Государственное образовательное учреждение высшего профессионального образования

"Московский государственный технический университет имени Н.Э.Баумана"

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №1
курса «Анализ алгоритмов»
на тему «Расстояния Левенштейна и Дамерау — Левенштейна»

Студент: Ильясов И.М., группа ИУ7-53Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Содержание

Введение

В современном мире ежедневно люди сталкиваются с вводом текста в Интернете при поиске какой-либо информации, при общении с друзьями и знакомыми в социальных сетях и мессенджерах. При этом многие допускают ошибки в правописании различных слов. Для исправления этих ошибок используются алгоритмы, которые определяют различие между двумя заданными строками. Советский математик Владимир Иосифович Левенштейн для этой цели ввел понятие «редакционного расстояния» между двумя строками и разработал алгоритм нахождения этого расстояния.

Целью лабораторной работы №1 является:

- изучение алгоритмов Левенштейна и Дамерау — Левенштейна нахождения расстояния между строками;
- применение метода динамического программирования для матричной реализации указанных алгоритмов;
- получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
- сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения рас- стояния между строками по затрачиваемым ресурсам (времени и памяти);
- экспериментальное подтверждение различий 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;
- описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе содержатся теоретическое описание алгоритмов и указание области их применения.

1.1 Описание алгоритмов

Расстояние Левенштейна — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Все три операции обладают штрафом 1. Операция совпадения обладает штрафом 0.

Задача по нахождению расстояния Левенштейна заключается в поиске минимального количества операций вставки/удаления/замены для превращения одной строки в другую.

При нахождении расстояния Дамерау — Левенштейна к перечисленным выше операциям добавляется операция перестановки соседних символов (транспозиции).

Рекурсивная реализация

Пусть на вход даны две строки S_1, S_2 длины i, j соответственно над некоторым алфавитом. Алгоритм нахождения расстояния Левенштейна можно описать следующей формулой:

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

где m находится из формулы, как:

$$m(S_1[i-1],\, S_2[j-1] = \begin{cases} 0, \text{если } S_1[i-1] = S_2[j-1] \\ 1, \text{иначе} \end{cases}$$

Алгоритм Дамерау — Левенштейна является модификацией алгоритма Левенштейна. Он учитывает транспозицию — перестановку двух соседних символов местами. Штраф данной операции также равен 1.

Расстояние Дамерау — Левенштейна для строк S_1 и S_2 длины i, j соответственно можно найти по формуле:

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

где m находится из формулы, как:

$$m(S_1[i-1],\ S_2[j-1] = \begin{cases} 0, \text{если } S_1[i-1] = S_2[j-1] \\ 1, \text{иначе} \end{cases}$$

Матричная реализация

Использование матрицы для расчёта редакционного расстояния: если длины строк S_1 и S_2 равны M и N соответственно, то найти расстояние Левенштейна можно, используя матрицу размерностью $(M + 1) * (N + 1)$.

Матрицу заполняют с ячейки (0, 0). В каждой ячейке содержится количество операций над частью первой подстроки, чтобы преобразовать ее в часть второй подстроки.

2 Конструкторская часть

Алгоритмы Левенштейна и Дамерау — Левенштейна можно реализовать, как рекурсивно, так и матрично. Ниже приведены IDEF0-диаграмма алгоритма нахождения редакционного расстояния и схемы трех реализаций: Левенштейн матрично, Дамерау — Левенштейн рекурсивно и матрично.

2.1 IDEF0-диаграмма

Рисунок 1: IDEF0-диаграмма алгоритма нахождения редакционного расстояния

2.2 Разработка алгоритмов

Ниже представлены схемы алгоритмов нахождения редакционного расстояния Левенштейна в матричной реализации (рис. 2), Дамерау — Левенштейна в матричной (рис. 3) и рекурсивной (рис. 4) реализациях.

