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»

Отчёт по лабораторной работе № 2 по курсу «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Гаврилюк В. А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель лабораторной работы — исследование методов умножения матриц с помощью:

– стандартного алгоритма;

– алгоритма Винограда;

– оптимизированного алгоритма Винограда.

Для достижения поставленной цели необходимо выполнить следующие задачи:

– построить схемы для стандартного алгоритма умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда;

– выполнить оценку трудоёмкости данных алгоритмов;

– создать программное обеспечение (ПО), реализующее перечисленные выше алгоритмы;

– провести сравнение затрат процессорного времени на работу алгоритмов.

1 Аналитический раздел

В данном разделе будут рассмотрены классический алгоритм умножения матриц и алгоритм Винограда.

Матрицы A и B могут быть перемножены, если они совместимы (число столбцов A равно числу строк B) [1].

1.1 Классический алгоритм умножения

Если $A = (a_{ij})$ — матрица размером $m \times n$, а $B = (b_{ij})$ — матрица размером $n \times p$, то их произведение $C = AB$ представляет собой матрицу $C = (c_{ij})$ размером mp. В классическом алгоритме умножения матриц элементы матрицы C определяются уравнением

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \tag{1.1}$$

для $i = \overline{1, m}$, $j = \overline{1, p}$ и $k = \overline{1, n}$ [1].

1.2 Алгоритм Винограда

Пусть $n = 4$, и пусть в матрице A n столбцов, а в матрице B n строк. Тогда элемент c_{ij} может быть вычислен по формуле (1.2).

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + a_{i4}b_{4j} \tag{1.2}$$

Формулу (1.2) можно заменить на эквивалентное выражение (1.3).

$$c_{ij} = (a_{i1}+b_{2j})(a_{i2}+b_{1j})+(a_{i3}+b_{4j})(a_{i4}+b_{3j})-a_{i1}a_{i2}-a_{i3}a_{i4}-b_{j1}b_{j2}-b_{j3}b_{j4} \tag{1.3}$$

Попарные умножения элементов в (1.3) могут быть предварительно вычислены по формулам (1.4) — (1.5).

$$E_i = \sum_{k=1}^{n/2} a_{i,2k-1} \cdot b_{i,2k} \tag{1.4}$$

$$F_j = \sum_{k=1}^{n/2} b_{2k-1,j} \cdot b_{2k,j} \tag{1.5}$$

Тогда элемент c_{ij} будет вычисляться по формуле (1.6) [2].

$$c_{ij} = \sum_{k=1}^{n/2}(a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - E_i - F_j \tag{1.6}$$

В алгоритме Винограда количество операций умножения уменьшается за счёт увеличения числа сложений. Алгоритм Винограда теоретически быстрее стандартного алгоритма, так как операция сложения на электронно-вычислительной машине (ЭВМ) выполняется быстрее, чем умножение [2].

2 Конструкторский раздел

В данном разделе будут приведены требования к входным, выходным параметрам и представлены схемы для классического алгоритма умножения матриц, алгоритма Винограда и его оптимизированной версии.

В качестве оптимизирующих операций были выполнены:

- замена умножения на 2 на двоичный сдвиг;
- использование инкремента «+=»;
- вынос начальной итерации из каждого внешнего цикла;

2.1 Требования к входным и выходным параметрам

Требования к входным и выходным параметрам:

- в качестве входных параметров алгоритм принимает две матрицы;
- корректными входными параметрами являются совместимые матрицы, в каждой из которых есть по крайней мере 1 строка и 1 столбец;
- выходным параметром является матрица, полученная в результате умножения первой матрицы на вторую.

2.2 Классический алгоритм умножения матриц

На рисунке 2.1 представлена схема классического алгоритма умножения матриц.

