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 дисциплине "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Задача удобного представления и хранения информации возникает в математике, физике, экономике, статистике и программировании. Для решения этой задачи используют матрицу.

Матрица - массив, элементы которого являются массивами [1]. Эти массивы называют строками матрицы. Элементы, стоящие на одной и той же позиции в разных строках, образуют столбец матрицы.

Над матрицами можно проводить следующие операции: сложение, вычитание, транспонирование, возведение в степень. Одной из часто применяющихся операций является умножение матриц. Например, в машинном обучении реализации распространения сигнала в слоях нейронной сети базируются на умножении матриц. Так, актуальной задачей является оптимизация алгоритмов матричного умножения.

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

- изучить алгоритмы стандартного умножения матриц и алгоритм Винограда;
- рассмотреть оптимизацию указанного алгоритма;
- привести схемы изучаемых алгоритмов;
- провести сравнительный анализ по трудоемкости рассмотренных алгоритмов;
- описать используемые типы и структуры данных;
- описать структуру разрабатываемого программного обеспечения;
- определить средства программной реализации выбранных алгоритмов;
- реализовать разработанные алгоритмы;
- провести функциональное тестирование программного обеспечения;

- провести сравнительный анализ по времени реализованных алгоритмов;
- подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут описаны алгоритмы умножения матриц: стандартный, Винограда и оптимизация алгоритма Винограда.

1.1 Стандартный алгоритм матричного умножения

Пусть даны две матрицы A и B размерности $N \cdot P$ и $P \cdot M$ соответсвенно:

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

Тогда матрица C размерностью N*M:

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

в которой:

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

называется их произведением.

Для вычисления произведения двух матриц, каждая строка первой матрицы почленно умножается на каждый столбец второй, затем подсчитывается сумма таких произведений и записывается в соответствующий элемент результирующей матрицы [2].

1.2 Алгоритм Винограда умножения матриц

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

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):

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

Выражение в правой части формулы 1.4 допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. При нечетном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов к результату [2].

1.3 Оптимизация алгоритма Винограда умножения матриц

Оптимизированная версия алгоритма Винограда [2] отличается:

- использованием битового сдвига, вместо деления на 2;
- последний цикл для нечетных элементов включен в основной цикл, используя дополнительные операции в случае нечетности;
- замены операций сложения и вычитания на += и -= соответственно.

1.4 Вывод

Были изучены способы умножения матриц при помощи классического алгоритма и алгоритма Винограда. Также была рассмотрена оптимизация алгоритма Винограда.

Программе, реализующей данные алгоритмы, на вход будут подаваться две матрицы. Выходными данными такой программы должна быть матрица - результат умножения введенных. Программа должна работать в рамках следующих ограничений:

- элементы матрицы - целые числа;
- количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы;
- должно быть выдано сообщение об ошибке при вводе пустых матриц.

Пользователь должен иметь возможность выбора алгоритма матричного умножения и вывода результата на экран. Также должны быть реализованы сравнение алгоритмов по времени работы с выводом результатов на экран и получение графического представления результатов сравнения. Данные действия пользователь должен выполнять при помощи меню.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов умножения матриц: стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда. Будут описаны типы и структуры данных, используемые для реализации, а также структура разрабатываемого программного обеспечения. Кроме того, будут выделены классы эквивалентности для тестирования.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц. Схема умножения матриц по алгоритму Винограда приведена на рисунках 2.2-2.4, схема оптимизированной версии приведена на рисунках 2.5-2.7.

