Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2 по курсу «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Фам Минь Хиеу

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л.Л., Строганов Д.В.

Москва — 2023 г.

Содержание

Введение

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

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

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

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

1) Описать алгоритмы стандартного умножения матриц и алгоритм Винограда;

2) Построить схемы следующих алгоритмов

   — классический алгоритм умножения матриц;

   — алгоритм Винограда;

   — оптимизированный алгоритм Винограда;

3) Создать программное обеспечение, реализующее перечисленные алгоритмы;

4) Провести сравнительный анализ по времени и по памяти реализованных алгоритмов;

5) Подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

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

1.1 Матрица

Матрицей [1] называют таблицу чисел a_{ik} вида

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

состоящую из m строк и n столбцов. Числа a_{ik} называются её *элементами*.

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

Можно выделить следующие операции над матрицами:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц в случае, если количество столбцов первой матрицы равно количеству строк второй матрицы. В итоговой матрице количество строк будет, как у первой матрицы, а столбцов – как у второй.

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

1.2 Стандартный алгоритм матричного умножения

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

$$A_{NP} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{PM} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.2}$$

Тогда матрица C размерностью N*M:

$$C_{NM} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.3}$$

в которой:

$$c_{ij} = \sum_{r=1}^{P} a_{ir} b_{rj} \quad (i = \overline{1, N}; j = \overline{1, M}) \tag{1.4}$$

называется их произведением.

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

1.3 Алгоритм Винограда умножения матриц

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение

соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$.

Их скалярное произведение равно: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.5):

$$V \cdot W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4. \quad (1.5)$$

Выражение в правой части формулы 1.5 допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. При нечетном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов к результату [2].

1.4 Оптимизация алгоритма Винограда умножения матриц

Оптимизированная версия алгоритма Винограда [2] отличается:

1) заменить умножение на 2 на побитовый сдвиг;

2) заменить операцию x = x + k; на x += k;

3) предвычислять некоторые слагаемые для алгоритма.

1.5 Вывод

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

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов умножения матриц: стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда. Будут описаны типы данных, используемые для реализации.

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц. Схема умножения матриц по алгоритму Винограда приведена на рисунках 2.2 – 2.3, схема оптимизированной версии приведена на рисунках 2.4 – 2.5.

