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 Цветков И.А.

Группа ИУ7-53Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

В математике и программировании часто приходится прибегать к использованию матриц. Существует огромное количество областей их применения в этих сферах. Например, матрицы активно используются при выводе различных формул в физике:

- градиент;
- дивергенция;
- ротор.

Также часто применяются и операции над матрицами - сложение, возведение в степень, умножение. При различных задачах размеры матрицы могут достигать больших значений. Поэтому оптимизация операций работы над матрицами является важной задачей в программировании. Об оптимизации операции умножения пойдет речь в данной лабораторной работе.

Целью данной работы является изучение, реализация и исследование алгоритмов умножения матриц - классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить и реализовать алгоритмы - классический, Винограда и его оптимизацию;
- провести тестирование по времени и по памяти для алгоритмов лабораторной работы;
- провести сравнительный анализ по времени классического алгоритма и алгоритма Винограда;
- провести сравнительный анализ по времени алгоритма Винограда и его оптимизации;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут представлены классический алгоритм умножения матриц и алгоритм Винограда.

1.1 Матрица

Матрица [1] - математический объект, который представляет собой двумерный массив, в котором элементы располагаются по строкам и столбцам.

Пусть A - матрица, тогда $A_{i,j}$ - элемент этой матрицы, который находится на *i-ой* строке и *j-ом* столбце.

Можно выделить следующие операции над матрицами:

- матрицы одинакового размера можно складывать и вычитать;
- количество столбцов одной матрицы равно количеству строк другой матрицы - их можно перемножить, причем количество строк будет, как у первой матрицы, а столбцов - как у второй.

Замечание: операция умножения матриц не коммутативна - если A и B - квадратные матрицы, а C - результат их перемножения, то произведение AB и BA дадут разный результат C.

1.2 Стандартный алгоритм

Пусть даны две 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.

1.3 Алгоритм Винограда

Алгоритм Винограда [2] — алгоритм умножения квадратных матриц. Начальная версия имела асимптотическую сложность алгоритма примерно $O(n^{2,3755})$, где n - размер стороны матрицы, но после доработки он стал обладать лучшей асимптотикой среди всех 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):

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

Прирост производительности заключается в идее предварительной обработки - полученное выражение требует большего количества операций, чем стандартное умножение матриц, но выражение в правой части крайнего равенства можно вычислить заранее и запомнить для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволит выполнить лишь два умножения и пять сложений, при учете, что потом будет сложено только с двумя предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее обычного алгоритма перемножения матриц.

Но стоит упомянуть, что при нечетном значении размера матрицы нужно

дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.4 Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц - стандартного и Винограда, который имеет большую эффективность за счет предварительных вычислений.

Алгоритмы на вход будут получать две матрицы, при чем количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы. При вводе пустой матрицы будет выведено сообщение об ошибке. Реализуемое ПО дает возможность выбрать один из алгоритмов или все сразу, ввести две матрицы и вывести результат перемножить, а также можно провести тестирование по времени для четных и нечетных размеров матриц и сравнить результаты, используя их графическое представление.

2 Конструкторская часть

В этом разделе будут представлено описание используемых типов данных, а также схемы алгоритмов перемножения матриц - стандартного, Винограда и оптимизации алгоритма Винограда.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

- количество строк - целое число типа *int*
- количество столбцов - целое число типа *int*
- матрица - двумерный список типа *int*

2.2 Сведения о модулях программы

Программа состоит из двух модулей:

- *main.py* - файл, содержащий весь служебный код;
- *algorythms.py* - файл, содержащий код всех алгоритмов перемножения матриц.

2.3 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма для стандартного умножения матриц. На рисунках 2.2-2.3 схема алгоритма Винограда умножения матриц, а на 2.4-2.5 - схема оптимизированного алгоритма Винограда.

