## 编辑距离问题

> 给你两个单词 word1 和 word2，请你计算出将 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

分析： 从题目中可以看出，数据变化时有如下操作：

```
if s1[i] == s2[j] {
   do nothing;
   i--,j--;
}
if (s1[i] != s2[j] ) {
  分别计算：
      插入，删除，替换操作的最小值返回
}

当遍历完成s1或s2之后，剩余的字符长度为base case
```

递归代码如下：

```
int minDistance(std::string word1,
                std::string word2,
                int         index1,
                int         index2) {
    if (index1 < 0) {
      return index2 + 1;
    }

    if (index2 < 0) {
      return index1 + 1;
    }

    if (word1[index1] == word2[index2]) {
      return minDistance(word1, word2, index1 - 1, index2 - 1);
    }

    if (word1[index1] != word2[index2]) {
      return std::min(
                 std::min(minDistance(word1, word2, index1 - 1, index2 -
1),
                          minDistance(word1, word2, index1 - 1, index2)),
                 minDistance(word1, word2, index1, index2 - 1)) +
             1;
```

```
    }
  }
```

通过分析可以，递归代码可以用memo(备忘录)的方案优化，优化代码如下：

```
  std::vector<std::vector<int>> memo;
  int                           minDistance(std::string word1,
                                            std::string word2,
                                            int         index1,
                                            int         index2) {
    if (index1 < 0) {
      return index2 + 1;
    }
    if (index2 < 0) {
      return index1 + 1;
    }
    if (memo[index1][index2] != 0) {
      return memo[index1][index2];
    }

    if (word1[index1] == word2[index2]) {
      memo[index1][index2] = minDistance(word1, word2, index1 - 1, index2
- 1);
    }

    if (word1[index1] != word2[index2]) {
      memo[index1][index2] =
          std::min(std::min(minDistance(word1, word2, index1 - 1, index2 -
1),
                            minDistance(word1, word2, index1, index2 -
1)),
                   minDistance(word1, word2, index1 - 1, index2)) +
          1;
    }

    return memo[index1][index2];
  }
```

由备忘录方法改为dp: 状态转移方程： $$dp[i][j] == dp[i-1][j-1] word1[i] == word2[j] \ dp[i][j] = std::min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1])+1 word1[i] != word2[j] $$ base case: dp[i][0] == i, dp[0][j] = j

```
class Solution {
public:
  int minDistance(std::string word1, std::string word2) {
    int len1 = word1.size(), len2 = word2.size();

    std::vector<std::vector<int>> dp(len1 + 1, std::vector<int>(len2 + 1,
0));
```

```
    for (int i = 1; i <= len1; i++) {
      dp[i][0] = i;
    }

    for (int j = 1; j <= len2; j++) {
      dp[0][j] = j;
    }

    for (int i = 1; i <= len1; i++) {
      for (int j = 1; j <= len2; j++) {
        if (word1[i - 1] == word2[j - 1]) {
          dp[i][j] = dp[i - 1][j - 1];
        } else {
          dp[i][j] = std::min(std::min(dp[i][j - 1] + 1, dp[i - 1][j] +
1),
                              dp[i - 1][j - 1] + 1);
        }
      }
    }

    return dp[len1][len2];
  }
};
```