Министерство науки и высшего образования 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 дисциплине "Анализ алгоритмов"

Тема Расстояния Левенштейна

Студент Власов Д.В.

Группа ИУ7-51Б

Оценка (баллы) ________________

Преподаватель: Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Расстояние Левенштейна – это минимальное количество редакторских операций, которые необходимы для превращения одной строки в другую. Оно может применяться для решения следующих задач:

- исправления ошибок в слове;
- предложение вариантов поиска в поисковой строке;
- в биоинформатике для сравнения белков.

Целью данной лабораторной работы является реализация и сравнение алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна. При выполнении лабораторной работы поставлены такие задачи:

1. дать математическое описание расстояния Левенштейна;
2. описать алгоритм поиска редакторского расстояния;
3. оценить затраты памяти на выполнение алгоритмов;
4. провести замеры процессорного времени работы на серии экспериментов;
5. провести сравнительный анализ алгоритмов.

1 Аналитический раздел

В данном разделе будет рассмотрено описание алгоритмов поиска расстояния Левенштейна и Дамерау-Левенштейна[1].

Допустимы следующие редакторские операции:

- M - совпадение, штраф – 0;
- I - вставка, штраф – 1;
- R - замена, штраф – 1;
- D - удаление, штраф – 1;

Пусть S_1 и S_2 – строки длиной M и N соответственно над некоторым алфавитом, тогда расстояние Левенштейна можно подсчитать по рекуррентной формуле [2]:

$$D(i,j) = \begin{cases} j, \quad i = 0 \\ i, \quad j = 0, i > 0 \\ min = (\\ D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[j]) \\ ) \end{cases} \tag{1.1}$$

где $m(S_1[i], S_2[j])$ равно нулю, если $S_1[i] = S_2[j]$ и единице в противном случае.

При поиске расстояние Дамерау-Левенштейна добавлена операция транспозиции, штраф которой равен 1, в связи с чем оно может быть вычислено по формуле:

$$D(i,j) = \begin{cases} j, & i = 0 \\ i, & j = 0, i > 0 \\ min \begin{cases} D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[i]), \\ D(i-2, j-2) + m(S_1[i], S_2[i]), \end{cases} & \begin{array}{l} \text{, если } i, j > 1 \\ \text{ и } S_1[i] = S_2[j-1] \\ \text{ и } S_1[i-1] = S_2[j] \end{array} \\ min \begin{cases} D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[i]), \end{cases} & \text{, иначе} \end{cases}$$

Вывод: были рассмотрены алгоритмы поиска расстояний Левенштейна и Дамерау-Левенштейна.

2 Конструкторский раздел

Требования к вводу:

1. на вход подаются 2 строки;

2. прописные и строчные буквы считаются разными символами.

Требования к программе: две пустые строки являются корректным вводом.

2.1 Схемы алгоритмов

На рисунках 2.1-2.4 представлены схемы алгоритмов поиска расстояния Левенштейна и Дамерау-Левенштейна.

