Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и система управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №1

по курсу «Анализ алгоритмов»

Расстояние Левенштейна и Дамерау-Левенштейна

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019г.

Оглавление

Введение

Расстояние Левенштейна определяет минимальное количество операций, необходимых для превращения одной строки в другую. Задача определения такого минимума актуальна, так как она решает множество проблем в теории информации и компьютерной лингвистике, например:

- исправление ошибок в словах при вводе (при в поисковых ситсемах, базах данных, программах автоматического определения текста);
- сравнении текстовых файлов (к примеру, утилита diff);
- сравнение белков, генов и хромосом в биоинформатике.

1. Аналитическая часть

1.1 Задачи

Цель лабораторной работы: исследовать расстояния Левенштейна и Дамерау-Левенштейна. Для достижения этой цели были поставлены следующие задачи:

- изучить алгоритмы вычисления расстояний между строками;
- применить методы динамического программирования для матричной реализации алгоритмов;
- сравнить матричную и рекурсивную реализацию алгоритмов;
- оценить эффективность каждой из реализаций по времени и памяти.

1.2 Описание алгоритмов

При нахождении расстояния Левенштейна определяется минимального количество операций следующих видов:

- вставка (I - insert);
- удаление (D - delete);
- замена (R - replace);
- совпадение (M - match).

При нахождении расстояния Дамерау-Левенштейна добавляется операция транспозиции (T - transpose), или перестановки двух соседних символов.

Таким образом, если заданы две строки S_1 и S_2 с длинами m и n соответственно над некоторым алфавитом, то расстояние Левенштейна можно вычислить по следующей рекуррентной формуле:

$$D(S_1[1..m], S_2[1..n]) = min(D(S_1[1..m-1], S_2[1..n] + 1), \tag{1.1}$$

$$D(S_1[1..m], S_2[1..n-1] + 1), \tag{1.2}$$

$$D(S_1[1..m-1], S_2[1..n-1] + m(S_1[m], S_2[n]))) \tag{1.3}$$

где

$$m(a,b) = \begin{cases} 0, a = b \\ 1, a \neq b \end{cases},$$

Соотношения в рекурретной формуле отвечают за соотвествующие разрешённые операции:

- (1.1) - вставка;
- (1.2) - удаление;
- (1.3) - замена или совпадение в зависимости от результата $m(a, b)$.

При вычислении расстояния Дамерау-Левенштейна в рекурретную формулу вносится дополнительное соотношение в минимум:

$$D(S_1[1..m-2], S_2[1..n-2]) + 1 \tag{1.4}$$

Соотношение (1.4) вносится в качестве дополнительного аргумента минимума только при выполнении следующих условий:

- $m > 2, n > 2$;
- $S_1[m] = S_2[n-1]$;
- $S_1[m-1] = S_2[n]$.

Тривиальным случаем в рекуррентной формуле является случай, когда одна из строк пустая. В этом случае расстояние Левенштейна равно длине другой строки.

Расстояния Левенштейна и Дамерау-Левенштейна можно также вычислить, используя матрицу, в которое разрешённые операции определены следующим образом:

- движение по столбцам вправо - вставка;
- движение по строкам вверх - удаление;
- движение по диагонали - замена/совпадение.

