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 № 2 по курсу "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Кузнецова А. В.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2022 г.

Оглавление

Введение

В современном мире приходится работать с большими массивами данных. Для упорядочивания данных и последующей их обработке проводится специальная операция по представлению данных в порядке увеличения (или уменьшения) их значения. Данную операцию называют сортировкой. В данной работе приведено сравнение трех популярных методов сортировки данных [1].

С матрицами можно выполнять стандартные алгебраические операции: сложение, вычитание, умножение, деление.

В данной лабораторной работе будут рассмотрены алгоритмы умножения матриц. Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Две матрицы могут быть перемножены, если число столбцов первой матрицы равно числу строк второй матрицы.

Умножение матриц является основным инструментом линейной алгебры и имеет многочисленные применения в математике, физике, программировании. Одним из самых эффективных по времени алгоритмов умножения матриц является алгоритм Винограда, имеющий асимптотическую сложность $O(n^{2,3755})$. Также существуют улучшения этого алгоритма [1].

Целью данной лабораторной работы является изучение алгоритмов умножения матриц. Для достижения поставленной цели требуется решить следующие задачи:

1. Реализация классического алгоритма умножения матриц;
2. Реализация алгоритма Винограда умножения матриц;
3. Реализация алгоритма Винограда умножения матриц с оптимизациями;
4. Создание схем изучаемых алгоритмов;
5. Оценка трудоёмкости алгоритмов;
6. Определение средств программной реализации алгоритмов;

7. Выполнение замеров процессорного времени работы реализаций алгоритмов умножения матриц;

8. Проведение сравнительного анализа по времени алгоритмов умножения матриц;

9. Подготовка отчета о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе представлено теоретическое описание алгоритмов умножения матриц: классического, Винограда и Винограда с оптимизациями.

1.1 Классический алгоритм умножения матриц

По определению умножения матриц для $n \times m$ матрицы А:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \tag{1.1}$$

и $m \times p$ матрицы В:

$$B = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mp} \end{pmatrix} \tag{1.2}$$

произведением $C = AB$ является $n \times p$ матрица, состоящая из элементов:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \quad (i = 1, 2, \dots n; j = 1, 2, \dots p) \tag{1.3}$$

Для того, чтобы умножить матрицу A размером $n \times m$ и матрицу B размером $m \times p$, нужно взять i-ю строку $(i = 1, 2, \dots n)$ в первой матрице и j-й $(j = 1, 2, \dots p)$ столбец во второй, перемножить их поэлементно, сложить полученные произведения и записать результат в i-й, j-й элемент результирующей матрицы.

1.2 Алгоритм Винограда

Каждый элемент результирующей матрицы представляет собой скалярное произведение соответствующих строки и столбца.

Рассмотрим два вектора:

$$A = (a_1, ..., a_n) \tag{1.4}$$

$$B = (b_1, ..., b_n) \tag{1.5}$$

Их скалярное произведение равно:

$$AB = \sum_{1}^{n} a_i b_i \tag{1.6}$$

$$AB = \sum_{1}^{n/2} (a_{2i} + b_{2i+1})(a_{2i+1} + b_{2i}) - t \tag{1.7}$$

$$t = \sum_{1}^{n/2} a_{2i} a_{2i+1} + \sum_{1}^{n/2} b_{2i} b_{2i+1} \tag{1.8}$$

В выражении 1.7 и 1.8 требуется большее число вычислений, чем в первом, но оно позволяет произвести предварительную обработку. Выражения $a_{2i}a_{2i+1}$ и $b_{2i}b_{2i+1}$ для $i = 1, 2, \ldots n/2$ можно вычислить заранее для каждой соответствующей строки и столбца. Это позволяет уменьшить число умножений [1].

1.3 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда схематично представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- замена операции $x = x + k$ на $x+ = k$;
- замена умножения на 2 на побитовый сдвиг;

- предвычисление некоторых слагаемых для алгоритма.

Вывод

В данном разделе были описаны алгоритмы умножения матриц: классический, Винограда и Винограда с оптимизациямми.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы вышеизложенных алгоритмов, а также вычислена их трудоемкость.

2.1 Разработка алгоритмов

На рисунке 2.1 представлен классический алгоритм умножения матриц.

