CSCI 3104 Fall 2022
Instructors: Prof. Grochow and Chandra Kanth Nagesh

Problem Set 9

Due Date .... November 7, 2022
Name .... **Tyler Huynh**
Student ID **109603994**
Collaborators .... **N/A**

Contents

Instructions

- The solutions **must be typed**, using proper mathematical notation. We cannot accept hand-written solutions. Useful links and references on LATEXcan be found here on Canvas.
- 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 Honor Code). Failure to do so will result in your assignment not being graded.

Honor Code (Make Sure to Virtually Sign the Honor Pledge)

Problem HC. On my honor, my submission reflects the following:

- 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.

In the specified region below, clearly indicate that you have upheld the Honor Code. Then type your name.

Honor Code. I, **Tyler Huynh** on my honor pledge that my submission is a reflection of my own understanding of the material, any and all collaborations/sources have been properly cited, I have not posted any material to external sources, and I have not copied other solutions as my own. □

1 Standard 23: Dynamic Programming: Using Recurrences to Solve

Consider the KNAPSACK problem from lecture. Fill in the dynamic programming table for the following input: $[(1,3),(2,3),(2,2),(3,3),(4,5)], W = 7$. Here, each pair (v_i, w_i) denotes an item with value v_i and weight w_i. What is the value of the optimal knapsack choice? (You **do not** need to fill in back-pointers.)

Answer. Referenced: **https://www.youtube.com/watch?v=8LusJS5-AGot=490s**
I will first specify what the items, weight, and values are for our input:

Items: 1, 2 ,3, 4, 5
Values: 1, 2, 2, 3, 4
Weights: 3, 3, 2, 3, 5

I will now fill in our table to find he most optimal knapsack code, as follows:

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
(2, 2) 1	0	0	2	2	2	2	2	2
(1, 3) 2	0	0	2	2	2	3	3	3
(2, 3) 3	0	0	2	2	2	4	4	4
(3, 3) 4	0	0	2	3	3	5	5	5
(4, 5) 5	0	0	2	3	3	5	5	6

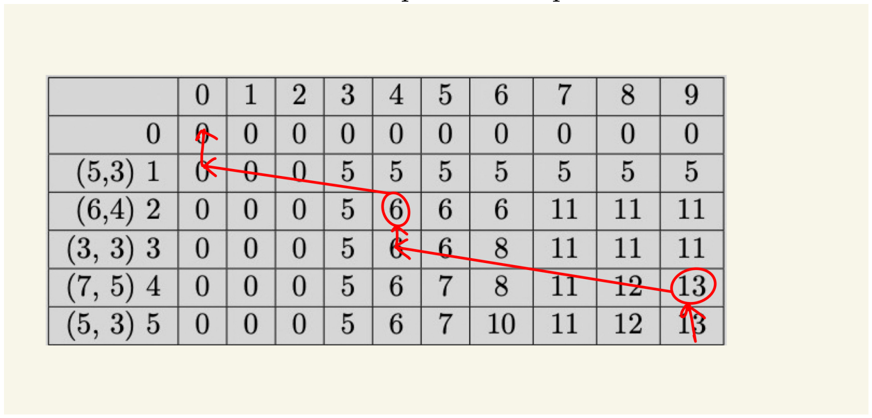
From the above we can see that the most optimal knapsack choice would be 6. □

2 Standard 24: Backtracking to find Solutions

Consider the Knapsack problem, with input $[(5,3),(6,4),(3,3),(7,5),(5,3)], W = 9$. Here, each pair (v_i, w_i) denotes an item with value v_i and weight w_i. The following is the dynamic programming table for the optimum value. From the table, **clearly indicate the steps you take to backtrace to find the optimum knapsack.** Be sure to include the steps at which an item was *not* chosen, as well as those at which it was chosen. That is, the number of steps your algorithm takes should be the same as the length of the list (5) (or maybe 6, depending on how you handle the last step).