2. Конструкторская часть

2.1 Схемы алгоритмов

2.2 Структуры данных

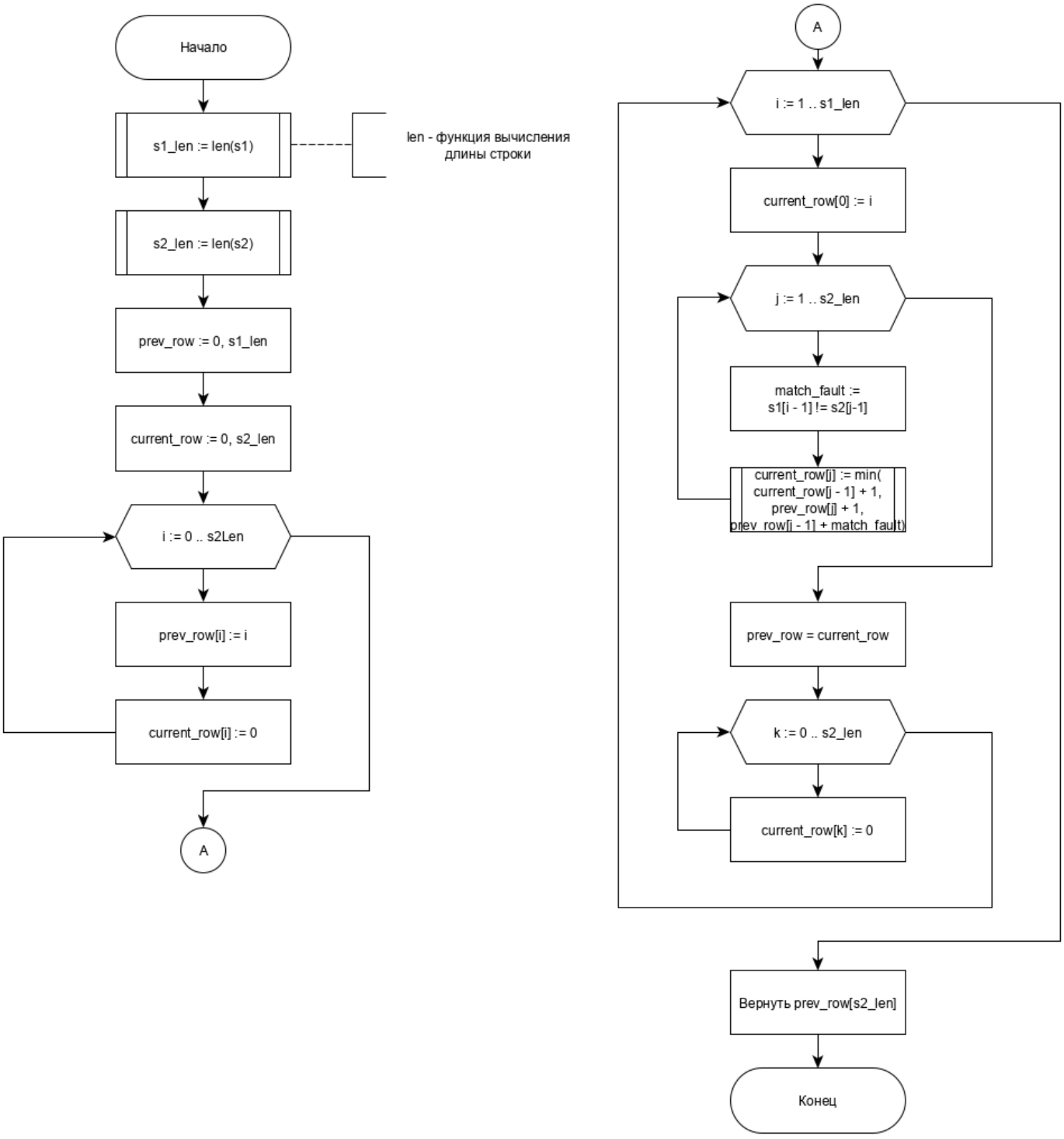


