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»

на тему: «Алгоритмы умножения матриц»

Студент группы ИУ7-51Б _____ (Подпись, дата) Шубенина Д. В. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Строганов Ю. В. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Умножение матриц является основным инструментом линейной алгебры и имеет многочисленные применения в математике, физике, программировании [1].

Целью данной лабораторной работы является изучение, реализация и исследование алгоритмов умножения матриц.

Необходимо выполнить следующие задачи:

1) изучить следующие алгоритмы умножения матриц:
   - классический алгоритм;
   - алгоритм Винограда;
   - оптимизированный алгоритм Винограда;
   - влгоритм Штрассена;
2) реализовать данные алгоритмы;
3) выполненить сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени, памяти);
4) описать и обосновать полученные результаты в отчете.

1 Аналитическая часть

В данном разделе будут рассмотрены классический алгоритм умножения матриц, алгоритм Винограда (неоптимизированный и оптимизированный), а также алгоритм Штрассена.

1.1 Матрица

Матрицей типа (или размера) $m \times n$ называют прямоугольную числовую таблицу, состоящую из $m \cdot n$ чисел, которые расположены в m строках и n столбцах [2].

Обозначаются:

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

или сокращенно $(a_{ij}), i = \overline{1, m}, j = \overline{1, n}$ [2].

Над матрицами возможны следующие операции:

— сложение матриц одинакового размера;

— умножение матриц, количество столбцов первой матрицы равно количеству строк второй матрицы [2].

1.2 Классический алгоритм

Пусть даны матрица $A = (a_{ij})$ размером $m \times n$ и матрица $B = (b_{ij})$ размером $n \times p$. Произведением матриц A и B называют матрицу $C = (c_{ij})$ размером $m \times p$ с элементами

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \tag{1.2}$$

которую обозначают $C = AB$ [2].

Классический алгоритм реализует формулу (1.2).

1.3 Алгоритм Винограда

Алгоритм Винограда является одним из самых эффективных алгоритмов умножения матриц, имея асимптотическую сложность $O(n^{2.3755})$ [1].

Рассмотрим два вектора $A = (a_1, \ldots, a_n)$ и $B = (b_1, \ldots, b_n)$.

Их скалярное произведение равно:

$$A \cdot B = \sum_{i=1}^{n} a_i b_i \tag{1.3}$$

$$A \cdot B = \sum_{i=1}^{\frac{n}{2}} (a_{2i} + b_{2i+1})(a_{2i+1} + b_{2i}) + \sum_{i=1}^{\frac{n}{2}} a_i a_{i+1} + \sum_{i=1}^{\frac{n}{2}} b_i b_{i+1} \tag{1.4}$$

В выражении (1.4) выполняется большее количество вычислений, чем в выражении (1.3), однако второе и третье слагаемые из (1.4) Виноград предложил вычислять предварительно [1]. Таким образом удается уменьшить количество операций умножения, являющихся более трудоемкими, чем операции сложения.

Если обрабатываемые матрицы имеют нечетный размер, то неоюходимо дополнительно рассчитать произведения крайних строк и столбцов.

1.4 Оптимизированный алгоритм Винограда

Для программной реализации алгоритма Винограда существует несколько оптимизаций:

— значение $d = \frac{n}{2}$, используемое в качестве ограничения цикла расчета второго и третьего слагаемых из соотношения (1.4) сохранить в переменную;

— заменить операцию умножения на 2 на операциб побитового сдвига влево на 1;

— при наличии операторов `+=, -=` в выбранном языке программирования использовать их при необходимости.

1.5 Алгоритм Штрассена

Пусть A и B — матрицы размером $n \times n$, где n — степень числа 2. Поделим эти матрицы на четыре части, пополам по вертикали и горизонтали, например

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{1.5}$$

где A_{ij} — подматрицы матрицы A, имеющие размер $\frac{n}{2} \times \frac{n}{2}$.

Пусть матрица C — результирующая матрица, элементы которой в случае выбранного разбиения матриц A и B будут равны

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned} \tag{1.6}$$

Разбиение матриц A_{ij}, B_{ij} выполняется рекурсивно до того момента, пока перемножение матриц не будет сведено к перемножению чисел [3].

При таком подходе для матрицы размером 2×2 количество операций умножения равно 8. Количество умножений можно снизить до 7, используя алгоритм Штрассена [4].

Для выбранного разбиения необходимо ввести новые матрицы:

$$
\begin{aligned}
M_1 &= (A_{12} - A_{22})(B_{21} + B_{22}),\\
M_2 &= (A_{11} + A_{22})(B_{11} + B_{22}),\\
M_3 &= (A_{11} - A_{21})(B_{11} + B_{12}),\\
M_4 &= (A_{11} + A_{12})B_{22},\\
M_5 &= A_{11}(B12 - B_{22}),\\
M_6 &= A_{22}(B21 - B_{11}),\\
M_7 &= (A_{21} + A_{22})B_{11}.
\end{aligned}
\tag{1.7}
$$

Тогда C_{ij} выражаются через эти матрицы [3]:

$$
\begin{aligned}
C_{11} &= M_1 + M_2 - M_4 + M_6,\\
C_{12} &= M_4 + M_5,\\
C_{21} &= M_6 + M_7,\\
C_{22} &= M_2 - M_3 + M_5 - M_7.
\end{aligned}
\tag{1.8}
$$

Алгоритм Штрассена рекурсивно производит разбиение исходных матриц в соответствии с (1.5), вычисление результирующей матрицы согласно (1.7), (1.8).

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц: классический, алгоритм Винограда, алгоритм Штрассена. Для алгоритма Винограда отдельно были рассмотрены возможные оптимизации, применимые при реализации.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов умножения матриц, описание используемых типов данных и структуры программного обеспечения.

2.1 Требования к ПО

К программе предъявлен ряд требований:

— на вход программе подаются две матрицы, каждая записана в отдельном текстовом файле;

— результатом умножения является матрица, выводимая на экран;

— программа должна позволять производить измерения процессорного времени, затрачиваемого на выполнение реализуемых алгоритмов.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема классического алгоритма умножения матриц.

На рисунках 2.2 – 2.3 приведена схема умножения матриц алгоритмом Винограда.

На рисунках 2.4 – 2.5 показана схема умножения матриц оптимизированным алгоритмом Винограда.

На рисунке 2.6 представлена схема алгоритма Штрассена.

