**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Анализ алгоритмов**

**Лабораторная работа №1**

**Отчёт на тему:**

# «Расстояния Левенштейна и Дамерау-Левенштейна»

Выполнила:

Янова Даниэлла

ИУ7-53

**Москва, 2018**

**Введение**

Расстояние Левенштейна (также редакционное расстояние или дистанция редактирования) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Впервые задачу упомянул в 1965 году советский математик Владимир Иосифович Левенштейн при изучении последовательностей 0-1. Впоследствии более общую задачу для произвольного алфавита связали с его именем. Большой вклад в изучение вопроса внёс Дэн Гасфилд. Расстояние Левенштейна и его обобщения активно применяется для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи); для сравнения текстовых файлов утилитой diff и ей подобными (здесь роль «символов» играют строки, а роль «строк» — файлы); в биоинформатике для сравнения генов, хромосом и белков. С точки зрения приложений определение расстояния между словами или текстовыми полями по Левенштейну обладает следующими недостатками: при перестановке местами слов или частей слов получаются сравнительно большие расстояния; расстояния между совершенно разными короткими словами оказываются небольшими, в то время как расстояния между очень похожими длинными словами оказываются значительными. Расстояние Дамерау — Левенштейна (названо в честь учёных Фредерика Дамерау и Владимира Левенштейна) — это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов. Расстояние Дамерау-Левенштейна, как и метрика Левенштейна, является мерой "схожести"двух строк. Алгоритм его поиска находит применение в реализации нечёткого 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).

**Задачи работы:**

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;

4. Сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);

5. Экспериментальное подтверждение различий во временн´oй эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк;

6. Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

**1. Аналитическая часть**

В этом разделе затронуты темы, описанные ниже.

1. Определение и формула нахождения расстояния левенштейна.
2. Определение и формула нахождения расстояния Дамерау-Левенштейна.
3. Принцип поиска расстояния для обоих случаев.
4. Применение

**1.1 Описание алгоритмов**

Ниже будут изложены словесные описания алгоритмов.

**1.1.1 Расстояние Левенштейна**

(также редакционное расстояние или дистанция редактирования) между двумя строками в теории информации и компьютерной лингвистике - это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую. Формула, для нахождения расстояния Левенштейна:

Для нахождения кратчайшего расстояния необходимо вычислить матрицу D, используя вышеприведённую формулу. Её можно вычислять как по строкам, так и по столбцам. Для восстановления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла (M,N) в левый верхний, на каждом шаге ища минимальное из трёх значений:

• если минимально (D(i-1, j) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i-1, j);

• если минимально (D(i, j-1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j-1);

• если минимально (D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i-1, j-1) Здесь (i, j) — клетка матрицы, в которой мы находимся на данном шаге. Если минимальны два из трёх значений (или равны все три), это означает, что есть 2 или 3 равноценных редакционных предписания.

**1.1.2 Расстояние Дамерау-Левенштейна**

Расстояние Дамерау-Левенштейна - это мера разницы двух строк символов, определяемая как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для перевода одной строки в другую. Является модификацией расстояния Левенштейна: к операциям вставки, удаления и замены символов, определённых в расстоянии Левенштейна добавлена операция транспозиции (перестановки) символов.

Формула, для нахождения расстояния:

Для нахождения кратчайшего расстояния необходимо вычислить матрицу D, используя вышеприведённую формулу. Её можно вычислять как по строкам, так и по столбцам. Для восстановления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла (M,N) в левый верхний, на каждом шаге ища минимальное из трёх значений:

• если минимально (D(i-1, j) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i-1, j);

• если минимально (D(i, j-1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j-1);

• если минимально (D(i-1, j-1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i-1, j-1) Здесь (i, j) — клетка матрицы, в которой мы находимся на данном шаге. Если минимальны два из трёх значений (или равны все три), это означает, что есть 2 или 3 равноценных редакционных предписания.

