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»

ЛАБОРАТОРНАЯ РАБОТА №1

Отчет на тему:

Расстояние Левенштейна

Монахов Д. И.

ИУ7-54

Москва 2018

Оглавление

[Введение 4](#_Toc525921184)

[1. Аналитическая часть 5](#_Toc525921185)

[1.1. Описание алгоритмов 5](#_Toc525921186)

[2. Конструкторская часть 6](#_Toc525921187)

[2.1 Разработка алгоритмов 6](#_Toc525921188)

[2.2 Сравнительный анализ рекурсивной и нерекурсивной реализаций 8](#_Toc525921189)

[3. Технологическая часть 9](#_Toc525921190)

[3.1 Требования к программному обеспечению 9](#_Toc525921191)

[3.2 Средства реализации 9](#_Toc525921192)

[3.3 Листинг кода 10](#_Toc525921193)

[4. Экспериментальная часть 13](#_Toc525921194)

[4.1 Примеры работы 13](#_Toc525921195)

[4.2 Постановка эксперимента 14](#_Toc525921196)

[4.3. Сравнительный анализ на материале экспериментальных данных 16](#_Toc525921197)

[Заключение 17](#_Toc525921198)

# Введение

В этом раздела будут рассмотрены цели и задачи данной работы.

**Цель работы**: изучить и применить метод динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна, а также получить практические навыки реализации указанных алгоритмов.

**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 реализации выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);
5. экспериментальное подтверждение различий во временной 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 этой части рассматривается, что такое расстояние Левенштейна и математически описываются алгоритмы, позволяющие найти это расстояние.

## 1.1. Описание алгоритмов

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.

Будем считать, что элементы строк нумеруются с первого, а не с нулевого.

Расстояние Левенштейна D(i,j) между i-ым элементом строки S1 и j-ым элементом строки S2 можно найти по следующей формуле:

D(i,j) = 

Расстояние Дамерау-Левенштейна D(i,j) между i-ым элементом строки S1 и j-ым элементом строки S2 можно найти по следующей формуле:

D(i,j) = 

# 2. Конструкторская часть

В этой части будут составлены блок-схемы алгоритмов, а сами алгоритмы будут подвергнуты сравнению аналитически.

## 2.1 Разработка алгоритмов

Псевдокод рекурсивной реализации алгоритма Левенштейна

функция ЛевенштейнаРекурсивно(i, j); возвращает целое

начало

если i равно 0 и j равно 0 то

вернуть 0

конец если

если только i или только j равны нулю то

вернуть отличное от нуля

конец если

вставка = ЛевенштейнаРекурсивно(i, j - 1) + 1

удаление = ЛевенштейнаРекурсивно(i - 1, j) + 1

М = ЛевенштейнаРекурсивно(i -1, j - 1) +

Совпадает(строка1[i - 1], строка2[j - 1])

вернуть наименьшее из (вставка, удаление, M)

конец

Псевдокод итеративной реализации алгоритма Левенштейна

функция ЛевенштейнаИтеративно; возвращает целое

начало

для i от 0 до длины строки 1; шаг = 1

для j от 0 до длины строки 2; шаг = 1

если i равно 0 и j равно 0 то

матрица[i,j] = 0

иначе если только i или только j равны нулю то

матрица[i,j] = отличное от нуля (i, j)

конец если

иначе

если строка1[i - 1] равно строка2[j - 1] то

разница = 0

иначе

разница = 1

конец если

вставка = Матрица[i - 1, j] + 1

удаление = Матрица[i, j - 1] + 1

М = Матрица[i - 1, j - 1] + разница

Матрица[i, j] = наименьшее из (вставка, удаление, М)

конец

конец

конец

конец

Псевдокод для итеративной реализации алгоритма Дамерау-Левенштейна

функция ДамерауЛевенштейнаИтеративно; возвращает целое

начало

для i от 0 до длины строки 1; шаг = 1

для j от 0 до длины строки 2; шаг = 1

если i равно 0 и j равно 0 то

матрица[i,j] = 0

иначе если только i или только j равны нулю то

матрица[i,j] = отличное от нуля (i, j)