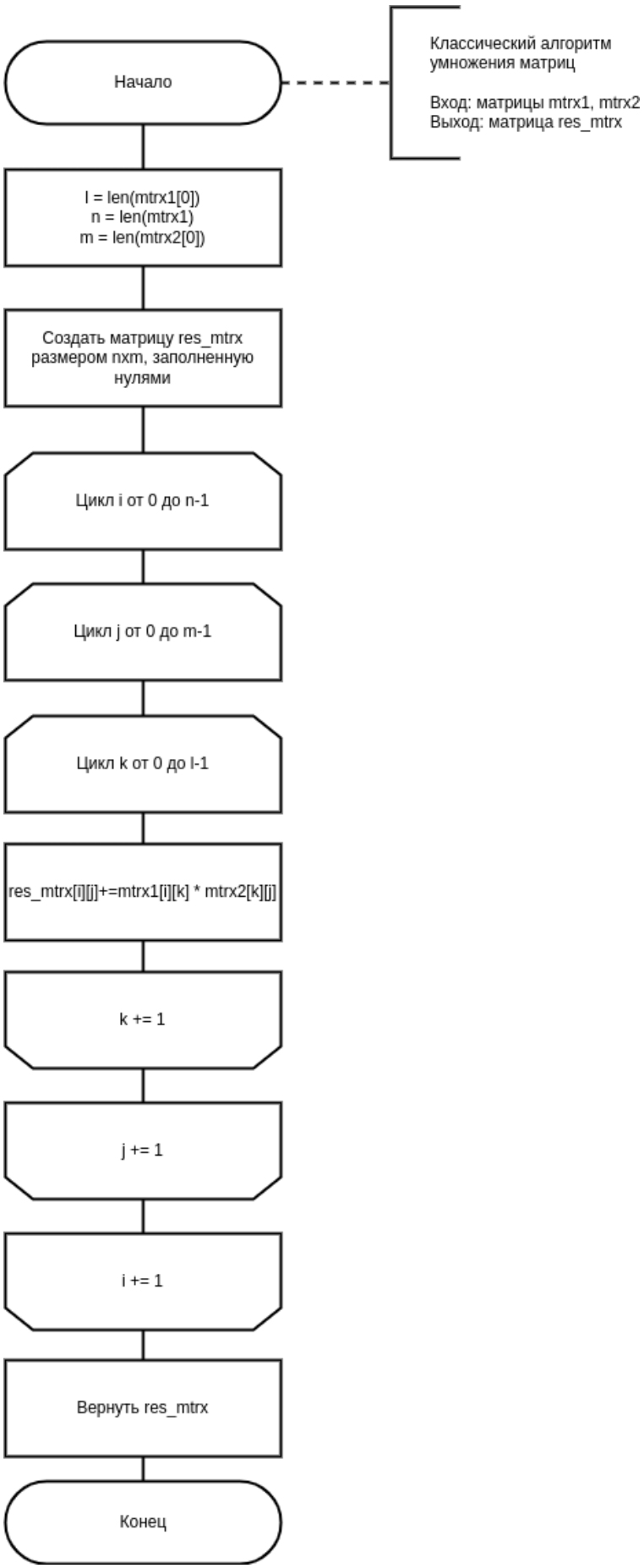

Рисунок 2.1 – Схема классического алгоритма умножения матриц

На рисунке 2.2 представлен алгоритм Винограда.

