Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. БАУМАНА

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

Расстояние Левенштейна. Расстояние Дамерау-Левенштейна

Студент: Зыкин Д.А. ИУ7-53(Б)

Москва 2018

Оглавление

Введение.

Цель работы: изучение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна.

В данной работе требуется изучить и применить метод динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна, а также получить практические навыки реализации алгоритмов.

Данные алгоритмы активно используются при проверке правописания в редакторских программах и в поисковых системах.

В данной работе рассматриваются 3 алгоритма:

1) Алгоритм Левенштейна;
2) Алгоритм Дамерау-Левенштейна;
3) Рекурсивный алгоритм Левенштейн.

1 Аналитическая часть

1.1.1 Расстояние Дамерау-Левенштейна, как и метрика Левенштейна, является мерой "схожести" двух строк. Алгоритм его поиска находит применение в реализации нечеткого поиска, а также в биоинформатике (сравнение ДНК), несмотря на то, что изначально алгоритм разрабатывался для сравнения текстов, набранных человеком (Дамерау показал, что 80% человеческих ошибок при наборе текстов составляют перестановки соседних символов, пропуск символа, добавление нового символа, и ошибка в символе. Поэтому метрика Дамерау-Левенштейна часто используется в редакторских программах для проверки правописания).

Наиболее часто применяемой метрикой является расстояние Левенштейна, или расстояние редактирования, алгоритмы вычисления которого можно найти на каждом шагу.

Исходный вариант этого алгоритма имеет временную сложность **O(mn)** и потребляет **O(mn)** памяти, где **m** и **n** — длины сравниваемых строк.

Расстояние Левенштейна – минимальное количество действий, необходимых для преобразования одного слова в другое. При вычислении расстояния Левенштейна следует выбирать минимальное количество действий необходимые для полного совпадения слов.

При вычислении расстояния Левенштейна используется понятие цена(штраф).

Всего в алгоритме Левенштейна есть 3 операции.

1) замена;
2) вставка;
3) удаления.

1.1.2 Расстояние Дамерау-Левенштейна

Если поиск слова осуществляется в тексте, который набран с клавиатуры, то вместо расстояния Левенштейна используют усовершенствованное расстояние Дамерау – Левенштейна. Исследования Ф. Дамерау показали, что наиболее частая ошибка при наборе слова – перестановка двух соседних букв, транспозиция T (transposition). В случае одной транспозиции расстояние Левенштейна равно 2. При использовании поправки Дамерау транспозиция принимается за единичное расстояние. При использовании расстояния Дамерау – Левенштейна за единичное расстояние принимают следующие действия: I (insert) – добавление символа; D (delete) – удаление символа; R (replace) – замена символа; T (transposition) – перестановка двух соседних символов.

2 Конструкторский раздел

В этом разделе будут рассмотрены алгоритмы с точки зрения формул и блок схем их работы.

2.1 Разработка алгоритмов

Создание и представление работы алгоритмов.

2.1.1 Математическое описание

Пусть имеются две строки 1 и 2 длиной S1, S2.

Тогда расстояние Левенштейна (1, 2) можно подсчитать по рекуррентной формуле:

$$
\mathrm{D(i,j)} = \begin{cases} 0; i = 0, j = 0 \\ i; j = 0, i > 0 \\ j; i = 0, j > 0 \\ D(i-1, j-1);\ S1[i] = S2[j] \\ \min(D(i, j-1) + insertCost, \\ D(i-1, j) + deleteCost, \\ D(i-1, j-1) + replaceCost); j > 0, i > 0, S1[i] \neq S2[j] \end{cases} .
$$

Классификация разрешенных операций и штрафы на выполнение операции:

1) замена символа = 1;
2) вставка символа = 1;
3) удаление символа = 1;
4) совпадение символа = 0.

В алгоритме Дамерау-Левенштейна добавляется еще одна операция - перестановка символа = 1.