иначе если i > 1 и j > 1 и

строка1[i - 1] равно строка2[j - 2] и

строка1[i - 2] равно строка2[j - 1]

если строка1[i - 1] равно строка2[j - 1] то

разница = 0

иначе

разница = 1

конец если

вставка = Матрица[i - 1, j] + 1

удаление = Матрица[i, j - 1] + 1

М = Матрица[i - 1, j - 1] + разница

перемещение = Матрица[i - 2, j - 2] + 1

Матрица[i, j] = наименьшее из (вставка, удаление, М, перемещение)

иначе

если строка1[i - 1] равно строка2[j - 1] то

разница = 0

иначе

разница = 1

конец если

вставка = Матрица[i - 1, j] + 1

удаление = Матрица[i, j - 1] + 1

М = Матрица[i - 1, j - 1] + разница

Матрица[i, j] = наименьшее из (вставка, удаление, М)

конец

конец

конец

конец

## 2.2 Сравнительный анализ рекурсивной и нерекурсивной реализаций

По псевдокодам алгоритмов видно, что рекурсивный метод не требует выделения памяти под хранение матрицы, однако почти каждая ветка дерева рекурсии будет иметь по три ветки. Также видно, что в рекурсивной реализации будет происходить пересчет одной и той же подстроки несколько раз. Отсюда следует, что рекурсивная реализация будет работать значительно медленнее нерекурсивной.

# 3. Технологическая часть

В этой части обосновывается выбор языка программирования, необходимых библиотек и будет приведен листинг кодов каждого из алгоритмов.

## 3.1 Требования к программному обеспечению

Требуется вводить две строки и выводить матрицу и значения расстояний, полученных различными реализациями. Допускается предоставление двух режимов ПО: для единичного эксперимента и для массовых экспериментов.

## 3.2 Средства реализации

В качестве языка программирования был выбран C#. Его выбор основан на следующих причинах:

1. язык программирования C# является частью платформы .NET Framework, которая хорошо документирована, обладает хорошей стабильностью работы, и имеет богатый набор библиотечных решений;
2. C# позволяет программисту уделять больше времени написанию кода и не тратить время на побочные действия;
3. C# просто отлаживать, он не обладает уязвимостями языка C++, например, проблемами с указателями;
4. C# по скорости работы почти не уступает языку C++.

Время выполнения замеряется средством .NET под названием StopWatch;

## 3.3 Листинг кода

Рекурсивная реализация алгоритма расстояния Левеншейна:

private int D(int i, int j)

{

if (i == 0 && j == 0)

{

\_matrix[i, j] = 0;

return 0;

}

if (j == 0 && i > 0)

{

\_matrix[i, j] = i;

return i;

}

if (i == 0 && j > 0)

{

\_matrix[i, j] = j;

return j;

}

int I = D(i, j - 1) + 1;

int R = D(i - 1, j) + 1;

int M = D(i - 1, j - 1) + Match(\_wordA[i - 1], \_wordB[j - 1]);

int min = Min(I, R, M);

\_matrix[i, j] = min;

return min;

}

Итеративная реализация расстояния Левенштейна:

private void Calculate()

{

int s1Len = \_first.Length;

int s2Len = \_second.Length;

for (int i = 0; i <= s1Len; i++)

{

for (int j = 0; j <= s2Len; j++)

{

if (i == 0 && j == 0)

{

\_matrix[i, j] = 0;

}

else if (j == 0 && i > 0)

{

\_matrix[i, j] = i;

}

else if (i == 0 && j > 0)

{

\_matrix[i, j] = j;

}

else

{

var diff = 0;

if (\_first[i - 1] == \_second[j - 1])

diff = 0;

else

diff = 1;

int insertion = \_matrix[i - 1, j] + 1;

int deletion = \_matrix[i, j - 1] + 1;

int substitution = \_matrix[i - 1, j - 1] + diff;

