



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 «Анализы Алгоритмов»

на тему: «Динамическое программирование»

Студент группы ИУ7-56Б

(Подпись, дата)

Разин А. В.
(Фамилия И.О.)

Преподаватель

(Подпись, дата)

Волкова Л. Л.
(Фамилия И.О.)

Преподаватель

(Подпись, дата)

Строганов Ю. В..
(Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Расстояние Левенштейна	4
2 Конструкторская часть	7
2.1 Реализации алгоритмов	7
2.1.1 Использование матрицы	7
2.1.2 Использование рекурсии	7
2.1.3 Использование рекурсии с мемоизацией	7
2.1.4 Схемы алгоритмов	8
2.1.5 Структуры данных	8
3 Технологическая часть	13
4 Исследовательская часть	14
4.1 Технические характеристики	14
4.2 Демонстрация работы программы	14
4.3 Временные характеристики	16
4.4 Характеристики по памяти	17
4.5 Вывод	23

Введение

Часто при работе со словами, необходимо их сравнивать, причем необходима конкретная метрика, которая покажет, насколько посимвольно одно слово отличается от другого. Одной из таких метрик является Расстояние Левенштейна. Данное расстояние является метрикой, измеряющей по модулю разность между двумя последовательностями символов.

Впервые задачу поставил в 1965 году советский математик Владимир Левенштейн при изучении последовательностей 0 — 1, впоследствии более общую задачу для произвольного алфавита связали с его именем.

Расстояние Левенштейна активно используется и по сей день:

- исправление ошибок в слове(в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи);
- сравнение текстовых файлов утилитой diff;
- для сравнения геномов, хромосом и белков в биоинформатике.

Данная метрика была модифицирована Фредриком Дамерау, путем введения операции перестановки соседних символов.

1 Аналитическая часть

Целью данной лабораторной работы является описание и исследование алгоритмов поиска расстояний Левенштейна и Дameraу-Левенштейна.

Для поставленной цели необходимо выполнить следующие задачи.

- 1) Исследовать расстояние Левенштейна.
- 2) Разработка алгоритмов поиска расстояний Левенштейна, Дameraу-Левенштейна.
- 3) Создать программное обеспечение, реализующее следующие алгоритмы.
- 4) Провести исследование, затрачиваемого процессорного времени и памяти при различных реализациях алгоритмов.
- 5) Провести сравнительный анализ алгоритмов.

1.1 Расстояние Левенштейна

Расстояние Левенштейна [**levenshtein**] (редакционное расстояние, дистанция редактирования) — метрика, измеряющая разность между двумя последовательностями символов.

Вводятся операции нескольких типов:

- 1 I (англ. insert) — операция вставки символа.
- 2 D (англ. delete) — операция удаления символа.
- 3 R (англ. replace) — операция замены символа.

Также введем символ λ , обозначающий пустой символ строки, не входящий ни в одну из рассматриваемых строк.

Будем считать стоимость каждой вышеизложенной операции равной 1:

— $D(a, b) = 1$, $a \neq b$, в противном случае замена не происходит;

- $D(\lambda, b) = 1;$
- $D(a, \lambda) = 1.$

Также обозначим совпадение символов как M (англ. match), таким образом $D(a, a) = 0$. Существует проблема взаимного выравнивания строк, в случае когда строки различной длины существует множество способов сопоставить соответствующие символы.

Решим проблему введением рекуррентной формулы, обозначим:

1. $L1$ — длина S_1 .
2. $L2$ — длина S_2 .
3. $S_1[1...i]$ — подстрока S_1 длиной i , начиная с первого символа.
4. $S_2[1...j]$ — подстрока S_2 длиной j , начиная с первого символа.

Расстояние Левенштейна между двумя строками S_1 и S_2 длиной M и N соответственно рассчитывается по рекуррентной формуле:

$$D(i, j) = \begin{cases} j, & \text{если } i = 0 \\ i, & \text{если } j = 0, i > 0 \\ \min(D(S_1[1..i], S_2[1..j-1]) + 1, & \\ D(S_1[1..i-1], S_2[1..j]) + 1, & \text{если } i > 0, j > 0 \\ D(S_1[1..i-1], S_2[1..j-1]) + m(S_1[i], S_2[j]), & \end{cases}, \quad (1.1)$$

где сравнение символов строк S_1 и S_2 рассчитывается как:

$$m(a, b) = \begin{cases} 0 & \text{если } a = b, \\ 1 & \text{иначе.} \end{cases} \quad (1.2)$$

В расстоянии Дамерау-Левенштейна вводится еще одна операция, обозначим ее как S (англ. swap) — данная операция применима, только если $S_1[i] = S_2[j-1]$ и $S_1[i-1] = S_2[j]$. Рекуррентная формула данной

