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»
в

По теме: «Алгоритм Левенштейна»

Студент:
Коротков Андрей Владимирович
Группа: ИУ7-55

Преподаватели:
Волкова Лилия Леонидовна
Строганов Юрий Владимирович

Москва, 2019 г.

Содержание

Введение

В данной работе требуется изучить и применить алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна, а также получить практические навыки реализации указанных алгоритмов.

Необходимо реализовать описанные ниже алгоритмы:

1. расстояние Левенштейна (табличный способ);
2. расстояние Дамерау-Левенштейна (табличный способ);
3. расстояние Дамерау-Левенштейна (рекурсивный способ).

1. Аналитическая часть

В данном разделе будет рассмотрено описание алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна.

1.1 Описание алгоритмов

Расстояние между двумя строками Левенштейна — это минимальное количество операций вставок, удалений и замен одного символа на другой, необходимых для превращения одной строки в другую.

Эти алгоритмы активно применяются:

- для исправления ошибок в слове;
- в поисковых системах;
- в базах данных;
- в биоинформатике для сравнения генов, хромосом и белков.

1.1.1 Алгоритм Левенштейна

Для двух строк S_1 и S_2 расстояние в алгоритме Левенштейна можно посчитать по рекуррентной формуле (1)

$d(S_1, S_2) = D(S_1, S_2)$, где

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

Где разрешены операции:

1) удаления (D). Штраф - 1;
2) добавления (I). Штраф - 1;
3) замены (R). Штраф - 1;
4) совпадения (M). Штраф - 0.

	λ	И	Г	Л	А
λ	0	1	2	3	4
С	1	1	2	3	4
И	2	1	2	3	4
Л	3	2	2	2	3
А	4	3	3	3	2

Таблица 1. Пример работы преобразования слова «сила» в слово «игла»

В таблице 1 минимальное расстояние между словом «сила» и словом «игла» - 2.

1.1.2 Алгоритм Дамерау-Левенштейна

Алгоритм нахождения расстояния Дамерау-Левенштейна является модификацией алгоритма Левенштейна: к операциям вставки, удаления и

замены символов, определённых в расстоянии Левенштейна, добавлена операция транспозиции (перестановки) символов. Модификация была введена, потому что большинство ошибок пользователей при наборе текста как раз и есть транспозиция.

$$D_{S_1,S_2}(i,j) = \begin{cases} max(i,j) & \text{i=0 } or \text{ j=0} \\ min \begin{cases} d_{S_1,S_2}(i-1,j)+1 \\ d_{S_1,S_2}(i,j-1)+1 \\ d_{S_1,S_2}(i-1,j-1)+m(S_1[i],S_2[j]) \\ d_{S_1,S_2}(i-2,j-2)+1 \end{cases} & S_1[i]=S_2[j-1], S_1[i-1]=S_2[j], i>1, j>1 \quad (2) \\ min \begin{cases} d_{S_1,S_2}(i-1,j)+1 \\ d_{S_1,S_2}(i,j-1)+1 \\ d_{S_1,S_2}(i-1,j-1)+m(S_1[i],S_2[j]) \end{cases} & \end{cases}$$

1.1.3 Вывод

Были рассмотрены поверхностно алгоритмы нахождения расстояния Левенштейна и его усовершенствованный алгоритм нахождения расстояния Дамерау-Левенштейна, принципиальная разница которого — наличие транспозиции, а также области применения данных алгоритмов.

2. Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов:

- Левенштейна;
- Дамерау-Левенштейна;
- рекурсивный алгоритм Левенштейна.