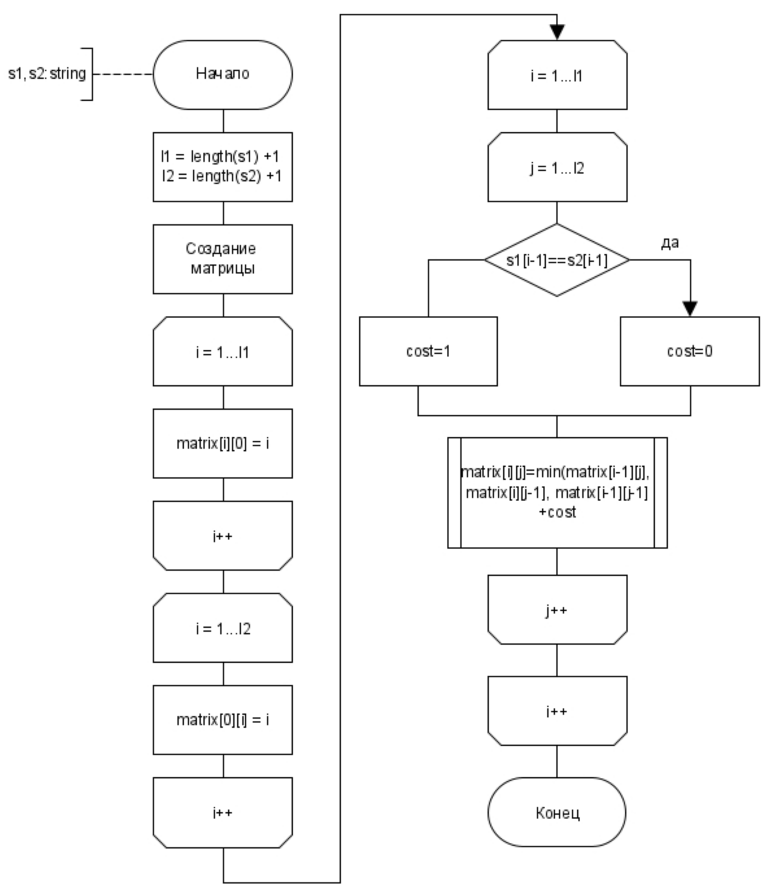

Рис. 2.1: Схема матричного алгоритма поиска расстояния Левенштейна

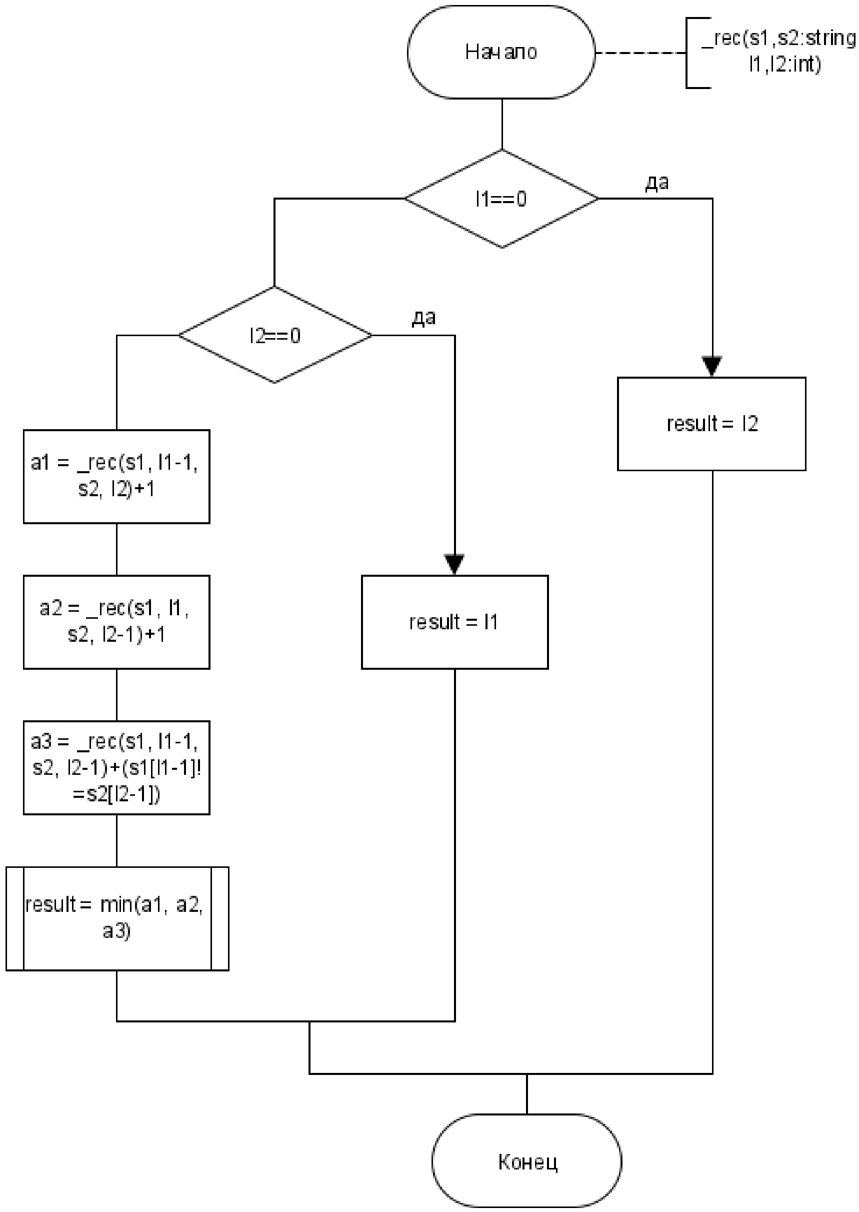

Рис. 2.2: Схема рекурсивного алгоритма поиска расстояния Левенштейна

Отличием матрично-рекурсивного алгоритма поиска расстояния Левенштейна от рекурсивного является сохранение результатов в матрицу, благодаря чему нет необходимости повторно пересчитывать значения функций.

