Lab 7 Edit Distance

Introduction

The objective of this lab is to implement a dynamic programming algorithm known as Edit Distance. You're given two strings and you're to calculate the distance between these strings. For example, if you're given "eddie" and "eddie" the distance would be 0. Or you're given "a" and "z" the distance would 1 since the only change needed is to replace "a" with "z", this would cost one 1. However, if you were to remove "a" and then add "z" then this would cost 2, since we made two moves. You're to implement this algorithm without using recursion. Our professor went over it in class, you need to use a 2D array.

Design

The biggest design hurdle was solved during class, the concept of how to solve this problem was given. Use a 2D array and compare every letter to each other, use the left cell, right cell, and top left corner cell to dictate the value of the cell you're currently on.

This was still challenge, at first I tried to fill the entire cell with the correct values but this proved to be too difficult, I also realized I had to manually create the 2D array. So I created a utility function for this purpose, filling every cell with infinity except the top row and most left column. The arrays dimensions were provided by the lengths of the strings plus one, as this algorithm requires an empty corner.

Next is to evaluate the strings, starting at index (1, 1), you would then compare the first element in the first string to to first element in the second string, if they're the same you get the minimum value of the left cell, top left cell, and right cell. To do this I created a utility function that receives a coordinates and the 2D array, and returns the values. Now if the values are out of bounds it'll return infinity as the value.

It repeats this process until it gets to the bottom left corner, this is the distance between strings.

Results

All test cases passed, I tested my utility functions, along with a full end to end test. The time complexity of the algorithm is length of the first string multiplied by the length of the second string.

Conclusion

If this algorithm was done using recursion lots of wasted work and backtracking would of been done, essentially it would have been a brute force implementation. Using the algorithm

above we’re able to get to the answer right away, it's still essentially n^2 but there isn’t wasted work.