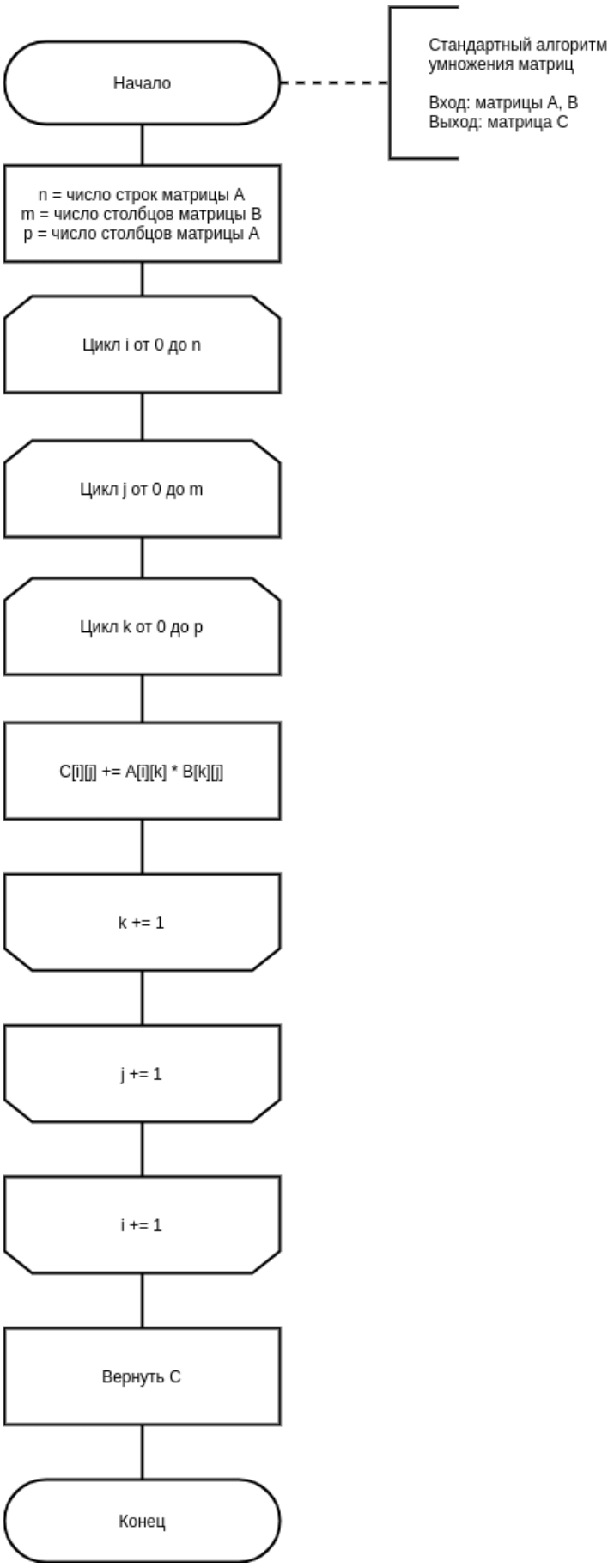

Рисунок 2.1 – Стандартный алгоритм умножения матриц

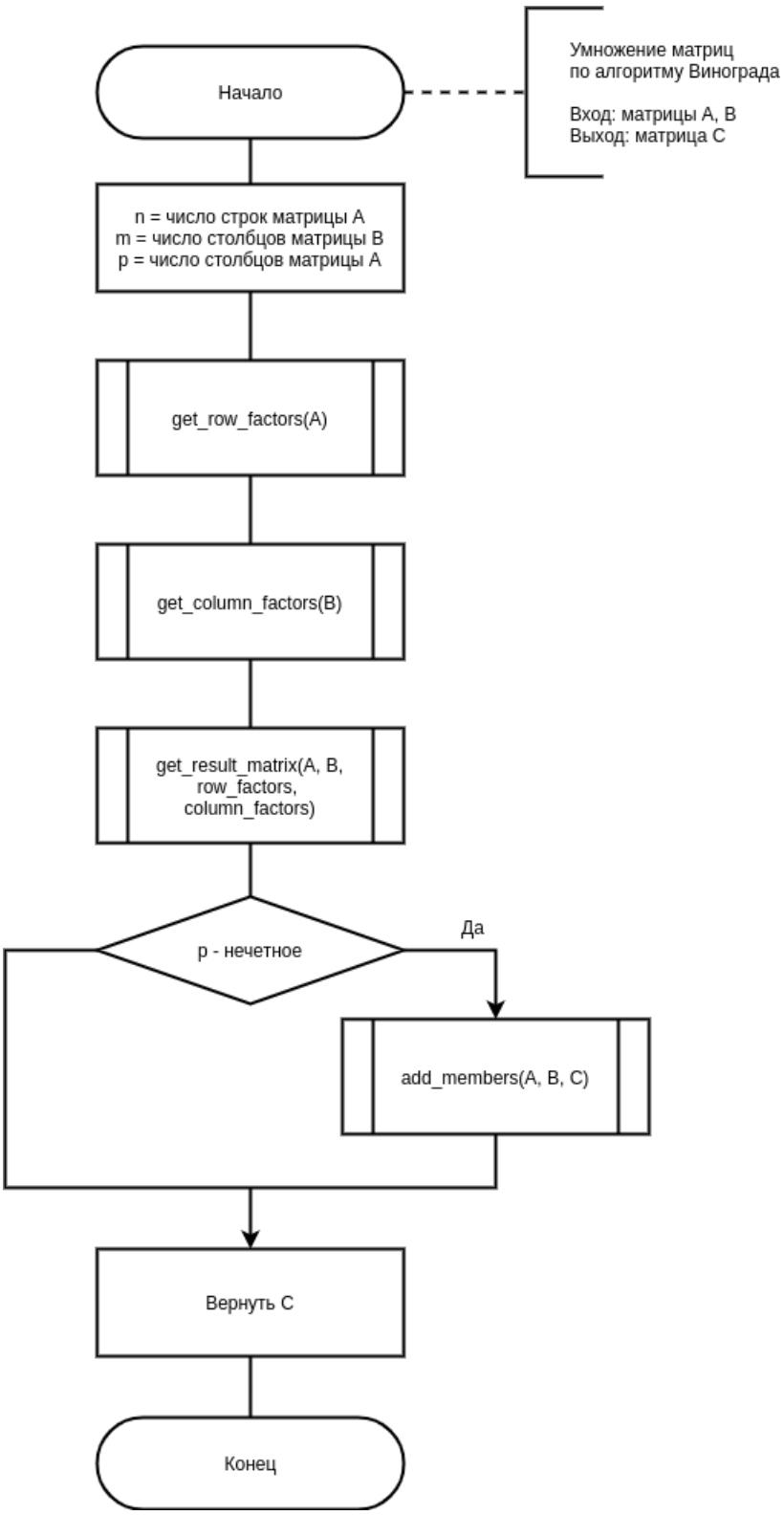