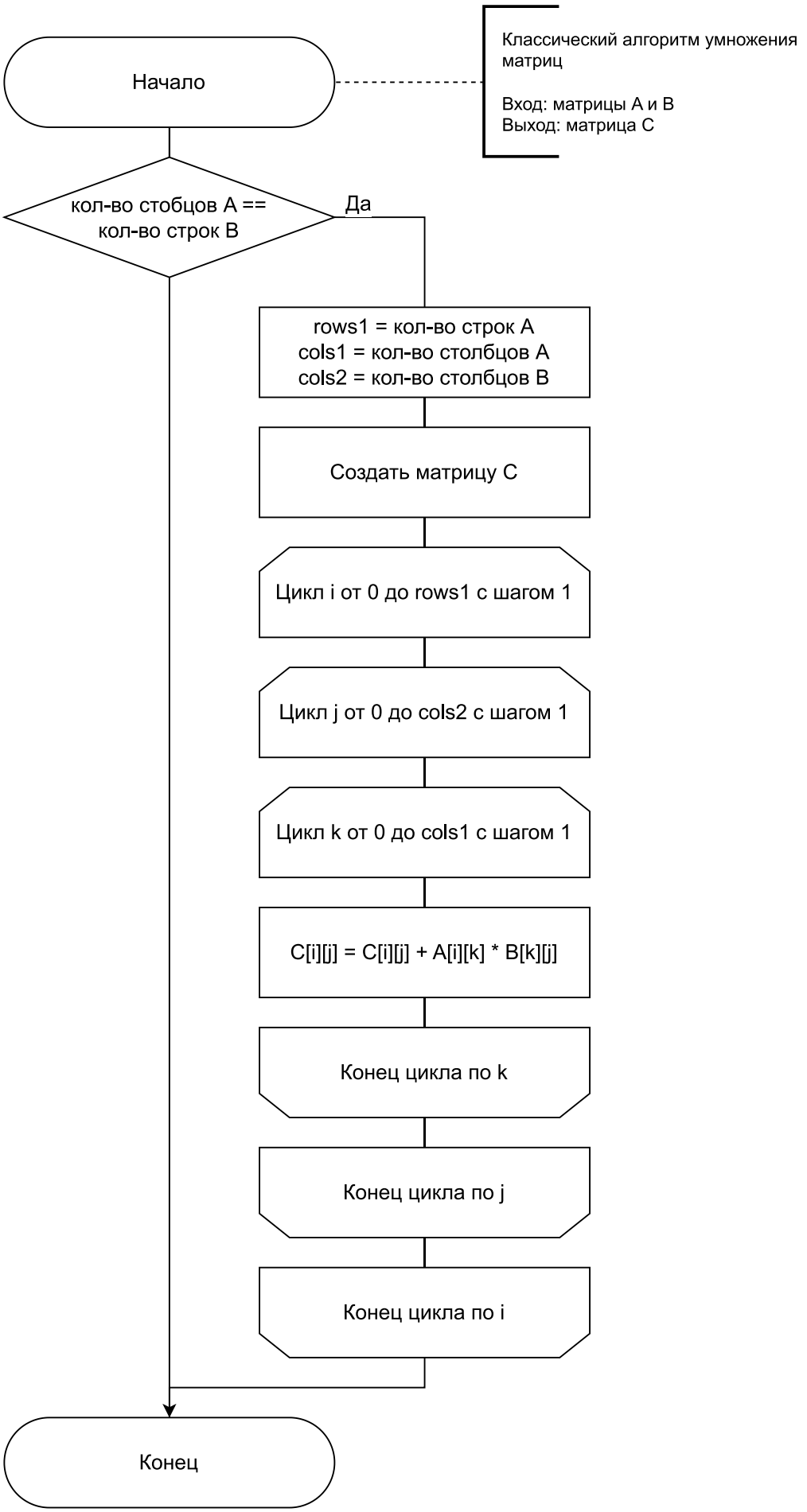

Рисунок 2.1 – Схема классического алгоритма умножения матриц

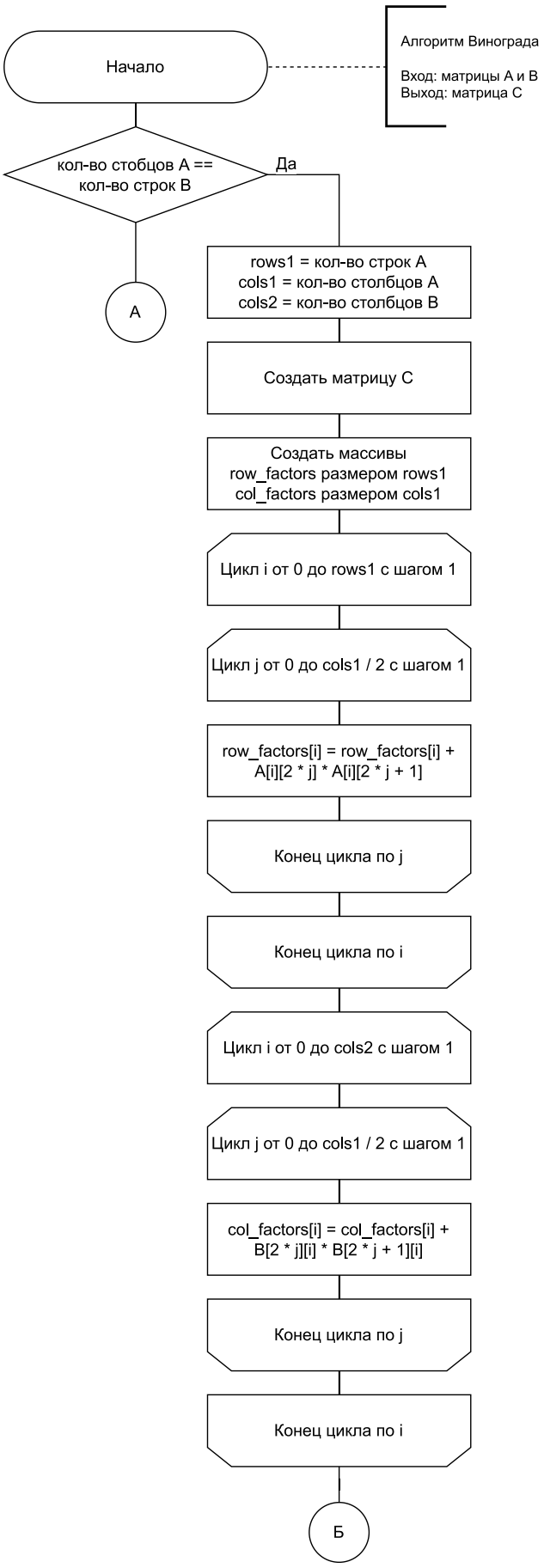

Рисунок 2.2 – Схема алгоритма Винограда

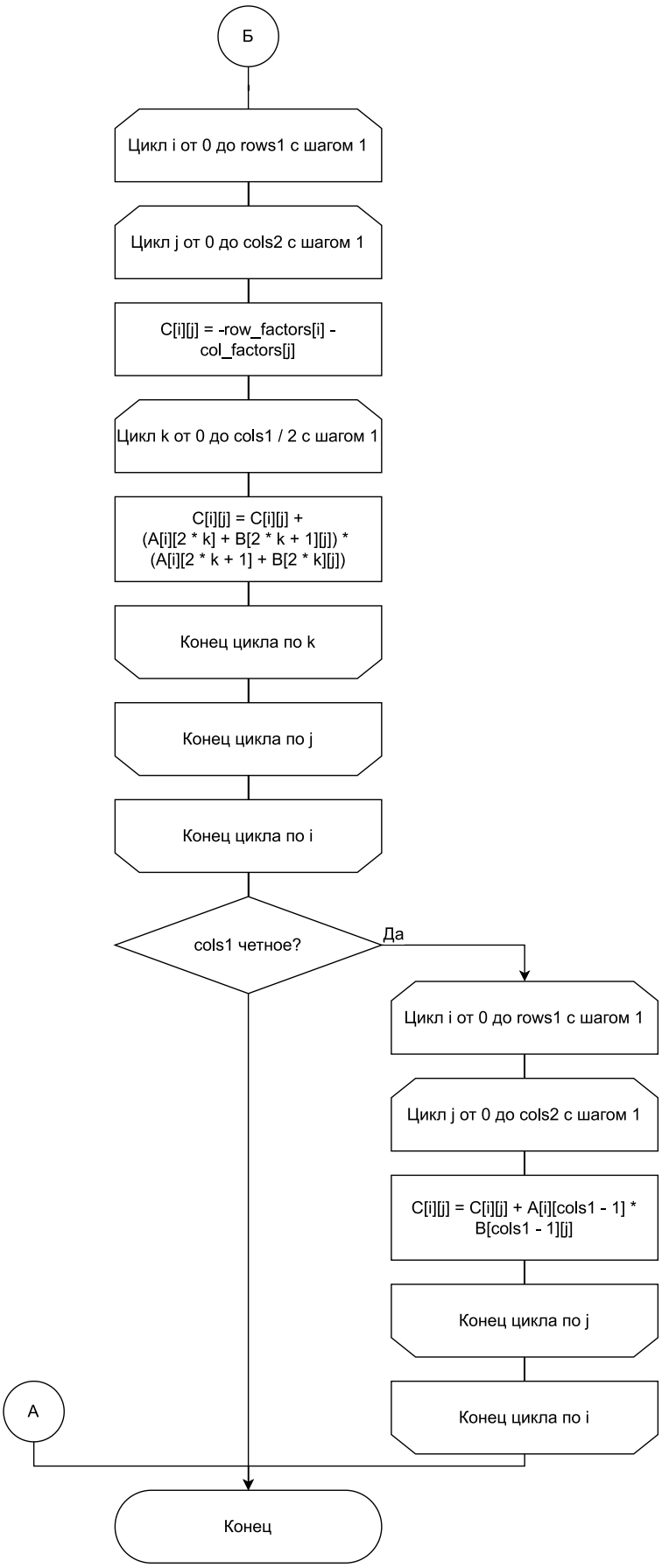

Рисунок 2.3 – Схема алгоритма Винограда (продолжение)