метрики выглядит следующим образом:

$$D(S1[1 \dots i], S2[1 \dots j]) = \begin{cases} j, & \text{если } i = 0 \\ i, & \text{если } j = 0, i > 1 \\ \min(D(S1[1..i], S2[1..j-1]) + 1, \\ D(S1[1..i-1], S2[1..j]) + 1, \\ D(S1[1..i-1], S2[1..j-1]) + \\ + \begin{cases} 0, & \text{если } S1[i] == S2[j], \\ 1, & \text{иначе} \end{cases} \\ \begin{cases} D(S1[1..i-2], S2[1..j-2]) + 1, & \text{если } i > 1, j > 1, \\ S1[i] == S2[j-1], \\ S1[i-1] == S2[j] \\ +\infty, & \text{иначе} \end{cases} \\ \min(D(S1[1..i], S2[1..j-1]) + 1, \\ D(S1[1..i-1], S2[1..j]) + 1, \\ D(S1[1..i-1], S2[1..j-1]) + \\ + \begin{cases} 0, & \text{если } S1[i] == S2[j], \\ 1, & \text{иначе} \end{cases}), & \text{если } i > 0, j > 0 \end{cases} \quad (1.3)$$

Идея в том, что после замены пары символов местами, полученные пары были поэлементно равны друг другу.

Вывод:

В данном разделе были рассмотрены цели и задачи работы, а также введены необходимые обозначения для поиска расстояний Левенштейна и Дамерау-Левенштейна и выведены рекуррентные отношения для поиска их значения.

2 Конструкторская часть

Рассмотрим различные реализации алгоритмов поиска расстояния Левенштейна и Дameraу-Левенштейна для строк S_1, S_2 , каждая имеет длину NM соответственно.

2.1 Реализации алгоритмов

2.1.1 Использование матрицы

Вводится матрица M с $N + 1$ на $M + 1$ элементами. Номер строки матрицы i соответствует длине подстроки S_1 , номер столбца j — длине подстроки S_2 . Значение $M[i][j]$ соответствует расстоянию Левенштейна между подстроками, $S_1[i]$ и $S_2[j]$ соответственно. После чего используется формула 1.3 или ?? для расчета значения в данной клетке, таким образом для получения значения, необходимо рассмотреть диагональную, верхнюю и нижнюю клетки (в ?? также рассматривается клетка $M[i - 2][j - 2]$).

2.1.2 Использование рекурсии

При использовании рекуррентной формулы возможно использование рекурсии. На вход подаются 2 строки, а также их длины на данный момент, получив строки с длинами i и j , необходимо рассмотреть те же строки с длинами $(i - 1, j - 1), (i, j - 1), (i - 1, j)$ и рассчитать их стоимость с помощью формул 1.3 и ?? (в ?? также рассматривается строка $(i - 2, j - 2)$). После чего необходимо выбрать вариант с наименьшей стоимостью из возможных.

2.1.3 Использование рекурсии с мемоизацией

Заметим, что метод, описанный в пункте 2.1.2, можно ускорить, если запоминать уже посчитанные значения, и использовать их в дальнейшем. Таким образом вводится матрица M , аналогичная матрице из пункта 2.1.1,

все ячейки которой заполняются $+\infty$, после чего перед вычислением текущего значения происходит проверка: в случае, если значение $M[i][j] = +\infty$, вычисляется текущее значение и заносится в матрицу, иначе вычисления значения не происходит и значение сразу берется из ячейки $M[i][j]$.

2.1.4 Схемы алгоритмов

2.1.5 Структуры данных

Для реализации выбранных алгоритмов были использованы следующие структуры данных:

1. Матрица — массив векторов типа `int`.
2. Строка — массив типа `wchar`.
3. Длина строки — целое значение типа `size_t`.

Вывод

На данном этапе были рассмотрены алгоритмы и записаны их схемы, также были описаны используемые структуры данных.

