CSE 521: Design and Analysis of Algorithms — Autumn 2006
Problem Set #3 — Instructor: Anna Karlin
Due on **November 7, 2006**

Turn-in: You can either bring the homework to class or email your homework to Roee or leave it in his mailbox on the first floor of the Allen Center (in the room with all the CSE grad student mailboxes).

Instructions: You are allowed to collaborate with fellow students taking the class in solving problem sets. You may also collaborate with one other classmate on writing up your solutions. If you do collaborate in any way, you must acknowledge for each problem the people you worked with on that problem.

The problems have been carefully chosen for their pedagogical value and hence might be similar or identical to those given out in past offerings of this course at UW, or similar courses at other schools. Using any pre-existing solutions from these sources, from the Web or other algorithms textbooks constitutes a violation of the academic integrity expected of you and is strictly prohibited.

Most of the problems require only one or two key ideas for their solution – spelling out these ideas should give you most of the credit for the problem even if you err in some finer details. So, make sure you clearly write down the main idea(s) behind your solution even if you could not figure out a complete solution.

A final **important** piece of advice: Begin work on the problem set early and don't wait till the deadline is only a few days away.

Readings: Kleinberg and Tardos: Chapter 7

Each problem is worth 10 points unless noted otherwise. All problem numbers refer to the Kleinberg-Tardos textbook.

1. Chapter 6, Problem 8
2. Chapter 6, Problem 23
3. Let Σ be a finite alphabet. For two strings $x, y \in \Sigma^*$, define the *Insert-Delete distance* between x and y, denoted $\mathrm{ID}(x, y)$, to be the minimum number of insertions and deletions needed to convert x to y. For example if $\Sigma = \{a, b, c\}$, $x = abbc$ and $y = bbac$, then $\mathrm{ID}(x, y) = 2$ (we delete the first a and insert an a before the last c).
   (a) Give a polynomial time algorithm that on input two strings x, y, computes the distance $\mathrm{ID}(x, y)$ as well as a sequence of $\mathrm{ID}(x, y)$ insert/delete operations that can be used to convert x to y.
   (b) Let us say that a string $p = p_1p_2 \dots p_k \in \Sigma^k$ is a substring of $z = z_1z_2 \dots z_n \in \Sigma^n$ if there exist a sequence of indices $1 \le i_1 < i_2 < \cdots < i_k \le n$ such that for all j, $1 \le j \le k$, we have $z_{i_j} = p_j$. The *longest common substring* (LCS) of two strings x, y is the largest sequence L such that L is a substring of both x and y. Likewise, the *shortest common superstring* (SCS) of two strings x, y is the smallest sequence L such that both x and y are substrings of L.

Design a polynomial time algorithm to find the LCS and SCS of two given strings. (<u>Hint</u>: It may be useful to figure out how, for two strings x, y, the quantities $\mathrm{ID}(x, y)$, length of $\mathrm{LCS}(x, y)$, and length of $\mathrm{SCS}(x, y)$ are related.)

4. State whether the following statements are True of False, and justify your answer.

   (a) Let $G = (V, E)$ be a directed graph. Let $a, b, c \in V$ be three distinct vertices such that in the graph G there exist k mutually edge-disjoint paths from a to b, as well as k mutually edge-disjoint paths from b to c. Then there also exist k mutually edge-disjoint paths between a and c.

   (b) Consider the following "Forward-Edge Only" algorithm for computing s-t flows. The algorithm runs in a sequence of augmentation steps till there is no s-t path in the residual graph, except that we use a variant of the residual graph that *only includes the forward edges*. In other words, the algorithm searches for s-t paths in a graph $\tilde{G}_f$ consisting only of edges e for which $f(e) < c(e)$, and terminates when there is no augmenting path consisting entirely of such edges. Note that we do not 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.

   Now to our claim: On every instance of the Maximum Flow problem, the forward-edge only algorithm returns a flow with value at least $1/4$ of the maximum-flow value (regardless of how it chooses it forward-edge paths).

5. Chapter 7, Problem 23

6. In this exercise, the goal is to develop a method to find augmenting paths that always computes the maximum flow using a number of augmentations that is independent of the edge capacities and depends only on the number of vertices and edges in the graph. This yields what is called a *strongly polynomial time* algorithm for computing maximum flow. The method is a very natural one: at each step, we perform a BFS on the residual graph from the source s till we reach t (if we never reach t, we stop the algorithm and output the flow), and augment flow along the s-t path in the BFS tree. In other words, we augment flow along the shortest path possible in each iteration. Let us call such an augmentation the BFS-augmentation.

   (a) Let f be a s-t flow on a network $G = (V, E)$ with source s and sink t. Suppose we perform a BFS-augmentation on f to compute a new flow f'. Prove that, for each $v \in V - \{s, t\}$, the shortest path distance $d_{f'}(s, v)$ from s to v in the residual graph $G_{f'}$ is not smaller than the shortest path distance $d_f(s, v)$ from s to v in G_f. (Here, we measure distance in residual graphs in terms of the number of hops, i.e., each edge in the residual graph has unit distance.)

   (b) Suppose we run the Ford-Fulkerson algorithm performing a BFS-augmentation at each augmentation step. Let $u, v \in V - \{s, t\}$. Prove that if the edge (u, v) is the bottleneck link on the chosen augmenting path in G_f when augmenting flow f and it is again, at a later point, the bottleneck link when augmenting flow f', then $d_{f'}(s, u) \geq d_f(s, u) + 2$.

   (c) Prove that the version of the Ford-Fulkerson algorithm where we perform a BFS-augmentation at each step terminates and outputs a maximum flow after only $O(|V||E|)$ augmentations, independent of the edge capacities.