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) Савинова М. Г. (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 исследование алгоритмов умножения матриц.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1) описать следующие алгоритмы умножения матриц:
   - классический алгоритм умножения;
   - алгоритм Винограда;
   - оптимизированный алогритм Винограда;
   - алгоритм Штрассена;
2) релизовать описанные алгоритмы;
3) дать оценку трудоемкости алгоритмов;
4) дать оценку потребляемой памяти реализациями алгоритмов;
5) провести замеры времени выполнения алгоритмов.

1 Аналитическая часть

В данном разделе будут рассмотрены классический алгоритм умножениям матриц, алгоритм Винограда и его же оптимизированная версия.

1.1 Матрица

Матрицей размером $m \times n$ называют прямоугольную числовую таблицу, состоящую из $m \cdot n$ чисел, которые расположены в m строках и n столбцах. Состовляющие матрицу числа называют *элементами* этой *матрицы* [2].

Матрицу обозначают

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

Над матрицами возможны следующие операции:

— сложение матриц одинакового размера;

— умножение матрицы на число;

— умножение матриц, которое определено лишь в случае, когда количество *столбцов* первого сомножителя равно количеству *строк* второго [2].

1.2 Классический алгоритм

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

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

которую обозначают $C = AB$.

Классический алгоритм реализует формулу 1.2.

1.3 Алгоритм Винограда

Одним из самых эффективных по времени алгоритмов умножения матриц является алгоритм Винограда, имеющий асимптотическую сложность $O(n^{2,3755})$ [1].

Рассмотрим два вектора:

$$U = (u_1, u_2, u_3, u_4), \tag{1.3}$$

$$V = (v_1, v_2, v_3, v_4). \tag{1.4}$$

Их скалярное произведение равно:

$$U \times V = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4, \tag{1.5}$$

что равносильно

$$U \times V = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3). \tag{1.6}$$

Для упомянутых раннее матриц A, B и C скалярное произведение, по замыслу Винограда, 1.5 можно свести к следующему выражению:

$$c_{ij} = \sum_{k=1}^{n/2}(a_{i,2k-1} + b_{j,2k})(a_{i,2k} + b_{j,2k-1}) - \\ - \sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{n/2} b_{2k-1,j}b_{2k,j}. \tag{1.7}$$

В целях экономии количества арифметических операций Виноград предложил находить второе и третье слагаемое в 1.7 заранее для каждой строки матрицы A и каждого столбца матрицы B.

Так, единожды вычислив для i-ой строки матрицы A значение выражения $\sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k}$, его можно использовать далее n раз для нахож-

дения элементов i-ой строки матрицы C.

Аналогично, единожды вычислив для j-ой столбца матрицы B значение выражения $\sum_{k=1}^{n/2} b_{2k-1,j} b_{2k,j}$, его можно использовать далее n раз для нахождения элементов j-ой столбца матрицы C.

Для примера, приведенного в формуле 1.6, в классическом умножении производится четыре умножения и три сложения; в алоритме Винограда — шесть умножений и девять сложений. Но, несмотря на увеличение количества операций, выражение в правой части можно вычислить заранее и запомнить для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволит выполнить лишь два умножения и пять сложений, складывая затем только лишь с двумя предварительно вычисленными суммами соседних элементов текущих строк и столбцов. Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее классического алгоритма умножения матриц [3].

При условии нечетного размера матрицы необходимо дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.4 Оптимизированный алгоритм Винограда

Для программной реализации алгоритма, рассмотренного в предыдущем пункте, можно выполнить следующие оптимизации:

1) значение $n/2$, используемое в качестве ограничения цикла подсчета предварительных данных, можно кешировать;

2) операцию умножения на 2 эффективнее реализовать как побитовый сдвиг влево на 1;

3) при условии существования операторов `+=, -=` в выбранном языке программирования, соответсвующие операции сложения и вычитания с присваиванием следует реализовывать с помощью данных операторов.

1.5 Алгоритм Штрассена

Произведение C двух матриц A и B размером 2×2 можно вычислить с помощью только **7**, а не **8** умножений, которые необходимы при использовании стандартного алгоритма. Действие осуществляются с использованием следующих формул:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \\ = \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}, \tag{1.8}$$

где

$$\begin{aligned} m_1 &= (a_{11} + a_{22}) \cdot (b_{11} + b_{22}), \\ m_2 &= (a_{21} + a_{22}) \cdot b_{11}, \\ m_3 &= (b_{12} - b_{22}) \cdot a_{11}, \\ m_4 &= a_{22} \cdot (b_{21} - b_{11}), \\ m_5 &= (a_{11} + a_{12}) \cdot b_{22}, \\ m_6 &= (a_{21} - a_{11}) \cdot (b_{11} + b_{12}), \\ m_7 &= (a_{12} - a_{22}) \cdot (b_{12} + b_{22}). \end{aligned} \tag{1.9}$$

Пусть матриц A и B размером $n \times n$, где n — стпень двойки. Матрицы A и B и их произведение C можно разделить на 4 подматрицы размером $\frac{n}{2} \times \frac{n}{2}$ каждую следующим образом:

$$\begin{pmatrix} {}_{11} & {}_{12} \\ {}_{21} & {}_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22}. \end{pmatrix} \tag{1.10}$$

Для получения корректного произведения эти подматрицы рассматривают как числа. Алгоритм Штрассена состоит в том, что требуемые 7 произведений подматриц размером $\frac{n}{2} \times \frac{n}{2}$ вычисляются рекурсивно и использованием описанного метода [4].

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц: классический, алгоритм Винограда, алгоритм Штрассена. Также были рассмотрены оптимизации, которые можно учесть при програмной реализации алгоритма Винограда.

Основным отличием этих алгоритмов является наличие предварительных вичислений — как следствие, количество операций умножения и сложения также различно.

2 Конструкторская часть

В данном разделе будут реализованы схемы алгоритмов умножения матриц и будут приведены рассчеты трудоемкостей для этих алгоритмов.

2.1 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма для стандартного умножения.

На рисунках 2.2–2.3 представлены схемы алгоритма умножения методом Винограда.

На рисунках 2.4–2.5 представлены схемы оптимизированного алгоритма умножения методом Винограда.

