«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №1

«Анализ алгоритмов»

Расстояние Левенштейна и Дамерау-Левенштейна

Студент: Нгуен Фыок Санг , гр. ИУ7-56Б

Москва, 2019 г.

Оглавление

Введение **2**

1 Аналитическая часть **3**
1.1 Задачи . . . 3
1.2 Описание алгоритмов . . . 3
1.2.1 Расстояние Левенштейна . . . 3
1.2.2 Расстояние Дамерау-Левенштейна . . . 4

2 Конструкторская часть **6**
2.1 Схемы алгоритмов . . . 6

3 Технологическая часть **9**
3.1 Требования к программному обеспечению . . . 9
3.2 Средства реализации . . . 9
3.3 Реализации алгоритмов . . . 9
3.4 Тесты . . . 12

4 Экспериментальная часть **13**
4.1 Примеры работы . . . 13
4.2 Сравнение работы алгоритмов Левенштейна и Дамерау-Левенштейна . . . 13
4.3 Сравнение работы реализаций алгоритма Левенштейна . . . 14

Заключение **16**

Литература **17**

Введение

В современном мире почти каждый человек пользуются компьютером и Интернетом в частности. Люди пишут текст в документах, выполняют поиск в поисковых системах, ищут переводы слов и текстов в онлайн-словарях. В таких ситуациях человек часто делает орфографические ошибки или опечатки, и на их исправление он тратит своё время. Чтобы этого избежать, в подобных системах есть опции поиска ошибок и автоисправления. Для такой опции необходим поиск расстояния между строками по алгоритмам Левенштейна и Дамерау-Левенштейна. Также эта задача необходима и в программировании (например, для сравнения текстовых файлов или файлов кода в системах контроля версий) и в биоинформатике (например, для сравнения белков, генов и хромосом).

1. Аналитическая часть

1.1 Задачи

Цель лабораторной работы: исследовать расстояния Левенштейна и Дамерау-Левенштейна. Для достижения этой цели были поставлены следующие задачи:

- изучить алгоритмы вычисления расстояний между строками;
- применить метод динамического программирования для матричных реализаций алгоритмов;
- сравнить матричную и рекурсивную реализации алгоритмов;
- оценить эффективность каждой из реализаций по времени и памяти.

1.2 Описание алгоритмов

1.2.1 Расстояние Левенштейна

Расстояние Левенштейна определяет минимальное количество операций, необходимых для превращения одной строки в другую, среди которых:

- вставка (I - insert);
- удаление (D - delete);
- замена (R - replace).

У каждой операции есть так называемая "цена или "штраф"за её выполнение. Цена каждой операции равна 1, кроме случая совпадения символов (M - match); цена в этом случае равна 0, т. к. при равенстве символов не требуется никаких действий. Соответственно, задача нахождения расстояния Левенштейна заключается в нахождении такой последовательности операции, приводящик одну строку к другой, суммарная цена которых минимальна.

Таким образом, если заданы две строки S_1 и S_2 с длинами m и n соответственно над некоторым алфавитом, то расстояние Левенштейна $D(S_1, S_2)$ между данными строками можно вычислить по следующей рекуррентной формуле [3]:

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

Соотношения в рекурретной формуле отвечают за соотвествующие разрешённые операции:

1. Вставка.
2. Удаление.
3. Замена или совпадение в зависимости от результата ($S_1[i] \neq S_2[j]$).

1.2.2 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является модификацией расстояние Левенштейна. К исходному набору возможных операций добавляется операция транспозиции (T - transpose), или перестановка двух соседних символов. В своих исследованиях Ф. Дамерау показал, что наиболее частой ошибкой при вводе текста является перестановка двух соседних букв слов [2]. "Цена"данной операции также равняется 1. При вычислении расстояния Левенштейна в такой ситуации потребовалось бы дважды заменить символ. Суммарная цена этих двух операций равнялась бы 2, а транспозиция добавляет в суммарную цену лишь 1. Исходя из этого, можно утверждать, что расстояние Дамерау-Левенштейна даёт лучший результат в сравнении с расстоянием Левенштейна.

