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»

на тему: «Расстояние Левенштейна»

Вариант № 7791

Студент ИУ7-56Б (Группа) _____ (Подпись, дата) Александрова А. Д. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Строганов Д. В. (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является изучение алгоритмов нахождения расстояния Левенштейна и расстояния Дамерау–Левенштейна.

В качестве **задач**, которые необходимо выполнить для достижения этой цели, можно выделить следующие:

1) Реализация алгоритмов поиска расстояния Левенштейна — итерационного, рекурсивного и рекурсивного с мемоизацией.

2) Реализация алгоритма нахождения расстояния Дамерау–Левенштейна.

3) Исследование ресурсной эффективности реализованных алгоритмов, сравнительный анализ результатов.

4) Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитический раздел

1.1 Определение расстояния Левенштейна

Расстояние Левенштейна между двумя строками — это минимальное количество операций вставки, удаления и замены, необходимых для превращения одной строки в другую.

Расстояние Левенштейна применяется в теории информации и компьютерной лингвистике для:

— исправления ошибок в слове

— сравнения текстовых файлов утилитой diff

— в биоинформатике для сравнения генов, хромосом и белков

Расстояние Дамерау–Левенштейна отличается от расстояния Левенштейна добавлением операции перестановки двух рядом стоящих букв в слове.

1.2 Вычисление расстояния Левенштейна

Пусть S_N и S_M — две строки длиной N и M соответственно. Цены всех операций примем равными 1. Тогда для этих строк расстояние Левенштейна $d(S_N, S_M) = D(N, M)$, где

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

где

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

Рассмотрим три алгоритма реализации нахождения расстояние Левенштейна.

1) **Рекурсивный алгоритм**. Напрямую реализует формулу 1.1.

2) **Матричный (итеративный) алгоритм**. Представляет собой построчное заполнение матрицы $[N \times M]$ промежуточными значениями $D(i, j)$.

3) **Рекурсивный алгоритм с мемоизацией**. Оптимизирует рекурсивный алгоритм кэшированием уже вычисленных значений $D(i, j)$. В случае, если рекурсивный алгоритм выполняет прогон для данных, которые еще не были обработаны, результат нахождения расстояния заносится в кэш. В случае, если обработанные ранее данные встречаются снова, для них расстояние повторно не находится, а алгоритм переходит к следующему шагу.

1.3 Вычисление расстояния Дамерау–Левенштейна

Расстояние Дамерау–Левенштейна вычисляется по приведенной ниже формуле, аналогичной формуле вычисления расстояния Левенштейна.

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

2 Конструкторский раздел

2.1 Требования к программе

К программе предъявлены следующие требования:

— вход: две строки

— выход: целое число — расстояние Левенштейна/Дамерау–Левенштейна

— наличие пользовательского интерфейса для выбора действий;

— возможность вывода заполняемой матрицы для итеративных алгоритмов;

— наличие замера процессорного времени работы реализаций алгоритмов поиска расстояний Левенштейна и Дамерау–Левенштейна.

2.2 Проектирование алгоритмов

В процессе проектирования программы были разработаны схемы алгоритмов, представленные на рисунках 2.1 — 2.6