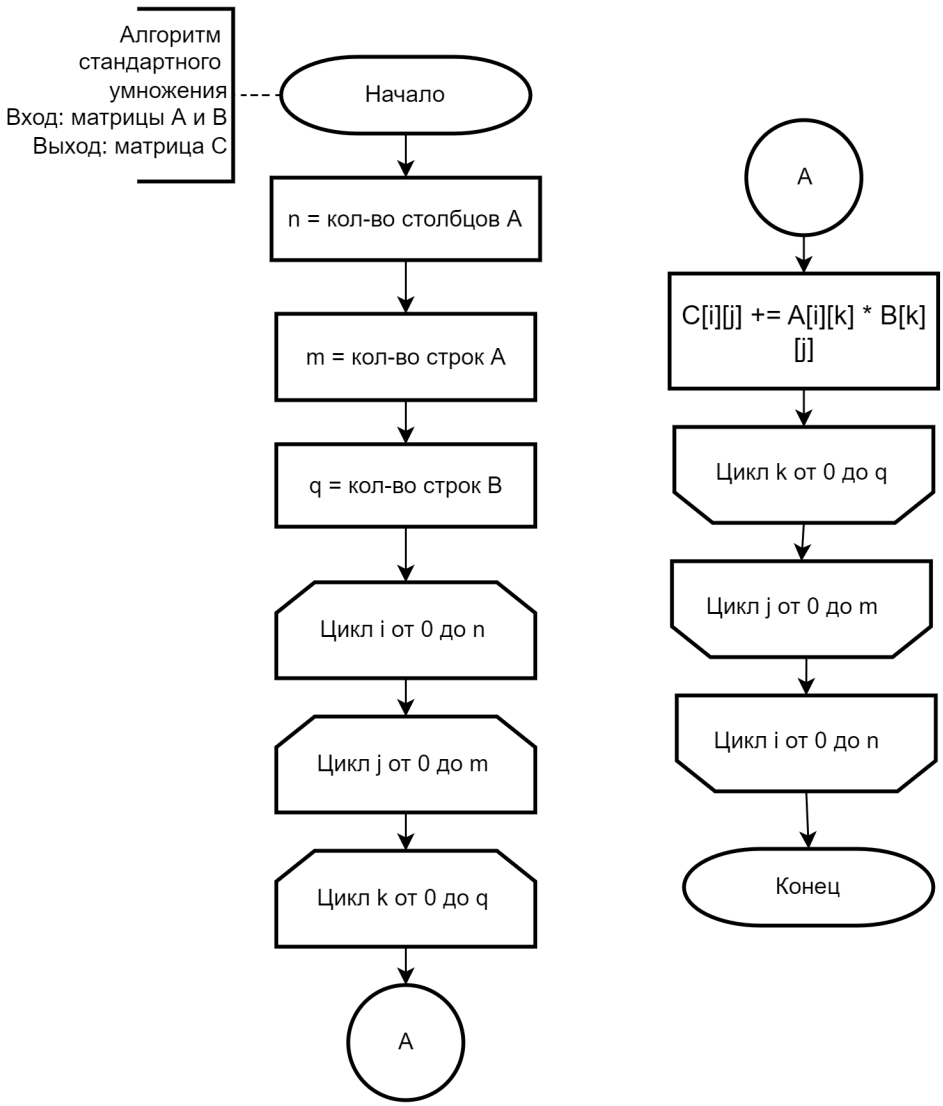

Рисунок 2.1 – Схема стандатного алгоритма умножения матриц

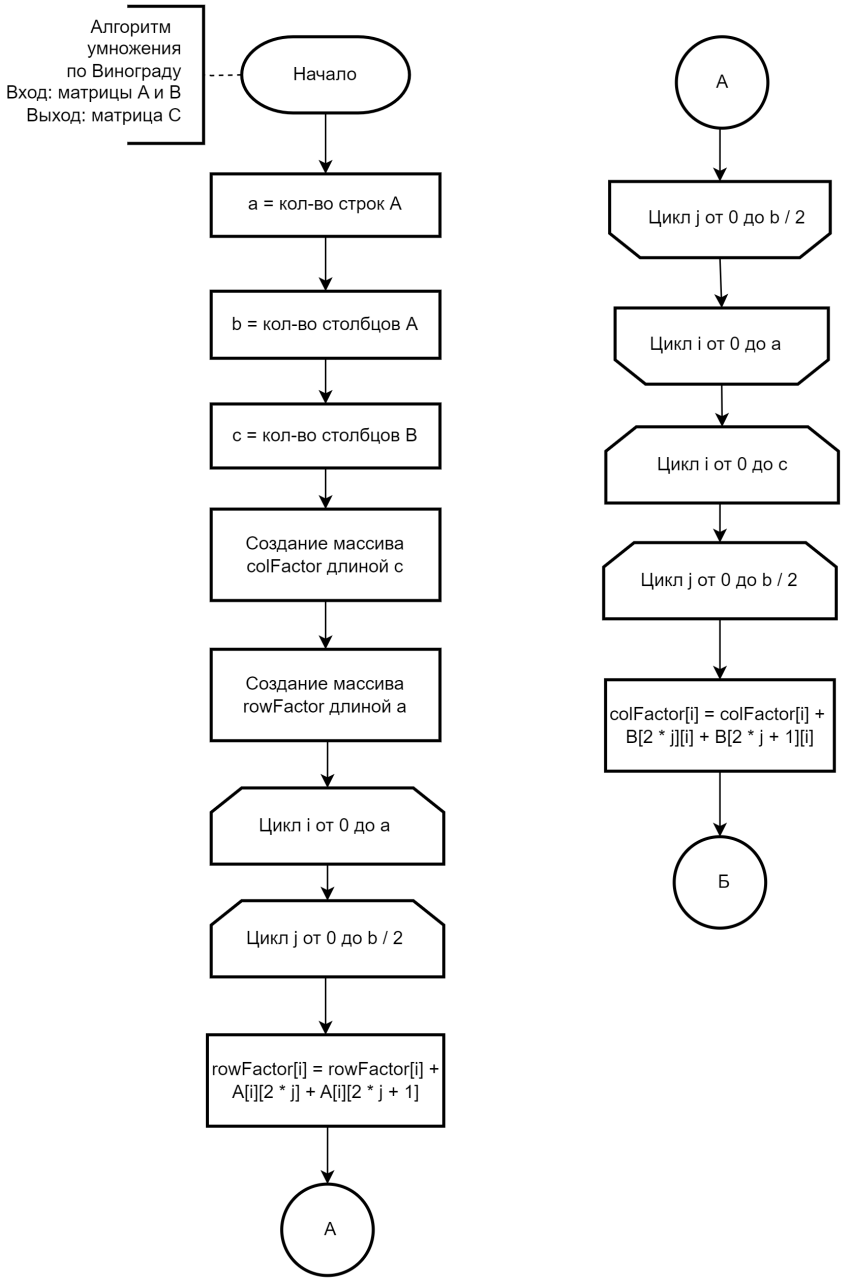

Рисунок 2.2 – Схема алгоритма Винограда (Часть 1)

