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 ИУ7-54Б (Группа) _____ (Подпись, дата) Разин А. (И. О. Фамилия)

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

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной лабораторной работе будет исследованы алгоритмы умножения матриц. Как и в математике, в программировании матрицы часто играют важную роль. Они находят широкое применение в разных областях, включая физику, где они используются для вывода формул, таких как:

1) градиент;
2) дивергенция;
3) ротор. [1]

Также матрицы неизбежно встречаются при выполнении различных операций, таких как сложение, возведение в степень и умножение. При работе над разными задачами размеры матриц могут быть значительными, поэтому оптимизация операций с матрицами становится важным аспектом в программировании. В данной лабораторной работе мы сосредоточимся на оптимизации умножения матриц.

Цель этой лабораторной работы включает в себя описание, реализацию и исследование алгоритмов умножения матриц.

Для достижения этой цели необходимо выполнить следующие задачи:

1) описать два алгоритма умножения матриц;
2) разработать программное обеспечение, которое реализует следующие алгоритмы:
    1) классический алгоритм умножения матриц;
    2) алгоритм Винограда;
    3) оптимизированный алгоритм Винограда;
    4) алгоритм Штрассена.
3) оценить трудоемкость умножения матриц;
4) провести анализ временных затрат работы программы и выявить важные факторы, влияющие на эти затраты;
5) сравнить алгоритмы между собой.

1 Аналитический раздел

В данном разделе рассматриваются классический алгоритм умножения матриц и алгоритм Винограда, а также его оптимизированная версия.

1.0.1 Определение матрицы

Матрица [1] представляет собой таблицу чисел a_{ik} следующего вида:

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

где m - количество строк, а n - количество столбцов. Элементы a_{ik} называются элементами матрицы.

Если A - матрица, то $A_{i,j}$ обозначает элемент матрицы, расположенный на i-й строке и j-м столбце.

Существуют различные операции, выполняемые с матрицами:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц, если количество столбцов в первой матрице равно количеству строк во второй матрице. Результатом является матрица, у которой количество строк равно количеству строк первой матрицы, а количество столбцов - количеству столбцов второй матрицы.

Замечание: в общем случае операция умножения матриц, когда в первой матрице число столбцов не равно числу строк во второй не определена. Также стоит отметить что операция умножения матриц не коммутативна ($A \cdot B$ не всегда равно $B \cdot A$) [1].

1.0.2 Классический алгоритм

Допустим, существует 2 матрицы.

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

В таком случае умножение происходит по принципу «строка на столбец», результом умножения матриц, заданных в 1.2 ($A \cdot B$), является матрица C, расчитанная по формуле (1.3).

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

1.0.3 Алгоритм Винограда

В 1987 году Дон Копперсмит и Виноград опубликовали метод, содержащий асимптотическую сложность $O(n^{2,3755})$ и улучшили его в 2011 до $O(n^{2,373})$, где n – размер сторон матрицы [2]. В основной идее алгоритма были заложены следующие рассуждения: Если посмотреть на 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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$. Это равенство можно переписать в виде (1.4).

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

Пусть матрицы A, B, C, ранее определенных размеров. Скалярное произведение (1.4), по замыслу Винограда, можно использовать при расчете

произведения матриц (1.5).

$$C_{ij} = \sum_{k=1}^{q/2}(a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \sum_{k=1}^{q/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{q/2} b_{2k-1,j}b_{2k,j} \quad (1.5)$$

Заметим, что в выражении (1.4), операнды со знаком − могут быть вычислены заранее для каждой строки и каждого столбца исходных матриц, а затем переиспользовать полученные значения что позволяет на практике заметно ускорить расчеты. Аналгично с (1.5), единожды вычислив $\sum_{k=1}^{q/2} a_{i,2k-1}a_{i,2k}$ для строки, можно использовать данное значение для получения значения любого произведения с операндами данной строки. При предварительном расчете значений столбцов используется аналогичная идея.