2.1 Разработка алгоритмов

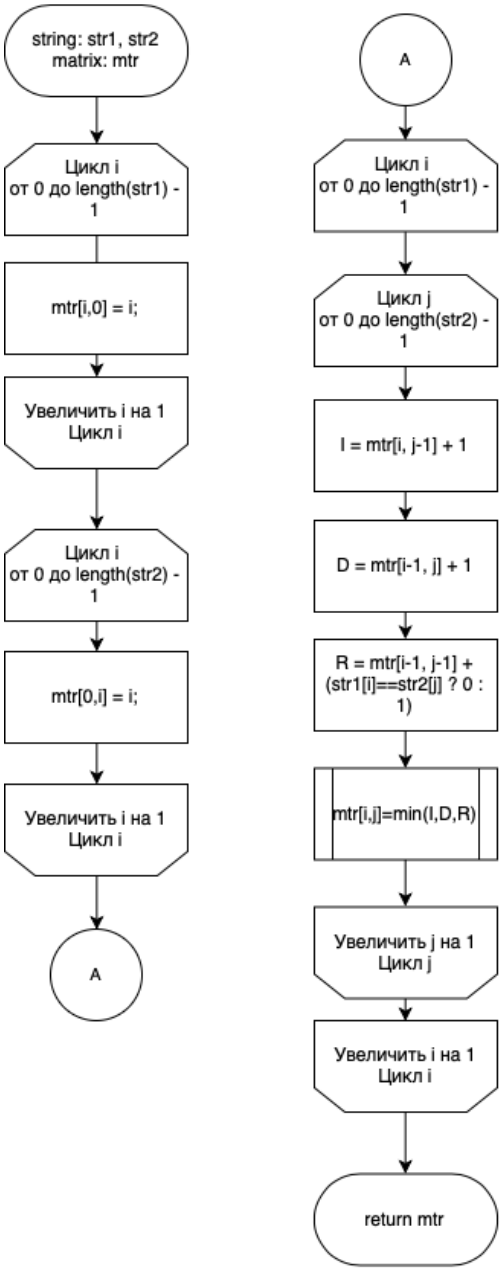

Рисунок 1. Схема алгоритма Левенштейна

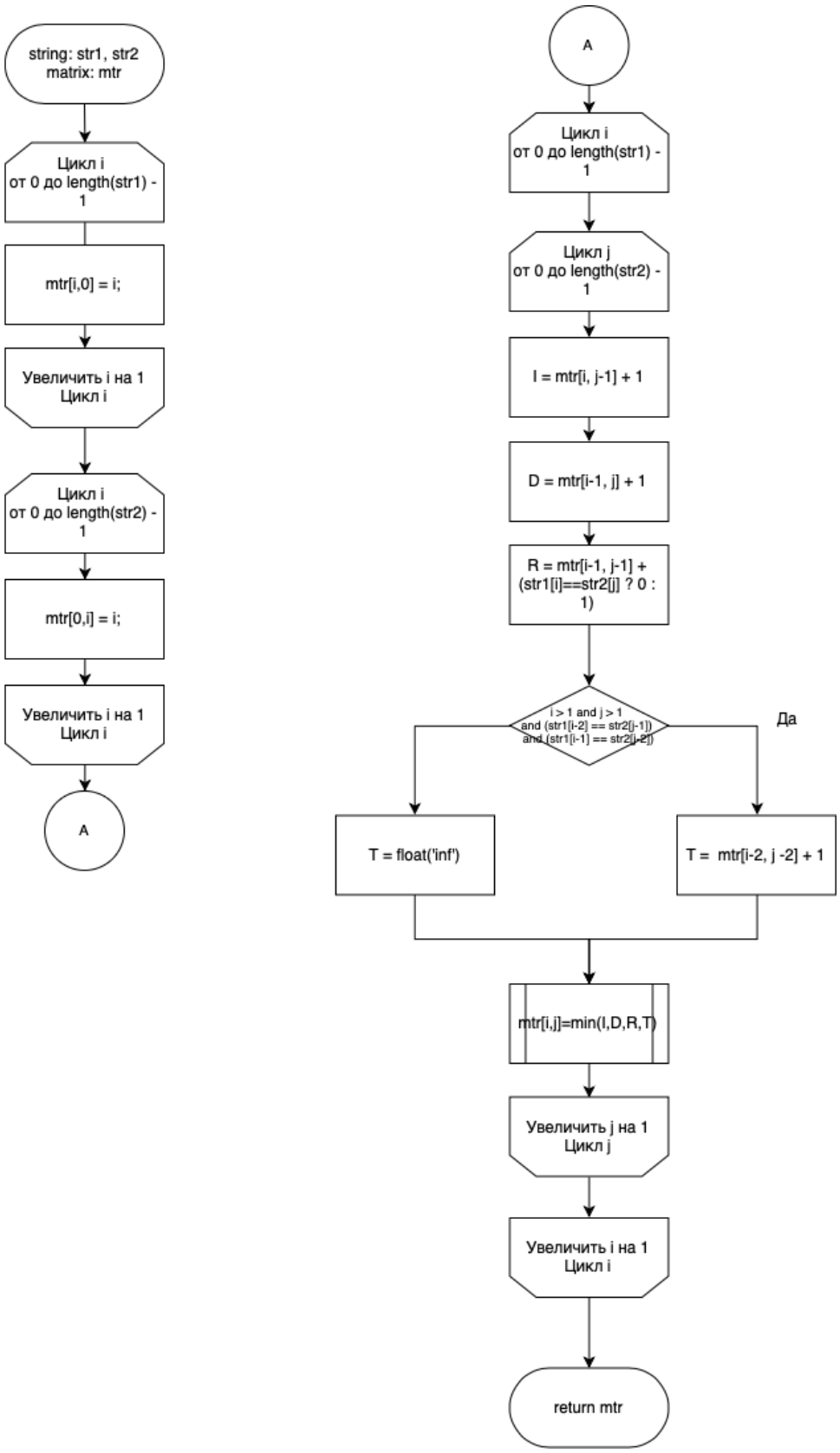

Рисунок 2. Схема алгоритма Дамерау-Левенштейна

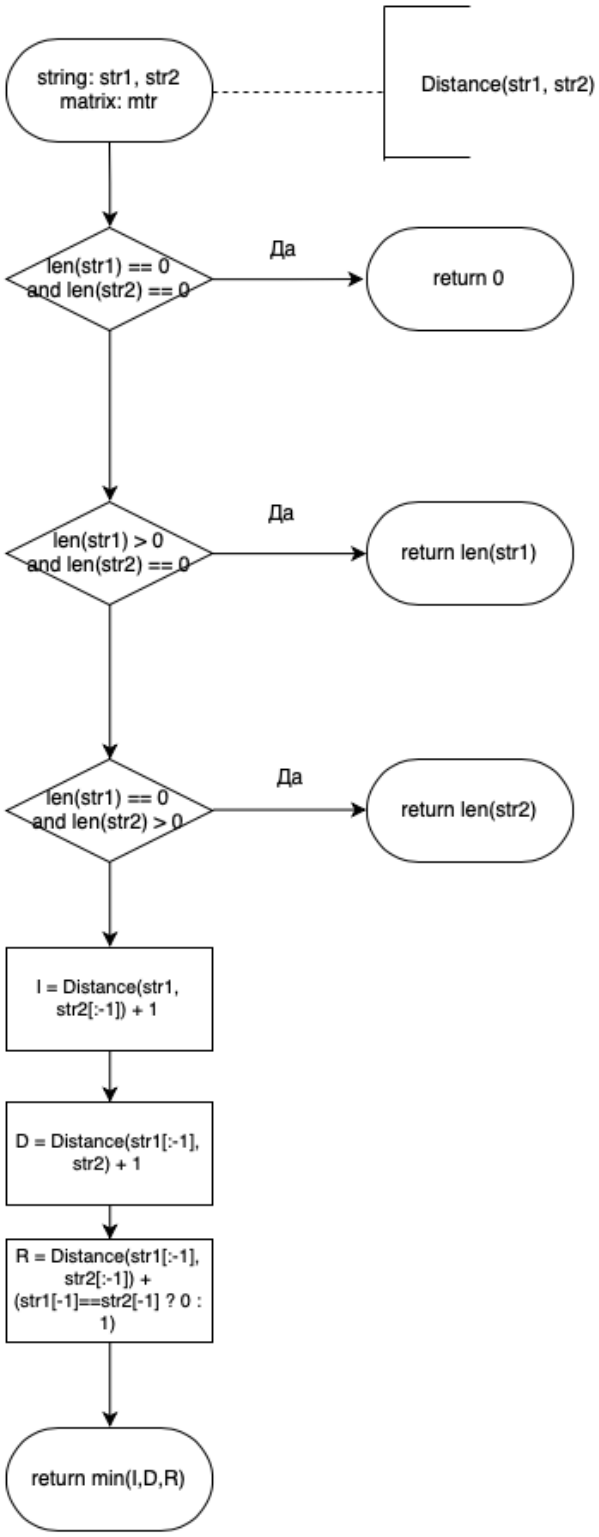

Рисунок 3. Схема рекурсивного алгоритма Левенштейна

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритмов нахождения расстояния Левенштейна, Дамерау-Левенштейна, а также рекурсивный алгоритм Левенштейна.

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: str1 - первое слово, str2 - второе слово.

Выходные данные: значение расстояния между двумя словами.