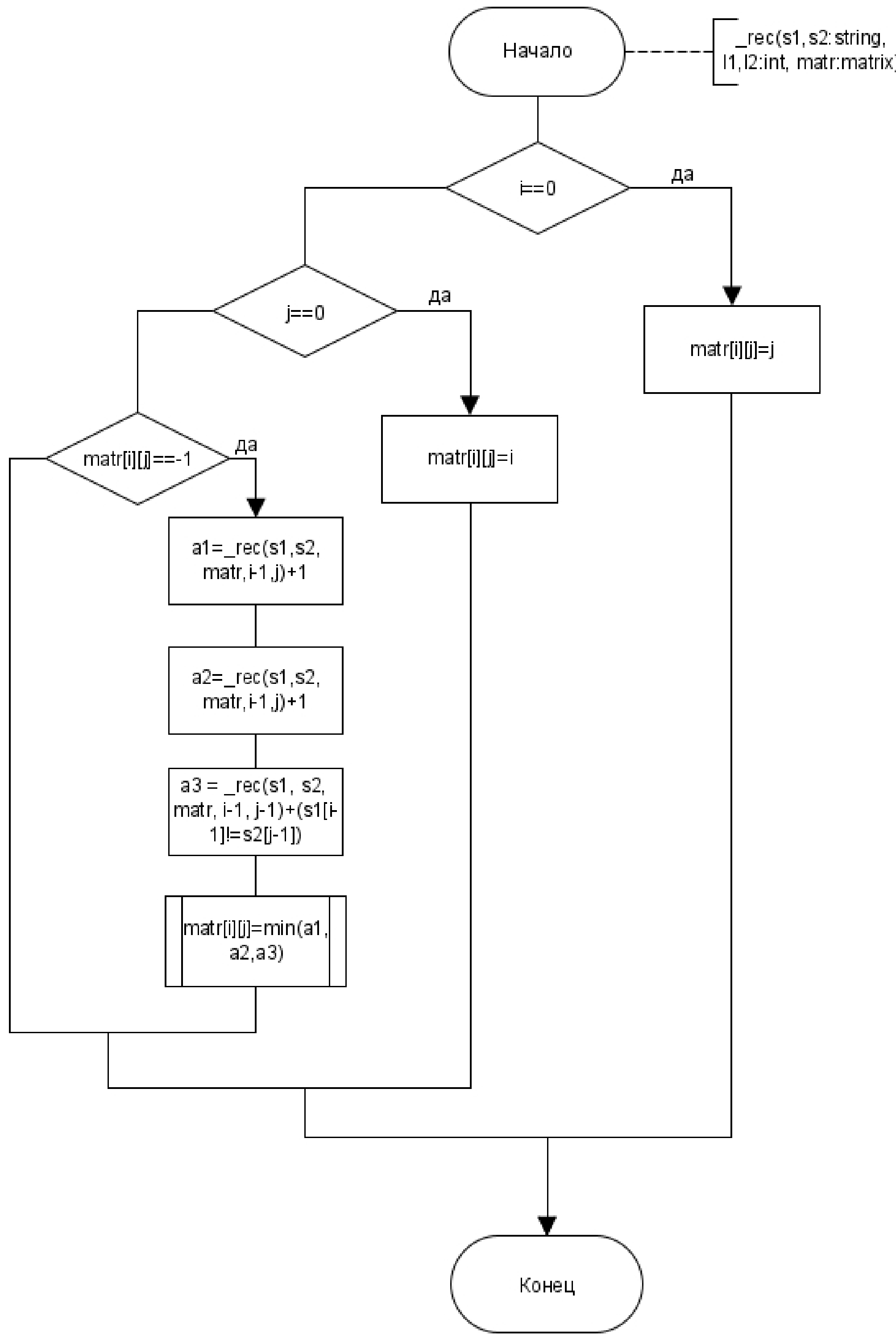

Рис. 2.3: Схема матрично-рекурсивного алгоритма поиска расстояния Левенштейна

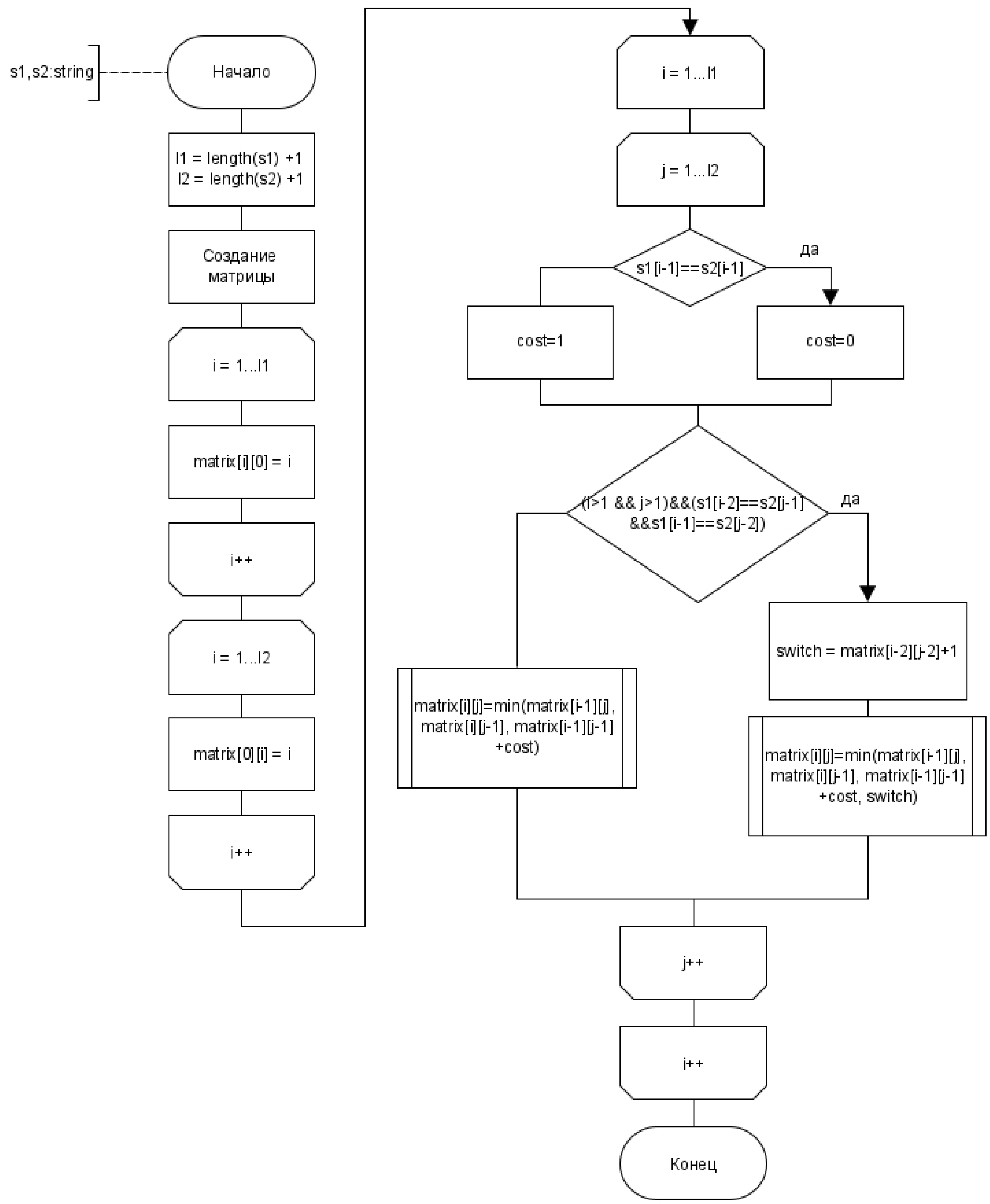

Рис. 2.4: Схема матричного алгоритма поиска расстояния Дамерау-Левенштейна

3 Технологический раздел

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации, представлен листинг кода.

3.1 Средства реализации

В данной работе используется язык программирования Python, в связи с тем, что имею большой опыт работы с ним. Среда разработки Visual Studio Code.
Для замера процессорного времени используется функция procees_time() из библиотеки time.

3.2 Листинг кода

В листингах 3.1-3.4 приведены алгоритмы поиска расстояния Левенштейна и Дамерау-Левенштейна.

