The American Journal of Engineering and Technology
(ISSN – 2689-0984)
Published: June 30, 2021 | **Pages:** 134-139
Doi: https://doi.org/10.37547/tajet/Volume03Issue06-24

IMPACT FACTOR
2021: 5. 705
OCLC - 1121105677

Journal Website: http://theamericanjournals.com/index.php/tajet

Understanding The Levenshtein Distance Equation For Beginners

Abdurakhmanova Umida Rustamovna
Lecturer Of The Department Of Information Technologies, Tashkent State University Of Uzbek Language And Literature Named After Alisher Navoi, An Independent Seeker Of TATU, Uzbekistan

ABSTRACT

Levenshtein Distance is a way to ascribe a numeric distance between two sequences (often characters in a string or word) by counting the minimum number of insertion, deletion and substitution operations required to transform one sequence to the other. The following article looks into the general definition, explanation and application of the theory.

KEYWORDS

Levenshtein distance equation, algorithm, string matching, spelling checking, symbol.

INTRODUCTION

Editorial distance, or Levenshtein distance, is a metric that allows you to determine the "similarity" of two strings - the minimum number of operations to insert one character, delete one character and replace one character with another, necessary to turn one string into another. This article describes a method for calculating the editorial distance using a small amount of memory, without significant loss of speed. This approach can be applied for large strings (about 105 characters, that is, in fact, for texts) when obtaining not only an assessment of "similarity", but also a sequence of changes to transfer one string to another.

A little formalization

Let there are two rows S1 and S2. We want to translate one into another (let the first into the second, it is easy to show that the operations are symmetric) using the following operations:

I: Insert a character at any place;

D: Deleting a character from an arbitrary position;

R: Replaces a character with another.

Then d (S1, S2) is the minimum number of I / D / R operations for transferring S1 to S2, and the editorial instruction is a listing of operations for transfer with their parameters.

The problem is easily solved by the Wagner - Fisher algorithm, but O (| S1 | * | S2 |) memory cells are required to restore the editorial instruction. I briefly outline the algorithm itself, since the "optimization" is based on it.

Wagner - Fisher Algorithm

The desired distance is formed through the auxiliary function D (M, N), which finds the editorial distance for substrings S1 [0..M] and S2 [0..N]. Then the total editorial distance will be equal to the distance for full-length substrings: d (S1, S2) = DS1, S2 (M, N).

The self-evident fact is that:

D (0,0) = 0.

Indeed, the empty lines are the same anyway.

Also, the meanings for:

D (i, 0) = i;

D (0, j) = j.

Indeed, any string can turn out to be empty by adding the required number of required characters; any other operations will only increase the score.

In general, it's a little more complicated:

D (i, j) = D (i-1, j-1), if S1 [i] = S2 [j],

otherwise D (i, j) = min (D (i-1, j), D (i, j-1), D (i-1, j-1)) + 1.

In this case, we choose what is more profitable: delete the symbol (D (i-1, j)), add (D (i, j-1)), or replace (D (i-1, j-1)).

It is easy to understand that the estimation algorithm does not require more than two columns of memory, the current one (D (i, *)) and the previous one (D (i-1, *)). However, the full matrix is needed to restore the editorial order. Starting from the lower right corner of the matrix (M, N), we go to the upper left, at each step looking for the minimum of three values:

If the minimum is (D (i-1, j) + 1), add the removal of the symbol S1 [i] and go to (i-1, j);

If the minimum is (D (i, j-1) + 1), add the insertion of the symbol S1 [i] and go to (i, j-1);

If minimal (D (i-1, j-1) + m), where m = 1, if S1 [i]! = S2 [j], otherwise m = 0; after which we go to (i-1, j-1) and add a replacement if m = 1.

Here (i, j) is the cell of the matrix in which we are at this step. If two of the three values are minimum (or all three are equal), this means that there are 2 or 3 equivalent editorial prescriptions.

