Оглавление

Введение

Целью лабораторной работы является изучение и реализация алгоритмов нахождения расстояний Левенштейна и Дамерлау-Левештейна, а также получения навыка динамического программирования. Для её достижения необходимо выполнить следующие задачи:

- Изучение алгоритмов Левенштейна и Дамерлау-Левештейна
- Разработать данные алгоритмы
- Применение методов динамического программирования для реализации алгоритмов
- Выполнить тестирование реализации алгоритмов методом черного ящика
- Провести сравнительный анализ этих алгоритмов по затратам памяти и процессорному выполнению времени на основе экспериментальных данных

1 Аналитический раздел

1.1 Расстояние Левенштейна

Расстояние Левенштейна (редакторское расстояние) между двумя строками [1] - минимальное количество операций вставки, удаления и замены, необходимых для превращения одной строки в другую.
При преобразовании одной строки в другую, используются следующие операции:

a) I (insert) - вставка;

b) D (delete) - удаление;

c) R (replace) - замена;

Будем считать, что стоимость каждой из этих операции (штраф) равна 1.
Введем еще одну операцию M (match) - совпадение. Её стоимость будет равна нулю. Необходимо найти последовательность замен с минимальным суммарным штрафом.

1.2 Рекурсивный алгоритм

Расстояние между двумя строками s1 и s2 рассчитывается по реккуретной формуле D:

$$D(s1[1..i], s2[1..j]) = \begin{cases} 0, & \text{i=0, j=0} \\ j, & \text{i=0, j>0} \\ i, & \text{i>0, j=0} \\ \min \begin{cases} D(s1[1..i], s2[1..j-1]) + 1 \\ D(s1[1..i-1], s2[1..j]) + 1 \\ D(s1[1..i-1], s2[1..j-1]) + f(s1, s2) \end{cases} \end{cases} \tag{1.2.1}$$

Функция f(s1, s2) определяется следующим образом:

$$f(s1, s2) = \begin{cases} 0, \text{ s1=s2} \\ 1, \text{ иначе} \end{cases} \tag{1.2.2}$$

1.3 Матрица расстояний

Реализация формулы 1.2.1 при больших значениях i,j, оказывается менее эффективной по времени ввиду того, что приходится вычислять промежуточные результаты неоднократно. Для оптимизации нахождения расстояния Левенштейна необходимо использовать матрицу стоимостей для хранения этих промежуточных значений. Таким образом, будет необходимо выполнять только построчное заполнение такой матрицы.

1.4 Рекурсивный алгоритм с кэшем в форме матрицы

При помощи использования матрицы можно выполнить оптимизацию рекурсивного алгоритма заполнения. Основная идея такого подхода заключается в том, что при каждом рекурсивном вызове алгоритма выполняется заполнение матрицы стоимостей. Главное отличие данного метода от того, что был описан в разделе 2.3 - начальная инициализация матрицы флагом ∞. То есть если рекурсивный алгоритм выполняет прогон для данных, которые не были обработаны, результат заносится в матрицу. Если же используются повторные данные (ячейка матрицы была заполнена), то алгоритм не выполняет вычисление расстояния для этой ячейки, а сразу переходит к следующему шагу.

1.5 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является модификацией расстояния Левенштейна, которая задействует еще одну редакторскую операцию - транспозицию T (transposition. Она выполняет обмен соседних символов в слове.

Дамерау показал, что 80% человеческих ошибок при наборе текстов является перестановка соседних символов, пропуск символа, добавление нового символа или ошибочный символ. Таким образом, расстояние Дамерау-Левенштейна часто используется в редакторских программах для проверки правописания. Это расстояние может быть вычислено по следующей реккуретной формуле:

$$D(s1[1..i], s2[1..j]) = \begin{cases} 0, & \text{i=0, j=0} \\ j, & \text{i=0, j>0} \\ i, & \text{i>0, j=0} \\ \min \begin{cases} D(s1[1..i], s2[1..j-1]) + 1 \\ D(s1[1..i-1], s2[1..j]) + 1 \\ D(s1[1..i-1], s2[1..j-1]) + f(s1, s2) \\ D(s1[1..i-1], s2[1..j-1]) + 1, \ \text{i,j} > 1, a_i = b_{j-1}, \ a_{i-1} \\ \infty, \ \text{иначе} \end{cases} \end{cases} \tag{1.5.1}$$

1.6 Вывод

В данном разделе были рассмотрены основные способы нахождения редакторского расстояния между двумя строками. Формулы для нахождения расстояния Левенштейна и Дамерау-Левенштейна задаются реккуретно, следовательно, алгоритмы могут быть реализованы как рекурсивно, так и итерационно.