При вычислении расстояния Дамерау-Левенштейна в рекурретную формулу вносится дополнительное соотношение в минимум:

$$D(S_1[i-2], S_2[j-2]) + 1 \tag{1.2}$$

Соотношение (1.2) вносится в выражение только при выполнении следующих условий:

$$\begin{cases} i > 1,\ j > 1 \\ S_1[i] = S_2[j-1] \\ S_1[i-1] = S_2[j] \end{cases} \tag{1.3}$$

Таким образом получаем следующую рекурретную формулу:

$$D(S_1[i], S_2[j]) = \begin{cases} i \ if \ j = 0 \\ j \ if \ i = 0 \\ min \begin{cases} D(S_1[i-1], S_2[j] + 1) \\ D(S_1[i], S_2[j-1] + 1) \\ D(S_1[i-1], S_2[j-1]) + \begin{cases} 1, if \ S_1[i] \neq S_2[j] \\ 0, else \end{cases} \\ D(S_1[i-2], S_2[j-2]) + 1 \end{cases} & \text{if (1.3)} \\ min \begin{cases} D(S_1[i-1], S_2[j] + 1) \\ D(S_1[i], S_2[j-1] + 1) \\ D(S_1[i-1], S_2[j-1] + \begin{cases} 1, if \ S_1[i] \neq S_2[j] \\ 0, else \end{cases} \end{cases} & \text{else} \end{cases} \quad (1.4)$$

2. Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы алгоритмов трёх реализаций алгоритмов поиск расстояния между строками.

