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»

Лабораторная работа №1
по дисциплине "Анализ Алгоритмов"

Тема Расстояние Левенштейна и Дамерау-Левенштейна

Студент Ковалец К. Э.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

В данной рабораторной работе будет рассмотрено расстояние Левенштейна. Данное расстояние показывает минимальное количество операций (вставки, удаления, замены), которое необходимо для перевода одной строки в другую. Это расстояние помогает определить схожесть двух строк.

Расстояние Левенштейна применяется в теории информации и компьютерной лингвистике для:

- исправления ошибок в слове;
- сравнения текстовых файлов утилитой diff;
- для сравнения генов, хромосом и белков в биоинформатике.

Целью данной лабораторной работы является изучение и применение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна, а также получение практических навыков реализации указанных алгоритмов. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить расстояния Левенштейна и Дамерау-Левенштейна;
- реализовать указвнные алгоритмы поиска расстояний (два алгоритма в матричной версии и один из алгоритмов в рекурсивной версии)
- примененить метод динамического программирования для матричной реализации указанных алгоритмов;
- провести сравнительный анализ линейной и рекурсивной реализаций алгоритмов по затраченному процессорному времени и памяти на основе экспериментальных данных;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы нахождения расстояний Левенштейна и Дамерау-Левенштейна.

1.1 Расстояние Левенштейна

Расстояние Левенштейна [1] – это минимальное количество операций вставки, удаления и замены, необходимох для превращения одной строки в другую.

Цены операций могут зависеть от вида операций (вставка, удаление, замена) и/или от участвующих в ней символов, отражая разную вероятность разных ошибок при вводе текста и т.п. В общем случае

- $w(a, b)$ – цена замены символа a на b, R (от англ. replace);
- $w(\lambda, b)$ – цена вставки символа b, I (от англ. insert);
- $w(a, \lambda)$ – цена удаления символа a, D (от англ. delete).

Для решения задачи о редакционном расстоянии необходимо найти последовательность замен, минимизирующую суммарную цену. Расстояние Левенштейна является частным случаем этой задачи при

- $w(a, a) = 0$;
- $w(a, b) = 1$, $a \neq b$;
- $w(\lambda, b) = 1$;
- $w(a, \lambda) = 1$.

Имеем две строки S_1 и S_2, длинной M и N соответственно. Расстояние Левенштейна рассчитывается по рекуррентной формуле формуле 1.1:

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

где функция 1.2 определена как:

$$m(a,b) = \begin{cases} 0 & \text{если a = b,} \\ 1 & \text{иначе} \end{cases}. \tag{1.2}$$

Рекурсивный алгоритм реализует формулу 1.1. Функция D составлена таким образом, что для перевода из строки a в строку b требуется выпол- нить последовательно некоторое количество операций (удаление, вставка, замена) в некоторой последовательности. Полагая, что a', b' — строки a и b без последнего символа соответственно, цена преобразования из строки a в строку b может быть выражена как:

- сумма цены преобразования строки a' в b' и цены проведения операции удаления, которая необходима для преобразования a' в a;
- сумма цены преобразования строки a в b' и цены проведения операции вставки, которая необходима для преобразования b' в b;
- сумма цены преобразования из a' в b' и операции замены, предполагая, что a и b оканчиваются на разные символы;
- цена преобразования из a' в b', предполагая, что a и b оканчиваются на один и тот же символ.

Минимальной ценой преобразования будет минимальное значение приведенных вариантов.

1.2 Расстояние Дамерау-Левенштейна

В алгоритме поиска расстояния Дамерау-Левенштейна, помимо вставки, удаления, и замены присутствует еще одна редакторская операция - транспозиция T (от англ. transposition).

Расстояние Дамерау-Левенштейна может быть вычисленно по рекуррентной формуле 1.3:

$$D(i,j) = \begin{cases} 0 & ,j=0, i=0 \\ i & ,j=0, i>0 \\ j & ,j>0, i=0 \\ min \begin{cases} D(i,j-1)+1 \\ D(i-1,j)+1 \\ D(i-1,j-1)+m(S_1[i],S_2[i]) \\ D(i-2,j-2)+m(S_1[i],S_2[i]) \end{cases} & \begin{array}{l} \text{, если i} > 1\text{, j} > 1 \\ ,S_1[i] = S_2[j-1] \\ ,S_1[j] = S_2[i-1] \end{array} \\ min \begin{cases} D(i,j-1)+1 \\ D(i-1,j)+1 \\ D(i-1,j-1)+m(S_1[i],S_2[i]) \end{cases} & \text{,иначе} \end{cases} \tag{1.3}$$

1.3 Вывод

В данном разделе были рассмотрены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна, формулы которых задаются реккурентно, а следовательно, данные алгоритмы могут быть реализованы рекурсивно и итерационно. На вход алгоритмам будут поступать две строки, которые могут содержать как русские, так и английские буквы, также будет предусмотрен ввод пустых строк. Реализуемое ПО будет давать возможность выбрать алгоритм и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна, приведено описание используемых типов данных, оценки памяти, а также описана структура ПО.

