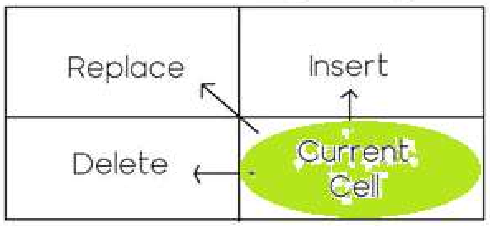
$$dp[i][j] = \begin{cases} dp[i\text{-}1], \text{ if } A[i] == B[j] \\ 1{+}\min\begin{pmatrix} dp[i,\text{-}1][j], \\ dp[i][j\text{-}1], \\ dp[i\text{-}1][j\text{-}1] \end{pmatrix} \text{ if } A[i]! = B[j] \end{cases}$$

Reference for Back tracing the changes



Lecture 24

Dynamic Programming (Edit Distance Problem): Edit, Edit Distance Designing DP Algorithm for Edit Distance Problem & its Time Complexity, and Applications Analysis of DP Edit Distance.

How similar are two strings?

- Spell correction
  - The user typed “graffe”

  Which is closest?
    - graf
    - graft
    - grail
    - giraffe
- Computational Biology
  - Align two sequences of nucleotides

    ```
    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC
    ```

  - Resulting alignment:

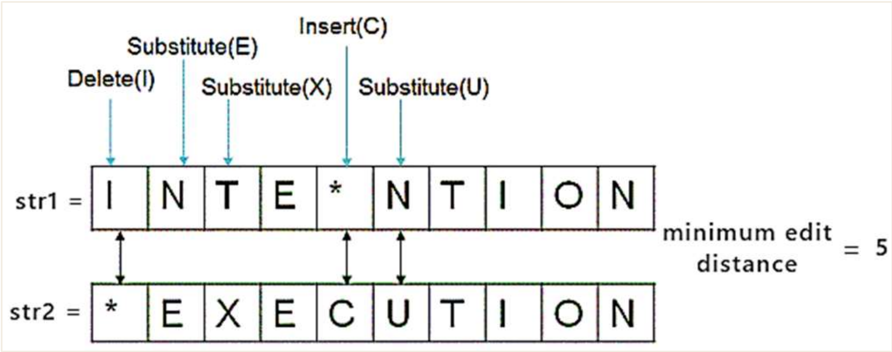
    ```
    -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
    TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
    ```

- Also for Machine Translation, Information Extraction, Speech Recognition

It has many applications, such as spell checkers, natural language translation, and bioinformatics.

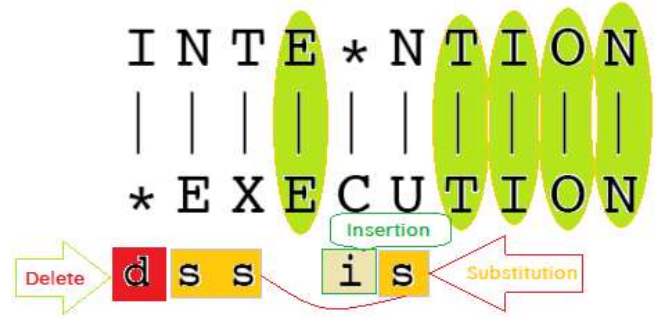
Minimum Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
  - Insertion
  - Deletion
  - Substitution
- Needed to transform one into the other



- Two strings and their **alignment**:

