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»

Отчёт по лабораторной работе №1 по курсу «Анализ Алгоритмов»

Студент Паламарчук А.Н.

Группа ИУ7-53Б

Преподаватель Кормановский М.В.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является исследование алгоритмов поиска расстояния Левенштейна, поиска расстояния Дамерау – Левенштейна. Для достижения поставленной цели необходимо выполнить следующие задачи:

— разработать алгоритмы поиска расстояний Левенштейна и Дамерау – Левенштейна;

— разработать программное обеспечение, содержащее данные алгоритмы;

— провести функциональное тестирование реализованных алгоритмов;

— провести сравнительный анализ затрат времени и памяти при различных реализациях алгоритмов.

1 Аналитическая часть

В данной части работы будут представлены алгоритмы нахождения расстояний Левенштейна и Дамерау – Левенштейна.

1.1 Расстояние Левенштейна

Расстояние Левенштейна (англ. Levenshtein distance) (также редакционное расстояние или дистанция редактирования) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую [1].

λ — пустой символ, не входящий ни в одну из рассматриваемых строк.

Обозначения операций:

— $w(a, b)$ – цена замены символа a на символ b;

— $w(\lambda, b)$ – цена вставки символа b;

— $w(a, \lambda)$ – цена удаления символа a.

Каждая операция имеет свою цену:

— $w(a, a) = 0$;

— $w(a, b) = 1$ при $a \neq b$;

— $w(\lambda, b) = 1$;

— $w(a, \lambda) = 1$.

1.2 Расстояние Дамерау – Левенштейна

Расстояние Дамерау – Левенштейна (англ. Damerau – Levenshtein distance) между двумя строками, состоящими из конечного числа символов — это минимальное число операций вставки, удаления, замены одного символа и транспозиции двух соседних символов, необходимых для перевода одной строки в другую [2].

В алгоритм поиска расстояния Дамерау – Левенштейна добавляется ещё одна операция — транспозиция $w(S_1[i], S_2[j]) = 1$ при $S_1[i] = S_2[j+1]$ и $S_1[i+1] = S_2[j]$.

2 Конструкторская часть

2.1 Описание используемых структур данных

При реализации алгоритмов будут использованы следующие структуры данных:

— исходные строки S_1, S_2 — строковый тип;

— вычисленное расстояние — целочисленный тип.

2.2 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены алгоритмы матричного поиска расстояния Левенштейна, рекурсивного поиска расстояния Левенштейна и матричного поиска расстояния Дамерау – Левенштейна.