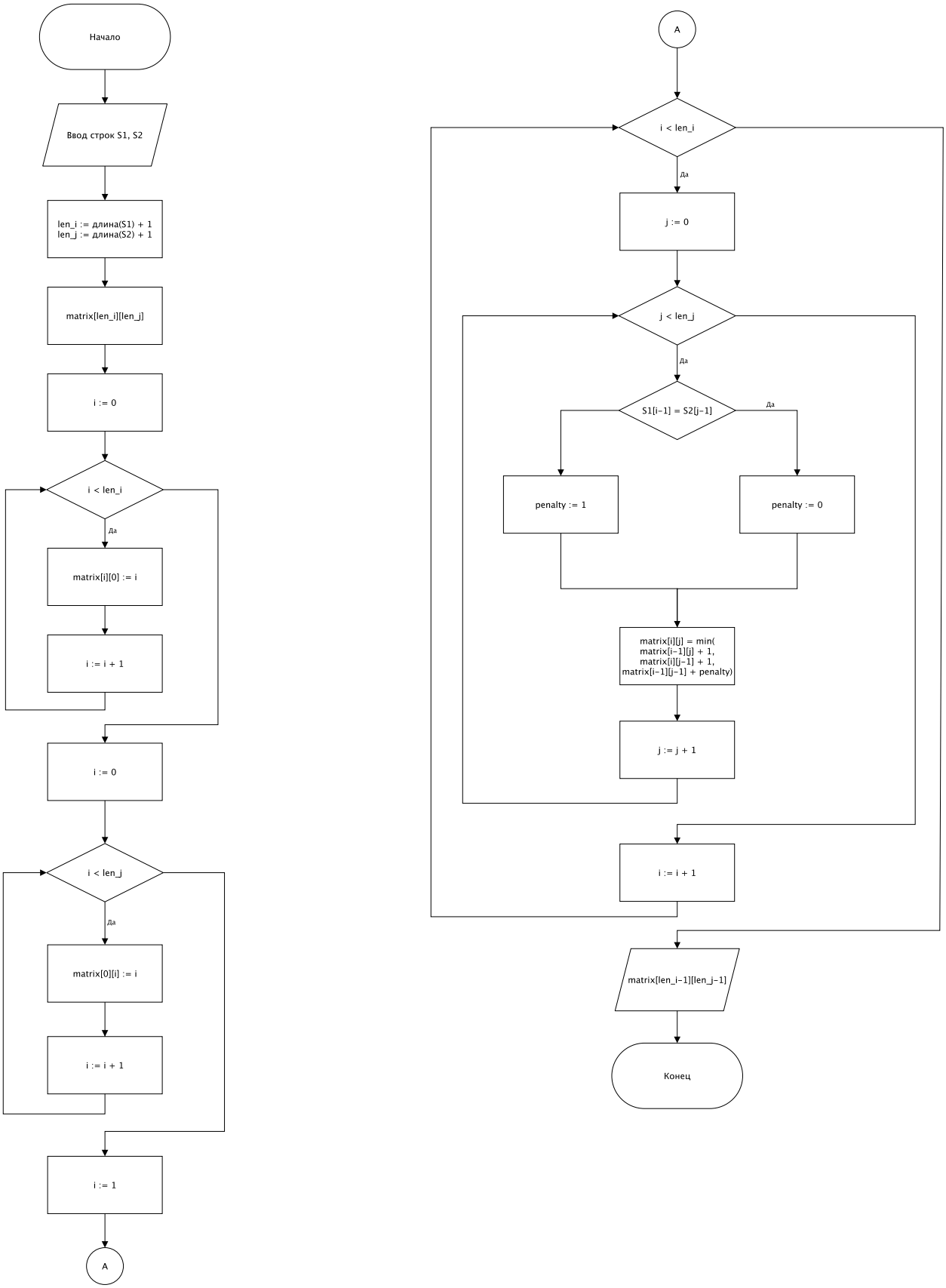

Рисунок 2: схема алгоритма нахождения редакционного расстояния Левенштейна (матрично)

