Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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 №1
по дисциплине «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Байгарин Алан

Группа ИУ7-52Б

Преподаватели Волкова Л.Л., Строганов Д.В.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Операция умножения матриц является одной из фундаментальных и наиболее ресурсоёмких задач в линейной алгебре и вычислительной математике. Она находит применение в различных областях науки и техники, включая решение систем линейных уравнений, компьютерную графику, машинное обучение, обработку сигналов и математическое моделирование систем.

Целью данной лабораторной работы является исследование алгоритмов умножения матриц.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать три алгоритма умножения матриц;
2) создать программное обеспечение, реализующее следующие алгоритмы:
    - классический алгоритм умножения матриц;
    - алгоритм Винограда;
    - оптимизированный алгоритм Винограда;
3) оценить трудоёмкость полученных реализаций;
4) провести анализ временных затрат работы программы и выявить влияющие на них факторы;
5) провести сравнительный анализ алгоритмов.

1 Аналитическая часть

В данном разделе будут приведены понятия матриц, их произведения, классический алгоритм умножения матриц, алгоритм Винограда и его оптимизированная версия.

1.1 Матрица

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов матрицы задают размер матрицы. [1]

$$A_{m\times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{1.1}$$

a_{ij} — элемент матрицы, находящийся в *i-ой* строке и *j-ом* столбце.

Для матриц определены следующие математические операции:

1) сложение с матрицей идентичного размера;

2) умножение на скаляр;

3) произведение двух матриц.

Произведением матриц $A_{n\times q}$ и $B_{q\times m}$ называется матрица $C_{n\times m}$ (обозначается $C = A\cdot B$), каждый элемент c_{ij} которой равен сумме произведений элементов i-й строки матрицы A на соответствующие элементы j-о столбца матрицы B:

$$\forall i = \overline{1\dots n},\ \forall j = \overline{1\dots m} \quad c_{ij} = \sum_{k=1}^{p} a_{ik}\cdot b_{kj} \tag{1.2}$$

При этом операция умножения $A_{n\times q}\cdot B_{k\times m}$ не определена в случае $q \neq k$

Замечание: операция умножения $A_{n\times m}\cdot B_{m\times n}$ в общем случае не коммутативна, то есть $A\cdot B \neq B\cdot A$

1.2 Классический алгоритм умножения матриц

Классический алгоритм умножения матриц вытекает из математического определения и реализует формулу (1.2). Асимптотическая сложность такого алгоритма равна $O(n^3)$ для двух матриц порядка $n \times n$ [2].

1.3 Алгоритм Винограда умножения матриц

Каждый элемент результирующей матрицы c_{ij} представляет собой скалярное произведение i-ой строки матрицы A на j-й столбец матрицы B. Подобное умножение допускает пред-

варительную подготовку данных для уменьшения суммарного числа операций умножения [3]

Для двух векторов V и W длины k каждый, скалярное произведение определяется как: $s = V \cdot W = \sum_{i=1}^{k} v_i \cdot w_i$.

Тот же результат можно получить следующей формулой:

$$s = \sum_{i=1}^{\lfloor k/2 \rfloor} (v_{2i} + w_{2i+1}) \cdot (v_{2i+1} + w_{2i}) - \sum_{i=1}^{\lfloor k/2 \rfloor} v_i * v_{2i} - \sum_{i=1}^{\lfloor k/2 \rfloor} w_i * w_{2i} + (w_k \cdot v_k \cdot k \bmod 2) \quad (1.3)$$

Несмотря на то, что суммарное число операций в формуле (1.3) больше, 2-е и 3-е слагаемые допускают предварительную обработку, так как зависят одновременно только от рядов или колонн одной матрицы. Так можно уменьшить итоговое число умножений, и, поскольку для ЭВМ (электронная вычислительная машина) операция умножения намного более ресурсоёмкая, чем операция сложения, реализация такого алгоритма на практике должна быть быстрее стандартной.

1.4 Оптимизация алгоритма Винограда

При программной реализации алгоритма Винограда, согласно варианту требуется провести следующие оптимизации:

1) избавиться от массива MulH, вычисляя значение для каждой отдельный строки (то есть избавится от предварительного вычисления 2-ого слагаемого в формуле (1.3), вычисляя его одновременно с первым);

2) заменить умножения $x \cdot 2$ на побитовый сдвиг $x << 1$.

Вывод

В данном разделе были приведены понятия матриц и операция их умножения, стандартный алгоритм умножения матриц, алгоритм Винограда для умножения матриц и его оптимизация.

2 Конструкторская часть

В данном разделе будут разработаны алгоритмы классического умножения матриц, умножения матриц с использованием алгоритма Винограда и его оптимизированной версии, и приведены схемы алгоритмов их реализации. Также будет приведена оценка трудоёмкости данных алгоритмов.

2.1 Разработка алгоритмов

2.1.1 Разработка алгоритма стандартного умножения матриц

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

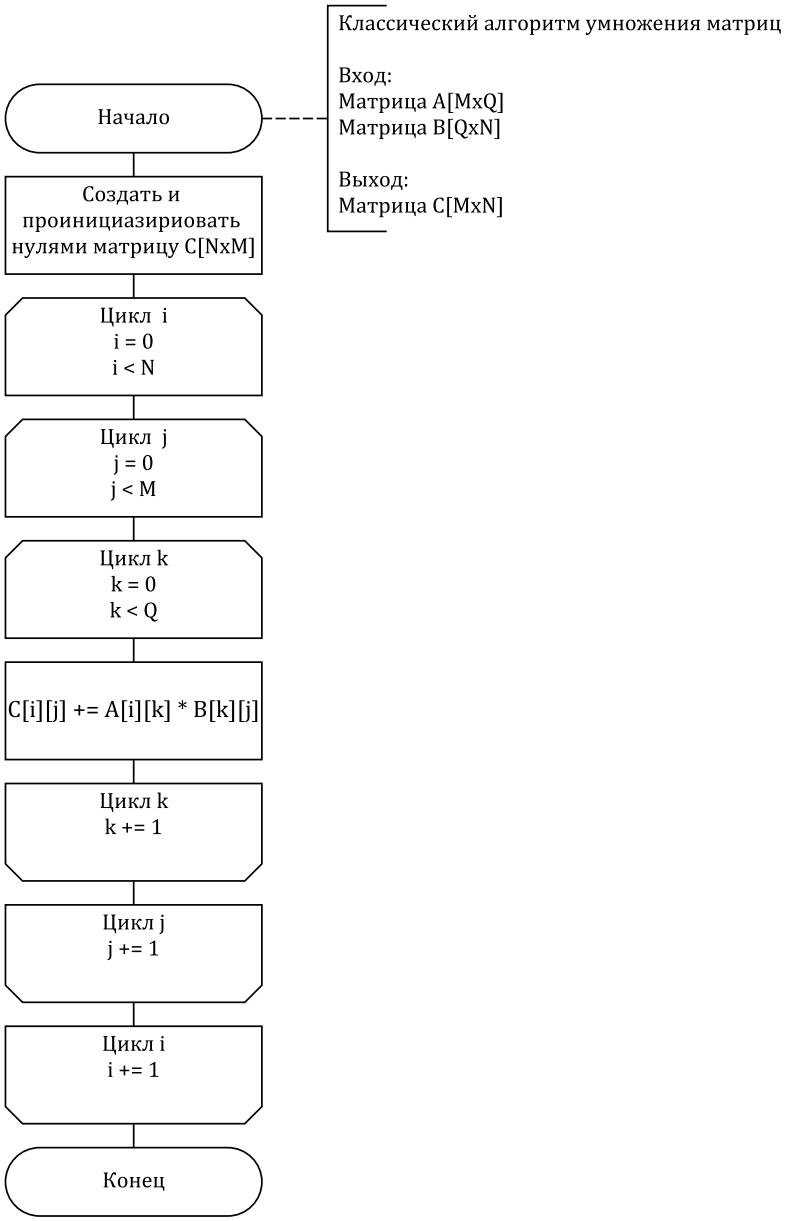

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

2.1.2 Разработка алгоритма Винограда для умножения матриц

На рисунке 2.2 приведена схема алгоритма Винограда для умножения матриц.