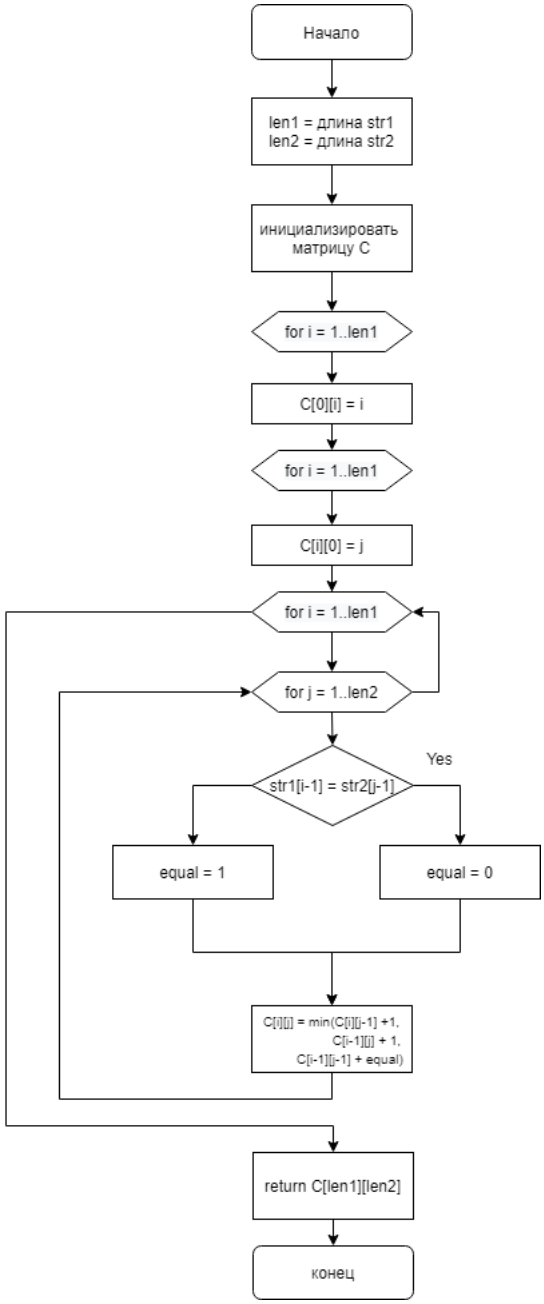

Рис. 2.1: Матричная реализация алгоритма Левенштейна

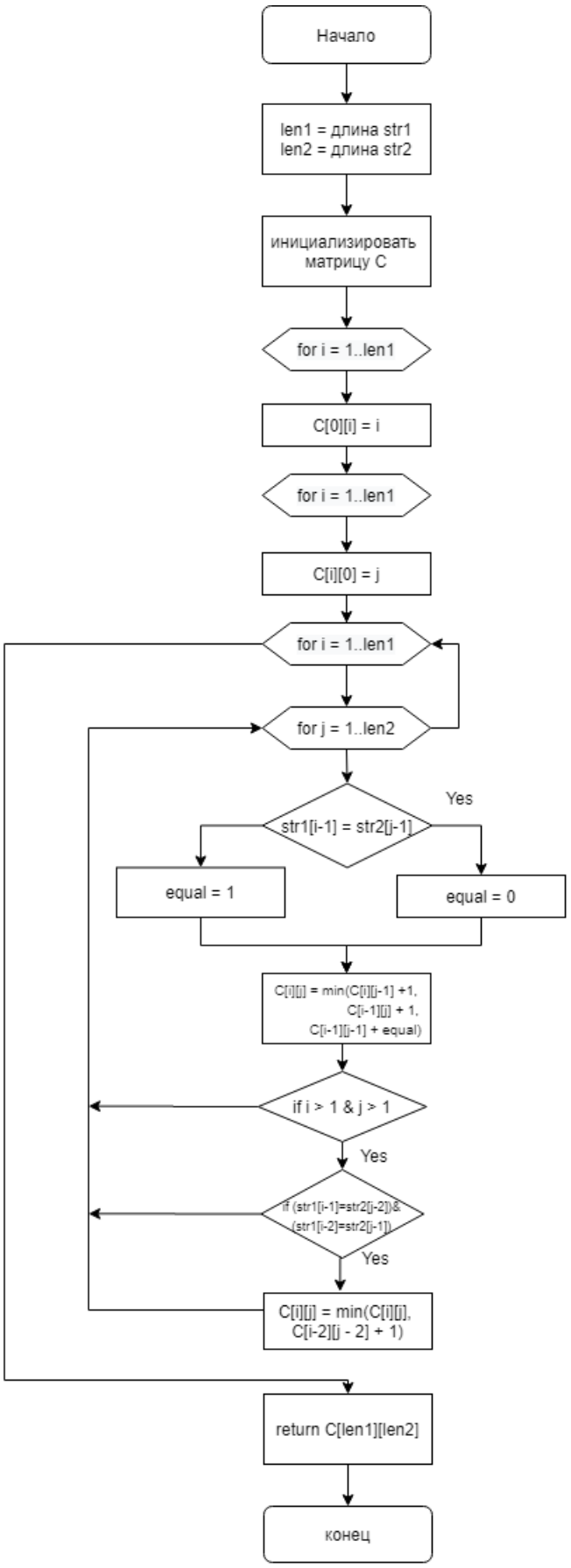

Рис. 2.2: Матричная реализация алгоритма Дамерау-Левенштейна

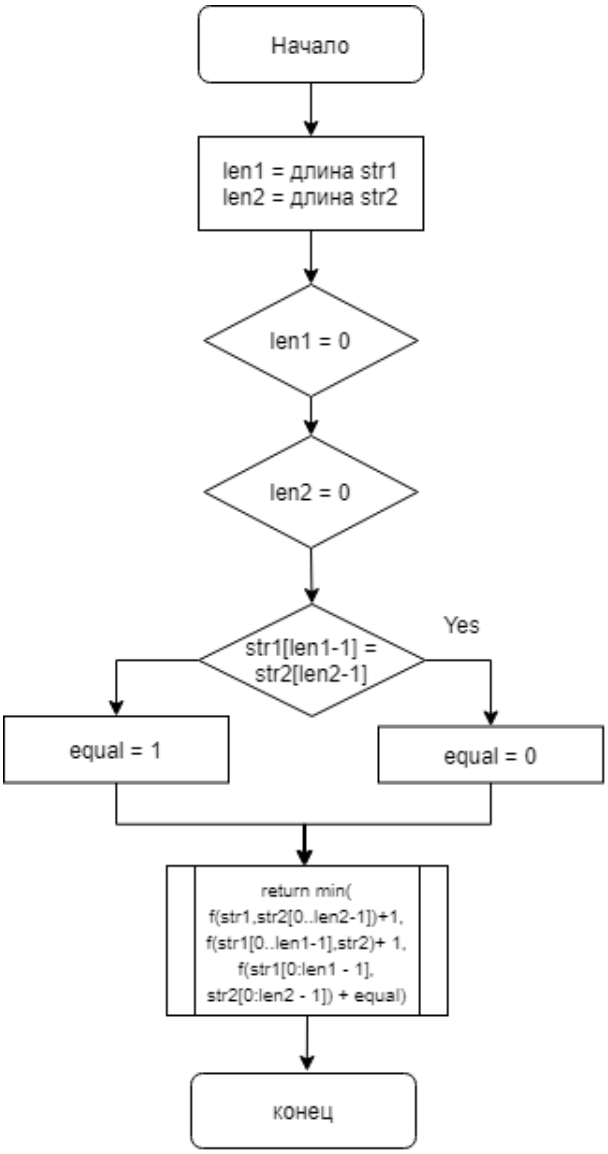

Рис. 2.3: Рекурсивная реализация алгоритма Дамерау-Левенштейна

3. Технологическая часть

3.1 Требования к программному обеспечению

На вход подаются две строки максимальной длины в 50 символов, которые входят в таблицу Юникода (UTF-8). На выход программа выдаёт три числовых значения, которые являются результатами вычисления расстояний тремя методам: матричными реализациями алгоритмов Левенштейна и Дамерау-Левенштейна и рекурсивной реализацией алгоритма Левенштейна. В качестве результата для матричных реализаций также выводится матрица расстояний.

3.2 Средства реализации

Для реализации программы был использован язык Python [4]. Для замера процессорного времени была использована функция time() из библиотеки time.

3.3 Реализации алгоритмов

На листингах 3.1 - 3.3 представлены коды реализации алгоритмов поиска расстояния.

Листинг 3.1: Матричная реализация алгоритма Левенштейна

```
1
2 def levenshtein(str1, str2):
3
4   len1 = len(str1)
5   len2 = len(str2)
6
7   C = [[0 for i in range(len2 + 1)] for j in range(len1 + 1)]
8
9   for i in range(0, len2 + 1):
10    C[0][i] = i