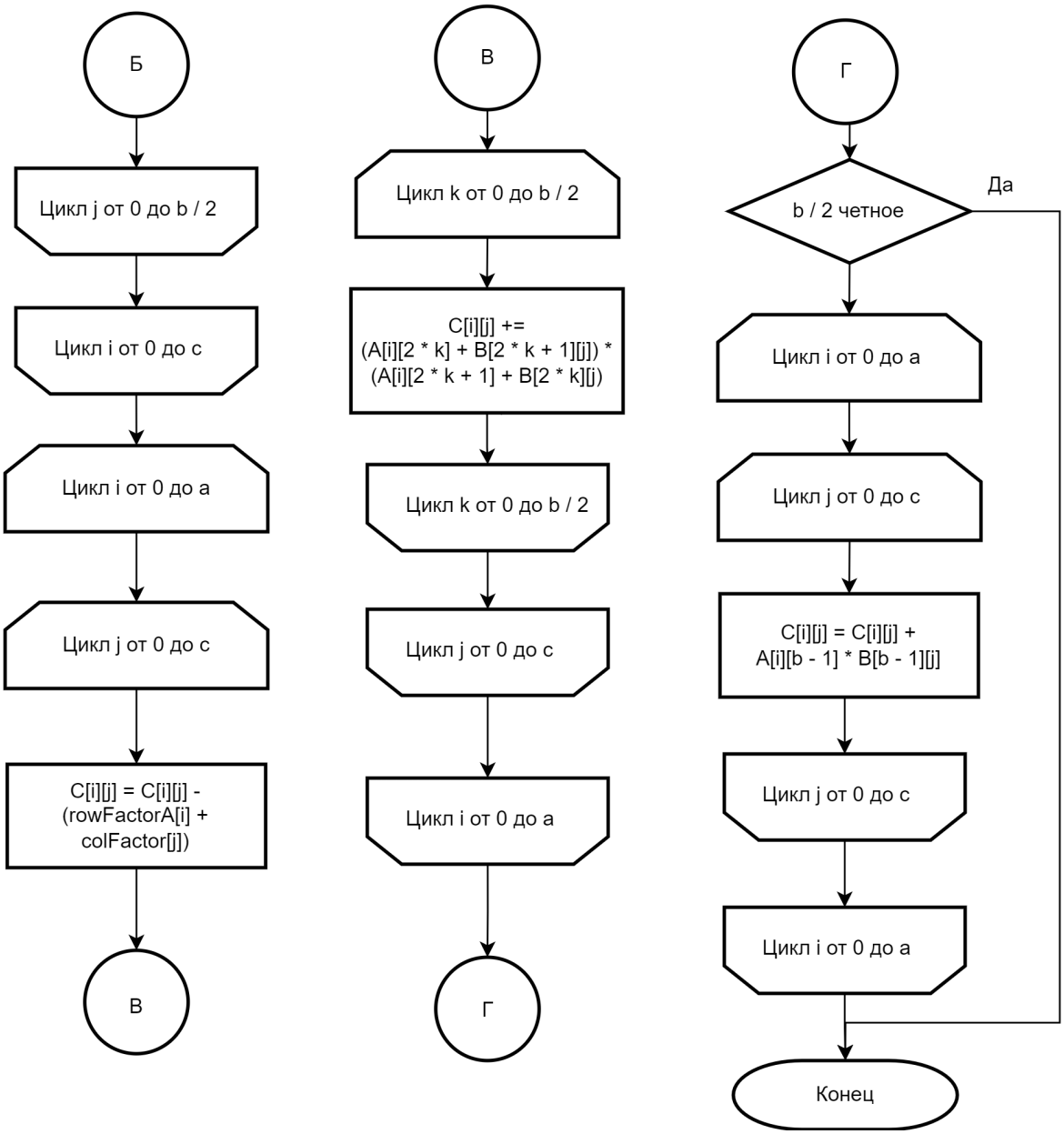

Рисунок 2.3 – Схема алгоритма Винограда (Часть 2)

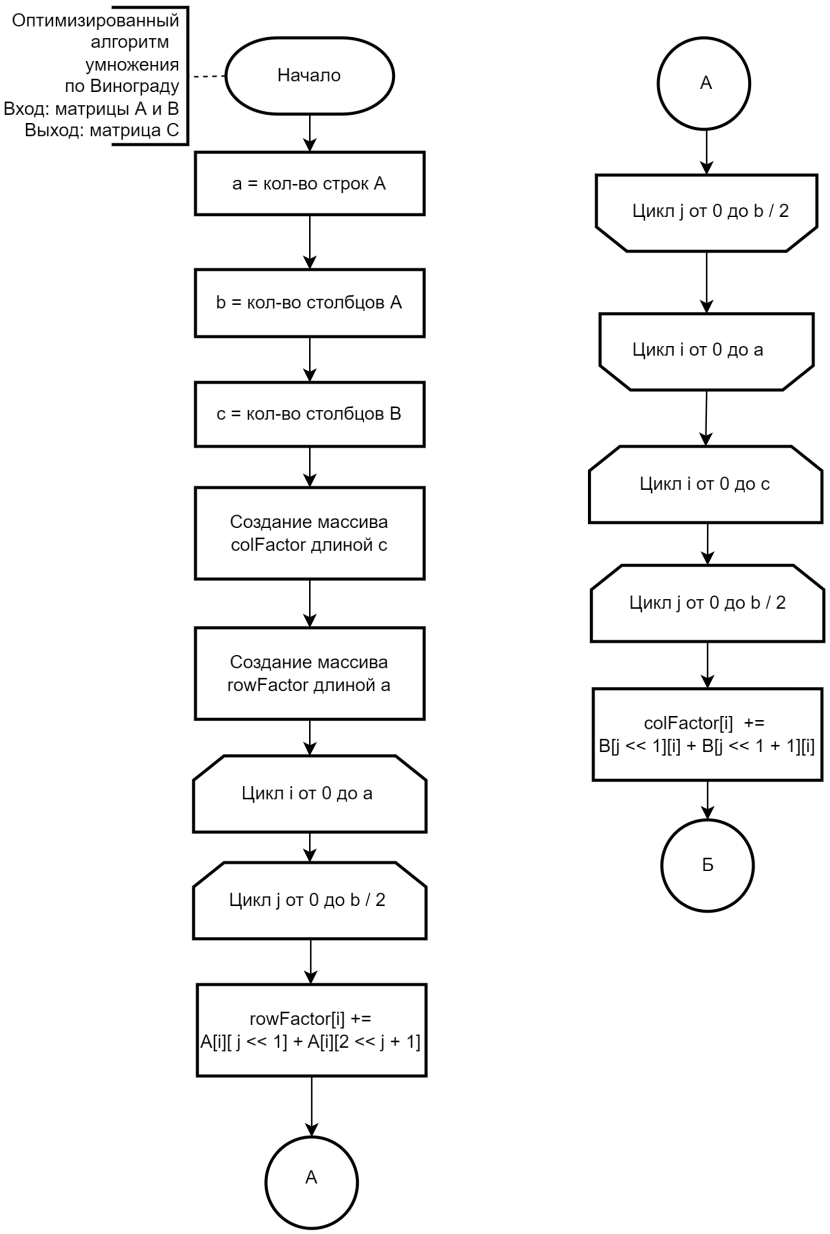

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда (Часть 1)