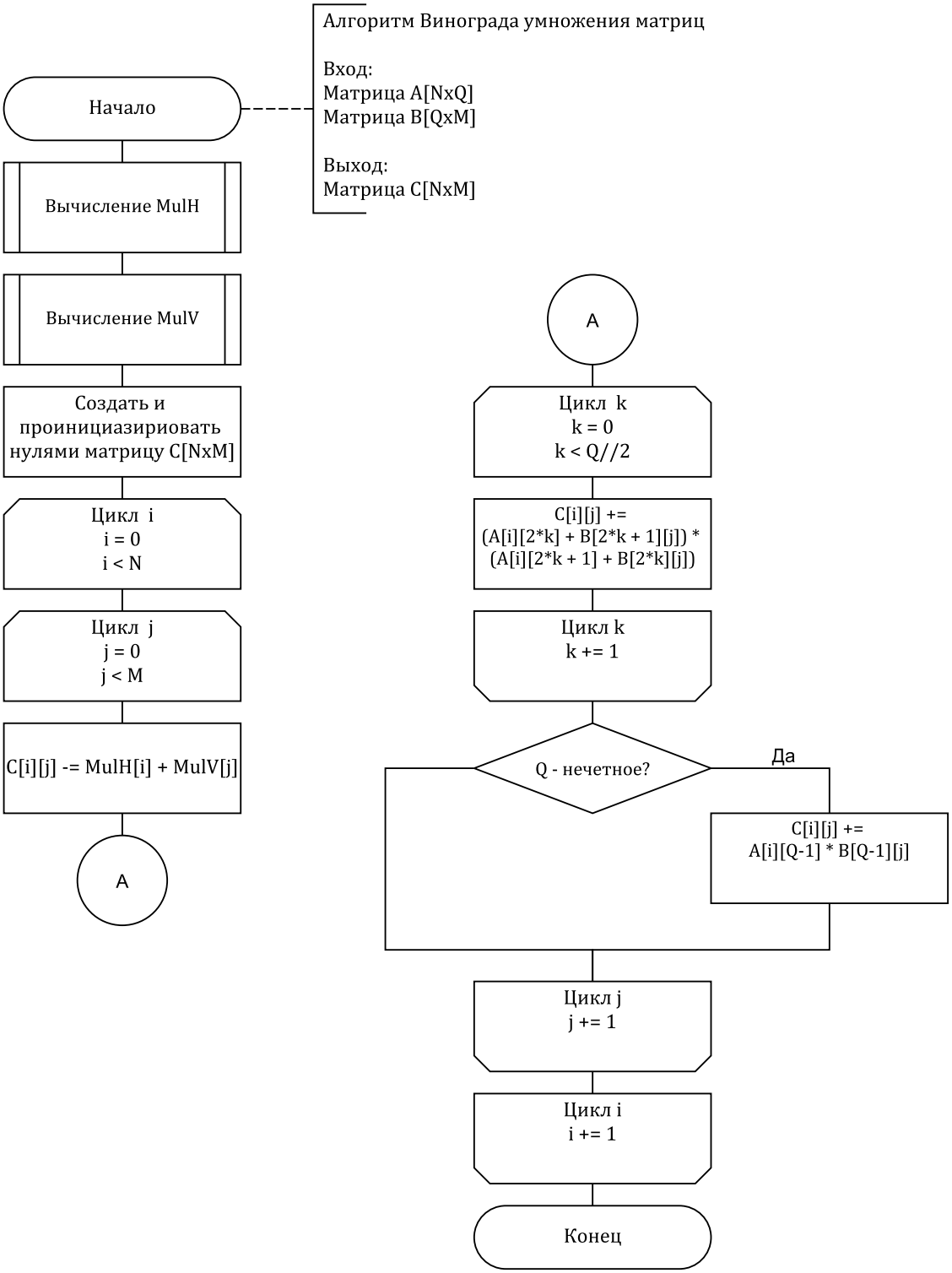

Рисунок 2.2 — Схема алгоритма Винограда для умножения матриц

На рисунке 2.3 приведена схема подпрограммы вычисления массива MulV сумм произведений пар соседних элементов строк матрицы.