11
12  for i in range(0, len1 + 1):
13    C[i][0] = i
14
15  for i in range(1, len1 + 1):
16    for j in range(1, len2 + 1):
17      if (str1[i - 1] == str2[j - 1]):
18        equal = 0
19      else:
20        equal = 1
21
22      C[i][j] = min(C[i][j-1] + 1,
```

```
23                 C[i-1][j] + 1,
24                 C[i-1][j-1] + equal)
25     #matrix_print(str1, str2, C)
26     return C[len1][len2]
```

Листинг 3.2: Матричная реализация алгоритма Дамерау-Левенштейна

```
1
2  def damerau_levenshtein(str1, str2):
3      len1 = len(str1)
4      len2 = len(str2)
5
6      C = [[0 for i in range(len2 + 1)] for j in range(len1 + 1)]
7
8      for i in range(0, len2 + 1):
9          C[0][i] = i
10
11     for i in range(0, len1 + 1):
12         C[i][0] = i
13
14     for i in range(1, len1 + 1):
15         for j in range(1, len2 + 1):
16             if (str1[i - 1] == str2[j - 1]):
17                 equal = 0
18             else:
19                 equal = 1
20
21             C[i][j] = min(C[i][j-1] + 1,
22                     C[i-1][j] + 1,
23                     C[i-1][j-1] + equal)
24
25             if (i > 1 and j > 1 and str1[i - 1] == str2[j - 2] and str1[i - 2] == str2[j - 1]):
26                 C[i][j] = min(C[i][j], C[i-2][j - 2] + 1)
27
28     #matrix_print(str1, str2, C)
29     return C[len1][len2]
```

