Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №1 по курсу "Анализ алгоритмов"

Тема Расстояния Левенштейна и Дамерау-Левенштейна

Студент Якуба Д. В.

Группа ИУ7-53Б

Оценка (баллы) ____________

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

Оглавление

Введение

Цели лабораторной работы

- Изучение алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна;
- Реализация алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна;
- Применение методов динамического программирования для реализации алгоритмов;
- Проведение сравнительного анализа алгоритмов на основе полученных данных;

Определение

Расстояние Левенштейна - это мера, определяющая различие двух последовательностей символов. По неформальному определению расстояние Левенштейна между двумя словами - это мимнимальное количество односимвольных изменений (вставок, удалений или замен), необходимых для преобразования одного слова в другое.

Расстояние Левенштейна применяется в теории информации и компьютерной лингвистике для:

- Автоматического исправления ошибок в слове;
- Сравнение введёных слов со словарями в поисковых запросах;
- Сравнения текстовых файлов утилитой diff;

- В биоинформатике для сравнения генов, хромосом и белков;

Расстояние Дамерау-Левенштейна - это также мера, определяющая различие двух последовательностей символов, однако набор доступных операций для преобразований строк расширяется - добавляется операция транспозиции (перестановка двух соседствующих символов).

1 Аналитическая часть

Расстояние Левенштейна - это минимальное ннобходимое количество редакторских операций (вставки, замены, удаления) для преобразования последовательности символов (строки) в другую.

При нахождении расстояния Дамерау — Левенштейна добавляется операция транспозиции (перестановки соседствующих символов).

Обозначение редакторских операций:

1. I - вставка символа;
2. R - замена символа;
3. D - удаление символа;
4. M - бездействие (применяется при совпадении символов);

При этом для каждой операции задаётся своя цена (или штраф). Для решения задачи необходимо найти последовательность операций, минимизирующую суммарную цену всех проведённых операций. При этом следует отметить, что:

1. $price(x, x) = 0$ - цена замены символа x самого на себя;
2. $price(x, y) = 1$ $(x \neq y)$ - цена замены символа x на символ y;
3. $price(\alpha, x) = 1$ - цена вставки символа x;
4. $price(x, \alpha) = 1$ - цена удаления символа x.

1.1 Рекурсивный алгоритм нахождения расстояния Левенштейна

Пусть s_1 и s_2 — две некоторые строки. Тогда расстояние Левенштейна может быть вычислено по формуле **??**:

$$D(i,j) = \begin{cases} 0 & i = 0, j = 0 \\ i & j = 0, i > 0 \\ j & i = 0, j > 0 \\ min(& \\ D(i, j-1) + 1 & \\ D(i-1, j) + 1 & j > 0, i > 0 \\ D(i-1, j-1) + m(s_1[i], s_2[j]) & \\ ) & \end{cases} \tag{1.1}$$

где функция $m(x, y)$ (x и y - символы) равна нулю, если $a = b$, и единице в противном случае; $x[i]$ - это i-ый символ строки x.

При этом очевидны следующие факты:

1. $D(s_1, s_2) \geq ||\mathrm{s}_1| - |s_2||$
2. $D(s_1, s_2) \leq \max(|\mathrm{s}_1|, |s_2|)$
3. $D(s_1, s_2) = 0 \Leftrightarrow \mathrm{s}_1 = s_2$

Суть рекурсивного алгоритма заключается в реализации формулы **??**.

1.2 Рекурсивный алгоритм нахождения расстояния Левенштейна с использованием матрицы

Для оптимизации рекурсивного алгоритма нахождения расстояния Левенштейна допустимо добавить матрицу для хранения значений $D(i, j)$ для того, чтобы не вычислять их заново раз за разом. Таким образом, при обработке ещё не затронутых данных, результат нахождения расстояния будет занесён в так называемую матрицу расстояний. В ином случае, если для рассматриваемого случая информация о расстоянии уже имеется в матрице, алгоритм будет переходить к следующему шагу.

1.3 Итеративный алгоритм нахождения расстояния Левенштейна с использованием матрицы

Данный алгоритм также заключается в решении задачи с использованием матрицы расстояний. От уже рассмотренного рекурсивного алгоритма нахождения расстояния Левенштейна с использованием матрицы итеративный алгоритм отличается построчным заполнением матрицы последовательно вычисляемыми $D(i,j)$.

1.4 Алгоритм нахождения расстояния Дамерау-Левенштейна с использованием матрицы

Расстояние Дамерау-Левенштейна вычисляется по следующей рекуррентной формуле:

$$D(i,j) = \begin{cases} 0, & i=0, j=0 \\ i, & j=0, i>0 \\ j, & i=0, j>0 \\ min(& \\ D(i,j-1)+1, & \\ D(i-1,j)+1, & j>0, i>0 \\ D(i-1,j-1)+m(s_1[i],s_2[j]) & \\ D(i-2,j-2)+1, & i,j>1,\ s_1[i]=s_2[j-1], s_1[i-1]=s_2[j] \\ ) & \end{cases} \tag{1.2}$$

Применение данной формулы для реализации рекурсивного алгоритма при больших значениях i, j будет работать достаточно долго по тем же причинам, что и реализация рекурсивного алгоритма поиска расстояния Левенштейна. Посему целесообразно вновь ввести матрицу, в которой будут храниться вычисленные по формуле значения.