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
(5,3) 1	0	0	0	5	5	5	5	5	5	5
(6,4) 2	0	0	0	5	6	6	6	11	11	11
(3, 3) 3	0	0	0	5	6	6	8	11	11	11
(7, 5) 4	0	0	0	5	6	7	8	11	12	13
(5, 3) 5	0	0	0	5	6	7	10	11	12	13

Answer. I have also included how I backed tracked to find the most optimum knapsack, the circles represent the items I **took** and the arrows represent the path that I took.



	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
(5,3) 1	0	0	0	5	5	5	5	5	5	5
(6,4) 2	0	0	0	5	6	6	6	11	11	11
(3, 3) 3	0	0	0	5	6	6	8	11	11	11
(7, 5) 4	0	0	0	5	6	7	8	11	12	13
(5, 3) 5	0	0	0	5	6	7	10	11	12	13

The steps to backtrack to find the optimum knapsack will be:

- 1. = The first step we would take would be to compare item 5 at column 9 with the row above it, we can see that since these two values are equal to each other we would **not** take it.

- 2. = The second step we would take would be to compare item 4 at column 9 with the row above it, we can see that since these two values are **not equal** to each other than we will traverse 5 towards the left side of our table.

- 3. = The third step we would take would be to compare item 3 at column 4 with the row above it, we can see that since these two values are equal to each other we would **not** take it.

- 4. = The fourth step we would take would be to compare item 2 at column 4 with the row above it, we can see that since these two values are **not equal** to each other than we will traverse 4 towards the left side of our table.

- 5. = The fifth step we would take would be to compare item 1 at column 0 with the row above it, we can see that since these two values are equal to each other we would **not** take it.
  From the above we have finished backtracking.

We can see that the most optimum knapsack would be:
13 where we take item 4 with value and weight (7, 5) and item 2 with value and weight (6, 4). □

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

Problem 1. Recall from class that 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**34567898535**, 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)

(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. Referenced **https://www.geeksforgeeks.org/longest-palindromic-subsequence-dp-12/**

For this problem we can see that the subproblem will be to denote the length of the longest reflective subsequence of $x[i, \ldots, j]$. From this I will write down a recurrence for $L[i, j]$, such that:
Let n equal the length of the string.

$$L[i,j] = \begin{cases} 0 & : n = 0 \\ 1 & : i = j \\ 2 & : x_i = x_j, j - i \leq 1 \\ L[i+1, j-1] + 2 & : x_i = x_j, j - 1 > 1 \\ max[L(i+1, j), (i, j-1)] & : x_i \neq x_j \end{cases}$$

I will now explain how I came to each of the cases above:

- **Base Cases:**
  For this case by the definition of the problem at hand we can see that if the length of the subsequence were to be of length 0 or of length 1 than the entirety of the subsequence is reflective. If it is of length 0 than we have a reflective subsequence of an empty string, if it is equal to 1 than we have a reflective subsequence of just that 1 character.

  Another base case would be when the entries in the subsequence are the same, but the index of the characters are right next to each other such that it results in it being 2.

- **Case 1**
  For case 1 we can see that for this case they are the same character meaning that $x_i = x_j$, but there is at least one character that exists in between these indexes, such that:

  $$L[i+1, j-1] + 2$$

  From the above we can see that if we increment i by 1 and subtract j by 1 we are looking at the string that does not include x_i or x_j since we know that the subsequence will already have these characters within it hence us adding two to it.

- **Case 2:**
  For case 2 we can see that for this case we are trying to find the max of the longest reflective subsequence when the characters x_i and x_j are not equal so they both cannot be included in the subsequence , such that:

  $$max(L[i+1, j], L[i, j-1])$$

  From the above we can see that if since we are trying to find the max of the longest reflective subsequence, where we are looking at the longest reflective subsequence in the substring where we do not include i and the longest subsequence in the substring where we do not include j since .

□

3.2 Problem 25(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 (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 the cell of $L[i, j]$, the sub-cases that we have to consider is that if the the subsequence is of length 0 than we have a reflective subsequence of an empty string, if it is of length 1 than we have a reflective subsequence of just 1 character.