Листинг 3.1: Матричный алгоритм поиска расстояния Левенштейна

```
def levenstein_m(origin, target):
  l1 = len(origin) + 1
  l2 = len(target) + 1
  matr = [[0 for i in range(l2)] for i in range(l1)]

  for i in range(1, l1):
    matr[i][0] = i
  for i in range(1, l2):
    matr[0][i] = i

  for i in range(1, l1):
    for j in range(1, l2):
      matr[i][j] = min(matr[i - 1][j] + 1,
              matr[i][j - 1] + 1,
              matr[i - 1][j - 1] + (origin[i - 1] != target[j - 1]))

  return matr[l1 - 1][l2 - 1]
```

Листинг 3.2: Рекурсивный алгоритм поиска расстояния Левенштейна

```
def _rec(origin, l1, target, l2):
  if not l1:
    return l2
  elif not l2:
    return l1

  a1 = _rec(origin, l1-1, target, l2) + 1
  a2 = _rec(origin, l1, target, l2-1) + 1
  a3 = _rec(origin, l1-1, target, l2-1) + \
      (origin[l1 - 1] != target[l2 - 1])

  return min(a1, a2, a3)
```

```

def levenstein_r(origin, target):
  l1 = len(origin)
  l2 = len(target)

  return _rec(origin, l1, target, l2)
```

Листинг 3.3: Матрично-рекурсивный алгоритм поиска расстояния Левенштейна

```
def _rec(s1, s2, matr, i, j):
  if not i:
    matr[i][j] = j
  elif not j:
    matr[i][j] = i

  elif matr[i][j] == -1:
    matr[i][j] = min(_rec(s1, s2, matr, i - 1, j) + 1,
            _rec(s1, s2, matr, i, j - 1) + 1,
            _rec(s1, s2, matr, i - 1, j - 1) + int(s1[i - 1] != s2[j - 1]))

  return matr[i][j]


def levenstein_rm(origin, target):
  l1 = len(origin) + 1
  l2 = len(target) + 1
  matr = [[-1 for i in range(l2)] for i in range(l1)]

  _rec(origin, target, matr, l1 - 1, l2 - 1)
  return matr[l1 - 1][l2 - 1]
```

Листинг 3.4: Матричный алгоритм поиска расстояния Дамерау-Левенштейна

```
def dl_matrix(origin, target):
  l1 = len(origin) + 1
  l2 = len(target) + 1
  matrix = [[0 for i in range(l2)] for i in range(l1)]

  for i in range(1, l1):
    matrix[i][0] = i
  for i in range(1, l2):
    matrix[0][i] = i

  for i in range(1, l1):
    for j in range(1, l2):
      if (i > 1 and j > 1) and (origin[i - 2] == target[j - 1] and origin[i - 1] == target
          [j - 2]):
        switch = matrix[i - 2][j - 2] + 1
        matrix[i][j] = min(matrix[i - 1][j] + 1,
                matrix[i][j - 1] + 1,
                matrix[i - 1][j - 1] + (origin[i - 1] != target[j - 1]),
                switch)
      else:
        matrix[i][j] = min(matrix[i - 1][j] + 1,
                matrix[i][j - 1] + 1,
                matrix[i - 1][j - 1] + (origin[i - 1] != target[j - 1]))

  return matrix[l1 - 1][l2 - 1]
```

3.3 Оценка затрат памяти

В таблице 3.1 приведены объёмы памяти, затрачиваемые различными типами данных в языке Python.

Таблица 3.1: Память, потребляемая разными типами данных в Python

Структура данных	**Занимаемая память(байт)**
Целое число	14
Пустой список	36
Список с 1 элементом	40
Пустая строка	25
Строка длиной 4	29

В таблицах 3.2-3.4 приведены оценки памяти, затрачиваемой на работу алгоритмов поиска расстояния Левенштейна.

Таблица 3.2: Память, потребляемая в матричном алгоритме поиска расстояния Левенштейна

Структура данных	**Занимаемая память(байт)**
Матрица	36+(len(s1)+1)*(36+4*(len(s2)+1))+ (len(s1)+1)*(len(s2)+1)*14
2 вспомогательные переменные(int)	28
2 счётчика(int)	28
передача параметров	2*(25+len(s))

В матричном алгоритме Дамерау-Левенштейна используется аналогичное количество памяти, однако на 1 вспомогательную переменную больше.

Для рекурсивного и матрично-рекурсивного алгоритмов поиска расстояния Левенштейна также будет оцениваться память при максимальной глубине рекурсивного вызова n равной длине большего слова.