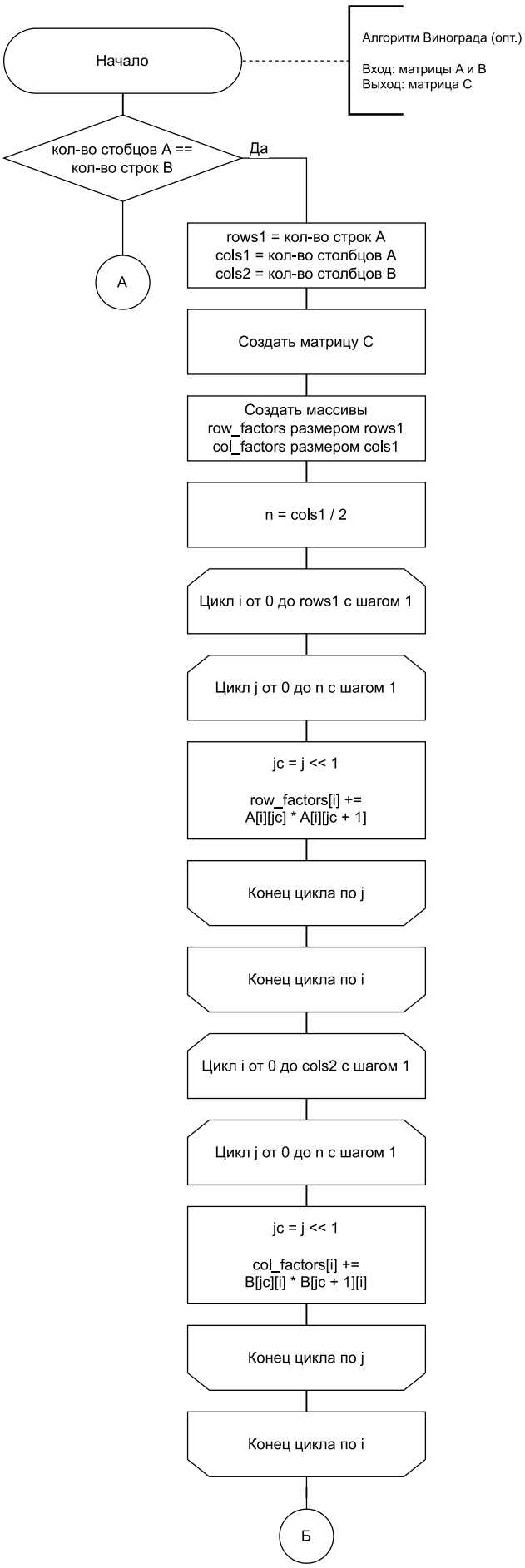

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда

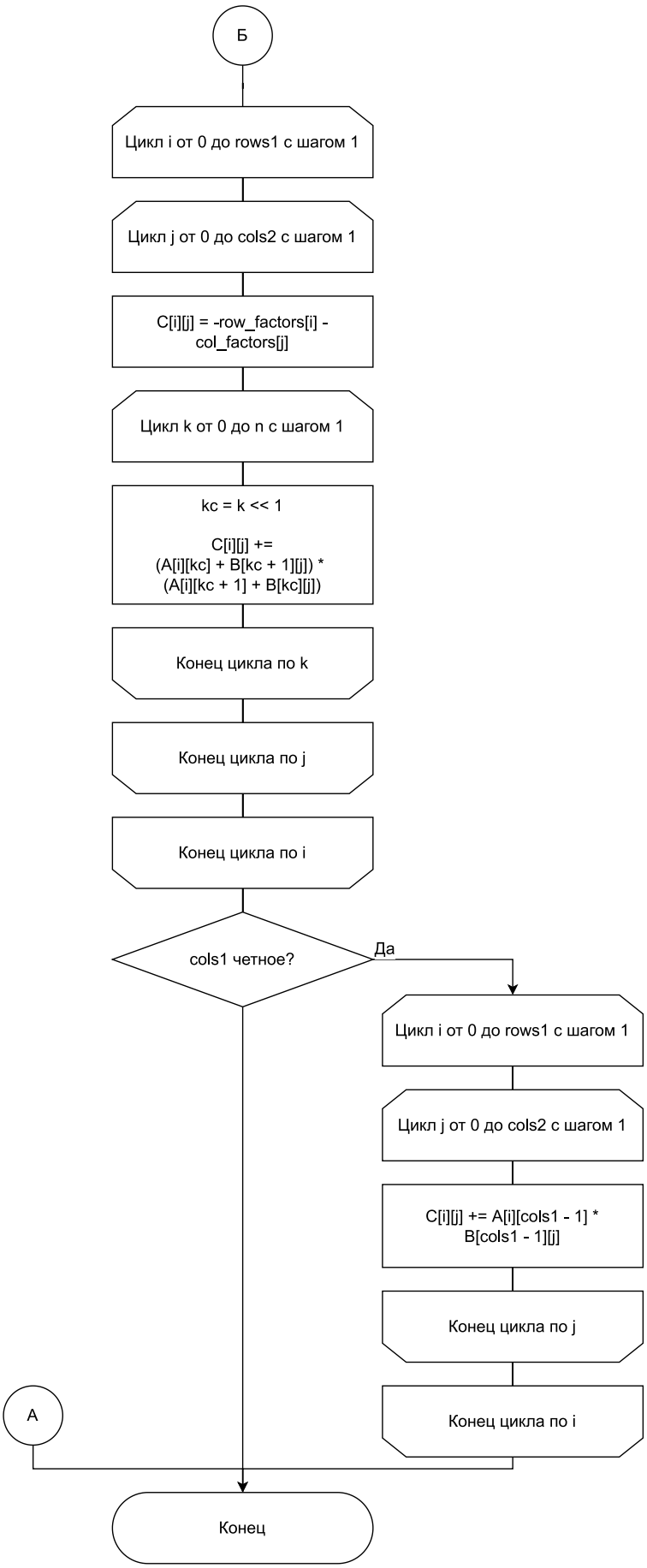

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда (продолжение)

