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: «Редакционное расстояние»

Студент группы ИУ7-51Б _____ (Подпись, дата) Шубенина Д. 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

Расстояние Левенштейна (также называемое редакционным расстоянием или дистанцией редактирования) — это метрика, которая измеряет разницу между двумя строками. Определяет минимальное количество операций вставки, удаления и замены символов, необходимых для преобразования одной строки в другую.

Расстояние Дамерау-Левенштейна является расширением расстояния Левенштейна, которое включает дополнительную операцию — транспозицию, чтобы обработать случаи, когда символы меняются местами или переупорядочиваются.

Расстояния Левенштейна и Дамерау-Левенштейна используются при решении следующих задач:

1) корректировка поискового запроса;
2) классификация текстов;
3) распознавание речи;
4) определение сходства между текстами;

Целью данной лабораторной работы является изучение, реализация и исследование алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна.

Необходимо выполнить следующие **задачи**:

1) изучить алгоритмы Левенштейна и Дамерау-Левенштейна для нахождения редакционного расстояния между строками;
2) реализовать данные алгоритмы;
3) выполненить сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени, памяти);
4) описать и обосновать полученные результаты в отчете.

1 Аналитическая часть

1.1 Расстояние Левенштейна

Расстояние Левенштейна между двумя строками — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения строки в другую [1].

Введем следующие обозначения операций:

— $w(a, b)$ – цена замены символа a на символ b;

— $w(\varepsilon, b)$ – цена вставки символа b;

— $w(a, \varepsilon)$ – цена удаления символа a;

Каждая операция имеет определенную цену:

— **M** (от англ. match): $w(a, a) = 0$

— **R** (от англ. replace): $w(a, b) = 1, a \neq b$

— **I** (от англ. insert): $w(\varepsilon, b) = 1$

— **D** (от англ. delete): $w(a, \varepsilon) = 1$

Пусть имеется две строки S_1 и S_2 длиной m и n соотвественно. Расстояние Левенштейна $d(S_1, S_2) = D(m, n)$ рассчитывается по следующей рекуррентной формуле [2]:

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

где сравнение символов строк S_1 и S_2 производится следующим образом:

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

1.1.1 Нерекурсивный алгоритм нахождения расстояния Левенштейна

При больших значениях m и n рекурсивная реализация алгоритма поиска расстояния Левенштейна малоэффективна по времени выполнения, так как промежуточные значения $D(i,j)$ вычисляются несколько раз. В таком случае можно воспользоваться итеративной реализацией данного алгоритма, использующего матрицу для хранения промежуточных значений.

Матрица имеет размеры

$$(m+1) \times (n+1) \tag{1.3}$$

где m — длина строки S_1, n — длина строки S_2.

В ячейке $[i,j]$ матрицы хранится значение $D(S_1[1..i], S_2[1..j])$. Первому элементу матрицы присвоено значение 0. Вся матрица заполняется в соотвествии с соотношением (1.1).

1.2 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является метрикой для измерения различий между двумя строками. Оно определяется как минимальное количество операций вставки, удаления, замены и транспозиции (перестановки двух соседних символов), необходимых для превращения одной строки в другую. Это расширение расстояния Левенштейна, так как, помимо трех базовых операций, оно также включает операцию транспозиции T (от англ. transposition).

Расстояние Дамерау-Левенштейна определяется по следующей рекуррентной формуле:

$$D(m,n) = \begin{cases} \max(i,j), & \min(i,j) = 0 \\ \min \begin{cases} D(i,j-1)+1, \\ D(i-1,j)+1, \\ D(i-1,j-1), \\ D(i-2,j-2)+1, \end{cases} & \text{если } i,j > 1, \\ & S_1[i] = S_2[j-1], \\ & S_1[i-1] = S_2[j] \\ \min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1) + \mathrm{m}(S_1[i], S_2[j]) \end{cases} & \text{иначе.} \end{cases} \tag{1.4}$$

1.2.1 Рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна

Рекурсивный алгоритм поиска расстояния Дамерау-Левенштейна реализует формулу (1.4) следующим образом:

1) Если одна из строк пустая, возвращается длина другой строки.