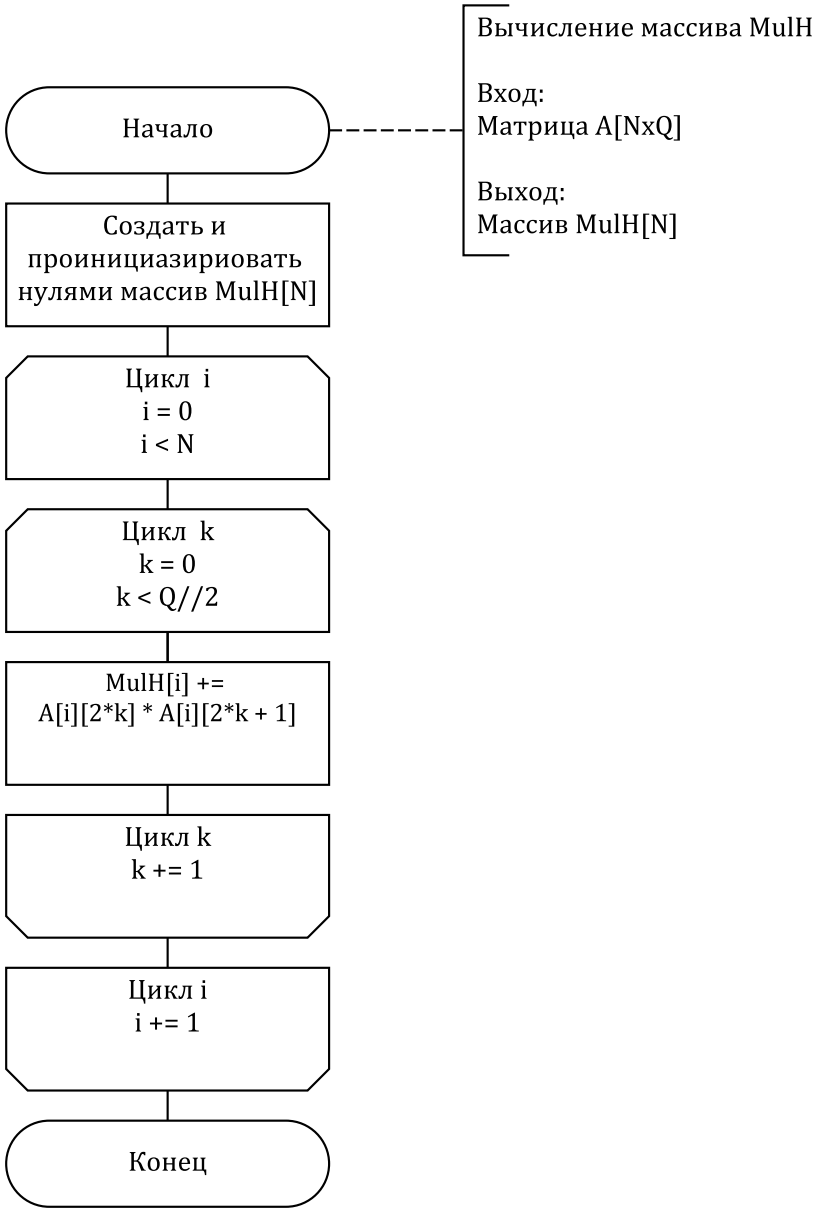

Рисунок 2.3 — Схема подпрограммы вычисления массива MulV сумм произведений пар соседних элементов строк матрицы

На рисунке 2.4 приведена схема подпрограммы вычисления массива MulH сумм произведений пар соседних элементов колонн матрицы.

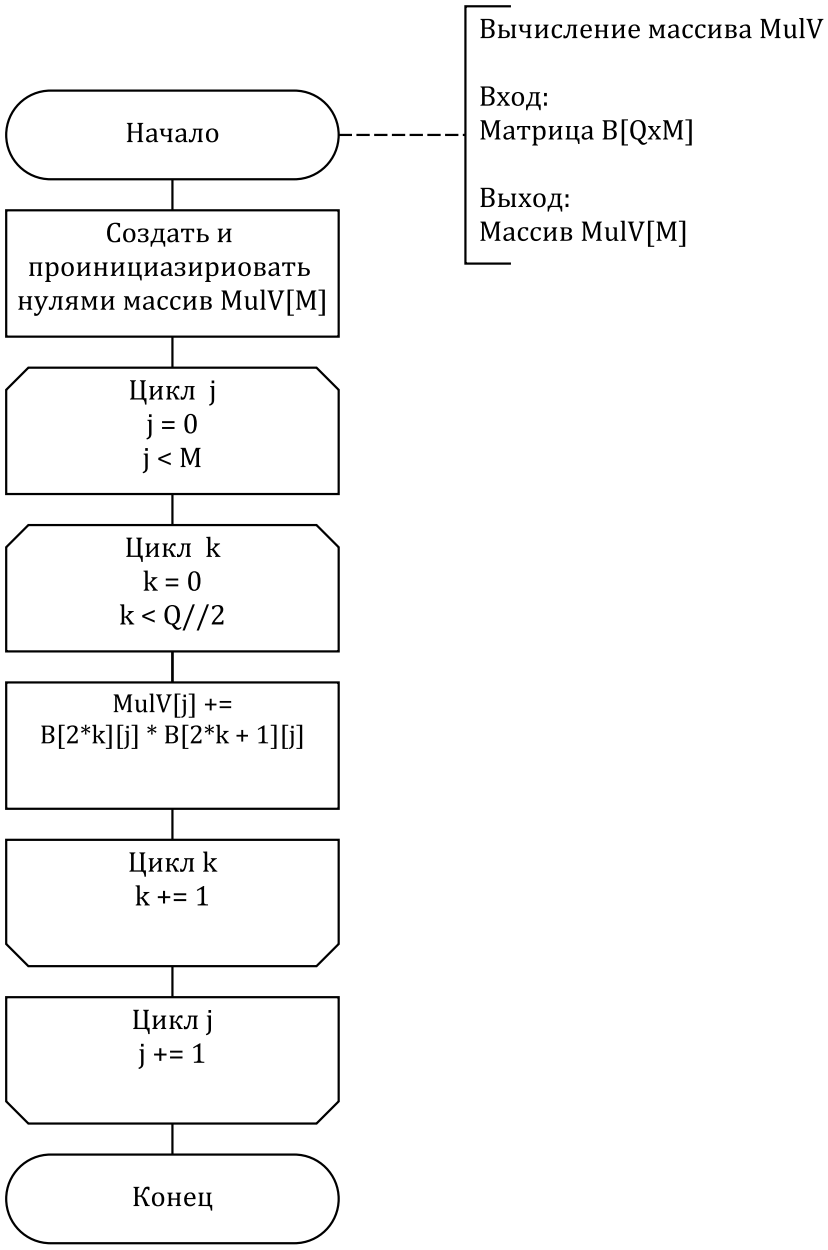

Рисунок 2.4 — Схема подпрограммы вычисления массива MulH сумм произведений пар соседних элементов колонн матрицы

2.1.3 Оптимизация алгоритма Винограда

На рисунке 2.5 приведена схема оптимизированного алгоритма Винограда для умножения матриц.