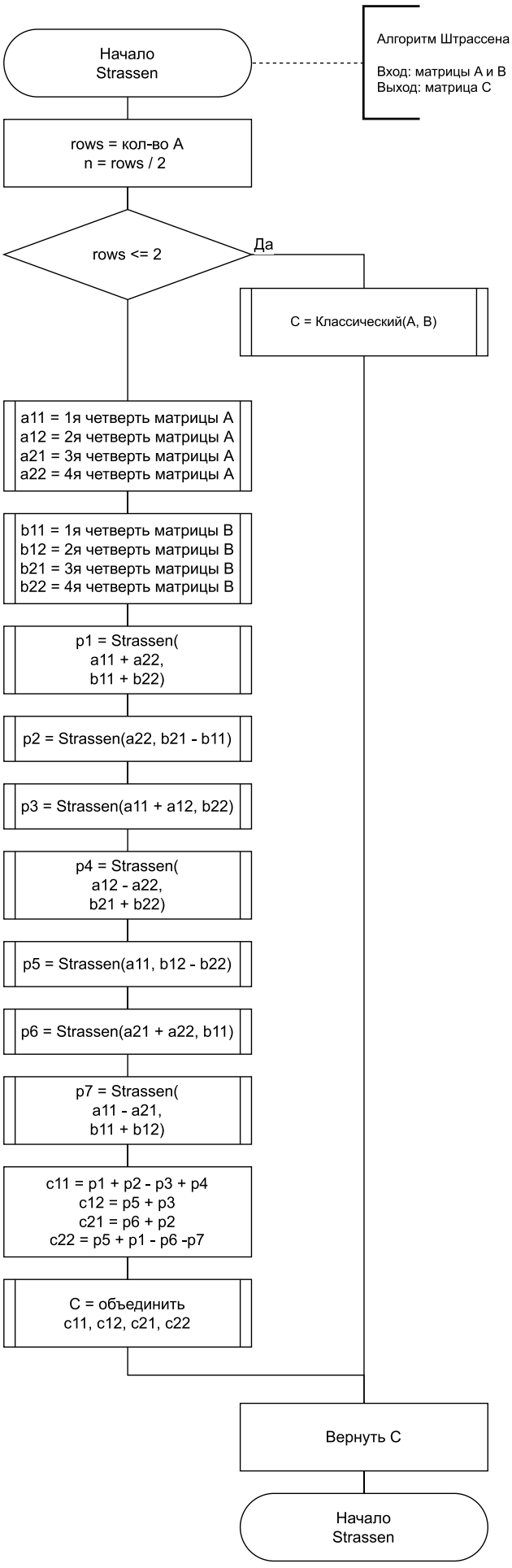

Рисунок 2.6 – Схема алгоритма Штрассена

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— *матрица* — двумерный массив значений типа `int`.

2.4 Модель вычисления для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма умножения матриц.

1) Трудоемкость базовых операций имеет:

— значение 1 для операций:

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

— значение 2 для операций:

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

2) Трудоемкость условного оператора:

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

3) Трудоемкость цикла

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

4) Трудоемкость передачи параметра в функцию и возврат из нее равен 0.

2.5 Трудоемкость алгоритмов

Далее будут приведены рассчеты трудоемкости реализуемых алгоритмов умножения матриц.

Пусть имеется две матрицы:

1) A размером $M \times N$,

2) B размером $N \times P$.

2.5.1 Классический алгоритм

Для стандартного алгоритма умножения матриц трудоемкость будет составлена из:

— цикла по $i \in [1 \dots M]$, трудоемкость которого $f = 2 + M \cdot (2 + f_{\text{тело}})$;

— цикла по $j \in [1 \dots P]$, трудоемкость которого $f = 2 + P \cdot (2 + f_{\text{тело}})$;

— цикла по $k \in [1 \dots N]$, трудоемкость которого $f = 2 + N \cdot (2 + 12)$;

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то

$$\begin{aligned} f_{Standard} &= 2 + M \cdot (2 + 2 + P \cdot (2 + 2 + N \cdot (3 + 3 + (2 + 2 + 2)))) = \\ &= 2 + 4M + 4MN + 14MNP \approx 14MNP = O(N^3) \end{aligned} \tag{2.5}$$

2.5.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда складывается из:

— трудоемкости создания и инициализации массивов $RowFactors$ и $ColFactors$:

$$f_{arrs} = f_{RowFactors} + f_{ColFactors} \tag{2.6}$$

— трудоемкость заполнения массива $RowFactors$:

$$f_{RowFactors} = 2 + M \cdot (2 + 4 + \frac{N}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ 2 + 6M + \frac{19MN}{2}; \tag{2.7}$$

— трудоемкость заполнения массива $ColFactors$:

$$f_{ColFactors} = 2 + P \cdot (2 + 4 + \frac{N}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ 2 + 6P + \frac{19NP}{2}; \tag{2.8}$$

— трудоемкость цикла умножения для четных размеров:

$$f_{mul} = 2 + M \cdot (4 + P \cdot (13 + 32\frac{N}{2})) = 2 + 4M + \\ +13MP + \frac{32MNP}{2} = 2 + 4M + 13MP + 16MNP; \tag{2.9}$$

— трудоемкость цикла, выполняемого в случае нечетных размеров матрицы:

$$f_{oddLoop} = 3 + \begin{cases} 0, & \text{четная} \\ 2 + M \cdot (4 + P \cdot (2 + 14)), & \text{иначе.} \end{cases} \tag{2.10}$$

Таким образом, для нечетного размера матрицы имеем:

$$f_{odd} = f_{arrs} + f_{RowFactors} + f_{ColFactors} + f_{mul} + f_{oddLoop} \approx \\ \approx 16MNP = O(N^3); \tag{2.11}$$

для четного:

$$f_{even} = f_{arrs} + f_{RowFactors} + f_{ColFactors} + f_{mul} + f_{oddLoop} \approx$$
$$\approx 16MNP = O(N^3). \quad (2.12)$$

2.5.3 Оптимизированный алгоритм Винограда

Оптимизация алгоритма Винограда осуществляется следующим образом:

— операция $x = x + k$ заменяется на операцию $x+ = k$;

— операция $x \cdot 2$ заменяется на $x << 1$;

— некоторые значения для алгоритма вычисляются заранее.

Тогда трудоемкость алгоритма Винограда с примененными оптимизациями складывается из:

— трудоемкости предвычисления значения $\frac{N}{2}$, равной 3;

— трудоемкости f_{arrs} (2.6) создания и инициализации массивов $RowFactors$ и $ColFactors$;

— трудоемкость заполнения массива $RowFactors$:

$$f_{RowFactors} = 2 + M \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 2 + 2 + 7)) = \\ 2 + 2M + \frac{13MN}{2}; \quad (2.13)$$

— трудоемкость заполнения массива $ColFactors$:

$$f_{ColFactors} = 2 + P \cdot (2 + 2 + \frac{N}{2} \cdot (2 + 2 + 2 + 7)) = \\ 2 + 2P + \frac{13NP}{2}; \quad (2.14)$$

— трудоемкость цикла умножения для четных размеров:

$$f_{mul} = 2 + M \cdot (4 + P \cdot (4 + 2 + \frac{N}{2} \cdot (2 + 2 + 3 + 6 + 2 + 6))) =$$
$$= 2 + 4M + 13MP + \frac{32MNP}{2} = 2 + 4M + 13MP + 16MNP; \tag{2.15}$$

— трудоемкость цикла, выполняемого в случае нечетных размеров матрицы:

$$f_{oddLoop} = 3 + \begin{cases} 0, & \text{четная} \\ 2 + M \cdot (4 + P \cdot (2 + 11)), & \text{иначе.} \end{cases} \tag{2.16}$$

Таким образом, для нечетного размера матрицы имеем:

$$f_{odd} = f_{arrs} + f_{RowFactors} + f_{ColFactors} + f_{mul} + f_{oddLoop} \approx$$
$$\approx \frac{19MNP}{2} = O(N^3); \tag{2.17}$$

для четного:

$$f_{even} = f_{arrs} + f_{RowFactors} + f_{ColFactors} + f_{mul} + f_{oddLoop} \approx$$
$$\approx \frac{19MNP}{2} = O(N^3). \tag{2.18}$$

2.5.4 Алгоритм Штрассена

Пусть матрицы A и B имеют размер $n \times n$, где n — степень 2.

