Анализ Алгоритмов

Лабораторная работа №1

Юмаев Артур Русланович

ИУ7-55

Преподаватель: Волкова Л. Л.

Москва, 2019

Оглавление

[Введение 3](#_Toc22593391)

[Аналитическая часть 3](#_Toc22593392)

[Расстояние Левенштейна 3](#_Toc22593393)

[Расстояние Домерау-Левенштейна 4](#_Toc22593394)

[Конструкторская часть 5](#_Toc22593395)

[Схема алгоритма Левенштейна методом динамического программирования 5](#_Toc22593396)

[Схема алгоритма Левенштейна рекурсивным методом 7](#_Toc22593397)

[Схема алгоритма Домерау-Левенштейна методом динамического программирования 8](#_Toc22593398)

[Схема алгоритма Домерау-Левенштейна рекурсивным методом 11](#_Toc22593399)

[Технологическая часть 12](#_Toc22593400)

[Алгоритм Левенштейна 12](#_Toc22593401)

[Алгоритм Домерау-Левенштейна 13](#_Toc22593402)

[Алгоритм Левенштейна рекурсивным способом 13](#_Toc22593403)

[Алгоритм Домерау-Левенштейна рекурсивным способом 14](#_Toc22593404)

[Исследовательская часть 14](#_Toc22593405)

[Заключение 15](#_Toc22593406)

# Введение

Данная работа посвящена исследованию “Редакторского алгоритма” или алгоритма Левенштейна и его дополнения – алгоритма Домерау-Левенштейна. Исследование является целью изучить математическую часть алгоритмов и запрограммировать их в нескольких вариантах (матричном и рекурсивном). Также будет проведена оценка алгоритмов по скорости работы и подготовлено их сравнение между собой.

# Аналитическая часть

Ставится задача поиска минимального расстояния D(S1, S2) между двумя словами S1 и S2. M = |S1|, N = |S2|.

Расстояние Левенштейна

Расстояние Левенштейна - расстояние между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Для расстояние Левенштейна рассмотрим граничные случаи:

D(char, 0) = 1 // Delete

D(0, char) = 1 // Insert

D(0, 0) = 0 // Match,

Где D – расстояние,

char – любой символ

Расстояние Левенштейна определяется как: D(S1[1…L(S1)], S2[1…L[S2]])

Общий случай для рекурсивного подхода:

D(S1[1…i], S2[1…j]) = min(

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]) + {0 if S1[i] = S2[j] else 1}

)

Общий случай для матричного подхода:

A = (L(S1) + 1) x (L(S2) + 1) – матрица

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | … | L(S1) |
| 1 | … | … | … |
| … | … | … | … |
| L(S2) | … | … | … |

Для I > 0, j > 0:

|  |  |
| --- | --- |
| X | Y |
| Z | Q |

Q = min(

Y + 1,

Z + 1,

X + {0 if S1[i] = S2[j] else 1}

)