2.1 Схемы алгоритмов

На вход алгоритмов падаются строки s1 и s2, на выходе получаем единственное число - искомое расстояние.

На рис. 2.1 - 2.4 приведены схемы рекурсивных и матричных алгоритмов Левенштейна и Дамерау-Левенштейна.

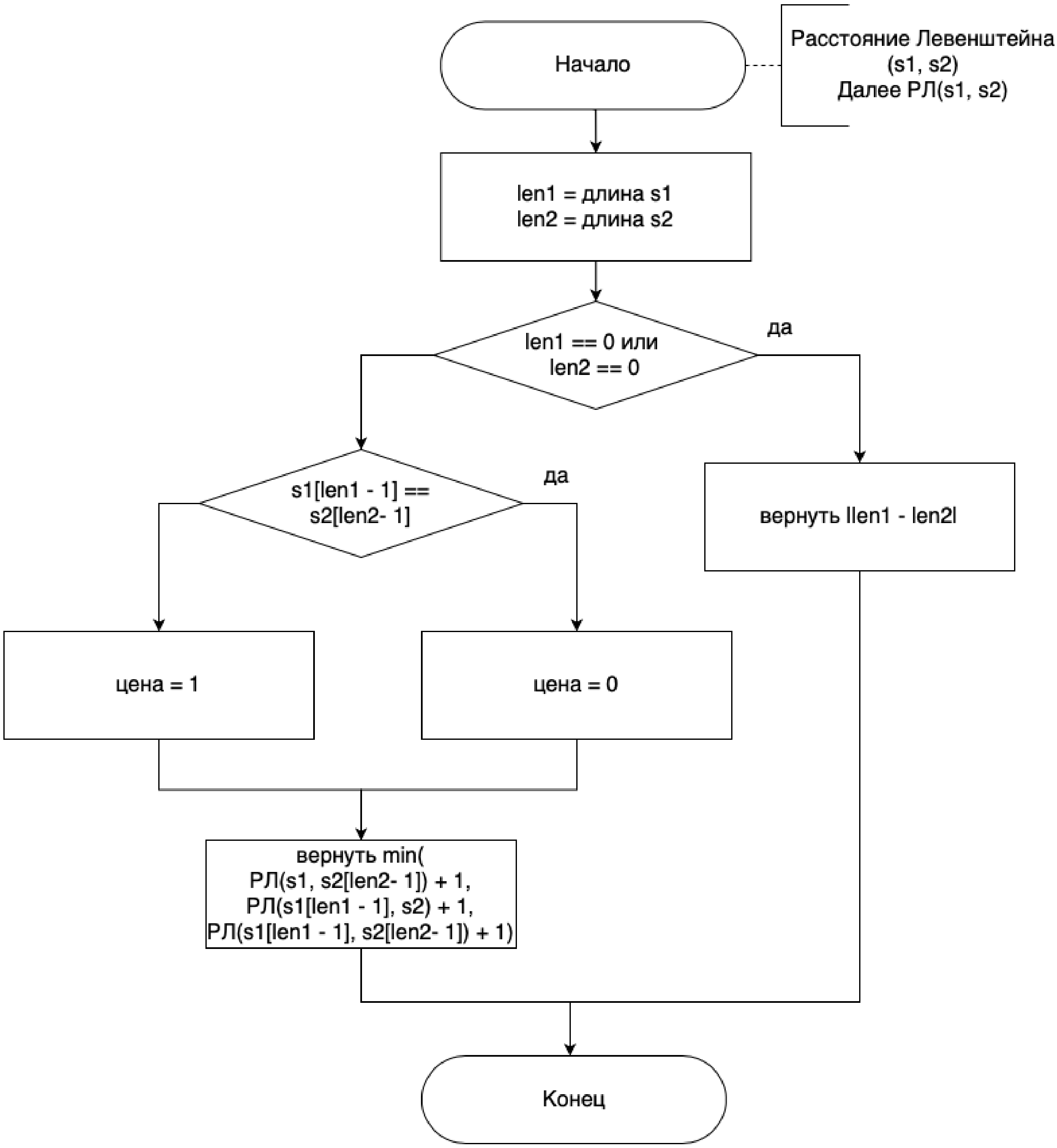

Рисунок 2.1 – Схема рекурсивного алгоритма нахождения расстояния Левенштейна

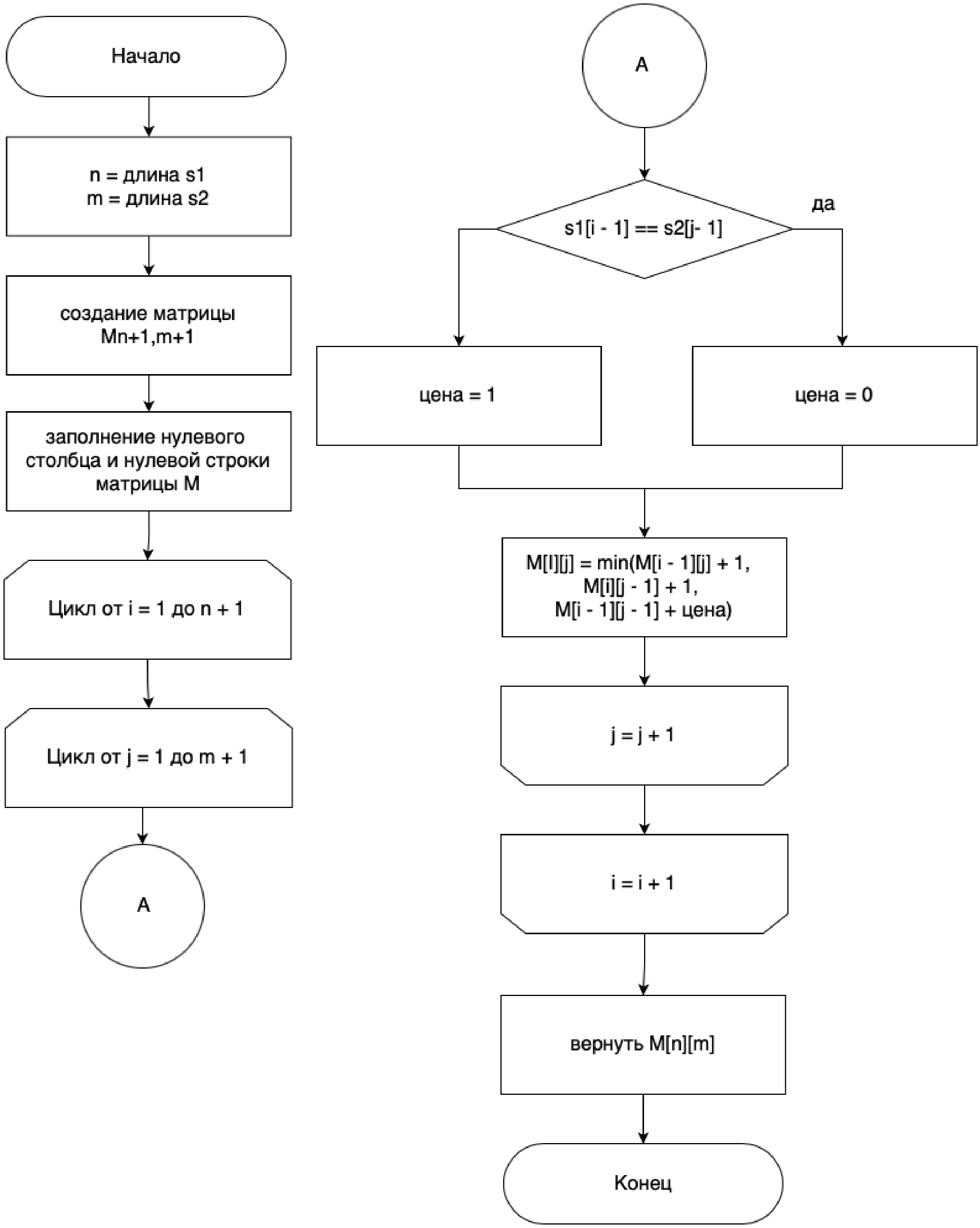

