CSCI311-DNAProject

Thomas Stone, Jake Etzler, Ryan Mosenkis

November 2021

1 Algorithm Analysis

1.1 Longest Common Substring

The Longest Common Substring algorithm we developed takes 2 strings as input and returns an integer representing the length of the longest common substring. Our algorithm achieves this by utilizing principles of dynamic programming to optimize our run time. We start by initializing a blank table of dimensions (string1.length + 1, string2.length + 1) where we will store results of previously done comparisons. We then compare compare every element in each string, storing results of each comparison in its spot in the table. Then, we iterate over every value in the table to determine the max found substring length. Because both the processes of comparing each value in every string as well as iterating over every single table value take $O(m * n)$ run time respectively, our Longest Common Substring algorithm's total rubntime ends up being $O(mn)$.

1.2 Longest Common Subsequence

Our Longest Common Subsequence algorithm is very similar to our Longest Common Substring algorithm. It also works by iterating across strings, comparing every value. It stores those values in a table, where another loop then iterates over the table to find the max found subsequence length. This also leaves us with a runtime of $O(mn)$.

1.3 Edit Distance

The edit distance algorithm creates a table using vectors that will be used to store the results of subproblems. This algorithm uses a bottom-up approach to determine the minimum number of actions that are needed to convert one string into another. These actions can either be insertion, removal, and replacement of characters. There are two base cases for this algorithm: when the first string is empty and when the second string is empty. In both of these cases, there will be either all removals or all insertions to get the strings to be equal. Additionally, if the last characters of the two strings are the same, that character can be ignored and we can recur for the remainder of each string. If the last characters of the two strings are different, all possible actions (insert, remove, replace) are considered and the minimum of these is taken and inserted into the table. The algorithm has nested for loops, where it loops through each character in the first string, and within this it loops through each character in the second string. Within the for loops, there are O(1) actions completed, resulting in an overall runtime of $O(mn)$, where m is the length of the first string and n is the length of the second string.