \text{ out of } R} f^*(u,v) - \sum_{(u,v) \text{ into } R} f^*(v,u)$$

"Net flow of f^* out of R" (def)

$$= \sum_{(u,v) \text{ out of } R} c(u,v) - \sum_{(u,w) \text{ into } R} 0 \overset{\text{def.}}{=} \text{"total capacity of the } R \to \overline{R} \text{ "cut""}$$

Summary: At end of F.F., you get a set $R \subseteq V$, [nodes reachable from s in G_{f^*}] with $s \in R$, $t \notin R$ (this is called an "s–t cut")

And net flow of f^* out of R = cap($R \to \overline{R}$ cut).

(♡)

This is amazing. Why?

(i) Suppose C is <u>any</u> s-t cut $(s \in C,\ t \notin C)$

Then it's easy to see: max-flow $\leq cap(C \to \overline{C})$.

[[The total amount of capacity getting out of C is the most you could ever ship from s to t; C is like one <u>possible</u> bottleneck.]]

(2) And if C is any cut, f is <u>any</u> flow,

value(f) = net flow of f out of C = $f_{out}(C) - f_{in}(C)$

[[just because flow can't get stuck anywhere. net flow out of s = net flow out of $\{s,a\}$, out of $\{s,a,c\}$, etc. by flow conservation.]]

Now look at (♡). (2) $\Rightarrow$ LHS = value(f^*)

(i) $\Rightarrow$ RHS $\geq$ max-flow.

So value$(f^*) \geq$ max-flow $\Rightarrow$ must <u>equal</u> max-flow!

☺ [[F.F. finds maximum flows]]

Also finds R with <u>minimum</u> possible cut value

[[famous]]

Theorem: Max s-t Flow value = Min s-t Cut value.

[[And F.F. finds the max flow & the Min-Cut in time $O(m \cdot Val(f^*))$. Great if you have reason to believe $val(f^*)$ not too large...]]