Get started

Open in app



## **Ethan Nam**

179 Followers - About Follow

# **Understanding the Levenshtein Distance Equation for Beginners**



Ethan Nam Feb 26, 2019 · 5 min read

I recently came across a situation where I needed fuzzy string matching functionality for a command line application I built. After googling a way to do this in Ruby, I found a Stack Overflow post telling me to install a ruby gem called *levenshtein-ffi* and be on my way. However, my curiosity got the best of me, and I went down an internet rabbit hole of trying to understand what is really going on behind the scenes. Ultimately, I landed on the <u>Levenshtein Distance Wikipedia page</u>, where I saw this:

$$\operatorname{lev}_{a,b}(i,j) = egin{cases} \max(i,j) & ext{if } \min(i,j) = 0, \ \min\left\{ egin{array}{ll} \operatorname{lev}_{a,b}(i-1,j) + 1 & ext{otherwise.} \ \operatorname{lev}_{a,b}(i,j-1) + 1 & ext{otherwise.} \ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i 
eq b_j)} & ext{What?} \end{cases}$$

I have minimal experience with matrices and have never taken Linear Algebra, so initially, I was bewildered. Eventually however, I was able to piece together an elementary understanding of what is happening, which I'll attempt to explain below. This explanation is meant for beginners — anyone who is confused by the equation above or has never taken higher level mathematics courses. **Warning**: if you don't fall into the above camp, you'll probably find the explanation below needlessly tedious.

## Introduction

The Levenshtein distance is a number that tells you how different two strings are. The higher the number, the more different the two strings are.

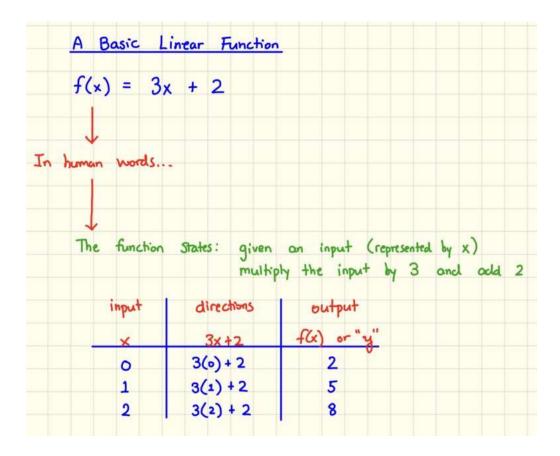
For example, the Levenshtein distance between "kitten" and "sitting" is 3 since, at a minimum, 3 edits are required to change one into the other.

- 1. **k**itten  $\rightarrow$  **s**itten (substitution of "s" for "k")
- 2. sitten  $\rightarrow$  sittin (substitution of "i" for "e")
- 3. sittin  $\rightarrow$  sitting (insertion of "g" at the end).

An "edit" is defined by either an insertion of a character, a deletion of a character, or a replacement of a character.

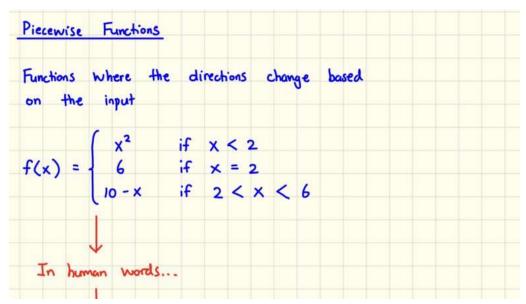
#### **Functions**

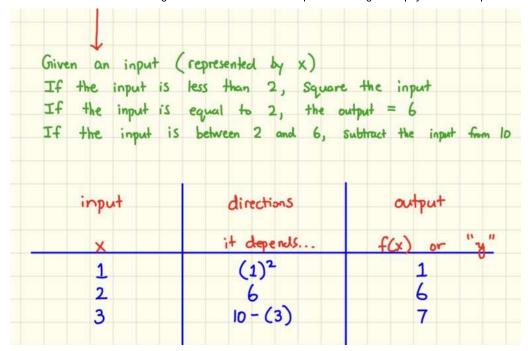
A quick refresher if you haven't looked at functions recently... The first thing to understand is that functions are a set of instructions that take a given input, follow a set of instructions, and yield an output. You probably saw lots of basic functions in your high school math courses.



## **Piecewise Functions**

Piecewise functions are more complex functions. In a piecewise function, there are multiple sets of instructions. You choose one set over another based on a certain condition. Consider the example below:

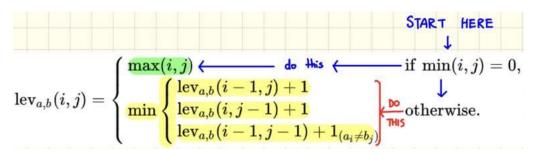




In the above example, we use different sets of instructions based on what the input is. Piecewise function are denoted by the brace { symbol.

