Longest Common Substring/Subsequence Pattern

This pattern helps solve problems involving **finding the optimal (longest, most similar) part of two strings**—either a substring (must be continuous) or a subsequence (order matters, but characters can be skipped).

- **Longest Common Substring:**
  The longest sequence of characters **appearing in both strings** as a contiguous block.
- **Longest Common Subsequence (LCS):**
  The longest sequence that **appears in both strings in order**, but not necessarily continuously.

Common Use Cases:

- Spell checking, diff tools
- DNA/protein similarity analysis
- "Edit distance" (how many edits to convert one string to another)

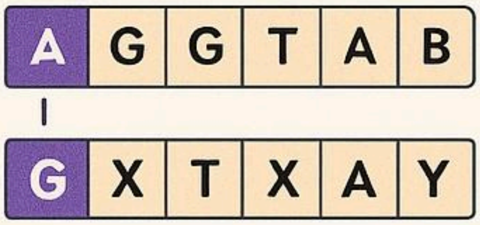
LONGEST COMMON SUBSTRING/SUBSEQUENCE PATTRN

LCS EXPLAINED

INPUT STRINGS: AGGTAB and GXTXAYB

Task: Find the length of their longest common subsequence (LCS).

1 **Goal:** Find the longest sequence present in both strings (not necessarily continuous)



2 Fill in the DP table: row by row, cell by c ll

1 + diagonal

	A	G	G	T	A	B
G	1	1	1	1	1	1
X	1	1	1	1	1	1
A	1	1	2	1	1	1
B	1	1	1	1	1	1
C	1	1	1	1	2	1

max of left

3 Build up the answer from the smallest subproblems

	A	G	G	T	A	B	
G	0	1	1	1	1	1	0
X	1	1	1	1	1	1	1
A	1	1	1	1	1	1	1
B	1	1	1	1	1	1	1
C	1	1	1	1	2	1	1
0	0	0	0	0	0	4	4

4 LCS Length = 4

G T A B

The LCS pattern is the foundation for diff tools. DNA sequen-ce alignment, and edit distance!

Longest common subsequence and substring problems use DP tables to reuse solutions for overlapping subproblems

Classic Problem: Longest Common Subsequence

Problem Statement:
Given two strings, find the length of their **Longest Common Subsequence** (LCS).

Example:
str1 = `"AGGTAB"`
str2 = `"GXTXAYB"`
Output: `4` (LCS is `"GTAB"`)

Explanation

1. **Log Session a DP table** with dimensions `[len1+1][len2+1]`.
2. **Fill table:**
   - If characters match, LCS length = 1 + LCS of previous indices.
   - Else, LCS length = max(LCS if skipping one from either string).
3. The answer is at `dp[len1][len2]`.

C Code Example: Longest Common Subsequence

```
#include <stdio.h>
#include <string.h>

// Returns length of LCS for str1 and str2
int lcs(char *str1, char *str2) {
    int m = strlen(str1);
    int n = strlen(str2);
    int dp[m+1][n+1];

    // Initialize first row and first column to 0
    for (int i = 0; i <= m; i++)
        dp[i][0] = 0;
    for (int j = 0; j <= n; j++)
        dp[0][j] = 0;

    // Build dp table
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (str1[i-1] == str2[j-1])
                dp[i][j] = 1 + dp[i-1][j-1];
            else
                dp[i][j] = (dp[i-1][j] > dp[i][j-1]) ? dp[i-1][j] : dp[i][j-1];
        }
    }
    return dp[m][n];
}

int main() {
    char str1[] = "AGGTAB";
    char str2[] = "GXTXAYB";
    printf("Length of LCS: %d\n", lcs(str1, str2));
    return 0;
}
```

How does this work?

- For each pair of indices, build the LCS length so far.
- If characters match, add 1 and move diagonally.
- If not, take the best length from skipping either character.

What about Longest Common Substring?

- Similar idea, but when characters match, count consecutive matches.
- Reset to 0 if they don’t match.
- (Ask if you want C code for substring as well!)

Where is this pattern useful?

- **File/sequence comparison tools** (like Git diff)
- **Bioinformatics** (DNA/RNA/protein similarity)
- **Spell checkers**
- **Edit distance problems** (minimum changes to turn one string into another)

Practice Challenge

- Try to print the actual LCS, not just its length.
- Try to write code for **Longest Common Substring**.