1 Intro

Given:
- Directed graph $G$
- $m$ edges, $n$ vertices
- source $s$, sink $t$
- a capacity for each edge, denoted by $c(u, v)$ for edge $e = (u, v)$

We also define $f(e) = f(u, v)$ to be the flow on edge $(u, v)$, with constraints:
- $0 \leq f(u, v) \leq c(u, v)$ (the flow is limited by the capacity)
- $\forall \ v \in V : v \neq s, v \neq t \rightarrow \Sigma f(v, w) = \Sigma f(u, v)$ (conversation of flow in and out, except at source and sink)

We want to find the maximum $s - t$ flow, $F$: that is, we want the maximum flow from source $s$ to sink $t$, given the above constraints.

2 Ford-Fulkerson Algorithm

A greedy algorithm that calculates the max $s - t$ flow in $O(mF)$ time.

Definitions:
- an augmenting path is a path of edges from $s$ to $t$, each of which still has some remaining flow (that is, $c(u, v) - f(u, v) > 0$)
- the residual graph $G_f$ is a graph specified by flow $f$ such that:
  - edge $e = (u, v)$ in $G_f$ has capacity $c(u, v) - f(u, v)$
  - edge $e' = (v, u)$ in $G_f$ has capacity $f(u, v)$

Starting with a flow of 0 (that is, $f(e) = 0 \ \forall \ e \in E$), we repeat the following:

1. Find an augmenting path $p$ from $s$ to $t$ in the residual graph. If none exists, exit.
2. Add $p$ to the flow $f$.
3. Update the residual graph.
4. Repeat, from step 1.

If there does not exist an augmenting path in the residual graph, then we cannot add any more flow from $s$ to $t$, and we have found the maximum flow.
Runtime analysis:

- $O(m)$ to find the augmenting path (using breadth-first or depth-first search, for example)
- $O(m)$ to update flow $f$ with path $p$
- $O(m)$ to update the residual graph

so it requires $O(m)$ per iteration, and each iteration adds at least a flow of 1 to the $s - t$ flow. Thus, we require at most $O(F)$ iterations, where $F$ is the maximum $s - t$ flow.

(Show example that requires $O(F)$ iterations.)

How do we improve? Pick the augmenting path ‘smartly’!

3 Edmonds-Karp Algorithm 2

The algorithm follows the steps of the Ford-Fulkerson algorithm, but picks the shortest augmenting path, rather than an arbitrary one.

Analysis:

- Let us rearrange the graph $G$ into levels, where all vertices in level $i$ have minimum distance from $s$ equal to $i$.
- Let $d$ be the current minimum distance from $s$ to $t$.
- Note that, when we find a path $p$ (and update the residual graph by removing it and adding the reverse edges), we never create edges going forward.
- Thus, the distance $d$ never decreases.
- Also note that, when we find a path, one or more edges have the limiting capacity, and when we push all flow along the path, we fill these edges.
- Thus, we ‘remove’ at least one edge from the residual graph each time.

We find augmenting paths of length $d$ until no such path exists (and no paths of lesser length exist, by the definition of $d$). If no such path exists, then the distance between $s$ and $t$ must have changed; as specified, it cannot have dropped, so it must have increased. It must have increased by at least 1, and the distance can be at most $n$. Thus:

- $O(m)$ time to find an augmenting path (using breadth-first search), add it to the flow, and update the residual graph
- $O(m)$ edges filled and removed before a path of length $d$ no longer exists
- $O(n)$ times that $d$ can increase

yielding a total runtime of $O(nm^2)$. 