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: Марабян К. ИУ7-55

Москва 2018

**Содержание**

[**Введение**](#_y1ea5kse8d8s)2

[**1 Аналитическая часть**](#_n91k6n79dzut)3

[**1.1 Постановка задачи**](#_3gqvaou3e8f5)3

[**1.2 Описание алгоритмов**](#_brhollpqxi2m)3

[**1.3 Область применения**](#_dg85m0w6uq69)4

[**2 Конструкторский раздел**](#_5g4cmomvb0gf)5

[**2.1 Разработка алгоритмов**](#_codkpmvflfqb)5

[**2.2. Сравнительный анализ рекурсивной и нерекурсивной реализаций**](#_a3ulpvywwkl3)7

[**3 Технологический раздел**](#_dgwh654ihhkp)9

[**3.1 Минимальные требования к программному обеспечению**](#_mncwt7me2c7x)9

[**3.2 Средства реализации**](#_80qaa7iwbkbc)9

[**3.3 Листинг кода**](#_l3quu9c8lbwe)10

[**4 Экспериментальная часть**](#_hlnbv34b5bsg) **13**

[**4.1 Примеры работы**](#_m9z4zuamn45l) **13**

[**4.2 Постановка эксперимента**](#_hcalz8peoopr) **16**

**4.3 Сравнительный анализ рекурсивного и не рекурсивного алгоритмов. 18**

[**Заключение**](#_1hl6wz3mwv0x) **25**

##### 

##### Введение

Алгоритмы нечеткого поиска (также известного как поиск по сходству или fuzzy string search) являются основой систем проверки орфографии и полноценных поисковых систем вроде Google или Yandex. Например, такие алгоритмы используются для функций наподобие «Возможно вы имели в виду …» в тех же поисковых системах.

В данной работе будут проведено сравнительное тестирование качества и производительности алгоритмов.

Цель данной лабораторной̆ 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 биохимической̆ информатике, данный̆ метод широко используют для срав­ нения генов, белков, а также хромосом.

Задачами данной лабораторной работы являются:

1) изучение алгоритмов Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками;

2) применение метода динамического программирования для матричной реализации указанных алгоритмов;

3) получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;

##### 1 Аналитическая часть

В данном разделе представлены теоретическое описание алгоритмов и указание области их применения.

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).

##### 1.1 Постановка задачи

Реализация алгоритма поиска расстояния Левенштейна:

а) базовая версия;

б) модификация Дамерау-Левенштейна;

в) базовая версия/модификация через рекурсию.

##### 1.2 Описание алгоритмов

Пусть s1 и s2 — две строки (длиной m и n соответственно) над некото­рым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) D(s1, s2) можно подсчитать по следующей рекуррентной 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]) + ), где = 0, если s1[m] = s2[n], иначе = 1

Разрешенные операции и штрафы описаны ниже.

1. Замена символа. Штраф 1.
2. Вставка символа. Штраф 1.
3. Удаление символа. Штраф 1.
4. Совпадение символа. Штраф 0.

В модифицированном алгоритме добавлена еще одна операция:

Перестановка символа. Штраф 1.

В рекуррентную формулу добавляется еще один член минимума:

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

##### 1.3 Область применения

Расстояние Левенштейна и его обобщения активно применяется:

1. для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи);
2. для сравнения текстовых файлов.

##### 2 Конструкторский раздел

В этом разделе приведены схемы алгоритмов и сравнительный анализ рекурсивной и не рекурсивной реализаций.

##### 2.1 Разработка алгоритмов

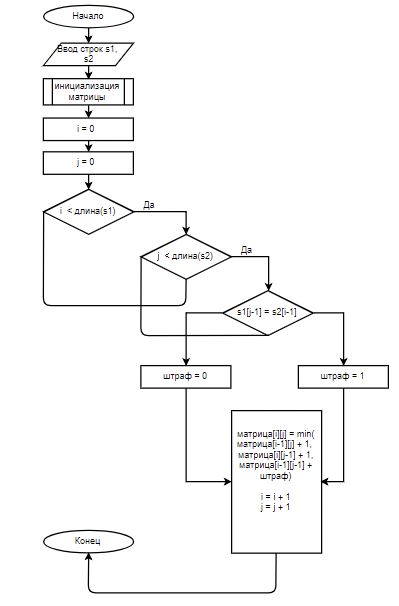


Рисунок 1 - схема алгоритма Левенштейна.

