CSCI 3104 Spring 2023
Instructors: Ryan Layer and Chandra Kanth Nagesh

Homework 25

Due Date ..... April 13, 2023
Name ..... **Blake Raphael**
Student ID **109752312**
Collaborators ..... **Alex Barry, Brody Cyphers, and Ben Kohav**

Contents

1 Instructions

- The solutions **should be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Here's a short intro to LaTeX.
- You should submit your work through the **class Gradescope page** only (linked from Canvas). Please submit one PDF file, compiled using this LaTeX template.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please submit this document with no fewer pages than the blank template (or Gradescope has issues with it).
- You are welcome and encouraged to collaborate with your classmates, as well as consult outside resources. You must **cite your sources in this document. Copying from any source is an Honor Code violation. Furthermore, all submissions must be in your own words and reflect your understanding of the material.** If there is any confusion about this policy, it is your responsibility to clarify before the due date.
- Posting to **any** service including, but not limited to Chegg, Reddit, StackExchange, etc., for help on an assignment is a violation of the Honor Code.
- You **must** virtually sign the Honor Code (see Section 2). Failure to do so will result in your assignment not being graded.

2 Honor Code (Make Sure to Virtually Sign)

- My submission is in my own words and reflects my understanding of the material.
- Any collaborations and external sources have been clearly cited in this document.
- I have not posted to external services including, but not limited to Chegg, Reddit, StackExchange, etc.
- I have neither copied nor provided others solutions they can copy.

Agreed (I agree to the above, Blake Raphael). □

3 Standard 25: Design a Dynamic Programming Algorithm (Synthesis Standard)

Problem 1. If x is a string, then a *subseequence* of x is a string of characters that occur in the same order as they do in x, but not necessarily contiguously. For example, abc is a subsequence of $aebeec$, but bca is not.

The Longest Reflective Subsequence problem is defined as follows.

- Input: A string x with characters from a finite alphabet Σ
- Output: A subsequence y of x such that for all $i \in \{1, \ldots, \ell = len(y)\}$, $y_i = y_{\ell - i + 1}$, that is as long as possible.

Examples.

1. Any subsequence of length 0 (the empty string) or length 1 (a single character) is reflective.
2. In $abasd; klfjasd; lfkjba$, the first two and last two characters together form a reflective subsequence $abba$. There are also several reflective subsequences of length 6, for example, $abjjba$, $abffba$, $abddba$, $ab;;ba$, $abkkba$. The longest are of length 9, for example $abkfdfkba$.
3. In 12**34**56**7898**5**3**5, the longest reflective subsequence is 3589853, which we've put in **bold** in the original string.

The goal of this problem is to design a dynamic programming algorithm to solve the Longest Reflective Subsequence problem.

3.1 Problem 25(a) (1 Point)

(a) Let $L[i, j]$ denote the length of the longest reflective subsequence of $x[i, \ldots, j]$. Write down a mathematical recurrence for $L[i, j]$. Clearly justify each case.

Answer.

$$L[i,j] = \begin{cases} 0 & : i > j, \\ 1 & : i = j, \\ \text{if } x[i] = x[j] & \\ L[i+1, j-1] + 2 & \\ \text{else} & \\ max\{L[i+1, j], L[i, j-1]\} & : i < j. \end{cases}$$

When $i > j$, we have an empty substring, so length of 0.
When $i = j$, we have a substring of length 1.
When $x[i] = x[j]$, we have a reflexive substring, so we add 2 to our length and continue our recursion inwards.
When $x[i] \neq x[j]$ we want to choose the recursion inwards that changes either i or j that provides us with the maximum length substring we are looking for, hence the max function application.

□

3.2 Problem 25(b) (2 Points)

(b) Clearly describe how to construct and fill in the lookup table. For the cell $L[i, j]$, clearly describe the sub-cases we consider (e.g., using a dependency diagram), which optimal sub-case we select, and any relevant pointers that should be included in the table of back-pointers.

Answer. For all the rows where $i > j$, the cells will be filled with 0. For all the rows where $i = j$, the cells will be filled with 1. For every other cell, we will see if $x[i] = x[j]$. If $x[i] = x[j]$, the cell will be filled with whatever $L[i+1, j-1]+2$ evaluates to. Otherwise, we will fill the table with whatever the maximum length of either $L[i+1, j]$ or $L[i, j-1]$ evaluates to.

The optimal sub-case we select will be the case where we always choose the maximum length of the reflexive sub-string. Since this is built into our recurrence relation, it will be the bottom right corner of our lookup table.

The relevant pointers that will be added are the ones that lead to the optimal case. So this will be the maximum of our cases where $x[i] \neq x[j]$ or when $x[i] = x[j]$, or when $i = j$. We will add up all the lengths of these sub-cases through our lookup table to get to the length of our optimal case. □

3.3 Problem 25(c) (1 Point)

(c) Work through an example of your algorithm using the input string $x =$ `uzwfzbu`. Clearly show how to recover an optimal solution by backtracing. You may hand-draw your table(s), but your explanation must be typed.

Answer. Here is the table:

$L[i,j]$	i $\rightarrow$	0	1	2	3	4	5	6
j $\downarrow$	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0
2	0	1	1	1	0	0	0	0
3	0	1	1	1	1	0	0	0
4	0	3	3	1	1	1	0	0
5	0	3	3	1	1	1	1	0
6	0	5	3	1	1	1	1	1

Our optimal solution, or the longest reflexive sub-string will be the following: `uzwzu`.

Here are the backtracking steps:
1: $L[i = 0, j = 6] = 5$, this is our maximum overall length.
2: $L[i = 1, j = 5] = 3$, we subtract both i and j by 1 since both characters were the same.
3: $L[i = 1, j = 4] = 3$, we take the maximum length of subtracting either i or j, in this case it was j since we get 3 instead of 1.
4: $L[i = 2, j = 3] = 1$, we subtract both i and j by one since both characters were the same.
5: $L[i = 2, j = 2] = 1$, we pick either i or j to subtract by one since both subtractions give us the same length so there is no maximum. In this case we picked to subtract j by 1. We have no more recursive steps, so we terminate here.

□