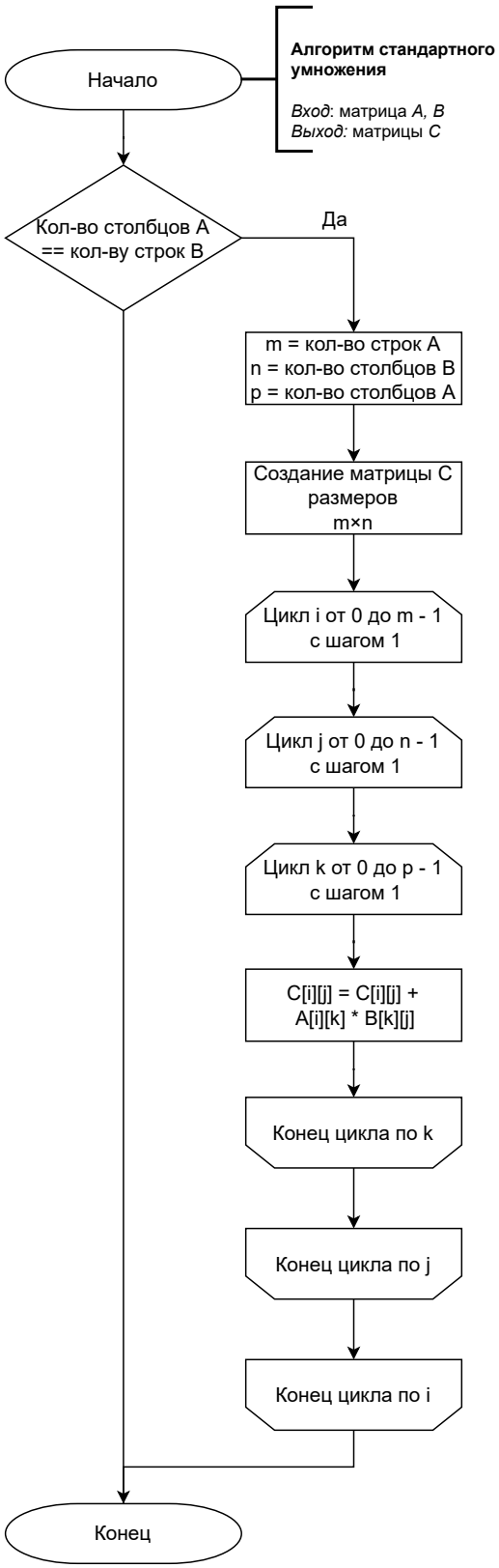

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

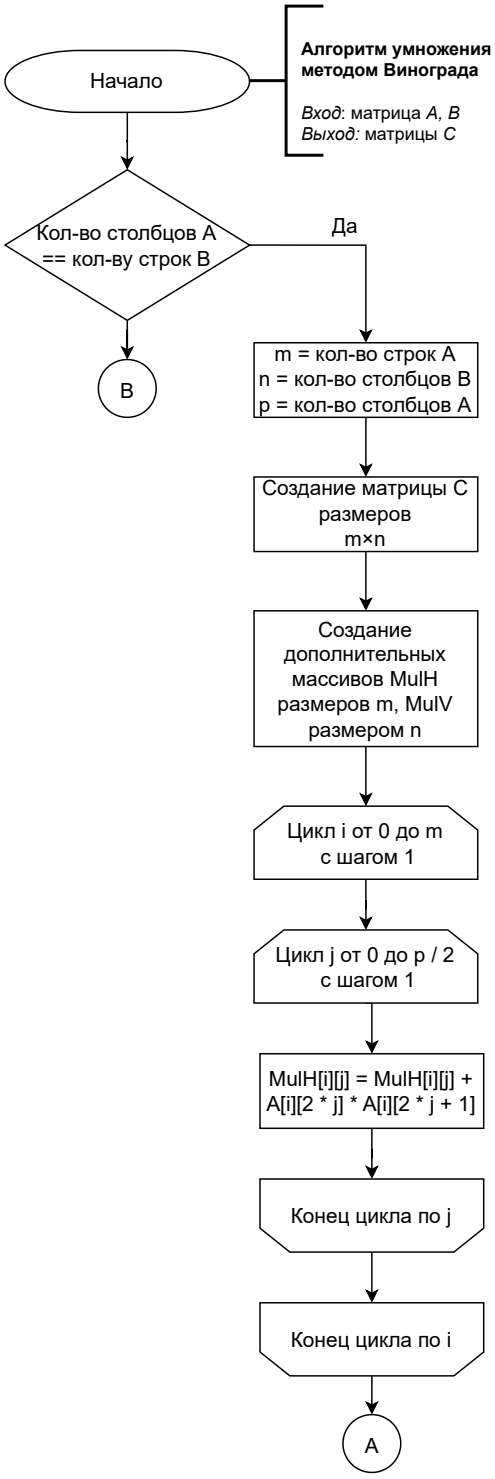

Рисунок 2.2 – Схема алгоритма умножения матриц методом Винограда (начало)