2) Если последние символы двух строк совпадают, рекурсивно вызывается функция для остатков строк (без последних символов).

3) Иначе рекурсивно вызываются четыре варианта преобразования строки:

   — **Вставка**: к результату рекурсивного вызова для остатка первой строки добавляется 1.

   — **Удаление**: к результату рекурсивного вызова для остатка второй строки добавляется 1.

   — **Замена**: к результату рекурсивного вызова для остатков строк добавляется 1.

— **Транспозиция**: если последние и предпоследние символы двух строк совпадают, к результату рекурсивного вызова для остатка строк добавляется 1.

4) Возвращается минимальное из четырех вариантов значение.

1.2.2 Рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна с кэшированием

При больших m и n рекурсивная реализация алгоритма поиска расстояния Дамерау-Левенштейна малоэффективна по времени, так как промежуточные значения расстояний между подстроками вычисляются неоднократно. Для оптимизации рекурсивного алгоритма по времени можно использовать матрицу в целях хранения соответствующих промежуточных значений. В таком случае алгоритм представляет собой рекурсивное заполнение матрицы $A_{m,n}$ промежуточными значениями $D(i, j)$.

1.2.3 Нерекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна

При больших значениях m, n алгоритм нахождения расстояния Дамерау-Левенштейна, использующий рекурсию, не является эффективным по времени. Вместо рекурсивной реализации можно использовать итерационную реализацию. В таком случае в качестве структуры для хранения промежуточных значений $D(i, j)$ используется матрица, имеющая размеры

$$(m + 1) \times (n + 1) \tag{1.5}$$

В ячейке $[i, j]$ матрицы хранится значение $D(S_1[1..i], S_2[1..j])$. Первому элементу матрицы присвоено значение 0. Вся матрица заполняется в

соотвествии с соотношением (1.4).

Вывод

В данном разделе были рассмотрены алгоритмы нахождения расстояний Левенштейна и Дамерау-Левенштейна — их рекурсивные и итеративные реализации. Также была рассмотрена оптимизация алгоритма нахождения расстояния Дамерау-Левенштейна с помощью кэширования.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна, приведены описание используемых типов данных и структуры программного обеспечения.

2.1 Требования к программному обеспечению

К программе предъявлен ряд функциональных требований:

— наличие интерфейса для выбора действий;

— возможность ввода строк;

— возможность обработки строк, состоящих как из латинских символов, так и из кириллических;

— возможность произвести замеры процессорного времени работы реализованных алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна.

2.2 Требования вводу

1) На вход реализованным алгоритмам подаются две строки.

2) Строки могут включать как латинские, так и кириллические символы.

3) Буквы нижнего и верхнего регистра считаются разными символами.

2.3 Разработка алгоритмов

На рисунке 2.1 представлена схема матричного алгоритма поиска расстояния Левенштейна. На рисунках 2.2 приведены схемы матричной,

рекурсивной и рекурсивной с кэшированием реализаций алгоритма нахождения расстояния Дамерау-Левенштейна.

