Understanding the Levenshtein Distance Equation for Beginners

Ethan Nam Follow
Feb 27, 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 Levenshtein Distance Wikipedia page, where I saw this:

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

What?

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.

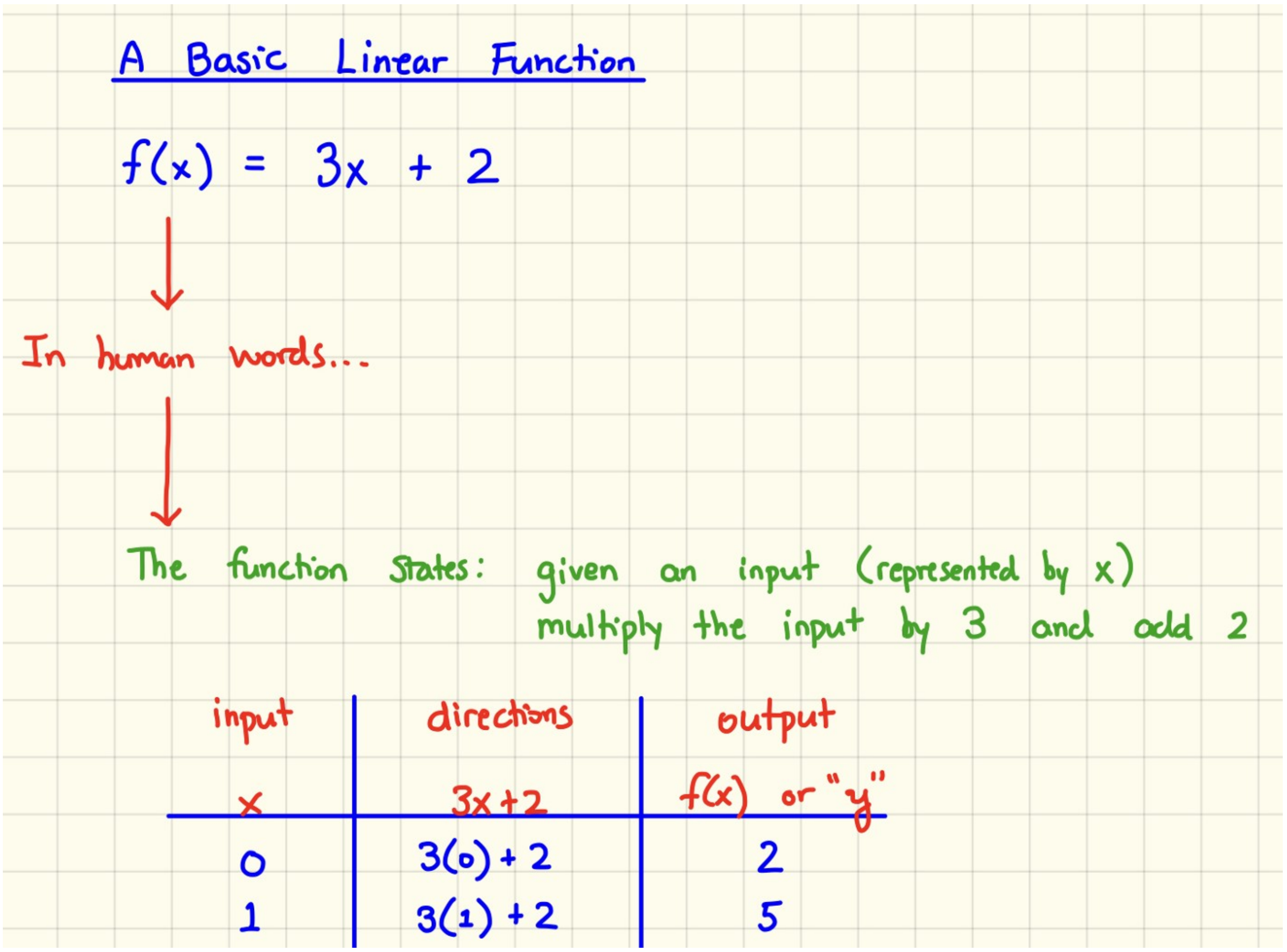
For example, the Levenshtein distance between “kitten” and “sitting” is 3 since, at a minimum, 3 edits are required to change one into the other.

1. **k**itten → **s**itten (substitution of “s” for “k”)
2. sitt**e**n → sitt**i**n (substitution of “i” for “e”)
3. sittin → sittin**g** (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.



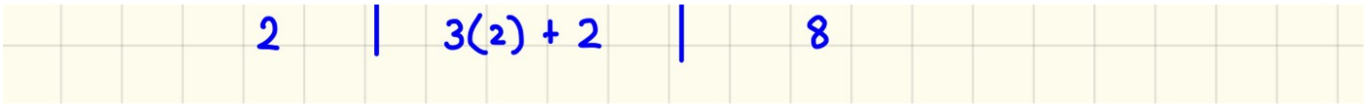
A Basic Linear Function

$f(x) = 3x + 2$

In human words...