1.0.4 Оптимизированный алгоритм Винограда

При программной реализации рассмотренного выше алгоритма Винограда можно сделать следующие оптимизации:

1) значение $\frac{N}{2}$, используемое как ограничения цикла подсчёта предварительных данных, можно кэшировать;

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

3) операции сложения и вычитания с присваиванием следует реализовывать при помощи соответствующего оператора += или -= (при наличии данных операторов в выбранном языке программирования).

1.0.5 Алгоритм Штрассена

Фолькер Штрассен предложил разбить матрицу на матрицы размером $2x2$, после чего посчитать их произведение. Было доказано, что в случае, если необходимо вычислить произведение 2 матриц размерности 2×2, над кольцом R. То это возможно сделать за 7 умножений и 18 сложений в R. В данном случае R — кольцо, т.е., ассоциативные операции сложения и умножения, сложение коммутативно, дистрибутивность, нейтральные элементы, противоположный элемент относительно сложения (без противоположного элемента — полукольцо) [3].

Стоит отметить, что данный алгоритм в общем случае применим только к квадратным матрицам размерности $n \times n$, где n - некая неотрицательная степень двойки. Исходная матрица рекурсивно разделяется на 4 матрицы размерностью $\frac{n}{2}$. Пока не будет достигнуты матрицы с размерностью 2×2, вычисление которых происходит тривиально по классическому алгоритму умножения матриц. После чего вычисляются 7 произведений (1.6).

$$\begin{aligned} d1 &= (a_{11} + a_{22})(b_{11} + b_{22}), \\ d2 &= (a_{21} + a_{22})b_{11}, \\ d3 &= a_{11}(b_{12} - b_{11}), \\ d4 &= a_{22}(b_{21} - b_{11}), \\ d5 &= (a_{11} + a_{12})b_{22}, \\ d6 &= (a_{21} - a_{11})(b_{11} + b_{12}), \\ d7 &= (a_{12} - a_{22})(b_{21} + b_{22}). \end{aligned} \tag{1.6}$$

После вычисления (1.6), результат искомого произведения $C = AB$, а именно матрицу C, можно записать как (1.7).

$$\begin{aligned} c_{11} &= d1 + d4 + d7 - d5, \\ c_{12} &= d3 + d5, \\ c_{21} &= d2 + d4, \\ c_{22} &= d1 + d3 + d6 - d2. \end{aligned} \tag{1.7}$$

Смысл умножения матриц 2×2 за 7 умножений и много сложений в том, что эту формулу для произведения матриц можно применять рекурсивно, сводя умножение матриц размера $n \times n$ к умножению семи пар матриц вдвое меньшего размера — семи, а не восьми [3].

Вывод

В данном разделе было введено понятие матрицы, определены несколько операций на ней, а также рассмотрена операция умножения матриц. В данном разделе мы изучили матрицы и операции, которые можно выполнять с ними, включая умножение. Было рассмотрено три алгоритма умножения матриц: классический и алгоритм Винограда, а также алгоритма Штрассена. Алгоритм Винограда выделяется своей эффективностью благодаря использованию предварительных вычислений. Алгоритм Штрассена позволяет уменьшить

число операций благодаря использованию рекурсивной формулы. Также были введены различные оптимизации, которые можно учесть при реализации алгоритма Винограда для достижения более высокой производительности.

2 Конструкторский раздел

В этом разделе будут рассмотрены различные реализации классического алгоритма умножения матриц и алгоритм умножения матриц Винограда, а также алгоритм умножения матриц Штрассена.

2.1 Реализации алгоритмов

В данной части работы будут рассмотрены различные реализации алгоритмов умножения матриц и представлены схемы алгоритмов данных реализаций.

На вход алгоритмов подаются матрицы m_1 и m_2 с размерностями $n_1 \times m_1$ и $n_2 \times m_2$.