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
```



- If each operation has cost of 1
  - Distance between these is 5

Alignment in Computational Biology

- Given a sequence of bases

  AGGCTATCACCTGACCTCCAGGCCGATGCCC
  TAGCTATCACGACCGCGGTCGATTTGCCCGAC

- An alignment:

  -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
  TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC

- Given two sequences, align each letter to a letter or gap

Other Uses of Edit Distance

- Evaluating Machine Translation and speech recognition

R	Spokesman	confirms		senior	government	adviser	was	shot	
H	Spokesman	said	the	senior		adviser	was	shot	dead
		S	I		D				I

- Named Entity Extraction and Entity Coreference
  - IBM Inc. announced today
  - IBM profits
  - Stanford President John Hennessy announced yesterday
  - for Stanford University President John Hennessy

Problem Statement

$$dp[i][j] = \begin{cases} dp[i-1], \text{ if } A[i] == B[j] \\ 1+\min\begin{pmatrix} dp[i,-1][j], \\ dp[i][j-1], \\ dp[i-1][j-1] \end{pmatrix} \text{ if } A[i]! = B[j] \end{cases}$$

Given two strings S1 and S2, find out the minimum edit distance to transform S1 to S2. (Edit operations: insertion, deletion, substitution)

Example:- “kitten” to “sitting”

1. kitten -> sitten, (substitution)
2. sitten -> sittin, (substitution)
3. sittin -> sitting. (insertion)

Output:3

Example: Edit Distance

- We find the distance between the words **DOG** and **COW** using the basic Minimum Edit Distance algorithm.

- **Step 1**: Draw the edit distance matrix. In this, each word is preceded by # symbol which marks the empty string.

	#	C	O	W
#				
D				
O				
G				

Example: Edit Distance

› **Step 2**: Find the edit-distance values using minimum edit distance algorithm to convert # (row side) to **#COW** (column side) and populate appropriate cells with the calculated distance.

› Number of operations required to convert;

- # to # is 0.
- # to #C is 1. That is, one insertion (no change with #, insert C)
- # to #CO is 2. That is, two insertions (no change with #, insert C and O)
- # to #COW is 3. That is, three insertions (no change with #, insert C, O and W)

