МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Н. Э. БАУМАНА

УДК _____
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

_____
головной исполнитель НИР

_____ _____

«_____» _____ 2019 г.

ОТЧЁТ ПО ДИСЦИПЛИНЕ "АНАЛИЗ АЛГОРИТМОВ"
О ЛАБОРАТОРНОЙ РАБОТЕ №1

по теме:

"Расстояние Левенштейна и Дамерау-Левенштейна"

(промежуточный)

Студент ИУ7-53Б ___ Пудов Дмитрий Юрьевич

Москва 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: изучение метода динамического программирования на материале алгоритмов Левенштейна и Дамерау-Левенштейна.

Постановка задачи:

— изучить метод метод динамического программирования на материала алгоритмов Левенштейна и Дамерау-Левенштейна;

— применить его;

— получить практические навыки реализации указанных алгоритмов.

1 Аналитическая часть

Начало части.

1.1 Описание алгоритмов

Какое-то описание.

2 Конструкторская часть

В данной части будут приведены схемы алгоритмов Левенштейна в рекурсивной и матричной реализации и Дамерау-Левенштейна.

2.1 Разработка алгоритмов

Далее указаны разработанные схемы алгоритмов Левенштейна и Дамерау-Левенштейна. Будем считать, что известны следующие функции: определения длины строки, поиска максимума и минимума среди нескольких чисел. Для матричных реализаций требуется наличие функции, динамически выделяющей память.