Листинг 3.3: Рекурсивная реализация алгоритма Левенштейна

```
1
2  def levenshtein_recursive(str1, str2):
3      len1 = len(str1)
4      len2 = len(str2)
5
6      if (len1 == 0):
7          return len2
8      if (len2 == 0):
9          return len1
10
11     is_equal = 1
12     if (str1[len1 - 1] == str2[len2 - 1]):
13         is_equal = 0
14
15     return min(levenshtein_recursive(str1, str2[0 : len2 - 1]) + 1,
```

```
16              levenshtein_recursive(str1[0 : len1 − 1], str2) + 1,
17              levenshtein_recursive(str1[0:len1 − 1], str2[0:len2 − 1]) + is_equal)
```

3.4 Тесты

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице 3.1. В данной таблице λ означает пустую строку, а числа в столбцах "Ожидание"и "Результат"соответствуют результатам работы алгоритмов в следующем порядке:

1. Матричная реализация алгоритма Левенштейна.
2. Матричная реализация алгоритма Дамерау-Левенштейна.
3. Рекурсивная реализация алгоритма Левенштейна.

Таблица 3.1: Функциональные тесты

Строка 1	**Строка 2**	**Ожидание**	**Результат**
λ	λ	0 0 0	0 0 0
λ	а	1 1 1	1 1 1
а	λ	1 1 1	1 1 1
а	а	0 0 0	0 0 0
а	б	1 1 1	1 1 1
азы	базы	1 1 1	1 1 1
компютер	компьютер	1 1 1	1 1 1
данны	данные	1 1 1	1 1 1
email.ru	mail.ru	1 1 1	1 1 1
programmmer	programmer	1 1 1	1 1 1
mail.rus	mail.ru	1 1 1	1 1 1
ашибка	ошибка	1 1 1	1 1 1
алгоритм	алгорифм	1 1 1	1 1 1
копия	копии	1 1 1	1 1 1
укрсовой	курсовой	2 1 1	2 1 1
аглоритм	алгоритм	2 1 1	2 1 1
универе	универ	2 1 1	2 1 1
курс	курсовой	4 4 4	4 4 4
курсовой	курс	4 4 4	4 4 4
курсовой	курсовик	2 2 2	2 2 2
код	закодировать	9 9 9	9 9 9
закодировать	код	9 9 9	9 9 9
ccoders	recoding	5 5 5	5 5 5
header	subheader	3 3 3	3 3 3
subheader	header	3 3 3	3 3 3
subheader	overheader	4 4 4	4 4 4

В результате проверки все реализации алгоритмов прошли все поставленные функциональные тесты.

4. Экспериментальная часть

4.1 Примеры работы

На рисунке 4.1 представлен пример работы программы, демонстрирующий различие в работе алгоритмов наглядно: различаются матрицы расстояний и результаты.

```
Input First String: people
Input Second String: poepel
-- --       p    o    e    p    e    l
-------------------------------------
      0    1    2    3    4    5    6
-------------------------------------
  p   1    0    1    2    3    4    5
-------------------------------------
  e   2    1    1    1    2    3    4
-------------------------------------
  o   3    2    1    1    2    3    4
-------------------------------------
  p   4    3    2    2    1    2    3
-------------------------------------
  l   5    4    3    3    2    2    2
-------------------------------------
  e   6    5    4    3    3    2    2
Result (damerau_levenshtein) is:          2
-- --       p    o    e    p    e    l
-------------------------------------
      0    1    2    3    4    5    6
-------------------------------------
  p   1    0    1    2    3    4    5
-------------------------------------
  e   2    1    1    1    2    3    4
-------------------------------------
  o   3    2    1    2    2    3    4
-------------------------------------
  p   4    3    2    2    2    3    4
-------------------------------------
  l   5    4    3    3    3    3    3
-------------------------------------
  e   6    5    4    3    4    3    4
Result(levenshtein) is:           4
Result(levenshtein_recursive) is:          4
```

