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

Дисциплина: анализ алгоритмов

Лабораторная работа №1
Расстояние Левенштейна

Студент: Власова Е. В., ИУ7-54

2019 г.

Содержание

Введение

Цель работы: изучение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна. В данной лабораторной работе изучаются алгоритмы нечеткого поиска.

Расстояние Левенштейна, как описано в [1], — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Расстояние Левенштейна и его модификации используются:

1) для выявления ошибок в словах (в поисковиках, различных BD (базах для хранения данных), в программах, которые разработаны для того, чтобы распознать текст;
2) для простого сравнения текстовых файлов;
3) в биохимической информатике данный метод широко используют для сравнения генов, белков, а также хромосом.

1 Аналитический раздел

В данном разделе представлены описания используемых алгоритмов.

1.1 Описание алгоритмов

1.1.1 Расстояние Левенштейна

В работе рассматриваются алгоритм Левенштейна, рекурсивный алгоритм Левенштейна и матричный алгоритм Дамерау-Левенштейна.

Ссылаясь на [1], расстояние Левенштейна использует понятие цена операции. Операций всего три: замена, вставка, удаления. Задача нахождения расстояния сводится к нахождению последовательности замен, минимизирующих суммарную цену (штраф).

Для префиксов строк a[1..i] и b[1..j] расстояние Левенштейна вычисляется по формуле (1):

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j), \ \ if \ \min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{a!=b} \end{cases} \quad otherwise. \end{cases} (1)$$

1.1.2 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является модификацией расстояния Левенштейна. Операции: замена (R), вставка (I), удаление (D) символа + транспозиция (T) — перестановка двух соседних символов Совпадение обозначают M (match).

a — строка длины |a|; b — строка длины |b|; d(a,b) = da,b(|a|,|b|)

Расстояние Дамерау-Левенштейна вычисляется по формуле (2):

$$d_{a,b}(i,j) == \begin{cases} \max(i,j), \ \ if \ \min(i,j) = 0, \\ min \begin{cases} d_{a,b}(i-1,j) + 1 \\ d_{a,b}(i,j-1) + 1 \quad if \ i,j > 1 \ and \ a_i = b_{j-1} \ and \ a_{i-1} = b_j \\ d_{a,b}(i-1,j-1) + 1_{a!=b} \\ d_{a,b}(i-2,j-2) + 1 \end{cases} \\ min \begin{cases} d_{a,b}(i-1,j) + 1 \\ d_{a,b}(i,j-1) + 1 \quad otherwise \\ d_{a,b}(i-1,j-1) + 1_{a!=b} \end{cases} \end{cases} (2)$$

1.2 Задание на выполнение лабораторной работы

Цели данной лабораторной работы:

1) изучение алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;
2) получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
3) сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени);
4) экспериментальное подтверждение различий во временной эффективности рекурсивной и не рекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк.

2 Конструкторская часть

В данном разделе представлены блок-схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