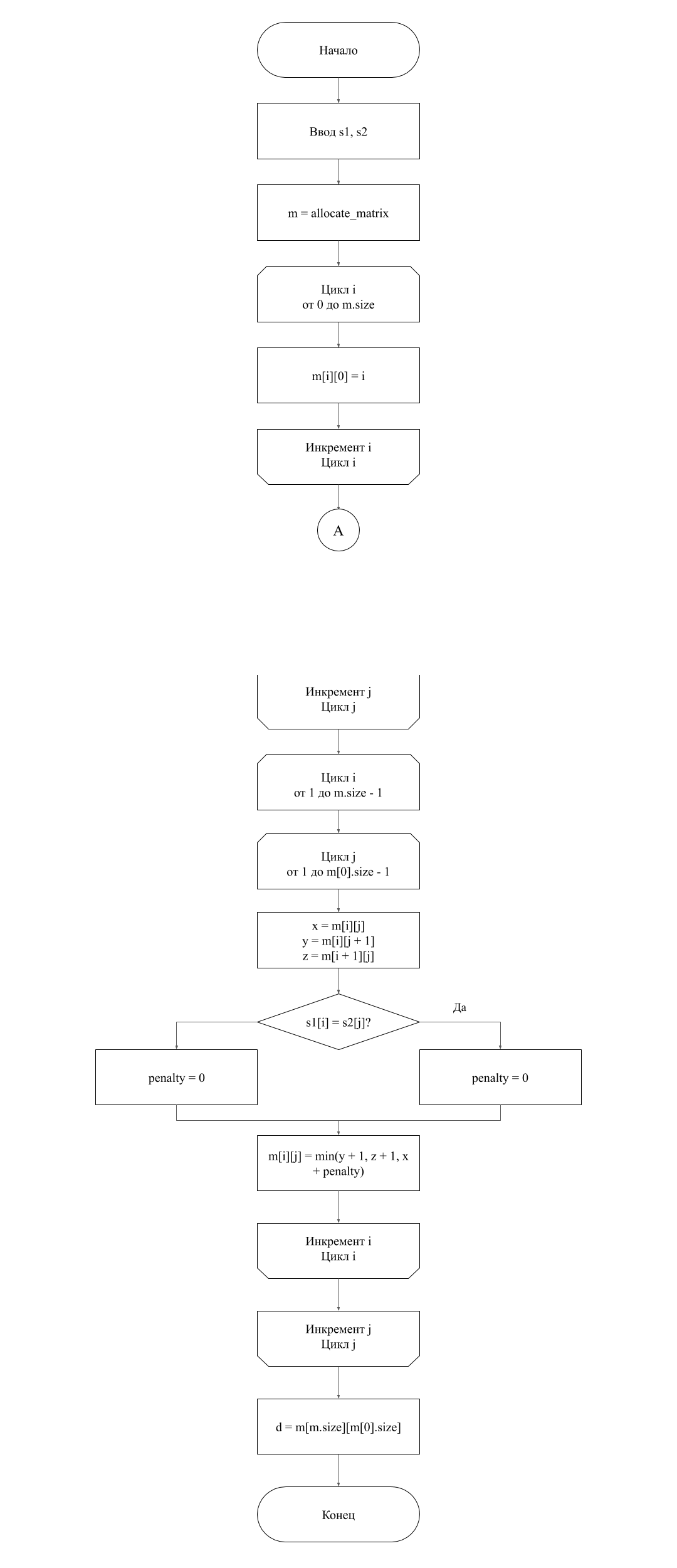
## Расстояние Домерау-Левенштейна

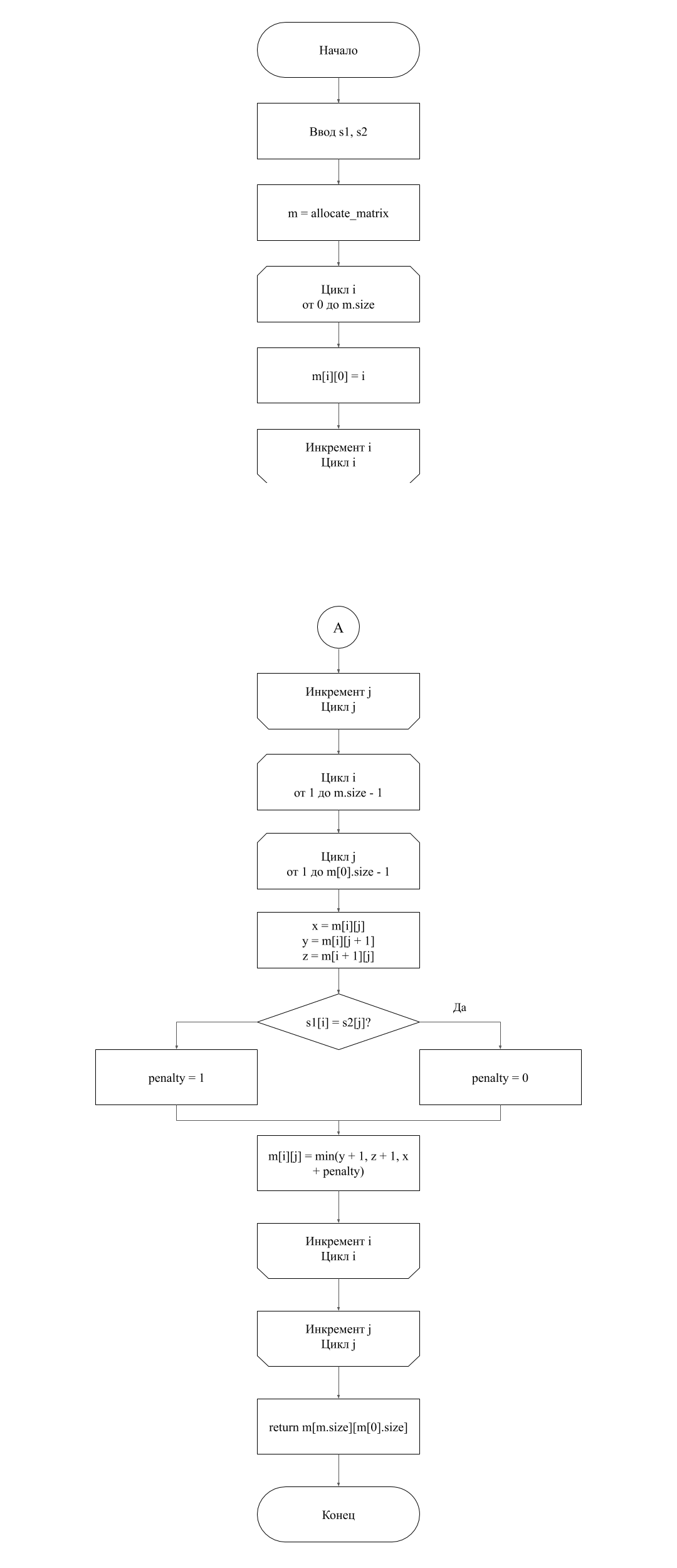
Алгоритм Домерау-Левенштейна заключается в том, что к аргументам функции min(…) добавляется еще одно условие:

+ {1 if S1[i] = S2[j - 1] & S2[j] = S1[i - 1] else 0}

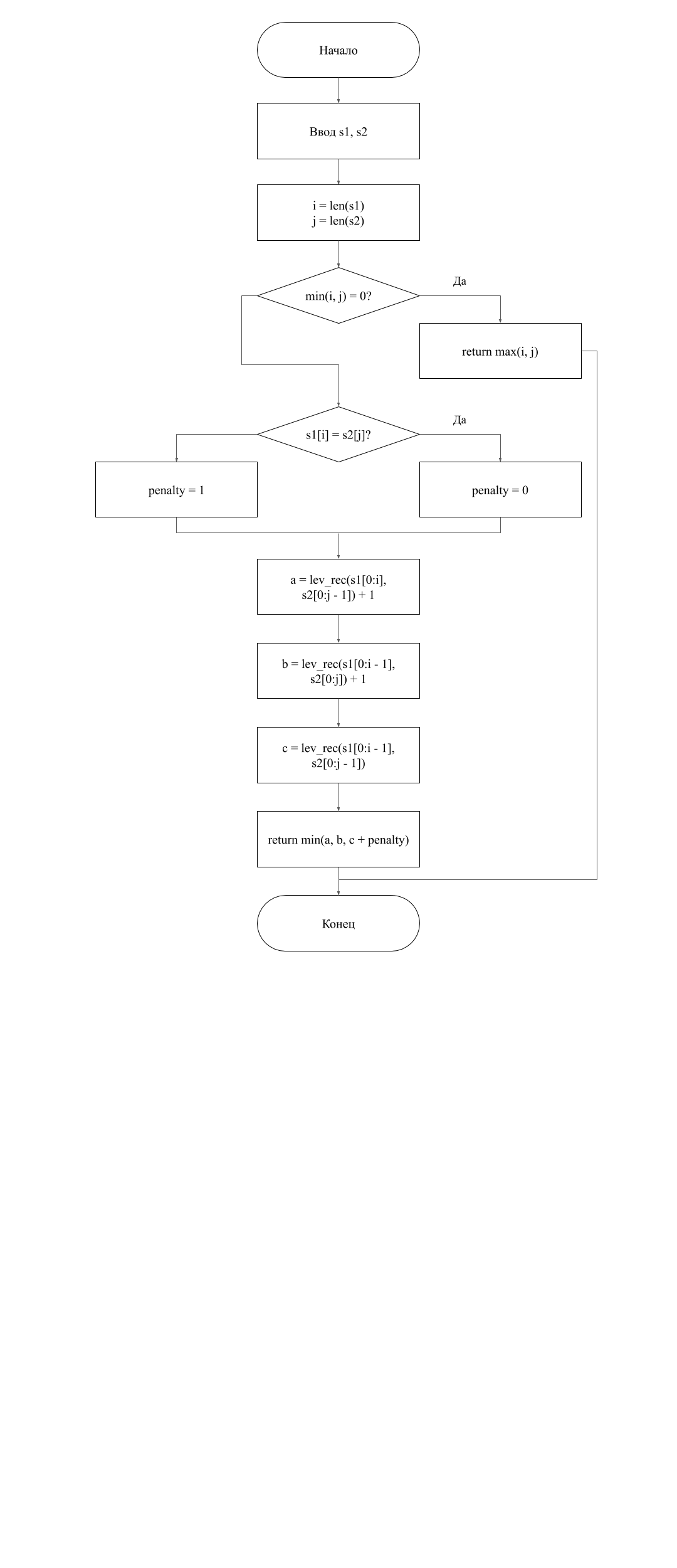
# Конструкторская часть