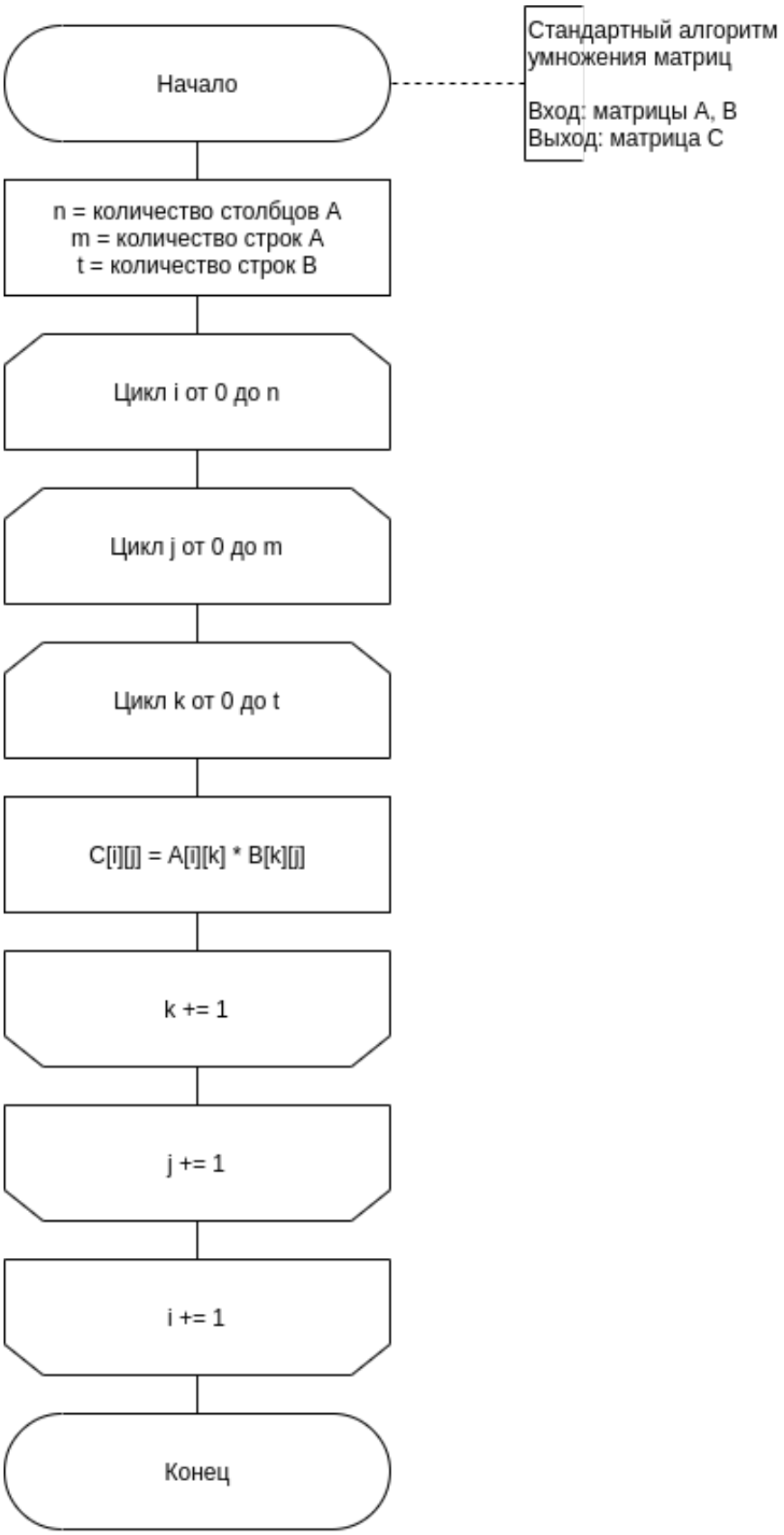

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

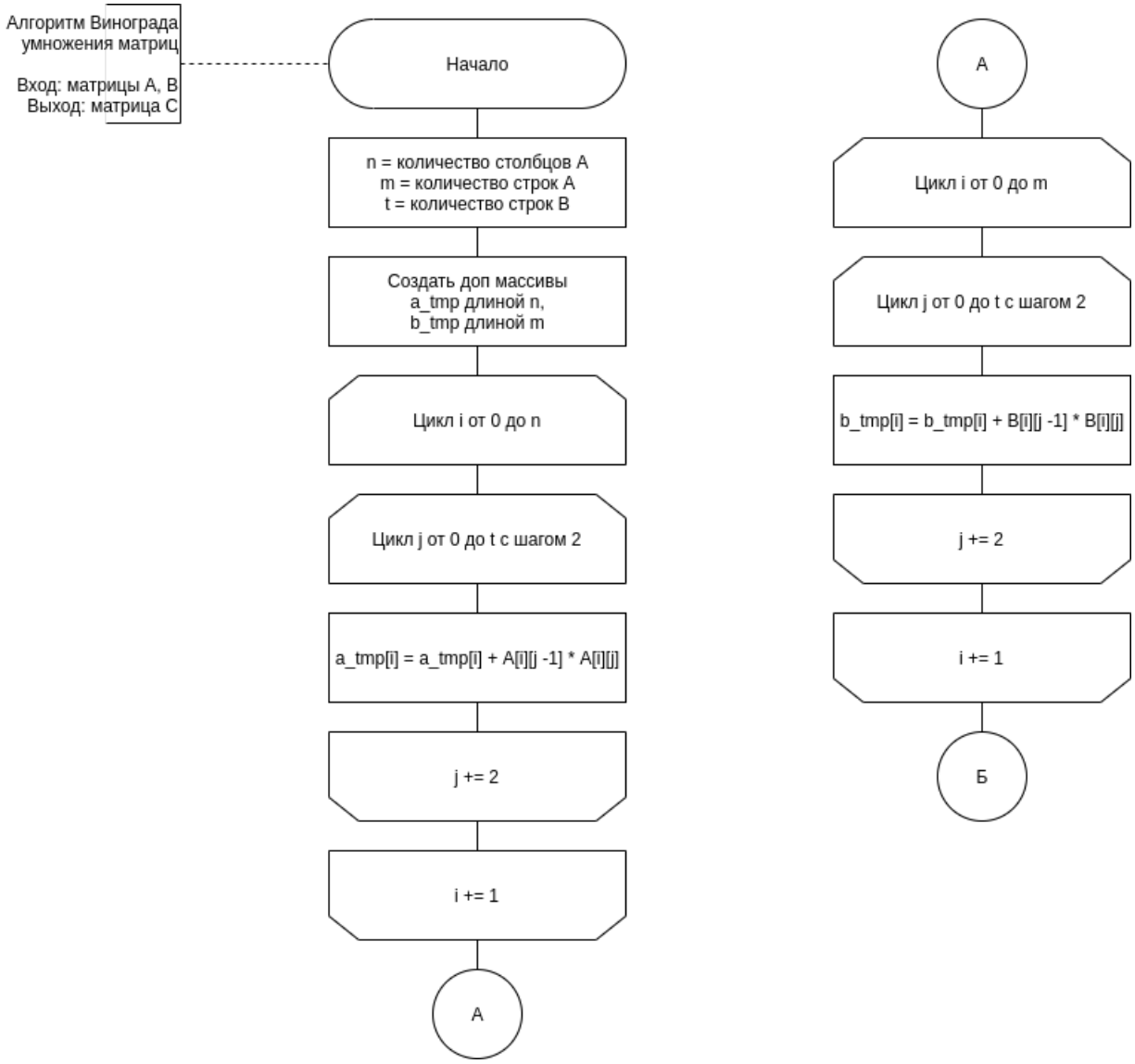