Рисунок 2.2 – Умножение матриц по алгоритму Винограда

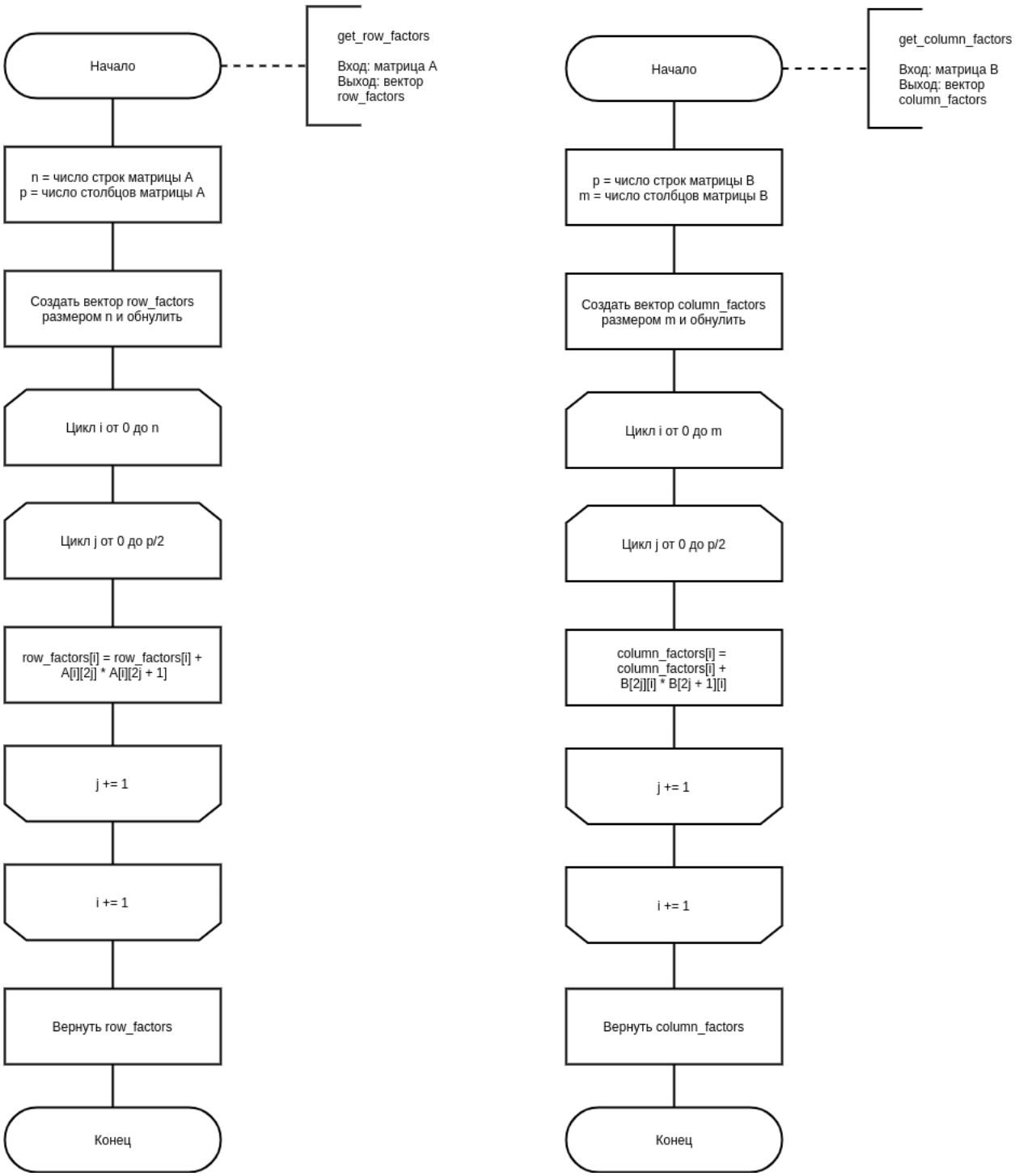

Рисунок 2.3 – Предварительные вычисления в умножении матриц по алгоритму Винограда

