QUICK REVIEW OF LAST TIME

Recall: Max-flow Min-cut Theorem

Theorem 3: Every max s-t flow has value equal to the capacity of a min s-t cut

We give an algorithmic proof of this theorem
- Showing that one algorithm solves both max-flow and min-cut at the same time)

FORD-FULKERSON METHOD

Algorithm development (mixed together with proof of max-flow min-cut theorem)

Residual Graph
- A residual graph $R_f$ is defined for a given flow $f$ and graph $G$
- $R_f$ has the same vertices as $G$
- For each edge $e = uv$ in $G$,
  - If $f(e) < c(e)$, then $R_f$ contains a forward edge $(u, v)$ with the remaining capacity $c(e) - f(e)$
  - If $f(e) > 0$, then $R_f$ contains a backwards edge $(v, u)$ with capacity $f(e)$ representing flow that could be "pushed back"

Ford-Fulkerson Method

Can undo previous decisions to improve the flow
Can effectively “push back” some flow using an augmenting path through a residual graph

Residual graph for this flow

Greedy flow $f$

Forward edge: remaining capacity

Backwards edge: can undo flow

Improved flow

"Augmenting path" (path that results in an improvement of the flow)
ANOTHER EXAMPLE RESIDUAL GRAPH
- Recall: for each edge $e = uv$ in $G$,
  - If $f(e) < c(e)$, then $R_f$ contains a forward edge $(u, v)$ with the remaining capacity $c(e) - f(e)$
  - If $f(e) > 0$, then $R_f$ contains a backwards edge $(v, u)$ with capacity $f(e)$ representing flow that could be "pushed back"

CONTINUING WITH NEW MATERIAL

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow

FORD-FULKERSON METHOD
- Find a shortest path $P$ from $s$ to $t$ in the residual graph
  - If it improves the flow, we call it an augmenting path
  - And use it to update the flow
FORD-FULKERSON METHOD

Find a shortest path $P$ from $s$ to $t$ in the residual graph
- If it improves the flow, we call it an augmenting path
- And use it to update the flow

Front page:

IMPROVING A FLOW $f$
GIVEN AN AUGMENTING PATH $P$

- An augmenting path w.r.t $f$ is a simple $s$-$t$ path in $R_f$
- Let $P$ be an augmenting path w.r.t $f$
- Let bottleneck($f$, $P$) be the minimum capacity of an edge in $P$
- We show this subroutine augment($f$, $P$) always improves the value of flow $f$

LEMMA 4: AUGMENT() IMPROVES FLOW $f$

Let $f$ be a flow in $G$ with $f^\text{in}(s) = 0$, and $P$ be an augmenting path w.r.t $f$
- Let $f'$ be the resulting flow after running augment($f$, $P$)
- Then $f'$ is a flow with $\text{value}(f') = \text{value}(f) + \text{bottleneck}(f, P)$
- That is, augment($f$, $P$) increases the flow by bottleneck($f$, $P$)

PROOF

- Claim: augment($f$, $P$) increases the flow by bottleneck($f$, $P$)
- How about conservation of flow?
  - Consider how the flow into and out of each vertex $u \notin \{s, t\}$ changes as a result of running augment($f$, $P$)
  - We show the change in $f^\text{in}(u)$ is the same as the change in $f^\text{out}(u)$
  - There are 4 cases, depending on whether the edges entering/leaving $u$ are forward or backward edges
**Case 3: forward / backwards**

Let bottleneck\((f, P) = b\)

![Diagram](image)

**Case 4: backwards / forwards**

is similar.

![Diagram](image)

**SHOWING \( f^{in}(s) = 0 \)**

- Last step in showing \( f' \) is a flow
  - Prove: \( s \) still has no flow into it
- Since \( f \) is a flow, \( f^{in}(s) = 0 \)
  - To get \( f^{in}(s) > 0 \), an augmenting path must include an edge into \( s \)
  - But then an augmenting path starts at \( s \), then returns to \( s \), forming a cycle -- contradiction!

