

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 ЛАБОРАТОРНОЙ РАБОТЕ №12**  
**дисциплины «Алгоритмизация»**

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

---

(подпись)

Руководитель практики:  
Воронкин Р.А.-доцент кафедры  
инфокоммуникаций

---

(подпись)

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

Ставрополь, 2023 г.

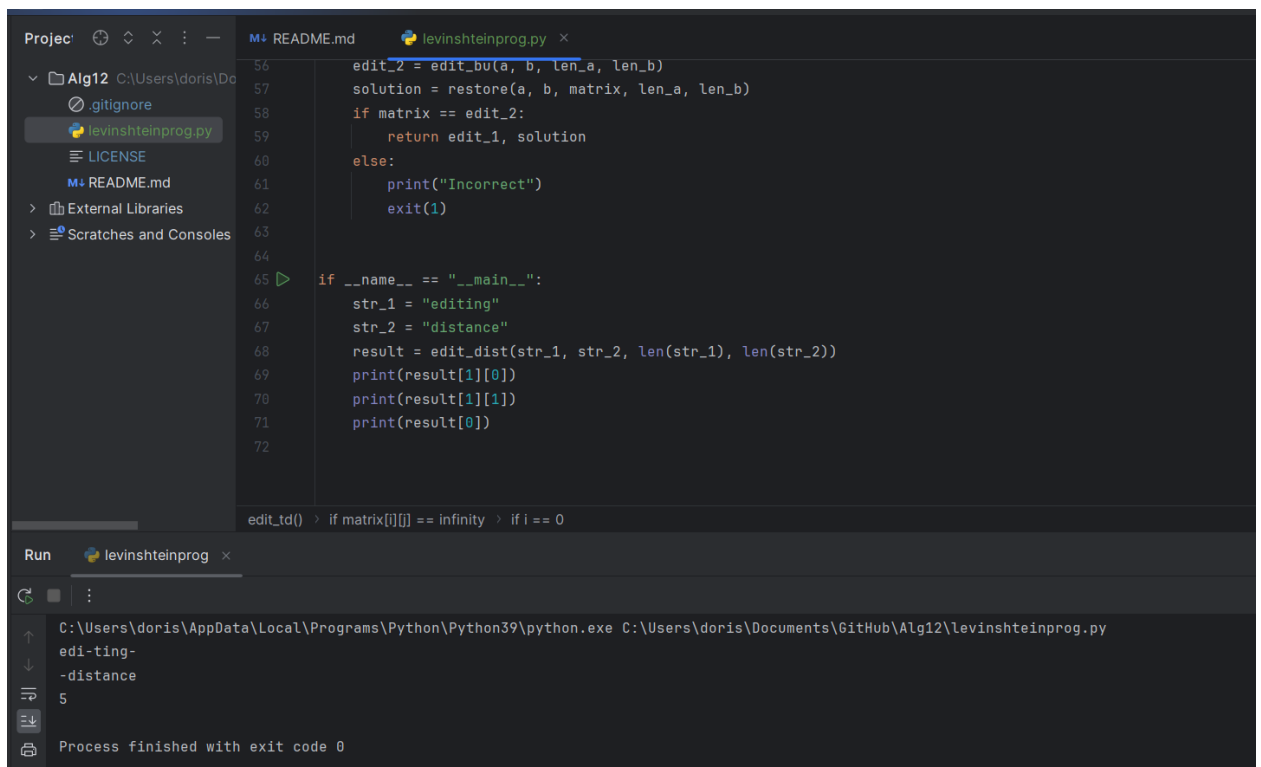
Порядок выполнения работы:

Задание 1. Выполнить алгоритм Левенштейна – расстояние редактирования.

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

Выход: минимальное количество вставок, удалений и замен символов, необходимое для преобразования A в B.

Для поиска расстояния редактирования используются 2 способа: 1) динамическое программирование сверху вниз, 2) динамическое программирование снизу вверх.