2 Конструкторская часть

2.1 Схемы алгоритмов Левенштейна

рис 2.1 - схема алгоритма итеративного алгоритма Левенштейна с использованием двух строк
рис 2.2 - схема алгоритма рекурсивного алгоритма Левенштейна без кэша
рис 2.3 - схема алгоритма рекурсивного алгоритма Дамерау-Левенштейна с использованием матрицы
рис 2.4 - схема алгоритма итеративного алгоритма Левенштейна с использованием двух строк

2.2 Вывод

На основе теоретических данных, полученных в аналатическом разделе, были построены схемы нужных алгоритмов.

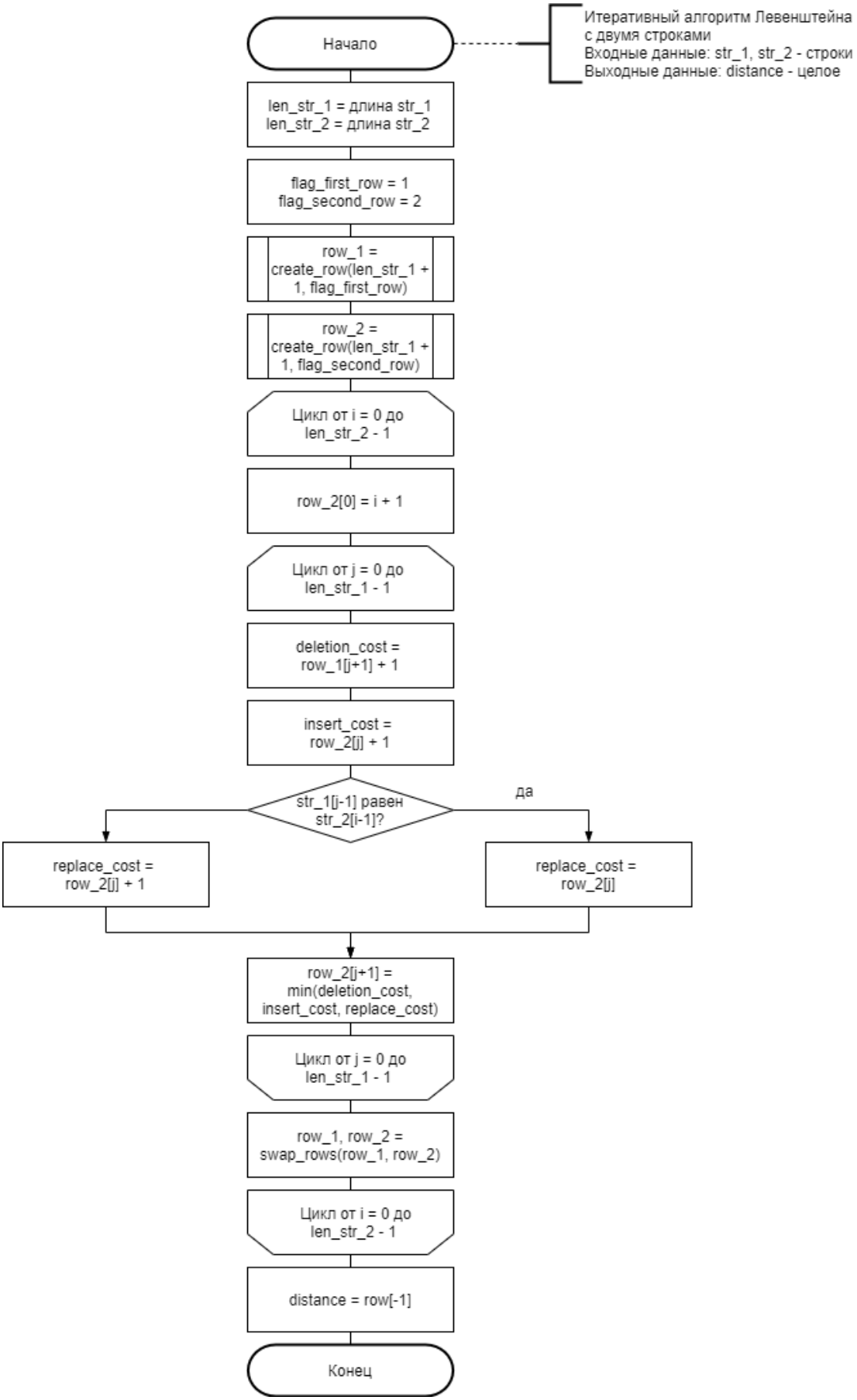

Рисунок 2.1 – Сравнение времени работы алгоритмов Левенштейна.

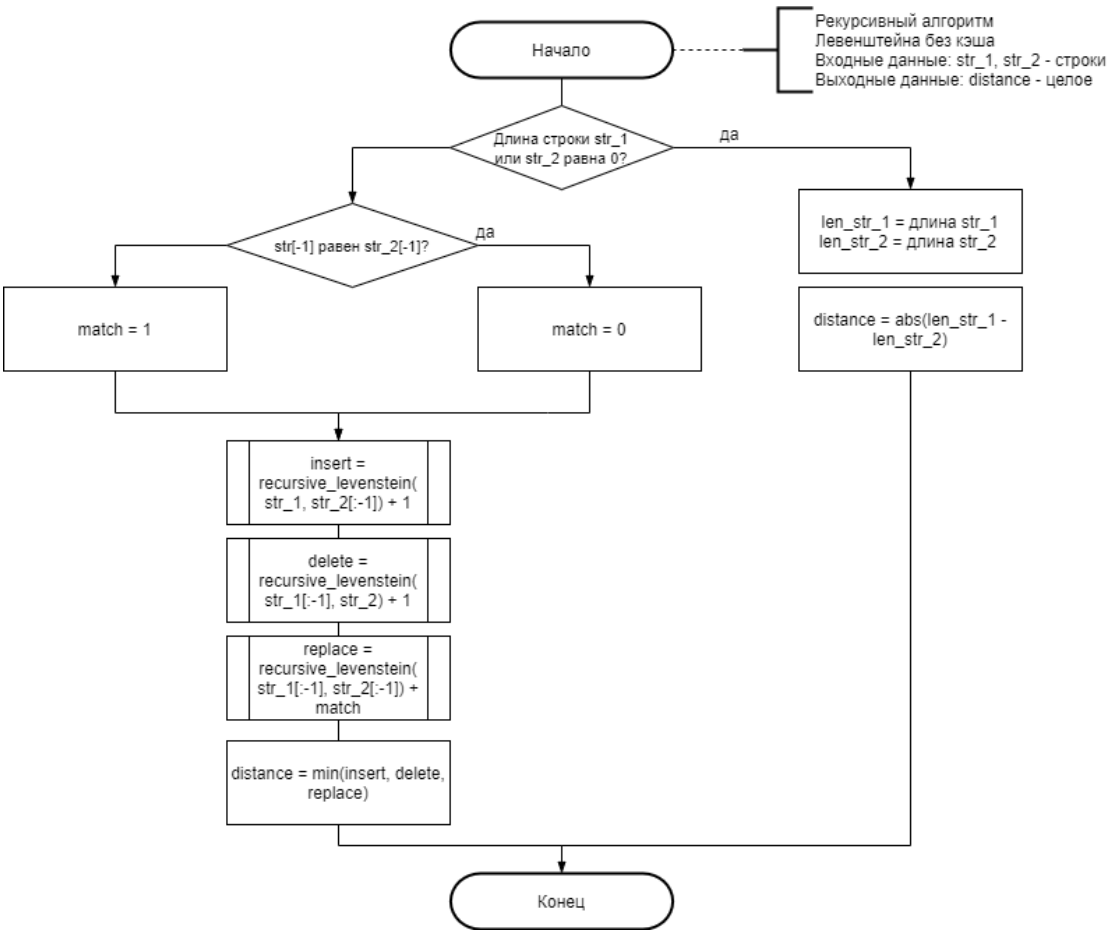

Рисунок 2.2 – Сравнение времени работы алгоритмов Левенштейна.

