

# Edit Distance

Levenshtein distance — 2D DP king ■

● HARD

## ■ Hook

Minimum operations (insert, delete, replace) to convert string word1 into word2. Used in spell checkers, DNA sequence alignment, autocorrect. One of the most important DP problems ever.

## ■ Core Intuition — Three Operations

```
dp[i][j] = min ops to convert word1[0..i-1] to word2[0..j-1]
If word1[i-1] == word2[j-1]: dp[i][j] = dp[i-1][j-1] (no op needed)
Else take minimum of three operations:
    INSERT into word1: dp[i][j] = dp[i][j-1] + 1
    DELETE from word1: dp[i][j] = dp[i-1][j] + 1
    REPLACE in word1: dp[i][j] = dp[i-1][j-1] + 1
Base cases:
dp[i][0] = i (delete all i chars of word1)
dp[0][j] = j (insert all j chars of word2)
```

## ■ Grid — word1='horse', word2='ros'

```

      ''  r  o  s
''  [0]  1  2  3
h   [1]  1  2  3
o   [2]  2  1  2
r   [3]  2  2  2
s   [4]  3  3  2
e   [5]  4  4  3] ← answer=3
```

## ■ Code

```

public int minDistance(String w1, String w2) {
    int m=w1.length(), n=w2.length();
    int[][] dp = new int[m+1][n+1];
    for (int i=0; i<=m; i++) dp[i][0]=i;
    for (int j=0; j<=n; j++) dp[0][j]=j;
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++) {
            if (w1.charAt(i-1)==w2.charAt(j-1))
                dp[i][j] = dp[i-1][j-1];
            else
                dp[i][j] = 1 + Math.min(dp[i-1][j-1],
                                         Math.min(dp[i-1][j], dp[i][j-1]));
        }
    }
    return dp[m][n];
}

```

■ TIME

**$O(m \times n)$**

■ SPACE

**$O(m \times n) \rightarrow O(n)$**

## ■ Practice

| # | Problem                                   | Diff   | Pattern     |
|---|-------------------------------------------|--------|-------------|
| 1 | Edit Distance (LC 72)                     | Hard   | 2D DP       |
| 2 | One Edit Distance (LC 161)                | Medium | Two Pointer |
| 3 | Delete Operation for Two Strings (LC 583) | Medium | LCS/Edit    |
| 4 | Longest Common Subsequence (LC 1143)      | Medium | 2D DP       |
| 5 | Minimum ASCII Delete Sum (LC 712)         | Medium | 2D DP       |

- ✓  $dp[i][j]$  = min ops to convert  $w1[0..i-1]$  to  $w2[0..j-1]$
- ✓ Match:  $dp[i][j] = dp[i-1][j-1]$  (free!)
- ✓ No match:  $1 + \min(\text{replace}=dp[i-1][j-1], \text{delete}=dp[i-1][j], \text{insert}=dp[i][j-1])$
- ✓ Base:  $dp[i][0]=i, dp[0][j]=j$
- ✓ Time:  $O(mn)$  | Space:  $O(mn)$ , optimizable to  $O(n)$

### ■ MENTAL MODEL — Remember This Forever

The DP grid answers: 'how different are the first  $i$  chars of word1 from the first  $j$  chars of word2?' Matching chars are free. For mismatches, pick the cheapest of: fix this position (replace), skip in word1 (delete), skip in word2 (insert). Each of these is literally one edit operation.