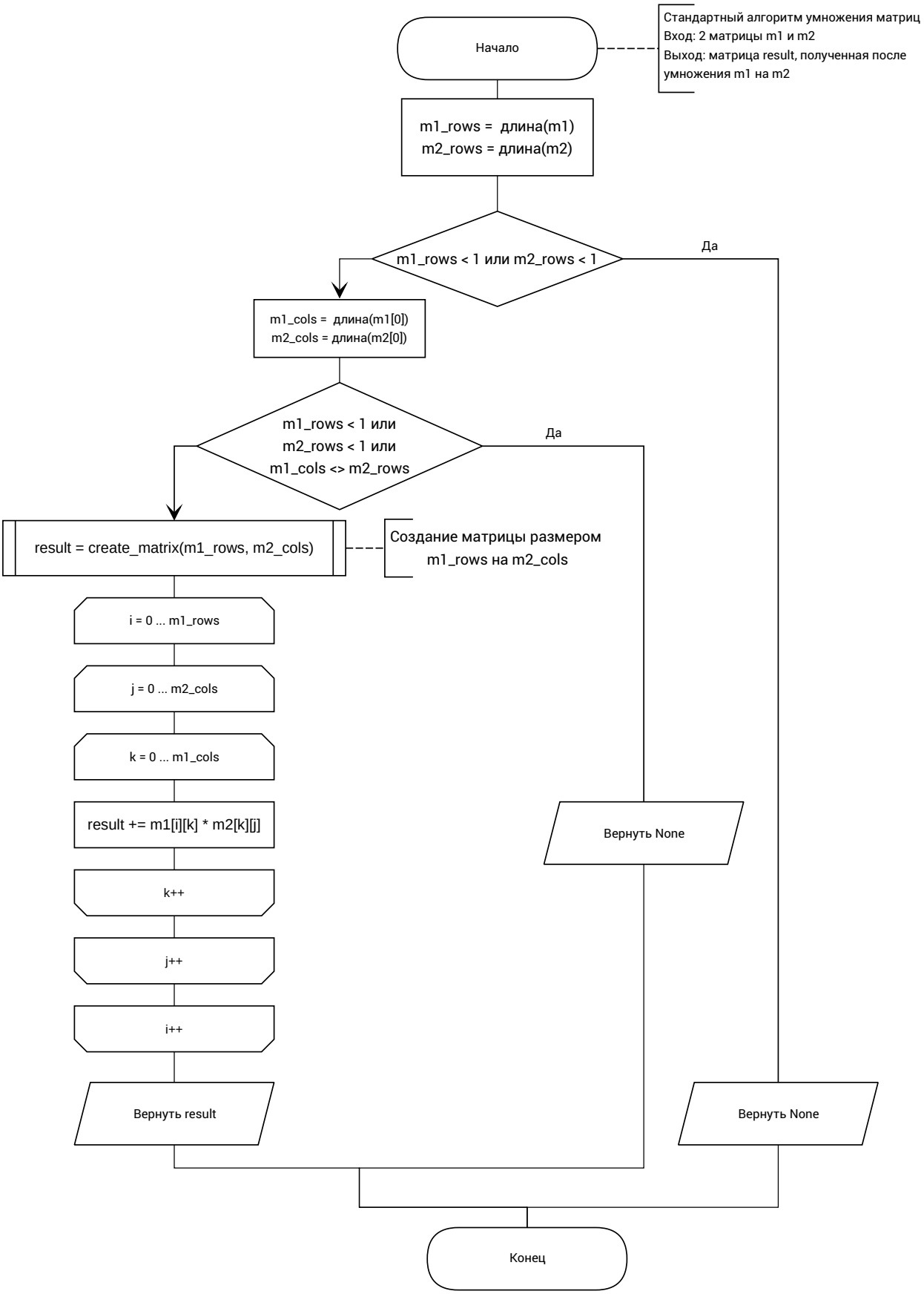

Рисунок 2.1 – Схема классического алгоритма умножения матриц

2.3 Алгоритм Винограда

На рисунках 2.2 — 2.3 представлена схема алгоритма Винограда.