2.2 Оценка трудоемкости реализаций алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1) Трудоемкость следующих базовых операций единична: +, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, –, «, ».

   Операции *, %, / имеют трудоемкость 2;

2) трудоемкость цикла for(k = 0; k < N; k++) {тело цикла} рассчитывается как:

$$f_{for} = f_{\text{инициал.}} + f_{\text{сравн.}} + N(f_{\text{тела}} + f_{\text{инкр.}} + f_{\text{сравн.}}); \tag{2.1}$$

3) трудоемкость передачи параметра в функции и возврат из функции равны 0;

4) трудоемкость условного оператора if (условие) then A else B рассчитывается как:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{л.с.} \\ max(f_A, f_B), & \text{х.с.} \end{cases} \tag{2.2}$$

В (2.2), л.с. обозначает лучший случай, х.с. - худший случай.

2.2.1 Трудоемкость реализации классического умножения матриц

Для реализации А.1 алгоритма умножения матриц трудоемкость будет слагаться из:

1) Внешнего цикла по переменной i, имеющим трудоемкость $f_{for_i} = 1 + 1 + n_1 \cdot (f_{for_j} + 1 + 1)$

2) Внутреннего цикла по переменной j, имеющим трудоемкость $f_{for_j} = 1 + 1 + m_2 \cdot (3 + f_{for_k} + 1 + 1)$

3) Цикла по переменной k, имеющим трудоемкость $f_{for_k} = 1 + 1 + n_2 \cdot (12 + 1 + 1)$

Таким образом трудоемкость реализации стандартного алгоритма А.1 вычисляется по формуле (2.3).

$$\begin{aligned} f_{std} = 2 + n_1 \cdot (2 + 2 + m_2 \cdot (5 + 2 + n_2 \cdot (12 + 1 + 1))) = \\ = 2 + 4n_1 + 7n_1m_2 + 14n_1m_2n_2 \end{aligned} \tag{2.3}$$

2.2.2 Трудоемкость реализации алгоритма Винограда

Для реализации А.2 алгоритма умножения матриц трудоемкость будет слагаться из:

1) Цикла заполнения массива, хранящего информацию о произведениях строк (2.4);

2) цикла заполнения массива, хранящего информацию о произведениях столбцов (2.5);

3) цикла перемножения матриц (2.6);

4) цикла дополнительных при расчетов при нечетных m_1, n_2.

$$f_1 = 2 + n_1(1 + 3 + \frac{m_1}{2} \cdot 19) = \frac{19}{2}m_1n_1 + 4n_1 + 2, \tag{2.4}$$

$$f_2 = 2 + m_2(1 + 3 + \frac{m_1}{2} \cdot 19) = \frac{19}{2}m_1m_2 + 4m_2 + 2, \tag{2.5}$$

$$f_3 = 2 + n_1 \cdot (4 + m_2 \cdot (2 + 7 + 4 + \frac{m_1}{2} \cdot (4 + 28))) = 2 + \\ + 4n_1 + 13n_1m_2 + 16n_1m_1m_2, \tag{2.6}$$

$$f4 = 3 + \begin{cases} 0, & \text{л.с.} \\ 2 + n_1(4 + (2 + 14)m_2) = 16n_1m_2 + 4n_1 + 2, & \text{х.с.} \end{cases} \tag{2.7}$$

2.2.3 Трудоемкость оптимизированной реализации алгоритма Винограда

В данном случае используется несколько оптимизаций:

1) Вместо операции умножения на 2 используется побитовый сдвиг влево, который имеет меньшую трудоемкость;

2) кешируется значение$\frac{m_1}{2}$;

3) При сложении используется операция +=, имеющая меньшую трудоемкость.