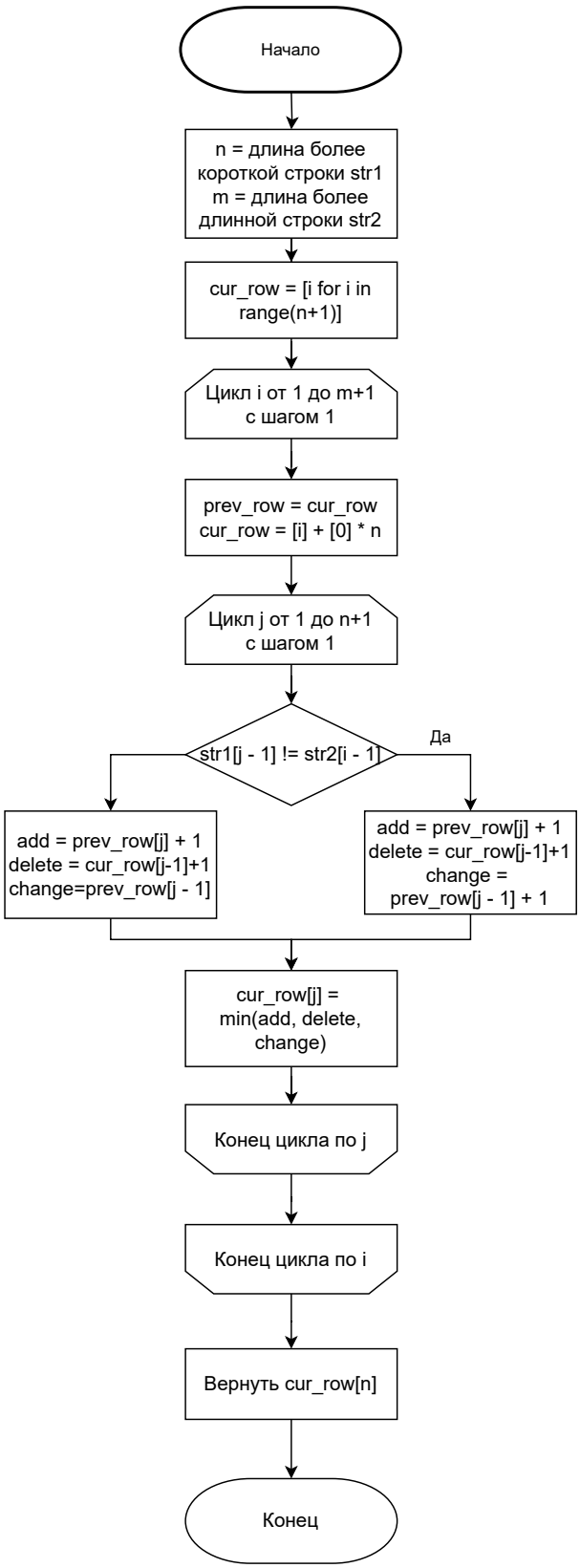

Рисунок 2.1 – Схема итеративного алгоритма нахождения расстояния Левенштейна

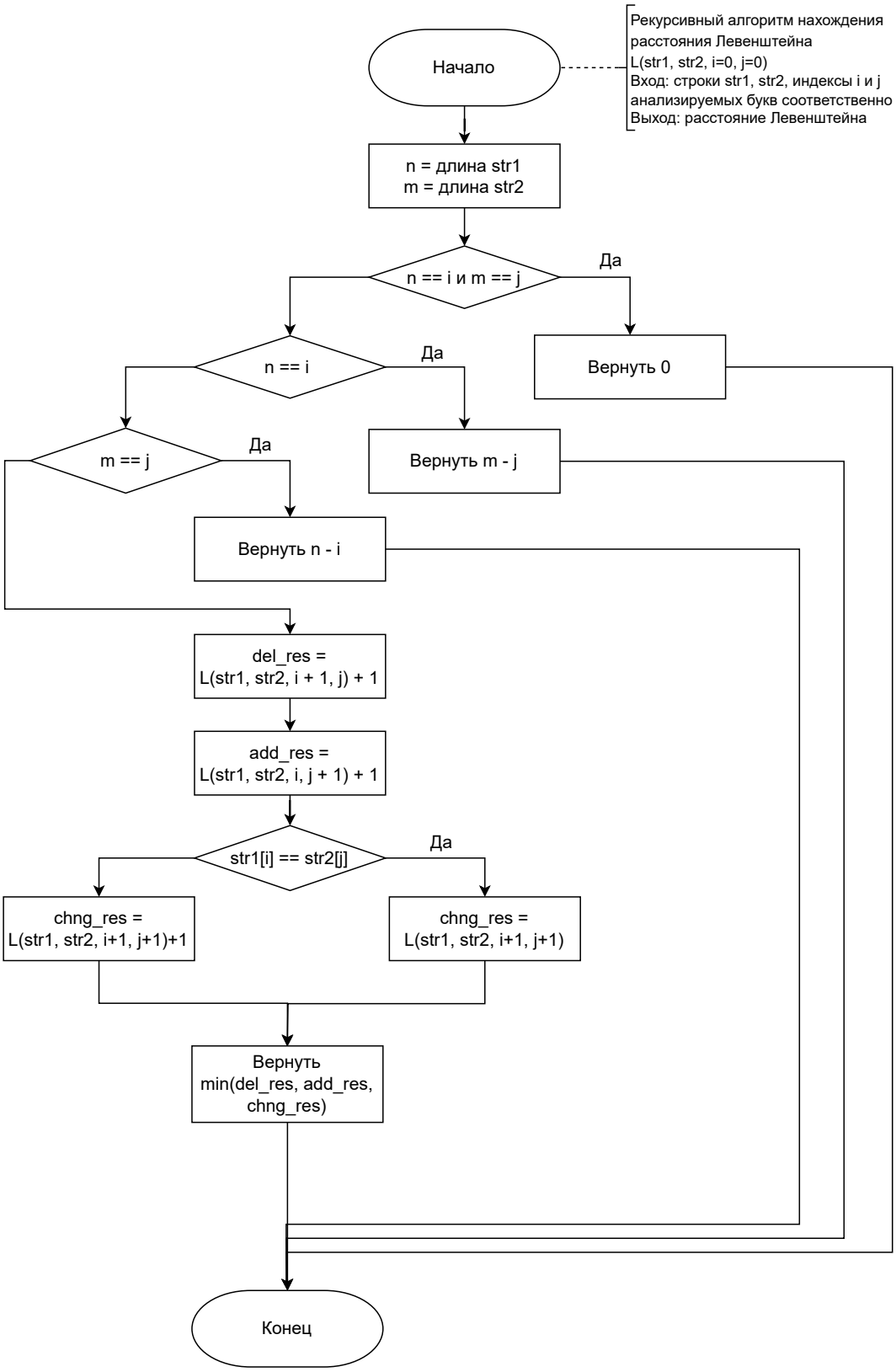

Рисунок 2.2 – Схема рекурсивного алгоритма нахождения расстояния Левенштейна