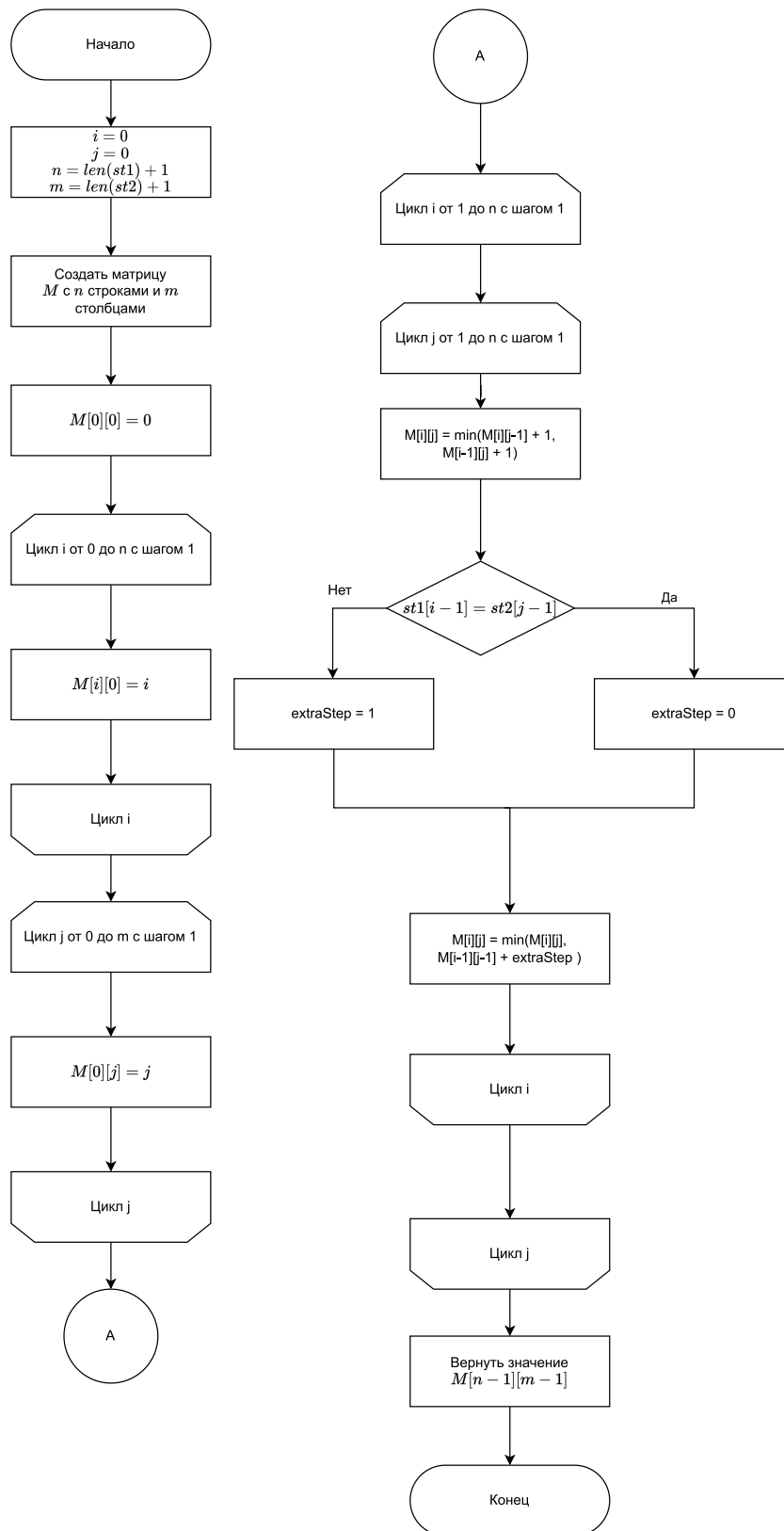


Рисунок 2.1 – Схема алгоритма поиска расстояния Левенштейна с помощью матрицы

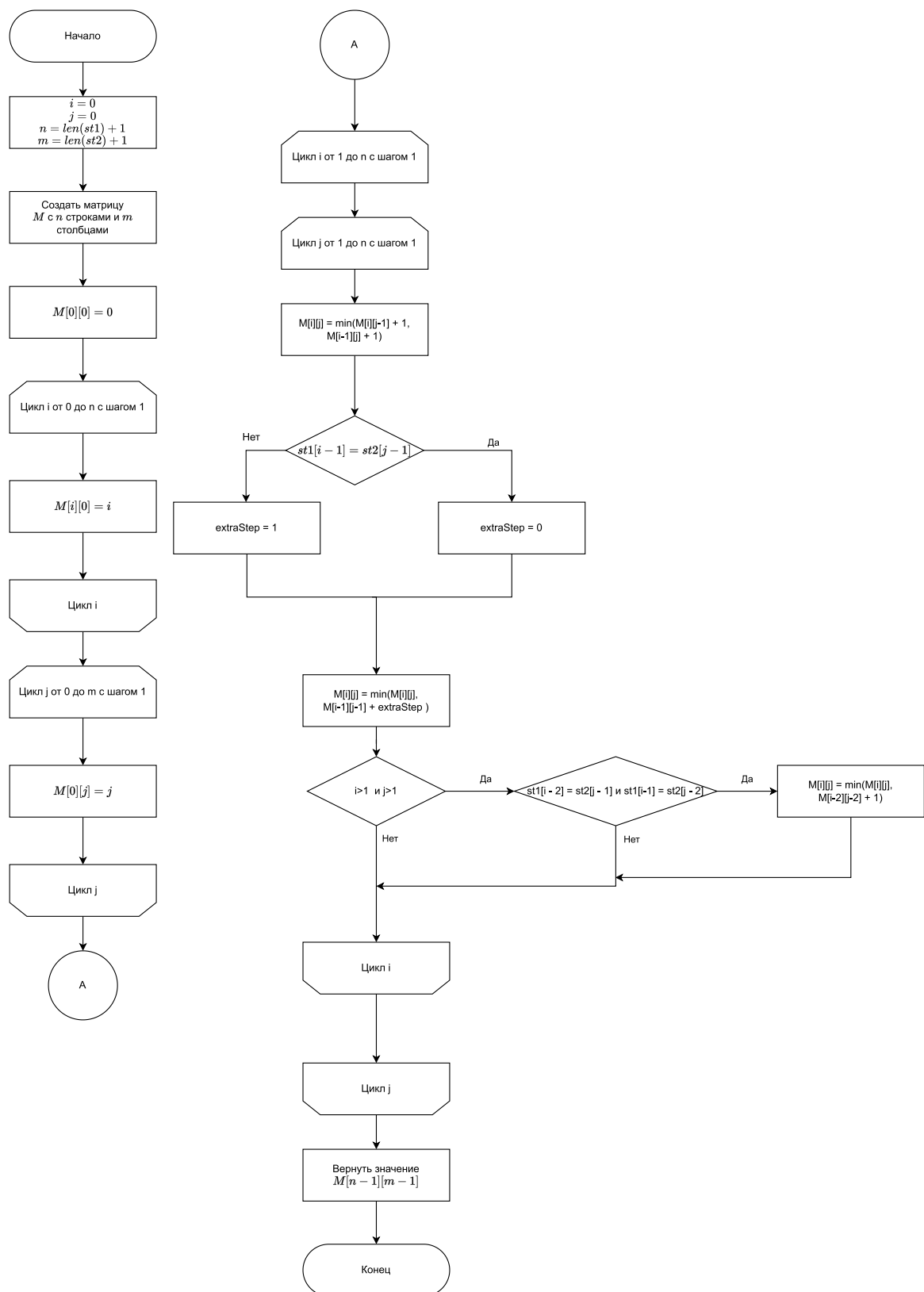


Рисунок 2.2 – Схема алгоритма поиска расстояния Дамерау-Левенштейна с помощью матрицы