Трудоемкость реализации А.3 слагается из циклов (2.8 – 2.11) и трудоемкости кеширования:

$$f_1 = 2 + n_1(1 + 1 + \frac{m_1}{2} \cdot 13) = \frac{13}{2}m_1n_1 + 2n_1 + 2, \tag{2.8}$$

$$f_2 = 2 + m_2(1 + 1 + \frac{m_1}{2} \cdot 13) = \frac{13}{2}m_1m_2 + 2m_2 + 2, \tag{2.9}$$

$$f_3 = 2 + n_1 \cdot (4 + m_2 \cdot (2 + 7 + 2 + \frac{m_1}{2} \cdot (2 + 21))) = 2 + \\ + 4n_1 + 11n_1m_2 + \frac{23}{2}n_1m_1m_2, \tag{2.10}$$

$$f4 = 3 + \begin{cases} 0, & \text{л.с.} \\ 2 + n_1(4 + (2 + 13)m_2) = 15n_1m_2 + 4n_1 + 2, & \text{х.с.} \end{cases} \tag{2.11}$$

Стоит учитывать значение $\frac{m_1}{2}$, трудоемкость кеширования которого равно 3.

Наилучшим случаем для данных алгоритмов являются матрицы с четным числом столбцов в первой матрице и четным числом строк во втором.

2.2.4 Трудоемкость реализации алгоритма Штрассена

Реализация алгоритма штрассена А.4 рекурсивна, соответственно трудоемкость рекурсивной функции в общем случае будет вычислена по формуле:

$$f_{rec} = \begin{cases} dir, & \text{при } N = sizeLimit \\ div(N) + \sum_1^l rec(size[i]) + com(N), & \text{при } N > sizeLimit \end{cases} \quad (2.12)$$

В формлуе (2.12), означают соответственно:

1) dir — трудоемкость нерекурсивного решения (база рекурсии);
2) div — трудоемкость разбиения задачи на подзадачи;
3) rec — трудоемкость решения подзадачи;
4) com — трудоемкость объеденения результатов решенных подзадач;
5) N — число входных элементов;

В случае реализации алгоритма Штрассена:

1) $dir = 9$;
2) $com = 2 + \frac{n}{2} \cdot (2 + 2 + \frac{n}{2} \cdot (2 + (5 + 6 + 6 + 7))) = 2 + 2n + \frac{13}{2}n^2$;
3) div В данном случае состоит из деления матриц на подматрицы.
   $div = 2 + \frac{n}{2} \cdot (2 + 2 + \frac{n}{2} \cdot (2 + 2 \cdot (5 + 6 + 6 + 7))) + 10 \cdot mat_{op} = 4 + 22n + 30n^2$;
4) $rec(n) = mat_{op} \cdot 18$,где $mat_{op} = 2 + \frac{n}{2} \cdot (2 + 2 + \frac{n}{2} \cdot (2 + 5))$, операция сложения/вычитания матриц.

Заметим, что в данной реализации матрицы - операнды дополняются до квадратных, порядок которых равен степени двойки. Так что наилучшим случаем будет квадратная матрица, порядок которой будет степень двойки.

$$f_{rec} = \begin{cases} 9, & \text{при } N = 1, \\ 49n^2 + 26n + 8 + \sum_1^7 rec(\frac{N}{2}), & \text{при } N > 1. \end{cases} \tag{2.13}$$

Таким образом трудоемкость рассчитывается по формуле (2.13). Стоит отметить, что Штрассен в своей работе оптимизирует умножение только квадратных матриц с порядком кратным двум, поэтому в реализации A.4, изначально матрицы приводятся к порядку, равному наименьшей степени двойки большей исходного порядка. Таким образом высота рекурсивного дерева будет равна $\log_2 N$. При учете предыдущего условия трудоемкость алгоритма Щтрассена вычисляется по формуле (2.14).

$$T = 7^{\log_2 N} \cdot 9 + \sum_0^{\log_2 N - 1} 7^i \cdot (49(\frac{N}{2^i})^2 + 4(\frac{N}{2^i}) + 4) \tag{2.14}$$

2.3 Структуры данных

Для реализации алгоритмов, будут использованы следующие типы данных: Для реализации выбранных алгоритмов были использованы следующие структуры данных:

1) Матрица — массив массивов типа int;

2) размерности матрицы — целые значение типа size_t.