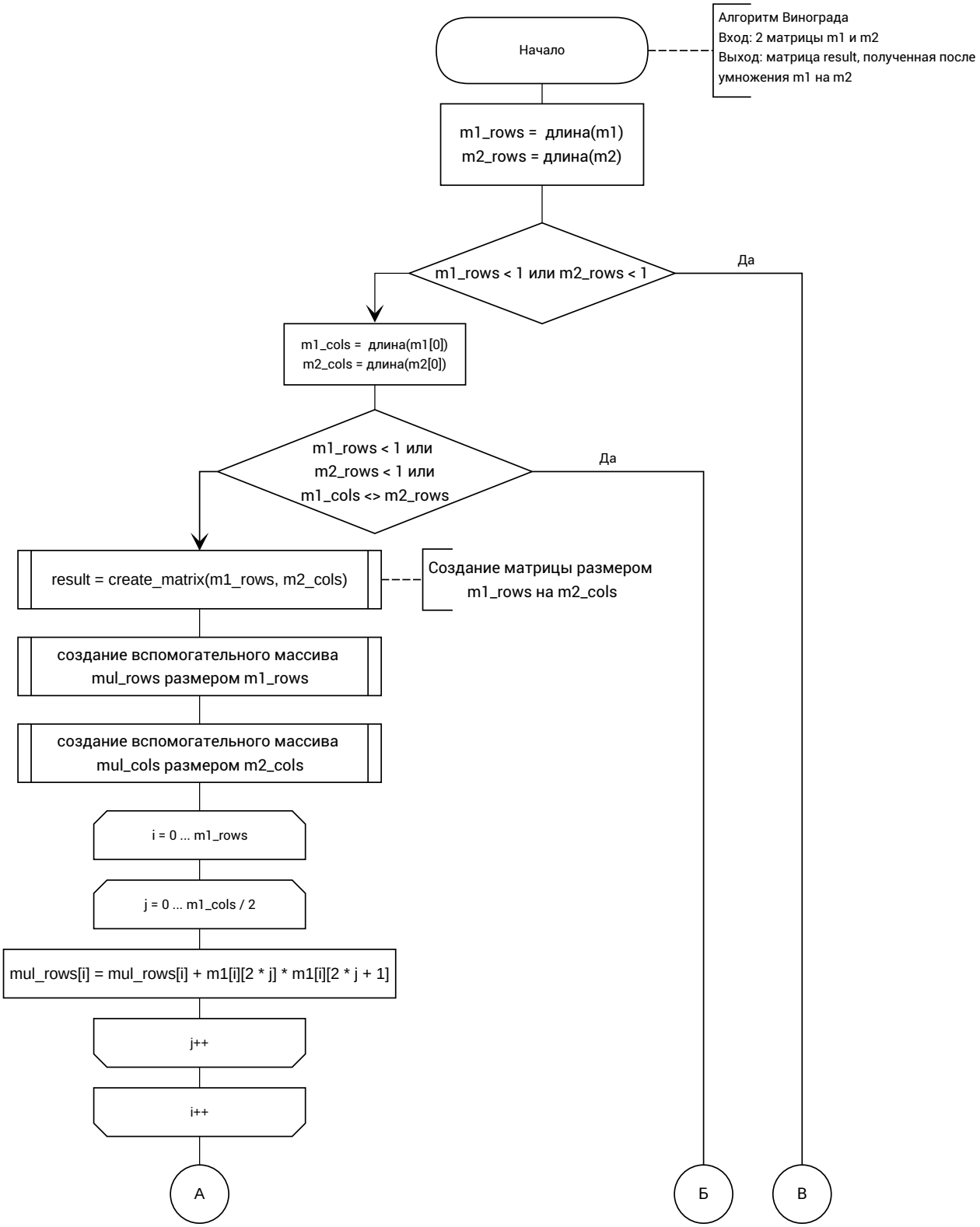

Рисунок 2.2 – Схема алгоритма Винограда. Часть 1

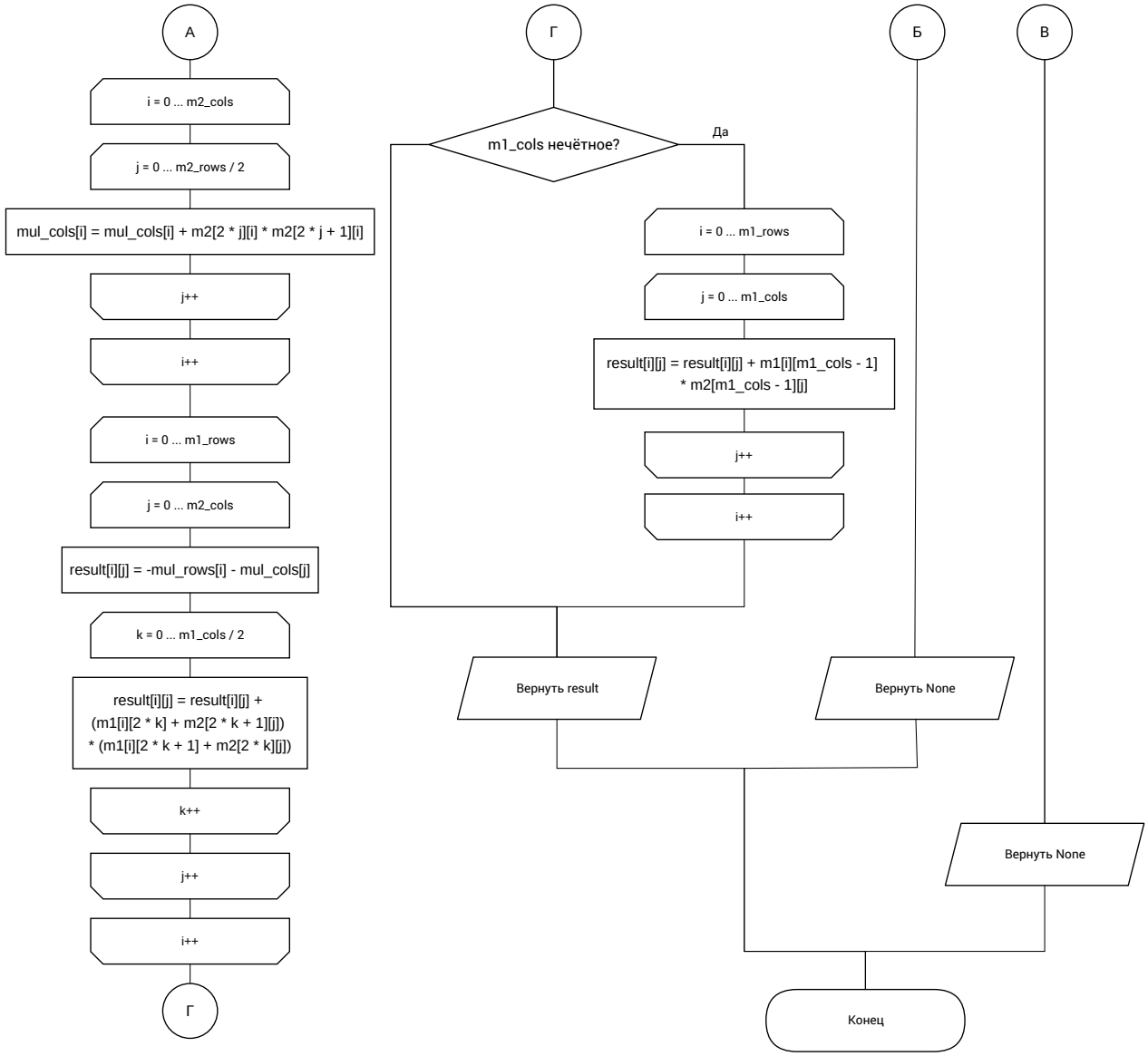

Рисунок 2.3 – Схема алгоритма Винограда. Часть 2

2.4 Оптимизированный алгоритм Винограда

На рисунках 2.4 — 2.6 представлена схема оптимизированного алгоритма Винограда.

