Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа № 1

Расстояния Левенштейна и Дамерау-Левенштейна

Студент: Сиденко Анастасия Генадьевна
Группа: ИУ7-53Б

Преподаватели: Строганов Юрий Владимирович
Волкова Лидия Леонидовна

2019 г.

Содержание

Введение

В данной лабораторной работе ставятся следующие задачи:

1. Изучение алгоритма Левенштейна и его модификации (алгоритма Дамерау-Левенштейна) для нахождения расстояние между строками
2. Получение практических навыков реализации данных алгоритма с использованием рекурсии и матрицы на одном из языков программирования
3. Сравнительный анализ матричного и рекурсивного алгоритмов по затрачиваемым ресурсам (зависимость времени от длины строки)
4. Экспериментальное подтверждение различий во временной эффективности рекурсивной и матричной реализаций выбранного алгоритма определения расстояния между строками на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк

1 Аналитическая часть

Рассмотрим понятия расстояния Левенштейна и расстояния Дамерау-Левенштейна, их использование в настоящее время.

1.1 Описание задачи

Расстояние Левенштейна (редакционное расстояние) — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Для двух строк $s1, s2$, их длины m, n соответстсвенно, расстояние Левенштейна равно $D(m, n)$, где

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & i > 0, j = 0 \\ j, & i = 0, j > 0 \\ min \begin{pmatrix} D(i, j-1) + 1 \\ D(i-1, j) + 1 \\ D(i-1, j-1) + \begin{cases} 0, & s1[i] = s2[j] \\ 1, & else \end{cases} \end{pmatrix} & j > 0, i > 0 \end{cases} \tag{1}$$

Редакторские операции:

1. I – Вставка – штраф 1
2. D – Удаление – штраф 1
3. R – Замена – штраф 1
4. M – Совпадение – штраф 0

Расстояние Дамерау-Левенштейна – если к списку разрешённых операций добавить транспозицию (два соседних символа меняются местами), получается расстояние Дамерау — Левенштейна. Дамерау показал, что 80% ошибок при наборе текста человеком являются транспозициями. Цена ошибки транспозиции равна 1.

Для двух строк $s1, s2$, их длины m, n соответстсвенно, расстояние Левенштейна равно $D(m, n)$, где

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & i > 0, j = 0 \\ j, & i = 0, j > 0 \\ min \begin{pmatrix} D(i, j-1) + 1 \\ D(i-1, j) + 1 \\ D(i-1, j-1) + \begin{cases} 0, & s1[i] = s2[j] \\ 1, & else \end{cases} \\ D(i-2, j-2) + 1 \end{pmatrix} & i, j > 1, s1[i] = s2[j-1], s1[i-1] = s2[j] \\ min \begin{pmatrix} D(i, j-1) + 1 \\ D(i-1, j) + 1 \\ D(i-1, j-1) + \begin{cases} 0, & s1[i] = s2[j] \\ 1, & else \end{cases} \end{pmatrix} & else \end{cases} \tag{2}$$

Расстояние Левенштейна и его модификация активно применяются:

1. для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи).
2. для сравнения текстовых файлов утилитой diff и ей подобными. Здесь роль «символов» играют строки, а роль «строк» — файлы.
3. в биоинформатике для сравнения генов, хромосом и белков.

1.2 Пути решения

Впервые задачу поставил в 1965 году советский математик Владимир Левенштейн при изучении последовательностей 0-1, а впоследствии более общую задачу для произвольного алфавита связали с его именем.

А пару лет назад Фредерик Дамерау доказал, что большинство ошибок при наборе текста — как раз и есть транспозиции. Поэтому именно данная метрика дает наилучшие результаты на практике.

Исходный вариант алгоритма имеет временную сложность O((m+1)(n+1)) и потребляет O((m+1)(n+1)) памяти, где m и n — длины сравниваемых строк. Весь процесс можно представить матрицей.

1.3 Выводы

Алгоритмы Дамерау и Дамерау-Левенштейна, для нахождения редакторского расстояния, особо актуальны во время развития технологий и биоинформатики. В основе нахождения минимального расстояния лежат редакторские операции - вставка, удаление, замена, совпадение и транспозиция, которая была введена Дамерау. В данной работе требуется изучить и применить данные алгоритмы, а также получить сравнение реализаций по скорости работы.

2 Конструкторская часть

В данной работе для нахождения расстояния Левенштейна используется матричный алгоритм, для Дамерау-Левенштейна - матричный и рекурсивный. Рассмотрим и изучим данные варианты реализации.