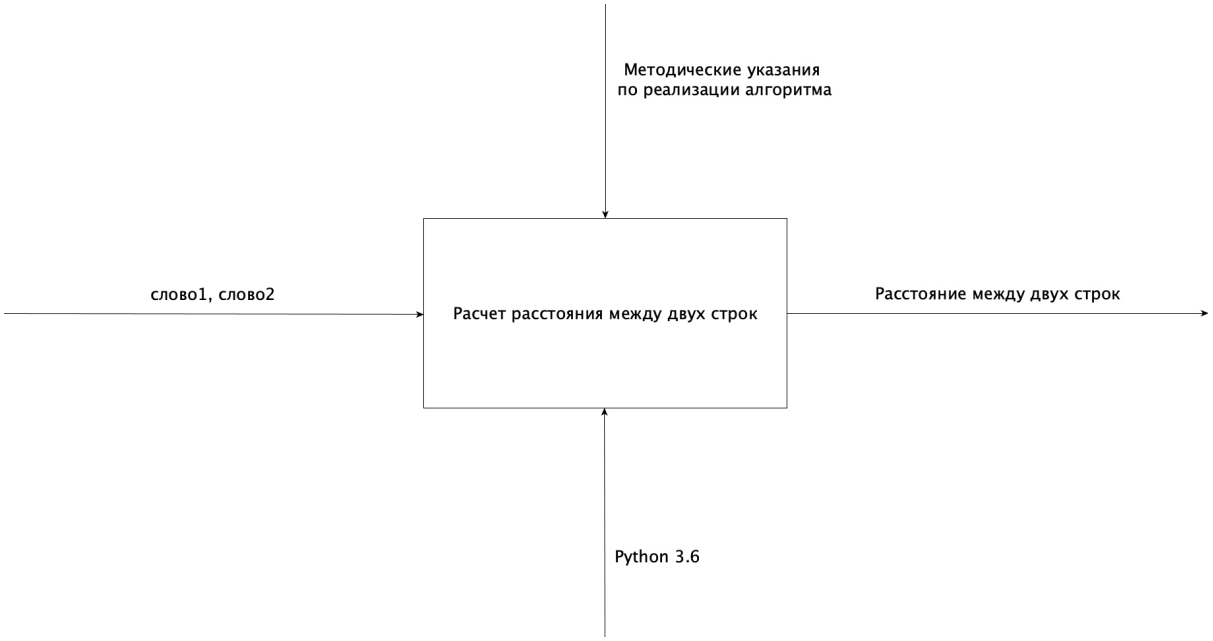

Рисунок 4. IDEF0-диаграмма, описывающая алгоритм нахождения расстояния Левенштейна.

3.2 Средства реализации

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

Для замера процессорного времени была использована библиотека, написанная на языке C и импортирована с помощью библиотеки ctypes.

```
unsigned long long getTicks(void)
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc" : "=A" (d) );
    return d;
}
```

Листинг 1. Функция получения тиков.

3.3 Листинг кода

В данном пункте представлен листинг кода, а именно:

- расстояние Левенштейна;
- расстояние Дамерау-Левенштейна;
- рекурсивное расстояние Дамерау-Левенштейна.

```
def LevesteinTable(str1, str2):
    matr = createMatr(str1,str2)

    for i in range(1, len(matr)):
        for j in range(1, len(matr[i])):
            c = matr[i-1][j-1]
            if str1[i-1] != str2[j-1]:
                c += 1

            D = min(matr[i][j-1] + 1, matr[i-1][j] + 1, c)
            matr[i][j] = D

    return matr
```

Листинг 2. Расстояние Левенштейна

```
def DameradLevensteinTable(str1, str2):
    matr = createMatr(str1,str2)

    for i in range(1, len(matr)):
        for j in range(1, len(matr[i])):
            c = matr[i-1][j-1]
            if str1[i-1] != str2[j-1]:
                c += 1

            if (i - 2 >= 0 and j - 2 >= 0 and str1[i-1] == str2[j-2]
                           and str2[j-1] == str1[i-2]):
                swap = matr[i-2][j-2] + 1
            else:
                swap = float('inf')

            D = min(matr[i][j-1] + 1, matr[i-1][j] + 1, c, swap)
            matr[i][j] = D

    return matr
```

Листинг 3. Расстояние Дамерау-Левенштейна

```
    def DameradLevensteinRec(str1, str2):
    if len(str1) == len(str2) == 0:
        return 0
    elif len(str1) > 0 and len(str2) == 0:
        return len(str1)
    elif len(str1) == 0 and len(str2) > 0:
        return len(str2)

    if str1[-1] == str2[-1]:
        M = 0
    else:
        M = 1
    I = DameradLevensteinRec(str1, str2[:-1]) + 1
    D = DameradLevensteinRec(str1[:-1], str2) + 1
    R = DameradLevensteinRec(str1[:-1], str2[:-1]) + M

    if len(str1) > 1 and len(str2) > 1 and str1[-1] == str2[-2] \
                                       and str1[-2] == str2[-1]:
        T = DameradLevensteinRec(str1[:-2], str2[:-2]) + 1
    else:
        T = float('inf')

    return min(I, D, R, T)
```

Листинг 4. Рекурсивное расстояние Дамерау-Левенштейна

3.4 Сравнительный анализ потребляемой памяти

Для проведения анализа замерим потребляемую память у разных классов:

Классы	Представление в коде	Занимаемая память (байты)
Пустой список	[ ]	40
Список с одним элементом	[0]	48
Список, заполненный нулями, длина списка 5 символов	[0, 0, 0, 0, 0]	80
Список, хранящий два пустых списка	[[ ], [ ]]	56
Список, хранящий два непустых списка	[[0,0,0], [0,0,0]]	56
Список, хранящий 5 пустых списка	[[ ], [ ], [ ], [ ], [ ]]	80
Целое число	int(100)	28
Пустая строка	str(‘’)	49

Таблица 2. Потребляемая память различными классами.

