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»

Лабораторная работа №2
по дисциплине "Анализ Алгоритмов"

Тема Алгоритм Копперсмита-Винограда

Студент Ковалец К. Э.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Термин «матрица» применяется во множестве разных областей: от программирования до кинематографии. Матрица в математике – это таблица чисел, состоящая из определенного количества строк (m) и столбцов (n). Мы встречаемся с матрицами каждый день, так как любая числовая информация, занесенная в таблицу, уже в какой-то степени считается матрицей.

Алгоритм Копперсмита — Винограда — алгоритм умножения квадратных матриц, предложенный Д. Копперсмитом и Ш. Виноградом. Он обладает лучшей асимптотикой среди известных алгоритмов умножения матриц. На практике алгоритм Копперсмита — Винограда не используется, так как он имеет очень большую константу пропорциональности и начина- ет выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров.

Целью данной лабораторной работы является изучение и реализация алгоритмов умножения матриц, вычисление трудоёмкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- исследовать классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда;
- сравнить существующие решения;
- привести схемы рассматриваемых алгоритмов (классического алгоритма умножения матриц, алгоритма Винограда и модифицированного алгоритма Винограда);
- описать используемые структуры данных;
- оценить трудоёмкость рассматриваемых алгоритмов;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;

- провести сравнительный анализ времени работы алгоритмов;
- провести модульное тестирование;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут представлены описания алгоритмов стандартного умножения матриц и алгоритм Копперсмита-Винограда. [1]

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{1.3}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.2 Алгоритм Копперсмита-Винограда

Алгоритм Копперсмита-Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита — Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \quad (1.4)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

В случае нечетного значений размера изначальной матрицы (n), следует произвести еще одну операцию - добавление произведения последних элементов соответствующих строк и столбцов.

Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которого от классического алгоритма — наличие предварительной обработки, а также количество операций умножения.

На вход алгоритмам будут поступать две матрицы, кол-во столбцов первой матрицы должно совпадать с кол-вом строк второй матрицы. При попытке ввести пустые матрицы будет выдано сообщение об ошибке. Реализуемое ПО будет давать возможность выбрать алгоритм и вывести для него результат вычисления, а также возможность произвести сравнение алгоритмов по затраченному времени.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов умножения матриц, приведено описание используемых типов данных, оценка трудоёмкости алгоритмов, а также описана структура ПО.

2.1 Схемы алгоритмов

На вход алгоритмов падаются матрицы $matr1$ и matr2, на выходе получаем результат перемножения двух матриц, записанный в res_matr.

На рис. 2.1 - 2.5 приведены схемы стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

