# 题目

给你两个单词 word1 和 word2， 请返回将 word1 转换成 word2 所使用的最少操作数 。

你可以对一个单词进行如下三种操作：

插入一个字符

删除一个字符

替换一个字符

示例 1：

输入：word1 = "horse", word2 = "ros"

输出：3

解释：

horse -> rorse (将 'h' 替换为 'r')

rorse -> rose (删除 'r')

rose -> ros (删除 'e')

示例 2：

输入：word1 = "intention", word2 = "execution"

输出：5

解释：

intention -> inention (删除 't')

inention -> enention (将 'i' 替换为 'e')

enention -> exention (将 'n' 替换为 'x')

exention -> exection (将 'n' 替换为 'c')

exection -> execution (插入 'u')

提示：

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

word1 和 word2 由小写英文字母组成

# 分析

要解决将一个单词转换为另一个单词的最少操作数问题，我们可以使用动态规划（DP）方法。该问题是经典的编辑距离问题，通过构建DP表来存储子问题的解，从而高效求解。

解题思路

1、问题分析：

转换过程允许三种操作：插入、删除、替换，每种操作的代价均为1。我们需要找到将word1转换为word2的最少操作次数。

2、DP表定义：

设dp[i][j]表示将word1的前i个字符（word1[0..i-1]）转换为word2的前j个字符（word2[0..j-1]）所需的最少操作次数。

3、状态转移方程：

- 若word1[i-1] == word2[j-1]（当前字符相同）：

无需操作，dp[i][j] = dp[i-1][j-1]。

- 若word1[i-1] != word2[j-1]（当前字符不同）：

需从三种操作中选最优：

- 替换：dp[i-1][j-1] + 1（替换word1[i-1]为word2[j-1]）。

- 删除：dp[i-1][j] + 1（删除word1[i-1]）。

- 插入：dp[i][j-1] + 1（在word1中插入word2[j-1]）。

因此，dp[i][j] = min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]) + 1。

4、边界条件：

- 若i == 0（word1为空）：需插入j个字符，dp[0][j] = j。

- 若j == 0（word2为空）：需删除i个字符，dp[i][0] = i。

代码实现（含详细注释）

class Solution {

public:

int minDistance(string word1, string word2) {

int m = word1.size();

int n = word2.size();

// 创建(m+1)x(n+1)的DP表，初始化为0

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

// 边界条件1：word1为空，需插入j个字符

for (int j = 0; j <= n; ++j) {

dp[0][j] = j;

}

// 边界条件2：word2为空，需删除i个字符

for (int i = 0; i <= m; ++i) {

dp[i][0] = i;

}

// 填充DP表

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

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

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

// 当前字符相同，无需操作

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

} else {

// 当前字符不同，取三种操作的最小值+1

dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) + 1;

}

}

}

// DP表右下角即为最终结果

return dp[m][n];

}

};

代码解释

1、初始化DP表：

表的大小为(m+1)x(n+1)（m、n分别为两单词长度），额外一行一列用于处理空字符串的边界情况。

2、填充边界：

- 第一行（i=0）表示word1为空，转换为word2的前j个字符需插入j次。

- 第一列（j=0）表示word2为空，转换为word1的前i个字符需删除i次。

3、填充DP表：

从i=1和j=1开始，根据当前字符是否相同，应用对应的状态转移方程，逐步计算出所有子问题的解。

4、结果提取：

dp[m][n]存储了将整个word1转换为整个word2的最少操作数，直接返回即可。

复杂度分析

- 时间复杂度：O(m\*n)，其中m和n分别为word1和word2的长度。需要填充(m+1)x(n+1)的DP表，每个单元格的计算为O(1)。

- 空间复杂度：O(m\*n)，用于存储DP表。（可优化为O(min(m,n))，通过滚动数组复用空间，但基础实现更直观。）

该方法通过动态规划高效求解编辑距离问题，适用于题目中单词长度≤500的约束。