• если транспозиция возможна, то возвращаем (D(i-2, j-2) + 1

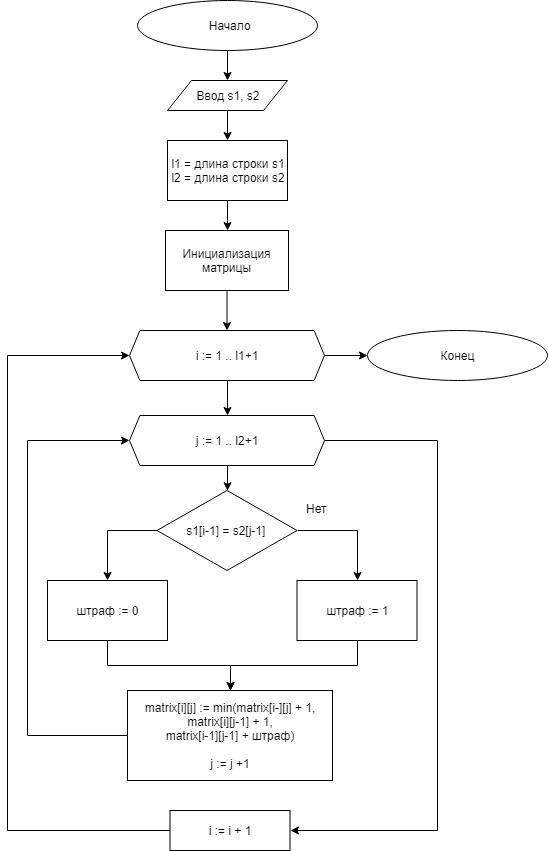
**1.2.3 Область применения алгоритмов**

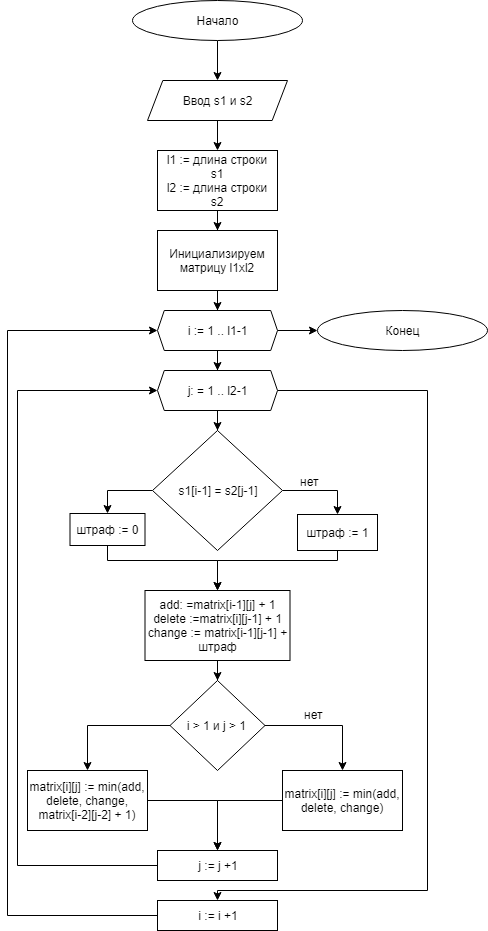
Данные алгоритмы применяются при поиске информации по запросу, с помощью них можно, найти, если пользователь допустил одну или несколько ошибок в слове, наиболее подходящее(имеющее наименьшее расстояние) к нему слово и заменить его в поисковой строке.

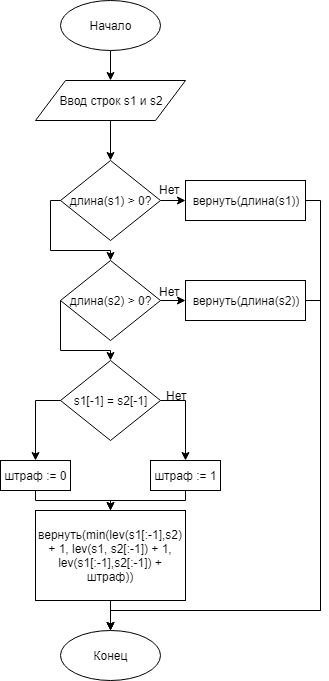
**2. Констукторский раздел**

В данном разделе размещены блоксхемы алгоритмов и сравнительный анализ линейной и рекурсивной реализаций.

**2.1 Разработка реализаций алгоритма**

 Рисунок 1.1 – Схема классической реализация алгоритма Левенштейна

Рисунок 1.2 – Схема алгоритма Дамерау-Левенштейна

Рисунок 1.3 – Схема рекурсивной реализации алгоритма Левенштейна

**2.2 Сравнительный анализ рекурсивной и нерекурсивной реализаций** Рекурсивная реализация работает медленнее по сравнение с линейной из-за повторных вычислений, возникающих в ходе работы рекурсивного алгоритма, это наглядно видно на Рис. 8, иллюстрирующей дерево рекурсивных вызовов выбранной реализации.Функция rec() принимает на вход 2 строки и 2 индекса, но чтобы не нагружать схему, 2 строки при вызове не подписываются. Для высчитывания цены перехода из диагонального элемента в текущий надо учитывать совпадают ли предыдущие символы строк, но поскольку это не влияет на дерево рекурсии, в схеме я опускаю этот момент и пишу просто число/вызов функции

**3. Технологический раздел**

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

**3.1 Требования к программному обеспечению**

Данная программа разрабатывалась на языке Python 3.7, поддерживаемом многими операционными системами. Для запуска программы необходим интрепритатор, поддерживающий Python 3.7 для 64-битной системы.