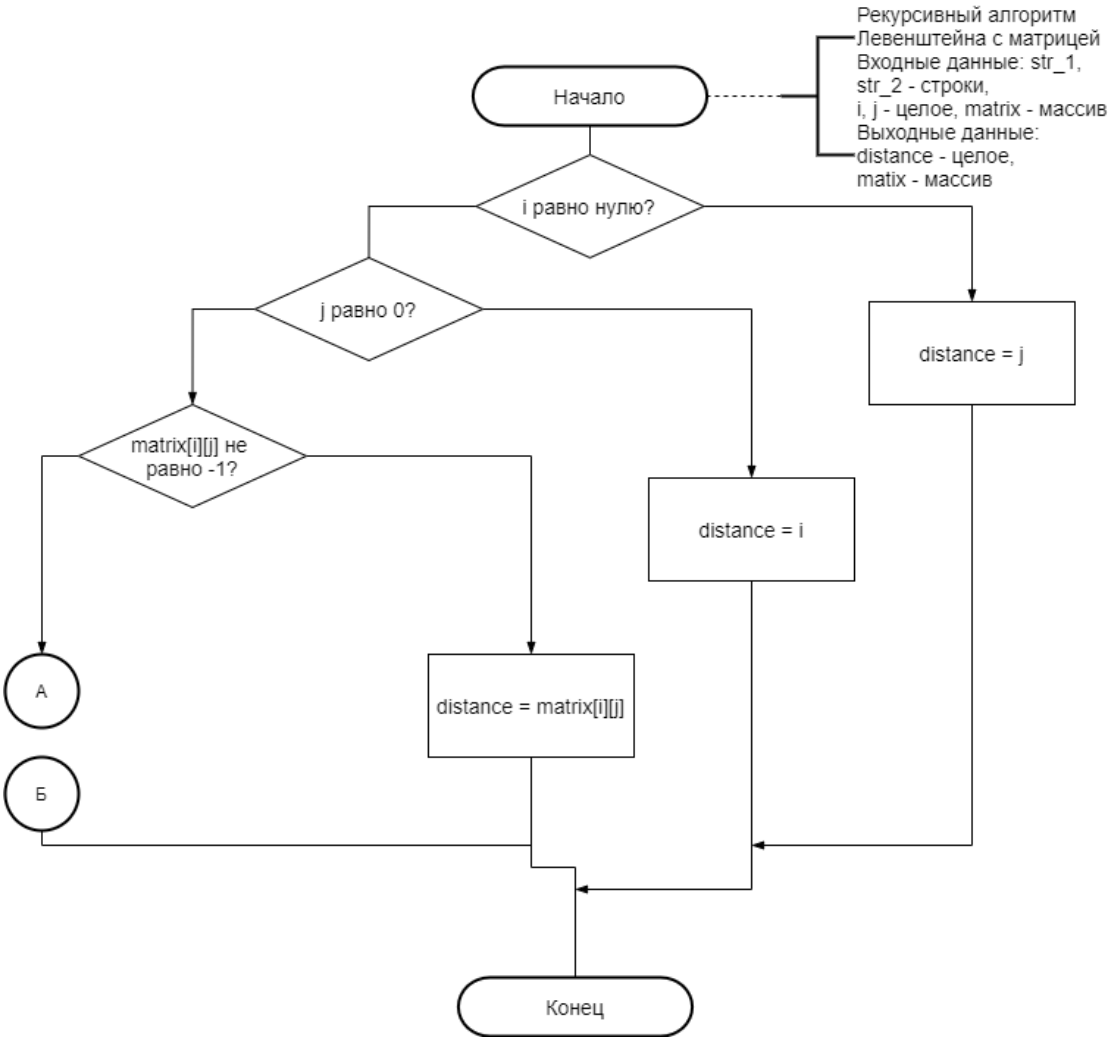

Рисунок 2.3 – Сравнение времени работы алгоритмов Левенштейна.

