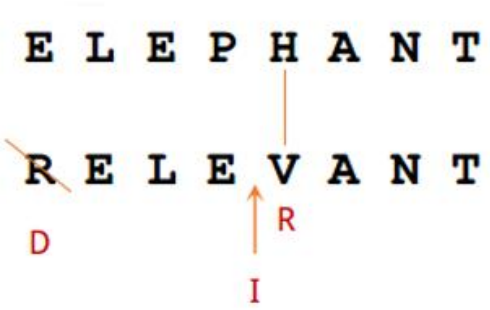
## What is Edit Distance?

The Edit Distance, most famously known as Levenshtein Distance, is a metric that quantifies the minimum number of single-character edits required to change one string (or sequence) into another.

| | | E | L | E | P | H | A | N | T |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| R | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| E | 2 | 1 | 2 | 2 | 3 | 4 | 5 | 6 | 7 |
| L | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| E | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| V | 5 | 4 | 3 | 2 | 2 | 3 | 4 | 5 | 6 |
| A | 6 | 5 | 4 | 3 | 3 | 3 | 3 | 4 | 5 |
| N | 7 | 6 | 5 | 4 | 4 | 4 | 4 | 3 | 4 |
| T | 8 | 7 | 6 | 5 | 5 | 5 | 5 | 4 | 3 |




Min Edit Distance: 3



Unlike the Hamming Distance, which only applies to strings of the same length and counts only substitutions, Edit Distance is much more flexible because it considers three fundamental types of edits:

- Insertion: Adding a character (e.g., "cat" to "cart" requires inserting 'r').
- Deletion: Removing a character (e.g., "cart" to "cat" requires deleting 'r').
- Substitution: Changing one character into another (e.g., "cat" to "bat" requires substituting 'c' with 'b').

Each of these operations is typically counted as one "edit cost." The goal is to find the cheapest way (the minimum number of operations) to transform one string into the other.

Example: Finding Similar Words in a Database (Spell Checking)

Imagine you have a list of correctly spelled product names in a database. A user types a search query, and you want to suggest the closest match even if there's a typo.

Let's say the correct product name is "chocolate". The user types: "choclate" (missing an 'o')

How many edits does it take to change "choclate" into "chocolate"?

Let's trace the transformation step-by-step:

- choclate
- chocolate (Insert 'o' at the 4th position)
- This transformation requires 1 edit (an insertion). So, the Edit Distance between "choclate" and "chocolate" is 1.

Another Example: "Saturday" vs. "Sunday"

Let's calculate the Edit Distance between "Saturday" and "Sunday".

S a t u r d a y

S u n d a y

Let's try to align them and see the changes:

- S A T U R D A Y
- S U N _ _ D A Y

We can transform "Saturday" into "Sunday" with the following minimum edits:

- S (match)
- A → U (Substitution: 'A' to 'U') - Cost: 1
- T → N (Substitution: 'T' to 'N') - Cost: 1
- U → D (No, U and D are not matched from here. We need to skip 'U' and 'R' in "Saturday" and match 'd'.)
- R (Deletion: delete 'R') - Cost: 1
- D (match)
- A (match)
- Y (match)

Let's re-evaluate the best path. It's often easier to think of it as aligning the strings and counting the non-matches.

S A T U R D A Y

S U N _ _ D A Y <-- One possible alignment for illustration

S A T U R D A Y

S U N D A Y

Let's trace again focusing on character-by-character transformation:

- S (match 'S')
- a → u (Substitution) - Cost: 1
- t → n (Substitution) - Cost: 1
- u (Delete 'u' from "Saturday") - Cost: 1
- r (Delete 'r' from "Saturday") - Cost: 1
- d (match 'd')
- a (match 'a')
- y (match 'y')

This path results in 4 edits. However, the optimal path is usually less:

Optimal Path (Edit Distance = 3):

- S (match)
- a → u (Substitution: 'a' to 'u') - Cost: 1
- t → n (Substitution: 't' to 'n') - Cost: 1

(Insert _ (placeholder for 'r') from "Saturday" to match 'r' in "Saturday") No, actually it's:

- S (match)
- a → u (Substitution) - Cost: 1
- t → n (Substitution) - Cost: 1
- (Delete 'r' from "Saturday") - Cost: 1
- d (match)
- a (match)
- y (match)

This is also 3 edits (2 substitutions, 1 deletion).

Let's try one more common interpretation:

- S (match)
- a → u (Substitution)
- t → n (Substitution)
- u (from saturday) -> _ (deletion)
- r (from saturday) -> _ (deletion)
- d (match)
- a (match)
- y (match) This is 4.

The standard way to get 3 is:

- s to s (match)
- a to u (substitution)
- t to n (substitution)
- delete r from saturday (or insert _ into sunday)
- d to d (match)
- a to a (match)
- y to y (match)

So, Edit Distance is 3.

Why it's useful:

Edit distance is a powerful tool because it tolerates small variations, making it ideal for:

- Spell Checkers: Suggesting "kitten" for "kitton" (1 substitution).
- Typo Correction: Correcting search queries.
- Bioinformatics: Measuring the evolutionary distance between two DNA sequences (where insertions, deletions, and point mutations are common).

- Plagiarism Detection: Identifying near-duplicate text.
- Fuzzy String Matching: Finding records that are "almost" the same in databases.