2.1.1 Инициализация матрицы

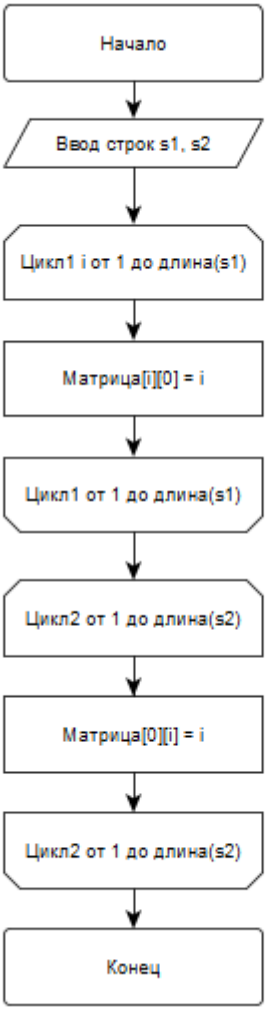

Рисунок 2.1 Инициализация матрицы

2.1.2 Матричный алгоритм Левенштейна

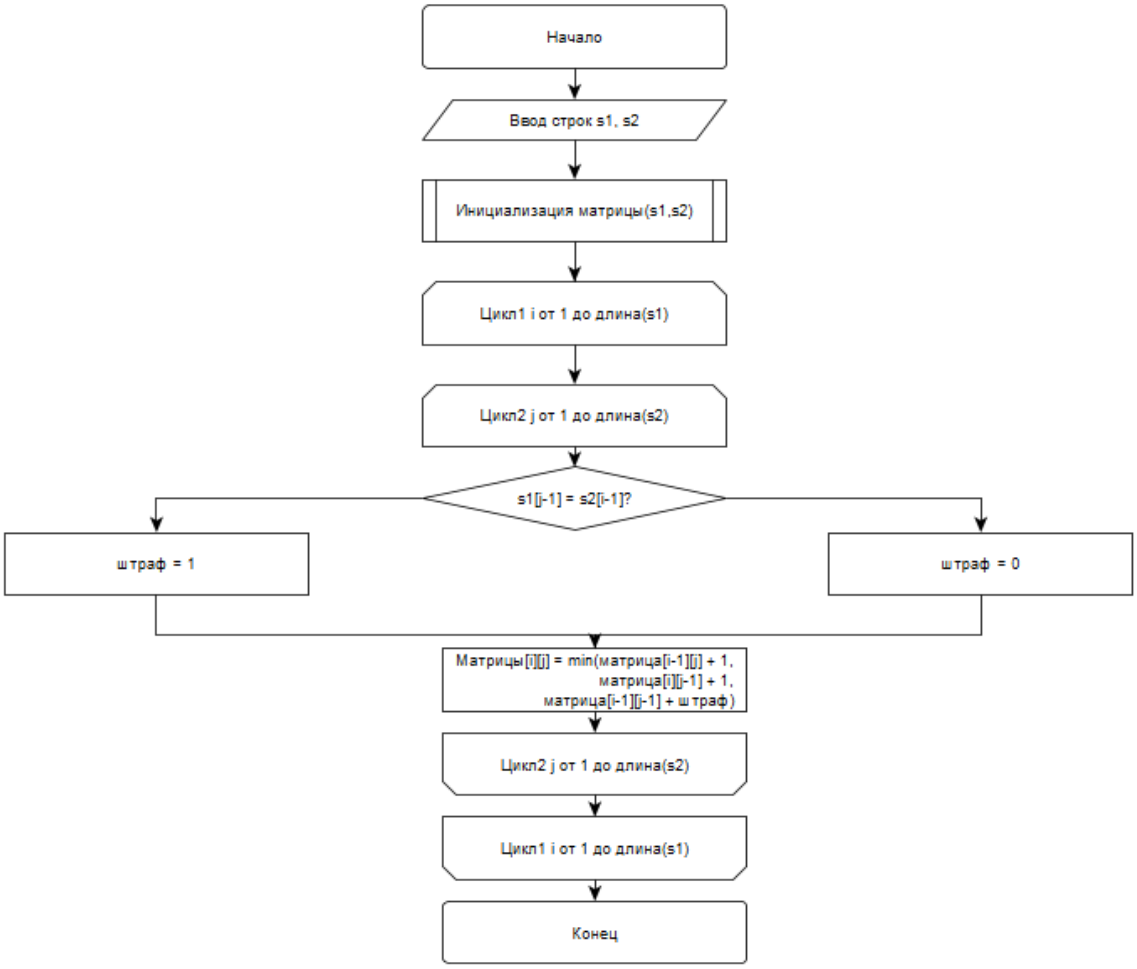

Рисунок 2.2 Матричный алгоритм Левенштейна.

