**15-451/651 Algorithm Design & Analysis, Spring 2024**

# Extra Review Problems

## Network Flows II

1. **(More bounds for unit-capacity networks)** Recall that a *unit-capacity* network is one where every edge has capacity one. In lecture we proved that we can find a max flow in such a network in $O(\sqrt{m})$ blocking flow computations. Suppose we are given a *simple* unit-capacity network (where simple means there are no duplicate edges between the same pair of vertices). We will show that such a network requires at most $n^{\frac{2}{3}}$ blocking flow computations.

   (a) Consider the network after we have found $k$ blocking flows. Prove that in the layered graph, at most half of the layers can have at least $\frac{2n}{k}$ vertices.

   (b) Using Part (a), prove that there exists a cut in the residual network of capacity at most $O\left(\left(\frac{n}{k}\right)^2\right)$

   (c) Using Part (b), prove that the number of blocking flows required is at most

   $$O\left(k+\left(\frac{n}{k}\right)^2\right),$$

   for any $k$, then pick $k$ to show that at most $n^{\frac{2}{3}}$ blocking flows are required.

   (d) Deduce that on a simple unit-capacity network, Dinic's algorithm runs in

   $$O\left(m\min\left(\sqrt{m}, n^{\frac{2}{3}}\right)\right)$$

2. **(Another polynomial-time algorithm for max flow)** In lecture, we saw the Edmonds-Karp Shortest Augmenting Paths algorithm, which we proved had a polynomial running time. Here's another natural strategy for picking augmenting paths: pick the augmenting path with the *largest capacity* (remember that the capacity of an augmenting path is the lowest capacity edge on the path, so we are trying to maximize the minimum capacity edge).

   (a) There are several ways to find such a path. Describe an algorithm that runs in $O(m\log n)$ time.

   (b) Prove the following lemma: In a graph with maximum $s$-$t$ flow $F$, there exists a $s$-$t$ path with capacity at least $F/m$. Hint: What happens if you delete all edges with capacity less than $F/m$?

   (c) Prove that this strategy requires at most $O(m\log F)$ augmenting paths to find a maximum flow, assuming the capacities are integers. Hint: Part (b) shows that we remove a fraction of $1/m$ of the remaining flow at each iteration. You might then want to use the inequality $(1-1/m)^m < 1/e$.

   (d) Deduce that this algorithm finds a maximum flow in $O(m^2\log n\log F)$ time. Explain why this is a polynomial-time algorithm for the usual definition.