Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

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

ОТЧЁТ ПО АНАЛИЗУ АЛГОРИТМОВ
к лабораторной работе №1 на тему:

Расстояние Левенштейна. Расстояние Дамерау-Левенштейна

Студент: Якубаускайте М.А. ИУ7-55

Москва 2018

Содержание

1 Введение

Цель данной лабораторной работы - изучение алгоритмов нечеткого поиска, которые используют:

а) для выявления ошибок в словах (в поисковиках, различных BD (базах для хранения данных), в программах, которые разработаны для того, чтобы распознать текст

б) для простого сравнения текстовых файлов

в) в биохимической информатике, данный метод широко используют для сравнения генов, белков, а также хромосом

В данной работе рассматриваются 3 алгоритма: алгоритм Левенштейна, Дамерау-Левенштейна и рекурсивная реализация алгоритма Левенштейна.

2 Аналитическая часть

2.1 Описание алгоритмов

2.1.1 Расстояние Левенштейна

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

Расстояние Левенштейна применяется для исправления ошибок в слове поискового запроса (поисковые системы, поиск по базе данных, распознавание рукописного текста и устной речи).

Расстояние Левенштейна использует понятие цена операции. Операций всего 3: замена, вставка, удаления. Задача нахождения расстояния сводится к нахождению последовательности замен, минимизирующих суммарную цену (штраф). Имеем:

а) cost(a, b) - цена замены символа "a"на "b"= 1, "a"$\neq$ "b"

б) cost(ϵ, b) - цена вставки символа "b"= 1

в) cost(a, ϵ) - цена удаления символа "a"= 1

г) cost(a, a) - цена совпадения символов = 0

2.1.2 Понятие расстояния Дамерау-Левенштейна

Определение расстояния по Левенштейну имеет недостатки:

а) при перестановке местами слов или частей слов получаются большие расстояния

б) расстояния между совершенно разными короткими словами будут меньше в отличии от расстояния между двумя длинными и очень похожими, но длинными словами

Расстояние Дамерау-Левенштейна - это модификация метода Левенштейна, которая также учитывает транспозицию символов (перестановка двух соседних элементов).

Пусть имеются две строки $s1$ и $s2$ длиной m и n. Тогда расстояние Левенштейна ($D(s1, s2)$) можно подсчитать по рекурентной формуле:

$D(s1[1...m], s2[1...n]) = min(D(s1[1...m-1], s2[1...n])+1, D(s1[1...m], s2[1...n-1])+1, D(s1[1...m-1], s2[1...n-1])+\alpha)$, где $\alpha = 0$, если $s1[m] = s2[n]$, иначе $\alpha = 1$

Классификация разрешенных операций и штрафы на выполнение операции:

а) Замена символа - $R = 1$

б) Вставка символа - $I = 1$

в) Удаление символа - $D = 1$

г) Совпадение символа - $M = 0$

В алгоритме Дамерау-Левенштейна добавляется еще одна операция - перестановка символа $x = 1$. А в рекурентную формулу добавляется еще один член:

$D(s1[1...m-1], s2[1...n-1] + \beta$, где $\beta = 1$, если $s1[m-1] = s2[n]$, иначе $\beta = 0$

2.1.3 Область применения алгоритмов.

а) Исправление ошибок в слове в:
   1) поисковых системах
   2) базах данных
   3) при вводе текста
   4) при автоматическом распознавании речи и отсканированного текста

б) Для сравнения содержимого строк текстовых файлов

3 Констукторский раздел

3.1 Разработка алгоритмов

3.1.1 Математическое описание

Пусть имеются две строки $s1$ и $s2$ длиной m и n. Тогда расстояние Левенштейна ($D(s1, s2)$) можно подсчитать по рекурентной формуле:

$D(s1[1...m], s2[1...n]) = min(D(s1[1...m-1], s2[1...n]) + 1, D(s1[1...m], s2[1...n-1]) + 1, D(s1[1...m-1], s2[1...n-1]) + \alpha)$, где $\alpha = 0$, если $s1[m] = s2[n]$, иначе $\alpha = 1$

Классификация разрешенных операций и штрафы на выполнение операции:

а) Замена символа - $R = 1$
б) Вставка символа - $I = 1$
в) Удаление символа - $D = 1$
г) Совпадение символа - $M = 0$

В алгоритме Дамерау-Левенштейна добавляется еще одна операция - перестановка символа $x = 1$. А в рекурентную формулу добавляется еще один член:

$D(s1[1...m-1], s2[1...n-1] + \beta$, где $\beta = 1$, если $s1[m-1] = s2[n]$, иначе $\beta = 0$