2.1 IDEF0

На рисунке 1 показана диаграмма нашего алгоритма.

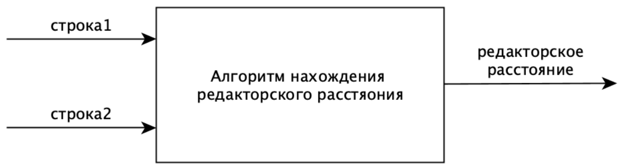

Рис. 1 - IDEF0

2.2 Схемы алгоритмов

Приведем схемы 3 исследуемых нами алгоритмов.

2.2.1 Расстояние Левенштейна матричным способом

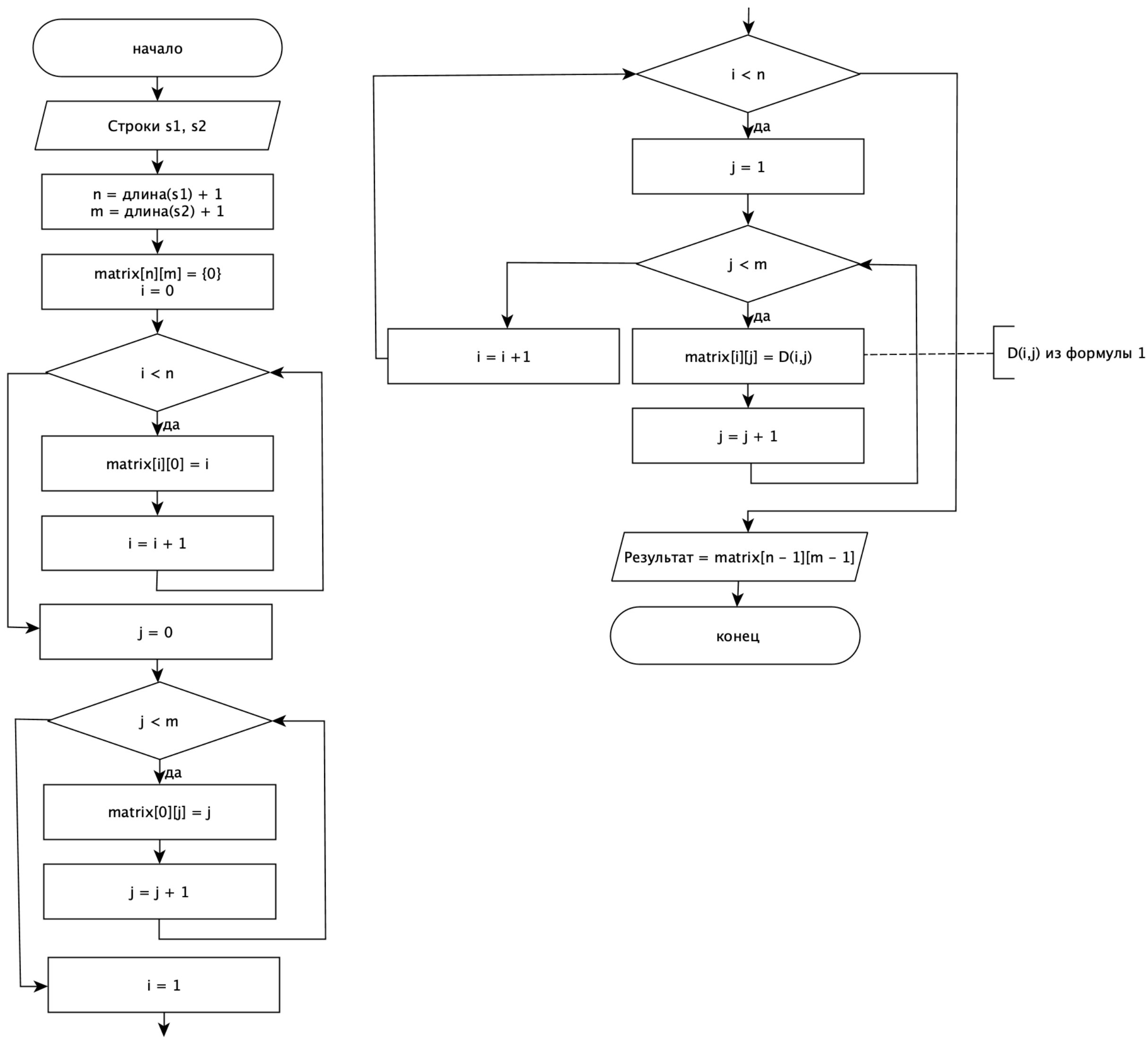

Схема 1 - алгоритм нахождения расстояния Левенштейна матричным способом

2.2.2 Расстояние Дамерау-Левенштейна матричным способом

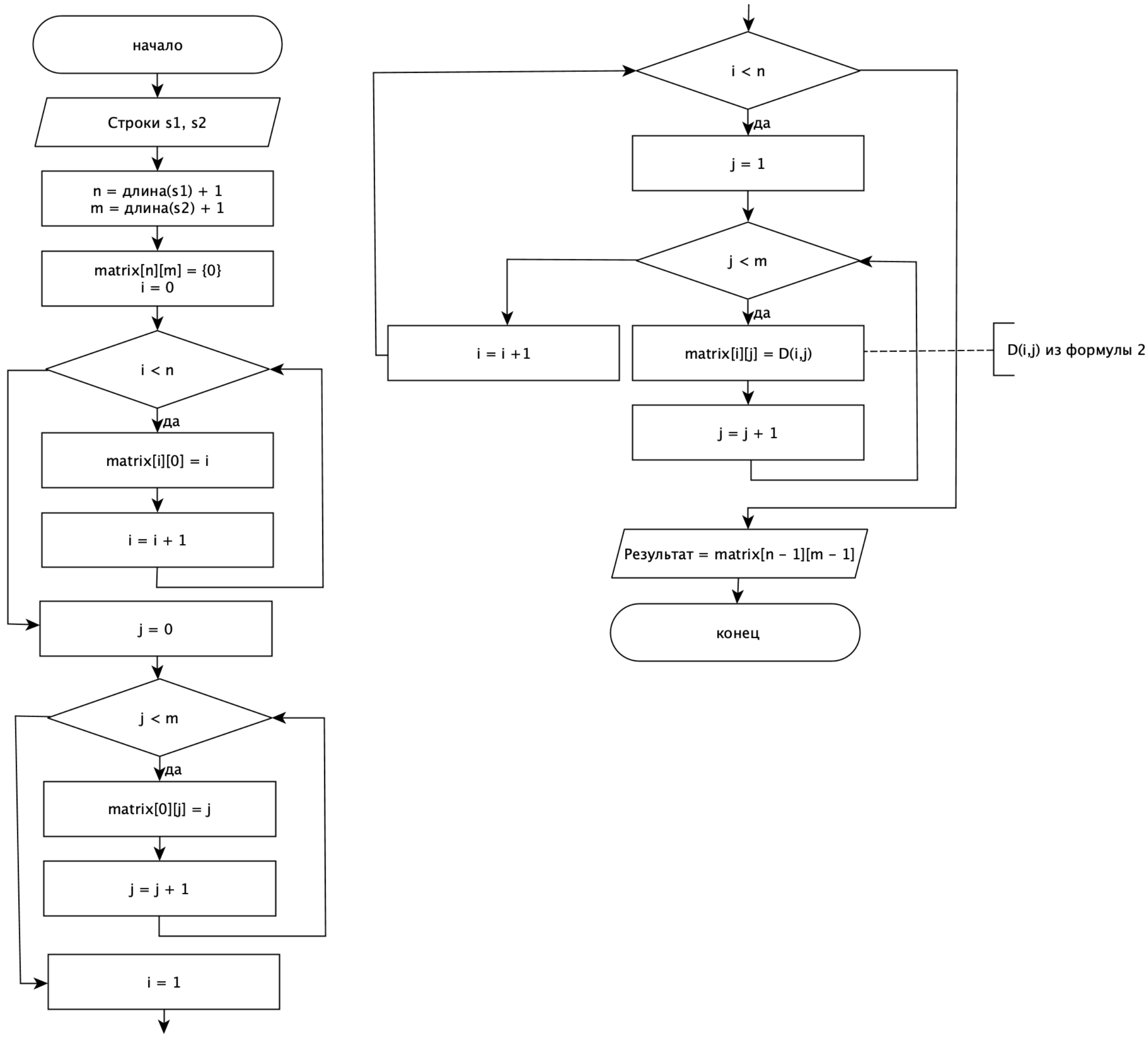

Схема 2 - алгоритм нахождения расстояния Дамерау-Левенштейна матричным способом

2.2.3 Расстояние Дамерау-Левенштейна рекурсивным способом

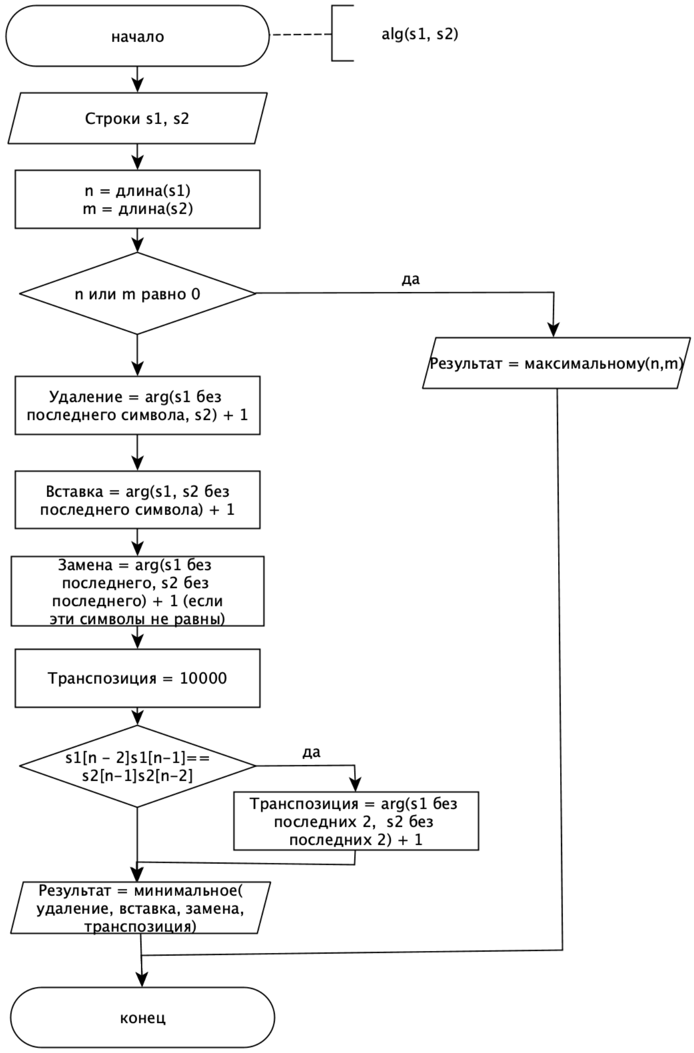

Схема 3 - алгоритм нахождения расстояния Дамерау-Левенштейна рекурсивным способом

2.3 Выводы

В самом коротком пути рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна имеет сложность $\Omega(4^{min(m,n)})$, а в максимальная