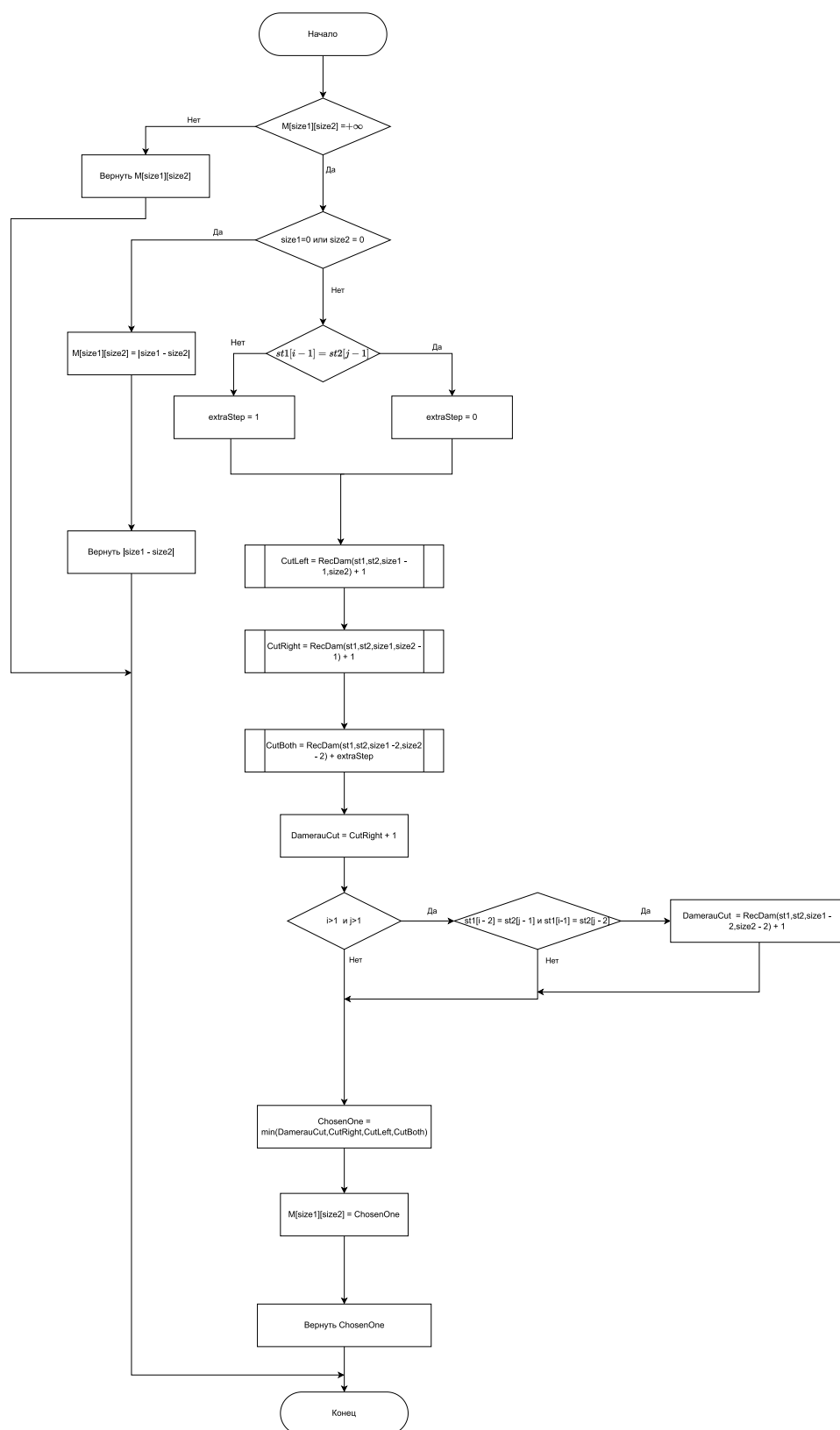


Рисунок 2.3 – Схема алгоритма поиска расстояния Дameraу-Левенштейна с помощью рекурсии с мемоизацией

