Министерство образования и науки 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"

Расстояние Левенштейна

Студент:
Турсунов Жасурбек Рустамович
Группа: ИУ7-56Б

Преподователи:
Волкова Лилия Леонидовна
Строганов Юрий Владимирович

Москва, 2020 г.

Оглавление

Введение

Расстояние Левенштейна - минимальное количество операций вставки, удаления, замены одного символа на другой, 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

Целью данной лабораторной работы является изучение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна.
Задачами данной лабораторной являются:

1. Изучение алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
2. Получение практических навыков реализации указанных алгоритмов;
3. Сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояний между строками по затрачиваемым ресурсам(времени);
4. Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе

Глава 1

Аналитическая часть

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 добавляется операция транспозиции(перестановки соседних символов).
Действия обозначаются так:

1. D (англ. delete) - удалить
2. I (англ. insert) - вставить
3. R (англ. replace) - заменить
4. M (англ. match) - совпадение

1.1 Алгоритм 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 Алгоритм Дамерау-Левенштейна

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

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

1.3 Вывод

Были рассмотрены поверхностно алгоритмы нахождения расстояния Левенштейна и его усовершенствованный алгоритм нахождения расстояния Дамерау-Левенштейна, принципиальная разница которого — наличие транспозиции.

Глава 2

Конструкторская часть

Требования к вводу:

1. На вход подаются две строки.

2. Строчные и заглавные буквы считаются разными.

1. Две пустые строки - корректный ввод, программа не должна аварийно завершаться.

2.1 Разработка алгоритмов

В данном разделе будут рассмотрены схемы алгоритмов:

- Табличный алгоритм Левенштейна
- Рекурсивный алгоритм Левенштейна
- Табличный алгоритм Дамерау-Левенштейна
- Рекурсивный алгоритм Дамерау-Левенштейна