**3.2 Листинг кода 1**

**def levenstain(a, b, print\_flag):**

n, m = len(a), len(b)

**if** n > m:

a, b = b, a

n, m = m, n

matrix = []

current\_row = [x **for** x **in** range(n+1)]

matrix.append(current\_row)

**for** i **in** range(1, m+1):

previous\_row, current\_row = current\_row, [i]+[0]\*n

**for** j **in** range(1, n+1):

add = previous\_row[j]+1

delete = current\_row[j-1]+1

change = previous\_row[j-1] + (a[j-1] != b[i-1])

current\_row[j] = min(add, delete, change)

matrix.append(current\_row)

**if** print\_flag != 0:

print\_matrix(matrix,a,b)

**return** current\_row[n]

Листинг кода 1.1 — Классический алгоритм Левенштейна

**def** damerau\_levenstain(a, b, print\_flag):

n, m = len(a), len(b)

**if** n > m:

a, b = b, a

n, m = m, n

matrix = []

previous\_row = [x **for** x **in** range(n+1)]

current\_row = [1]+[0]\*n

**for** i **in** range(1, n+1):

add = previous\_row[i]+1

delete = current\_row[i-1]+1

change = previous\_row[i-1] + (a[i-1] != b[0])

current\_row[i] = min(add, delete, change)

matrix.append(previous\_row)

matrix.append(current\_row)

**for** i **in** range(2, m+1):

preprevious\_row, previous\_row, current\_row = previous\_row, current\_row, [i]+[0]\*n

**for** j **in** range(1, n+1):

add = previous\_row[j]+1

delete = current\_row[j-1]+1

change = previous\_row[j-1] + (a[j-1] != b[i-1])

**if** j == 1:

current\_row[j] = min(add, delete, change)

**else**:

**if** a[j-2] == b[i-1] **and** a[j-1] == b[i-2]:

current\_row[j] = min(preprevious\_row[j-2] + 1, add, delete, change)

**else**:

current\_row[j] = min(add, delete, change)

matrix.append(current\_row)

**if** print\_flag != 0:

print\_matrix(matrix,a,b)

**return** current\_row[n]

Листинг кода 1.2 — Алгоритм Дамерау-Левенштейна

**def** recurr\_levenstain(a, b):

n, m = len(a), len(b)

**if** n > m:

a, b = b, a

n, m = m, n

**return** find\_eom(n, m, a, b)

**def** find\_eom(i, j, a, b):

find\_eom.count += 1

**if** i == 0:

**return** j

**elif** j == 0:

**return** i

**else**:

el = min(find\_eom(i-1, j, a, b)+1, find\_eom(i, j-1, a, b)+1,

find\_eom(i-1, j-1, a, b)+ (a[i-1] != b[j-1]))

**return** el

Листинг кода 1.3 — Рекурсивный алгоритм Левенштейна

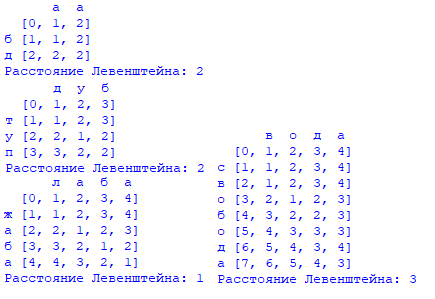
**4. Исследовательский раздел**

В этом разделе будут изложены примеры работы и исследования быстродействия алгоритмов.