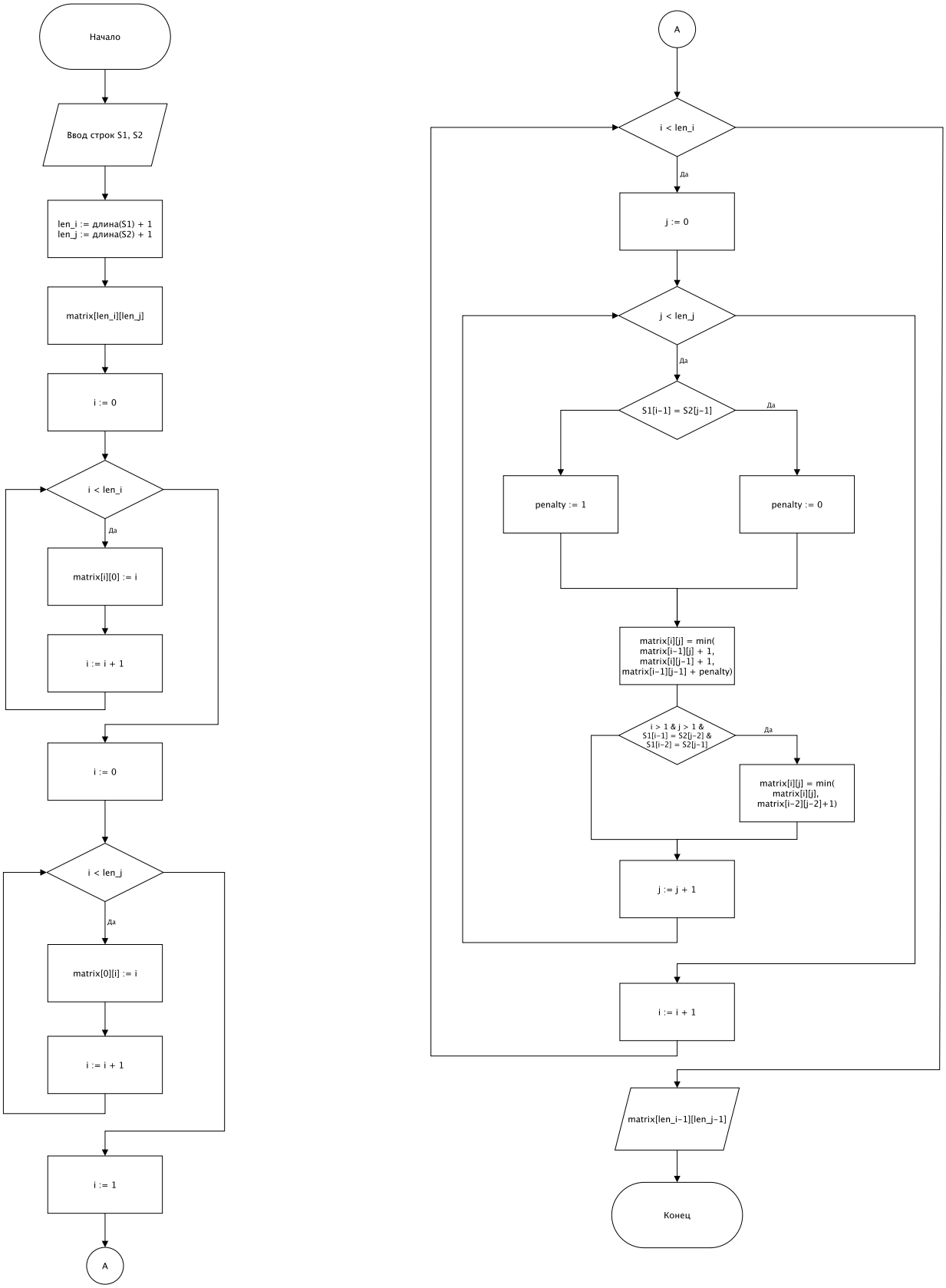

Рисунок 3: схема алгоритма нахождения редакционного расстояния Дамерау — Левенштейна (матрично)

