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 "Функциональное и логическое программирование"

Тема Язык Lisp

Студент Варин Д.В.

Группа ИУ7-66Б

Оценка (баллы) _____

Преподаватели Толпинская Н.Б.

Москва — 2022 г.

Оглавление

Введение

Цель лабораторной работы - изучение, реализация и исследование алгоритмов нахождения расстояний Левенштейна и Дамерау – Левенштейна.

Расстояние Левенштейна (редакционное расстояние) — метрика, позволяющая определить минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую [?]. Операциям, используемым в этом преобразовании, можно назначить разные цены. Широко используется в биоинформатике и компьютерной лингвистике.

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 применяется:

1) для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи);

2) для сравнения текстовых файлов утилитой `diff` unix систем (здесь роль «символов» играют строки, а роль «строк» — файлы);

3) в биоинформатике для сравнения генов, белков и др..

Расстояние Дамерау — Левенштейна (названо в честь учёных Фредерика Дамерау и Владимира Левенштейна) — этот алгоритм является модификацией расстояния Левенштейна. В нём к операциям удаления, вставки, замены добавляется операция транспозиции (перестановки двух соседних символов).

Задачи лабораторной работы:

- изучение алгоритмов Левенштейна и Дамерау–Левенштейна;
- получение практических навыков реализации алгоритмов Левенштейна и Дамерау — Левенштейна;
- исследование алгоритмов;

- применение методов динамического программирования;
- проведение сравнительного анализа алгоритмов на основе полученных экспериментальных данных;
- сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовка отчета по лабораторной работе.

1 Аналитическая часть

Расстояние Левенштейна [?] между двумя строками — это минимальное количество операций вставки, удаления и замены, необходимых для превращения одной строки в другую.

Каждая операция имеет цену, которая зависит от вида операции (вставка, удаление, замена) и/или от участвующих в ней символов, отражая разную вероятность разных ошибок при вводе текста, и т. п. В общем случае:

- $w(a, b)$ — цена замены символа a на символ b.
- $w(\lambda, b)$ — цена вставки символа b.
- $w(a, \lambda)$ — цена удаления символа a.

Для решения задачи о редакционном расстоянии необходимо найти последовательность замен, минимизирующую суммарную цену. Расстояние Левенштейна является частным случаем этой задачи при

- $w(a, a) = 0$.
- $w(a, b) = 1,\ a \neq b$.
- $w(\lambda, b) = 1$.
- $w(a, \lambda) = 1$.

1.1 Рекурсивный алгоритм нахождения расстояния Левенштейна

Расстояние Левенштейна между двумя строками a и b может быть вычислено по формуле 1.1, где $|a|$ означает длину строки a; $a[i]$ — i-ый символ строки a , функция $D(i,j)$ определена как:

$$D(i,j) = \begin{cases} 0 & \text{i} = 0, \text{j} = 0 \\ i & \text{j} = 0, \text{i} > 0 \\ j & \text{i} = 0, \text{j} > 0 \\ \min\{ & \\ \quad D(i,j-1)+1 & \\ \quad D(i-1,j)+1 & \text{i} > 0, \text{j} > 0 \\ \quad D(i-1,j-1)+m(a[i],b[j]) & (1.2) \\ \} & \end{cases}, \qquad (1.1)$$

где функция 1.2 имеет следующий вид:

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

Рекурсивный алгоритм реализует формулу 1.1. Функция D удовлетворяет следующим правилам:

1) для перевода из пустой строки в пустую требуется ноль операций;

2) для перевода из пустой строки в строку a требуется $|a|$ операций;

3) для перевода из строки a в пустую требуется $|a|$ операций;

Для перевода из строки a в строку b требуется некоторое количество операций (удаление, вставка, замена) в некоторой последовательности. Последовательность проведения любых двух операций можно поменять, порядок проведения операций не имеет никакого значения. Полагая, что a', b' —

строки a и b без последнего символа соответственно, цена преобразования из строки a в строку b может быть выражена как:

1) сумма цены преобразования строки a в b и цены проведения операции удаления, которая необходима для преобразования a' в a;

2) сумма цены преобразования строки a в b и цены проведения операции вставки, которая необходима для преобразования b' в b;

3) сумма цены преобразования из a' в b' и операции замены, предполагая, что a и b оканчиваются разные символы;

4) цена преобразования из a' в b', предполагая, что a и b оканчиваются на один и тот же символ.

Минимальной ценой преобразования будет минимальное значение приведенных вариантов.