Из полученных данных можно сделать вывод о том, что для хранения элементов списка используются указатели, и нужно просматривать занимаемую память у каждого списка, находящегося внутри списка по отдельности.

В алгоритме Левенштейна используются:

Структура данных	Занимаемая память (байты)
Матрица	$40 + 8 * [len(str1) + 1] + [len(str1) + 1] * [40 + 8 * (len(str2) + 1)]$
Две вспомогательные переменные (int)	56
Два счетчика (int)	56
Передача парметров	$2 * [49 + len(str)]$

Таблица 3. Потребляемая память структурами данных в алгоритме нахождения расстояния Левенштейна.

В алгоритме Дамерау-Левенштейна используются:

Структура данных	Занимаемая память (байты)
Матрица	$40 + 8 * [len(str1) + 1] + [len(str1) + 1] * [40 + 8 * (len(str2) + 1)]$
Три вспомогательных переменных (int)	84
Два счетчика (int)	56
Передача парметров	$2 * [49 + len(str)]$

Таблица 4. Потребляемая память структурами данных в алгоритме нахождения расстояния Дамерау-Левенштейна.

В рекурсивном алгоритме Дамерау-Левенштейна используются:

Структура данных	Занимаемая память (байты)
Пять переменных для подсчета IDRT (I - insert, D - delete, R - replace, T - transponing)	140
Передача парметров	$2 * [49 + len(str)]$

Таблица 5. Потребляемая память структурами данных в рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна.

Причем, максимальная глубина рекурсивного вызова функции - максимальная длина из двух слов.

3.4.1 Оценка потребляемой памяти на словах длинной 4 и 1000 символов

Оценим алгоритмы на словах длинной 4 символа:

Структура данных	Занимаемая память (байты)
Матрица	480
Две вспомогательные переменные (int)	56
Два счетчика (int)	56
Передача парметров	106
Сумма данных	698

Таблица 6. Потребляемая память структурами данных в алгоритме нахождения расстояния Левенштейна.

Структура данных	Занимаемая память (байты)
Матрица	480
Три вспомогательных переменных (int)	84
Два счетчика (int)	56
Передача парметров	106
Сумма данных	726

Таблица 7. Потребляемая память структурами данных в алгоритме нахождения расстояния Дамерау-Левенштейна.

Структура данных	Занимаемая память (байты)
Пять переменных для подсчета IDRT (I - insert, D - delete, R - replace, T - transponing)	140 * 4 (максимальная глубина вызовов функции) = **560**
Передача парметров	106 * 4/2 (усредненное значение) = **212**
Сумма данных	772

Таблица 8. Потребляемая память структурами данных в рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна.

Оценим алгоритмы на словах длинной 1000 символов:

Структура данных	Занимаемая память (байты)
Матрица	8 064 096
Две вспомогательные переменные (int)	56
Два счетчика (int)	56
Передача парметров	2 098
Сумма данных	8 066 306

Таблица 9. Потребляемая память структурами данных в алгоритме нахождения расстояния Левенштейна.

Структура данных	Занимаемая память (байты)
Матрица	8 064 096
Три вспомогательных переменных (int)	84
Два счетчика (int)	56
Передача парметров	2 098
Сумма данных	8 066 334

Таблица 10. Потребляемая память структурами данных в алгоритме нахождения расстояния Дамерау-Левенштейна.

Структура данных	Занимаемая память (байты)
Пять переменных для подсчета IDRT (I - insert, D - delete, R - replace, T - transponing)	140 * 1000 (максимальная глубина вызовов функции) = **140 000**
Передача парметров	2098 * 1000/2 (усредненное значение) = **1 049 000**
Сумма данных	1 189 000

Таблица 11. Потребляемая память структурами данных в рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна.

Таким образом рекурсивный алгоритм выигрывает по памяти в сравнении с итеративными алгоритмами при обработке длинных строк.

3.5 Вывод

В данном разделе была представлена реализация алгоритмов нахождения расстояния Левенштейна, Дамерау-Левенштейна, а также рекурсивный алгоритм Дамерау-Левенштейна. Произведен анализ по потребляемой памяти каждым из алгоритмов в входе которого был сделан вывод о том, что рекурсивный алгоритм выигрывает по памяти при обработке длинных строк.

4. Экспериментальная часть

В данном разделе будут рассмотрены примеры работы программы. Также будет проведен эксперимент и сравнительный анализ данных.

4.1 Примеры работы

Примеры работы операций:

- удаление;
- замена;
- добавление;
- перестановка.

Также проверки:

- пустая строка;
- равенство строк.