The function states: given an input (represented by x) multiply the input by 3 and add 2

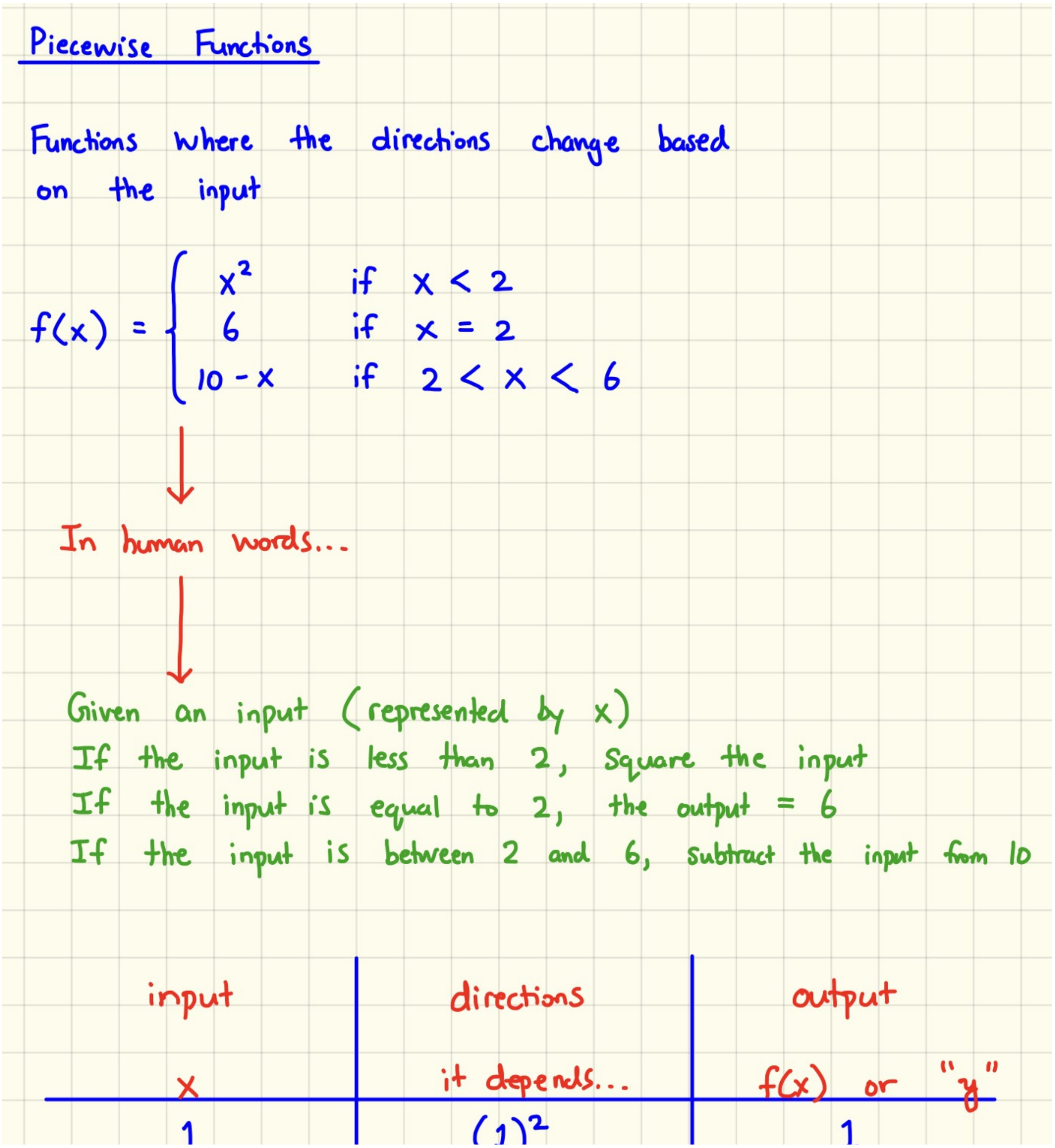
input	directions	output
x	$3x+2$	$f(x)$ or "y"
0	$3(0)+2$	2
1	$3(1)+2$	5



2	$3(2) + 2$	8

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:



Piecewise Functions

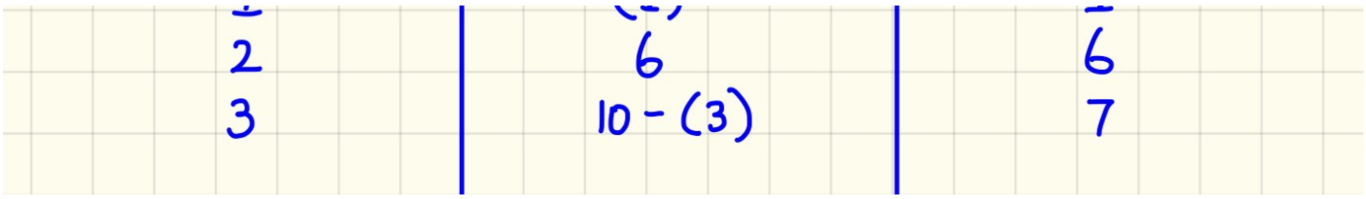
Functions where the directions change based on the input

$$f(x) = \begin{cases} x^2 & \text{if } x < 2 \\ 6 & \text{if } x = 2 \\ 10 - x & \text{if } 2 < x < 6 \end{cases}$$

In human words...

