

# Απόσταση Levenshtein

# Απόσταση Μεταξύ Λέξεων

Πώς ορίζεται;

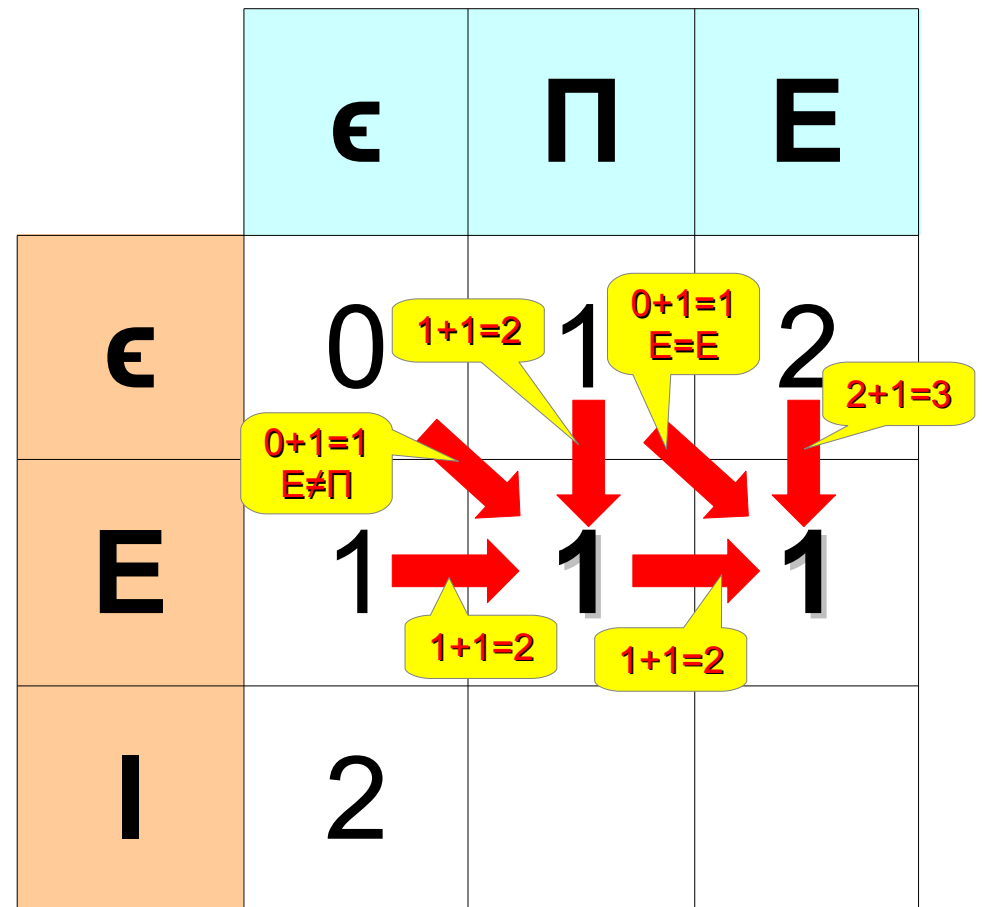
Από το πλήθος των διορθώσεων που χρειάζεται να γίνουν για να μετατρέψουμε τη μία λέξη στην άλλη.

(edit distances)

# Απόσταση Levenshtein

	€	Π	Ε	Δ	Ι	Ν	Η	Σ
€	0	1	2	3	4	5	6	7
Ε	1							
Ι	2							
Δ	3							
Η	4							
Σ	5							
Η	6							

	€	Π	Ε	Δ	Ι	Ν	Η	Σ
€	0	1	2	3	4	5	6	7
Ε	1	1	1	2	3	4	5	6
Ι	2	2	2	2	2	3	4	5
Δ	3	3	3	2	3	3	4	5
Η	4	4	4	3	3	4	3	4
Σ	5	5	5	4	4	4	4	3
Η	6	6	6	5	5	5	4	4