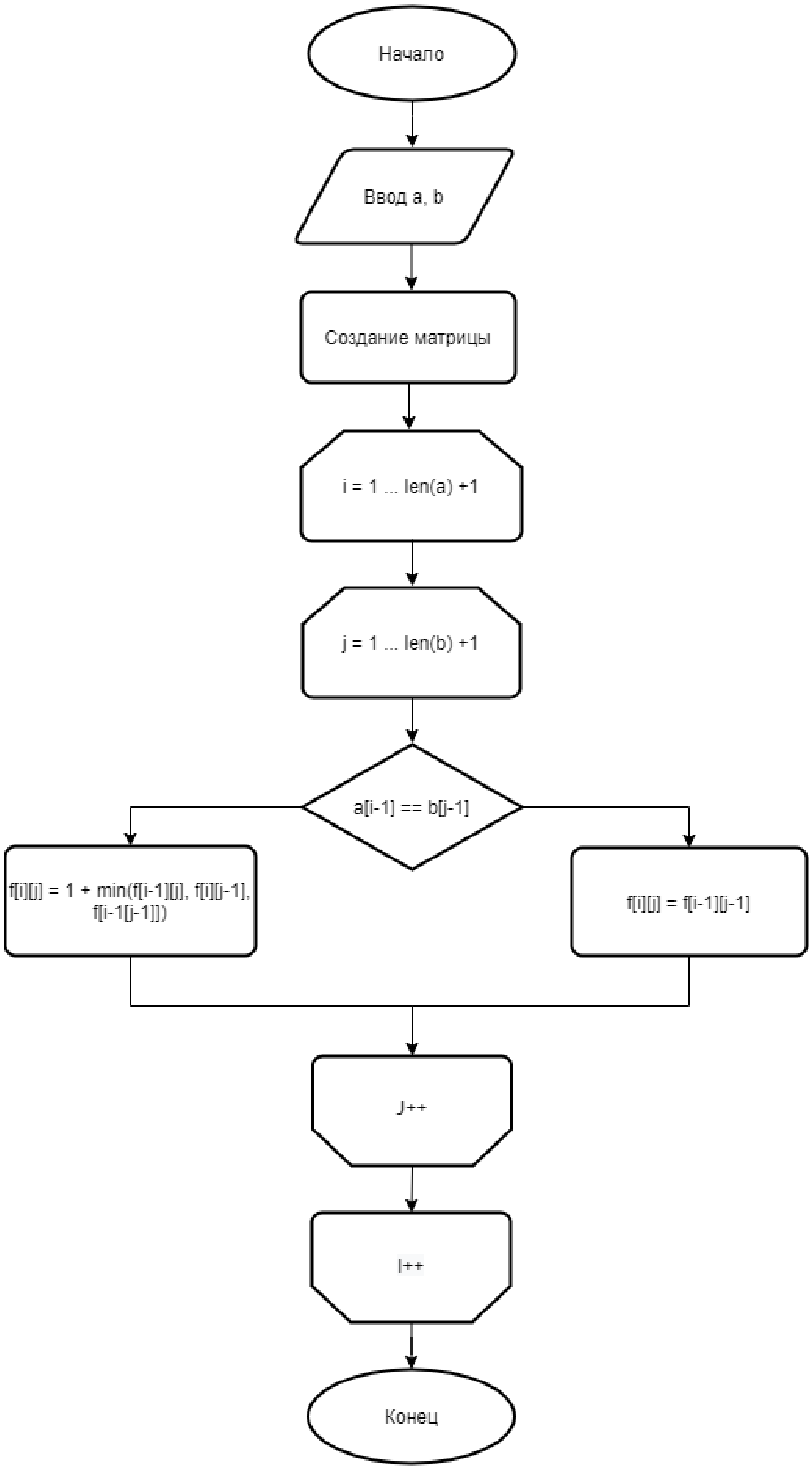

Рис. 2.1: Схема матричного алгоритма нахождения расстояния Левенштейна

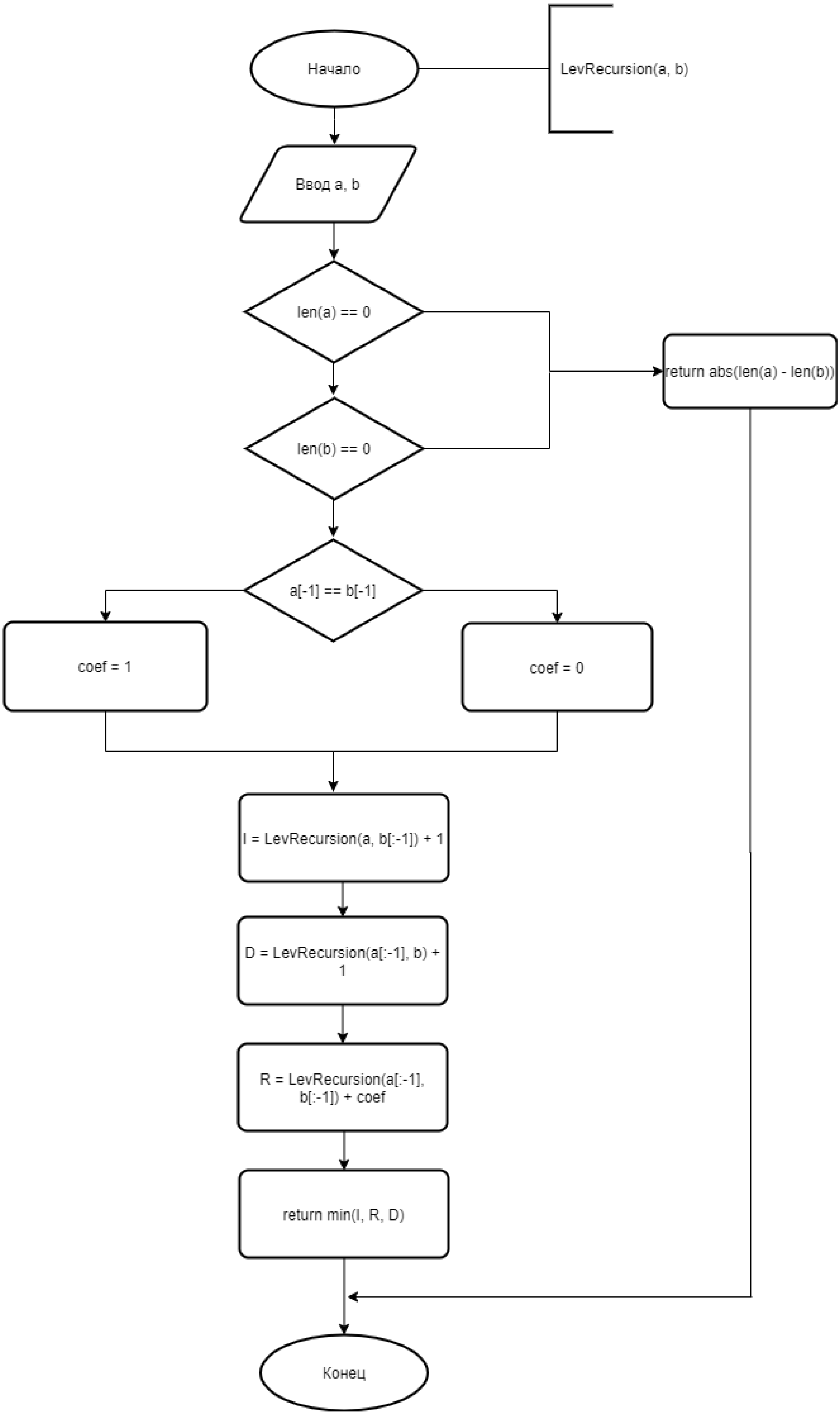

Рис. 2.2: Схема рекурсивного алгоритма нахождения расстояния Левенштейна

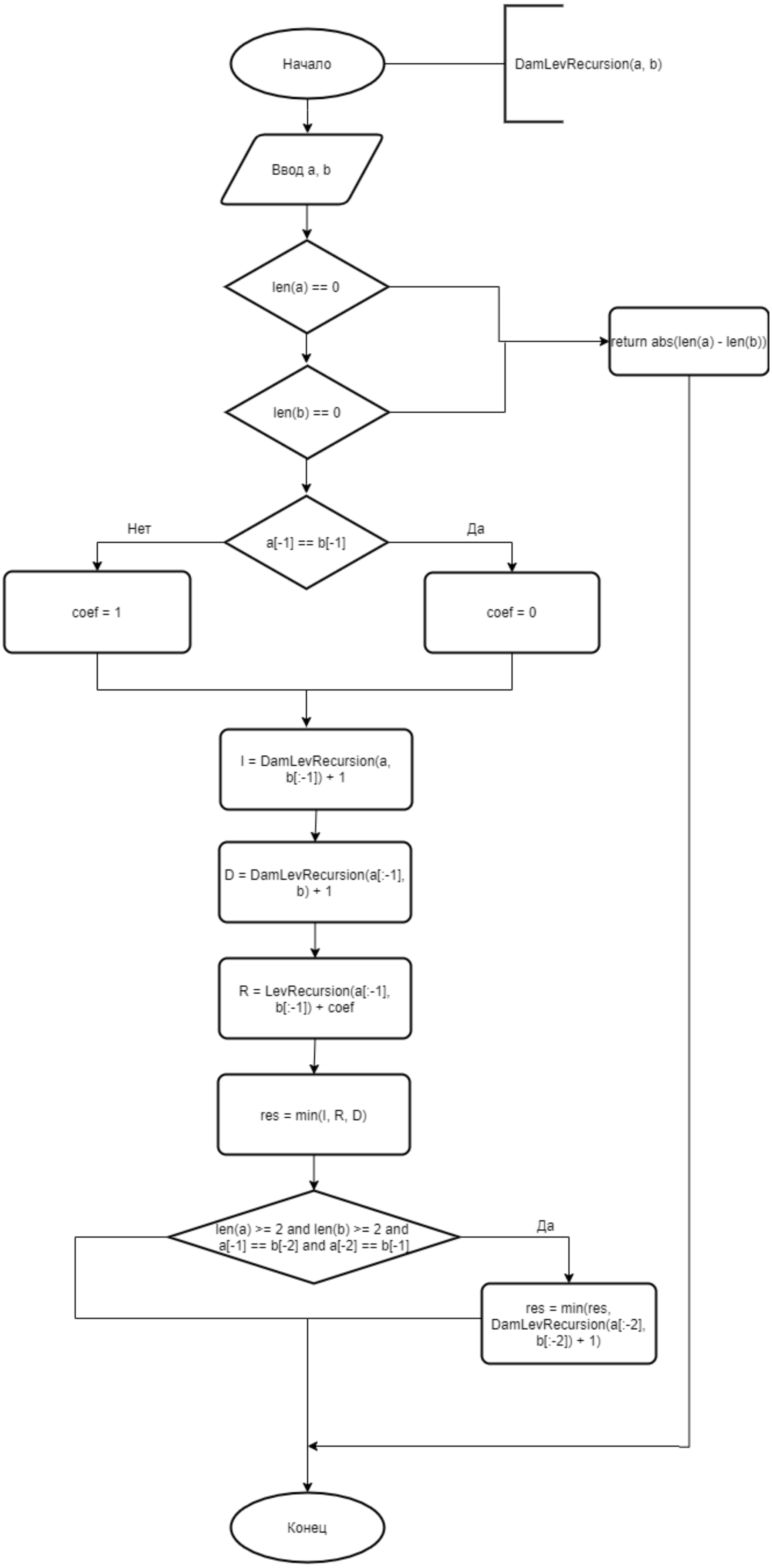

Рис. 2.3: Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна

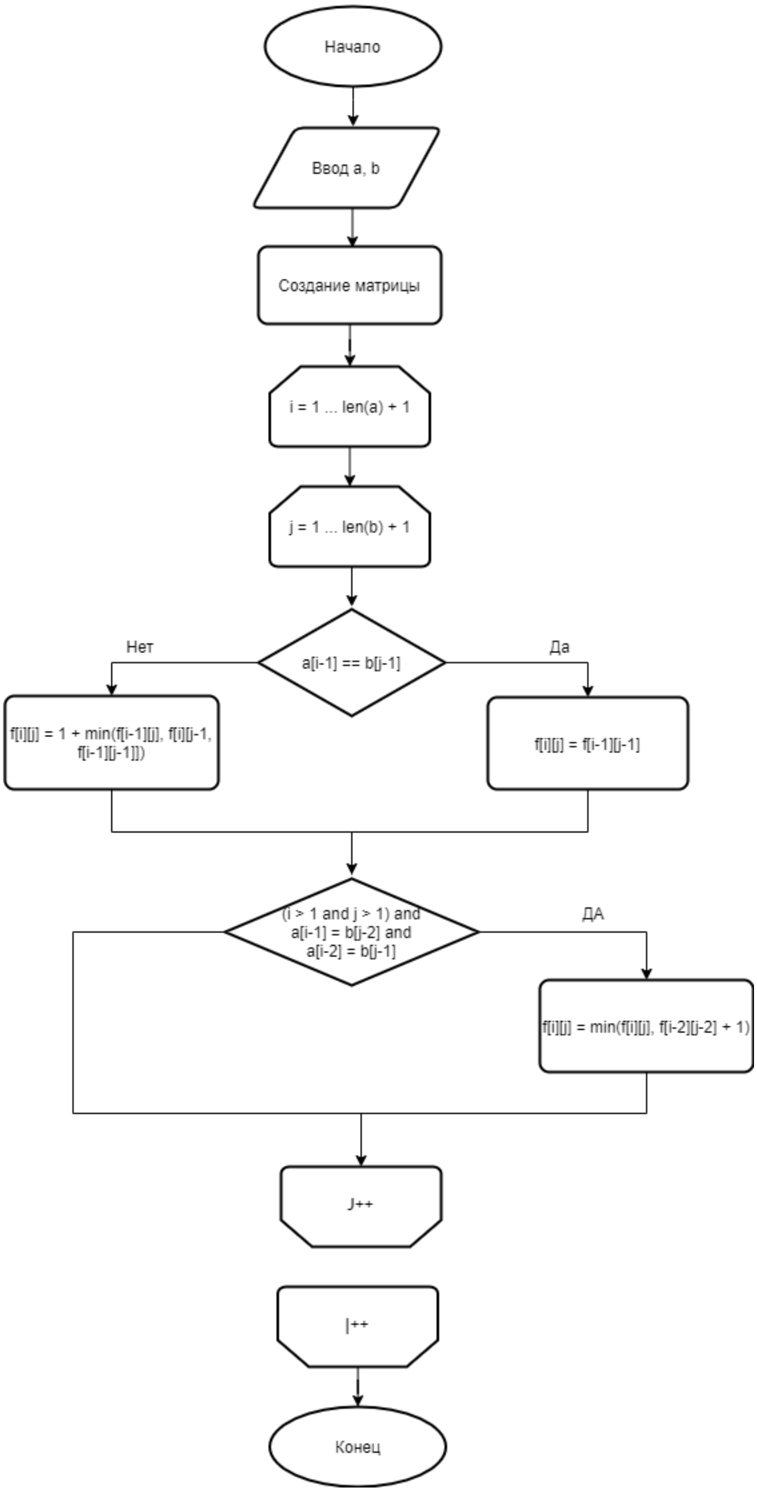

Рис. 2.4: Схема матричного алгоритма нахождения расстояния Дамерау-Левенштейна

Глава 3

Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

Входные данные: a - первое слово, b - второе слово.
Выходные данные: значение расстояние между двумя словами.

Рис. 3.1: IDEF0-диаграмма, описывающая алгоритм нахождения расстояния Левенштейна.