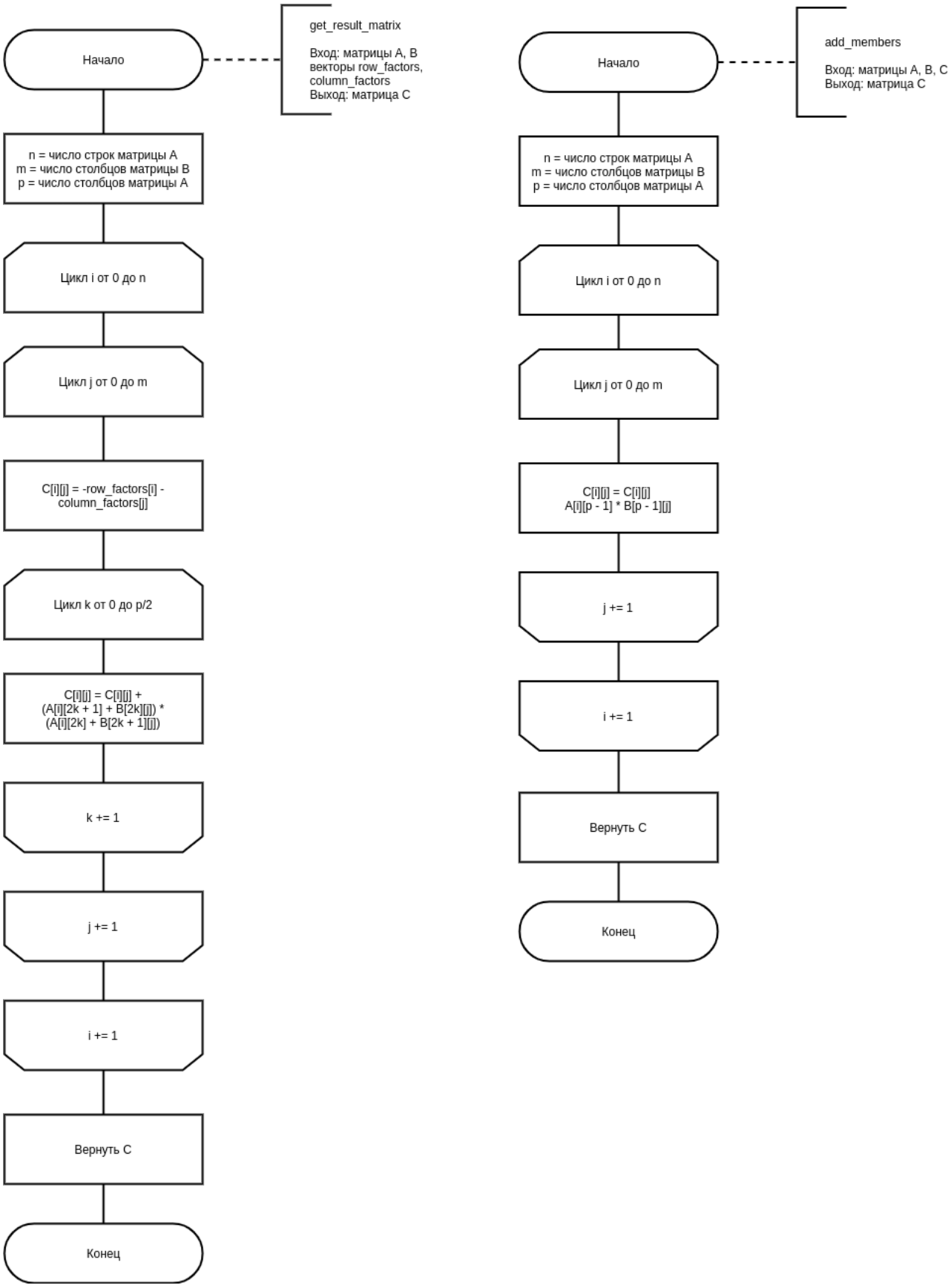

Рисунок 2.4 – Вычисление результата умножения матриц по алгоритму Винограда

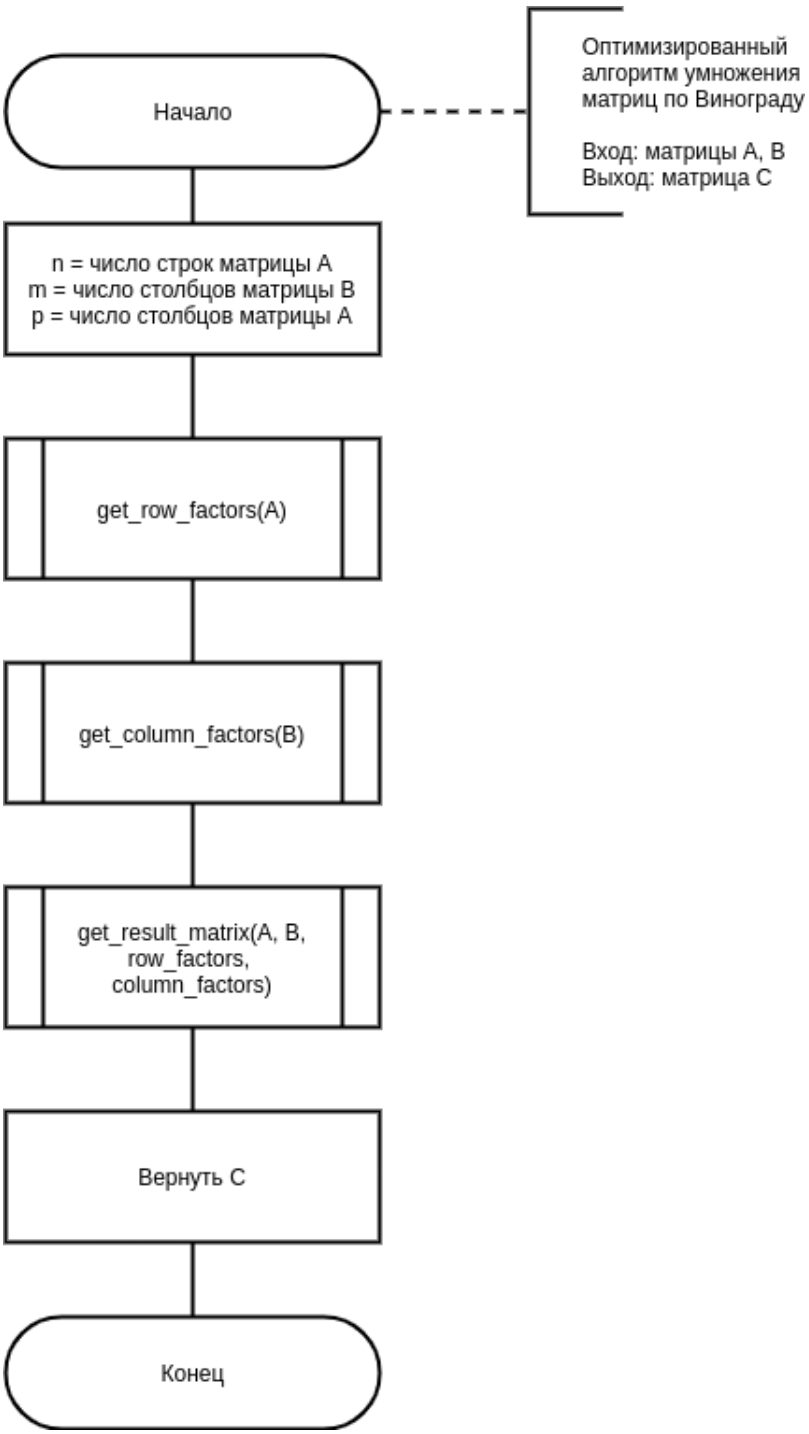

Рисунок 2.5 – Умножение матриц по оптимизированному алгоритму Винограда

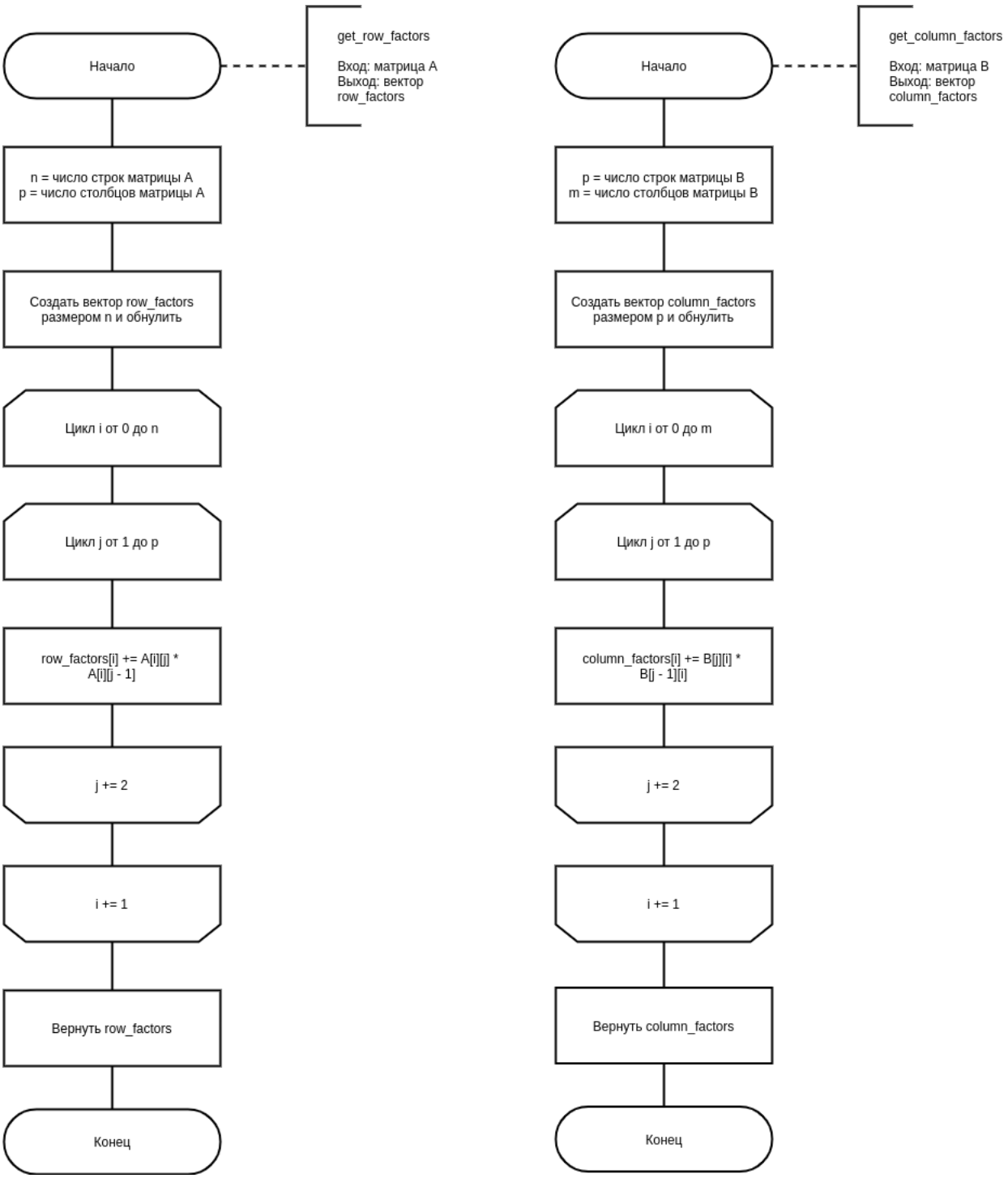

Рисунок 2.6 – Предварительные вычисления в умножении матриц по оптимизированному алгоритму Винограда

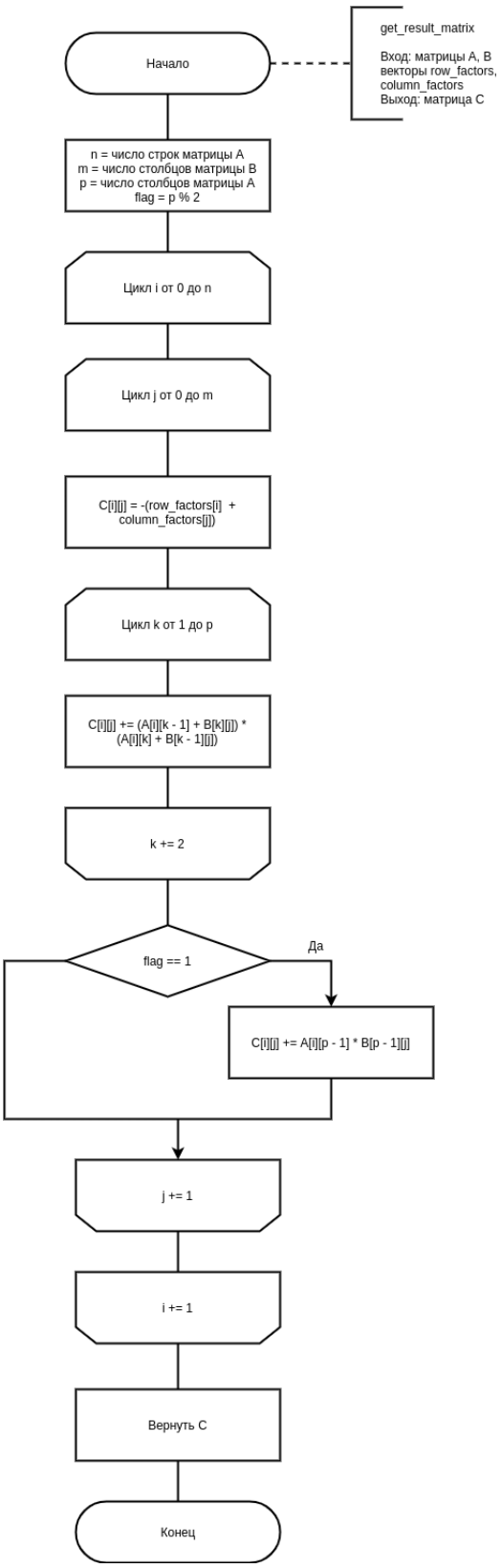

Рисунок 2.7 – Вычисление результата умножения матриц по оптимизированному алгоритму Винограда

2.2 Модель вычислений для оценки трудоемкости алгоритмов

Для определения трудоемкости алгоритмов необходимо ввести модель вычислений [3]:

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

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

4. трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости для умножения матрицы A[N][P] и матрицы B[P][M].

2.3.1 Стандартный алгоритм умножения матриц

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости цикла по $k \in [0..P-1]$, равной $1 + P(3+8) = 1 + 11P$;