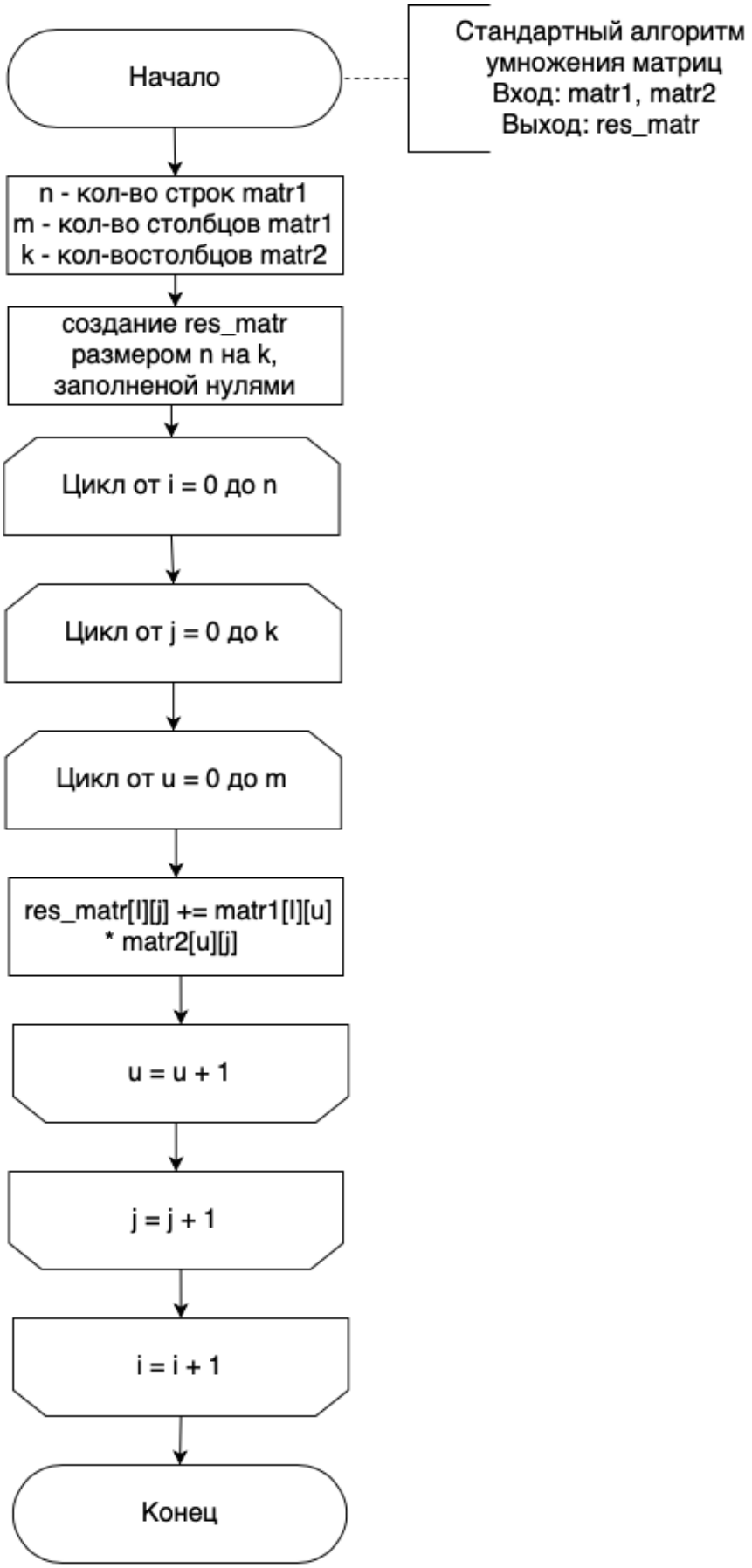

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

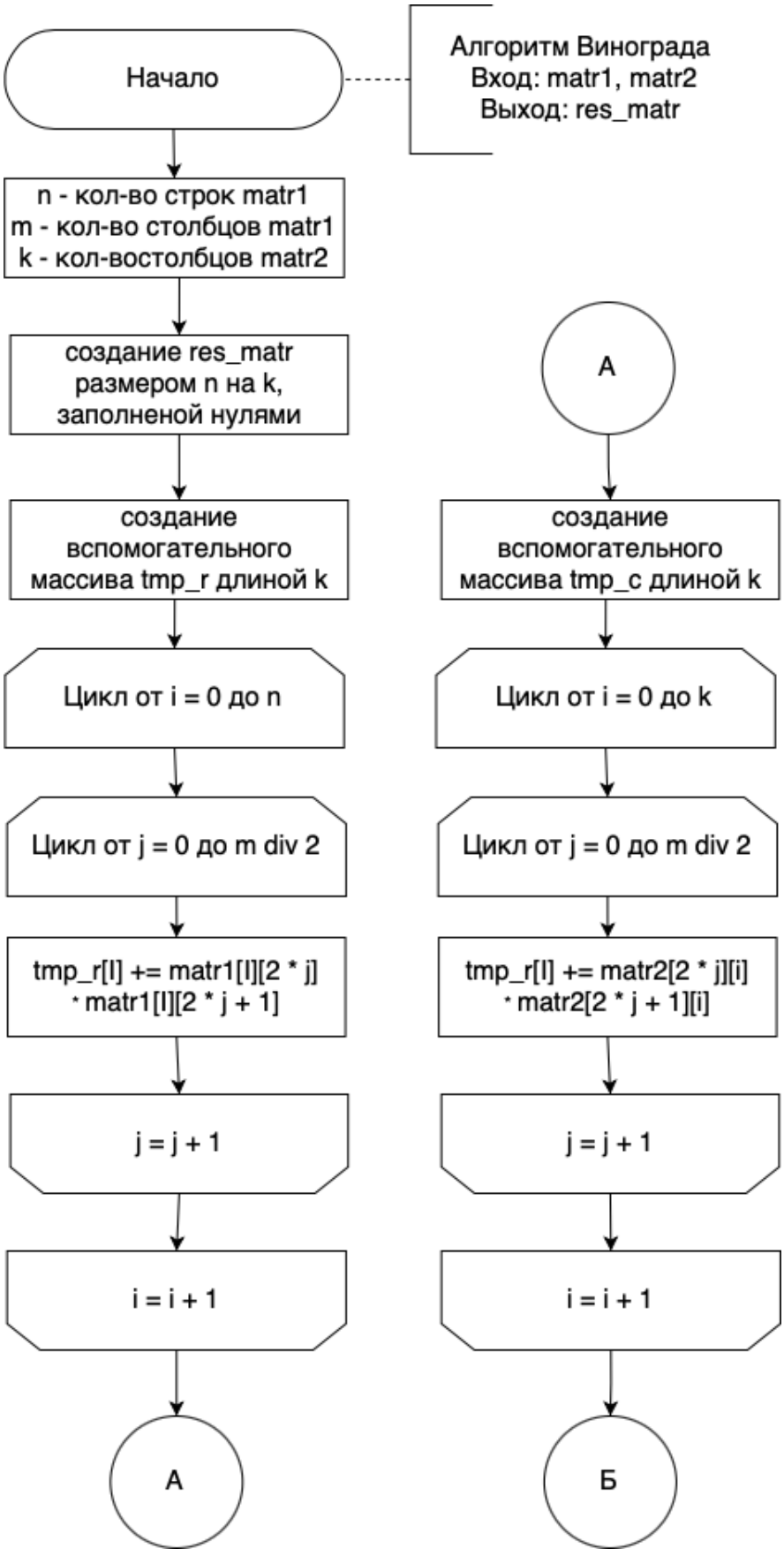