длина $\Omega(4^{m+n+1})$, матричная реализация имеет сложность $\Omega(mn)$. Следует вывод, что рекурсивная реализация является гораздо более затратной по времени. Реализуем алгоритмы поиска редакторских расстояний и проверим наше предположение.

3 Технологическая часть

Стоит задача разработки и сравнительного анализа алгоритмов, вычисляющих редакторские расстояния.

В реализациях в целях увеличения точности подсчета времени вывод матрицы был вынесен за пределы функций-алгоритмов. В целях наглядности были опущены части программ, не относящиеся к работе алгоритмов.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность замеров процессорного времени выполнения реализации каждого алгоритма. Требуется провести замеры для варьирующихся длин строк: от 100 до 1000. Один эксперимент ставится не менее 100 раз, результат одного эксперимента рассчитывается как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

Используемое ПО - Mac OS.

3.2 Средства реализации

В качестве языка программирования был выбран Python.

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python обладает динамической типизацией, автоматическим управлением памятью.

3.3 Листинг кода

Расстояние Левинштейна, матричный способ:

```
def levinstein(s1, s2):
    n_matrix = len(s1) + 1
    m_matrix = len(s2) + 1
    matrix = [[0] * m_matrix for i in range(n_matrix)]

    for i in range(n_matrix):
        matrix[i][0] = i
    for j in range(m_matrix):
        matrix[0][j] = j

    for i in range(1, n_matrix):
        for j in range(1, m_matrix):
            matrix[i][j] = min(matrix[i - 1][j] + 1, matrix[i][j - 1] + 1,
            matrix[i - 1][j - 1] + (s1[i - 1] != s2[j - 1]))

    return matrix[n_matrix - 1][m_matrix - 1]
```