- трудоекмости цикла по $j \in [0..M-1]$, равной $1 + M(3 + f_{body}) = 1 + M(3 + 1 + 11P) = 1 + 4M + 11PM$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $1 + N(3 + f_{body}) = 1 + N(3 + 1 + 4M + 11MP) = 1 + 4N + 4NM + 11NMP$.

Таким образом, трудоемкость стандартного алгоритма умножения равна $f_{standard} = 1 + 4N + 4NM + 11NMP \approx 11NMP$.

2.3.2 Алгоритм умножения матриц по Винограду

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости создания и заполнения нулями векторов row_factors и column_factors, равной $N + M$;
- трудоемкости предварительных вычислений для строк, равной $1 + N(3 + 1 + P/2(3 + 12)) = 1 + 4N + 7.5NP$;
- трудоемкости предварительных вычислений для столбцов, равной $1 + M(3 + 1 + P/2(3 + 12)) = 1 + 4M + 7.5MP$;
- трудоемкости цикла по $k \in [0..P/2 - 1]$, равной $1 + P/2(3 + 23) = 1 + 13P$;
- трудоекмости цикла по $j \in [0..M-1]$, равной $1 + M(3 + f_{body}) = 1 + M(3 + 7 + 1 + 13P) = 1 + 11M + 13PM$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $1 + N(3 + f_{body}) = 1 + N(3 + 1 + 11M + 13MP) = 1 + 4N + 11NM + 13NMP$;
- трудоемкости условия, равной 2;
- трудоемкости двойного цикла добавления в случае нечетного размера матрицы, равной $1 + N(3 + 1 + M(3 + 13)) = 1 + 4N + 16NM$.