3.1.2 Блок-схемы алгоритмов

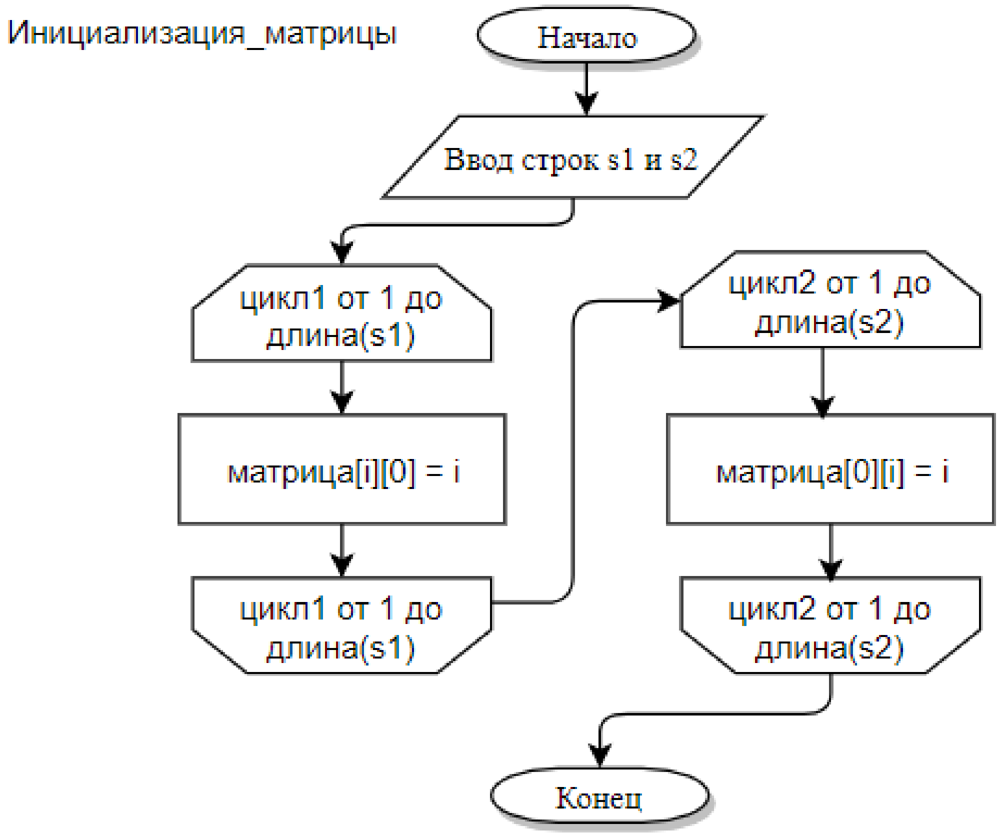

Рисунок 3.1 — Процесс инициализации матрицы.

