Edit Distance

Idea:

Use DP where dp[i][j] = minimum edits to convert the first i chars of word1 into the first j chars of word2.

Base cases:

- Converting empty → j chars: insert j times → dp[0][j] = j.
- Converting i chars → empty: delete i times → dp[i][0] = i.

Transition (build up by last operation):

For $i \geq 1, j \geq 1$, compare word1[i-1] vs word2[j-1].

- If they match: no new edit needed dp[i][j] = dp[i-1][j-1].
- If they don't match: take the best of three actions and add 1:
  - Insert the word2[j-1] character (aim to match more of word2):
    cost from dp[i][j-1] + 1
  - Delete word1[i-1] (drop a char from word1):
    cost from dp[i-1][j] + 1
  - Replace word1[i-1] with word2[j-1]:
    cost from dp[i-1][j-1] + 1

So: dp[i][j] = min(dp[i][j-1] + 1, dp[i-1][j] + 1, dp[i-1][j-1] + cost)
where cost = 0 if characters equal, else 1.

Answer:

dp[m][n] where m = len(word1), n = len(word2).

Complexity:

- Time: $O(m \cdot n)$
- Space: $O(m \cdot n)$ (or $O(n)$ with a rolling 1D array if you like optimizing).

1D space trick (optional):

Keep a single row dp[j] for the current i, plus a variable to remember the diagonal (dp[i-1][j-1]). Update left-to-right, carefully preserving values:

· prevDiag holds the old dp[j-1] from the previous row (i.e, the dp[i-1][j-1] term).

· For each j, compute the new cur = min(insert, delete, replace Or Match)

using: · insert: dp[j-1]+1 (already updated this row).

· delete: dp[j]+1 (value from previous row before overwrite)

· replace/match: prevDiag + (word1[i-1] != word2[j-1])

· Then set prevDiag = old dp[j] before overwrite.