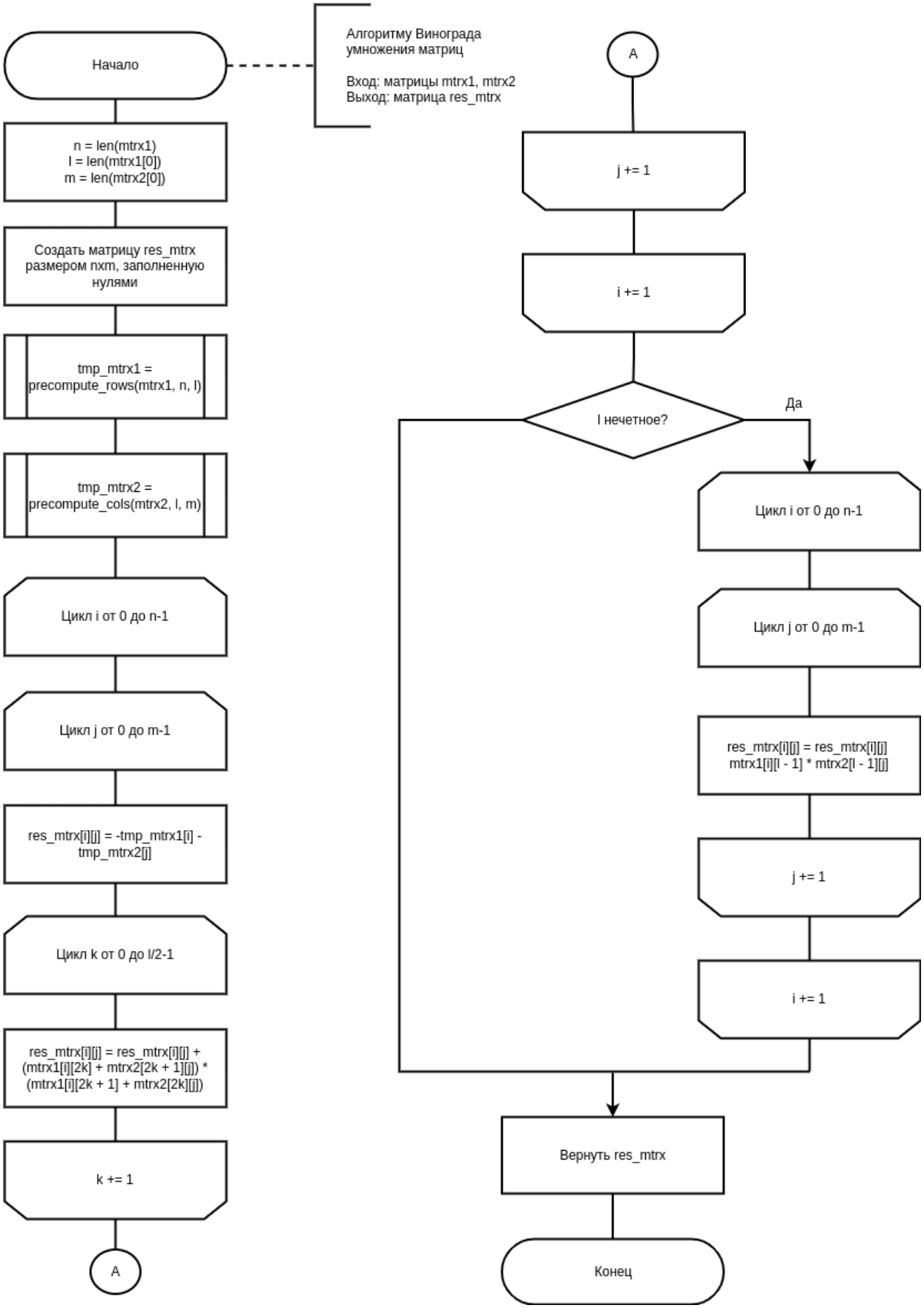

Рисунок 2.2 – Схема алгоритма Винограда

На рисунке 2.3 представлены алгоритмы предварительных вычислений слагаемых для алгоритма Винограда.