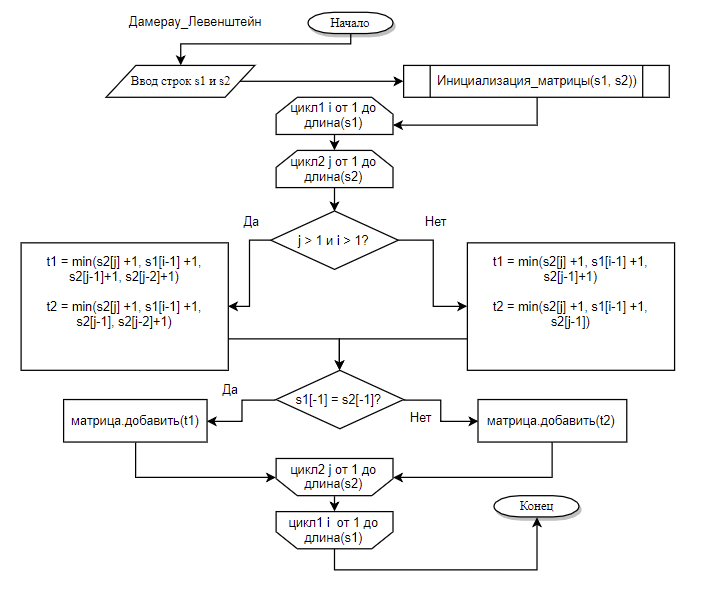


Рисунок 2 - схема алгоритма Дамерау-Левенштейна.

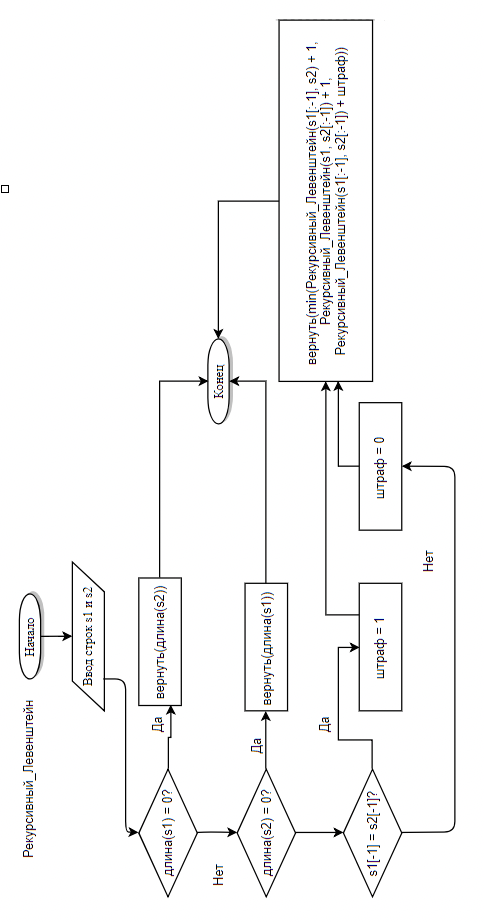


Рисунок 3 - схема рекурсивного алгоритма Левенштейна.

##### 2.2. Сравнительный анализ рекурсивной и не рекурсивной реализаций

Результат выполнения рекурсивного алгоритма совпадает с результатом, полученным не рекурсивным алгоритмом. Однако, алгоритм с рекурсией является менее неэффективным за счет того, что некоторые подзадачи решаются по несколько раз. Это отражается на времени выполнения программы. Не рекурсивные реализации работают значительно быстрее.

##### 3 Технологический раздел

Данный раздел содержит указание использованного языка программирования и использованных средств замера процессорного времени, а также листинг кода.

##### 3.1 Минимальные требования к программному обеспечению

Операционные системы: winXP/7/10, MacOS.

Объем жесткого диска: 50 Мб.

Оперативная память: 512 MB ОЗУ.

##### 3.2 Средства реализации

Для реализации алгоритмов был выбран язык программирования C#. Приоритетом разработчиков этого языка была его простота, и так как это язык высокого уровня, он больше похож на английский, чем другие. C# позволяет разработчику сосредоточиться на алгоритме, а не на деталях реализации, так как сложные конструкции в нём заключены в абстракции.

##### 3.3 Листинг кода

Алгоритм Левенштейна:

|  |
| --- |
| public int Levenshtein()  {  if (firstString == null) throw new ArgumentNullException("string1");  if (secondString == null) throw new ArgumentNullException("string2");  int diff;  for (int i = 0; i <= firstString.Length; i++) { matrixLevenshtein[i, 0] = i; }  for (int j = 0; j <= secondString.Length; j++) { matrixLevenshtein[0, j] = j; }  for (int i = 1; i <= firstString.Length; i++)  {  for (int j = 1; j <= secondString.Length; j++)  {  diff = (firstString[i - 1] == secondString[j - 1]) ? 0 : 1;  matrixLevenshtein[i, j] = Math.Min(Math.Min(matrixLevenshtein[i - 1, j] + 1,  matrixLevenshtein[i, j - 1] + 1),  matrixLevenshtein[i - 1, j - 1] + diff);  }  }  return matrixLevenshtein[firstString.Length, secondString.Length];  } |

