CS 473: Algorithms, Fall 2010
HW 8 (due Tuesday, November 2)

This homework contains four problems. **Read the instructions for submitting homework on the course webpage**. In particular, *make sure* that you write the solutions for the problems on separate sheets of paper; the sheets for each problem should be stapled together. Write your name and netid on each sheet.

Collaboration Policy: For this home work, Problems 1-3 can be worked in groups of up to 3 students each.

Problem 0 should be answered in Compass as part of the assessment HW8-Online and should be done individually.

0. (10 pts) HW8-Online on Compass.

1. (35 pts) Suppose we want to write an efficient function Shuffle(n) that returns a permutation of the set $\{1, 2, ..., n\}$ chosen uniformly at random.

   (a) Prove that the following algorithm is *not* correct.[Hint: Consider $n = 3$.]

   Algorithm *Shuffle*(n)
   1. **for** $i \leftarrow 1$ **to** n
   2. **do** $\pi[i] \leftarrow i$
   3. **for** $i \leftarrow 1$ **to** n
   4. **do** swap $\pi[i] \leftrightarrow \pi[\text{Random}(n)]$
   5. **return** $\pi[1 \ldots n]$

   (b) Prove that the following implementation of Shuffle(n) is correct. What is its expected running time?

   Algorithm *Shuffle*(n)
   1. **for** $i \leftarrow 1$ **to** n
   2. **do** $\pi[i] \leftarrow$ NULL
   3. **for** $i \leftarrow 1$ **to** n
   4. $j \leftarrow \text{Random}(n)$
   5. **while** $\pi[j] \neq$ NULL
   6. **do** $j \leftarrow \text{Random}(n)$
   7. $\pi[j] \leftarrow i$
   8. **return** $\pi[1 \ldots n]$

   (c) Describe and analyze an implementation of Shuffle(n) that runs in $O(n)$ time. (An algorithm that runs in $O(n)$ expected time is fine, but $O(n)$ *worst case* time is possible).

2. (25 pts) Your friends have written a very fast piece of maximum-flow code based on repeatedly finding augmenting paths. However, after you have looked at a bit of output from it, you realize that it's not always finding a flow of maximum value. The bug turns out to be pretty easy to find; your friends hadn't really gotten into the whole backward-edge thing when

writing the code, and so their implementation builds a variant of the residual graph that *only includes the forward edges.* In other words, it searches for $s-t$ paths in a graph $\tilde{G}_f$ consisting only of edges e for which $f(e) < c_e$, and it terminates when there is no augmenting path consisting entirely of such edges. We'll call this Forward-Edge-Only Algorithm. (Note that we do not try to prescribe how this algorithm chooses its forward-edge paths; it may choose them in any fashion it wants, provided that it terminates only when there are no forward-edge paths.)

It's hard to convince your friends they need to reimplement the code. In addition to its blazing speed, they claim, in fact, that it never returns a flow whose value is less than a fixed fraction of optimal. Do you believe this? The crux of their claim can be made precise in the following statement.

> There is an absolute constant $b > 1$ (independent of the particular input flow network), so that on every instance of Maximum-Flow problem, the Forward-Edge-Only Algorithm is guaranteed to find a flow of value at least $1/b$ times the maximum-flow value (regardless of how it chooses forward-edge paths).

Decide whether you think this algorithm is true or false, and give a proof of either the statement or its negation.

3. (30 pts) Given a flow network G with integer capacities you have computed a maximum flow f between s and t. However you have made a mistake in noting the capacity of an edge e.

   - (10 pts) Suppose you *under* estimated the capacity of e by $k > 0$ units. Show that you can compute the correct maximum flow in $O(km)$ time using the current flow f.
   - (20 pts) Do the same as above if you *over* estimated the capacity of e by $k > 0$ units. *Hint:* First assume that f is acyclic. How do you reduce flow on e?