Minimum Edit Distance

Notebook:	Stanford NLP Book		
Created:	30-07-2020 19:14	**Updated:**	31-07-2020 03:16
Author:	kopalsharma2000@gmail.com		
URL:	https://www.google.co.in/search?sxsrf=ALeKk01REyJPHkFn6YTDsr_xadnhv9O6Mg%3A1...		

Application
Autocorrect
Find error rate in speech recognition
Machine Translation
Identifying if two strings are coreferent
(In linguistics, **coreference**, sometimes written co-reference,
occurs when two or more expressions in a text refer to the same person or thing;
they have the same referent)

Minimum Edit Distance - defined as the minimum number of editing operations needed to transform one string to another
This can be done by visualizing the alignment of two strings. Placing one string below the other
and seeing which letters corresponds to which letter (or does it corresponds to an empty string)
Below it an operation list is written which tells the transformation needed in each letter alignment
to covert above string to the below one.

There is a way to assign cost to the above way. It's called the Levenshtein Distance. Actually there are two ways he
proposed. They are:-

1. To assign every operation (deletion substitution and insertion a cost of 1 and keeping the letter same a cost 0)
2. To not allow substitution but just insertion and deletion and assign in cost 1. If you think about it substitution will

   be there but it will be cost 2 (as substitution = one deletion(1) and one insertion (1) = 2)

The Minimum Edit Distance Algorithm
Use dynamic programming for this. Dynamic programming suggests that the a large problem can be solved by combining
solutions to various sub problems.
Edit distance between two strings X and Y of length n and m respectively is represented by D[n,m]
The value of D[i; j] is computed by taking the minimum of the three possible paths through the matrix which
arrive there:

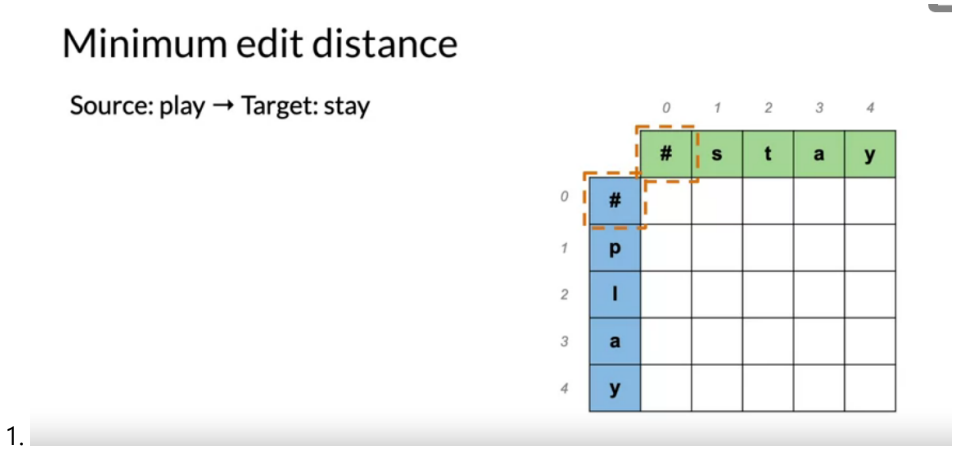
$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

If del-cost = ins-cost = 1 and sub-cost = 2 then

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

Vertibri Algorithm - extension to minimum edit distance algorithm except it computes the "maximum probability alignment" of one string
with another.

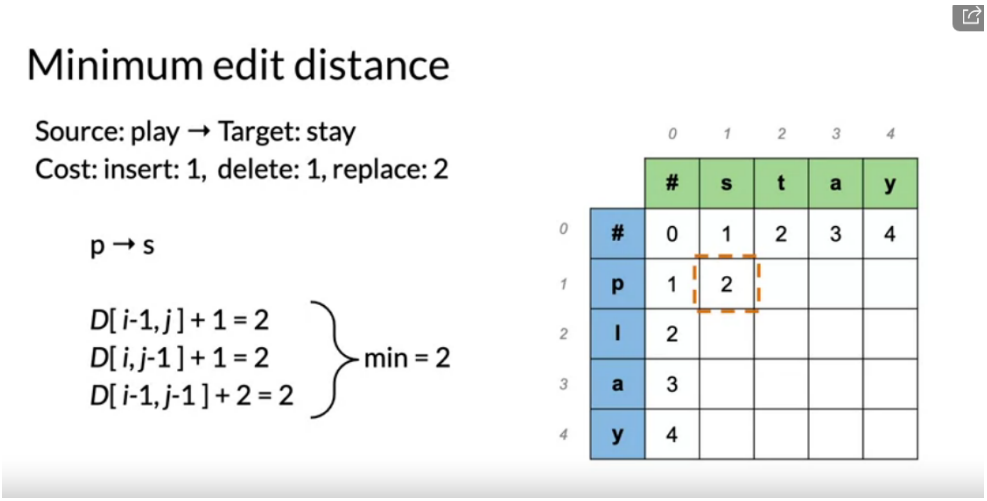
Below are the steps of how Dynamic Programming works :-

1. 

There hashes are used as a convention in the starting. Now in the cell (0,0) value 0 will be filled as the edit distance between empty string of source
and empty string of target is the 0. Similarly now in cell (1,0) you delete p to convert into empty string so edit distance is 1 and in cell (0,1) to insert s
in empty string the edit distance is again 1 so you fill those values in there respectively. Similarly all cells are filled. It is not necessary that all the
cells are needed to be done for the transformation but we still fill it to find the optimal edit distance.

2.

First we fill the leftmost column and the top row (the insertion and deletion one and the formula for the same is generalized in the D[i,j]



every cell uses the same formula. to explain the formula. there are 3 ways to calculate edit distance for the cell. One through row, one
through column and one direct replacement. The minimum distance from all three methods in finally inserted in the cell.

Urging all the readers to fill this table by themselves and verify from below

Minimum edit distance

Source: play → Target: stay
Cost: insert: 1, delete: 1, replace: 2

play → stay

D[*m*, *n*] = 4

		0	1	2	3	4
		#	s	t	a	y
0	#	0	1	2	3	4
1	p	1	2	3	4	5
2	l	2	3	4	5	6
3	a	3	4	5	4	5
4	y	4	5	6	5	4

What is Backtrace?
Sometimes its not enough just to find the minimum edit distance. So it's important to keep a backtrace of how you got there
at that edit distance by keeping pointers in the table.