1.2 Матричный алгоритм нахождения расстояния Левенштейна

Реализация формулы 1.1 может быть малоэффективна по времени исполнения при больших i, j, т. к. множество промежуточных значений $D(i, j)$ вычисляются множество раз подряд. Можно оптимизировать алгоритм, используя подход динамического программирования [**?**]. Использование матрицы для хранения промежуточных значений позволит оптимизировать алгоритм. В таком случае алгоритм представляет собой построчное заполнение матрицы $A_{|a|,|b|}$ значениями $D(i, j)$.

1.3 Рекурсивный алгоритм нахождения расстояния Левенштейна с заполнением матрицы

Рекурсивный алгоритм заполнения можно оптимизировать по времени выполнения с использованием матричного алгоритма. Суть данного метода заключается в параллельном заполнении матрицы при выполнении рекурсии. В случае, если рекурсивный алгоритм выполняет прогон для данных, которые еще не были обработаны, результат нахождения расстояния заносится в матрицу. В случае, если обработанные ранее данные встречаются снова, для них расстояние не находится и алгоритм переходит к следующему шагу.

1.4 Расстояния Дамерау — Левенштейна

Расстояние Дамерау — Левенштейна может быть найдено по формуле 1.3, которая задана как

$$d_{a,b}(i,j) = \begin{cases} \max(i,j), & \text{если } \min(i,j) = 0, \\ \min\{ & \\ \quad d_{a,b}(i,j-1) + 1, & \\ \quad d_{a,b}(i-1,j) + 1, & \text{иначе} \\ \quad d_{a,b}(i-1,j-1) + m(a[i],b[j]), & \\ \quad \begin{cases} d_{a,b}(i-2,j-2) + 1, & \text{если } i,j > 1; \\ & a[i] = b[j-1]; \\ & b[j] = a[i-1] \\ \infty, & \text{иначе} \end{cases} & \\ \} & \end{cases}, \tag{1.3}$$

Формула представляет собой формулу 1.1 с поправкой на анализ транс-

позиции соседних символов. Как и в случае с рекурсивным методом, прямое применение этой формулы неэффективно по времени исполнения, то аналогично методу из 1.3 производится добавление матрицы для хранения промежуточных значений рекурсивной формулы.

1.5 Вывод

В данном разделе были рассмотрены алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна, модифицирует обычный алгоритм, учитывая возможность перестановки соседних символов. Формулы Левенштейна и Дамерау — Левенштейна для рассчета расстояния между строками заданы рекурсивно, но были рассмотрены способы, позволяющие свести задачу к иттерационному вычислению.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов нахождения расстояние Левештейна

2.1 Схема алгоритма Левенштейна

На рисунке 2.1 приведена схема рекурсивного алгоритма Левенштейна.