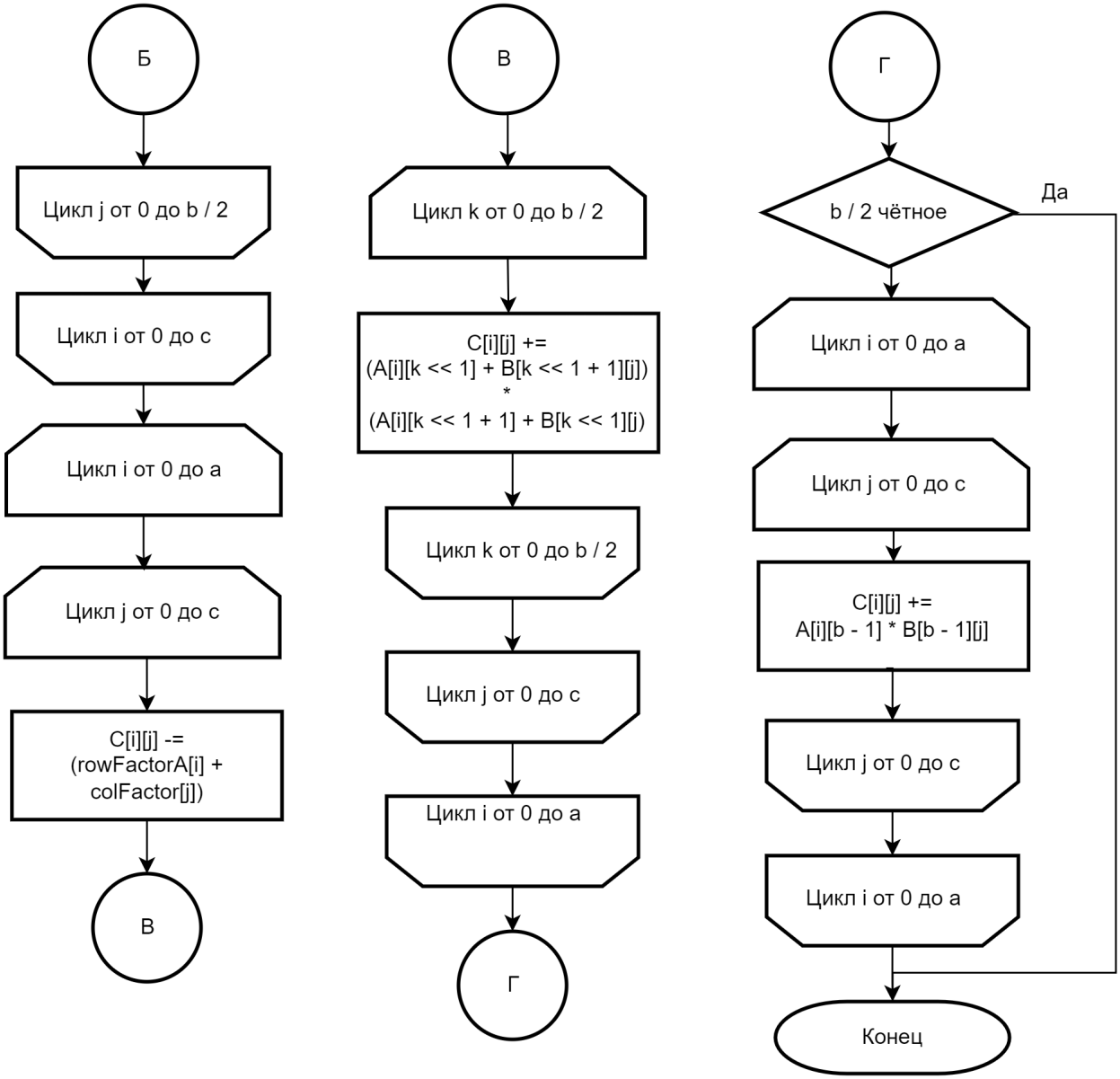

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда (Часть 2)

2.2 Модель вычислений для проведения оценки трудоемкости

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

1) Трудоемкость базовых операций:

   — равна 1 для:

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

   — равна 2 для:

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

2) Трудоемкость 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) Трудоемкость 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) Трудоемкость передачи параметра в функции и возврат из функции равны 0.

2.3 Трудоемкость алгоритмов

2.3.1 Классический алгоритм

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из:

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

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

— цикла по $k \in [1 \dots M]$, трудоёмкость которого: $f = 2 + 2 + 14M$;

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

$$f_{standart} = 2 + N \cdot (2 + 2 + T \cdot (2 + 2 + M \cdot (2 + 8 + 1 + 1 + 2))) = \\ = 2 + 4N + 4NT + 14NMT \approx 14NMT = O(N^3) \tag{2.5}$$

2.3.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, нужно учесть следующее:

— создания и инициализации массивов a_{tmp} и b_{tmp}, трудоёмкость которых указана в формуле (2.6);

$$f_{init} = N + M \tag{2.6}$$

— заполнения массива a_{tmp}, трудоёмкость которого указана в формуле (2.7);

$$\begin{aligned} f_{atmp} &= 2 + N \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ &= 2 + 4N + \frac{19NM}{2} = 2 + 4N + 9,5NM \end{aligned} \tag{2.7}$$

— заполнения массива b_{tmp}, трудоёмкость которого указана в формуле (2.8);

$$\begin{aligned} f_{btmp} &= 2 + T \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ &= 2 + 4T + \frac{19TM}{2} = 2 + 4T + 9,5TM \end{aligned} \tag{2.8}$$

— цикла заполнения для чётных размеров, трудоёмкость которого указана в формуле (2.9);

$$f_{cycle} = 2 + N \cdot (4 + T \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 28))) = \\ = 2 + 4N + 13NT + \frac{32NTM}{2} = 2 + 4N + 13NT + 16NTM \tag{2.9}$$

— цикла, который дополнительно нужен для подсчёта значений при нечётном размере матрицы, трудоемкость которого указана в формуле (2.10);

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

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

$$f_{worst} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3) \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f_{best} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3) \tag{2.12}$$

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

— кэширования значения $\frac{M}{2}$ в циклах, которое равно 3;

— создания и инициализации массивов $mulH$ и $mulV$ (2.6);

— заполнения массива $mulH$, трудоёмкость которого (2.7);

— заполнения массива $mulV$, трудоёмкость которого (2.8);

— цикла заполнения для чётных размеров, трудоёмкость которого указана в формуле (2.13);

$$f_{cycle} = 2 + N \cdot (4 + T \cdot (4 + 7 + \frac{M}{2} \cdot (2 + 10 + 5 + 2 + 4))) =$$
$$= 2 + 4N + 11NT + \frac{23NTM}{2} = 2 + 4N + 11NT + 11,5 \cdot NTM \tag{2.13}$$

— условие, которое нужно для дополнительных вычислений при нечётном размере матрицы, трудоемкость которого указана в формуле (2.14);

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + T \cdot (2 + 10)), & \text{иначе} \end{cases} \tag{2.14}$$

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

$$f_{worst} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3) \tag{2.15}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f_{best} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3) \tag{2.16}$$

2.4 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

— количество строк — целое число;

— количество столбцов — целое число;

— матрица — двумерный список целых чисел

2.5 Вывод

Были представлены схемы алгоритмов умножения матриц. Были указаны типы данных, используемые для реализации. Проведённая теоретическая оценка трудоемкости алгоритмов показала, что в трудоёмкость выполнения алгоритма Винограда в случае оптимизации в 1.2 раза меньше, чем у простого алгоритма Винограда.