На рисунках ниже будут приведены тесты с целью демонстрации корректности работы программы.

```
Слово1 = "" Слово2 = ""
Лев. табл. D = 0
Дам.-Лев. табл. D =  0
Дам.-Лев. рек. D = 0
```

Рисунок 5. Пустые слова

```
Слово1 = "hello" Слово2 = "hell"
Лев. табл. D = 1
Дам.-Лев. табл. D =  1
Дам.-Лев. рек. D = 1
```

Рисунок 6. Удаление символа

```
Слово1 = "base" Слово2 = "bose"
Лев. табл. D = 1
Дам.-Лев. табл. D =  1
Дам.-Лев. рек. D = 1
```

Рисунок 7. Замена символа

```
Слово1 = "" Слово2 = "exit"
Лев. табл. D = 4
Дам.-Лев. табл. D = 4
Дам.-Лев. рек. D = 4
```

Рисунок 8. Добавление символов

```
Слово1 = "IlovBMSTU" Слово2 = "IlovBMSTU"
Лев. табл. D = 0
Дам.-Лев. табл. D = 0
Дам.-Лев. рек. D = 0
```

Рисунок 9. Одинаковые строки

```
Слово1 = "Жора" Слово2 = "Жроа"
Лев. табл. D = 2
Дам.-Лев. табл. D = 1
Дам.-Лев. рек. D = 1
```

Рисунок 10. Перестановка символов

4.2 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;

4.3 Сравнительный анализ на материале экспериментальных данных

Сравнение времени работы алгоритмов Левенштейна и Дамерау-Левенштейна:

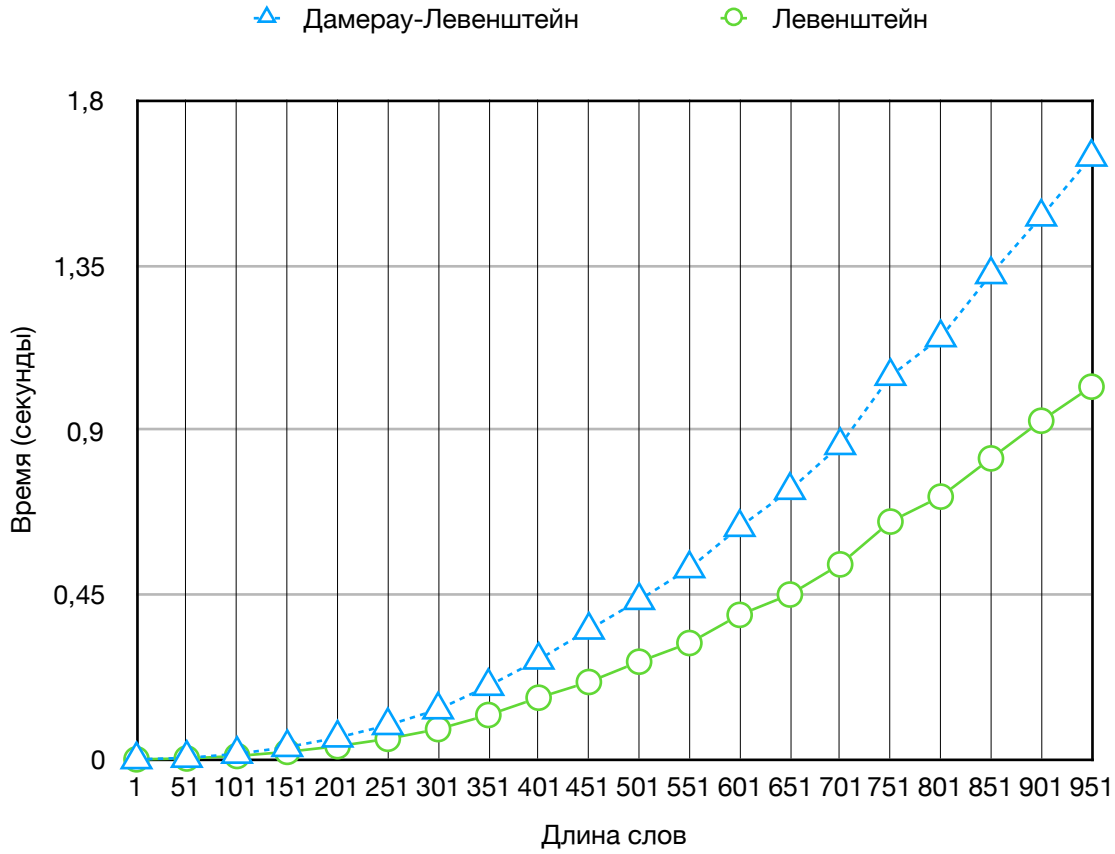

Рисунок 11. График работы алгоритмов Левенштейна и Дамерау-Левенштейна (ось абсцисс - время работы алгоритмов(секунды), ось ординат - длина слов).