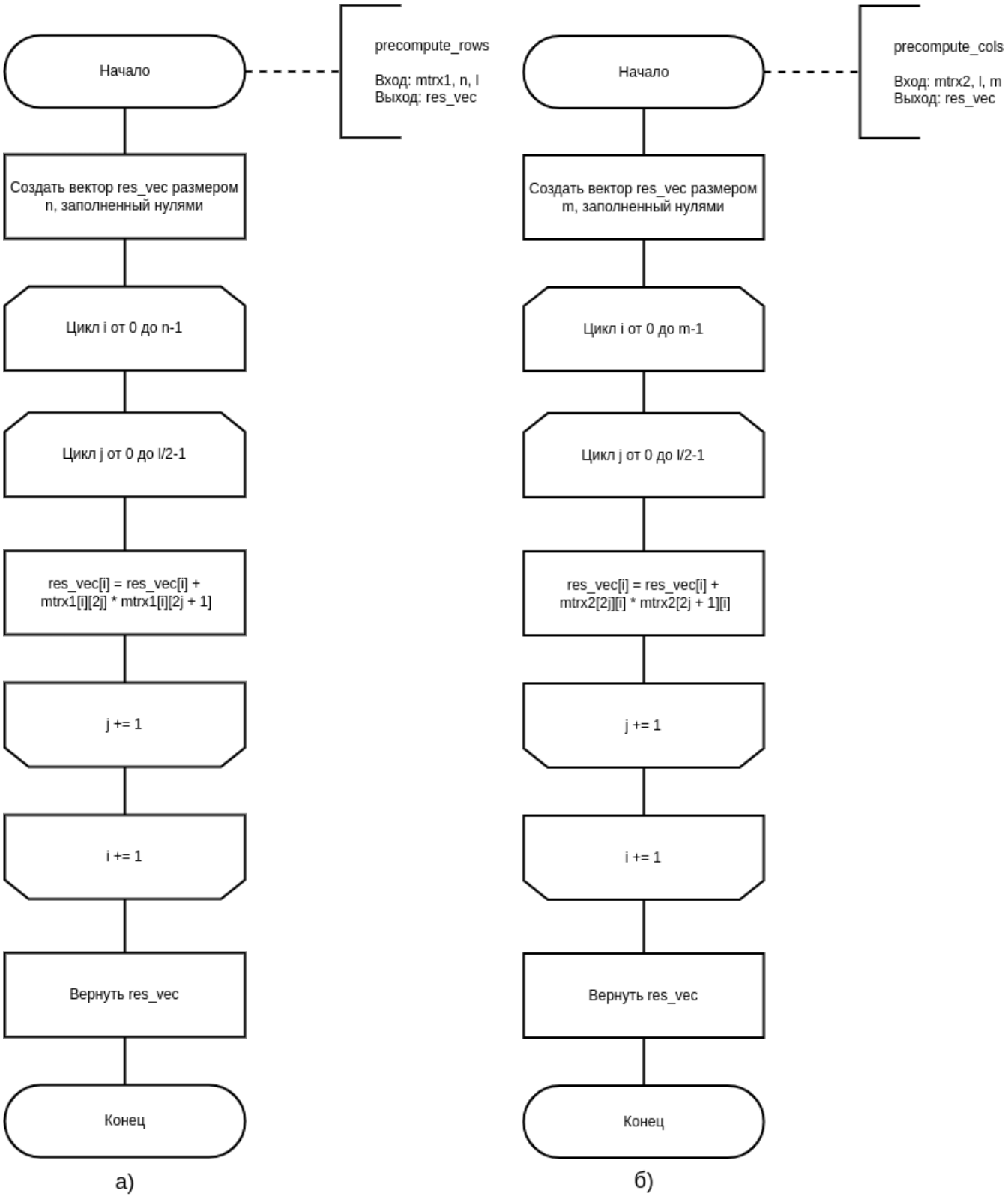

Рисунок 2.3 – Схемы алгоритмов предварительных вычислений слагаемых для алгоритма Винограда: а)для строк первой матрицы; б)для столбцов второй матрицы

На рисунке 2.4 представлен алгоритм Винограда с оптимизациями.

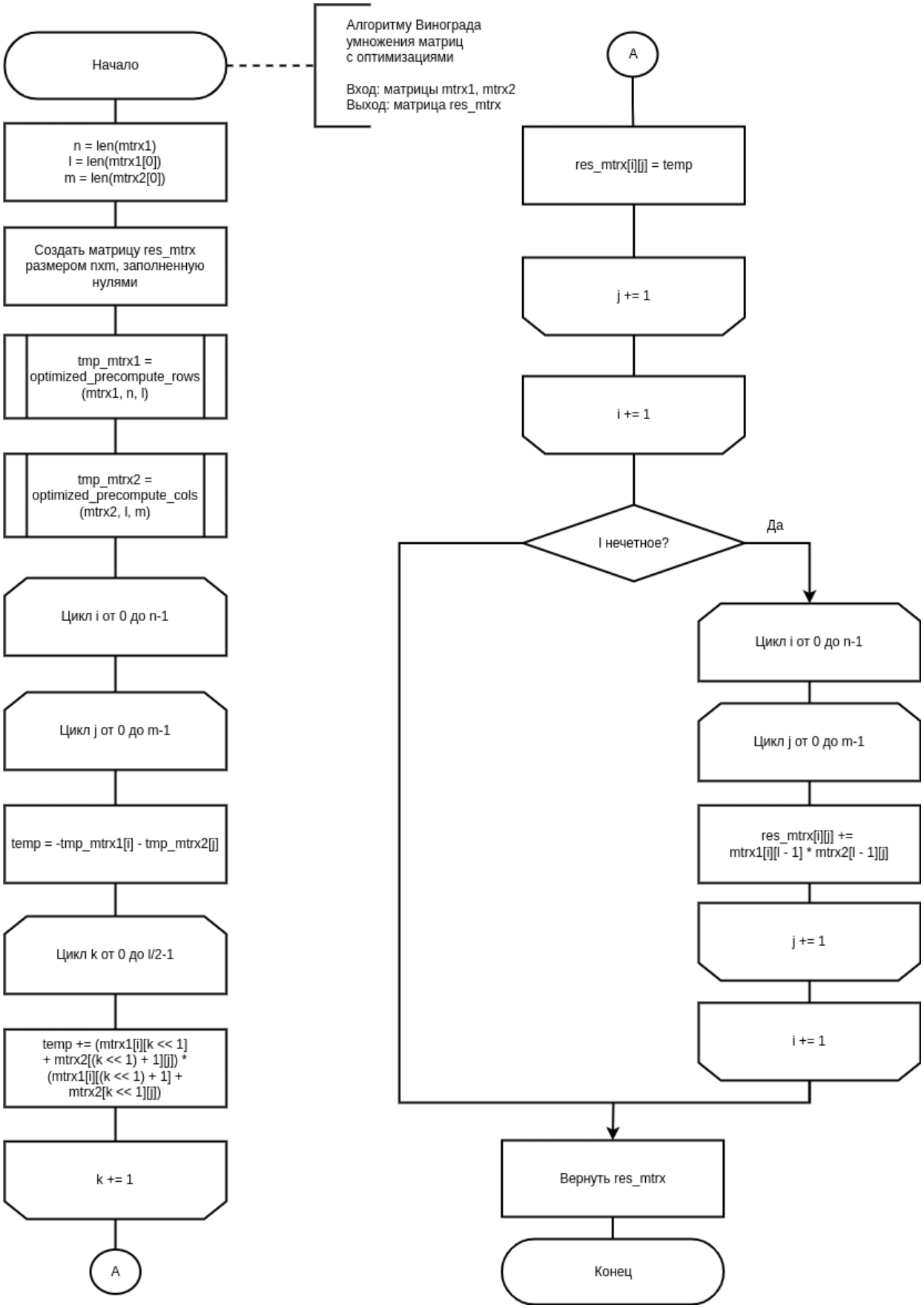

Рисунок 2.4 – Схема алгоритма Винограда с оптимизациями

На рисунке 2.5 представлены алгоритмы предварительных вычислений слагаемых для алгоритма Винограда с оптимизациями.