	#	C	O	W
#	0	1	2	3
D				
O				
G				

Example: Edit Distance

› **Step 3**: Find the edit-distance values using minimum edit distance algorithm to convert # (column side) to **#DOG** (row side) and populate appropriate cells with the calculated distance.

› Number of operations required to convert

- # to # is 0.
- # to #D is 1. [one insertion (no change with #, insert D]
- # to #DO is 2. [two insertions (no change #, insert D and O]
- # to #DOG is 3. [three insertions (no change #, insert D, O and G]

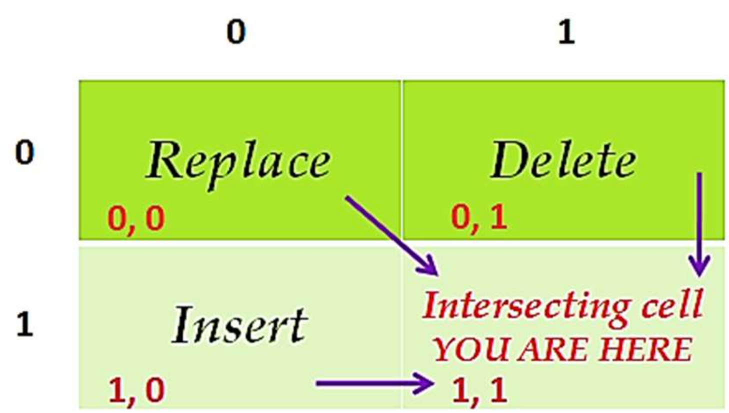
	#	C	O	W
#	0	1	2	3
D	1			
O	2			
G	3			

So far, we have found the minimum edit distance for 7 sub-problems.

[# - # = 0, # - #C = 1, # - #CO = 2, # - #COW = 3, # - #D = 1, # - #DO = 2, and # - #DOG = 3]

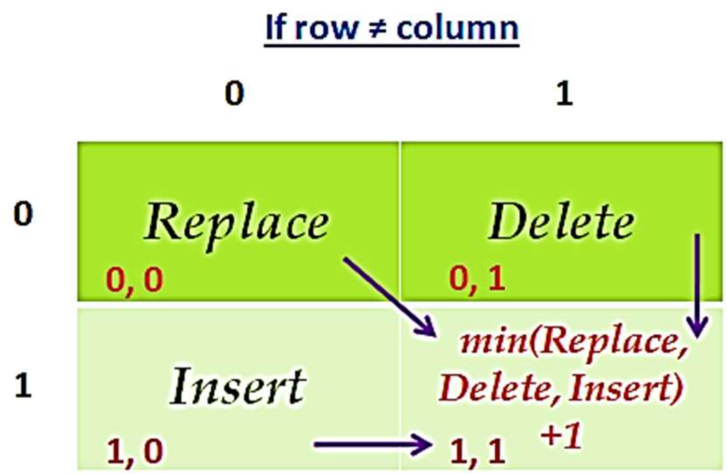
Example: Edit Distance

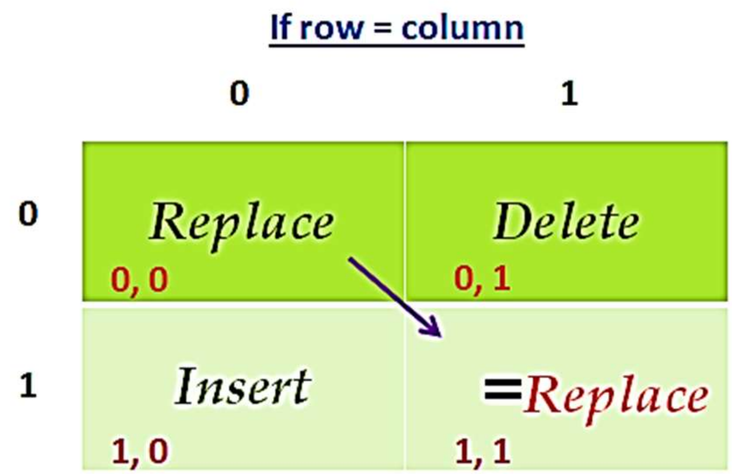
- **<u>Step 4</u>**: From this step onwards, we try to find the cost for a sub-problem by finding the minimum cost of three sub-problems and add 1 with that if the characters intersect at that cell are different. If the intersecting characters are same, then we add 0 with the diagonal cell value.
- [*Note: we have used A as the name for this matrix and included the index numbers for easy understanding*]
- Use the following table as **<u>a key to find the cost</u>**.



Example: Edit Distance

- If the ***row character ≠ column character***,
- The cost of the intersecting cell = min(replace, delete, insert).
- If the ***row character = column character***,
- The cost of the intersecting cell = cost of the Replace cell





Example: Edit Distance

Sub-problem: #D ➔ #C.

Intersecting cell: **A[1, 1]**

Same intersecting characters? NO.

Cost(A[1, 1]) = Min(A[0, 0], A[0, 1], A[1, 0]) + 1

= A[0, 0] + 1

= 0 + 1 = 1

	#	C	O	W
#	0	1	2	3
D	1	1		
O	2			
G	3			

As the cost derived from the cost of cell A[0, 0], a pointer is included from the current cell A[1, 1] to mark the back-trace path.

Sub-problem: #D ➔ #CO.

Intersecting cell: **A[1, 2]**

Same intersecting characters? NO.

Cost(A[1, 2]) = Min(A[0, 1], A[0, 2], A[1, 1]) + 1