Алгоритм Дамерау-Левенштейна:

|  |
| --- |
| public int DamerauLevenshtein()  {  if (firstString == null) throw new ArgumentNullException("string1");  if (secondString == null) throw new ArgumentNullException("string2");  int diff;  for (int i = 0; i <= firstString.Length; i++)  matrixDamerauLevenshtein[i, 0] = i;  for (int j = 0; j <= secondString.Length; j++)  matrixDamerauLevenshtein[0, j] = j;  for (int i = 1; i <= firstString.Length; i++)  {  for (int j = 1; j <= secondString.Length; j++)  {  diff = secondString[j - 1] == firstString[i - 1] ? 0 : 1;  var vals = new int[]  {  matrixDamerauLevenshtein[i - 1, j] + 1,  matrixDamerauLevenshtein[i, j - 1] + 1,  matrixDamerauLevenshtein[i - 1, j - 1] + diff  };  matrixDamerauLevenshtein[i, j] = vals.Min();  if (i > 1 && j > 1 && firstString[i - 1] == secondString[j - 2] && firstString[i - 2] == secondString[j - 1])  matrixDamerauLevenshtein[i, j] = Math.Min(matrixDamerauLevenshtein[i, j], matrixDamerauLevenshtein[i - 2, j - 2] + 1);  }  }  return matrixDamerauLevenshtein[firstString.Length, secondString.Length];  } |

Рекурсивный алгоритм Левенштейна:

|  |
| --- |
| public int RecLevenshtein(string firstString, int firstStringLen, string secondString, int secondStringLen)  {  int diff;  if (firstStringLen == 0)  return secondStringLen;  if (secondStringLen == 0)  return firstStringLen;  if (firstString[firstStringLen - 1] == secondString[secondStringLen - 1])  diff = 0;  else  diff = 1;  return Math.Min(Math.Min(  RecLevenshtein(firstString, firstStringLen - 1, secondString, secondStringLen) + 1,  RecLevenshtein(firstString, firstStringLen, secondString, secondStringLen - 1) + 1),  RecLevenshtein(firstString, firstStringLen - 1, secondString, secondStringLen - 1) + diff);  } |

##### 4 Экспериментальная часть

В данном разделе представлены примеры работы для нескольких случаев с указанием матричных значений для нерекурсивных реализаций и результирующих длин для всех реализаций

##### 4.1 Примеры работы

В этом разделе представлены примеры работы для нескольких случаев с указанием матричных значений для нерекурсивных реализаций и результирующих длин для всех реализаций.

Таблица 1

Тестовые данные (пример 1)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Первое слово | Второе слово | Левенштейн  (мсек) | Дамерау - Левенштейн  (мсек) | Рек. Левенштейн  (мсек) | Результат |
| телефон | телефно | 3 | 17 | 2615 | 212 |

Матрица Левенштейна:

0 1 2 3 4 5 6 7

1 0 1 2 3 4 5 6

2 1 0 1 2 3 4 5

3 2 1 0 1 2 3 4

4 3 2 1 0 1 2 3

5 4 3 2 1 0 1 2

6 5 4 3 2 1 1 1

7 6 5 4 3 2 1 2

Матрица Дамерау-Левенштейна:

0 1 2 3 4 5 6 7

1 0 1 2 3 4 5 6

2 1 0 1 2 3 4 5

3 2 1 0 1 2 3 4

4 3 2 1 0 1 2 3

5 4 3 2 1 0 1 2

6 5 4 3 2 1 1 1

7 6 5 4 3 2 1 1

Таблица 2

Тестовые данные (пример 2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Первое слово | Второе слово | Левенштейн  (мсек) | Дамерау - Левенштейн  (мсек) | Рек. Левенштейн  (мсек) | Результат |
| рост | гост | 0 | 2 | 13 | 111 |

Матрица Левенштейна:

0 1 2 3 4

1 1 2 3 4

2 2 1 2 3

3 3 2 1 2

4 4 3 2 1

Матрица Дамерау-Левенштейна:

0 1 2 3 4

1 1 2 3 4

2 2 1 2 3

3 3 2 1 2

4 4 3 2 1

Таблица 3

Tестовые данные (пример 3)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Первое слово | Второе слово | Левенштейн  (мсек) | Дамерау - Левенштейн  (мсек) | Рек. Левенштейн  (мсек) | Результат |
| лебедь | ьдебел | 1 | 8 | 454 | 444 |

Матрица Левенштейна:

0 1 2 3 4 5 6

1 1 2 3 4 5 5

2 2 2 2 3 4 5

3 3 3 3 2 3 4

4 4 4 3 3 2 3

5 5 4 4 4 3 3

6 5 5 5 5 4 4

Матрица Дамерау-Левенштейна:

0 1 2 3 4 5 6

1 1 2 3 4 5 5

2 2 2 2 3 4 5

3 3 3 3 2 3 4

4 4 4 3 3 2 3

5 5 4 4 4 3 3

6 5 5 5 5 4 4

##### 4.2 Постановка эксперимента

В данной работе было решено провести два исследования на время работы (в тиках процессора) в зависимости от количества символов в строке:

1. Берется среднее время на 100 тестов. Размер строк по 1000.
2. Размер строк 10 и 100. Количество тестов увеличено до 10000.