Given an input (represented by x)
If the input is less than 2, square the input
If the input is equal to 2, the output = 6
If the input is between 2 and 6, subtract the input from 10

input	directions	output
x	it depends...	f(x) or "y"
1	$(1)^2$	1



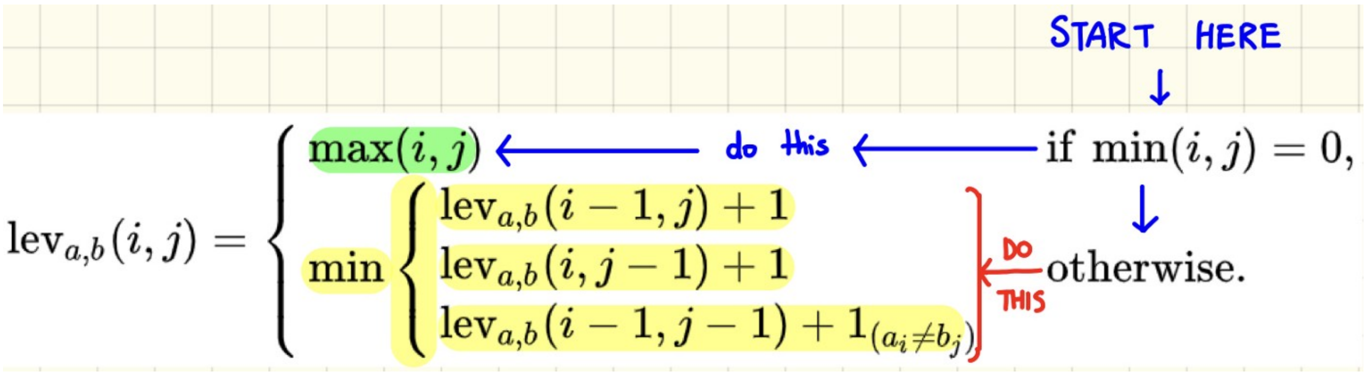
2	6	6
3	10 - (3)	7

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.

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

Original



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

In other words…

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

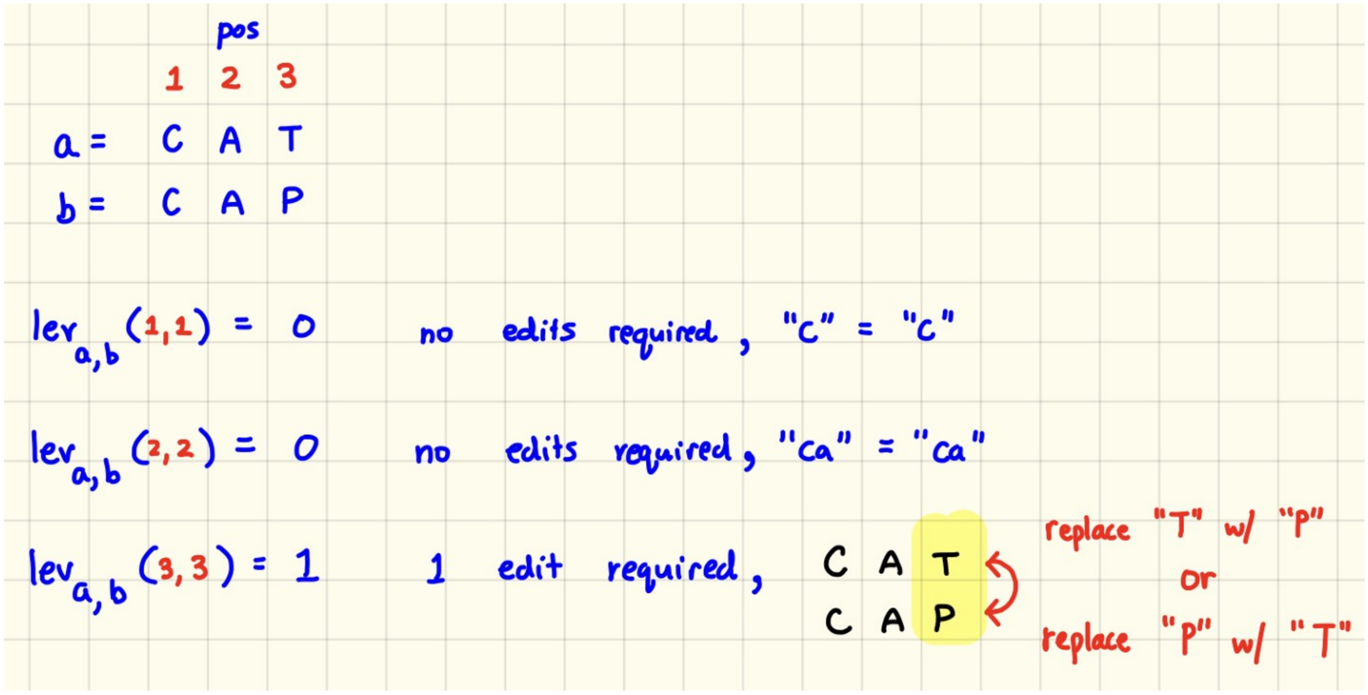
a = string #1

b = string #2

i = the terminal character position of string #1

j = the terminal character position of string #2.