Рис. 2.1: Алгоритм Левенштейна

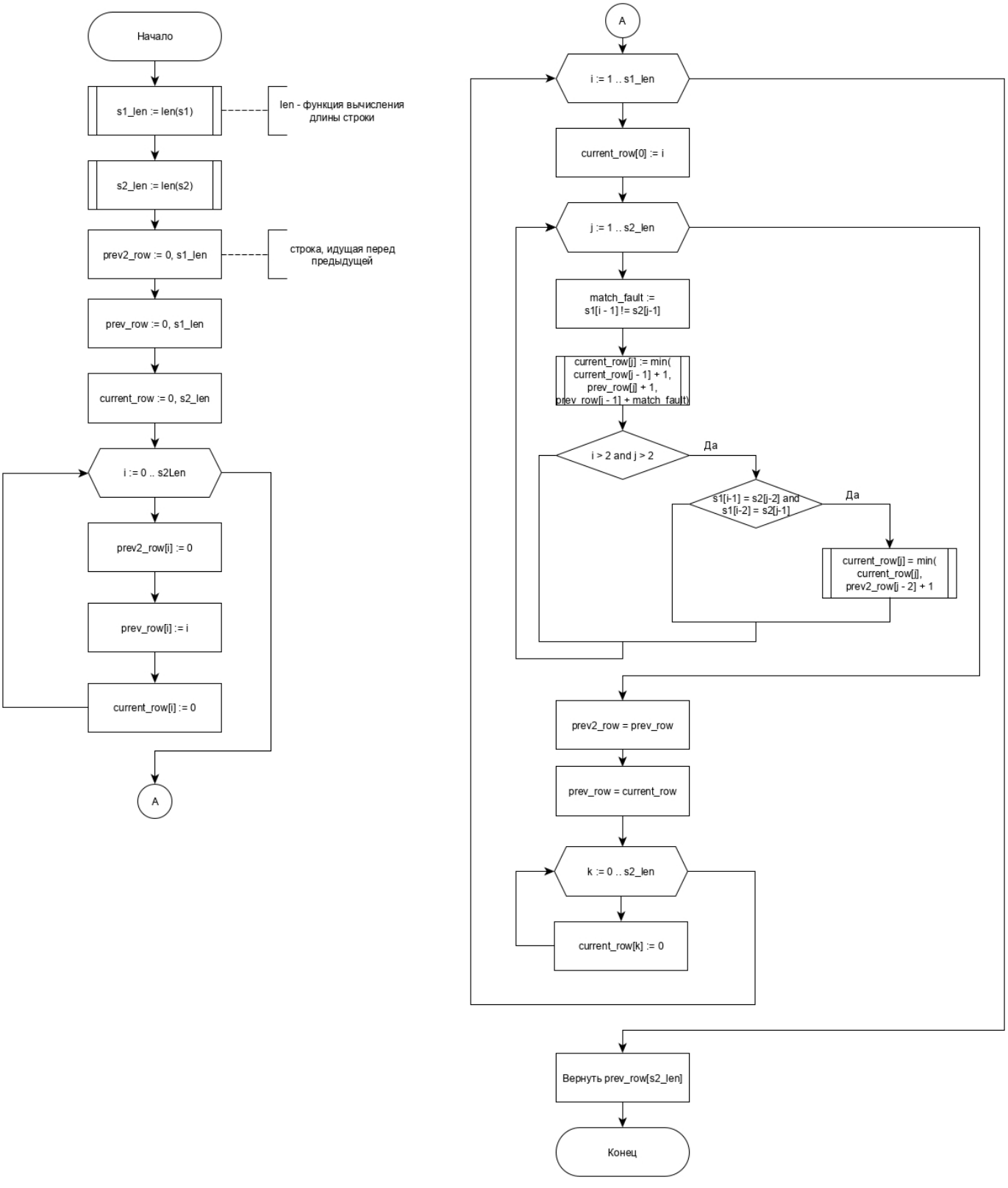


Рис. 2.2: Алгоритм Дамерау-Левенштейна (матричная реализация)

