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

Студент Сабуров С. М.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение

Расстояние Левенштейна - минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Расстояние Левенштейна применяется в теории информации и компьютерной лингвистике для следущего:

— исправления ошибок в слове;

— сравнения текстовых файлов утилитой diff;

— в биоинформатике для сравнения генов, хромосом и белков.

Цель данной лабораторной работы: реализация и анализ алгоритмов нахождения расстояние Левенштейна и Дамерау-Левенштейна.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Изучение алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками.
2. Применение метода динамического программирования для матричной реализации указанных алгоритмов.
3. Получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии.
4. Сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти).
5. Экспериментальное подтверждение различий во временно́й 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.
6. Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

Расстояние Левенштейна [2] между двумя строками – это минимальное количество операций вставки, удаления и замены, необходимых 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;

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

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

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

— $w(a, a) = 0$;

— $w(a, b) = 1, a \neq b$;

— $w(\lambda, b) = 1$;

— $w(a, \lambda) = 1$.

1.1 Расстояние Левенштейна

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда расстояние Левенштейна можно подсчитать по следующей рекуррентной формуле:

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

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

1.2 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна вычисляется по следующей рекуррентной формуле:

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

1.3 Вывод

В данном разделе были рассмотрены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна, который является модификаций первого, учитывающего возможность перестановки соседних символов.

— Входные данные — на вход подаются две строки.Заглавные и прописные буквы считаются разными.

— Выходные данные — на выходе имеем целое число.

— Ограничения, в рамках которых будет работать программа — Две пустые строки - корректный ввод, программа не должна аварийно завершаться.

— Функциональные требования — функции, представленные на листингах 1 - 3 должны вычислять расстояние Левенштейна, принимая на вход 2 строки и длины этих строк и возвращая целое число. Функция пред-

ставленная на листинге 4 должна вычислять расстояние Дамерау - Левенштейна, принимая на вход 2 строки и длины этих строк и возвращая целое число.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов

2.1 Схемы алгоритмов

На рис. 1-7 показаны схемы рассматриваемых алгоритмов.

