Homework 4: Dynamic Programming and Network Flows

Handed out Friday, October 20. Due at the start of class Monday, October 30.
Upload your code for Problem 5 to Moodle by 2pm (10/30/2017).

Problem 1. This problem involves some simple variants of the LCS problem. In each case you are given two sequences, $X =< x_1, ..., x_m >$ and $Y =< y_1, ..., y_n >$ as input.

(a) It is sometimes useful to allow mismatches in the LCS, but at a penalty. Suppose, for example, that the alphabet is $\Sigma = a, b, c, d, e$. Your scanning equipment sometimes confuses c's and e's. When evaluating the length of a common subsequence, we allow c's to be matched with e's (and vice versa), but we penalize each such match by counting it as just half a character.

In the LCS, we show these special half-character matches with the symbol "ε". This character can match either a c or an e. An example is shown in Fig. 1(a). The standard LCS is $< bdccade >$ of length 7. By allowing matches between c's and e's, we have a generalized LCS $Z =< bd\varepsilon c\varepsilon a\varepsilon de >$ of length $6 + 3/2 = 7.5$.

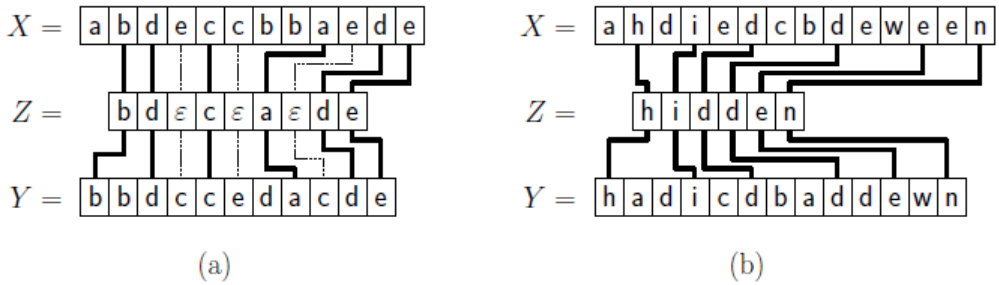

Figure 1: Problem 1.

Present a DP algorithm for computing the length of the generalized LCS of two input strings X and Y . (It is sufficient to present just the recursive rule, not a complete algorithm.) Briefly explain. What is the running time of your algorithm (if you had implemented it)?

(b) The FBI is developing a new document analyzer that is looking for hidden messages in documents. The criminals conceal the same message in multiple documents, but they hide the secret message by embedding it as a subsequence. To make these concealed messages harder to find, whenever they place a hidden character, they make sure that two consecutive characters of the hidden message are never consecutive in the document.

The FBI asks you to design an algorithm to find the LCS within a pair of documents, but subject to the condition that consecutive characters of the LCS do not appear consecutively either in X or in Y . An example is given in Fig. 1(b). The standard LCS is $< adidbdewn >$ (of length 9), but this involves many consecutive characters of X and Y . The longest LCS without consecutive characters is (I believe) $Z =< hidden >$ (of length 6).

Present a DP algorithm for computing the length of the nonconsecutive LCS of two input strings X and Y . (It is sufficient to present just the recursive rule, not a complete algorithm.) Briefly explain. What is the running time of your algorithm (if you had implemented it)?

Problem 2. The objective of this problem is to write a dynamic programming algorithm to play a game. Two players, called Jen and Ben alternate in taking moves, with Jen always going first. Initially the board consists of three piles of diamonds, which we denote (A, B, C), meaning that there are A diamonds in the first pile, B in the second pile, and C in the third. The board always consists of three numbers that are nonnegative. During a move a player can do any one of the following:

(1) Remove 1 diamond from pile 1.

(2) Remove either 1 or 2 diamonds from pile 2.

(3) Remove either 2 or 3 diamonds from pile 3.