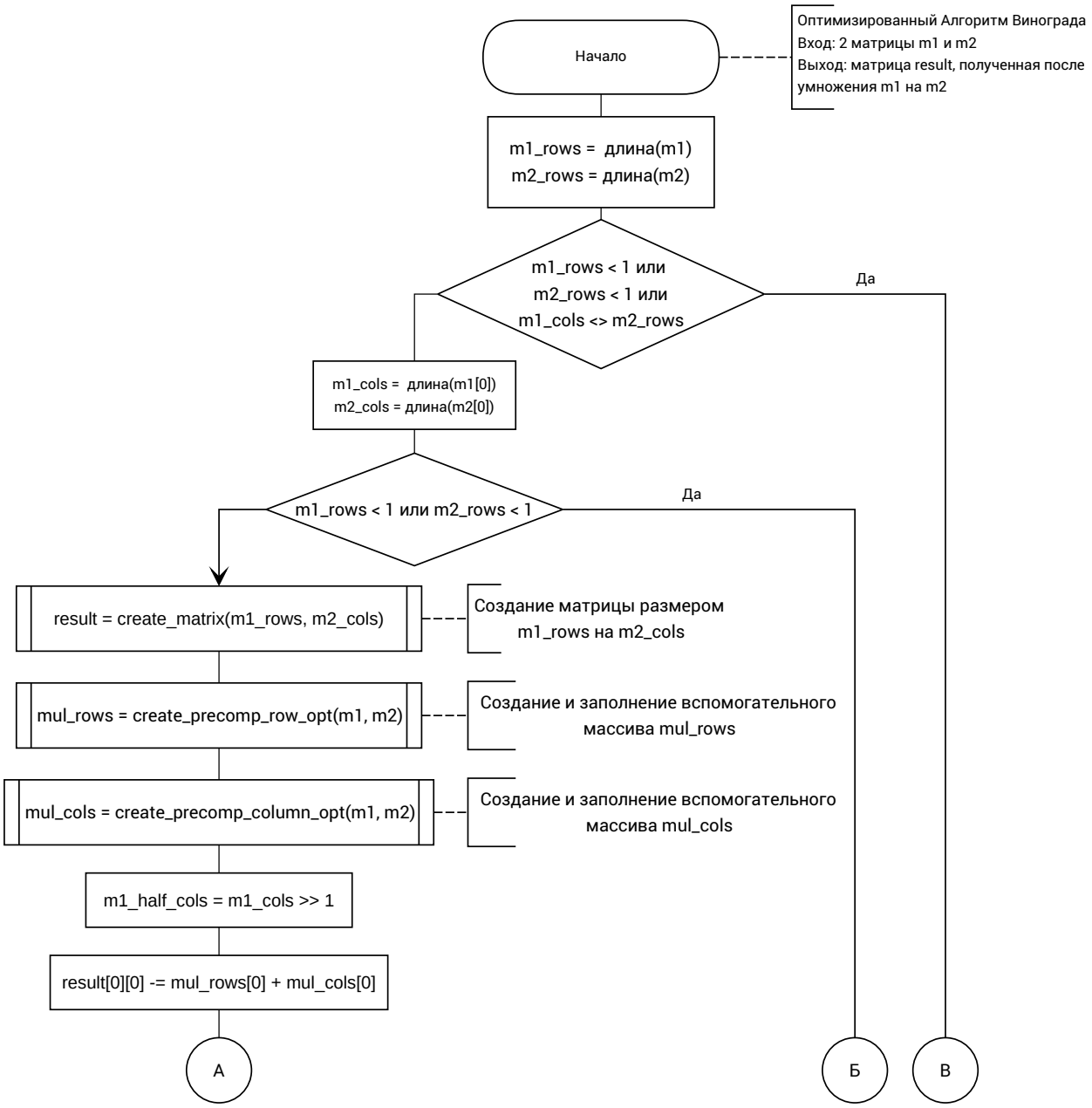

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда. Часть 1

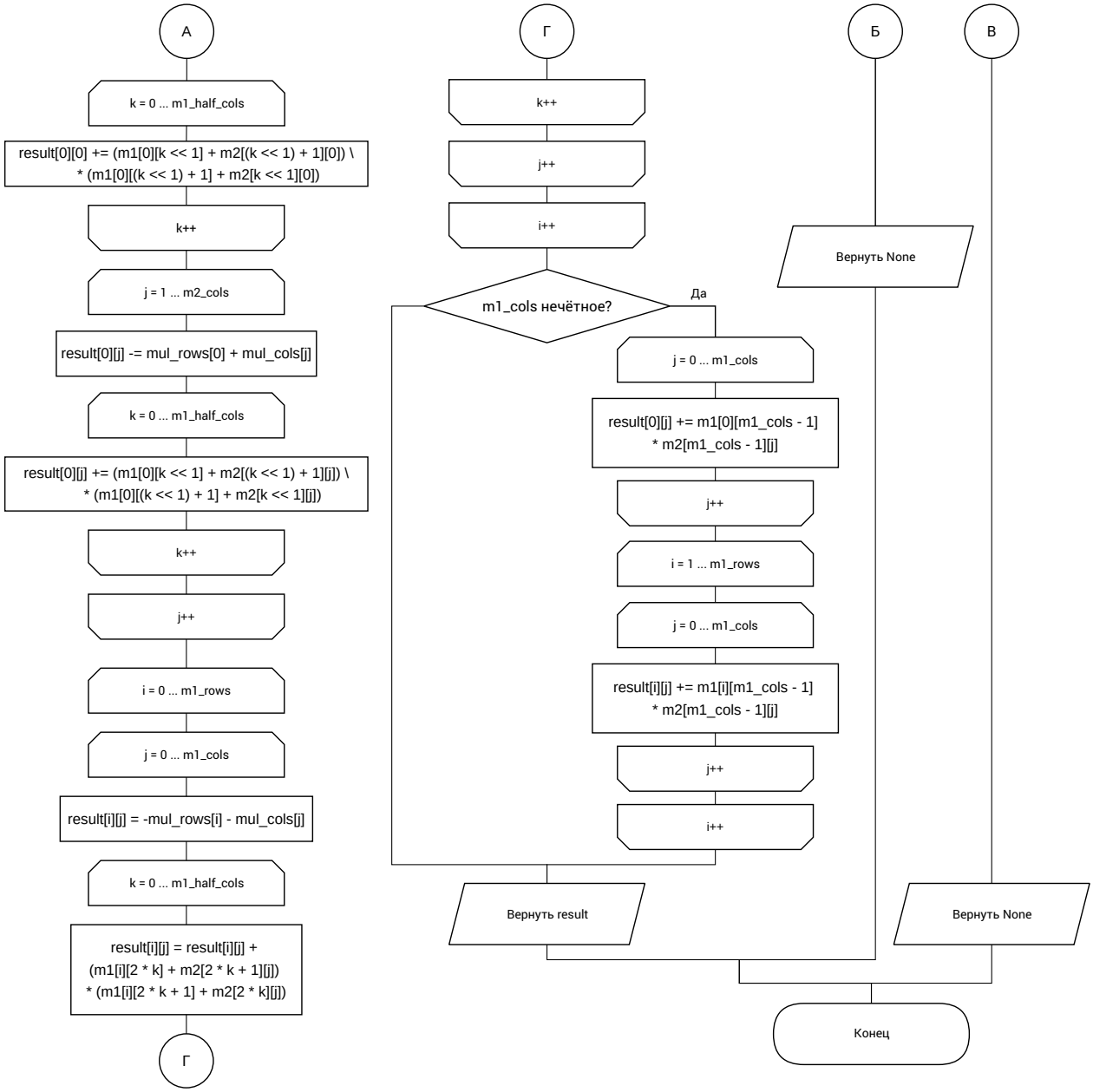

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда. Часть 2

На рисунке 2.6 представлены схемы алгоритмов заполнения вспомогательных массивов попарными операциями умножения.

