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:
Середа Кирилл Витальевич
1 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы:

Алгоритм Левенштейна. Алгоритм поиска расстояния редактирования динамического программирования сверху вниз:

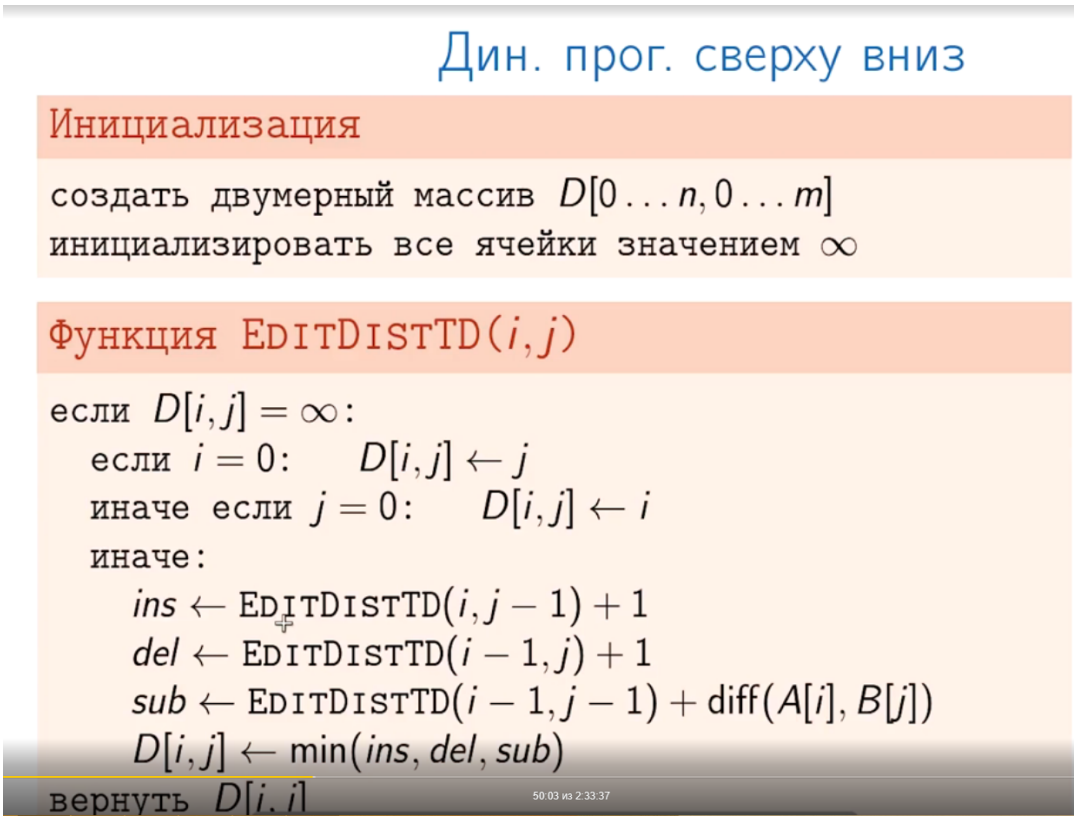


Рисунок 2 – Алгоритм

```
def edit_dist(a, b, len_a, len_b):
    """
    Поиск расстояния редактирования и восстановление решения.
    """
    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 EDITDISTBU($A[1\ldots n], B[1\ldots m]$)

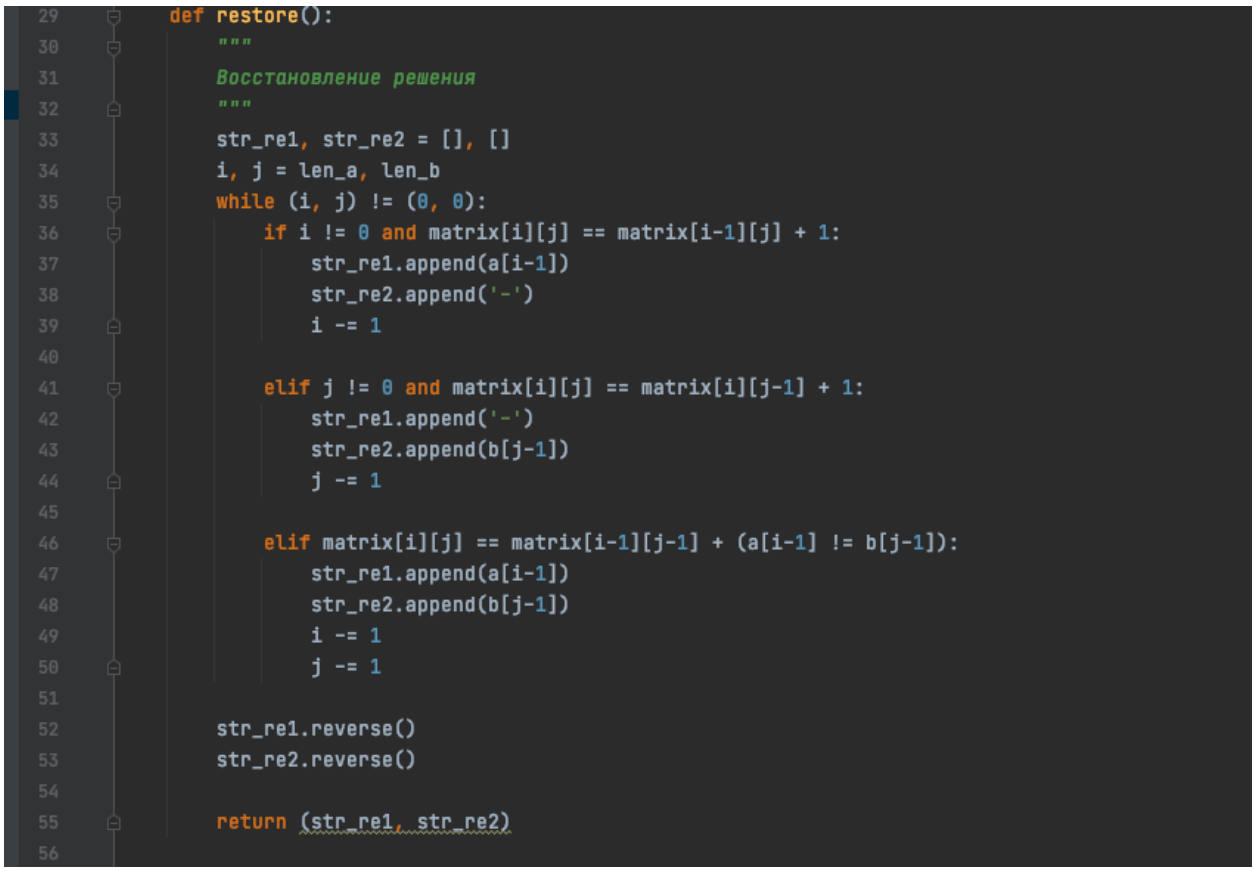
создать массив $D[0\ldots n, 0\ldots m]$
для i от 0 до n:
  $D[i, 0] \leftarrow i$
для j от 0 до m:
  $D[0, j] \leftarrow j$
для i от 1 до n:
  для j от 1 до m:
    $c \leftarrow \text{diff}(A[i], B[j])$
    $D[i, j] \leftarrow \min(D[i-1, j]+1, D[i, j-1]+1, D[i-1, j-1]+c)$
вернуть $D[n, m]$

Рисунок 4 – Алгоритм

```
58      def edit_dist_bu():
59          """
60          Динамическое программирование снизу вверх
61          """
62          matrix = []
63          for i in range(len_a+1):
64              matrix.append([i])
65          for j in range(1, len_b+1):