Алгоритм Дамерау-Левенштейна работает дольше, так как имеет более сложную логику. Заметим, что оба алгоритма имеют одинаковый характер.

Сравнение времени работы рекурсивного и итеративного алгоритмов Дамерау-Левенштейна:

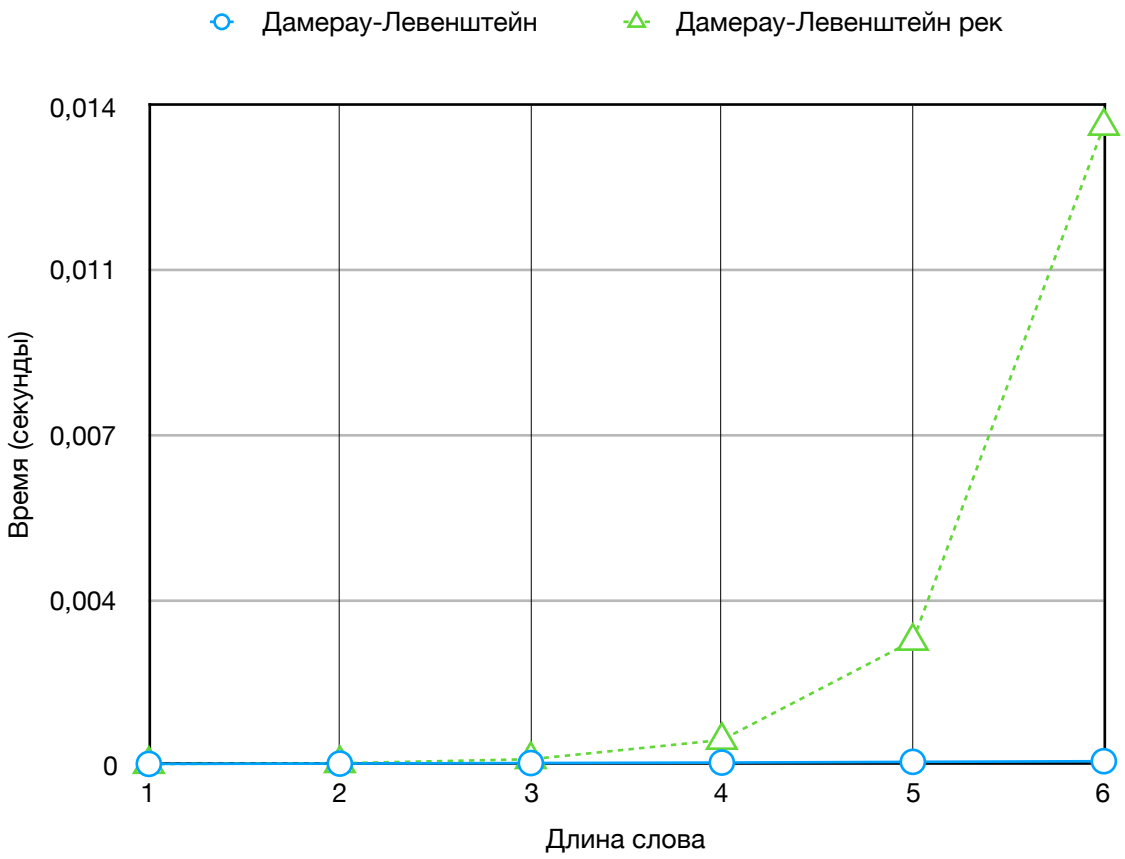

Рисунок 12. График работы работы рекурсивного и итеративного алгоритмов Дамерау-Левенштейна (ось абсцисс - время работы алгоритмов(секунды), ось ординат - длина слов)

Дальнейшие замеры не имеют смыла, так как время выполнения рекурсивного алгоритма увеличивается экспоненциально, пропорционально количеству рекурсивных вызовов. При одинаковом размере строк рекурсивный алгоритм сильно проигрывает итеративному.