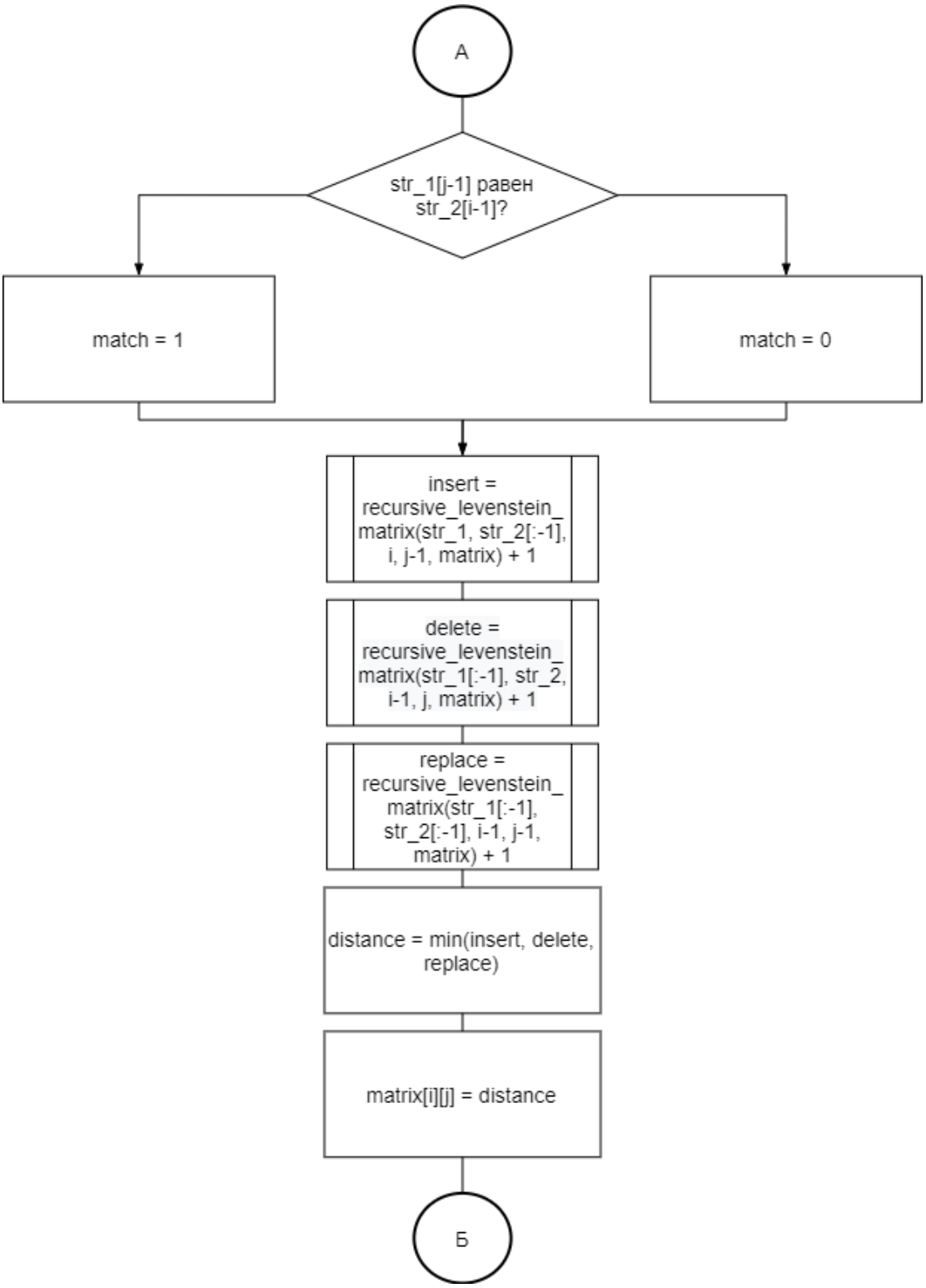

Рисунок 2.4 – Сравнение времени работы алгоритмов Левенштейна.