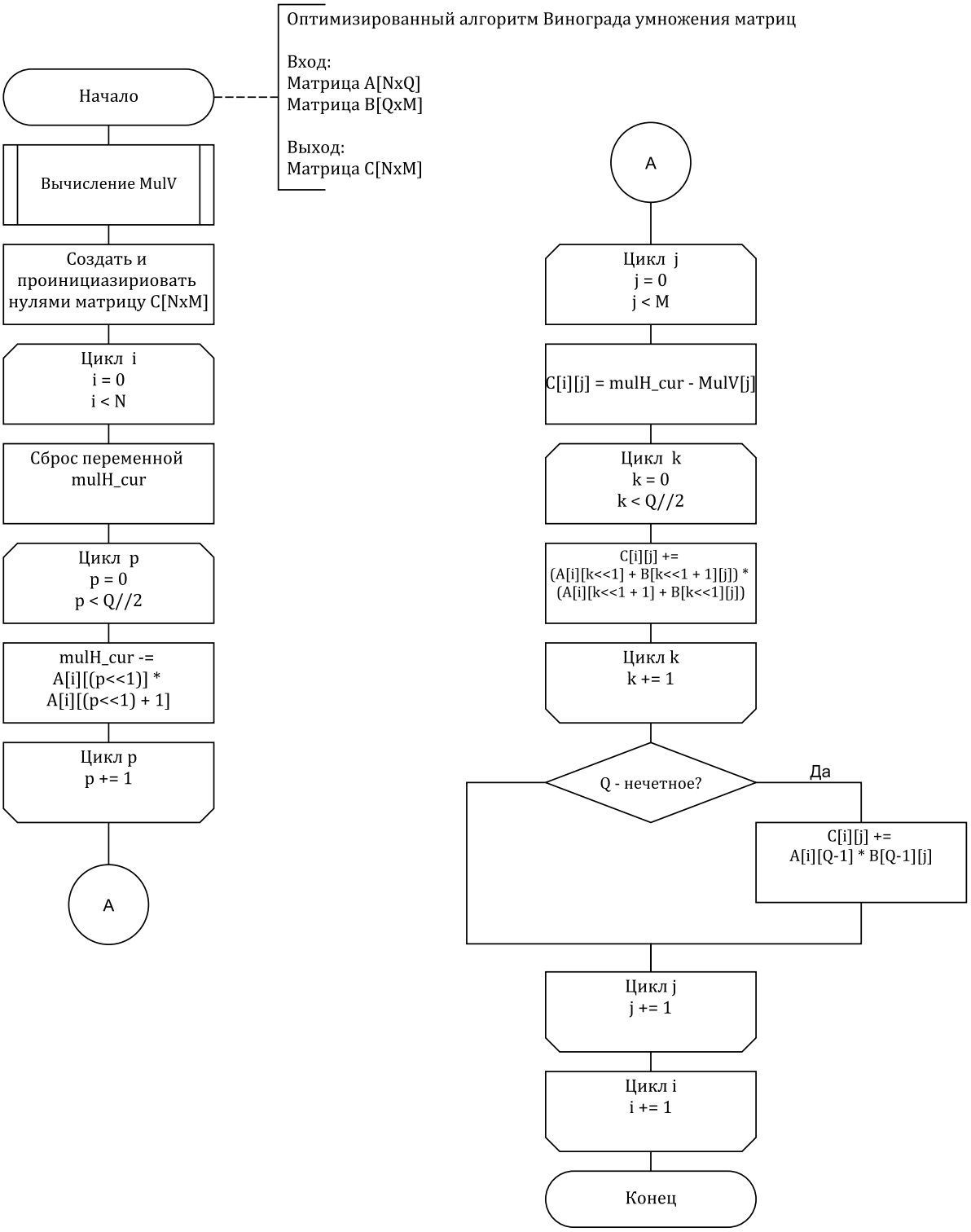

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда для умножения матриц

2.2 Оценка трудоёмкости алгоритмов

2.2.1 Модель вычислений для оценки трудоёмкости

Была введена модель вычислений для определения трудоёмкости каждого отдельного взятого алгоритма сортировки.

1) Трудоёмкость базовых операций имеет:
   — равную 1:

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

   — равную 2:

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

2) Трудоёмкость условного оператора:

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

3) Трудоёмкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

4) Трудоёмкость передачи параметра в функции и возврат из функции равны 0.

2.2.2 Трудоёмкость стандартного алгоритма умножения матриц

Суммарная трудоёмкость f_{std} складывается из трудоёмкостей инициализации матрицы $C_{N\times M}$ (f_{init}) и вложенного цикла (f_{for})

Трудоёмкость инициализации матрицы указана в формуле (2.5);

$$f_{init} = 1 \cdot (N \cdot M) = NM \tag{2.5}$$

Трудоёмкость тройного цикла указана в формуле (2.6)

$$f_{for} = 2 + N \cdot (2 + 2 + M \cdot (2 + 2 + Q \cdot (2 + 9))) = 2 + 4N + 4MN + 11MNQ \tag{2.6}$$

Тогда суммарная трудоёмкость находится как сумма (2.5) + (2.6) и указана в формуле (2.7)

$$f_{std} = f_{init} + f_{for} = 11QNM + 5MN + 4N + 2 \approx 11MNQ = O(n^3) \tag{2.7}$$

2.2.3 Трудоёмкость алгоритма Винограда умножения двух матриц

Суммарная трудоёмкость f_{Win} складывается из трудоёмкостей инициализации матрицы $C_{N\times M}$ (f_{init}), расчёта массивов MulH и MulV (f_{mulH} f_{mulV} соответственно), и трудоёмкости вложенного цикла (f_{for})

Трудоёмкость инициализации матрицы указана ранее в формуле (2.5);

Трудоёмкость расчёта MulH указана в формуле (2.8);

$$f_{mulH} = N + 2 + N \cdot (2 + (4 + \frac{Q}{2} \cdot (4 + 13))) = 2 + 7N + \frac{17NQ}{2} \tag{2.8}$$

Трудоёмкость расчёта MulV (с учётом оптимизации умножения) указана в формуле (2.9);

$$f_{mulH} = M + 2 + M \cdot (2 + (4 + \frac{Q}{2} \cdot (4 + 13))) = 2 + 7M + \frac{17MQ}{2} \tag{2.9}$$

Трудоёмкость цикла f_{for} указана в формуле (2.10);

$$f_{for} = 2 + N \cdot (2 + (2 + M \cdot (2 + (6 + 4 + \frac{Q}{2} \cdot (4 + 25) + 3 + \begin{cases} 0, & \text{Q — чётное,} \\ 11, & \text{иначе} \end{cases})))) =$$

$$2 + 4N + 15MN + \frac{29MNQ}{2} + \begin{cases} 0, & \text{Q — чётное,} \\ 11MN, & \text{иначе} \end{cases} \tag{2.10}$$

Суммарная трудоёмкость находится как сумма (2.5) + (2.9) + (2.8) + (2.10) и приведена в формуле (2.11)

$$f_{Win} = f_{init} + f_{mulH} + f_{mulV} + f_{for} =$$

$$6 + 7M + 11N + \frac{17MQ}{2} + \frac{17NQ}{2} + 16MN + \frac{29MNQ}{2} + \begin{cases} 0, & \text{Q — чётное,} \\ 11MN, & \text{иначе} \end{cases} \tag{2.11}$$

$$\approx \frac{29MNQ}{2} = O(n^3)$$

2.2.4 Трудоёмкость оптимизированного алгоритма Винограда умножения двух матриц

Суммарная трудоёмкость $f_{optimized\_Win}$ складывается из трудоёмкости инициализации матрицы $C_{N\times M}$ (f_{init}), трудоёмкости расчёта массива MulV (f_{mulV}), и трудоёмкости вложенного цикла (f_{for})

Трудоёмкость инициализации матрицы указана ранее в формуле (2.5);

Трудоёмкость расчёта MulV (с учётом оптимизации умножения) указана в формуле (2.9);

$$f_{mulV} = M + 2 + M \cdot (2 + (4 + \frac{Q}{2} \cdot (4 + 11))) = 2 + 7M + \frac{15MQ}{2} \tag{2.12}$$

Трудоёмкость цикла f_{for} указана в формуле (2.13);

$$f_{for} = 2 + N \cdot (7 + \frac{Q}{2} \cdot (4 + 10) + (2 + M \cdot (2 + 12 + \frac{Q}{2} \cdot (4 + 21) + \begin{cases} 0, & \text{Q — чётное,} \\ 11MN, & \text{иначе} \end{cases}))) =$$

$$2 + 9N + 7NQ + 14MN + \frac{25MNQ}{2} + \begin{cases} 0, & \text{Q — чётное,} \\ 11MN, & \text{иначе} \end{cases} \tag{2.13}$$

Суммарная трудоёмкость находится как сумма (2.5) + (2.12) + (2.13) и приведена в формуле (2.14)

$$f_{optimised\_Win} = f_{init} + f_{mulV} + f_{for} =$$

$$4 + 7M + 9N + 7NQ + \frac{15MQ}{2} + 15MN + \frac{25MNQ}{2} + \begin{cases} 0, & \text{Q — чётное,} \\ 11MN, & \text{иначе} \end{cases} \tag{2.14}$$

$$\approx \frac{25MNQ}{2} = O(n^3)$$

Вывод

В данном разделе были построены схемы алгоритмов умножения матриц, рассматриваемых в лабораторной работе, введена модель вычислений и проведена теоретическая оценка трудоёмкости представленных алгоритмов. Согласно приведённой оценке, трудоёмкость стандартного алгоритма Винограда на 21% больше, чем у его оптимизированной версии, и на 32% больше трудоёмкости стандартного алгоритма в введённой модели вычислений.

