Spell check algorithm based on Wagner-Fischer matrix model

TechGenX

July 8, 2024

The search bar in the top section can compensate for the spelling mistakes in the input and return the best possibility by **Levenshtein distance**

This algorithm uses reccursion and checks three subcases.The Levenshtein distance d between two strings a and b is defined as follows:

$$lev(a,b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |a| = 0 \\ lev(tail(a), tail(b)) & \text{if staring charecters match} \\ \min \begin{cases} lev(tail(a), b) + 1 \\ lev(a, tail(b)) + 1 \\ lev(tail(a), tail(b)) + 1 \end{cases} & \text{else} \end{cases}$$

where: $tail(a)$ is the substring of a without the starting string. In the video and the code, ill do the opposite and define $head(a)$ which is the substring except the last charecter

The equation for the reccursion in the worst case is

$$T(n) = 3 \cdot T(n-1) + \Theta(1)$$

and the time complexity becomes $T(n) = O(3^n)$ which is bad. This algorithm involves repetitions i.e. for some cases, same thing is being calculated more than once. To improve, Wagner and Fischer devoloped a matrix based approach which can find the distance in **linear time**

Algorithm

The distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.

1. for every loop, distance is calculated by creating a matrix with rows as input (from) and coloumns as movie name (to) with empty spaces as shown in 1. Let's take the example from "BOATS' to 'FLOATS'

2. Each edge box is filled with the distance from 'cumulative of charecters in row above the box' to 'cumulative of charecters in coloumns left of box'. For example, take the 3rd box in the first row _ to BOA needs three operations (3 additions) as shown in 2

3. The inbetween boxes is filled by selecting the box on left, up, and diagonally left-up with least number. If the box is

   left box : going from left box to current is equivalent to adding at the end. Thus fill the box with 'leftbox + 1'

   upper box : going from upper box to current is equivalent to deletion at the end. Thus fill the box with 'upperbox + 1'

   diagonal box : going from diagonal box to currentUserName is equivalent to substitution at the end. If the end charecters are not same, then value is 'diagonalbox + 1' as in 3 and if it is, then its same ('diagonalbox') as in 4

4. Do this till the last box to get the **edit distance**

Time complexity

Each step to fill a cell in matrix requires a constant amount $\Theta(1)$ time and thus the total time taken for the algorithm is

$$T(m, n) = m \cdot n \cdot \Theta(1)$$

where m and n are the lengths of dictionary word and input word

If the intention is to check for each for each dictionary word in the set of words of size d, filter words less than a set limit l and sort in linear time with algorithms such as 'counting sort' or 'radix sort', The overall time taken for a word of length n would be

$$T(n) = \sum_{i=1}^{d} m_i \cdot n \cdot \Theta(1) = \Theta(d \cdot n) = \Theta(n)$$

and for sorting, the time taken would be $\Theta(l + n')$ where n' is the count of words from dictionary filtered and since $n' \leq d$, the time taken including sorting can be written as

$$T(n) = \Theta(d \cdot n) + O(l + d) = \Theta(n)$$

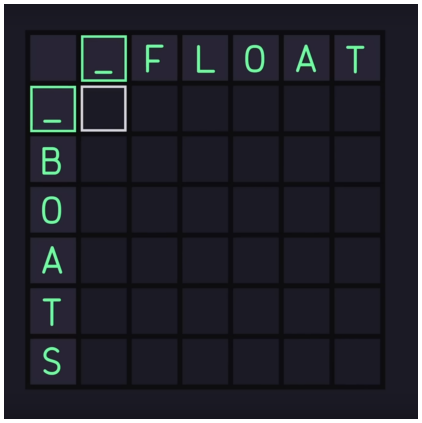

Figure 1: Labelling matrix

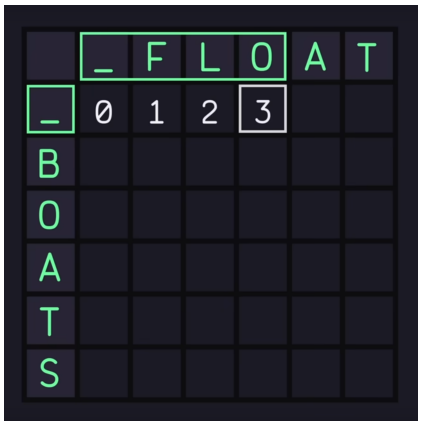

Figure 2: Filling edges

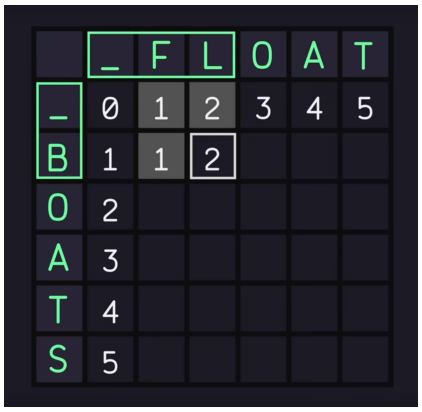

Figure 3: Addition from left or substitution from diagonal

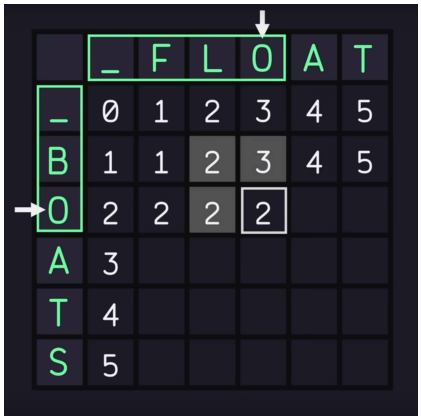

Figure 4: Substitution from diagonal but end at same character

credit for the photos: b001

Weighted Algorithm

Search results can be tuned to show favorable options by setting weights for addition, deletion and substitution. Special case in which the if an input string is in the movie name, its effective weight is decreased, giving a better result