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 дисциплине "Анализ алгоритмов"

Тема Расстояние Левенштейна

Студент Романов А.В.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение

Расстояние Левенштейна - минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Расстояние Левенштейна применяется в теории информации и компьютерной лингвистике для:

- исправления ошибок в слове
- сравнения текстовых файлов утилитой diff
- в биоинформатике для сравнения генов, хромосом и белков

Цели данной лабораторной работы:

1. Изучение метода динамического программирования на материале алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна.
2. Оценка реализаций алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна.

Задачи данной лабораторной работы:

1. Изучение алгоритмов Левенштейна и Дамерау-Левенштейна;
2. Применение метода динамического программирования для матричной реализации указанных алгоритмов;
3. Получение практических навыков реализации указанных алгоритмов: матричные и рекурсивные версии;
4. Сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);
5. Экспериментальное подтверждение различий во временно́й эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма;
6. Описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

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

Цены операций могут зависеть от вида операции (вставка (insert), удаление (delete), замена (replace) 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:

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

а функция 1.2 определена как:

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

Рекурсивный алгоритм реализует формулу 1.1. Функция D составлена из следующих соображений:

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;
4. Для перевода из строки a в строку b требуется выполнить последовательно 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:
   (a) Сумма цены преобразования строки a в b и цены проведения операции удаления, которая необходима для преобразования a' в a;
   (b) Сумма цены преобразования строки a в b и цены проведения операции вставки, которая необходима для преобразования b' в b;
   (c) Сумма цены преобразования из a' в b' и операции замены, предполагая, что a и b оканчиваются разные символы;
   (d) 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

Прямая реализация формулы 1.1 может быть малоэффективна по времени исполнения при больших i, j, т. к. множество промежуточных значений $D(i, j)$ вычисляются заново множество раз подряд. Для оптимизации нахождения расстояния Левенштейна можно использовать матрицу в целях хранения соответствующих промежуточных значений. 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

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

Формула выводится по тем же соображениям, что и формула (1.1). 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, который является модификаций первого, учитывающего возможность перестановки соседних символов. Формулы Левенштейна и Дамерау — Левенштейна для рассчета расстояния между строками задаются рекурсивно, а следовательно, алгоритмы могут быть реализованы рекурсивно или итерационно.

2 | Конструкторская часть

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов нахождения расстояние Левештейна и Дамерау - Левенштейна. На рисунках 2.1 - 2.4 представлены рассматриваемые алгоритмы.