Вывод

Для каждого рассмотренного алгоритма имеется некоторая рекуррентная формула, что даёт возможность изучить как рекурсивные, так и итеративные реализации алгоритмов. Для оптимизации рекурсивных алгоритмов в рассмотрение вводится матрица, в которую записываются все промежуточные вычисленные значения. Эта же матрица применяется и при реализации итеративных алгоритмов.

2 Конструкторская часть

2.1 Блок-схема рекурсивного алгоритма Левенштейна

Блок-схема рекурсивного алгоритма поиска расстояния Левенштейна предоставлена на рисунке ??.

2.2 Блок-схема рекурсивного алгоритма Левенштейна с использованием матрицы

Блок-схема рекурсивного алгоритма поиска расстояния Левенштейна с использованием матрицы расстояний предоставлена на рисунке ??.

2.3 Блок-схема итеративного алгоритма Левенштейна

Блок-схема итеративного алгоритма поиска расстояния Левенштейна с использованием матрицы расстояний предоставлена на рисунке ??.

2.4 Блок-схема алгоритма Дамерау-Левенштейна

Блок-схема алгоритма поиска расстояния Дамерау-Левенштейна предоставлена на рисунке ??.

Вывод

С помощью информации, предоставленной в аналитическом разделе, были построены блок-схемы, описывающие работу рассматриваемых алгоритмов.

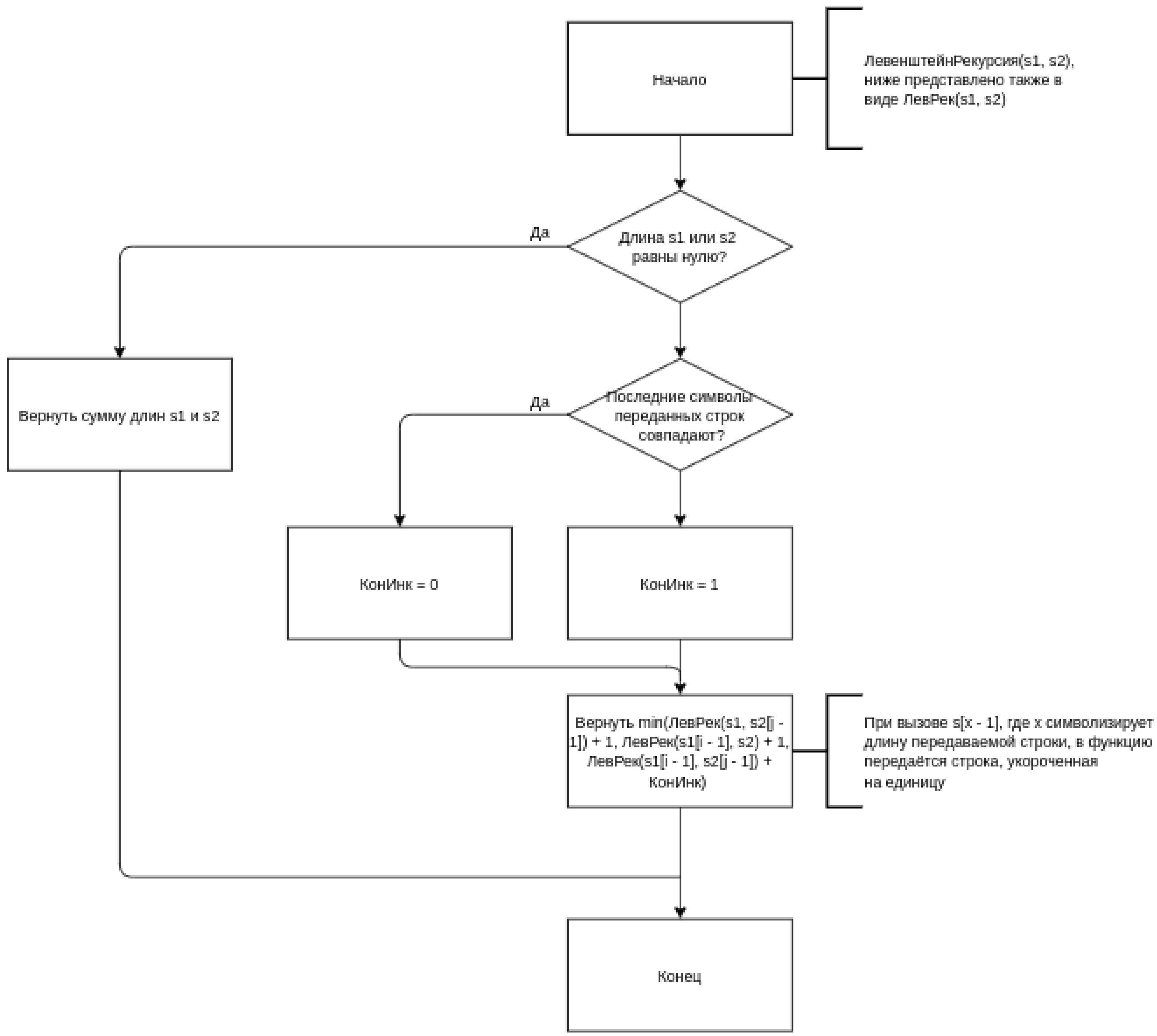


Рис. 2.1: Блок-схема, описывающая работу рекурсивного алгоритма поиска расстояния Левенштейна.