= A[1, 1] + 1

= 1 + 1 = 2

	#	C	O	W
#	0	1	2	3
D	1	1	2	
O	2			
G	3			

As the cost derived from the cost of cell A[1, 1], a pointer is included from the current cell A[1, 2] to mark the back-trace path.

Example: Edit Distance

› ***Justification for choosing cell A[1, 1] as the previous cell***: We have two cells (A[0,1] and A[1, 1]) with the same minimum cost 1. Which one to choose? We choose A[1, 1]. The reasons are;

› We are trying to convert 'D' to 'CO' in this sub-problem

› We already used the ***substitution operation*** (substitute character D with C).

› So, the next character 'O' in the output can be derived with the ***insert operation*** only.

› Hence, we have chosen the cell A[1, 1] which represents **insert operation** (refer the **key table** above) as the minimum valued cell

Example: Edit Distance

<u>Sub-problem: #D → #COW.</u>

Intersecting cell: **A[1, 3]**

Same intersecting characters? NO.

Cost(A[1, 3]) = Min(A[0, 2], A[0, 3], A[1, 2]) + 1

= A[1, 2] + 1

= 2 + 1 = 3

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2			
G	3			

As the cost derived from the cost of cell A[1, 2], a pointer is included from the current cell A[1, 3] to mark the back-trace path.

Example: Edit Distance

Justification for choosing cell A[1, 2] as the previous cell: We have two cells (A[0, 2] and A[1, 2]) with the same minimum cost 1. Which one to choose? We choose A[1, 2]. The reasons are;

We are trying to convert 'D' to 'COW' in this sub-problem

We already used the ***substitution operation*** (substitute character D with C).

So, the next characters 'O' and 'W' in the output can be derived with the ***insert operation*** only.

Hence, we have chosen the cell A[1, 2] which represents **insert operation** (refer the **key table** above) as the minimum valued cell.

Example: Edit Distance

Sub-problem: #DO ➔ #C.

Intersecting cell: **A[2, 1]**

Same intersecting characters? NO.

Cost(A[2, 1]) $= \text{Min}(A[1, 0], A[1, 1], A[2, 0]) + 1$

$= A[1, 1] + 1$

$= 1 + 1 = 2$

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2		
C	3			