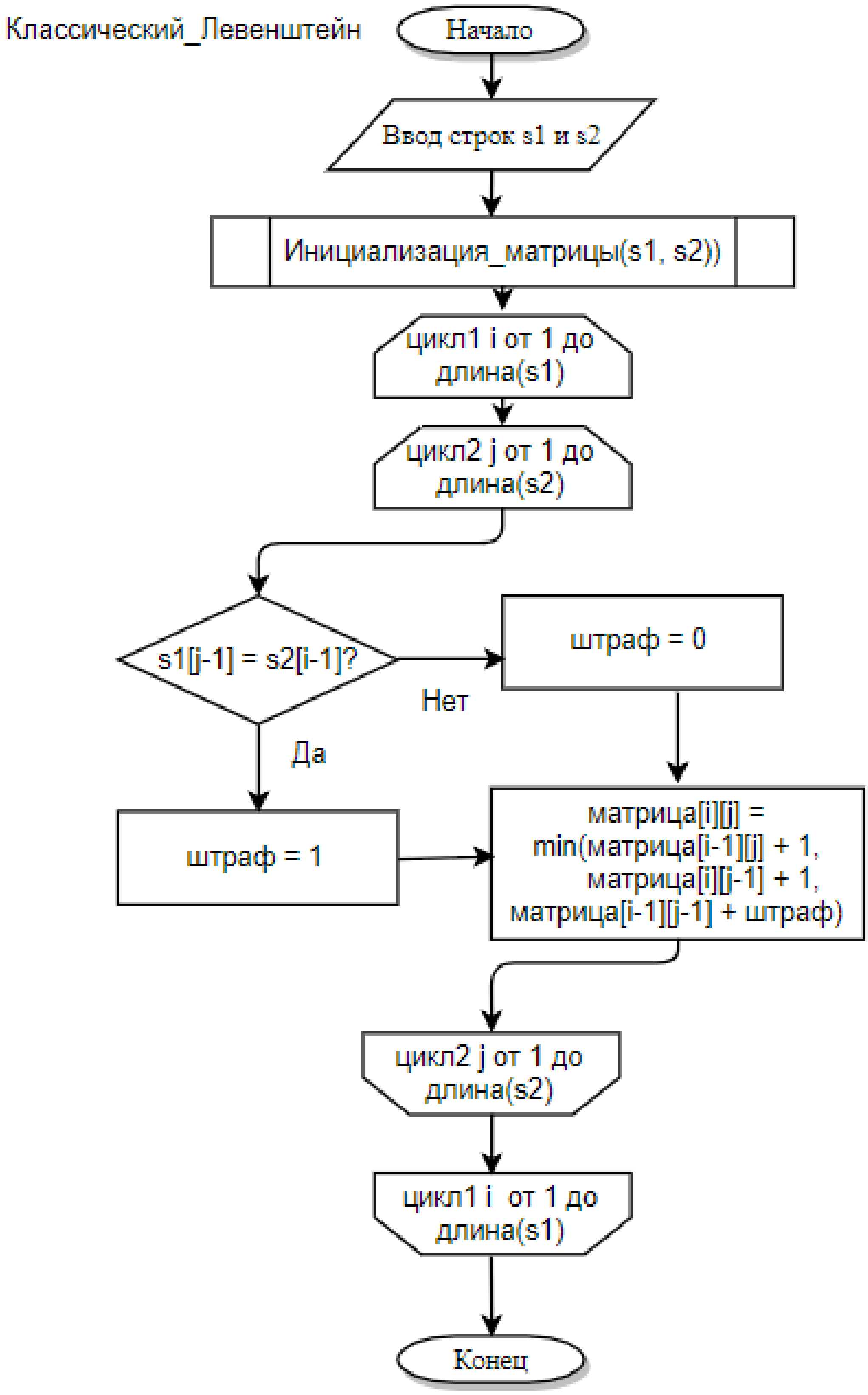

Рисунок 3.2 — Классический алгоритм Левенштейна.