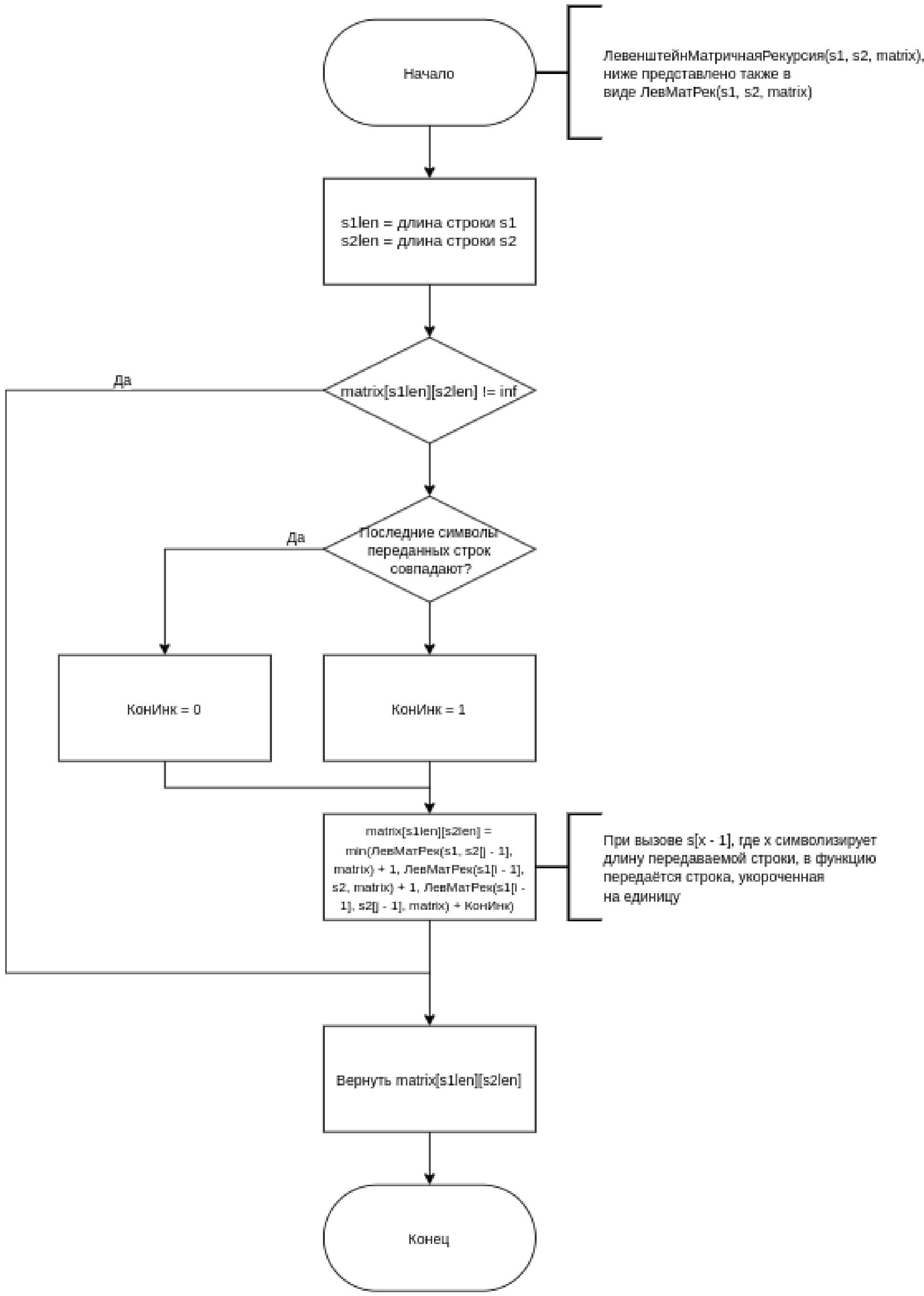


Рис. 2.2: Блок-схема, описывающая работу рекурсивного алгоритма поиска расстояния Левенштейна с использованием матрицы расстояний.

