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 Иммореева М.А.

Группа ИУ7-52Б

Оценка (баллы) ________________

Преподаватель Строганов Д.В.

Москва — 2023 г.

Оглавление

Введение

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) — метрика, измеряющая разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций (вставки, удаления, замены), необходимых для превращения одной строки в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены. Широко используется в теории информации и компьютерной лингвистике.

Расстояние Левенштейна и его обобщения активно применяются для:

- исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи);
- сравнения текстовых файлов утилитой `diff` и ей подобными (здесь роль «символов» играют строки, а роль «строк» — файлы);
- в биоинформатике для сравнения генов;

Расстояние Дамерау — Левенштейна (названо в честь учёных Фредерика Дамерау и Владимира Левенштейна) — это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна, так как к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов.

1 Аналитическая часть

1.1 Цель и задачи

Целью данной лабораторной работы является: Изучение метода динамического программирования на материале расстояний Левенштейна и Дамерау-Левенштейна. Для достижения цели следует поставить следующщте задачи:

1. изучение алгоритмов Левенштейна, Дамерау-Левенштейна нахождения расстояния между строками;
2. разработка и реализация алгоритмов поиска расстояний Левенштейна в форме: матричной; Дамерау-Левенштейна в формах: матричной, рекурсивной, рекурсивной с кешем;
3. сравнительный анализ алгоритмов определения расстояния между строками по затрачиваемому времени, памяти;
4. выполнить замеры процессорного времени работы реализаций алгоритмов;
5. провести анализ полученных результатов в отчете.

Далее этом разделе будут представлены описания алгоритмов нахождения расстояний Левенштейна, Дамерау-Левенштейна и их практическое применение.

1.2 Алгоритм нахождения расстояния Левенштейна

Расстояние Левенштейна - это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления и замены, необходимых для перевода одной строки в другую.

Расстояние Левенштейна может быть найдено по формуле 1.1, которая задана как

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), \quad \text{если } \min(i,j) = 0, \\ \min\{ \\ \qquad d_{a,b}(i, j-1) + 1, \\ \qquad d_{a,b}(i-1, j) + 1, \\ \qquad d_{a,b}(i-1, j-1) + m(a[i], b[j]) \\ \} \end{cases}, \tag{1.1}$$

1.3 Алгоритм нахождения расстояния Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна - это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов.

Расстояние Дамерау — Левенштейна может быть найдено по формуле

1.2, которая задана как

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), \quad \text{если } \min(i,j) = 0, \\ \min\{ \\ \qquad d_{a,b}(i,j-1) + 1, \\ \qquad d_{a,b}(i-1,j) + 1, \\ \qquad d_{a,b}(i-1,j-1) + m(a[i],b[j]), \quad \text{иначе} \\ \qquad \left[\begin{array}{ll} d_{a,b}(i-2,j-2) + 1, & \text{если } i,j > 1; \\ & a[i] = b[j-1]; \\ & b[j] = a[i-1] \\ \infty, & \text{иначе} \end{array} \right. \\ \} \end{cases}, \tag{1.2}$$

1.4 Рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна

Рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна выполняет прогон для данных, которые еще не были обработаны. Строится дерево из сравняемых букв, и ,доходя до максимальной глубины, запрос возвращается рекурсивно, выбирая минимальное из возможных преобразований. Результат нахождения расстояния возвращается рекурсивно к исходному запросу.

1.5 Рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна с использованием кеша

Рекурсивный алгоритм заполнения можно оптимизировать по времени выполнения с использованием кеша. В качестве кеша используется матрица. Суть данной оптимизации заключается в последовательном заполнении матрицы при выполнении рекурсии. В случае, если рекурсивный алгоритм выполняет прогон для данных, которые еще не были обработаны, результат нахождения расстояния заносится в матрицу. В случае, если обработанные ранее данные встречаются снова, для них расстояние не находится и алгоритм переходит к следующему шагу.

Вывод

Формулы Левенштейна, Дамерау — Левенштейна для рассчета расстояния между строками задаются рекурентно, а следовательно, алгоритмы могут быть реализованы рекурсивно или итерационно.

2 Конструкторская часть

В этом разделе будут приведены требования к вводу и программе, а также схемы алгоритмов нахождения расстояний Левенштейна, Дамерау-Левенштейна.

2.1 Алгоритмы нахождения расстояния Левенштейна, Дамерау — Левенштейна

На рисунке 2.1 приведена схема рекурсивного алгоритма нахождения расстояния Дамерау – Левенштейна. На рисунке 2.2 приведена схема матричного алгоритма нахождения расстояния Левенштейна. На рисунках 2.3, 2.4 приведена схема рекурсивного алгоритма нахождения расстояния Дамерау – Левенштейна с кешем. На рисунке 2.5 приведена схема матричного алгоритма нахождения расстояния Дамерау – Левенштейна.