The first player who cannot make a move loses. (And the winner gets all the diamonds.) That is, if it is a player's turn to move and the board is either $(0, 0, 0)$ or $(0, 0, 1)$ then he/she loses. Given the initial configuration, (A, B, C), and with the assumptions that Jen plays first and both players play as well as possible, determine which of the two players can force a win. (Since there is no element of chance, and the game is finite in length, one of the two can always force a win.) For example, suppose that the initial board is $(0, 1, 4)$. In this case Jen can force a win. She uses rule (3) to remove 2 diamonds from pile 3, resulting in the board (0, 1, 2). Ben's only choices are to remove 1 from pile 2 or 2 from pile 3, resulting in the possible boards $(0, 0, 2)$ and $(0, 1, 0)$. In either case, Jen can remove the final diamonds (using either rules (3) or (2), respectively) leaving Ben without a move.

a. Derive a (recursive) dynamic programming rule to determine the winner, given the initial board (A, B, C). (Be sure to include a description of the basis cases.) Justify the correctness of your formulation. (For this part I do not want a full algorithm, just the recursive rule.)

b. Present an implementation of recursive rule of part (a). (You may use memoization or the bottom-up method.) Express your running time as a function of A, B, and C.

Problem 3. The following problem explores a few modifications to Bellman-Ford. You should assume that you are given a weighted digraph $G = (V, E)$ with no negative weight cycles, and as usual, the shortest path between two vertices is the path that minimizes the total weight of edges.

(a) Describe how to find the shortest path in a directed graph from u to v with **exactly** k edges. It is sufficient to describe the modifications you would make to Bellman-Ford to do this, or you may state the recurrence relationship and give details about how you would use it to find this path.

Now, let M be the maximum, over all pairs of vertices $u, v \in V$, of the minimum number of edges in any shortest path from u to v.

(b) Describe a simple modification to the Bellman-Ford algorithm that allows it to terminate in at most $M + 1$ stages.

(c) Prove that your modified algorithm (from part b) is correct (that is, on termination the distance to each node is the same as it would be for the original Bellman-Ford algorithm), and that it terminates in at most $M + 1$ stages.

Problem 4. Network flow issues come up in dealing with natural disasters and other crises, since major unexpected events often require the movement and evacuation of large numbers of people in a short amount of time. Consider the following scenario. Due to large-scale flooding in a region, paramedics have identified a set of n injured people distributed across the region who need to be rushed to hospitals. There are k hospitals in the region, and each of the n people needs to be brought to a hospital that is within a half-hour's driving time of their current location (so different people will have different options for hospitals, depending on where they are right now). At the same time, one doesn't want to overload any one of the hospitals by sending too many patients its way. The paramedics are in touch by cell phone, and they want to collectively work out whether they can choose a hospital for each of the injured people in such a way that the load on the hospitals is balanced: Each hospital receives at most $\lceil n/k \rceil$ people. Give a polynomial-time algorithm that takes the given information about the people's locations and determines whether this is possible.

Problem 5. Programming Consider the shortest common supersequence (SCS) problem. The input to the SCS problem is two strings $A =< A_1...A_m >$ and $B =< B_1...B_n >$, and the output is the length of the shortest string that contains both A and B as subsequences.

Example: $A =< ABCBABA >$, $B =< BCAABAB >$, then the $SCS =< ABCAABABA >$, so the $len(SCS) = 9$.

Given that the recurrence relationship is as follows, implement a dynamic programming algorithm to solve the SCS problem. Let $SCS(i, j)$ be the length of the shortest common supersequence of $A[1..i]$ and $B[1..j]$. Your algorithm must run in $O(mn)$ time. Your program should prompt the user to enter two strings and output the length of the shortest common supersequence, not the actual SCS.

$$SCS(i,j) = min \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ SCS(i-1, j-1) + 1 & i, j > 0 \text{ and } x_i = y_j \\ SCS(i, j-1) + 1 & i, j > 0 \text{ and } x_i \neq y_j \\ SCS(i-1, j) + 1 & i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Extra Credit: Add backtracking hints to your code and use them to output the SCS as well as the length of the SCS.