

72-编辑距离

题述

72. 编辑距离

难度 **困难** 2363 ☆ 10 文 0 0

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')

思路

这道题看起来感觉好复杂！

但其实这是动态规划经典问题！！

老生常谈，动态规划五部曲

1. 确定dp数组（dp table）以及下标的含义

$dp[i][j]$ 表示以下标 $i-1$ 为结尾的字符串 $word1$ ，和以下标 $j-1$ 为结尾的字符串 $word2$ ，最近编辑距离为 $dp[i][j]$ 。

至于为什么表示的是 $i-1$ 和 $j-1$ ，看完后面的流程就会理解一些。

2、确定递推公式

在确定递推公式的时候，首先要考虑清楚编辑的几种操作，整理如下：

```
if (word1[i - 1] == word2[j - 1])
    不操作
if (word1[i - 1] != word2[j - 1])
    增
    删
    换
```

也就是如上4种情况。

```
if word1[i-1] == word2[j-1]:
    无需编辑
    那么dp[i][j] = dp[i-1][j-1]
```

回顾上面讲过的 $dp[i][j]$ 的定义， $word1[i - 1]$ 与 $word2[j - 1]$ 相等了，那么就不用编辑了，以下标 $i-2$ 为结尾的字符串 $word1$ 和以下标 $j-2$ 为结尾的字符串 $word2$ 的最近编辑距离 $dp[i - 1][j - 1]$ 就是 $dp[i][j]$ 了。

在整个动规的过程中，最为关键就是正确理解 $dp[i][j]$ 的定义！

$if (word1[i - 1] != word2[j - 1])$ ，此时就需要编辑了，如何编辑呢？

- 操作一： $word1$ 删除一个元素，那么就是以下标 $i - 2$ 为结尾的 $word1$ 与 $j-1$ 为结尾的 $word2$ 的最近编辑距离 再加上一个操作。

即 $dp[i][j] = dp[i - 1][j] + 1;$

- 操作二： $word2$ 删除一个元素，那么就是以下标 $i - 1$ 为结尾的 $word1$ 与 $j-2$ 为结尾的 $word2$ 的最近编辑距离 再加上一个操作。

即 $dp[i][j] = dp[i][j - 1] + 1;$

这里有同学发现了，怎么都是删除元素，添加元素去哪了。

$word2$ 添加一个元素，相当于 $word1$ 删除一个元素，例如 $word1 = "ad"$ ， $word2 = "a"$ ， $word1$ 删除元素' d '和 $word2$ 添加一个元素' d '，变成 $word1="a"$ ， $word2="ad"$ ，最终的操作数是一样！ dp 数组如下图所示的：

Diagram illustrating the transformation of a 2D array structure:

a

+	-	-	-	+	-	-	-	+
	0		1					
+	-	-	-	+	-	-	-	+

a

	1		0					
+	-	-	-	+	-	-	-	+

d

	2		1					
+	-	-	-	+	-	-	-	+

==>

a d

+	-	-	-	+	-	-	-	+
	0		1		2			
+	-	-	-	+	-	-	-	+

a

	1		0		1			
+	-	-	-	+	-	-	-	+

- 操作三：替换元素，`word1` 替换 `word1[i - 1]`，使其与 `word2[j - 1]` 相同，此时不用增加元素，那么以下标 `i-2` 为结尾的 `word1` 与 `j-2` 为结尾的 `word2` 的最近编辑距离 加上一个替换元素的操作。

即 $dp[i][j] = dp[i - 1][j - 1] + 1;$

综上, 当 `if (word1[i - 1] != word2[j - 1])` 时取最小的, 即: `dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) + 1;`

```
if (word1[i - 1] == word2[j - 1]) {
    dp[i][j] = dp[i - 1][j - 1];
}
else {
    dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) + 1;
}
```

3、dp数组初始化

$dp[i][j]$ 表示以下标 $i-1$ 为结尾的字符串 $word1$ ，和以下标 $j-1$ 为结尾的字符串 $word2$ ，最近编辑距离为 $dp[i][j]$ 。