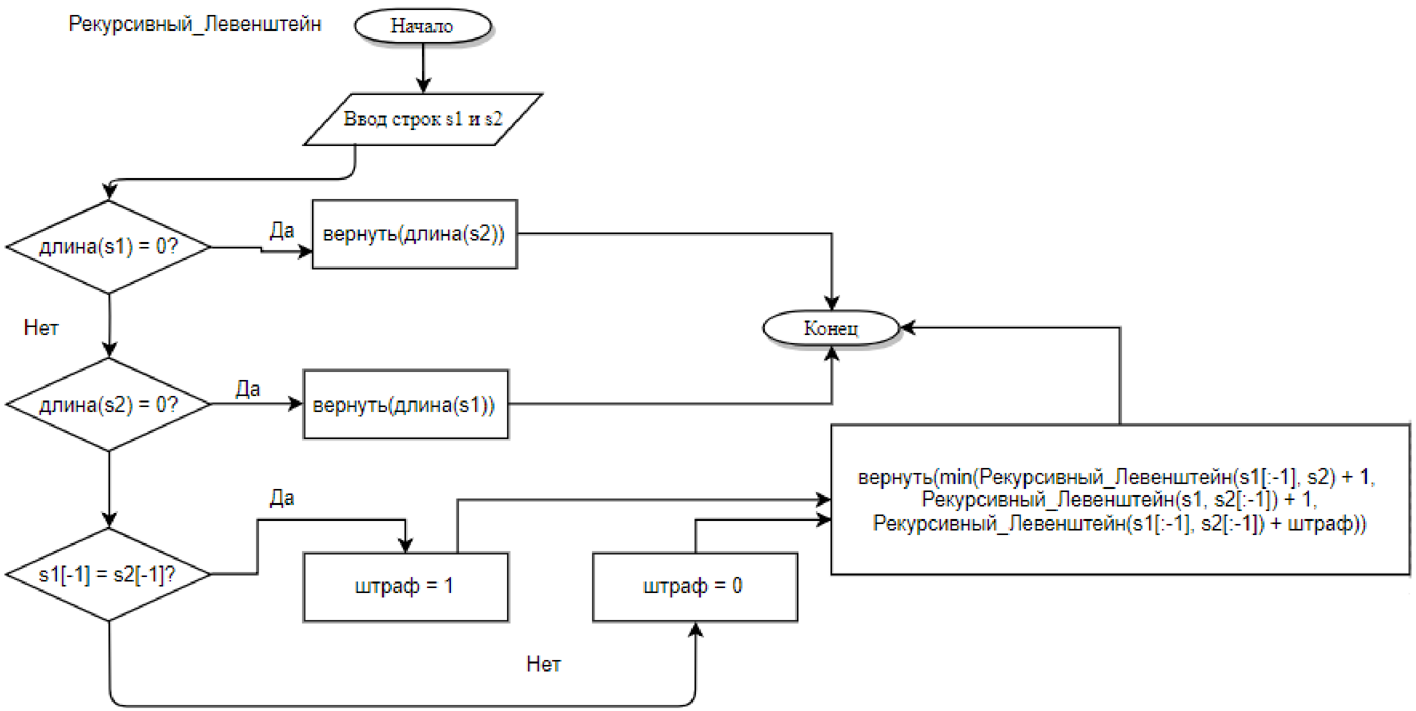

Рисунок 3.3 — Рекурсивный алгоритм Левенштейна.

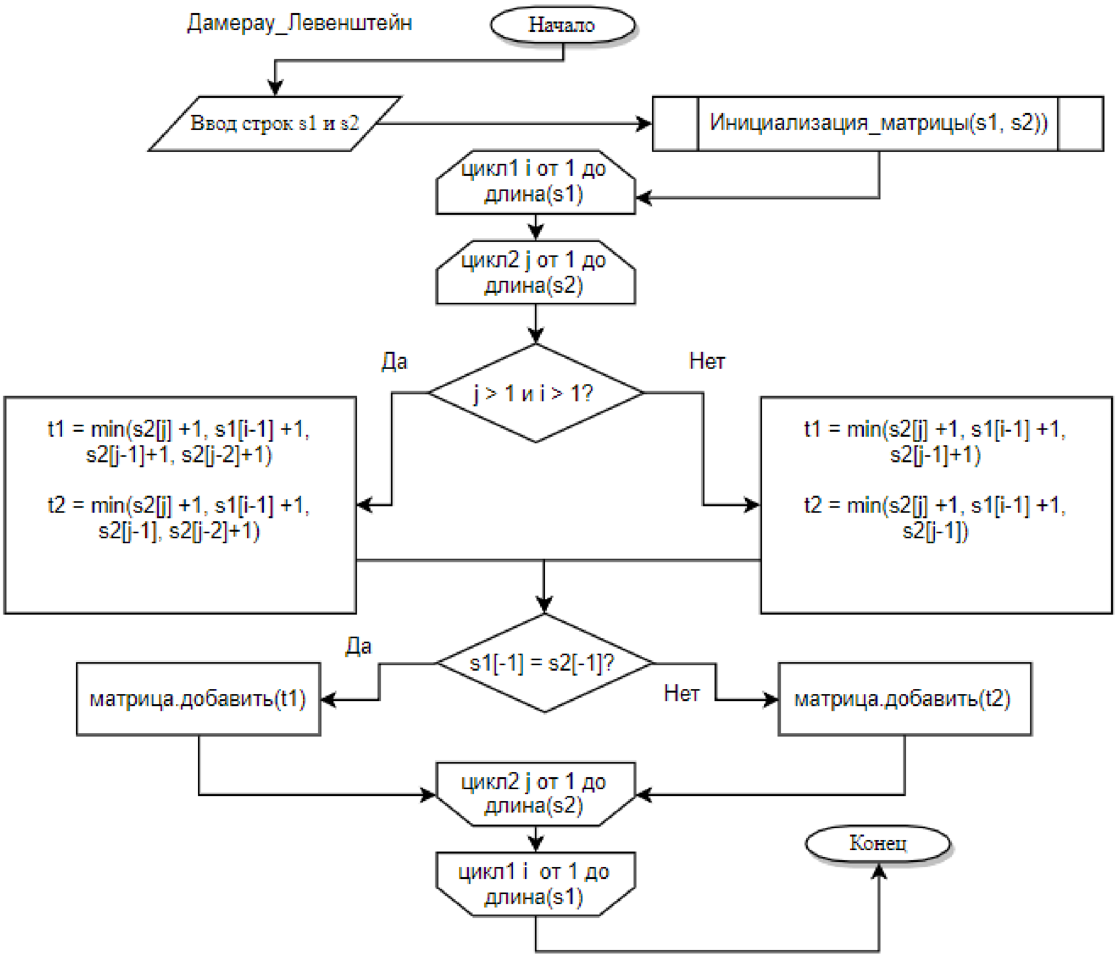

Рисунок 3.4 — Классический алгоритм Дамерау-Левенштейна.