Рисунок 2.3 – Схема рекурсивного с мемоизацией алгоритма нахождения расстояния Левенштейна

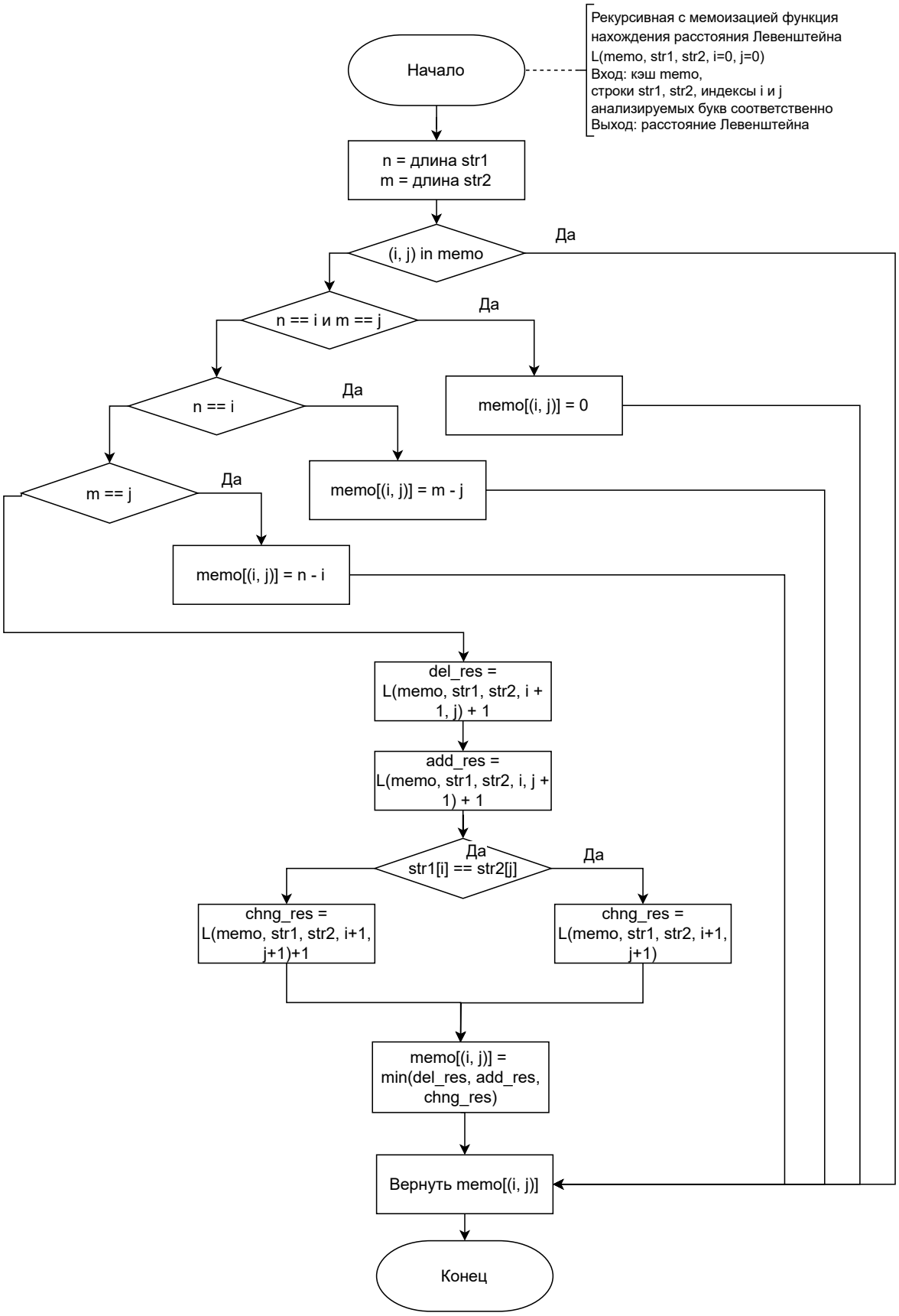

Рисунок 2.4 – Схема функции L из рекурсивного с мемоизацией алгоритма нахождения расстояния Левенштейна