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



$\text{lev}_{a,b}(1,1) = 0$ no edits required, "c" = "c"

$\text{lev}_{a,b}(2,2) = 0$ no edits required, "ca" = "ca"

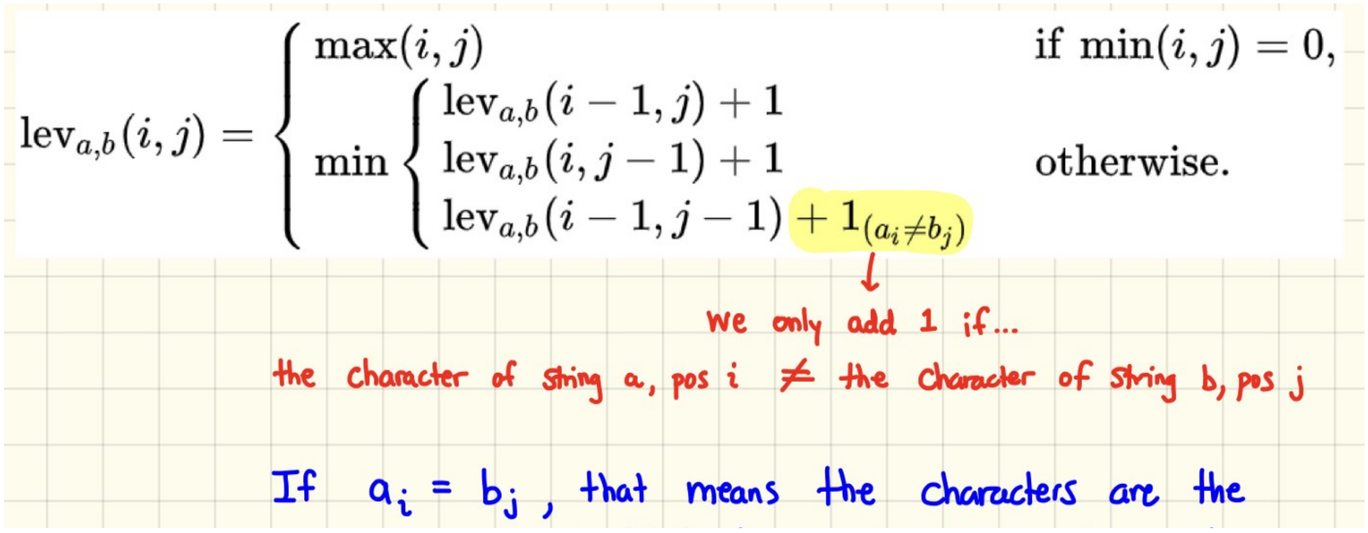
$\text{lev}_{a,b}(3,3) = 1$ 1 edit required,

The conditional ($a_i \neq b_j$)

a_i refers to the character of string a at position i

b_j 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.



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

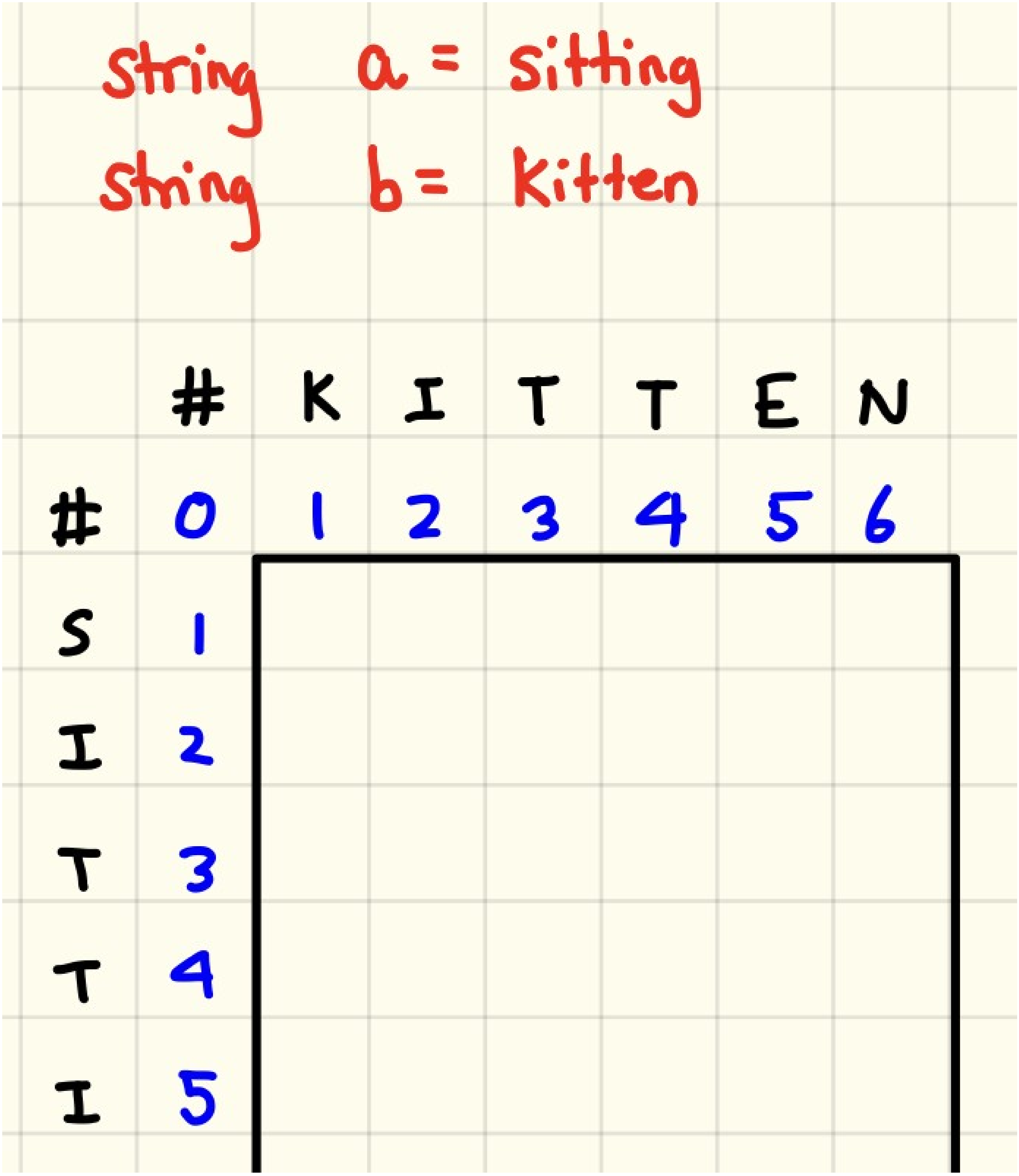
we only add 1 if...