Таким образом, трудоемкость алгоритма умножения по Винограду в худшем случае (нечетный размер матрицы) равна $f_{worst} = N + M + 2(1 +$

$4N+7.5M)+1+4N+11NM+13NMP+2+1+4N+16NM = 6+17N+16M+27NM+13NMP \approx 13NMP$.

Трудоемкость алгоритма умножения по Винограду в лучшем случае (четный размер матрицы) равна $f_{best} = N + M + 2(1 + 4N + 7.5M) + 1 + 4N+11NM+13NMP+2 = 5+13N+16M+11NM+13NMP \approx 13NMP$.

2.3.3 Оптимизированный алгоритм умножения матриц по Винограду

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости создания и заполнения нулями векторов row_factors и column_factors, равной $N + M$;
- трудоемкости предварительных вычислений для строк, равной $1 + N(3 + 1 + P/2(3 + 8)) = 1 + 4N + 5.5NP$;
- трудоемкости предварительных вычислений для столбцов, равной $1+ M(3 + 1 + P/2(3 + 8)) = 1 + 4M + 5.5MP$;
- трудоемкости вычисления признака четности размера матрицы, равной 2;
- трудоемкости цикла по $k \in [0..P/2 - 1]$, равной $1 + P/2(3 + 16) = 1 + 9.5P$;
- трудоемкости условия, равной 1;
- трудоемкости добавления в случае нечетного размера матрицы, равной 10;
- трудоекмости цикла по $j \in [0..M - 1]$, равной $1 + M(3 + f_{body})$;
- трудоемкости цикла по $i \in [0..N - 1]$, равной $1 + N(3 + f_{body})$.

Таким образом, трудоемкость оптимизированного алгоритма умножения по Винограду в худшем случае (нечетный размер матрицы) равна

$f_{worst} = N + M + 2(1 + 4N + 5.5M) + 2 + 1 + N(3 + 1 + M(3 + 7 + 1 + 9.5P + 1 + 10)) = 5 + 13N + 12M + 22MN + 9.5MNP \approx 9.5NMP$.

Трудоемкость оптимизированного алгоритма умножения по Винограду в лучшем случае (четный размер матрицы) равна $f_{best} = N + M + 2(1 + 4N + 5.5M) + 2 + 1 + N(3 + 1 + M(3 + 7 + 1 + 9.5P + 1)) = 5 + 13N + 12M + 11MN + 9.5MNP \approx 9.5NMP$.

2.4 Описание используемых типов и структур данных

Для реализации рассмотренных алгоритмов будет использован тип данных *int* - для числа строк и числа столбцов каждой матрицы.

Структура данных - матрица - представляет собой двумерный список значений типа *int*.

2.5 Структура разрабатываемого ПО

При реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Для взаимодействия с пользователем будет выделена функция process(), из которой будут вызываться методы умножения матриц и функции сравнительного анализа. Для работы с матрицами будут разработаны следущие функции:

- создание матрицы случайных чисел, входным параметром функции является размер, выходным - матрица;
- ввод матрицы, функция не имеет входных параметров, выходным параметром является введенная матрица;
- процедура вывода матрицы, входным параметром которой является матрица;
- функции умножения матриц для каждого алгоритма (классического, Винограда и оптимизированного Винограда), у которых на входе - две матрицы для умножения, а на выходе - результат умножения матриц.

Для сравнительного анализа будут реализованы:

- процедура замеров времени, выходным парметром которой является массив временных значений;
- функция графического представления замеров времени, у которой на входе - массив временных значений, на выходе - его графическое представление.

2.6 Классы эквивалентности при тестировании

Для тестирования разрабатываемой программы будут выделены следующие классы эквивалентности:

- две пустые матрицы;
- одна пустая матрица, одна нет;
- число столбцов первой матрицы не равно числу строк второй;
- умножение квадратных матриц;
- умножение матриц с одинаковым числом стоблцов первой и числом строк второй, но разным числом строк первой и числом столбцов второй.

2.7 Вывод

Были представлены схемы алгоритмов умножения матриц. Были указаны типы и структуры данных, используемые для реализации, и описана структура разрабатываемого программного обеспечения. Также были выделены классы эквивалентности для тестирования ПО.

3 Технологическая часть

В данном разделе будут указаны средства реализации, будут представлены листинги кода, а также функциональные тесты.

3.1 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования Python [4]. Выбор ЯП обусловлен простотой синтаксиса, большим числом библиотек и эффективностью визуализации данных.

Замеры времени проводились при помощи функции process_time из библиотеки time [5].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
- matrix.py - файл, содержащий функции работы с матрицами;
- time_test.py - файл, содержащий функции замеров времени работы указанных алгоритмов;
- graph_result.py - файл, содержащий функции визуализации временных характеристик описанных алгоритмов.

3.3 Листинги кода

Стандартный алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда умножения матриц приведены в листингах 3.1-3.3.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
def standard_mult(A, B):
    n = len(A)
    m = len(B[0])
    p = len(A[0])

    C = [[0] * m for i in range(n)]

    for i in range(n):
        for j in range(m):
            for k in range(p):
                C[i][j] += A[i][k] * B[k][j]

    return C
```

Листинг 3.2 – Алгоритм Винограда умножения матриц

```
def winograd_mult(A, B):
    n = len(A)
    m = len(B[0])
    p = len(A[0])

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

    row_factors = [0] * n

    for i in range(n):
        for j in range(p // 2):
            row_factors[i] = row_factors[i] + \
                             A[i][2 * j] * A[i][2 * j + 1]

    column_factors = [0] * m

    for i in range(m):
        for j in range(p // 2):
            column_factors[i] = column_factors[i] + \
                                B[2 * j][i] * B[2 * j + 1][i]

    for i in range(n):
        for j in range(m):
            C[i][j] = -row_factors[i] - column_factors[j]

            for k in range(p // 2):
                C[i][j] = C[i][j] + (A[i][2 * k + 1] + B[2 *
                   k][j]) * \
                                     (A[i][2 * k] + B[2 * k +
                                        1][j])

    if p % 2 != 0:
        for i in range(n):
            for j in range(m):
                C[i][j] = C[i][j] + A[i][p - 1] * B[p - 1][j]

    return C
```

Листинг 3.3 – Оптимизированный алгоритм Винограда умножения матриц

```
def optimized_winograd_mult(A, B):
    n = len(A)
    m = len(B[0])
    p = len(A[0])