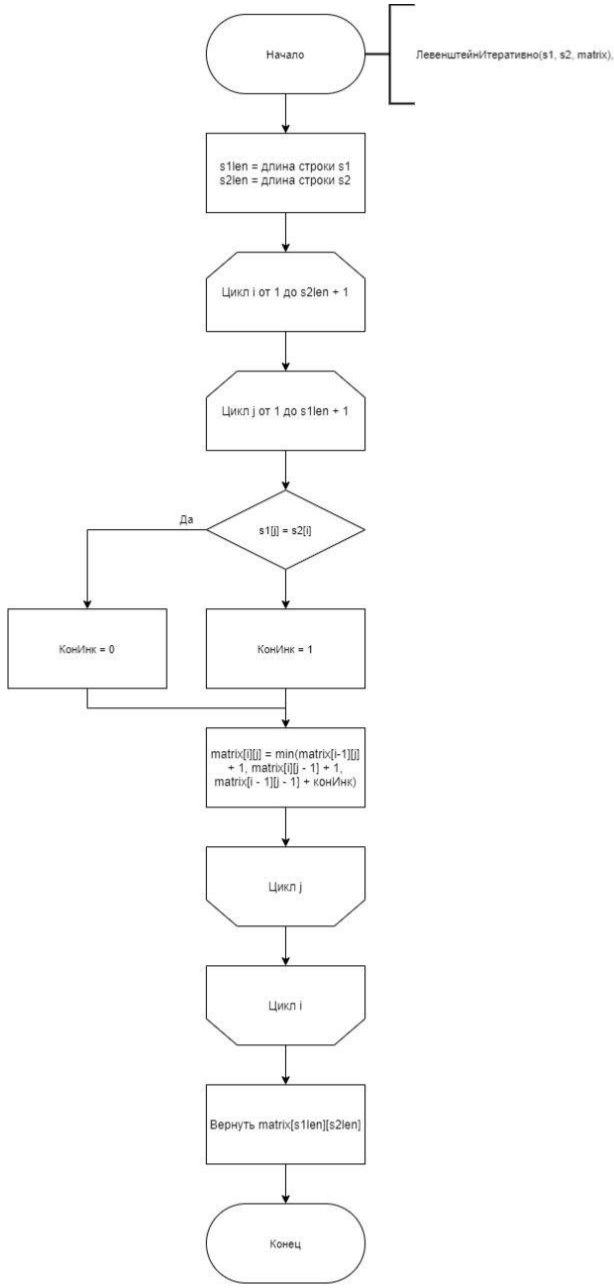
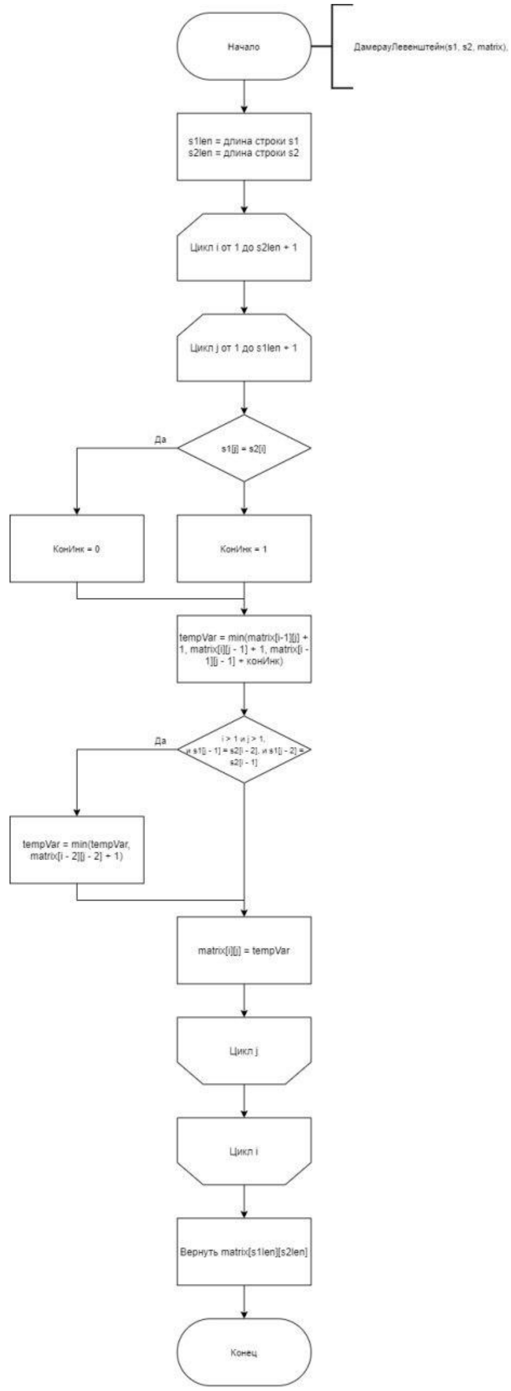


Рис. 2.3: Блок-схема, описывающая работу итеративного алгоритма поиска расстояния Левенштейна с использованием матрицы расстояний.



...

Рис. 2.4: Блок-схема, описывающая работу алгоритма поиска расстояния Дамерау-Левенштейна.

3 Технологическая часть

3.1 Требования к программному обеспечению

- Входные данные - строки s_1 и s_2;
- Выходные данные - искомое расстояние для выбранного метода и матрица расстояний (при её наличии).

3.2 Средства реализации программного обеспечения

При написании программного продукта был использован язык программирования C++.

Данный выбор обусловлен следующими факторами:

- Данный язык программирования преподавался в рамках курса объектно-ориентированного программирования;
- Высокая вычислительная производительность;
- Большое количество справочной литературы.