As the cost derived from the cost of cell A[1, 1], a pointer is included from the current cell A[2, 1] to mark the back-trace path.

Justification for choosing cell A[1, 1] as the previous cell: We have two cells (A[1, 0] and A[1, 1]) with the same minimum cost 1. Which one to choose? We choose A[1, 1]. The reasons are;

We are trying to convert 'DO' to 'C' in this sub-problem

We already used the *substitution operation* (substitute character D with C).

So, the next character 'O' has to be removed in the output. Hence, a *delete operation* used.

Hence, we have chosen the cell A[1, 1] which represents **delete operation** (refer the **key table** above) as the minimum valued cell.

Example: Edit Distance

Sub-problem: #DO ➔ #CO.

Intersecting cell: **A[2, 2]**

Same intersecting characters? YES

Cost(A[2, 2]) = Min(A[1, 1], A[1, 2], A[2, 1]) + 0

= A[1, 1] + 0

= 1 + 0 = 1

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2	1	
G	3			

As the cost derived from the cost of cell A[1, 1], a pointer is included from the current cell A[2, 2] to mark the back-trace path.

Sub-problem: #DO ➔ #COW.

Intersecting cell: **A[2, 3]**

Same intersecting characters? NO.

Cost(A[2, 3]) = Min(A[1, 2], A[1, 3], A[2, 2]) + 1

= A[2, 2] + 1

= 1 + 1 = 2

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2	1	2
G	3			

As the cost derived from the cost of cell A[2, 2], a pointer is included from the current cell A[2, 3] to mark the back-trace path.

Example: Edit Distance

Sub-problem: #DOG → #C.

Intersecting cell: **A[3, 1]**

Same intersecting characters? NO.

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2	1	2
G	3	3		

Cost(A[3, 1]) = Min(A[2, 0], A[2, 1], A[3, 0]) + 1

= A[2, 1] + 1

= 2 + 1 = 3

Justification for choosing cell A[3, 1] as the previous cell: We have two cells (A[2, 0] and A[2, 1]) with the same minimum cost 1. Which one to choose? We choose A[2, 1]. The reasons are;

We are trying to convert 'DOG' to 'C' in this sub-problem

We already used the *substitution operation* (substitute character D with C) and *delete operation* (deleted a character 'O'). **[refer sub-problem #DO → #C]**