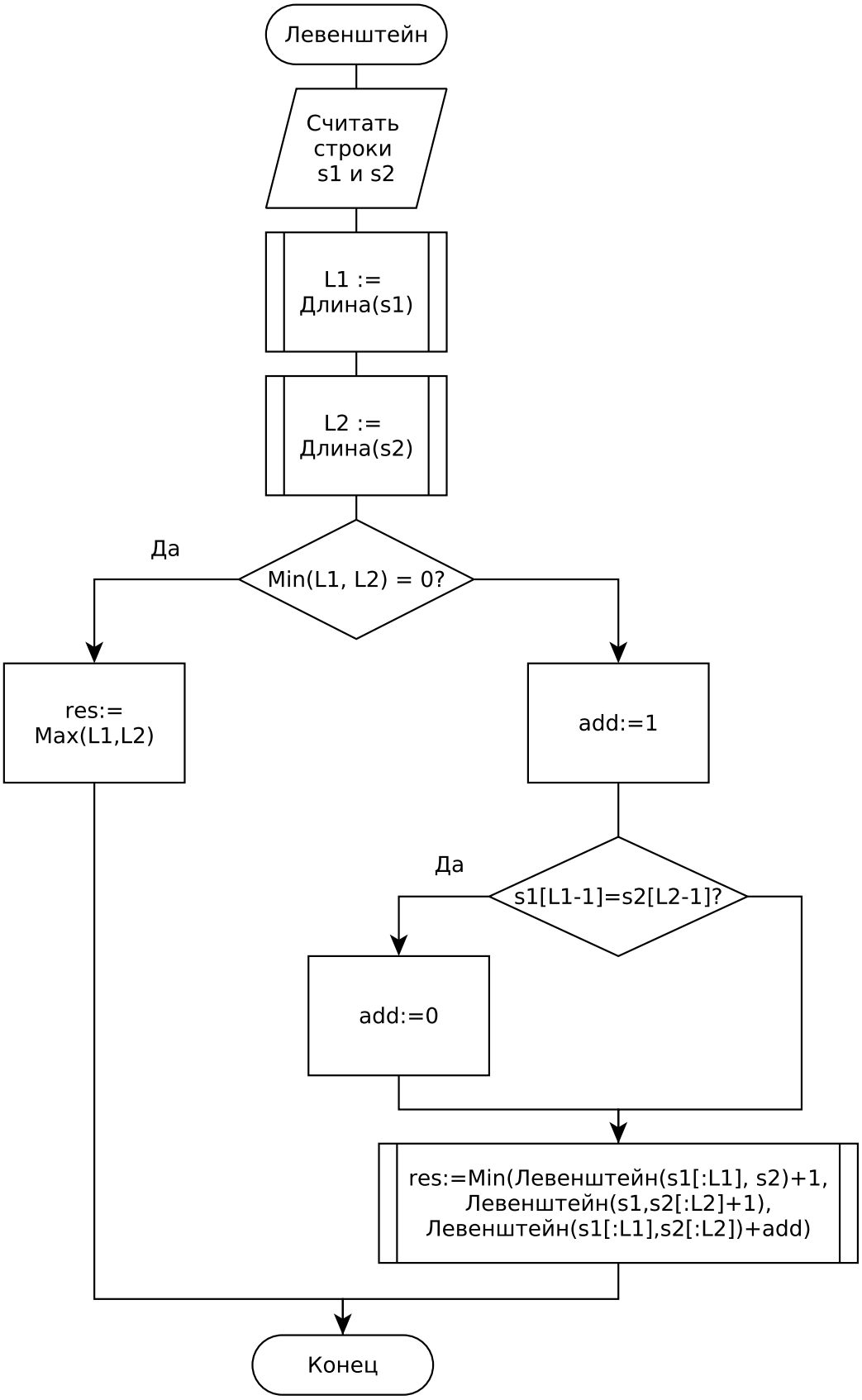

Рисунок 2.1 — Схема рекурсивного алгоритма Левенштейна

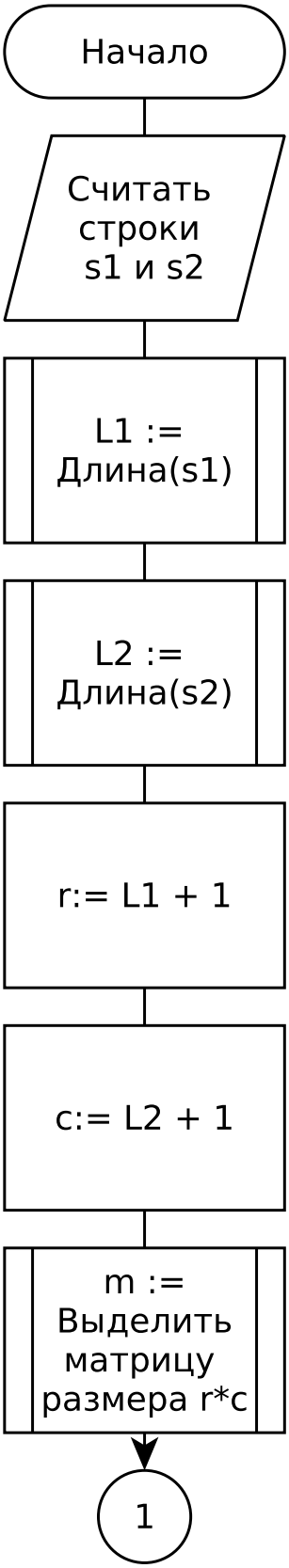

Рисунок 2.2 — Схема матричного алгоритма Левенштейна. Часть 1.

