Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №1 по курсу "Анализ алгоритмов"

Тема Расстояние Левенштейна и Дамерау-Левенштейна

Студент Городский Ю.Н.

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2024 г.

Содержание

Введение

Расстояние Левенштейна и Дамерау-Левенштейна представляют минимальное количество односимвольных операций (вставка, удаление и замена символов), необходимых для преобразования одной последовательности символов в другую. Используются для исправления ошибок в слове, сравнения текстовых файлов и иных операций с символьными последовательностями.

Цель лабораторной работы — реализация и сравнение алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна. Для достижения поставленной цели необходимо выполнить следующие задачи:

- реализовать алгоритмы поиска расстояний (Левенштейна с рекурсией, Левенштейна и Дамерау-Левенштейна с использованием динамического программирования);
- проанализировать затраченное процессорное время рекурсивной и матричной реализации алгоритмов на основе эксперементальных данных;
- описать и обосновать полученные результаты.

1 Аналитическая часть

В данном разделе рассматриваются алгоритмы нахождения расстояний Левенштейна и Дамерау-Левенштейна.

1.1 Описание алгоритмов

Растояние Левенштейна[1] равно минимальному числу односимвольных операций преобразования (удаление, вставка, замена), необходимых для получения одной строки из другой.

1.1.1 Расстояние Левенштейна

Заданы две строки S_1 и S_2 над некоторым алфавитом длиной M и N соответственно, тогда расстояние Левенштейна $d(S_1, S_2)$ можно найти по рекуррентной формуле:

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ min\{ & \\ \quad D(i, j-1) + 1, & \\ \quad D(i-1, j) + 1, & j > 0, i > 0 \\ \quad D(i-1, j-1) + m(S_1[i], S_2[j]) & \\ \} & \end{cases} \tag{1.1}$$

где $m(a,b)$ равна нулю, если $a = b$ и единице в противном случае
$min(a,b,c)$ возвращает наименьший из аргументов.

1.1.2 Расстояние Дамерау-Левенштейна

В Алгоритме нахождения Дамерау-Левенштейна[1] добавляется операция перестановки символов. В таком случае расстояние Дамерау-Левенштейна $d(S_1, S_2)$ может быть найдено по формуле:

$$d(i,j) = \begin{cases} max(i,j), & min(i,j) = 0 \\ min \begin{cases} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1)+1_{(a_i \neq b_i)} \\ d(i-2,j-2)+1 \end{cases} & , i,j > 1, a_i = b_{j-1}, a_{i-1} = b_j \\ min \begin{cases} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1)+1_{(a_i \neq b_i)} \end{cases} & , \end{cases} \tag{1.2}$$

Вывод

В разделе были рассмотрены два алгоритма нахождения расстояний между строками: Левенштейна и Дамерау-Левенштейна.

2 Конструкторская часть

В разделе будут представлены схемы алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна, дано описание используемх типов данных, оценки памяти, описана структура ПО.

2.1 Представление алгоритмов

Входными данными для алгоритмов являются строки S_1 и S_2, выходными данными - число, искомое расстояние.

На рисунках 2.1 - 2.3 приведены схемы реализованных алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна.

