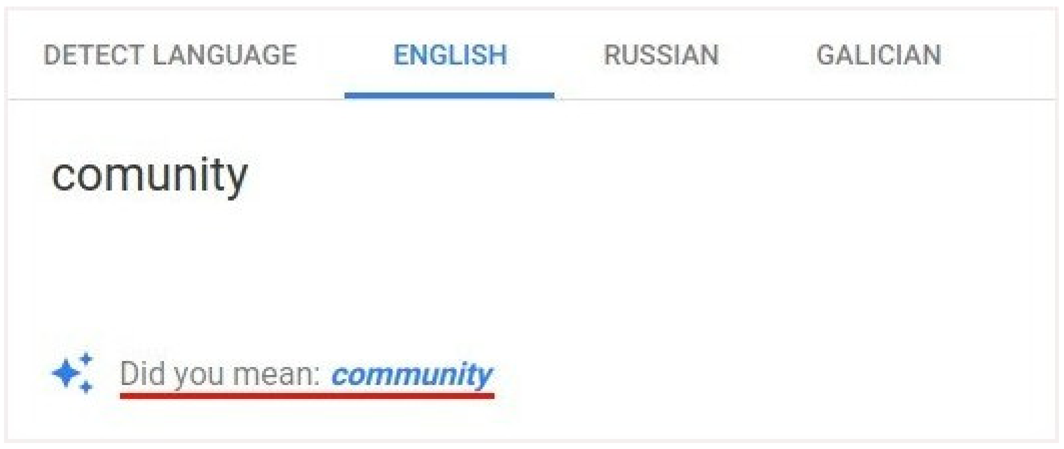
Theory: Edit distance

§1. Introduction

Sometimes people make a spelling mistake while typing a word. Fortunately, almost all text editing interfaces nowadays have an autocorrection function:



Have you ever thought about autocorrection algorithms? One of the possible approaches to this problem is to create a vocabulary of correct words and, for every received word with a typo, find the most similar one there and use it. The only problem here is to come up with a method allowing to estimate the similarity between two words. One of the widely used metrics for comparing two arbitrary strings is called **Edit Distance**. The metric has many uses: it finds applications in spelling mistakes correction (as described above) and in bioinformatics to measure the similarity between two DNA sequences and in other areas. The metric will be the main focus of this topic, so let's examine it in more detail.

§2. Definition

For two strings s and t, the edit distance $d_E(s,t)$ is the minimal number of operations required to transform s into t. Available operations are:

- **Insertion**: rat $\to$ rapt;
- **Deletion**: rat $\to$ at;
- **Substitution**: rat $\to$ cat.

For example, the word **rat** can be transformed into **arm** with three operations:

$$\text{rat} \to \text{at} \to \text{art} \to \text{arm}.$$

First, we delete the letter **r**, then we insert **r** after **a**, and finally, we substitute **t** with **m**.

Now, let's consider an algorithm for finding the edit distance between two arbitrary strings s and t.

§3. Calculating the edit distance

An algorithm for edit distance calculation uses a so-called **dynamic programming** technique. The idea of this approach is that we split the initial problem into smaller ones, find the answers for them and then combine the answers to get the solution for the initial problem. Let's see how we can calculate edit distance with this approach.

For strings s and t, let's denote $d_{i,j}$ as the edit distance for a prefix $s[0..i]$ of s and a prefix $t[0..j]$ of t:

$$d_{i,j} = d_E(s[0..i], t[0..j]).$$

Then, let's try to calculate $d_{i,j}$ assuming that we already know the answers for all smaller prefixes of $s[0..i]$ and $t[0..j]$. To do this, consider the possible variants of transformation of $s[0..i]$ into $t[0..j]$:

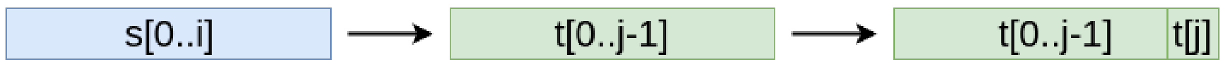
Current topic:
Edit distance

Topic depends on:
✓ String basics
✗ Dynamic programming basics

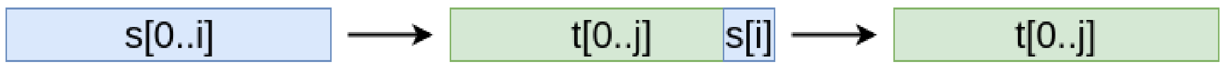
Topic is required for:
Edit distance alignment
Edit distance in Java

Table of contents:

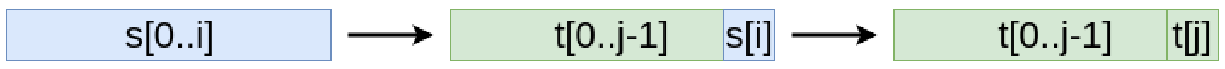
1. $s[0..i]$ may be transformed into $t[0..j-1]$ first, then the last symbol $t[j]$ needs to be inserted:



2. $s[0..i-1]$ may be transformed into $t[0..j]$ first, then the last symbol $s[i]$ needs to be deleted:



3.$s[0..i-1]$ may be transformed into $t[0..j-1]$ first, then $s[i]$ needs to be substituted by $t[j]$ if $s[i]$ and $t[j]$ don't match:



Note that we assume the first part of each transformation is optimal. Therefore, at least one of the final results will be optimal too.