Рисунок 2.2 – Схема умножения матриц по алгоритму Винограда (часть 1)

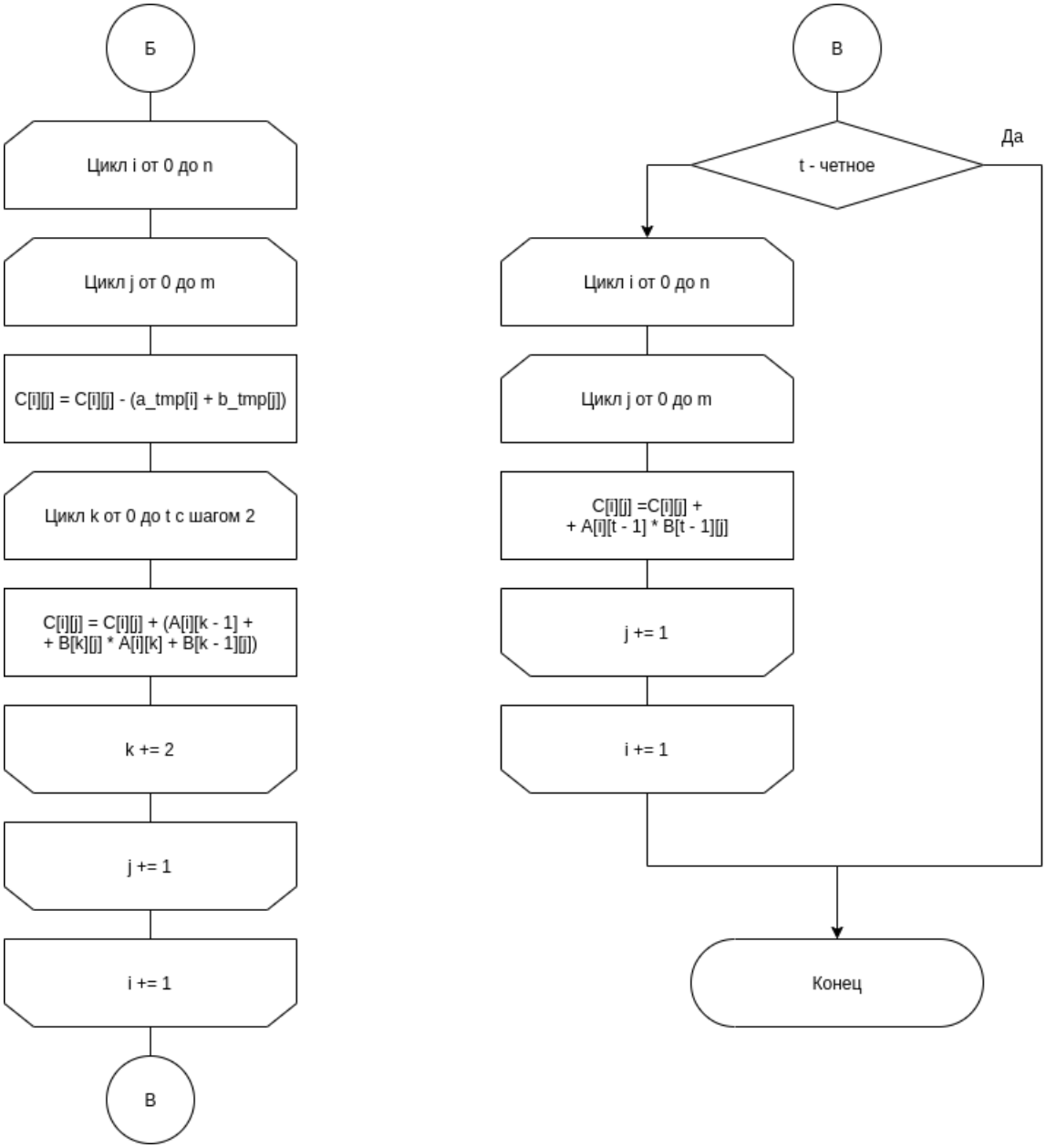

Рисунок 2.3 – Схема умножения матриц по алгоритму Винограда (часть 2)