Рисунок 2.2 – Схема умножения матриц алгоритмом Винограда

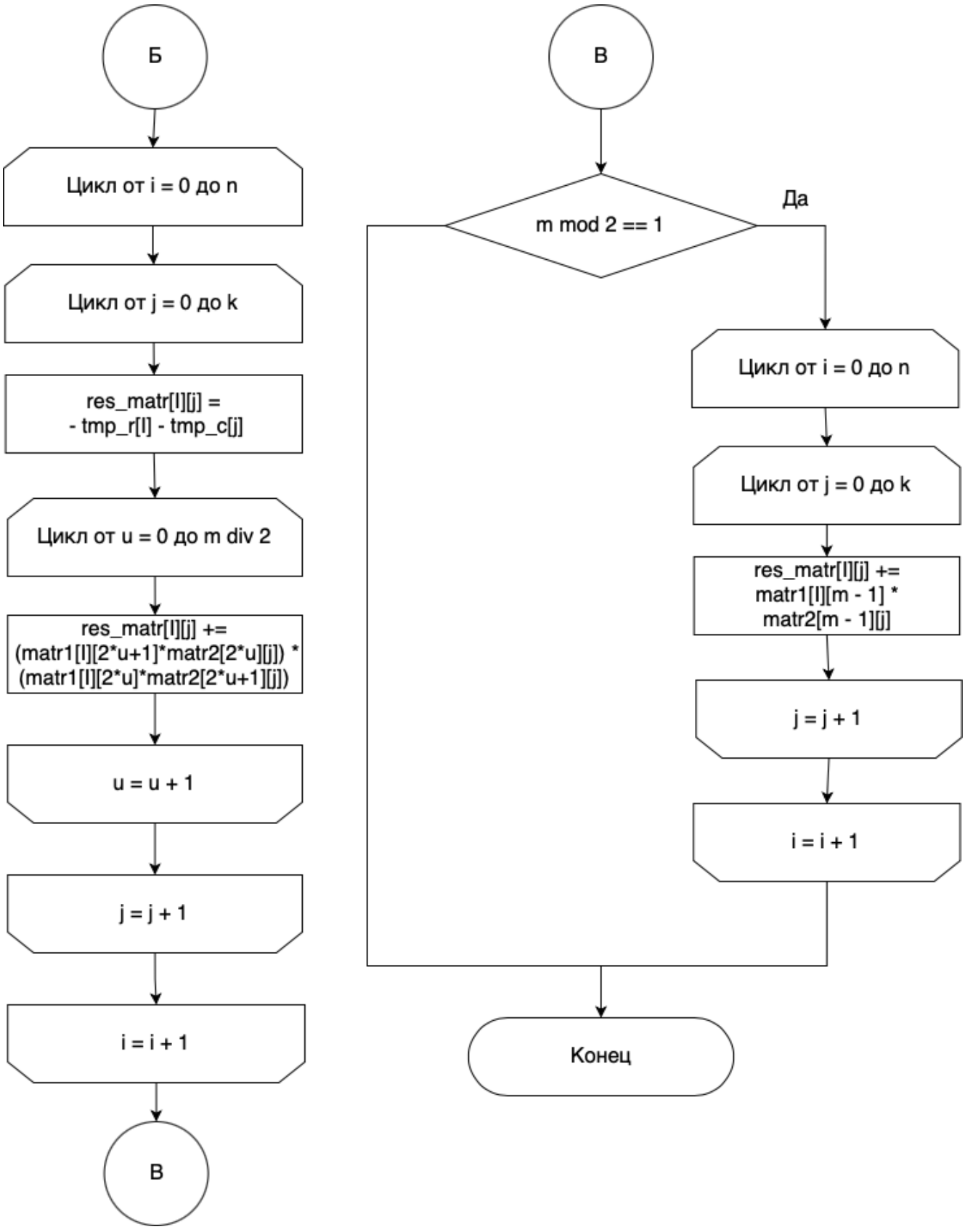

Рисунок 2.3 – Схема умножения матриц алгоритмом Винограда (продолжение)