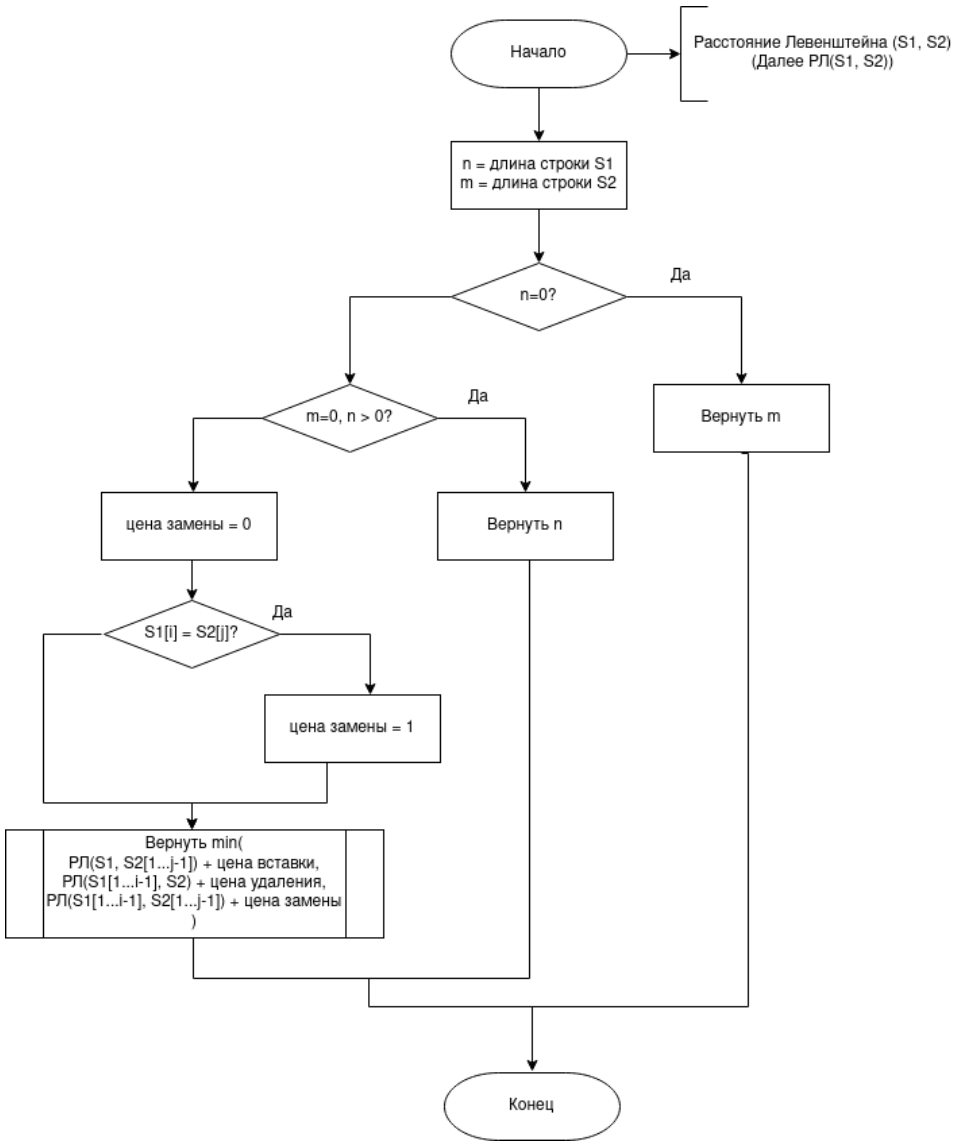

Рисунок 2.1 – Схема рекурсивного алгоритма Левенштейна

На рисунке 2.2 приведена схема рекурсивного алгоритма Левенштейна с заполнением матрицы.

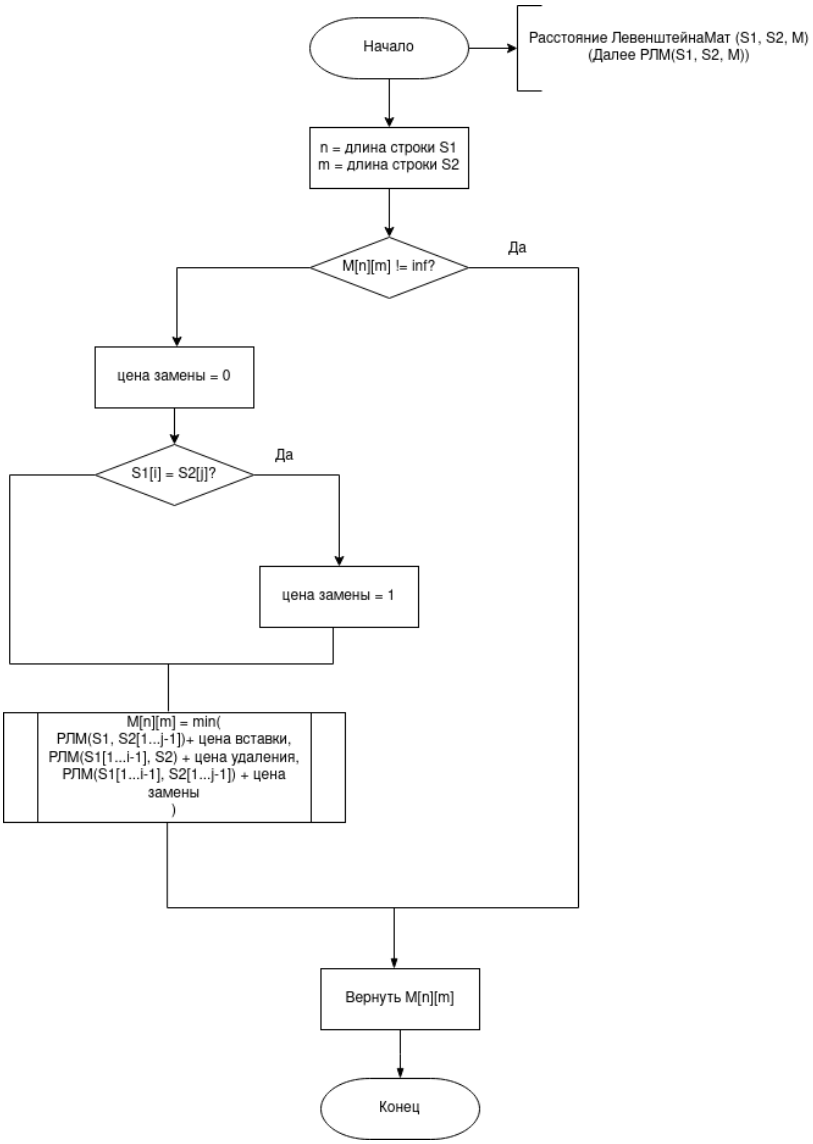

Рисунок 2.2 – Схема рекурсивного алгоритма Левенштейна с заполнением матрицы

На рисунке 2.3 приведена схема матричного алгоритма Левенштейна.