Рис. 4.1: Пример работы программы

4.2 Сравнение работы алгоритмов Левенштейна и Дамерау-Левенштейна

Для сравнения времени работы алгоритмов Левенштейна и Дамерау-Левенштейна были использованы строки длиной от 10 до 70 с шагом 10. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.1 и на рисунке 4.2.

Таблица 4.1: Время работы матричных реализаций алгоритмов (ms) процессора

Длина слова	Алгоритм Левенштейна	Алгоритм Дамерау-Левенштейна
10	0.19946	0.19927
20	0.59840	0.79860
30	1.39639	1.69157
40	2.29361	3.68990
50	2.49347	7.08122
60	4.38809	7.87875
70	6.98132	10.77120
80	5.88438	14.36564
90	10.37225	16.85559
100	12.16747	26.12550

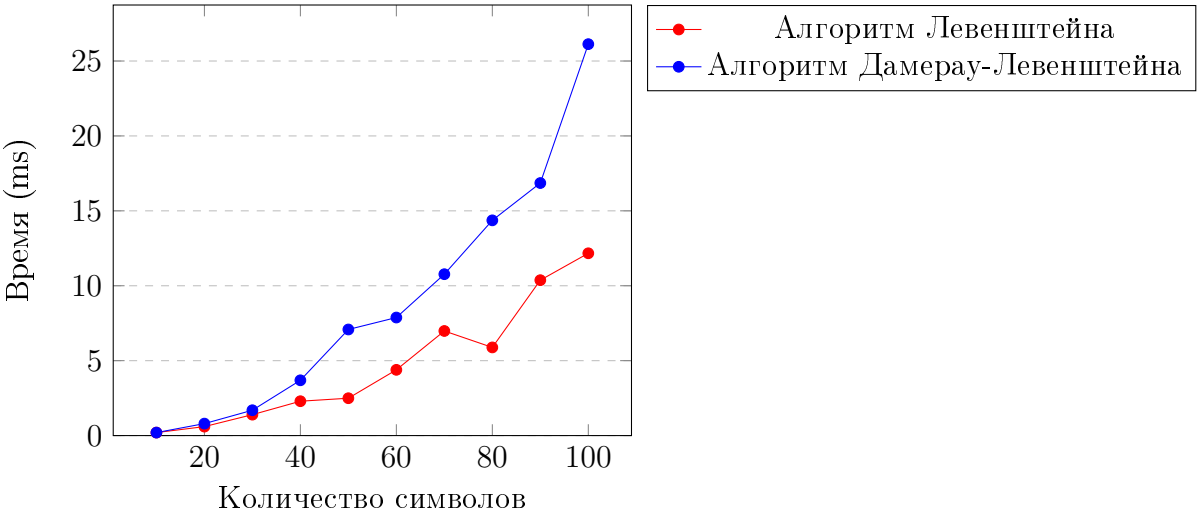

Рис. 4.2: График времени работы матричных реализаций алгоритмов Левенштейна и Дамерау-Левенштейна

Алгоритм Левенштейна выигрывает по времени. Алгоритм Дамерау-Левенштейна выполняется дольше за счёт добавления небольшого количества операций.

4.3 Сравнение работы реализаций алгоритма Левенштейна

Для сравнения времени работы матричной и рекурсивной реализаций алгоритма Левенштейна были использованы строки длиной от 1 до 10 с шагом 1. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.2 и на рисунках 4.3 и 4.4.