## Схема алгоритма Левенштейна методом динамического программирования

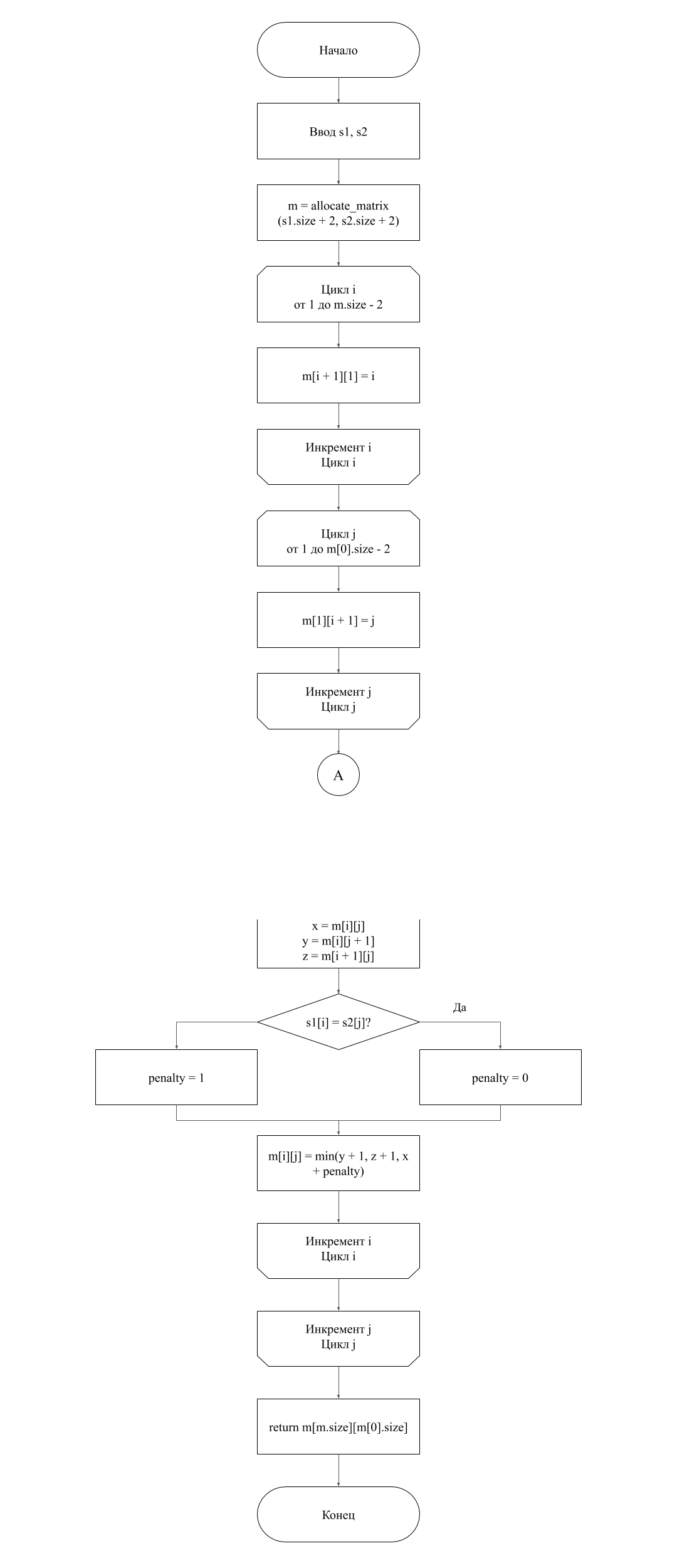