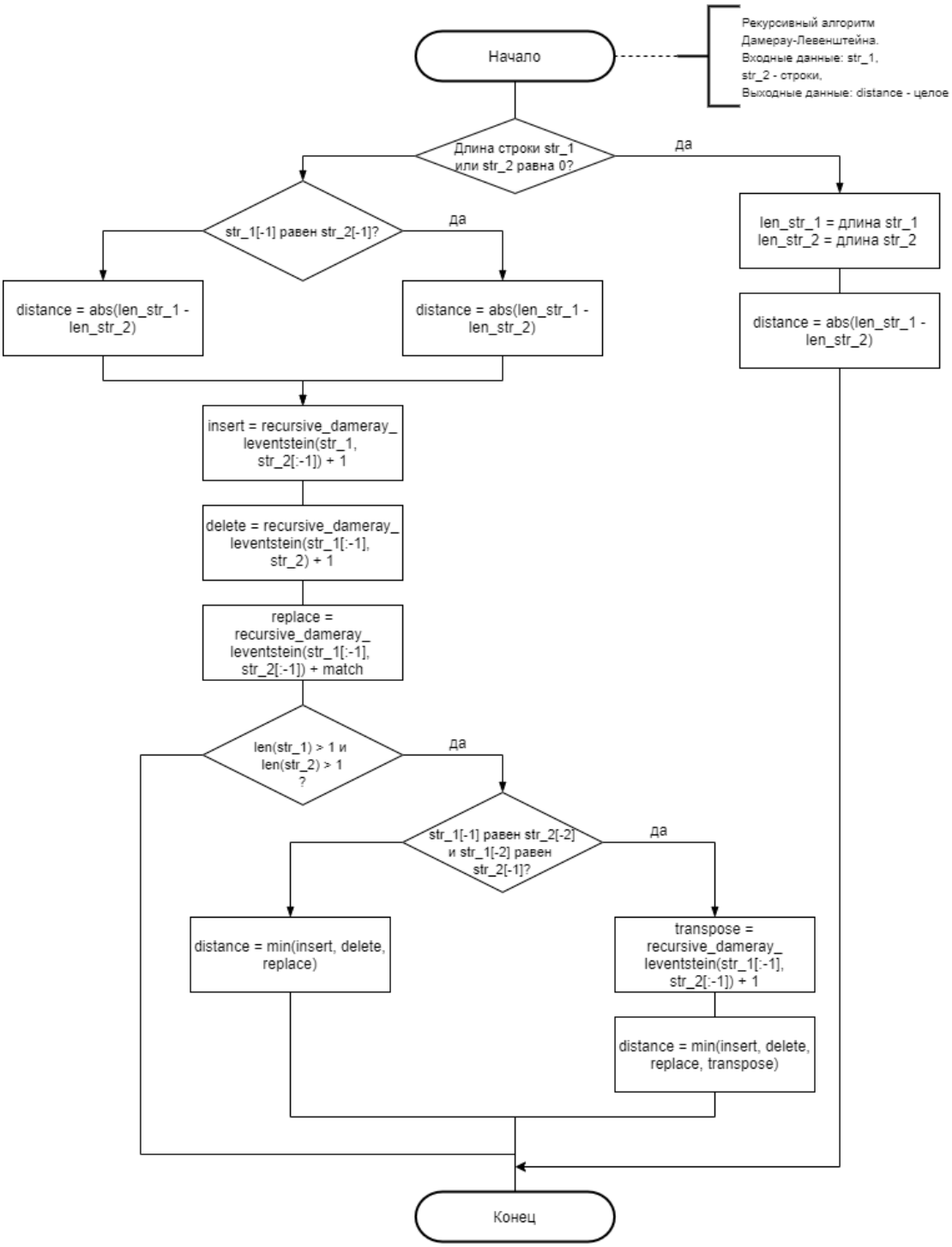

Рисунок 2.5 – Сравнение времени работы алгоритмов Левенштейна.