3.2 Сравнительный анализ рекурсивной и нерекурсивной реализации.

Для временного сравнительного анализа рекурсивной реализации алгоритма Левенштейна, нерекурсивной реализации алгоритма Левенштейна и алгоритма Дамерау-Левенштейна использовалась методика многочисленных запусков 2 типов. Сначала проводились вычисления по времени для слов одинаковой длины (7 символов), количество запусков было более 100. После вычислялись средние значения по времени и при помощи графических средств используемого языка строилась первая столбчатая диаграмма. Второй способ полагал исследование слов различной длины. Максимальная длина слова была 10, минимальная - 4, количество запусков - 10 для каждой пары разности длин.

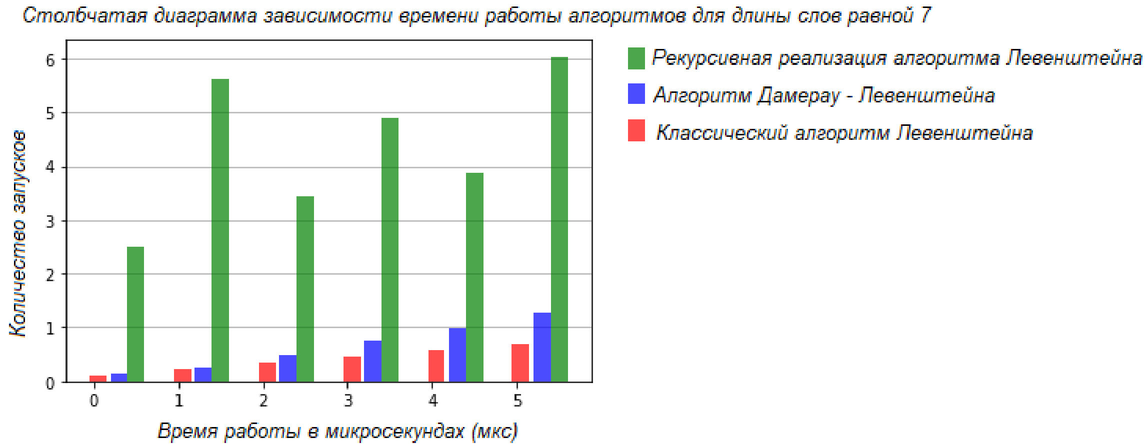

Рисунок 3.5 — Пример тестового замера количества вызовов.