Для тестирования производительности реализаций алгоритмов использовалась утилита QueryPerfomanceCounter, объявленная в заголовочном файле "windows.h".

При написаннии программного продукта использовалась среда разработки QT Creator.

Данный выбор обусловлен следующими факторами:

- Основы работы с данной средой разработки преподавался в рамках курса программирования на Си;
- QT Creator позволяет работать с расширением QtDesign, которое позволяет создать удобный интерфейс для программного продукта в сжатые сроки.

3.3 Листинг кода

В листингах **??** - **??** предоставлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Функция реализации рекурсивного алгоритма Левенштейна

```
size_t MainWindow::levenshteinRecursive(QString fWord,
    QString sWord)
{
    if (!fWord.size() || !sWord.size())
        return fWord.size() + sWord.size();

    return std::min({levenshteinRecursive(fWord, sWord.mid
        (0, sWord.size() - 1)) + 1,
    levenshteinRecursive(fWord.mid(0, fWord.size() - 1),
        sWord) + 1,
    levenshteinRecursive(fWord.mid(0, fWord.size() - 1),
        sWord.mid(0, sWord.size() - 1)) +
    ((fWord.back() == sWord.back()) ? 0 : 1)});
}
```

Листинг 3.2: Функция реализации рекурсивного алгоритма Левенштейна с использованием матрицы расстояний

```
size_t MainWindow::levenshteinRecursiveMatrix(
QString fWord, QString sWord, std::vector<std::vector<int>>
    &matrix)
{
    if (matrix[sWord.size()][fWord.size()] != std::
        numeric_limits<int>().max())
        return matrix[sWord.size()][fWord.size()];
```

```
    matrix[sWord.size()][fWord.size()] =
    std::min({levenshteinRecursiveMatrix(fWord.mid(0, fWord
        .size() - 1), sWord, matrix) + 1,
    levenshteinRecursiveMatrix(fWord, sWord.mid(0, sWord.
        size() - 1), matrix) + 1,
    levenshteinRecursiveMatrix(
    fWord.mid(0, fWord.size() - 1), sWord.mid(0, sWord.size
        () - 1), matrix) +
    ((fWord.back() == sWord.back()) ? 0 : 1)});

    return matrix[sWord.size()][fWord.size()];
}
```

Листинг 3.3: Функция реализации итеративного алгоритма Левенштейна

```
size_t MainWindow::levenshteinNonRecursiveMatrix(
QString fWord, QString sWord, std::vector<std::vector<int>>
    &matrix)
{
    for (int i = 1; i <= sWord.size(); i++)
        for (int j = 1; j <= fWord.size(); j++)
            matrix[i][j] = std::min({matrix[i - 1][j] + 1,
                matrix[i][j - 1] + 1,
            matrix[i - 1][j - 1] +
            (((fWord.mid(0, j)).back() == sWord.mid(0, i).
                back()) ? 0 : 1)});

    return matrix[sWord.size()][fWord.size()];
}
```

Листинг 3.4: Функция реализации алгоритма Дамерау-Левенштейна

```
bool canBeTranspose(QString fStr, QString sStr, size_t i,
    size_t j)
{
    return fStr.at(j - 1) == sStr.at(i - 2) && fStr.at(j -
        2) == sStr.at(i - 1);
}

```

```
size_t MainWindow::damerauLev(
QString fWord, QString sWord, std::vector<std::vector<int>>
    &matrix)
{
    for (int i = 1; i <= sWord.size(); i++)
        for (int j = 1; j <= fWord.size(); j++)
        {
            int temp = std::min({matrix[i - 1][j] + 1,
               matrix[i][j - 1] + 1,
            matrix[i - 1][j - 1] +
            (((fWord.mid(0, j)).back() == sWord.mid(0, i).
               back()) ? 0 : 1)});
            if (i > 1 && j > 1 && canBeTranspose(fWord,
               sWord, i, j))
                temp = std::min(temp, matrix[i - 2][j - 2]
                   + 1);
            matrix[i][j] = temp;
        }
    return matrix[sWord.size()][fWord.size()];
}
```

3.4 Тестирование программного продукта

В таблице **??** приведены тестовые данные и вывод программы для алгоритмов вычисления расстояния Левенштейна и Дамерау-Левенштейна. Тесты пройдены успешно.

Таблица 3.1: Тесты

		Ожидаемый результат	
Строка 1	Строка 2	Алг. Левенштейна	Алг. Дамерау-Левенштейна
table	tumbler	3	3
hell	help	1	1
KillUsAll	KlilUsAll	2	1
smoke	mssql	5	4
OfMiceAndMen	OfMonstersAndMen	6	6
roofer	killer	4	4
orange	orangina	3	3
prolifer	profiler	2	2
cat	dog	3	3

Вывод

Спроектированные алгоритмы вычисления расстояния Левенштейна рекурсивно, рекурсивно с использованием матрицы расстояний, итеративно с использованием матрицы расстояний, а также алгоритм вычисления расстояния Дамерау-Левенштейна итеративно с использованием матрицы были реализованы и протестированы.