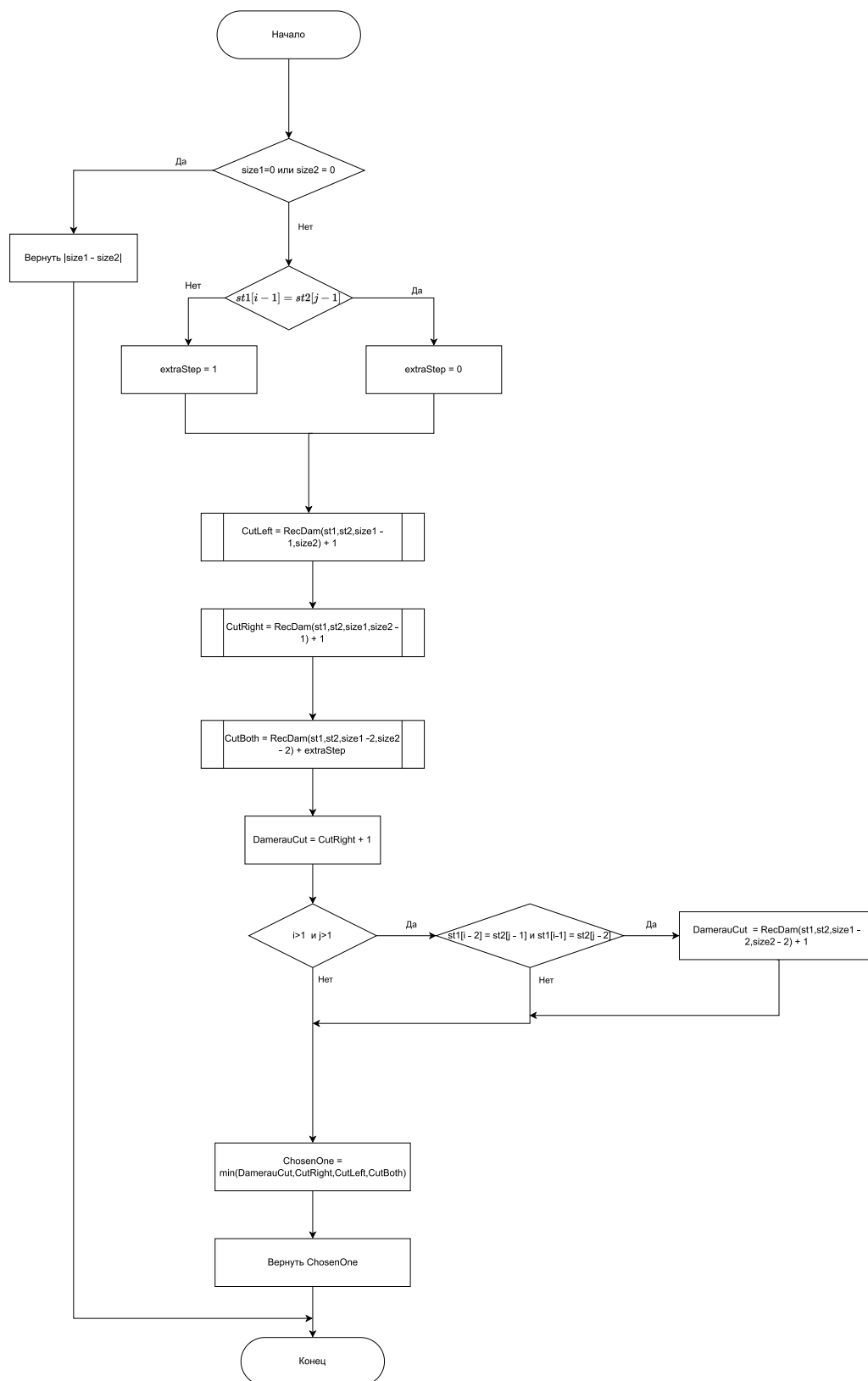


Рисунок 2.4 – Схема алгоритма поиска расстояния Дameraу-Левенштейна с помощью рекурсии