So, the next character 'G' has to be removed in the output as well. Hence, a *delete operation* used.

Hence, we have chosen the cell A[2, 1] which represents delete operation (refer the **key table** above) as the minimum valued cell.

Example: Edit Distance

Sub-problem: #DOG ➔ #CO.

Intersecting cell: **A[3, 2]**

Same intersecting characters? NO

Cost(A[3, 2]) = Min(A[2, 1], A[2, 2], A[3, 1]) + 1

= A[2, 2] + 1

= 1 + 1 = 2

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2	1	2
G	3	3	2	

Sub-problem: #DOG ➔ #COW.

Intersecting cell: **A[3, 3]**

Same intersecting characters? NO.

Cost(A[3, 3]) = Min(A[2, 2], A[2, 3], A[3, 2]) + 1

= A[2, 2] + 1

= 1 + 1 = 2

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2	1	2
G	3	3	2	2

Example: Edit Distance

we need to traverse back from the last cell (the minimum edit distance between DOG and COW) using the pointer that starts at that cell.

The cost at cell A[3, 3] was derived from cell A[2, 2] by incrementing the cost at A[2, 2] due to the **substitution operation**.

The cost at cell A[2, 2] was derived from cell A[1, 1]. The cost of A[1, 1] is used as it is because of **No change**. (the column character = row character in the intersecting cell).

The cost at cell A[1, 1] was derived from cell A[0, 0] by incrementing the cost at A[0, 0] due to the **substitution operation**.

	#	C	O	W
#	0	1	2	3
D	1	1	2	3
O	2	2	1	2
G	3	3	2	2

Example: Edit Distance

Hence, the result of Minimum Edit Distance with alignment is;

D O G

| | |

C O W

Substitution *No change* *Substitution*

Edit Distance: Levenshtein Distance

Edit distance between 2 words:

word1	→	word2
horse		ros

OPERATIONS

1. REPLACE
2. INSERT
3. DELETE

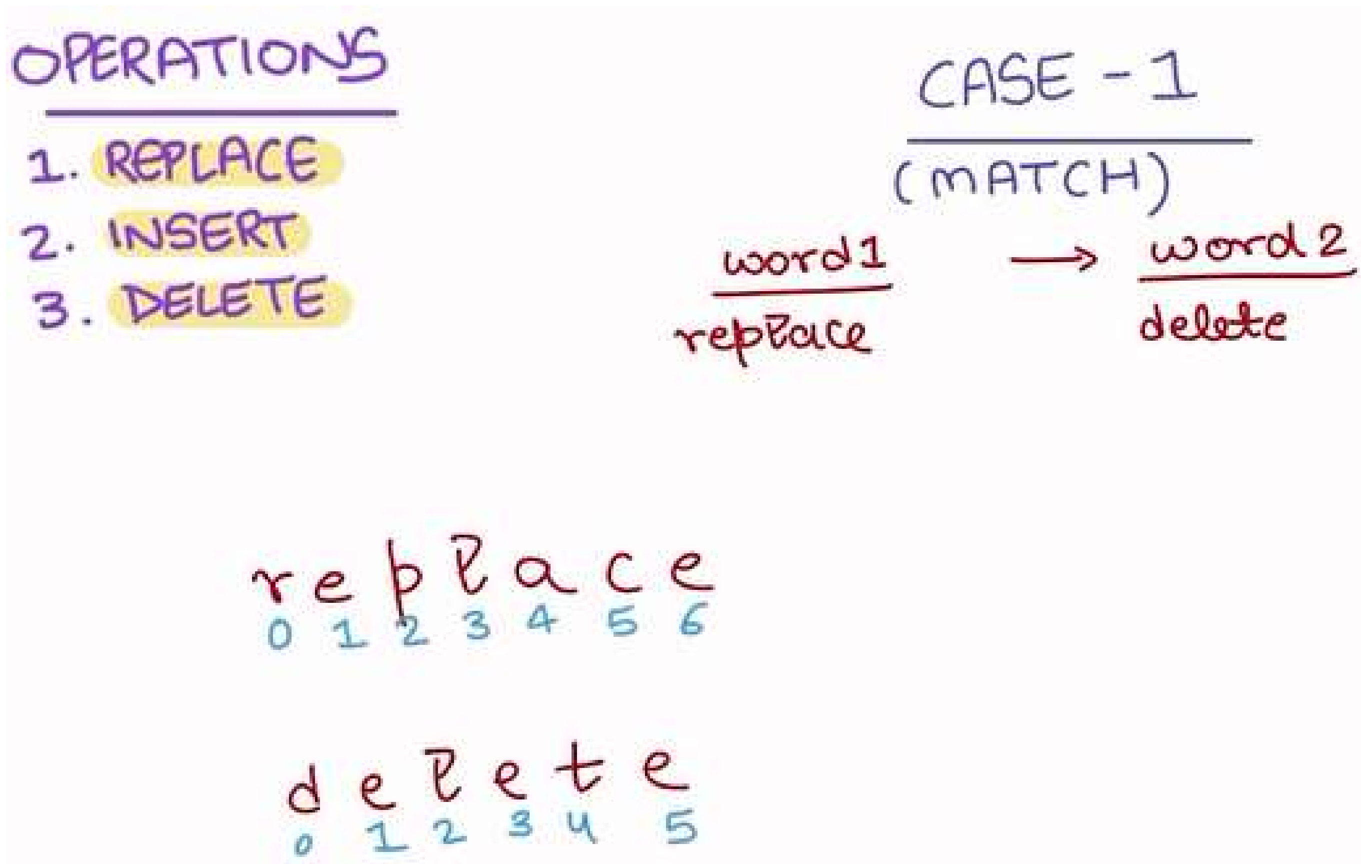
EDIT DISTANCE: 3

horse → rorse (replace 'h' with 'r')

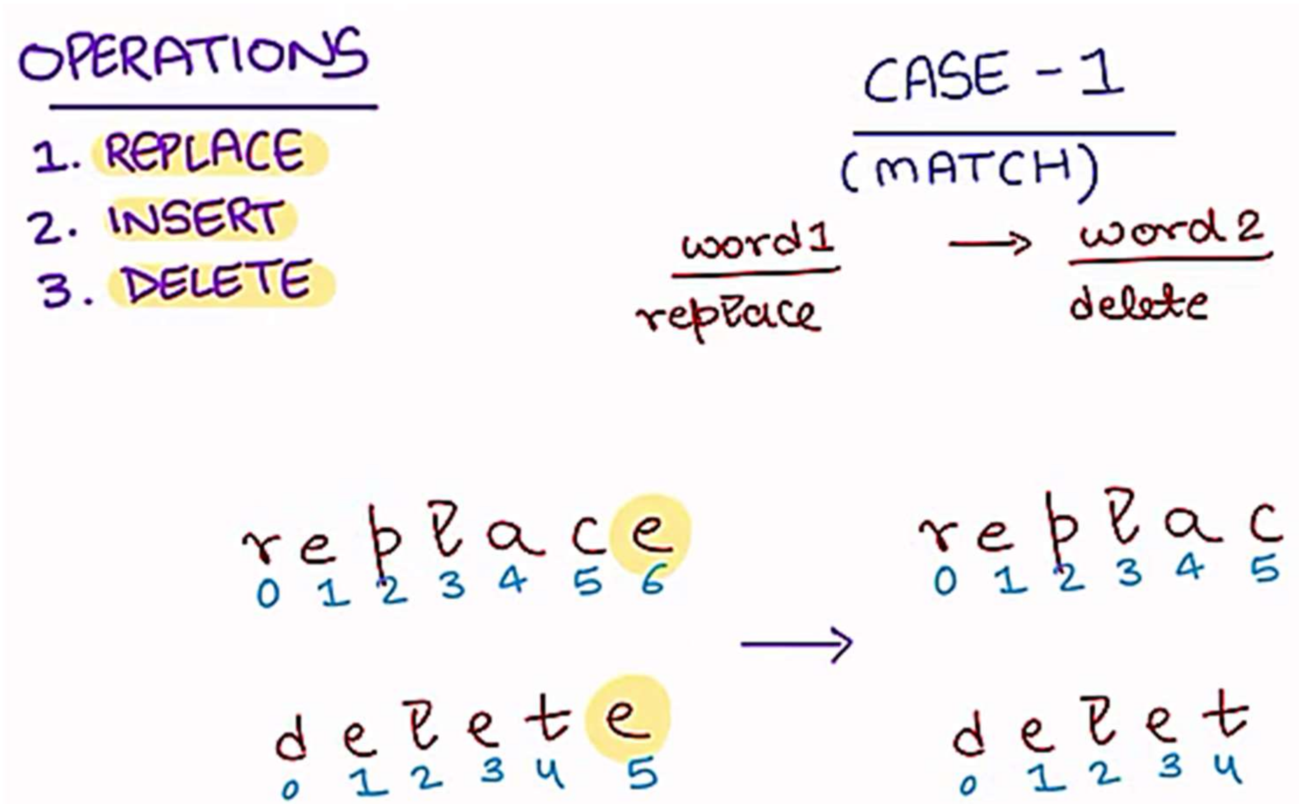
rorse → rose (remove 'r')

rose → ros (remove 'e')

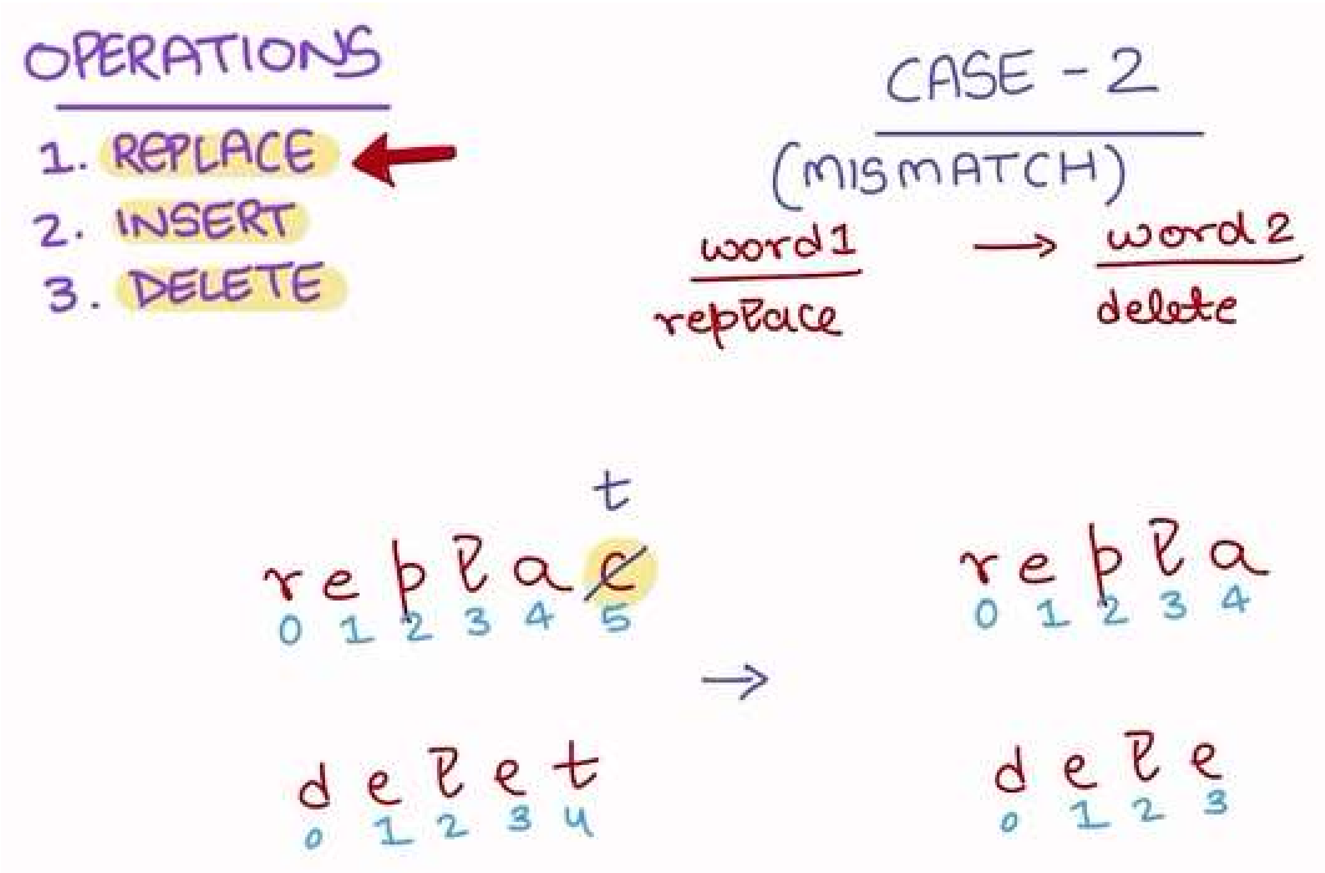
Edit Distance: Levenshtein Distance



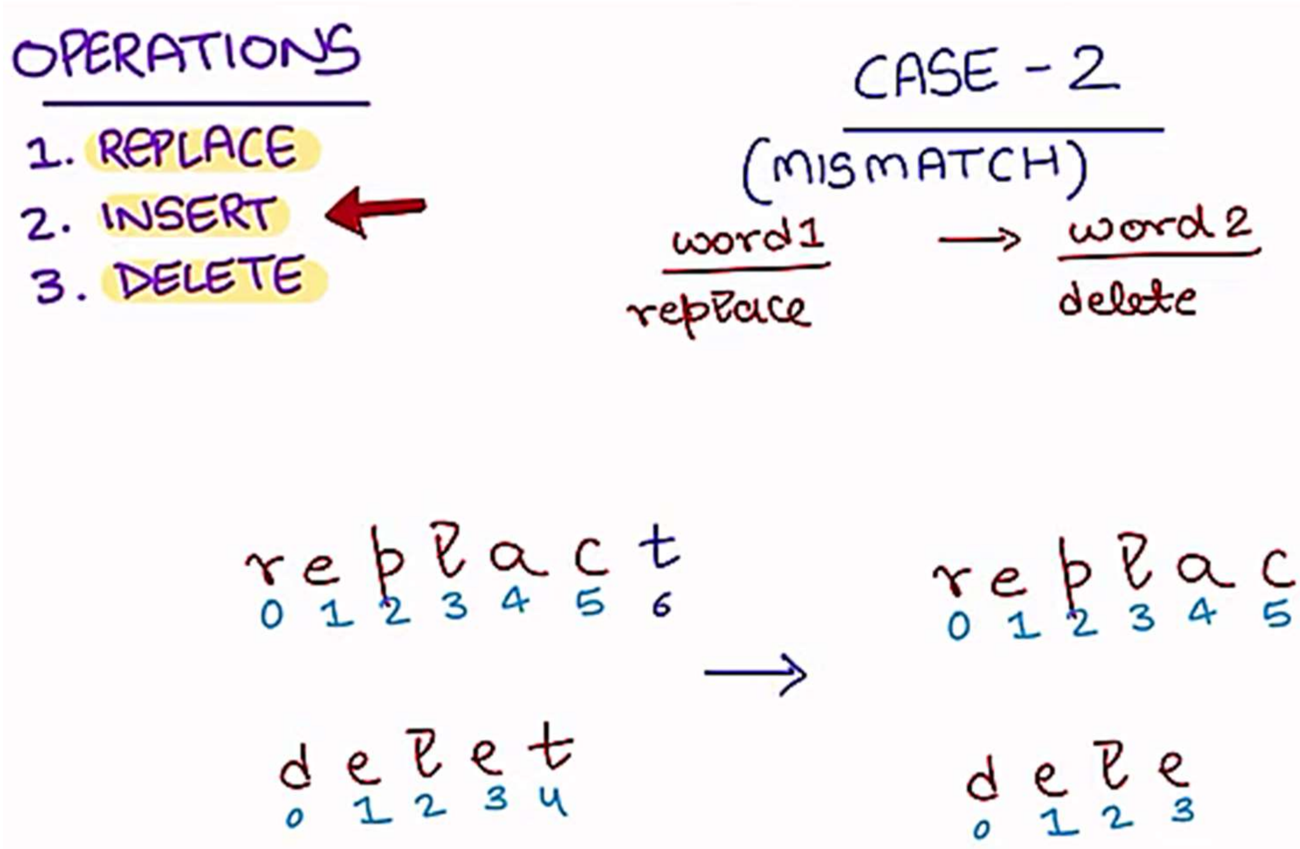
Edit Distance: Levenshtein Distance



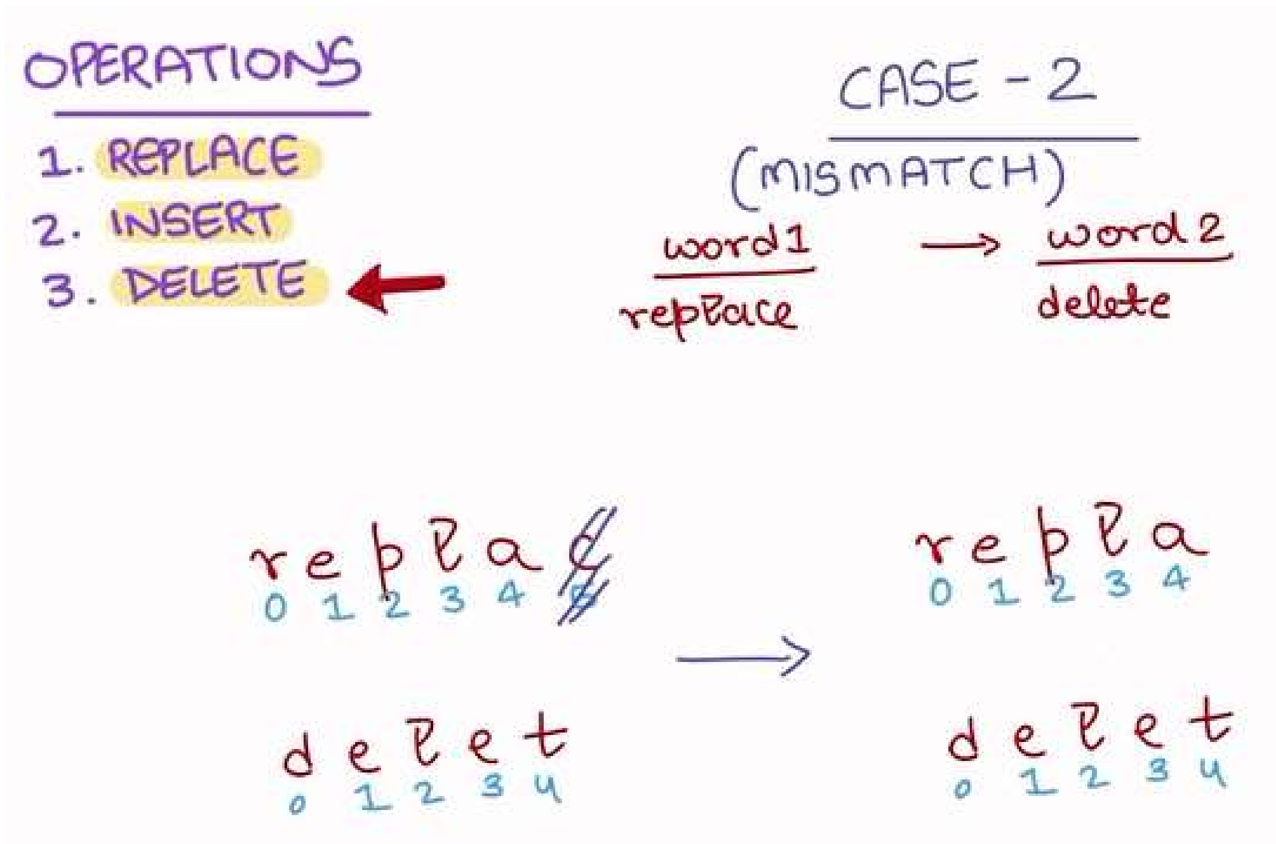
Edit Distance: Levenshtein Distance {REPLACE}