2.1.3 Матричный алгоритм Дамерау-Левештейна

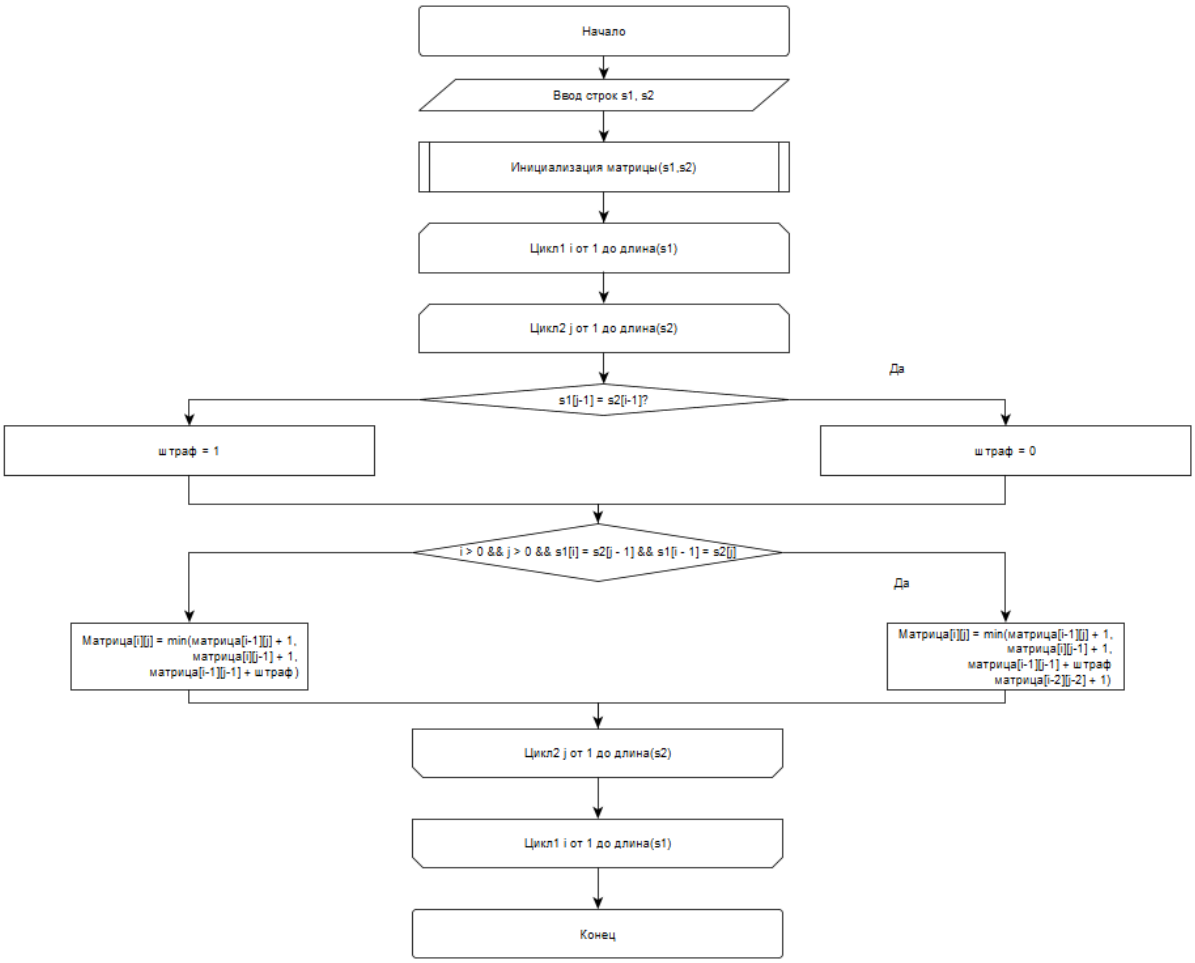

Рисунок 2.3 Матричный алгоритм Дамерау-Левенштейна.

