Announcements

- HW due (written) next Wednesday in class
Network Flow

- A directed graph $G = (V, E)$
- Each edge has a maximum capacity $c_e$
- Two special vertices $s, t \in V$
  - $s$ is the source
  - $t$ is the sink

Note: $s$ has no incoming edges,
$t$ has no outgoing edges.
Formally:

A flow is a function \( f: E \rightarrow \mathbb{R}^+ \) (some amount sent along each edge) such that:

- capacity constraint: \( \forall e \in E, \ 0 \leq f(e) \leq c_e \)

- conservation constraint: \( \forall v \in V, \ \text{if } v \neq s \text{ or } t, \)
  \[ \sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e) \]

\( f^{\text{in}}(v) + f^{\text{out}}(v) \)
So: graph
A flow in this graph:

An \((s, t)\)-flow with value 10. Each edge is labeled with its flow/capacity.
\textbf{Def.} An s-t cut is a partition of \( V \) into 2 sets \( (S, T) \) with \( s \in S \), \( t \in T \).

The capacity of a cut \( c(S, T) = \sum_{e \text{ out of } S} c_e \).
Our algorithm (in pictures)

Consider some flow:

Form the residual graph $G_f$ (of $G$ with respect to flow $f$):
Find a path in this residual graph, and let $F =$ bottleneck edge.

Consider the edges in this path, and update the original flow.
path in $G_f$:

\[
\begin{align*}
\text{new flow: (in } G)\end{align*}
\]

\[
f'(u \rightarrow v) = \begin{cases} 
  f(u \rightarrow v) + F & \text{if } u \rightarrow v \text{ is in the augmenting path} \\
  f(u \rightarrow v) - F & \text{if } v \rightarrow u \text{ is in the augmenting path} \\
  f(u \rightarrow v) & \text{otherwise}
\end{cases}
\]
Pseudo code

Max flow \((G)\):
\[
\begin{align*}
& f(e) \leftarrow 0 \quad \forall e \in E \\
& G_f \leftarrow G
\end{align*}
\]

while there is an s-t path in \(G_f\)
\[
\begin{align*}
& \text{Let } P < \text{s-t path in } G_f \\
& f' = \text{Augment}(f, P) \quad \leftarrow \text{update flow using path } P \text{ in } G_f \\
& f \leftarrow f' \\
& \text{Update } G_f \\
& \text{return } f
\end{align*}
\]

Need to show that this algorithm gives maximum flow
Strategy: 2 things

1. Theorem: let $f$ be any $s$-$t$ flow, and $(S, T)$ any $s$-$t$ cut.

   Then $v(f) = c(S, T)$

2. Given a flow $f$ where there is no $s$-$t$ path in $G_f$, we can find a cut $(S^*, T^*)$ with:

   $v(f) = c(S^*, T^*)$.

   max flow  min cut
First, a lemma:

**Lemma:** Let $f$ be any s-t flow, and $(S, T)$ any S-T cut. Then $v(F) = f^{out}(S) - f^{in}(S)$.

**Proof:** By definition: $v(f) = f^{out}(S)$ also $f^{in}(S) = 0$

So: $v(F) = f^{out}(S) - f^{in}(S)$

For any other $v \in S$, $f^{out}(v) = f^{in}(v)$

So $v(F) = \sum_{v \in S} (f^{out}(v) - f^{in}(v))$
\[ v(f) = \sum_{v \in S} (f^{out}(v) - f^{in}(v)) \]

Think about edges in \( S \).
Any edge with 2 endpoints in \( S \) appears twice in sum.
Any edge out of \( S \) appears once, in a \( f^{out} \) term.
Any edge into \( S \) appears once, in a \( f^{in} \) term.

So:
\[ \sum_{v \in S} (f^{out}(v) - f^{in}(v)) = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in } S} f(e) \]
\[ = f^{out}(S) - f^{in}(S) \]
**Thm:** Let $f$ be any s-t flow, and $(S, T)$ any s-t cut.

Then $\nu(f) \leq c(S, T)$

**pf:**

$\nu(f) = f^{out}(S) - f^{in}(S)$ (by lemma)

$\leq f^{out}(S)$

$\leq \sum_{e \text{ out of } S} c(e)$

$= c(S, T)$
Thm: Given a flow $f$ where there is no $s$-$t$ path in $G_f$, we can find a cut $(S^*, T^*)$ with:
$v(f) = c(S^*, T^*)$.

pf: Consider $G_f$.
No $s$-$t$ path, so let $S = \{ v \in V \mid G_f \text{ has an } s \text{ to } v \text{ path} \}$
(Note: $t \notin S$)

let $T = V - S$

Diagram:

$G_f$

$S$

$T$

$s$

$t$
Consider $e \in G$. 

**PF (cont.)** Consider $e \in G$ going from $S$ to $T$, $e = (u, v)$. 
If $v \in T$, so $f(e) = C_G$ (or else $v$ would be reachable in $G_f$).

Consider $e' \in G$ from $T$ to $S$, $e' = (u', v')$ where $u' \notin S$, so reversed edge.

$\Rightarrow f(e') = 0$.

$\Rightarrow v(f) = f^{out}(S) - f^{in}(S) = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ into } S} f(e) = \sum_{e \text{ out of } S} C_e - \sum_{e \text{ into } S} 0 = C(S, T)$.
Runtime: (A first try)

- In each loop, flow increases by at least 1.
- Each time in loop takes $O(m+n)$

$\Rightarrow O(m|f|)$

$value of max flow
Ideas for improving:

- Choose path with largest bottleneck edge
- Choose path with min. # of edges

both lead to "good" poly...