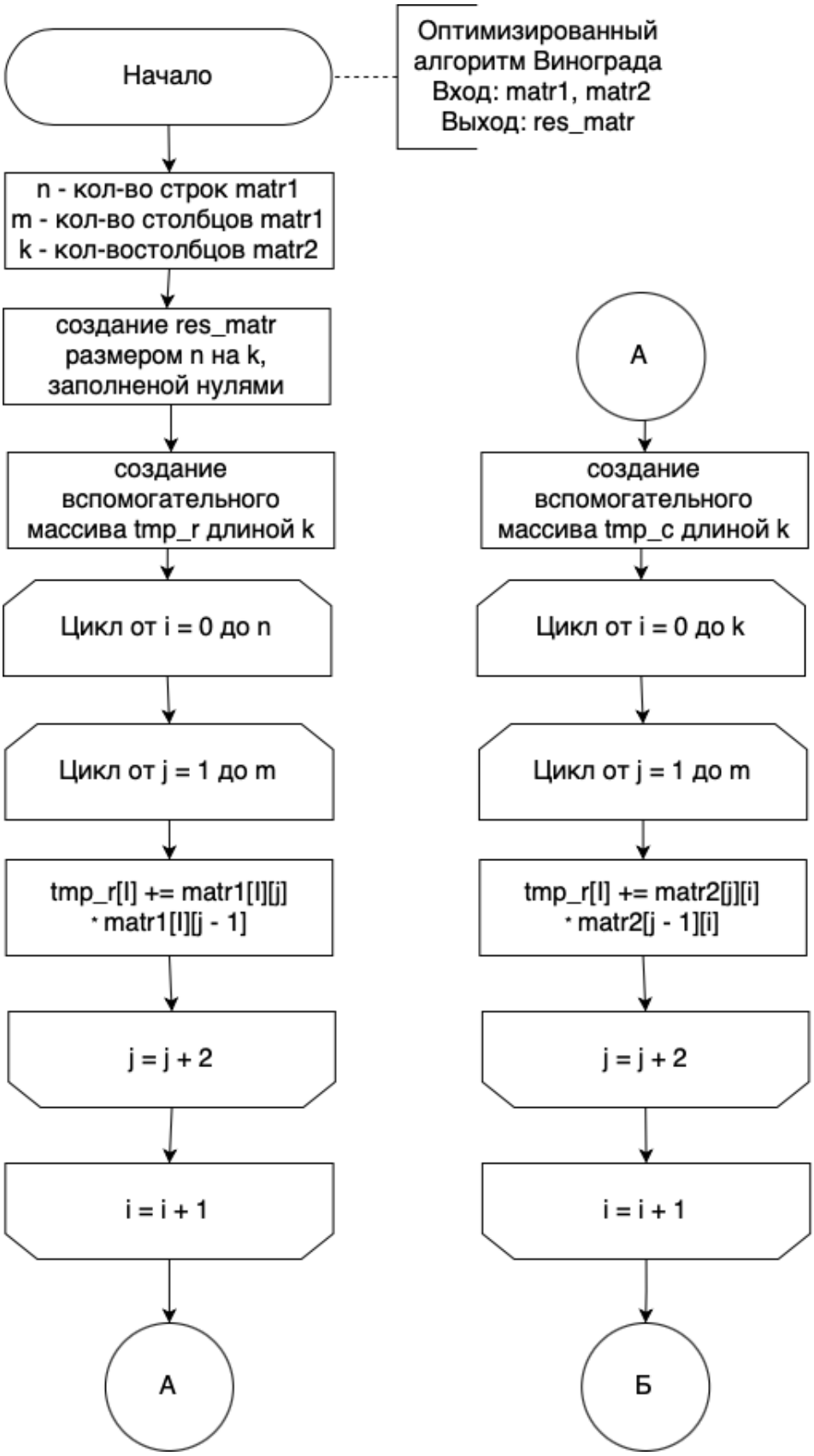

Рисунок 2.4 – Схема умножения матриц оптимизированным алгоритмом Винограда

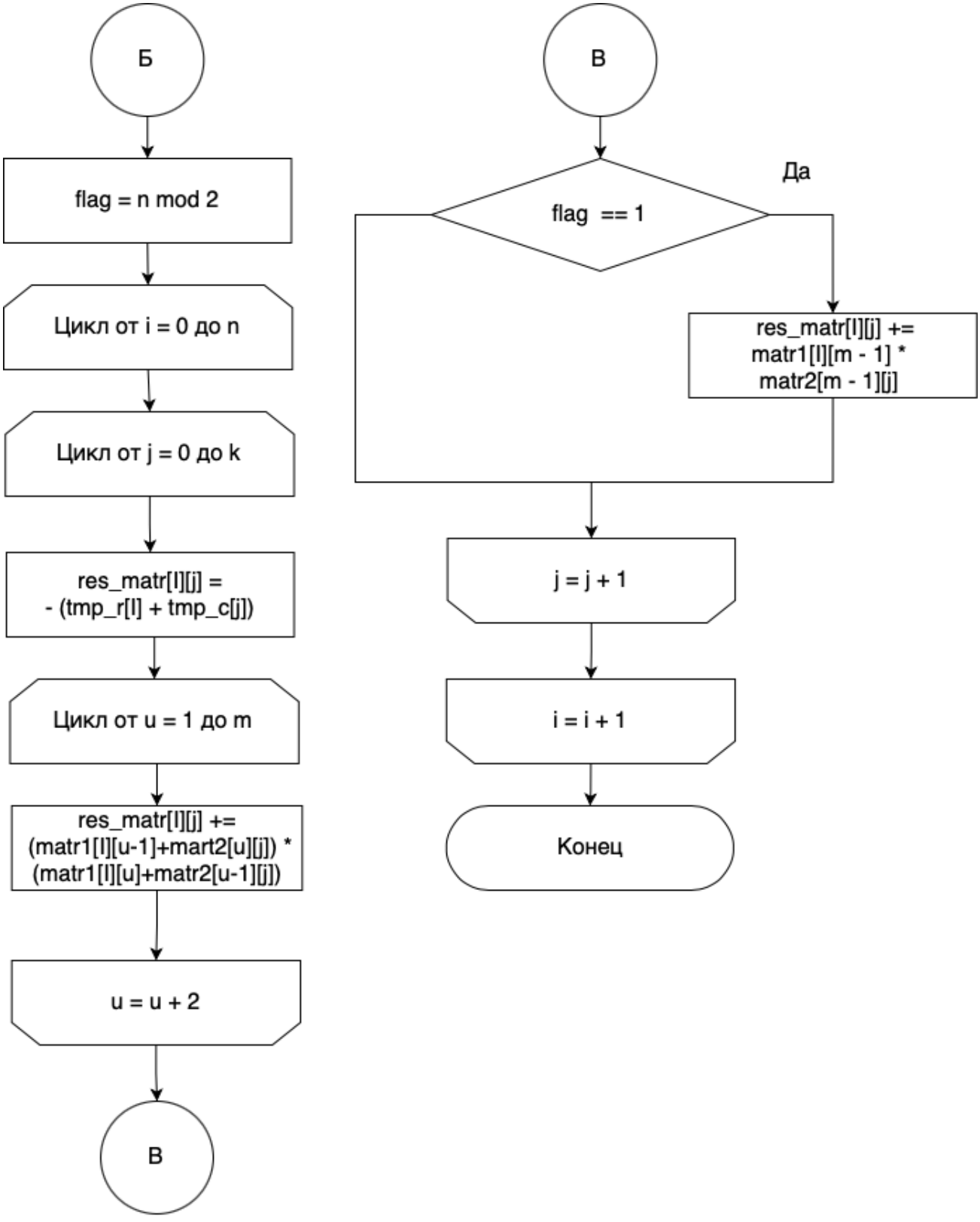

Рисунок 2.5 – Схема умножения матриц оптимизированным алгоритмом Винограда (продолжение)

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.3}$$

4. трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут расчитаны трудоемкости алгоритмов умножения матриц.