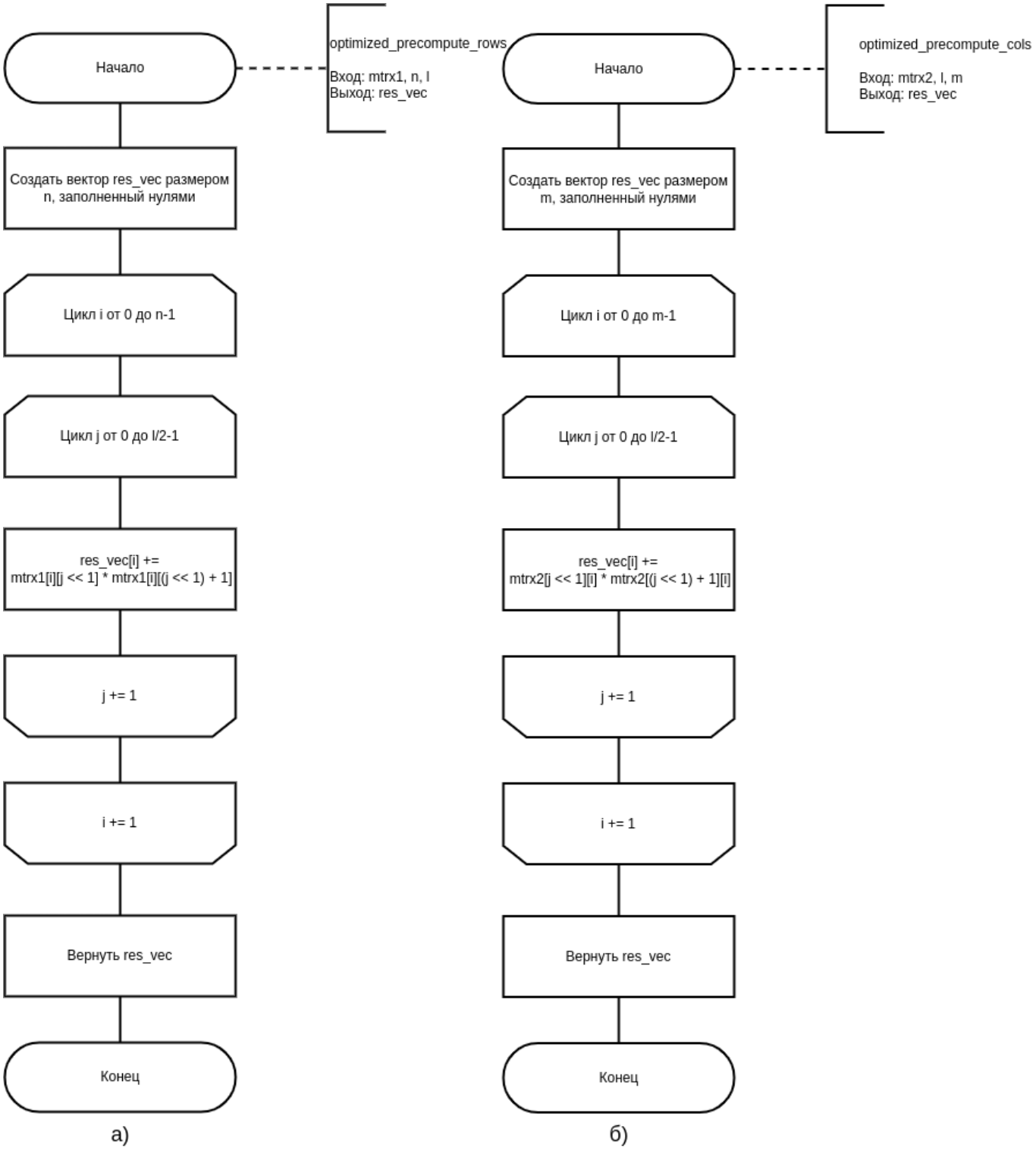

Рисунок 2.5 – Схемы алгоритмов предварительных вычислений слагаемых для алгоритма Винограда с оптимизациями: а)для строк первой матрицы; б)для столбцов второй матрицы

2.2 Вычисление трудоемкости алгоритмов

2.2.1 Классический алгоритм умножения матриц

Трудоёмкость классического алгоритма равна (2.1):

$$f_{classic} = 2 + n \cdot (2 + 2 + m \cdot (2 + 2 + l \cdot 11)) = \\ 2 + 4 \cdot n + 4 \cdot mn + 11 \cdot lmn \approx 11 \cdot lmn \tag{2.1}$$