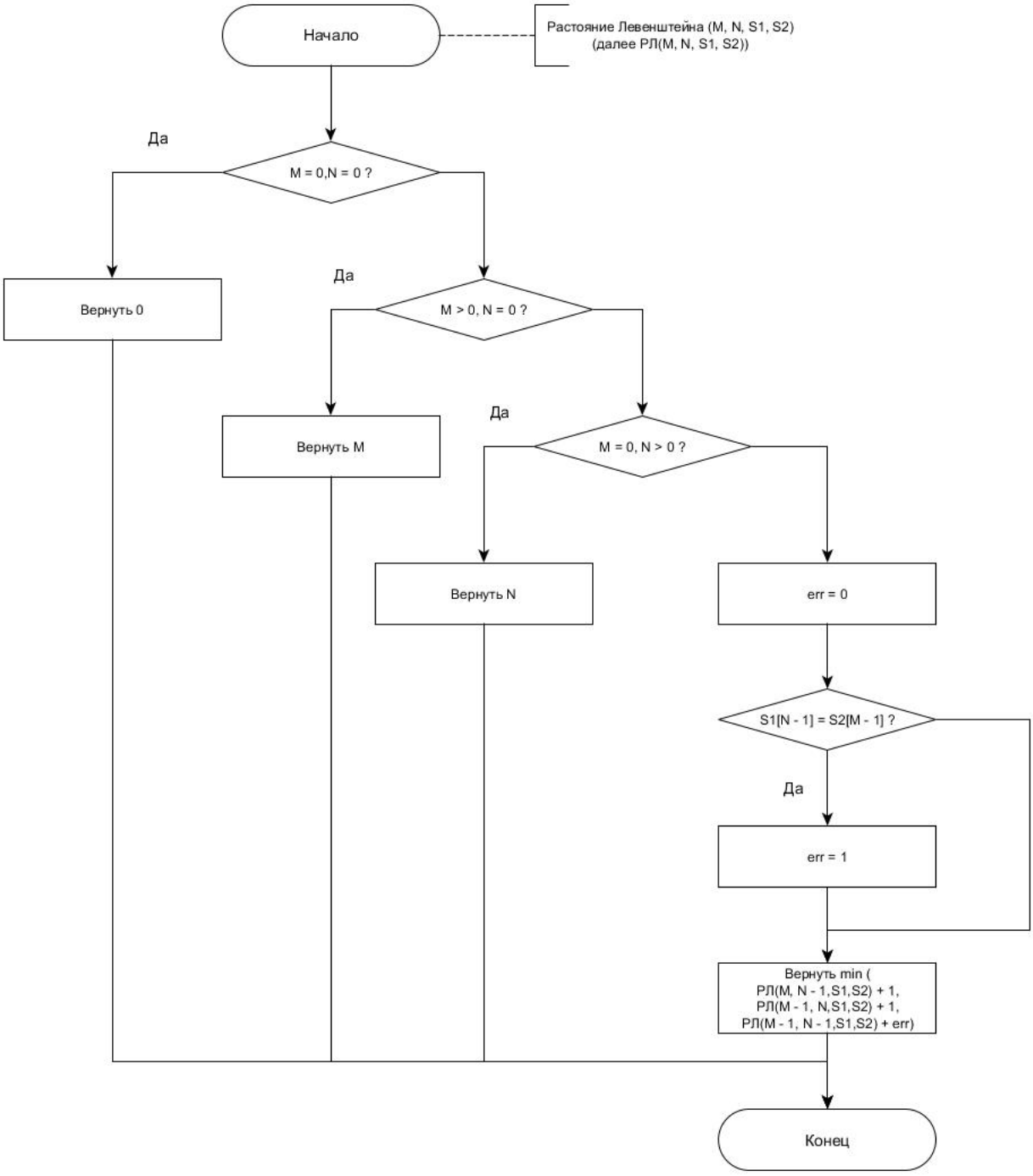

Рис. 1 — Схема рекурсивного алгоритма нахождения расстояния Левенштейна

2.2 Описание структур данных

В алгоритмах, где подразумевается работа с матрицей, используются 2 массива, содержащий последовательно идущие строки матрицы. Такой выбор был сделан из соображений экономии памяти.

2.3 Способы тестирования и классы эквивалентности

Была выбрана методика тестирования черным ящиком. Классы эквивалентности:

— Одинаковые строки.

— Строки, отличающиеся в один символ .

— Пустые строки.

— Строки, полностью отличающиеся по символьному содержанию .

2.4 Использование памяти

Алгоритмы Левенштейна и Дамерау-Левенштейна не отличаются по использованию памяти, соответственно достаточно рассмотреть рекурсивный и матричный реализации этих алгоритмов.

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк, а на каждый вызов функции требуется еще 5 дополнительных переменных типа *usize*, соответственно, максимальный расход памяти

$$(Size(S_1) + Size(S_2) \cdot (2 \cdot Size(\text{string}) + 5 \cdot Size(\text{usize}))) \quad (1)$$

где Size - функция, возвращающая размер аргумента; string - строковый тип, usize - целочисленный, беззнаковый тип.

Использование памяти при итеративной реализации теоритически равно

$$(Size(S_1 + 1) \cdot Size(S_2 + 1)) \cdot Size(usize) + 2 \cdot Size(string) \quad (2)$$

Использование памяти рекурсивной реализации алгоритма Левенштейна с кэшем теоритически равно

$$(Size(S_1) + Size(S_2) \cdot (2 \cdot Size(string) + \\ + 5 \cdot Size(usize)) + (Size(S_1 + 1) \cdot Size(S_2 + 1)) \cdot Size(usize)) \quad (3)$$

2.5 Вывод

Рекурсивный алгоритм Левенштейна работает на порядок дольше итеративных реализаций, время его работы увеличивается в геометрической прогрессии. На словах длиной 9 символов, матричная реализация превосходит рекурсивную в 4800 раз. Рекурсивные алгоритмы Левенштейна и Дамерау - Левенштейна сопостовимы по времени. Однако, использование кэша значительно ускоряет рекурсивный алгоритм, но он все еще не превосходит матричную реализацию.

Из формул 2-3 можно сделать вывод, что рекурсивные алгоритмы потребляют больше памяти, чем матричные, при одинаковых длин строк.

