Finish the Proof

- Last week, we defined the maximum flow and minimum cut problems, and then we saw most of a proof that for any input to the two problems, the value of the maximum flow equals the capacity of the minimum cut.
- For the proof, we started with an arbitrary feasible (s, t)-flow \( f : E \rightarrow \mathbb{R}_{\geq 0} \) and built the residual graph \( G_f \) which contained all edges with positive residual capacity according to \( c_f : V \times V \rightarrow \mathbb{R}_{\geq 0} \):

\[
c_f(u \rightarrow v) = \begin{cases} 
  c(u \rightarrow v) - f(u \rightarrow v) & \text{if } u \rightarrow v \in E \\
  f(v \rightarrow u) & \text{if } v \rightarrow u \in E \\
  0 & \text{otherwise}
\end{cases}
\]

- If \( G_f \) contained an augmenting path \( P \) from \( s \) to \( t \), we pushed flow along \( P \), creating a new flow of higher value. The amount we pushed was equal to the minimum residual capacity of edges along \( P \).

- Now, suppose there is no path from source \( s \) to target \( t \) in the residual graph \( G_f \).
  - Let \( S \) be the vertices reachable from \( s \) in \( G_f \), and let \( T = V \setminus S \).
  - Partition \( (S, T) \) is an \( (s, t) \)-cut, and for every \( u \) in \( S \) and \( v \) in \( T \):
    - If \( u \leftrightarrow v \) in \( E \), then \( 0 = c_f(u \leftrightarrow v) = c(u \leftrightarrow v) - f(u \leftrightarrow v) \)
      - i.e., \( f(u \leftrightarrow v) = c(u \leftrightarrow v) \); the edge is saturated.
    - If \( v \leftrightarrow u \) in \( E \), then \( 0 = c_f(u \leftrightarrow v) = f(v \leftrightarrow u) \)
      - i.e., the edge is avoided.
  - We see \( f \) saturates every edge from \( S \) to \( T \) and avoids every edge from \( T \) to \( S \).
  - And if you remember from earlier, that means is a maximum flow, \( (S, T) \) is a minimum cut, and \( |f| = ||S, T|| \).
To summarize, exactly one of these two cases holds:

1. There is an augmenting path from s to t in the residual graph. We can strictly increase the value of f by pushing along that path, so f was not a maximum flow to begin with.
2. There is no path from s to t in the residual graph. f is already a maximum flow with value equal to the capacity of the minimum cut.

In particular, the only situation where f is a maximum flow shows us its value is equal to the capacity of the minimum cut.

**Ford-Fulkerson Augmenting Path Algorithm**

- You can turn this proof into an algorithm for computing a maximum flow usually referred to as the Ford-Fulkerson Augmenting Path Algorithm.
- In short: start with all flow values equal to 0 and repeatedly push flow along augmenting paths until you can’t find one anymore.
- But will this process actually result in a maximum flow?
- First, let’s assume all the capacities are integers. This has a few repercussions.
  - The initial flow is all integers since 0 is an integer.
  - If we assume inductively that f is all integers, then all the residual capacities are integers.
  - Meaning any amount of flow we push is always a positive integer.
  - Meaning any new flow is all integers and its value is at least 1 greater than the old flow.
- So if we let f^* denote the maximum flow, we do at most |f^*| augmentations and f^* is all integers.
- We can build and search the residual graph in O(E) time, so these |f^*| augmentations take O(E |f^*|) time total.
- But there are two issues with this analysis.
- At the beginning of the semester, I talked about different classes of running times. Our usual goal was to find algorithms with running time polynomial in the input size. For example, we could compute edit distance in O(n^2) time or all pairs shortest paths in O(V^3) time.
- O(E |f^*|) is what we call a *pseudo-polynomial time* algorithm. It runs in time polynomial in |E| and |f^*|, but |f^*| may not be polynomial in the input size.
- In particular, consider the example below: we might try pushing along the 1 edge or its reverse every augmentation, leading to a running time of Theta(X). But X can be written using O(log X) bits; the running time is exponential in the input size!
Ford-Fulkerson is often efficient in practice, though, or in situations where you can guarantee $|f^*|$ is small.

The other issue with this analysis is that we’re assuming the capacities are integers. But we defined flows and capacities using real numbers.

You can set up examples with real number capacities where every augmentation gets smaller and smaller and smaller. You always get higher value flows, but you never get to a maximum flow. There’s not even a guarantee that you’ll approach the maximum flow value in the limit.

Of course, computers don’t actually store real numbers, but you should still be nervous. If your floating point additions or comparisons start doing rounding, you may actually enter an infinite loop where you never make real progress on increasing the flow value!

But here’s the trick. We get to choose which augmenting paths to use. If we pick carefully, maybe the algorithm will run faster.

Both of the following algorithms were discovered by Edmonds and Karp (and others) in the 1970s.

**Edmonds-Karp 1: Fattest Augmenting Paths**

- Edmonds-Karp: Choose the augmenting path with the largest bottleneck value.
- You can find this path using a variant of the Prim-Jarník minimum spanning tree algorithm: Build a spanning tree from $s$ in the residual graph, repeatedly adding the edge of largest residual capacity that leaves the tree.
- You can pull edges out of a priority queue implemented with, say, a binary heap, so $O(\log V)$ time per edge or $O(E \log V)$ time total to find each augmenting path.