MATERIALS AND METHODS

As a result, it will take O (| S1 | * | S2 |) time and O (| S1 | * | S2 |) memory.

General reasons

The prerequisite for the creation of this method is a simple fact: in real systems, memory is more expensive than time. Indeed, for strings with a length of 216 characters, about 232 computational operations are needed (which with modern capacities will fall within ten seconds of computations) and 232 memory, which in most cases is already more than the size of the physical memory of the machine.

The idea of how to use the linear amount of memory (2 * | S1 |) to estimate the distance lies on the surface, it remains to competently transfer it to the calculation of the editorial prescription. If you think of transforming from sequence ‘a’ to sequence ‘b’ the first recursion represents a deletion from the ‘a’ sequence. The second recursion represents an insertion into the ‘b’ sequence, and the last recursion is either a substitution or a match (and skip over) depending on the equality check. The length is incremented in all cases except where there is a match.

When one sequence is exhausted (i.e. length=0) the length of the remaining sequence is added to the result (as either deletions or insertions depending which sequence is left).

Application

In approximate string matching, the objective is to find matches for short strings in many longer texts, in situations where a small number of differences is to be expected. The short strings could come from a dictionary, for instance. Here, one of the strings is typically short, while the other is arbitrarily long. This has a wide range of applications; for instance, spell checkers, correction systems for optical character recognition, and software to assist natural language translation based on translation memory.

The Levenshtein distance can also be computed between two longer strings, but the cost to compute it, which is roughly proportional to the product of the two string lengths, makes this impractical. Thus, when used to aid in fuzzy string searching in applications such as record linkage, the compared strings are usually short to help improve speed of comparisons.[citation needed]

In linguistics, the Levenshtein distance is used as a metric to quantify the linguistic distance, or how different two languages are from one another. It is related to mutual intelligibility, the higher the linguistic distance, the lower the mutual intelligibility, and the lower the linguistic distance, the higher the mutual intelligibility.

String Matching.

Spelling Checking.

Dynamic Programming Approach

The Levenshtein algorithm calculates the least number of edit operations that are necessary to modify one string to obtain another string. The most common way of calculating this is by the dynamic programming approach:

A matrix is initialized measuring in the (m, n) cell the Levenshtein distance between the m-character prefix of one with the n-prefix of the other word.

The matrix can be filled from the upper left to the lower right corner.

Each jump horizontally or vertically corresponds to an insert or a delete, respectively.

The cost is normally set to 1 for each of the operations.

The diagonal jump can cost either one, if the two characters in the row and column do not match else 0, if they match. Each cell always minimizes the cost locally.

This way the number in the lower right corner is the Levenshtein distance between both words.

Use Case

In approximate string matching, the objective is to find matches for short strings in many longer texts, in situations where a small number of differences are to be expected. The short strings could come from a dictionary, for instance. Here, one of the strings is typically short, while the other is arbitrarily long. This has a wide range of applications, for instance, spell checkers, correction systems for optical character recognition and software to assist natural language translation based on translation memory.

The Levenshtein distance can also be computed between two longer strings. But the cost to compute it, which is roughly proportional to the product of the two string lengths, makes this impractical. Thus, when used to aid in fuzzy string searching in applications such as record linkage, the compared strings are usually short to help improve the speed of comparisons.

There are other popular measures of edit distance, which are calculated using a different set of allowable edit operations. For instance, the Damerau–Levenshtein distance allows the transposition of two adjacent characters alongside insertion, deletion, substitution;

The longest common subsequence (LCS) distance allows only insertion and deletion, not substitution;

The Hamming distance allows only substitution; hence, it only applies to strings of the same length.

The Jaro distance allows only transposition.

Edit distance is usually defined as a parameterizable metric calculated with a specific set of allowed edit operations, and each operation is assigned a cost (possibly infinite). This is further generalized by DNA sequence alignment algorithms such as the Smith–Waterman algorithm, which make an operation's cost depend on where it is applied.