**4.1 Пример работы**

Далее представлены матрицы, выводимые алгоритмами:

а) Алгоритм Левенштейна

Рисунок 2.1 — Примеры работы классического алгоритма Левенштейна

б)Алгоритм Дамерау-Левенштейна

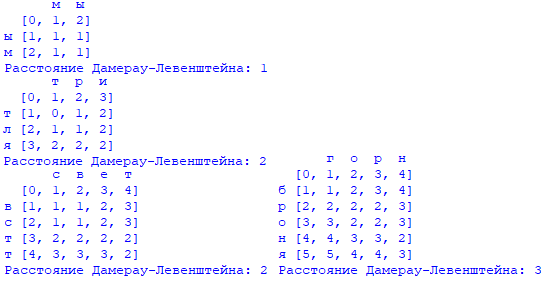
Рисунок 2.2 — Примеры работы Алгоритма Дамерау-Левенштейна

Таблица 1

Примеры работы алгоритмов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **s1** | **s2** | **Ожидание** | **Результат** | **Оценка (0, 1)** |
| 1 | лубянка | китаянка | 4 4 4 | 4 4 4 | 1 1 1 |
| 2 | анализ | алгоритмов | 8 8 8 | 8 8 8 | 1 1 1 |
| 3 | фрактал | фарктал | 2 1 2 | 2 1 2 | 1 1 1 |
| 4 | олло | лоол | 3 2 3 | 3 2 3 | 1 1 1 |
| 5 | ошибка | я | 6 6 6 | 6 6 6 | 1 1 1 |
| 6 | рота | подъем | 5 5 5 | 5 5 5 | 1 1 1 |
| 7 | программист | аутист | 7 7 7 | 7 7 7 | 1 1 1 |
| 8 | зима | Кодзима | 3 3 3 | 3 3 3 | 1 1 1 |
| 9 | примите | лабу | 7 7 7 | 7 7 7 | 1 1 1 |

1. алгоритм Левенштейна;

2. алгоритм Дамерау-Левенштейна;

3. рекурсивный алгоритм Левенштейна.

**4.2 Исследование скорости работы алгоритма**

Для исследования скоростных характеристик был использован компьютер на базе процессора Intel Core i7-4500U, содержащий 8 гигабайт оперативной памяти. Модуль тестирования запускался с жестокого диска под операционной системой Windows 10. Жесткий диск имел среднюю скорость передачи данных при чтении 502 Мбайт/с и 199 Мбайт/с при записи.

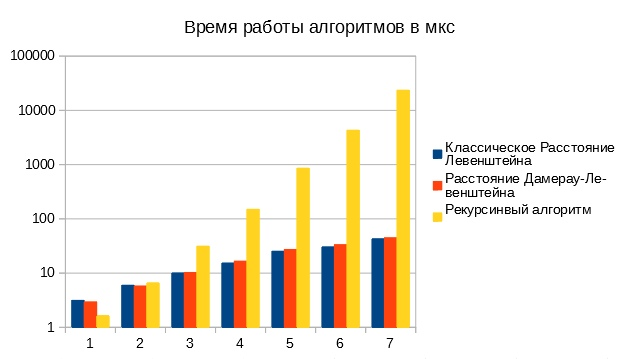
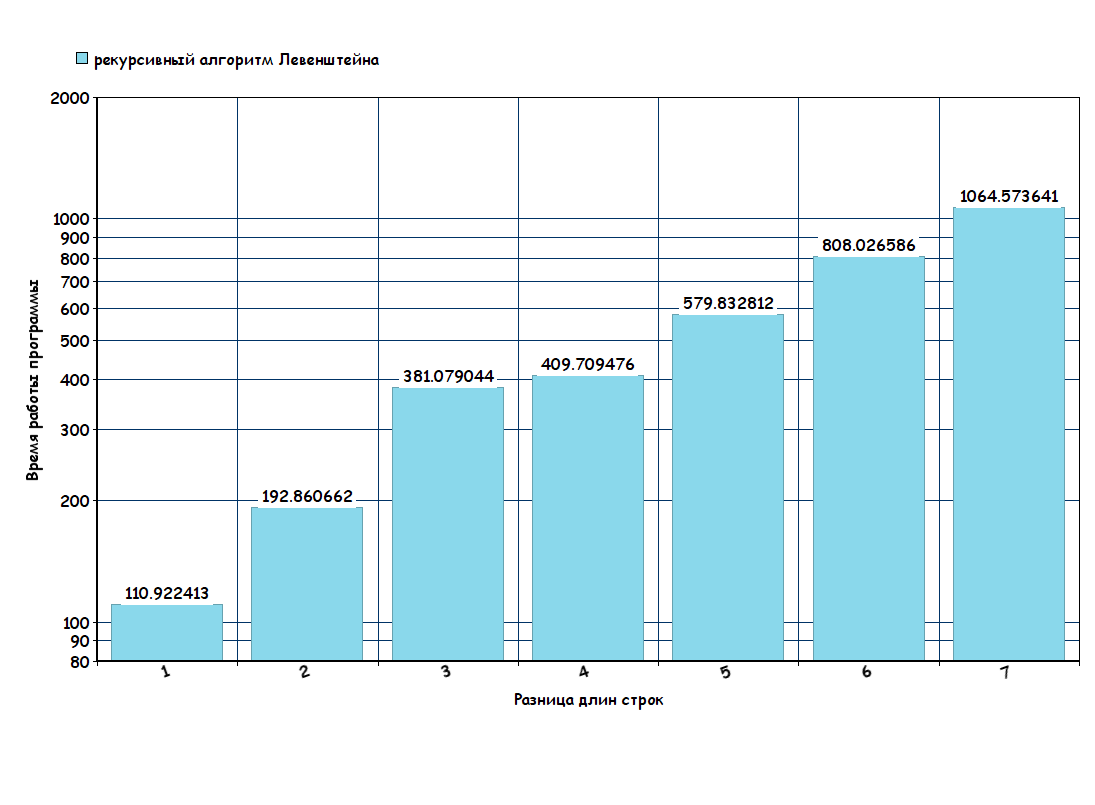
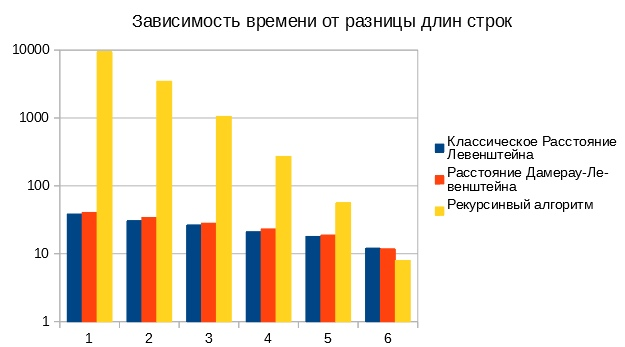