Если $M(n)$ — количество умножений, выполняемых алгоритмом Штрассена для умножения двух матриц размером $n \times n$, то рекуррентное соотношение для $M(n)$ будет иметь следующий вид [4]:

$$M(n) = 7M\left(\frac{n}{2}\right), n > 1, M(1) = 1. \tag{2.19}$$

Поскольку $n = 2^k$:

$$M(2^k) = 7M(2^{k-1}) = 7 \cdot [7M(2^{k-2})] = 7^2 \cdot M(2^{k-2}) = \ldots = \\ = 7^i \cdot M(2^{k-i}) = \ldots = 7^k \cdot M(2^{k-k}) = 7^k. \tag{2.20}$$

Так как $k = \log_2(n)$:

$$M(n) = 7^{\log_2(7)} = n^{\log_2(7)} \approx n^{2.807}. \tag{2.21}$$

Рассмотрим количество сложений $A(n)$. Для умножения двух матриц порядка $n > 1$ алгоритму требуется 7 умножений и 18 сложений матриц размером $\frac{n}{2} \times \frac{n}{2}$ [4]; при $n > 1$ сложений не требуется, так как задача вырождается в перемножение двух чисел, поэтому количество сложений оценивается следующим образом:

$$A(n) = 7 \cdot A\left(\frac{n}{2}\right) + 18 \cdot \left(\frac{n}{2}\right)^2, n > 1, A(1) = 0. \tag{2.22}$$

Итоговая трудоемкость рассчитывается как:

$$T(n) = A(n) + M(n). \tag{2.23}$$

Вывод

В данном разделе на основе теоретических данных были построены схемы реализуемых алгоритмов умножения матриц. Также была проведена оценка трудоемкостей алгоритмов. В результате оценки алгоритма Винограда выяснилось, что оптимизированная версия в 1.6 раз менее трудоемка, чем неоптимизированная.

3 Технологическая часть

В данном разделе приведены средства реализации программного обеспечения, сведения о модулях программы, листинг кода и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [5], так как в нем имеется контейнер `std::vector`, представляющий собой массив динамический массив данных произвольного типа, и библиотека `<ctime>` [6], позволяющая производить замеры процессорного времени.

В качестве средства написания кода была выбрана кроссплатформенная среда разработки *Visual Studio Code* за счет того, что она предоставляет функционал для проектирования, разработки и отладки ПО.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл, содержащий точку входа в программу;

— `matrix.cpp` — файл, содержащий класс `Matrix`, реализующий необходимые для работы с матрицами функции;

— `algorithms.cpp` — файл, содержащий реализации алгоритмов умножения матриц;

— `measure.cpp` — файл, содержащий функции, замеряющие процессорное время выполнения реализуемых алгоритмов.

3.3 Реализация алгоритмов

Листинг 3.1 – Реализация классического алгоритма умножения матриц

```
1  Matrix Common(const Matrix &m1, const Matrix &m2)
2  {
3      size_t rows1 = m1.rows();
4      size_t cols1 = m1.columns();
5      size_t cols2 = m2.columns();
6
7      Matrix res(rows1, cols2, 0);
8
9      for (size_t i = 0; i < rows1; ++i)
10         for (size_t j = 0; j < cols2; ++j)
11             for (size_t k = 0; k < cols1; ++k)
12                 res[i][j] = res[i][j] + m1[i][k] * m2[k][j];
13
14     return res;
15 }
```

Листинг 3.2 – Реализация алгоритма Винограда

```
1  Matrix Winograd(const Matrix &m1, const Matrix &m2)
2  {
3      size_t rows1 = m1.rows();
4      size_t cols1 = m1.columns();
5      size_t cols2 = m2.columns();
6
7      Matrix res(rows1, rows1);
8
9      std::vector<int> row_factors(rows1, 0);
10     std::vector<int> col_factors(cols2, 0);
11
12     for (size_t i = 0; i < rows1; ++i)
13         for (size_t j = 0; j < cols1 / 2; ++j)
14             row_factors[i] = row_factors[i] + m1[i][2 * j] *
                   m1[i][2 * j + 1];
15
16     for (size_t i = 0; i < cols2; ++i)
17         for (size_t j = 0; j < cols1 / 2; ++j)
18             col_factors[i] = col_factors[i] + m2[2 * j][i] *
                   m2[2 * j + 1][i];
19
20     for (size_t i = 0; i < rows1; ++i)
21     {
22         for (size_t j = 0; j < cols2; ++j)
23         {
24             res[i][j] = -row_factors[i] - col_factors[j];
25             for (size_t k = 0; k < cols1 / 2; ++k)
26             {
27                 res[i][j] = res[i][j] + (m1[i][2 * k] + m2[2 *
                       k + 1][j]) *
28                     (m1[i][2 * k + 1] + m2[2 * k][j]);
29             }
30         }
31     }
32
33     if (cols1 % 2)
34     {
35         for (size_t i = 0; i < rows1; ++i)
36             for (size_t j = 0; j < cols2; ++j)
37                 res[i][j] = res[i][j] + m1[i][cols1 - 1] *
```

```
38                  m2[cols1 - 1][j];
39      }
40
41      return res;
42  }
```

Листинг 3.3 – Реализация алгоритма Винограда с оптимизациями

```
1  Matrix WinogradOpt(const Matrix &m1, const Matrix &m2)
2  {
3      size_t rows1 = m1.rows();
4      size_t cols1 = m1.columns();
5      size_t cols2 = m2.columns();
6
7      Matrix res(rows1, rows1);
8
9      std::vector<int> row_factors(rows1, 0);
10     std::vector<int> col_factors(cols2, 0);
11
12     size_t half_cols1 = cols1 / 2;
13
14     for (size_t i = 0; i < rows1; ++i)
15     {
16         for (size_t j = 0; j < half_cols1; ++j)
17         {
18             size_t j_mul = j << 1;
19             row_factors[i] += m1[i][j_mul] * m1[i][j_mul + 1];
20         }
21     }
22
23     for (size_t i = 0; i < cols2; ++i)
24     {
25         for (size_t j = 0; j < half_cols1; ++j)
26         {
27             size_t j_mul = j << 1;
28             col_factors[i] += m2[j_mul][i] * m2[j_mul + 1][i];
29         }
30     }
31
32     for (size_t i = 0; i < rows1; ++i)
33     {
34         for (size_t j = 0; j < cols2; ++j)
35         {
36             res[i][j] = -row_factors[i] - col_factors[j];
37             for (size_t k = 0; k < half_cols1; ++k)
38             {
39                 size_t k_mul = k << 1;
40                 res[i][j] += (m1[i][k_mul] + m2[k_mul + 1][j]) *
```

```
41                                  (m1[i][k_mul + 1] + m2[k_mul][j]);
42              }
43          }
44      }
45
46      if (cols1 % 2)
47      {
48          for (size_t i = 0; i < rows1; ++i)
49              for (size_t j = 0; j < cols2; ++j)
50                  res[i][j] += m1[i][cols1 - 1] *
51                               m2[cols1 - 1][j];
52      }
53
54      return res;
55  }
```

Листинг 3.4 – Реализация алгоритма Штрассена

```
Matrix Strassen(const Matrix &m1, const Matrix &m2)
{
    size_t rows = m1.rows();

    if (rows <= 2)
    {
        return Common(m1, m2);
    }

    size_t n = m1.rows() / 2;

    auto a11 = m1.slice(0, n, 0, n);
    auto a12 = m1.slice(0, n, n, rows);
    auto a21 = m1.slice(n, rows, 0, n);
    auto a22 = m1.slice(n, rows, n, rows);

    auto b11 = m2.slice(0, n, 0, n);
    auto b12 = m2.slice(0, n, n, rows);
    auto b21 = m2.slice(n, rows, 0, n);
    auto b22 = m2.slice(n, rows, n, rows);

    auto p1 = Strassen(a11 + a22, b11 + b22);
    auto p2 = Strassen(a22, b21 - b11);
    auto p3 = Strassen(a11 + a12, b22);
    auto p4 = Strassen(a12 - a22, b21 + b22);
    auto p5 = Strassen(a11, b12 - b22);
    auto p6 = Strassen(a21 + a22, b11);
    auto p7 = Strassen(a11 - a21, b11 + b12);

    auto c11 = p1 + p2 - p3 + p4;
    auto c12 = p5 + p3;
    auto c21 = p6 + p2;
    auto c22 = p5 + p1 - p6 - p7;

    return Matrix::combine(c11, c12, c21, c22);
}
```

3.4 Функциональные тесты

На таблице 3.1 представлены функциональные тесты стандартного алгоритма умножения матриц.