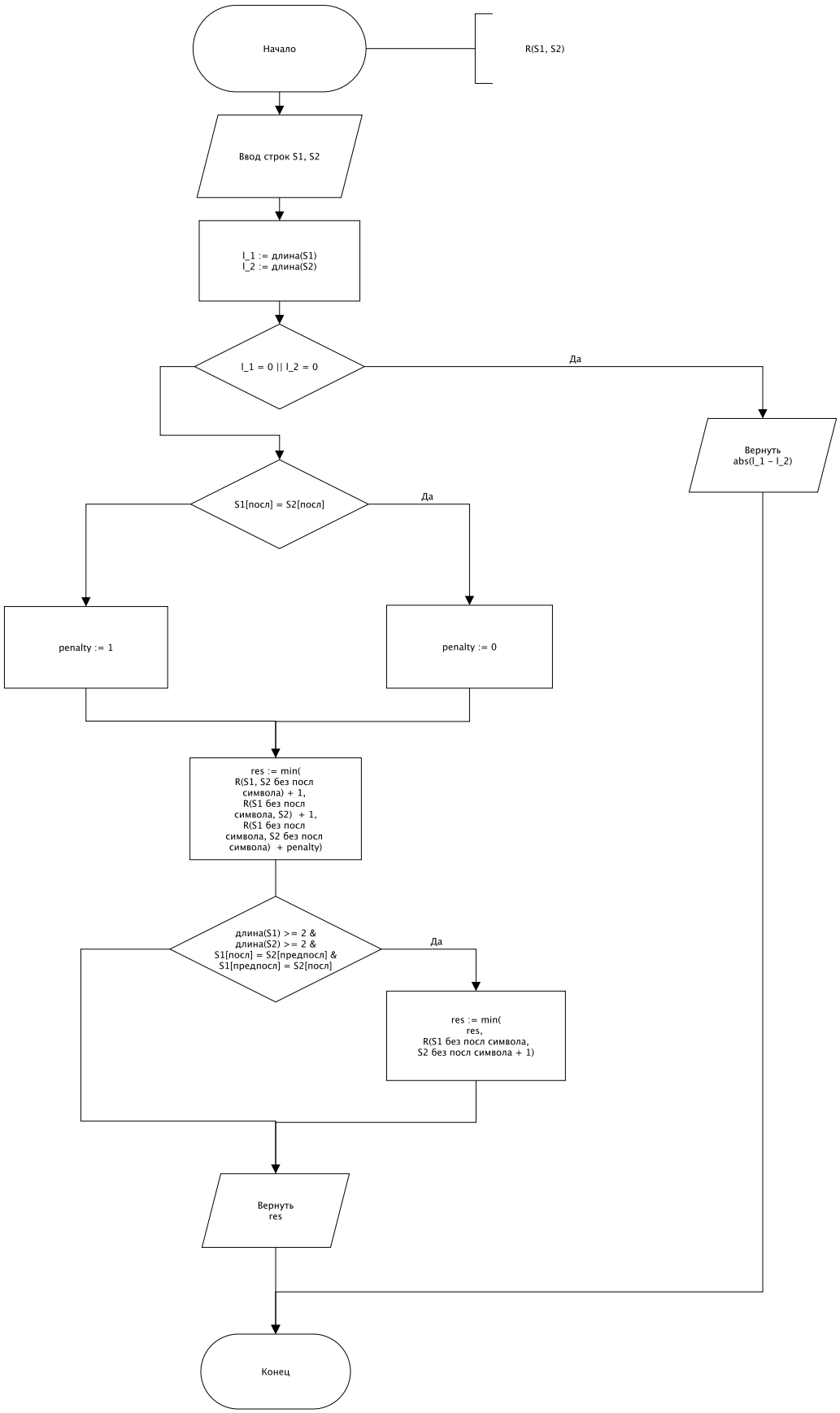

Рисунок 4: схема алгоритма нахождения редакционного расстояния Дамерау — Левенштейна (рекурсивно)