3 Технологическая часть

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

3.1 Требования к программному обеспечению

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

— элементы матрицы - целые числа;

— количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы;

— должно быть выдано сообщение об ошибке при вводе пустых матриц.

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

3.2 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования C++. Данный выбор обусловлен наличием у языка функции *clock*() [3] измерения процессорного времени.

Визуализация графиков с помощью библиотеки *Matplotlib* [4].

3.3 Сведения о модулях программы

Программа состоит из следующих модулей:

— main.cpp - точка входа программы;

— matrix.h - заголовочный файл, содержащий объявлении функций работы с матрицами;

— matrix.cpp - файл, содержащий реализации функций работы с матрицами;

— measure.h - заголовочный файл, содержащий объявлении функций замеров времени работы и затравоченной памяти рассматриваемых алгоритмов;

— measure.cpp - файл, содержащий реализации функций замеров времени работы и затравоченной памяти рассматриваемых алгоритмов;

3.4 Реализация алгоритмов

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

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
1  Matrix Matrix::standartMul(Matrix& m)
2  {
3      size_t a = arr.size();
4      size_t b = arr[0].size();
5      size_t c = m.arr[0].size();
6
7      Matrix res(a, c);
8
9      for (size_t i = 0; i < a; i++)
10         for (size_t j = 0; j < c; j++)
11             for (size_t k = 0; k < b; k++)
12                 res.arr[i][j] = res.arr[i][j] + arr[i][k] *
                       m.arr[k][j];
13     return res;
```

```
14 }
```

Листинг 3.2 – Алгоритм Винограда умножения матриц

```
1  Matrix Matrix::vinogradMul(Matrix& m)
2  {
3      size_t a = arr.size();
4      size_t b = arr[0].size();
5      size_t c = m.arr[0].size();
6
7      Matrix res(a, c);
8      vector<int> mulH(a, 0);
9      vector<int> mulV(c, 0);
10
11     size_t d = b / 2;
12     for (size_t i = 0; i < a; i++)
13         for (size_t j = 0; j < d; j++)
14             mulH[i] = mulH[i] + arr[i][2 * j] * arr[i][2 * j + 1];
15
16     for (size_t i = 0; i < c; i++)
17         for (size_t j = 0; j < d; j++)
18             mulV[i] = mulV[i] + m.arr[2 * j][i] * m.arr[2 * j + 1][i];
19
20     for (size_t i = 0; i < a; i++)
21         for (size_t j = 0; j < c; j++)
22         {
23             res.arr[i][j] = res.arr[i][j] - mulH[i] - mulV[j];
24             for (size_t k = 0; k < d; k++)
25                 res.arr[i][j] = res.arr[i][j] + (arr[i][2 * k] +
                       m.arr[2 * k + 1][j]) * \
26                 (arr[i][2 * k + 1] + m.arr[2 * k][j]);
27         }
28
29     if (b % 2)
30     {
31         for (size_t i = 0; i < a; i++)
32             for (size_t j = 0; j < c; j++)
33                 res.arr[i][j] = res.arr[i][j] + arr[i][b - 1] *
                       m.arr[b - 1][j];
34     }
35
36     return res;
37 }
```

Листинг 3.3 – Оптимизированный алгоритм Винограда умножения матриц

```
1  Matrix Matrix::optVinogradMul(Matrix& m)
2  {
```

```
3       size_t a = arr.size();
4       size_t b = arr[0].size();
5       size_t c = m.arr[0].size();
6
7       Matrix res(a, c);
8       vector<int> mulH(a, 0);
9       vector<int> mulV(c, 0);
10
11      size_t d = b / 2;
12      int tmp;
13      for (size_t i = 0; i < a; i++)
14          for (size_t j = 0; j < d; j++)
15              mulH[i] += arr[i][j << 1] * arr[i][(j << 1) + 1];
16
17      for (size_t i = 0; i < c; i++)
18          for (size_t j = 0; j < d; j++)
19              mulV[i] += m.arr[j << 1][i] * m.arr[(j << 1) + 1][i];
20
21      for (size_t i = 0; i < a; i++)
22      {
23          tmp = mulH[i];
24          for (size_t j = 0; j < c; j++)
25          {
26              res.arr[i][j] -= (tmp + mulV[j]);
27              for (size_t k = 0; k < d; k++)
28                  res.arr[i][j] += (arr[i][k << 1] + m.arr[(k << 1) +
                        1][j]) * \
29                  (arr[i][(k << 1) + 1] + m.arr[k << 1][j]);
30          }
31      }
32
33      if (b % 2)
34      {
35          for (size_t i = 0; i < a; i++)
36          {
37              tmp = arr[i][b - 1];
38              for (size_t j = 0; j < c; j++)
39                  res.arr[i][j] += tmp * m.arr[b - 1][j];
40          }
41      }
42
43      return res;
44  }
```

3.5 Функциональные тесты

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

Таблица 3.1 – Функциональные тесты

Матрица A	Матрица B	Ожидаемый результат
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 4 & 1 & 2 \\ -1 & 5 & -2 \\ -7 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 4 & 1 \\ 1 & 1 & 3 \\ 0 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 5 & 17 & 11 \\ 4 & 1 & 10 \\ -7 & -28 & -5 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 10 \end{pmatrix}$

3.6 Вывод

Были представлены листинги реализаций всех алгоритмов умножения матриц – стандартного, Винограда и оптимизированного алгоритма Винограда. Также в данном разделе была приведены функциональные тесты.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будет проведен сравнительный анализ реализованных алгоритмов умножения матриц по затраченному процессорному времени.

4.1 Технические характеристики

Тестирование проводилось на устройстве со следующими техническими характеристиками:

— Операционная система Window 10 Home Single Language;

— Память 8 Гб;

— Процессор 11th Gen Intel(R) Core(TM) i7-1165G7 2.80 ГГц, 4 ядра.

4.2 Демонстрация работы программы

На рисунках 4.1, 4.2 приведены демонстрации работы программы.