4 Исследовательская часть

4.1 Пример работы программного обеспечения

Ниже на рисунках ?? - ?? предоставлены csv-таблицы, сгенерированные по окончанию работы каждого из алгоритмов при передаче строк, описанных на рисунке ??.

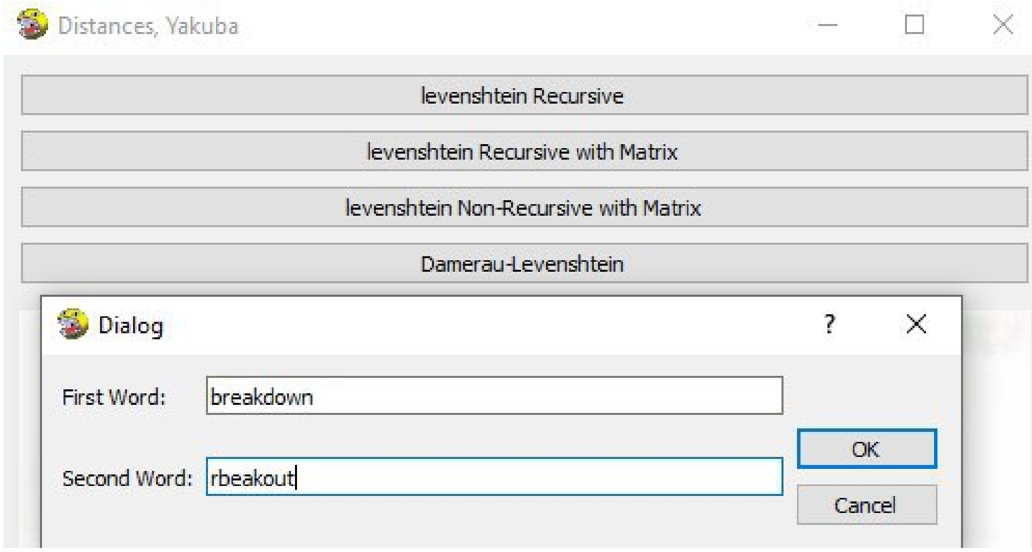


Рис. 4.1: Передача тестовых данных в ПО.

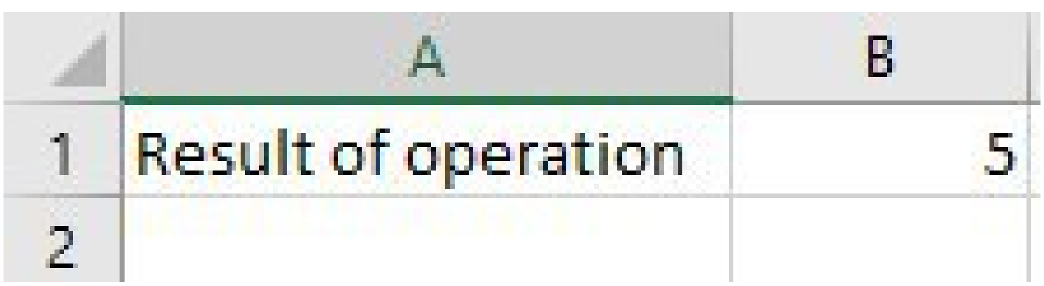


Рис. 4.2: Итоговая таблица для рекурсивной реализации поиска расстояния Левенштейна.

	A	B	C	D	E	F	G	H	I	J	K
1	Result of (	5									
2											
3	Result Matrix:										
4			b	r	e	a	k	d	o	w	n
5		0	1	2	3	4	5	6	7	8	9
6	r	1	1	1	2	3	4	5	6	7	8
7	b	2	1	2	2	3	4	5	6	7	8
8	e	3	2	2	2	3	4	5	6	7	8
9	a	4	3	3	3	2	3	4	5	6	7
10	k	5	4	4	4	3	2	3	4	5	6
11	o	6	5	5	5	4	3	3	3	4	5
12	u	7	6	6	6	5	4	4	4	4	5
13	t	8	7	7	7	6	5	5	5	5	5

Рис. 4.3: Итоговая таблица для рекурсивной реализации поиска расстояния Левенштейна с использованием матрицы расстояний.

	A	B	C	D	E	F	G	H	I	J	K
1	Result of (	5									
2											
3	Result Matrix:										
4			b	r	e	a	k	d	o	w	n
5		0	1	2	3	4	5	6	7	8	9
6	r	1	1	1	2	3	4	5	6	7	8
7	b	2	1	2	2	3	4	5	6	7	8
8	e	3	2	2	2	3	4	5	6	7	8
9	a	4	3	3	3	2	3	4	5	6	7
10	k	5	4	4	4	3	2	3	4	5	6
11	o	6	5	5	5	4	3	3	3	4	5
12	u	7	6	6	6	5	4	4	4	4	5
13	t	8	7	7	7	6	5	5	5	5	5

Рис. 4.4: Итоговая таблица для итеративной реализации поиска расстояния Левенштейна с использованием матрицы расстояний.