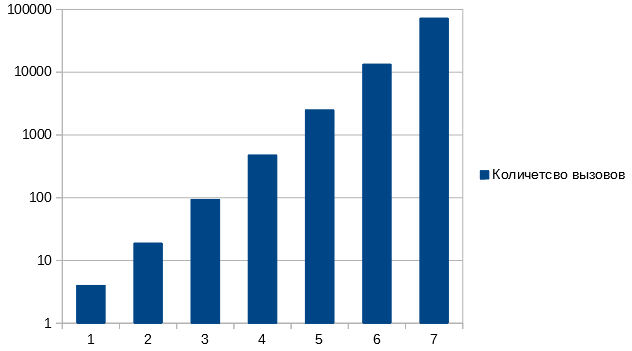
Рисунок 3.1 – Время выполнения программы в зависимости от длины строки

Рисунок 3.2 – Таблица тестовых результатов для строк разной длины

**4.3 Сравнительный анализ рекурсивной и нерекурсивной реализации**

Для временного сравнительного анализа рекурсивной реализации алгоритма Левенштейна, нерекурсивной реализации алгоритма Левенштейна и алгоритма Дамерау-Левенштейна использовалась методика многочисленных запусков двух типов: для слов одинаковой длины (от 1 до 7 символов) и для слов разной длины (минимальная длина строки – 3 символа, максимальная – 10 символов). В обоих случаях худшие результаты показывает рекурсивная реализация алгоритма Левенштейна. Можно сделать вывод, что данная версия алгортма не рекомендована к использованию в реальных проектах, так как она может существенно замедлить их работу.

Такое замедление по времени вызвано частыми обращениями к рекурсивным запросам, которые можно проследить на примере анализа вызова рекурсивной реализации от строк разной длины:



**Заключение**

В результате выполнения лабораторной работы были получены следующие основные навыки:

* изучены теоритеческие понятия в алгоритмах для нахождения расстояния Левенштейна и Дамерау-Левенштейна;
* рассмотрен один из современных способов оптимизаций и ускорения рекурсивный программ с помощью втроенного счетчика вызовов функций на Python 3;
* проведен аналитический вывод формул для заполнения матриц расстояний;
* проведено сравнение трех реализаций заданного алгоритма, выявлены их слабые места;
* 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;
* oпределение расстояния по Левенштейну имеет недостатки:
  + при перестановке местами слов или частей слов получаются большие расстояния;
  + расстояния между совершенно разными короткими словами будут меньше в отличии от расстояния между двумя длинными и очень похожими, но длинными словами.