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

    row_factors = [0] * n

    for i in range(n):
        for j in range(1, p, 2):
            row_factors[i] += A[i][j] * A[i][j - 1]

    column_factors = [0] * m

    for i in range(m):
        for j in range(1, p, 2):
            column_factors[i] += B[j][i] * B[j - 1][i]

    flag = p % 2

    for i in range(n):
        for j in range(m):
            C[i][j] = -(row_factors[i] + column_factors[j])

            for k in range(1, p, 2):
                C[i][j] += (A[i][k - 1] + B[k][j]) * \
                           (A[i][k] + B[k - 1][j])

            if flag:
                C[i][j] += A[i][p - 1] * B[p - 1][j]

    return C
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы умножения матриц. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица A	Матрица B	Ожидаемый результат
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & -1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{pmatrix}$	$\begin{pmatrix} -30 & -36 & -42 \\ -66 & -81 & -96 \\ -102 & -126 & -150 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$

3.5 Вывод

Были реализованы функции алгоритмов умножения матриц. Было проведено функциональное тестирование указанных функций.

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

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система: Ubuntu 20.04.1 Linux x86_64 [6];
- память : 8 GiB;
- процессор: AMD® Ryzen™ 3 3200u @ 2.6 GHz [7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы.

```
МЕНЮ:
  1 - стандартное умножение матриц;
  2 - умножение матриц по алгоритму Винограда;
  3 - умножение матриц по оптимизированному алгоритму Винограда;
  4 - построить графики;
  5 - замерить время;
  0 - выход.
Выбор: 3

Введите матрицу A!
Введите число строк: 3
Введите число столбцов: 2
1
2
3
4
5
6

Введенная матрица A:
 1 2
 3 4
 5 6

Введите матрицу B!
Введите число строк: 2
Введите число столбцов: 4
1
2
3
4
5
6
7
8

Введенная матрица B:
 1 2 3 4
 5 6 7 8

Полученная матрица:
 11 14 17 20
 23 30 37 44
 35 46 57 68
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Функция process_time из библиотеки time ЯП Python возвращает процессорное время в секундах - значение типа float.

Для замера времени:

- получить значение времени до начала выполнения алгоритма, затем после её окончания. Чтобы получить результат, необходимо вычесть из второго значения первое;
- первый шаг необходимо повторить iters раз (в программе iters равно 10), суммируя полученные значения, а затем усреднить результат.

Замеры проводились для квадратных матриц целых чисел, заполненных случайным образом, размером от 10 до 100 и от 11 до 101. Результаты измерения времени для четного размера матриц приведены в таблице 4.1 (в мс).

Таблица 4.1 – Результаты замеров времени

Размер	Стандартный	Винограда	Опт-ый Винограда
10	0.2875	0.6851	0.2061
20	1.6220	3.5965	1.2381
30	5.3519	4.4189	4.0736
40	12.2201	9.7817	9.0212
50	23.4212	18.8085	18.8640
60	39.3355	32.1766	30.2312
70	61.6586	52.5354	50.6840
80	91.7189	81.2931	73.1704
90	129.2349	110.4791	104.2271
100	177.5043	158.5571	134.2171