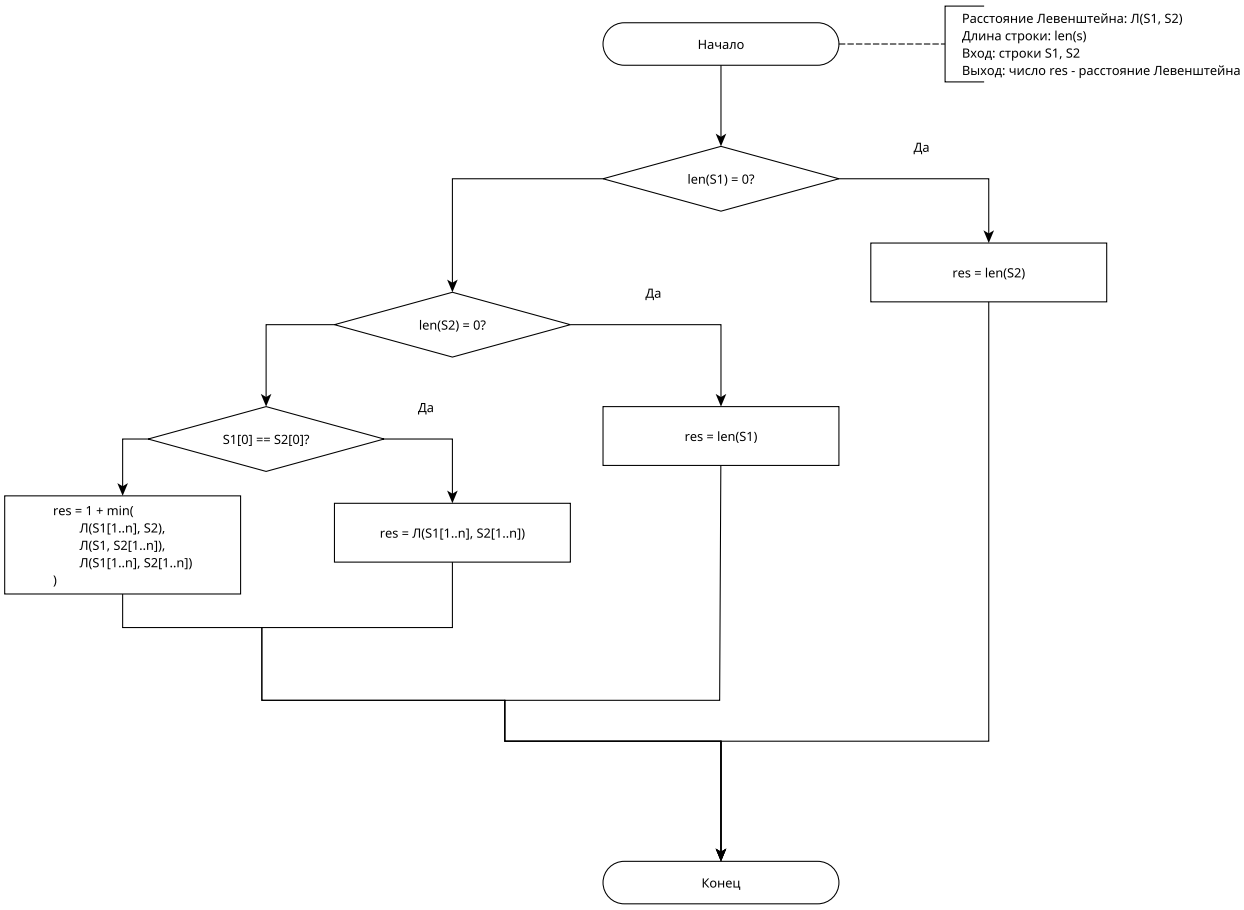

Рисунок 2.1 – Схема рекурсивного алгоритма нахождения расстояния Левенштейна