66              matrix[0].append(j)
67          for i in range(1, len_a+1):
68              for j in range(1, len_b+1):
69                  c = a[i-1] != b[j-1]
70                  matrix[i].append(min(
71                      matrix[i-1][j] + 1,
72                      matrix[i][j-1] + 1,
73                      matrix[i-1][j-1] + c
74                  ))
75          return matrix
76
```

Рисунок 5 – Фрагмент кода

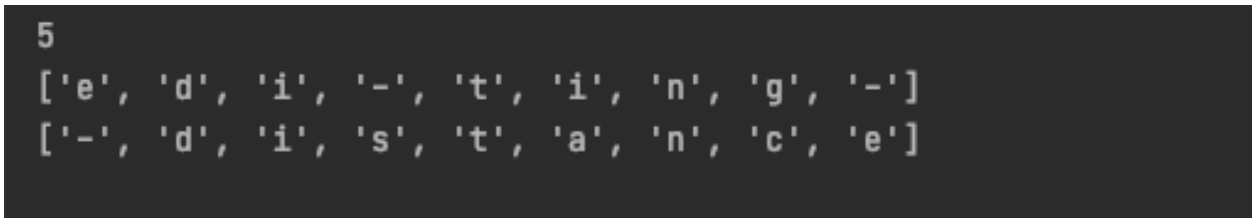
Восстановление решения по матрице:



```
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:



```
5
['e', 'd', 'i', '-', 't', 'i', 'n', 'g', '-']
['-', 'd', 'i', 's', 't', 'a', 'n', 'c', 'e']
```

Рисунок 7 – Результат работы программы levinshtein.py

Вывод: в процессе выполнения работы мы изучили алгоритм Левенштейна для определения расстояния редактирования, применив два различных подхода к его реализации. Кроме того, следует подчеркнуть, что в алгоритме динамического программирования, применяемом сверху вниз, используется рекурсия: для расчета верхних значений используются все нижние, тогда как подход снизу вверх предполагает последовательное заполнение матрицы.