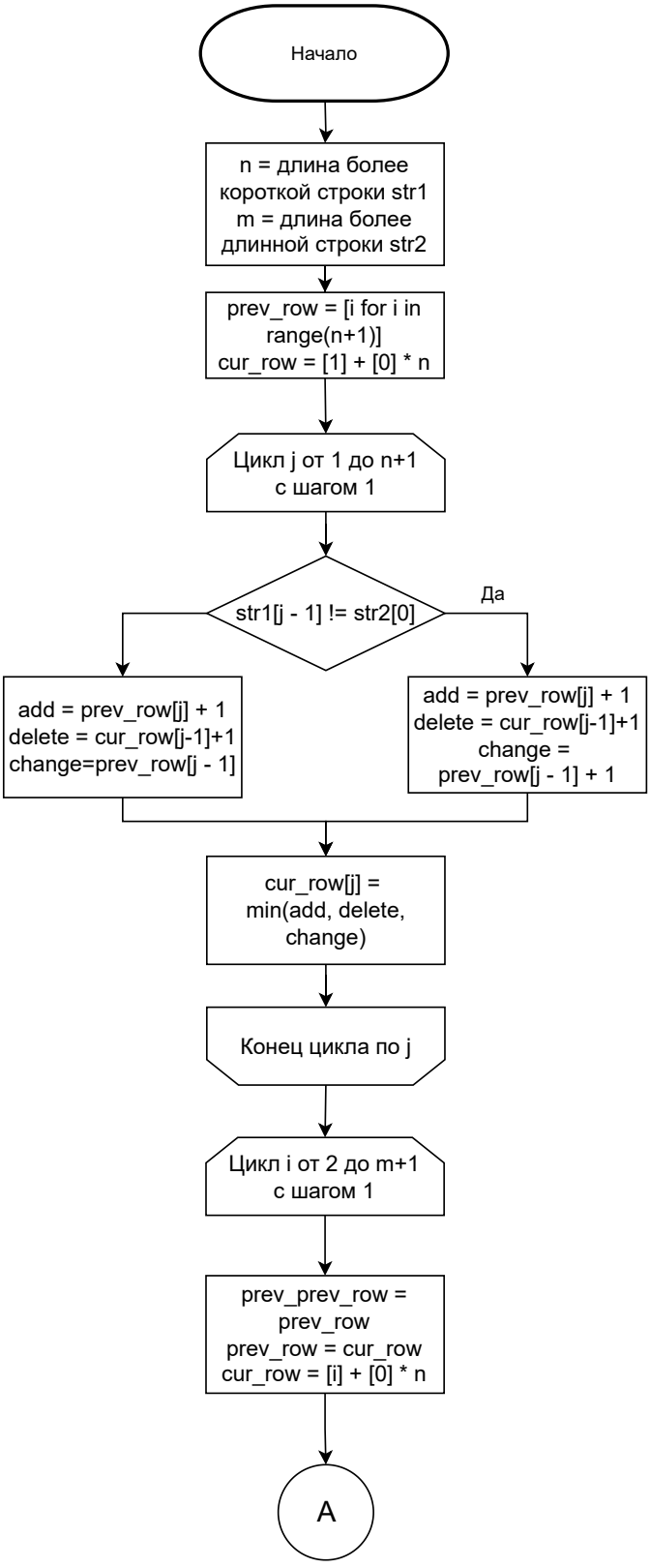

Рисунок 2.5 – Схема итеративного алгоритма нахождения расстояния Дамерау–Левенштейна. 1 часть