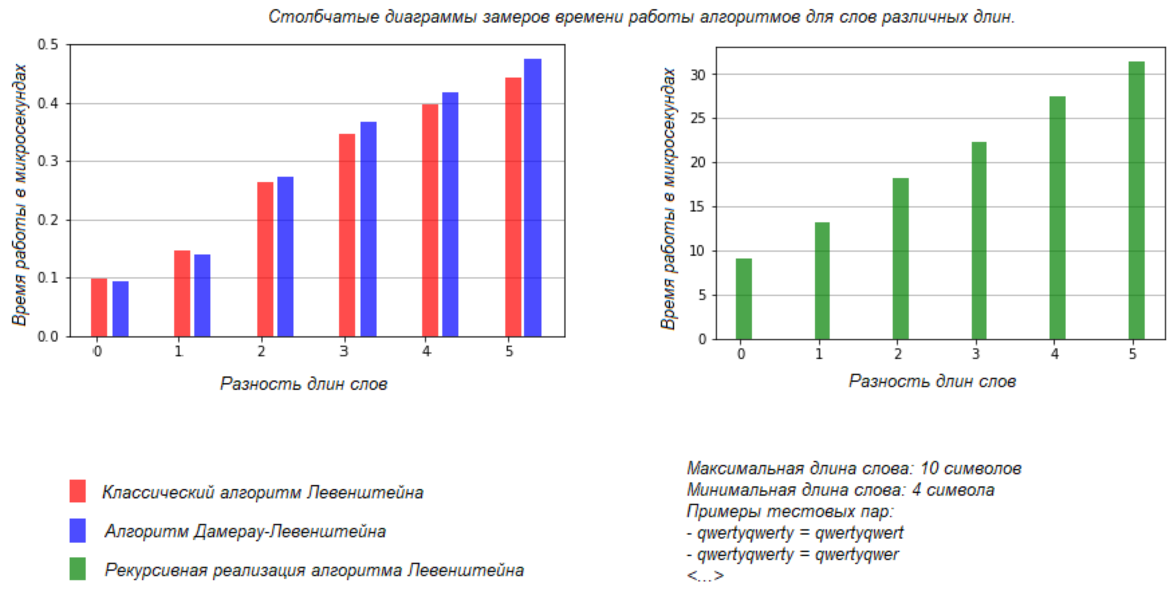

Рисунок 3.6 — Таблица тестовых результатов

Из столбчатых диаграмм видно, что худшие показатели по времени в первом и втором случае у рекурсивной реализации алгоритма Левенштейна. Алгоритм Левенштейна находит расстояние между словами длиной 10 и 4 символа за 0.5 микросекунды, а рекурсивная версия алгоритма затрачивает на решение той же задачи уже 31 микросекунду. Таким образом вычислительная разница составляет 30.5 микросекунд. Можно сделать вывод, что рекурсивную версию алгоритма Левенштейна не стоит применять для поиска соотвествий в крупных базах данных, так как он существенно может замедлить работу.

Такое замедление по времени вызвано частыми обращениями к рекурсивным запросам, которые можно проследить на примере анализа вызова рекурсивной реализации:

```
январь март
4
LD was called 1933 times!
[("('я', '')({}", 360), ("('', 'м')({}", 292), ("('я', 'м')({}", 231), ("('', '')({}", 231),
("('ян', '')({}", 192), ("('ян', 'м')({}", 129), ("('янв', '')({}", 88), ("('', 'ма')({}", 72),
("('янв', 'м')({}", 63), ("('я', 'ма')({}", 61), ("('ян', 'ма')({}", 41), ("('янва', '')({}", 32),
("('янв', 'ма')({}", 25), ("('янва', 'м')({}", 25), ("('янва', 'ма')({}", 13), ("('', 'мар')({}",
12), ("('я', 'мар')({}", 11), ("('ян', 'мар')({}", 9), ("('январ', '')({}", 8), ("('янв', 'мар')
{}", 7), ("('январ', 'м')({}", 7), ("('янва', 'мар')({}", 5), ("('январ', 'ма')({}", 5),
("('январ', 'мар')({}", 3), ("('январь', 'март')({}", 1), ("('январ', 'март')({}", 1), ("('янва',
'март')({}", 1), ("('янв', 'март')({}", 1), ("('ян', 'март')({}", 1), ("('я', 'март')({}", 1),
("('', 'март')({}", 1), ("('январь', 'мар')({}", 1), ("('январь', 'ма')({}", 1), ("('январь', 'м')
{}", 1), ("('январь', '')({}", 1)]
```

Рисунок 3.7 — Пример тестового замера количества вызовов.

Данные количества вызовов рекурсии для следующих слов представлены в таблице ниже:

s1	*s1*	*Кол-во вызовов*
январь	март	1933
матр	март	481
октябрь	Октябрь	72958
июнь	ноябрь	162817
ферваль	ьлавреф	72958

Рисунок 3.8 — Таблица тестовых результатов

4 Технологический раздел

4.1 Требования к программному обеспечению

Данная программа разрабатывалась на языке Python 3.6, поддерживаемом многими операционными системами. Для запуска программы необходим интреприатор, поддерживающий Python 3.6 для 64-битной системы.

4.2 Листинг кода

Классический алгоритм Левенштейна.

```
rows = len(s1) + 1
cols = len(s2) + 1
dist = [[0 for x in range(cols)] for x in range(rows)]

for i in range(1, rows):
    dist[i][0] = i

for i in range(1, cols):
    dist[0][i] = i

for col in range(1, cols):
    for row in range(1, rows):
        cost = 0 if s1[row-1] == s2[col-1] else 1

        dist[row][col] = min(dist[row-1][col] + 1,
                             dist[row][col-1] + 1,
                             dist[row-1][col-1] + cost)
return dist[row][col]
```

Рекурсивный алгоритм Левенштейна.

```
if s1 == "": return len(s2)

if s2 == "": return len(s1)

cost = 0 if s1[-1] == s2[-1] else 1

result = min([Lrecursion(s1[:-1], s2) + 1,
        Lrecursion(s1, s2[:-1]) + 1,
        Lrecursion(s1[:-1], s2[:-1]) + cost])
return result
```