3 Технологическая часть

В данном разделе

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

- Процессор: Intel(R) Core(TM) i5-10300H CPU 2.50 ГГц [intel].
- Оперативная память: 16 ГБайт.
- Операционная система: Windows 10 Pro 64-разрядная система версии 21H2 [windows].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного программного обеспечения, а именно показаны результаты вычислений реализаций алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна на примере двух строк «друзья» и «рдузия». При этом выводятся матрицы для алгоритмов, использующих матрицы для промежуточных результатов.

img/example.png

Рисунок 4.1 – Демонстрация работы программы при поиске расстояний
Левенштейна и Дамерау-Левенштейна

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблице 4.1, в которой есть поля, обозначенные «-». Это обусловлено тем, что для рекурсивной реализации алгоритмов достаточно приведенных замеров для построения графика. По полученным замерам по времени для рекурсивной реализации понятно, что проведения замеров на длин строк больше 15 будет достаточно долгим, поэтому нет смысла проводить замеры по времени рекурсивных реализаций алгоритмов поиска расстояния.

Замеры проводились на одинаковых длин строк от 1 до 200 с различным шагом.

Таблица 4.1 – Замер по времени для строк, размер которых от 1 до 200

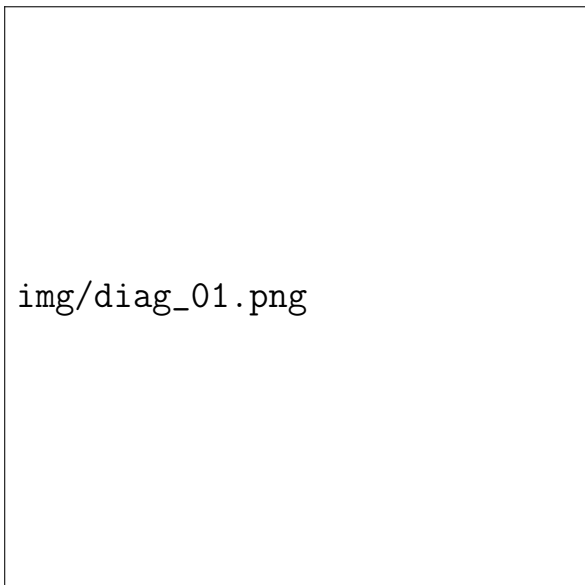
Длина (символ)	Время, нс			
	Левенштейн	Дамерау-Левенштейн		
	Итеративный	Итеративный	Рекурсивный	
			Без кеша	С кешом

Отдельно сравниваются итеративные алгоритмы поиска расстояний Левенштейна и Дамерау-Левенштейна. Сравнение будет производится на основе данных, представленных в таблице 4.1. Результат можно рассмотреть на рисунке 4.2.

При длинах строк менее 30 символов разница по времени между итеративными реализациями незначительна, однако при увеличении длины строки алгоритм поиска расстояния Левенштейна оказывается быстрее вплоть до полутора раз (при длинах строк равных 200). Это обосновывается тем, что у алгоритма поиска расстояния Дамерау-Левенштейна задействуется дополнительная операция, которая замедляет алгоритм.

Также сравним рекурсивную и итеративную реализации алгоритма поиска расстояния Дамерау-Левенштейна. Данные представлены в таблице 4.1 и отображены на рисунке 4.3.

На рисунке 4.3 продемонстрировано, что рекурсивный алгоритм становится менее эффективным по времени (вплоть до 21 раз при длине строк равной 7 элементов), чем итеративный.



img/diag_01.png

Рисунок 4.2 – Сравнение по времени нерекурсивных реализаций алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна

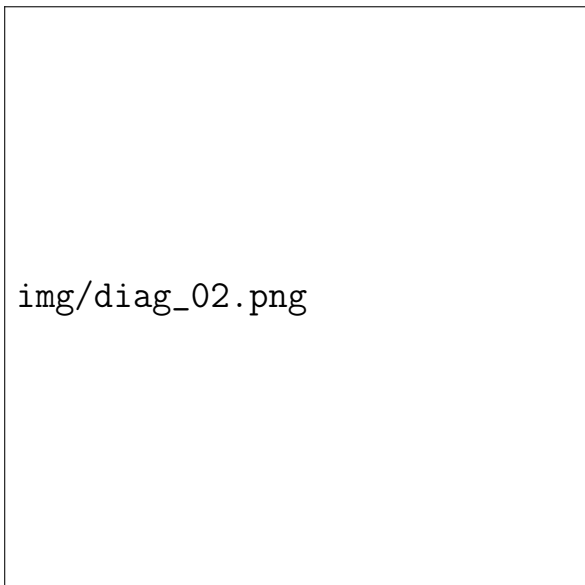
Кроме того, согласно данным, приведенным в таблице 4.1, рекурсивные алгоритмы при длинах строк более 10 элементов не пригодны к использованию в силу экспоненциально роста затрат процессорного времени, в то время, как затраты итеративных алгоритмов по времени линейны.