Рисунок 2.2 – Схема матричного алгоритма нахождения расстояния Левенштейна

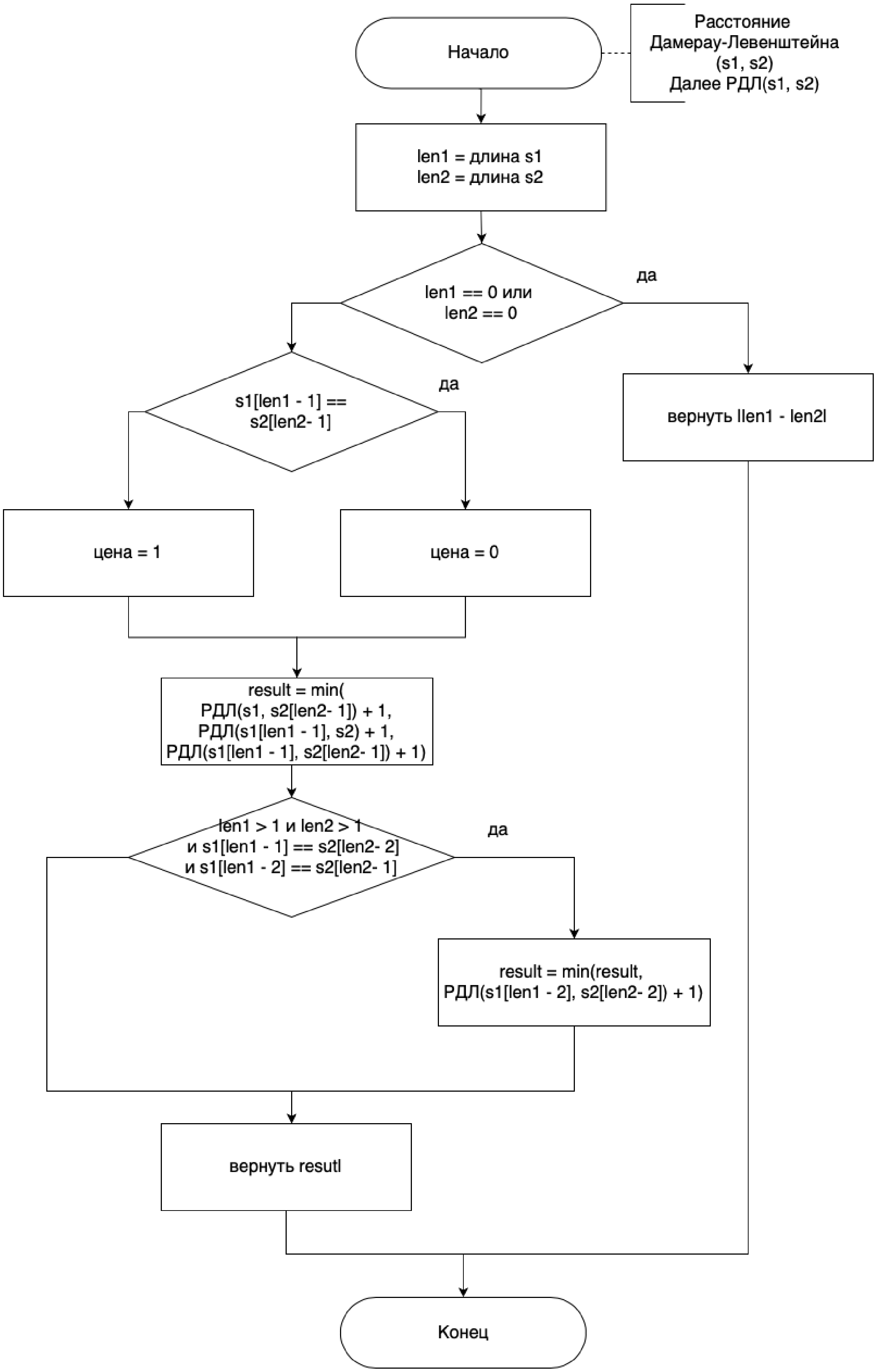

Рисунок 2.3 – Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна

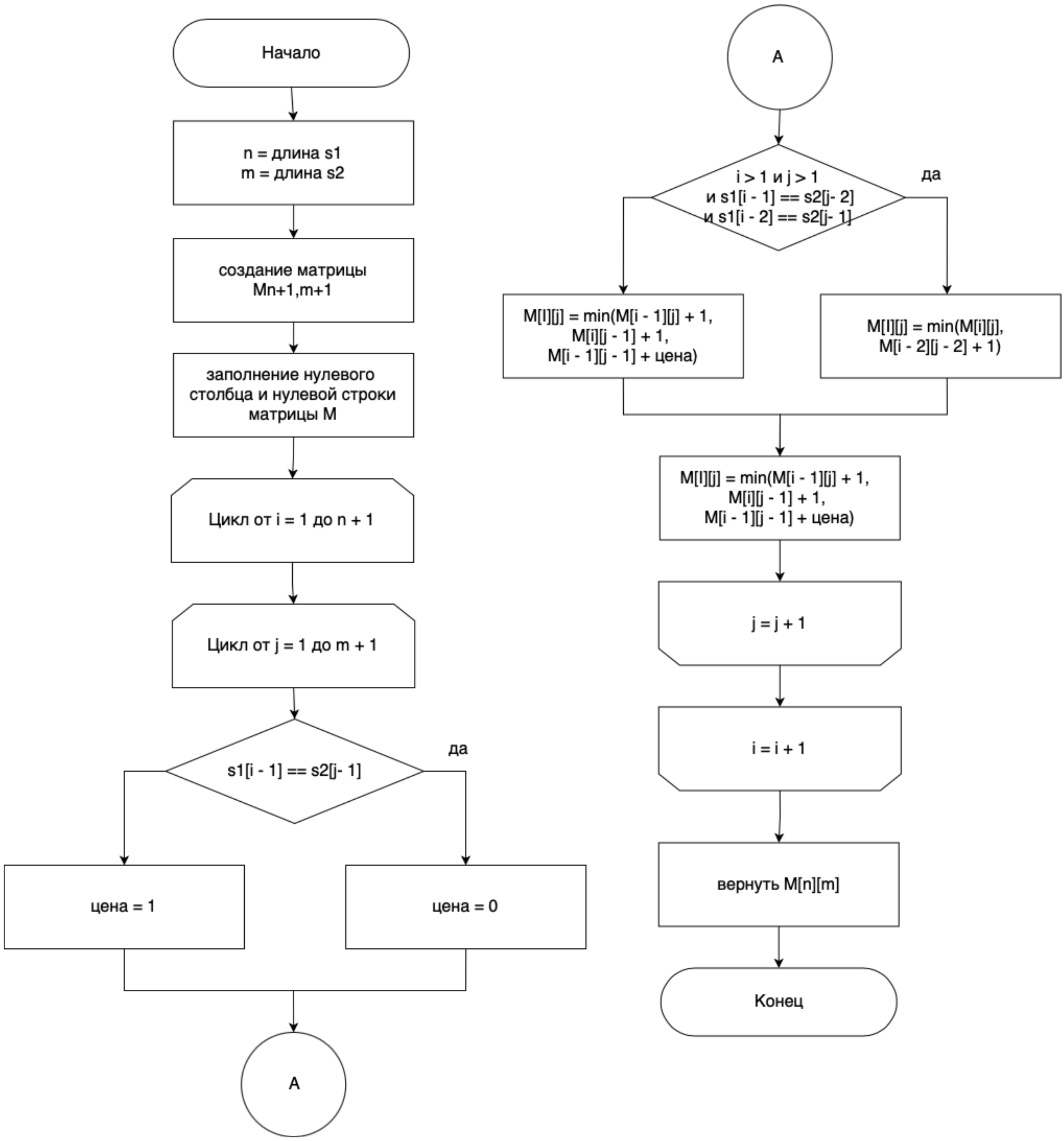

Рисунок 2.4 – Схема матричного алгоритма нахождения расстояния Дамерау-Левенштейна

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- строка - массив типа $char$ размером длины строки;
- длина строки - целое число типа int;
- матрица - двумерный массив типа int.

2.3 Оценка памяти

Алгоритмы Левенштейна и Дамерау-Левенштейна не отличаются по использованию памяти, поэтому достаточно рассмотреть рекурсивную и матричную реализации одного из этих алгоритмов.

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк, а на каждый вызов функции требуется еще 3 дополнительных переменных типа int, соответственно, максимальный расход памяти

$$(Len(S_1) + Len(S_2)) \cdot (2 \cdot Size(\text{string}) + 3 \cdot Size(\text{int})), \tag{2.1}$$

где S_1, S_2 - строки, Size - функция, возвращающая размер аргумента; Len - функция, возвращающая длину строки, string - строковый тип, int - целочисленный.