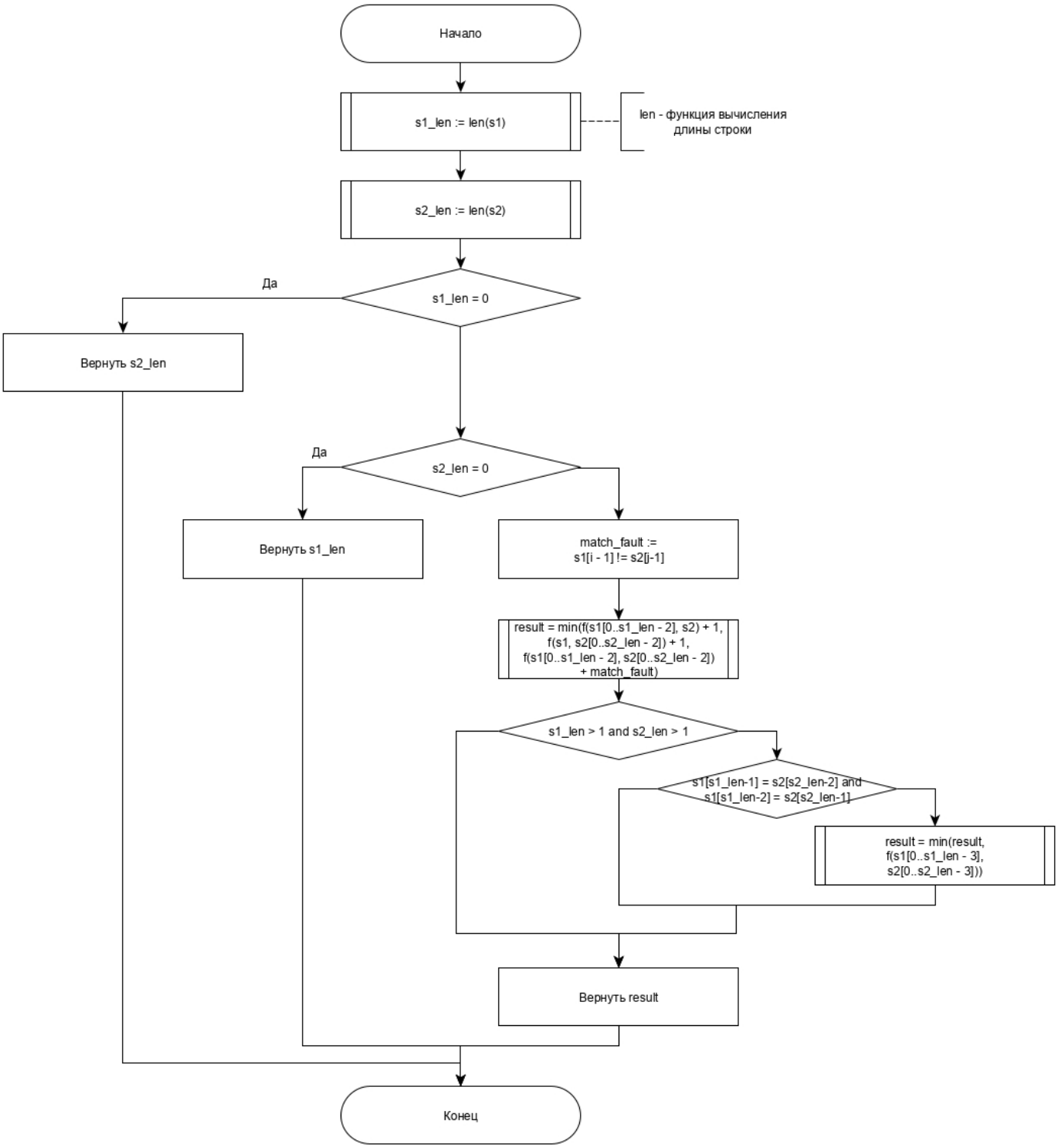


Рис. 2.3: Алгоритм Дамерау-Левенштейна (рекурсивная реализация)

3. Технологическая часть

3.1 Требования к программному обеспечению

3.2 Средства реализации

3.3 Листинг кода

Листинг 3.1: Расстояние Левенштейна (матрично)

```
def str_distance(s1, s2, to_print=False):
    s1_len = len(s1)
    s2_len = len(s2)

    # initialization of first two rows
    prev_row = [i for i in range(s2_len + 1)]   # first row - [0, 1, ..., n]
    current_row = [0] * (s2_len + 1)

    if to_print:
        print(prev_row)

    for i in range(1, s1_len + 1):          # row loop
        # current row fill
        current_row[0] = i
        for j in range(1, s2_len + 1):                  # column loop
            match_fault = int(s1[i - 1] != s2[j - 1])           # symbol
                match
            current_row[j] = min(current_row[j - 1] + 1,        #
                horizontal
                                 prev_row[j] + 1,               # vertical
                                 prev_row[j - 1] + match_fault)  # diagonal

        if to_print:
            print(current_row)

        # row switching
        prev_row = current_row
        current_row = [0] * (s2_len + 1)

    return prev_row[-1]     # value in bottom right corner of table
```

Листинг 3.2: Расстояние Дамерау-Левенштейна (матрично)

```
def str_distance(s1, s2, to_print=False):
    s1_len = len(s1)
    s2_len = len(s2)

    # initialization of first two rows
    prev2_row = [0] * (s2_len + 1)
    prev_row = [i for i in range(s2_len + 1)]
    current_row = [0] * (s2_len + 1)

    if to_print:
        print(prev_row)

    for i in range(1, s1_len + 1):          # row loop
        # current row fill
        current_row[0] = i
        for j in range(1, s2_len + 1):                  # column loop
            match_fault = int(s1[i - 1] != s2[j - 1])      # if symbol matches
            current_row[j] = min(current_row[j - 1] + 1,         # horizontal
                                 prev_row[j] + 1,                # vertical
                                 prev_row[j - 1] + match_fault)  # diagonal

            # transposition check
            if i > 2 and j > 2:
                if s1[i - 1] == s2[j - 2] and s1[i - 2] == s2[j - 1]:
                    current_row[j] = min(current_row[j],
                                         prev2_row[j - 2] + 1)

        if to_print:
            print(current_row)

        # row switching
        prev2_row = prev_row
        prev_row = current_row
        current_row = [0] * (s2_len + 1)

    return prev_row[-1]         # value in bottom right corner of table
```