2.3.1 Стандартный алгоритм умножения матриц

Во всех последующих алгоритмах не будем учитывать инициализацию матрицы, в которую записывается результат, потому что данное действие есть во всех алгоритмах и при этом не является самым трудоёмким.

Трудоёмкость стандартного алгоритма умножения матриц состоит из:

- внешнего цикла по $i \in [1..M]$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;
- цикла по $j \in [1..N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;
- цикла по $k \in [1..K]$, трудоёмкость которого: $f = 2 + 10K$;

Учитывая, что трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела (2.4):

$$f_{standard} = 2 + M \cdot (4 + N \cdot (4 + 10K)) = 2 + 4M + 4MN + 10MNK \approx 10MNK \tag{2.4}$$

2.3.2 Алгоритм Копперсмита — Винограда

Трудоёмкость алгоритма Копперсмита — Винограда состоит из:

- создания и инициализации массивов MH и MV, трудоёмкость которого (2.5):

$$f_{init} = M + N; \tag{2.5}$$

- заполнения массива MH, трудоёмкость которого (2.6):

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 11); \tag{2.6}$$

- заполнения массива MV, трудоёмкость которого (2.7):

$$f_{MV} = 2 + K(2 + \frac{N}{2} \cdot 11); \tag{2.7}$$

- цикла заполнения для чётных размеров, трудоёмкость которого (2.8):

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)); \tag{2.8}$$

- цикла, для дополнения умножения суммой последних нечётных строки

и столбца, если общий размер нечётный, трудоемкость которого (2.9):

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \tag{2.9}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.10):

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.10}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.11):

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.11}$$

2.3.3 Оптимизированный алгоритм Копперсмита — Винограда

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

- вычисление происходит заранее;
- используется битовый сдвиг, вместо деления на 2;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

Трудоёмкость улучшенного алгоритма Копперсмита — Винограда состоит из:

- создания и инициализации массивов MH и MV, трудоёмкость которого (2.12):

$$f_{init} = M + N; \tag{2.12}$$

- заполнения массива MH, трудоёмкость которого (2.13):

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 8); \tag{2.13}$$

- заполнения массива MV, трудоёмкость которого (2.14):

$$f_{MV} = 2 + K(2 + \frac{M}{2} \cdot 8); \tag{2.14}$$

- цикла заполнения для чётных размеров, трудоёмкость которого (2.15):

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)); \tag{2.15}$$

- условие, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоемкость которого (2.16):

$$f_{last} = \begin{cases} 1, & \text{чётная,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \tag{2.16}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.17):

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.17}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.18):

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.18}$$

2.4 Классы эквивалентности

Выделенные классы эквивалентности для тестирования:

- ввод пустых матриц;
- одна из матриц - пустая;
- кол-во столбцов первой матрицы не равно кол-ву строк второй матрицы;
- перемножение квадратных матриц;
- перемножение матриц разных размеров (кол-во столбцов первой матрицы равно кол-ву строк второй матрицы).

2.5 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- кол-во строк в матрице - целое число типа *int*;
- кол-во столбцов в матрице - целое число типа *int*;
- матрица - двумерный массив типа *int*.

2.6 Структура ПО

ПО будет состоять из следующих модулей:

- *main.py* - файл, содержащий функцию *main*;
- *matrix_mult.py* - файл, содержащий код всех алгоритмов умножения матриц и функций, отвечающих за умножение;
- *compare_time.py* - файл, в котором содержатся функции для замера времени работы алгоритмов;
- *in_out_matrix.py* - файл, в котором содержатся функции ввода-вывода матриц;
- *color.py* - файл, который содержит переменные типа *string* для цветного вывода результата работы программы в консоль.

2.7 Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов умножения матриц, выбраны используемые типы данных, выделены классы эквивалентности для тестирования, а также была проведена оценка трудоёмкости алгоритмов и описана структура ПО.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги кода, а также функциональные тесты.

3.1 Требования к программному обеспечению

- входные данные - две матрицы $matr1$ и $matr2$, кол-во столбцов матрицы $matr1$ должно совпадать с кол-вом строк матрицы $matr2$;
- выходные данные - резульат умножения входных матриц (res_matr).

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [2]. Выбор обсуловлен наличием опыта работы с ним. Время работы было замерено с помощью функции *process_time* из библиотеки *time* [3].

3.3 Листинги кода

В листингах 3.1 - 3.3 представлены реализации алгоритмов умножения матриц.