2.1.4 Рекурсивный алгоритм Левенштейна

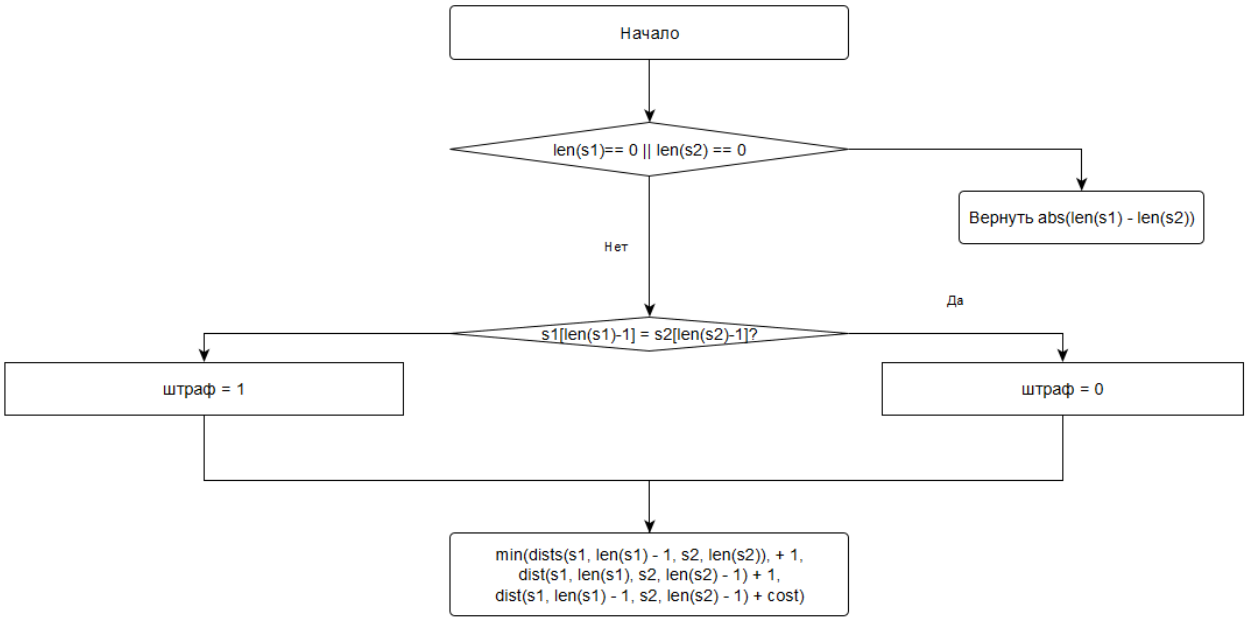

Рисунок 2.3 Рекурсивный алгоритм Левенштейна.

2.2 Выводы по конструкторскому разделу

Были представлены блок-схемы рассматриваемых алгоритмов.

3 Технологическая часть

В данном разделе представлены выбор языка программирования и реализации алгоритмов.

Требования к программе:

a) Программа отрабатывает корректно при пустом вводе;

b) Буквы верхнего и нижнего регистра считаются разными.

3.1 Средства реализации

В качестве языка программирования в данной работе был выбран Forth – конкатентативный язык программирования, в котором программы записываются последовательностью лексем. Является стековым ЯП: математические выражения представляются постфиксной записью при использовании стека. При написании программы использовался интерпретатор Gforth.

