

3. Approximate String Matching

Often in applications we want to search a text for something that is [similar](#) to the pattern but not necessarily exactly the same.

To formalize this problem, we have to specify what does “similar” mean. This can be done by defining a [similarity](#) or a [distance measure](#).

A natural and popular distance measure for strings is the [edit distance](#), also known as the [Levenshtein distance](#).

Edit distance

The **edit distance** $ed(A, B)$ of two strings A and B is the minimum number of edit operations needed to change A into B . The allowed edit operations are:

- S **Substitution** of a single character with another character.
- I **Insertion** of a single character.
- D **Deletion** of a single character.

Example 3.1: Let $A = \text{Lewensteinn}$ and $B = \text{Levenshtein}$. Then $ed(A, B) = 3$.

The set of edit operations can be described

with an **edit sequence**: **NNSNNNINNNND**
or with an **alignment**: **Lewens-teinn**
Levenshtein-

In the edit sequence, N means No edit.

There are many variations and extension of the edit distance, for example:

- **Hamming distance** allows only the substitution operation.
- **Damerau–Levenshtein distance** adds an edit operation:
T **Transposition** swaps two adjacent characters.
- With **weighted edit distance**, each operation has a cost or weight, which can be other than one.
- Allow insertions and deletions (indels) of **factors** at a cost that is lower than the sum of character indels.

We will focus on the basic Levenshtein distance.

Levenshtein distance has the following two useful properties, which are not shared by all variations (exercise):

- Levenshtein distance is a **metric**.
- If $ed(A, B) = k$, there exists an edit sequence and an alignment with k edit operations, but no edit sequence or alignment with less than k edit operations. An edit sequence and an alignment with $ed(A, B)$ edit operations is called **optimal**.

Computing Edit Distance

Given two strings $A[1..m]$ and $B[1..n]$, define the values d_{ij} with the recurrence:

$$\begin{aligned}d_{00} &= 0, \\d_{i0} &= i, \quad 1 \leq i \leq m, \\d_{0j} &= j, \quad 1 \leq j \leq n, \text{ and} \\d_{ij} &= \min \begin{cases} d_{i-1,j-1} + \delta(A[i], B[j]) \\ d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \end{cases} \quad 1 \leq i \leq m, 1 \leq j \leq n,\end{aligned}$$

where

$$\delta(A[i], B[j]) = \begin{cases} 1 & \text{if } A[i] \neq B[j] \\ 0 & \text{if } A[i] = B[j] \end{cases}$$

Theorem 3.2: $d_{ij} = ed(A[1..i], B[1..j])$ for all $0 \leq i \leq m$, $0 \leq j \leq n$.
In particular, $d_{mn} = ed(A, B)$.

Example 3.3: $A = \text{ballad}, B = \text{handball}$

d		h	a	n	d	b	a	l	l
	0	1	2	3	4	5	6	7	8
b	1	1	2	3	4	4	5	6	7
a	2	2	1	2	3	4	4	5	6
l	3	3	2	2	3	4	5	4	5
l	4	4	3	3	3	4	5	5	4
a	5	5	4	4	4	4	4	5	5
d	6	6	5	5	4	5	5	5	6

$$ed(A, B) = d_{mn} = d_{6,8} = 6.$$

Proof of Theorem 3.2. We use induction with respect to $i + j$. For brevity, write $A_i = A[1..i]$ and $B_j = B[1..j]$.

Basis:

$$d_{00} = 0 = ed(\epsilon, \epsilon)$$

$$d_{i0} = i = ed(A_i, \epsilon) \quad (i \text{ deletions})$$

$$d_{0j} = j = ed(\epsilon, B_j) \quad (j \text{ insertions})$$

Induction step: We show that the claim holds for d_{ij} , $1 \leq i \leq m, 1 \leq j \leq n$. By induction assumption, $d_{pq} = ed(A_p, B_q)$ when $p + q < i + j$.

Let E_{ij} be an optimal edit sequence with the cost $ed(A_i, B_j)$. We have three cases depending on what the last operation symbol in E_{ij} is:

N or S: $E_{ij} = E_{i-1,j-1}N$ or $E_{ij} = E_{i-1,j-1}S$ and
 $ed(A_i, B_j) = ed(A_{i-1}, B_{j-1}) + \delta(A[i], B[j]) = d_{i-1,j-1} + \delta(A[i], B[j]).$

I: $E_{ij} = E_{i,j-1}I$ and $ed(A_i, B_j) = ed(A_i, B_{j-1}) + 1 = d_{i,j-1} + 1.$

D: $E_{ij} = E_{i-1,j}D$ and $ed(A_i, B_j) = ed(A_{i-1}, B_j) + 1 = d_{i-1,j} + 1.$

One of the cases above is always true, and since the edit sequence is optimal, it must be one with the minimum cost, which agrees with the definition of d_{ij} . □

The recurrence gives directly a **dynamic programming** algorithm for computing the edit distance.