Использование памяти при итеративной реализации теоритически равно

$$(Len(S_1)+1) \cdot (Len(S_2)+1) \cdot Size(int) + 3 \cdot Size(int) + 2 \cdot Size(string) \tag{2.2}$$

По расходу памяти итеративные алгоритмы проигрывают рекурсивным: максимальный размер используемой памяти в них растёт как произведение длин строк, в то время как у рекурсивного алгоритма — как сумма длин строк.

2.4 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

- ввод пустых строк;
- одна из строк - пустая;
- расстояния Левенштейна и Дамерау–Левенштейна равны;
- расстояния Левенштейна и Дамерау–Левенштейна различны;

2.5 Структура ПО

ПО будет состоять из следующих модулей:

- *main.py* - файл, содержащий функцию *main*;
- *algorithms.py* - файл, содержащий код всех алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна;
- *compare_time.py* - файл, в котором содержатся функции для замера времени работы алгоритмов;
- *in_out.py* - файл, в котором содержатся функции ввода-вывода;
- *color.py* - файл, который содержит переменные типа *string* для цветного вывода результата работы программы в консоль.

2.6 Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов, выбраны используемые типы данных, а также была проведена оценка затрачиваемого объёма памяти и описана структура ПО.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги кода, а также функциональные тесты.

3.1 Требования к программному обеспечению

- входные данные - две строки на русском или английском языке в любом регистре;
- выходные данные - искомое расстояние для выбранного метода и матрицы расстояний для матричных реализаций.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [2]. Выбор обсуловлен наличием опыта работы с ним. Время работы было замерено с помощью функции *process_time* из библиотеки *time* [3].

3.3 Листинги кода

В листингах 3.1 - 3.4 представлены реализации алгоритмов нахождения расстояния Левенштейна и Дамерау–Левенштейна.

Листинг 3.1 – Функция нахождения расстояния Левенштейна рекурсивно

```
def lev_recursion(s1, s2, output = False):
    len1 = len(s1)
    len2 = len(s2)

    if len1 == 0 or len2 == 0:
        return abs(len1 - len2)

    m = 0 if s1[-1] == s2[-1] else 1

    return min(lev_recursion(s1,      s2[:-1]) + 1,
               lev_recursion(s1[:-1], s2     ) + 1,
               lev_recursion(s1[:-1], s2[:-1]) + m)
```

Листинг 3.2 – Функция нахождения расстояния Левенштейна итеративно

```
def lev_table(s1, s2, output = False):
    len1 = len(s1)
    len2 = len(s2)

    M = [[i + j for j in range(len2 + 1)]
                for i in range(len1 + 1)]

    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):

            m = 0 if s1[i - 1] == s2[j - 1] else 1

            M[i][j] = min(M[i - 1][j    ] + 1,
                          M[i    ][j - 1] + 1,
                          M[i - 1][j - 1] + m)
    if output:
        output_table(M, s1, s2)

    return M[-1][-1]
```

Листинг 3.3 – Функция нахождения расстояния Дамерау–Левенштейна рекурсивно

```
def dam_lev_recursion(s1, s2, output = False):
    len1 = len(s1)
    len2 = len(s2)

    if len1 == 0 or len2 == 0:
        return abs(len1 - len2)

    m = 0 if s1[-1] == s2[-1] else 1

    result = min(dam_lev_recursion(s1,      s2[:-1]) + 1,
                 dam_lev_recursion(s1[:-1], s2     ) + 1,
                 dam_lev_recursion(s1[:-1], s2[:-1]) + m)

    if len1 > 1 and len2 > 1 and s1[-1] == s2[-2] \
                             and s1[-2] == s2[-1]:
        result = min(result, dam_lev_recursion(s1[:-2], s2[:-2]) +
            1)

    return result
```