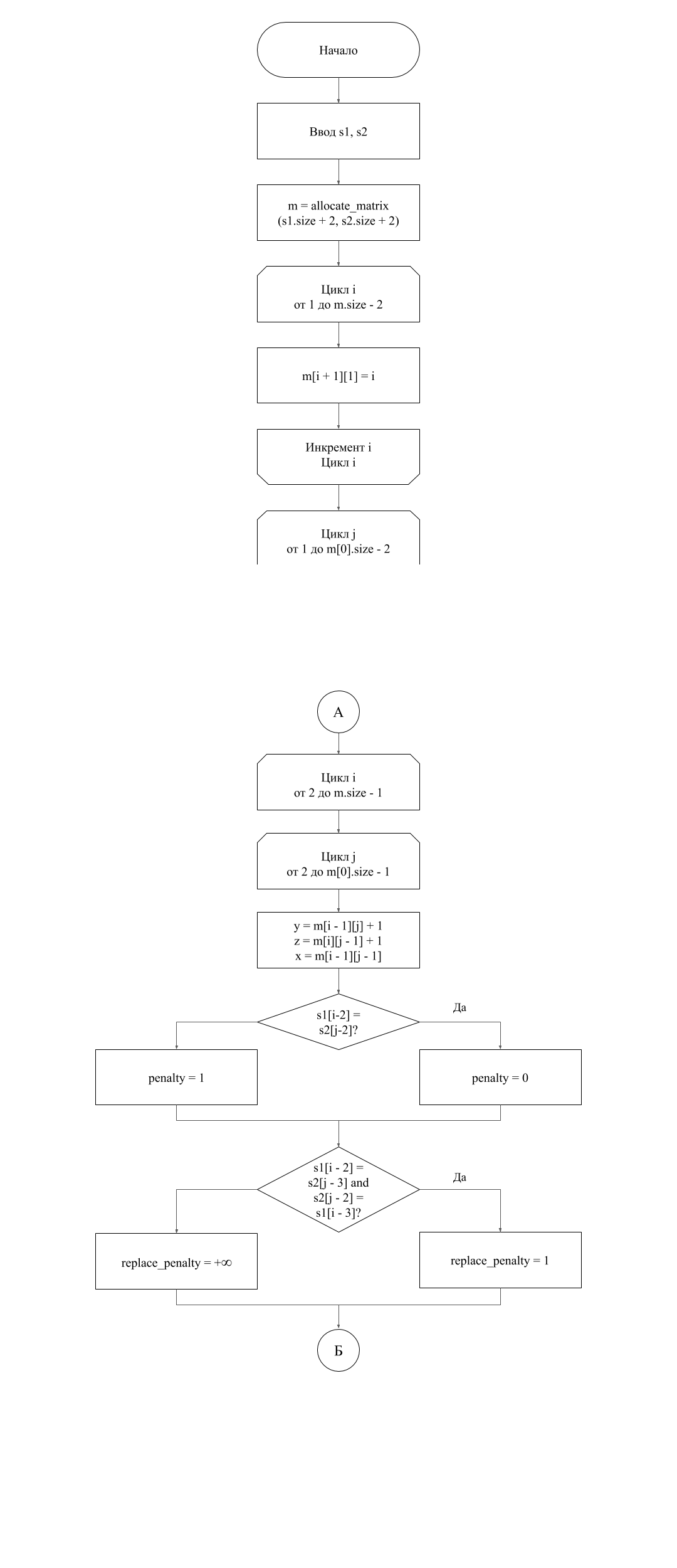


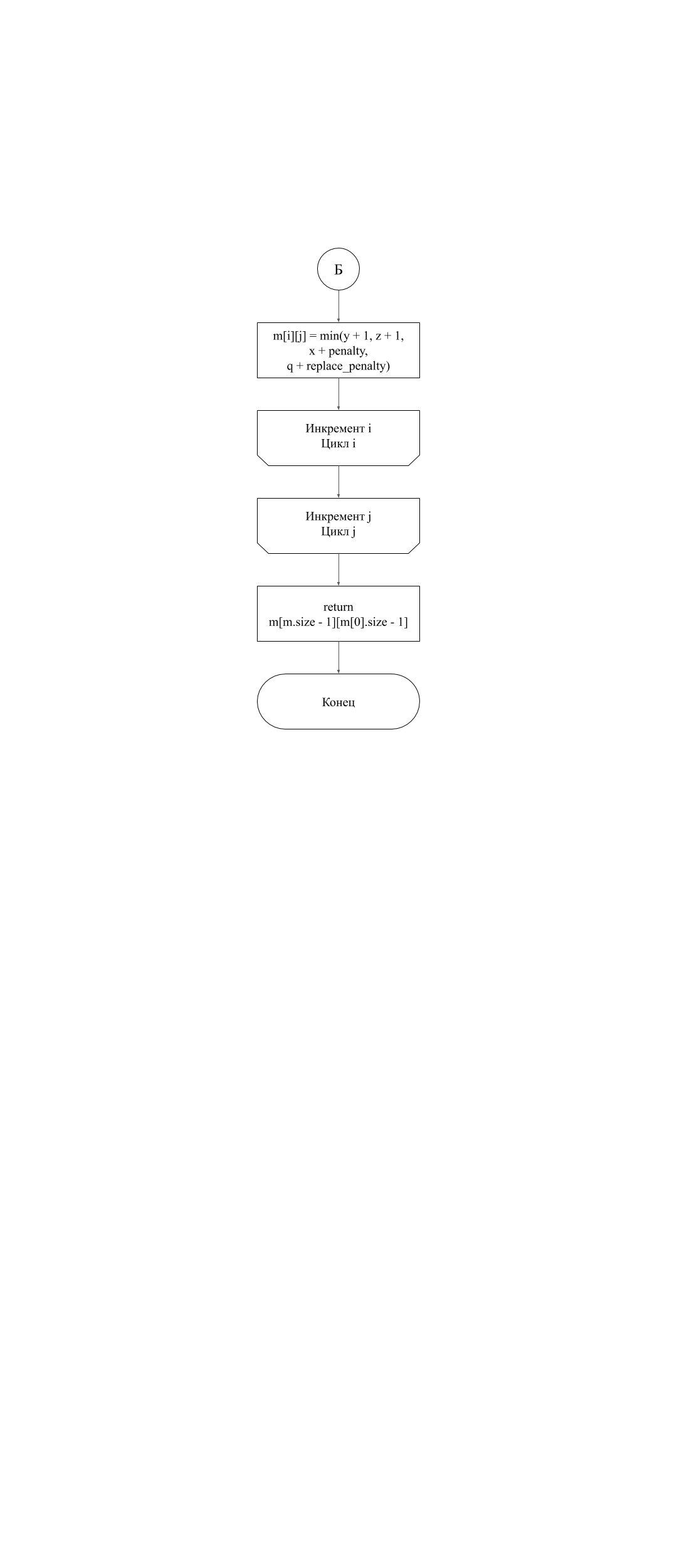
## Схема алгоритма Левенштейна рекурсивным методом