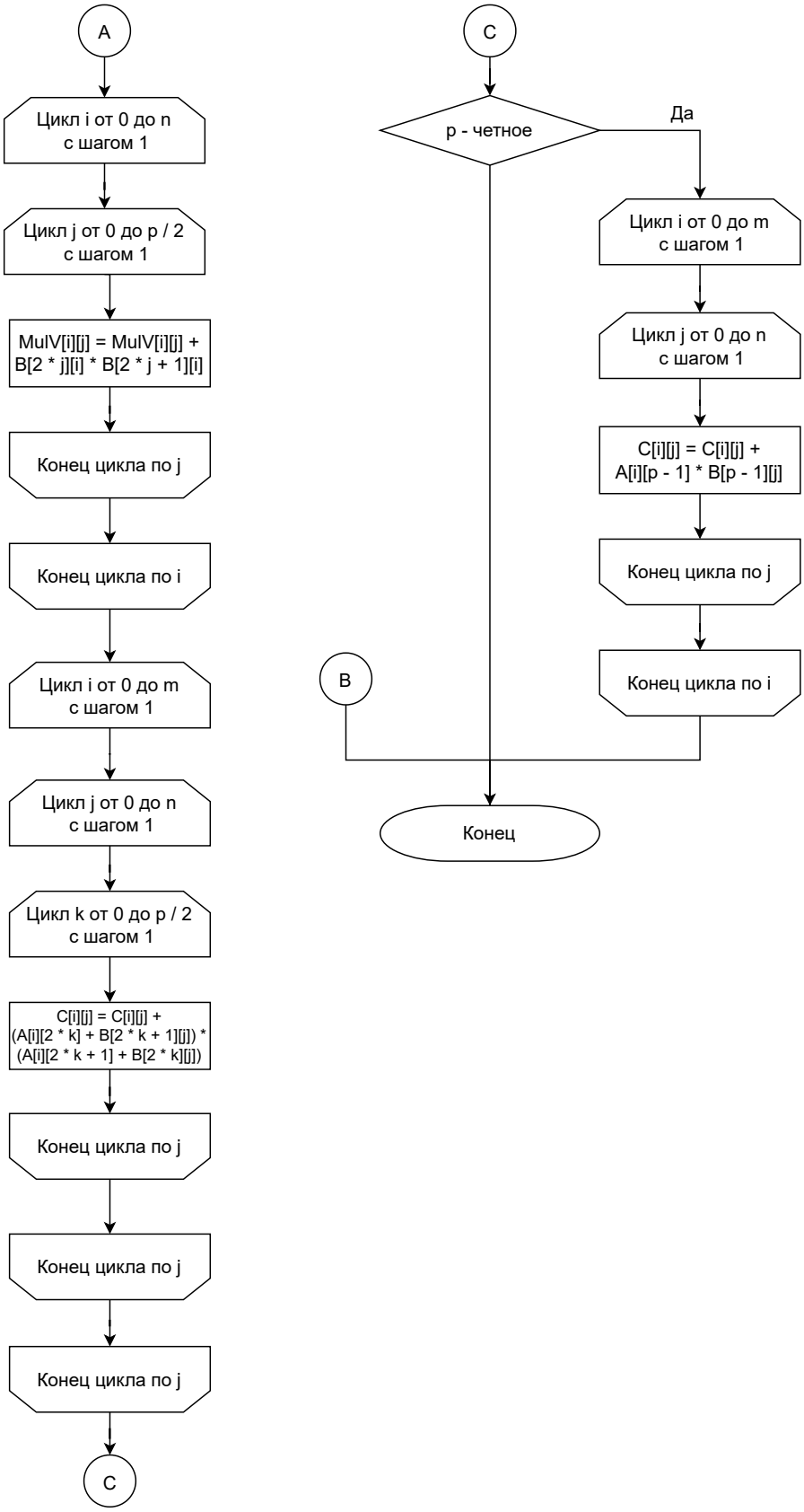

Рисунок 2.3 – Схема алгоритма умножения матриц методом Винограда (конец)

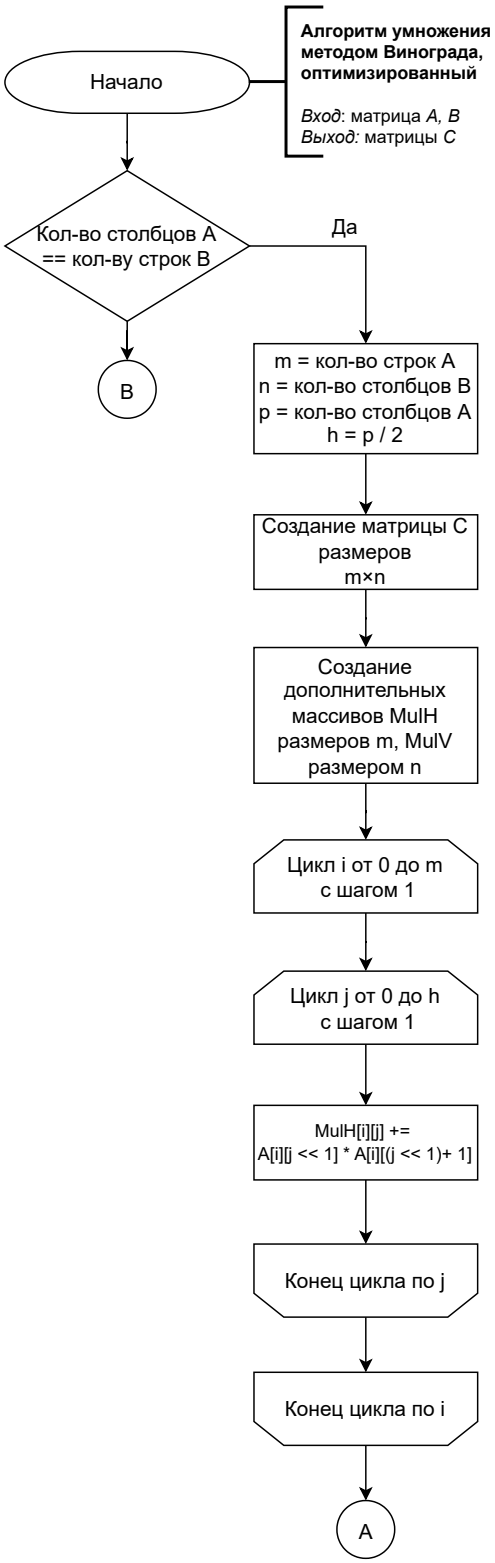

Рисунок 2.4 – Схема оптимизированного алгоритма умножения матриц методом Винограда (начало)