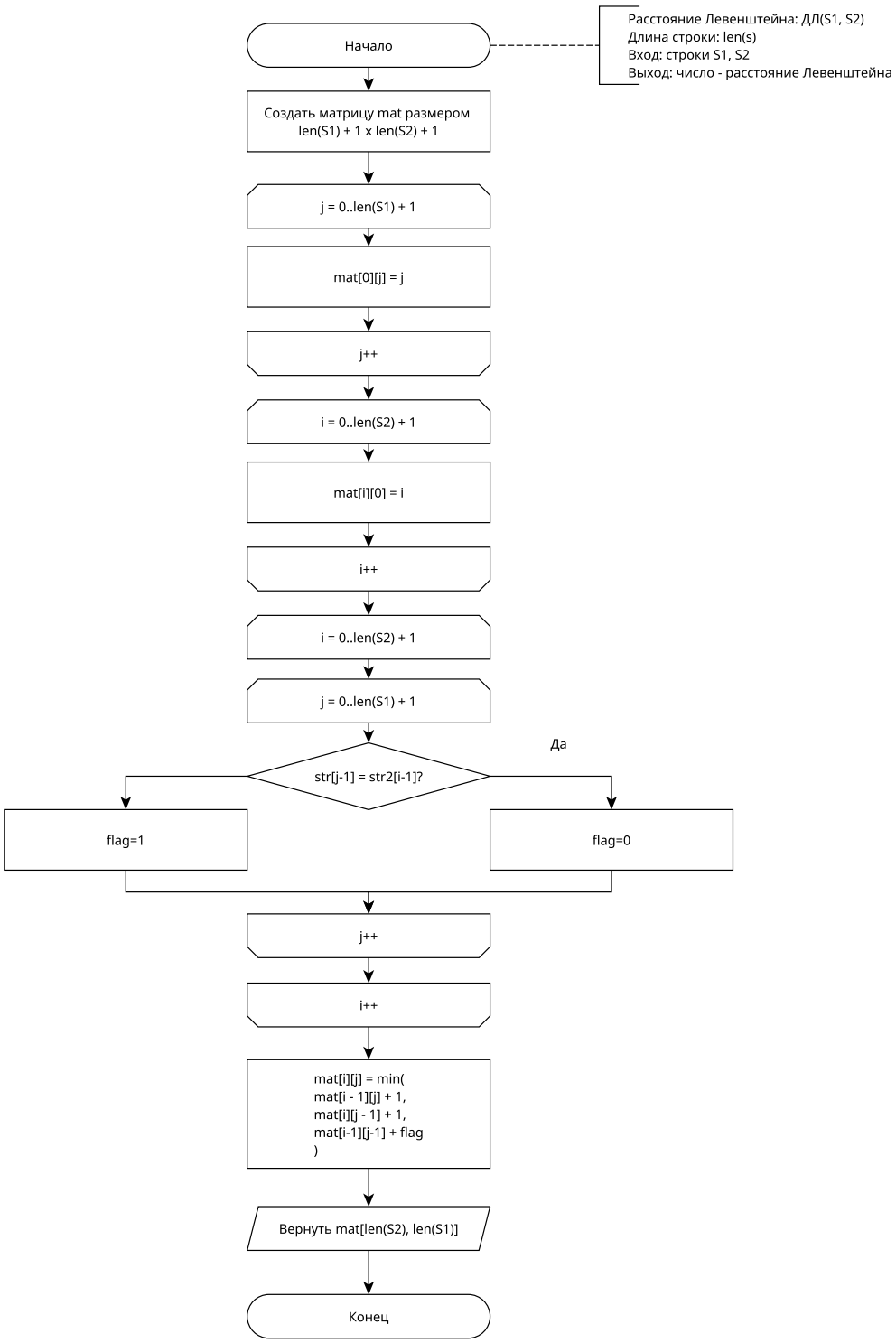

Рисунок 2.2 – Схема алгоритма нахождения расстояния Левенштейна с использованием динамического программирования