2.2.2 Алгоритм Винограда

Трудоёмкость алгоритма Винограда равна (2.2):

$$\begin{gathered} f_{winograd} = 2 + n \cdot (4 + 2 + l/2 \cdot 19) + 2 + l \cdot (4 + 2 + m/2 \cdot 19) + \\ 2 + n \cdot (2 + 2 + m \cdot (7 + 4 + 2 + l/2 \cdot 32)) + \\ 3 + \left[\begin{array}{c} 0, \text{ лучший случай (размерность четная)} \\ 2 + n \cdot (2 + 2 + m \cdot 16), \text{ худший случай (нечетная)} \end{array} \right. \\ = 9 + 10n + 6l + 19/2ln + 19/2lm + 13mn + 16lmn + \\ \left[\begin{array}{c} 0, \text{ л. с.} \\ 2 + 4n + 16mn, \text{ х. с.} \end{array} \right. \\ \approx 16lmn \end{gathered} \tag{2.2}$$

2.2.3 Оптимизированный алгоритм Винограда

Трудоёмкость оптимизированного алгоритма Винограда равна (2.3):

$$
\begin{gathered}
f_{optimized\_winograd} = 2 + n \cdot (4 + 2 + l/2 \cdot 15) + 2 + \\
l \cdot (4 + 2 + m/2 \cdot 15) + 2 + \\
n \cdot (2 + 2 + m \cdot (8 + 4 + 2 + l/2 \cdot 19)) + 3 + \\
\left[\begin{array}{c} 0, \text{ лучший случай (размерность четная)} \\ 2 + n \cdot (2 + 2 + m \cdot 13), \text{худший случай (нечетная)} \end{array} \right. \\
= 9 + 10n + 6l + 15/2ln + 15/2lm + 14mn \\
19/2lmn + \left[\begin{array}{c} 0, \text{ л. с.} \\ 2 + 4n + 13mn, \text{ х. с.} \end{array} \right. \\
\approx 9,5lmn
\end{gathered}
\tag{2.3}
$$

Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов. Кроме того, была произведена оценка трудоемкости алгоритмов. Наименее трудоемким алгоритмом оказался оптимизированный алгоритм Винограда.

3 Технологическая часть

В данном разделе приведены средства реализации, сведения о модулях программы, листинги кода, тесты.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Python [2]. Данный выбор обусловлен тем, что он позволяет реализовывать сложные задачи за короткие сроки за счет наличия большого количества подключаемых библиотек и простоты синтаксиса.

Замеры времени проводились при помощи функции process_time_ns из библиотеки time [3].

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

- main.py - файл, содержащий точку входа в программу, в нем происходит вызов алгоритмов;
- multiplication.py - файл, содержащий алгоритмы умножения матриц;
- matrix.py - файл, содержащий функции работы с матрицами;
- measurements.py - файл, содержащий функции замеров времени работы алгоритмов;

3.3 Листинги кода

В листингах 3.1–3.3 представлены реализации алгоритмов умножения матриц: классического, Винограда и Винограда с оптимизациями.

Листинг 3.1 – Классический алгоритм умножения матриц

```
def classic_multiplication(mtrx1, mtrx2):
    n, l, m = len(mtrx1), len(mtrx1[0]), len(mtrx2[0])

    res_mtrx = [[0] * m for _ in range(n)]

    for i in range(n):
        for j in range(m):
            for k in range(l):
                res_mtrx[i][j] += mtrx1[i][k] * mtrx2[k][j]

    return res_mtrx
```

Листинг 3.2 – Алгоритм Винограда

```
def winograd_multiplication(mtrx1, mtrx2):
    n, l, m = len(mtrx1), len(mtrx1[0]), len(mtrx2[0])

    res_mtrx = [[0] * m for _ in range(n)]

    tmp_mtrx1 = precompute_rows(mtrx1, n, l)
    tmp_mtrx2 = precompute_cols(mtrx2, l, m)

    for i in range(n):
        for j in range(m):
            res_mtrx[i][j] = -tmp_mtrx1[i] - tmp_mtrx2[j]
            for k in range(l // 2):
                res_mtrx[i][j] = res_mtrx[i][j] + (mtrx1[i][k *
                    2] + mtrx2[k * 2 + 1][j]) * (mtrx1[i][k * 2 +
                    1] + mtrx2[k * 2][j])

    if l % 2 != 0:
        for i in range(n):
            for j in range(m):
                res_mtrx[i][j] = res_mtrx[i][j] + mtrx1[i][l - 1]
                    * mtrx2[l - 1][j]
    return res_mtrx

def precompute_rows(mtrx, n, m):
    res_vec = [0] * n