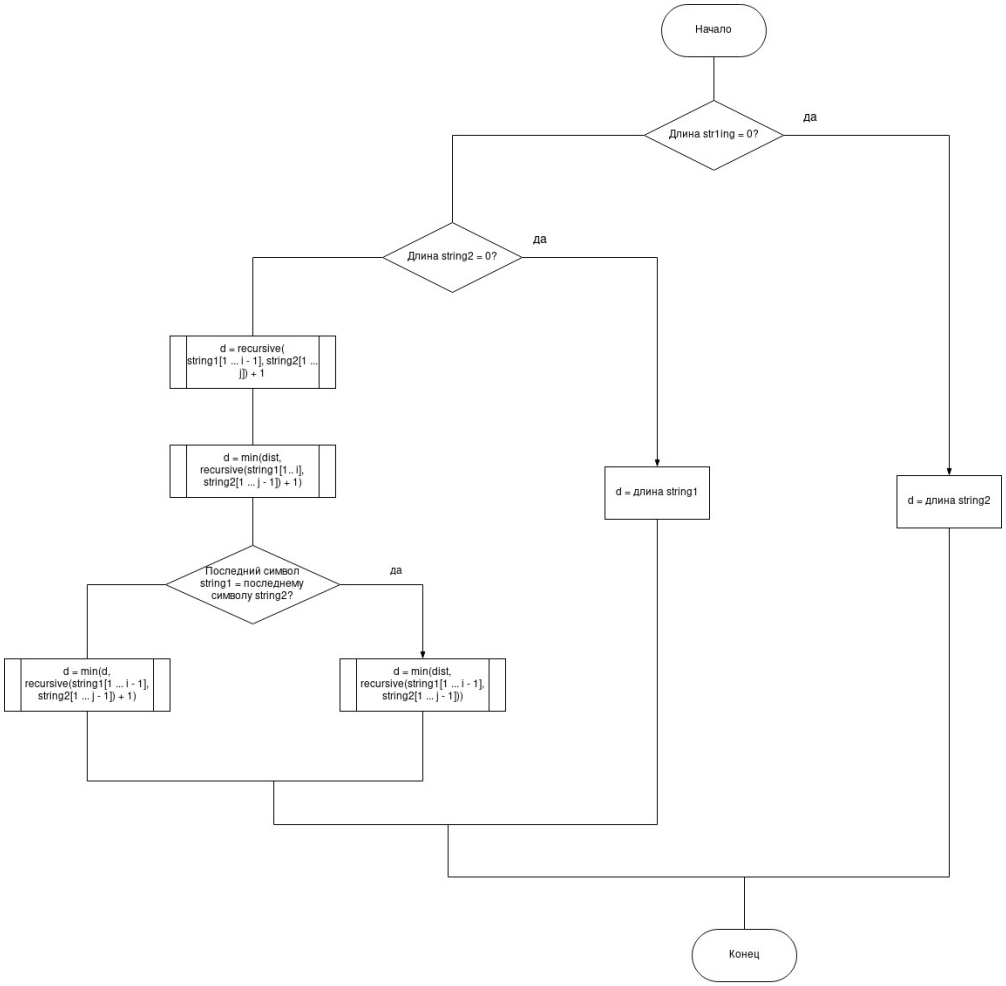

Рис. 2.1: Схема рекурсивного алгоритма нахождения расстояния Левенштейна

2.2 Вывод

На основе теоретических данных, полученные в аналитическом разделе были построены схемы иследуеммых алгоритмов.

