CMSC 451 Quiz 3

This quiz is closed-book and closed-notes. You may use any algorithms or results given in class. The total point value is 50 points. Good luck!

Problem 1. (12 points) For each of the following DP algorithms, indicate the worst-case asymptotic time for:

- Computing the numeric *objective value* and
- Computing the *actual solution*, assuming that the *helper values* have already been computed.

(For example, for LCS, the objective value is the length of the LCS and the solution is the actual LCS string.) You may assume either a memoized or bottom-up implementation of each algorithm.

As an example, we have done the first one (Weighted Interval Scheduling) for you.

Algorithm	**Parameters**	**Objective Value**	**Solution**
Weighted Interval Scheduling	n (no. of intervals)	n	n
Longest Common Subsequence (LCS)	m, n (sequence lengths)		
Chain Matrix Multiplication	n (no. of matrices)		
Floyd-Warshall (all-pairs shortest paths)	n (no. of vertices)		

Problem 2. (10 points) Given two strings, $X = \langle x_1 x_2 \ldots x_m \rangle$ and $Y = \langle y_1 y_2 \ldots y_n \rangle$, the *shortest common supersequence* (SCS) is a string Z of minimum length such that both X and Y are subsequences of Z. For example, if $X = \langle$ABCBACA$\rangle$ and $Y = \langle$BCABCB$\rangle$, then $\text{SCS}(X, Y) = \langle$ABCABACBA$\rangle$ (see Fig. 1).

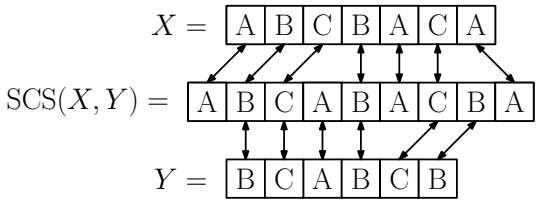


Figure 1: Shortest common supersequence.

Present an efficient dynamic programming algorithm which given two strings X and Y as input computes the *length* of $\text{SCS}(X, Y)$. Present your answer as a recursive DP formulation

(not pseudocode). Briefly justify your algorithm's correctness. Be sure to include the basis case(s) and indicate what the final length is. (**Hint**: If implemented, it should run in $O(mn)$ time.)

Problem 3. (8 points) You are given a sequence of n points $P = \{p_1, \ldots, p_n\}$ in 2-dimensional space sorted by x-coordinates. Let $p_i = (x_i, y_i)$. A *non-increasing subsequence* is any subsequence of P such that each successive y-coordinate is either equal to or smaller than its predecessor. The *longest non-increasing subsequence* (LNS) is the non-increasing subsequence having the largest number of points. For example, the LNS for the set of points in Fig. 2 has length 6, consisting of the subsequence $\langle p_1, p_3, p_5, p_6, p_9, p_{10} \rangle$.

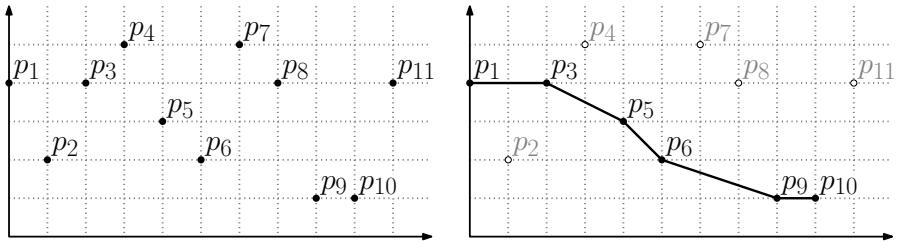
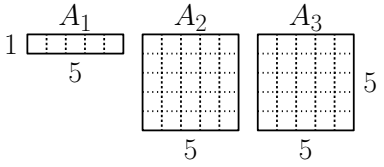


Figure 2: Computing the length of the maximum LNS.

Present an efficient dynamic programming algorithm which given a sequence of n points computes the *length* of the longest non-increasing subsequence. Present your answer as a recursive DP formulation (not pseudocode). Briefly justify your algorithm's correctness. Be sure to include the basis case(s) and indicate what the final length is. You may assume the points are given in increasing order of x-coordinates and there are no duplicate x-coordinates. (**Hint**: If implemented, it should run in $O(n^2)$ time.)

Problem 4. (10 points) Let's consider the chain matrix multiplication problem, with a small twist. Recall that if we multiply a $p \times q$ matrix with a $q \times r$ matrix the result is a $p \times r$ matrix, and the multiplication time is pqr. However, if the matrices are both square (that is, $p = q = r$), we can use a new chip that does this multiplication in $p^2/5$ time.

For example, consider the matrices in Fig. 3. Without the chip, the optimal multiplication time is 50, but with the chip, the optimal time is just 30.



$$\text{cost}((A_1A_2)A_3) = 25 + 25 = 50$$

$$\text{cost}(A_1(A_2A_3)) = \frac{25}{5} + 25 = 30$$

Figure 3: Modified chain matrix multiplication.

Present an efficient dynamic programming algorithm which given the dimensions $\langle p_0, \ldots, p_n \rangle$ of a sequence of matrices $A_1, \ldots, A_n$, where A_i is a $p_{i-1} \times p_i$ matrix, computes the optimal *multiplication time* for the entire sequence. Present your answer as a recursive DP formulation (not pseudocode). Briefly justify your algorithm's correctness. Be sure to include the basis

case(s) and indicate what the final multiplication time is. (**Hint**: If implemented, it should run in $O(n^3)$ time.)

Problem 5. (10 points) Consider the s-t network G and flow f shown in Fig. 4.

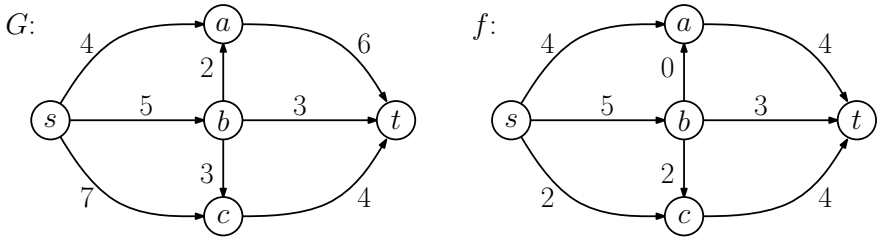


Figure 4: Network flow.

(a) (5 points) Show the residual network G_f for this flow by drawing the network. (Do not show edges that have a capacity of zero.)

(b) (3 points) List the sequence of vertices on any s to t path in G_f or indicate that there is no such path.

(c) (2 points) What is the maximum amount of flow that can be pushed along your path?