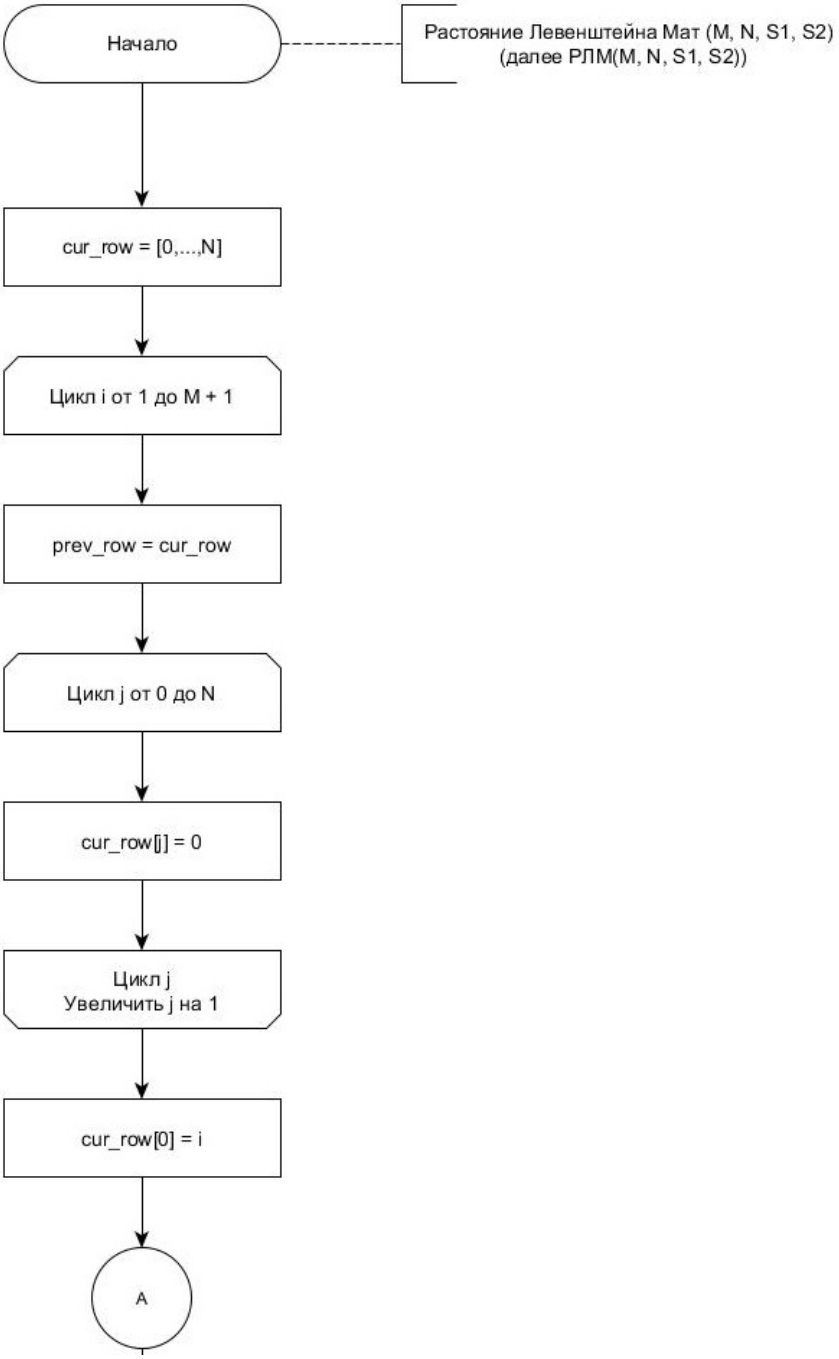

Рис. 2 — Схема матричного алгоритма нахождения расстояния Левенштейна. Часть 1

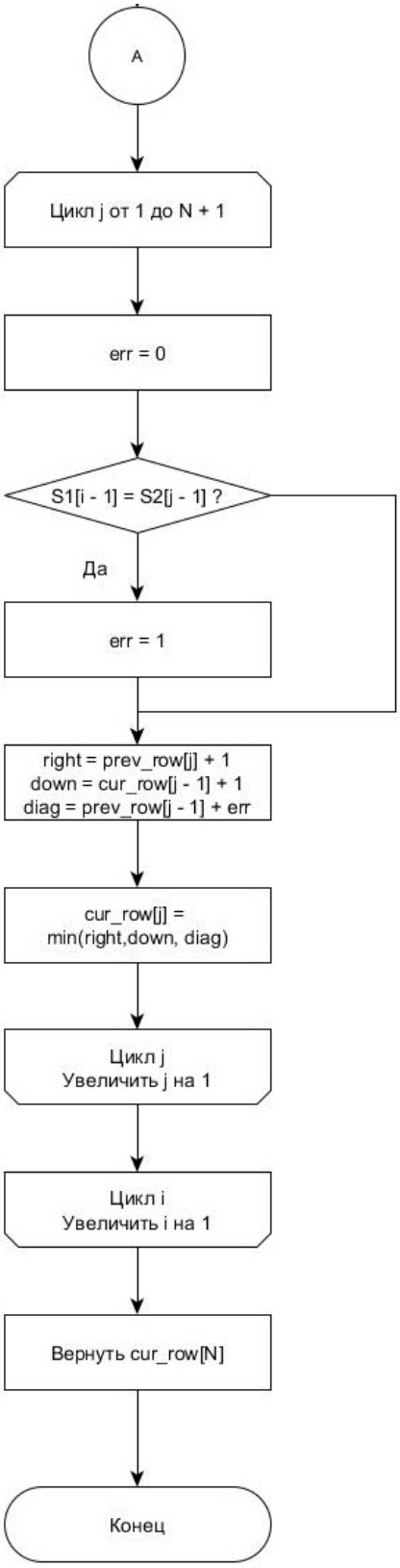

Рис. 3 — Схема матричного алгоритма