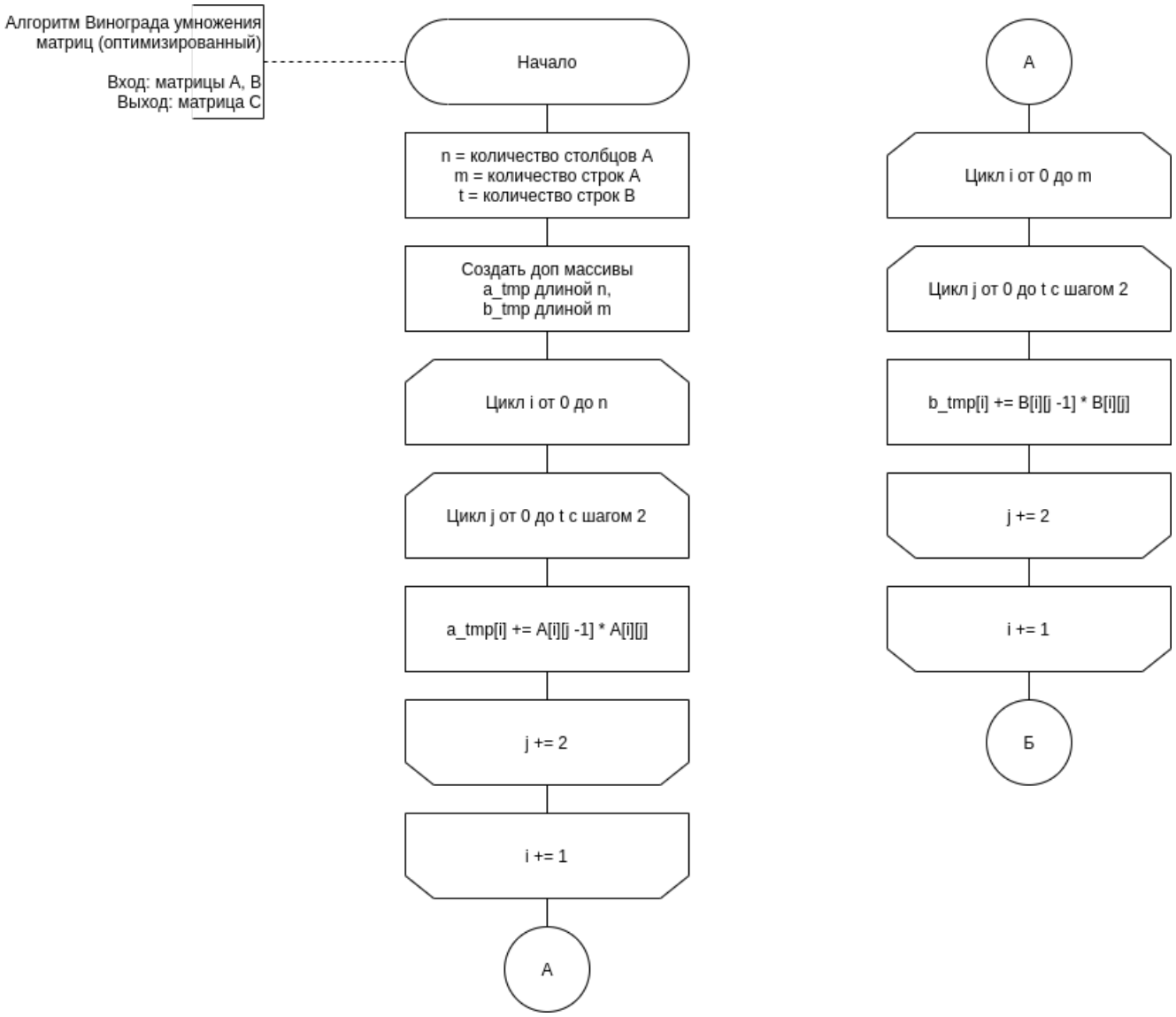

Рисунок 2.4 – Схема умножения матриц по оптимизированному алгоритму Винограда (часть 1)