На таблице 3.2 представлены функциональные тесты алгоритма Винограда.

На таблице 3.3 показаны функциональные тесты алгоритма Штрассена.

Таблица 3.1 – Функциональные тесты для классического алгоритма

Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix} 9 \end{pmatrix}$	$\begin{pmatrix} 3 \end{pmatrix}$	$\begin{pmatrix} 27 \end{pmatrix}$	$\begin{pmatrix} 27 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{pmatrix}$
$\begin{pmatrix} 2 & 10 \\ 3 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 10 \\ 3 & 7 \end{pmatrix}$	$\begin{pmatrix} 2 & 10 \\ 3 & 7 \end{pmatrix}$
$\begin{pmatrix} 4 & 2 & 9 \\ 1 & 3 & 1 \\ 0 & 5 & 3 \end{pmatrix}$	$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 8 & 4 & 9 \\ 2 & 6 & 1 \\ 0 & 10 & 3 \end{pmatrix}$	$\begin{pmatrix} 8 & 4 & 9 \\ 2 & 6 & 1 \\ 0 & 10 & 3 \end{pmatrix}$
$\begin{pmatrix} 4 & 2 & 9 \\ 1 & 3 & 1 \\ 0 & 5 & 3 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*

Таблица 3.2 – Функциональные тесты для алгоритма Винограда

Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix}9\end{pmatrix}$	$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}27\end{pmatrix}$	$\begin{pmatrix}27\end{pmatrix}$
$\begin{pmatrix}1\\2\\3\end{pmatrix}$	$\begin{pmatrix}4 & 5 & 6\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*
$\begin{pmatrix}2 & 10\\3 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	$\begin{pmatrix}2 & 10\\3 & 7\end{pmatrix}$	$\begin{pmatrix}2 & 10\\3 & 7\end{pmatrix}$
$\begin{pmatrix}4 & 2 & 9\\1 & 3 & 1\\0 & 5 & 3\end{pmatrix}$	$\begin{pmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}8 & 4 & 9\\2 & 6 & 1\\0 & 10 & 3\end{pmatrix}$	$\begin{pmatrix}8 & 4 & 9\\2 & 6 & 1\\0 & 10 & 3\end{pmatrix}$
$\begin{pmatrix}4 & 2 & 9\\1 & 3 & 1\\0 & 5 & 3\end{pmatrix}$	$\begin{pmatrix}\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*
$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}4 & 5 & 6\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*

Таблица 3.3 – Функциональные тесты для алгоритма Штрассена

Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix}9\end{pmatrix}$	$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}27\end{pmatrix}$	$\begin{pmatrix}27\end{pmatrix}$
$\begin{pmatrix}1\\2\\3\end{pmatrix}$	$\begin{pmatrix}4 & 5 & 6\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*
$\begin{pmatrix}2 & 10\\3 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	$\begin{pmatrix}2 & 10\\3 & 7\end{pmatrix}$	$\begin{pmatrix}2 & 10\\3 & 7\end{pmatrix}$
$\begin{pmatrix}4 & 2 & 9\\1 & 3 & 1\\0 & 5 & 3\end{pmatrix}$	$\begin{pmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 1\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*
$\begin{pmatrix}4 & 2 & 9\\1 & 3 & 1\\0 & 5 & 3\end{pmatrix}$	$\begin{pmatrix}\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*
$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}4 & 5 & 6\end{pmatrix}$	*Сообщение об ошибке*	*Сообщение об ошибке*

Вывод

Были реализованы алгоритмы умножения матриц (классический, алгоритм Винограда, алгоритм Штрассена). Проведено тестирование реализованных алгортимов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Intel i5-1035G1 (8) @ 3.600 ГГц.

— Оперативная память: 16 ГБайт.

— Операционная система: Manjaro Linux x86_64 (версия ядра Linux 5.15.131-1-MANJARO).

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 показан пример работы разработанной программы для случая, когда пользователь выбирает опцию «Умножение алгоритмом Винограда» и затем — «Умножение алгоритмом Штрассена». Входными данными являются матрицы

$$\begin{pmatrix} 0 & 4 & 1 & 0 \\ 5 & 1 & 4 & 6 \\ 9 & 3 & 5 & 1 \\ 0 & 5 & 7 & 9 \end{pmatrix} \text{ и } \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
                Меню
1. Стандартное умножение матриц.
2. Умножение алгоритмом Винограда.
3. Умножение оптимизированным алгоритмом Винограда.
4. Умножение алгоритмом Штрассена.
5. Замерить время для реализованных алгоритмов.
6. Редактировать матрицы
0. Выход.

Выберите опцию (0-6): 2

[       0     20      5      0 ]
[      25      5     20      6 ]
[      45     15     25      1 ]
[       0     25     35      9 ]

                Меню
1. Стандартное умножение матриц.
2. Умножение алгоритмом Винограда.
3. Умножение оптимизированным алгоритмом Винограда.
4. Умножение алгоритмом Штрассена.
5. Замерить время для реализованных алгоритмов.
6. Редактировать матрицы
0. Выход.

Выберите опцию (0-6): 4

[       0     20      5      0 ]
[      25      5     20      6 ]
[      45     15     25      1 ]
[       0     25     35      9 ]

                Меню
1. Стандартное умножение матриц.
2. Умножение алгоритмом Винограда.
3. Умножение оптимизированным алгоритмом Винограда.
4. Умножение алгоритмом Штрассена.
5. Замерить время для реализованных алгоритмов.
6. Редактировать матрицы
0. Выход.

Выберите опцию (0-6): 0
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Временные характеристики

Исследование временных характеристик реализуемых алгоритмов производилось три раза:

1) на квадратных матрицах нечетного размера, который изменяется от 1 до 101 с шагом 10;

2) на квадратных матрицах четного размера, который изменяется от 10 до 100 с шагом 10;

3) на квадратных матрицах, размер которых — степень двойки от 2 до 128.

В силу того, что время работы алгоритмов может колебаться в связи с различными процессами, происходящими в системе, для обеспечения более точных результатов измерения для каждого алгоритма повторялись 100 раз, а затем бралось их среднее арифметическое значение.

На рисунке 4.2 показаны зависимости времени выполнения классического алгоритма умножения, алгоритма Винограда (без оптимизации и с ней) от нечетного размера квадратных матриц.

На рисунке 4.3 представлены зависимости времени выполнения классического алгоритма умножения, алгоритма Винограда (без оптимизации и с ней) от четного размера квадратных матриц.

На рисунке 4.4 представлены зависимости времени выполнения классического алгоритма умножения, алгоритмов Винограда (без оптимизации и с ней) и Штрассена от матриц, размер которых — степень 2.