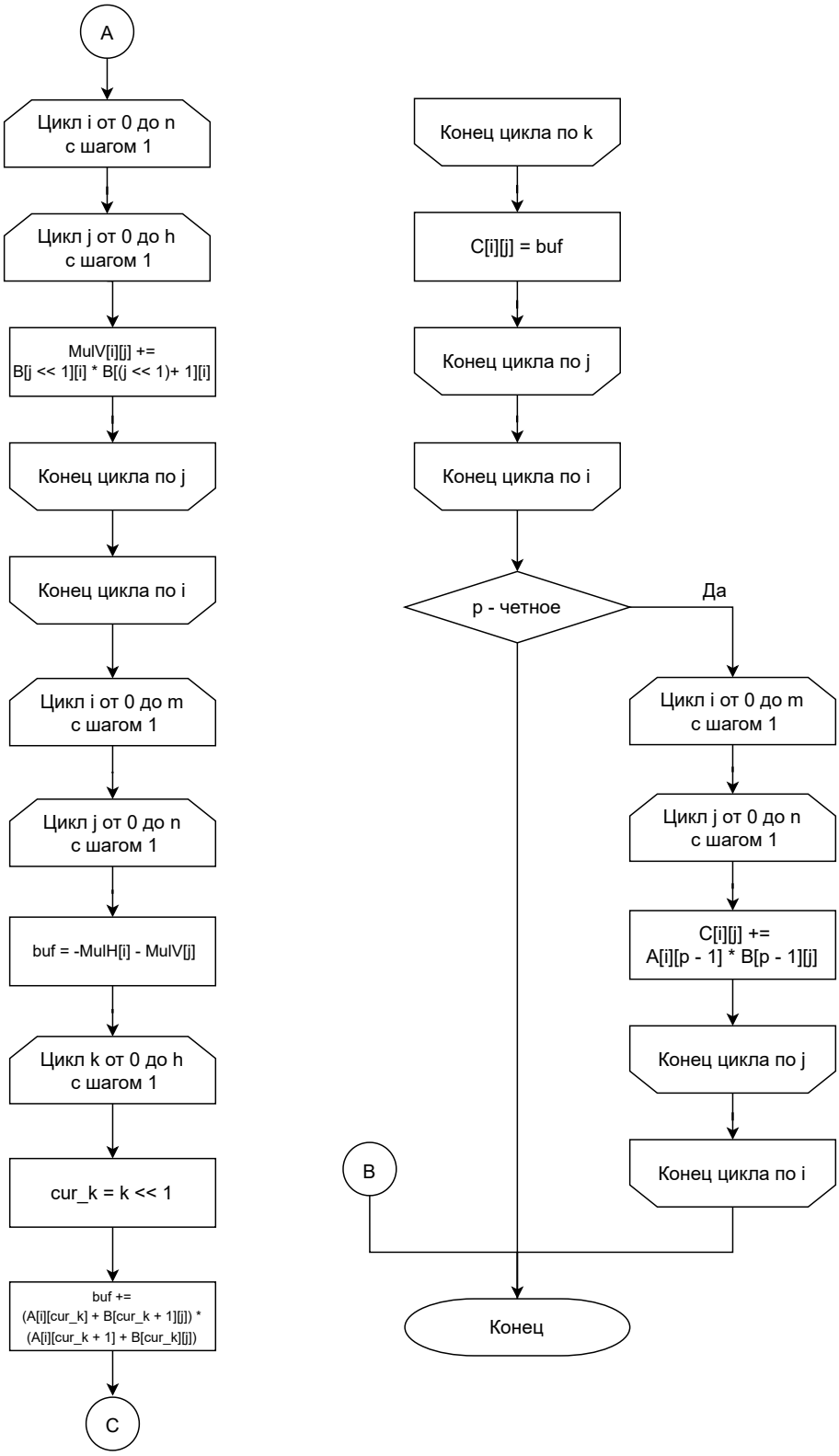

Рисунок 2.5 – Схема оптимизированного алгоритма умножения матриц методом Винограда (конец)

2.2 Модель вычислений для проведения оценки трудоемкости алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1) операции из списка 2.1 имеют трудоемкость **1**;

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

2) операции из списка 2.2 имеют трудоемкость **2**;

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

3) трудоемкость условного оператора `if условие then A else B` рассчитывается как 2.3;

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{в случае выполнеия условия,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4) трудоемкость цикла рассчитывается как 2.4

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

5) трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут приведены рассчеты трудоемкостей алгоритмов для умножения матриц.

Пусть у нас есть 2 матрицы:

1) A размером $M \times P$;

2) B размером $P \times N$.

Стандартный алгоритм

Трудоемкость стандартного алгоритма умножения матриц состоит из:

— внешнего цикла по $i \in [1 \dots M]$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;

— цикла по $j \in [1 \dots N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;

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

Так как трудоемкость стандартного алгоритма равно трудоемкости внешнего цикла — можно вычислить ее, подставив циклы тела:

$$\begin{aligned} f_{standart} &= 2 + M \cdot (2 + 2 + N \cdot (2 + 2 + P \cdot (2 + 8 + 1 + 1 + 2))) = \\ &= 2 + 4M + 4MN + 14MNP \approx 14MNP = O(N^3). \end{aligned} \tag{2.5}$$

Алгоритм Винограда

При вычислении трудоемкости алгоритма Винограда необходимо учесть следующее:

— трудоемкость создания и инициализации массивов $MulH$ и $MulV$:

$$f_{init} = f_{MulH} + f_{MulV}; \tag{2.6}$$

— трудоемкость заполнения массива $MulH$:

$$\begin{aligned} f_{MulH} &= 2 + M \cdot (2 + 4 + \frac{P}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ &= 2 + 6M + \frac{19MP}{2}; \end{aligned} \tag{2.7}$$

— трудоемкость заполнение массива $MulV$:

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

— трудоемкость цикла заполения для четных размеров:

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

— трудоемкость дополнительного цикла, в случае нечетного размера матрицы:

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

В итоге, для худшего случая (т. е. когда размер матрицы нечетный) получаем следующую трудоемкость:

$$f_{worst} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx 16MNP = O(N^3). \tag{2.11}$$

Для лучшего случая (т. е. когда размер матрицы четный):

$$f_{best} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx 16MNP = O(N^3). \tag{2.12}$$

Оптимизированный алгоритм Винограда

Оптимизация алгоритма Винограда осуществляется за счет следующего набора действий:

— замены операции $x = x + k$ на $x+ = k$;

— замены $\cdot 2$ на $<<$;

— предвычисления некоторых слагаемых для алгоритма.