4.4 Характеристики по памяти

Введем следующие обозначения:

- n — длина строки S_1 ;
- m — длина строки S_2 ;
- $size()$ — функция вычисляющая размер в байтах;
- *string* — строковый тип;
- *int* — целочисленный тип;
- *size_t* — беззнаковый целочисленный тип.

Максимальная глубина стека вызовов при рекурсивной реализации нахождения расстояния Дамерау-Левенштейна равна сумме входящих строк,



img/diag_02.png

Рисунок 4.3 – Сравнение по времени алгоритмов поиска расстояния
Дамерау-Левенштейна

а на каждый вызов требуется 2 дополнительные переменные, соответственно, максимальный расход памяти равен:

$$(n + m) \cdot (2 \cdot \text{size}(\text{string}) + 3 \cdot \text{size}(\text{int}) + 2 \cdot \text{sizeof}(\text{size_t})), \quad (4.1)$$

где:

- $2 \cdot \text{size}(\text{string})$ — хранение двух строк;
- $2 \cdot \text{size}(\text{size_t})$ — хранение размеров строк;
- $2 \cdot \text{size}(\text{int})$ — дополнительные переменные;
- $\text{size}(\text{int})$ — адрес возврата.

Для рекурсивного алгоритма с кешированием поиска расстояния Дамерау-Левенштейна будет теоретически схож с расчетом в формуле (4.1), но также учитывается матрица, соответственно, максимальный расход памяти равен:

$$(n + m) \cdot (2 \cdot \text{size}(\text{string}) + 3 \cdot \text{size}(\text{int}) + 2 \cdot \text{size}(\text{size_t})) + \\ + (n + 1) \cdot (m + 1) \cdot \text{size}(\text{int}) \quad (4.2)$$

Использование памяти при итеративной реализации алгоритма поиска рас-

стояния Левенштейна теоретически равно:

$$(n + 1) \cdot (m + 1) \cdot \text{size}(\text{int}) + 2 \cdot \text{size}(\text{string}) + 2 \cdot \text{size}(\text{size_t}) + \text{size}(\text{int} **) + (n + 1) \cdot \text{size}(\text{int}*) + 2 \cdot \text{size}(\text{int}), \quad (4.3)$$

где

- $2 \cdot \text{size}(\text{string})$ — хранение двух строк;
- $2 \cdot \text{size}(\text{size_t})$ — хранение размеров матрицы;
- $(n + 1) \cdot (m + 1) \cdot \text{size}(\text{int})$ — хранение матрицы;
- $\text{size}(\text{int} **) + (n + 1) \cdot \text{size}(\text{int}*)$ — указатель на матрицу;
- $\text{size}(\text{int})$ — дополнительная переменная для хранения результата;
- $\text{size}(\text{int})$ — адрес возврата.

Использование памяти при итеративной реализации алгоритма поиска расстояния Дамерау-Левенштейна теоретически равно:

$$(n + 1) \cdot (m + 1) \cdot \text{size}(\text{int}) + 2 \cdot \text{size}(\text{string}) + 2 \cdot \text{size}(\text{size_t}) + \text{size}(\text{int} **) + (n + 1) \cdot \text{size}(\text{int}*) + 3 \cdot \text{size}(\text{int}), \quad (4.4)$$

где

- $2 \cdot \text{size}(\text{string})$ — хранение двух строк;
- $2 \cdot \text{size}(\text{size_t})$ — хранение размеров матрицы;
- $(n + 1) \cdot (m + 1) \cdot \text{size}(\text{int})$ — хранение матрицы;
- $\text{size}(\text{int} **) + (n + 1) \cdot \text{size}(\text{int}*)$ — указатель на матрицу;
- $2 \cdot \text{size}(\text{int})$ — дополнительные переменные;
- $\text{size}(\text{int})$ — адрес возврата.

По расходу памяти итеративные алгоритмы проигрывают рекурсивным: максимальный размер используемой памяти в итеративном растет как произведение длин строк, в то время как у рекурсивного алгоритма — как сумма длин строк.

По формулам 4.1 – 4.3 затрат по памяти в программе были написаны соответствующие функции для подсчета расходуемой памяти, результаты расчетов, которых представлены в таблице 4.2, где размеры строк находятся в диапазоне от 10 до 200 с шагом 10.

Таблица 4.2 – Замер памяти для строк, размером от 10 до 200

Длина (символ)	Размер в байтах			
	Левенштейн	Дамерау-Левенштейн		
	Итеративный	Итеративный	Рекурсивный	
			Без кеша	С кешом

Из данных, приведенных в таблице 4.2, понятно, что рекурсивные алгоритмы являются более эффективными по памяти, так как используется только память под локальные переменные, передаваемые аргументы и возвращаемое значение, в то время как итеративные алгоритмы затрачивают память линейно пропорционально длинам обрабатываемых строк.

