# 题目

给定两个单词 word1 和 word2 ，返回使得 word1 和 word2 相同所需的最小步数。

每步可以删除任意一个字符串中的一个字符。

示例 1：

输入: word1 = "sea", word2 = "eat"

输出: 2

解释: 第一步将 "sea" 变为 "ea" ，第二步将 "eat "变为 "ea"

示例 2:

输入：word1 = "leetcode", word2 = "etco"

输出：4

提示：

1 <= word1.length, word2.length <= 500

word1 和 word2 只包含小写英文字母

# 分析

## 方法一：最长公共子序列

思路：

这个问题可以转换为求解两个字符串的最长公共子序列（Longest Common Subsequence，简称 LCS），然后用两个字符串的长度减去最长公共子序列的长度，即为所需的最小步数。

动态规划可以用来求解最长公共子序列。具体思路如下：

1、定义一个二维数组dp，其中dp[i][j]表示word1的前i个字符和word2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word2和word1的最长公共子序列，因此都是 0。

3、遍历word1和word2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[word1.size()][word2.size()]即为最长公共子序列的长度。

最小步数即为word1.size() + word2.size() - 2 \* dp[word1.size()][word2.size()]。

代码：

class Solution {

public:

int minDistance(string word1, string word2) {

int m = word1.size();

int n = word2.size();

// Initialize dp array

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

// Calculate LCS length

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

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

if (word1[i - 1] == word2[j - 1]) {

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

} else {

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

}

}

}

// Calculate min steps

return m + n - 2 \* dp[m][n];

}

};

这段代码中，首先定义了一个二维数组 dp，用来保存最长公共子序列的长度。然后通过两层循环遍历 word1 和 word2 的所有字符，根据字符是否相等来更新 dp 数组。最后，返回最小步数，即 word1.size() + word2.size() - 2 \* dp[word1.size()][word2.size()]。