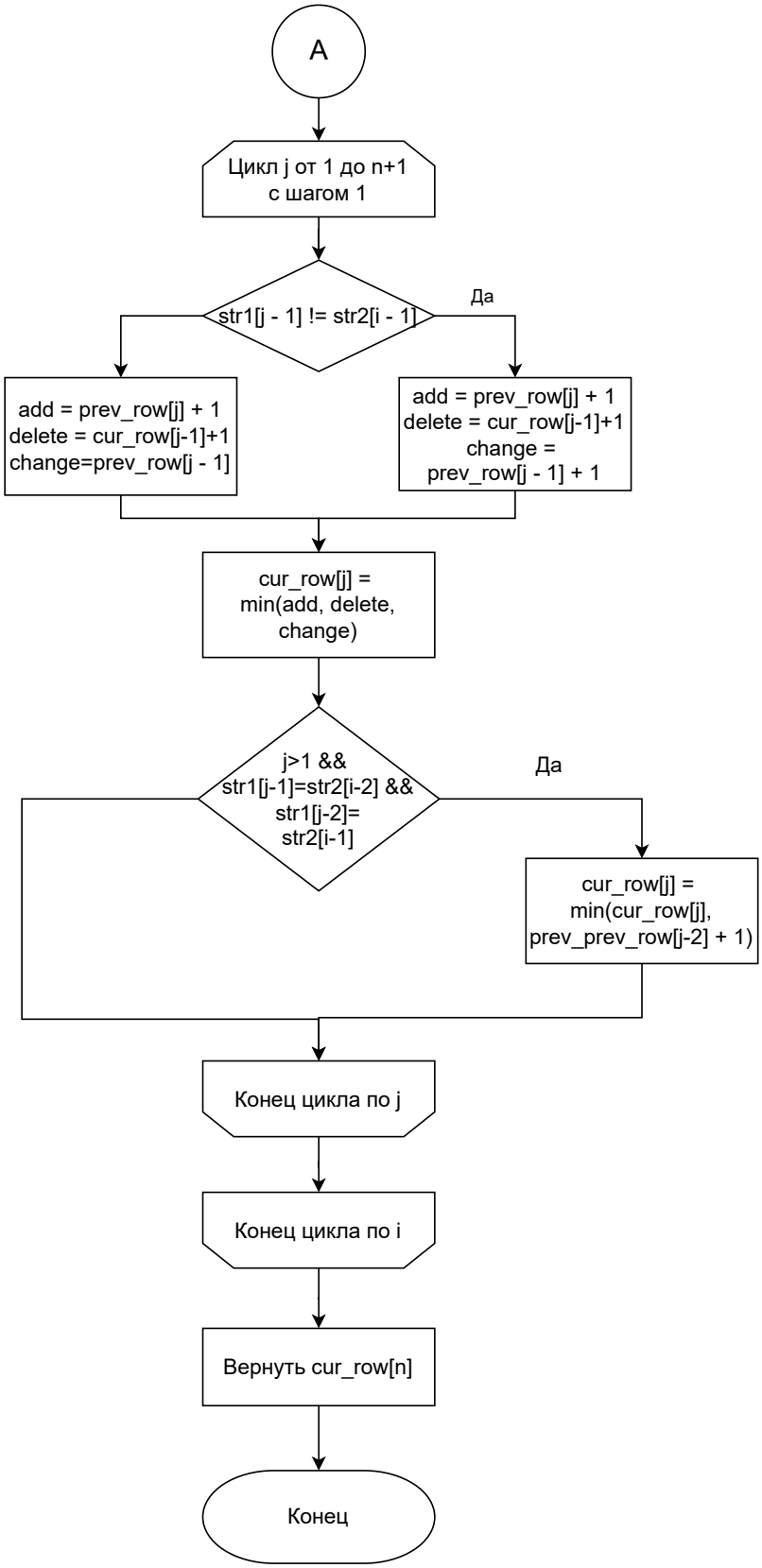

Рисунок 2.6 – Схема итеративного алгоритма нахождения расстояния Дамерау–Левенштейна. 2 часть

3 Технологический раздел

3.1 Средства реализации

Для реализации данной лабораторной был выбран язык Python, как простой язык с широким функционалом, содержащий все необходимые для работы инструменты и позволяющий завершить этап кодирования в кратчайшие сроки. Для замеров времени используется функция process_time() из модуля time(), возвращающая процессорное время.

3.2 Программные модули

Программа разбита на следующие модули:

— **main.py** — модуль, реализующий пользовательский интерфейс. Вызывает функции алгоритмов

— **funcs.py** — модуль содержит функции поиска расстояния Левенштейна и Дамерау–Левенштейна. Функции итеративных алгоритмов с выводом матрицы работы вызываются в main.py, функции без вывода используются для замеров времени работы алгоритмов.

— **tests.py** — модуль для проведения функционального тестирования.

— **time_mes.py** — модуль, реализующий замеры времени.

— **graph.py** — модуль для графического отображения полученных замеров времени.