2.3 Сравнительный анализ рекурсивной и нерекурсивной реализаций алгоритма Дамерау — Левенштейна

Итеративная реализация алгоритма имеет сложность $O(m, n)$ (где m, n - длины 1 и 2 строк соответственно), а рекурсивная - $O(4^{max(m,n)})$. Другими словами, ре- курсивная реализация в несколько раз уступает матричной по времени, в чем нетрудно убедиться, проведя ряд тестов по замеру времени работы обоих упомянутых вариантов.

2.4 Вывод

В данном разделе были разработаны схемы для итерационных реализаций указанных алгоритмов и рекурсивной реализации алгоритма Дамерау — Левенштейна. Для последнего также было приведено сравнение рекурсивной и итеративной реализации, из которого можно прийти к выводу, что рекурсивная реализация алгоритма Дамерау — Левенштейна должна уступать нерекурсивной по времени и по памяти.

3 Технологическая часть

В данном разделе представлены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также листинг кода программы.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать два алгоритма нахождения расстояния между двумя строками – алгоритм Левенштейна и алгоритм Дамерау — Левенштейна, причем алгоритм Дамерау — Левенштейна должен быть реализован как рекурсивным, так и матричным способом. Пользователь должен иметь возможность проводить как единичный замер расстояния для выбранной пары строк, так и возможность сравнить скорость работы этих алгоритмов.

Разработанное ПО должно предоставлять возможность замеров процессорного времени выполнения реализации каждого алгоритма. Требуется провести замеры для варьирующихся длин строк (длины сравниваемых строк полагать одинаковыми): не менее чем от 100 до 1000 с шагом 100. Один эксперимент ставится не менее 100 раз, результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования Python. Для измерения процессорного времени была использован метод time.process_time().

3.3 Листинг кода

В приведенных ниже листингах представлены реализации алгоритма Левенштейна матрично (листинг 1) и алгоритма Дамерау — Левенштейна матрично (листинг 2) и рекурсивно (листинг 3).

Листинг 1: Расстояние Левенштейна (матричная реализация):

```
def levenshtein_matrix(string_1, string_2):
    len_i = len(string_1) + 1
    len_j = len(string_2) + 1
    matrix = [[i + j for j in range(len_j)] for i in range(len_i)]

    for i in range(1, len_i):
        for j in range(1, len_j):
            penalty = 0 if (string_1[i - 1] == string_2[j - 1]) else 1
            matrix[i][j] = min(matrix[i - 1][j] + 1,
                               matrix[i][j - 1] + 1,
                               matrix[i - 1][j - 1] + penalty)
    return(matrix[-1][-1])
```

Листинг 2: Расстояние Дамерау — Левенштейна (матричная реализация):

```
def damerau_levenshtein_matrix(string_1, string_2):
    len_i = len(string_1) + 1
    len_j = len(string_2) + 1
    matrix = [[i + j for j in range(len_j)] for i in range(len_i)]

    for i in range(1, len_i):
        for j in range(1, len_j):
            penalty = 0 if (string_1[i - 1] == string_2[j - 1]) else 1
            matrix[i][j] = min(matrix[i - 1][j] + 1,
                               matrix[i][j - 1] + 1,
                               matrix[i - 1][j - 1] + penalty)
            if (i > 1 and j > 1) and string_1[i-1] == string_2[j-2] and
string_1[i-2] == string_2[j-1]:
                matrix[i][j] = min(matrix[i][j], matrix[i-2][j-2] + 1)
    return(matrix[-1][-1])
```