Листинг 3.4 – Функция нахождения расстояния Дамерау–Левенштейна итеративно

```
def dam_lev_table(s1, s2, output = False):
    len1 = len(s1)
    len2 = len(s2)

   M = [[i + j for j in range(len2 + 1)]
              for i in range(len1 + 1)]

    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):

            m = 0 if s1[i - 1] == s2[j - 1] else 1

            M[i][j] = min(M[i - 1][j     ] + 1,
                          M[i     ][j - 1] + 1,
                          M[i - 1][j - 1] + m)

            if i > 1 and j > 1 and s1[i - 1] == s2[j -2 ] \
                               and s1[i - 2] == s2[j - 1]:
                M[i][j] = min(M[i][j], M[i - 2][j - 2] + 1)

    if output:
        output_table(M, s1, s2)

    return M[-1][-1]
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов вычисления расстояния Левенштейна и Дамерау—Левенштейна. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

1-ая строка	2-ая строка	Ожидаемый результат	
		Левенштейн	Дамерау-Левенштейн
[]	[]	0	0
[]	цветы	5	5
бегемот	бегемот	0	0
скат	кот	2	2
красивый	карсивый	2	1
вагон	гонки	4	4
бар	раб	2	2
слон	слоны	1	1

3.5 Вывод

В данном разделе были разработаны алгоритмы поиска расстояний Левенштейна и Дамерау–Левенштейна (рекурсивные и с заполнением матрицы), а также проведено тестирование.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: macOS 11.5.2. [4]
- Память: 8 GiB.
- Процессор: 2,3 GHz 4-ядерный процессор Intel Core i5. [5]

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

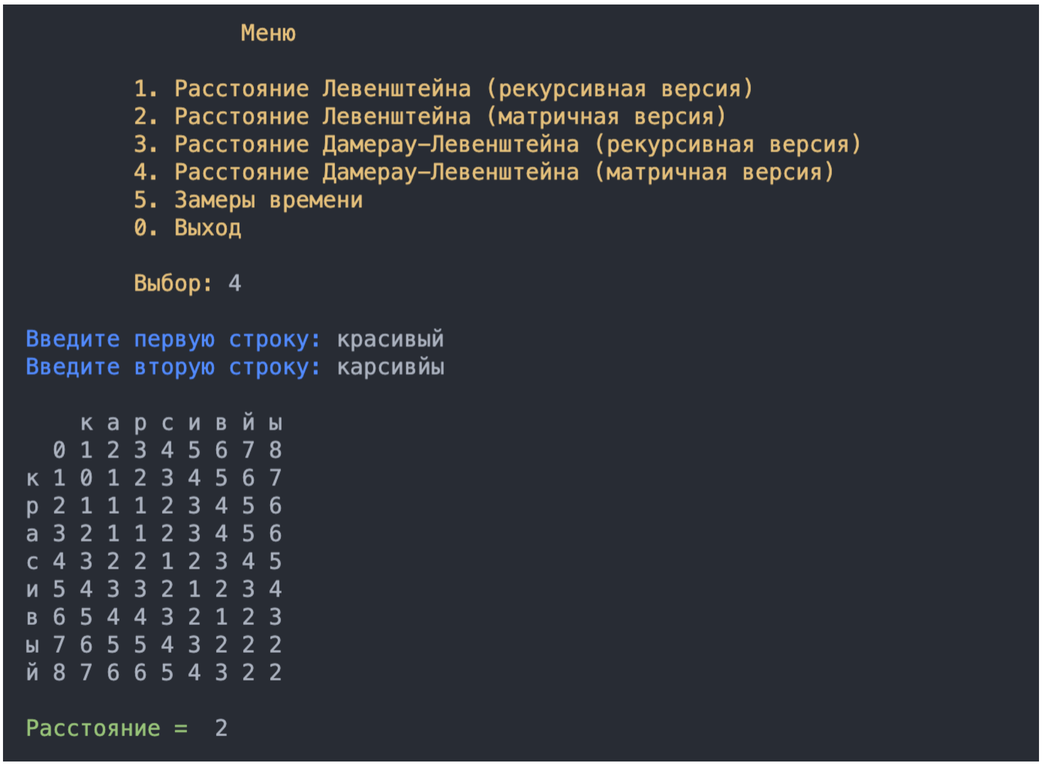

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Результаты замеров времени работы алгоритмов нахождения расстояний Левенштейна и Дамерау–Левенштейна приведены в таблице 4.1. Замеры времени проводились на строках одинаковой длины и усреднялись для каждого набора одинаковых экспериментов.

Таблица 4.1 – Время работы алгоритмов (в секундах)

Длина строк	Лев рек.	Дам-Лев рек.	Лев итер.	Дам-Лев итер.
1	5.64e-06	4.93e-06	2.60e-06	2.20e-06
2	6.05e-06	7.61e-06	9.30e-06	9.30e-06
3	9.85e-06	1.02e-05	4.39e-05	4.51e-05
4	1.51e-05	1.61e-05	2.02e-04	2.07e-04
5	2.08e-05	2.34e-05	1.09e-03	1.09e-03
6	2.82e-05	3.22e-05	5.69e-03	5.83e-03