Расстояние Дамерау-Левинштейна, матричный способ:

```
def damerau_levinstein_matrix(s1, s2):
    n_matrix = len(s1) + 1
    m_matrix = len(s2) + 1
    matrix = [[0] * m_matrix for i in range(n_matrix)]

    for i in range(n_matrix):
        matrix[i][0] = i
    for j in range(m_matrix):
        matrix[0][j] = j

    for i in range(1, n_matrix):
        for j in range(1, m_matrix):
             if (i > 1 and s1[i - 1] == s2[j - 2] and s1[i - 2] == s2[j - 1]):
```

```
                    matrix[i][j] = min(matrix[i - 1][j] + 1,
                    matrix[i][j - 1] + 1, matrix[i - 1][j - 1] +
                    (s1[i - 1] != s2[j - 1]), matrix[i - 2][j - 2] + 1)
                else:
                    matrix[i][j] = min(matrix[i - 1][j] + 1,
                    matrix[i][j - 1] + 1, matrix[i - 1][j - 1] +
                    (s1[i - 1] != s2[j - 1]))

        return matrix[n_matrix - 1][m_matrix - 1]
```

Расстояние Дамерау-Левинштейна, рекурсивный способ:

```
    def damerau_levinstein_recursive(s1, s2):
        n = len(s1)
        m = len(s2)
        if (0 == n):
            return m
        if (0 == m):
            return n

        delete = damerau_levinstein_recursive(s1[:-1], s2) + 1
        insert = damerau_levinstein_recursive(s1, s2[:-1]) + 1
        replace = damerau_levinstein_recursive(s1[:-1], s2[:-1]) +
        (s1[-1] != s2[-1])
        transposition = 1000
        if (n > 1 and m > 1 and s1[-1] == s2[-2] and s1[-2] == s2[-1]):
            transposition = damerau_levinstein_recursive(s1[:-2], s2[:-2]) + 1

        return min(delete, insert, replace, transposition)
```

3.4 Тестирование

В таблице 1 представлена заготовка данных для тестирования наших алгоритмов.

Строка1	**Строка2**	**Ожидаемый результат**
dessert	desert	1
cook	cooker	2
mother	money	3
woman	water	4
program	friend	6
house	girl	5
probelm	problem	1 or 2
head	ehda	2 or 3
bring	brought	4
happy	happy	0
minute	moment	5
person	eye	5
week	weeks	1
member	morning	6
death	health	2
education	question	4
room	moor	2
car	city	3
air	area	3
country	office	6

Таблица 1. Подготовленные тестовые данные.

3.5 Выводы

Реализовано 3 алгоритма, подготовлены тесты для оценки качества их работы.

Получены практические навыки реализации матричного поиска расстояния Левинштейна и матричного и рекрсивного – Дамерау-Левенштейна.

4 Экспериментальная часть

Оценка качества работы алгоритмов. Экспериментальное сравнение работы различных алгоритмов (зависимость времени выполнения от длины входных слов).

4.1 Примеры работы

На рисунке 2 представлены примеры работы программы на разных входных данных.