Листинг 3: Расстояние Дамерау — Левенштейна (рекурсивная реализация):

```
def damerau_levenshtein_recursion(string_1, string_2):
    if string_1 ==  '' or string_2 == '':
        return abs(len(string_1) - len(string_2))
    penalty = 0 if (string_1[-1] == string_2[-1]) else 1
    res = min(damerau_levenshtein_recursion(string_1, string_2[:-1]) + 1,
              damerau_levenshtein_recursion(string_1[:-1], string_2) + 1,
              damerau_levenshtein_recursion(string_1[:-1], string_2[:-1])
+ penalty)
    if (len(string_1) >= 2 and len(string_2) >= 2 and string_1[-1] ==
string_2[-2] and string_1[-2] == string_2[-1]):
        res = min(res, damerau_levenshtein_recursion(string_1[:-2],
string_2[:-2]) + 1)
    return res
```

3.4 Вывод

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка Python было разработано программное обеспечение, содержащее реализации выбранных алгоритмов.

4 Экспериментальная часть

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

4.1 Примеры работы

На приведенных ниже рисунках продемонстрирована работа выбранных алгоритмов.

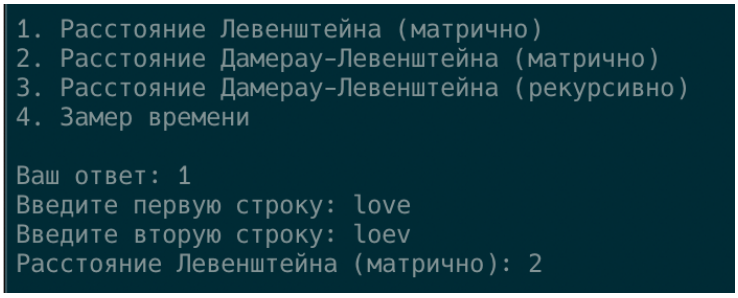

Рисунок 5: пример работы алгоритмов (Левенштейн матрично)

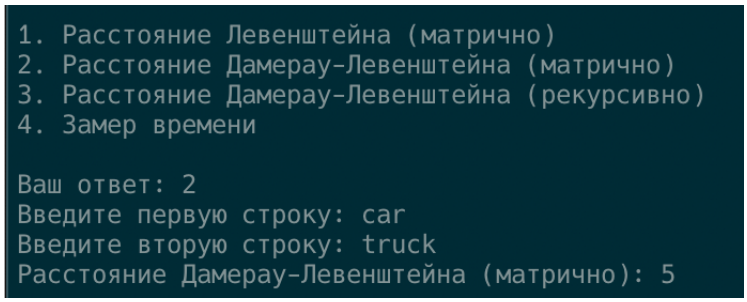

Рисунок 6: пример работы алгоритмов (Дамерау — Левенштейн матрично)

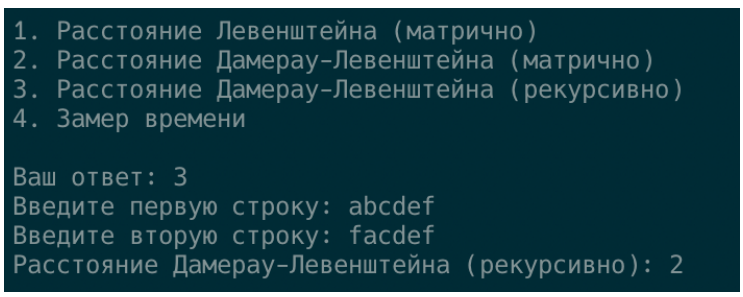

Рисунок 7: пример работы алгоритмов (Дамерау — Левенштейн рекурсивно)

4.2 Результаты тестирования

Таблица 1:

Результаты замеров времени рекурсивной реализации алгоритма Дамерау — Левенштейна