# Βέλτιστο Μονοπάτι

	€	Π	Ε	Δ	Ι	Ν	Η	Σ
€	0	1	2	3	4	5	6	7
Ε	1	1	1	2	3	4	5	6
Ι	2	2	2	2	2	3	4	5
Δ	3	3	3	2	3	3	4	5
Η	4	4	4	3	3	4	3	4
Σ	5	5	5	4	4	4	4	3
Η	6	6	6	5	5	5	4	4

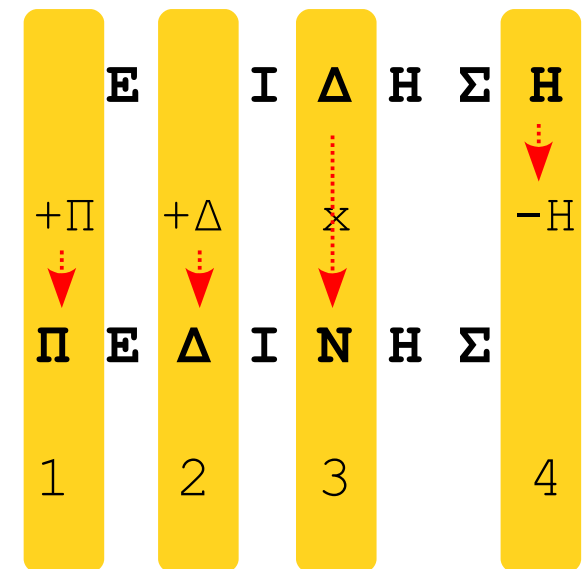
	€	Π	Ε	Δ	Ι	Ν	Η	Σ
€	0	1						
Ε			1	2				
Ι					2			
Δ						3		
Η							3	
Σ								3
Η								4

Ξεκινά πάντα από το πάνω αριστερό τετράγωνο δημιουργώντας μια αύξουσα ακολουθία τιμών μέχρι το κάτω δεξιό τετράγωνο.

Για κάθε οριζόντια ή κατακόρυφη μετακίνηση η αύξηση γίνεται κατά 1.  
Για κάθε διαγώνια μετακίνηση η αύξηση μπορεί να είναι 0 ή 1.

Κάθε φορά που αυξάνεται η τιμή στο μονοπάτι έχουμε και μια αλλαγή:

- > οριζόντια μετακίνηση = προσθήκη του χαρακτήρα της στήλης
- > κατακόρυφη μετακίνηση = αφαίρεση του χαρακτήρα γραμμής
- > διαγώνια μετακίνηση = αντικατάσταση του συμβόλου της γραμμής με το σύμβολο της στήλης



# Υλοποίηση

	€	Π	Ε
€	0	1	2
Ε	1	1	1
Ι	2		

Diagram illustrating the calculation of edit distance (Levenshtein distance) between two strings, S and T, using a dynamic programming table. The table shows the minimum number of operations (insertions, deletions, substitutions) required to transform S into T.

The table structure is as follows:

	€	Π	Ε
€	0	1	2
Ε	1	1	1
Ι	2		

Red arrows indicate the path from (0,0) to (2,2). Yellow callouts show the calculations for each step:

- From (0,0) to (0,1):  $0+1=1$  (E ≠ Π)
- From (0,1) to (1,1):  $1+1=2$
- From (0,1) to (1,2):  $0+1=1$  (E = Ε)
- From (1,1) to (2,2):  $2+1=3$
- From (1,1) to (2,1):  $1+1=2$
- From (1,2) to (2,2):  $1+1=2$

```
def Lev(S,T):
```

```
    if S=="": return len(T)
    if T=="": return len(S)
```

```
    p1=Lev(S,T[:-1])+1
```

```
    p2=Lev(S[:-1],T)+1
```

```
    p3=Lev(S[:-1],T[:-1])
```

```
    if S[-1]!=T[-1]: p3+=1
```

```
    return min(p1,p2,p3)
```