3.3 Реализация алгоритмов

В результате разработки алгоритмов были получены листинги 3.1 — **??**

Листинг 3.1 – Исходный код итеративного алгоритма вычисления расстояния Левенштейна

```
def matrix_levenstein(str1, str2):
    n, m = len(str1), len(str2)
    if n > m:
        str1, str2 = str2, str1
        n, m = m, n
    cur_row = [i for i in range(n + 1)]
    for i in range(1, m + 1):
        prev_row, cur_row = cur_row, [i] + [0] * n
        for j in range(1, n + 1):
            if str1[j - 1] != str2[i - 1]:
                add, delete, change = prev_row[j] + 1, cur_row[j
                    - 1] + 1, prev_row[j - 1] + 1
            else:
                add, delete, change = prev_row[j] + 1, cur_row[j
                    - 1] + 1, prev_row[j - 1]
            cur_row[j] = min(add, delete, change)
    return cur_row[n]
```

Листинг 3.2 – Исходный код рекурсивного алгоритма вычисления расстояния Левенштейна

```
def recurs_levenstein(str1, str2, i=0, j=0):
    if len(str1) == i and len(str2) == j:
        return 0
    if len(str1) == i:
        return len(str2) - j
    if len(str2) == j:
        return len(str1) - i
    del_res = 1 + recurs_levenstein(str1, str2, i + 1, j)
    add_res = 1 + recurs_levenstein(str1, str2, i, j + 1)
    if str1[i] == str2[j]:
        chng_res = recurs_levenstein(str1, str2, i + 1, j + 1)
    else:
        chng_res = 1 + recurs_levenstein(str1, str2, i + 1, j +
           1)
    return min(del_res, add_res, chng_res)
```

Листинг 3.3 – Исходный код рекурсивного с мемоизацией алгоритма вычисления расстояния Левенштейна

```
def memo_recurs_levenstein(memo, str1, str2, i=0, j=0):
    if (i, j) in memo:
        pass
    elif len(str1) == i and len(str2) == j:
        memo[(i, j)] = 0
        return 0
    elif len(str1) == i:
        memo[(i, j)] = len(str2) - j
        return len(str2) - j
    elif len(str2) == j:
        memo[(i, j)] = len(str1) - i
        return len(str1) - i
    else:
        del_res = 1 + memo_recurs_levenstein(memo, str1, str2, i
           + 1, j)
        add_res = 1 + memo_recurs_levenstein(memo, str1, str2,
           i, j + 1)
        if str1[i] == str2[j]:
            chng_res = memo_recurs_levenstein(memo, str1, str2,
               i + 1, j + 1)
        else:
            chng_res = 1 + memo_recurs_levenstein(memo, str1,
               str2, i + 1, j + 1)
        memo[(i, j)] = min(del_res, add_res, chng_res)
    return memo[(i, j)]


def memo_levenstein(str1, str2):
    memo = {}
    return memo_recurs_levenstein(memo, str1, str2)
```

Листинг 3.4 – Исходный код итеративного алгоритма вычисления расстояния Дамерау–Левенштейна

```
def damerau_levenstein(str1, str2):
    n, m = len(str1), len(str2)
    if n > m:
        str1, str2 = str2, str1
        n, m = m, n
    prev_row = [i for i in range(n + 1)]
    cur_row = [1] + [0] * n
    for j in range(1, n + 1):
        if str1[j - 1] != str2[0]:
            add, delete, change = prev_row[j] + 1, cur_row[j -
               1] + 1, prev_row[j - 1] + 1
        else:
            add, delete, change = prev_row[j] + 1, cur_row[j -
               1] + 1, prev_row[j - 1]
        cur_row[j] = min(add, delete, change)

    for i in range(2, m + 1):
        prev_prev_row, prev_row, cur_row = prev_row, cur_row,
           [i] + [0] * n
        for j in range(1, n + 1):
            if str1[j - 1] != str2[i - 1]:
                add, delete, change = prev_row[j] + 1, cur_row[j
                   - 1] + 1, prev_row[j - 1] + 1
            else:
                add, delete, change = prev_row[j] + 1, cur_row[j
                   - 1] + 1, prev_row[j - 1]
            cur_row[j] = min(add, delete, change)
            if j > 1 and str1[j - 1] == str2[i - 2] and str1[j -
               2] == str2[i - 1]:
                cur_row[j] = min(cur_row[j], prev_prev_row[j -
                   2] + 1)
    return cur_row[n]
```

3.4 Функциональные тесты

Функциональные тесты приведены в таблице 3.1 и были пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входные данные		**Полученное расстояние**			
Строка 1	**Строка 2**	**Левенштейн**			**Дамерау Левенштейн**
		Итер.	**Рекурс.**	**Рекурс. с мемоиз.**	
а	а	0	0	0	0
а	б	1	1	1	1
вход	вдох	2	2	2	2
кошка	собака	3	3	3	3
абвгд	авбдг	3	3	3	2
трон	ртопа	4	4	4	3
пример	примеры	1	1	1	1

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz.

— Оперативная память: 8 ГБайт.