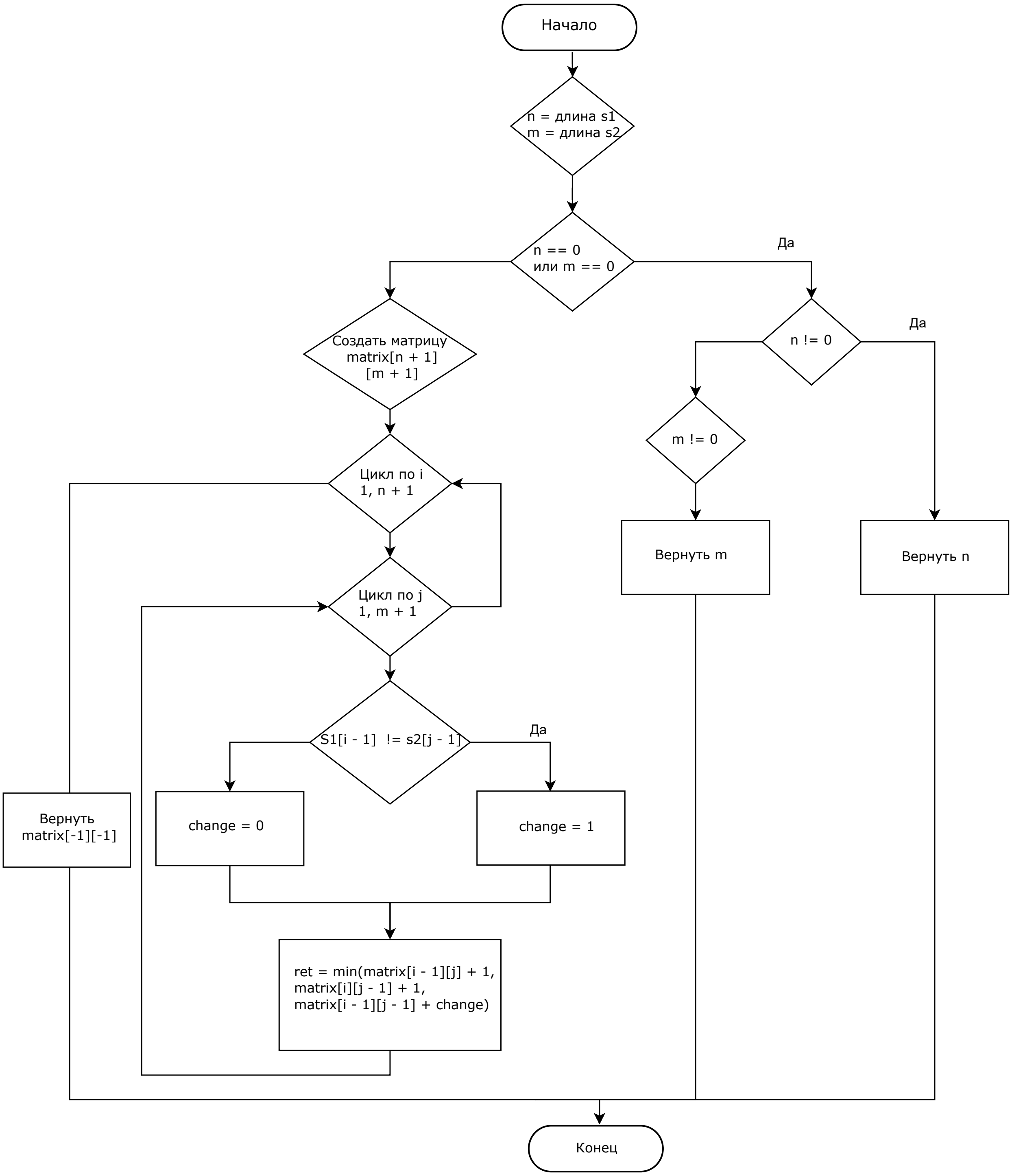

Рисунок 2.1 – Схема матричного алгоритма нахождения расстояния Левенштейна

