

Longest Common Subsequence

2D DP grid — the classic string DP ■

● MEDIUM

■ Hook

Given two strings, find the longest subsequence common to both. A subsequence maintains order but doesn't need to be contiguous. Used in diff tools (git diff), DNA analysis, spell checkers. This is the gateway to all 2D string DP problems.

■ Core Intuition

Build a 2D table $dp[i][j]$ = LCS length of $text1[0..i-1]$ and $text2[0..j-1]$. If characters match: $dp[i][j] = dp[i-1][j-1] + 1$. If not: $dp[i][j] = \max(dp[i-1][j], dp[i][j-1])$.

■ DP Grid — $text1='abcde', text2='ace'$

```

      ' ' a c e
' ' [ 0 0 0 0 ]
a  [ 0 1 1 1 ] ← a==a: dp[1][1]=dp[0][0]+1=1
b  [ 0 1 1 1 ] ← b!=a,c,e: take max from up/left
c  [ 0 1 2 2 ] ← c==c: dp[3][2]=dp[2][1]+1=2
d  [ 0 1 2 2 ]
e  [ 0 1 2 3 ] ← e==e: dp[5][3]=dp[4][2]+1=3
★ ANSWER = dp[5][3] = 3 (LCS = 'ace')
```

■ Code

```

public int longestCommonSubsequence(String text1, String text2) {
    int m=text1.length(), n=text2.length();
    int[][] dp = new int[m+1][n+1];
    for (int i=1; i<=m; i++) {
        for (int j=1; j<=n; j++) {
            if (text1.charAt(i-1) == text2.charAt(j-1))
                dp[i][j] = dp[i-1][j-1] + 1; // chars match
            else
                dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]); // skip one
        }
    }
    return dp[m][n];
}
```

■ TIME

$O(m \times n)$

■ SPACE

$O(m \times n)$

Common Mistakes

Subsequence vs Substring

Subsequence: characters don't need to be contiguous. Substring: they do. For LCS (subsequence), characters can be far apart — just maintain order.

Practice

#	Problem	Diff	Pattern
1	Longest Common Subsequence (LC 1143)	Medium	2D DP
2	Longest Common Substring	Medium	2D DP (contiguous)
3	Edit Distance (LC 72)	Hard	2D DP
4	Shortest Common Supersequence (LC 1092)	Hard	LCS + construction
5	Delete Operation for Two Strings (LC 583)	Medium	LCS variant

- ✓ $dp[i][j]$ = LCS of $text1[0..i-1]$ and $text2[0..j-1]$
- ✓ Match: $dp[i][j] = dp[i-1][j-1] + 1$
- ✓ No match: $dp[i][j] = \max(dp[i-1][j], dp[i][j-1])$
- ✓ Grid size: $(m+1) \times (n+1)$ with row/col 0 as empty string
- ✓ Time: $O(mn)$ | Space: $O(mn)$, reducible to $O(n)$

MENTAL MODEL — Remember This Forever

Think of the 2D grid as 'matching progress'. $dp[i][j]$ = 'how long is the best story I can tell using the first i letters of $text1$ and first j letters of $text2$?' When characters match, extend the story. When they don't, inherit the best story so far.