```
C:\Users\abcd2\OneDrive\Desktop\5-SEM-HIEU\AA\xf21iu26\src\x64\Debug\main.exe
Меню:
                    0) Выйти
                    1) Посчитать стандартным алгоритмом
                    2) Посчитать алгоритмом Винограда
                    3) Посчитать оптимизированным алгоритмом Винограда
                    4) Сравнить алгоритмы по времени
                    5) Сравнить алгоритмы по памяти
Введите команду: 1
Введите первую матрицу
Введите количество строк: 2
Введите количество столбцов: 3
4 -1 7
2 0 1
Введите вторую матрицу
Введите количество строк: 3
Введите количество столбцов: 2
1 2
1 0
-1 -1

Результат:
-4 1
1 3
```

Рисунок 4.1 – Демонстрация работы программы при стандартном умножении

```
Меню:
                    0) Выйти
                    1) Посчитать стандартным алгоритмом
                    2) Посчитать алгоритмом Винограда
                    3) Посчитать оптимизированным алгоритмом Винограда
                    4) Сравнить алгоритмы по времени
                    5) Сравнить алгоритмы по памяти
Введите команду: 2
Введите первую матрицу
Введите количество строк: 2
Введите количество столбцов: 3
4 -1 7
2 0 1
Введите вторую матрицу
Введите количество строк: 3
Введите количество столбцов: 2
1 2
1 0
-1 -1