3 Технологическая часть

В данном разделе будут перечислены средства реализации, листинги кода и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования для выполнения данной лабораторной работы был выбран C++ [4], так как он является статически типизированным языком с гарантированным временем выполнения, что соотносится с требованиями к лабораторной работе.

Для замера процессорного времени выполнения программы используется функция *clock()* библиотеки *ctime* [5] [6]

3.2 Сведения о модулях программы

Программа состоит из 6 модулей:

1) definitions.hpp — модуль, содержащий определения типов данных;
2) err.hpp — модуль, содержащий описание классов исключений;
3) extras.cpp — модуль, содержащий функции инициализации объектов;
4) solution.cpp — модуль, содержащий реализации всех алгоритмов;
5) io.cpp — модуль, содержащий функции ввода-вывода;
6) main.cpp — точка входа программы;

3.3 Реализация алгоритмов

В листинге 3.1 содержится описание используемых типов данных, в листингах 3.2 – 3.4 приведены реализации алгоритмов умножения матриц

Листинг 3.1 — Используемые типы данных

```
#include <vector>

using namespace std;
using number = double;
using Matrix = vector<vector<number>>;
```

Листинг 3.2 — Реализация стандартного алгоритма умножения матриц

```
matrix_time multiply_standart(Matrix& A, Matrix& B)
{
    size_t n = A.size();
    size_t q = A[0].size();
    size_t m = B[0].size();

    if (q != B.size()) throw Unmultipliable();

    clock_t time;

    time = clock();

    Matrix C = createMatrix(n, m);

    for (size_t i = 0; i < n; ++i)
    {
        for (size_t j = 0; j < m; ++j)
        {
            for (size_t k = 0; k < q; ++k)
            {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    time = clock() - time;

    return {C, time};
}
```

Листинг 3.3 — Реализация алгоритма Винограда умножения матриц

```
matrix_time multiply_Vinograd(Matrix& A, Matrix& B)
{
    size_t n = A.size();
    size_t q = A[0].size();
    size_t m = B[0].size();

    if (q != B.size()) throw Unmultipliable();

    clock_t time;

    time = clock();

    Matrix C = createMatrix(n, m);
    vector<number> mul_v = createVector(m);
    vector<number> mul_h = createVector(n);

    for (size_t i = 0; i < n; ++i)
    {
        for (size_t k = 0; k < (q / 2); ++k)
        {
            mul_h[i] += A[i][2*k] * A[i][2*k + 1];
        }
    }

    for (size_t j = 0; j < m; ++j)
    {
        for (size_t k = 0; k < (q / 2); ++k)
        {
            mul_v[j] += B[2*k][j] * B[2*k + 1][j];
        }
    }

    for (size_t i = 0; i < n; ++i)
    {
        for (size_t j = 0; j < m; ++j)
        {
            C[i][j] -= mul_h[i] + mul_v[j];

            for (size_t k = 0; k < (q / 2); ++k)
            {
```

```
                C[i][j] += (A[i][2*k] + B[2*k + 1][j]) * (A[i][2*k + 1]
                    + B[2*k][j]);
            }

            if (q % 2 == 1)
                C[i][j] += A[i][q-1] * B[q-1][j];

        }
    }

    time = clock() - time;

    return {C, time};
}
```

Листинг 3.4 — Реализация оптимизированного алгоритма Винограда умножения матриц

```
matrix_time multiply_Vinograd_upd(Matrix& A, Matrix& B)
{
    size_t n = A.size();
    size_t q = A[0].size();
    size_t m = B[0].size();

    if (q != B.size()) throw Unmultipliable();

    number mul_cur_h{};

    clock_t time;

    time = clock();

    Matrix C = createMatrix(n, m);
    vector<number> mul_v = createVector(m);

    for (size_t j = 0; j < m; ++j)
    {
        for (size_t k = 0; k < (q / 2); ++k)
        {
            mul_v[j] += B[(k<<1)][j] * B[(k<<1) + 1][j];
        }
    }

    for (size_t i = 0; i < n; ++i)
    {
        mul_cur_h = 0;
        for (size_t k = 0; k < (q / 2); ++k)
        {
            mul_cur_h += A[i][(k<<1)] * A[i][(k<<1) + 1];
        }

        for (size_t j = 0; j < m; ++j)
        {
            C[i][j] -= mul_cur_h + mul_v[j];

            for (size_t k = 0; k < (q / 2); ++k)
            {
                C[i][j] += (A[i][(k<<1)] + B[(k<<1) + 1][j]) * (A[i][(k
```

```
                    <<1) + 1] + B[(k<<1)][j]);
            }

            if (q % 2 == 1)
                C[i][j] += A[i][q-1] * B[q-1][j];

        }
    }

    time = clock() - time;

    return {C, time};
}
```

3.4 Функциональные тесты

Тестирование проведено по методологии чёрного ящика.

В ходе выполнения программы тестовые данные подаются на вход всем трём реализациям; совпадение результатов их работы проверяется внутри программы. Функциональные тесты приведены в таблице 3.1

Таблица 3.1 — Функциональные тесты для реализаций алгоритмов умножения матриц

Входные данные		**Результат для классического алгоритма**	
Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 1 & 5 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 10 \\ 100 \end{pmatrix}$	$\begin{pmatrix} 321 \\ 654 \end{pmatrix}$	$\begin{pmatrix} 321 \\ 654 \end{pmatrix}$
$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 35 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$

Все тесты пройдены успешно.

Вывод

В данном разделе были перечислены использованные технические средства, приведены и протестированы реализации трёх алгоритмов умножения матриц. В результате тестирования корректность реализаций была подтверждена.

4 Исследовательская часть

4.1 Характеристики ЭВМ

В листинге 4.1 приведены характеристики ЭВМ и системы, использованных для проведения замеров времени выполнения реализаций алгоритмов.

Листинг 4.1 — Частичный вывод команды systeminfo.exe — характеристика ЭВМ

```
Имя ОС:                                    Майкрософт Windows 11 Домашняя
    для одного языка

Версия ОС:                                 10.0.26100 Н/Д построение
   26100

Конфигурация ОС:                           Изолированная рабочая станция

Тип сборки ОС:                             Multiprocessor Free

Модель системы:                            ASUS Zenbook 14
   UX3405MA_UX3405MA

Тип системы:                               x64-based PC

Процессоры:                                Число процессоров - 1.
[01]: Intel64 Family 6 Model 170 Stepping 4 GenuineIntel ~1400 МГц

Версия BIOS:                               American Megatrends
   International, LLC. UX3405MA.308, 23.07.2024

Полный объем физической памяти:            15 741 МБ

Виртуальная память: максимальный размер: 22 653 МБ
```

4.2 Время выполнения алгоритмов

Время работы алгоритмов измерялось с использованием функции $clock$ библиотеки $ctime$.

Время умножения для каждого размера матриц считалось как среднее арифметическое из 100 повторений. На вход подавались квадратные матриц $A_{N\times N}$, заполненные числами 0 до N^2.

В таблицах 4.1 – 4.2 приведены результатов времени умножения на различных размерах входных матриц.