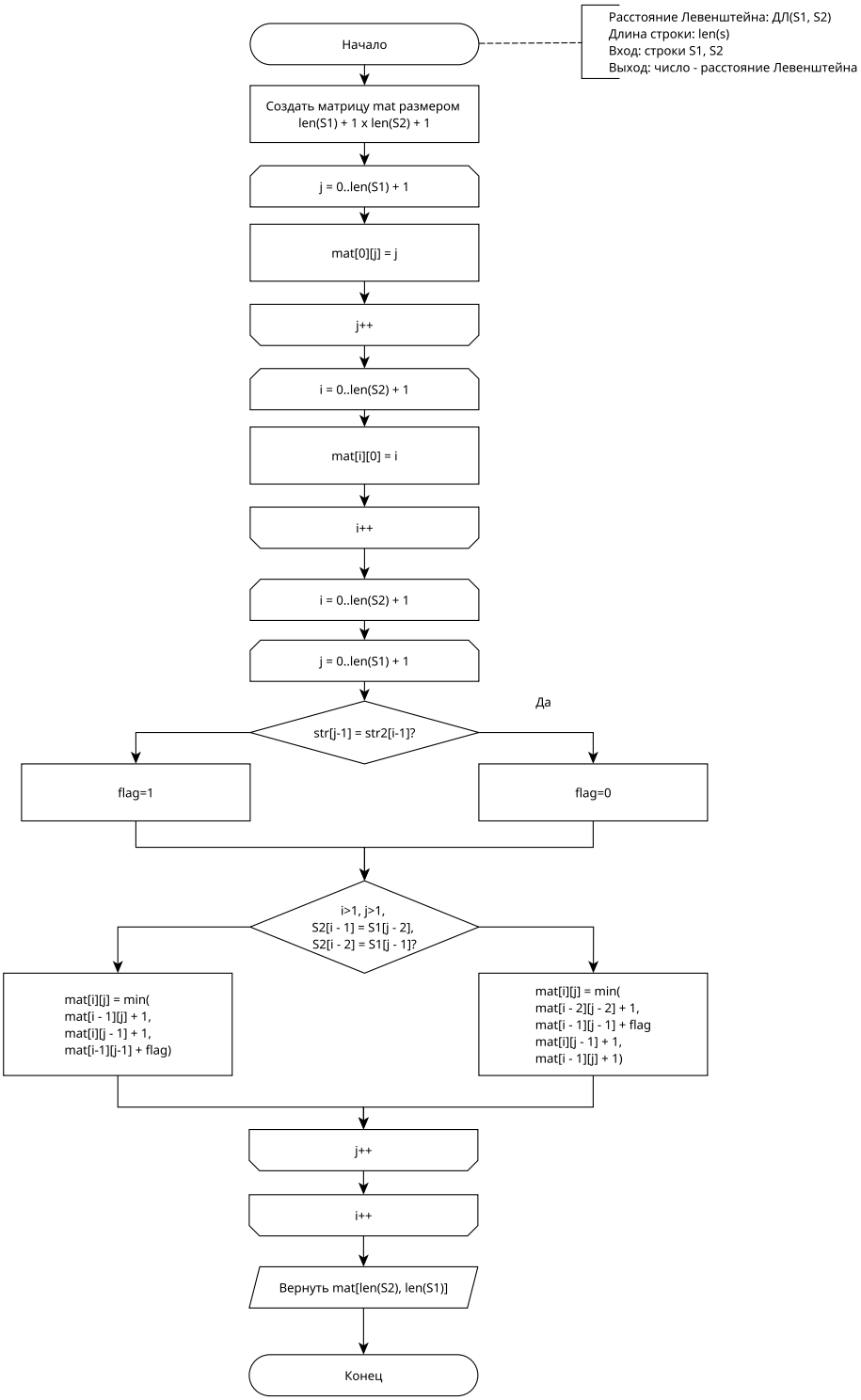

Рисунок 2.3 – Схема алгоритма нахождения расстояния Дамерау-Левенштейна с использованием динамического программирования

Вывод

В разделе были представлены схемы реализованных алгоритмов для нахождения расстояний Левенштейна и Дамерау-Левенштейна.

3 Технологическая часть

В разделе будут приведены требования к программному обеспечению, средства реализации, листиги кода.

3.1 Требования к программному обеспечению

Входные данные: 2 символьных строки произвольной длины.

Выходные данные: искомое расстояние для выбранного метода и матрица расстояний для методов с использованием динаического программирования.

3.2 Средства реализации

В работе был выбран язык программирования Python[2]. Т.к. в нем присутствуют функции для вычисления процессорного времени в библиотеке time[3]. Время замерялось с помощью функции process_time().

3.3 Реализация алгоритмов

На листингах 3.1 - 3.4 представлены реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна.

Листинг 3.1 – Вспомогательные функции

```
1 def head(str: str) -> str:
2     return str[0]
3
4 def tail(str: str) -> str:
5     return str[1:]
```

Листинг 3.2 – Рекурсивный алгоритм нахождения расстояния Левенштейна

```
1  def levenshtein_distance_recursion(str1: str, str2: str) -> int:
2      if len(str1) == 0:
3          return len(str2)
4      elif len(str2) == 0:
5          return len(str1)
6      elif (head(str1) == head(str2)):
7          return levenshtein_distance_recursion(tail(str1),
               tail(str2))
8      return 1 + min(
9          levenshtein_distance_recursion(tail(str1), str2),
10         levenshtein_distance_recursion(str1, tail(str2)),
11         levenshtein_distance_recursion(tail(str1), tail(str2)))
```

Листинг 3.3 – Алгоритм нахождения расстояния Левенштейна с использованием динамического программирования

```
def levenshtein_distance_mem(str1: str, str2: str) -> int:
    mat = [[0] * (len(str2) + 1) for i in range(len(str1) + 1)]

    for j in range(len(str1) + 1):
        mat[0][j] = j

    for i in range(len(str2) + 1):
        mat[i][0] = i

    for i in range(1, len(str2) + 1):
        for j in range(1, len(str1) + 1):
            mat[i][j] = min(
                    mat[i - 1][j] + 1,
                    mat[i][j - 1] + 1,
                    mat[i - 1][j - 1] + (1 if str1[j - 1] != str2[i
                       - 1] else 0))
    return mat[-1][-1]
```