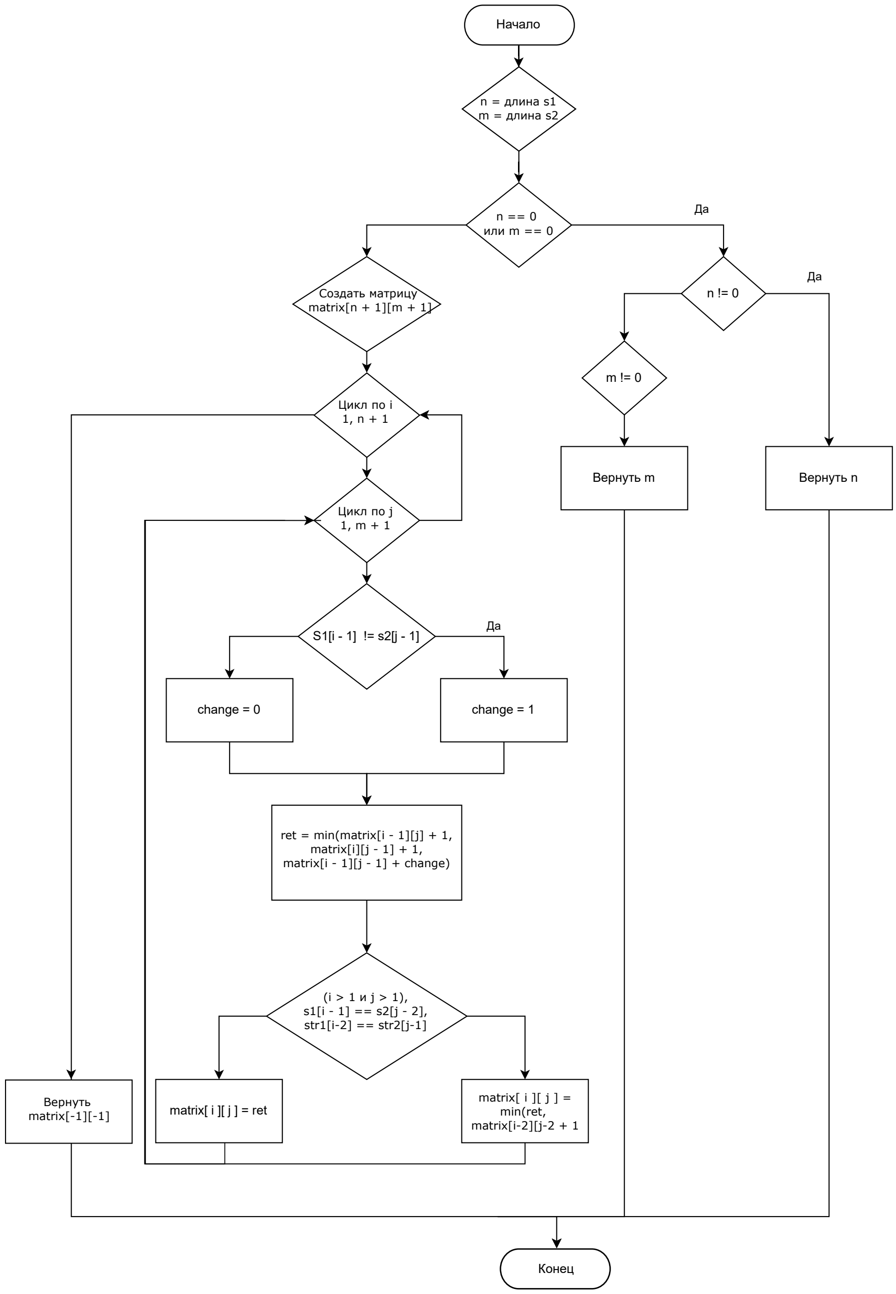

Рисунок 2.2 – Схема матричного алгоритма нахождения расстояния Дамерау–Левенштейна

