

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 Геннадьевна

Группа: ИУ7-53Б

Преподаватели:

Строганов Юрий Владимирович

Волкова Лилия Леонидовна

2019 г.

Содержание

| | |
|---|-----------|
| Введение | 2 |
| 1 Аналитическая часть | 3 |
| 1.1 Описание задачи | 3 |
| 1.2 Выводы | 4 |
| 2 Конструкторская часть | 5 |
| 2.1 Функциональная модель | 5 |
| 2.2 Схемы алгоритмов | 5 |
| 2.3 Выводы | 9 |
| 3 Технологическая часть | 10 |
| 3.1 Требования к программному обеспечению | 10 |
| 3.2 Средства реализации | 10 |
| 3.3 Листинг кода | 10 |
| 3.4 Тестирование | 11 |
| 3.5 Выводы | 11 |
| 4 Экспериментальная часть | 13 |
| 4.1 Примеры работы | 13 |
| 4.2 Результаты тестирования | 13 |
| 4.3 Замеры времени | 14 |
| 4.4 Выводы | 16 |
| Заключение | 17 |

Введение

Все мы в повседневной жизни давно пользуемся поисковыми системами, такими как Яндекс или Google, чтобы найти необходимую нам информацию. С каждым годом поисковые системы учатся понимать нас всё лучше, предлагают в результатах наиболее подходящие запросы, чтобы мы не тратили много времени на просмотр поисковой выдачи. Сегодня широко применяется автоматическая обработка естественного языка, проверка орфографии и ошибок в написаниях слов, не только в поисковых системах[1, стр. 13], но и в текстовых редакторах[1, стр. 26], системах распознавания отсканированного текста[1, стр. 14]. Задача сводится к поиску информации не только по заданному образцу, но и близкому к этому образцу значению – нечеткий поиск. Нечеткий поиск активно применяется в биоинформатике для решения задач сравнения генов, хромосом и белков[2, глава 18].

Алгоритмы нечеткого поиска характеризуются метрикой — расстоянием между двумя словами, позволяющей оценить степень их сходства[3]. Примерами таких расстояний являются расстояние Левенштейна и Дамерау-Левенштейна.

В данной лабораторной работе ставятся следующие задачи.

1. Изучение алгоритма Левенштейна и его модификации (алгоритма Дамерау-Левенштейна) для нахождения расстояния 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

Алгоритмы нечеткого поиска являются основой систем проверки орфографии и полноценных поисковых систем вроде Google или Yandex[1], также применяются в биоинформатике[2]. В числе наиболее известных метрик нечеткого поиска – расстояния Левенштейна и Дамерау-Левенштейна.

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[4].

Для двух строк $s1, s2$, их длины m, n соответственно, расстояние Левенштейна равно $D(m, n)$, которое находится по формуле 1.

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

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[4]. Дамерау показал, что 80% ошибок при наборе текста человеком являются транспозициями. Цена ошибки транспозиции равна 1.[6]

Для двух строк $s1, s2$, их длины m, n соответственно, расстояние Дамерау-Левенштейна равно $D(m, n)$, которое находится по формуле 2.

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

Расстояние Левенштейна и его модификация активно применяются:

- а) для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи)[1];
- б) для сравнения текстовых файлов утилитой diff и ей подобными. Здесь роль «символов» играют строки, а роль «строк» — файлы[1];

с) в биоинформатике для сравнения генов, хромосом и белков[2].

Впервые задачу поставил в 1965 году советский математик Владимир Левенштейн при изучении последовательностей 0-1 [5], а впоследствии более общую задачу для произвольного алфавита связали с его именем.

Фредерик Дамерау доказал, что большинство ошибок при наборе текста — как раз и есть транспозиции. Поэтому именно данная метрика дает наилучшие результаты на практике[6].

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.

1.2 Выводы

Проблема поиска, услуг или продукта, возникает на подавляющем большинстве сайтов. И в основной своей массе реализация подобной возможности ограничивается поиском по точному слову, которое ввели в поисковой строке.

Алгоритмы нечеткого поиска решают поставленную задачу, основной метрикой в этом случае является расстояние между двумя словами.