**FINISHING LEMMA 4: AUGMENT() IMPROVES FLOW**

- Finally we argue \( \text{value}(f') = \text{value}(f) + \text{bottleneck}(f, P) \)
  - \( f \) and \( f' \) are flows, so \( \text{value}(f') = f^{\text{out}}(s) \) and \( \text{value}(f) = f^{\text{in}}(s) \)
  - We thus show \( f^{\text{out}}(s) = f^{\text{in}}(s) + \text{bottleneck}(f, P) \)
- The augmenting path \( P \) is a simple path (leaving \( s \) exactly once)
  - And there is no flow into \( s \), so the edge \( e \in P \) leaving \( s \) is a forward edge
  - This means \( \text{augment}(f, P) \) adds \( \text{bottleneck}(f, P) \) to \( f(e) \)
- So \( f'^{\text{out}}(s) = f'^{\text{in}}(s) + \text{bottleneck}(f, P) \)

**FORD-FULKERSON METHOD**

- By Lemma 4, starting from any flow \( f \)
  - if we can find an augmenting path \( P \) w.r.t \( f \) in \( R_f \),
  - then we can use \( \text{augment}(f, P) \) to improve our flow
- Ford-Fulkerson does this repeatedly starting from an empty flow

**PROOF STRATEGY**

Claim: when there is no augmenting path, there is a cut with capacity equal to the value of the current flow.

- Proving this will simultaneously
  - prove the max-flow min-cut theorem,
  - prove correctness of the Ford-Fulkerson method,
  - solve the max flow problem, and
  - solve the min cut problem

**MAX-FLOW MIN-CUT THEOREM PROOF**
PROVING MAX FLOW = MIN CUT

Two directions:
- max flow ≤ min cut and max flow ≥ min cut

We actually proved the ≤ direction already (Lemma 2 last time) when discussing upper bounds for max flow.

If remains to prove the ≥ direction:

PROVING THE PROPOSITION

- Since there is no s-t path in Rf, there is a subset S of vertices with s ∈ S, t ∉ S such that S has no outgoing edges in Rf.
- What does this statement look like?

PROVING THE PROPOSITION

- Claim: c^in(S) = value(f)
- Consider two types of edges. Type 1:
  - u ∈ entering S in G
  - (u, v) ∈ E(S) in G, u ∈ S, v ∈ S
  - Since S has no outgoing edge in Rf, we know there is no edge (u, v) in Rf
  - This implies f(u) = 0, as otherwise u would be a backwards edge in Rf

PROVING THE PROPOSITION

- We just showed
  - For edge (u, v) directed out of S, f(u) = c(u)
  - For edge (u, v) directed into S, f(u) = 0
  - So c^in(S) = c^in(S) - f(u) = c^in(S) - 0 = c^in(S)
  - This proves the proposition. I.e., given flow f, if there are no s-t paths in Rf, then there is a cut matching the flow.
TIME COMPLEXITY
of the Ford-Fulkerson method

RUNTIME OF FORD-FULKERSON
- Assume we find any arbitrary augmenting path \( P \),
  using any technique, in \( O(n + m) \) time
- Then every time \( \text{augment}(f, P) \) is run,
  we know only that the flow increases
- If capacities are integers, the increase is at least 1
- In this case, if max flow is \( k \) then runtime is \( O(k(n + m)) \)
  - For max flow we assume a connected graph, so this is \( O(km) \)
  - Very bad if \( k \) is large

If capacities are real (and in particular some are irrational),
this may never terminate!

RUNTIME OF FORD-FULKERSON
- Depends on the implementation
- How do we find an augmenting path?
- How many times do we need to augment before we terminate?

WORST CASE FOR THIS APPROACH

EDMONDS-KARP APPROACH
- Use BFS to find a shortest path (in terms of number of edges)
  and use that as an augmenting path
- It turns out this always terminates after \( O(nm) \) augmenting paths
  (even with real capacities)
- BFS takes \( O(n + m) \) time; \( O(n) \) since the graph is connected
- So total runtime is \( O(nm^2) \)

In 2022, researchers found a \textit{near-linear} time algorithm
which leverages techniques from convex optimisation
and sophisticated data structures.