Итоговая трудоемкость оптимизированного алгоритма Винограда состоит из:

— трудоемкости кеширования значения $\frac{P}{2}$ — 3;

— трудоемкость заполнения массива $MulH$:

$$f_{MulH} = 2 + M \cdot (2 + 2 + \frac{P}{2} \cdot (2 + 5 + 1 + 2 + 3)) = \\ = 2 + 2M + \frac{13MP}{2}; \tag{2.13}$$

— трудоемкость заполнение массива $MulV$:

$$f_{MulV} = 2 + N \cdot (2 + 2 + \frac{P}{2} \cdot (2 + 5 + 1 + 2 + 3)) = \\ = 2 + 2N + \frac{13NP}{2}; \tag{2.14}$$

— трудоемкость цикла заполения для четных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (2 + 10 + 2 + \frac{P}{2} \cdot 19)) = \\ = 2 + 4M + 14MN + \frac{19MNP}{2}; \tag{2.15}$$

— трудоемкость дополнительного цикла, в случае нечетного размера матрицы:

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

В итоге, для худшего случая (т. е. когда размер матрицы нечетный) получаем следующую трудоемкость:

$$f_{worst} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx \frac{19MNP}{2} = O(N^3). \tag{2.17}$$

Для лучшего случая (т. е. когда размер матрицы четный):

$$f_{best} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx \frac{19MNP}{2} = O(N^3). \tag{2.18}$$

Алгоритм Штрассена

Если $M(n)$ — количество умножений, выполняемых алгоритмом для умножения двух матриц размером $n \times n$ (где n — степень двойки), то получим следующее рекурретное соотношение для $M(n)$:

$$M(n) = 7M(\frac{n}{2}). \tag{2.19}$$

При $n > 1, M(1) = 1$. Поскольку $n = 2^k$,

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

Подставляя $k = \log_2 n$, получаем:

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

что меньше, чем n_3, необходимое для стандартного алгоритма.

Также необходимо рассмотреть количество сложений $A(n)$, выполняемых алгоритмом. Для умножения двух матриц порядка $n > 1$ алгоритму требуется **7** умножений и **18** сложений матриц размером $\frac{n}{2} \times \frac{n}{2}$. Это сводится к следующему реккурентному уравнению:

$$A(n) = 7 \cdot A \cdot \frac{n}{2} + 18 \cdot (\frac{n}{2})^2. \tag{2.22}$$

При $n > 1, A(1) = 0$.

Итоговую трудоемкость можно рассчитать как:

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

Вывод

В данном разделе на основе теоретических данных, полученных в аналитическом разделе, были построены схемы алгоритмов умножения матриц. Оценены трудоемкости в лучшем и худшем случаях. Так же теорети-

ческий расчет показал, что трудоемкость оптимизированного алгоритма Винограда в 1.6 меньше, чем у неоптимизированного алгоритма Винограда.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинг кода и функциональные тесты.

3.1 Требования к ПО

К программе предъявлен ряд требований:

— входными данными являются две матрицы, каждя из которых хранится в файле, с расширением `*.txt`;

— результат выполнения программы — матрица, полученная в результате умножения введенных матриц;

— возможность произвести замеры процессорного времени выполнения реализованных алгоритмов.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык `C++` [5], так как в нем есть стандартная библиотека `ctime` [5], которая позволяет производить замеры процессорного времени выполнения программы;

В качестве среды разработки был выбран *Visual Studio Code*: он является кроссплатформенным и предоставляет полный набор интсрументов для проектирования и отладки кода.

3.3 Сведения о модулях программы

Данная программа разбита на следующие модули:

— `main.cpp` — файл содержит точку входа в программу, из которой происходит вызов алгоритмов по разработанному интерфейсу;

— `matrix.cpp` — файл содержит реализацию класса `MatrixT`;

— `multiply.cpp` — файл содержит функции умножения матриц;

— `measure.cpp` — файл содержит функции, замеряющие процессорное время выполнения алгоритмов умножения матриц;

3.4 Реализация алгоритмов

В листинге 3.1 приведена реализация алгоритма стандартного умножения матриц.

В листинге 3.2 приведена реализация алгоритма умножения матриц методом Винограда.

В листинге 3.3 приведена реализация оптимизированного алгоритма умножения матриц методом Винограда.

В листинге 3.4 приведена реализация алгоритма умножения матриц методом Штрассена.

Листинг 3.1 – Функция стандартного умножения матриц

```
1  MatrixT Standard::multiply(MatrixT& m1, MatrixT& m2) {
2
3      int rows = m1.getRows(),
4          columns = m2.getColumns(),
5          tmp = m1.getColumns();
6
7      MatrixT res{rows, columns};
8
9      for (int i = 0; i < rows; ++i) {
10
11         for (int j = 0; j < columns; ++j) {
12
13             for (int k = 0; k < columns; ++k) {
14
15                 res(i, j) = res(i, j) + m1(i, k) * m2(k, j);
16             }
17         }
18     }
19
20     return res;
21 }
```

Листинг 3.2 – Функция умножения матриц методом Винограда

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

```
38                  res(i, j) = res(i, j) + m1(i, rows - 1) *
                        m2(rows - 1, j);
39              }
40          }
41
42          return res;
43      }
```

Листинг 3.3 – Оптимизированная функция умножения матриц методом Винограда

```
1  MatrixT VinogradOpt::multiply(MatrixT& m1, MatrixT& m2) {
2
3      int rows = m1.getRows();
4
5      MatrixT res{rows, rows};
6
7      vector<int> mulH(rows, 0);
8      vector<int> mulV(rows, 0);
9
10     int stepHalf = rows / 2;
11
12     for (int i = 0; i < rows; ++i) {
13
14         for (int j = 0; j < stepHalf; ++j)
15             mulH[i] += m1(i, j << 1) * m1(i, (j << 1) + 1);
16     }
17
18     for (int i = 0; i < rows; ++i) {
19
20         for (int j = 0; j < stepHalf; ++j)
21             mulV[i] += m2(j << 1, i) * m2((j << 1) + 1, i);
22     }
23
24     for (int i = 0; i < rows; ++i) {
25
26         for (int j = 0; j < rows; ++j) {
27
28             int buf = -mulH[i] - mulV[j];
29
30             for (int k = 0; k < stepHalf; ++k) {
31
32                 int curK = k << 1;
33
34                 buf += (m1(i, curK) + m2(curK + 1, j)) * (m1(i,
                       curK + 1) + m2(curK, j));
35             }
36
37             res(i, j) = buf;
38         }
```

```
39      }
40
41      if (rows % 2) {
42
43          for (int i = 0; i < rows; ++i) {
44
45              for (int j = 0; j < rows; ++j)
46                  res(i, j) += m1(i, rows - 1) * m2(rows - 1, j);
47          }
48      }
49
50      return res;
51  }
```