В основе алгоритмов поиска расстояний Дамерау и Дамерау-Левенштейна, для нахождения минимального редакционного 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. Необходимо рассмотреть и изучить данные варианты реализации.

2.1 Функциональная модель

На рисунке 1 представлена функциональная модель нашей задачи.



Рис. 1 - Функциональная модель алгоритма поиска редакторского расстояния

2.2 Схемы алгоритмов

Приведем схемы 3 исследуемых алгоритмов (см. рисунки 2-4).

Расстояние Левенштейна матричным способом

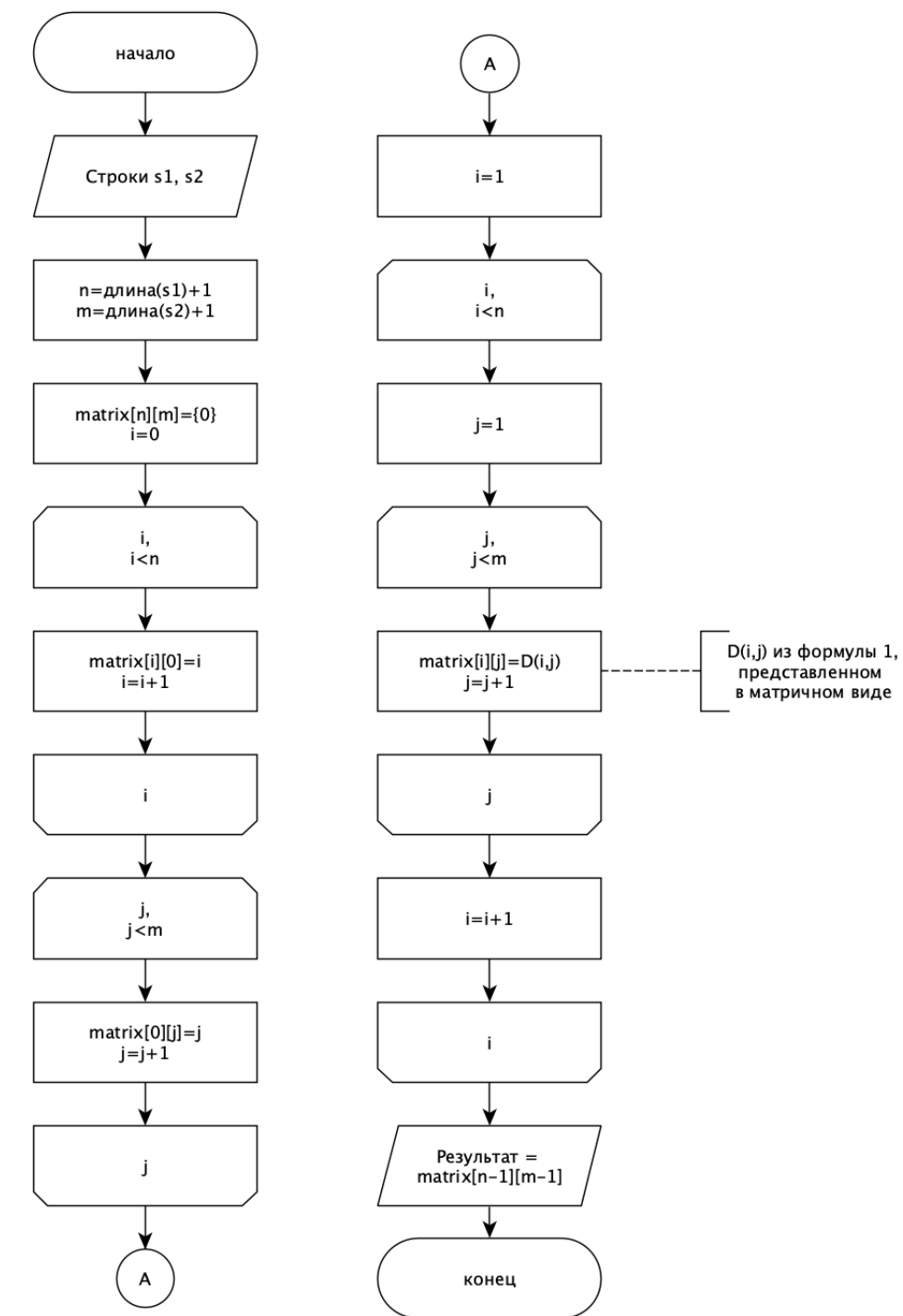


Рис. 2 - Алгоритм нахождения расстояния Левенштейна матричным способом

Расстояние Дамерау-Левенштейна матричным способом

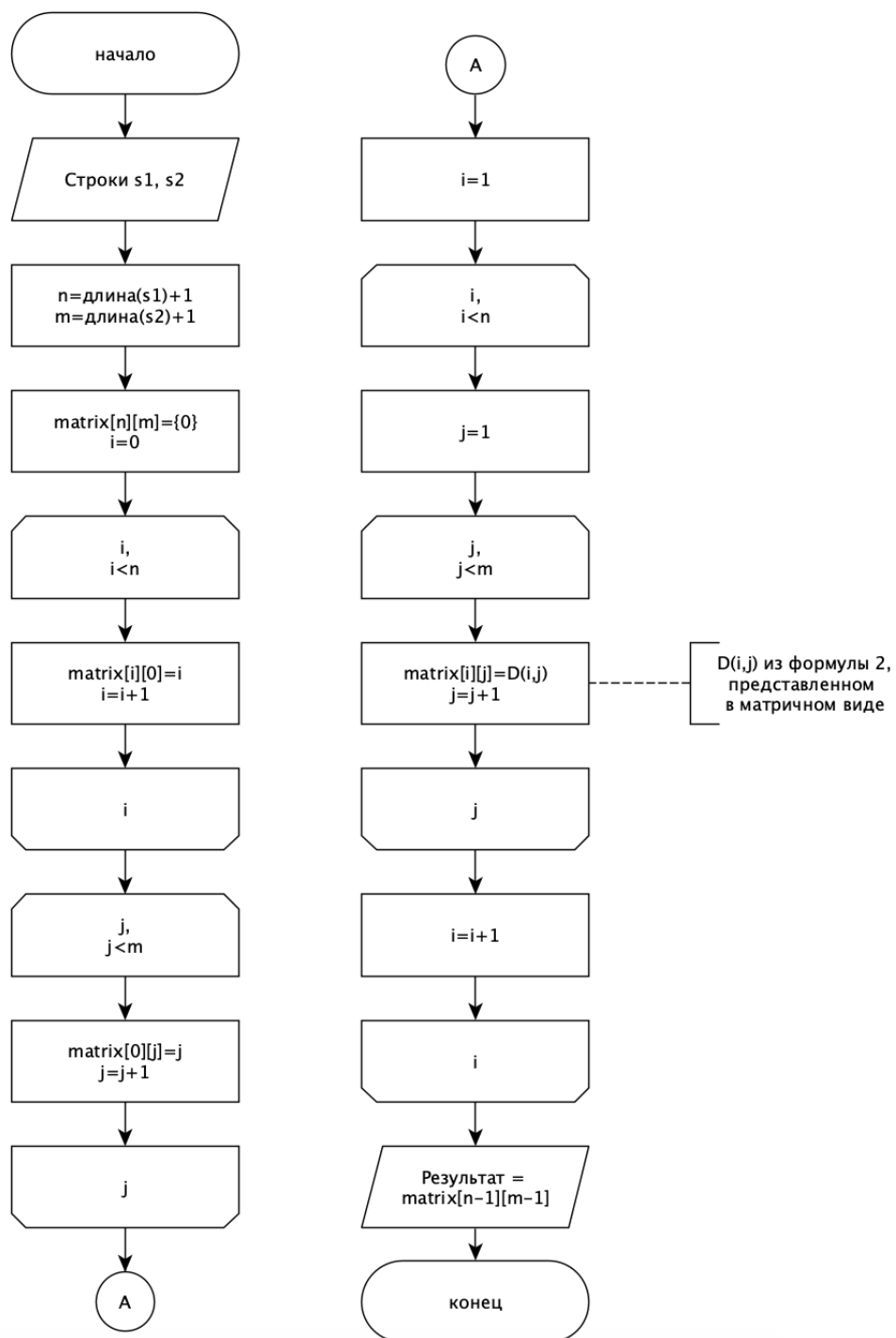


Рис. 3 - Алгоритм нахождения расстояния Дameraу-Левенштейна матричным способом

Расстояние Дameraу-Левенштейна рекурсивным способом

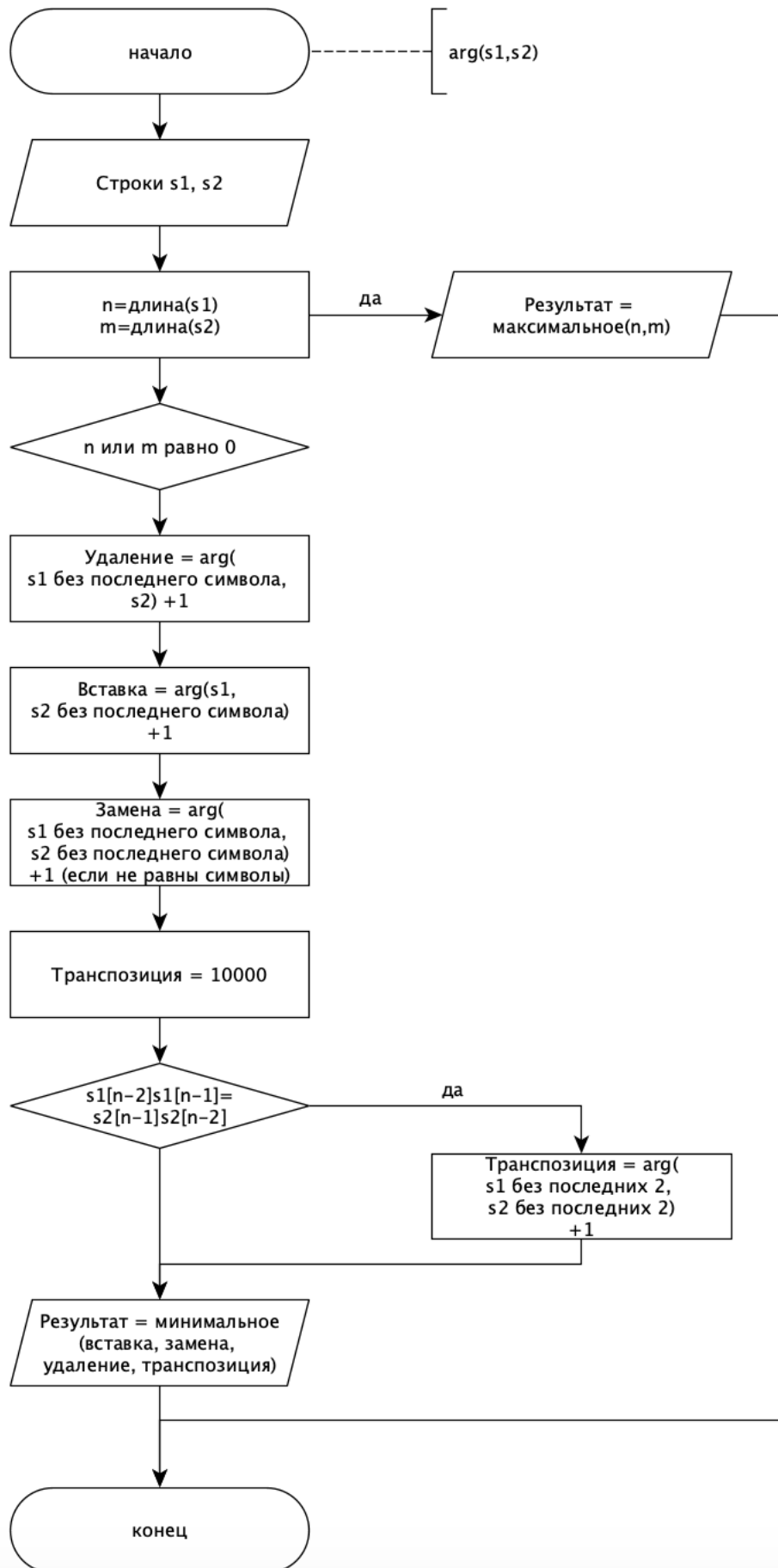


Рис. 4 - Алгоритм нахождения расстояния Дамерау-Левенштейна рекурсивным способом

2.3 Выводы

В самом коротком пути рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна имеет сложность $\Omega(4^{\min(m,n)})$, а максимальная сложность $\Omega(4^{m+n+1})$, матричная реализация имеет сложность $\Omega(mn)$. Следует вывод, что рекурсивная реализация является гораздо более затратной по времени. Необходимо проверить предположение, реализовав данные алгоритмы.