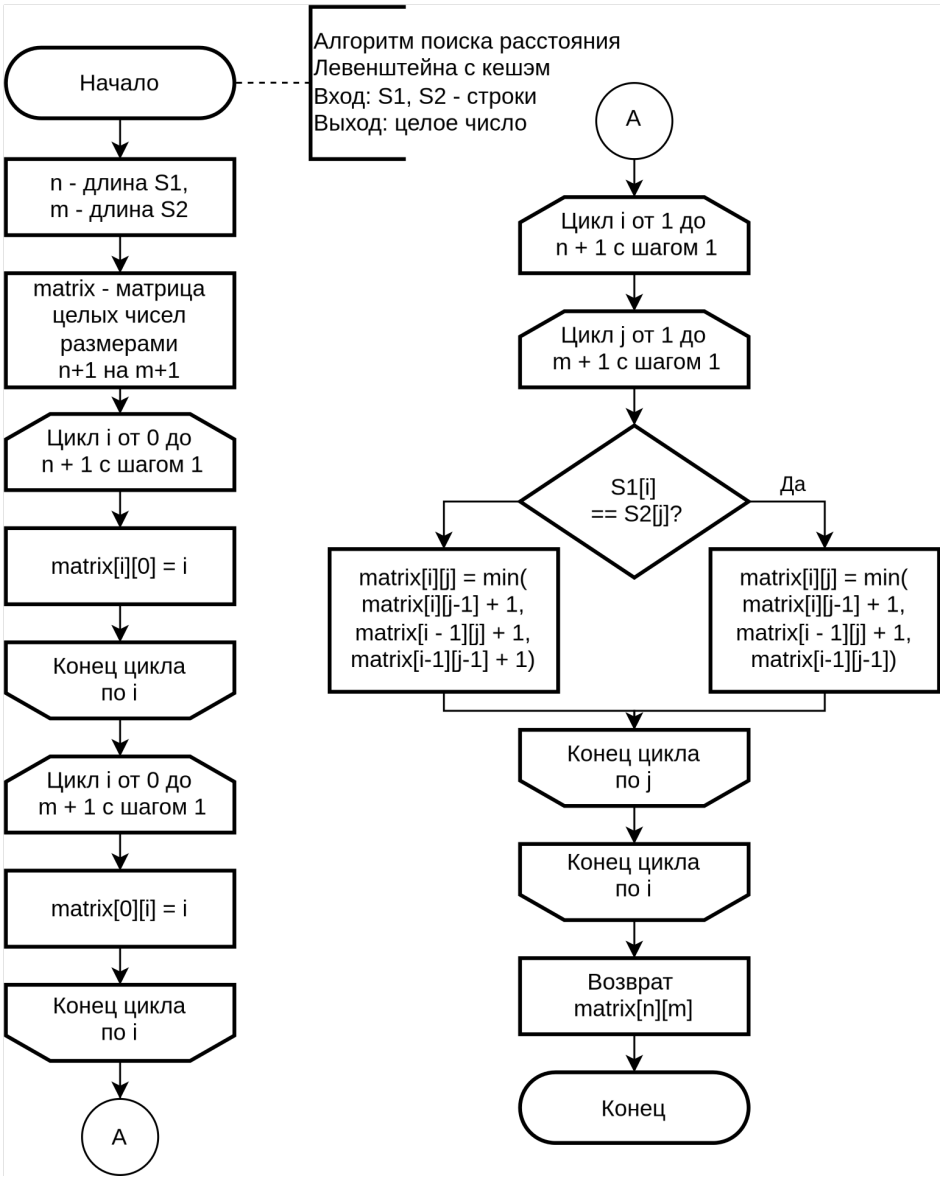

Рисунок 2.1 – Матричный алгоритм поиска расстояния Левенштейна

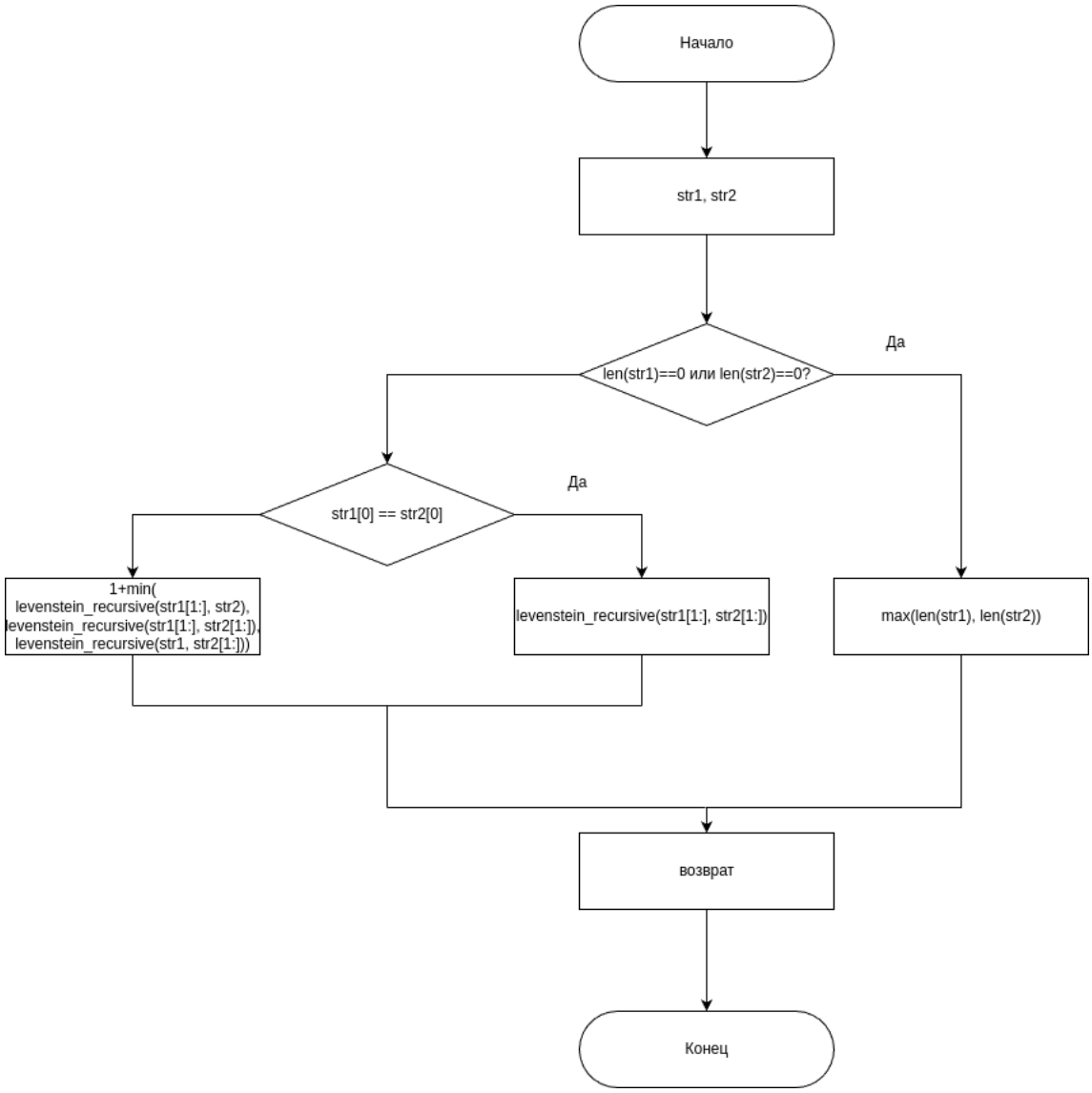

Рисунок 2.2 – Рекурсивный алгоритм поиска расстояния Левенштейна

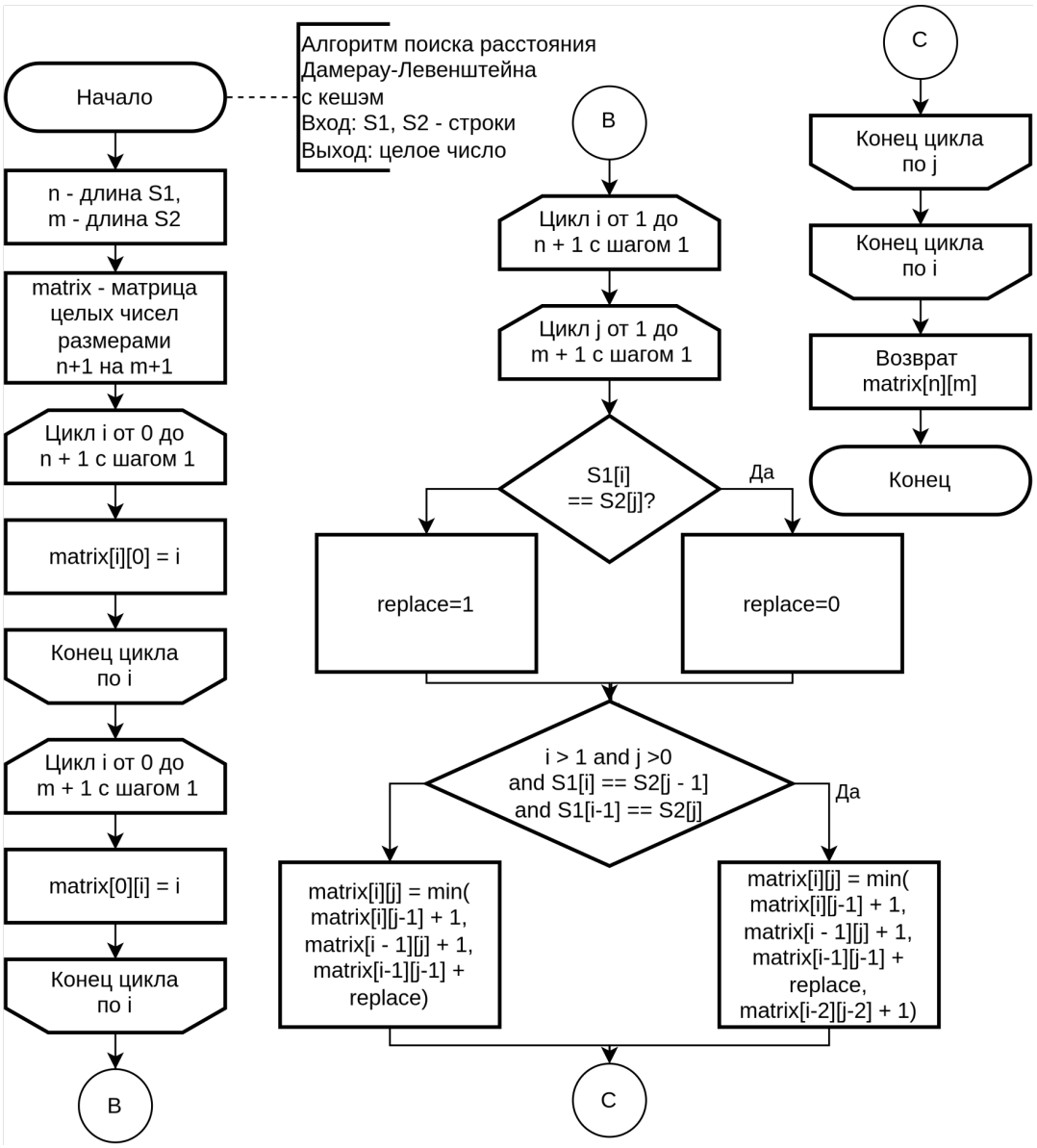

Рисунок 2.3 – Матричный алгоритм поиска расстояния Дамерау – Левенштейна

3 Технологическая часть

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*. Требуется измерить затрачиваемое время и объём необходимой памяти, для проведения замеров времени использовалась библиотека *time*, объёма затрачиваемаемой памяти использовалась библиотека *tracemalloc*. Для построения графиков использовалась библиотека *mathplotlib*.

3.2 Реализация алгоритмов

В листингах 3.1, 3.2 и 3.3 представлены алгоритмы матричного поиска расстояния Левенштейна, рекурсивного поиска расстояния Левенштейна и матричного поиска расстояния Дамерау – Левенштейна.

Листинг 3.1 – Матричный алгоритм поиска расстояния Левенштейна

```