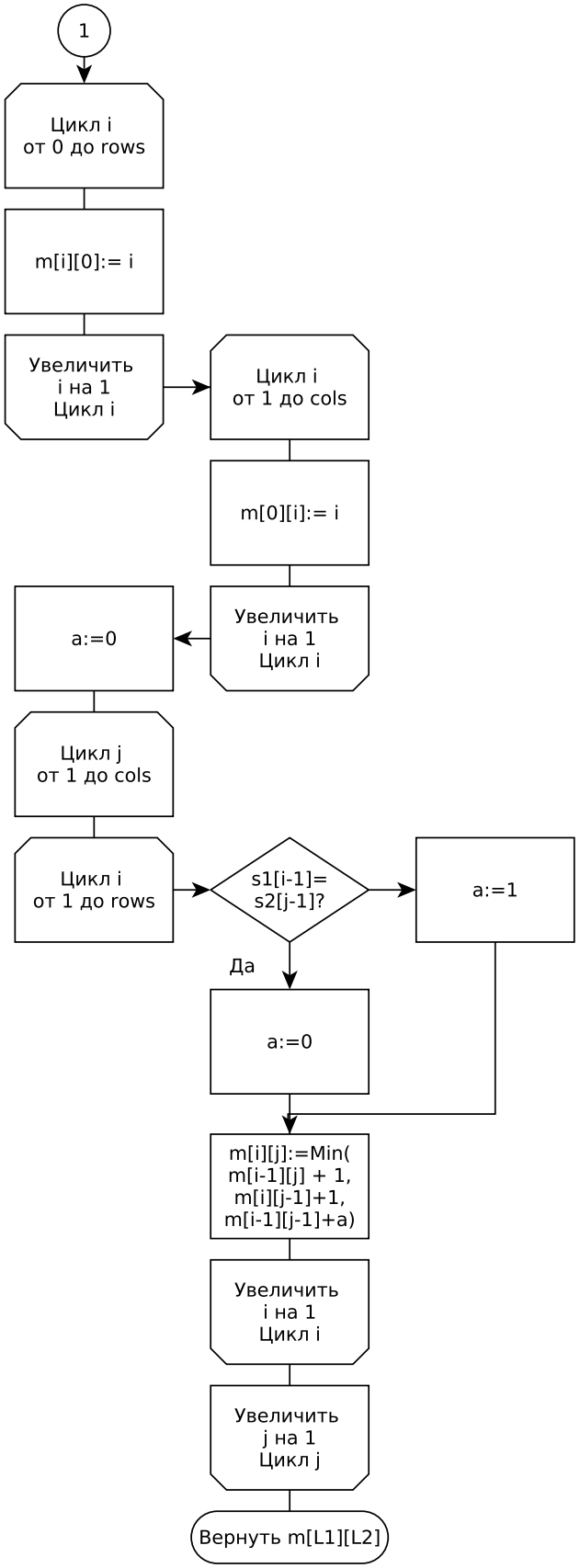

Рисунок 2.3 — Схема матричного алгоритма Левенштейна. Часть 2.

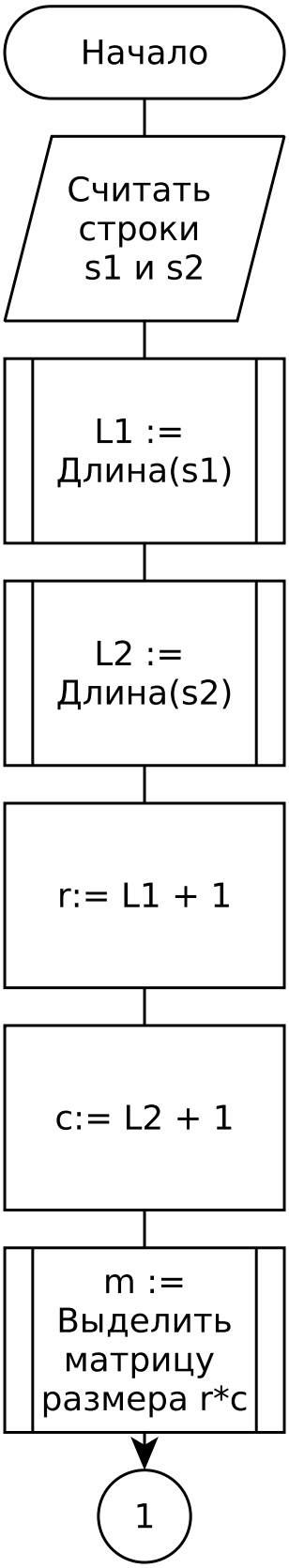

Рисунок 2.4 — Схема алгоритма Дамерау-Левенштейна. Часть 1.

