CSE 102: Spring 2021

Advanced Homework # 5

Dynamic Programming Algorithms

Find a problem (in decreasing order of preference) that

1. You can do on your own without help from any human being (or discord) or any internet/social media resources.
2. You can do on your own without help from any human being (or discord) but possibly with the help of some internet/social media resources (cite these)
3. You can do with the help of some human being or discord (cite these) with or without the help of some internet/social media resources (cite these)

For each problem, be sure to,

1. Describe the algorithm in words (at a level of detail that should be clear to students in this class).
2. Present the pseudo-code for the algorithm with comments. Comments should clarify whether the algorithm will run top to down, right to left, etc.
3. Discuss all the base cases properly.
4. Present a few examples to clearly illustrate how the algorithm works on some specific inputs.
5. Describe the computational and space complexity of the algorithm.

Problems

In the comments part of the section, please state which problem you are attempting. Use the problem numbers as they appear below.

Attempt only **ONE** of the following problems:

1. Problem 8.28
2. Problem 8.30
3. Problem 8.31
4. Problem 10.20
5. Common Subsequence Problem 10.19
   [Solution available in CLRS and many other sources]
6. Regular Expression Problem Given an input string s and a pattern p, design a dynamic programming algorithm that implements regular expression matching with support for . and $*$ where:

   - '.' Matches any single character.
   - '*' Matches zero or more of the preceding element.

   The matching should cover the entire input string (not partial.

   If needed, you may use any or all of the following or similar constraints (state clearly which constraints, if any, you are using):

   - $0 \leq$ s.length ≤ 20
   - $0 \leq$ p.length ≤ 30
   - s contains only lowercase English letters.
   - p contains only lowercase English letters, '.', and '*'.
   - It is guaranteed for each appearance of the character '*', there will be a previous valid character to match.

   [Solution available on leetcode and many other sources:
   https://leetcode.com/problems/regular-expression-matching/
   Please look up the examples 1-5 to understand the problem better without any loss of points.]

Problem 8.28. Consider the alphabet $\Sigma = \{a, b, c\}$. The elements of Σ have the following multiplication table, where the rows show the left-hand symbol and the columns show the right-hand symbol.

$$\begin{array}{c|ccc} & a & b & c \\ \hline a & b & b & a \\ b & c & b & a \\ c & a & c & c \end{array}$$

Thus $ab = b$, $ba = c$, and so on. Note that the multiplication defined by this table is neither commutative nor associative.

Find an efficient algorithm that examines a string $x = x_1 x_2 \cdots x_n$ of characters of Σ and decides whether or not it is possible to parenthesize x in such a way that the value of the resulting expression is a. For instance, if $x = bbbba$, your algorithm should return "yes" because $(b(bb))(ba) = a$. This expression is not unique. For example, $(b(b(b(ba)))) = a$ as well. In terms of n, the length of the string x, how much time does your algorithm take?

Problem 8.30. Let u and v be two strings of characters. We want to transform u into v with the smallest possible number of operations of the following three types: delete a character, add a character, or change a character. For instance, we can transform $abbac$ into $abcbc$ in three stages:

$$\begin{array}{lll} abbac & \rightarrow abac & \text{(delete } b\text{)} \\ & \rightarrow ababc & \text{(add } b\text{)} \\ & \rightarrow abcbc & \text{(change } a \text{ into } c\text{).} \end{array}$$

Show that this transformation is not optimal.

Write a dynamic programming algorithm that finds the minimum number of operations needed to transform u into v and tells us what these operations are. As a function of the lengths of u and v, how much time does your algorithm take?

Problem 8.31. You have n objects that you wish to put in order using the relations "$<$" and "$=$". For example, with three objects 13 different orderings are possible.

$$\begin{array}{lllll} a = b = c & a = b < c & a < b = c & a < b < c & a < c < b \\ a = c < b & b < a = c & b < a < c & b < c < a & b = c < a \\ c < a = b & c < a < b & c < b < a & & \end{array}$$

Give a dynamic programming algorithm that can calculate, as a function of n, the number of different possible orderings. Your algorithm should take a time in $O(n^2)$ and space in $O(n)$.

10.19 Let A and B be arrays of n integers each. A *common subsequence* of A and B is a sequence that is a subsequence of A and is a subsequence of B. The subsequence is not required to be contiguous in A or B. For example, if the entries of A are 5, 8, 6, 4, 7, 1, 3, and the entries of B are 4, 5, 6, 9, 7, 3, 2, a longest common subsequence is 5, 6, 7, 3. Give an algorithm to find a longest common subsequence of A and B. Analyze the worst-case running time and space requirements of your algorithm.

10.20 Suppose we are given three strings of characters: $X = x_1x_2 \cdots x_m$, $Y = y_1y_2 \cdots y_n$, and $Z = z_1z_2 \cdots z_{m+n}$. Z is said to be a *shuffle* of X and Y if Z can be formed by interspersing the characters from X and Y in a way that maintains the left-to-right ordering of the characters from each string. For example, *cchocohilaptes* is a shuffle of *chocolate* and *chips*, but *chocochilatspe* is not. Devise a dynamic programming algorithm that takes as input X, Y, Z, m, and n, and determines whether or not Z is a shuffle of X and Y. Analyze the worst-case running time and space requirements of your algorithm. *Hint*: The values in your dictionary should be Boolean, not numeric.