1  def levenstein_matrix(str1: str, str2: str) -> int:
2      rows = len(str1) + 1
3      cols = len(str2) + 1
4      matrix = [[0] * cols for i in range(rows)]
5      for i in range(1, cols):
6          matrix[0][i] = i
7      for i in range(1, rows):
8          matrix[i][0] = i
9
10     for i in range(1, rows):
11         for j in range(1, cols):
12             if str1[i - 1] == str2[j - 1]: cost = 0
13             else: cost = 1
14             matrix[i][j] = min(
15                 matrix[i][j - 1] + 1,
16                 matrix[i - 1][j] + 1,
17                 matrix[i - 1][j - 1]  + cost
18             )
19
20     return matrix[-1][-1]
```

Листинг 3.2 – Рекурсивный алгоритм поиска расстояния Левенштейна

```
def levenstein_recursive(str1: str, str2: str) -> int:
    if (len(str1) == 0 or len(str2) == 0):
        return max(len(str1), len(str2))
    if str1[0] == str2[0]:
        return levenstein_recursive(str1[1:], str2[1:])
    return 1 + min(
        levenstein_recursive(str1[1:], str2),
        levenstein_recursive(str1[1:], str2[1:]),
        levenstein_recursive(str1, str2[1:])
    )
```

Листинг 3.3 – Матричный алгоритм поиска расстояния Дамерау – Левенштейна

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

    for i in range(1, rows):
        for j in range(1, cols):
            if i >= 2 and j >= 2 and str1[i - 1] == str2[j - 2] and
               str1[i - 2] == str2[j - 1]:
                matrix[i][j] = min(
                    matrix[i - 1][j] + 1,
                    matrix[i - 1][j - 1] + (0 if str1[i - 1] ==
                       str2[j - 1] else 1),
                    matrix[i - 2][j - 2] + 1,
                    matrix[i][j - 1] + 1,
                )
            else:
                matrix[i][j] = min(
                    matrix[i][j - 1] + 1,
                    matrix[i - 1][j] + 1,
                    matrix[i - 1][j - 1]  + (0 if str1[i - 1] ==
                       str2[j - 1] else 1)
```