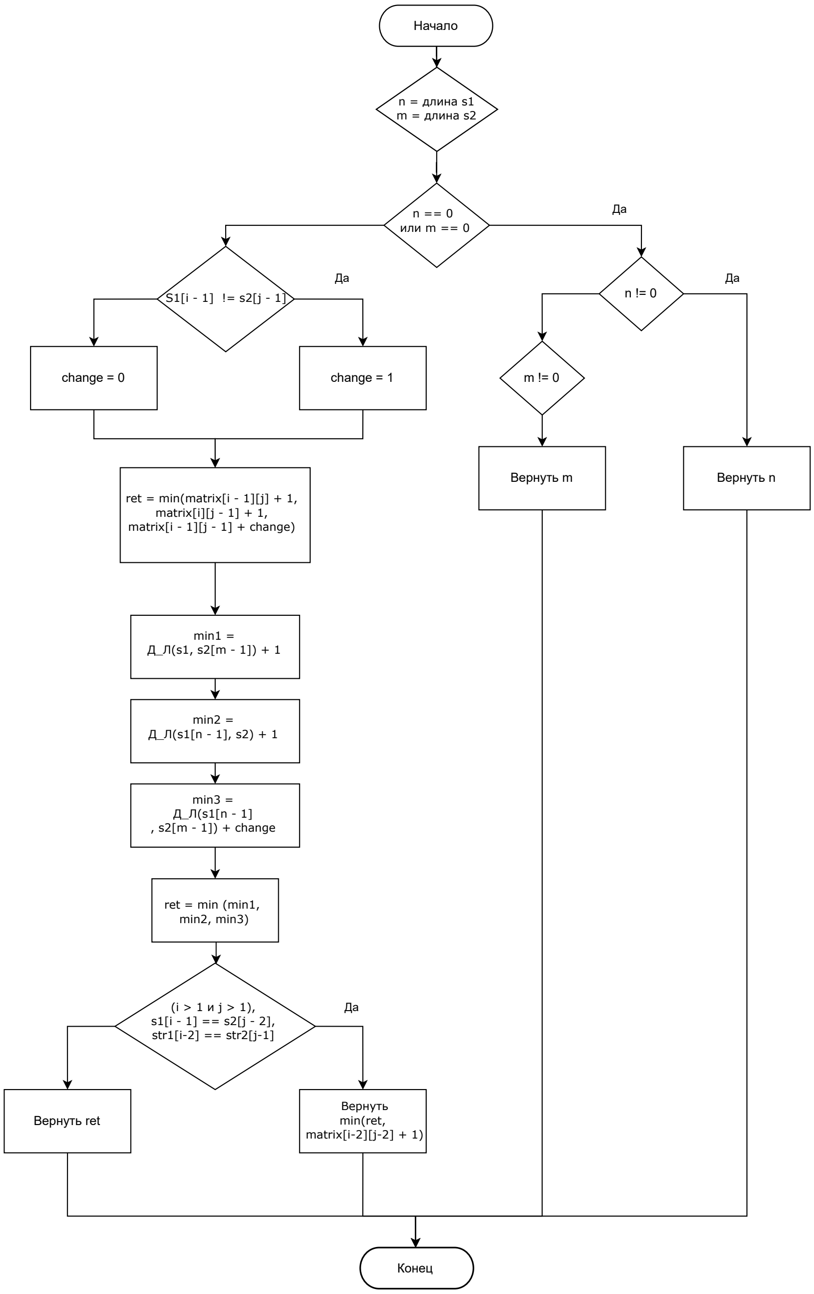

Рисунок 2.3 – Схема рекурсивного алгоритма нахождения расстояния Дамерау–Левенштейна с кешем