```

```
    for i in range(n):
        for j in range(m // 2):
            res_vec[i] = res_vec[i] + mtrx[i][j * 2] * mtrx[i][j
                * 2 + 1]

    return res_vec

def precompute_cols(mtrx, n, m):
    res_vec = [0] * m

    for i in range(m):
        for j in range(n // 2):
            res_vec[i] = res_vec[i] + mtrx[j * 2][i] * mtrx[j * 2
                + 1][i]

    return res_vec
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
def optimized_winograd_multiplication(mtrx1, mtrx2):
    n, l, m = len(mtrx1), len(mtrx1[0]), len(mtrx2[0])

    res_mtrx = [[0] * m for _ in range(n)]

    tmp_mtrx1 = optimized_precompute_rows(mtrx1, n, l)
    tmp_mtrx2 = optimized_precompute_cols(mtrx2, l , m)

    for i in range(n):
        for j in range(m):
            temp = -tmp_mtrx1[i] - tmp_mtrx2[j]
            for k in range(l // 2):
                temp += (mtrx1[i][k << 1] + mtrx2[(k << 1) +
                    1][j]) * (mtrx1[i][(k << 1) + 1] + mtrx2[k <<
                    1][j])
            res_mtrx[i][j] = temp

    if l % 2 != 0:
        for i in range(n):
            for j in range(m):
                res_mtrx[i][j] += mtrx1[i][l - 1] * mtrx2[l -
                    1][j]
    return res_mtrx
```

```
21
22 def optimized_precompute_rows(mtrx, n, m):
23     res_vec = [0] * n
24
25     for i in range(n):
26         for j in range(m // 2):
27             res_vec[i] += mtrx[i][j << 1] * mtrx[i][(j << 1) + 1]
28
29     return res_vec
30
31 def optimized_precompute_cols(mtrx, n, m):
32     res_vec = [0] * m
33
34     for i in range(m):
35         for j in range(n // 2):
36             res_vec[i] += mtrx[j << 1][i] * mtrx[(j << 1) + 1][i]
37
38     return res_vec
```

3.4 Тестирование

В таблице 3.1 приведены функциональные тесты для алгоритмов умножения матриц.

Таблица 3.1 – Тесты

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 & 2 & 1 \\ 3 & 2 & 1 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 18 & 12 & 6 \\ 18 & 12 & 6 \\ 18 & 12 & 6 \end{pmatrix}$

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функциональное тестирование пройдено успешно.

Вывод

Был реализован классический алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будет проведен сравнительный анализ реализованных алгоритмов умножения матриц по затраченному процессорному времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование представлены ниже.

- операционная система: Manjaro Linux [4];
- память : 7,6 GiB;
- процессор: 8 × Intel® Core™ i5-10210U CPU @ 1.60GHz [5].

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы.

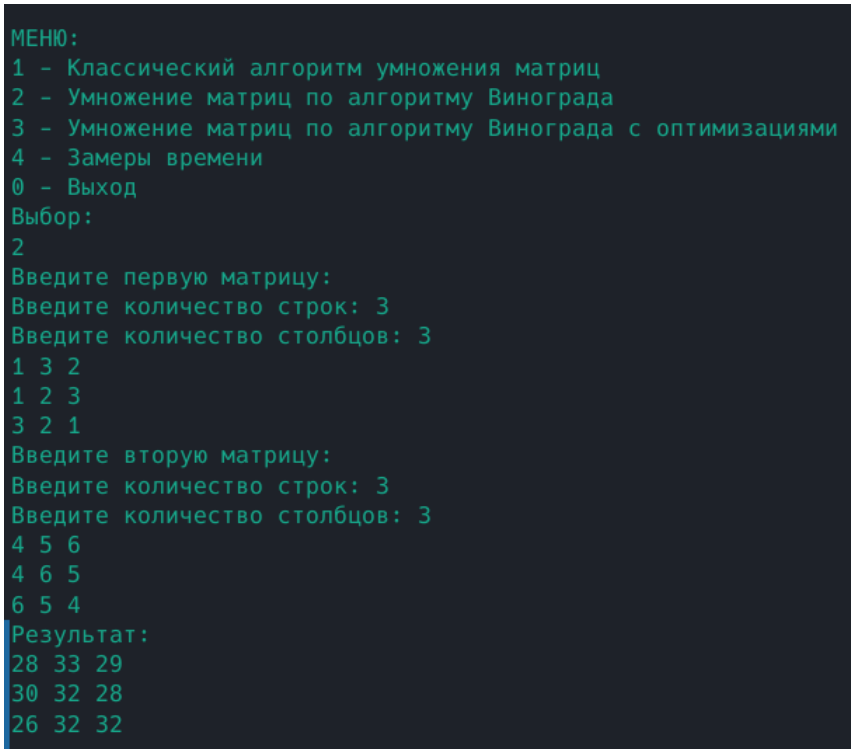

Рисунок 4.1 – Пример работы программы

4.3 Временные характеристики

Функция process_time_ns из библиотеки time языка программирования Python возвращает процессорное время в наносекундах.

Замеры проводились для матриц размерами от 10x10 до 101x101, заполненных случайными элементами.

На рисунке 4.2 приведены графические результаты сравнения временных характеристик.