3.2 Средства реализации

В данной работе используется язык программирования Python, так как ЯП позволяет написать программу за кратчайшее время. Проект выполнен в среде разработки Visual Studio Code.

3.3 Листинг кода

В данном пункте представлен листинг кода, а именно:

- Расстояние Левенштейна
- Рекурсивное расстояние Левенштейна
- Расстояние Дамерау-Левенштейна
- Рекурсивное расстояние Дамерау-Левенштейна

```
    def LevTable(a, b):
        f=[[i+j if i*j==0 else 0 for j in range(len(b)+1)]
                                 for i in range (len(a)+1)]
        for i in range(1, len(a) + 1):
            for j in range (1, len(b) + 1):
                if a[i-1] == b[j-1]:
                    f[i][j] = f[i-1][j-1]
                else:
                    f[i][j]=1+min(f[i-1][j], f[i][j-1], f[i-1][j-1])
        return f[len(a)][len(b)]

```

Листинг 3.1: Расстояние Левенштейна

```
        def DamLevRecursion(a, b):
            if a ==  "" or b == "":
                return abs(len(a) - len(b))
            coef = 0 if (a[-1] == b[-1]) else 1
            res = min(DamLevRecursion(a, b[:-1]) + 1,
                      DamLevRecursion(a[:-1], b) + 1,
                      DamLevRecursion(a[:-1], b[:-1]) + coef)
            if (len(a) >= 2 and len(b) >= 2 and a[-1] == b[-2]
                                             and a[-2] == b[-1]):
                res = min(res, DamLevRecursion(a[:-2], b[:-2]) + 1)
            return res

```

Листинг 3.2: Рекурсивное расстояние Левенштейна

```
def DamLevTable(a, b):
    f = [[i+j if i*j == 0 else 0 for j in range(len(b) + 1)]
                                 for i in range (len(a) + 1)]
    for i in range(1, len(a) + 1):
        for j in range(1, len(b) + 1):
            if a[i-1] == b[j-1]:
                f[i][j] = f[i-1][j-1]
            else:
                f[i][j]=1+min(f[i-1][j],f[i][j-1],f[i-1][j-1])
            if (i > 1 and j > 1) and a[i-1] == b[j-2]
                                 and a[i-2] == b[j-1]:
                f[i][j] = min(f[i][j], f[i-2][j-2] + 1)
    return f[len(a)][len(b)]
```

