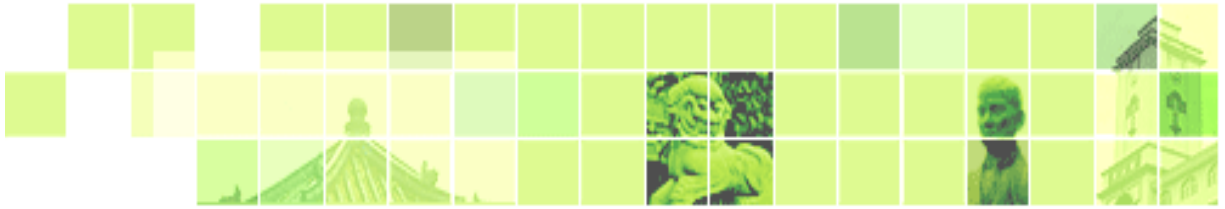
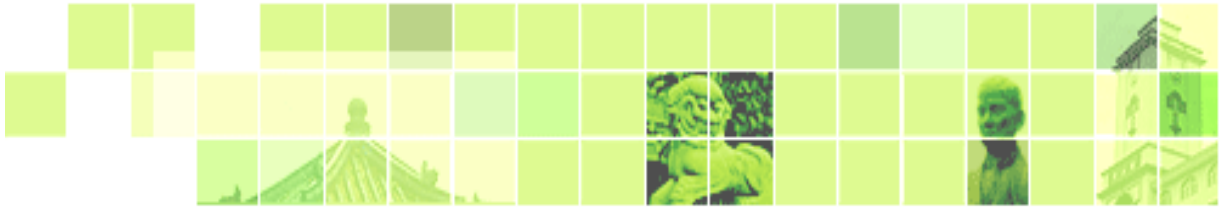


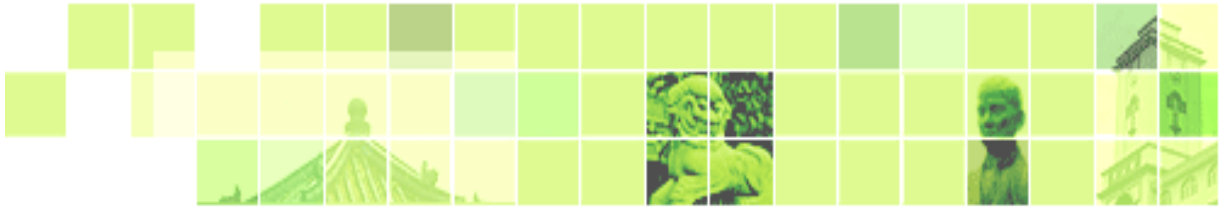
第3节练习：编辑距离



编辑距离

给定两个字符串S和T，对于T，我们允许三种操作：

1. 在任意位置添加任意字符
2. 删除存在的任意字符
3. 修改任一字符



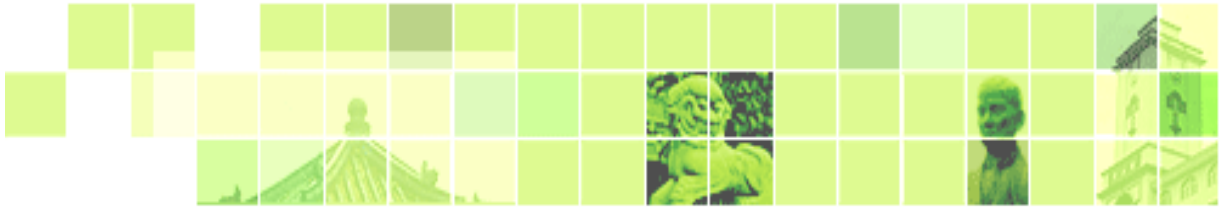
编辑距离

问最少操作多少次可以把字符串T变为S？

例：S=“ABCF”　　T=“DBFG”

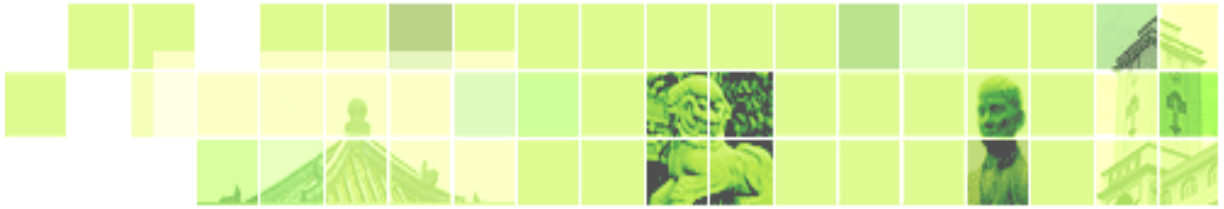
则可以

1. 把D替换为A
2. 删掉G
3. 插入C



分析

- 上例中最少的操作次数，通常被称为编辑距离，即最短编辑距离。
- 从字符串对齐的角度来理解：给定字符串S和T，我们可以用一种特殊字符促成两个字符串对齐。
- 在S和T中任意添加特殊字符“-”使得它们长度相同，将这两个串对齐。
- 最终两个串相同位置出现了不同字符，扣1分
- 我们要使得这两个串对齐扣分尽量少