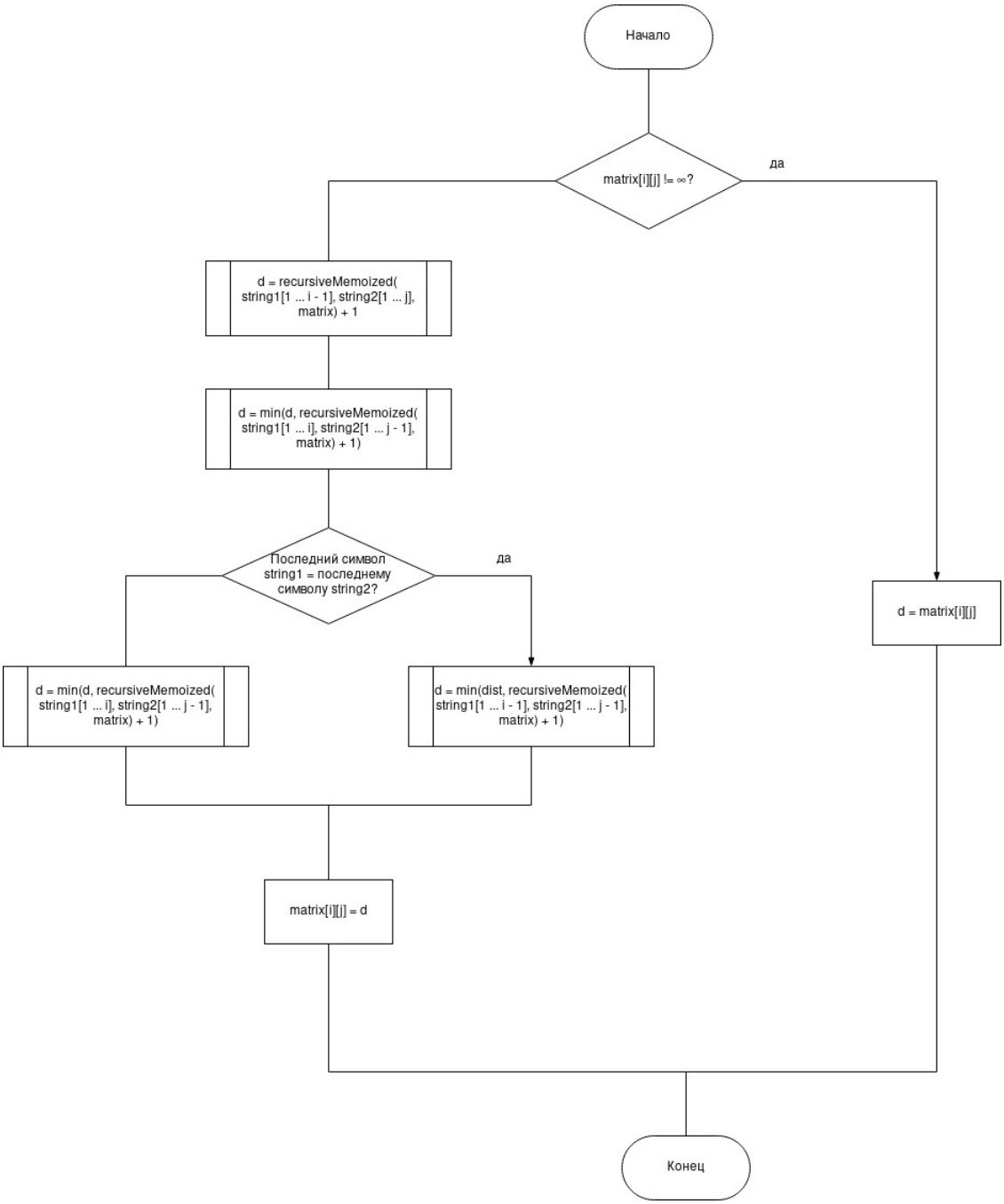

Рис. 2.2: Схема рекурсивного алгоритма с мемоизацией нахождения расстояния Левенштейна