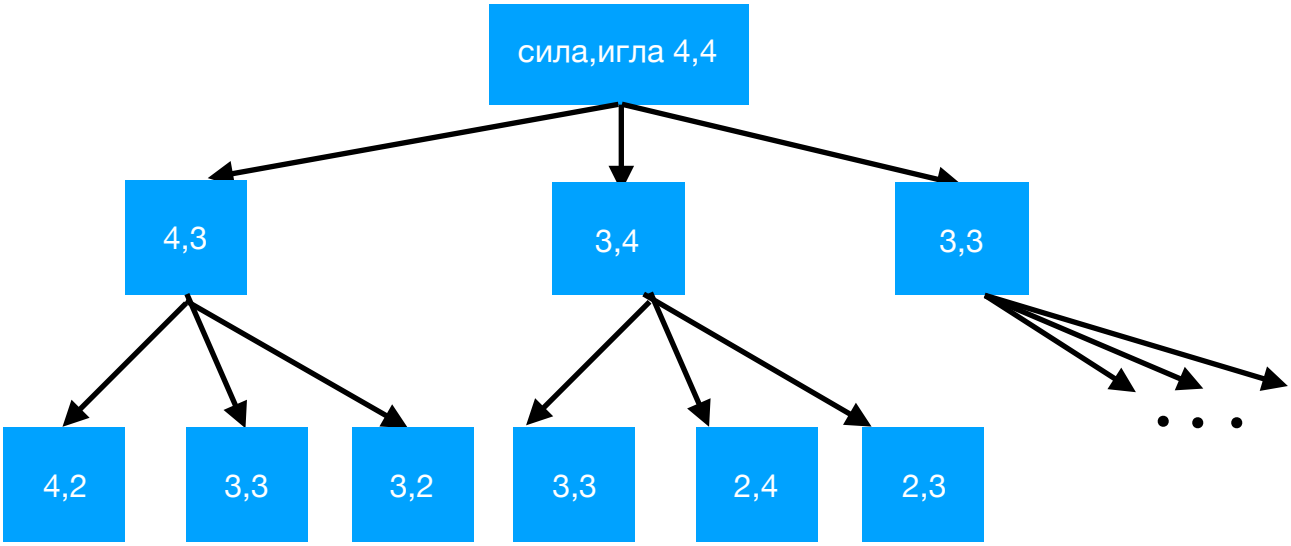

Рисунок 13. Схема рекурсивного вызова в алгоритме Левенштейна (описывает часть работы алгоритма)

```
Levestein Table:
    и г л а
  0 1 2 3 4
с 1 1 2 3 4
и 2 1 2 3 4
л 3 2 2 2 3
а 4 3 3 3 2

Damerad Levenstein Table:
    и г л а
  0 1 2 3 4
с 1 1 2 3 4
и 2 1 2 3 4
л 3 2 2 2 3
а 4 3 3 3 2

Levenstein Rec: 2
Кол-во вызовов функции при рекурсии:  481
```

Рисунок 14. Демонстрация кол-ва вызовов функций при рекурсии

На рисунке (13) видно, что рекурсивная реализация алгоритма поиска расстояния между строками использует многократный вызов функций, причем функции могут вызываться с одинаковыми аргументами. Например, 3 раза вызывается функция с аргументами 3 и 3. При табличном способе создается матрица, размерность которой определяется длинной слов, что позволяет оптимизировать кол-во необходимой памяти. В отличии от табличного способа, рекурсивный требует много памяти (кол-во вызовов функций при рекурсии 481, при длине слов: 4 символа). Очевидно, что рекурсивный метод проигрывает табличному по памяти и по скорости, поэтому использование такого метода становится нецелесообразным.

4.4 Вывод

В данном разделе был поставлен эксперимент по замеру времени выполнения каждого алгоритма. По итогам замеров алгоритм нахождения расстояния Левенштейна оказался самым быстрым (на 30 % быстрее, чем алгоритм Дамерау-Левенштейна), а самым медленным — рекурсивный алгоритм Дамерау-Левенштейна (на 98% медленнее, чем итеративный алгоритм Дамерау-Левенштейна). Также была продемонстрирована правильность реализации каждого алгоритма с помощью тестов.

Заключение

В ходе работы были изучены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна (рекурсивный и итеративный). Выполнено сравнение рекурсивного и итеративного алгоритмов Дамерау-Левенштейна. В ходе исследования было установлено, что итеративные алгоритмы Левенштейна и Дамерау-Левенштейна занимают намного больше памяти при обработке длинных строк, чем рекурсивная реализация тех же алгоритмов (при длине 1000 символов, рекурсивный алгоритм потребляет в 7 раз меньше памяти, чем итерационные). Изучены зависимости времени выполнения алгоритмов от длин строк. При сравнении времени выполнения алгоритмов, стало понятно, что самый быстрый среди рассматриваемых алгоритмов — алгоритм Левенштейна. Также реализован программный код продукта.

Список использованных источников

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.
2. Нечёткий поиск в тексте и словаре // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/114997/ (дата обращения: 10.09.19).
3. Вычисление расстояния Левенштейна // [Электронный ресурс]. Режим доступа: https://foxford.ru/wiki/informatika/vychislenie-rasstoyaniya-levenshteyna (дата обращения: 10.09.19).
4. Нечеткий поиск, расстояние левенштейна алгоритм // [Электронный ресурс]. Режим доступа: https://steptosleep.ru/antananarivo-106/ (дата обращения: 10.09.19).