With that in mind, the Levenshtein Distance equation should look a little more readable.

$$\operatorname{lev}_{a,b}(i,j) = egin{cases} \max(i,j) & ext{if } \min(i,j) = 0, \ \min egin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \ \operatorname{lev}_{a,b}(i,j-1) + 1 \ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i 
eq b_j)} \end{cases} & ext{otherwise.} \end{cases}$$



In other words..

# What do a, b, i, and j stand for?

a = string #1

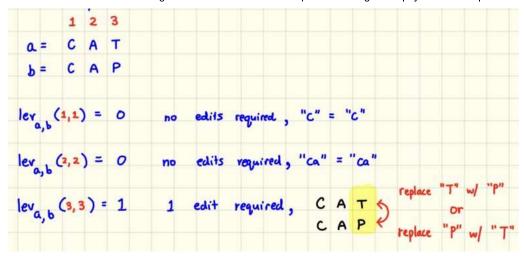
b = string #2

i = the terminal character position of string #1

j =the terminal character position of string #2.

https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0

The positions are 1-indexed. Consider the below example where we compare string "cat" with string "cap":



## The conditional $(a_i \neq b_i)$

ai refers to the character of string a at position i

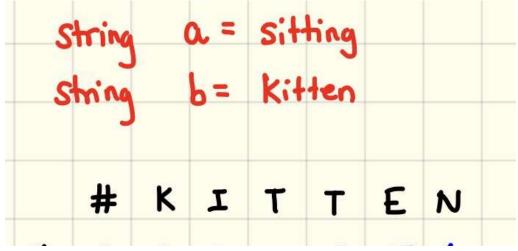
b<sub>i</sub> refers to the character of string b at position j

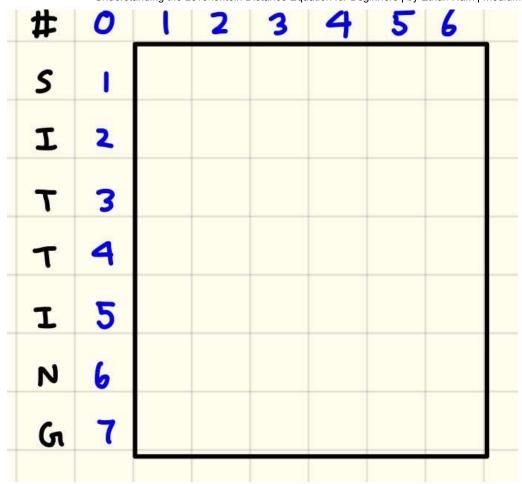
We want to check that these are not equal, because if they are equal, no edit is needed, so we should not add 1. Conversely, if they are not equal, we want to add 1 to account for a necessary edit.

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \operatorname{lev}_{a,b}(i-1,j) + 1 & \text{otherwise.} \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} & \text{we only add 1 if ...} \\ \text{the character of string a, pos i} \neq \text{the character of string b, pos j} \end{cases}$$
 If  $a_i = b_j$ , that means the characters are the some, so we shouldn't increment the edit number.

### Solving Using a Matrix

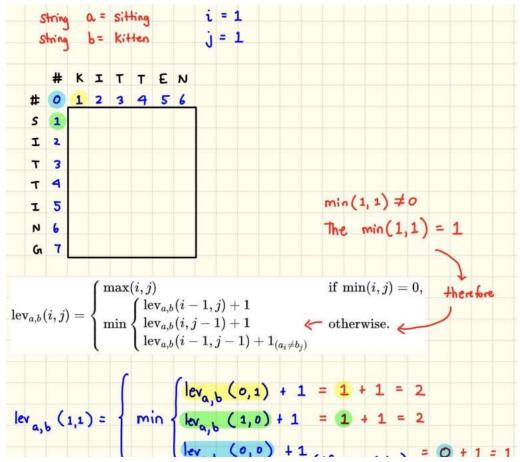
The Levenshtein distance for strings A and B can be calculated by using a matrix. It is initialized in the following way:

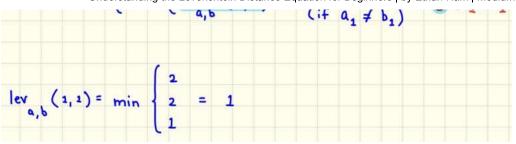


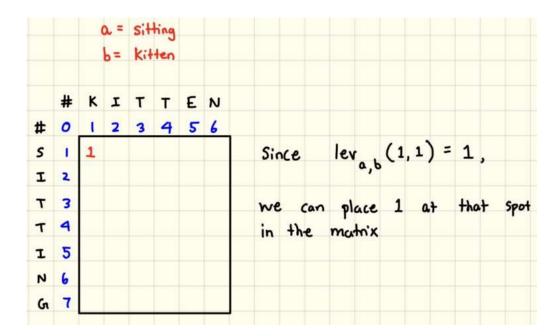


From here, our goal is to fill out the entire matrix starting from the upper-left corner. Afterwards, the bottom-right corner will yield the Levenshtein distance.