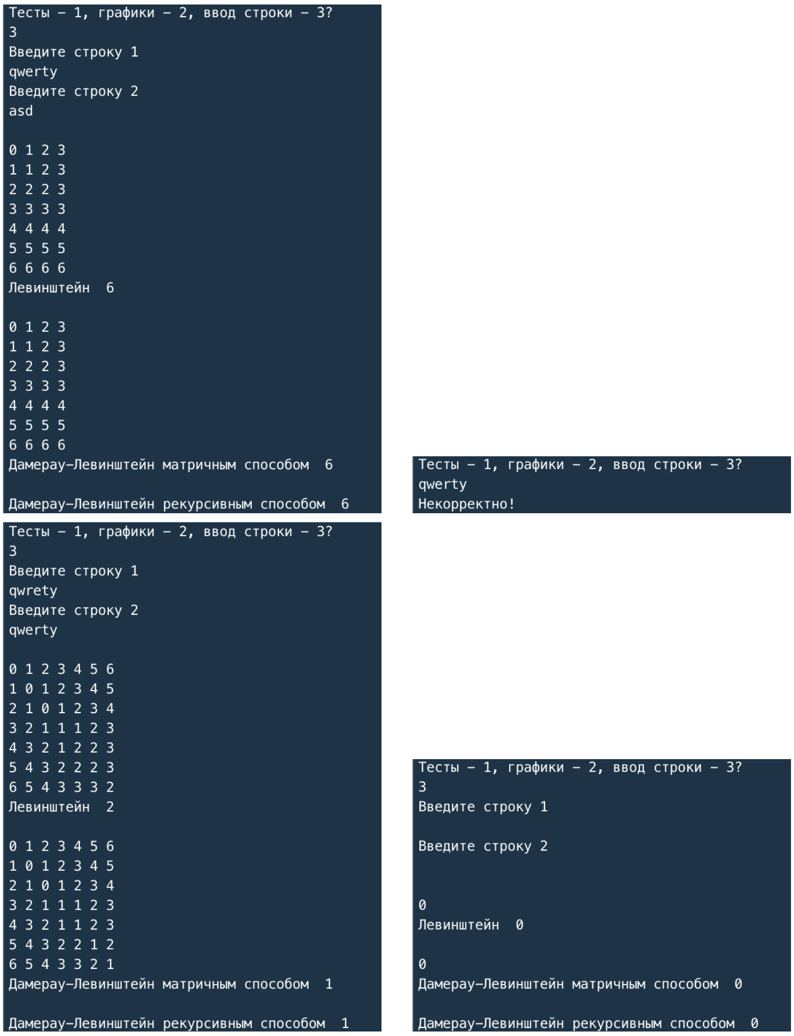

Рис. 2 - Примеры работы

4.2 Результаты тестирования

Проверяем нашу программу на тестах из таблицы 1. Полученные результаты представлены в таблице 2.

Строка1	Строка2	Л. матричный	Д.-Л. матричный	Д.-Л. рекурсивный
dessert	desert	1	1	1
cook	cooker	2	2	2
mother	money	3	3	3
woman	water	4	4	4
program	friend	6	6	6
house	girl	5	5	5
probelm	problem	2	1	1
head	ehda	3	2	2
bring	brought	4	4	4
happy	happy	0	0	0
minute	moment	5	5	5
person	eye	5	5	5
week	weeks	1	1	1
member	morning	6	6	6
death	health	2	2	2
education	question	4	4	4
room	moor	2	2	2
car	city	3	3	3
air	area	3	3	3
country	office	6	6	6

Таблица 2. Тестирование программы.

Тесты пройдены

4.3 Замеры времени

На графиках 1-3 представлено сравнение алгоритмов поиска редакционного расстояния по времени.

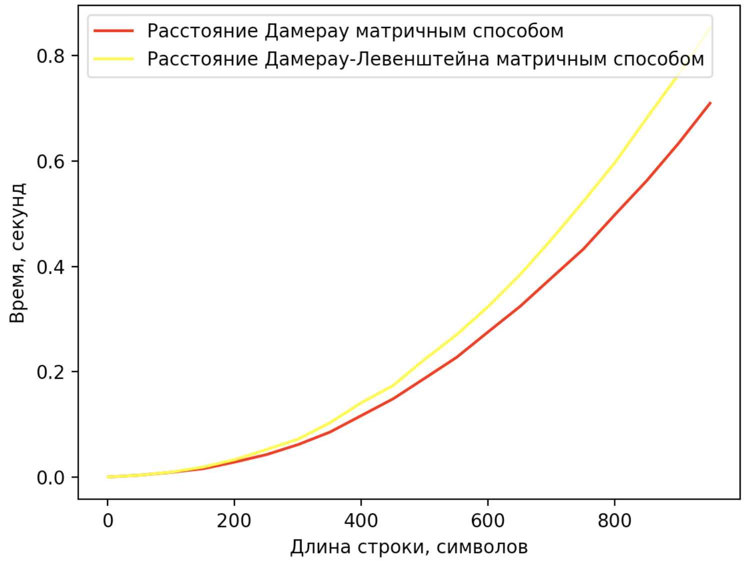

График 1 - Сравнение реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна матричным способом