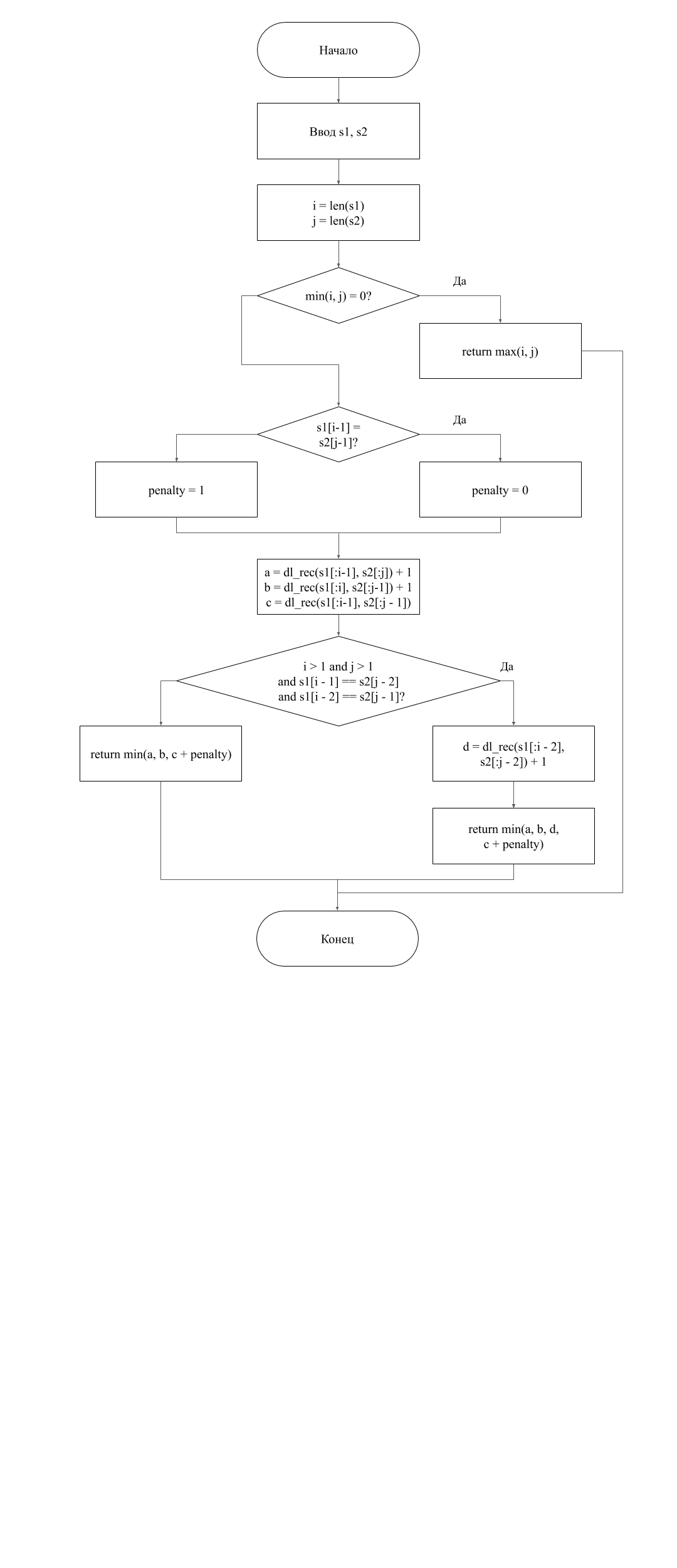
## Схема алгоритма Домерау-Левенштейна методом динамического программирования







## Схема алгоритма Домерау-Левенштейна рекурсивным методом



# Технологическая часть

В качестве языка программирования был выбран Python из-за того, что для него существуют простые и быстрые библиотеки для работы с матрицами, которые упрощают процесс разработки. Библиотеки для работы с матрицами – Pandas и NumPy. Статистические данные отображаются с помощью библиотеки Matplotlib.