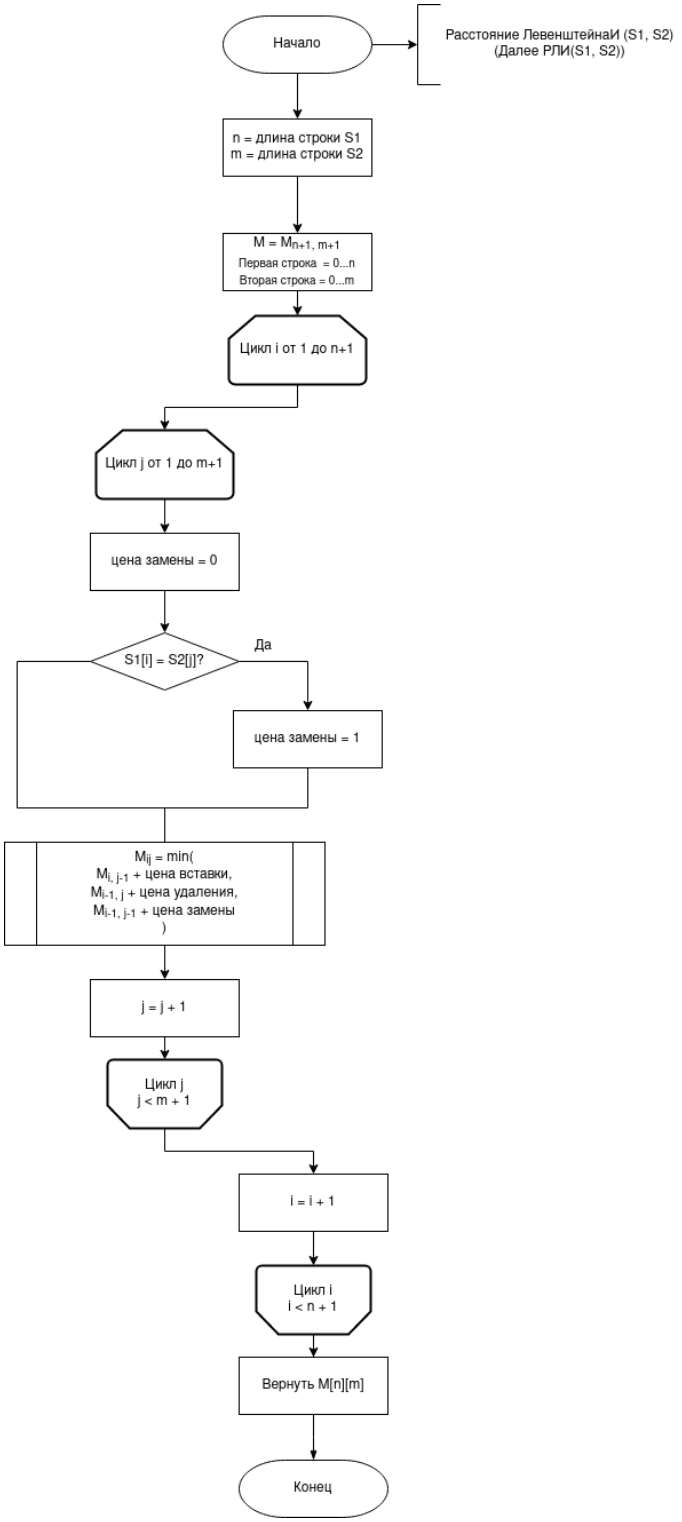

Рисунок 2.3 – Схема матричного алгоритма Левенштейна

2.2 Схема алгоритма Дамерау — Левенштейна

На рисунке 2.4 приведена схема матричного алгоритма Дамерау – Левенштейна.