3 Технологическая часть

Стоит задача разработки и сравнительного анализа алгоритмов, вычисляющих редакционные 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.

3.2 Средства реализации

В качестве языка программирования был выбран Python[8], так как я знакома с этим языком программирования.

Для замеров времени была выбран метод *process_time()*, возвращает текущее время процессора как число с плавающей запятой, выраженное в секундах в Unix.

Для генерации случайных строк, заданного размера использовался метод *random.choice()*.

3.3 Листинг кода

Расстояние Левинштейна, матричный способ:

```
1 def levinstein(s1, s2):
2     n_matrix = len(s1) + 1
3     m_matrix = len(s2) + 1
4     matrix = [[0] * m_matrix for i in range(n_matrix)]
5
6     for i in range(n_matrix):
7         matrix[i][0] = i
8     for j in range(m_matrix):
9         matrix[0][j] = j
10
11    for i in range(1, n_matrix):
12        for j in range(1, m_matrix):
13            matrix[i][j] = min(matrix[i - 1][j] + 1, matrix[i][j - 1] + 1,
14                               matrix[i - 1][j - 1] + (s1[i - 1] != s2[j - 1]))
15
16    return matrix[n_matrix - 1][m_matrix - 1]
```

Расстояние Дамерау-Левинштейна, матричный способ:

```
1 def damerau_levinsein_matrix(s1, s2):
2     n_matrix = len(s1) + 1
3     m_matrix = len(s2) + 1
4     matrix = [[0] * m_matrix for i in range(n_matrix)]
5
6     for i in range(n_matrix):
7         matrix[i][0] = i
8     for j in range(m_matrix):
9         matrix[0][j] = j
10
11    for i in range(1, n_matrix):
12        for j in range(1, m_matrix):
13            if (i > 1 and s1[i - 1] == s2[j - 2] and s1[i - 2] == s2[j - 1]):
```

```

14         matrix[i][j] = min(matrix[i - 1][j] + 1,
15         matrix[i][j - 1] + 1, matrix[i - 1][j - 1] +
16         (s1[i - 1] != s2[j - 1]), matrix[i - 2][j - 2] + 1)
17     else:
18         matrix[i][j] = min(matrix[i - 1][j] + 1,
19         matrix[i][j - 1] + 1, matrix[i - 1][j - 1] +
20         (s1[i - 1] != s2[j - 1]))
21
22     return matrix[n_matrix - 1][m_matrix - 1]

```

Расстояние Дамерау-Левинштейна, рекурсивный способ:

```

1     def damerau_levinstein_recursive(s1, s2):
2         n = len(s1)
3         m = len(s2)
4         if (0 == n):
5             return m
6         if (0 == m):
7             return n
8
9         delete = damerau_levinstein_recursive(s1[:-1], s2) + 1
10        insert = damerau_levinstein_recursive(s1, s2[:-1]) + 1
11        replace = damerau_levinstein_recursive(s1[:-1], s2[:-1]) +
12        (s1[-1] != s2[-1])
13        transposition = 1000
14        if (n > 1 and m > 1 and s1[-1] == s2[-2] and s1[-2] == s2[-1]):
15            transposition = damerau_levinstein_recursive(s1[:-2], s2[:-2]) + 1
16
17        return min(delete, insert, replace, transposition)

```

3.4 Тестирование

В таблице 1 представлена заготовка данных для тестирования наших алгоритмов.