7	3.67e-05	4.28e-05	3.07e-02	3.22e-02
8	4.72e-05	5.48e-05	1.68e-01	1.73e-01

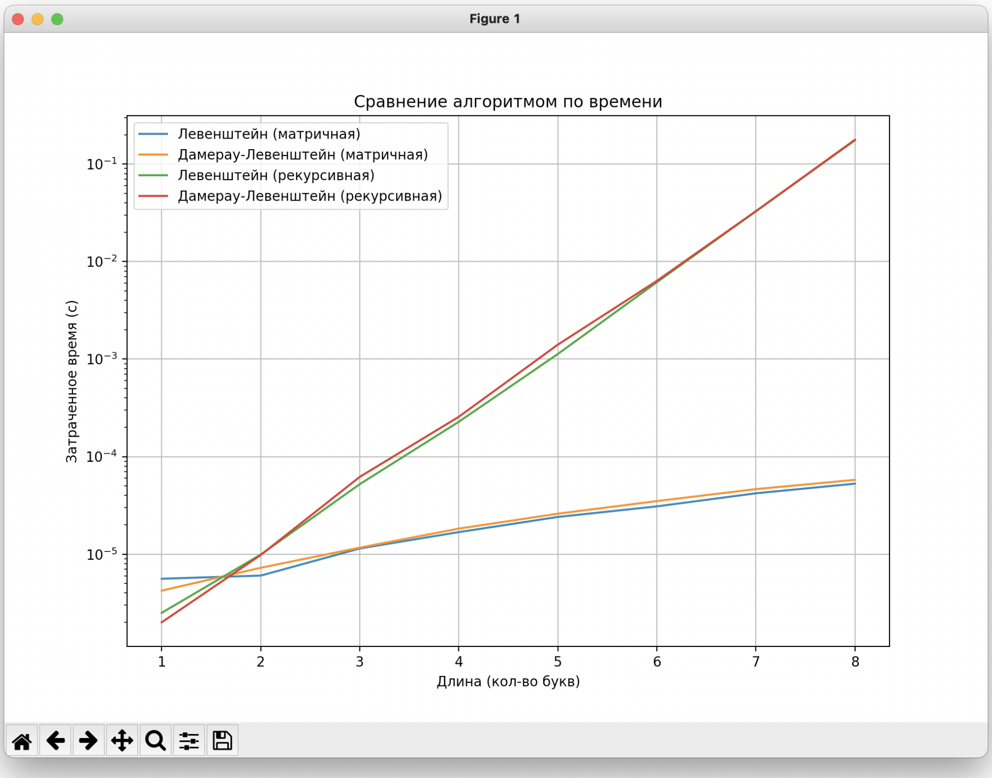

Рисунок 4.2 – Сравнение алгоритмов по времени

Наиболее эффективными являются алгоритмы, использующие матрицы, так как в рекурсивных алгоритмах большое количество повторных расчетов.

4.4 Вывод

Рекурсивные алгоритмы Левенштейна и Дамерау–Левенштейна работают на несколько порядков дольше реализаций, использующих матрицы (в 10 раз при длине строки - 4, в 100 раз при длине - 6 и в 1000 раз при длине строки - 8), время их работы увеличивается в геометрической прогрессии. Также стоит заметить, что как рекурсивные, так и итеративные алгоритмы сопостовимы между собой по времени выполнения и примерно равны.

Заключение

Было экспериментально подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранных алгоритмов нахождения расстояния между строками при помощи разработаного программного обеспечения на материале замеров процессорного времени выполнения реализаций на различных длинах строк.

В результате исследований можно сделать вывод о том, что матричная реализация данных алгоритмов заметно выигрывает по времени при росте длин строк, но проигрывает по количеству затрачиваемой памяти.

В ходе выполнения данной лабораторной работы были решены следующие задачи:

- изучены алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
- применены методы динамического программирования для матричной реализации указанных алгоритмов;
- получены практические навыки реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
- проведен сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);
- экспериментально подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Литература

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

[2] Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 17.10.2021).

[3] time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 17.10.2021).

[4] macOS Monterey - Apple(RU) [Электронный ресурс]. Режим доступа: `https://www.apple.com/ru/macos/monterey/` (дата обращения: 17.10.2021).

[5] Intel [Электронный ресурс]. Режим доступа: `https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html` (дата обращения: 17.10.2021).