Таблица 3.3: Память, потребляемая в рекурсивном алгоритме поиска расстояния Левенштейна

Структура данных	**Занимаемая память(байт)**
3 переменные(int)	52
передача параметров	2*(25+len(s))+2*14
максимальная	n*(52+2*(25+len(s))+2*14)

Таблица 3.4: Память, потребляемая в матрично-рекурсивном алгоритме поиска расстояния Левенштейна

Структура данных	**Занимаемая память(байт)**
матрица	36+(len(s1)+1)*(36+4*(len(s2)+1))+ (len(s1)+1)*(len(s2)+1)*14
передача параметров	2*(25+len(s))+2*14+ 36+(len(s1)+1)*4
максимальная	n*(28+28+36+(len(s1)+1)*(36+4*(len(s2)+1))+ (len(s1)+1)*(len(s2)+1)*14)

Используя таблицы 3.2-3.4 можно оценить память, затрачиваемую на вычисление расстояния между двумя словами, длиной 10 символов.

Таблица 3.5: Память, потребляемая алгоритмами вычисления редакторского расстояния для двух строк длиной 10 символов

Алгоритм	**Затрачиваемая память (байт)**
Матричный Левенштейна	2736
Рекурсивный Левенштейна	1500
Матрично-рекурсивный Левенштейна	26660
Матричный Дамерау-Левенштейна	2750

Из таблицы 3.5 видно, что рекурсивный алгоритм потребляет наименьшее количество памяти для поиска расстояния между словами длиной 10 символов

3.4 Проведение тестирования:

Проведём тестирование программы по методу чёрного ящика. В столбцах "Ожидаемый результат"и "Полученный результат"находится 4 числа, соответствующие матричному, рекурсивному, матрично-рекурсивному алгоритмам поиска расстояния Левенштейна и матричному алгоритму поиска расстояния Дамерау-Левенштейна.

Таблица 3.6: Тестирование программы

Входные данные	**Ожидаемый результат**	**Полученный результат**
,	0 0 0 0	0 0 0 0
, abcd	4 4 4 4	4 4 4 4
abcd,	4 4 4 4	4 4 4 4
telo, stolb	3 3 3 3	3 3 3 3
abcd, cbef	3 3 3 3	3 3 3 3
abcd, bacd	2 2 2 1	2 2 2 1
1234, 5678	4 4 4 4	4 4 4 4

Все тесты пройдены успешно.

4 Исследовательский раздел

Постановка эксперимента В рамках проекта были проведены эксперименты, описанные ниже.

1. Сравнение матричного, матрично-рекурсивного алгоритмов Левенштейна и матричного алгоритма Дамерау-Левенштейна проводилось на словах длиной от 1 до 500, с шагом 50, для каждой длины было представлено 20 слов.

2. Для сравнения матричного, рекурсивного, матрично-рекурсивного алгоритмов Левенштейна и матричного алгоритма Дамерау-Левенштейна использовались слова длиной от 1 до 8 с шагом 1, аналогично по 20 слов каждой длины.

Слова случайно генерировались и состояли их цифр от 1 до 9.

Сравнительный анализ на материале экспериментальных данных На рисунках представлены 4.1 и 4.2 представлены графики зависимости времени работы алгоритмов поиска редакционного расстояния от длины слов.