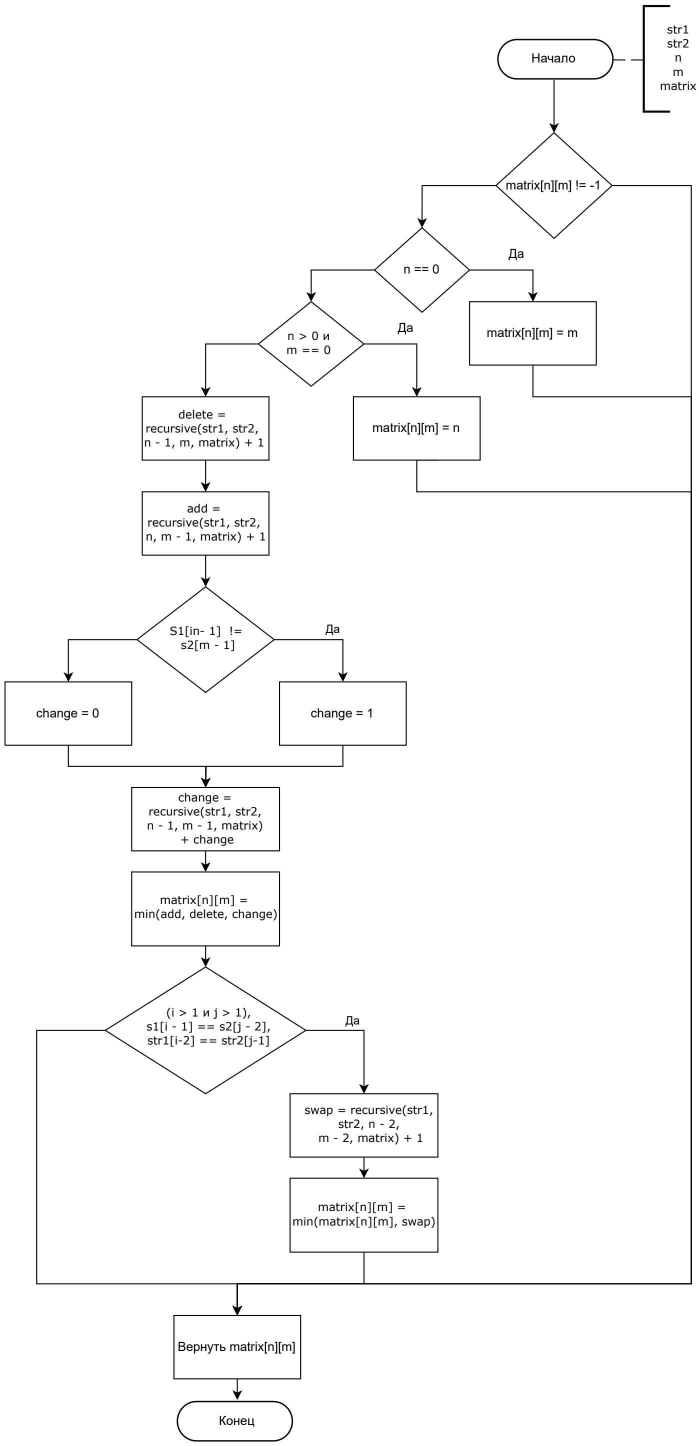

Рисунок 2.4 – Схема функции recursive алгоритма нахождения расстояния Дамерау–Левенштейна с кешем

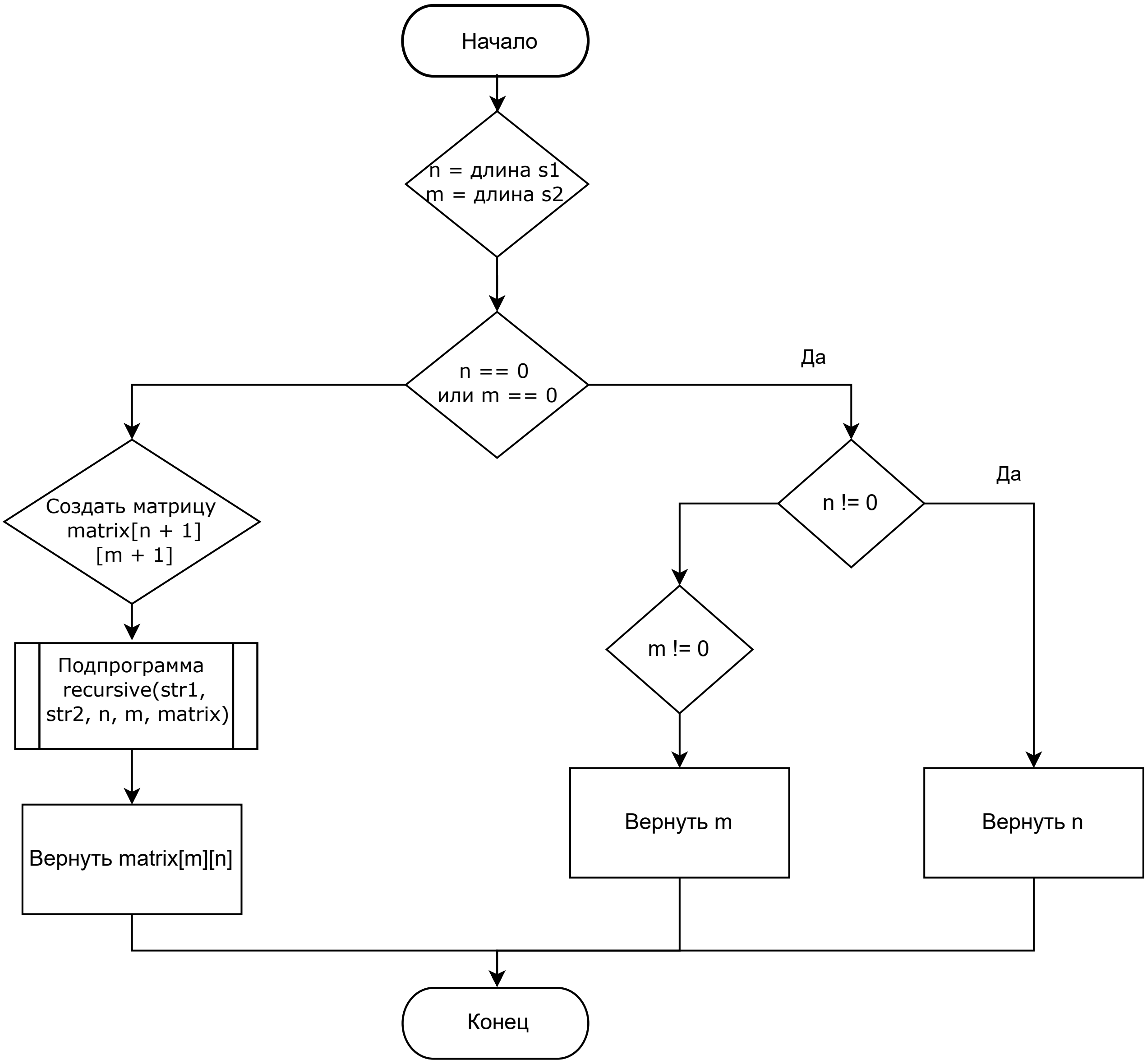

Рисунок 2.5 – Схема рекурсивного алгоритма нахождения расстояния с кешем Дамерау–Левенштейна

Вывод

Перечислены требования к вводу и программе, а также на основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к вводу

К вводу программы прилагаются данные требования:

1. На вход подаются две строки.
2. Буквы верхнего и нижнего регистров считаются различными.
3. Допускается ввод пустых строк.