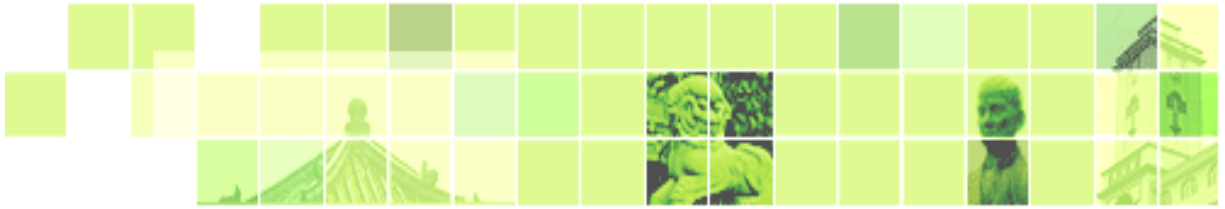


解答

对于上例，我们采取了这样的对齐方式：

```
12345
ABCF-
DB-FG
```

注意：两个“-”相对是没有意义的

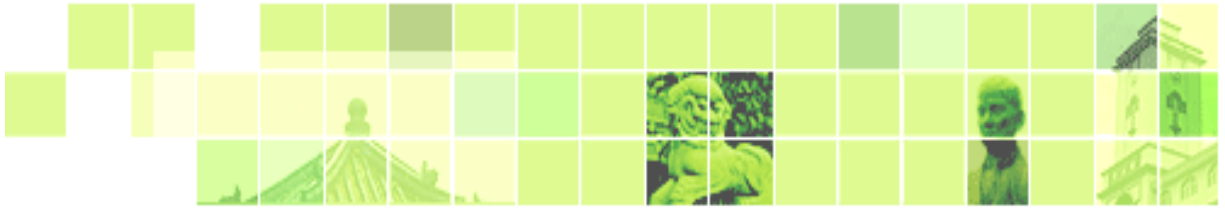


解答

对于上例，我们采取了这样的对齐方式：

```
    12345
S   ABCF-
T   DB-FG
```

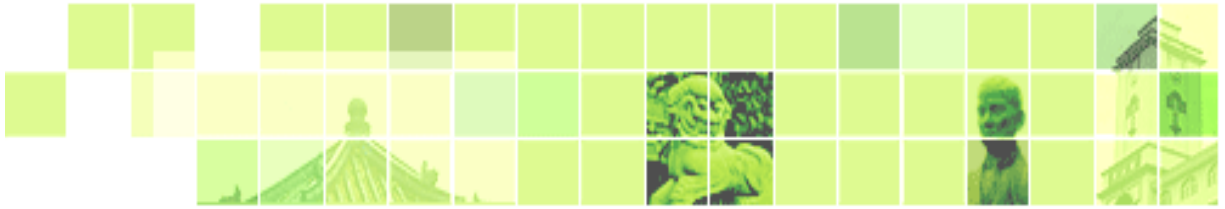
1. S，T对应位置都是普通字符，若相同，则不扣分。例如位置2，4
2. S，T对应位置都是普通字符，若不同，则扣1分。例如位置1
3. S在该位置是特殊字符，T在该位置是普通字符，扣1分。例如位置5
4. S在该位置是普通字符，T在该位置是特殊字符，扣1分。例如位置3



扣分项目对应着什么？

1. S，T对应位置都是普通字符，若相同，则不扣分。例如位置2，4
2. S，T对应位置都是普通字符，若不同，则扣1分。例如位置1
3. S在该位置是特殊字符，T在该位置是普通字符，扣1分。例如位置5
4. S在该位置是普通字符，T在该位置是特殊字符，扣1分。例如位置3