The screenshot displays a Python IDE with a project named 'Alg12'. The file explorer on the left shows the project structure, including 'levinshteinprog.py'. The main editor window shows the code for the Levenshtein distance algorithm. The code defines a function 'edit\_dist' and a main block that tests the function with the strings 'editing-' and '-distance'. The output of the program is shown in the 'Run' console at the bottom, indicating the distance is 5 and the process finished successfully.

```
56 edit_2 = edit_bu(a, b, len_a, len_b)
57 solution = restore(a, b, matrix, len_a, len_b)
58 if matrix == edit_2:
59     return edit_1, solution
60 else:
61     print("Incorrect")
62     exit(1)
63
64
65 if __name__ == "__main__":
66     str_1 = "editing-"
67     str_2 = "-distance"
68     result = edit_dist(str_1, str_2, len(str_1), len(str_2))
69     print(result[1][0])
70     print(result[1][1])
71     print(result[0])
72
```

Run levinshteinprog x

C:\Users\doris\AppData\Local\Programs\Python\Python39\python.exe C:\Users\doris\Documents\GitHub\Alg12\levinshteinprog.py

edi-ting-  
-distance  
5

Process finished with exit code 0

Рисунок 1. Программа и результат выполнения

```
M: README.md levinshteinprog.py x
4 usages new *
1 def edit_td(a, b, matrix, infinity, i, j):
2     if matrix[i][j] == infinity:
3         if i == 0:
4             matrix[i][j] = j
5         elif j == 0:
6             matrix[i][j] = i
7         else:
8             ins = edit_td(a, b, matrix, infinity, i, j - 1) + 1
9             del_val = edit_td(a, b, matrix, infinity, i - 1, j) + 1
10            sub = edit_td(a, b, matrix, infinity, i - 1, j - 1) + (a[i - 1] != b[j - 1])
11            matrix[i][j] = min(ins, del_val, sub)
12    return matrix[i][j]
13
14
15 1 usage new *
16 def edit_bu(a, b, len_a, len_b):
17     matrix = [[0] * (len_b + 1) for _ in range(len_a + 1)]
18     for i in range(len_a + 1):
19         matrix[i][0] = i
20     for j in range(1, len_b + 1):
21         matrix[0][j] = j
22     for i in range(1, len_a + 1):
23         for j in range(1, len_b + 1):
24             c = a[i - 1] != b[j - 1]
25             matrix[i][j] = min(
26                 matrix[i - 1][j] + 1,
27                 matrix[i][j - 1] + 1,
28                 matrix[i - 1][j - 1] + c
29             )
edit_td() > if matrix[i][j] == infinity > if i == 0
```

Рисунок 2. Реализация первых двух алгоритмов

```
M: README.md levinshteinprog.py x
32 def restore(a, b, matrix, len_a, len_b):
33     str_re1, str_re2 = "", ""
34     i, j = len_a, len_b
35     while i != 0 or j != 0:
36         if i != 0 and matrix[i][j] == matrix[i - 1][j] + 1:
37             str_re1 += a[i - 1]
38             str_re2 += '-'
39             i -= 1
40         elif j != 0 and matrix[i][j] == matrix[i][j - 1] + 1:
41             str_re1 += '-'
42             str_re2 += b[j - 1]
43             j -= 1
44         else:
45             str_re1 += a[i - 1]
46             str_re2 += b[j - 1]
47             i -= 1
48             j -= 1
49     return str_re1[::-1], str_re2[::-1]
50
51
52 1 usage new *
53 def edit_dist(a, b, len_a, len_b):
54     infinity = float('inf')
55     matrix = [[infinity] * (len_b + 1) for _ in range(len_a + 1)]
56     edit_1 = edit_td(a, b, matrix, infinity, len_a, len_b)
57     edit_2 = edit_bu(a, b, len_a, len_b)
58     solution = restore(a, b, matrix, len_a, len_b)
59     if matrix == edit_2:
60         return edit_1, solution
edit_td() > if matrix[i][j] == infinity > if i == 0
```

Рисунок 3. Алгоритм восстановления матрицы

Вывод: в работе был изучен алгоритм Левенштейна для поиска расстояния редактирования, а также были рассмотрены два способа его вычисления: первый способ использует рекурсию и заполняет матрицу по порядку, вычисляя сначала нижние ячейки, а затем верхние.