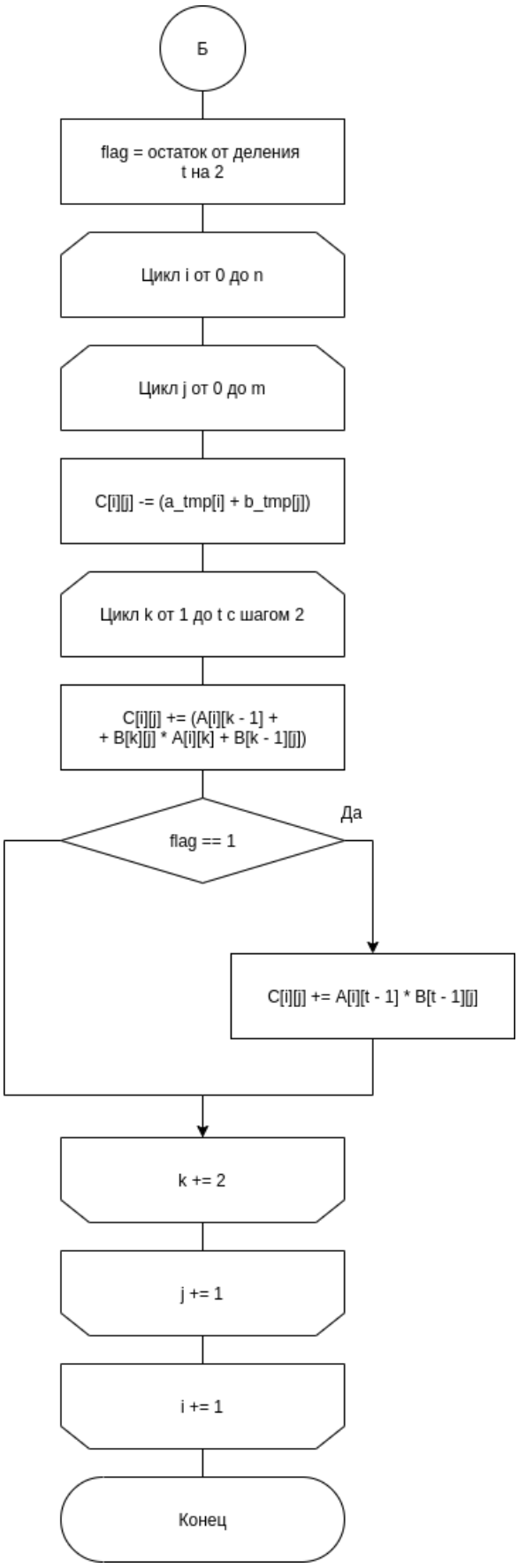

Рисунок 2.5 – Схема умножения матриц по оптимизированному алгоритму Винограда (часть 2)

2.4 Модель вычислений

Чтобы провести вычисление трудоемкости алгоритмов умножения матриц, введем модель вычислений [3]:

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.

2.5 Трудоемкость алгоритмов

Рассчитаем трудоемкость алгоритмов умножения матриц.

2.5.1 Стандартный алгоритм умножения матриц

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться 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: $f = 2 + 10K$.

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

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

2.5.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, нужно учесть следующее:

- создания и инициализации массивов a_tmp и b_tmp, трудоёмкость которых (2.5):

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

- заполнения массива a_tmp, трудоёмкость которого (2.6):

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

- заполнения массива b_tmp, трудоёмкость которого (2.7):

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

- 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, который дополнительно нужен для подсчета значений при нечет-

ном размере матрицы, трудоемкость которого (2.9):

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

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

$$f_{worst} = f_{a\_tmp} + f_{b\_tmp} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.10}$$

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

$$f_{best} = f_{a\_tmp} + f_{a\_tmp} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.11}$$

2.5.3 Оптимизированный алгоритм Винограда

Оптимизация заключается в:

- использовании побитового сдвига вместо деления на 2;
- цикл, который был вынесен при нечетном размере матрицы, был занесен в общий цикл, тем самым в общем цикле происходят дополнительные вычисления при нечетном размере матрицы;
- операции сложения и вычитания заменены на операции $+=$ и $-=$ соответственно.

Тогда трудоемкость оптимизированного алгоритма Винограда состоит из:

- создания и инициализации массивов a_tmp и b_tmp (2.5);
- заполнения массива a_tmp, трудоёмкость которого (2.6);
- заполнения массива MV, трудоёмкость которого (2.7);
- цикла заполнения для чётных размеров, трудоёмкость которого (2.12):

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)); \tag{2.12}$$

- условие, которое нужно для дополнительных вычислений при нечетном размере матрицы, трудоемкость которого (2.13):

$$f_{last} = \begin{cases} 1, & \text{чётная,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \tag{2.13}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем (2.14):

$$f_{worst} = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.14}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.15):

$$f_{best} = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.15}$$

2.6 Классы эквивалентности при тестировании

Для тестирования выделены классы эквивалентности, представленные ниже.

1. Одна из матриц - пустая.
2. Количество столбцов одной матрицы не равно количеству строк второй матрицы.
3. Перемножение квадратных матриц.
4. Перемножение матриц разных размеров (при этом количество столбцов одной матрицы не равно количеству строк второй матрицы).

2.7 Вывод

В данном разделе были построены схемы алгоритмов умножения матриц рассматриваемых в лабораторной работе, были описаны классы эквивалент-

ности для тестирования, модули программы, а также проведена теоретическая оценка трудоемкости алгоритмов.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритмов умножения матриц - стандартный, Винограда и оптимизированный алгоритм Винограда.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*[4]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time*[5].

3.2 Листинги кода

В листингах 3.1-3.3 представлены реализации алгоритмов умножения матриц - стандартного, Винограда и оптимизированного алгоритма Винограда.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
1  def standart_alg(mat_a, mat_b):
2      n = len(mat_a)
3      m = len(mat_a[0])
4      t = len(mat_b[0])
5      res_matrix = [[0] * t for _ in range(n)]
6
7      for i in range(n):
8          for j in range(m):
9              for k in range(t):
10                 res_matrix[i][j] += mat_a[i][k] * mat_b[k][j]
11
12     return res_matrix
```

Листинг 3.2 – Алгоритм Винограда умножения матриц