3.2 Требования к программе

К программе прилагаются данные требования:

1. Две пустые строки - корректный ввод, программа не должна аварийно завершаться.
2. На выход программа должна вывести число – расстояние Дамерау-Левенштейна, матрицу при необходимости.
3. Программа позволяет тестировать каждый метод поиска расстояния Левенштейна, Дамерау–Левенштейна отдельно или все алгоритмы вместе.

3.3 Требования к программному обеспечению

К программе предъявляется ряд требований:

- у пользователя есть выбор алгоритма, или какой-то один, или все сразу, а также есть выбор тестирования времени;

- на вход подаются две строки на русском или английском языке в любом регистре;
- на выходе — искомое расстояние для выбранного метода (выбранных методов) и матрицы расстояний для матричных реализаций.

3.4 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python.

Данный язык достаточно распространен и гибок в использовании.

Время работы алгоритмов было замерено с помощью функции process_time() из библиотеки time.

3.5 Сведения о модулях программы

Программа состоит из двух модулей:

1. main.py - главный файл программы, в котором располагается меню;
2. funcs.py - файл с дополнительными функциями, использующимися главными алгоритмами;
3. algs.py - файл со всеми алгоритмами, использующимися в программе;
4. test.py - файл с замерами времени для графического изображения результата.

3.6 Код программы

В листингах 3.1, 3.2, 3.3, 3.4 приведены реализации алгоритмов нахождения расстояния Дамерау–Левенштейна.

Листинг 3.1 – Функция нахождения расстояния Левенштейна с использованием матрицы.

```
def levenshtein_matrix(str1, str2, output = False):
n = len(str1)
m = len(str2)

if n == 0 or m == 0:
if n != 0:
return n
if m != 0:
return m
return 0

matrix = create_matrix(n + 1, m + 1)

for i in range(1, n + 1):
for j in range(1, m + 1):
change = 0
if (str1[i - 1] != str2[j - 1]):
change += 1

matrix[i][j] = min(matrix[i - 1][j] + 1, matrix[i][j - 1] +
    1, matrix[i - 1][j - 1] + change)

if output:
print_matrix(str1, str2, matrix)
return(matrix[-1][-1])
```

Листинг 3.2 – Функция нахождения расстояния Дамерау-Левенштейна с использованием рекурсии.

```
def damerau_levenshtein_recursive(str1, str2, out_put = False):
n = len(str1)
m = len(str2)

if n == 0 or m == 0:
if n != 0:
return n
if m != 0:
return m
return 0

change = 0
if str1[-1] != str2[-1]:
change += 1

min_ret = min(damerau_levenshtein_recursive(str1[:n - 1], str2) + 1,
damerau_levenshtein_recursive(str1, str2[:m - 1]) + 1,
damerau_levenshtein_recursive(str1[:n - 1], str2[:m - 1]) + change)
if n > 1 and m > 1 and str1[-1] == str2[-2] and str1[-2] == str2[-1]:
min_ret = min(min_ret,
damerau_levenshtein_recursive(str1[:n - 2], str2[:m - 2]) + 1)

return min_ret
```

Листинг 3.3 – Функция нахождения расстояния Дамерау-Левенштейна с использованием матрицы.

```
def damerau_levenshtein_matrix(str1, str2, output = False):
n = len(str1)
m = len(str2)

if n == 0 or m == 0:
if n != 0:
return n
if m != 0:
return m
return 0

matrix = create_matrix(n + 1, m + 1)

for i in range(1, n + 1):
for j in range(1, m + 1):
change = 0
if (str1[i - 1] != str2[j - 1]):
change += 1

matrix[i][j] = min(matrix[i - 1][j] + 1, matrix[i][j - 1] + 1, matrix[i - 1][j - 1] + change)

if (i > 1 and j > 1) and str1[i-1] == str2[j-2] and str1[i-2] == str2[j-1]:
matrix[i][j] = min(matrix[i][j], matrix[i-2][j-2] + 1)
if output:
print_matrix(str1, str2, matrix)
return(matrix[-1][-1])
```