3.2 Листинг кода

3.2.1 Классический алгоритм Левенштейна

Листинг 1

```
: [!] ( value index array -- ) swap cells + ! ;

: cost?
  = if 0 + else 1 + then ;

: levenshtein col 1 ?DO ( --  . I - rows J - cols )
  rows 1 ?DO
  1 +
  lev_m I col * J 1 - + cells + @
  1 +
  lev_m I 1 - col * J 1 - + cells + @
  word1 I 1 - chars + c@
  word2 J 1 - chars + c@
  cost?
  min
  min
  I col * J + lev_m [!]
  clearstack
  LOOP LOOP
;

: fill_rows ( 0 -- )
  rows col * 0 ?DO
  DUP
  I lev_m [!]
  1 +
  col +LOOP ;

: fill_col ( -- )
  col 0 DO I I lev_m [!] LOOP ;

: levenshtein_basic ( -- )
  CR ." LEVENSHTEIN_MATRIX:"
  clearstack
  0
  fill_rows
  fill_col
  levenshtein
  lev_m col rows * 1 - cells + @
;
```

3.2.2 Классический алгоритм Дамерау-Левенштейна

Листинг 2

```
1   : transp? ( i j -- arr[i-2][j-2]+1 / infiniity )
1    0 > 0 >
2    OVER OVER
3    word2 2 - chars + c@
4    word1 1 - chars + c@
5    =
6    OVER OVER
7    word2 1 - chars + c@
8    word1 2 - chars + c@
9    =
10   and and and
11   IF lev_m I 1 - col * J 1 - 1 + + cells + @ ELSE 1000
12   THEN ;
13  : cost?
14    = if 0 + else 1 + then ;
15  : dam-lev col 1 ?DO ( --  . I - rows J - cols )
16    rows 1 ?DO
17    lev_m I 1 - col * J + cells + @
18    1 +
19    lev_m I col * J 1 - + cells + @
20    1 +
21    lev_m I 1 - col * J 1 - + cells + @
22    word1 I 1 - chars + c@
23    word2 J 1 - chars + c@
24    cost?
25    min min
26    I J  transp?
27    min
28    I col * J + lev_m [!]
29   LOOP LOOP
30  ;
31
32  : damerau–levenshtein
33    clearstack
34    0
```

```
35    fill_rows
36    fill_col
37    dam-lev
38    lev_m col rows * 1 - cells + @ ;
```

3.2.3 Рекурсивный алгоритм Левенштейна:

Листинг 3

```
: distl                    ( a1 n1 a2 n2 -- distance )
  dup
  if
     2>r dup
    if
     2dup 1- chars + c@ 2r@ 1- chars + c@ =
     if
       1- 2r> 1- recurse exit
     else
       2dup 1- 2r@ 1- recurse -rot
       2dup    2r@ 1- recurse -rot
       1- 2r> recurse min min 1+
     then
    else
     2drop  2r> swap drop
    then
  else
    2drop swap drop
  then
;
```

3.3 Выводы по технологическому разделу

Был выбран язык программирования, описаны требования к программе, а также предоставлен код рассматриваемых алгоритмов.

4 Экспериментальная часть

В данном разделе показаны примеры работы готового ПО, анализ работы алгоритмов по времени.

4.1 Примеры работы

Классический алгоритм Левенштейна:

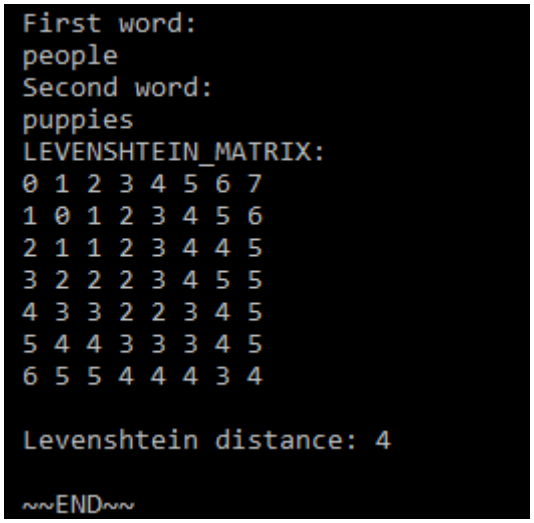

Рисунок 1. Пример работы программы алгоритма Левенштейна

Классический алгоритм Дамерау-Левенштейна:

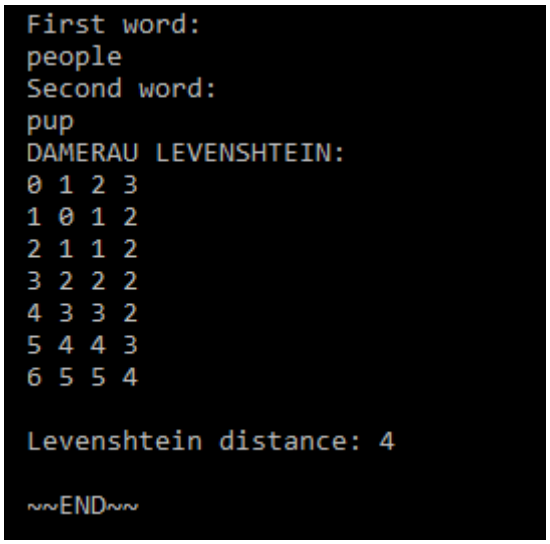

Рисунок 2 пример работы программы алгоритма Дамерау-Левенштейна

Рекурсивный алгоритм Левенштейна:

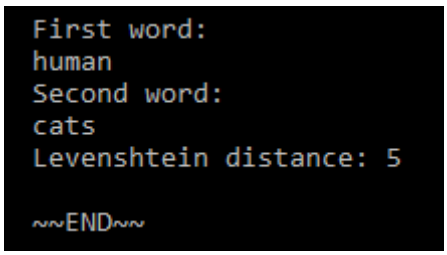

Рисунок 3 пример работы рекурсивного алгоритма

4.2 Результаты тестирования

Данная программа была проверена на ожидаемых и граничных наборах входных данных, а также на потенциально ошибочных ситуациях (пустой ввод).

№	Первое слово	Второе слово	Ожидаемый результат	Алгоритм Левенштейна	Алгоритм Дамерау-Левенштейна	Рекур. алгоритм Левеншт.
0	cats	cast	2	2	1	2
1	left	tfel	4	4	4	4
2	abcdefg	Qwerty	7	7	7	7
3	d	lalalalla	9	9	9	9
4	lalalal	g	7	7	7	7
5	Cats	cats	1	1	1	1
6	CATS	cats	4	4	4	4
7	[пустой ввод]	Star	4	4	4	4
8	star	[пустой ввод]	4	4	4	4
9	[пустой ввод]	[пустой ввод]	0	0	0	0

4.3 Постановка эксперимента по замеру времени

Измерение времени работы программы производится с помощью функции, возвращающей результат в секундах:

```
: time: ( "word" -- )
utime 2>R ' EXECUTE
utime 2R> D-
<# # # # # # # [CHAR] . HOLD #S #> TYPE ." seconds" ;
```

Замеры времени производятся для каждого алгоритма на строках разной длины.

4.4 Сравнительный анализ на основе экспериментальных данных

Длины строк s1 и s2	Матричный алгоритм	Матричный алгоритм	Рекурсивный алгоритм

	Левенштейна, сек.	Дамерау-Левенштейна, сек.	Левенштейна, сек.
3 3	**0.000**	**0.000**	**0.000**
7 7	**0.000**	**0.000**	**0.001**
10 10	**0.000**	**0.000**	**0.136**
100 100	**0.001**	**0.003**	
8 27	**0.000**	**0.000**	**0.002**
18 12	**0.000**	**0.000**	**0.048**
70 70	**0.001**	**0.001**	
90 90	**0.001**	**0.001**	
100 100	**0.001**	**0.002**	