Листинг 3.3: Расстояние Дамерау-Левенштейна (рекурсивно)

```
    def str_distance(s1, s2):
        s1_len = len(s1)
        s2_len = len(s2)

        if s1_len == 0:
            return s2_len
        if s2_len == 0:
            return s1_len

        match_fault = int(s1[-1] != s2[-1])

        result = min(str_distance(s1[:-1], s2) + 1,
                     str_distance(s1, s2[:-1]) + 1,
                     str_distance(s1[:-1], s2[:-1]) + match_fault)
```

```
15
16          if s1_len > 1 and s2_len > 1:
17              if s1[-1] == s2[-2] and s1[-2] == s2[-1]:
18                  result = min(result, str_distance(s1[:-2], s2[:-2]) + 1)
19
20          return result
```

Строка 1	Строка 2	Ожидание	Результат
<пустая>	<пустая>	0 0 0	0 0 0
<пустая>	а	1 1 1	1 1 1
а	<пустая>	1 1 1	1 1 1
а	а	0 0 0	0 0 0
а	б	1 1 1	1 1 1
азы	базы	1 1 1	1 1 1
компютер	компьютер	1 1 1	1 1 1
данны	данные	1 1 1	1 1 1
email.ru	mail.ru	1 1 1	1 1 1
programmmer	programmer	1 1 1	1 1 1
mail.rus	mail.ru	1 1 1	1 1 1
ашибка	ошибка	1 1 1	1 1 1
алгоритм	алгорифм	1 1 1	1 1 1
копия	копии	1 1 1	1 1 1
укрсовой	курсовой	2 1 1	2 1 1
аглоритм	алгоритм	2 1 1	2 1 1
универе	универ	2 1 1	2 1 1
курс	курсовой	4 4 4	4 4 4
курсовой	курс	4 4 4	4 4 4
курсовой	курсовик	2 2 2	2 2 2
код	закодировать	9 9 9	9 9 9
закодировать	код	9 9 9	9 9 9
ccoders	recoding	5 5 5	5 5 5
header	subheader	3 3 3	3 3 3
subheader	header	3 3 3	3 3 3
subheader	overheader	4 4 4	4 4 4

Таблица 3.1: Функциональные тесты

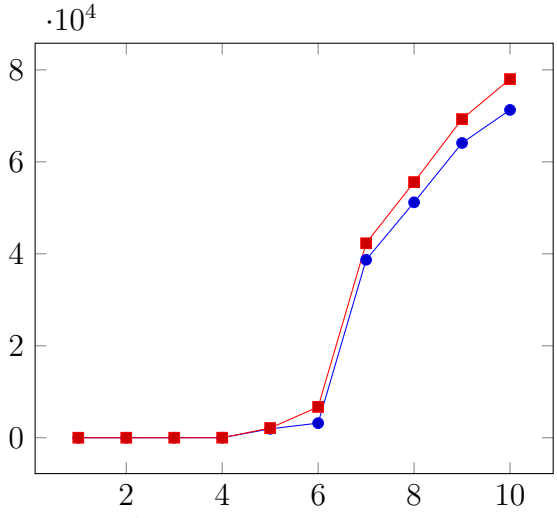
3.4 Тесты

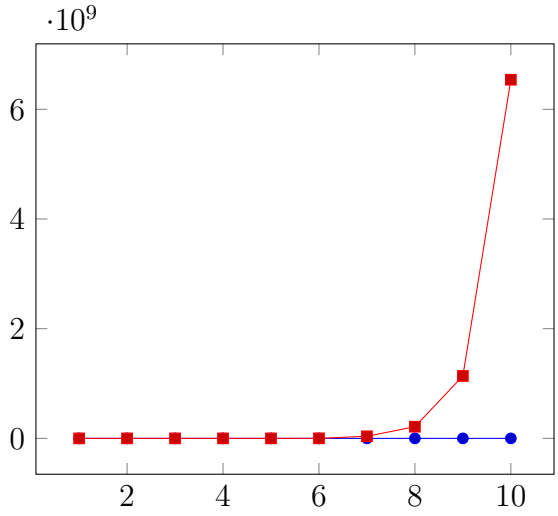
Для проверки корректности работы были подготовлены следующие функциональные тесты:

4. Экспериментальная часть

4.1 Примеры работы

4.2 Исследование времени выполнения





Заключение

Алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна между строками были изучены и реализованы: были реализованы три варианта алгоритма для получения навыка динамического программирования.
Были исследованы затраты данных вариантов реализации по времени и памяти. Экспериментально было подтверждено, что рекурсивный вариант реализации алгоритма значительно проигрывает матричным вариантам при росте длины входных строк по обоим показателям.

Список литературы