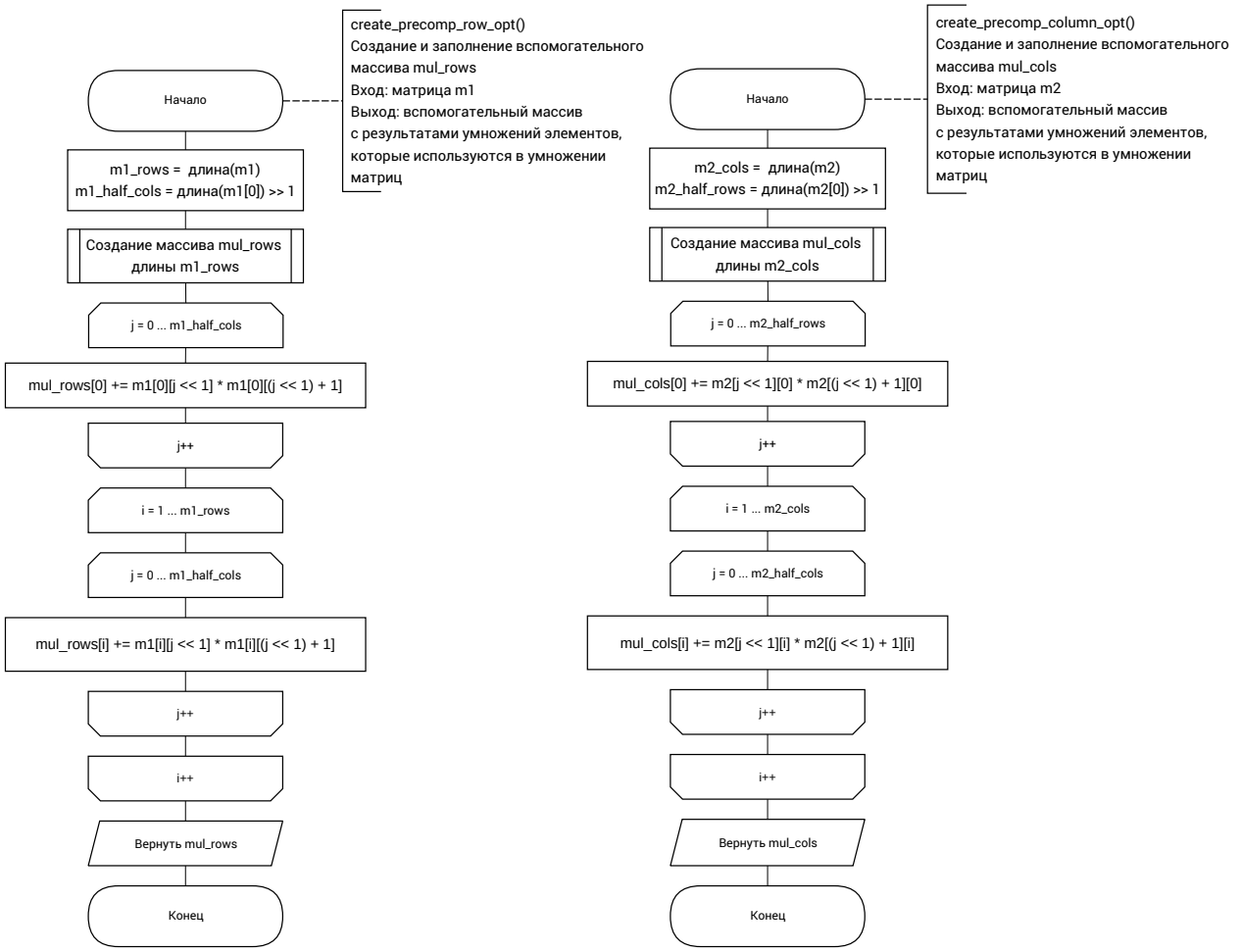

Рисунок 2.6 – Схемы алгоритмов заполнения вспомогательных массивов

2.5 Оценка трудоёмкости

2.5.1 Модель вычислений

Операции, имеющие единичную стоимость:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [idx], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

Операции, имеющие двойную стоимость:

$$*, /, \%, * =, / =, \% =, len() \tag{2.2}$$

Трудоёмкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} \min(f_1, f_2), & \text{лучший случай} \\ \max(f_1, f_2), & \text{худший случай} \end{cases} \tag{2.3}$$

Трудоёмкость цикла:

$$f_{loop} = f_{\text{инициализация}} + f_{\text{сравнения}} + N \times (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

Трудоёмкость вызова функции равна нулю.

2.5.2 Трудоёмкость стандартного алгоритма

Стоимость начальной проверки значений и инициализации переменных:

$$f_{init} = 3 \cdot 2 + 3 + (2 + 1 + 1) \cdot 2 + 5 + (2 + m \cdot (2 + p)). \tag{2.5}$$

Стоимость основного цикла рассчитывается по формуле (2.6).

$$f_{loop} = 2 + m \cdot (4 + p \cdot (4 + n \cdot (2 + 9))) \tag{2.6}$$

Итоговая трудоёмкость рассчитывается по формуле (2.7).

$$f = f_{init} + f_{loop} = 26 + 6 \cdot m + 5 \cdot m \cdot p \\ + 11 \cdot m \cdot n \cdot p \approx 11 \cdot m \cdot n \cdot p = O(N^3) \tag{2.7}$$

2.5.3 Трудоёмкость алгоритма Винограда

Стоимость начальной проверки значений и инициализации переменных совпадает со стоимостью одноимённого пункта в стандартном алгоритме:

$$f_{init} = 3 \cdot 2 + 3 + (2 + 1 + 1) \cdot 2 + 5 + (2 + m \cdot (2 + p)) \\ = 24 + 2 \cdot m + m \cdot p. \tag{2.8}$$

Стоимости инициализации вспомогательных массивов вычисляются по формулам (2.9) и (2.10.

$$f_{row\_init} = 2 + 2 \cdot m + 2 + m \cdot (4 + 2 + \frac{n}{2} \cdot (1 + 1 + 15)). \tag{2.9}$$

$$f_{col\_init} = 2 + 2 \cdot p + 2 + p \cdot (4 + 2 + \frac{n}{2} \cdot (1 + 1 + 15)). \tag{2.10}$$

Стоимость основного цикла в алгоритме Винограда вычисляется по формуле (2.11).

$$\begin{aligned} f_{loop} &= 2 + m \cdot (2 + 2 + p \cdot (13 + \frac{n}{2} \cdot 30)) \\ &= 2 + 4 \cdot m + 13 \cdot m \cdot p + 15 \cdot m \cdot n \cdot p \end{aligned} \tag{2.11}$$

Стоимость учёта нечётности в алгоритме Винограда рассчитывается по формуле (2.12).