нахождения расстояния Левенштейна. Часть 2

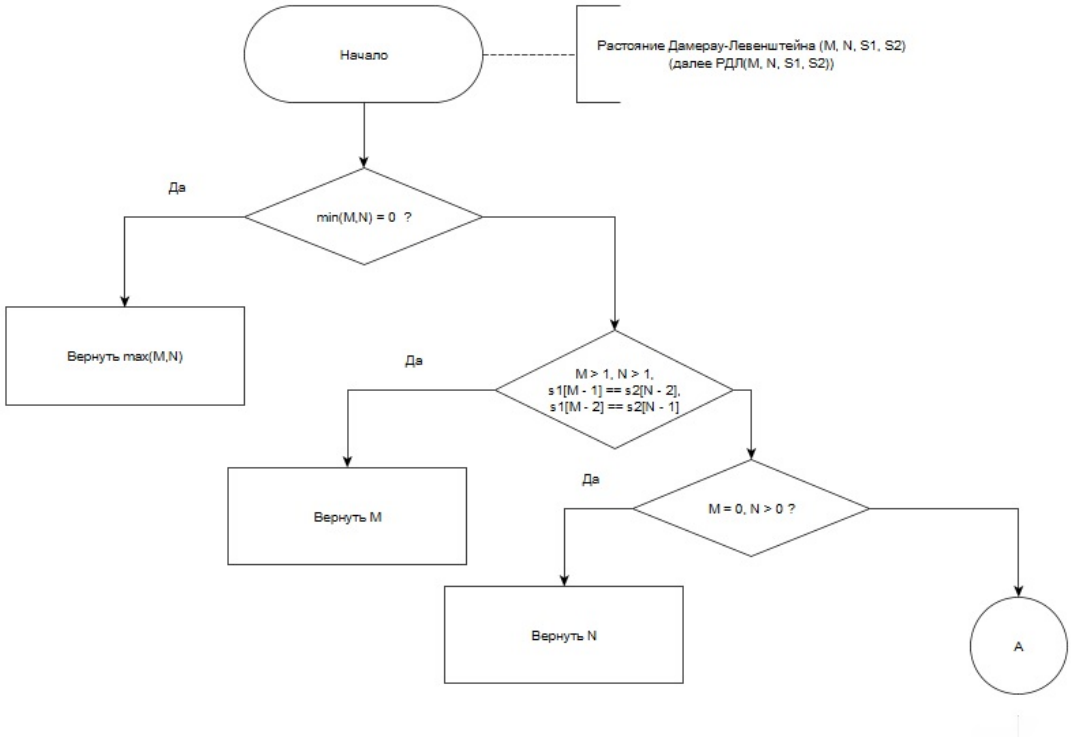

Рис. 4 — Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна. Часть 1

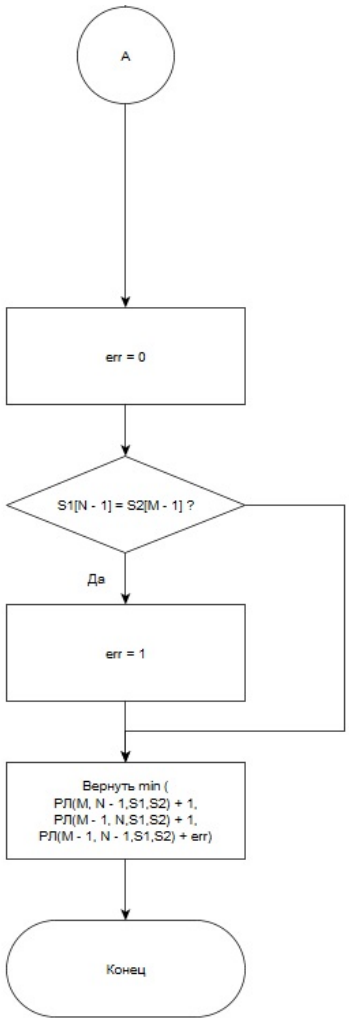