Листинг 3.4 – Алгоритм нахождения расстояния Дамерау-Левенштейна с использованием динамического программирования

```
def damerau_levenshtein(str1: str, str2: str) -> int:
    mat = [[0] * (len(str1) + 1) for i in range(len(str2) + 1)]

    for i in range(len(str1) + 1):
        mat[0][i] = i

    for i in range(len(str2) + 1):
        mat[i][0] = i

    for i in range(1, len(str2) + 1):
        for j in range(1, len(str1) + 1):
            if (i > 1 and j > 1 and str2[i - 1] == str1[j - 2] and
               str2[i - 2] == str1[j - 1]):
                mat[i][j] = min(
                    mat[i - 2][j - 2] + 1,
                    mat[i - 1][j - 1] + (1 if str1[j - 1] != str2[i
                       - 1] else 0),
                    mat[i][j - 1] + 1,
                    mat[i - 1][j] + 1
                )
            else:
                mat[i][j] = min(
                    mat[i - 1][j] + 1,
                    mat[i][j - 1] + 1,
                    mat[i - 1][j - 1] + (1 if str1[j - 1] != str2[i
                       - 1] else 0)
                )
    return mat[-1][-1]
```

Вывод

В разделе были рассмотерны требования для программного обеспечения, используемые средства реализации, приведены листинги кода для вычисления расстояний Левенштейна (рекурсивный алгоритм и динамический алгоритм), Дамерау-Левенштейна(динамический алгоритм), и их вспомогательных функций.

4 Исследовательская часть

4.1 Технические характеристики

Характеристики используемого оборудования:

- Операционная система - Ubuntu 24.04.1 LTS [4] Linux X86_64 [5];
- Оперативная память - 16 ГБ;
- Процессор - Intel® Core™ i5-13420H 2.1GHz [6].

4.2 Описание используемых типов данных

Используемые типы данных

- строка - последовательность символов типа str;
- длина строки - целое число типа int;
- матрица - двумерный массив типа int.

4.3 Оценка памяти

Рекурсивный алгоритм нахождения расстояния Левенштейна не использует структур для хранения промежуточных результатов, но каждый вызов функции обрабатывает только фрагмент строк, после чего происходит повторный вызов функции. В худшем случае глубина рекурсии будет равна:

$$len(S_1) + len(S_2) \tag{4.1}$$

При этом на каждом вызове используются локальные переменные: 2 типа str. В таком случае максимальное количество используемой паямти будет равно:

$$(len(S_1) + len(S_2)) \cdot (2 \cdot sizeof(str)) \tag{4.2}$$

где $sizeof$ - функция вычисления размера параметра.

Алгоритм нахождения расстояния Левенштейна с использованием дианмического программирования задействует в себе матрицу размером $len(S_1)+1 \times len(S_2)+1$ типа int. В матрице хранятся промежуточные результаты вычислений (Расстояние Левенштейна для каждой подстроки). Кроме матрицы в алгоритме используются 2 переменные типа int и 2 перемнные типа str. Тогда количество используемой памяти будет равно:

$$(len(S_1) + len(S_2)) \cdot sizeof(int) + 2 \cdot sizeof(int) + 2 \cdot sizeof(str). \quad (4.3)$$

Динамический алгоритм нахождения расстояния Дамерау-Левенштейна аналогичен динамическому для нахождения расстояния Левенштейна, иных переменных в нум не задействуется, поэтому объем используемый паямти так же равен:

$$(len(S_1) + len(S_2)) \cdot sizeof(int) + 2 \cdot sizeof(int) + 2 \cdot sizeof(str). \quad (4.4)$$

4.4 Время выполнения алгоритмов

Результаты замеров времени работы алгоритмов приведены в таблице 4.1. Замеры проводились на строках одинаковой длины и усреднялись для каждого набора входных данных одной длины. Каждое значение получено с помощью взятия среднего из 100 измерений.

Таблица 4.1 – Время работы алгоритмов (в микросекундах)

Длина строк(символов)	Лев.(рек.)	Лев.(мат.)	Дам.-Лев.(мат.)
1	0.61	0.93	0.90
2	2.32	1.55	1.62
3	10.05	2.50	2.65
4	48.82	3.60	4.05
5	229.41	5.12	5.72
6	1125.48	6.78	7.82
7	6229.95	8.96	10.54
8	30163.73	11.10	12.95

На рисунках 4.1 - 4.2 представлены графики зависимости времни выполнения алгоритмов от длины входных слов.