the character of string a, pos i ≠ the character of string b, pos j

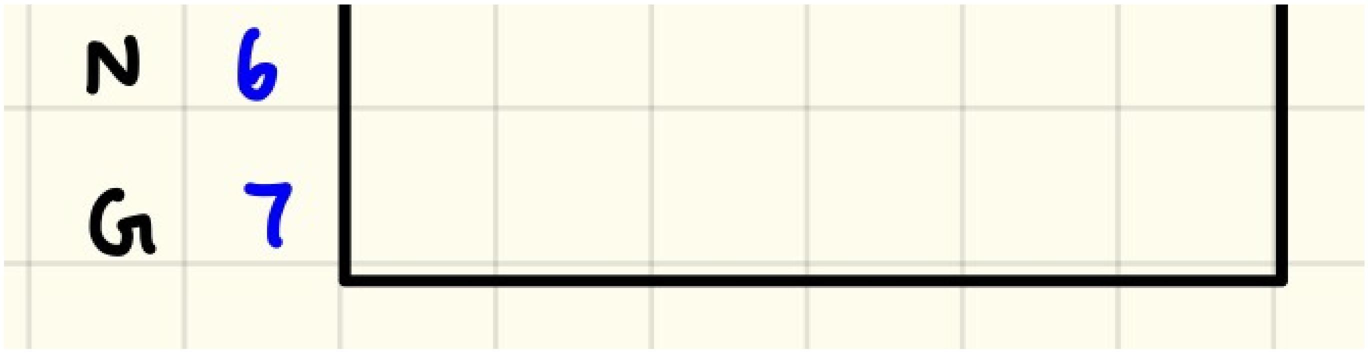
If $a_i = b_j$, that means the characters are the

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:

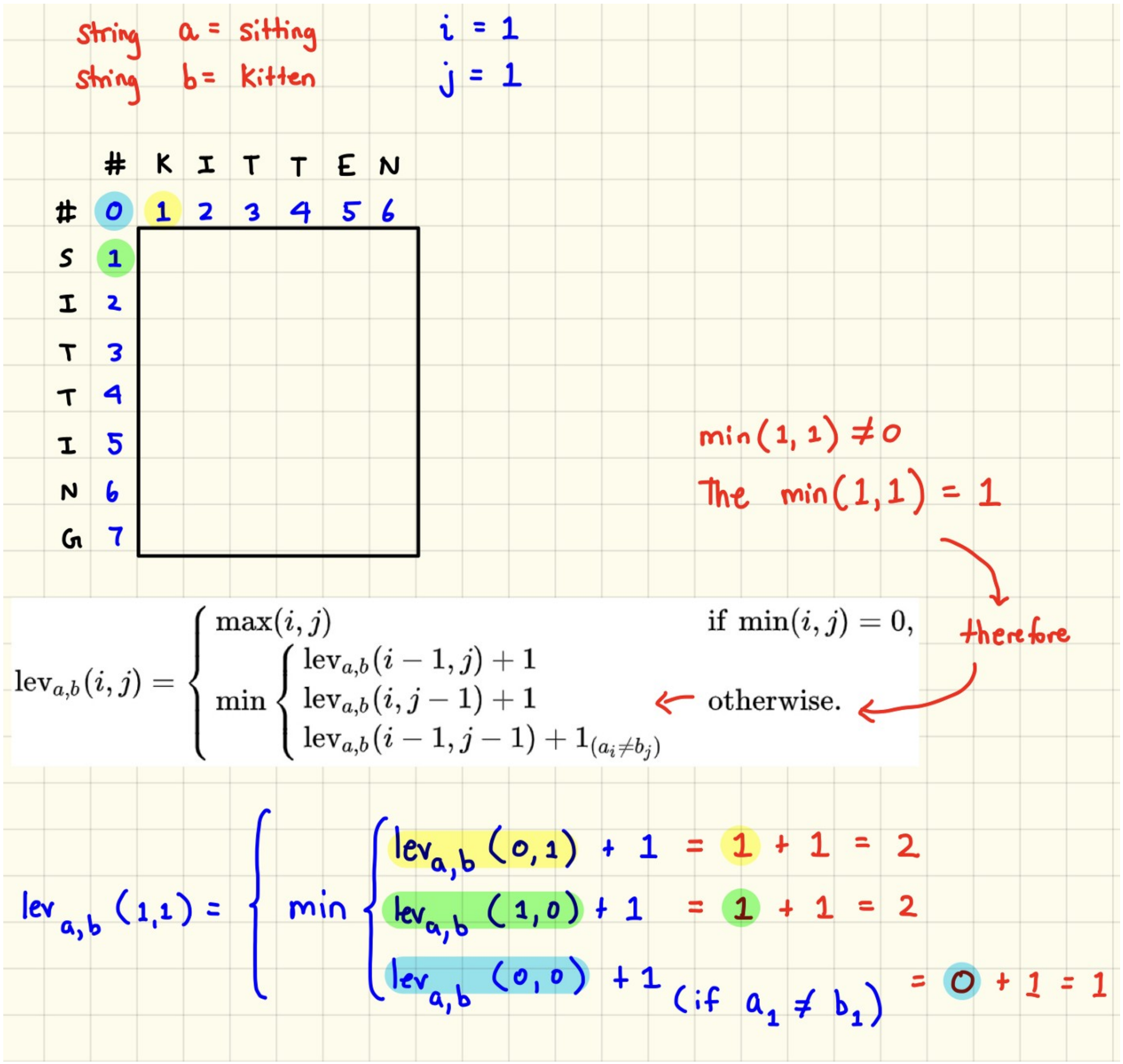

String a = sitting
String b = Kitten

$$
\begin{array}{c|ccccccc}
 & \# & K & I & T & T & E & N \\
\hline
\# & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
S & 1 & & & & & & \\
I & 2 & & & & & & \\
T & 3 & & & & & & \\
T & 4 & & & & & & \\
I & 5 & & & & & &
\end{array}
$$



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.



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

$$\operatorname{lev}_{a,b}(1,1) = \left\{ \min \begin{cases} \operatorname{lev}_{a,b}(0,1) + 1 = 1 + 1 = 2 \\ \operatorname{lev}_{a,b}(1,0) + 1 = 1 + 1 = 2 \\ \operatorname{lev}_{a,b}(0,0) + 1 \text{ (if } a_1 \neq b_1) = 0 + 1 = 1 \end{cases} \right.$$

$$\operatorname{lev}_{a,b}(1,1) = \min \begin{cases} 2 \\ 2 \\ 1 \end{cases} = 1$$

$a =$ sitting

$b =$ kitten

$$\begin{array}{c|c|cccccc} & \# & K & I & T & T & E & N \\ \# & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline S & 1 & 1 & & & & & \\ I & 2 & & & & & & \\ T & 3 & & & & & & \\ T & 4 & & & & & & \\ I & 5 & & & & & & \\ N & 6 & & & & & & \\ G & 7 & & & & & & \end{array}$$

Since $\operatorname{lev}_{a,b}(1,1) = 1$,

we can place 1 at that spot in the matrix

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

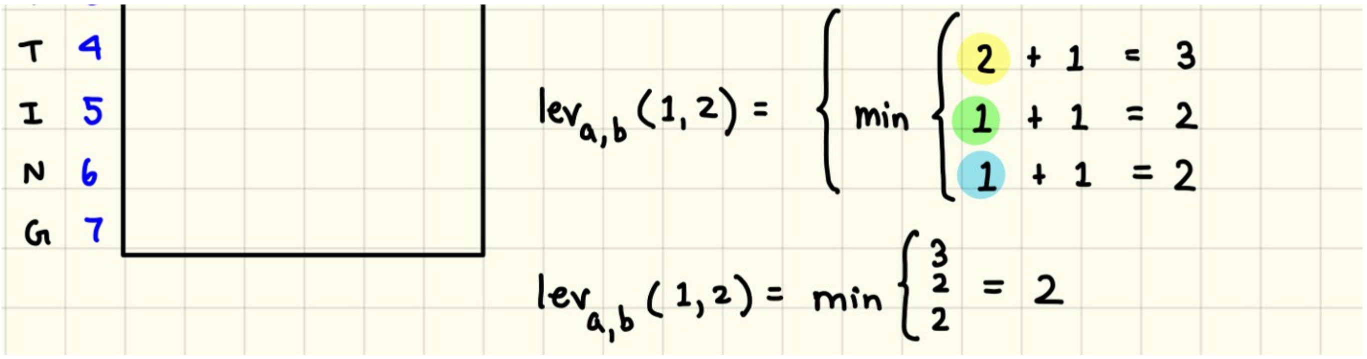
One more example:

$a =$ sitting $\quad i = 1$

$b =$ kitten $\quad j = 2$

$$\begin{array}{c|c|cccccc} & \# & K & I & T & T & E & N \\ \# & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline S & 1 & 1 & & & & & \\ I & 2 & & & & & & \\ T & 3 & & & & & & \end{array}$$