Algorithm 3.4: Edit distance

Input: strings $A[1..m]$ and $B[1..n]$

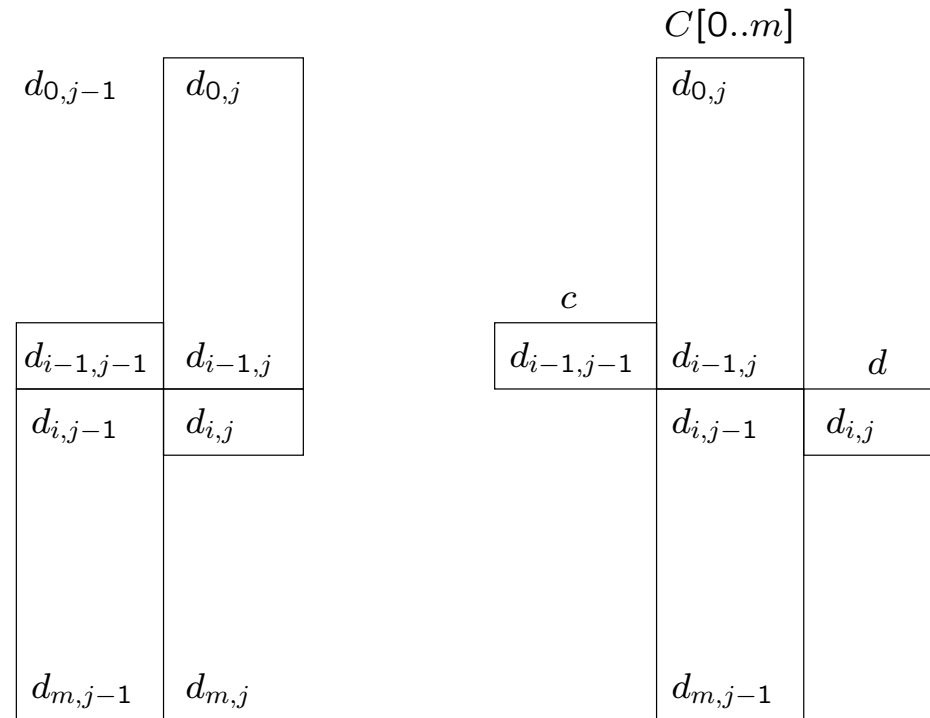
Output: $ed(A, B)$

- (1) for $i \leftarrow 0$ to m do $d_{i0} \leftarrow i$
- (2) for $j \leftarrow 1$ to n do $d_{0j} \leftarrow j$
- (3) for $j \leftarrow 1$ to n do
- (4) for $i \leftarrow 1$ to m do
- (5) $d_{ij} \leftarrow \min\{d_{i-1,j-1} + \delta(A[i], B[j]), d_{i-1,j} + 1, d_{i,j-1} + 1\}$
- (6) return d_{mn}

The time and space complexity is $\mathcal{O}(mn)$.

The space complexity can be reduced by noticing that each column of the matrix (d_{ij}) depends **only on the previous column**. We do not need to store older columns.

A more careful look reveals that, when computing d_{ij} , we only need to store the bottom part of column $j - 1$ and the already computed top part of column j . We store these in an array $C[0..m]$ and variables c and d as shown below:



Algorithm 3.5: Edit distance in $\mathcal{O}(m)$ space

Input: strings $A[1..m]$ and $B[1..n]$

Output: $ed(A, B)$

```
(1) for  $i \leftarrow 0$  to  $m$  do  $C[i] \leftarrow i$ 
(2) for  $j \leftarrow 1$  to  $n$  do
(3)    $c \leftarrow C[0]$ ;  $C[0] \leftarrow j$ 
(4)   for  $i \leftarrow 1$  to  $m$  do
(5)      $d \leftarrow \min\{c + \delta(A[i], B[j]), C[i - 1] + 1, C[i] + 1\}$ 
(6)      $c \leftarrow C[i]$ 
(7)      $C[i] \leftarrow d$ 
(8) return  $C[m]$ 
```

- Note that because $ed(A, B) = ed(B, A)$ (exercise), we can assume that $m \leq n$.

It is also possible to find optimal edit sequences and alignments from the matrix d_{ij} .

An edit graph is a directed graph, where the nodes are the cells of the edit distance matrix, and the edges are as follows:

- If $A[i] = B[j]$ and $d_{ij} = d_{i-1,j-1}$, there is an edge $(i-1, j-1) \rightarrow (i, j)$ labelled with N.
- If $A[i] \neq B[j]$ and $d_{ij} = d_{i-1,j-1} + 1$, there is an edge $(i-1, j-1) \rightarrow (i, j)$ labelled with S.
- If $d_{ij} = d_{i,j-1} + 1$, there is an edge $(i, j-1) \rightarrow (i, j)$ labelled with I.
- If $d_{ij} = d_{i-1,j} + 1$, there is an edge $(i-1, j) \rightarrow (i, j)$ labelled with D.

Any path from $(0, 0)$ to (m, n) is labelled with an optimal edit sequence.

Example 3.6: $A = \text{ballad}, B = \text{handball}$

d		h	a	n	d	b	a	l	l	
	0	⇒ 1	⇒ 2	⇒ 3	⇒ 4	→ 5	→ 6	→ 7	→ 8	
b	↓	↘	↘	↘	↘	↘				
	1		1 → 2	→ 3	→ 4		4 → 5	→ 6	→ 7	
a	↓	↘	↓	↘			↘			
	2		2	1 ⇒ 2	→ 3	→ 4		4 → 5	→ 6	
l	↓	↘	↓	↘	↘	↘	↘	↓	↘	
	3		3	2	2 ⇒ 3	→ 4	→ 5		4 → 5	
l	↓	↘	↓	↘	↘	↘	↘	↘	↘	
	4		4	3	3	3 ⇒ 4	→ 5		5	4
a	↓	↘	↓	↘	↓	↘	↘	↘		↘
	5		5	4	4	4	4	4 ⇒ 5		5
d	↓	↘	↓	↘	↓	↘	↘	↓	↘	↘
	6		6	5	5	4 → 5		5	5 ⇒ 6	

There are 7 paths from $(0,0)$ to $(6,8)$ corresponding to 7 different optimal edit sequences and alignments, including the following three:

IIIIINNND	SNISSNIS	SNSSINSI
----ballad	ba-lla-d	ball-ad-
handball--	handball	handball