Время выполнения рекурсивной реализации алгоритма резко возрастает с увеличением длины слов. Можно сделать вывод о том, что матричная реализация алгоритма значительно эффективнее рекурсивной при любой длине слова.

Таблица 4.2: Время (ms) работы реализаций алгоритма Левенштейна процессора

Длина слова	Матричная реализация	Рекурсивная реализация
1	0.00947	0.00219
2	0.01944	0.01216
3	0.02293	0.06726
4	0.03590	0.34368
5	0.04488	2.12764
6	0.06332	9.37488
7	0.09275	52.40877
8	0.10023	295.03857

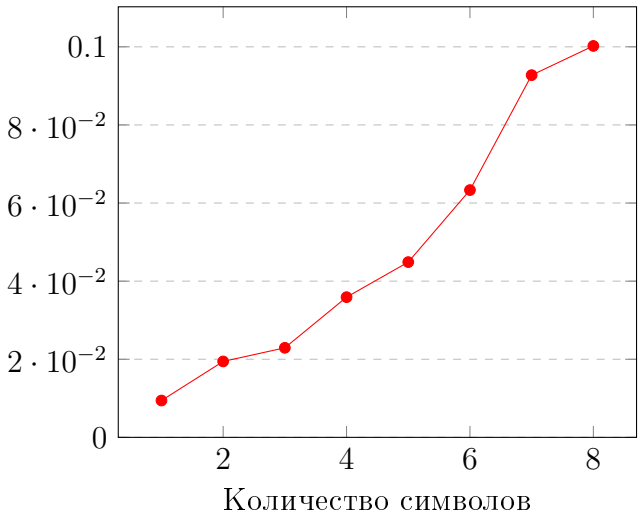

Рис. 4.3: График времени работы матричной реализации алгоритма Левенштейна

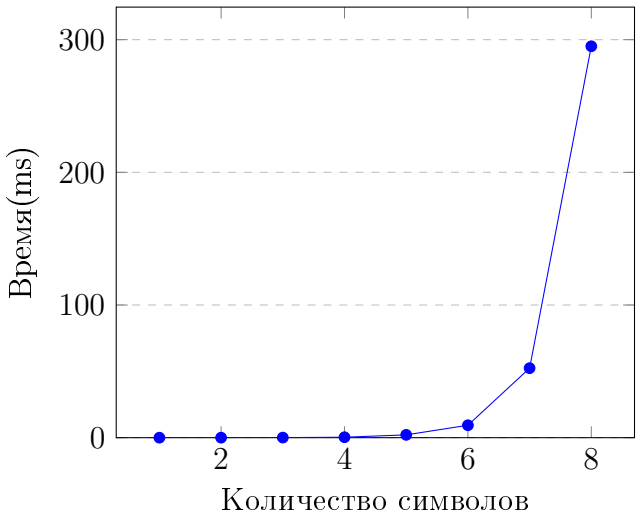

Рис. 4.4: График времени работы рекурсивной реализации алгоритма Левенштейна

Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна. Для этого были реализованы три различные реализации алгоритмов с применением навыка динамического программирования для матричных.
Экспериментально потверждена эффективность матричных реализаций над рекурсивной: при длине слов выше 10 символов применение рекурсивной реализации алгоритма Дамерау-Левенштейна является нецелесообразной, т. к. проигрывает по памяти и по времени матричных в несколько порядков. Также было экспериментально установлено, что применение матричной реализации Дамерау-Левенштейна допустимо, так как данный алгоритм уступает по времени алгоритму Левенштейна лишь на 10%, но при это при решении определённых задач может давать меньший результат.

Литература

[1] Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. В. И. Левенштейн.

[2] A technique for computer detection and correction of spelling errors. Damerau Fred J.

[3] Indexing methods for approximate dictionary searching. Journal of Experimental Algorithmics, 2011. L. M. Boytsov

[4] https://cppreference.com/ [Электронный ресурс]