

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №12**  
**дисциплины «Алгоритмизация»**

Выполнил:  
Степанов Леонид Викторович  
2 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение  
средств вычислительной  
техники и автоматизирование  
систем», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А., канд. техн. наук,  
доцент, доцент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2023 г.

Порядок выполнения работы:

Алгоритм Левенштейна

## Расстояние редактирования

**Вход:** строки  $A[1 \dots n]$  и  $B[1 \dots m]$ .

**Выход:** минимальное количество вставок, удалений и замен символов, необходимое для преобразования  $A$  в  $B$ . Данное число называется **расстоянием редактирования** и **расстоянием Левенштейна**.

Рисунок 1 – Входные и выходные данные

Алгоритм поиска расстояния редактирования динамического программирования сверху вниз:

## Дин. прог. сверху вниз

### Инициализация

создать двумерный массив  $D[0 \dots n, 0 \dots m]$   
инициализировать все ячейки значением  $\infty$

### Функция $\text{EDITDISTTD}(i, j)$

```
если  $D[i, j] = \infty$ :  
    если  $i = 0$ :  $D[i, j] \leftarrow j$   
    иначе если  $j = 0$ :  $D[i, j] \leftarrow i$   
    иначе:  
         $ins \leftarrow \text{EDITDISTTD}(i, j - 1) + 1$   
         $del \leftarrow \text{EDITDISTTD}(i - 1, j) + 1$   
         $sub \leftarrow \text{EDITDISTTD}(i - 1, j - 1) + \text{diff}(A[i], B[j])$   
         $D[i, j] \leftarrow \min(ins, del, sub)$ 
```

вернуть  $D[i, j]$

50:03 из 2:33:37

Рисунок 2 – Алгоритм

```
def edit_dist_td(i, j):
    """
    Динамическое программирование сверху вниз
    """
    if matrix[i][j] == infinity:
        if i == 0:
            matrix[i][j] = j
        elif j == 0:
            matrix[i][j] = i
        else:
            ins = edit_dist_td(i, j-1) + 1
            delete = edit_dist_td(i-1, j) + 1
            sub = edit_dist_td(i-1, j-1) + (a[i-1] != b[j-1])
            matrix[i][j] = min(ins, delete, sub)

    return matrix[i][j]
```

Рисунок 3 – Фрагмент кода

Алгоритм поиска расстояния редактирования динамического программирования снизу вверх:

Дин. прог. снизу вверх

Функция  $\text{EDITDISTBU}(A[1 \dots n], B[1 \dots m])$

создать массив  $D[0 \dots n, 0 \dots m]$   
 для  $i$  от 0 до  $n$ :  
 $D[i, 0] \leftarrow i$   
 для  $j$  от 0 до  $m$ :  
 $D[0, j] \leftarrow j$   
 • для  $i$  от 1 до  $n$ :  
 • для  $j$  от 1 до  $m$ :  
 •  $c \leftarrow \text{diff}(A[i], B[j])$   
 $D[i, j] \leftarrow \min(D[i-1, j] + 1, D[i, j-1] + 1, D[i-1, j-1] + c)$   
 вернуть  $D[n, m]$

Рисунок 4 – Алгоритм

```
def edit_dist_bu():
    """
    Динамическое программирование снизу вверх
    """
    matrix = []
    for i in range(len_a+1):
        matrix.append([i])
    for j in range(1, len_b+1):
        matrix[0].append(j)
    for i in range(1, len_a+1):
        for j in range(1, len_b+1):
            c = a[i-1] != b[j-1]
            matrix[i].append(min(
                matrix[i-1][j] + 1,
                matrix[i][j-1] + 1,
                matrix[i-1][j-1] + c
            ))
    return matrix
```

Рисунок 5 – Фрагмент кода

Восстановление решения по матрице:

```
def restore():
    """
    Восстановление решения
    """
    str_re1, str_re2 = [], []
    i, j = len_a, len_b
    while (i, j) != (0, 0):
        if i != 0 and matrix[i][j] == matrix[i-1][j] + 1:
            str_re1.append(a[i-1])
            str_re2.append('-')
            i -= 1

        elif j != 0 and matrix[i][j] == matrix[i][j-1] + 1:
            str_re1.append('-')
            str_re2.append(b[j-1])
            j -= 1

        elif matrix[i][j] == matrix[i-1][j-1] + (a[i-1] != b[j-1]):
            str_re1.append(a[i-1])
            str_re2.append(b[j-1])
            i -= 1
            j -= 1

    str_re1.reverse()
    str_re2.reverse()

    return (str_re1, str_re2)
```

Рисунок 6 – Фрагмент кода

Результат работы алгоритма:

```
PS C:\Users\Leo\Desktop\gitalg\lab12-alg> python3 .\prog\levenshtein.py
5
['e', 'd', 'i', '-', 't', 'i', 'n', 'g', '-']
['-', 'd', 'i', 's', 't', 'a', 'n', 'c', 'e']
PS C:\Users\Leo\Desktop\gitalg\lab12-alg> █
```

Рисунок 7 – Результат работы программы levinshtein.py

Вывод: в ходе выполнения работы мы познакомились с алгоритмом Левенштейна по поиску расстояния редактирования: реализовали его двумя способами. Также стоит отметить, что в алгоритме динамического программирования сверху вниз используется рекурсия: для вычисления верхних используются все нижние, а снизу вверх заполняет матрицу по порядку.