The famous Soviet and Russian mathematician Vladimir Iosifovich Levenshtein (by the way, who passed away a little over two months ago) at the beginning of the second half of the last century introduced the concept of editing distance, which we still use today in various fields - from search engines to bioinformatics. In this article, we will apply its principle for fuzzy search in MySQL (since MySQL does not offer a built-in solution at the moment), calculating the most efficient (i.e. fastest) method out of several found on the Internet, build an algorithm for such a search and implement it on PHP.

What to use:

Levenshtein distance and Damerau-Levenshtein distance: both represent the minimum number of operations for converting one string to another, differing in operations - Levenshtein proposed the operations of insertion, deletion and replacement of one character, and Damerau supplemented them with the operation of transposition, i.e. when two adjacent symbols are swapped; the following implementations have been proposed for MySQL:

Levenshtein-style query by Gordon Lesti a custom function for calculating the Levenshtein distance, published in Get It Done With MySQL 5 & Up (ed. by Peter Brawley and Arthur Fuller), by Jason Rust custom function for calculating the Damerau-Levenshtein distance, based on a C function by Linus Torvalds, by Diego Torres Oliver's algorithm: calculates the similarity of two strings in PHP represented by the function similar_text metaphone: phonetic indexing algorithm, works only with letters of the English alphabet in PHP represented by the metaphone function.

Fuzzy search algorithm

DISCUSSIONS AND RESULTS

Obviously, it makes no sense to calculate the Levenshtein distance between the entered word and each word from the dictionary in the database for each search, since it will take a lot of time. By the way, a few years ago on Habré a method was described in which, with each search, the entire dictionary from the database was driven into a PHP array, transliterated, and then similar words were selected, alternately using the levenshtein function, the megaphone, the similar text, or two at once. The decision of preliminary rapid filtering and subsequent refining of the found options suggests itself.

Thus, the essence of the fuzzy search algorithm can be summarized as follows:

Calculate the metaphone of the search term.

Find all words in the dictionary in the database by metaphone with Levenshtein (or Damerau-Levenshtein) distance <2 characters.

If nothing was found, the user made too many mistakes in the word, we stop tormenting the database and write that the user goes to the bathhouse, nothing was found.

If 1 word is found, return it.

If> 1 word is found, we refine it: find the percentage of similarity of the search query with each found word from the dictionary in the database; we find the maximum percentage of similarity; return all words with this percentage (in case several words have the same percentage, which will be the maximum).

Each search will need to calculate the Levenshtein distance. To do this, you need to find the fastest implementation of the algorithm for MySQL.

The fastest was the implementation of the Damerau-Levenshtein distance, written by

Linus Torvalds in C and adapted by Diego Torres for MySQL as a user-defined function. In second place, with a small time difference, is a primitive imitation of the Levenshtein distance in the form of an SQL query with a large number of LIKE operators, by Gordon Flattery. In third place, the user-defined function for MySQL from Jason Rats is far behind.

CONCLUSION

In conclusion, it can be added that it is necessary to use the calculation of the Levenshtein distance in MySQL in production only in cases where the string to be compared with is short, and the table with the words to be compared with the string is small. Otherwise, a possible solution in the case of a table with a large vocabulary may be to divide it into several tables, for example, by the first letter, or by the length of a word or its megaphone.

REFERENCES

1. Goday berdieva G.X. Nationality: problem and solutions.// Monograph. Publishing house" surkhon-edition " Termez-2019 y. 300 p.
2. Goday berdieva G.X. The basis of the National Goya.// Brochure. Publishing house" surkhon-edition " Termez-2016 y. 80 p.
3. Goday berdieva G.X formation and evolution of classical ideologies.// Brochure. Publishing house" Tafakkur " Tashkent-2016 44 p.
4. Godayberdieva G.X traditional education and youth.// Brochure. Publishing house" surkhon-edition " Termez-2016 y. 60p.