Рис. 5 — Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна. Часть 2

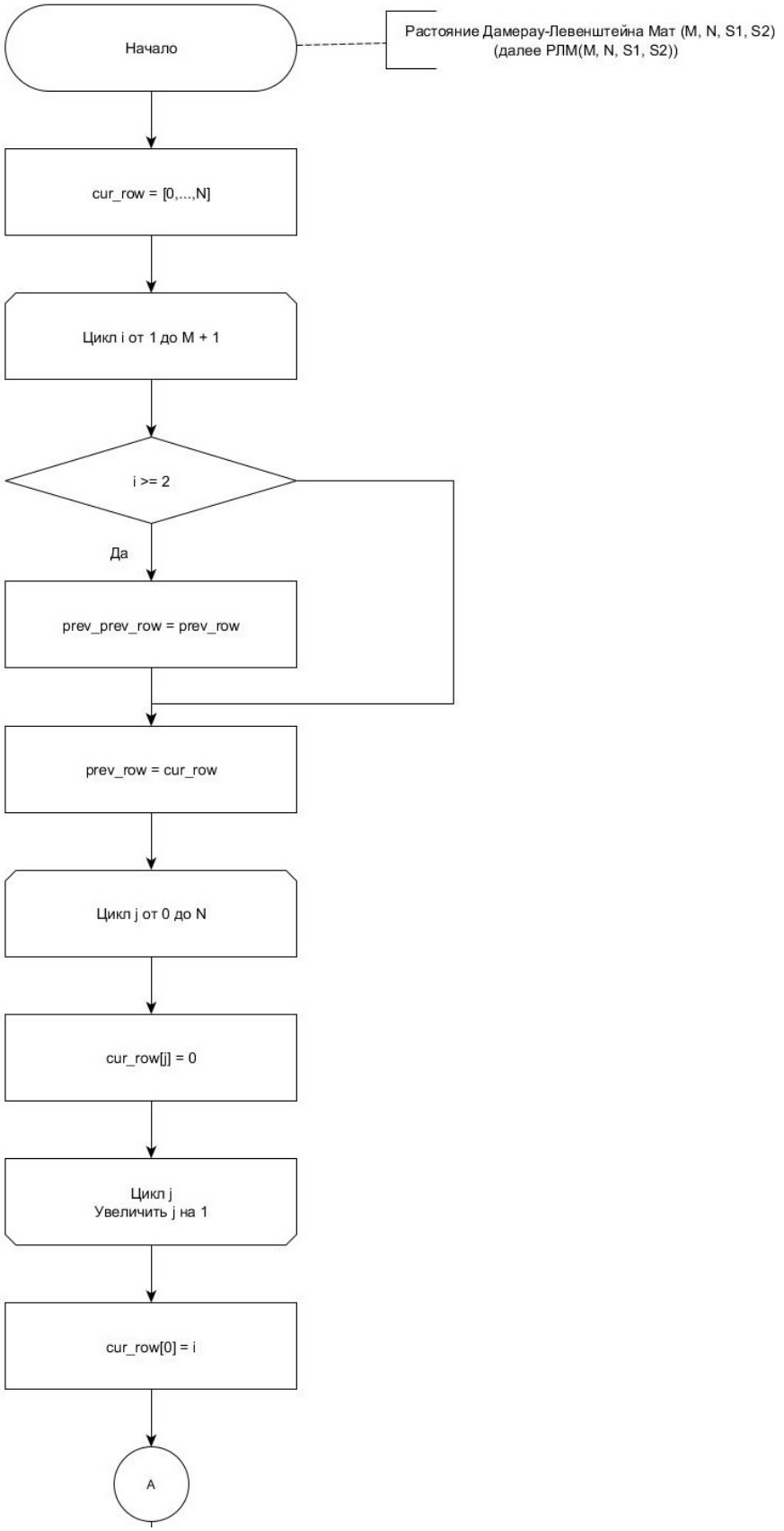

Рис. 6 — Схема матричного алгоритма нахождения расстояния Дамерау-Левенштейна. Часть 1