Листинг 3.1 – Функция стандартного алгоритма умножения матриц

```
def classical_alg(matr1, matr2):

    n = len(matr1)
    m = len(matr1[0])
    k = len(matr2[0])

    res_matr = [[0] * k for _ in range(n)]

    for i in range(n):
        for j in range(k):
            for u in range(m):
                res_matr[i][j] += matr1[i][u] * matr2[u][j]

    return res_matr
```

Листинг 3.2 – Функция умножения матриц алгоритмом Винограда

```
1  def winograd_alg(matr1, matr2):
2
3      n = len(matr1)
4      m = len(matr1[0])
5      k = len(matr2[0])
6
7      res_matr = [[0] * k for _ in range(n)]
8
9      tmp_r = [0] * n
10     for i in range(n):
11         for j in range(0, m // 2, 1):
12             tmp_r[i] += matr1[i][2 * j] * matr1[i][2 * j + 1]
13
14     tpm_c = [0] * k
15     for i in range(k):
16         for j in range(0, m // 2, 1):
17             tpm_c[i] += matr2[2 * j][i] * matr2[2 * j + 1][i]
18
19     for i in range(n):
20         for j in range(k):
21             res_matr[i][j] = -tmp_r[i] - tpm_c[j]
22
23             for u in range(0, m // 2, 1):
24                 res_matr[i][j] += (matr1[i][2 * u + 1] + matr2[2 *
                       u    ][j]) * \
25                                   (matr1[i][2 * u    ] + matr2[2 *
                                        u + 1][j])
26
27     if m % 2 == 1:
28         for i in range(n):
29             for j in range(k):
30                 res_matr[i][j] += matr1[i][m - 1] * matr2[m - 1][j]
31
32     return res_matr
```