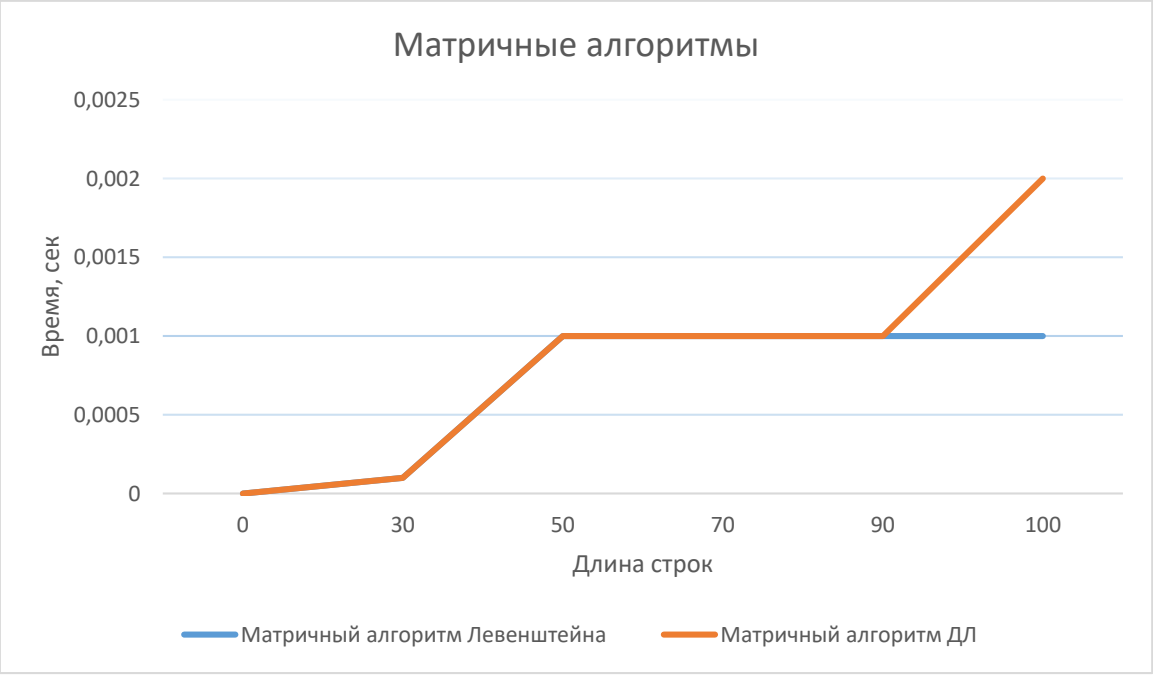

Рисунок 4.1 Сравнение матричных алгоритмов

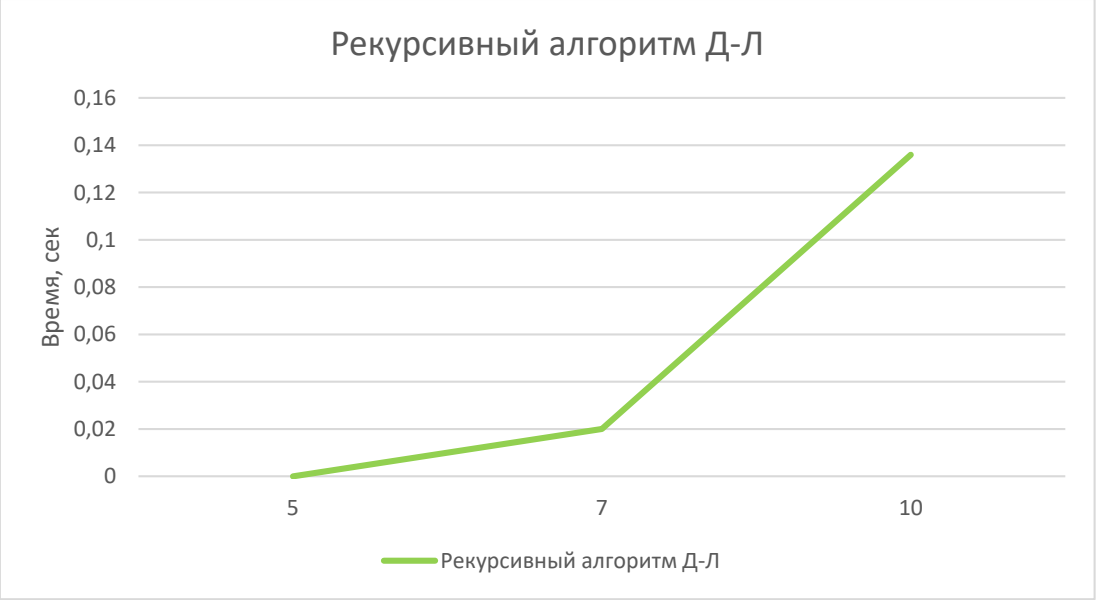

Рисунок 4.2 Работа рекурсивного алгоритма

4.5 Выводы по экспериментальной части

Алгоритмы были проверены на словах различной длины. На основе полученных данных можно сделать следующие выводы: матричный алгоритм Левенштейна работает быстрее, чем его модификация. Чем больше размер строк, тем больше разница во времени. Так, уже при длине слов в 100 символов, классический алгоритм Левенштейна показывает результат в 0.001 сек., в то время как алгоритм Дамерау-Левенштейна – 0.003. Рекурсивный алгоритм, по сравнению с нерекурсивными, является неэффективным по времени из-за частого обращения к рекурсивным запросам и осуществления лишни вычислений. При длине слов в 10 символов, рекурсивный алгоритм Левенштейна выполняется за 0.136 сек. В связи с этим, рекурсивный алгоритм не стоит применять к строкам большой длины, так как это может существенно увеличить время работы программы.

Заключение

В результате выполнения данной работы были изучены и реализованы матричные алгоритмы Левенштейна и Дамерау-Левенштейна, а также рекурсивный алгоритм Левенштейна. Был проведён сравнительный анализ данных алгоритмов: экспериментально было подтверждено различие во временной эффективности рекурсивной и нерекурсивным реализаций выбранного алгоритма. На основе полученных данных были сделаны следующие выводы: матричный алгоритм Левенштейна работает быстрее, чем его модификация. Чем больше размер строк, тем больше разница во времени. Так, уже при длине слов в 100 символов, классический