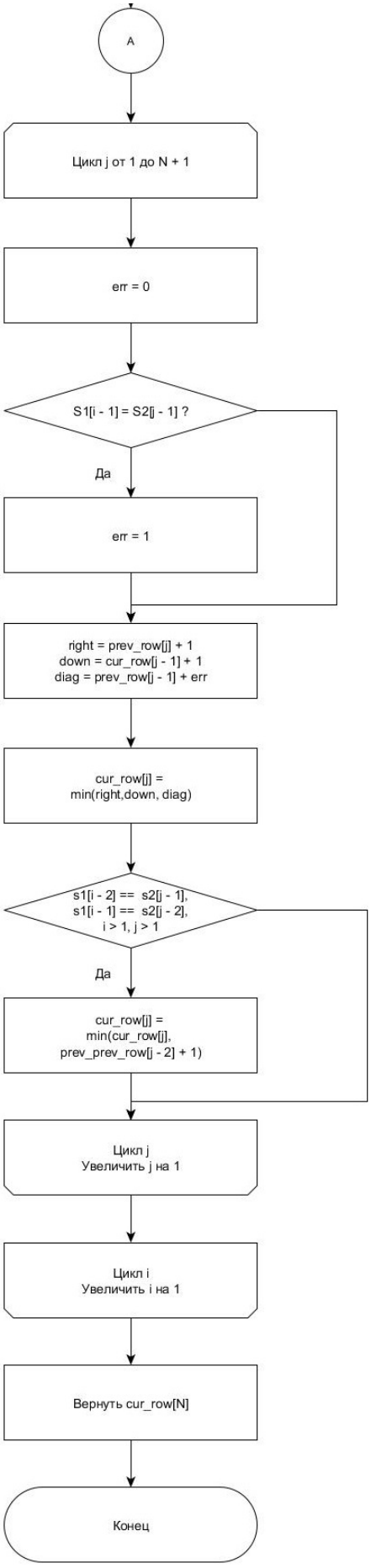

Рис. 7 — Схема матричного алгоритма нахождения расстояния Дамерау-Левенштейна. Часть 2

3 Технологическая часть

В данном разделе приведены листинги реализаций алгоритмов, тестирование и выбор языка программирования.

3.1 Выбор языка программирования

Для реализации программ я выбрал язык программирования Python [1], так как этот язык предоставляет как низкоуровневые интерфейсы, так и высокоуровневые. Также он является безопасным языком. Среда разработки IDLE. Также данный язык был выбран потому, что в нем присутствует инструментарий для замера процессорного времени и тестирования.

3.2 Листинги реализации алгоритма

В листингах 3.1 - 3.4 приведена реализация алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна.

Листинг 1: Функция нахождения расстояния Левенштейна рекурсивно

```
1  def lev_rec(M,N, s1,s2):
2      if (M == 0 and N == 0):
3          return 0
4      elif (M > 0 and N == 0):
5          return M
6      elif (M == 0 and N > 0):
7          return N
8      else:
9          return min(lev_rec(M, N - 1,s1,s2) + 1,\
10                        lev_rec(M - 1, N,s1,s2) + 1,\
11                        lev_rec(M - 1, N - 1,s1,s2) + bool(ord(s1[
     M - 1]) - ord(s2[N - 1])))
```

Листинг 2: Функция нахождения расстояние Левенштейна рекурсивно с мемоизацией

```
def rec_matr_lev(Matr, M, N, s1, s2):
    if Matr[M][N] == sys.maxsize:
        if (M == 0 and N == 0):
            Matr[M][N] = 0
            return int(0)
        elif (M > 0 and N == 0):
            Matr[M][N] = M
            return int(M)
        elif (M == 0 and N > 0):
            Matr[M][N] = N
            return int(N)
        else:
            err = 1
            if (s1[M - 1] == s2[N - 1]):
                err = 0
            Matr[M][N] =  min (rec_matr_lev(Matr,M, N - 1,s1, s2) + 1, \
                               rec_matr_lev(Matr,M - 1, N,s1, s2) + 1, \
                               rec_matr_lev(Matr,M - 1, N - 1,s1,s2) + err)
            return int(Matr[M][N])
    else:
        return Matr[M][N]
```

Листинг 3: Функция нахождения расстояния Левенштейна итеративно

```
def matr_lev(M, N, s1, s2):

    cur_row = 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):
            err = 1
            if s1[i - 1] == s2[j - 1]:
                err = 0
            right, down, diag = prev_row[j] + 1, \
                                cur_row[j - 1] + 1, \
```

```
12                                      prev_row[j - 1] + err
13                cur_row[j] = min(right, down, diag)
14        return cur_row[N]
```