Since we don't know in advance which operation gives the optimal result, we need to check all of them and then choose the best one. The formula for finding the best one is:

$$d_{i,j} = min\{d_{i-1,j}+1, d_{i,j-1}+1, d_{i-1,j-1}+match(s[i],t[j])\}\,(1)$$

Here, $match(s_1, s_2)$ is 0 if the symbols match and 1 otherwise.

To complete the algorithm description, we need to specify edit distance in the case when one of the strings is empty (let's denote it as e). In this case,

$$d_E(e,t) = d_E(t,e) = |t|,$$

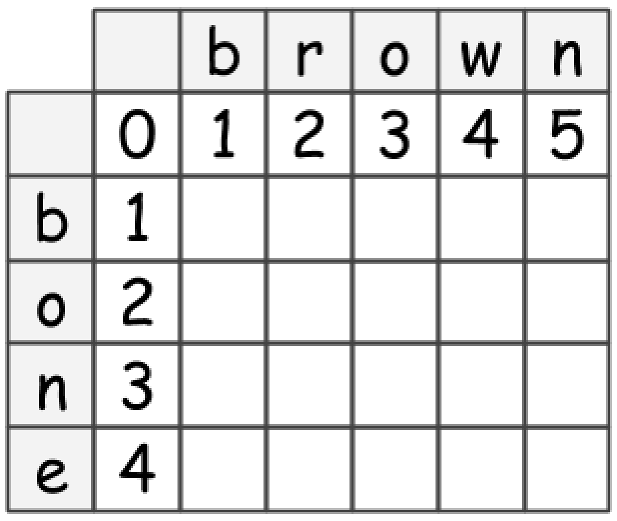
since to transform the empty string into t (or vice versa), one needs to perform exactly $|t|$ operations.

The formula (1) immediately gives us an idea how we can find the edit distance for two arbitrary strings s and t. We can start calculating the edit distance for smaller prefixes and move to the bigger ones to finally get the answer for the initial two strings.

§4. An example

Consider how the algorithm works for the strings "bone" and "brown". Intermediate answers will be stores in a 2d-array d, where $d[i][j] = d_E(s[0..i], t[0..j])$.

Initially, we need to fill the first row and the first column of the array:



		b	r	o	w	n
	0	1	2	3	4	5
b	1					
o	2					
n	3					
e	4					

The first row and the first column correspond to the transformation of the empty string to some prefix of s or t. So, $d[i][0] = i$ and $d[0][j] = j$.

Next, we can start calculating intermediate answers for the prefixes using the formula (1) (arrows outgoing from the green cells correspond to the optimal transformation on the current step):

		b	r	o	w	n
	0	1	2	3	4	5
b	1	0				
o	2					
n	3					
e	4					

		b	r	o	w	n
	0	1	2	3	4	5
b	1	0	1			
o	2					
n	3					
e	4					

		b	r	o	w	n
	0	1	2	3	4	5
b	1	0	1	2		
o	2					
n	3					
e	4					

Filling the table row by row, we finally get the following:

		b	r	o	w	n
	0	1	2	3	4	5
b	1	0	1	2	3	4
o	2	1	1	1	2	3
n	3	2	2	2	2	2
e	4	3	3	3	3	3

The lower right corner of the table corresponds to the edit distance for the initial strings. So, $d_E(\text{bone}, \text{brown}) = 3$.

For two arbitrary strings s and t, the algorithm for edit distance calculation is:

1. Create a $(|s| + 1) \times (|t| + 1)$ table to store intermediate answers.
2. Fill in the first row and the first column of the table (like in the example above).
3. Fill in the remaining cells using the formula (1).
4. Return a value in the lower right corner of the table as a final result.

§5. The complexity analysis

For strings s and t, the running time of the algorithm is $O(|s| \cdot |t|)$ since we calculate each value in a table of intermediate answers only once, and each such calculation requires a constant number of operations.

The table itself requires $O(|s| \cdot |t|)$ additional memory to be stored. However, one may note that to calculate $d[i][j]$, only the previous row of the table is required. Therefore, the algorithm can be implemented using only $O(min(|s|, |t|))$ additional memory.

Report a typo

25 users liked this theory. **8** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

This content was created almost 2 years ago and updated 5 days ago. Share your feedback below in comments to help us improve it!

Comments (3) Hints (0) Useful links (3) Show discussion