\_matrix[i, j] = Min(insertion, deletion, substitution);

}

}

}

\_result = \_matrix[s1Len, s2Len];

}

Итеративная реализация расстояния Дамерау-Левеншейна:

private void Calculate()

{

int s1Len = \_first.Length;

int s2Len = \_second.Length;

for (int i = 0; i <= s1Len; i++)

{

for (int j = 0; j <= s2Len; j++)

{

if (i == 0 && j == 0)

{

\_matrix[i, j] = 0;

}

else if (j == 0 && i > 0)

{

\_matrix[i, j] = i;

}

else if (i == 0 && j > 0)

{

\_matrix[i, j] = j;

}

else if(i > 1 && j > 1 && \_first[i - 1] == \_second[j - 2] && \_first[i - 2] == \_second[j - 1])

{

var diff = 0;

if (\_first[i - 1] == \_second[j - 1])

diff = 0;

else

diff = 1;

int insertion = \_matrix[i - 1, j] + 1;

int deletion = \_matrix[i, j - 1] + 1;

int substitution = \_matrix[i - 1, j - 1] + diff;

int replacement = \_matrix[i - 2, j - 2] + 1;

\_matrix[i, j] = Min(insertion, deletion, substitution, replacement);

}

else

{

var diff = 0;

if (\_first[i - 1] == \_second[j - 1])

diff = 0;

else

diff = 1;

int insertion = \_matrix[i - 1, j] + 1;

int deletion = \_matrix[i, j - 1] + 1;

int substitution = \_matrix[i - 1, j - 1] + diff;

\_matrix[i, j] = Min(insertion, deletion, substitution);

}

}

}

\_result = \_matrix[s1Len, s2Len];

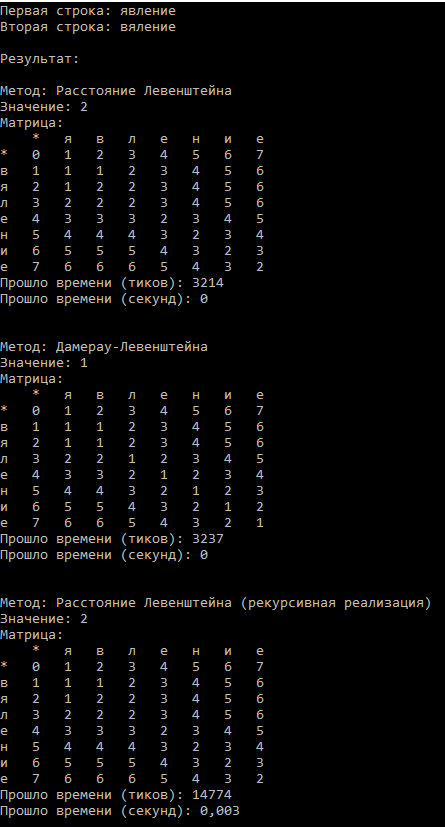
}

# 4. Экспериментальная часть

В этой части лабораторной работы будут поставлены эксперименты над реализованными алгоритмами и сделаны выводы.

## 4.1 Примеры работы

Пример работы программы для строк “явление” и “вяление”



## 4.2 Постановка эксперимента

Проведем эксперимент, где будем сравнивать две строки с одинаковой длиной, замеряя время работы каждого из алгоритмов. Начнем со строк нулевой длины, далее будем увеличивать длину строк на единицу до девяти включительно. По полученным результатами можно построить график, чтобы понять, у какой из трёх рассмотренных реализаций алгоритмов наилучшая скорость работы.

На Рисунке 1 виден экспоненциальный рост рекурсивной реализации.