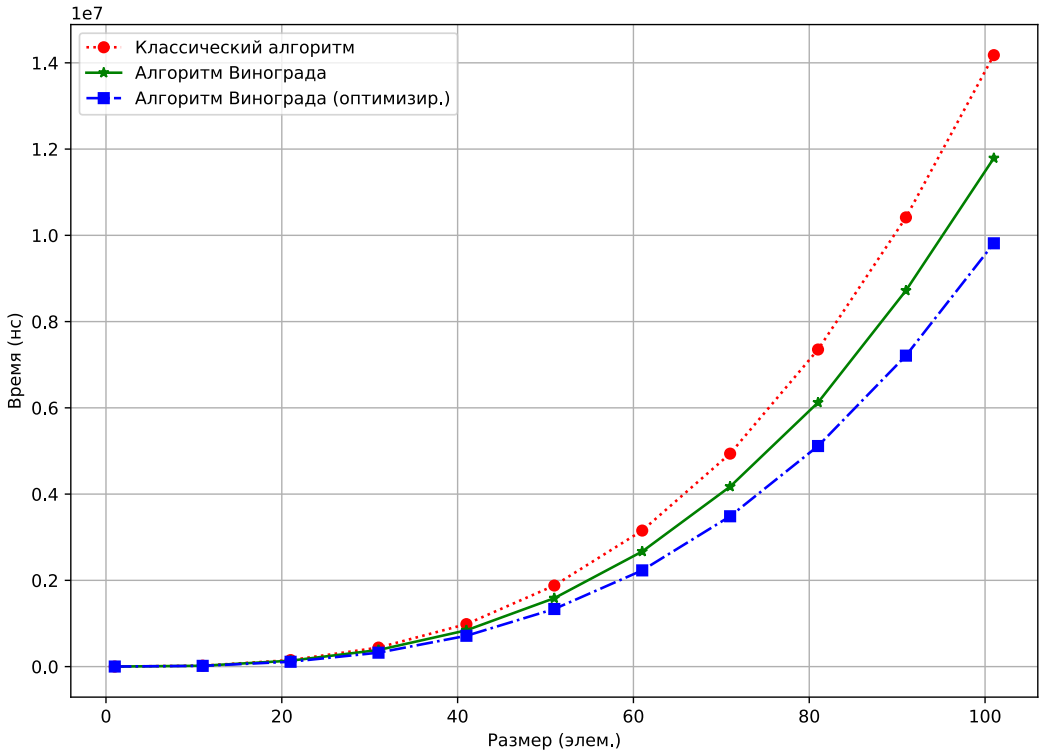

Рисунок 4.2 – Результат измерений времени работы реализуемых алгоритмов на матрицах нечетных размеров

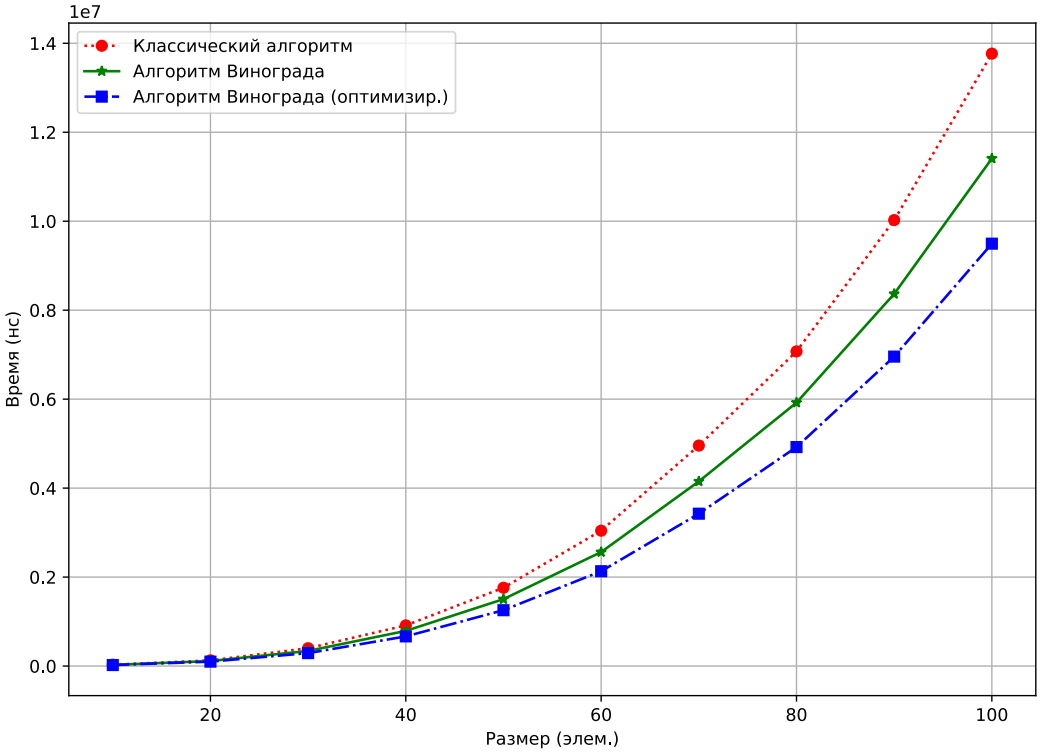

Рисунок 4.3 – Результат измерений времени работы реализуемых алгоритмов на матрицах четных размеров

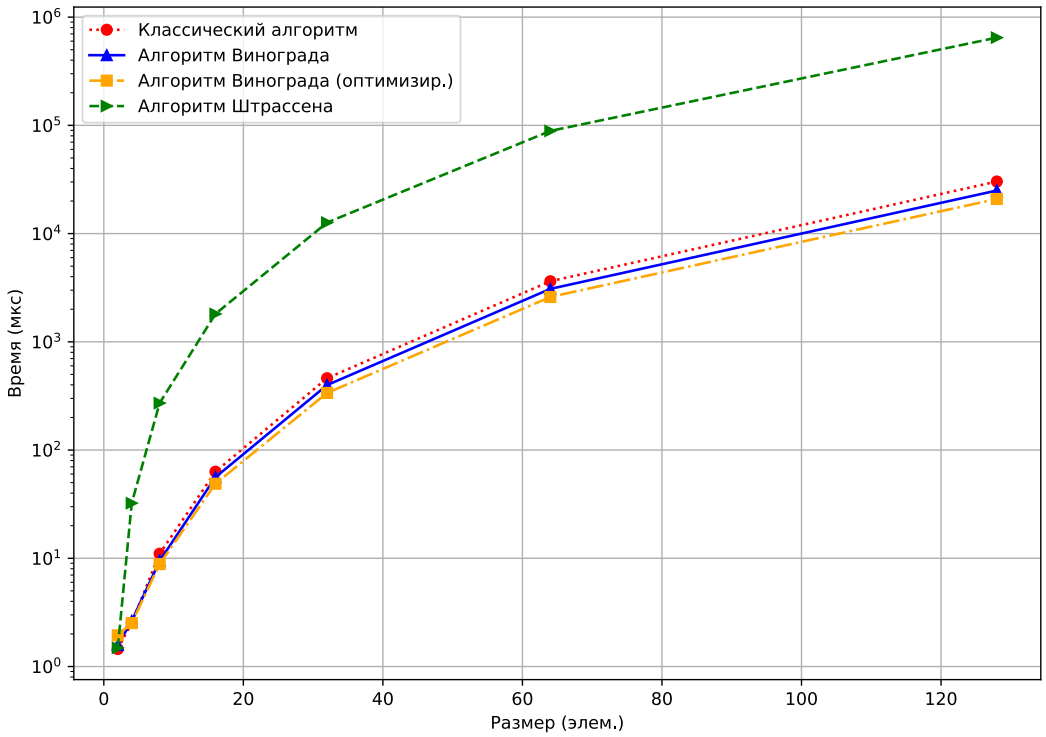

Рисунок 4.4 – Результат измерений времени работы реализуемых алгоритмов на матрицах, размеры которых — степень 2

4.4 Характеристики по памяти

Введем следующие обозначения:

— size(v) — функция, вычисляющая размер входного параметра v в байтах;

— int — целочисленный тип данных.

Теоретически оценим объем используемой памяти алгоритмов умножения для целочисленной матрицы размером $n \times n$.

4.4.1 Классический алгоритм

Оценка используемой классическим алгоритмом умножения памяти приведена в формуле (4.1).

$$M_{Classic} = 3 \cdot \text{size}(int) + 3 \cdot \text{size}(int) + n \cdot n \cdot \text{size}(int) = \\ = \text{size}(int) \cdot (6 + n^2), \tag{4.1}$$

где $3 \cdot \text{size}(int)$ — размер переменных для хранения размеров матриц,
$3 \cdot \text{size}(int)$ — размер переменных цикла,
$n \cdot n \cdot \text{size}(int)$ — размер результирующей матрицы.

