C++ with picture, O(nm)

[votrubac](https://leetcode.com/votrubac/)

80720

82753

Aug 02, 2019

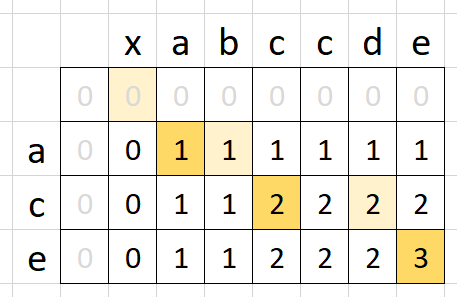
**Intuition**

LCS is a well-known problem, and there are similar problems here:

* [1092. Shortest Common Supersequence](https://leetcode.com/problems/shortest-common-supersequence/)
* [1062. Longest Repeating Substring](https://leetcode.com/problems/longest-repeating-substring/)
* [516. Longest Palindromic Subsequence](https://leetcode.com/problems/longest-palindromic-subsequence/)

Bottom-up DP utilizes a matrix m where we track LCS sizes for each combination of i and j.

* If a[i] == b[j], LCS for i and j would be 1 plus LCS till the i-1 and j-1 indexes.
* Otherwise, we will take the largest LCS if we skip a charracter from one of the string (max(m[i - 1][j], m[i][j - 1]).

This picture shows the populated matrix for "xabccde", "ace" test case.  


**C++**

int longestCommonSubsequence(string &a, string &b) {

short m[1001][1001] = {};

for (auto i = 0; i < a.size(); ++i)

for (auto j = 0; j < b.size(); ++j)

m[i + 1][j + 1] = a[i] == b[j] ? m[i][j] + 1 : max(m[i + 1][j], m[i][j + 1]);

return m[a.size()][b.size()];

}

**Complexity Analysis**

* Time: O(nm), where n and m are the string sizes.
* Memory: O(nm).

**Memory-Optimized Solution**

You may notice that we are only looking one row up in the solution above. So, we just need to store two rows.

**C++**

int longestCommonSubsequence(string &a, string &b) {

short m[2][1000] = {};

for (int i = 0; i < a.size(); ++i)

for (int j = 0; j < b.size(); ++j)

m[!(i % 2)][j + 1] = a[i] == b[j] ? m[i % 2][j] + 1 : max(m[i % 2][j + 1], m[!(i % 2)][j]);

return m[a.size() % 2][b.size()];

}

**Complexity Analysis**

* Time: O(nm), where n and m are the string sizes.
* Memory: O(min(n,m)), assuming that we will use a smaller string for the column dimension.