When the subsequence is of length 2 and when $x_i = x_j$ that is when the characters of i and j are the same and the indexes of the characters are next to each other.

When $x_i = x_j$, meaning they are the same character, another sub-case would be when we increment i by 1 and subtract j by 1 we are looking at the string that does not include x_i or x_j since we know that the subsequence will already have these characters within it hence we add two to the sub-sequence, since the characters are the same, which will be reflective.

The last sub-case that we have to consider is when $x_i \neq x_j$. We are looking at the longest reflective subseqeunce in the substring where we do not include i and the longest subsequence in the substring where we do not include j, where we would be finding the max between the two.

For the lookup table, we will have the rows of the table as i where i is 0 to $n-1$ and the columns will be j is 0 to $n-1$. i indicates the starting index and j indicates the ending index. The way that the lookup table will be filled is looking at the previous cases stated above to see what case will fit the most and calculate it.

The back-pointers that we would use is when we consider where the value came from by considering the above cases. □

3.3 Problem 25(c)

(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. I will now run the algorithm on the example input string $x =$ `uzwfzbu`:

L[i, j]	0, u	1, z	2, w	3, f	4, z	5, b	6, u
0, u	1	1	1	1	3	3	5
1, z	x	1	1	1	3	3	3
2, w	x	x	1	1	1	1	1
3, f	x	x	x	1	1	1	1
4, z	x	x	x	x	1	1	1
5, b	x	x	x	x	x	1	1
6, u	x	x	x	x	x	x	1

From the above we can see that the combination of uz(possible options include f, w)zu would be the most optimal solution via backtracking, such that:

- **Step 1:** We will begin at i = 0 and j = 6 where we can see that 5 will be the longest reflective subsequence of the string $x =$ `uzwfzbu`.

- **Step 2:** We will back track by comparing $i = 0$ and $j = 6$ to the cell below it and the cell to the left of it, we can see that this is **not** equal to either so we will refer to our algorithm and see that since we got the value from case 1 from our recurrence relation where case 1 dictates that since x_i and x_j are of the same character, but we know that at least one character exists between these indexes. So we increment i and decrement j by 1, from here we are looking at a string that does not include either x_i or x_j, since we know that these subsequence will already have these characters within it, so we add 2. From here we know that since we had to use case 1 from our recurrence relation we backtrack to $i = 1$ and $j = 5$. This means that both u's is in the longest reflective subsequence so our current subsequence is uu.

- **Step 3:** We compare the index $i = 2$ and $j = 5$, to the cell below it and the cell to the left of it. Since it is equal to the cell to the left of it, this index's value came from the cell at $i = 1$, $j = 4$. Hence we will backtrack to this cell to find the most optimal solution.

- **Step 4:** We will now compare the index at $i = 1$ and $j = 4$, to the cell below it and to the left we can see that this value is not equal to either. This is case 1 where since $x_i = x_j$ are the same character, but there exists at least one character in between them. We must increment i and decrement j by 1, from here we are looking at a string that does not include either x_i or x_j, since we know that these subsequence will already have these characters within it, so we add 2. From here we know that since we had to use case 1 from our recurrence relation we backtrack to $i = 2$ and $j = 3$. We also know that both z's must be in our subsequence. So, our current subsequence is uzzu.

- **Step 5:** From here we compare the index of $i = 2$ and $j = 3$ to the cell below it and the cell to the left of it , we can see that since these values are equal, there exists two possible solutions here as the index of $i = 2$ and $j = 3$, got its values from either of these indexes since they are equal.
- **Step 6:** Here we can see that since we have backtracked, thus far we have reached the most optimal solution for the input string of $x =$ `uzwfzbu`, such that the most optimal solution would be of length = 5. From our backtracking we can see that we chose both u's and both z's to be in our longest

subsequence. However, since there are other characters in our string that could exist in between the two z's to make it a length of 5. Thus our final longest reflective subsequence could be of the following: uzwzu, uzfzu.

□