4.4.2 Алгоритм Винограда

Оценка используемой алгоритмом Винограда памяти приведена в формуле (4.1).

$$M_{Winograd} = M_{RowFactors} + M_{ColFactors} + M_{res} + M_{mul} + M_{odd}, \tag{4.2}$$

где $M_{RowFactors}, M_{ColFactors}$ — размер вспомогательных массивов,
M_{res} — размер результирующей матрицы,
M_{mul} — размер памяти, используемой при умножении матриц,
M_{odd} — размер памяти, используемой при обработке условия о нечетности размеров матриц.

Размер результирующей матрицы рассчитывается по формуле (4.3)

$$M_{res} = n \cdot n \cdot \text{size}(int). \tag{4.3}$$

Количество памяти, затрачиваемой на хранение массивов $RowFactors$ и $ColFactors$ равно

$$M_{RowFactors} = M_{ColFactors} = n \cdot \text{size}(int). \tag{4.4}$$

При умножении размер используемой памяти равен

$$M_{mul} = 3 \cdot \text{size}(int), \tag{4.5}$$

где $3 \cdot \text{size}(int)$ — переменные цикла.

Количество памяти, затрачиваемой на обработку случая, когда мат-

рица имеет нечетный размер, равно

$$M_{odd} = \begin{cases} 0, & \text{четная} \\ 2 \cdot \text{size}(int), & \text{нечетная} \end{cases} \tag{4.6}$$

4.4.3 Оптимизированный алгоритм Винограда

Оценка используемой оптимизированной версией алгоритма Винограда памяти приведена в формуле (4.1).

$$M_{WinogradOpt} = M_{RowFactors} + M_{ColFactors} + M_{res} + M_{mul} + M_{odd} + \text{size}(int), \tag{4.7}$$

где $M_{RowFactors}, M_{ColFactors}$ — размер вспомогательных массивов,
M_{res} — размер результирующей матрицы,
M_{mul} — размер памяти, используемой при умножении матриц,
M_{odd} — размер памяти, используемой при обработке условия о нечетности размеров матриц,
size(int) — размер переменной, используемой для кеширования.

Размер памяти, необходимой для обработки массивов $RowFactors$ и $ColFactors$, равен

$$M_{RowFactors} = M_{ColFactors} = n \cdot \text{size}(int) + \text{size}(int), \tag{4.8}$$

где size(int) — размер переменной, используемой для кеширования.

Количество памяти, затрачиваемой на умножение, равно

$$M_{mul} = 3 \cdot \text{size}(int) + \text{size}(int), \tag{4.9}$$

где size(int) — размер переменной, используемой для кеширования.

M_{res} и M_{odd} рассчитываются согласно соотношениям (4.3) и (4.6) соответственно.

4.4.4 Алгоритм Штрассена

Ниже приведена оценка количества памяти, затрачиваемой при каждом рекурсивном вызове алгоритма Штрассена.

$$M_{StrassenCall} = (4 \cdot \frac{n}{2} \cdot \frac{n}{2} + 4 \cdot \frac{n}{2} \cdot \frac{n}{2} + 11 \cdot \frac{n}{2} \cdot \frac{n}{2} + n \cdot n) \cdot \text{size}(int) + \\ +2 \cdot \text{size}(int), \tag{4.10}$$

где $2 \cdot \text{size}(int)$ — размер переменных, используемых для кеширования,
$4 \cdot \frac{n}{2} \cdot \frac{n}{2} \cdot \text{size}(int)$ — размер матриц, полученных в результате разбиения исходных на 4 части,
$11 \cdot \frac{n}{2} \cdot \frac{n}{2} \cdot \text{size}(int)$ — размер промежуточных матриц,
$n \cdot n \cdot \text{size}(int)$ — размер результирующей матрицы.

4.5 Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы:

1) оптимизированный алгоритм Винограда оказался самым эффективным по времени независимо от размерности входных матриц (см. рисунки 4.2 – 4.4).

2) время работы классического алгоритма умножения и оптимизированного и неоптимизированного алгоритмов Винограда на матрицах нечетного размера больше, чем на матрицах четного размера (см. рисунки 4.2, 4.3); для алгоритма Винограда меньшая скорость работы на матрицах нечетного размера объясняется необходимостью дополнительных вычислений крайних строк и столбцов в результирующей матрице;

3) алгоритм Штрассена показал наименьшую производительность среди всех алгоритмов, исследуемых на матрицах, размер которых равен степени двойки (см. рисунок 4.4); вероятно, низкая производи-

тельность алгоритма обуславливается необходимостью выполнения дополнительных операций сложения/вычитания, разбиения/слияния матриц;

В результате теоретической оценки алгоритмов по памяти можно сделать вывод о том, что стандартный алгоритм умножения матриц требует наименьших расходов по памяти. Алгоритм Штрассена, напротив, является самым требовательным по памяти за счет использования вспомогательных подматриц для выполнения рассчетов и рекурсивных вызовов.

Оптимизированный алгоритм Винограда является более ресурсозатратным, чем неоптимизированный, так как первый алгоритм использует дополнительные переменные для хранения промежуточных рассчетов.

Заключение

В результате выполнения лаботарторной работы по исследованию алгоритмов умножения матриц решены следующие задачи:

1) описаны алгоритмы умножения матриц;
2) разработаны и реализованы соответствующие алгоритмы;
3) создан программный продукт, позволяющий протестировать реализованные алгоритмы;
4) проведен сравнительный анализ процессорного времени выполнения реализованных алгоритмов:
   - оптимизированный алгоритм Винограда оказался самым эффективным по времени независимо от размерности входных матриц;
   - время работы классического алгоритма умножения и оптимизированного и неоптимизированного алгоритмов Винограда на матрицах нечетного размера больше, чем на матрицах четного размера; для алгоритма Винограда меньшая скорость работы на матрицах нечетного размера объясняется необходимостью дополнительных вычислений крайних строк и столбцов в результирующей матрице;
   - алгоритм Штрассена показал наименьшую производительность среди всех алгоритмов, исследуемых на матрицах, размер которых равен степени двойки; низкая производительность алгоритма обуславливается необходимостью выполнения дополнительных операций сложения/вычитания, разбиения/слияния матриц;
5) выполнена теоретическая оценка объема затрачиваемой памяти каждым из реализованных алгоритмов: стандартный алгоритм умножения матриц является наименее ресурсозатратным из всех реализованных алгоритмов; самым же требовательным по памяти оказался алгоритм Штрассена за счет использования вспомогательных подматриц

для выполнения рассчетов и рекурсивных вызовов; оптимизированный алгоритм Винограда использует больше ресурсов по сравнению с неоптимизированным, за счет предвычисления некоторых выражений.

Список использованных источников

1 С. Анисимов Н., В. Строганов Ю. Реализация умножения матриц по Винограду на языке Haskell. // Новые информационные технологии в автоматизированных системах. 2018. URL: — Режим доступа: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell (дата обращения 03.11.2023).

2 Н. Канатников А., П. Крищенко А. Аналитическая геометрия. Конспект лекций. 2009. Т. 132.

3 Лекция 6: Сложность рекурсивных алгоритмов. Умножение матриц (над кольцом и булевых). Простой рекурсивный алгоритм для умножения целых чисел. Нахождение пары ближайших точек на плоскости. — Режим доступа: https://logic.pdmi.ras.ru/~hirsch/students/cs2005/lecture6.pdf (дата обращения: 07.11.2023).

4 Р. Лапина Н., Е. Булатникова М. Алгоритм Штрассена для умножения матриц // Математика и ее приложения в современной науке и практике. 2014. URL: — Режим доступа: https://elibrary.ru/item.asp?id=23140890 (дата обращения 07.11.2023).

5 Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017 (дата обращения: 25.09.2023).

6 Standard library header <ctime> [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/cpp/header/ctime.