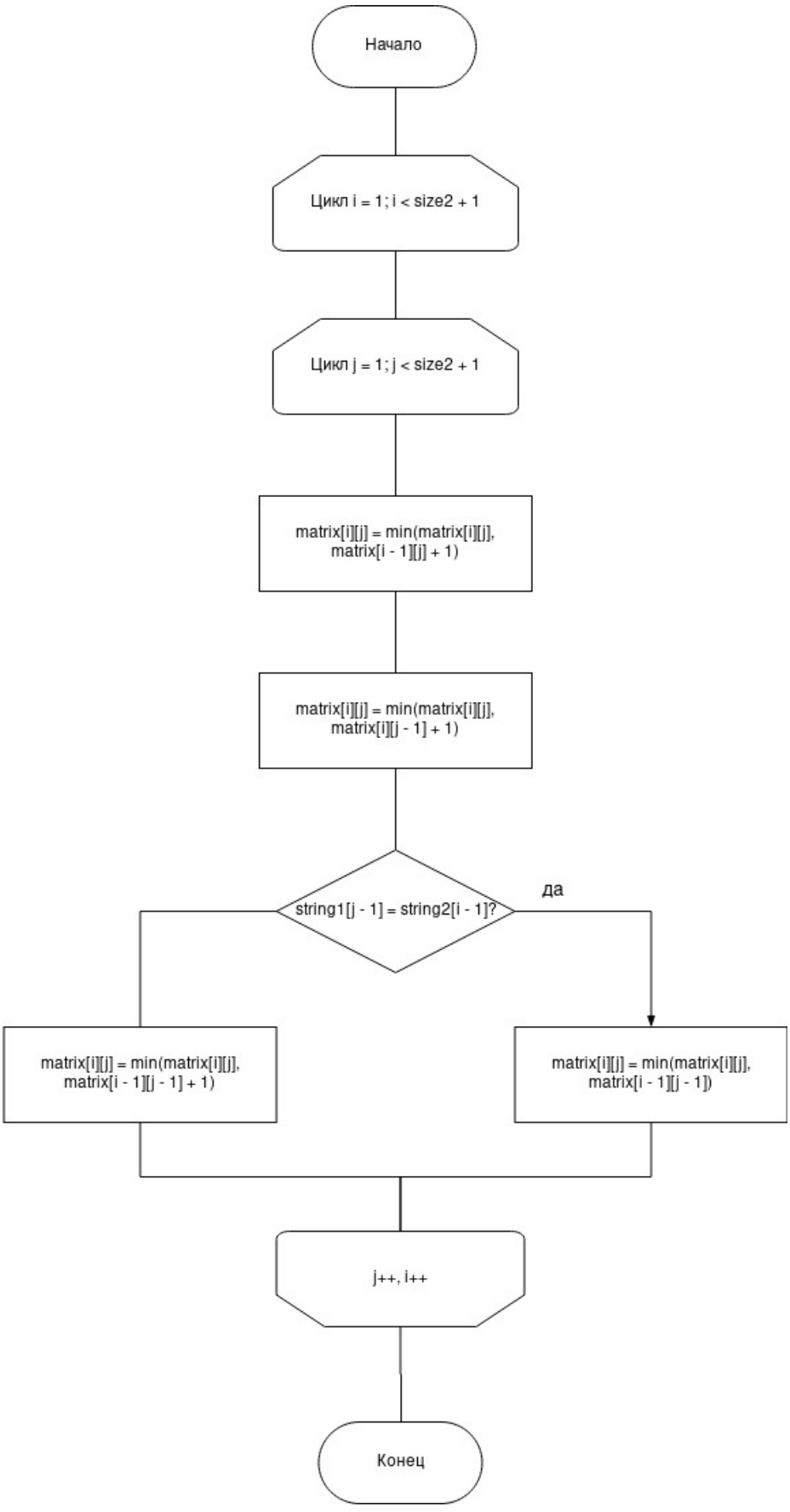

Рис. 2.3: Схема итеративного алгоритма нахождения расстояния Левенштейна

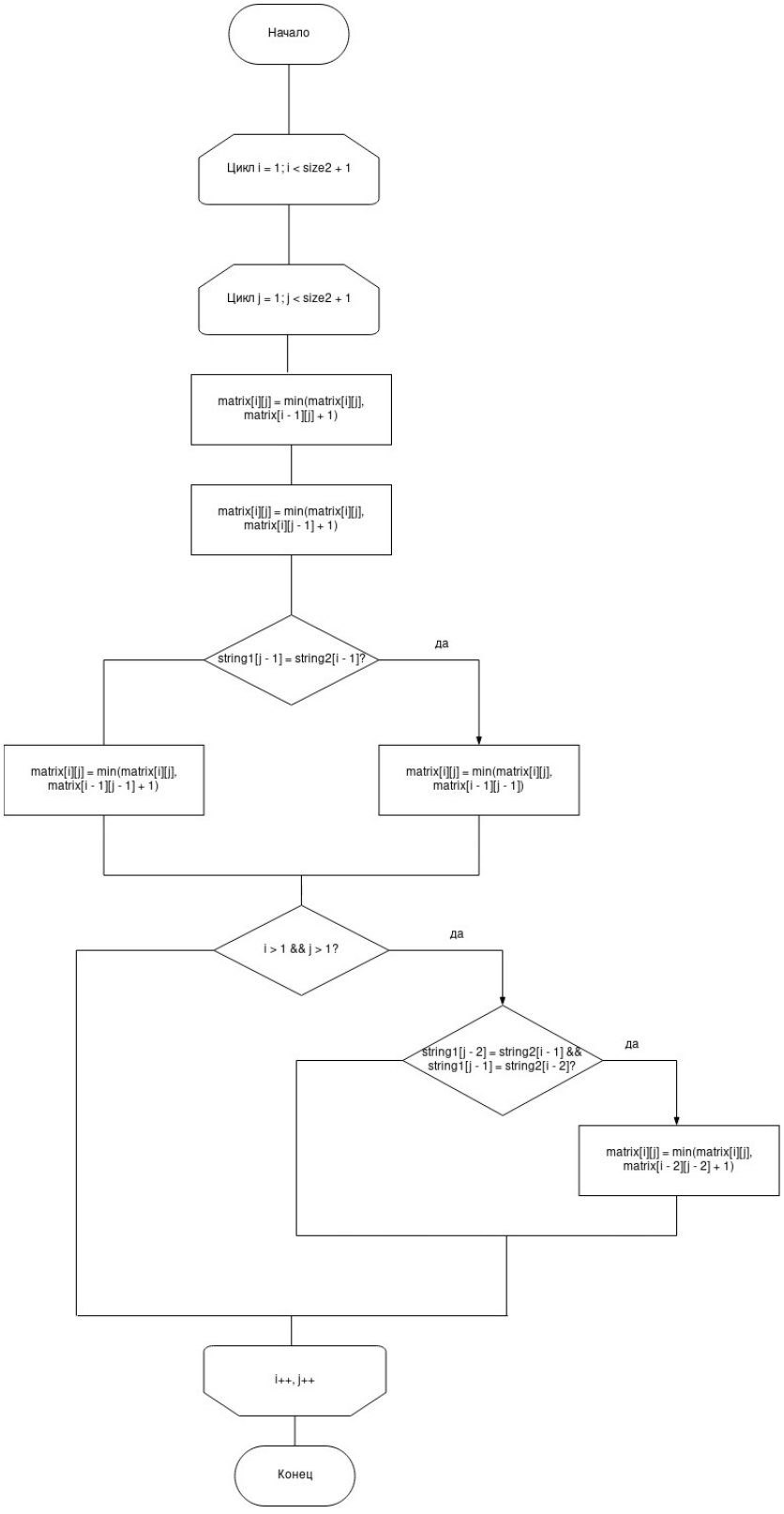

Рис. 2.4: Схема итеративного алгоритма нахождения расстояния Дамерау-Левенштейна

3 | Технологическая часть

3.1 Требование к ПО

Требования к вводу:

1. На вход подаются две строки в любой раскладке (в том числе и пустые);
2. ПО должно выводить полученное расстояние и вспомогательны матрицы;
3. ПО должно выводить потраченную память и время;

3.2 Средства реализации

Для реализации программы нахождения расстояние Левенштейна я выбрал язык программирования Haskell [2]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа программирования.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.4 приведена реализация алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна.

Листинг 3.1: Функция нахождения расстояния Левенштейна рекурсивно

```
levenshteinRecursion :: String -> String -> (Distance, Depth)
levenshteinRecursion s1 s2 = _recursion s1 s2 0
  where _recursion s1 "" n = (length s1, n)
    _recursion "" s2 n = (length s2, n)
    _recursion s1 s2 n = (score, depth) where
    (insert, curr1) = _recursion (init s1) s2 (n + 1)
    (delete, curr2) = _recursion s1 (init s2) (n + 1)
    (replace, curr3) = _recursion (init s1) (init s2) (n + 1)

    match = if last s1 == last s2 then 0 else 1
    score = min3 (insert + 1) (delete + 1) (replace + match)
    depth = max3 curr1 curr2 curr3
```

Листинг 3.2: Функция нахождения расстояние Левенштейна рекурсивно с мемоизацией

```
levenshteinMemoized :: String -> String -> (Distance, Depth)
```

```
levenshteinMemoized s1 s2 = _memoized s1 s2 matrix 0
  where matrix = fromList (length s1 + 1) (length s2 + 1) $ repeat (-1)
    _memoized s1 "" _ n = (length s1, n)
    _memoized "" s2 _ n = (length s2, n)
    _memoized s1 s2 mtr n = (score, depth) where
    score = min3 (insert + 1) (delete + 1) (replace + match)
    memoized = (getElem (length s1) (length s2) mtr, n + 1)
    new_mtr = setElem score (length s1, length s2) mtr

    (insert, curr1) = if fst memoized == -1 then _memoized (init s1) s2
       new_mtr (n + 1)
      else memoized
    (delete, curr2) = if fst memoized == -1 then _memoized s1 (init s2)
       new_mtr (n + 1)
      else memoized
    (replace, curr3) = if fst memoized == -1 then _memoized (init s1) (init
       s2) new_mtr (n + 1)
      else memoized

    match = if last s1 == last s2 then 0 else 1
    depth = max3 curr1 curr2 curr3
```

Листинг 3.3: Функция нахождения расстояния Левенштейна итеративно

```
levenshteinIterative :: String -> String -> Matrix Int
levenshteinIterative s1 s2 = fromLists $ reverse $ foldl
  (\mtr i -> if head mtr == [] then [[0..length s1]] else calcRow mtr i :
     mtr) [[]] [0..length s2]
  where calcRow mtr i = foldl (\row j ->
      row ++ if length row == 0 then [length mtr] else [
        min3 (last row + 1) (head mtr !! j + 1) $ head mtr !! (j - 1) +
          if s1 !! (j - 1) == s2 !! (i - 1) then 0 else 1]
      ) [] [0..length s1]
```

Листинг 3.4: Функция нахождения расстояния Дамерау-Левенштейна матрично

```
damerauLevenshtein :: String -> String -> Matrix Int
damerauLevenshtein s1 s2 = fromLists $ reverse $ foldl
  (\mtr i -> if head mtr == [] then [[0..length s1]] else calcRow mtr i :
     mtr) [[]] [0..length s2]
  where cell mtr row i j = min3 (last row + 1) (head mtr !! j + 1) $ head
     mtr !! (j - 1) +
      if s1 !! (j - 1) == s2 !! (i - 1) then 0 else 1
      transposition i j = i > 1 && j > 1 && s1 !! (j - 1) == s2 !! (i - 2)
         && s1 !! (j - 2) == s2 !! (i - 1)
      calcRow mtr i = foldl (\row j -> row ++ [
        if length row == 0 then length mtr
        else if transposition i j then min (cell mtr row i j) (((head $ tail
            mtr) !! i - 2) + 1)
        else cell mtr row i j]
      ) [] [0..length s1]
```

3.4 Тестовые данные

В таблице 3.1 приведены тестовые данные, на которых было протестированно разработанное ПО.

