FIT2004 S1/2019

Assignment 2: Analysis

Nicholas Chong (29808146)

**Introduction**

This document will give a brief outline of my solution along with the worst-case space and time complexity of my solution. Each section will be about a single task in the assignment.

For reference, let n and m be the size of the first and second text respectively; and let N be the number of words in the dictionary and M be the maximal size of the words in the list. Furthermore, assume that I am discussing the **worst-case complexity for both time and space complexity** for any operations.

**Task 1: Finding the longest subsequence of common alphabets**

The function works by retrieving the two words from the file and creating a table with dimensions based on the length of the first word and the second word (adding 1 to each for a null row and column). For example, if the two messages were “abcde” and “cdgfee”, the table would have (5 + 1 = 6) rows and (6 + 1 = 7) columns.

Each element in the table represents the length of the longest common subsequence out of two substrings of the original words. Using our previous example, an element in the 3rd row and 4th column would contain the length of the longest common subsequence of “abc” and “cdgf”. We fill out the table based on:

Base case: DP[i][j] = 0 if i or j is 0

Recurrence relations:

DP[i][j] = 1 + DP[i-1][j-1] if the letters at the position match

DP[i][j] = max(DP[i-1][j], DP[i][j-1]) otherwise

There will never be a common subsequence of two words if one of the words are empty, so it will always have a length of 0. If the two letters at the position match, that means we have found another letter to add to our building subsequence and so the length increases by 1 and we add the optimal solution if the letters weren’t there. If the two letters at the position do not match, we consider the better solution of having the first word without the newest letter and the second word without the newest letter.

We then acquire the longest common subsequence using backtracking. Starting at the very bottom right of the table, we add the letter if the elements to the left and above of our current element are the same AND that value being different to the top-left element. If not, we move towards the element with the higher value. At the end, we get our encrypted word.

**Worst-case Time Complexity**

We begin by opening the file’s contents and retrieving both lines which will take O(n) and O(m) to iterate and retrieve.

**Worst-case Space Complexity**

**Task 2: Words separating**

**Worst-case Time Complexity**

**Worst-case Space Complexity**