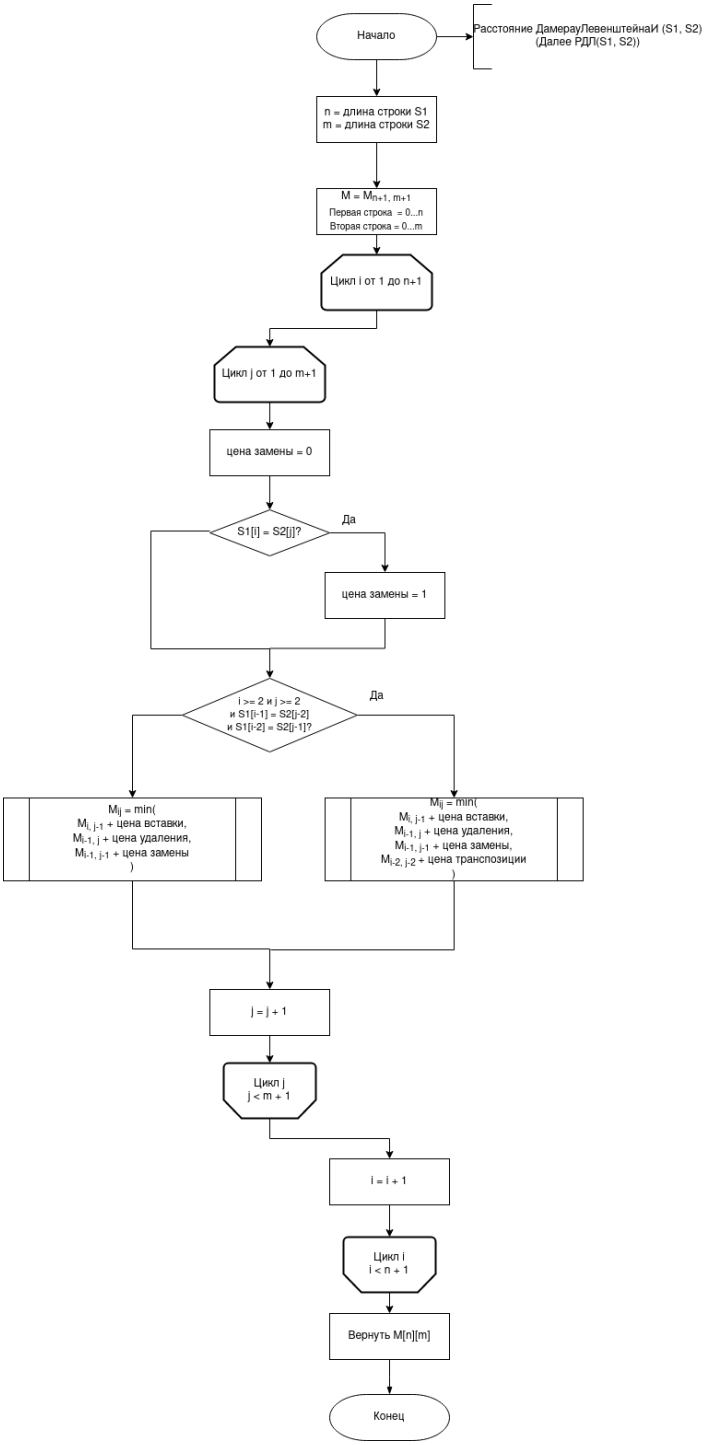

Рисунок 2.4 – Схема алгоритма Дамерау – Левенштейна

2.3 Вывод

В данном разделе на основе теоретических данных, полученных из аналитического раздела были разработаны и построены схемы исследуемых алгоритмов: Левенштейна с помощью рекурсии, Левенштейна с помощью рекурсии и кеширования, Левенштейна матричного и Дамерау-Левенштейна матричного.

Заключение

В ходе выполнения работы были выполнены все поставленные задачи и изучены методы динамического программирования на основе алгоритмов вычисления расстояния Левенштейна.

С помощью экспериментов были установлены различия в производительности алгоритмов вычисления расстояния Левенштейна. Для слов длины 10 рекурсивный алгоритм Левенштейна работает на несколько порядков медленнее (20000 раз) матричной реализации. Рекурсивный алгоритм с кешированием работает быстрее простого рекурсивного, но все еще медленнее матричного (150 раз). Если длина сравниваемых строк превышает 10, рекурсивный алгоритм становится неприемлимым для использования по времени выполнения программы. Матричная реализация алгоритма Дамерау — Левенштейна сопоставимо с алгоритмом Левенштейна. В ней добавлены дополнительные проверки, но, эти алгоритмы находятся в разном поле использования.

Теоретически было рассчитано использования памяти в каждом из алгоритмов вычисления расстояния Левенштейна. Обычные матричные алгоритмы потребляют намного больше памяти, чем рекурсивные, за счет дополнительного выделения памяти под матрицы и большее количество локальных переменных, однако позволяют обрабатывать более длинные строки.