| Строка1 | Строка2 | Ожидаемый результат (Дамерау or Дамерау-Левинштейна) |
|-----------|----------|--|
| dessert | desert | 1 |
| cook | cooker | 2 |
| mother | money | 3 |
| woman | water | 4 |
| program | friend | 6 |
| house | girl | 5 |
| probelm | problem | 2 or 1 |
| head | ehda | 3 or 2 |
| 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 рекурсивного – 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.

```
Тесты - 1, графики - 2, ввод строки - 3?  
3  
Введите строку 1  
qwerty  
Введите строку 2  
asd  
  
0 1 2 3  
1 1 2 3  
2 2 2 3  
3 3 3 3  
4 4 4 4  
5 5 5 5  
6 6 6 6  
Левинштейн 6  
  
0 1 2 3  
1 1 2 3  
2 2 2 3  
3 3 3 3  
4 4 4 4  
5 5 5 5  
6 6 6 6  
Дамерау-Левинштейн матричным способом 6  
Дамерау-Левинштейн рекурсивным способом 6  
  
Тесты - 1, графики - 2, ввод строки - 3?  
3  
Введите строку 1  
qwerty  
Введите строку 2  
qwerty  
  
0 1 2 3 4 5 6  
1 0 1 2 3 4 5  
2 1 0 1 2 3 4  
3 2 1 1 1 2 3  
4 3 2 1 2 2 3  
5 4 3 2 2 2 3  
6 5 4 3 3 3 2  
Левинштейн 2  
  
0 1 2 3 4 5 6  
1 0 1 2 3 4 5  
2 1 0 1 2 3 4  
3 2 1 1 1 2 3  
4 3 2 1 1 2 3  
5 4 3 2 2 1 2  
6 5 4 3 3 2 1  
Дамерау-Левинштейн матричным способом 1  
Дамерау-Левинштейн рекурсивным способом 1  
  
Тесты - 1, графики - 2, ввод строки - 3?  
3  
Введите строку 1  
  
Введите строку 2  
  
0  
Левинштейн 0  
  
0  
Дамерау-Левинштейн матричным способом 0  
Дамерау-Левинштейн рекурсивным способом 0
```

Рис. 5 - 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 (см. рисунки 6-8) представлено сравнение алгоритмов поиска редакционного расстояния по времени.

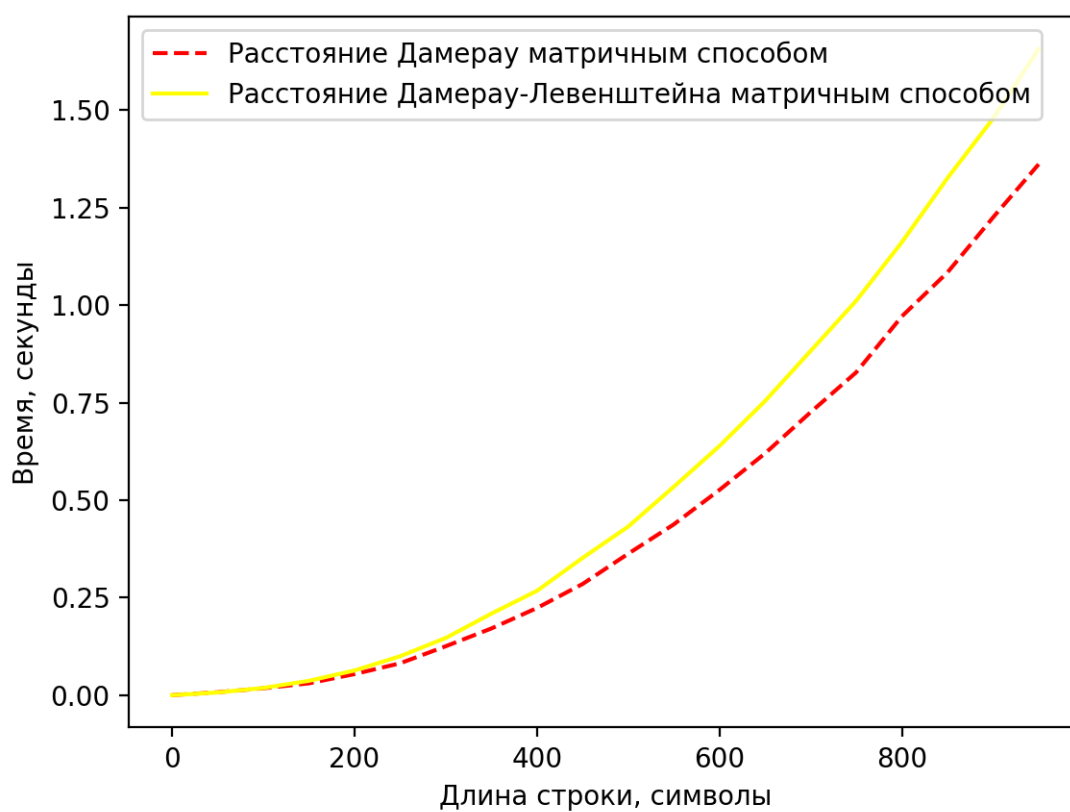


Рис. 6 - Сравнение реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна матричным способом