那么 $dp[i][0]$ 和 $dp[0][j]$ 表示什么呢?

dp[i][0]：以下标i-1为结尾的字符串word1，和空字符串word2，最近编辑距离为dp[i][0]。

那么dp[i][0]就应该是i，对word1里的元素全部做删除操作，即：dp[i][0] = i;

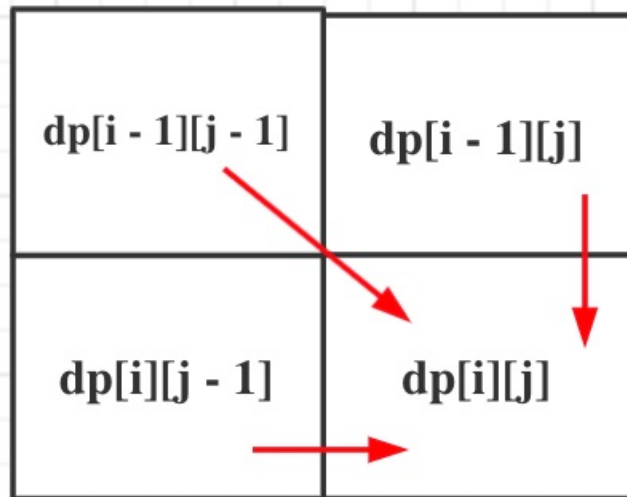
同理 $dp[0][j] = j$;

```
for (int i = 0; i <= word1.size(); i++) dp[i][0] = i;
for (int j = 0; j <= word2.size(); j++) dp[0][j] = j;
```

4、确定遍历顺序

从如下四个递推公式：

- $dp[i][j] = dp[i - 1][j - 1]$
- $dp[i][j] = dp[i - 1][j - 1] + 1$
- $dp[i][j] = dp[i][j - 1] + 1$
- $dp[i][j] = dp[i - 1][j] + 1$
- $dp[i][j]$ 是依赖左方, 上方和左上方元素的
-




公众号：代码随想录

- dp矩阵中一定是从左到右从上到下去遍历。

```
for (int i = 1; i <= word1.size(); i++) {
    for (int j = 1; j <= word2.size(); j++) {
        if (word1[i - 1] == word2[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1];
        }
        else {
            dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) +
1;
        }
    }
}
```

5、举例推导

		r	o	s	
		0	1	2	3
h		1	1	2	3
o		2	2	1	2
r		3	2	2	2
s		4	3	3	2
e		5	4	4	3



公众号: 代码随想录

题解

C++贪心

```

class Solution {
public:
    int minDistance(string word1, string word2) {
        vector<vector<int>>> dp(word1.size() + 1, vector<int>(word2.size() + 1, 0));
        for (int i = 0; i <= word1.size(); i++) dp[i][0] = i;
        for (int j = 0; j <= word2.size(); j++) dp[0][j] = j;
        for (int i = 1; i <= word1.size(); i++) {
            for (int j = 1; j <= word2.size(); j++) {
                if (word1[i - 1] == word2[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1];
                }
                else {
                    dp[i][j] = min({dp[i - 1][j - 1], dp[i - 1][j], dp[i][j - 1]}) + 1;
                }
            }
        }
    }
}
  
```

```

        }
    }
    return dp[word1.size()][word2.size()];
}
};

```

Python

```

class Solution:
    def minDistance(self, word1: str, word2: str) -> int:
        dp = [[0] * (len(word2)+1) for _ in range(len(word1)+1)]
        for i in range(len(word1)+1):
            dp[i][0] = i
        for j in range(len(word2)+1):
            dp[0][j] = j
        for i in range(1, len(word1)+1):
            for j in range(1, len(word2)+1):
                if word1[i-1] == word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    dp[i][j] = min(dp[i-1][j-1], dp[i-1][j], dp[i][j-1]) + 1
        return dp[-1][-1]

```

思考