Листинг 3.3: Расстояние Дамерау-Левенштейна

```
def DamLevRecursion(a, b):
    if a ==  "" or b == "":
        return abs(len(a) - len(b))
    coef = 0 if (a[-1] == b[-1]) else 1
    res = min(DamLevRecursion(a, b[:-1]) + 1,
              DamLevRecursion(a[:-1], b) + 1,
              DamLevRecursion(a[:-1], b[:-1]) + coef)
    if (len(a) >= 2 and len(b) >= 2 and a[-1] == b[-2]
                                    and a[-2] == b[-1]):
        res = min(res, DamLevRecursion(a[:-2], b[:-2]) + 1)
    return res
```

Листинг 3.4: Рекурсивное расстояние Дамерау-Левенштейна

3.4 Результаты тестирования

В данном разделе будут показаны результаты тестирования.

```
Basic tests:
Number of failed tests in LevRecursion method:  0
Number of failed tests in LevTable method:  0
Number of failed tests in DamLevRecursion method:  0
Number of failed tests in DamLevTable method:  0
Pro tests(Here we check how works table and recursions methods):
Levenstein method:  0
Damerau-Levenstein method:  0
```

Рис. 3.2: Результаты проведенных тестов.

3.5 Вывод

В данном разделе была представлена реализация алгоритмов нахождения расстояния Левенштейна, Дамерау-Левенштейна, а также рекурсивные алгоритмы Левенштейна и Дамерау-Левенштейна. Было проведено тестирование направленное на правильность её работы. Все тесты прошли успешно.

Глава 4

Исследовательская часть

В данном разделе будет проведен эксперимент и сравнительный анализ.

4.1 Постановка эксперимента

В рамках данного проекта были проведены эксперименты, описанные ниже:

1. Сравнение алгоритмов Левенштейна и Дамерау-Левенштейна. Количество символов в слове от 1 до 1000 с шагом 50. Один эксперимент ставился 100 раз;

2. Сравнение рекурсивного и итеративного алгоритмов Дамерау-Левенштейна. Количество символов в слове от 1 до 6 с шагом 1. Один эксперимент ставился 20 раз;

4.2 Сравнительный анализ на материале экспериментальных данных

Сравнение времени работы алгоритмов Левенштейна и Дамерау-Левенштейна:

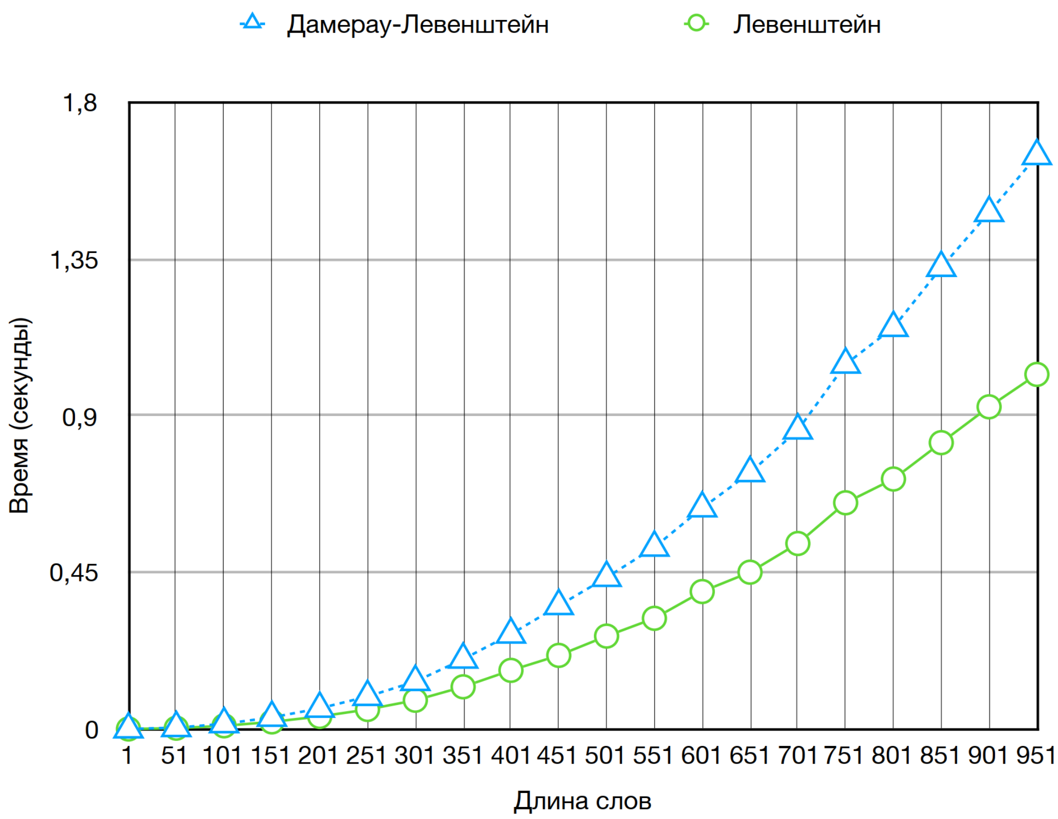

Рис. 4.1: График работы алгоритмов Левенштейна и Дамерау-Левенштейна

Можно заметить что алгоритм Дамерау-Левенштейна работает дольше, так как имеет более сложную логику. Также стоит отметить, что оба графика имеет схожий характер.

Сравнение времени работы рекурсивных алгоритмов Левенштейна и Дамерау-Левенштейна:

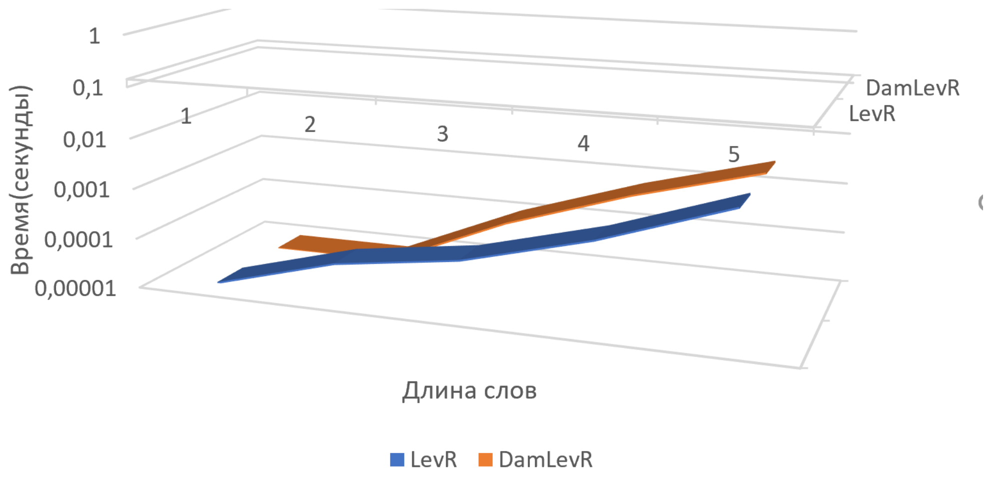

Рис. 4.2: График работы рекурсивных алгоритмов Левенштейна и Дамерау-Левенштейна

В приведенном выше графике можно заметить, что оба алгоритмы имеют одинаковый характер, но по скорости рекурсивный алгоритм Левенштейна работает быстрее, чем рекурсивный алгоритм Дамерау-Левенштейна.Замеры рекурсивного алгоритма с итеративным не имеют смысла, так время выполнения рекурсивного алгоритма увеличивается экспоненциально. При одинаковом размере строк рекурсивный алгоритм сильно проигрывает итеративному.

4.3 Вывод

В данном разделе был поставлен эксперимент по замеру времени выполнения алгоритма. По итогам замеров алгоритм нахождения расстояния Левенштейна оказался самым быстрым, а самым медленным - рекурсивный алгоритм Дамерау-Левенштейна.

Заключение

В ходе работы были изучены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна(рекурсивный и итеративный). Выполнено сравнение рекурсивных и итеративных алгоритмов. Изучены зависимости времени выполнения алгоритмов от длин строк. При сравнении времени выполнения алгоритмов, стало понятно, что самый быстрый среди рассматриваемых алгоритмов - алгоритм Левенштейна, а самый медленный рекурсивный алгоритм Дамерау-Левенштейна. Также реализован программный код продукта.