CSCI 3104 Fall 2021
Instructors: Profs. Grochow and Waggoner

Problem Set 9

Due Date ..... TODO
Name ..... **John Blackburn**
Student ID **jobl2177**
Collaborators ..... **List Your Collaborators Here**

Contents

1 Instructions **1**

2 Honor Code (Make Sure to Virtually Sign) **2**

3 Standard 24- Dynamic Programming: Design DP Algorithms **3**
3.1 Problem 2(a) 3
3.2 Problem 2(b) 4
3.3 Problem 2(c) 5

1 Instructions

- The solutions **must 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 Canvas page** only. 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)

Problem 1.

- 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 john blackburn. □

3 Standard 24- Dynamic Programming: Design DP Algorithms

Problem 2. The Highest Reflective Subsequence problem is defined as follows.

- Instance: A string $\omega \in \Sigma^n$, where Σ is our fixed alphabet.
- Solution: A subsequence $\psi \in \Sigma^\ell$ of ω of maximum length such that for all $i \in \{1, \ldots, \ell\}$, $\psi_i = \psi_{\ell-i+1}$.

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

3.1 Problem 2(a)

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

Answer.

$$L[i,j] = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ L[i+1, j-1] + 2 & \text{if } i, j > 0,\ \omega[i] = \omega[j] \\ MAX(L[i+1, j], L[i, j-1]) & \text{if } i, j > 0,\ \omega[i]! = \omega[j] \end{cases}$$

The base cases occur when $i = j$ and when $i > j$. if $i = j$ we know the length of the subsequence must be 1 since i and j are in the same spot of ω. As for when $i > j$ we know that the subsequence will be of length 0 because the sequence cannot be a palindrome when $i > j$.

Now for the recursive cases: They can be broken into two cases, them being when the outer two letters are equivalent and when they are not. When $\omega[i] = \omega[j]$ we know that the subsequence will be 2 links longer than the subsequence of the string ω where $i = i + 1$ and $j = j - 1$. This effectively is saying we take the subsequence from the string ω without the outer two letters and add 2 to its length because the last and first letters of the new string are equivalent. As for when $\omega[i]! = \omega[j]$ we take the max of the two subsequences that make up the inside of the current string we have, one string including i up to $j - 1$, and the other including j going down to $i + 1$.

□

3.2 Problem 2(b)

(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, which optimal sub-case we select, and any relevant pointers that should be included in the table.

Answer. Let w be our input string of length n. We initialize a lookup table $T[0, ..., n][0, ..., n]$ to be a two-dimensional array, where each cell stores: A natural number which indicates the max length of a palindrome sequence at that length

The first row and column are both filled with the letters of the input string. let the string along the row be denoted as w and the string along the column be denoted by t.

let i denote the row number and j denote the column number.

We now proceed to fill in the cells in a bottom up manner, row-by-row, the cells $T[i][j]$ are filled as follows.

Case 1 : suppose the cell we are looking at is where $i = j$. For example $T[1][1]$ or $T[3][3]$. So, in this case we always set: -Set $T[i][j]$.length = 1;

Case 2: suppose $i > j$. For these cases we set: - Set T [i][j].length = 0;

Case 3: Suppose $w_i = t_j$. This tells us that the characters at this subproblem are equivalent and should be added to our sequence. So we take the following actions: -Set $T[i][j]$.length= $T[i + 1][j–1]$.length +2;

Case 4: Suppose $w_i! = t_j$. We need to consider the two subproblems, whose solutions are stored in: T [i+ 1][j] and T [i][j – 1], respectively.

