LCS Explanation and Code

Longest Common Subsequence (LCS) explain every thing and code in c++

Longest Common Subsequence (LCS) Explanation

The **Longest Common Subsequence (LCS)** is a classic dynamic programming problem. The goal is to find the longest sequence that appears in both given sequences (not necessarily contiguous).

Example

Given two sequences:

- Sequence 1: `ABCBDAB`
- Sequence 2: `BDCAB`

The **LCS** is `BCAB`, which has a length of 4.

Properties

1. **Order matters**: The characters must appear in the same relative order as in the original sequences.
2. **Not necessarily contiguous**: Unlike substrings, characters in the subsequence may not be adjacent.

Dynamic Programming Approach

We use a 2D table `dp` to solve the problem, where:

- `dp[i][j]` represents the length of the LCS of the first `i` characters of `sequence1` and the first `j` characters of `sequence2`.

Recurrence Relation

- If the characters match:
  `dp[i][j] = dp[i-1][j-1] + 1`
- If the characters do not match:
  `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`

Base Case

- If either sequence is empty, the LCS length is `0`:
  `dp[i][0] = dp[0][j] = 0`

Algorithm Steps

1. Create a 2D array `dp` of size `(n+1) x (m+1)` where `n` and `m` are the lengths of the two sequences.
2. Initialize the first row and column to `0`.
3. Fill the `dp` table using the recurrence relation.
4. The value at `dp[n][m]` gives the length of the LCS.
5. To reconstruct the LCS, trace back through the table.

C++ Code for LCS

cpp

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

// Function to find the LCS
string findLCS(const string &s1, const string &s2) {
    int n = s1.size();
    int m = s2.size();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));

    // Fill the dp table
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            if (s1[i - 1] == s2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }

    // Reconstruct the LCS
    string lcs;
    int i = n, j = m;
    while (i > 0 && j > 0) {
        if (s1[i - 1] == s2[j - 1]) {
            lcs = s1[i - 1] + lcs;
            --i;
            --j;
        } else if (dp[i - 1][j] > dp[i][j - 1]) {
            --i;
        } else {
            --j;
        }
    }

    return lcs;
}

int main() {
    string s1, s2;
    cout << "Enter first string: ";
    cin >> s1;
    cout << "Enter second string: ";
    cin >> s2;

    string lcs = findLCS(s1, s2);
    cout << "Longest Common Subsequence: " << lcs << endl;

    return 0;
}
```

Explanation of the Code

1. **Input**: Two strings `s1` and `s2`.
2. **DP Table Construction**: The `dp` table is filled using the recurrence relation.
3. **LCS Reconstruction**: Starting from `dp[n][m]`, trace back to determine the LCS.
4. **Output**: Print the LCS.

Time and Space Complexity

- **Time Complexity**: $O(n \times m)$, where n and m are the lengths of the two sequences.
- **Space Complexity**: $O(n \times m)$ for the `dp` table.