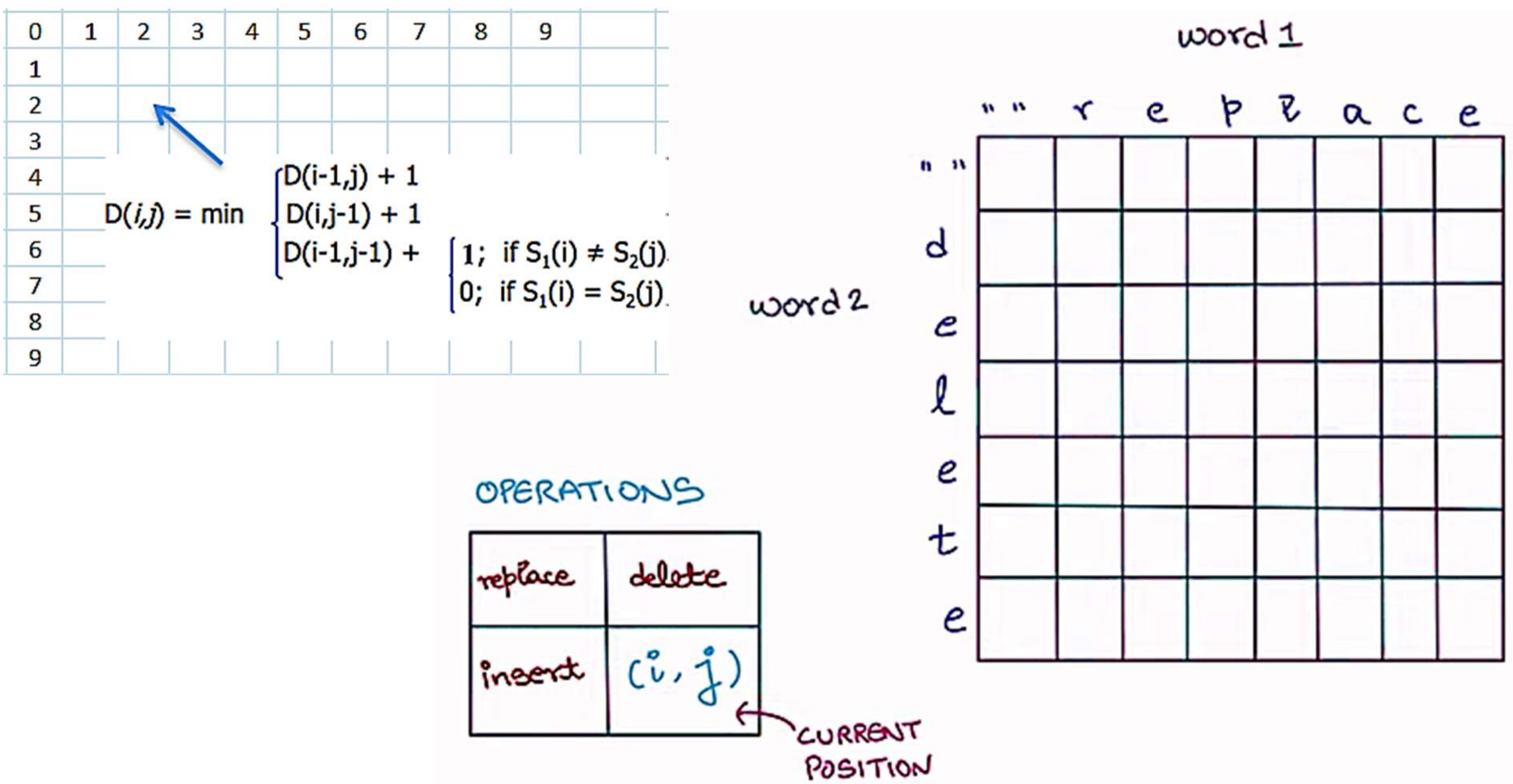
Edit Distance: Levenshtein Distance {INSERT}



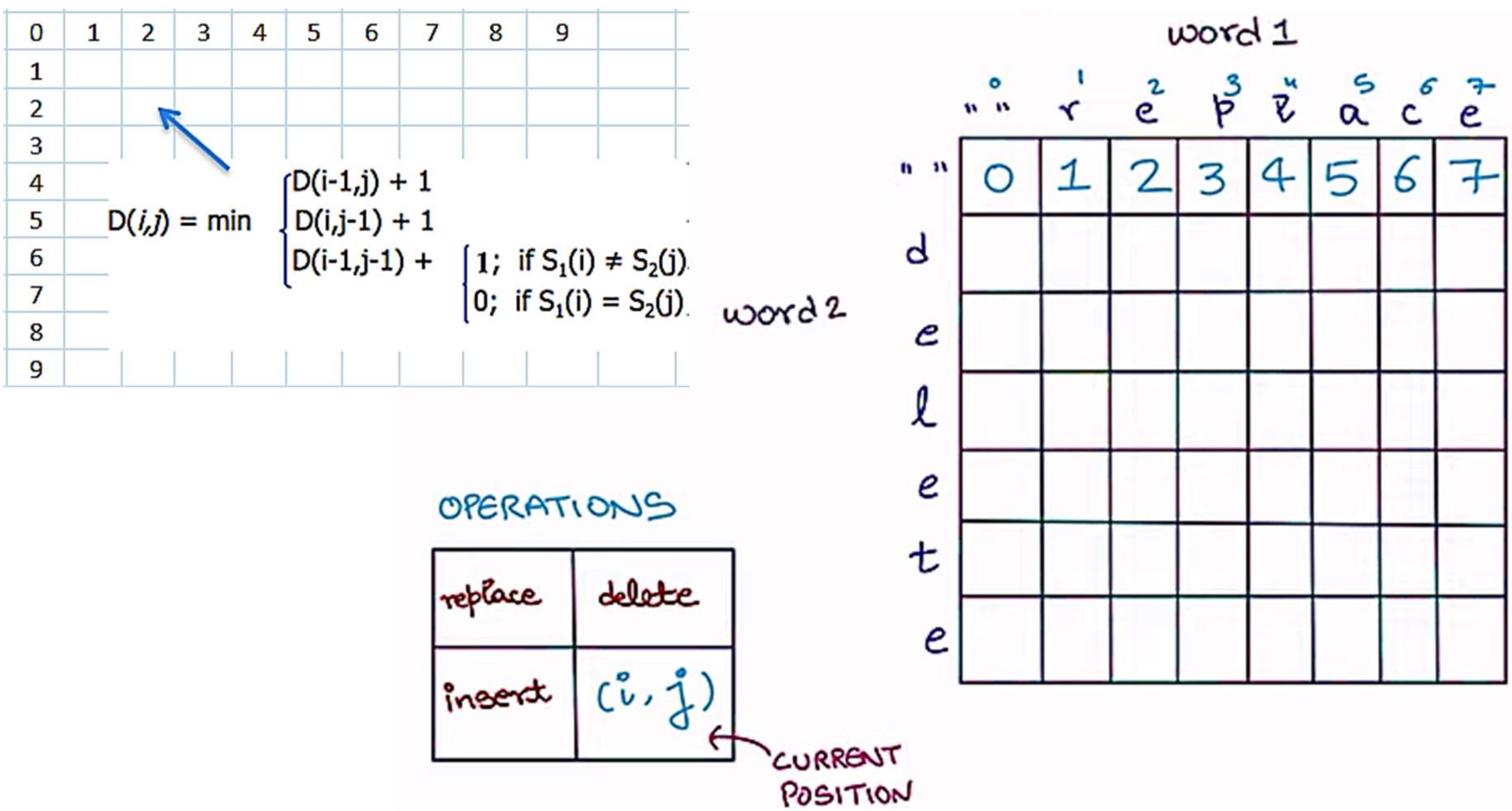
Edit Distance: Levenshtein Distance {DELETE}



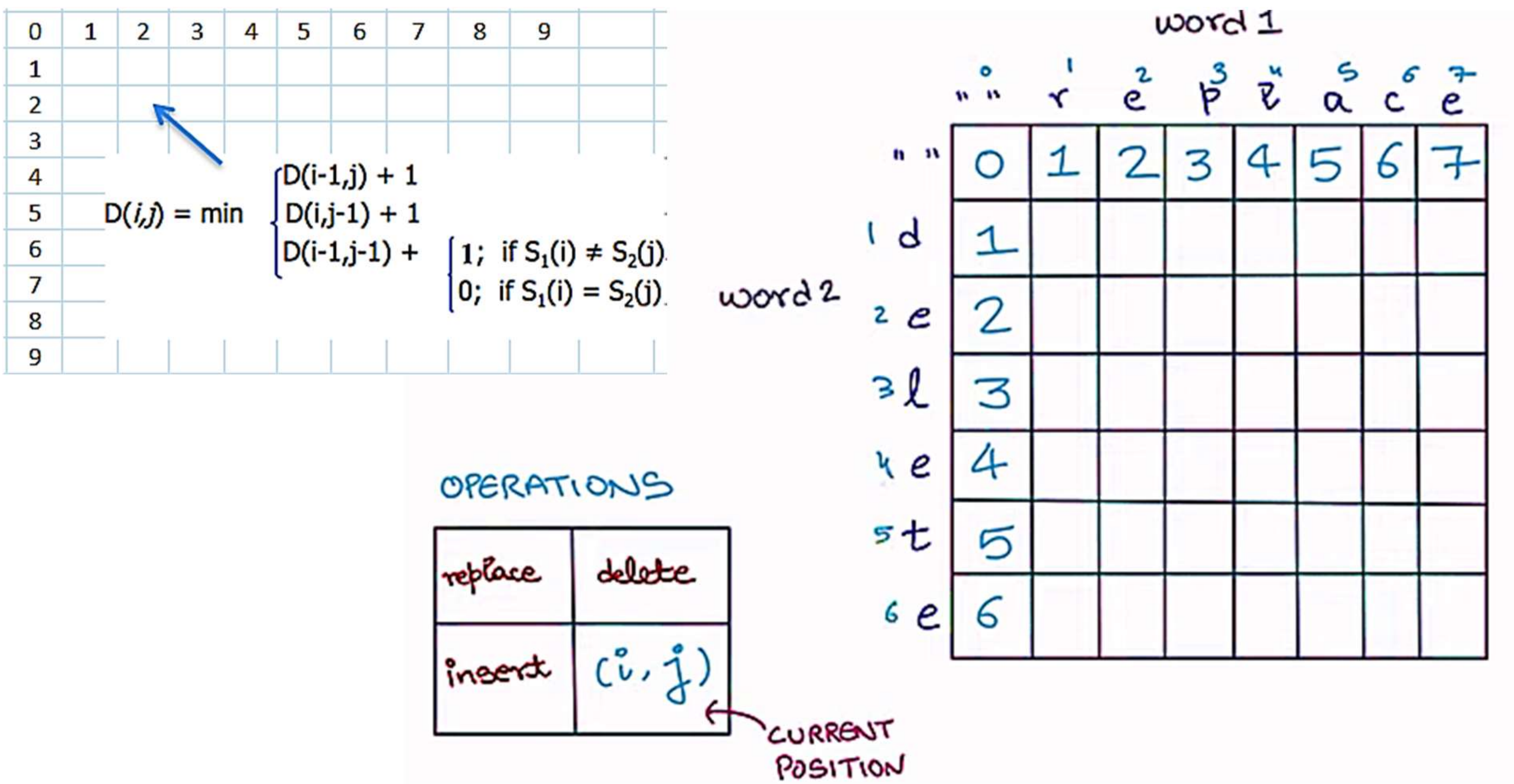
Edit Distance: Levenshtein Distance



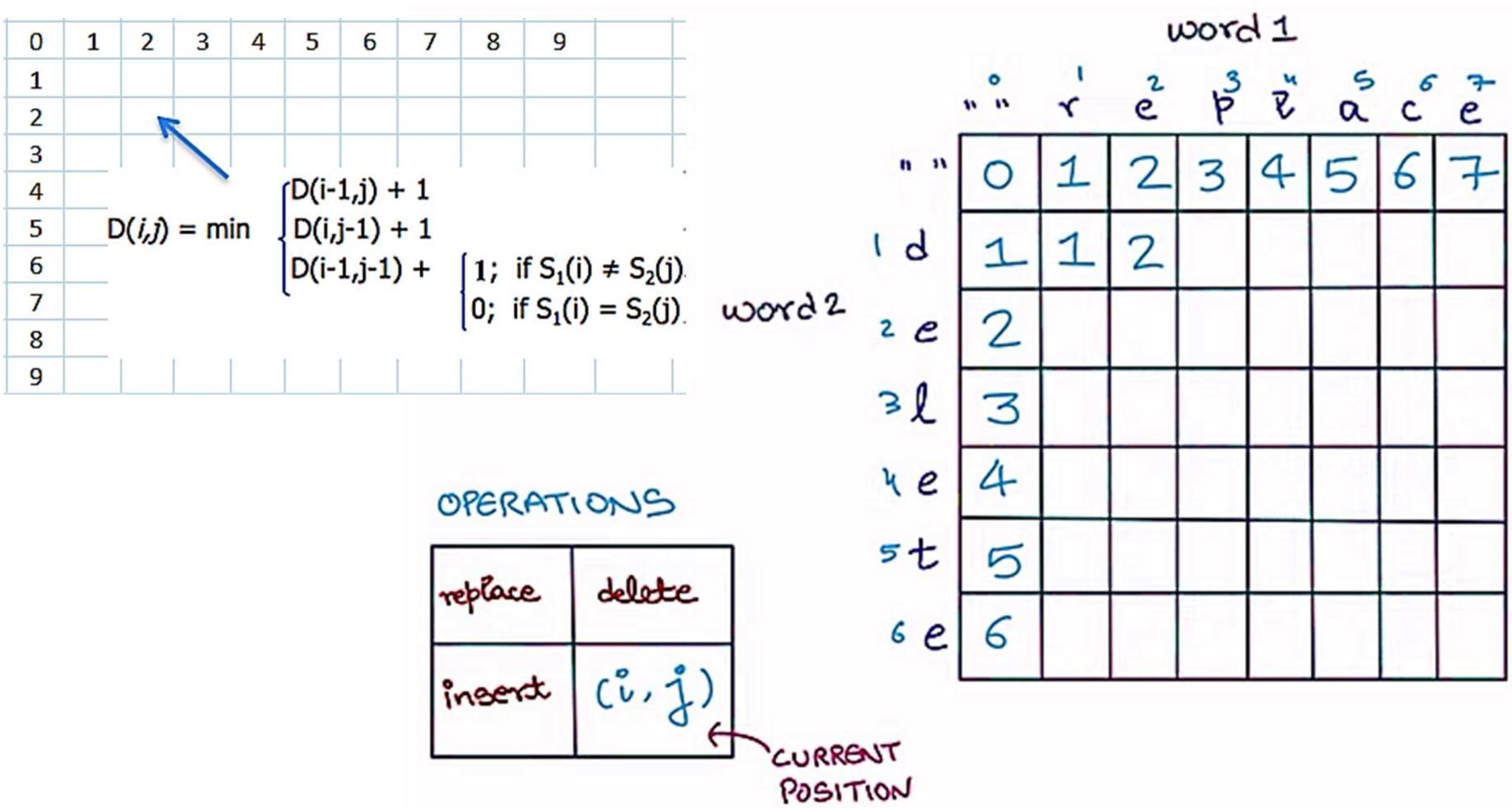
Edit Distance: Levenshtein Distance



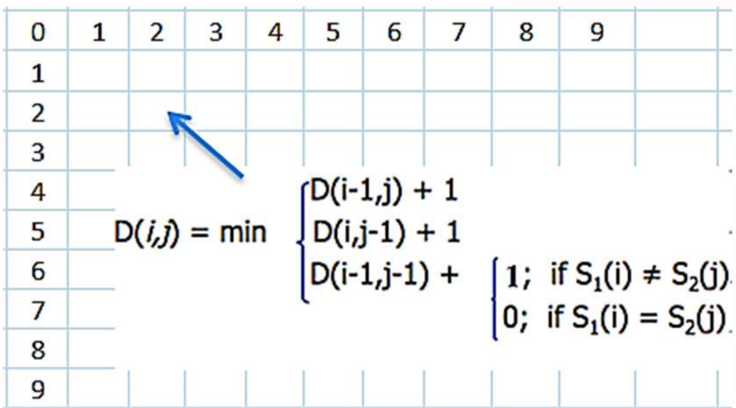
Edit Distance: Levenshtein Distance

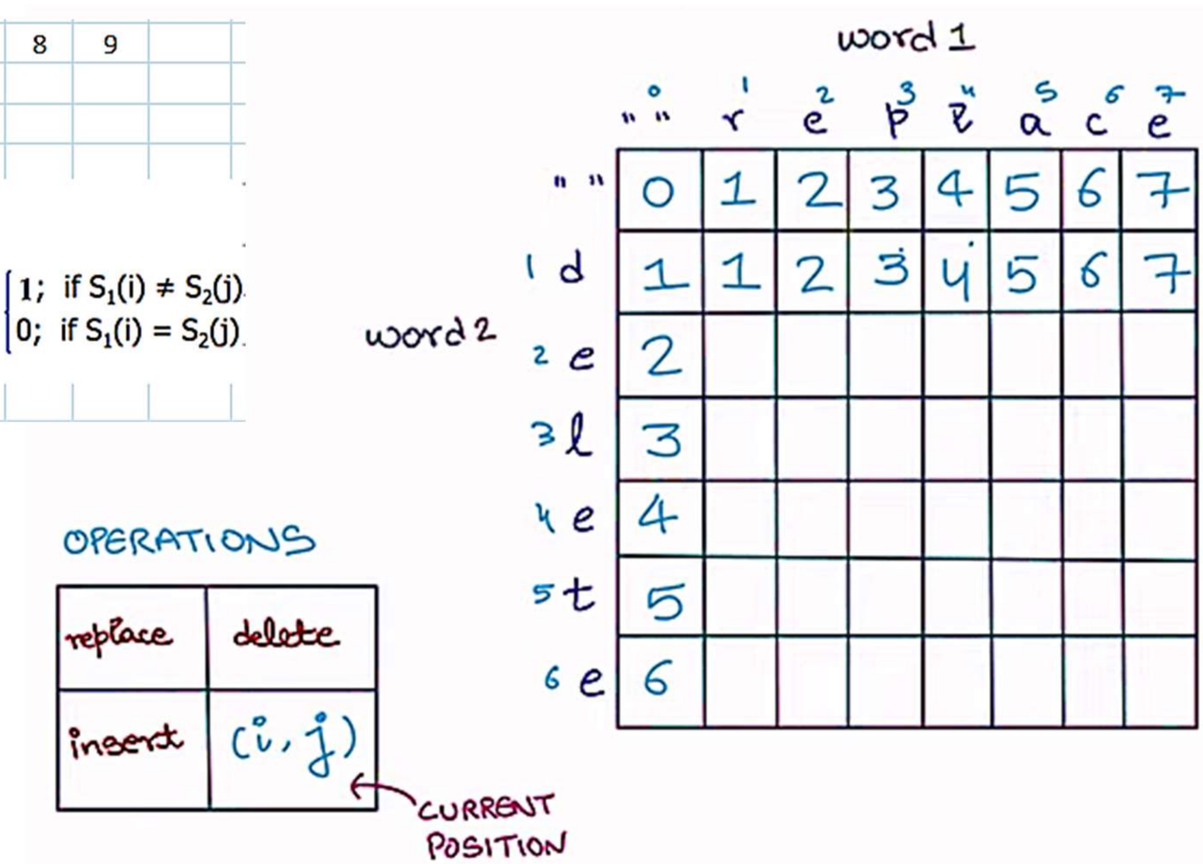


Edit Distance: Levenshtein Distance

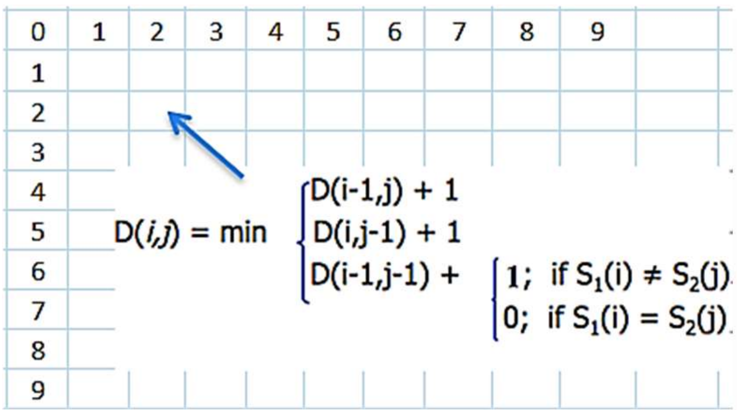


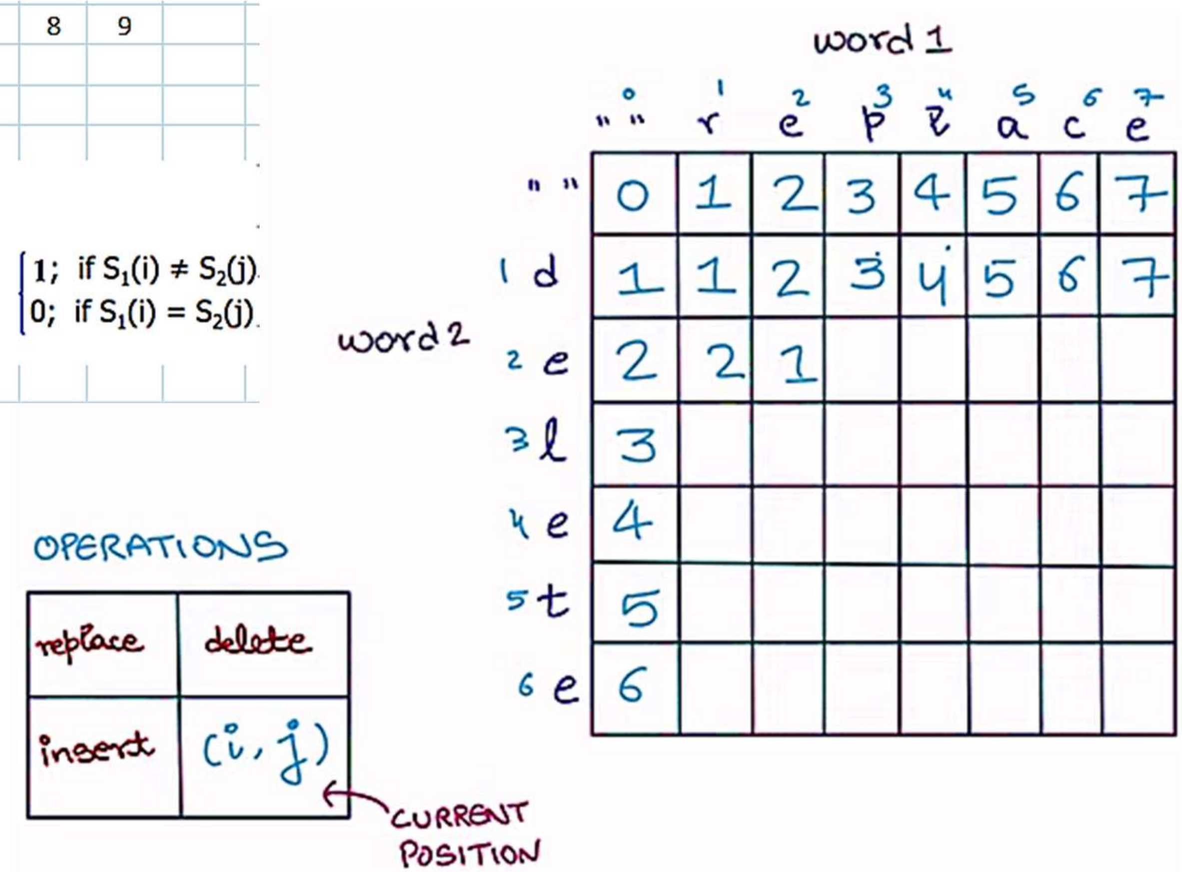
Edit Distance: Levenshtein Distance



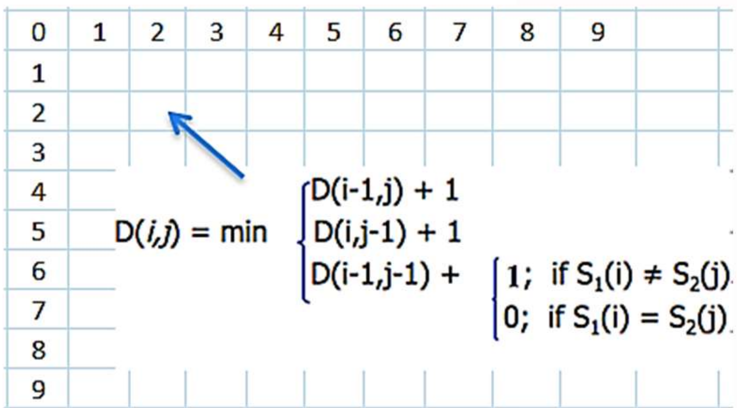


Edit Distance: Levenshtein Distance





Edit Distance: Levenshtein Distance



$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

word 1

word 2

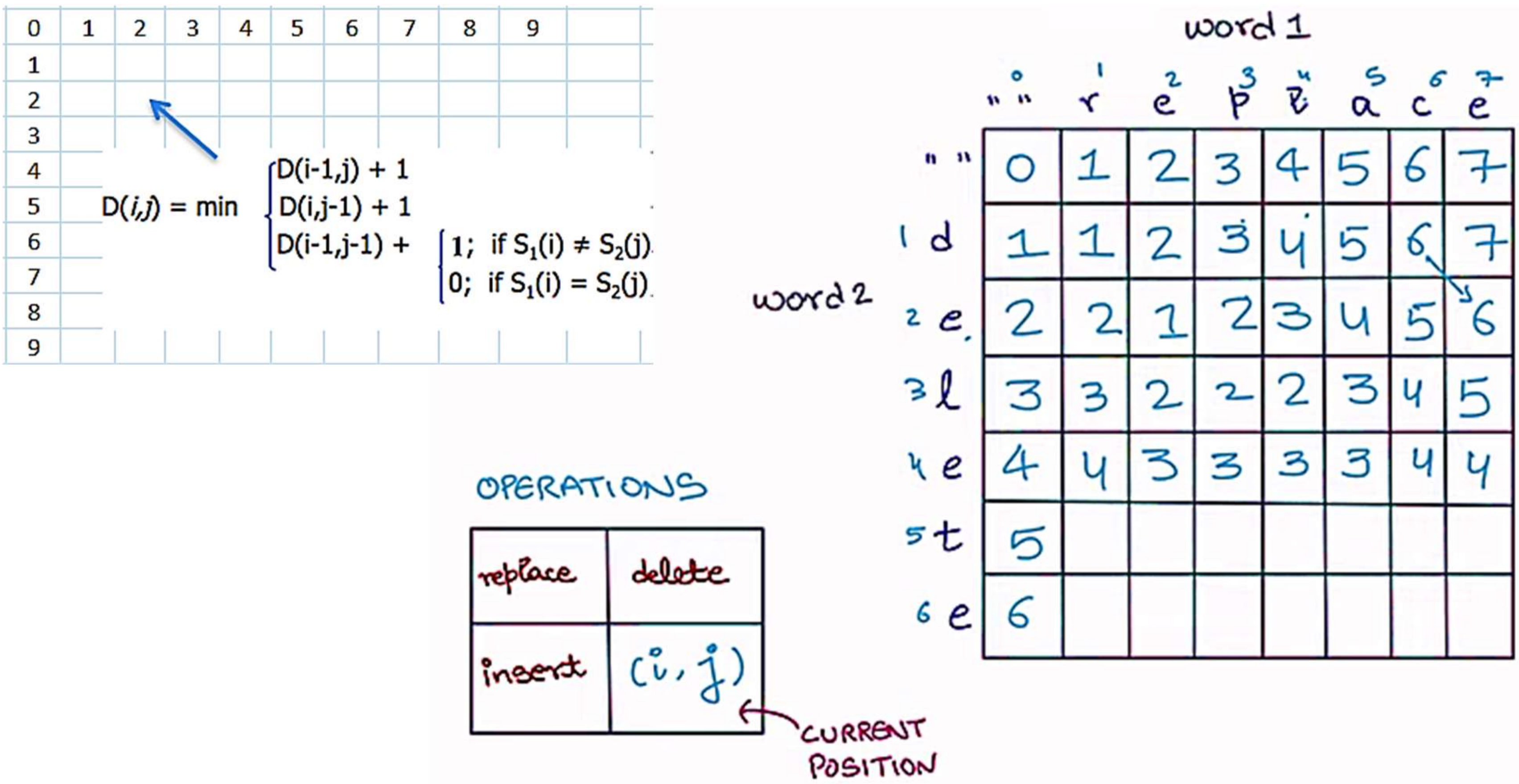
	0 ""	1 r	2 e	3 p	4 l	5 a	6 c	7 e
""	0	1	2	3	4	5	6	7
1 d	1	1	2	3	4	5	6	7
2 e	2	2	1	2	3	4	5	6
3 l	3							
4 e	4							
5 t	5							
6 e	6							

OPERATIONS

replace	delete
insert	(i, j)

CURRENT POSITION

Edit Distance: Levenshtein Distance



$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

	""	r	e	p	l	a	c	e
""	0	1	2	3	4	5	6	7
d	1	1	2	3	4	5	6	7
e	2	2	1	2	3	4	5	6
l	3	3	2	2	2	3	4	5
e	4	4	3	3	3	3	4	4
t	5							
e	6							

Edit Distance: Levenshtein Distance

0	1	2	3	4	5	6	7	8	9
1									
2									
3									
4									
5									
6									
7									
8									
9									

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 1; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$

$$b_{i,j} \leftarrow \begin{cases} \text{"}\uparrow\text{"} & \text{if } s_{i,j} = s_{i-1,j} \\ \text{"}\leftarrow\text{"} & \text{if } s_{i,j} = s_{i,j-1} \\ \text{"}\nwarrow\text{"} & \text{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$$

OPERATIONS

replace	delete
insert	(i, j)

CURRENT POSITION

word 1

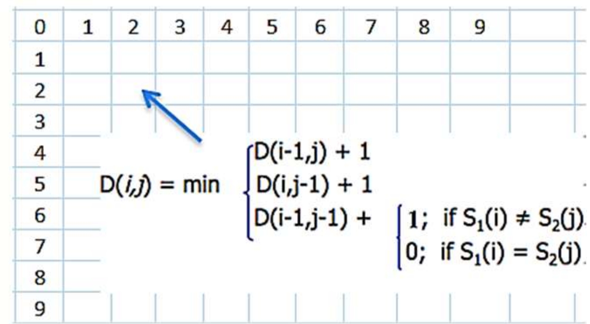
word 2

	0 ""	1 r	2 e	3 p	4 l	5 a	6 c	7 e
""	0	1	2	3	4	5	6	7
1 d	1	1	2	3	4	5	6	7
2 e	2	2	1	2	3	4	5	6
3 l	3	3	2	2	2	3	4	5
4 e	4	4	3	3	3	3	4	4
5 t	5	5	4	4	4	4	4	5
6 e	6	6	5	5	5	5	5	4

Edit Distance: Levenshtein Distance - Code