В рекуррентную формулу добавляется еще один член:

$$da,b(i,j)\begin{cases}\max(i,j)\, if min(i,j)=0\\ \min(da,b(i-1,j)+1, if\ i,j>1\ and\ ai=bj-1\ and\ ai-1=bj\\ da,b(i,j-1)+1,\\ da,b(i-1,j-1)+1(ai\neq bj)\\ da,b(i-2,j-2)+1)\\ \min(da,b(i-1,j)+1,\\ da,b(i,j-1)+1,\\ da,b(i-1,j-1)+1(ai\neq bj))\end{cases}$$

2.1.2 Блок-схемы алгоритмов

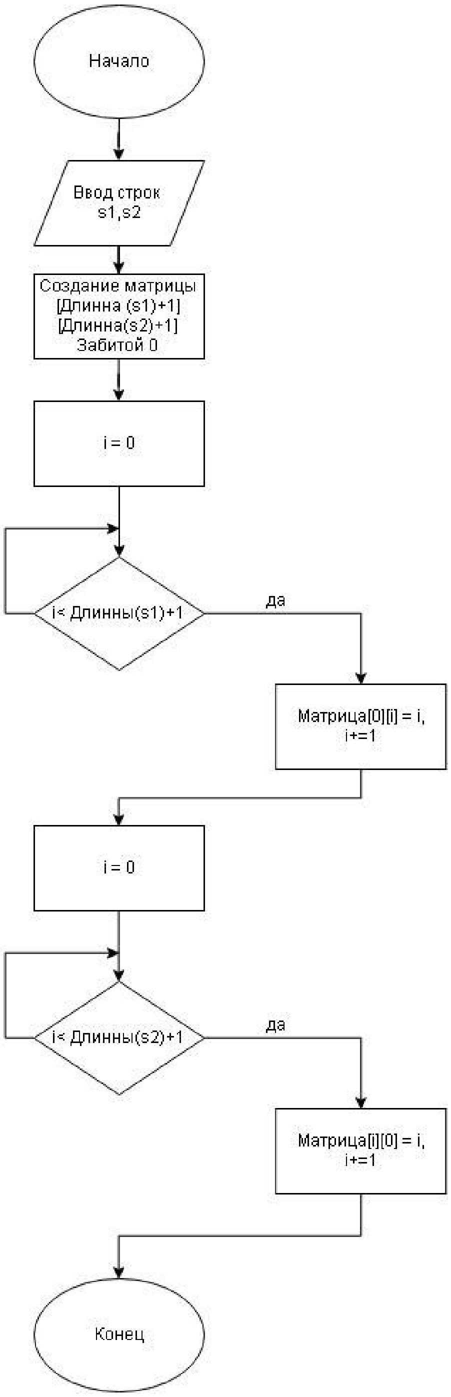

Рисунок 2.1 — процесс инициализации матрицы.