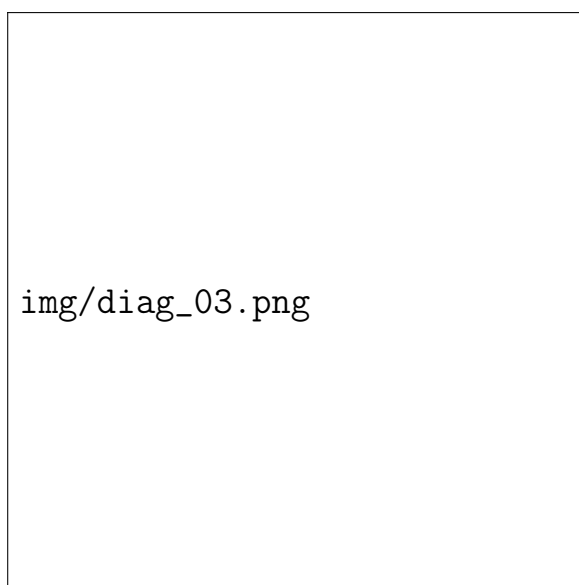
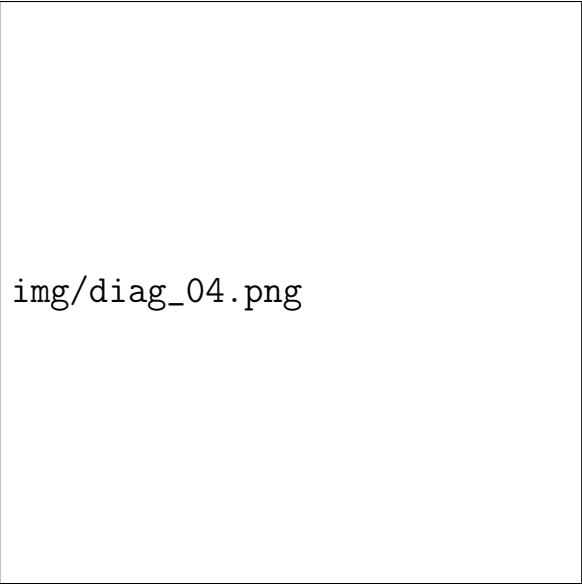


Рисунок 4.4 – Сравнение по памяти алгоритмов поиска расстояния Левенштейна и Дамерау-Левенштейна — итеративной и рекурсивной реализации

Из рисунка 4.5 понятно, что рекурсивная реализация алгоритма поиска расстояния Дамерау-Левенштейна эффективная по памяти, чем итеративная.



img/diag_04.png

Рисунок 4.5 – Сравнение по памяти алгоритмов поиска расстояния Дамерау-Левенштейна — итеративной и рекурсивной реализации

4.5 Вывод

В данном разделе было произведено сравнение количества затраченного времени и памяти алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна. Наименее затратным по времени оказался итеративный алгоритм нахождения расстояния Левенштейна.

Приведенные характеристики показывают, что рекурсивная реализация алгоритма в 21 раз проигрывает по времени. В связи с этим, рекурсивные алгоритмы следует использовать лишь для малых размерностей строк (1 – 4 символа).

Так как во время печати очень часто возникают ошибки связанные с транспозицией букв [ulianov], алгоритм поиска расстояния Дамерау-Левенштейна является наиболее предпочтительным, не смотря на то, что он проигрывает по времени и памяти алгоритму Левенштейна.

Рекурсивная реализация алгоритма поиска расстояния Дамерау-Левенштейна будет более затратным по времени по сравнению с итеративной реализацией алгоритма поиска расстояния Дамерау-Левенштейна, но менее затратным по памяти по отношению к итеративному алгоритму Дамерау-Левенштейна.

Заключение

В результате исследования было определено, что время алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна растет в геометрической прогрессии при увеличении длин строк. Лучшие показатели по времени дает матричная реализация алгоритма нахождения расстояния Дамерау-Левенштейна и его рекурсивная реализация с кешем, использование которых приводит к 21-кратному превосходству по времени работы уже на длине строки в 4 символа за счет сохранения необходимых промежуточных вычислений. При этом итеративная реализации с использованием матрицы занимают довольно много памяти при большой длине строк.

Цель данной лабораторной работы были достигнуты, а именно описание и исследование особенностей задач динамического программирования на алгоритмах Левенштейна и Дамерау-Левенштейна.

Для достижения поставленной целей были выполнены следующие задачи.

- 1) Описаны алгоритмы поиска расстояния Левенштейна и Дамерау-Левенштейна;
- 2) Создано программное обеспечение, реализующее следующие алгоритмы.
 - нерекурсивный метод поиска расстояния Левенштейна;
 - нерекурсивный метод поиска расстояния Дамерау-Левенштейна;
 - рекурсивный метод поиска расстояния Дамерау-Левенштейна;
 - рекурсивный с кешированием метод поиска расстояния Дамерау-Левенштейна.
- 3) Выбраны инструменты для замера процессорного времени выполнения реализаций алгоритмов.
- 4) Проведены анализ затрат работы программы по времени и по памяти, выяснить влияющие на них характеристики.