求解最小编辑距离问题

原创 娄梁山 LOA算法学习笔记 2021-02-02 14:13

01 问题描述

给定两个单词 word_1 和 word_2，请计算出将 word_1 转换成 word_2 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

02 问题分析

此问题与卜老师课堂上所讲的序列连配问题类似。对于原问题，我们可以分解多个子问题。假设单词 word_1 和 word_2的长度分别为M和N，并假设OPT[m][n]表示word_1[1...m]转换成word_2[1...n]所需最少的编辑次数。这是一个典型的动态规划问题，采取动态规划的思想，通过分解成子问题缩小问题规模。因此针对子问题word_1[1...m]和word_2[1...n]，只需要考虑word_1[m]和word_2[n]应进行哪种操作（插入，删除，替换）。

如果word_1[m] == word_2[n]，则只需0次编辑，即OPT[m][n]=OPT[m-1][n-1]。

否则，当word_1[m] != word_2[n]时，有三种编辑方式：

1．把 word_1[m] 替换为 word_2[n]，需要1次编辑，然后问题变为求 word_1[1...m-1] 到 word_2[1...n-1] 的编辑距离，即 OPT[m-1][n-1];

2．把 word1_[m] 删除，需要1次编辑，然后问题变为求 word_1[1...m-1] 到 word_2[1...n] 的编辑距离，即 OPT[m-1][n];

3．在 word_1 后面插入字符 word_2[n]，需要 1 次编辑，然后问题变为求 word_1[1...m] 到 word_2[1 ...n-1] 的编辑距离，即 OPT[m][n-1];

最后比较哪种操作下编辑的次数最少，取最小值即可。

考虑到边界问题：

当n=0时，OPT[m][0]=m;

当m=0时，OPT[0][n]=n;

总结，此问题的递推关系式如下：

当word_1[m]==word_2[n] 时，OPT[m][n]=OPT[m-1][n-1]

当word_1[m]!=word_2[n]时，

OPT[m][n]=1+min(OPT[m-1][n-1],OPT[m][n-1],OPT[m-1][n])

当n=0时，OPT[m][0]=m;

当m=0时，OPT[0][n]=n;

03 正确性证明

循环不变量：OPT[m][n]表示将word_1[1...m]转换成word_2[1...n]所需要的最少编辑次数。

初始化：当两个单词中任一个长度为0时，所需要的编辑次数即为另一个长度不为0的单词长度，即当n=0时，OPT[m][0]=m；当m=0时，OPT[0][n]=n;

维护：对于原问题的子问题，考察word_1的前m个字符和word_2前n个字符。当word_1[m]==word_2[n]时，当前位置所需编辑次数为0，即OPT[m][n]=OPT[m-1][n-1]；当word_1[m]!=word_2[n]时，所以肯定需要1次编辑（插入、删除、替换），取三种情况中的最小值即可，即：OPT[m][n]=1+min(OPT[m-1][n-1],OPT[m][n-1],OPT[m-1][n])。

终止：当word_1和word_2分别遍历到最后一个字符时，同样需要考虑比较当前两个字符，当word_1[M]==word_2[N]时，当前位置所需编辑次数为0，即OPT[M][N]=OPT[M-1][N-1]；当word_1[M]!=word_2[N]时，所以肯定需要进行1次编辑（插入、删除、替换），取三种情况中的最小值即可，即：OPT[M][N]=1+min(OPT[M-1][N-1],OPT[M][N-1],OPT[M-1][N])。因此OPT[M][N]即为最终所求结果。

04 代码实现

```
intminDistance(string word1, string word2) {
        int m = word1.length();
        int n = word2.length();
        vector<vector<int>>OPT(m+1, vector<int>(n+1));
        for (int i = 0; i <= m; ++i) {
            OPT[i][0] = i;
        }
        for (int j = 0; j <= n; ++j) {
            OPT[0][j] = j;
        }
        for (int i = 1; i <= m; ++i) {
            for (int j = 1; j <= n; ++j) {
                if (word1[i-1] == word2[j-1]) {
                    OPT[i][j] = OPT[i-1][j-1];
                } else {
                    OPT[i][j] = 1 +min(OPT[i-1][j-1], min(OPT[i][j-1], OPT[i-1][j]));
                }
            }
        }
        return OPT[m][n];
    }
```

05 总结

此问题与卜老师课堂上所讲的序列连配问题类似。当遇到问题较大时，需要缩减问题规模，学会将原问题试着划分成多个子问题，可以选择从最简单的问题着手。类似的问题还有正则表达式匹配、通配符匹配等问题，都可以用上述的思想进行求解。动态规划类的问题本质是大同小异的，子问题的引入可以大大减少求解问题的复杂度。