## Алгоритм Левенштейна

|  |
| --- |
| def levenshtein\_matrix(s1, s2, return\_matrix=False):  matrix = alloc\_matrix(s1, s2)  # 1. Simple cases  for i in range(matrix.shape[0]): # Fill the first column  matrix[i][0] = i  for i in range(matrix.shape[1]): # Fill the first row  matrix[0][i] = i  # 2. i > 0, j > 0  for i in range(1, matrix.shape[0]):  for j in range(1, matrix.shape[1]):  x = matrix[i - 1][j - 1]  y = matrix[i - 1][j]  z = matrix[i][j - 1]  matrix[i][j] = min(y + 1,  z + 1,  x + (0 if s1[i - 1] == s2[j - 1]  else 1))  distance = matrix[matrix.shape[0] - 1][matrix.shape[1] - 1]    if return\_matrix:  return matrix, distance  else:  return distance |

## Алгоритм Домерау-Левенштейна

|  |
| --- |
| def domerau\_levenshtein\_matrix(s1, s2, return\_matrix=False):  matrix = np.zeros((len(s1) + 2, len(s2) + 2))  # 1. Simple cases  for i in range(1, matrix.shape[0] - 1):  matrix[i + 1][1] = i  for i in range(1, matrix.shape[1] - 1):  matrix[1][i + 1] = i  for i in range(2, matrix.shape[0]):  for j in range(2, matrix.shape[1]):  y = matrix[i - 1][j] + 1  z = matrix[i][j - 1] + 1  x = matrix[i - 1][j - 1] + \  (0 if s1[i - 2] == s2[j - 2] else 1)  q = matrix[i - 2][j - 2] + \  (1 if s1[i - 2] == s2[j - 3] and  s2[j - 2] == s1[i - 3]  else np.inf)  matrix[i][j] = min(y, z, x, q)  distance = matrix[matrix.shape[0] - 1][matrix.shape[1] - 1]  if return\_matrix:  return matrix, distance  else:  return distance |

## Алгоритм Левенштейна рекурсивным способом

|  |
| --- |
| def levenshtein\_rec(s1, s2):  i, j = len(s1), len(s2)  if min(i, j) == 0:  return max(i, j)  return min(levenshtein\_rec(s1[0:i], s2[0:j - 1]) + 1,  levenshtein\_rec(s1[0:i - 1], s2[0:j]) + 1,  levenshtein\_rec(s1[0:i - 1], s2[0:j - 1]) + \  (0 if s1[i - 1] == s2[j - 1] else 1)) |

## Алгоритм Домерау-Левенштейна рекурсивным способом

|  |
| --- |
| def domerau\_levenshtein\_rec(s1, s2):  i = len(s1)  j = len(s2)  if min(i, j) == 0:  return max(i, j)  elif (i > 1 and j > 1 and s1[i - 1] == s2[j - 2] and s1[i - 2] == s2[j - 1]):  return min(  domerau\_levenshtein\_rec(s1[:i - 1], s2[:j]) + 1,  domerau\_levenshtein\_rec(s1[:i], s2[:j - 1]) + 1,  domerau\_levenshtein\_rec(s1[:i - 2], s2[:j - 2]) + 1,  domerau\_levenshtein\_rec(s1[:i - 1], s2[:j - 1]) + \  (0 if s1[i - 1] == s2[j - 1] else 1))  else:  return min(  domerau\_levenshtein\_rec(s1[:i - 1], s2[:j]) + 1,  domerau\_levenshtein\_rec(s1[:i], s2[:j - 1]) + 1,  domerau\_levenshtein\_rec(s1[:i - 1], s2[:j - 1]) + \  (0 if s1[i - 1] == s2[j - 1] else 1)) |

# Исследовательская часть

На рис. 1 приведено сравнение алгоритмов Левенштейна и Домерау-Левенштейна по времени работы в зависимости от длины слов. Опыт показал, что алгоритм Домерау-Левенштейна работает медленнее, что было логично, из-за дополнительной операции сравнения букв на неверный порядок. Для каждой пары слов опыт проводился 100 раз и далее считалось среднее время, затраченное на поиск расстояния для каждой конкретной пары. Весь опыт занял около 50 минут.