```
24                     )
25
26     return matrix[-1][-1]
```

3.3 Функциональные тесты

В таблице 3.1 приведены тесты для тесты для алгоритмов вычисления расстояния. Тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Строка 1	Строка 2	Расстояние Левенштейна	Расстояние Дамерау – Левенштейна
строка	строка	0	0
qwerty	ytrewq	6	5
звезда	звезды	1	1
бобака	бобик	2	2
цска	мгту	4	4
ерер	репе	2	2

4 Исследовательская часть

В данной части будут использоваться следующие обозначения для различных алгоритмов поиска расстояний.

— A1 — матричный алгоритм поиска расстояния Левенштейна

— A2 — рекурсивный алгоритм поиска расстояния Левенштейна

— A3 — матричный алгоритм поиска расстояния Дамерау – Левенштейна

4.1 Технические характеристики

Технические характеристики устройства:

— операционная система Manjaro Linux x86_64;

— процессор Ryzen 5500U 6 ядер, тактовая частота 2.1 ГГц;

— оперативная память 16 Гбайт.

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

4.2 Сравнительный анализ временных затрат

Замеры времени для каждой длины слов проводились 100 раз. Результат замера — среднее арифметическое время работы алгоритма, на вход подавались сгенерированные случайным образом строки.

В таблице 4.1 представлены результаты измерений времени работы алгоритмов в зависимости от длины исходных строк.

Таблица 4.1 – Временные затраты

Длина	A1	A2	A3
1	0.0035	0.0020	0.0034
2	0.0018	0.0013	0.0017
3	0.0028	0.0036	0.0029
4	0.0045	0.0147	0.0048
5	0.0064	0.0728	0.0073
6	0.0091	0.3495	0.0101
7	0.0122	1.6686	0.0143
8	0.0159	9.0077	0.0182
9	0.0201	44.0207	0.0233
10	0.0249	248.8653	0.0290

На рисунке 4.1 представлено сравнение временных затрат для алгоритмов матричного поиска расстояния Левенштейна, рекурсивного поиска расстояния Левенштейна и матричного поиска расстояния Дамерау – Левенштейна.

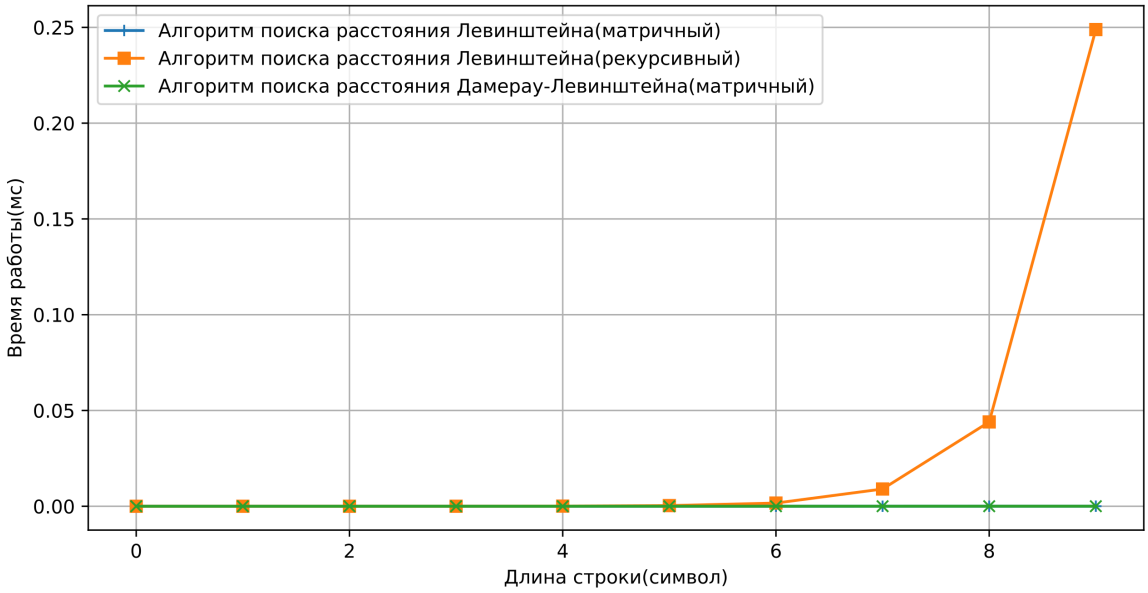

Рисунок 4.1 – Временные затраты

4.3 Сравнительный анализ затрат памяти

Для проведения замеров объёма затрачиваемаемой памяти, на вход подавались сгенерированные случайным образом строки различной длины.

В таблице 4.2 представлены результаты измерений затрат памяти при работе алгоритмов в зависимости от длины исходных строк.

Таблица 4.2 – Затраты памяти

Длина	A1	A2	A3
1	128	24	128
2	192	47	192
3	232	47	232
4	288	134	288
5	392	222	392
6	480	312	480
7	584	404	584
8	704	498	704