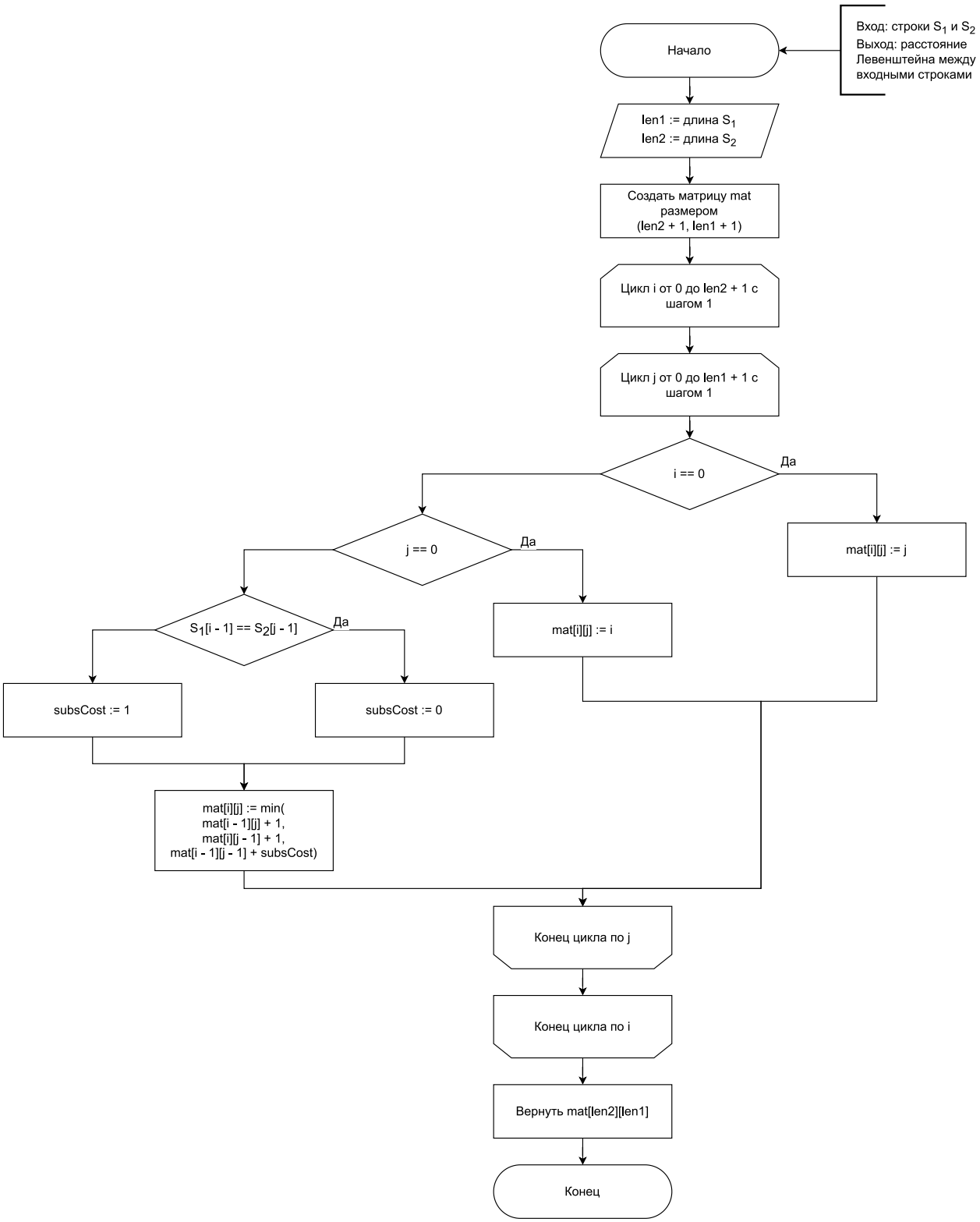

Рисунок 2.1 – Схема матричного алгоритма Левенштейна

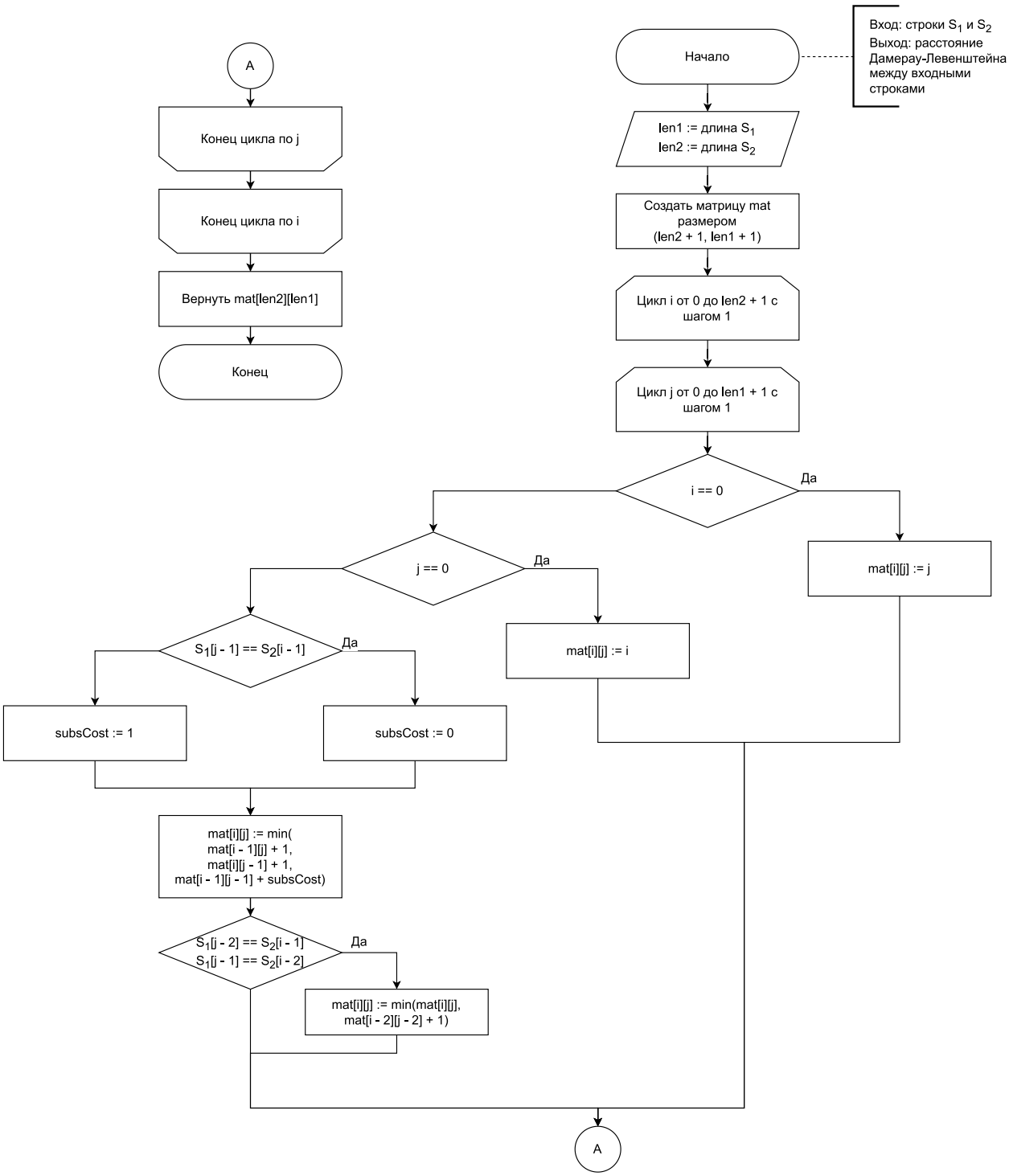

Рисунок 2.2 – Схема матричного алгоритма Дамерау-Левенштейна

2.4 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— *строка* — массив символов типа `wchar_t`;

— *длина строки* — значение длины строки типа `int`;

— *матрица* — двумерный массив значений типа `int`.

Вывод

Были реализованы алгоритмы Левенштейна (итеративно) и Дамерау-Левенштейна (итеративно, рекурсивно, рекурсивно с кэшированием). Проведено тестирование реализованных алгортимов.

3 Технологическая часть

В данном разделе приведены средства реализации программного обеспечения, сведения о модулях программы, листинг кода и функциональные тесты.