	A	B	C	D	E	F	G	H	I	J	K
1	Result of (	4									
2											
3	Result Matrix:										
4			b	r	e	a	k	d	o	w	n
5		0	1	2	3	4	5	6	7	8	9
6	r	1	1	1	2	3	4	5	6	7	8
7	b	2	1	1	2	3	4	5	6	7	8
8	e	3	2	2	1	2	3	4	5	6	7
9	a	4	3	3	2	1	2	3	4	5	6
10	k	5	4	4	3	2	1	2	3	4	5
11	o	6	5	5	4	3	2	2	2	3	4
12	u	7	6	6	5	4	3	3	3	3	4
13	t	8	7	7	6	5	4	4	4	4	4

Рис. 4.5: Итоговая таблица для реализации поиска расстяния Дамерау-Левенштейна.

4.2 Технические характеристики

Технические характеристики ЭВМ, на котором выполнялись исследования:

- ОС: Windows 10
- Оперативная память: 16 Гб
- Процессор: Intel Core i7-10510U

  При проведении замеров времени ноутбук был подключен к сети электропитания.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались на данных, сгенерированных случайным образом один раз. При увеличении количества символов во входных данных - в конец к уже сгенерированным строкам добавлялись новые сгенерированные символы.

Тестовые данные:

- 5 символов:
VxgtU (строка 1),
jRMFA (строка 2)

- 7 символов:
VxgtUsx (строка 1),
jRMFAyC (строка 2)

- 10 символов:
VxgtUsx2u3 (строка 1),
jRMFAyCfiV (строка 2)

- 20 символов:
VxgtUsx2u39dtX81sxy8 (строка 1),
jRMFAyCfiVxyhmILtGMG (строка 2)

- 30 символов:
VxgtUsx2u39dtX81sxy8GInrYeVNmJ (строка 1),
jRMFAyCfiVxyhmILtGMG4IVZTjPQ7l (строка 2)

- 50 символов:
VxgtUsx2u39dtX81sxy8GInrYeVNmJvvG7WkaA7Qjs82qP6bJG (строка 1),
jRMFAyCfiVxyhmILtGMG4IVZTjPQ7laMIEG6xv9zbdXq9WcJY2 (строка 2)

- 100 символов:
VxgtUsx2u39dtX81sxy8GInrYeVNmJvvG7WkaA7Qjs82qP6bJG
Ooryez5fYpJWcPRhm7TEjeUoD49M26XDt CJrGtjJXf3aZ9La9n (строка 1),
jRMFAyCfiVxyhmILtGMG4IVZTjPQ7laMIEG6xv9zbdXq9WcJY2
G4J0JV1XP8ecmHkTYdY1uzSm8WFY3KjgG ggAw3GrPISl76Mzb1 (строка 2)

- 200 символов:
VxgtUsx2u39dtX81sxy8GInrYeVNmJvvG7WkaA7Qjs82qP6bJGO
oryez5fYpJWcPRhm7TEjeUoD49M26XDtCJrGtjJXf3aZ9La9nsh
v3cAbwuAJuKc00ndp6EWNHQcArjwXQzAtdpnHs2uOF1kfhWjzXU
S44zKnHVNCaeLyzBlce3RCdGwbJx8s2SlfvYoyBZsKrN1cX (строка 1),
jRMFAyCfiVxyhmILtGMG4IVZTjPQ7laMIEG6xv9zbdXq9WcJY2G

4J0JV1XP8ecmHkTYdY1uzSm8WFY3KjgGggAw3GrPISl76Mzb1f3
ElDEyOeorQGS6CxLWS3lH8sNgZta9vSDMLvnbPaXP24H5dYkBXL
RruvzSlLs1T8hyezy0U3awz65ctATEclCBG4H1pC9mMusWF (строка 2)

Результаты замеров времени приведены в таблице **??**. Прочерк в таблице означает, что для заданных значений тестирование не проводилось. На рисунках **??**, **??** приведены графики зависимостей времени работы алгоритмов от длины строк, подаваемых на вход.

Таблица 4.1: Замеры времени для строк различной длины

Длина строк	LevRec	LevMatRec	LevMatIter	DamLev
5	10867	-	-	-
7	258961	-	-	-
10	33589820	3146	2001	2137
20	-	12896	4686	6251
30	-	29325	10744	13631
50	-	70918	29277	38427
100	-	184238	86268	118891
200	-	642895	248651	299743

4.4 Оценка затрат памяти

Максимальная глубина стека вызовов при исполнении рекурсивного алгоритма Левенштейна определяется выражением **??**:

$$(sizeof(s_1) + sizeof(s_2)) * (2 * sizeof(string) + sizeof(int)) \tag{4.1}$$

Здесь sizeof - оператор вычисления размера; s_1, s_2 - строки; *string* - строковый тип; *int* - целочисленный тип.