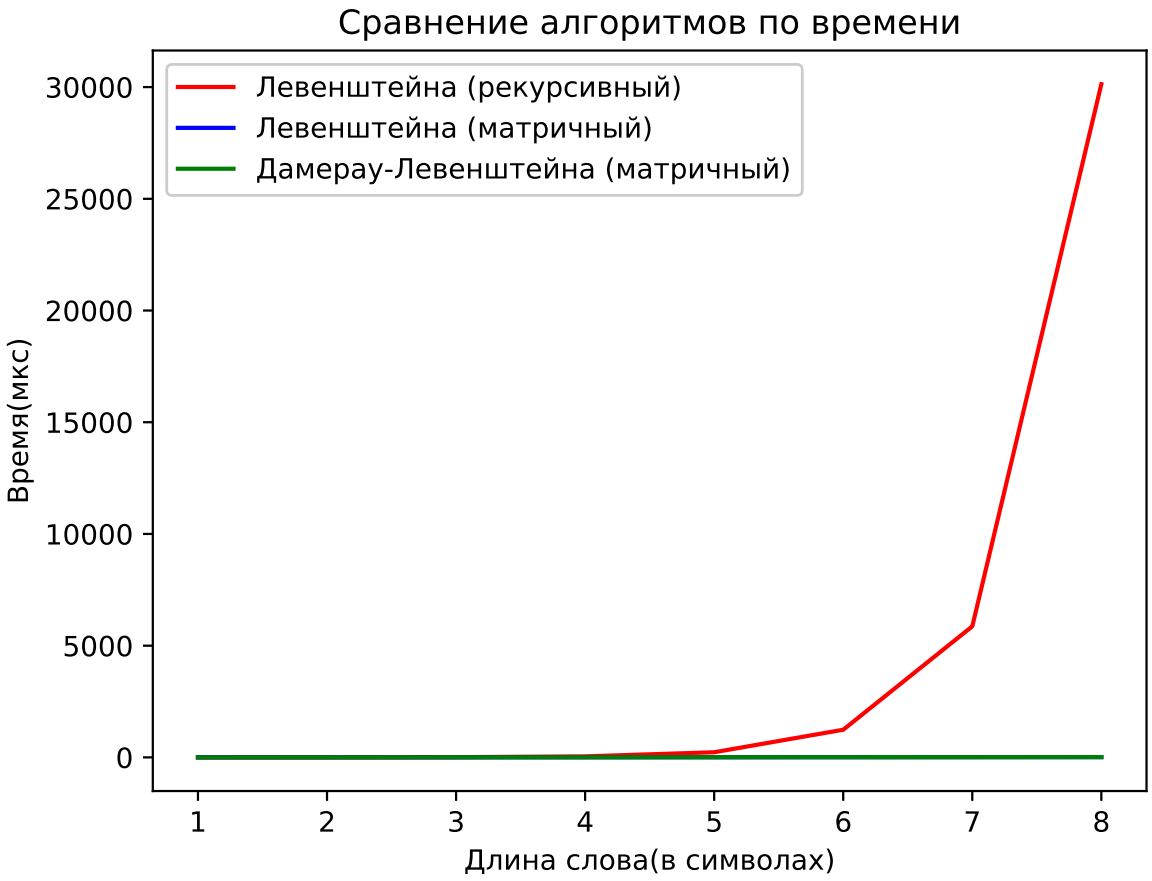

Рисунок 4.1 – Сравнение всех алгоритмов по времени

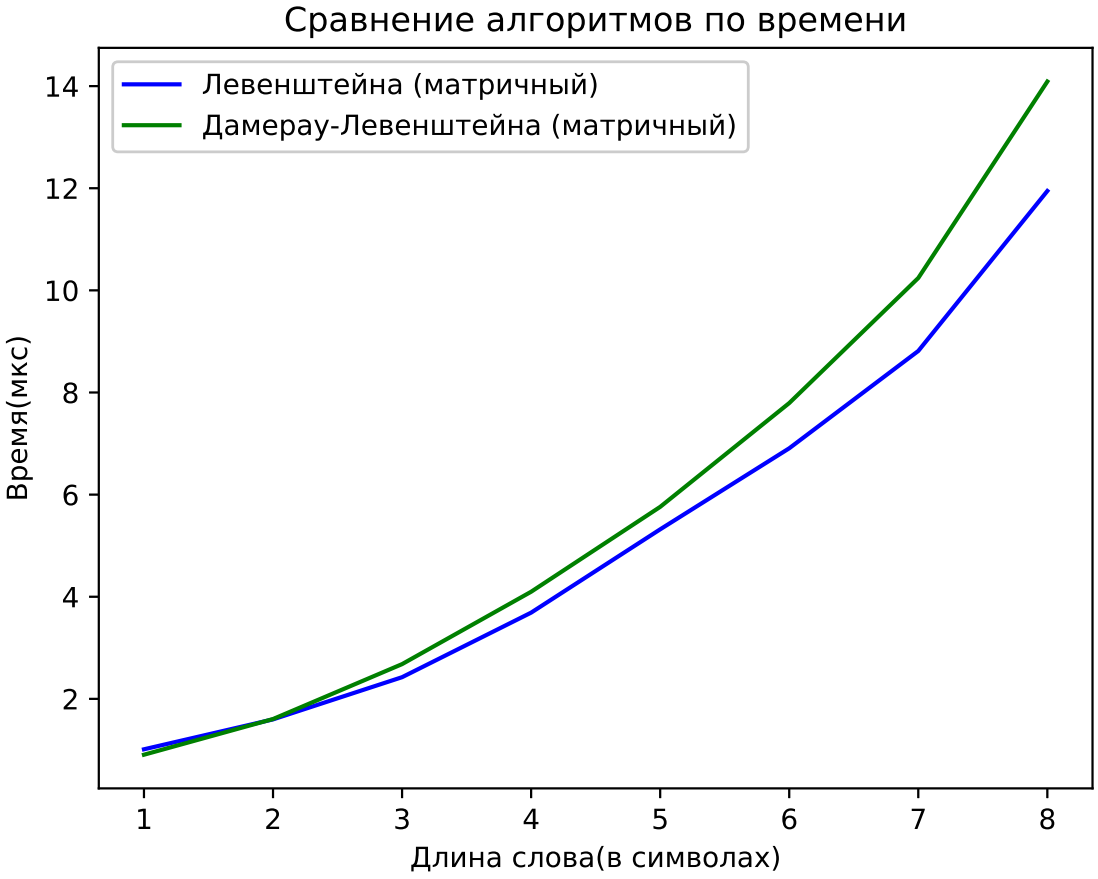

Рисунок 4.2 – Сравнение динамических алгоритмов по времени

Наиболее эффективными являются динамические алгоритмы (использующие матрицу), так как в рекурсивном алгоритме большое количество повторных расчетов.

Вывод

Рекурсивный алгоритм работает на несколько порядков медленне, чем алгоритм, использующий динамический подход. Динамические алгоритм вычисления расстояния Левенштейна и Дамерау-Левенштейна мало отличаются по времени работы. Оценка используемой памяти показала, что рекурсивный алгоритм требует меньше памяти, нежели алгоритмы с использованием матриц. Расход памяти у динамического алгоритма вычисления расстояния Дамерау-Левенштейна равен расходу памяти у динамического алгоритма вычисления расстояния Левенштейна несмотря на добавление новой операции.

Заключение

Эксперементально подтверждено различие временной эффективности рекурсивной и динамической реализации выбранных алгоритмов нахождения расстояния между строками при помощи разработанного программного обеспечения на данных, полученных замерами процессорного времени выполнения алгоритмов.