Листинг 3.4 – Функция нахождения расстояния Дамерау-Левенштейна с использованием рекурсии и кэша.

```
def recursive(str1, str2, n, m, matrix):
if (matrix[n][m] != -1):
return matrix[n][m]

if (n == 0):
matrix[n][m] = m
return matrix[n][m]

if (n > 0 and m == 0):
matrix[n][m] = n
return matrix[n][m]

delete = recursive(str1, str2, n - 1, m, matrix) + 1
add = recursive(str1, str2, n, m - 1, matrix) + 1
change = 0

if (str1[n - 1] != str2[m - 1]):
change = 1

change = recursive(str1, str2, n - 1, m - 1, matrix) + change

matrix[n][m] = min(add, delete, change)
if n > 1 and m > 1 and str1[n - 1] == str2[m - 2] and str1[n
    - 2] == str2[m - 1]:
swap = recursive(str1, str2, n - 2, m - 2, matrix) + 1
matrix[n][m] = min(matrix[n][m], swap)
return matrix[n][m]

def damerau_levenshtein_recursive_cash(str1, str2, output =
    False):
n = len(str1)
m = len(str2)

if n == 0 or m == 0:
if n != 0:
return n
if m != 0:
return m
return 0
```

3.7 Функциональные тесты

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

Таблица 3.1 – Функциональные тесты

№	Строка 1	Строка 2	Ожидаемый результат	матричный Левенштейн	матричный Дамерау-Левенштейн	рекурсивный Дамерау–Левенштейн	кеш Дамерау–Левенштейн
1	скат	кот	2	2	2	2	2
2	видео	вадео	1	1	1	1	1
3	картина	тюна	4	4	4	4	4
4	-	робот	5	5	5	5	5
5	мир	-	3	3	3	3	3
8	Срок	срок	1	1	1	1	1
9	мера	мероприятие	8	8	8	8	8
10	studio	stand	4	4	4	4	4

Вывод

Были разработаны и протестированы алгоритмы: нахождения расстояния Левенштейна матрично, Дамерау - Левенштейна рекурсивно, с заполнением матрицы и рекурсивно с кешем.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: 64-разрядная операционная система, процессор x64.
- Память: 16 Гб.
- Процессор: Intel(R) Core(TM) i7–4700HQ CPU @ 2.40 ГГц.

Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Алгоритмы тестировались при помощи функции process_time() из библиотеки time языка Python. Данная функция возвращает сумму системного и пользовательского процессорного времени текущего процессора, типа float в секундах.

Замеры времени для каждой длины слов проводились 100 раз. В качестве результата взято среднее время работы алгоритма.

Результаты замеров приведены в таблице 4.1 (время в мкс).

Таблица 4.1 – Результаты замеров времени.

Длина	Левенштейн (матрица)	Дамерау–Левенштейн (матрица)	Дамерау–Левенштейн (рекурсия)	Дамерау–Левенштейн (рекурсия + кеш)
0	0.000000	0.000000	0.000000	0.000000
1	0.000000	0.000000	0.000000	0.000000
2	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	312.5	0.000000
5	0.000000	0.000000	1250.0	156.25
6	0.000000	0.000000	6093.75	156.25
7	156.25	132.745	32656.25	312.5