Таблица 4.1 — Замер времени для чётных матриц размером от 2 до 1000

Линейный размер	**Время, мс**		
	Классический	**Виноград**	**Виноград (опт)**
2	1.28	1.56	1.34
10	15.3	15.12	14.36
50	665.88	669.58	640.26
100	5273.9	5006.5	4992.96
200	41825.8	39184.7	39130.6
300	145676	138221	138054
400	355970	338145	337549
500	688205	653745	656862
750	2397640	2224810	2230590
1000	5876470	5460200	5455430

Таблица 4.2 — Замер времени для нечётных матриц размером от 1 до 1001

Линейный размер	**Время, мс**		
	Классический	**Виноград**	**Виноград (опт)**
1	0.86	1.14	1
11	13.92	14.38	13.42
51	714.8	708.2	687.58
101	5465.68	5209.92	5179.46
201	43404	40891.2	41091
301	152297	144547	144932
401	355688	338194	338850
501	698691	664296	664542
751	2426420	2249880	2263940
1001	5891340	5432400	5479120

На рисунках 4.1 – 4.5 приведены графики времени умножения от размеров входных матриц.

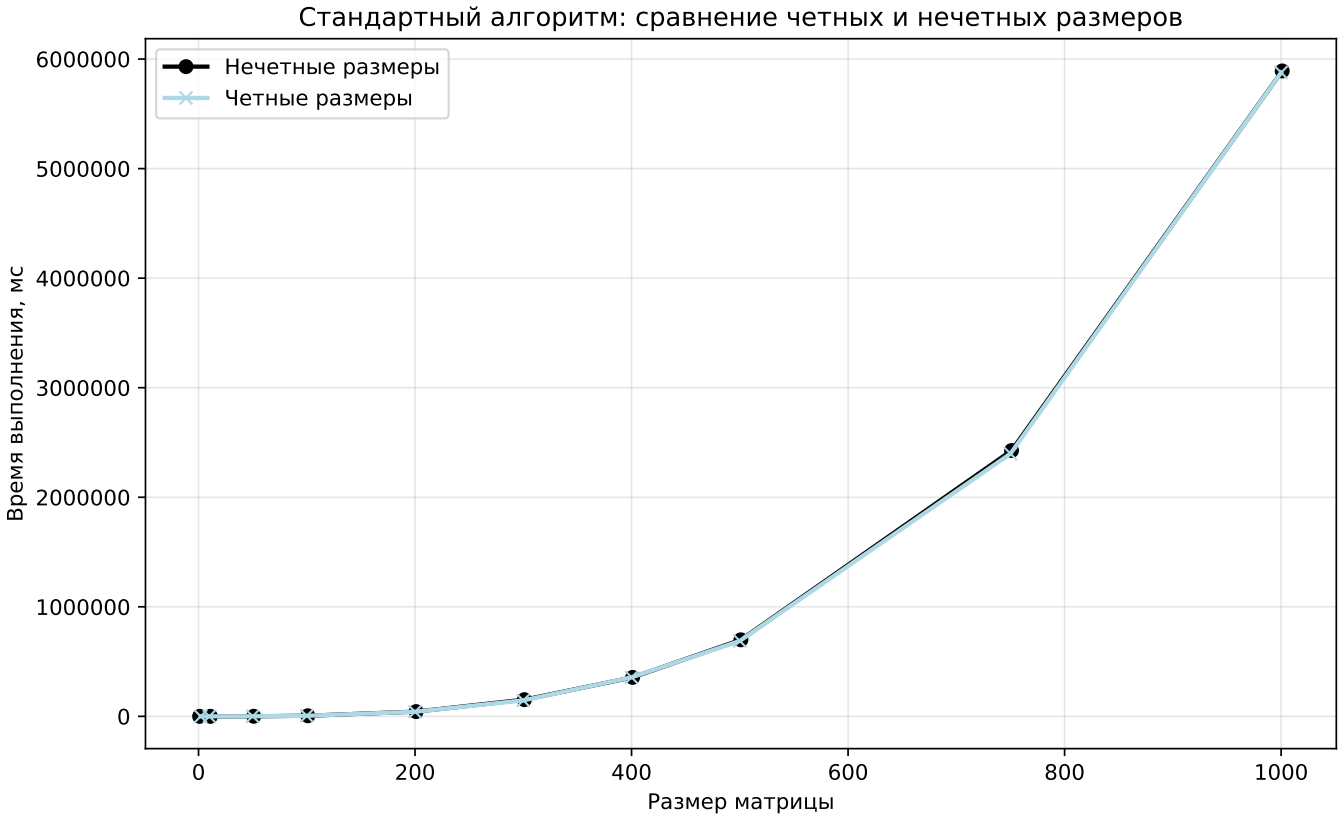

Рисунок 4.1 — Сравнение стандартного алгоритма на чётных и нечётных размерах

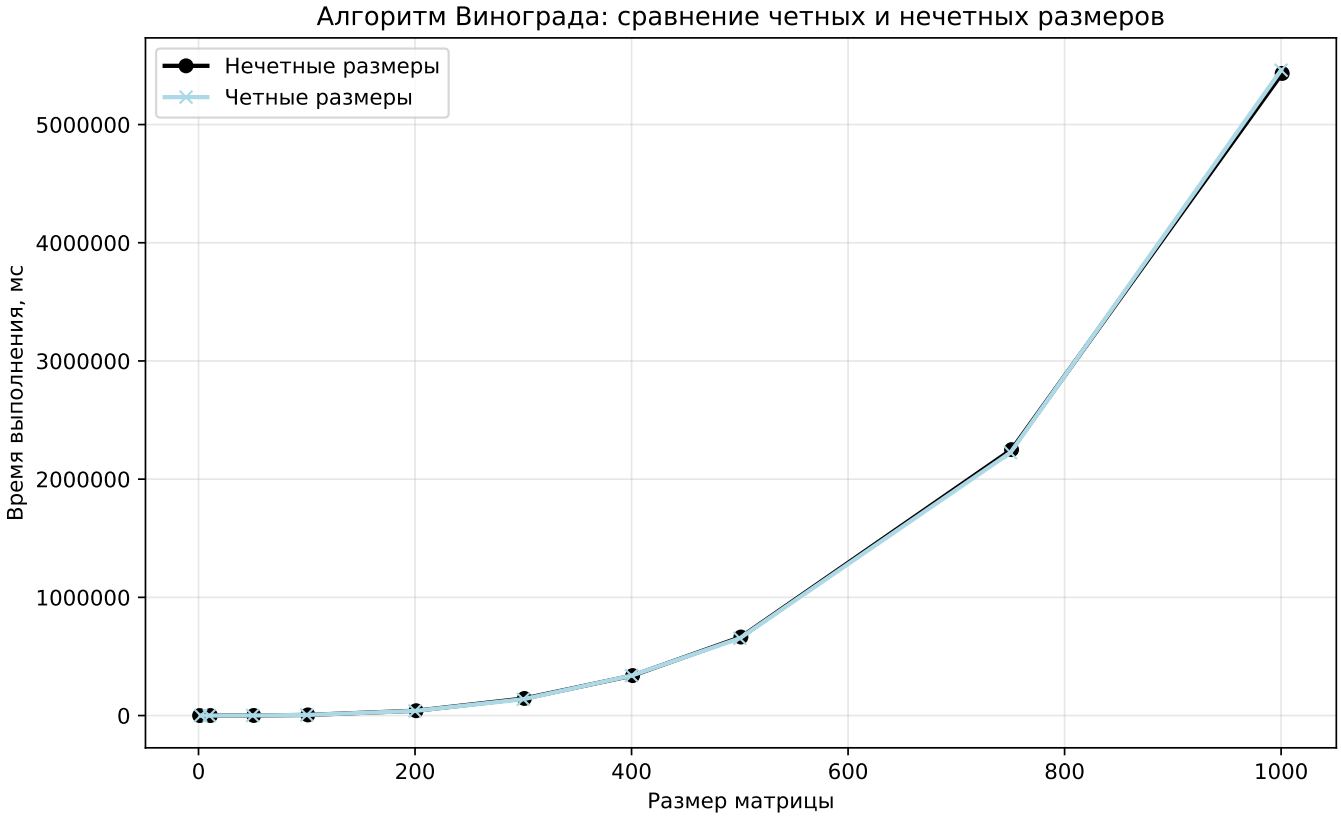

Рисунок 4.2 — Сравнение алгоритма Винограда на чётных и нечётных размерах

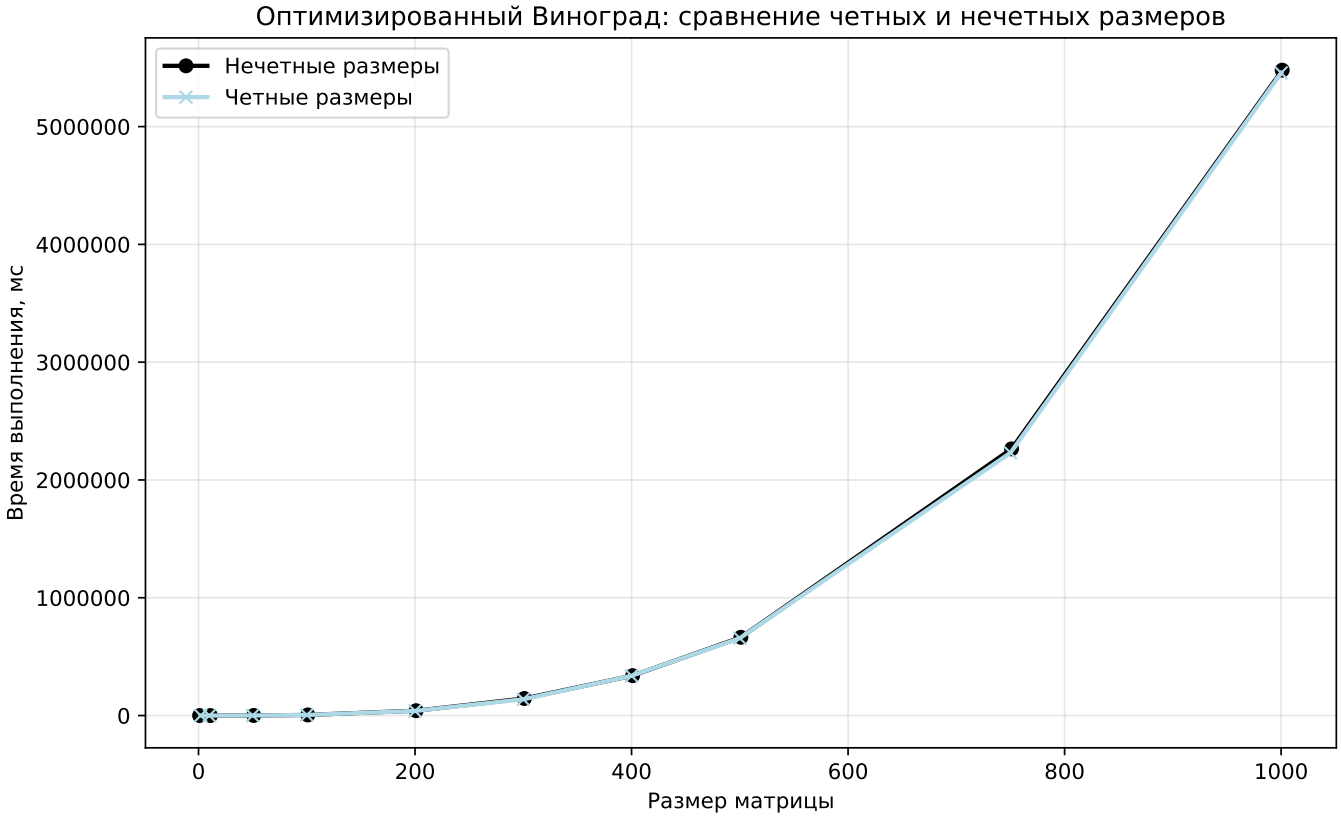

Рисунок 4.3 — Сравнение оптимизированного алгоритма Винограда на чётных и нечётных размерах