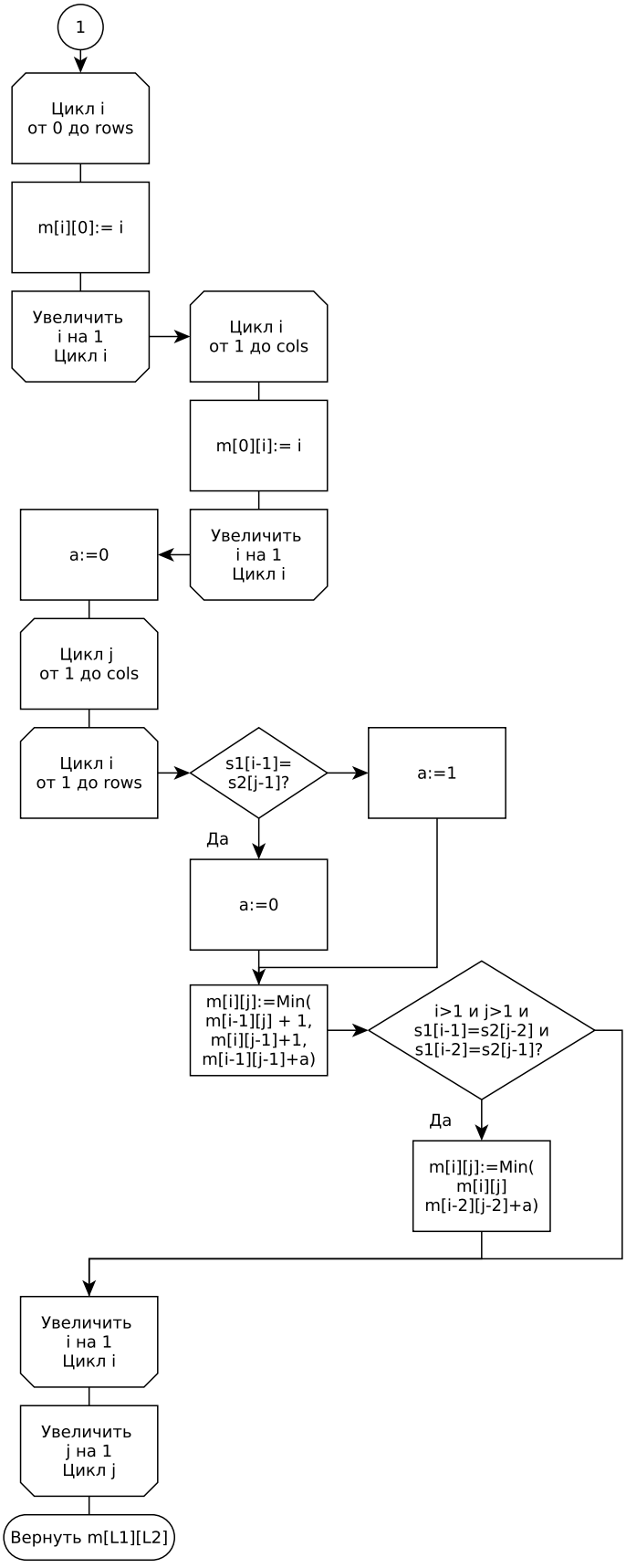

Рисунок 2.5 — Схема алгоритма Дамерау-Левенштейна. Часть 2.