На полученных результатах можно сделать вывод, что матричная реализация значительно выигрывает в скорости при увеличении длины строк, но проигрывает по количеству используемой памяти.

В ходе выполнения лабораторной работы были решены задачи:

- реализованы алгоритмы нахождения расстояний Левенштейна (матричный и рекурсивный) и Дамерау-Левенштейна (матричный);
- реализации проанализированы по затрачиваемым ресурсам (памяти и процессорного времени выполнения);
- полученные результаты описаны и обоснованы.

Список литературы

[1] Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады АН СССР, 1965. Т. 163. С. 845– 848.

[2] Welcome to Python [Электронный ресурс]. Режим доступа: `https://docs.python.org` (дата обращения: 15.09.2024).

[3] time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 15.09.2024).

[4] Ubuntu 24.04.1 (Noble Numbat) [Электронный ресурс]. Режим доступа: `https://releases.ubuntu.com/24.04/` (дата обращения: 15.09.2024).

[5] Linux [Электронный ресурс]. Режим доступа: `https://www.linux.org` (дата обращения: 15.09.2024).

[6] Core i5-13420H: технические характеристики и тесты [Электронный ресурс]. Режим доступа: `https://technical.city/ru/cpu/Core-i5-13420H` (дата обращения: 15.09.2024).