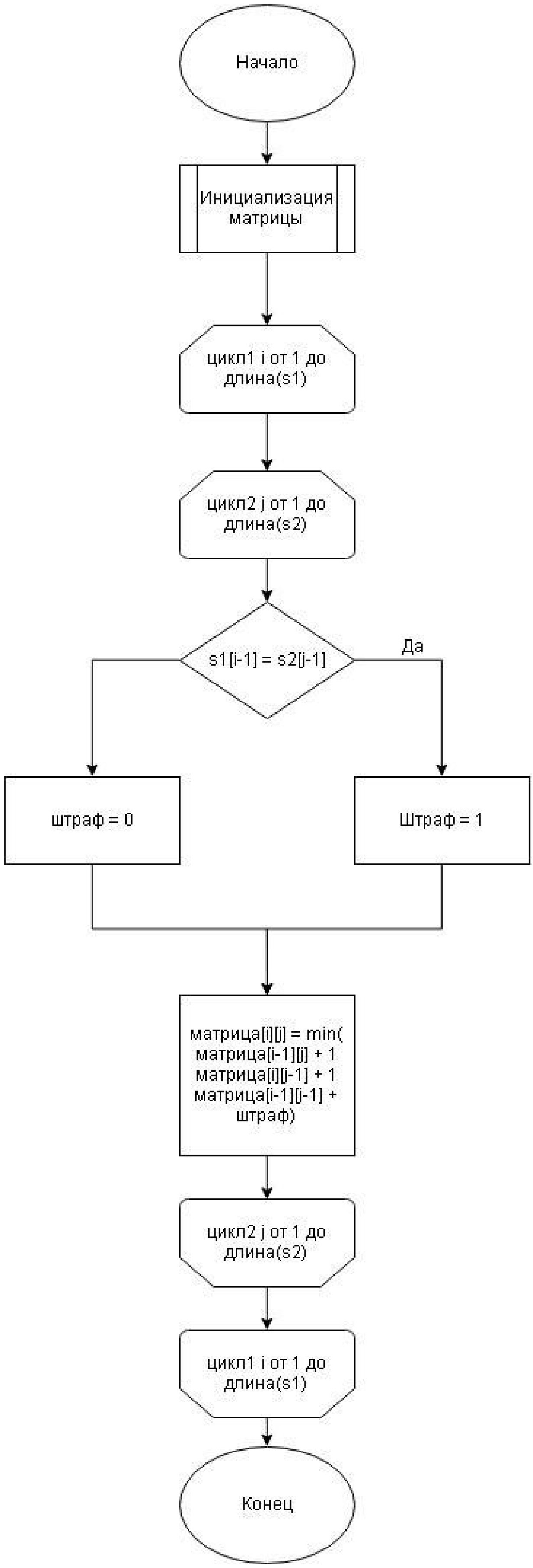

Рисунок 2.2 - алгоритм Левенштейна

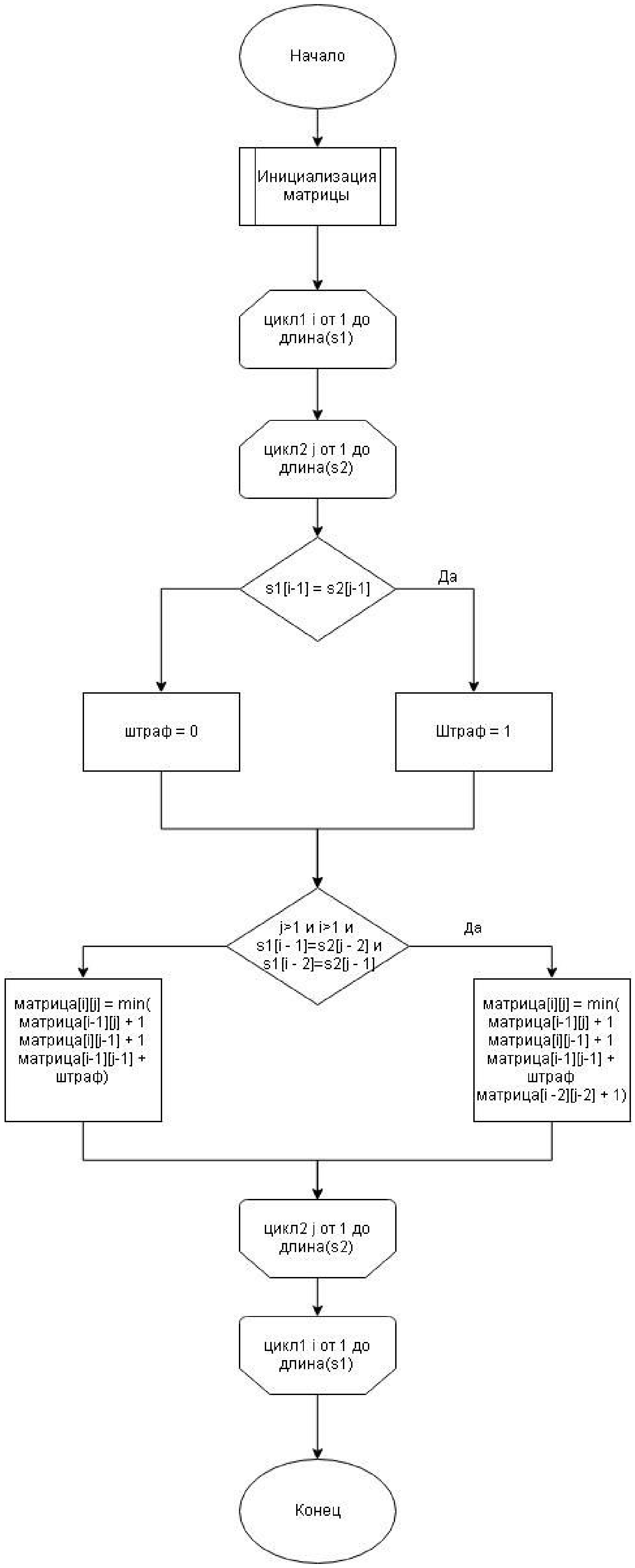

Рисунок 2.3 – алгоритм Дамерау-Левенштейна

