# 题目

给定两个字符串 text1 和 text2，返回这两个字符串的最长 公共子序列 的长度。如果不存在 公共子序列 ，返回 0 。

一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。

例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。

两个字符串的 公共子序列 是这两个字符串所共同拥有的子序列。

示例 1：

输入：text1 = "abcde", text2 = "ace"

输出：3

解释：最长公共子序列是 "ace" ，它的长度为 3 。

示例 2：

输入：text1 = "abc", text2 = "abc"

输出：3

解释：最长公共子序列是 "abc" ，它的长度为 3 。

示例 3：

输入：text1 = "abc", text2 = "def"

输出：0

解释：两个字符串没有公共子序列，返回 0 。

提示：

1 <= text1.length, text2.length <= 1000

text1 和 text2 仅由小写英文字符组成。

# 分析

方法一：动态规划

思路：

上述代码使用了动态规划来解决最长公共子序列的问题。具体思路如下：

1、定义一个二维数组dp，其中dp[i][j]表示text1的前i个字符和text2的前j个字符的最长公共子序列的长度。

2、初始化dp的第一行和第一列，即dp[0][j]和dp[i][0]，它们分别表示一个空字符串与text2和text1的最长公共子序列，因此都是 0。

3、遍历text1和text2的所有字符，如果当前字符相等，那么 dp[i][j] = dp[i-1][j-1] + 1，表示当前字符可以加入最长公共子序列；如果不相等，那么dp[i][j]的值取dp[i-1][j]和dp[i][j-1]中的较大值，表示当前字符不可能同时出现在最长公共子序列中。

4、最终dp[text1.size()][text2.size()]即为最长公共子序列的长度。

这样，通过动态规划的方法，可以在O(m\*n)的时间复杂度内解决最长公共子序列的问题。

代码：

class Solution {

public:

int longestCommonSubsequence(string text1, string text2) {

int m = text1.size();

int n = text2.size();

vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

for (int i = 1; i <= m; ++i) {

for (int j = 1; j <= n; ++j) {

if (text1[i - 1] == text2[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + 1;

} else {

dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

return dp[m][n];

}

};