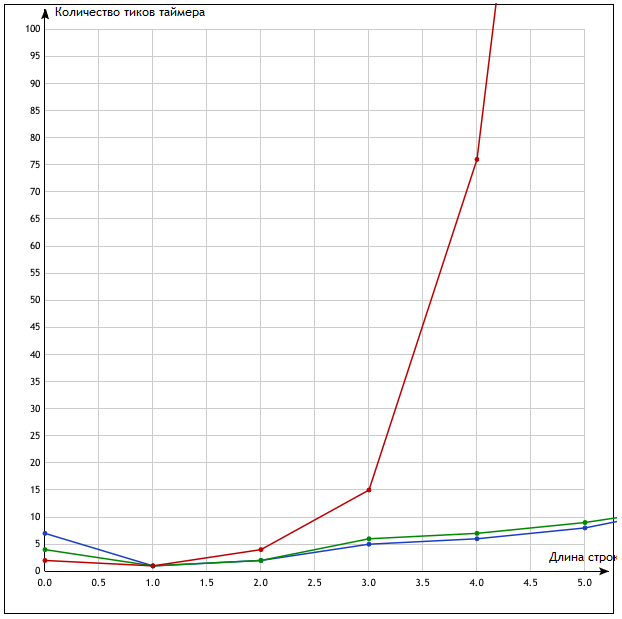


Рисунок 1. В масштабе по оси абсцисс от нуля до пяти.

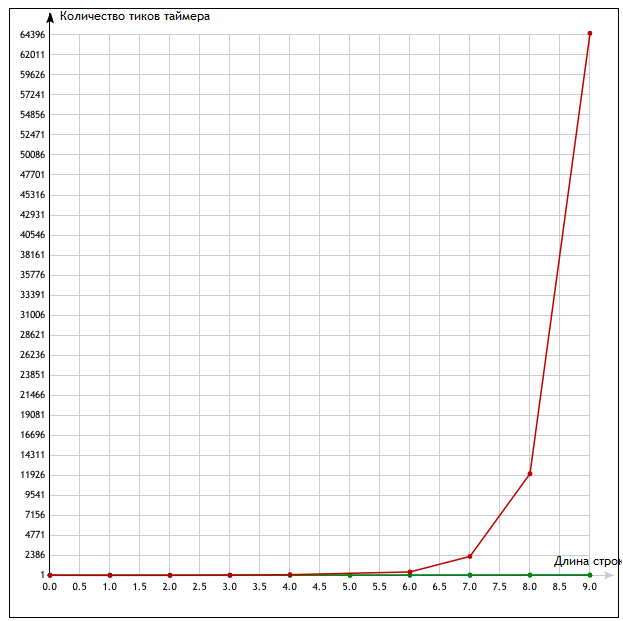


Рисунок 2. В масштабе по оси абсцисс от нуля до девяти.

Синий – итеративная реализация алгоритма Левенштейна.

Зелёный – итеративная реализация алгоритма Дамерау-Левенштейна.

Красный – рекурсивная реализация алгоритма Левенштейна.

Дерево вызовов рекурсивной реализации алгоритма Левенштейна

Для слов «Дверь» и «Двор» показана часть дерева рексурсии. Это ярко показывает, что даже на словах небольшой длины ветвистость дерева довольно высока.

4.3. Сравнительный анализ на материале экспериментальных данных

Рисунок 3. Часть дерева рекусрсии

D(4,0)

D(3,1)

D(4,1)

D(5,0)

D(5,1)

D(5,2)

D(5,3)

D(5,4)

В результате эксперимента видно, что итеративные реализации алгоритмов Левенштейна и Дамерау-Левенштейна работают за линейное время, а время работа рекурсивной реализации растет экспоненциально.

# Заключение

Реализован алгоритм Левенштейна, и его расширение, Алгоритм Дамерау-Левенштейна, позволяющие решать множество прикладных задач. В них входит исправление ошибок в словах, сравнения массивов данных и т.д. Изучено поведение рекурсивных реализаций при увеличении глубины рекурсии и увеличении количества веток. Показано, что рекурсивные функции с большим количеством веток (в данной работе 3 ветви) имеют экспоненциальные показатель времени выполнения, а, следовательно, их необходимо замещать на итерационные варианты.