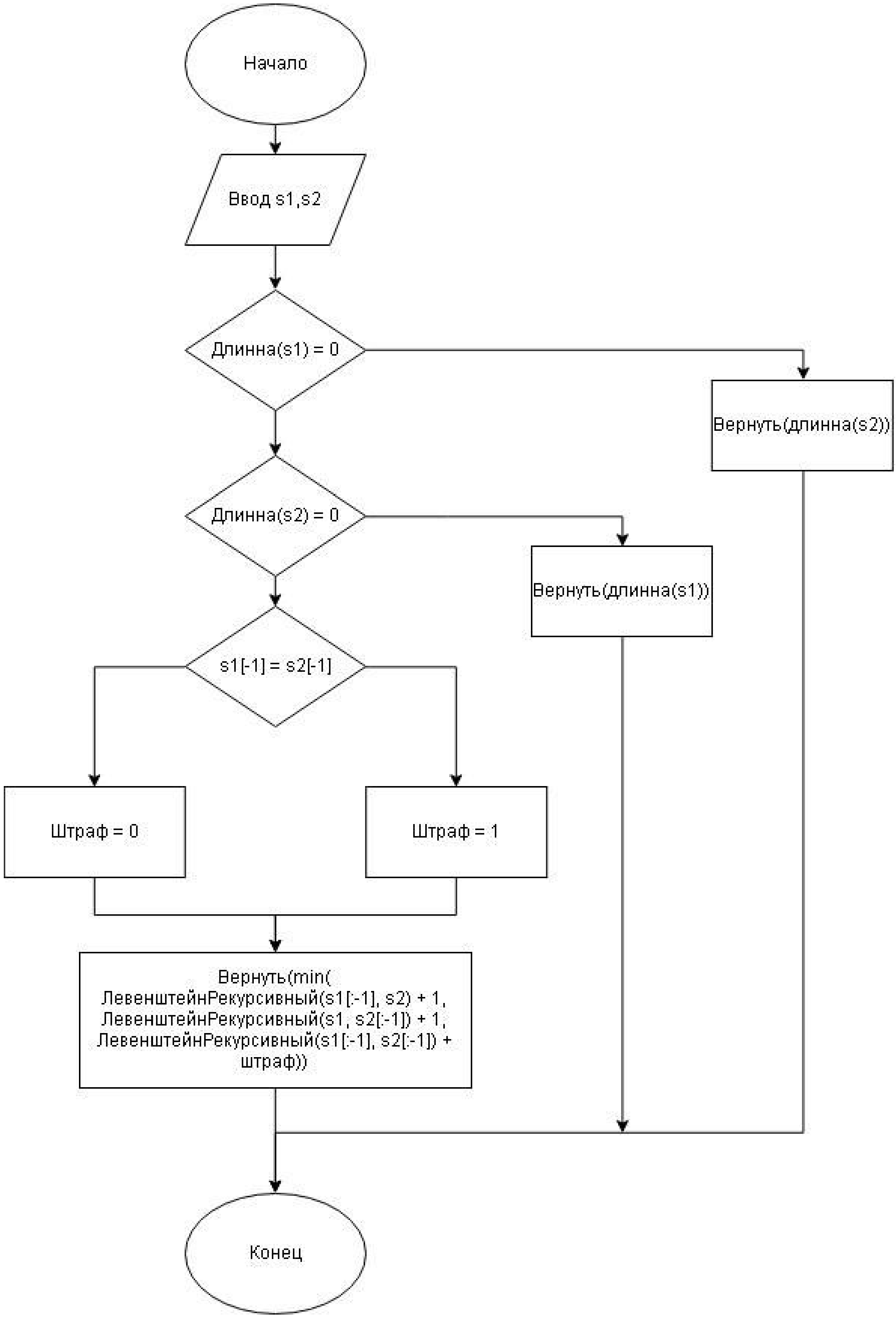

Рисунок 2.4 - алгоритм Левенштейна Рекурсивный

3 Технологический раздел

Создание алгоритмов на языке программирование и создание готового продукта.