При исполнении интеративной реализации задействованная память будет определяться выражением **??**:

$$(sizeof(s_1+1)*(sizeof(s_2+1)*sizeof(int)+sizeof(int)+2*sizeof(string)) \tag{4.2}$$

Вывод

Чистый рекурсивный вариант реализации алгоритма Левенштейна работает дольше реализаций с задействованием матриц расстояний. Также следует отметить факт того, что рекурсивные реализации совместно отстают от итеративных реализаций. Легко заметить, что при величине

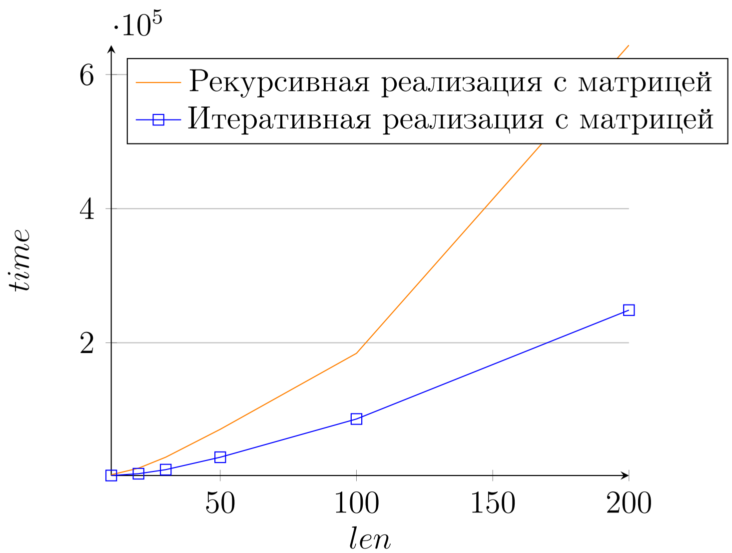


Рис. 4.6: Зависимость времени работы рекурсивных реализаций алгоритмов вычисления расстояния Левенштейна от длины строк

строк в 10 символов (единственное общее поле для всех четырёх реализаций), чистый рекурсивный алгоритм медленнее матричных реализаций примерно в 100600 раз. При этом уже на 20 символах можно заметить отставание по скорости исполнения матричного рекурсивного алгоритма - в приведённом случае он медленнее итеративной реализации алгоритма поиска расстояния Левенштейна примерно в 3 раза, и медленнее реализации алгоритма поиска расстояния Дамерау-Левенштейна примерно в 2 раза. С повышением длин строк отставание в скорости работы рекурсивного алгоритма становится всё более заметным.

Также легко заметить, что алгоритм Дамерау-Левенштейна работает несколько дольше алгоритма Левенштейна, но это объясняется внесением в реализацию новой операции, а, следовательно, и новой проверки, что и добавляет времени исполнения алгоритму. При этом стоит отметить, что алгоритмы Левенштейна и Дамерау-Левенштейна - это алгоритмы разного назначения и сравнение их быстродействия - некая формальность.

Несмотря на всё вышеизложенное рекурсивная реализация всё же имеет своё достоинство: пиковая используемая память у него ниже, чем

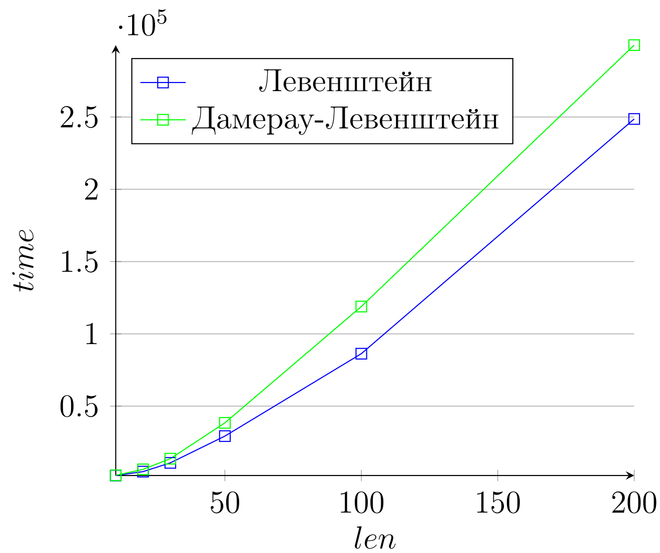


Рис. 4.7: Зависимость времени работы итеративных реализаций алгоритмов вычисления расстояния Левенштейна и Дамерау-Левенштейна от длины строк.

у итеративных реализаций, что позволяет использовать его в условиях дефицита памяти.

Заключение

В ходе выполнения лабораторной работы:

- Были изучены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна;
- Были реализованы алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна;
- Были изучены методы динамического программирования на основе алгоритмов вычисления расстояния Левенштейна и Дамерау-Левенштейна;
- Был проведён сравнительный анализ производительности конкретных реализаций алгоритмов, показавший динамику роста времени исполнения каждого из алгоритмов;
- Были рассчитаны пиковые затраты памяти для каждого из алгоритмов, что позволило выявить достоинство рекурсивной нематричной реализации алгоритма поиска расстояния Левенштейна;
- Были получены практические навыки реализации алгоритмов на ЯП C++.

Список источников

- Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений сиволов. - М.: Доклады АН СССР, 1965. Т.163. С.845-848.

- Qt Documentation [Электронный ресурс]. Режим доступа: `https://doc.qt.io/` (дата обращения: 11.09.2000).

- Документация Разработчика приложений для Windows [Электронный ресурс]. Режим доступа: `https://docs.microsoft.com/` (дата обращения: 11.09.2000).