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + 4 \cdot m + 16 \cdot m \cdot p, & \text{нечётная} \end{cases} \tag{2.12}$$

Итоговая трудоёмкость алгоритма Винограда вычисляется по формуле (2.21).

$$f = f_{init} + f_{row\_init} + f_{col\_init} + f_{loop} + f_{check} \tag{2.13}$$

В лучшем случае трудоёмкость алгоритма составляет:

$$\begin{aligned} f = 37 + 14 \cdot m + 8 \cdot p + 14 \cdot m \cdot p + 8.5 \cdot m \cdot n + 8.5 \cdot n \cdot p \\ +15 \cdot m \cdot n \cdot p \approx 15 \cdot m \cdot n \cdot p = O(N^3) \end{aligned} \tag{2.14}$$

В худшем случае трудоёмкость алгоритма составляет:

$$\begin{aligned} f = 39 + 18 \cdot m + 8 \cdot p + 30 \cdot m \cdot p + 8.5 \cdot m \cdot n + 8.5 \cdot n \cdot p \\ +15 \cdot m \cdot n \cdot p \approx 15 \cdot m \cdot n \cdot p = O(N^3) \end{aligned} \tag{2.15}$$

2.5.4 Трудоёмкость оптимизированного алгоритма Винограда

Стоимость начальной проверки значений и инициализации переменных вычисляется по формуле:

$$f_{init} = 2 + 3 \cdot 2 + 3 + (2 + 1 + 1) \cdot 2 + 5 + (2 + m \cdot (2 + p)) \\ = 26 + 2 \cdot m + m \cdot p. \tag{2.16}$$

Стоимости инициализации вспомогательных массивов вычисляются по формулам (2.17) и (2.18.

$$f_{row\_init} = 13 + 2 \cdot m + \frac{n}{2} \cdot 13 + (m - 1) \cdot (4 + \frac{n}{2} \cdot 13) \\ = 9 + 6 \cdot m + 6.5 \cdot m \cdot n \tag{2.17}$$

$$f_{col\_init} = 13 + 2 \cdot p + \frac{n}{2} \cdot 13 + (p - 1) \cdot (4 + \frac{n}{2} \cdot 13) \\ = 9 + 6 \cdot p + 6.5 \cdot p \cdot n \tag{2.18}$$

Стоимость основного цикла оптимизированного алгоритма Винограда с учётом выноса первых итераций каждого внешнего цикла вычисляется по формуле:

$$f_{main} = 6 + 2 + \frac{n}{2} \cdot 23 + 2 + (p - 1) \cdot (10 + \frac{n}{2} \cdot 23) + 2 \\ +(m - 1) \cdot (4 + p \cdot (10 + \frac{n}{2} \cdot 23)) = -2 + 4 \cdot m + 10 \cdot m \cdot p + 11.5 \cdot m \cdot n \cdot p \tag{2.19}$$

Стоимость учёта нечётности в алгоритме Винограда рассчитывается по формуле (2.20).

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 4 + 13 \cdot p + (m - 1)(4 + 13 \cdot p), & \text{нечётная} \end{cases} \tag{2.20}$$

Итоговая трудоёмкость оптимизированного алгоритма Винограда вычисляется по формуле (2.21).

$$f = f_{init} + f_{row\_init} + f_{col\_init} + f_{main} + f_{check} \tag{2.21}$$

В лучшем случае трудоёмкость алгоритма составляет:

$$f = 45 + 12 \cdot m + 6 \cdot p + 11 \cdot m \cdot p + 6.5 \cdot m \cdot n \\ +6.5 \cdot n \cdot p + 11.5 \cdot m \cdot n \cdot p \approx 11.5 \cdot m \cdot n \cdot p = O(N^3) \tag{2.22}$$

В худшем случае трудоёмкость алгоритма составляет:

$$f = 45 + 16 \cdot m + 6 \cdot p + 24 \cdot m \cdot p + 6.5 \cdot m \cdot n$$
$$+6.5 \cdot n \cdot p + 11.5 \cdot m \cdot n \cdot p \approx 11.5 \cdot m \cdot n \cdot p = O(N^3) \tag{2.23}$$

Вывод

В данном разделе были приведены схемы алгоритмов умножения матриц и рассчитана их трудоёмкость. В рамках модели вычислений, определённой в подразделе 2.5.1, все приведённые алгоритмы имеют кубическую сложность. Оптимизация уменьшила коэффициент при старшем члене в формуле трудоёмкости для алгоритма Винограда до 11.5, но наименьший коэффициент при старшем члене все же принадлежит стандартному алгоритму (11).

3 Технологический раздел

В данном разделе будет представлена реализация алгоритмов поиска редакционного расстояния. Также будут указаны средства реализации алгоритмов и результаты тестирования.

3.1 Средства реализации

Для реализации был выбран язык программирования Python [3]. Выбор обусловлен наличием функции вычисления процессорного времени time.process_time() в библиотеке time [4].

3.2 Реализация алгоритмов

В листинге 3.1 представлена функция для создания матрицы размером $rows \times cols$.

Листинг 3.1 – Функция создания матрицы

```
def create_matrix(rows: int, cols: int) -> list[list[int]]:
    return [[0] * cols for _ in range(rows)]
```

В листинге 3.2 продемонстрирована реализация стандартного алгоритма умножения матриц.

Листинг 3.2 – Реализация стандартного алгоритма умножения матриц

```
def standard_mult(m1: list[list[int]], m2: list[list[int]]) ->
   list[list[int]]:
    m1_rows, m2_rows = len(m1), len(m2)
    if m1_rows < 1 or m2_rows < 1:
        return None
    m1_cols, m2_cols = len(m1[0]), len(m2[0])
    if m1_cols < 1 or m2_cols < 1 or m1_cols != m2_rows:
        return None

    result = create_matrix(m1_rows, m2_cols)
    for i in range(m1_rows):
        for j in range(m2_cols):
            for k in range(m1_cols):
                result[i][j] += m1[i][k] * m2[k][j]

    return result
```

В листингах 3.3 и 3.4 представлены реализации алгоритмов Винограда и оптимизированного алгоритма Винограда.

Листинг 3.3 – Реализация алгоритма Винограда

```
def winograd_mult(m1: list[list[int]], m2: list[list[int]]) ->
   list[list[int]]:
    m1_rows, m2_rows = len(m1), len(m2)
    if m1_rows < 1 or m2_rows < 1:
        return None
    m1_cols, m2_cols = len(m1[0]), len(m2[0])
    if m1_cols < 1 or m2_cols < 1 or m1_cols != m2_rows:
        return None

    result = create_matrix(m1_rows, m2_cols)
    mul_rows, mul_cols = [0] * m1_rows, [0] * m2_cols

    for i in range(m1_rows):
        for j in range(m1_cols // 2):
            mul_rows[i] = mul_rows[i] + m1[i][2 * j] * m1[i][2 *
               j + 1]
    for i in range(m2_cols):
        for j in range(m2_rows // 2):
            mul_cols[i] = mul_cols[i] + m2[2 * j][i] * m2[2 * j
               + 1][i]
    for i in range(m1_rows):
        for j in range(m2_cols):
            result[i][j] = -mul_rows[i] - mul_cols[j]
            for k in range(m1_cols // 2):
                result[i][j] = result[i][j] + (m1[i][2 * k] +
                   m2[2 * k + 1][j]) \
                    * (m1[i][2 * k + 1] + m2[2 * k][j])

    if m1_cols % 2 == 1:
        for i in range(m1_rows):
            for j in range(m2_cols):
                result[i][j] = result[i][j] + m1[i][m1_cols - 1]
                   * m2[m1_cols - 1][j]

    return result
```