3.1 Требования к программному обеспечению

Средние требования:

- Python 3.4.3 для 64-битной системы;
- ОС: OS 64-bit Windows 7;
- Процессор: Intel CPU Core i5-2500K 3.3GHz;
- Оперативная память: 8 GB ОЗУ.

3.2 Средства реализации

Данная программа разрабатывалась на языке Python, т.к. python высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

3.3 Листинг кода

Алгоритм Левенштейна:

```
def Livenshtein(s1, s2):
    matrix = []
    for i in range(len(s2) + 1):
        matrix.append([0] * (len(s1) +1))
    for i in range(1, len(s2)+1 ):
        matrix[i][0] = i
    for i in range(1, len(s1)+1):
        matrix[0][i] = i
    for i in range (1, len(s1)+1):
        for j in range (1, len(s2)+1):
            cost = 0 if s2[j - 1] == s1[i - 1] else 1
            matrix[j][i] = min(matrix[j-1][i]+1,
                               matrix[j][i-1]+1,
                               matrix[j-1][i-1] + cost)
```

Алгоритм Дамерау-Левенштейна:

```
def DamLivenshtein(s1, s2):
    matrix = []
    string = s1


    for i in range(len(s2) + 1):
        matrix.append([0] * (len(s1) +1))
    for i in range(1, len(s2)+1 ):
        matrix[i][0] = i
    for i in range(1, len(s1)+1):
        matrix[0][i] = i
    for i in range (1, len(s1)+1):
```

```
    for j in range (1, len(s2)+1):
        cost = 0 if s2[j - 1] == s1[i - 1] else 1
        if j>1 and i>1 and s1[i - 1] == s2[j - 2] and s1[i - 2] == s2[j - 1]:
            matrix[j][i] = min(matrix[j - 1][i] +1,
                               matrix[j][i - 1] +1,
                               matrix[j - 1][i - 1] + cost,
                               matrix[j - 2][i - 2] +1)
        else:
            matrix[j][i] = min(matrix[j - 1][i] +1,
                               matrix[j][i - 1] +1,
                               matrix[j - 1][i - 1] + cost)
```

Рекурсивный алгоритм Левенштейна:

```
def RecLivenshtein(s1, s2):
    if s1 == "":
        return len(s2)
    if s2 == "":
        return len(s1)
    cost = 0 if s1[-1] == s2[-1] else 1

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

4 Исследовательский раздел

4.1 Пример работы

Левенштейн

```
    п  р  о  л  е  т
  [0, 1, 2, 3, 4, 5, 6]
з [1, 1, 2, 3, 4, 5, 6]
а [2, 2, 2, 3, 4, 5, 6]
л [3, 3, 3, 3, 3, 4, 5]
е [4, 4, 4, 4, 4, 3, 4]
т [5, 5, 5, 5, 5, 4, 3]
```

```
    к  о  р  о  в  а
  [0, 1, 2, 3, 4, 5, 6]
к [1, 0, 1, 2, 3, 4, 5]
о [2, 1, 0, 1, 2, 3, 4]
о [3, 2, 1, 1, 1, 2, 3]
р [4, 3, 2, 1, 2, 2, 3]
в [5, 4, 3, 2, 2, 2, 3]
а [6, 5, 4, 3, 3, 3, 2]
```

```
    л  а  м  а
  [0, 1, 2, 3, 4]
д [1, 1, 2, 3, 4]
а [2, 2, 1, 2, 3]
л [3, 2, 2, 2, 3]
а [4, 3, 2, 3, 2]
й [5, 4, 3, 3, 3]
л [6, 5, 4, 4, 4]
а [7, 6, 5, 5, 4]
м [8, 7, 6, 5, 5]
а [9, 8, 7, 6, 5]
```