На рисунке 4.2 приведены графические результаты сравнения временных характеристик для четного размера матриц.

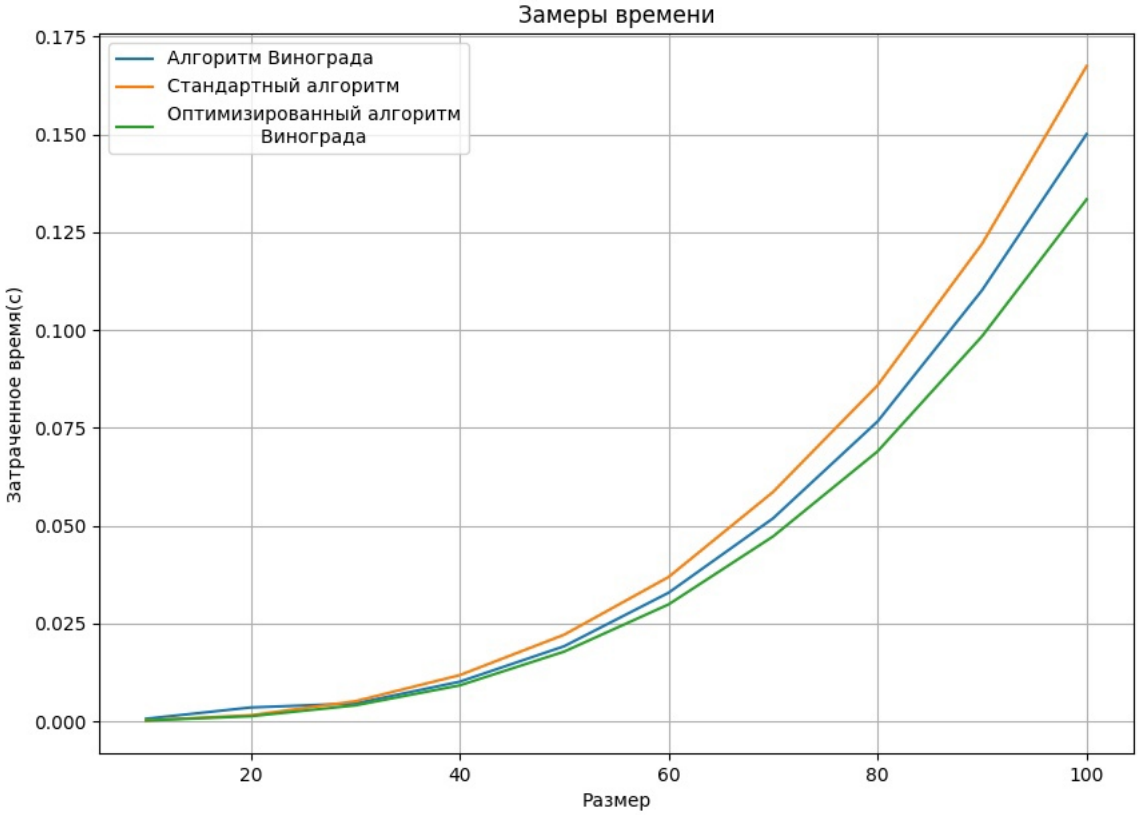

Рисунок 4.2 – Сравнение по времени алгоритмов умножения матриц четного размера

Результаты измерения времени для четного размера матриц приведены в таблице 4.2 (в мс).

Таблица 4.2 – Результаты замеров времени

Размер	Стандартный	Винограда	Опт-ый Винограда
11	0.2942	0.8884	0.2907
21	1.8007	3.2802	1.6107
31	5.5211	4.6575	4.8589
41	12.0378	10.6847	10.6740
51	23.3503	20.2865	19.9010
61	41.0413	34.2695	32.9210
71	65.2536	54.6966	50.5432
81	93.8148	82.2679	72.8342
91	131.0258	118.6902	105.8783
101	177.7476	158.8673	142.5004

На рисунке 4.3 приведены графические результаты сравнения временных характеристик для нечетного размера матриц.

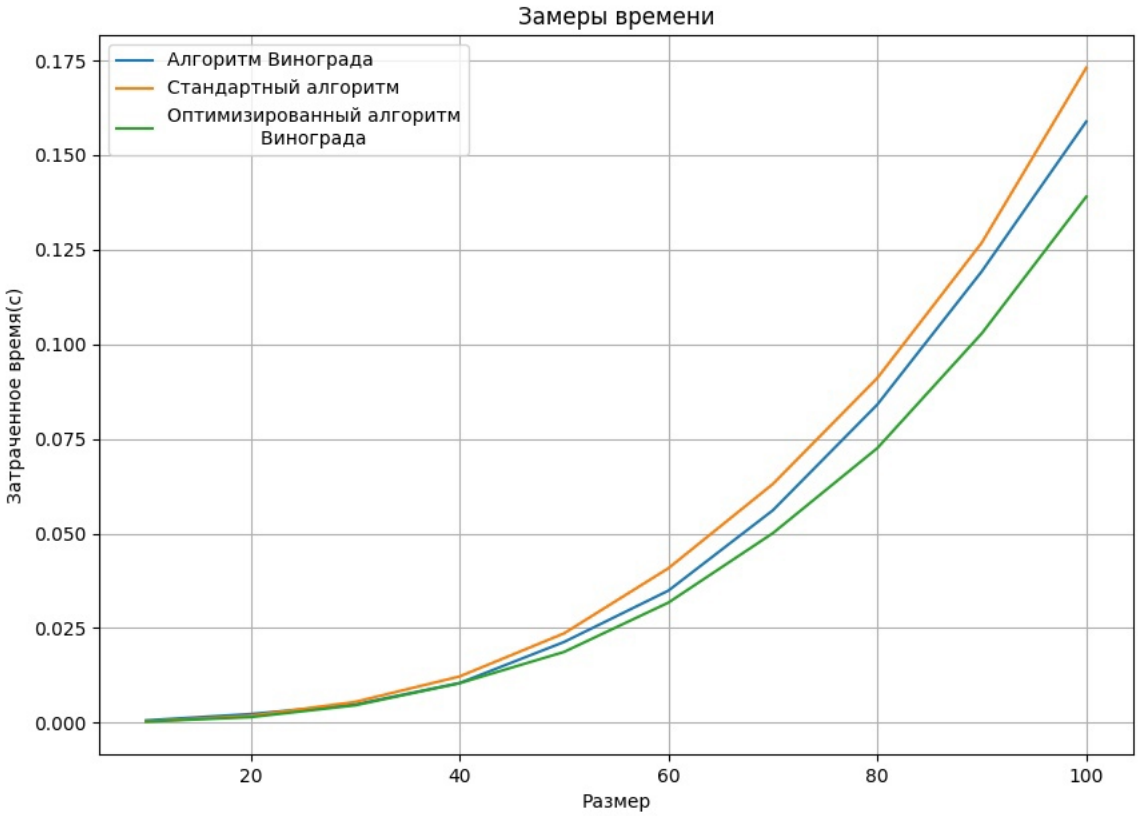

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц нечетного размера

4.4 Вывод

В результате эксперимента было получено, что при размере матриц, большем 30, оптимизированный алгоритм Винограда работает работает быстрее стандартного алгоритма в 1.3 раза. При этом стандартный алгоритм медленнее алгоритма Винограда в 1.2 раза. Тогда, для размера матриц, начиная с 30 элементов, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду.

Также в результате эксперимента было установлено, что при четном размере матриц, алгоритм Винограда работает быстрее, чем на матрицах с нечетным размером в 1.2 раза в связи с проведением дополнительных вычислений для крайних строк и столбцов. Можно сделать вывод, что алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров.

Заключение

В результате исследования было получено, что при размере матриц, большем 30, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду, так как данный алгоритм работает быстрее стандартного алгоритма в 1.3 раза. При этом стандартный алгоритм медленнее алгоритма Винограда в 1.2 раза.

Кроме того алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров, так как указанный алгоритм работает в 1.2 раза быстрее, чем на матрицах с нечетным размером. Это связано с проведением дополнительных вычислений для крайних строк и столбцов.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены следующие задачи:

- были изучены классический алгоритм, алгоритм Винограда и его оптимизированная версия умножения матриц;
- были разработаны изученные алгоритмы;
- был проведен сравнительный анализ реализованных алгоритмов;
- был подготовлен отчет о выполненной лабораторной работе.

Список литературы

[1] Н.Вирт Алгоритмы и структуры данных. 1989.

[2] Д. А. Погорелов, А. М. Таразанов Оптимизация классического алгоритма Винограда для перемножения матриц. 2019.

[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: 18.10.2021).

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

[6] Ubuntu 20.04 LTS (Focal Fossa) Beta [Электронный ресурс]. Режим доступа: `http://old-releases.ubuntu.com/releases/20.04.1/` (дата обращения: 18.10.2021).

[7] Мобильный процессор AMD Ryzen™ 3 3200U с графикой Radeon™ Vega 3 [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-3-3200u` (дата обращения: 18.10.2021).