3.1 Средства реализации

В качестве языка, используемого при написании данной лабораторной работы, был выбран язык C++ [3]. Этот выбор обусловлен тем, что в данном языке программирования имеется контейнер `std::wstring`, представляющий собой массив символов типа `std::wchar_t`. Также в языке C++ имеется библиотека `<ctime>`, позволяющая выполнять замеры процессорного времени.

В качестве средства написания кода была выбрана кроссплатформенная среда разработки *Visual Studio Code*, т.к. она предоставляет широкий функционал для проектирования, разработки и отладки ПО.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу;

— `matrix.cpp` — файл, содержащий функции создания матрицы, ее освобождения и вывода на экран;

— `algorithms.cpp` — файл, содержащий реализации алгоритмов поиска расстояний Левенштейна и Дамерау-Левенштейна;

— `measure.cpp` — файл, содержащий функции, замеряющие процессорное время выполнения реализуемых алгоритмов.

3.3 Реализация алгоритмов

3.4 Функциональные тесты

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Intel i5-1035G1 (8) @ 3.600GHz.

— Оперативная память: 16 ГБайт.

— Операционная система: Manjaro Linux x86_64 (версия ядра Linux 5.15.131-1-MANJARO).

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 показан пример работы разработанной программы для случая, когда пользователь выбирает действие «Запуск алгоритмов поиска расстояния Левенштейна» и вводит строки «кот» и «кошка».