Let's fill out the matrix by following the piecewise function.

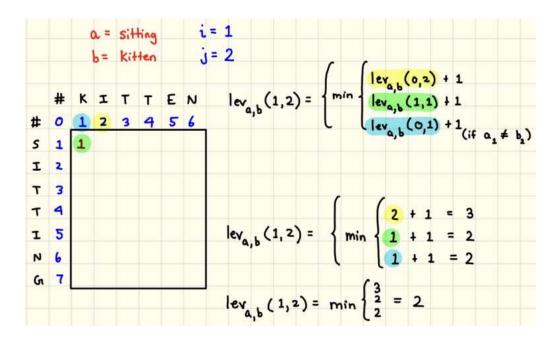


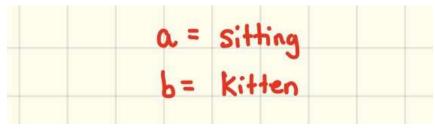


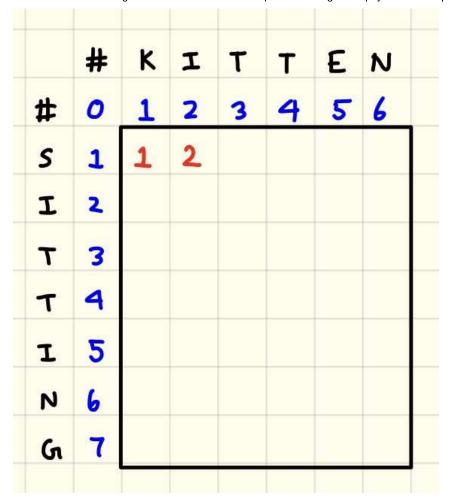


Now we can fill out the rest of the matrix using the same piecewise function for all the spots in the matrixes.

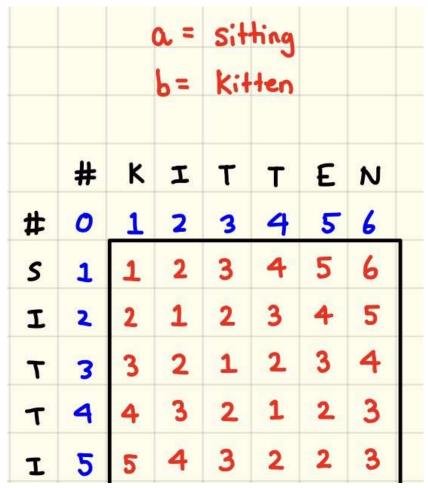
## One more example:







If you feel comfortable with the equation at this point, try to fill out the rest of the matrix. The result is posted below:



1	6	6	5	4	3	3	2
G	7	7	6	5	4	4	3

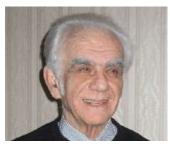
Since the lower-right corner is 3, we know the Levenshtein distance of "kitten" and "sitting" is 3. This is what we expected since we already knew the Levenshtein distance was 3 (as explained in the introduction). This is also reflected in the matrix shown on the Levenshtein Distance Wikipedia page.

		k	i	t	t	е	n
	0	1	2	3	4	5	6
S	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

#### Conclusion

$$\mathrm{lev}_{a,b}(i,j) = egin{cases} \max(i,j) & ext{if } \min(i,j) = 0, \ \min egin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \ \mathrm{lev}_{a,b}(i,j-1) + 1 \ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i 
eq b_j)} \end{cases}$$
 otherwise.

Hopefully, the above looks less intimidating to you now. Even better, I hope you understand what it means!





Vladimir Levenshtein, inventor of the Levenshtein algorithm

Now that you understand to a certain degree what is happening under the hood, you can really appreciate all that's happening when you download a gem like *levenshtein-ffi* and run a method like Levenshtein.distance("kitten", "sitting") and get a return value of 3. All the hard work has been abstracted away so that you get a simple numerical representation of how different two strings are. We should all thank Vladimir Levenshtein, who came up with his algorithm in 1965. The algorithm hasn't been improved in over 50 years and for good reason. <u>According to MIT</u>, it may very well be that Levenshtein's algorithm is the best that we'll ever get in terms of efficiency.

Works Used / Cited

#### The Levenshtein Algorithm

The Levenshtein distance is a string metric for measuring difference between two sequences. Informally, the Levenshtein...

www.cuelogic.com

#### Levenshtein distance - Wikipedia

In information theory, linguistics and computer science, the Levenshtein distance is a string metric for measuring the...

en wikipedia org

## Easy to understand Dynamic Programming - Edit distance

IMPORTANT: This blog has been moved to jlordiales.me. The content here might be outdated. To see this post on the new...

jlordiales wordpress.com

Programming Edit Distance Levenshtein Levenshtein Distance Fuzzy Logic

About Help Legal

Get the Medium app



