Project Report

Title: Building a spell checker which checks for misspelling in a text and suggester for misspelled words

Project Members

Ibrahim Gulmez - 19050111032
Bulent Oral - 19050111028
Furkan Riza Parlak - 19050111040
Enes Batuhan Caliskan - 21050151006
Huseyin Ates - 20050111013

Abstract

Our aim in this project is to detect misspelled words in a text and to offer suggestions to the user for misspellings. In the project, we first created the dictionary. When the user enters any text into the program, the program splits the text in to the words in it and checks it one by one.

We determined our suggestions based on the minimum number of letter additions, removals, and replacements. We used the Levenshtein algorithm to do this. It is also known as Edit Distance algorithm. Actually we didn't find the actual distance but similarity between two strings. Then, using the keyboard layout, we found the actual distance of the filtered word to the string typed by the user.

To find the distance between any two characters, we used the Euclidean distance. Additionally, we have taken into account whether strings are anagrams to improve the suggestions, because it is a common mistake to mix up letter orders.

Implementations and Acknokwlendments

What is the Levenshtein algorithm and how it works?

Levenshtein Algorithm: This algorithm gives the minimum character arrangement required to convert two strings to each other. These edits can be deletion, addition or modification.

In our project we implemented this algorithm using dynamic programming. The algorithm uses a matrix to store the minimum number of single-character edits needed to convert one string to the prefix of another string.

For each letter pair, the algorithm calculates the minimum number of edits needed to convert the first string up to that letter to the second string up to that letter. Adds one to the minimum number of edits required for the previous sub-problem if the letters are different. If the letters are the same, it will be the same as the minimum number of edits of the previous subproblem.

At the end of the algorithm, it gives us the solution in the lower right corner of the same matrix. This is the minimum number of edits required to convert one string to another.

Since we are using two dimensional array in its implementation, the time and space complexity of Levenshtein algorithm using dynamic programming is O(m*n).

Levenshtein Algorithm

```
function getLevenshteinDistance(word1, word2):
   distance = max(len(word1), len(word2))
   if word1 is equal to word2:
      return 0

   if len(word1) is 0 or len(word2) is 0:
      return distance

   dpMatrix = [len(word1) + 1][len(word2) + 1]

   for i = 0 to len(word1):
      dpMatrix[i][0] = i
   for j = 0 to length(word2):
      dpMatrix[0][j] = j

   for i = 1 to length of word1:
      for j = 1 to length of word2:
         cost = 0 if word1[i-1] == word2[j-1], else 1
         dpMatrix[i][j] = min(dpMatrix[i-1][j] + 1,
                              dpMatrix[i][j-1] + 1,
                              dpMatrix[i-1][j-1] + cost)

   return dpMatrix[length of word1][length of word2]
```

What is the Eucledian distance?

Eucledian Distance: In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.

In our project, we can think of each key on the keyboard as a point. To find the distance of any two letter, we just need to apply the formula.

Formula:

$$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}$$

For example, the distance between the letter Q(0, 0) and the letter E(0, 2) is 2.

References

https://en.wikipedia.org/wiki/Euclidean_distance
https://en.wikipedia.org/wiki/Levenshtein_distance
https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0