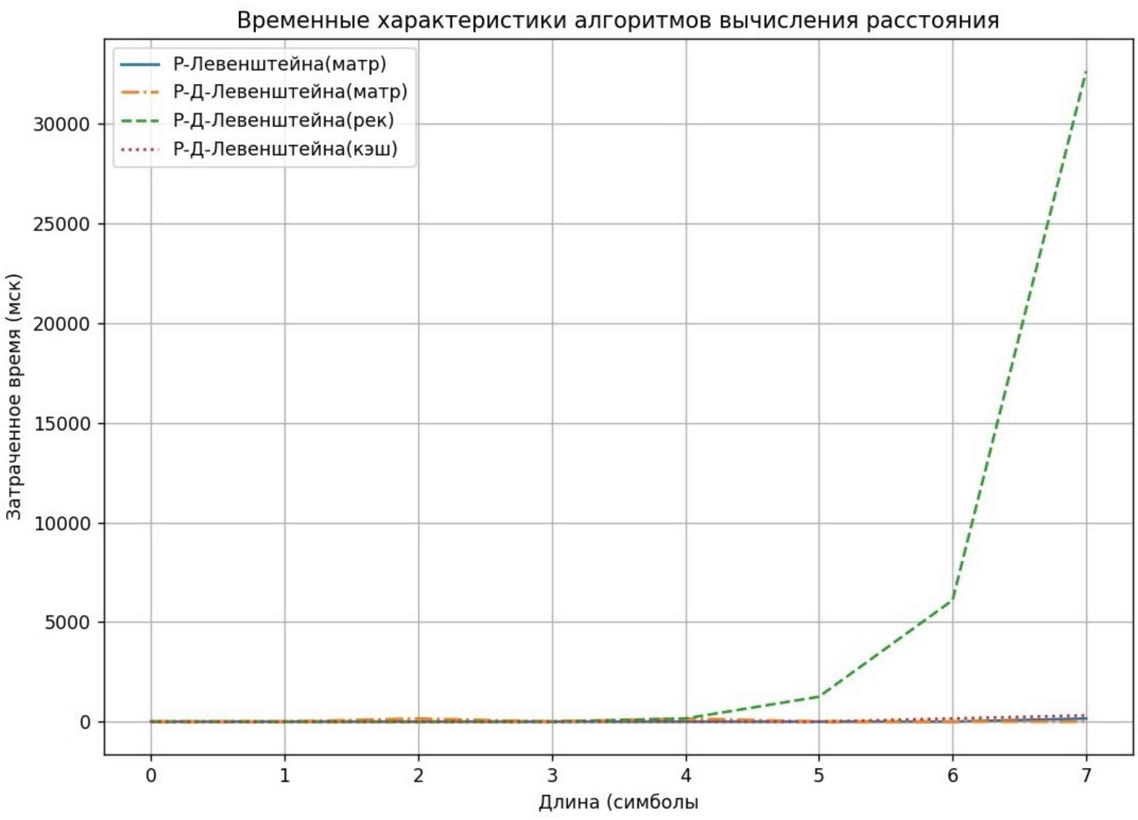

Рисунок 4.1 – Временные затраты алгоритмов

4.3 Использование памяти

Пусть длина строки S1 – n, длина строки S2 – m, тогда затраты памяти на приведенные выше алгоритмы будут следующими:

- матричный алгоритм поиска расстояния Левенштейна, Дамерау – Левенштейна:
  - строки S1, S2 - (m + n) * sizeof(char)
  - матрица – ((m + 1) * (n + 1)) * sizeof(int)
  - длины строк – 2 * sizeof(int)
  - вспомогательные переменные – 3 * sizeof(int)

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящий строк.

- рекурсивный алгоритм поиска расстояния Дамерау – Левенштейна(для каждого вызова):
  - строки S1, S2 – (m + n) * sizeof(char)
  - длины строк – 2 * sizeof(int)
  - вспомогательная переменная - sizeof(int)
  - адрес возврата
- рекурсивный алгоритм поиска расстояния Дамерау – Левенштейна с использованием кеша (для каждого вызова): Для всех вызовов еще память для хранения самой матрицы - ((m + 1) * (n + 1)) * sizeof(int)
  - строки S1, S2 – (m + n) * sizeof(char)
  - длины строк – 2 * sizeof(int)
  - вспомогательные переменные – 5 * sizeof(int)
  - ссылка на матрицу – 8 байт
  - адрес возврата

Вывод

Рекурсивный алгоритм нахождения расстояния Дамерау - Левенштейна работает на порядок дольше итеративных реализаций, время его работы увеличивается в геометрической прогрессии. На словах длиной 7 символов, матричная реализация алгоритма нахождения расстояния Левенштейна превосходит по времени работы рекурсивную на несколько порядков.

Рекурсивный алгоритм с заполнением матрицы превосходит простой рекурсивный и сравним по времени работы с матричными алгоритмами.

Алгоритм нахождения расстояния Дамерау — Левенштейна по времени выполнения сопоставим с алгоритмом нахождения расстояния Левенштейна. В нём добавлена дополнительная проверка, позволяющая находить ошибки пользователя, связанные с неверным порядком букв, в связи с чем он работает незначительно дольше, чем алгоритм нахождения расстояния Левенштейна.

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

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- изучены алгоритмы Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
- разработаны и реализованы алгоритмы поиска расстояний Левенштейна, Дамерау-Левенштейна;
- проведен сравнительный анализ алгоритмов определения расстояния между строками по затрачиваемому времени, памяти;
- выполнены замеры процессорного времени работы реализаций алгоритмов;
- проведены анализы полученных результатов.

Экспериментально было подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранных алгоритов на материале замеров процессорного времени.

В результате исследований можно прийти к выводу, что матричная реализация алгоритмов нахождения расстояний заметно выигрывает по времени при росте строк, но проигрывает по количеству затрачиваемой памяти.

Список использованных источников

[1] Welcome to Python [Электронный ресурс]. Режим доступа:: https://www.python.org – (Дата обращения: 18.09.2023),

[2] Кеш – Википедия [Электронный ресурс]. Режим доступа:: https://ru.wikipedia.org/wiki/Кэш – (Дата обращения: 18.09.2023),

[3] Damerau-Levenshtein distance and strings [Электронный ресурс]. Режим доступа: https://docs.stack-assessment.org/en/Topics/Levenshtein_distance – (Дата обращения: 18.09.2023),

[4] Алгоритм Левенштейна [Электронный ресурс]. Режим доступа: https://ru.hexlet.io/courses/algorithms-graphs/lessons/levenshtein-distance/theory_unit – (Дата обращения: 18.09.2023);