алгоритм Левенштейна показывает результат в 0.001 сек., в то время как алгоритм Дамерау-Левенштейна – 0.003. Рекурсивный алгоритм, по сравнению с нерекурсивными, является неэффективным по времени из-за частого обращения к рекурсивным запросам и осуществления лишни вычислений. Уже при длине слов в 10 символов, рекурсивный алгоритм Левенштейна существенно замедляется и выполняется за 0.136 сек. Был сделан вывод, что рекурсивный алгоритм, по сравнению с нерекурсивными, является неэффективным по времени из-за частого обращения к рекурсивным запросам.

Список использованной литературы.

1. Обзор методов нечёткого поиска информации, Журнал "Вестник Московского государственного университета печати" Выпуск № 2 / 2013
2. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965.
3. В. И. Левенштейн. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965.
4. Редакционное расстояние между двумя строками [Электронный ресурс]. URL: http://www.math.spbu.ru/user/jvr/DA_html/_lec_1_13.html
5. R. A. Wagner, M. J. Fischer. The string-to-string correction problem. (1974).
6. Стандарт Forth, [Электронный ресурс], URL: https://forth-standard.org/
7. Л. Броуди, Начальный курс программирования на языке ФОРТ, 1990.
8. Stephen Pelc, Programming Forth Stephen Pelc, 2005.
9. Leo Brodie, Thinking Forth, 1984.
10. J. L. Bezemer, And so forth, 2004.
11. С. Н. Баранов, Н. Р. Ноздрунов, Язык Форт и его реализации, 1988.
12. Язык Форт строками [Электронный ресурс]. URL: https://www.forth.org.ru/
13. Gforth Manual строками [Электронный ресурс]. URL: http://www.complang.tuwien.ac.at/forth/gforth/Docs-html/