Результат:
-4 1
1 3
```

Рисунок 4.2 – Демонстрация работы программы при умножении Винограда

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблицах 4.1, 4.2.

В таблице 4.1 приведены результаты замеров по времени алгоритмов умножения матриц при четных размеров квадратных матриц от 1 до 100 с шагом 10 на различных входных данных.

Таблица 4.1 – Результаты замеров времени (чётные размеры матриц)

Размер матрицы	**Время, мс**		
	Классический	**Винограда**	**(опт.) Винограда**
1	0.028	0.052	0.056
10	0.424	0.592	0.596
20	1.624	1.952	1.956
30	3.624	4.112	4.116
40	6.424	7.072	7.076
50	10.024	10.832	10.836
60	14.424	15.392	15.396
70	19.624	20.752	20.756
80	25.624	26.912	26.916
90	32.424	33.872	33.876
100	40.024	41.632	41.636

По таблице 4.1 был построен график 4.3. Исходя из этих данных можно понять, что лучшего всего работает оптимизированный алгоритм Винограда, а классический алгоритм работаем в 1,3 раз хуже, чем алгоритм Винограда.

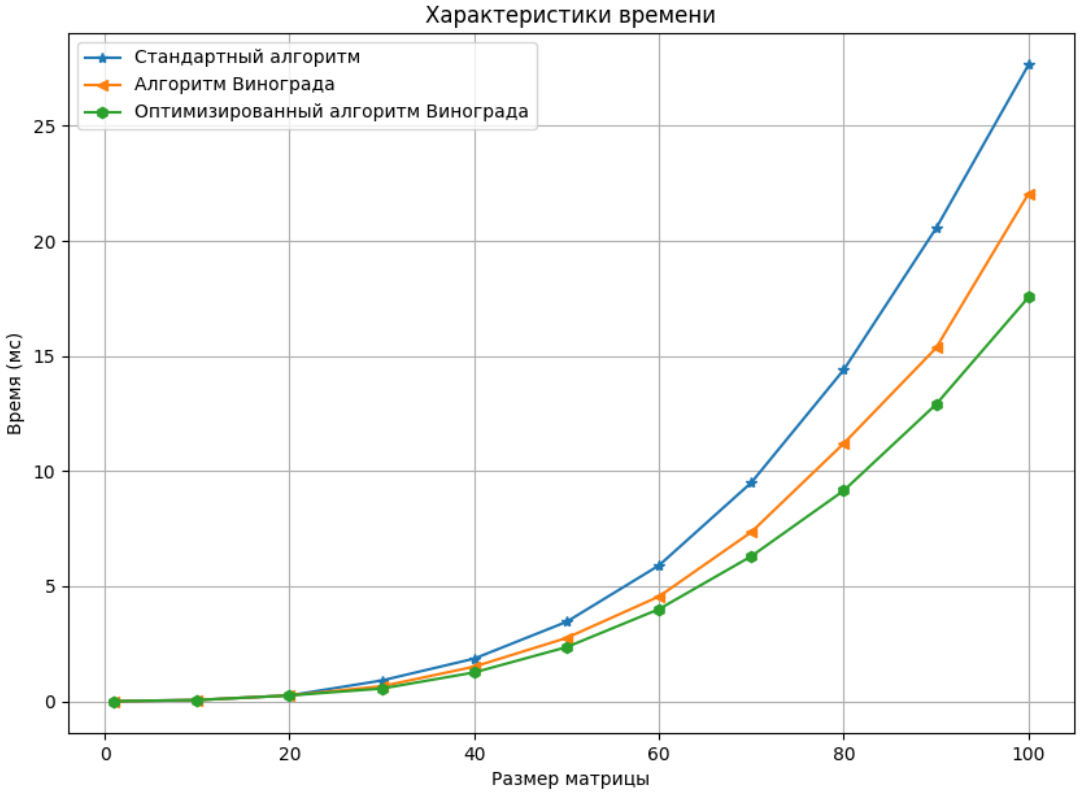

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц на чётных размерах матриц

В таблице 4.2 приведены результаты замеров по времени алгоритмов умножения матриц при нечетных размеров квадратных матриц, размеров от 1 до 101 с шагом 10 на различных данных.

Таблица 4.2 – Результаты замеров времени (нечётные размеры матриц)

Размер матрицы	**Время, мс**		
	Классический	**Винограда**	**(опт.) Винограда**
1	0	0	0
11	0.05	0.1	0.05
21	0.3	0.25	0.25
31	0.85	0.75	0.65
41	2.05	1.65	1.35
51	3.8	3.2	2.55
61	6.4	4.95	4.1
71	10.15	7.9	6.6
81	15	11.95	9.8
91	20.8	16.4	13.45
101	28.85	21.75	18.05

По таблице 4.2 был построен график 4.4. Исходя из этих данных можно понять, что лучшего всего работает оптимизированный алгоритм Винограда, а классический алгоритм работаем в 1,2 раз хуже, чем алгоритм Винограда.

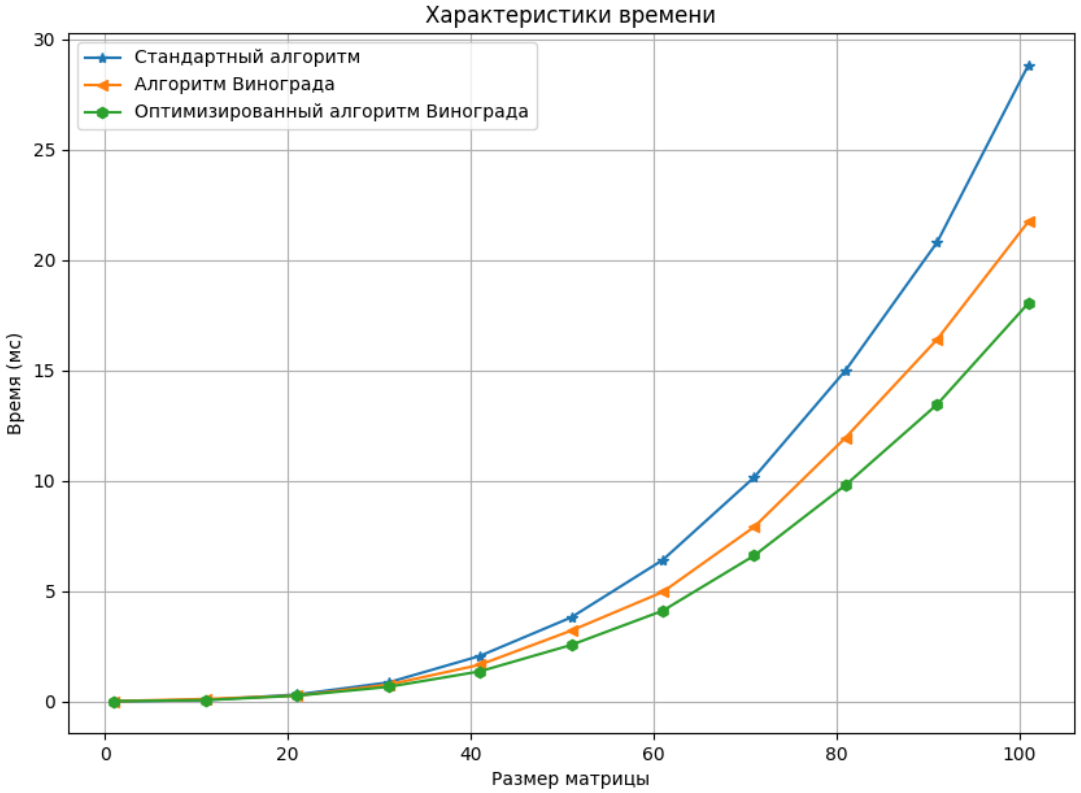

Рисунок 4.4 – Сравнение по времени алгоритмов умножения матриц на нечётных размерах матриц

4.4 Характеристики по памяти

Пусть M - количество строк первой матрицы, N - количество столбцов первой матрицы, Q - количество столбцов второй матрицы, тогда затраты памяти на рассматриваемые алгоритмы будут следующими.

Стандартный алгоритм умножения матриц:

— размер матрицы — 3 · sizeof(size_t);

— матрица — (M · N + N · Q) · sizeof(int)

Таким образом, общая затраченная память в стандартном алгоритме умножения матриц равна $4 \cdot M \cdot N + 4 \cdot N \cdot Q + 3 \cdot 8 = 4 \cdot M \cdot N + 4 \cdot N \cdot Q + 24$.

Алгоритм Винограда:

— размер матрицы — 3 · sizeof(size_t);

— вспомогательная переменная — sizeof(size_t);

— вспомогательные матрицы — (M + Q) · sizeof(int);

— матрица — (M · N + N · Q) · sizeof(int).

Таким образом, общая затраченная память в алгоритме Винограда равна $3 \cdot 8 + 1 \cdot 8 + (M+Q) \cdot 4 + 4 \cdot M \cdot N + 4 \cdot N \cdot Q = 4 \cdot M \cdot N + 4 \cdot N \cdot Q + 4 \cdot M + 4 \cdot Q + 32$.

Оптимизированный алгоритм Винограда:

— размер матрицы — 3 · sizeof(size_t)

— вспомогательная переменная — 1 · sizeof(size_t) + 1 · sizeof(int)

— вспомогательные матрицы — (M + Q) · sizeof(int),

— матрица — (M · N + N · Q) · sizeof(int)

Таким образом, общая затраченная память в алгоритме оптимизированного Винограда равна $3 \cdot 8 + 1 \cdot 8 + 1 \cdot 4 + (M + Q) \cdot 4 + 4 \cdot M \cdot N + 4 \cdot N \cdot Q = 4 \cdot M \cdot N + 4 \cdot N \cdot Q + 4 \cdot M + 4 \cdot Q + 36$.