2.2 Сравнительный анализ рекурсивной и нерекурсивной реализаций

Какой-то текст

3 Технологическая часть

3.1 Требования к программному обеспечению

3.2 Средства реализации

3.3 Листинг кода

В данном разделе приведены листинги реализаций ранее указанных алгоритмов на языке Go.

```
1  func LevenshteinDamerau(first string, second string) (int, [][] int) {
2      lenFirst := len(first)
3      lenSecond := len(second)
4      rows := lenFirst + 1
5      cols := lenSecond + 1
6      matrix := allocateMatrix(rows, cols)
7
8      for i := 0; i < rows; i++ {
9          matrix[i][0] = i
10     }
11     for i := 1; i < cols; i++ {
12         matrix[0][i] = i
13     }
14
15     add := 0
16     for j := 1; j < cols; j++ {
17         for i := 1; i < rows; i++ {
18             if first[i-1] == second[j-1] {
19                 add = 0
20             } else {
21                 add = 1
22             }
23             matrix[i][j] = MinThree(matrix[i-1][j]+1,
24                 matrix[i][j-1]+1,
25                 matrix[i-1][j-1]+add)
26
27             if i > 1 && j > 1 && first[i-1] == second[j-2] && first[i-2] ==
                   second[j-1] {
28                 matrix[i][j] = Min(matrix[i][j], matrix[i-2][j-2]+add)
29             }
30         }
31     }
32
33     return matrix[lenFirst][lenSecond], matrix
34 }
```

```
func LevenshteinIterative(first string, second string) (int, [][]int) {
    lenFirst := len(first)
    lenSecond := len(second)
    rows := lenFirst + 1
    cols := lenSecond + 1
    matrix := allocateMatrix(rows, cols)

    for i := 0; i < rows; i++ {
        matrix[i][0] = i
    }
    for i := 1; i < cols; i++ {
        matrix[0][i] = i
    }

    add := 0
    for j := 1; j < cols; j++ {
        for i := 1; i < rows; i++ {
            if first[i-1] == second[j-1] {
                add = 0
            } else {
                add = 1
            }
            matrix[i][j] = MinThree(matrix[i-1][j]+1,
                matrix[i][j-1]+1,
                matrix[i-1][j-1]+add)
        }
    }

    return matrix[lenFirst][lenSecond], matrix
}
```

```
func LevenshteinRecursive(first string, second string) int {
    lenFirst := len(first)
    lenSecond := len(second)
    if Min(lenFirst, lenSecond) == 0 {
        return Max(lenFirst, lenSecond)
    } else {
        add := 1
        if first[lenFirst-1] == second[lenSecond-1] {
            add = 0
        }

        return MinThree(
            LevenshteinRecursive(first[:lenFirst-1], second)+1,
            LevenshteinRecursive(first, second[:lenSecond-1])+1,
```

```
15                 LevenshteinRecursive(first[:lenFirst-1],
                       second[:lenSecond-1])+add)
16         }
17     }
```

```
1  func levenshteinRecursiveModule(first, second string, result, minval int)
       int {
2      lenFirst := len(first)
3      lenSecond := len(second)
4      if result >= minval {
5          return minval
6      } else if lenFirst == 0 {
7          return result +lenSecond
8      } else if lenSecond == 0 {
9          return result + lenFirst
10     } else {
11         add := 1
12         if first[lenFirst-1] == second[lenSecond-1] {
13             add = 0
14         }
15         r1 := levenshteinRecursiveModule(first[:lenFirst-1],
               second[:lenSecond-1], result+add, minval)
16         r2 := levenshteinRecursiveModule(first, second[:lenSecond-1],
               result+1, Min(r1, minval))
17         r3 := levenshteinRecursiveModule(first[:lenFirst-1], second,
               result+1, MinThree(r1, r2, minval))
18         return MinThree(r1, r2, r3)
19     }
20 }
21
22 func LevenshteinRecursiveOptimized(first string, second string) int {
23     minval := Max(len(first), len(second))
24     result := 0
25     return levenshteinRecursiveModule(first, second, result, minval)
26 }
```

3.4 Описание тестирования

4 Экспериментальная часть

4.1 Примеры работы

4.2 Результаты тестирования

4.3 Постановка эксперимента по замеру времени и памяти

4.4 Сравнительный анализ на материале экспериментальных данных

ЗАКЛЮЧЕНИЕ