Листинг 3.4 – Реализация оптимизированного алгоритма Винограда

```
def winograd_optimized_mult(m1: list[list[int]], m2:
   list[list[int]]) -> list[list[int]]:
    m1_rows, m2_rows = len(m1), len(m2)
    if m1_rows < 1 or m2_rows < 1:
        return None
    m1_cols, m2_cols = len(m1[0]), len(m2[0])
    if m1_cols < 1 or m2_cols < 1 or m1_cols != m2_rows:
        return None
    result = create_matrix(m1_rows, m2_cols)
    mul_rows, mul_cols = create_precomp_row_opt(m1),
       create_precomp_column_opt(m2)
    m1_half_cols = m1_cols >> 1

    result[0][0] -= mul_rows[0] + mul_cols[0]
    for k in range(m1_half_cols):
        result[0][0] += (m1[0][k << 1] + m2[(k << 1) + 1][0]) \
            * (m1[0][(k << 1) + 1] + m2[k << 1][0])
    for j in range(1, m2_cols):
        result[0][j] -= mul_rows[0] + mul_cols[j]
        for k in range(m1_half_cols):
            result[0][j] += (m1[0][k<<1] + m2[(k << 1)+1][j]) \
                * (m1[0][(k << 1) + 1] + m2[k << 1][j])
    for i in range(1, m1_rows):
        for j in range(m2_cols):
            result[i][j] -= mul_rows[i] + mul_cols[j]
            for k in range(m1_half_cols):
                result[i][j] += (m1[i][k<<1]+m2[(k<<1)+1][j]) \
                    * (m1[i][(k<<1)+1]+m2[k<<1][j])

    if m1_cols % 2 == 1:
        for j in range(m2_cols):
                result[0][j] += m1[0][m1_cols-1]*m2[m1_cols-1][j]
        for i in range(1, m1_rows):
            for j in range(m2_cols):
                result[i][j] += m1[i][m1_cols-1]*m2[m1_cols-1][j]

    return result
```

В листинге 3.5 изображены функции для инициализации вспомогательных массивов для оптимизированного алгоритма Винограда.

Листинг 3.5 – Реализация оптимизированного алгоритма Винограда

```
def create_precomp_row_opt(m1: list[list[int]]) -> list[int]:
    m1_rows, m1_half_cols = len(m1), len(m1[0]) >> 1
    mul_rows = [0] * m1_rows

    for j in range(m1_half_cols):
        mul_rows[0] += m1[0][j << 1] * m1[0][(j << 1) + 1]
    for i in range(1, m1_rows):
        for j in range(m1_half_cols):
            mul_rows[i] += m1[i][j << 1] * m1[i][(j << 1) + 1]

    return mul_rows


def create_precomp_column_opt(m2: list[list[int]]) -> list[int]:
    m2_cols, m2_half_rows = len(m2[0]), len(m2) >> 1
    mul_cols = [0] * m2_cols

    for j in range(m2_half_rows):
        mul_cols[0] += m2[j << 1][0] * m2[(j << 1) + 1][0]
    for i in range(1, m2_cols):
        for j in range(m2_half_rows):
            mul_cols[i] += m2[j << 1][i] * m2[(j << 1) + 1][i]

    return mul_cols
```

3.3 Тестирование

В таблице 3.1 представлены тесты для алгоритмов умножения матриц, входные данные указаны в (3.2). Тестирование проводилось по методологии чёрного ящика. Все тесты пройдены успешно.

$$m_1 = [] \tag{3.1}$$

$$m_2 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$m_3 = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

$$m_4 = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

$$m_5 = \begin{bmatrix} 5 & 6 & 7 \end{bmatrix} \tag{3.2}$$

Таблица 3.1 – Тесты для алгоритмов умножения матриц

№	**Описание теста**	**Алгоритм**	**Входные данные**	**Выходные данные**	**Ожидаемые данные**
1	Пустые матрицы	Стандартный	m_1, m_1	None	None
2	Пустые матрицы	Виноград	m_1, m_1	None	None
3	Пустые матрицы	Оптимизированный Виноград	m_1, m_1	None	None
4	Несовместимые матрицы	Стандартный	m_2, m_5	None	None
5	Несовместимые матрицы	Виноград	m_2, m_5	None	None
6	Несовместимые матрицы	Оптимизированный Виноград	m_2, m_5	None	None
7	Корректные данные	Стандартный	m_2, m_3	m_4	m_4
8	Корректные данные	Виноград	m_2, m_3	m_4	m_4
9	Корректные данные	Оптимизированный Виноград	m_2, m_3	m_4	m_4

4 Исследовательский раздел

4.1 Технические характеристики

Исследование проводилось на ЭВМ со следующими характеристиками:

– операционная система Ubuntu 22.04.5 LTS;

– объем оперативной памяти 16 ГБ;

– процессор Intel Core i7-8700K CPU 3.70Ггц × 12 [5].

4.2 Проведение исследования

Было проведено исследование зависимости времени работы алгоритмов умножения матриц от размера входных квадратных матриц. Для каждого размера было проведено 10 замеров времени. Измерение времени проводилось с помощью функции *process_time* из модуля *time*.

В таблице 4.1 приведены результаты замера процессорного времени работы алгоритмов.

Таблица 4.1 – Зависимость времени работы алгоритмов от линейного размера входных матриц

Размер матрицы	**Стандартный, с**	**Виноград, с**	**Оптимизированный Виноград, с**
21	0.0013	0.0014	0.0013
41	0.0077	0.0100	0.0090
61	0.0248	0.0318	0.0283
81	0.0579	0.0746	0.0660
101	0.1118	0.1436	0.1268
121	0.1924	0.2462	0.2174
141	0.3040	0.3882	0.3433
161	0.4522	0.5765	0.5086
181	0.6419	0.8139	0.7194
201	0.8778	1.1183	0.9856
221	1.1668	1.4779	1.3059
241	1.5149	1.9216	1.6916
261	1.9474	2.4351	2.1496
281	2.4148	3.0461	2.7247
301	2.9777	3.7617	3.3824
321	3.6667	4.5926	4.1631
341	4.3503	5.5051	5.0158

На рисунке 4.1 продемонстрированы графики, построенные на основе табличных данных.