Результаты расчетов памяти, которых представлены в таблице 4.3, где размеры матриц находятся в диапазоне от 1 до 100 с шагом 10.

Таблица 4.3 – Результаты замеров памяти

Размер матрицы	**Память, байт**		
	Классический	**Винограда**	**(опт.) Винограда**
1	0.028	0.052	0.056
10	0.424	0.592	0.596
20	1.624	1.952	1.956
30	3.624	4.112	4.116
40	6.424	7.072	7.076
50	10.024	10.832	10.836
60	14.424	15.392	15.396
70	19.624	20.752	20.756
80	25.624	26.912	26.916
90	32.424	33.872	33.876
100	40.024	41.632	41.636

По таблице 4.3 был построен график 4.5. Исходя из этих данных можно понять, что стандартный алгоритм потребляет меньше всего памяти, также алгоритм Винограда и оптимизированный алгоритм Виноград используют примерно одинаковое количество памяти.

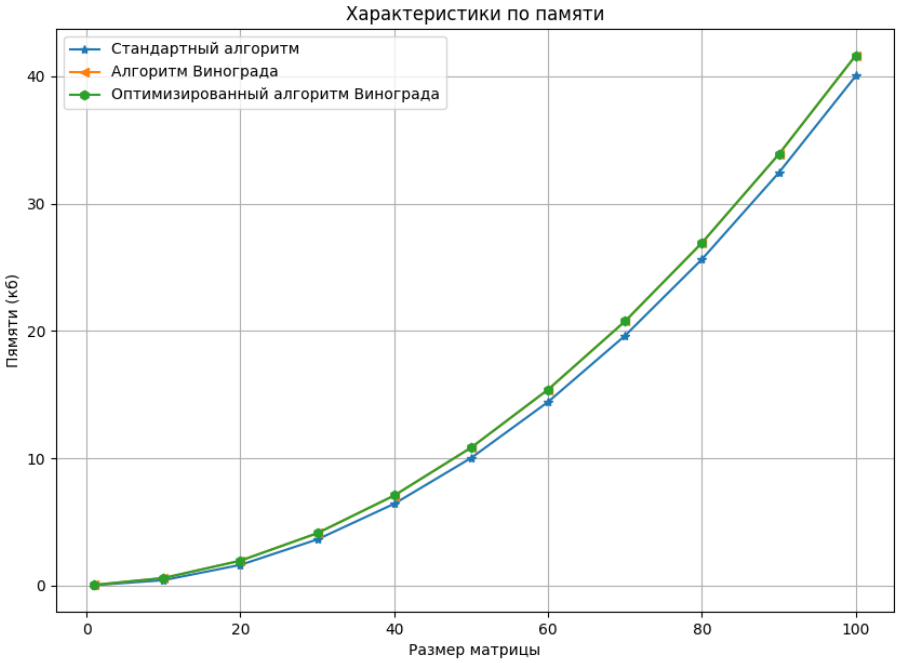

Рисунок 4.5 – Сравнение по памяти алгоритмов умножения матриц

4.5 Вывод

В результате эксперимента было получено, что при размере матриц, большем 40, оптимизированный алгоритм Винограда работает быстрее стандартного алгоритма в 1.45 раза. При этом стандартный алгоритм медленнее алгоритма Винограда в 1.3 раза. Тогда, для размера матриц, начиная с 40 элементов, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду. Стандартный алгоритм использует меньше всего памяти. Затрата памяти алгоритма Винограда и его оптимизированной версии немного различается.

Также в результате эксперимента было установлено, что при четном размере матриц, алгоритм Винограда работает быстрее, чем на матрицах с нечетным размером в 1.08 раза в связи с проведением дополнительных вычислений для крайних строк и столбцов. Можно сделать вывод, что алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров.

Заключение

В результате исследования было получено, что при размере матриц, большем 40, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду, так как данный алгоритм работает быстрее стандартного алгоритма в 1.45 раза. При этом стандартный алгоритм медленнее алгоритма Винограда в 1.3 раза. Стандартный алгоритм использует меньше всего памяти. Затрата памяти алгоритма Винограда и его оптимизированной версии немного различается.

Кроме того алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров, так как указанный алгоритм работает в 1.08 раза быстрее, чем на матрицах с нечетным размером. Это связано с проведением дополнительных вычислений для крайних строк и столбцов.

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

1) описаны алгоритмы стандартного умножения и алгоритм Винограда;
2) построены схемы следующих алгоритмов:
   - классический алгоритм умножения матриц;
   - алгоритм Винограда;
   - оптимизированный алгоритм Винограда.
3) создано программное обеспечение, реализующее перечисленные алгоритмы;
4) проведен сравнительный анализ реализованных алгоритмов;
5) подготовлен отчет о выполненной лабораторной работе.

Список использованных источников

1. Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. — М.: Гуманит. изд. центр ВЛАДОС, 2003.

2. Черненький В. М. Гапанюк Ю. Е. Методика идентификации пассажира по установочным данным. – М.: Вестник МГТУ им. Н.Э. Баумана. Сер. “Приборостроение”, 2012. Т. 163. С. 30–34.

3. C library function clock() [Электронный ресурс]. Режим доступа: `https://en.cppreference.com/w/cpp/chrono/c/clock`.

4. Matplotlib documentation [Электронный ресурс]. Режим доступа: `https://matplotlib.org/stable/index.html` (дата обращения: 02.11.2023)