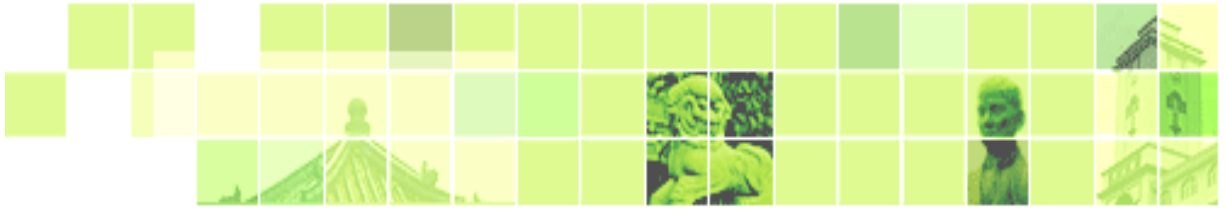
1. **不扣分，直接对应**
2. **替换T中对应位置的字符**
3. **删除T中对应位置的字符**
4. **在T中插入该字符**



算法详解

设 f(i, j) 表示 S 的前 i 位和 T 的前 j 位对齐后的最少扣分，则最后一位对齐的情况为：

1. **S[i] == T[j]，这时前i-1和j-1位都已对齐，这种情况下最少的扣分为f(i-1, j-1)**
2. **S[i] ≠ T[j]，此时最少扣分为f(i-1, j-1)+1**
3. **S的前i位和T的前j-1位已经对齐，此时最少扣分为f(I, j-1)+1**
4. **S的前i-1位和T的前j位对齐，此时的最少扣分为f(I, j-1)+1**



算法详解

定义函数same(i, j)，表示如果S[i] == T[j]则为0，否则为1

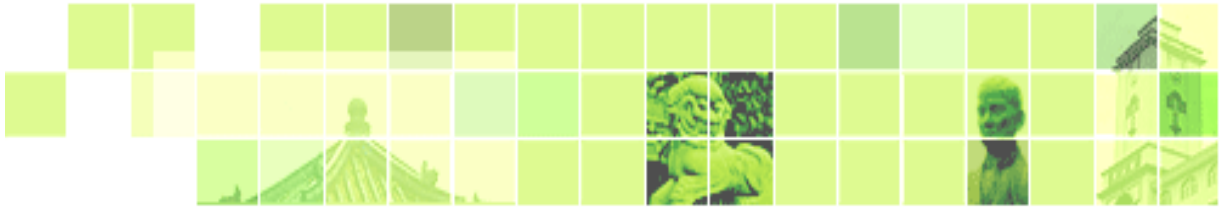
则递推表达式为：

$$f(i,j) = min(f(i-1,j-1) + same(i,j), f(i-1,j) + 1, f(i,j-1) + 1)$$

初值：

$$f(0,j) = j$$

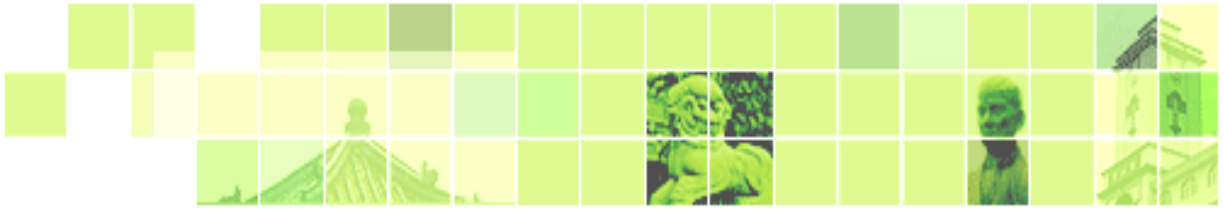
$$f(i,0) = i$$



伪代码

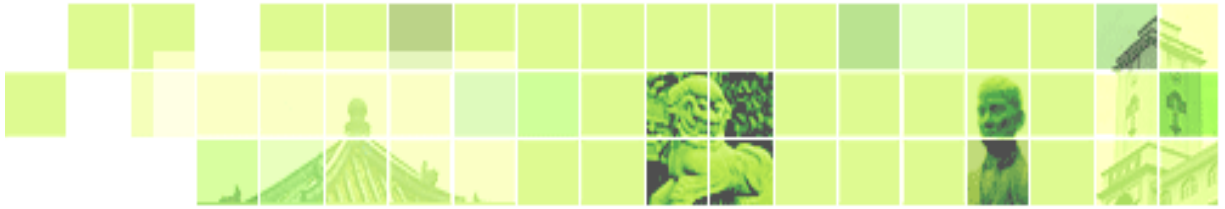
```
for j = 0 to n do
    f[j] = j
endfor

for i = 1 to m do
    last = f[0]
    f[0] = i
    for j = 1 to n do
        temp = f[i,j]
        f[i,j] = min(last + same(i,j), temp + 1, f[j - 1] + 1)
        last = temp
    endfor
endfor
```