```
def vinograd_alg(mat_a, mat_b):

    n = len(mat_a)
    m = len(mat_a[0])
    t = len(mat_b[0])

    res_matrix = [[0] * t for _ in range(n)]

    tmp_row = [0] * n
    tmp_col = [0] * t

    for i in range(n):
        for j in range(0, m // 2):
            tmp_row[i] = tmp_row[i] + mat_a[i][2 * j] * mat_a[i][2
               * j + 1]

    for i in range(t):
        for j in range(0, m // 2):
            tmp_col[i] = tmp_col[i] + mat_b[2 * j][i] * mat_b[2 * j
               + 1][i]


    for i in range(n):
        for j in range(t):

            res_matrix[i][j] = -tmp_row[i] - tmp_col[i]

            for k in range(0, m // 2):

                res_matrix[i][j] = res_matrix[i][j] + (mat_a[i][2 *
                   k + 1] + mat_b[2 * k][j]) * (mat_a[i][2 * k] +
                   mat_b[2 * k + 1][j])

    if (m % 2 == 1):
        for i in range(n):
            for j in range(t):
                res_matrix[i][j] = res_matrix[i][j] + mat_a[i][m -
                   1] * mat_b[m - 1][j]

    return res_matrix
```

Листинг 3.3 – Оптимизированный алгоритм Винограда умножения матриц

```
def optimized_vinograd_alg(mat_a, mat_b):

    n = len(mat_a)
    m = len(mat_a[0])
    t = len(mat_b[0])

    res_matrix = [[0] * t for _ in range(n)]

    tmp_row = [0] * n
    tmp_col = [0] * t

    for i in range(n):
        for j in range(0, m // 2):
            tmp_row[i] = tmp_row[i] + mat_a[i][2 * j] * mat_a[i][2
               * j + 1]

    for i in range(t):
        for j in range(0, m // 2):
            tmp_col[i] = tmp_col[i] + mat_b[2 * j][i] * mat_b[2 * j
               + 1][i]


    flag = m % 2

    for i in range(n):
        for j in range(t):

            res_matrix[i][j] -= (tmp_row[i] + tmp_col[i])

            for k in range(1, m, 2):

                res_matrix[i][j] += (mat_a[i][k - 1] + mat_b[k][j])
                   * (mat_a[i][k] + mat_b[k - 1][j])

            if (flag):
                res_matrix[i][j] += mat_a[i][m - 1] * mat_b[m -
                   1][j]