Таблица 3.1: Таблица тестовых данных

№	Первое слово	Второе слово	Ожидаемый результат	Полученный результат
1			0 0 0 0	0 0 0 0
2	kot	skat	2 2 2 2	2 2 2 2
3	kate	ktae	2 2 1 1	2 2 1 1
4	abacaba	aabcaab	4 4 2 2	4 4 2 2
5	sobaka	sboku	3 3 3 3	3 3 3 3
6	qwerty	queue	4 4 4 4	4 4 4 4
7	apple	aplpe	2 2 1 1	2 2 1 1
8		cat	3 3 3 3	3 3 3 3
9	parallels		9 9 9 9	9 9 9 9
10	русскиебуквы	русскиебукыв	2 2 2 1	2 2 2 1

3.5 Вывод

В данном разделе были разработаны исходные коды четырех алгоритмов: вычисления расстояния Левенштейна рекурсивно, с заполнением матрицы и рекурсивно с заполнением матрицы, а также вычисления расстояния Дамерау — Левенштейна с заполнением матрицы.

4 | Исследовательская часть

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

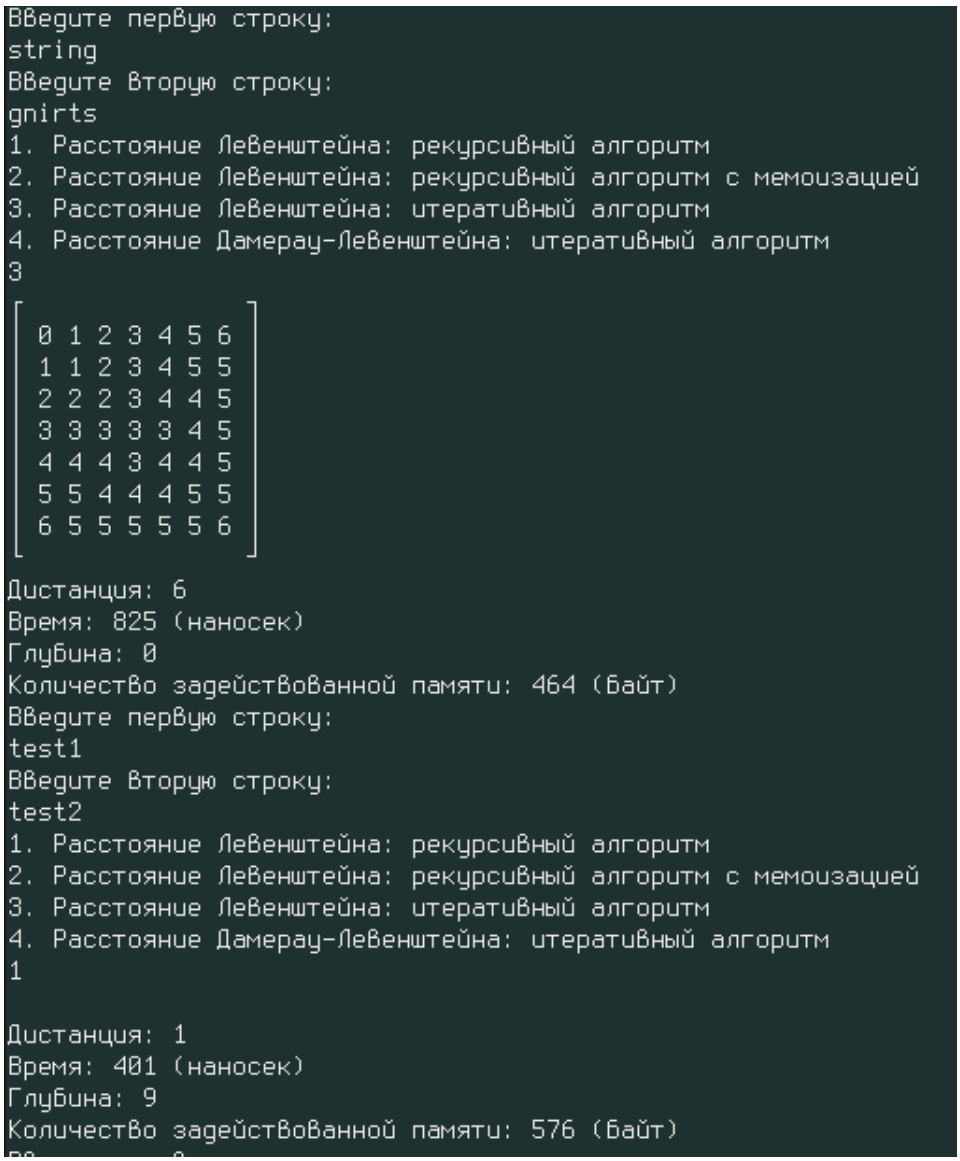

Рис. 4.1: Работа алгоритмов нахождения расстояния Левенштейна и Дамерау – Левенштейна.

4.2 Технические характеристики

Ниже приведеные технические характеристики устройства, на котором было проведенно тестирование ПО:

- Операционная система: Debian [3] Linux [4] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [5].

4.3 Время выполнения алгоритмов

Время выполнения алгоритм замерялось с помощью применения технологии профайлинга [6]. Данный инстрмуент даёт детальное описание количества вызовов и количества времени CPU, занятого каждой функцией.

В таблице 4.1. представленны замеры времени работы для каждого из алгоритмов.

Таблица 4.1: Таблица времени выполнения алгоритмов (в наносекундах)

Длина строк	Rec	MatRec	Iter	DamIter
10	30500400	1300	650	680
20	NaN	5100	2300	2500
30	NaN	11000	4800	5200
50	NaN	30000	12500	13500
100	NaN	115000	48000	55000
200	NaN	530000	249000	528000

4.4 Использование памяти

Алгоритмы нахождения расстояния Левенштейна и Дамерау — Левенштейна не отличаются друг от друга с точки зрения использования памяти.

Максимальная глубина стека вызовов при рекурсивной реализации равна сумме длин входящих строк. Поэтому, максимальный расход памяти равен:

$$(\mathcal{S}(STR_1) + \mathcal{S}(STR_2)) \cdot (2 \cdot \mathcal{S}(\text{string}) + 3 \cdot \mathcal{S}(\text{integer})), \tag{4.1}$$

где $\mathcal{S}$ — оператор вычисления размера, STR_1, STR_2 — строки, string — строковый тип, integer — целочисленный тип.