Длина строк	Время рекурсивного
1	0.0029
2	0.006
3	0.0172
4	0.0531
5	0.2473
6	1.2832
7	6.769
8	37.259
9	203.533
10	1126.851

Таблица 2:

Результаты замеров времени матричной реализации алгоритмов Дамерау — Левенштейна и Левенштейна

Длина строк	Время базового	Время модифицированного
100	1.309	1.618
200	5.135	6.411
300	11.638	14.569
400	21.627	26.909
500	35.315	43.081
600	51.524	63.319
700	70.74	87.068
800	94.351	115.728
900	120.24	147.158
1000	149.115	182.367

4.3 Постановка эксперимента по замеру времени

На рис. 8 представлены результаты замеров для варьирующихся длин строк от 100 до 1000 с шагом 100 и на рис.9 - для длин строк от 1 до 10 с шагом 1. Один эксперимент ставился 100 раз, вычислялось среднее время работы.

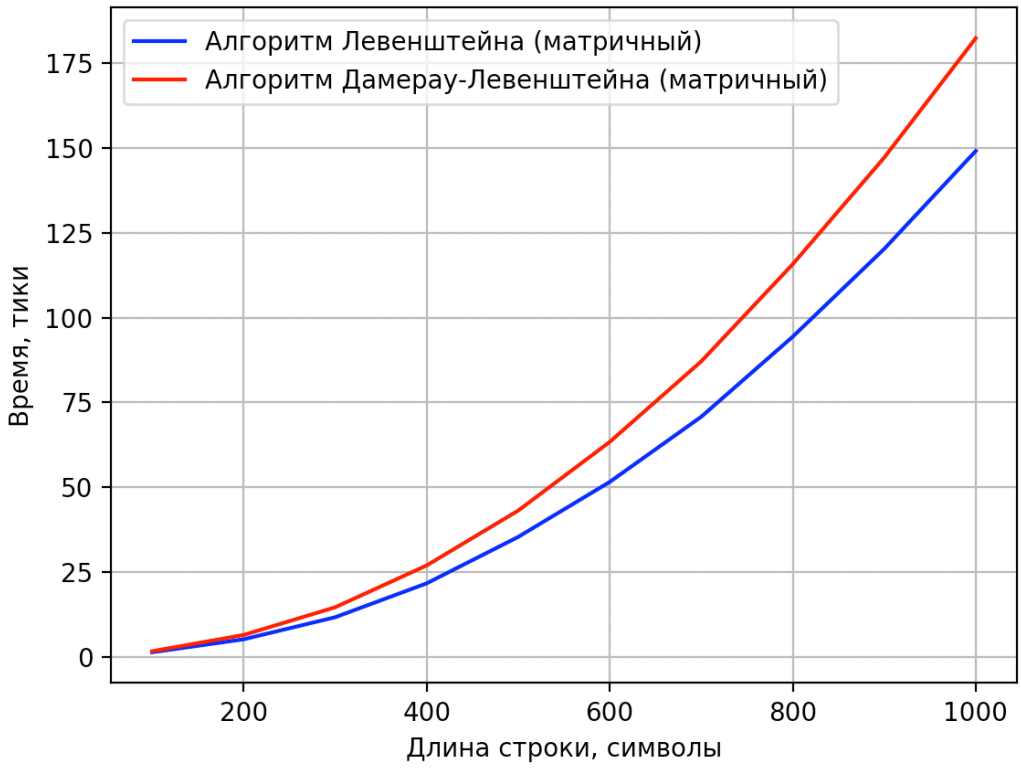

Рисунок 8: график зависимости времени работы алгоритма Левенштейна (матрично) и Дамерау—Левенштейна от длины строки.