    return res_matrix
```

3.3 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы умножения матриц, рассматриваемых в данной лабораторной работе. Тесты *для всех алгоритмов* пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$

3.4 Вывод

Были представлены листинги всех алгоритмов умножения матриц - стандартного, Винограда и оптимизированного алгоритма Винограда. Также в данном разделе была приведена информации о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программа, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

- операционная система: Ubuntu 20.04.3 [6] Linux [7] x86_64;
- память: 8 GiB;
- процессор: Intel® Core™ i5-7300HQ CPU @ 2.50GHz [8].

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

На рисунке 4.1 представлен результат работы программы.

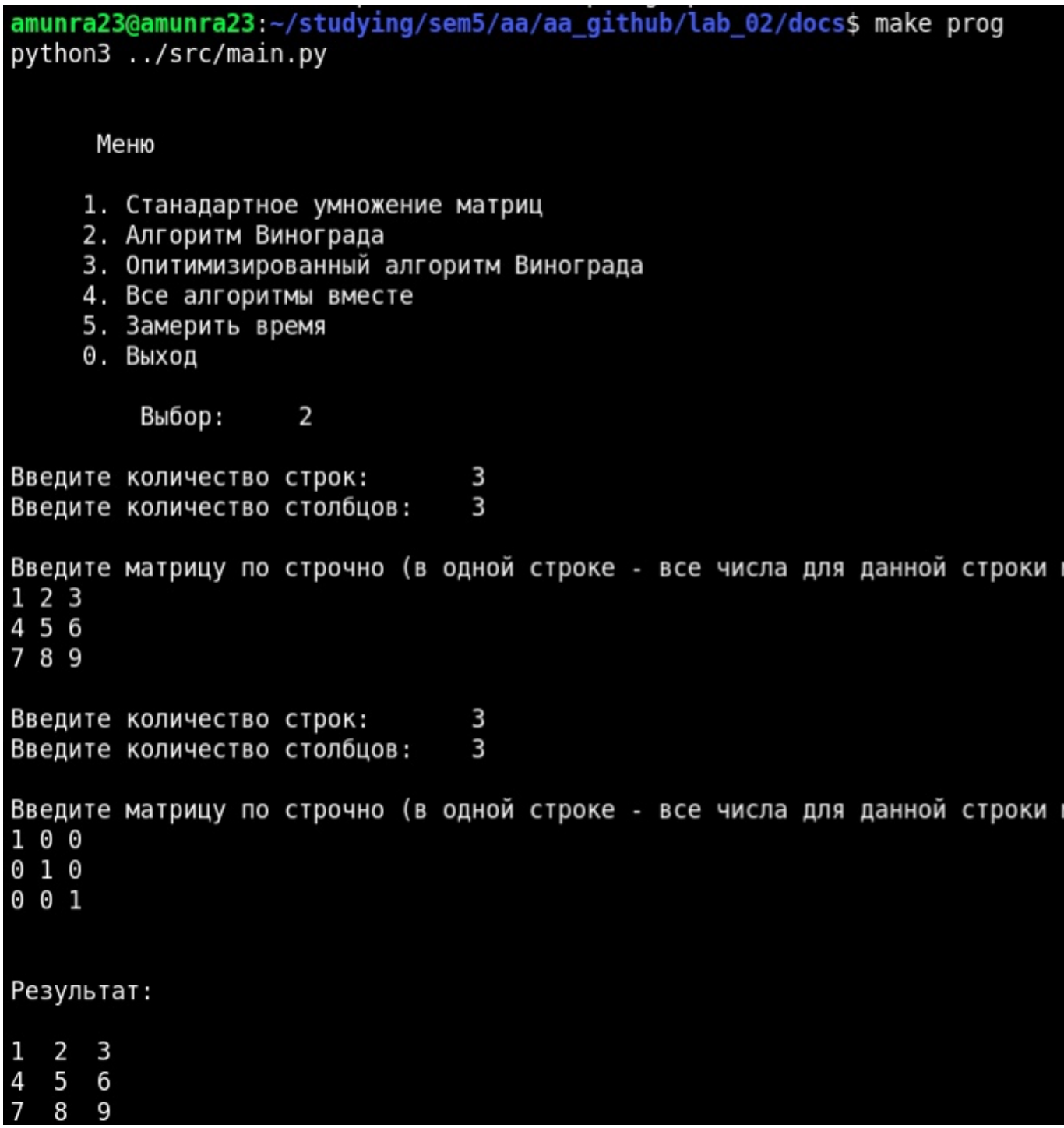

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени process_time(...) из библиотеки time на Python. Функция возвращает пользовательское процессорное время типа float.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для четных размеров матриц от 10 до 100 по 100 раз на различных входных матрицах. А также - для нечетных размеров матриц от 11 до 101 по 100 раз на различных данных.

Результаты замеров приведены в таблицах 4.1-4.2 (время в мс).

Таблица 4.1 – Результаты замеров времени (четные размеры матриц)

Размер	Виноград	Стандартный	Виноград (опт)
10	0.3189	0.2338	0.1962
20	1.2911	1.5249	1.2745
30	4.2726	4.9073	4.2689
40	10.3607	11.9060	9.2739
50	19.8466	22.4358	20.1878
60	38.2797	42.2143	32.1537
70	54.8869	56.4940	55.7448
80	74.8601	87.8505	69.6352
90	113.5103	126.9989	101.6805
100	162.9821	179.0071	142.2032

Таблица 4.2 – Результаты замеров времени
(нечетные размеры матриц)

Размер	Виноград	Стандартный	Виноград (опт)
11	0.3610	0.3169	0.2721
21	1.5591	1.8751	1.5605
31	4.5730	5.3056	4.4258
41	11.2068	13.3446	10.5558
51	21.6583	24.3008	20.1238
61	37.2238	45.4626	37.7527
71	61.7743	69.6095	60.5679
81	95.3371	98.5012	75.7777
91	125.1572	135.4359	110.7858
101	157.5460	182.0805	139.2139

Также на рисунках 4.2–4.3 приведены графические результаты замеров.

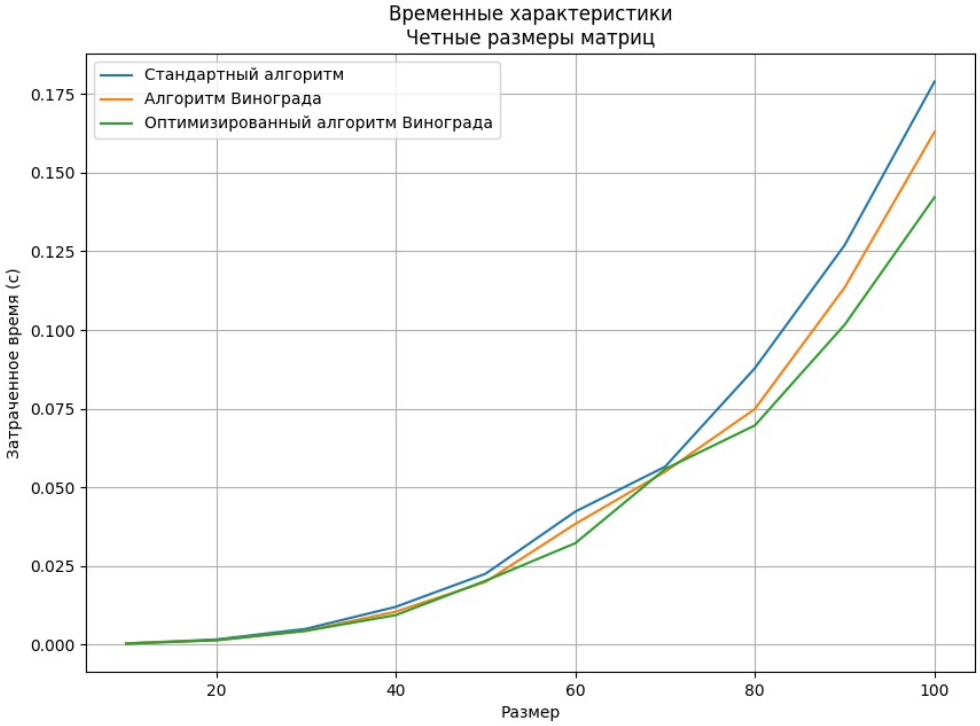

Рисунок 4.2 – Сравнение по времени алгоритмов умножения матриц на четных размерах матриц

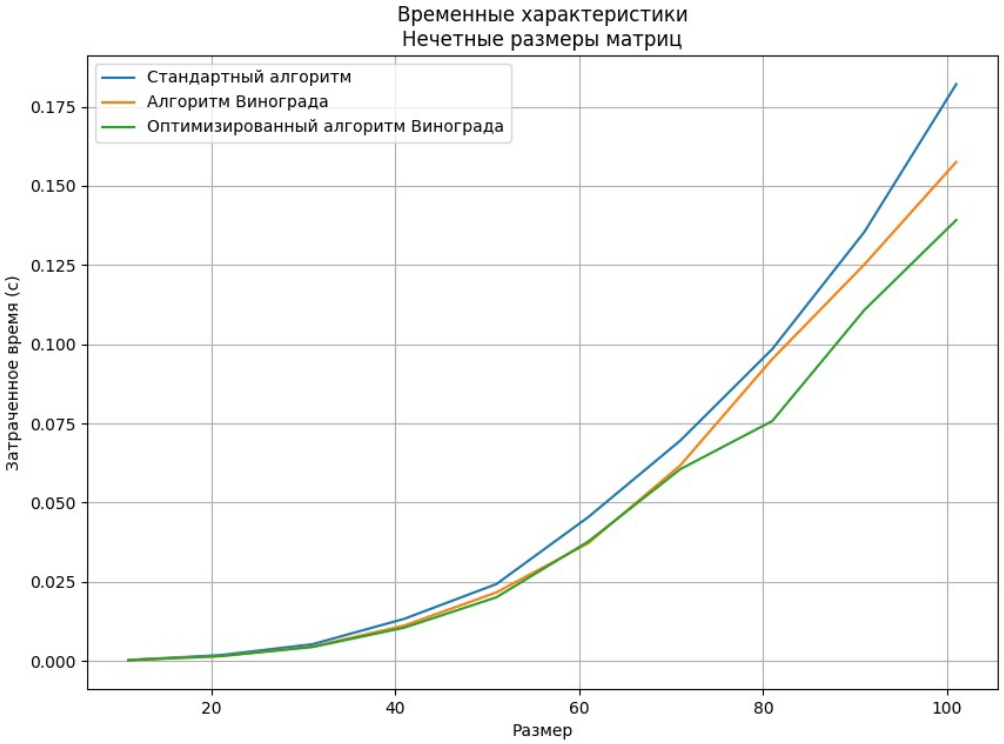

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц на нечетных размерах матриц

4.4 Вывод

В результате эксперимента было получено, что при больших размерах матриц (свыше 10), алгоритм Винограда быстрее стандартного алгоритма более, чем 1.2 раза, а оптимизированный алгоритм Винограда быстрее стандартного алгоритма в 1.3 раза. В итоге, можно сказать, что при таких данных следует использовать оптимизированный алгоритм Винограда.