If T [i + 1][j].length ≥T [i][j – 1].length, we set:

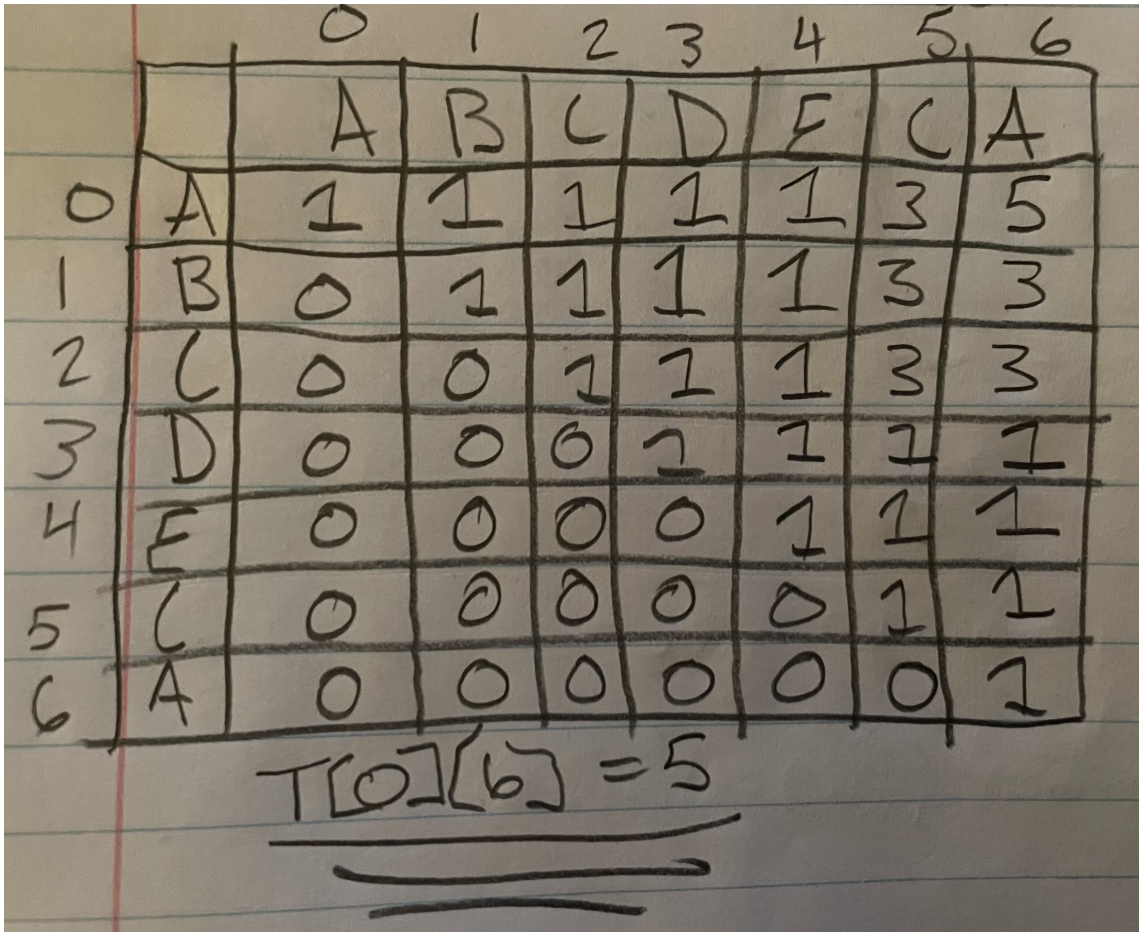
– T[i][j].length=T[i+1][j].length

Otherwise, we set: – T[i][j].length=T[i][j–1].length

The table gives us the max length of our highest reflective subsequence at T[0][n-1]. No pointers are neccessary with this lookup table to find the length of our subsequence. □

3.3 Problem 2(c)

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

Answer. 

I worked through the above input string and made a table for it. I started at the bottom left of the table and made my way across the row and then went up and across each row and finally finished at T[0][6] to get the final answer of 5. My answer means that the highest reflective subsequence of the input string has a length of 5. This table structure works becuase each new problem builds on the previous subproblems before it. Leaving the final problem having built upon all the previous ones giving the final answer in the final box. You can only solve the final problem in the table when every previous subproblem has been completed. □