— Операционная система: Windows 11 Home версии 23H2.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Временные характеристики

Замеры времени работы алгоритмов проводились на строках длиной от 1 до 100 символов для всех алгоритмов кроме рекурсивного алгоритма Левенштейна, так как он работает на порядки раз дольше остальных даже на размере строки до 10 символов, и дальнейшие замеры не имеют смысла и выполняются слишком долго. Для каждой длины строки замеры были выполнены 300 раз и получено среднее значение, занесенное в результат.

Результаты представлены в таблице 4.1. Прочерк в таблице означает, что замеры не проводились.

По приведенным результатам можно увидеть упомянутое выше значительное различие во времени работы рекурсивного алгоритма Левенштейна и остальных трёх. Рекурсивный с мемоизацией алгоритм Левенштейна в свою очередь в разы медленнее итеративного. Алгоритм Дамерау–Левенштейна же на небольших значениях отличается от итеративного незначительно, но с увеличением длины строки растет и разница во времени в силу наличия дополнительной проверки (при длине строки в 100 символов Дамерау–Левенштейн на 30% медленнее обычного Левенштейна).

Таблица 4.1 – Результаты замеров времени работы алгоритмов

Длина строки, символов	Время, мкс			
	Левенштейн			**Дамерау-Левенштейн**
	Итер.	**Рекурс.**	**Рекурс. с мемоиз.**	
1	1.833	1.316	2.296	1.521
2	3.197	5.313	5.942	3.268
3	5.658	27.062	12.569	6.159
4	8.879	142.767	21.676	10.286
5	13.221	770.604	36.284	17.050
6	17.051	3766.914	46.917	21.460
7	21.546	19985.579	62.689	28.966
8	27.777	108254.140	87.913	38.761
9	35.359	607252.302	115.044	49.243
10	40.872	3338151.763	132.939	57.089
20	154.048	-	459.356	193.552
30	332.489	-	1024.083	424.310
40	578.928	-	1849.603	738.465
50	886.866	-	3173.269	1153.285
60	1304.971	-	4915.784	1674.950
70	1771.331	-	6825.604	2254.049
80	2292.528	-	8963.647	2938.029
90	2887.955	-	11290.257	3704.933
100	3595.309	-	14109.328	4667.696

Те же выводы можно сделать, изучив графическое представление замеров времени, представленные на Рисунках 4.1 и 4.2