Также при проведении эксперимента было выявлено, что на четных размерах реализация алгоритма Винограда в 1.2 раза быстрее, чем на нечетных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов. Следовательно, стоит использовать алгоритм Винограда для матриц, которые имеют четные размеры.

Заключение

В результате исследования было определено, что стандартный алгоритм умножения матриц проигрывает по времени алгоритму Винограда примерно в 1.2 раза из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций - операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает оптимизированный алгоритм Винограда – он примерно в 1.2 раза быстрее алгоритма Винограда на размерах матриц свыше 10 из-за замены операций равно и плюс на операцию плюс-равно, а также за счет замены операции умножения операцией сдвига, что дает проводить часть вычислений быстрее. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда. Также стоит упомянуть, что алгоритм Винограда работает на четных размерах матриц примерно в 1.3 раза быстрее, чем на нечетных, что связано с тем, что нужно произвести часть дополнительных вычислений для крайних строк и столбцов матриц, поэтому алгоритм Винограда лучше работает четных размерах матриц.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи:

- были изучены и реализованы алгоритмы умножения матриц - стандартный, Винограда и оптимизированный алгоритм Винограда;
- проведен сравнительный анализ по времени алгоритмов умножения матриц на четных размерах матриц
- проведен сравнительный анализ по времени алгоритмов умножения матриц на нечетных размерах матриц
- проведен сравнительный анализ по времени алгоритмов алгоритмов между собой
- подготовлен отчет о лабораторной работе.

Список литературы

[1] Матрица [Электронный ресурс]. Режим доступа: https://terme.ru/termin/matrica.html (дата обращения: 23.10.2021).

[2] Умножение матриц [Электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 23.10.2021).

[3] М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. 2007.

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 04.10.2021).

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 04.10.2021).

[6] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 04.10.2021).

[7] Linux [Электронный ресурс]. Режим доступа: https://www.linux.org/forums/#linux-tutorials.122 (дата обращения: 04.10.2021).

[8] Процессор Intel® Core™ i5-7300HQ [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/97456/intel-core-i5-7300hq-processor-6m-cache-up-to-3-50-ghz.html (дата обращения: 04.10.2021).