Листинг 4: Функция нахождения расстояния Дамерау-Левенштейна матрично

```
def damer_leven_opt(M, N, s1, s2):
    cur_row = range(N + 1)
    for i in range (1, M + 1):
        if i >= 2:
            prev_prev_row = prev_row
        prev_row, cur_row = cur_row, [i] + [0] * N
        for j in range (1, N + 1):
            err = 1
            if s1[i - 1] == s2[j - 1]:
                err = 0
            right, down, diag = prev_row[j] + 1, cur_row[j - 1] + 1, prev_row[j - 1] + err
            cur_row[j] = min(right, down, diag)
            if s1[i - 2] == s2[j - 1] and s1[i - 1] == s2[j - 2] and i > 1 and j > 1:
                cur_row[j] = min(cur_row[j], prev_prev_row[j - 2] + 1)
    return cur_row[N]
```

3.3 Тестирование

Тестовые данные были подобраны таким образом, чтобы покрыть все возможнные случаи.

В Таблице 1 приведены функциональные тесты для алгоритмов вычисления расстояния Левенштейна и Дамерау-Левенштейна.

Все тесты были пройдены успешно (ожидаемый результат совпал с фактическим).

Таблица 1 — Функциональные тесты

Строка 1	Строка 2	Ожидаемый результат	
		Левенштейн	Дамерау-Левенштейн
Take	Took	3	3
Art	Atr	2	1
car	city	3	3
head	ehda	3	2
laptop	notebook	7	7
peek	peeks	1	1
rain	pain	1	1
Пусто	a	1	1
a	Пусто	1	1
Пусто	Пусто	0	0

4 Исследовательская часть

В данном разделе будут представлены замеры времени работы реализаций алгоритмов и потребления памяти.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Время выполнения алгоритм замерялось с помощью метода clock() . Этот метод возвращает текущее время процессора как число с плавающей запятой, выраженное в секундах в Unix. . В таблице 2 содержится результат исследований.

Замер времени был выполнен со строками одинаковой длины. Длина строки – количество символов, содержащихся в этой строке.

Таблица 2 — Таблица времени выполнения алгоритмов (в наносекундах)

Длина строк	Rec	MatRec	Iter	DamIter
5	2143800	80000	60999	56999
10	10393456200	365599	188599	201799
15	None	673600	372799	454400
20	None	1398999	712599	819800
30	None	3075799	1553199	1917799
50	None	8534000	4737799	5200999
100	None	36561600	17005799	19993799
200	None	125299799	68700799	75188600

На рис. 8 - 9 представлена зависимость времени работы реализаций алгоритмов от длины строк. Для удобства ось времени работы представлена в логарифмической шкале

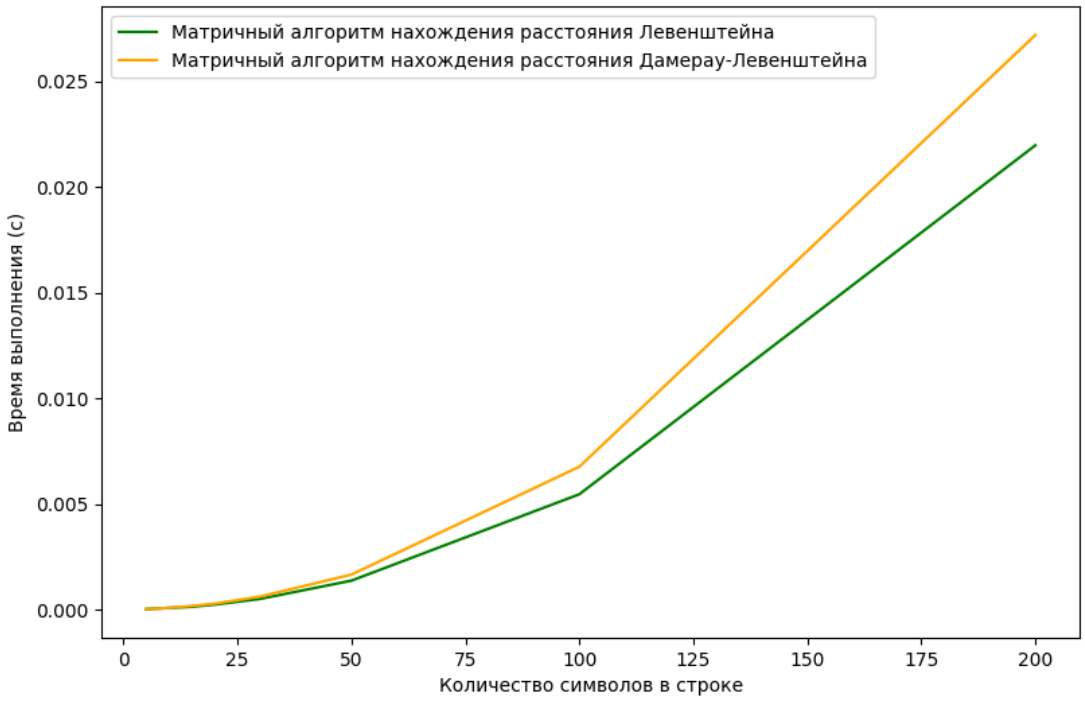

Рис. 8 — Зависимость времени работы алгоритмов от длины строк. Часть1.