Листинг 3.4 – Функция умножения матриц методом Штрассена

```
MatrixT Strassen::multiply(MatrixT& m1, MatrixT& m2) {

    int rows = m1.getRows();

    if (rows <= 2)
        return bruteForce(m1, m2);

    int n = rows / 2;

    auto a = m1.copy(0, n, 0, n);
    auto b = m1.copy(0, n, n, rows);
    auto c = m1.copy(n, rows, 0, n);
    auto d = m1.copy(n, rows, n, rows);

    auto e = m2.copy(0, n, 0, n);
    auto f = m2.copy(0, n, n, rows);
    auto g = m2.copy(n, rows, 0, n);
    auto h = m2.copy(n, rows, n, rows);

    auto ad = a + d;
    auto eh = e + h;
    auto p1 = multiply(ad, eh);

    auto ge = g - e;
    auto p2 = multiply(d, ge);

    auto ab = a + b;
    auto p3 = multiply(ab, h);

    auto bd = b - d;
    auto gh = g + h;
    auto p4 = multiply(bd, gh);

    auto fh = f - h;
    auto p5 = multiply(a, fh);

    auto cd = c + d;
    auto p6 = multiply(cd, e);

    auto ac = a - c;
```

```
41      auto ef = e + f;
42      auto p7 = multiply(ac, ef);
43
44      auto c11 = p1 + p2 - p3 + p4;
45      auto c12 = p5 + p3;
46      auto c21 = p6 + p2;
47      auto c22 = p5 + p1 - p6 - p7;
48
49      MatrixT res{rows, rows};
50
51      res.merge(c11, c12, c21, c22);
52
53      return res;
54  }
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функции, реализующей алгоритм стандартного умножения матриц.

В таблице 3.2 приведены тесты для функции, реализующих алгоритмы умножения матриц методом Винограда.

В таблице 3.3 приведены тесты для функции, реализующих алгоритмы умножения матриц методом Штрассена.

Таблица 3.1 – Функциональные тесты для стандартного алгоритма умножения матриц

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$(\ \ )$	*Неверный размер*
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \end{pmatrix}$
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 \\ 4 & 9 \\ 3 & 2 \end{pmatrix}$

Таблица 3.2 – Функциональные тесты для алгоритма умножения матриц методом Винограда

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$(\ \ )$	*Неверный размер*
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	*Неверный размер*

Таблица 3.3 – Функциональные тесты для алгоритма умножения матриц методом Штрассена

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$(\ \ )$	*Неверный размер*
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 5 & 7 \\ 4 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 7 \\ 4 & 8 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 14 & 16 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 14 & 16 \end{pmatrix}$

Вывод

Были представлены листинги функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда и алгоритм Штрассена. Также в данном разделе была представлена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: AMD Ryzen 5 5500U – 2.10 ГГц;

— Оперативная память: 16 ГБайт;

— Операционная система: Windows 10 Pro 64-разрядная система версии 22H2.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация работы разработанного ПО.

```
                Меню
1. Стандартный метод;
2. Алгоритм Винограда;
3. Оптимизированный п. 2;
4. Алгоритм Штрассена;
5. Замерить время;
0. Выход.

Выберете пункт (0-5): 4

Первая матрица:
        1 2 3 4
        5 6 7 8
        9 10 11 12
        13 14 15 16

Вторая матрица:
        1 2 3 4
        5 6 7 8
        9 10 11 12
        13 14 15 16

Результирующая матрица:
        90 100 110 120
        202 228 254 280
        314 356 398 440
        426 484 542 600
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Затраты по времени выполнения реализаций алгоритмов

Все замеры проводились на квадратных матрицах. Поскольку замеры по времени имеют некоторую погрешность, замеры производились 100 раз, а затем вычислялось среднее арифметическое значение.

Результаты замеров приведены в таблицах 4.1–4.2.

На рисунках 4.2–4.4 приведены графики зависимостей работы алгоритмов от размеров матриц.

Таблица 4.1 – Результаты замеров времени (четные размеры матрицы)

Размер матрицы	Время, мкс		
	Стандартный	Виноград	(опт.) Виноград
10	20.03	21.08	15.39
20	144.02	137.92	87.42
30	484.81	450.17	269.18
40	1139.56	1045.93	631.92
50	2099.01	1930.60	1113.76
60	3623.58	3325.67	1930.44
70	5760.55	5267.34	3018.87
80	8507.87	7797.31	4448.39
90	12116.53	11080.79	6292.83
100	16577.28	15180.71	8620.49

Таблица 4.2 – Результаты замеров времени (нечетные размеры матрицы)

Размер матрицы	Время, мкс		
	Стандартный	Виноград	(опт.) Виноград
1	0.88	1.33	1.19
11	25.44	27.93	19.20
21	161.92	158.98	99.83
31	552.63	506.58	308.10
41	1184.07	1091.53	646.86
51	2254.23	2079.07	1210.11
61	3826.80	3511.72	2024.81
71	6179.90	5608.31	3225.67
81	9278.50	8483.07	4896.62
91	13029.55	11768.70	6864.01
101	17648.53	16087.79	9208.68

Таблица 4.3 – Результаты замеров времени (размер матрицы — степень двойки)

Размер матрицы	Время, мкс			
	Стандартный	Виноград	(опт.) Виноград	Штрассен
2	1.05	1.50	1.42	1.90
4	2.58	2.65	2.48	29.32
8	10.80	10.40	7.16	187.77
16	73.07	62.58	45.89	1404.03
32	665.42	523.15	360.38	9975.97
64	4324.61	3395.31	2276.56	70209.42
128	34420.27	26537.17	17532.16	495987.57