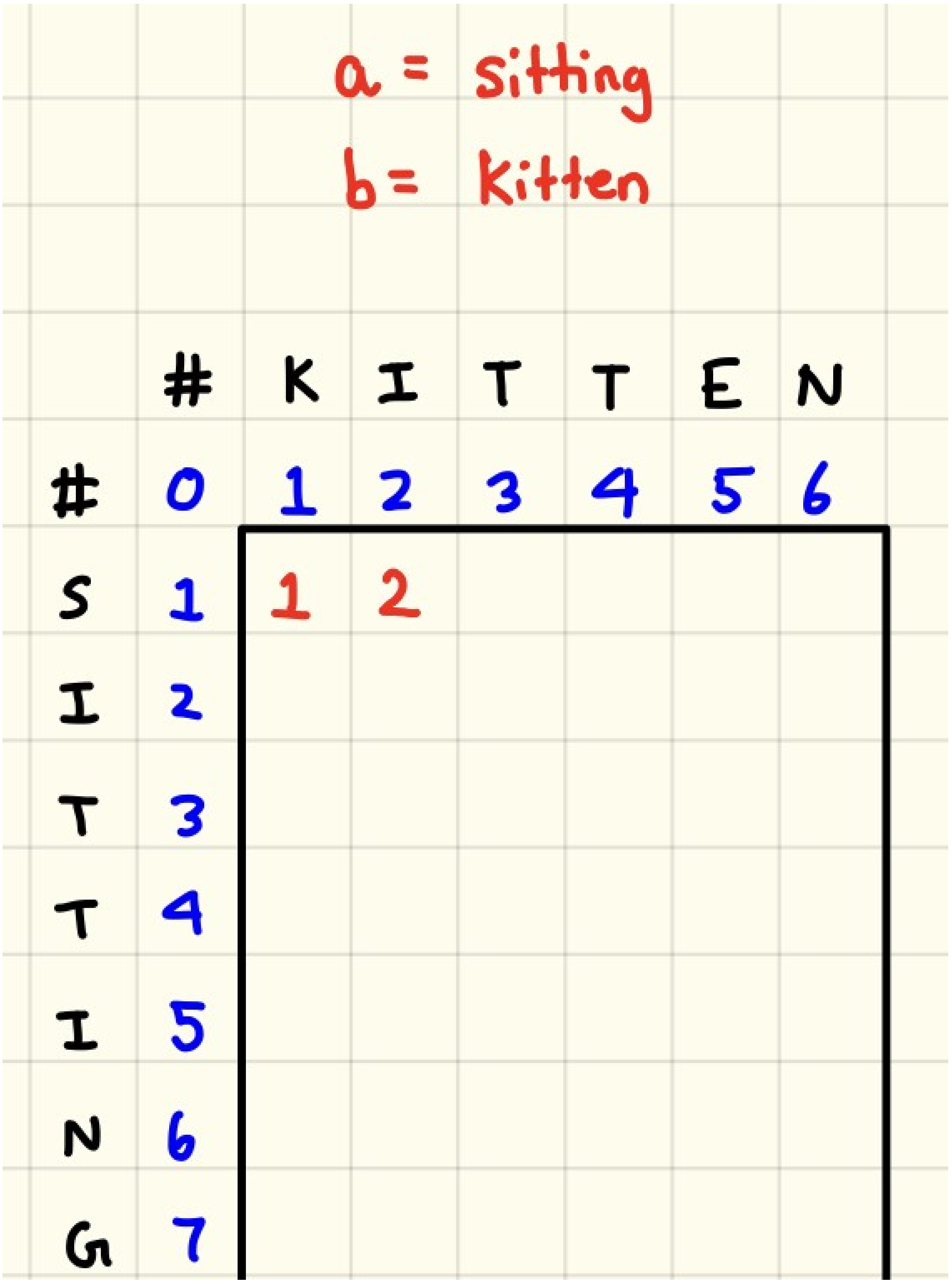
$$\operatorname{lev}_{a,b}(1,2) = \begin{cases} \min \begin{cases} \operatorname{lev}_{a,b}(0,2) + 1 \\ \operatorname{lev}_{a,b}(1,1) + 1 \\ \operatorname{lev}_{a,b}(0,1) + 1_{(\text{if } a_1 \neq b_2)} \end{cases} \end{cases}$$



T	4	
I	5	
N	6	
G	7	

$$lev_{a,b}(1,2) = \begin{cases} \min \begin{cases} 2 + 1 = 3 \\ 1 + 1 = 2 \\ 1 + 1 = 2 \end{cases} \end{cases}$$