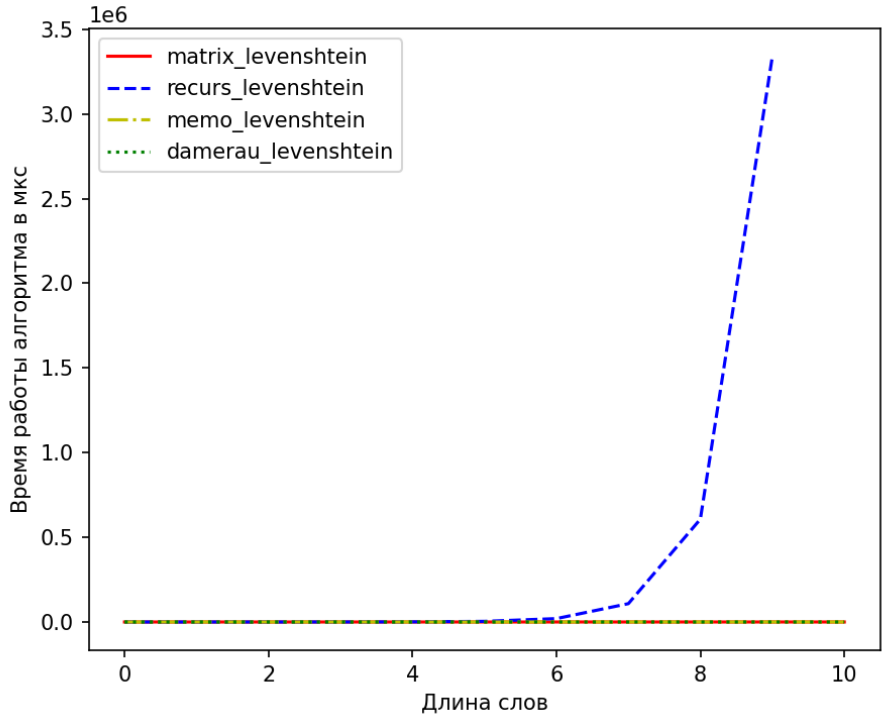

Рисунок 4.1 – Результаты замеры времени работы всех алгоритмов

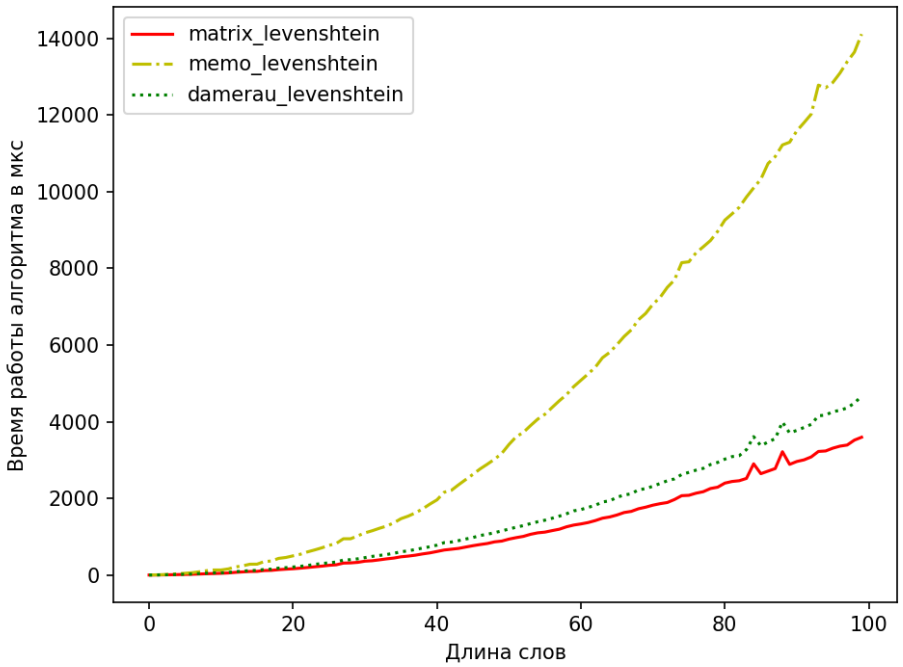

Рисунок 4.2 – Результаты замеров без рекурсивного алгоритма на большем интервале

4.3 Характеристики памяти

В данной реализации итеративного алгоритма поиска расстояния Левенштейна в памяти хранятся 2 последних строки матрицы размером $(N+1) \cdot (M+1)$, где N - длина наименьшей входной строки, а M - наибольшей. В таком случае размер используемой памяти примерно равен $2 \cdot (N+1) \cdot sizeof(int)$.

Для итеративного алгоритма поиска расстояния Дамерау–Левенштейна ситуация аналогичная, только память выделяется на 3 последние строки в размере $3 \cdot (N+1) \cdot sizeof(int)$.

В рекурсивной реализации без мемоизации, для каждого рекурсивного вызова функции будет необходимо выделять память под стековый фрейм. Максимальная глубина стека вызовов на 2 больше суммы длин исходных строк. Следовательно, пиковая затрачиваемая память растет пропорционально сумме длин строк.

Рекурсивный алгоритм с мемоизацией выделяет память как под полную матрицу расстояний размером $(N+1) \cdot (M+1)$, так и под стековые фреймы для каждого вызова.

ЗАКЛЮЧЕНИЕ

В результате исследования можно сделать вывод, что итеративные алгоритмы являются наиболее быстрыми из всех изученных, притом расстояние Левенштейна вычисляется быстрее расстояния Дамерау–Левенштейна. Память используемая итеративными алгоритмами пропорциональна длине наименьшей строки, а рекурсивным алгоритмом без мемоизации — сумме двух входных строк. Рекурсивный алгоритм с мемоизацией же является самым затратным по памяти.

Таким образом, в результате выполнения лабораторной работы были изучены и реализованы алгоритмы поиска расстояния Левенштейна — итерационный, рекурсивный и рекурсивный с мемоизацией, а также алгоритм поиска расстояния Дамерау–Левенштейна, и проведена оценка их ресурсной эффективности, то есть выполнены все поставленные задачи, и цель выполнения работы — изучение алгоритмов поиска расстояний Левенштейна и Дамерау–Левенштейна — можно считать достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов //. Т. 163. — Доклады АН СССР, 1965. — Гл. 4. С. 845—848.

2. *Ульянов М. В.* Ресурсно-эффективные компьютерные алгоритмы: учебное пособие //. — М.: Издательство «Наука», ФИЗМАТЛИ, 2007. — Гл. 6.

3. *Кормановский М. В.* Граф на основе расстояния Левенштейна и его визуализация / М. В. Кормановский, Н. П. Артюхин, А. А. Косарев [и др.] // Проблемы лингвистики и лингводидактики в неязыковом вузе : 5-я Международная научно-практическая конференция: сборник материалов конференции. В 2-х томах, Москва, 15 декабря 2022 года. Том 1. – Москва: Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), 2023. – С. 310-319.

4. Документация по Python time.process_time() [Электронный ресурс]. — — Режим доступа: `https://docs.python.org/3/library/time.html#time.process_time` (Дата обращения: 30.09.2024).