В строке переставлены соседние буквы в середине слова.

Под str1 и str2 подразумевается количество символов.

Таблица 4 - тестовые данные (эксперимент 1).

|  |  |  |  |
| --- | --- | --- | --- |
| str1 | str2 | Левенштейна | Дамерау-Левенштейна |
| 30 | 24 | 46 | 47 |
| 33 | 27 | 42 | 47 |
| 34 | 26 | 46 | 63 |
| 79 | 47 | 46 | 47 |
| 89 | 49 | 46 | 47 |

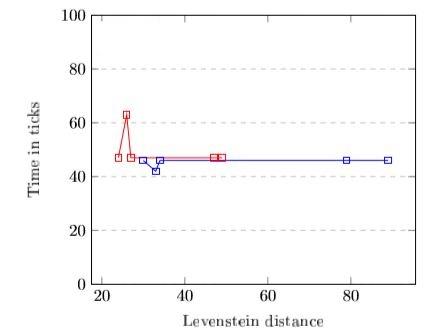


Рисунок 4 - график для тестовых данных из таблицы 4.

Таблица 5 - тестовые данные (эксперимент 2).

|  |  |  |  |
| --- | --- | --- | --- |
| str1 | str2 | Левенштейна | Дамерау-Левенштейна |
| 90 | 90 | 327 | 359 |
| 90 | 90 | 405 | 421 |
| 93 | 92 | 403 | 473 |
| 93 | 93 | 421 | 546 |

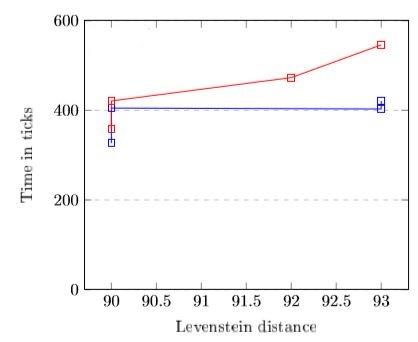
****

Рисунок 5 - график для тестовых данных из таблицы 5.

Подводя итоги проделанной работы, можно сделать вывод, что рекурсивный алгоритм при более простой реализации работает чрезвычайно долго, что делает его использование нецелесообразным. Не рекурсивный алгоритм значительно превосходит его по эффективности. Алгоритм с добавленной операцией работает дольше, т.к. имеет более сложную логику, но все равно значительно превосходит по скорости рекурсивный алгоритм.

##### 4.3 Сравнительный анализ рекурсивного и не рекурсивного алгоритмов

Первое слово: рост

Второе слово: гост

Пример вызова не рекурсивного алгоритма:

[First string: р First string: г]

[First string: р First string: о]

[First string: р First string: с]

[First string: р First string: т]

[First string: о First string: г]

[First string: о First string: о]

[First string: о First string: с]

[First string: о First string: т]

[First string: с First string: г]

[First string: с First string: о]

[First string: с First string: с]

[First string: с First string: т]

[First string: т First string: г]

[First string: т First string: о]

[First string: т First string: с]

[First string: т First string: т]

Пример вызова рекурсивного алгоритма (481 раз):

[Rec First string: т First string: т]

[Rec First string: с First string: т]

[Rec First string: о First string: т]

[Rec First string: р First string: т]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: с]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: с First string: с]

[Rec First string: о First string: с]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: с First string: о]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: с]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: т First string: с]

[Rec First string: с First string: с]

[Rec First string: о First string: с]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: с First string: о]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: т First string: о]

[Rec First string: с First string: о]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: т First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: о]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: с]

[Rec First string: о First string: с]

[Rec First string: р First string: с]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: с First string: о]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

[Rec First string: с First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: о First string: о]

[Rec First string: р First string: о]

[Rec First string: р First string: г]

[Rec First string: о First string: г]

[Rec First string: р First string: г]

[Rec First string: р First string: г]

##### Заключение

Изучены алгоритмы Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками. Применен метод динамического программирования для матричной реализации указанных алгоритмов. Получены практические навыки реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии. Проведен сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти). Было получено экспериментальное подтверждение различий во временнóй 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. Сделан отчет о выполненной лабораторной работе.