Fundamental Algorithms, Section 003
Homework 9, Fall 22. Due date: Wednesday, November 16, 4:55pm on Gradescope

1. Recall Problem 4 from the previous homework:

Consider the problem of merging a sequence of sorted lists $L_1, L_2, \ldots, L_n$ of lengths $\ell_1, \ell_2, \ldots, \ell_n$. In this problem we are only allowed to merge adjacent lists: the allowed action is to remove two adjacent lists from the sequence and then replace them in the sequence with their merge. Thus, if after some sequence of merges we have obtained the shorter sequence $L'_1, L'_2, \ldots, L'_m$, and we merge $L'_i$ and $L'_{i+1}$ yielding $L''_i$, then the new sequence is $L'_1, \ldots L'_{i-1}, L''_i, L'_{i+2}, \ldots, L'_m$.

Suppose it costs $|L_a| + |L_b|$ to merge lists $L_a$ and $L_b$, for any pair of lists $L_a$ and $L_b$.

Give an iterative implementation of the dynamic programming solution for finding the cost of an optimum solution. (You may want to refer to the solution set for homework 8 for the recursive formulation of the solution.)

2. Consider the String Edit problem, with inputs $u = u_n u_{n-1} \ldots u_1$ and $v = v_m v_{m-1} \ldots v_1$. The problem is to find the smallest number of edits that changes $u$ to $v$: an edit is one of the following three actions: removal of a character, insertion of a character, replacement of a character somewhere in $u$ by another character. (See the lecture notes for a solution to the general string edit problem.)

Suppose we want to determine the minimum number of edits to change $u$ to $v$ so long as it is at most $k$; otherwise, the result is set to $k+1$ (but this is understood as meaning that at least $k+1$ edits would be needed). Let MinEdit$(i, j)$ be the minimum edit cost for changing $u_i u_{i-1} \ldots u_1$ to $v = v_j v_{j-1} \ldots v_1$.

a. Observe that MinEdit$(i, j) \geq |j - i|$.

b. By limiting which recursive calls are made, improve the running time of the efficient recursive algorithm to $O((n+m)k)$. (In fact, a bound of $O(\min\{n, m\} \cdot k)$ can be achieved.) It suffices to give the recursive expression for MinEdit. You also need to explain why your algorithm has this running time.

3. Let $G = (V, E)$ be a dag, a directed acyclic graph, and let $s \in V$ be a designated vertex. Suppose each edge has a postive integer length. Give a linear time algorithm to find the number of shortest paths from $s$ to each vertex $v \in V$.

4. Let $T$ be a rooted tree. By preprocessing the tree and storing suitable results in space $O(|T|)$, enable the following type of query to be answered in $O(1)$ time: given two vertices (nodes) in $T$ answer whether one vertex is an ancestor of the other, and if so which one?
Hint. You want to use suitable vertex numberings.

Challenge problem. Do not submit.
This problem concerns 2-SAT (short for 2-satisfiability). The input consists of a logical formula, e.g. $F = (x \vee y) \wedge (\overline{x} \vee z) \wedge (y \vee \overline{z})$. Each term $x, y, z$ is a Boolean variable, which can take on the values True or False. $\overline{x}$ denotes the negation of $x$, i.e. if $x =$ True then $\overline{x} =$ False, and conversely. We call the collection $x, \overline{x}, y, \overline{y}, z, \overline{z}$ the literals. The formula consists of the logical "and" of a sequence of clauses, and each clause is the logical "or" of two literals.

The question being asked is whether there is an assignment of Truth values to the Boolean variables that causes $F$ to evaluate to True. Here setting $x = y = z =$ True causes $F$ to evaluate to True, so it is satisfiable.

The version of the problem in which each clause contains 3 literals, called 3-SAT, is an NP-Complete problem, a class of problems for which there are no known polynomial time algorithms, and for which essentially all researchers believe there are no sub-exponential time algorithms. In contrast, there is a linear time algorithm for the 2-SAT problem, which you are to find.

To this end, we build a directed graph $G = (V, E)$. If the formula contains $n$ variables, there will be $2n$ vertices, corresponding to the $2n$ literals these $n$ variables induce. For each clause $(x \vee y)$ we create two directed edges: $(\overline{x}, y)$ and $(\overline{y}, x)$. Henceforth, we refer to a literal and the corresponding vertex interchangeably.

a. Show that if we assign literal $\ell$ = False, then for every literal $\ell'$ reachable from $\overline{\ell}$ in $G$, if there is a variable assignment with $\ell$ = False that causes $F$ to evaluate to True, then in every such assignment, $\ell'$ must be set to True.

b. Conclude that if both $x$ and $\overline{x}$ are in the same strong component of $G$ then $F$ cannot evaluate to True under any assignment of truth values to the variables. Furthermore, conclude that $F$ can evaluate to True only if all the literals in each strong component receive the same truth assignment for $F$.

c. If each component contains only non-conflicting literals (i.e. not both $x$ and $\overline{x}$ for any $x$) then obtain a truth assignment as follows. Assign True to the literals in each sink component, and assign False to the negations of these literals. Now remove the components containing these variables and iterate. Argue that this does indeed produce an assignment that causes $F$ to evaluate to True, and show it can be computed in linear time.