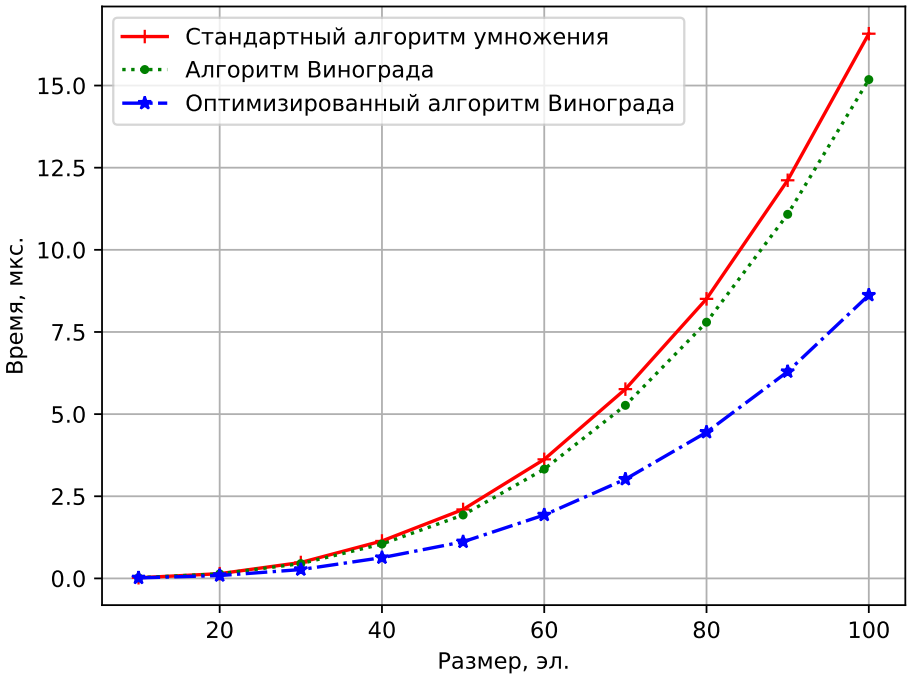

Рисунок 4.2 – Сравнение по времени алгоритмов умножения матриц на четных размерах матрицы

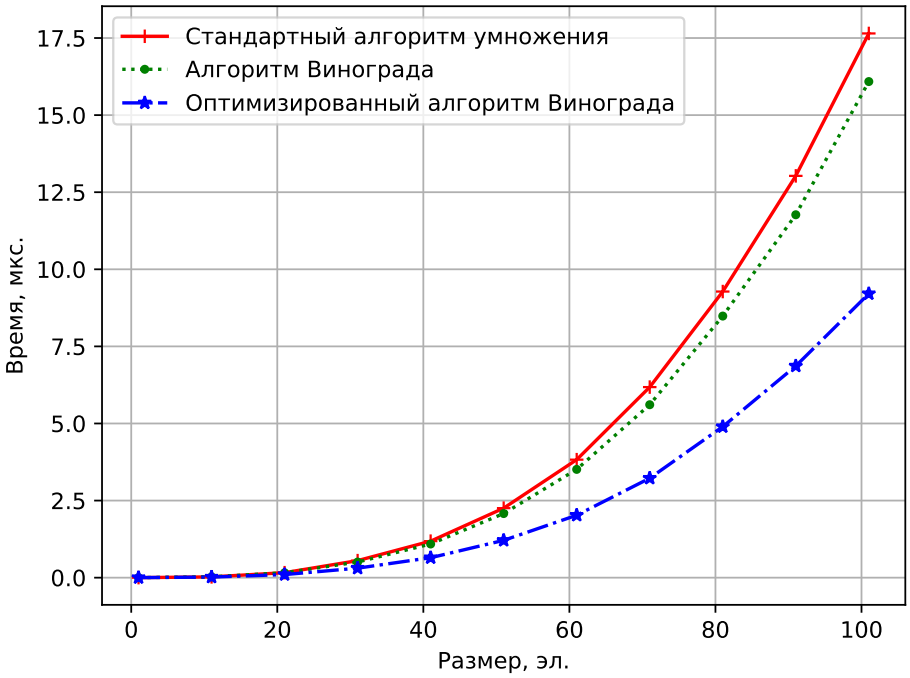

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц на нечетных размерах матрицы

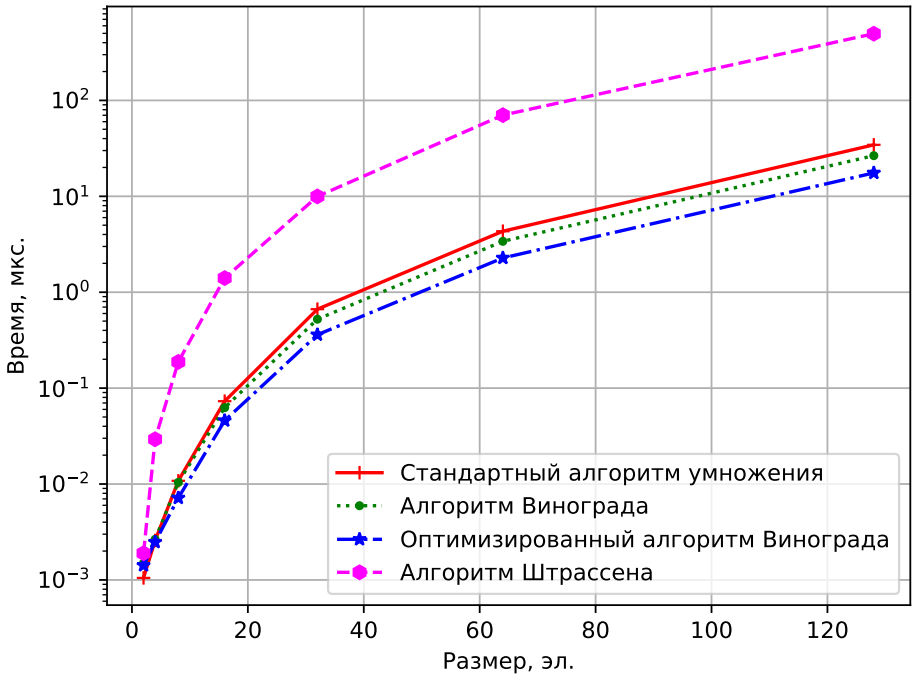

Рисунок 4.4 – Сравнение по времени алгоритмов умножения матриц на размерах, равных степени двойки

4.4 Затраты по памяти реализаций алгоритмов

Введем следующие обозначения:

— $size()$ — функция вычисляющая размер в байтах;

— int — целочисленный тип.

Приведем теоретический расчет затрат по памяти для умножения двух матриц A и B размером $n \times n$, элементы которых типа int.

Стандартный алгоритм

Затраты по памяти для реализации стандартного алгоритма приведены в формуле 4.1:

$$\begin{aligned} M_{std} = 3 \cdot size(int) + 3 \cdot size(int) + n \cdot n \cdot size(int) = \\ = (n^2 + 6) \cdot size(int), \end{aligned} \tag{4.1}$$

где, $3 \cdot size(int)$ — переменные, хранящие размеры матриц;
$3 \cdot size(int)$ — переменные, используемые в циклах;
$n \cdot n \cdot size(int)$ — результирующая матрица.