So how many augmenting paths are there?

- Let $f$ be the current flow and $f'$ be the maximum flow in the current residual graph $G_f$. In other words, $f + f'$ is the maximum flow in $G$.
- Let $e$ be the bottleneck edge in the current iteration, so we’re about to push $c_f(e)$ units of flow.
- There is a decomposition of $f'$ that includes at most $|E|$ path flows, so $c_f(e) \geq |f'| / |E|$.
- So pushing down the maximum-bottleneck path multiplies the residual maximum flow value by $(1 - 1 / |E|)$ or less.
- After $|E| \ln |f^*|$ iterations, the residual value of the maximum flow is at most

$$|f^*| \cdot (1 - 1/E)^{E\ln|f^*|} < |f^*|e^{-\ln|f^*|} = 1.$$
• In other words, we can’t do another augmentation after $|E| \ln |f^*|$ iterations if the capacities are integers, because there won’t be an integral amount of flow left to push.
• The total running time assuming integer capacities is $O(E^2 \log V \log |f^*|)$.
• This running time is polynomial in the problem size, but it still relies on integer capacities.

Edmonds-Karp 2: Shortest Augmenting Paths

• Edmonds-Karp (again): Choose an augmenting path with the smallest number of edges.
• Can be found in $O(E)$ time by running a breadth-first search in the residual graph.
• Now to bound the number of iterations.
• Let $f_i$ be the flow after $i$ iterations, and $G_i = G_{f_i}$. Flow $f_0$ is zero everywhere and $G_0 = G$.
• Let level$_i(v)$ be the unweighted shortest path distance from $s$ to $v$ in $G_i$.

**Lemma:** level$_i(v)$ $\geq$ level$_{i-1}(v)$ for all vertices $v$ and non-negative integers $i$.

• We’ll do induction on level$_i(v)$.
• level$_i(s) = 0 =$ level$_{i-1}(s)$. Check.
• If we cannot reach $v$ from $s$, then level$_i(v)$ = infty $\geq$ level$_{i-1}(v)$. Check.
• Otherwise, let $s \rightarrow \ldots \rightarrow u \rightarrow v$ be a shortest path to $v$ in $G_i$.
• level$_i(v) = \text{level}_{i-1}(u) + 1$, so the induction hypothesis shows level$_i(u) \geq \text{level}_{i-1}(u)$.
• If $u \rightarrow v$ is in $G_{i-1}$, then level$_i(u) + 1 \geq \text{level}_{i-1}(v)$.
• If $u \rightarrow v$ is not in $G_{i-1}$, then we must have pushed along $v \rightarrow u$ to create residual capacity in $u \rightarrow v$. Meaning $v \rightarrow u$ was on the shortest $s$ to $t$ path. So level$_{i-1}(u) + 1 > \text{level}_{i-1}(u) - 1 = \text{level}_{i-1}(v)$.
• Either way, level$_i(v) = \text{level}_{i-1}(u) + 1 \geq \text{level}_{i-1}(u) + 1 \geq \text{level}_{i-1}(v)$.

**Lemma:** Any edge $u \rightarrow v$ disappears from the residual graph at most $|V| / 2$ times.

• Suppose $u \rightarrow v$ is in $G_i$ and $G_{i+1}$ but not in $G_{i+1}$, ..., $G_j$ for some $i < j$.
• $u \rightarrow v$ must be in the $i$th augmenting path, so level$_i(v) = \text{level}_{i}(u) + 1$.
• and $v \rightarrow u$ must be in the $j$th augmenting path, so level$_j(u) = \text{level}_{j}(v) + 1$.
• So, level$_i(u) = \text{level}_{i-1}(v) + 1 \geq \text{level}_{i-1}(u) + 1 = \text{level}_{i-1}(v) + 2$.
• So the distance from $s$ to $u$ increased by 2 between the disappearance and reappearance of $u \rightarrow v$. Every level is less than $|V|$ or infinite (if there is no path to $u$), so an edge can disappear at most $|V| / 2$ times.

• There are $2|E|$ possible residual edges so $|E| |V|$ disappearances total. Each augmentation makes its bottleneck edge disappear, so there are at most $|E| |V|$ iterations.
• The total running time is $O(VE^2)$.
• And this running time is correct even for arbitrary non-negative real number edge capacities.
• A variation on this idea was independently proposed by Dinitz in 1970. His algorithm was more complicated, but it runs in only $O(V^2 E)$ time.
And there have been many more algorithms discovered since these. Some rely on fancy data structures. Some use methods other than augmenting paths.

Building upon decades of more or less steady progress from several researchers, Orlin in 2012 described an algorithm that runs in only O(VE) time.

Very few people understand this algorithm, and I am not one of them, so it’s well beyond the scope of this class.

But for the purposes of doing homework or exams, you should feel free to cite it.

Orlin [2012]: Maximum flows and minimum cuts can be computed in O(VE) time.