Использование памяти при итеративной реализации теоретически равно:

$$(\mathcal{S}(STR_1) + 1) \cdot (\mathcal{S}(STR_2) + 1) \cdot \mathcal{S}(\text{integer}) + 5 \cdot \mathcal{S}(\text{integer}) + 2 \cdot \mathcal{S}(\text{string}). \tag{4.2}$$

4.5 Вывод

Рекурсивная реализация алгоритма нахождения расстояния Левенштейна работает дольше итеративных реализаций, время работы этой реализации увеличивается в геометрической прогрессии. Рекурсивный алгоритм с мемоизацией проигрывает по памяти: так как в языке Haskell каждый объект является неизменяемым, каждый рекурсивный вызов создается новая вспомогательная матрица, хранящаю данные о прошлых вызовах. Стоит отметить, что, скорее всего, в других языках программирования, где присутствует передача по ссылке (например в языке программирования C++ [7]) такого бы не наблюдалось.

Заключение

В ходе проделанной работы был изучен метод динамического программирования на материале реализации алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна. Также были изучены алгоритмы поиска расстояния Левенштейна и Дамерау-Левенштейна нахождения расстояния между строками и получены практические навыки раелизации указанных алгоритмов в матричной и рекурсивных версиях, а так же в версиях с мемоизацией.

Экспериментально было подтверждено различие во временной эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработаного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк.

Теоретически было рассчитано использования памяти в каждой из реализаций алгоритмов нахождения расстояния Левенштейна и Дамерау - Левенштейна.

Литература

[1] Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады АН СССР, 1965. Т. 163. С. 845–848.

[2] The Haskell purely functional programming language [Электронный ресурс]. Режим доступа: https://haskell.org/. Дата обращения: 16.09.2020.

[3] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 20.09.2020.

[4] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux. Дата обращения: 20.09.2020.

[5] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html. Дата обращения: 20.09.2020.

[6] Profiling – Википедия [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Profiling. Дата обращения: 20.09.2020.

[7] C++ – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/C++. Дата обращения: 20.09.2020.