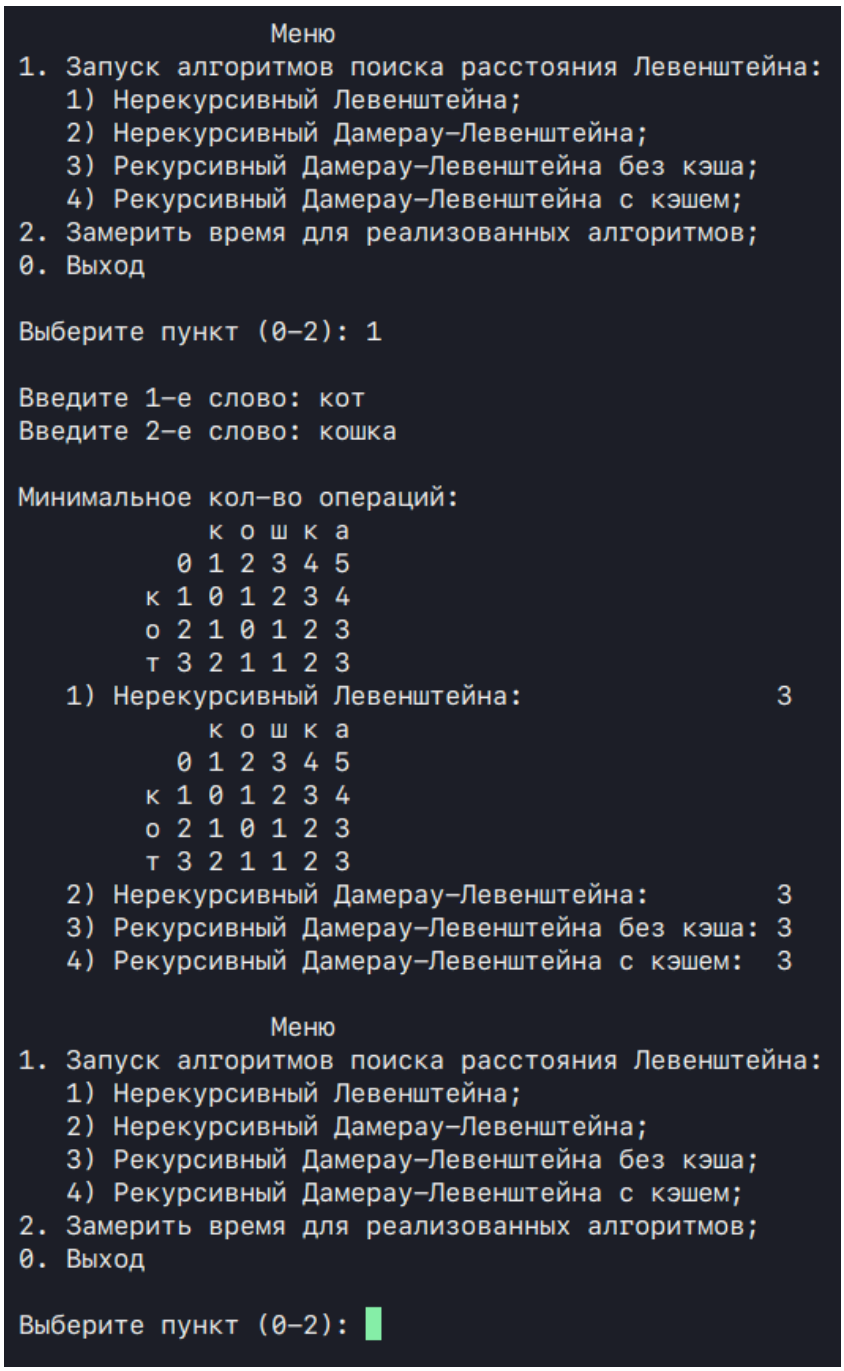

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Исследование временных характеристик алгоритмов производилось на случайно сгенерированных строках длинами от 1 до 10, длины изменяются с шагом 1. Для нерекурсивных алгоритмов отдельно производилось сравнение на строках длинами от 20 до 100 с шагом изменения длины 10. Во

избежание погрешности измерения для каждой строки производились 50 раз, затем вычислялось среднее арифметическое всех полученных значений времени.

4.4 Характеристики по памяти

Введем следующие обозначения:

— m — длина строки S_1;

— n — длина строки S_2;

— $size(v)$ — функция, вычисляющая размер входного параметра v в байтах;

— $char$ — тип данных, используемый для хранения символа строки;

— int — целочисленный тип данных.

Теоретически оценим объем используемой памяти итеративной реализацией алгоритма поиска расстояния Левенштейна:

$$\begin{aligned} M_{LevIter} = (m+1)\cdot(n+1)\cdot size(int) + (m+n)\cdot size(char) + \\ + size(int**) + (m+1)\cdot size(int*) + \\ + 3\cdot size(int) + 2\cdot size(int), \end{aligned} \tag{4.1}$$

где:

— $(m+1)\cdot(n+1)\cdot size(int)$ — размер матрицы;

— $size(int**)$ — размер указателя на матрицу;

— $(m+1)\cdot size(int*)$ — размер указателей на строки матрицы;

— $(m+n)\cdot size(char)$ — размер двух входных строк;

— $2\cdot size(int)$ — размер переменных, хранящих длину строк;

— $3 \cdot size(int)$ — размер переменных цикла i, j и промежуточной переменной $subsCost$.

Для алгоритма поиска расстояния Дамерау-Левенштейна теоретическая оценка объема используемой памяти идентична.

Произведем оценку затрат по памяти для рекурсивных реализаций алгоритма нахождения расстояния Дамерау-Левенштейна

4.5 Вывод

Список использованных источников

1 А. Погорелов Д., М. Таразанов А., Л. Волкова Л. Сравнительный анализ алгоритмов редакционного расстояния Левенштейна и Дамерау-Левенштейна // Синергия Наук. 2019. URL: https://elibrary.ru/item.asp?id=36907767.

2 В. Траулько М. Программная реализация нечеткого поиска текстовой информации в словаер с помощью расстояния Левенштейна // Форум молодых ученых. 2017. URL: https://cyberleninka.ru/article/n/programmnaya-realizatsiya-nechetkogo-poiska-tekstovoy-informatsii-v-slovare-s-pomoschyu-rasstoyaniya-levenshteyna.

3 Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017` (дата обращения: 25.09.2022).