Алгоритм Дамерау-Левенштейна.

```
rows = len(s1)
cols = len(s2)

preprs = None
```

```python
precur = [x for x in range(cols + 1)]
cur = [1]

for i in range(1, rows + 1):
    for j in range(1, len(precur)):
        if j > 1 and i > 1:
            correct = min(precur[j] + 1,
                          cur[j - 1] + 1,
                          precur[j - 1] + 1,
                          preprs[j - 2] + 1)

            incorrect = min(precur[j] + 1,
                            cur[j - 1] + 1,
                            precur[j - 1],
                            preprs[j - 2] + 1)

            cur.append(correct) if s1[i - 1] != s2[j - 1] else
                cur.append(incorrect)

        else:
            correct = min(precur[j] + 1,
                          cur[j - 1] + 1,
                          precur[j - 1] + 1)

            incorrect = min(precur[j] + 1,
                            cur[j - 1] + 1,
                            precur[j - 1])

            cur.append(correct) if s1[i - 1] != s2[j - 1] else
                cur.append(incorrect)

        preprs = precur
        precur = cur
        cur = [i + 1]
return precur[-1]
```

5 Исследовательский раздел

5.1 Пример работы

На нижеприведенном изображении представлены матрицы, выводимые нерекурсивными реализациями.

```
матр март
[0, 1, 2, 3, 4]
[1, 0, 1, 2, 3]
[2, 1, 0, 1, 2]
[3, 2, 1, 1, 1]
[4, 3, 2, 1, 2]
2
```

```
октябрь Октябрь
[0, 1, 2, 3, 4, 5, 6, 7]
[1, 1, 2, 3, 4, 5, 6, 7]
[2, 2, 1, 2, 3, 4, 5, 6]
[3, 3, 2, 1, 2, 3, 4, 5]
[4, 4, 3, 2, 1, 2, 3, 4]
[5, 5, 4, 3, 2, 1, 2, 3]
[6, 6, 5, 4, 3, 2, 1, 2]
[7, 7, 6, 5, 4, 3, 2, 1]
1
```

```
ферваль феерваль
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[1, 0, 1, 2, 3, 4, 5, 6, 7]
[2, 1, 0, 1, 2, 3, 4, 5, 6]
[3, 2, 1, 1, 1, 2, 3, 4, 5]
[4, 3, 2, 2, 2, 1, 2, 3, 4]
[5, 4, 3, 3, 3, 2, 1, 2, 3]
[6, 5, 4, 4, 4, 3, 2, 1, 2]
[7, 6, 5, 5, 5, 4, 3, 2, 1]
1
```

```
ферваль ьлавреф
[0, 1, 2, 3, 4, 5, 6, 7]
[1, 1, 2, 3, 4, 5, 6, 6]
[2, 2, 2, 3, 4, 5, 5, 6]
[3, 3, 3, 3, 4, 4, 5, 6]
[4, 4, 4, 4, 3, 4, 5, 6]
[5, 5, 5, 4, 4, 4, 5, 6]
[6, 6, 5, 5, 5, 5, 5, 6]
[7, 6, 6, 6, 6, 6, 6, 6]
6
```

```
123цук 12цук3
[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, 2, 3, 3]
[4, 3, 2, 1, 2, 3, 4]
[5, 4, 3, 2, 1, 2, 3]
[6, 5, 4, 3, 2, 1, 2]
2
```

Рисунок 5.1 — Таблица тестовых результатов алгоритма Левенштейна.

```
матр март
[0, 1, 2, 3, 4]
[1, 0, 1, 2, 3]
3
```

```
октябрь Октябрь
[0, 1, 2, 3, 4, 5, 6, 7]
[1, 1, 2, 3, 4, 5, 6, 7]
7
```

```
ферваль феерваль
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[1, 0, 1, 2, 3, 4, 5, 6, 7]
7
```

```
ферваль ьлавреф
[0, 1, 2, 3, 4, 5, 6, 7]
[1, 1, 2, 3, 4, 5, 6, 6]
6
```

```
123цук 12цук3
[0, 1, 2, 3, 4, 5, 6]
[1, 0, 1, 2, 3, 4, 5]
5
```

Рисунок 5.2 — Таблица тестовых результатов алгоритма Дамерау-Левенштейна.