练习

请用python实现求解最小编辑距离函数



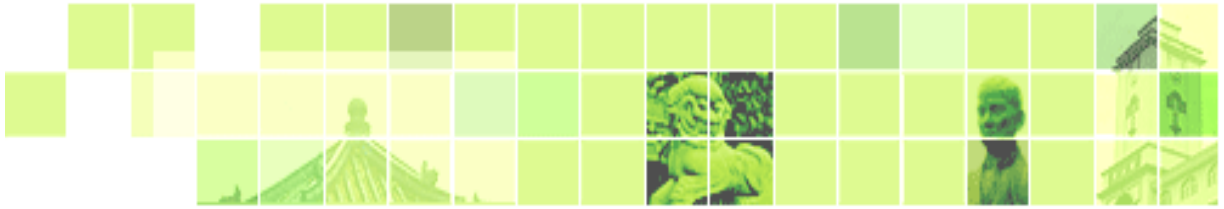
解答

请用python实现求解最小编辑距离函数

```
def editdis(str1, str2):
    len_str1 = len(str1) + 1
    len_str2 = len(str2) + 1
    #create matrix
    matrix = [0 for n in range(len_str1 * len_str2)]
    #init x axis
    for i in range(len_str1):
        matrix[i] = i
    #init y axis
    for j in range(0, len(matrix), len_str1):
        if j % len_str1 == 0:
            matrix[j] = j

    for i in range(1, len_str1):
        for j in range(1, len_str2):
            if str1[i-1] == str2[j-1]:
                cost = 0
            else:
                cost = 1
            matrix[j*len_str1+i] = min(matrix[(j-1)*len_str1+i]+1,
                                       matrix[j*len_str1+(i-1)]+1,
                                       matrix[(j-1)*len_str1+(i-1)] + cost)

    return matrix[-1]
```

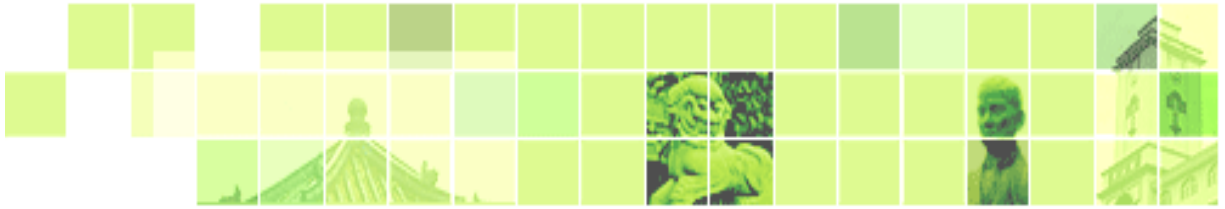


解答

调用difflib包实现：

```
def difflib_editdis(str1, str2):
    leven_cost = 0
    s = difflib.SequenceMatcher(None, str1, str2)
    for tag, i1, i2, j1, j2 in s.get_opcodes():


        if tag == 'replace':
            leven_cost += max(i2-i1, j2-j1)
        elif tag == 'insert':
            leven_cost += (j2-j1)
        elif tag == 'delete':
            leven_cost += (i2-i1)
    return leven_cost
```