Алгоритм Винограда

Затраты по памяти для реализации алгоритма умножения методом Винограда приведены в формуле 4.2:

$$M_{vin} = M_{mulH} + M_{mulV} + M_{mul}, \tag{4.2}$$

где M_{mulH}, M_{mulV} — память, используемая для хранения дополнительных массивов;
M_{mul} — память, используемая при самом умножении матриц;

Соответствующие затраты представлены в формулах 4.3–4.4

$$M_{mulH} = M_{mulV} = n \cdot size(int) + 2 * size(int); \tag{4.3}$$

$$M_{mul} = n \cdot n \cdot size(int) + 3 \cdot size(n) + \begin{cases} 0, & \text{чётная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \tag{4.4}$$

Итоговые затраты по памяти для реализации алгоритма Винограда:

$$M_{mul} = (n^2 + 2 \cdot n + 7) \cdot size(int) + \begin{cases} 0, & \text{чётная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \tag{4.5}$$

Оптимизированнный алгоритм Винограда

Затраты по памяти для оптимизированной реализации алгоритма умножения методом Винограда идентичны формуле 4.2. M_{mulH} и M_{mulV} для данной релизации также совпадают.

Отличия появляются в самом умножении матриц, поскольку в целях оптимизации необходимо было некоторые значения хранить в отдельных переменных.

$$M_{mul} = n \cdot n \cdot size(int) + 4 \cdot size(n) + \\ + n \cdot n \cdot \frac{n}{2} * size(int) + n \cdot n * size(int) \begin{cases} 0, & \text{чётная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \quad (4.6)$$

Итоговые затраты по памяти для оптимизированной реализации алгоритма Винограда:

$$M_{mul} = (\frac{n^3}{2} + 2 \cdot n^2 + 6 \cdot n + 4) \cdot size(int) + \begin{cases} 0, & \text{чётная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \quad (4.7)$$

Алгоритм Штрассена

Рассчитаем затраты по памяти для каждого рекурсивного вызова.

$$M_{mul} = (4 \cdot \frac{n}{2} \cdot \frac{n}{2} + 4 \cdot \frac{n}{2} \cdot \frac{n}{2} + \\ + 21 \cdot \frac{n}{2} \cdot \frac{n}{2} + n \cdot n) \cdot size(int), \quad (4.8)$$

где $4 \cdot \frac{n}{2} \cdot \frac{n}{2}$ — 4 матрицы, на которые мы разбиваем нашу исходную матрицу;
$13 \cdot \frac{n}{2} \cdot \frac{n}{2}$ — временные матрицы, получаемые в ходе вычислений;
$n \cdot n$ — результирующая матрица.

Итоговые затраты по памяти для реализации алгоритма Штрассена:

$$M_{mul} = \frac{33}{4} \cdot n^2 \cdot size(int). \quad (4.9)$$

Вывод

Исходя из данных, полученных в таблицах 4.1–4.3, можно сделать вывод, что лучше всего работает отмизированный алгоритм Винограда. Стандартный же алгоритм умножения матриц работает медленнее по сравнению с двумя реалазиями алгоритма Винограда. Модификации, используемые в оптимизированной реализации алгоритма также повлияли на его скорость работы. Также алгоритм Винограда при четном размере матрицы работает быстрее — это обусловлено проведением дополнительных вычислений для крайних строк и столбцов при нечетном размере. Таким образом, алгоритм Винограда стоит использовать при работе именно с матрицами, размер которых — четный.

Также на рис. 4.4 видно, что реализация алгоритма Штрассена выполняется намного дольше реализаций остальных алгоритмов, так как он требует дополнительных операций сложения и вычитаний матриц.

Из теоретических расчетов для потребляемой памяти, можно сделать вывод о том, что стандартное умножение требует наименьшее количество памяти. Наибольшие же затраты требует реализация алгоритма Штрассена, так как при каждом рекурсивном вызове необходимо разбивать исходные матрицы на 4 подматрицы, а также производить дополнительные операции сложения и умножения. Оптимизированная реализация алгоритма также требует больше памяти, по сравнению с неоптимизированной, так как используются дополнительные переменные для хранения некоторых предварительных вычислений.

Заключение

В ходе исследования было определено, что оптимизированный алгоритм Винограда продемонстрировал лучшую производительность. В то же время стандартный метод умножения матриц работает медленнее по сравнению с обеими реализациями алгоритма Винограда. Внесенные модификации в оптимизированную реализацию также оказали влияние на ее скорость. Кроме того, при использовании алгоритма Винограда с четным размером матриц производительность выше, поскольку нет необходимости в дополнительных вычислениях для крайних строк и столбцов в случае нечетного размера. В результате рекомендуется использовать алгоритм Винограда при работе с матрицами четного размера.

Кроме того, реализация алгоритма Штрассена требует больше времени на выполнение из-за дополнительных операций сложения и вычитания матриц.

С учетом теоретических расчетов потребляемой памяти можно сделать вывод о том, что стандартное умножение требует наименьшее количество памяти. Однако реализация алгоритма Штрассена требует наибольших затрат памяти из-за необходимости разбивать исходные матрицы на 4 подматрицы при каждом рекурсивном вызове, а также производить дополнительные операции сложения и умножения. Оптимизированная реализация алгоритма Винограда также требует больше памяти по сравнению с неоптимизированной из-за использования дополнительных переменных для хранения некоторых предварительных вычислений.

Цель данной лабораторной работы была достигнута, а именно были исследованы алгоритмы умножения матриц.

В результате выполнения лабораторной работы для достижения этой цели были выполнены следующие задачи:

1) описаны следующие алгоритмы умножения матриц:

   — стандартный алгоритм умножения;

   — алгоритм Винограда;

   — оптимизированный алгоритм Винограда;

   — алгоритм Штрассена.

2) релизованы описанные алгоритмы;

3) дана оценка трудоемкости алгоритмов;

4) дана оценка потребляемой памяти реализациями алгоритмов;

5) проведены замеры времени выполнения алгоритмов;

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.

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

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

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://www.elibrary.ru/item.asp?id=23140890` (дата обращения 08.11.2023).

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).