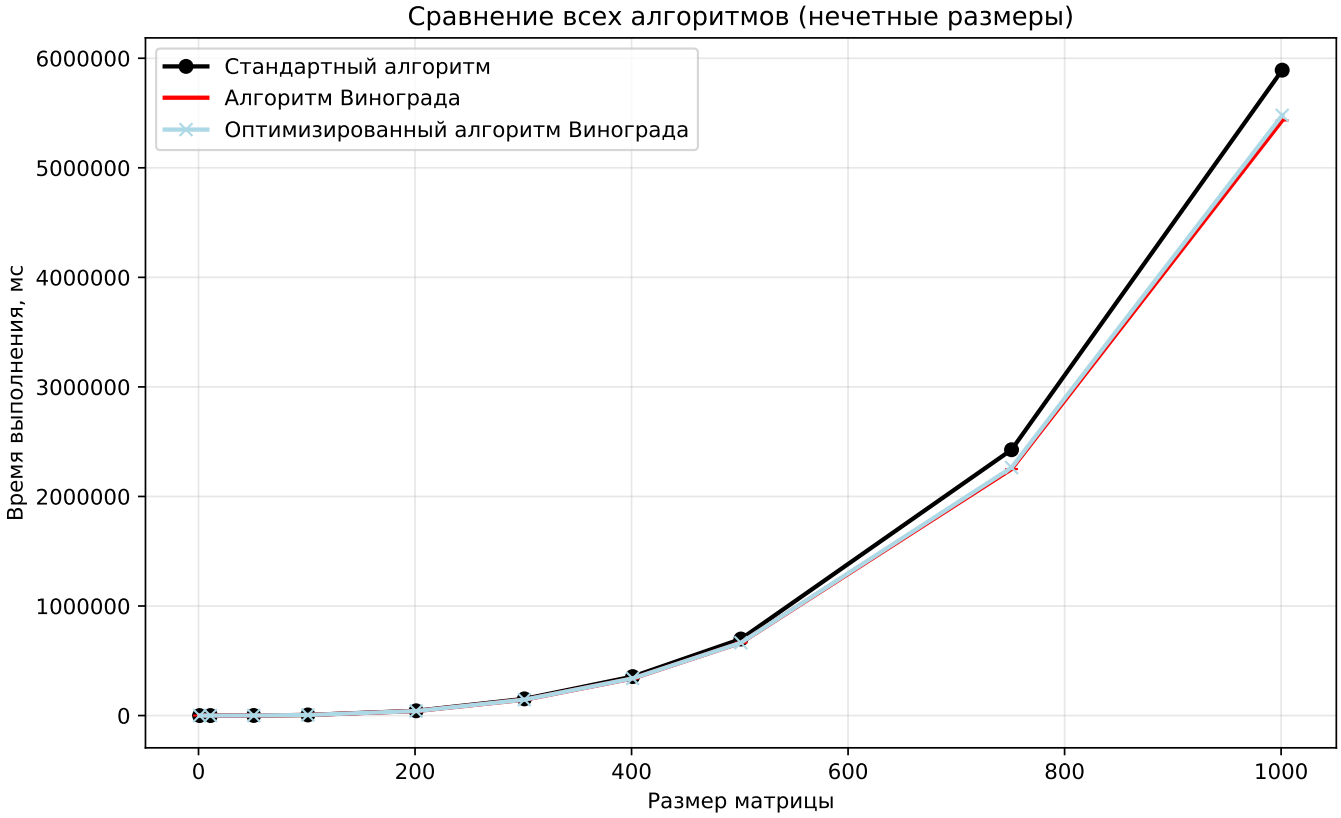

Рисунок 4.4 — Сравнение всех алгоритмов на чётных размерах

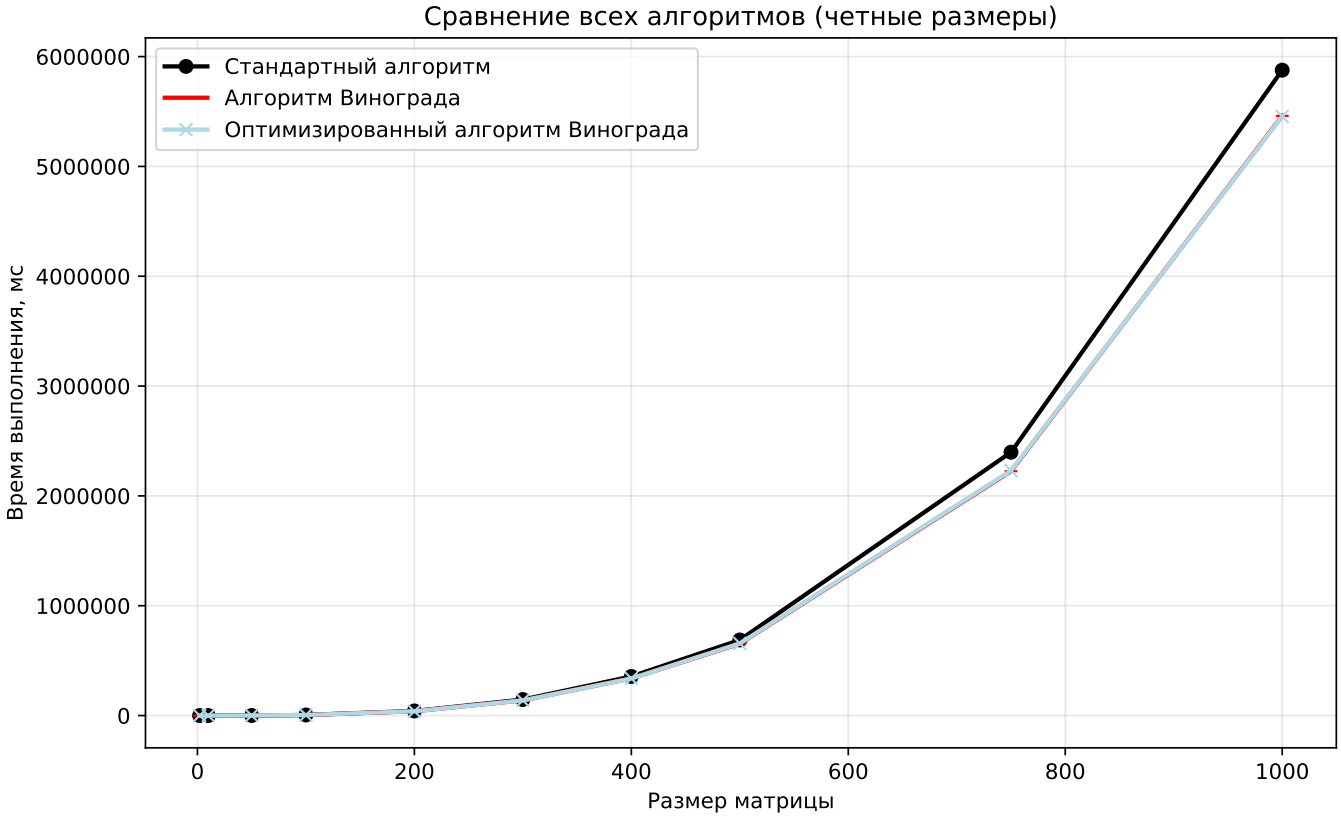

Рисунок 4.5 — Сравнение всех алгоритмов на нечётных размерах

Приведённые данные показывают, что полученная реализация алгоритма винограда эффективнее реализации стандартного по времени выполнения (на 7% при максимальном размере входных матриц). Разница между временем выполнения реализаций алгоритма Винограда и его оптимизированной версии оказалась минимальной.

Несущественной оказалась и разность времени работы описанных реализаций на близких по значению чётных и нечётных линейных размерах входных матриц (менее 1% при линейном размере матриц более 200)

Вывод

В ходе анализа замеренного времени выполнения, было получено, что применять приведённую реализацию алгоритма Винограда имеет смысл при линейном размере входных матриц более 50. Также было установлено, что дополнительные вычисления в реализации алгоритма Винограда при нечётном линейном размере входных матриц не вносят существенного вклада в итоговое время работы.

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы было установлено, что реализация алгоритма Винограда оказывается эффективнее по времени, чем реализация стандартного алгоритма умножения матриц, при больших размерах линейной матрицы (более 50). Выполненная реализация оптимизированного алгоритма Винограда оказалась практически идентичной своей обычной версии.

Все оставленные задачи, а именно:

— описание алгоритмов умножение матриц;

— оценка трудоёмкости этих алгоритмов;

— реализация стандартного алгоритма, алгоритма Винограда и его; оптимизированной версии;

— анализ времени выполнения полученных реализаций;

— сравнительный анализ алгоритмов умножения матриц;

Были выполнены.

Цель лабораторной работы, заключающаяся в исследовании алгоритмов умножения матриц, была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Матрица, её история и применение [Электронный ресурс]. Режим доступа: https://urok.1sept.ru/articles/637896 (дата обращения: 18.09.2025).

2. Пан В. Я. Быстрое умножение матриц и смежные вопросы алгебры, Матем. сб, том 208. 2017. — С. 90–91.

3. Головашкин Д. Л. Векторные алгоритмы вычислительной линейной алгебры: учеб. пособие. — Самара: Изд-во Самарского университета, 2019. — С. 28–35.

4. Справочник по языку C++ [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170 (дата обращения: 18.09.2025).

5. Time library [Электронный ресурс]. Режим доступа: https://eel.is/c++draft/ctime.syn (дата обращения: 18.09.2025).

6. Information technology — Programming languages — C [Электронный ресурс]. Режим доступа: https://www.open-std.org/JTC1/SC22/WG14/www/docs/n3220.pdf (дата обращения: 18.09.2025).