```
    1  д  в  а  3  4
  [0, 1, 2, 3, 4, 5, 6]
1 [1, 0, 1, 2, 3, 4, 5]
2 [2, 1, 1, 2, 3, 4, 5]
3 [3, 2, 2, 2, 3, 3, 4]
д [4, 3, 2, 3, 3, 4, 4]
в [5, 4, 3, 2, 3, 4, 5]
а [6, 5, 4, 3, 2, 3, 4]
4 [7, 6, 5, 4, 3, 3, 3]
5 [8, 7, 6, 5, 4, 4, 4]
```

Дамерау-Левенштейн

```
    п  р  о  л  е  т
  [0, 1, 2, 3, 4, 5, 6]
з [1, 1, 2, 3, 4, 5, 6]
а [2, 2, 2, 3, 4, 5, 6]
л [3, 3, 3, 3, 3, 4, 5]
е [4, 4, 4, 4, 4, 3, 4]
т [5, 5, 5, 5, 5, 4, 3]
```

```
    к  о  р  о  в  а
  [0, 1, 2, 3, 4, 5, 6]
к [1, 0, 1, 2, 3, 4, 5]
о [2, 1, 0, 1, 2, 3, 4]
о [3, 2, 1, 1, 1, 2, 3]
р [4, 3, 2, 1, 1, 2, 3]
в [5, 4, 3, 2, 2, 1, 2]
а [6, 5, 4, 3, 3, 2, 1]
```

```
    л  а  м  а
  [0, 1, 2, 3, 4]
д [1, 1, 2, 3, 4]
а [2, 2, 1, 2, 3]
л [3, 2, 2, 2, 3]
а [4, 3, 2, 3, 2]
й [5, 4, 3, 3, 3]
л [6, 5, 4, 4, 4]
а [7, 6, 5, 5, 4]
м [8, 7, 6, 5, 5]
а [9, 8, 7, 6, 5]
```

```
    1  д  в  а  3  4
  [0, 1, 2, 3, 4, 5, 6]
1 [1, 0, 1, 2, 3, 4, 5]
2 [2, 1, 1, 2, 3, 4, 5]
3 [3, 2, 2, 2, 3, 3, 4]
д [4, 3, 2, 3, 3, 4, 4]
в [5, 4, 3, 2, 3, 4, 5]
а [6, 5, 4, 3, 2, 3, 4]
4 [7, 6, 5, 4, 3, 3, 3]
5 [8, 7, 6, 5, 4, 4, 4]
```

```
    1  3  2  4
  [0, 1, 2, 3, 4]
1 [1, 0, 1, 2, 3]
2 [2, 1, 1, 1, 2]
3 [3, 2, 1, 1, 2]
4 [4, 3, 2, 2, 1]
5 [5, 4, 3, 3, 2]
```

4.2 Постановка эксперимента

Исследование скорости работы алгоритма. Для временного сравнительного анализа рекурсивной реализации алгоритма Левенштейна, нерекурсивной реализации алгоритма Левенштейна и алгоритма Дамерау-Левенштейна использовалось несколько запусков. Проводились вычисления по тикам для слов одинаковой длины (5, 10, 12). Можно сделать вывод, что рекурсивную версию алгоритма Левенштейна не стоит применять в крупных базах данных, так как он существенно замедляет работу.

```
Левинштеин:
4.346780077378723e-05
Дамерал-Левинштеин:
5.584404960521250 5e-05
Левинштеин Рекурсивный:
0.0012421527790564893

Левинштеин:
0.00012406434804185105
Дамерал-Левинштеин:
0.00016813586827082918
Левинштеин Рекурсивный:
5.965388461774144