Результат запусков приведен в таблице ниже:

1. Алгоритм Левенштейна
2. Рекурсивный алгоритм Левенштейна
3. Алгоритм Дамерау-Левенштейна

№ теста	s1(длина s1)	s2(длина s2)	Ожидание	Результат	Оценка (0,1)
0	каждый	охотник	7 7 4	7 7 4	1 1 1
1	желает	знать	5 5 3	5 5 3	1 1 1
2	где	сидит	4 4 4	4 4 4	1 1 1
3	сидит	фазан	5 5 3	5 5 3	1 1 1
4	и	происходит	9 10 9	9 10 9	1 1 1
5	сейчас	сейчас	1 1 1	1 1 1	1 1 1
6	сейчас	сачйес	4 2 4	4 2 4	1 1 1
7	шалаш	шаалш	2 1 2	2 1 2	1 1 1
8	котик	бегемотик	5 5 5	5 5 5	1 1 1
9	кот	слок	3 3 3	3 3 3	1 1 1
10	слон	кот	3 3 3	3 3 3	1 1 1
11	Энштейн	ншЭтейн	2 2 3	2 2 3	1 1 1

Рисунок 5.3 — Проверка правильности работы алгоритма.

5.2 Исследование скорости работы алгоритма

Для исследования скоростных характеристик был использован компьютер на базе процессора Intel Core i7-7700HQ, содержащий 16 гигабайт оперативной памяти. Стоит учесть, что модуль тестирования запускался с жестокго диска под операционной системой Windows. Жесткий диск имел среднюю скорость передачи данных при чтении 105,3 Мбайт/с, а время доступа — 16,3 мс. Столбчатые диаграммы анализа времени работы алгоритмов представлены в разделе 3.2 на Рисунок 3.5 и Рисунок 3.6.

s1	s1	Левенш. (сек)	Рек. Лев.(сек)	Дам-Лев. (сек)	Результат
январь	март	0.88229	1.34251	0.20778	4 4 4
матр	март	0.18123	0.17921	0.18423	2 2 1
октябрь	Октябрь	0.23121	0.32412	0.19821	1 1 1
июнь	ноябрь	0.81232	1.23124	0.76543	5 5 4
ферваль	ьлавреф	0.98768	1.54343	0.87231	6 6 6

Рисунок 5.4 — Таблица тестовых результатов по времени

Ниже представлены матрицы для метода Дамерау-Левенштейна и Левенштейна для наглядного сравнения.

Дамерау-Левенштейн:

	const	л	е	в	е	н	ш	т	е	й	н
const	0	1	2	3	4	5	6	7	8	9	10
э	1	1	2	3	4	5	6	7	8	9	10
н	2	2	1	2	3	4	5	6	7	8	9
ш	3	3	2	2	3	4	4	5	6	7	8
т	4	4	3	3	2	3	4	4	5	6	7
е	5	5	4	4	3	3	4	5	4	5	6
й	6	6	5	5	4	4	3	4	5	4	5
н	7	7	6	6	5	4	4	4	5	5	4

Дамерау - Левенштейн:

	const	т	о	п
const	0	1	2	3
о	1	1	1	2
т	2	1	1	2
п	3	2	2	1

	const	т	о	п
const	0	1	2	3
п	1	1	2	2
о	2	2	1	2
т	3	2	2	2

Рисунок 5.5 — Матрицы расстояний, полученные с помощью алгоритма Дамерау-Левенштейна

Левенштейн:

	const	л	е	в	е	н	ш	т	е	й	н
const	0	1	2	3	4	5	6	7	8	9	10
э	1	1	2	3	4	5	6	7	8	9	10
н	2	2	2	3	4	4	5	6	7	8	9
ш	3	3	3	3	4	5	4	5	6	7	8
т	4	4	4	4	4	5	5	4	5	6	7
е	5	5	4	5	4	5	6	5	4	5	6
й	6	6	5	5	5	5	6	6	5	4	5
н	7	7	6	6	6	5	6	7	6	5	4

Левенштейн:

	const	т	о	п
const	0	1	2	3
о	1	1	1	2
т	2	1	2	2
п	3	2	2	2

	const	т	о	п
const	0	1	2	3
п	1	1	2	2
о	2	2	1	2
т	3	2	2	2

Рисунок 5.6 — Матрицы расстояний, полученные с помощью алгоритма Левенштейна

Заключение

В результате выполнения лабораторной работы были получены следующие основные навыки:

а) изучены теоритические понятия в алгоритмах для нахождения расстояния Левенштейна и Дамерау-Левенштейна

б) рассмотрен один из современных способов оптимизаций и ускорения рекурсивный программ с помощью втроенного счетчика вызовов функций на Python 3

в) проведен аналитический вывод формул для заполнения матриц расстояний

г) проведено сравнение трех реализаций заданного алгоритма, выявлены их слабые места

д) в рамках данной работы было сделано заключение, что рекурсивный алгоритм сильно проигрывает по скорости двум другим реализациям. Скоростные отличия между алгоритмом Дамерау-Левенштейна и Левенштейна в рамках данной работы найдены не были.