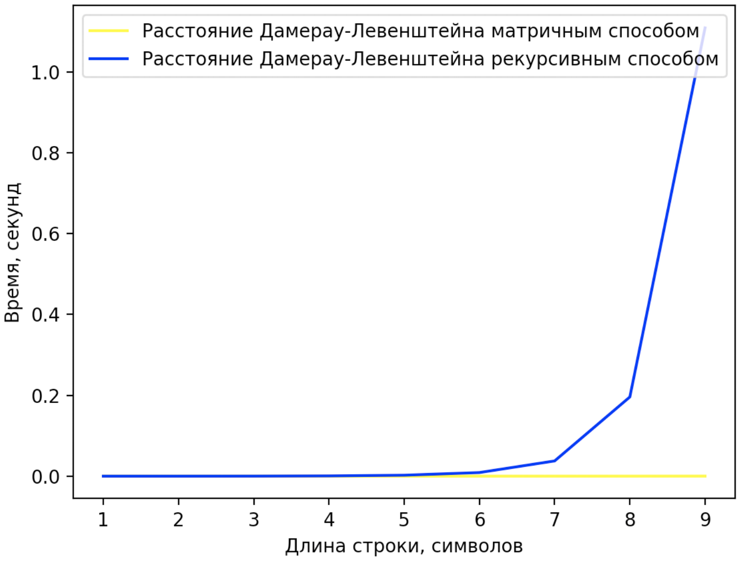

График 2 - Сравнение реализации алгоритмов нахождения расстояния Дамерау-Левенштейна матричным и рекурсивным способом

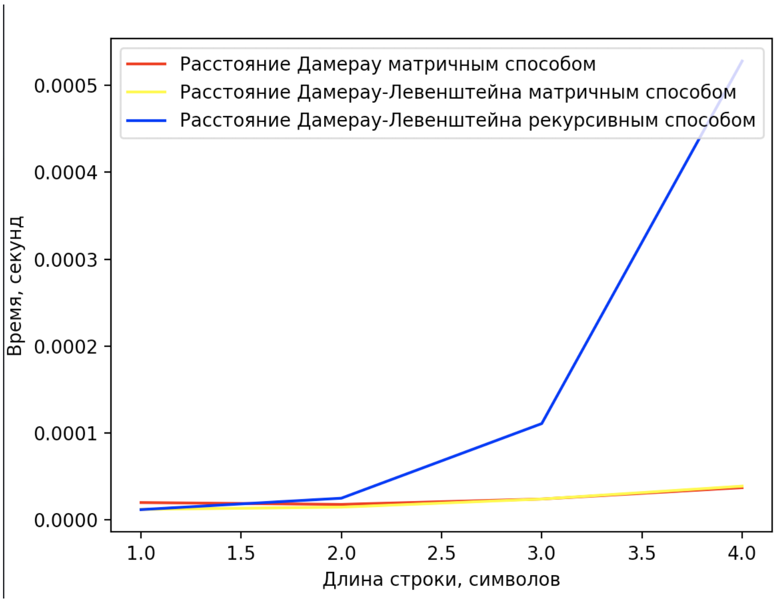

График 3 - Сравнение реализации всех 3 алгоритмов

4.4 Выводы

Как видно из графиков рекурсивный алгоритм является самым медленным, гораздо быстрее использовать алгоритмы матричные. Дамерау-Левенштейна проигрывает обычному Левенштейну только на очень больших длинах слов, но цена ошибки, в некоторых случаях, у него меньше.

Заключение

В данной лабораторной работе было реализовано и проанализировано 3 алгоритма нахождения редакционных расстояний:

1. нахождение расстояния Левенштейна матричным способом

2. нахождение расстояния Дамерау-Левенштейна матричным способом

3. нахождение расстояния Дамерау-Левенштейна рекурсивным способом

Расстояние Левенштейна и его модификация активно применяются:

1. для исправления ошибок в слове (компьютерная или машинная лингвистика)
2. для сравнения текстовых файлов утилитой diff и ей подобными
3. в биоинформатике для сравнения генов, хромосом и белков

При сравнении данных алгоритмов пришли к следующим выводам:

1. Рекурсивный алгоритм является самым медленным, гораздо быстрее использовать алгоритмы матричные.

2. Дамерау-Левенштейна проигрывает обычному Левенштейну только на очень больших длинах слов, но цена ошибки, в некоторых случаях, у него меньше.

Список литературы

[1] Сложность дистанции редактирования (расстояние Левенштейна)[Электронный ресурс]. – Режим доступа: http://qaru.site/questions/3908099/complexity-of-edit-distance-levenshtein-distance-recursion-top-down-implementation (дата обращения: 15.09.2019).

[2] Вычисление редакционного расстояния[Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/117063/ (дата обращения: 15.09.2019).

[3] Нечёткий поиск в тексте и словаре[Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/114997/ (дата обращения: 15.09.2019).