4.2 Вывод

Наиболее эффективными по времени при маленькой длине слова являются рекурсивные реализации алгоритмов, но как только увеличивается длина слова, их эффективность резко снижается, что обусловлено большим количеством повторных рассчетов. Время работы алгоритма, использующего матрицу, намного меньше благодаря тому, что в нем требуется только (m + 1)*(n + 1) операций заполнения ячейки матрицы. Также установлено, что алгоритм Дамерау-Левенштейна работает немного дольше алгоритма Левенштейна, т.к. в нем добавлены дополнительные проверки, однако алгоритмы сравнимы по временной эффективности.

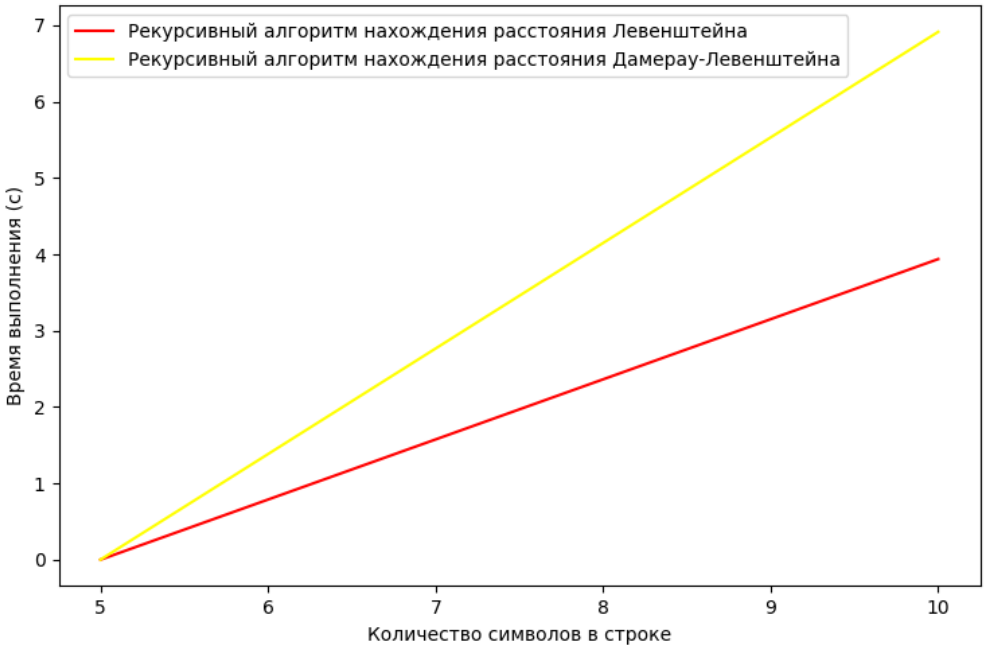

Рис. 9 — Зависимость времени работы алгоритмов от длины строк. Часть2.

Заключение

Был изучен метод динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна. Также изучены алгоритмы Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками, получены практические навыки раелизации указанных алгоритмов в матричной и рекурсивных версиях.

Экспериментально было подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработаного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк.

В результате исследований можно сделать вывод, что матричная реализация данных алгоритмов заметно выигрывает по времени при росте длины строк, следовательно более применима в реальных проектах.

При выполнение данной лабораторной работы была выполнена цель и достигнуты следующие задачи:

— были изучены алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
— были применены методы динамического программирования для матричной реализации указанных алгоритмов;
— были получены практические навыки реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
— были проведен сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);
— было экспериментально подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разрабо-

танного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк;

— были описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Список литературы

1. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. 845-848.
2. Марк Лутц . Изучаем Python. O'REILLY, 2019. 731-732.
3. Васильев. А. Н. Python на примерах. Практический курс по программированию. СПб.: Наука и Техника, 2016. 175-214.
4. Лучано Рамальо. Python. К вершинам мастерства. O'REILLY,, 2016. 44 - 68.
5. Даг Хеллман. Стандартная библиотека Python 3. Компьютерное издательство "Диалектика", 2019. 237-247.