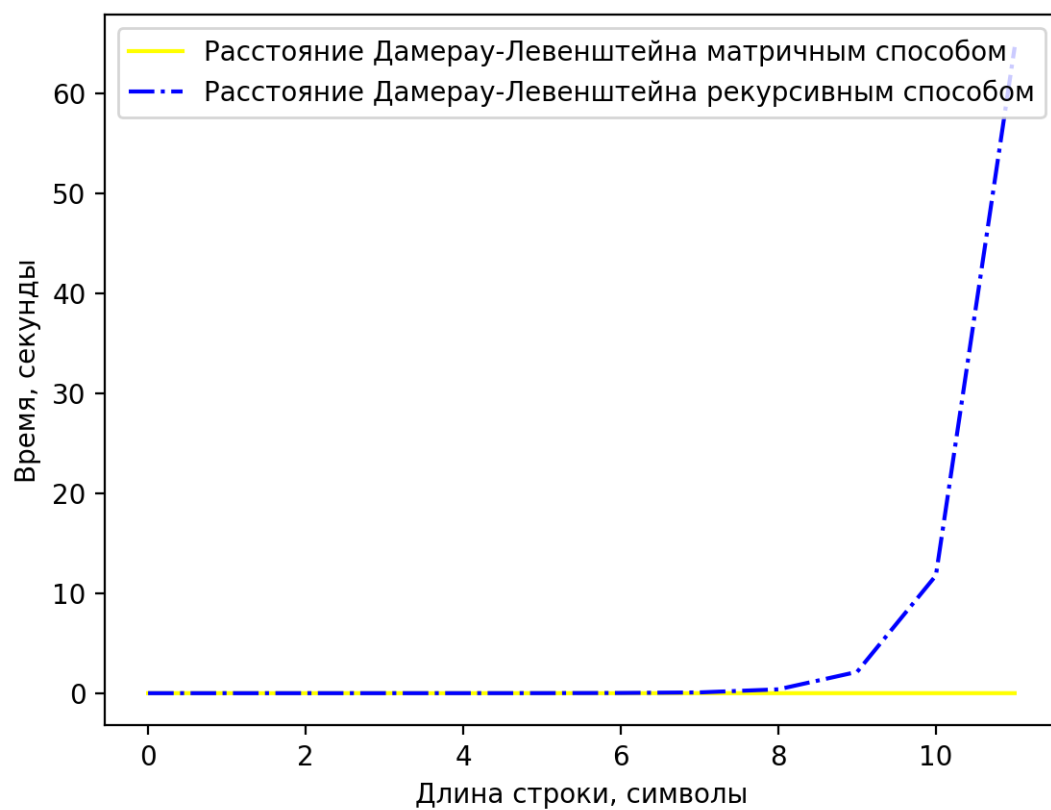


Рис. 7 - Сравнение реализации алгоритмов нахождения расстояния Дамерау-Левенштейна матричным и рекурсивным способом