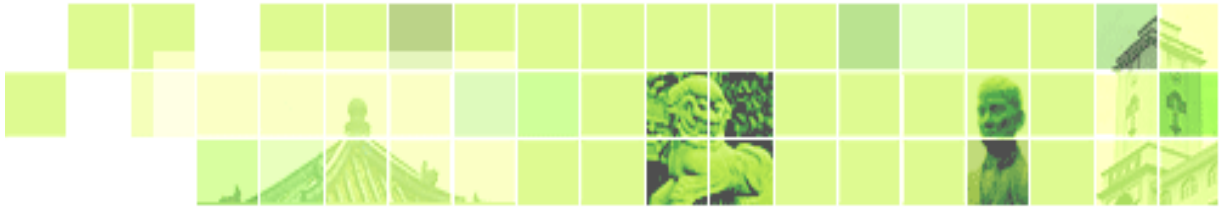


练习

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character



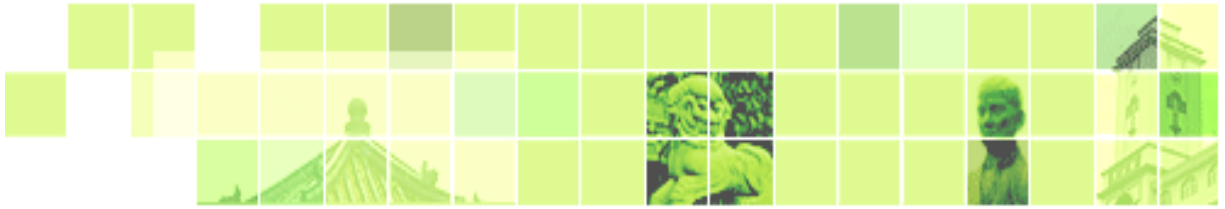
练习

Example 1

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2

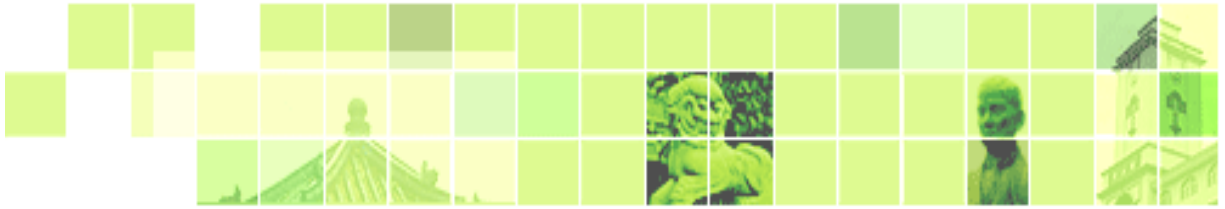
```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```



解答

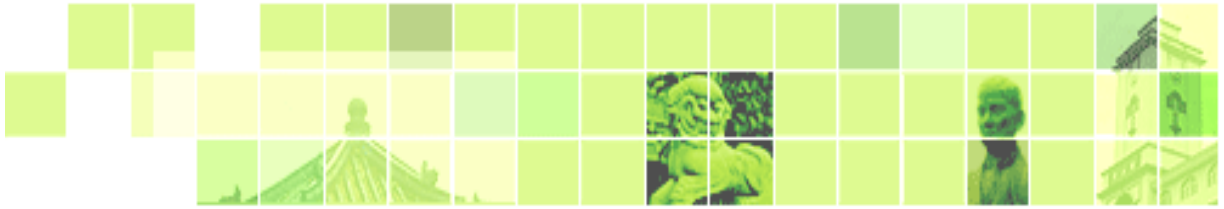
编辑距离求解是典型的动态规划问题。

- 定义状态矩阵dp[m][n]，其中m=length(*word1*)+1，n=length(*word2*)+1（考虑到word1或word2为空的情况）
- 定义dp[i][j]为*word1*中前i个字符组成的串，与*word2*中前j个字符组成的串的编辑距离



解答

- 插入操作：在*word1*的前i个字符后插入一个字符，使得插入的字符等于新加入的*word2*[j]。插入操作对于原*word1*字符来说，i是没有前进的；而对于*word2*来说，j前进了一位。此时dp[i][j]=dp[i][j-1]+1
- 删除操作：在*word1*的第i个字符后删除一个字符，使得删除后的字符串*word1*[:i-1]与*word2*[:j]相同。删除操作对于原*word2*字符来说，j是没有前进的；而对于*word1*来说，i前进了一位。此时dp[i][j]=dp[i-1][j]+(0/1)



解答

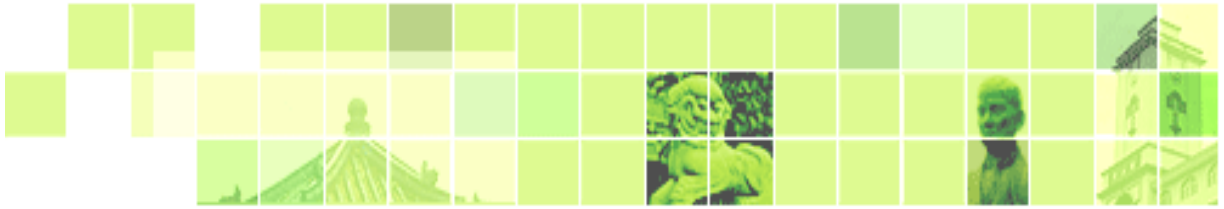
```
class Solution:

    def minDistance(self, word1, word2):
        m=len(word1)+1; n=len(word2)+1
        dp = [[0 for i in range(n)] for j in range(m)]

        for i in range(n):
            dp[0][i]=i
        for i in range(m):
            dp[i][0]=i
        for i in range(1,m):
            for j in range(1,n):
                if word1[i-1] == word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    dp[i][j] = min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1]) + 1

        return dp[m-1][n-1]

word1 = "intention"
word2 = "execution"
test = Solution()
print(test.minDistance(word1, word2))
```



Thank you !