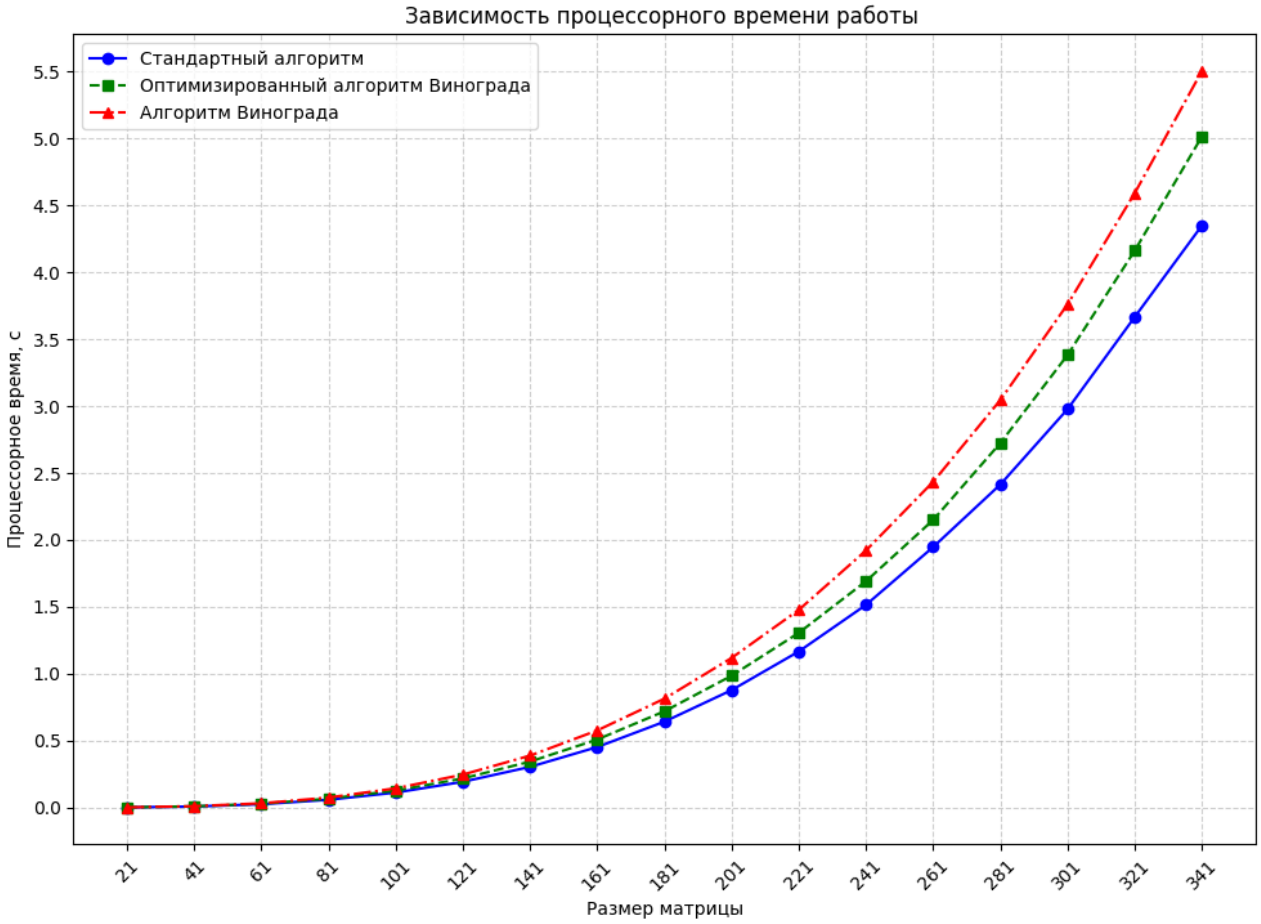

Рисунок 4.1 – Зависимость времени работы алгоритмов от линейного размера входных матриц

Вывод

Исследование процессорного времени подтвердило теоретические расчёты трудоёмкости алгоритмов. Оптимизированный алгоритм Винограда для размеров матриц от 21 до 341 в среднем требует на 12.17% процессорного времени меньше, чем неоптимизированный алгоритм. Но выполненных оптимизаций оказалось не достаточно для достижения показателей, меньших чем у стандартного алгоритма умножения матриц.

ЗАКЛЮЧЕНИЕ

Исследование процессорного времени работы подтвердило теоретические расчёты трудоёмкости алгоритмов умножения матриц. Оптимизированный алгоритм Винограда для размеров матриц от 21 до 341 в среднем требует на 12.17% процессорного времени меньше, чем неоптимизированный алгоритм. Но выполненных оптимизаций оказалось не достаточно для достижения показателей, меньших чем у стандартного алгоритма умножения матриц.

В рамках лабораторной работы:

- были построены схемы для стандартного алгоритма умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда;
- были вычислены трудоёмкости данных алгоритмов на основе введённой модели;
- было разработано ПО, реализующее перечисленные выше алгоритмы;
- проведено сравнение процессорного времени работы алгоритмов.

Все задачи выполнены. Цель лабораторной работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Кормен Т. Х.* Алгоритмы: построение и анализ. — Москва : Издательский дом «Вильямс», 2011.
2. *Kakaradov B.* Ultra-fast matrix multiplication: An empirical analysis of highly optimized vector algorithms // Stanford Undergraduate Research Journal. — 2004.
3. Welcome to Python [Электронный ресурс]. — Режим доступа: `https://www.python.org/` (дата обращения: 07.11.2024).
4. time — Time access and conversions [Электронный ресурс]. — Режим доступа: `https://docs.python.org/3/library/time.html#module-time` (дата обращения: 07.11.2024).
5. Intel Core i7-8700K Processor [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/us/en/ark/products/126775/intel-core-i78700k-processor-12m-cache-up-to-4-70-ghz.html).