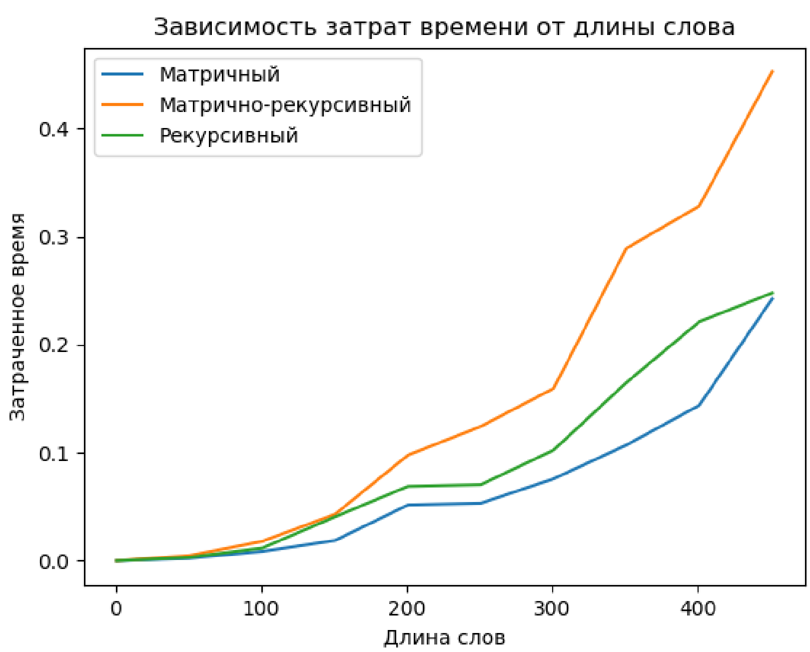

Рис. 4.1: График зависимости времени работы матричного, матрично-рекурсивного алгоритмов Левенштейна и матричного алгоритма Дамерау-Левенштейна от длины слов (ось абсцисс-время работы в секундах, ось ординат-длина слов)

По графику видно, что наиболее быстрым является матричный алгоритм поиска расстояния Левенштейна, несколько медленнее - матричный алгоритм поиска расстояния Дамерау-Левенштейна, что связано с более сложной логикой второго, и наиболее медленный - матрично-рекурсивный алгоритм, в связи с количеством дополнительных вызовов функции.

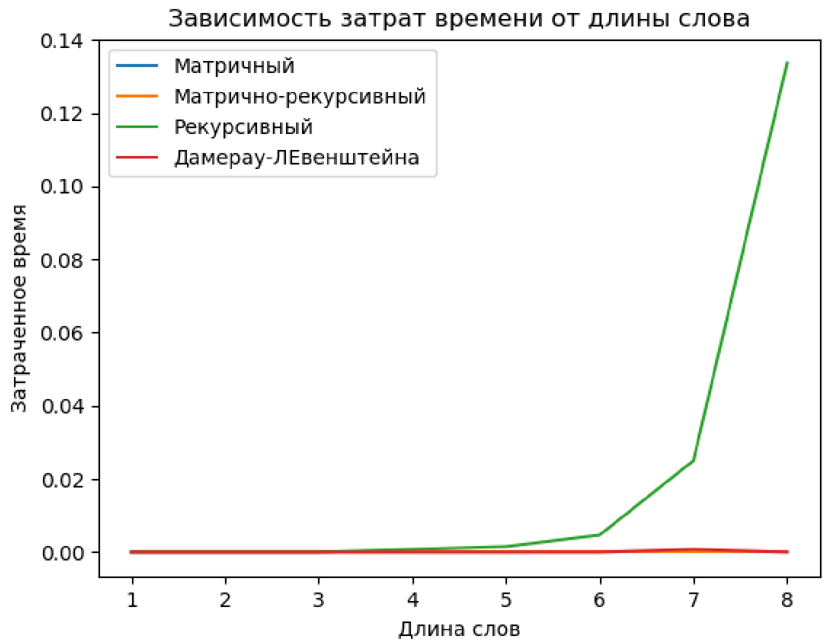

Рис. 4.2: График зависимости времени работы матричного, рекурсивного, матрично-рекурсивного алгоритмов Левенштейна и матричного алгоритма Дамерау-Левенштейна от длины слов (ось абсцисс-время работы в секундах, ось ординат-длина слов)

Время выполнения рекурсивного алгоритма увеличивается экспоненциально, в связи с чем итеративные или матрично-рекурсивные алгоритмы выполняются значительно быстрее.

Таким образом, самым быстрым алгоритмом оказался матричный алгоритм поиска расстояния Левенштейна, а самым медленным- рекурсивный. Можно сделать вывод, что матричный алгоритм эффективен для поиска редакционного расстояния для всех слов, но в случаях, когда ограничены ресурсы памяти и длина слов мала, лучше будет использовать рекурсивный алгоритм.

Заключение

В ходе лабораторной работы цель достигнута реализованы алгоритмы поиска расстояний Левенштейна и Дамерау-Левенштейна, проведён их сравнительный анализ. Все задачи были выполнены: дано математическое описание расстояния Левенштейна, описаны алгоритмы поиска редакторского расстояния, оценены затраты памяти на выполнение алгоритмов, проведены замеры процессорного времени.

Литература

[1] В.И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. —1965.

[2] sic. Вычисление редакционного расстояния[Электронный ресурс]. — 2011. —Режим доступа: https://habr.com/ru/post/117063/ (дата обращения: 20.09.2021).