```
class Solution {
public:
    int minDistance(string word1, string word2) {
        vector<vector<int>> dp(word2.size() + 1, vector<int>(word1.size() + 1, 0));

        for(int i = 0; i < dp.size(); ++i) {
            dp[i][0] = i;
        }
        for(int i = 0; i < dp[0].size(); ++i) {
            dp[0][i] = i;
        }
        for(int row = 1; row < dp.size(); ++row) {
            for(int col = 1; col < dp[0].size(); ++ col) {
                if(word1[col - 1] == word2[row - 1]) {
                    dp[row][col] = dp[row - 1][col - 1];
                } else {
                    dp[row][col] = min({dp[row][col - 1], dp[row - 1][col], dp[row - 1][col - 1]}) + 1;
                }
            }
        }
        return dp[dp.size() - 1][dp[0].size() - 1];
    }
};
```



Order of the Algorithm

Time Complexity is O(mn)

Space Complexity is O(mn)

Assignment No. 04 CLO-4

(a) Compute Edit Distance matrix for the following strings by considering first string as rows and second string as column wise, and then to show alignment with operations.

A. X= S T A L L , Y=T A B L E

B. X=E X E C U T I O N Y= I N T E N T I O N

C. X= n u m p y Y= n u m e x p r

D. X=G C G T A T G C A C G C
Y=G C T A T G C C A C G C

Thank You!!!

Have a good day