Листинг 3.3 – Функция умножения матриц оптимизированным алгоритмом Винограда

```
def optimized_winograd_alg(matr1, matr2):

    n = len(matr1)
    m = len(matr1[0])
    k = len(matr2[0])

    res_matr = [[0] * k for _ in range(n)]

    tmp_r = [0] * n
    for i in range(n):
        for j in range(1, m, 2):
            tmp_r[i] += matr1[i][j] * matr1[i][j - 1]

    tpm_c = [0] * k
    for i in range(k):
        for j in range(1, m, 2):
            tpm_c[i] += matr2[j][i] * matr2[j - 1][i]

    flag = n % 2
    for i in range(n):
        for j in range(k):
            res_matr[i][j] = -(tmp_r[i] + tpm_c[j])

            for u in range(1, m, 2):
                res_matr[i][j] += (matr1[i][u - 1] + matr2[u
                    ][j]) * \
                                  (matr1[i][u    ] + matr2[u -
                                      1][j])
            if flag:
                res_matr[i][j] += matr1[i][m - 1] * matr2[m - 1][j]

    return res_matr
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов умнжения матриц (стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда). Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

1-ая матрица	2-ая матрица	Ожидаемый результат
$()$	$()$	Сообщение об ошибке
$()$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 5 \end{pmatrix}$	$\begin{pmatrix} 5 \end{pmatrix}$	$\begin{pmatrix} 25 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$	$\begin{pmatrix} 14 & 32 \\ 32 & 77 \\ 50 & 122 \end{pmatrix}$

3.5 Вывод

В данном разделе были разработаны алгоритмы умножения матриц (стандартный, Винограда и оптимизированный Винограда), проведено тестирование, описаны средства реализации и требования к ПО.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: macOS 11.5.2. [4]
- Память: 8 GiB.
- Процессор: 2,3 GHz 4-ядерный процессор Intel Core i5. [5]

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

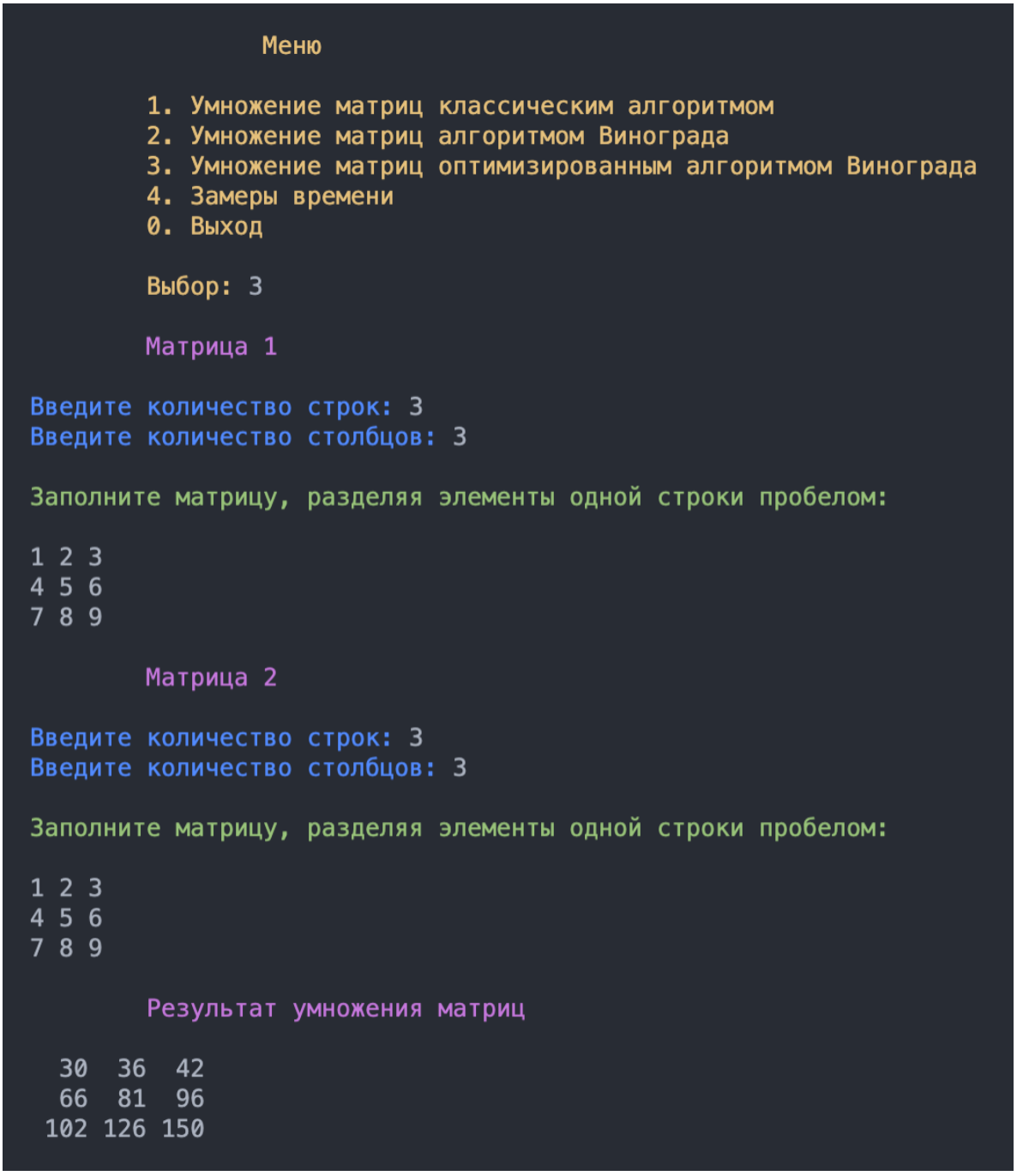

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Результаты замеров времени работы алгоритмов умножения матриц (стандартного, Винограда и оптимизированного Винограда) приведены на рисунках 4.2 - 4.5. Замеры времени проводились на квадратных матрицах чётного и нечётного размеров (в секундах) и усреднялись для каждого набора одинаковых экспериментов.

Размер	Стандартный алгоритм	Алгоритм Винограда	Оптим. алгоритм Винограда
50	2.63e-02	3.15e-02	2.31e-02
100	1.86e-01	2.22e-01	1.71e-01
150	5.81e-01	7.16e-01	5.75e-01
200	1.34e+00	1.66e+00	1.33e+00
250	2.57e+00	3.22e+00	2.56e+00
300	4.54e+00	5.70e+00	4.42e+00
350	7.21e+00	9.22e+00	7.17e+00

Рисунок 4.2 – Сравнение алгоритмов по времени при чётных размерах квадратных матриц в таблице

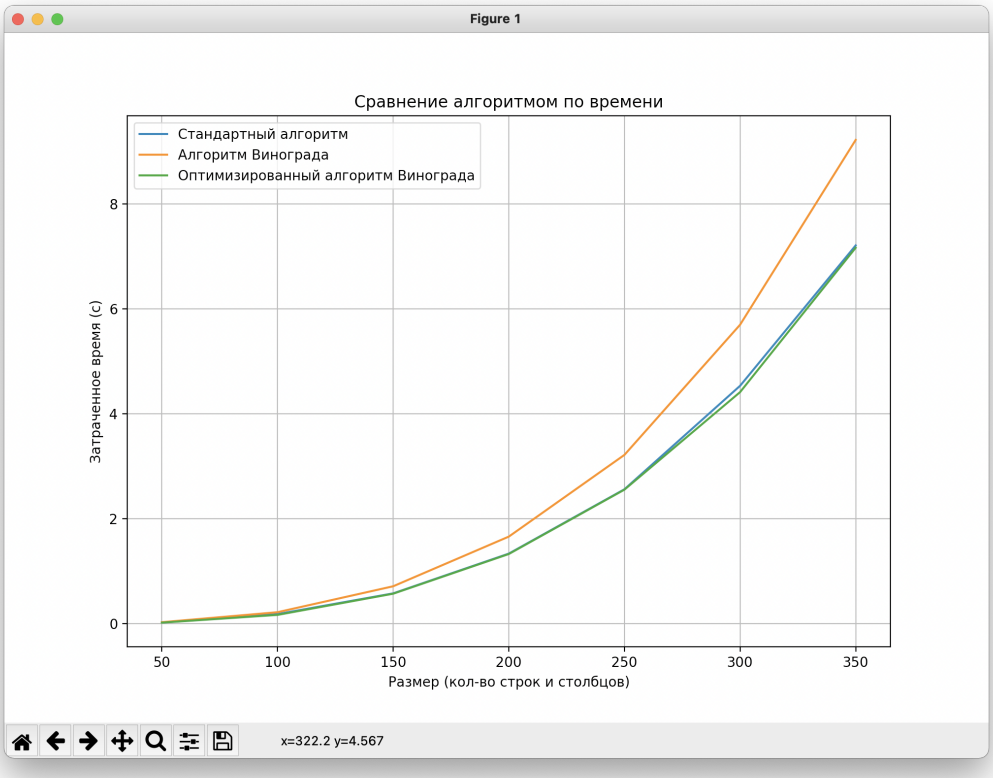

Рисунок 4.3 – Сравнение алгоритмов по времени при чётных размерах квадратных матриц