Левинштеин:
0.00016481541126727654
Дамерал-Левинштеин:
0.0002327338499763191
Левинштеин Рекурсивный:
185.55000261108663
```

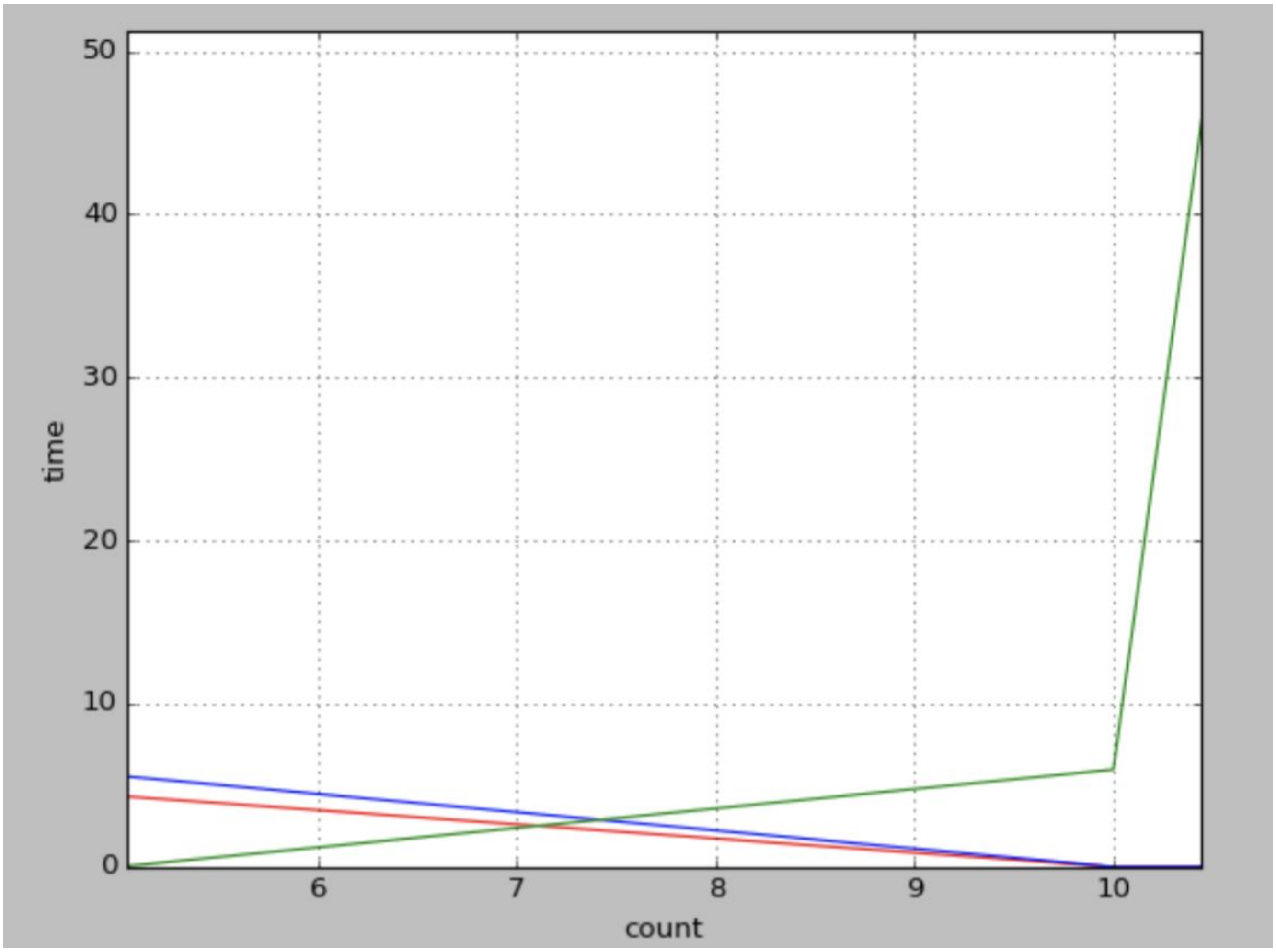

Рисунок 4.2.1 – Пример работы алгоритмов по скорости со словами одной длины

Эксперимент с разными длинами слов, где первое слово (10), второе (9, 8, 7)

```
Левинштеин:
0.0001465059864402284
Дамерал-Левинштеин:
0.00021384733459345676
Левинштеин Рекурсивный:
3.3649366451568463
```

```
Левинштеин:
0.00013416864021368284
Дамерал-Левинштеин:
0.0001927710347897748
Левинштеин Рекурсивный:
1.4498592257389942
```