9	904	594	904
10	1056	692	1056

На рисунке 4.2 представлено сравнение затрат памяти для алгоритмов матричного поиска расстояния Левенштейна, рекурсивного поиска расстояния Левенштейна и матричного поиска расстояния Дамерау – Левенштейна.

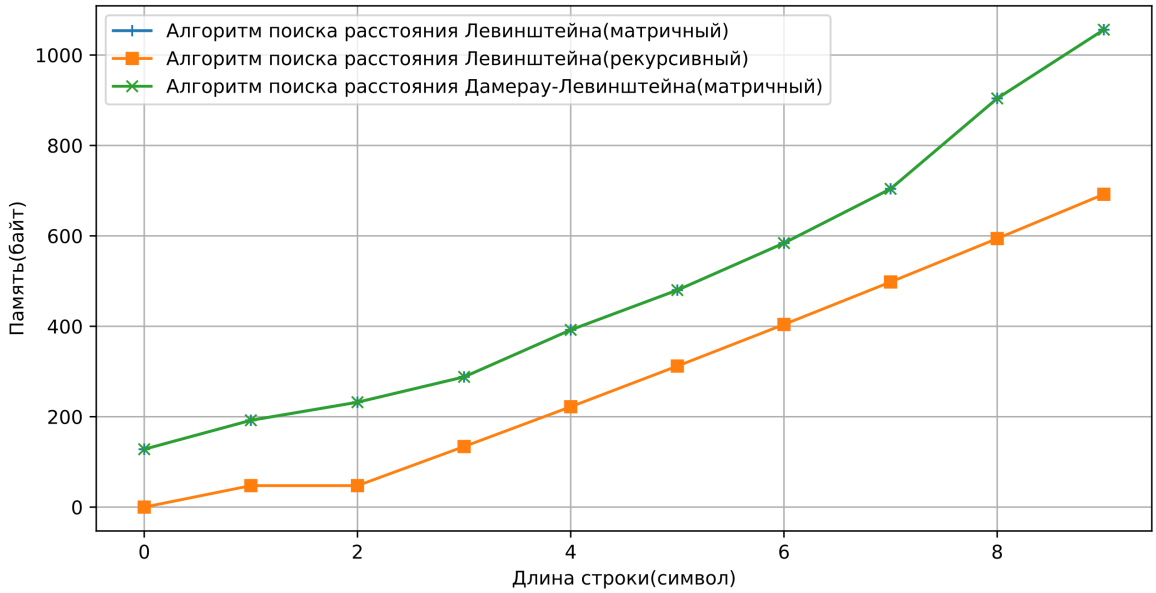

Рисунок 4.2 – Затраты памяти

4.4 Результаты проведенных исследований

По полученным данным измерений временных затрат был сделан вывод о том, что рекурсивный алгоритм поиска расстояния Левенштейна имеет экспоненциальный рост временных затрат при увеличении длины строки, что является значительно менее эффективным по сравнению с матричными алгоритмами поиска расстояний Левенштейна и Дамерау – Левенштейна.

По полученным данным измерений затрат памяти был сделан вывод о том, что рекурсивный алгоритм поиска расстояния Левенштейна требует меньшего количества затрат памяти, по сравнению с матричными алгоритмами поиска расстояний Левенштейна и Дамерау – Левенштейна, которые проигрывают из-за необходимости использовать дополнительную память под матрицу промежуточных значений.

ЗАКЛЮЧЕНИЕ

Исследованы алгоритмы поиска расстояния Левенштейна, поиска расстояния Дамерау – Левенштейна. Цель работы достигнута. В ходе выполнения лабораторной работы были решены следующие задачи:

— разработаны алгоритмы поиска расстояний Левенштейна и Дамерау – Левенштейна;

— разработаны программное обеспечение, содержащее данные алгоритмы;

— проведено функциональное тестирование реализованных алгоритмов;

— проведен сравнительный анализ затрат времени и памяти при различных реализациях алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Задача о редакционном расстоянии, алгоритм Вагнера-Фишера. [Электронный ресурс]. Санкт-Петербург, 2022. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%BC_%D1%80%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B8,_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%92%D0%B0%D0%B3%D0%BD%D0%B5%D1%80%D0%B0-%D0%A4%D0%B8%D1%88%D0%B5%D1%80%D0%B0. (Дата обращения: 09.12.2024).

[2] Задача о расстоянии Дамерау-Левенштейна. [Электронный ресурс]. Санкт-Петербург, 2022. URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D1%80%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B8_%D0%94%D0%B0%D0%BC%D0%B5%D1%80%D0%B0%D1%83-%D0%9B%D0%B5%D0%B2%D0%B5%D0%BD%D1%88%D1%82%D0%B5%D0%B9%D0%BD%D0%B0. (Дата обращения: 09.12.2024).