Размер	Стандартный алгоритм	Алгоритм Винограда	Оптим. алгоритм Винограда
49	2.98e-02	2.74e-02	2.10e-02
99	1.60e-01	2.04e-01	1.62e-01
149	5.43e-01	6.91e-01	5.48e-01
199	1.30e+00	1.63e+00	1.29e+00
249	2.53e+00	3.18e+00	2.52e+00
299	4.46e+00	5.90e+00	4.44e+00
349	7.10e+00	9.12e+00	7.17e+00

Рисунок 4.4 – Сравнение алгоритмов по времени при нечётных размерах квадратных матриц в таблице

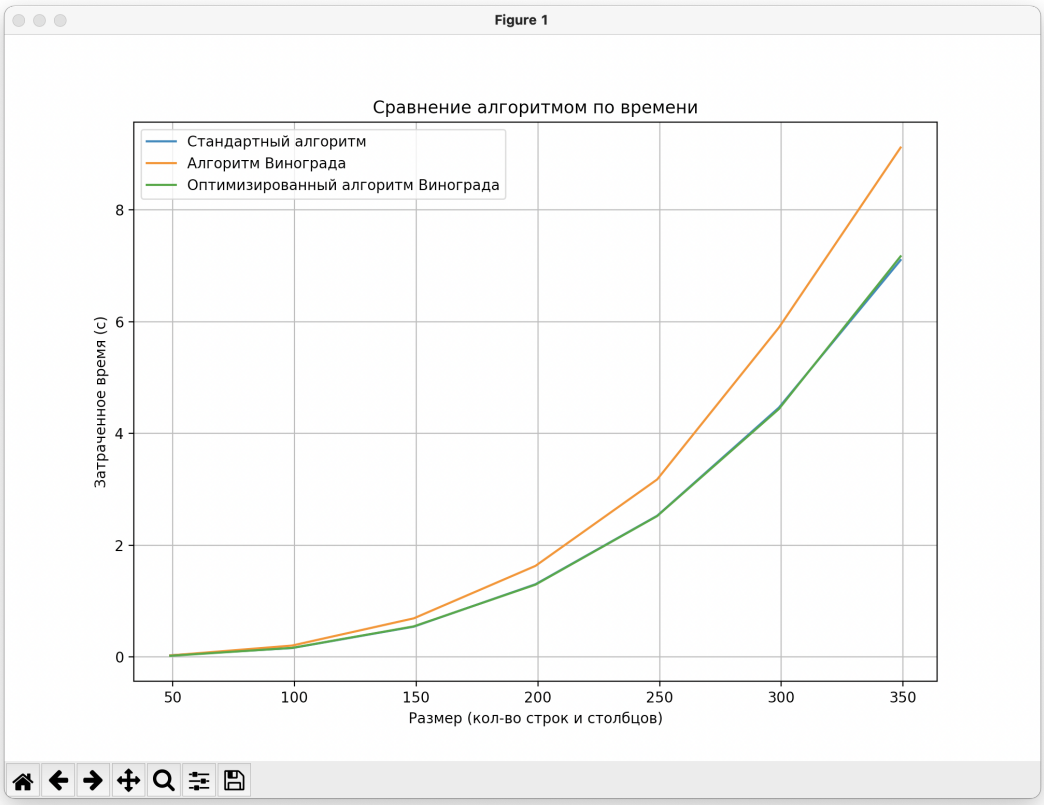

Рисунок 4.5 – Сравнение алгоритмов по времени при нечётных размерах квадратных матриц

4.4 Вывод

В этом разделе были указаны технические характеристики машины, на которой происходило сравнение времени работы алгоритмов умножения матриц, а также приведены результаты исследования.

Сравнения проводились на квадратных матрицах чётного и нечётного размеров. В обоих случаях алгоритм Винограда показал наихудший результат. На матрицах размером 200x200 он работал на 24.8% времени дольше, чем оптимизированный алгоритм Винограда и на 23.8% дольше стандартного алгоритма. На матрицах 199x199 на 26.3% и 25.3% соответственно. На матрицах большего размера разница оставалась похожей (28.5% и 27.8% на матрицах 350x350 и 27.2% и 28.5% на матрицах 349x349). Стандартный алгоритм умножения матриц и оптимизированный алгоритм Винограда показали схожие результаты при чётных и нечётных размерах матриц. На матрицах размером 350x350 оптимизированный алгоритм Винограда работает на 0.6% быстрее стандартного алгоритма, а на матрицах 349x349 на 1.0% медленее его.

Заключение

Было экспериментально подтверждено различие во временной эффективности стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

В результате исследований можно сделать вывод о том, что алгоритм Винограда неэффективен по времени на матрицах малого размера (350x350 и меньше).

В ходе выполнения данной лабораторной работы были решены следующие задачи:

- изучены стандартный алгоритма умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда;
- реализованы указанные алгоритмы;
- проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- экспериментально подтверждено различие во временной эффективности алгоритмов умножения матриц при помощи разработанного программного обеспечения на материале замеров процессорного времени;
- описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Поставленная цель была достигнута.

Литература

[1] Умножение матриц [Электронный ресурс], howpublished="Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 18.10.2021).

[2] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 18.10.2021).

[3] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 18.10.2021).

[4] macOS Monterey - Apple(RU) [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/macos/monterey/ (дата обращения: 18.10.2021).

[5] Intel [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/details/processors/core/i5.html (дата обращения: 18.10.2021).