```
Левинштеин:
0.00012234535007991002
Дамерал-Левинштеин:
0.0002919838606949103
Левинштеин Рекурсивный:
0.5419934992466507
```

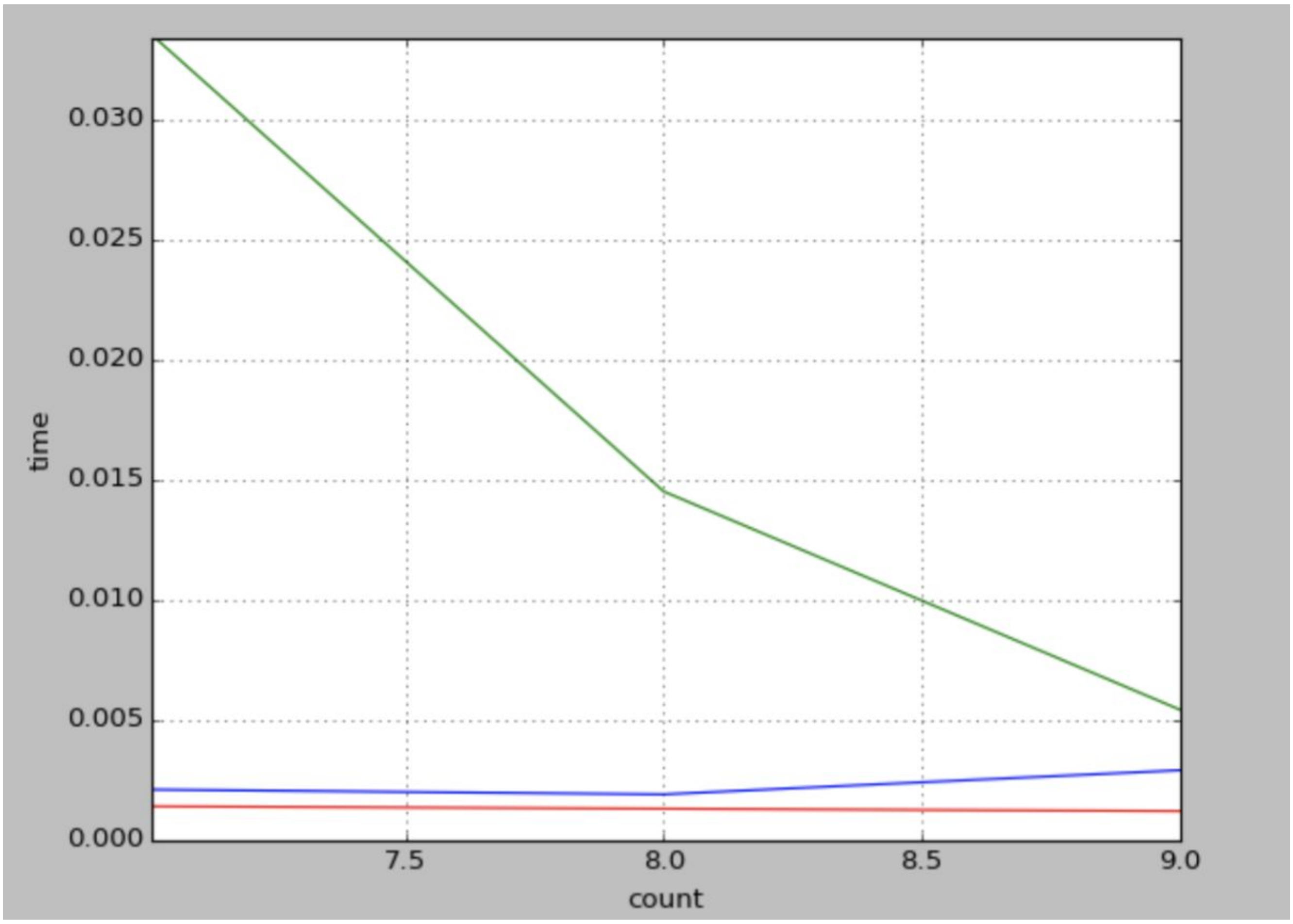

Рисунок 4.2.2 – Пример работы алгоритмов по скорости с словами разной длины

Заключение

1. Изучены теоретические понятия в алгоритмах для нахождения расстояния Левенштейна и Дамерау-Левенштейна;

2. Проведен аналитический вывод формул для заполнения матриц расстояний;

3. Проведено сравнение трех реализаций заданного алгоритма

4. 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;

5. Определение расстояния по Левенштейну имеет недостатки:

   1) при перестановке местами слов или частей слов получаются большие расстояния;

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