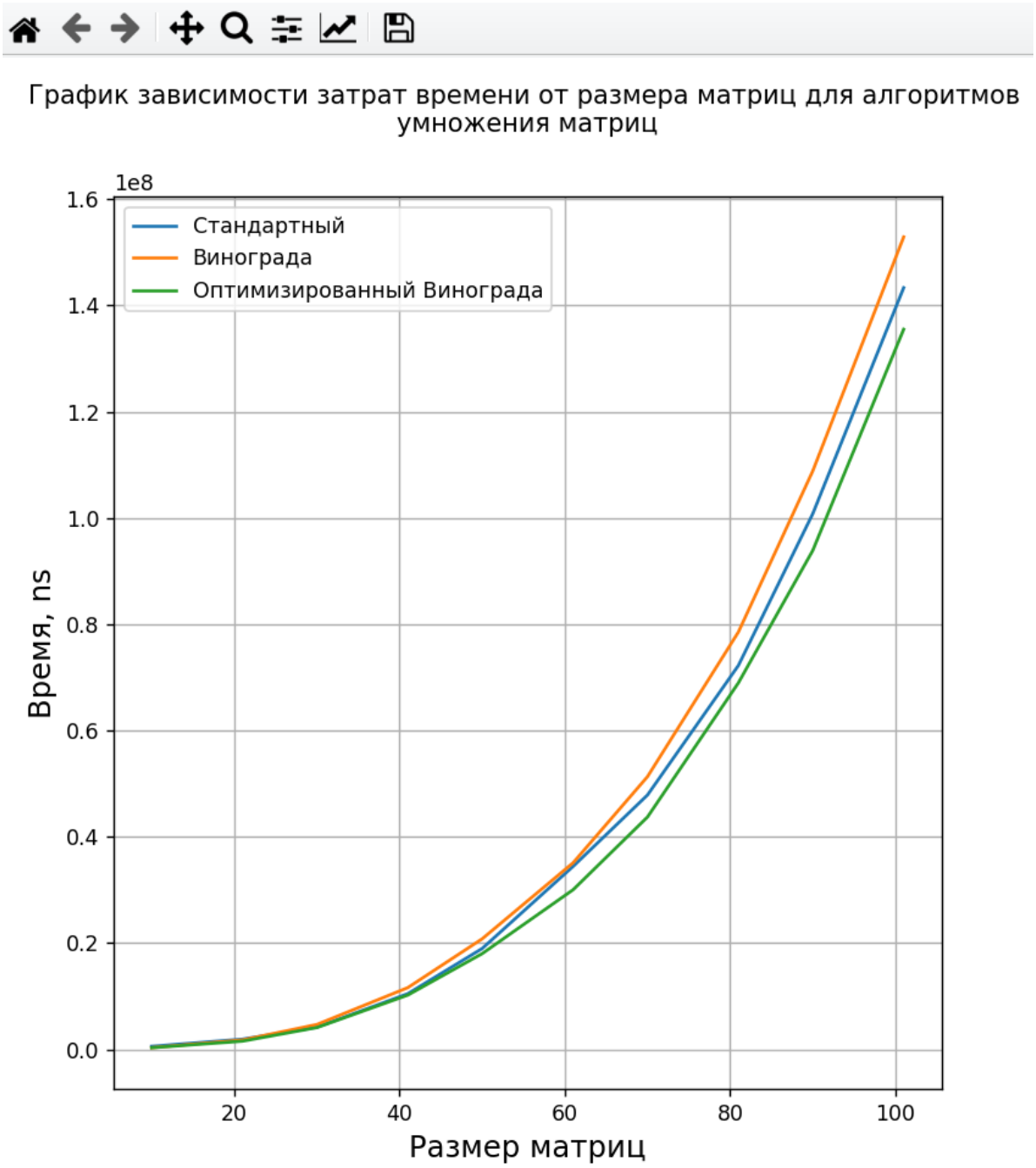

Рисунок 4.2 – Сравнение по времени алгоритмов умножения матриц

4.4 Вывод

Приведенные характеристики времени показывают нам, что наименее затратным по времени является оптимизированный алгоритм Винограда.

Заключение

В результате выполнения данной лабораторной работы были рассмотрены алгоритмы умножения матриц (классический, Винограда и оптимизированный Винограда), построены схемы, соответствующие данным алгоритмам. В рамках выполнения работы цель достигнута и решены следующие задачи:

- реализованы алгоритмы умножения матриц;
- проведена оценка трудоемкости алгоритмов;
- проведено сравнение временных характеристик;
- подготовлен отчет о выполненной лабораторной работе.

Список используемых источников

[1] Реализация алгоритма умножения матриц по Винограду на языке Haskell [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell (дата обращения: 19.10.2022).

[2] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 19.10.2022).

[3] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 19.10.2022).

[4] Manjaro Linux [Электронный ресурс]. Режим доступа: https://manjaro.org (дата обращения: 19.10.2022).

[5] Процессор Intel® Core™ i5-10210U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/195436/intel-core-i510210u-processor-6m-cache-up-to-4-20-ghz.html (дата обращения: 19.10.2022).