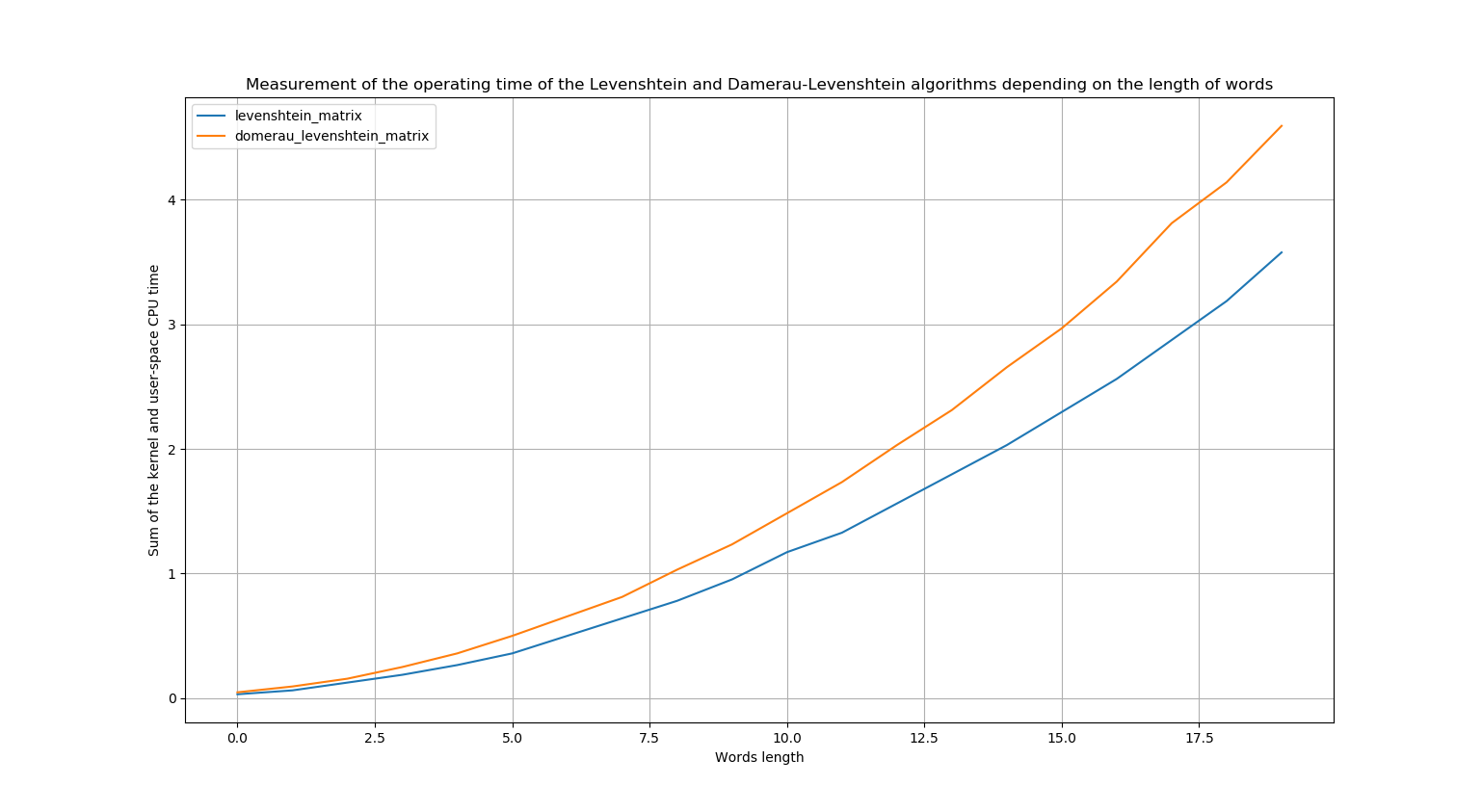


Рис. 1

# Заключение

В результате выполнения данной работы рассмотрены и изучены понятия расстояния Левенштейна и расстояния Домерау-Левенштейна. Реализованы два варианта алгоритма нахождения расстояния Левенштейна (рекурсивного и не рекурсивного вида). Сравнены их временные характеристики как следствие проведённых экспериментов. Реализован алгоритм нахождения расстояния Домерау-Левенштейна. Были сделаны выводы об эффективности по времени рекурсивного и не рекурсивного вариантов алгоритмов. Применение рекурсивного варианта алгоритма нахождения расстояния Левенштейна неэффективно по времени. Рекомендуется использовать не рекурсивный алгоритм.