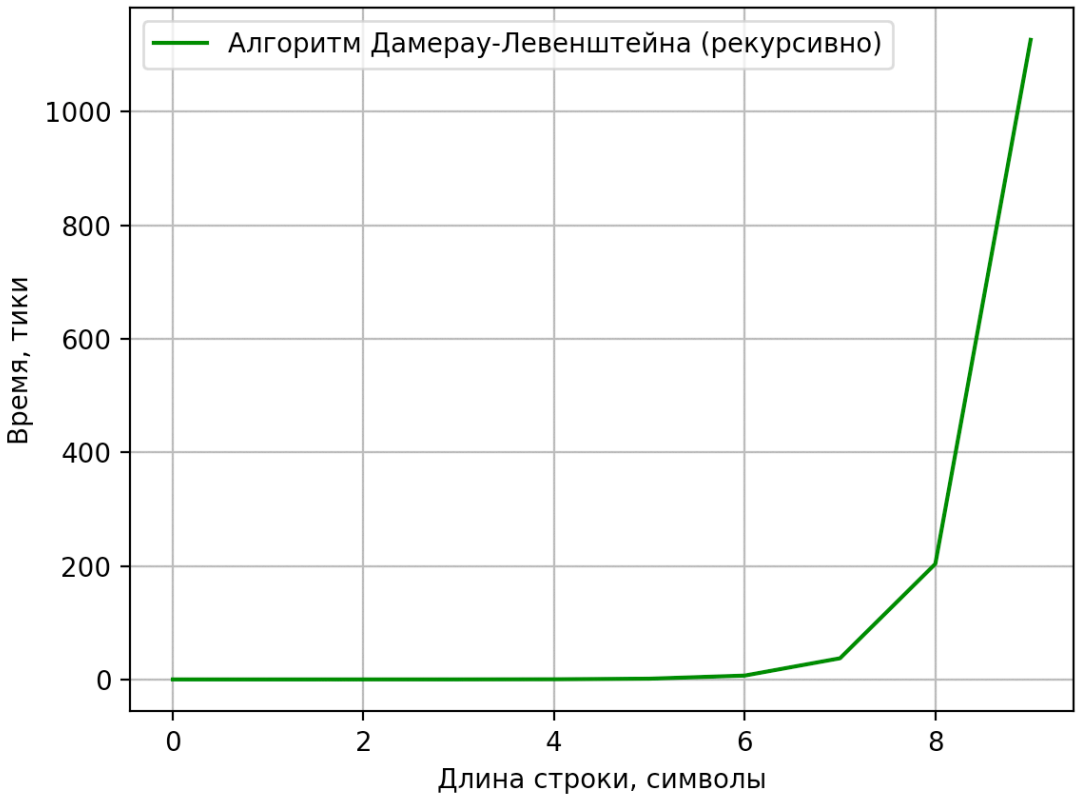

Рисунок 9: график зависимости времени работы алгоритма Дамерау — Левенштейна (рекурсивно) от длины строки.

4.4 Вывод

Время работы матричных реализаций обоих алгоритмов имеет линейную зависимость от произведения длин двух строк. Время работы рекурсивной реализация алгоритма Дамерау — Левенштейна имеет экспоненциальную зависимость, использование этой реализации непредпочтительно.

Заключение

В процессе выполнения лабораторной работы было проведено исследование алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками. Во время разработки программного обеспечения в соответствии с поставленными требованиями были получены практические навыки реализации указанных алгоритмов: алгоритмов Левенштейна и Дамерау — Левенштейна в матричной версии и алгоритма Дамерау — Левенштейна в рекурсивной версии. При помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк были экспериментально подтверждены различия во временной эффективности рекурсивной и нерекурсивной версий выбранного алгоритма определения расстояния между строками.

Список литературы

1. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965

2. Нечеткий поиск в словаре с универсальным автоматом Левенштейна. URL:https://habr.com/ru/post/275937/ (дата обращения — 12.09.2019).

3. Нечёткий поиск в тексте и словаре. URL:https://habr.com/ru/post/114997/ (дата обращения — 12.09.2019).