Levenshtein Distance/Edit Distance

The Levenshtein distance, also known as the edit distance, is a measure of the minimum number of operations required to transform one string into another. These operations can include insertion, deletion, and replacement of characters in the strings.

The Levenshtein distance is commonly used in applications such as spell-checking, search engine query suggestions, and text-to-speech alignment. The algorithm involves constructing a matrix of distances between substrings of the two input strings and computing the minimum number of operations required to transform the first string into the second string.

Here are the steps to perform the Levenshtein algorithm:

- Initialize a matrix M with dimensions (m+1)x(n+1), where m and n are the lengths of the two strings being compared. Each element in the matrix represents the minimum number of edits required to convert a substring of the first string to a substring of the second string.
- Fill in the first row and column of the matrix with the values 0, 1, 2, 3, ..., m and 0, 1, 2, 3, ..., n, respectively. These values represent the number of edits required to convert an empty string into a substring of the other string.
- For each element in the matrix M[i,j], where i > 0 and j > 0, compute the minimum of the following three values:
  a. M[i-1,j] + 1, which represents the cost of deleting a character from the first string.
  b. M[i,j-1] + 1, which represents the cost of inserting a character into the first string.
  c. M[i-1,j-1] + (0 if s1[i-1] == s2[j-1], 1 otherwise), which represents the cost of substituting a character in the first string with a character in the second string. If the characters are the same, the cost is 0, otherwise the cost is 1.
- The value of the bottom-right element in the matrix, M[m+1,n+1], represents the minimum number of edits required to convert the entire first string into the entire second string.

Calculate Minimum number of operations required to convert a given string into another string?

Allowed operations: **Insertion, Deletion, Replacing**

Try to convert String Hall into Hello:

	“ ”	**H**	**A**	**L**	**L**
“ “	0	1	2	3	4
H	1				
E	2				
L	3				
L	4				
O	5				

The time complexity of the Levenshtein algorithm is O(mn), where m and n are the lengths of the two strings being compared. The space complexity of the algorithm is also O(mn), as it requires a matrix of size (m+1)x(n+1) to store the edit distances.