$$lev_{a,b}(1,2) = \min \begin{cases} 3 \\ 2 \\ 2 \end{cases} = 2$$

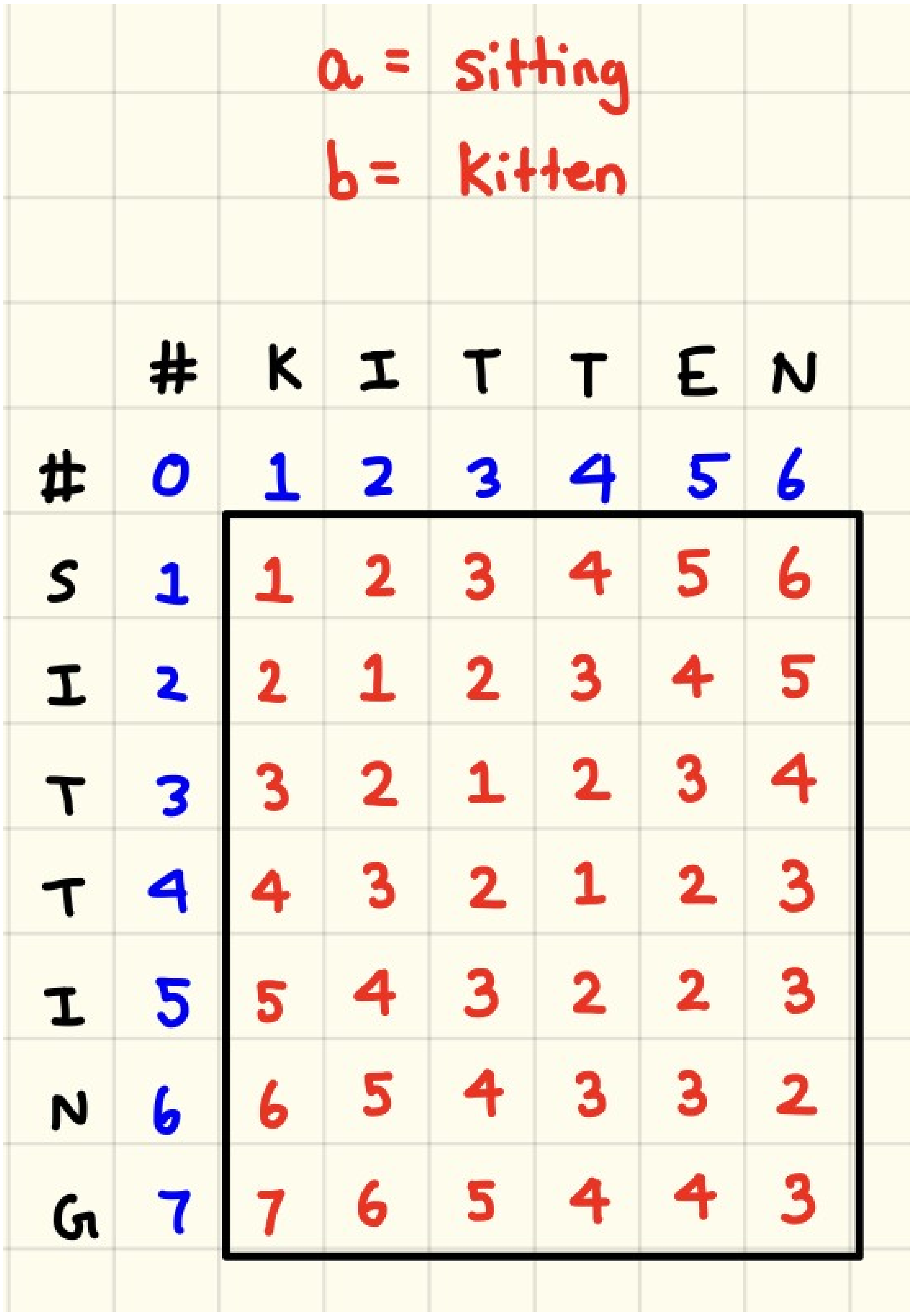


a = sitting

b = kitten

	#	K	I	T	T	E	N
#	0	1	2	3	4	5	6
S	1	1	2				
I	2						
T	3						
T	4						
I	5						
N	6						
G	7						

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

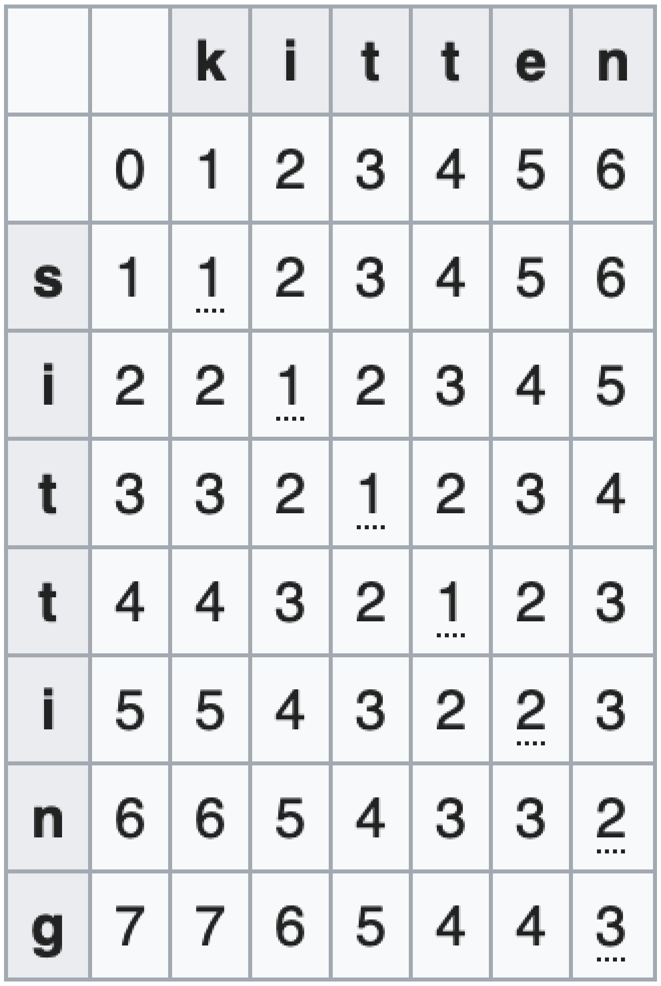


a = sitting

b = kitten

	#	K	I	T	T	E	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

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.



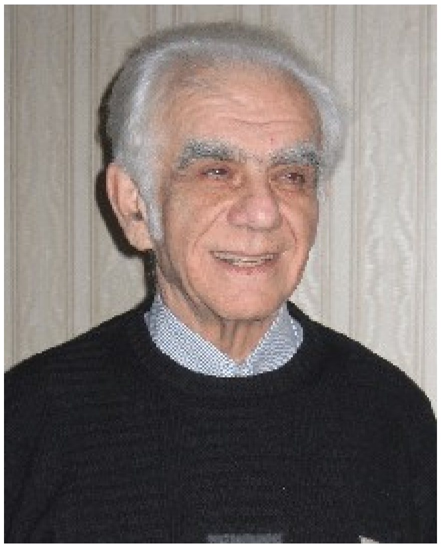
		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Conclusion

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

Oh, I see!

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. According to MIT, 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

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