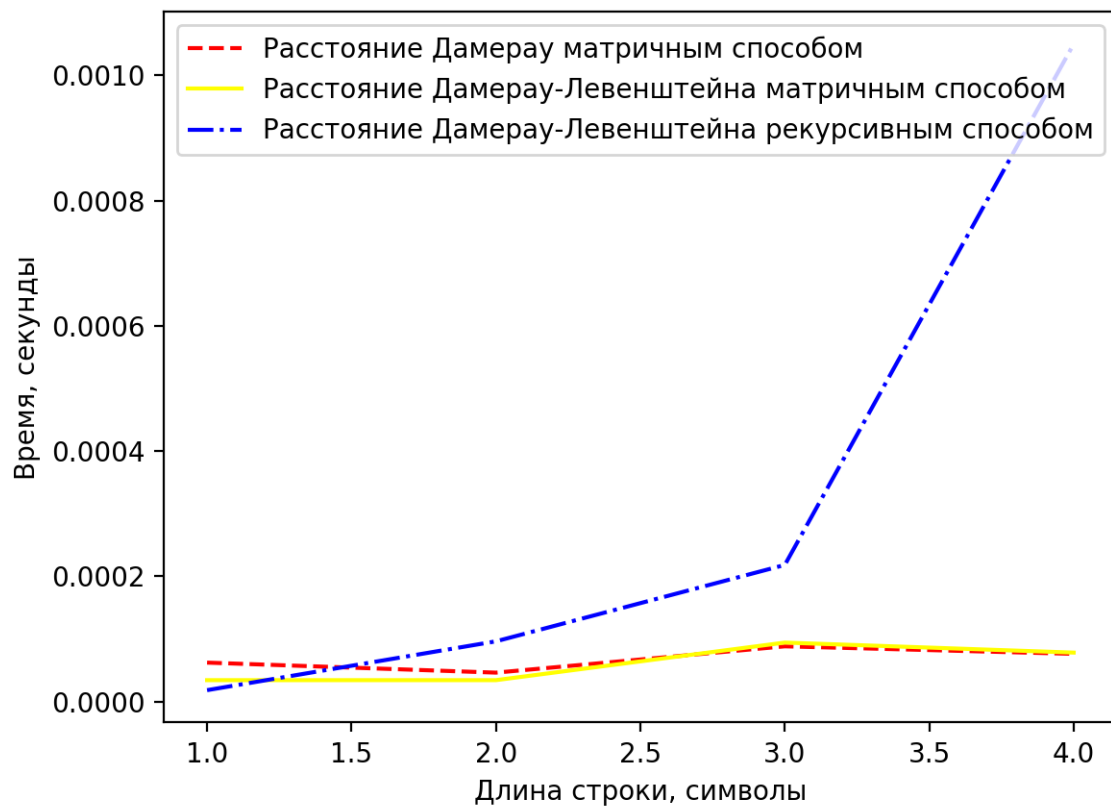


Рис. 8 - Сравнение реализации всех 3 алгоритмов

4.4 Выводы

Как видно из графиков рекурсивный алгоритм является самым медленным, гораздо быстрее использовать алгоритмы матричные.

Дамерау-Левенштейна проигрывает обычному Левенштейну на 20% на длинах слов, которые больше 200 (рисунок 6), 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

В данной лабораторной работе выполнены следующие задачи.

1. Изучен алгоритм Левенштейна и его модификации (алгоритм Дамерау-Левенштейна) для нахождения расстояния между строками.
2. Получены практические навыки реализации данных алгоритмов с использованием рекурсии и матрицы на одном из языков программирования.
3. Проведен сравнительный анализ матричного и рекурсивного алгоритмов по затрачиваемым ресурсам (зависимость времени от длины строки).
4. Экспериментальное подтверждено различие временной эффективности рекурсивной и матричной реализаций выбранного алгоритма определения расстояния между строками на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк.

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 на 20% на длинах слов, которые больше 200, но цена ошибки, в некоторых случаях, у него меньше.

Список литературы

- [1] Соснина Е. П. Введение в прикладную лингвистику: учебное пособие. – Ульяновск: УлГТУ, 2012.
- [2] Д. Гасфилд. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. – СПб.: Невский диалект, 2003.
- [3] Вычисление редакционного расстояния[Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/117063/> (дата обращения: 15.09.2019).
- [4] Нечёткий поиск в тексте и словаре[Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/114997/> (дата обращения: 15.09.2019).
- [5] В. И. Левенштейн. Двоичные коды с исправлением выпадений и вставок символа 1. – Москва, 1965.
- [6] Желудков А. В., Макаров Д. В., Фадеев П. В. Особенности алгоритмов нечеткого поиска – Москва: МГТУ Инженерный вестник, 2014.
- [7] 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).
- [8] Python[Электронный ресурс]. – Режим доступа: <https://www.python.org> (дата обращения: 15.09.2019).