# CS 224 Advanced Algorithms — Fall 2014

## Problem Set 8

Due: 11:59pm, Wednesday, November 26th
Submit to: `cs224-f14-assignments@seas.harvard.edu`
Solution maximum page limit: 6 pages

See homework policy at `http://people.seas.harvard.edu/~minilek/cs224/hmwk.html`

Throughout this problem set, whenever a graph is discussed, $n$ is the number of vertices and $m$ is the number of edges. We also assume the graph is connected, so $m \geq n - 1$.

**Problem 1:** This problem shows how to combine scaling with blocking flows.

(a) (3 points) In class we showed that the Dinic's blocking flow algorithm takes time $O(mn^2)$ on capacitated graphs. Show that another valid time bound is $O(mn + nf^*)$, where $f^*$ is the value of the optimal flow. **Hint:** Find a different way of accounting for work done by advances and augments.

(b) (2 points) Show how to achieve an $O(mn \log U)$ time algorithm for max flow by combining blocking flow with scaling.

**You only need to do one of problems 2 and 3. If you do both, then your higher score among the two will count, and the other will be worth bonus points equal to your score on it divided by 3, rounded down.**

**Problem 2:** Recall the single source shortest paths problem: given a directed graph with edge weights, we would like to find all shortest path distances from some source vertex $s$ to all other vertices in the graph $G = (V, E)$. In this problem, assuming integer edge weights in $\{-C, \ldots, C\}$, we will design a scaling algorithm taking time $O(m\sqrt{n} \log C)$. In what follows, $c(u, v)$ denotes the cost of an edge $(u, v)$.

(a) (2 points) For a "price function" $p : V \to \mathbb{Z}$ define a reduced cost function $c_p(u, v) = c(u, v) + p(u) - p(v)$. Suppose we had an algorithm $\mathcal{A}$ running in time $T$ which, when given a graph $G$, either (1) finds a price function $p$ such that all edge reduced costs are nonnegative (we call such a price function "feasible"), or (2) correctly outputs that no such price function exists. Show how $\mathcal{A}$ implies an $O(m + n \log n + T)$ time algorithm for shortest paths with arbitrary (i.e. possibly negative) edge weights.

(b) (2 points) We now develop such an algorithm $\mathcal{A}$. Suppose we had an algorithm $\mathcal{B}$ which, given a graph with edge integer edge weights all at least $-1$, finds a feasible price function in time $T'$. Show we can then implement an algorithm $\mathcal{A}$ from part (a) with running time $T = O(T' \log C)$. **Hint:** Use scaling.

(c) (3 points) We now develop $\mathcal{B}$. We say vertex $v \in V$ is *bad* if there is a negative edge $(u, v) \in E$ entering it. Note if there are no bad vertices then the zero price function $p = 0$ is feasible. Give an algorithm $\mathcal{B}$ such that if there are $b$ bad vertices then in $O(mb)$ time either (1) it will be correctly reported that $G$ has a negative weight cycle, or (2) if no such cycle exists then a feasible price function will be output. Note, however, that $b$ can be $\Theta(n)$ in the worst case, and $O(mn)$ is already achieved via Bellman-Ford. **Hint:** Define a graph $G^- = (V, E^-)$ whose edges are $E^- = \{e \in E : c(e) \leq 0\}$. If any strongly connected component (SCC) of $G^-$ has a negative edge with both endpoints inside the component, then note $G$ has a negative cycle and we can halt. Otherwise, replace $G^-$ by a version where each SCC is contracted down to a single vertex (note SCC's can be found in $O(m)$ time); imagine also contracting the same vertices in $G$. Note now $G^-$ is acyclic, and $G$ has a negative cycle iff the original uncontracted version of $G$ did. Now, start with $p = 0$ and adjust prices to fix bad vertices one by one without creating new bad vertices in the process (reasoning about $G^-$ may be helpful for this).

(d) (2 points) The key to improving (c) is to fix not just one bad vertex in a time, but many simultaneously in $O(m)$ time. Suppose we had an algorithm $\mathcal{C}$ such that if there are $b$ bad vertices, it could in $O(m)$ time find a price function $p$ decreasing the number of bad vertices (according to $c_p$) to at most $b - \lceil \sqrt{b} \rceil$ (or conclude that $G$ has a negative weight cycle). Show that such a $\mathcal{C}$ implies an implementation of $\mathcal{B}$ in time $O(m\sqrt{n})$.

(e) (6 points) Show how to implement the algorithm $\mathcal{C}$ from part (d). **Hint:** Try to find either a long path or many paths of the same length in the SCC-contracted $G^-$, then handle the two cases separately. It might help to create a new source vertex $\alpha$ and connect it to every vertex in $G^-$ with a weight 0 edge, then partition $V$ according to shortest path distances from $\alpha$.

**Problem 3:** (15 points) In lecture we showed that the total runtime of $m$ operations on a link-cut tree using splay trees to implement auxiliary trees is $O((\log n) \cdot (m + \text{PCC}))$, where PCC is the total number of preferred child changes ove those $m$ operations. Show that in fact the stronger bound $O(m \log n + \text{PCC})$ holds. Given that we showed $\text{PCC} = O(m \log n)$ in class, this implies an $O(\log n)$ amortized bound per operation. For this problem, we only require you to consider the case when all $m$ operations are either link, cut, or access. **Hint:** Remember our main tool for analyzing splay trees: Lemma 1 from Lecture 7 (choose the weight function $w(\cdot)$ appropriately).

**Problem 4:** (10 points) Give an $O(m(m + n \log n) \log U)$ time scaling algorithm for min cost max flow in graphs with capacities in $\{1, \ldots, U\}$. **Hint:** Initially all edges have capacity 0 and $p = 0$ is feasible. As capacity bits are shifted in, $p$ must be fixed to stay feasible.

**Problem 5:** (1 point) How much time did you spend on this problem set? If you can remember the breakdown, please report this per problem. (sum of time spent solving problem and typing up your solution)