Вывод

Были разработаны схемы алгоритмов, позволяющих с помощью различных подходов находить произведение матриц, также была дана оценка трудоемкости для рассмотренных алгоритмов, кроме того были рассмотрены структуры данных, необходимые при реализации данных алгоритмов.

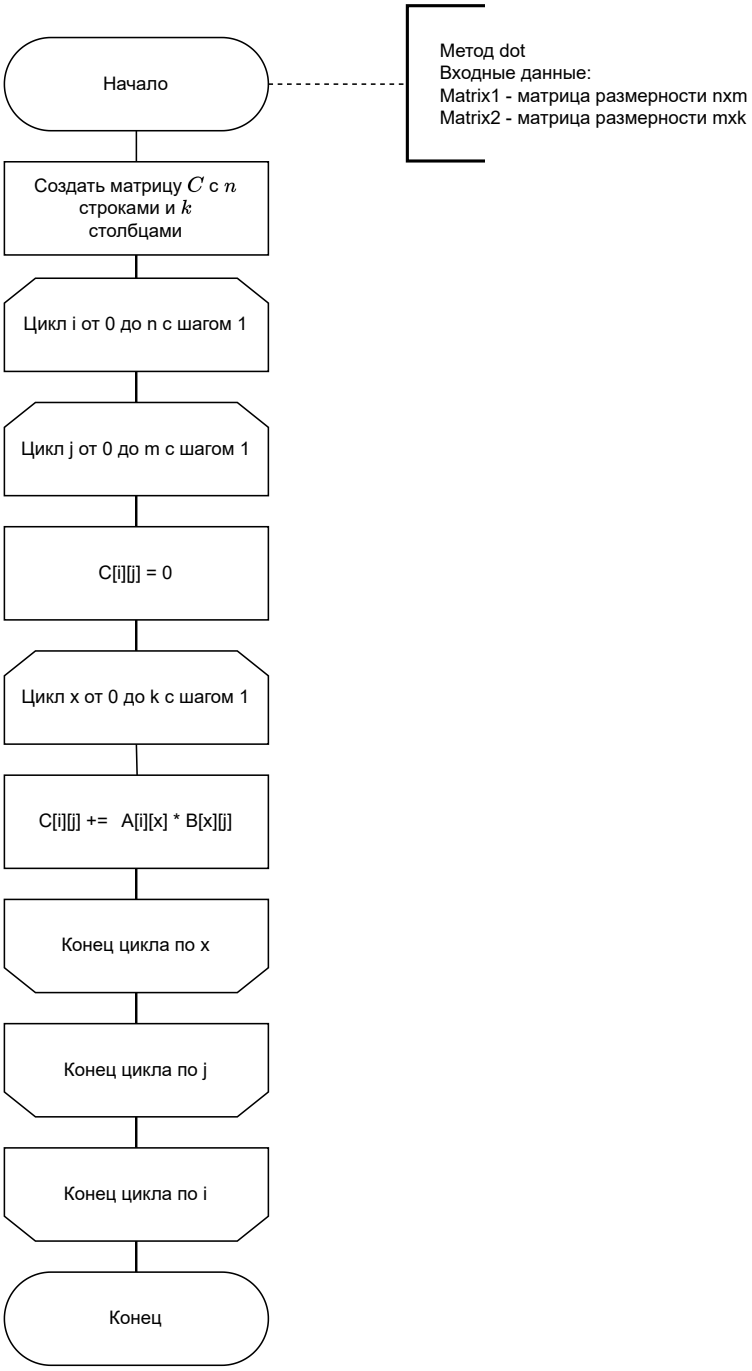

Рисунок 2.1 – Стандартный алгоритм расчета произведения матриц

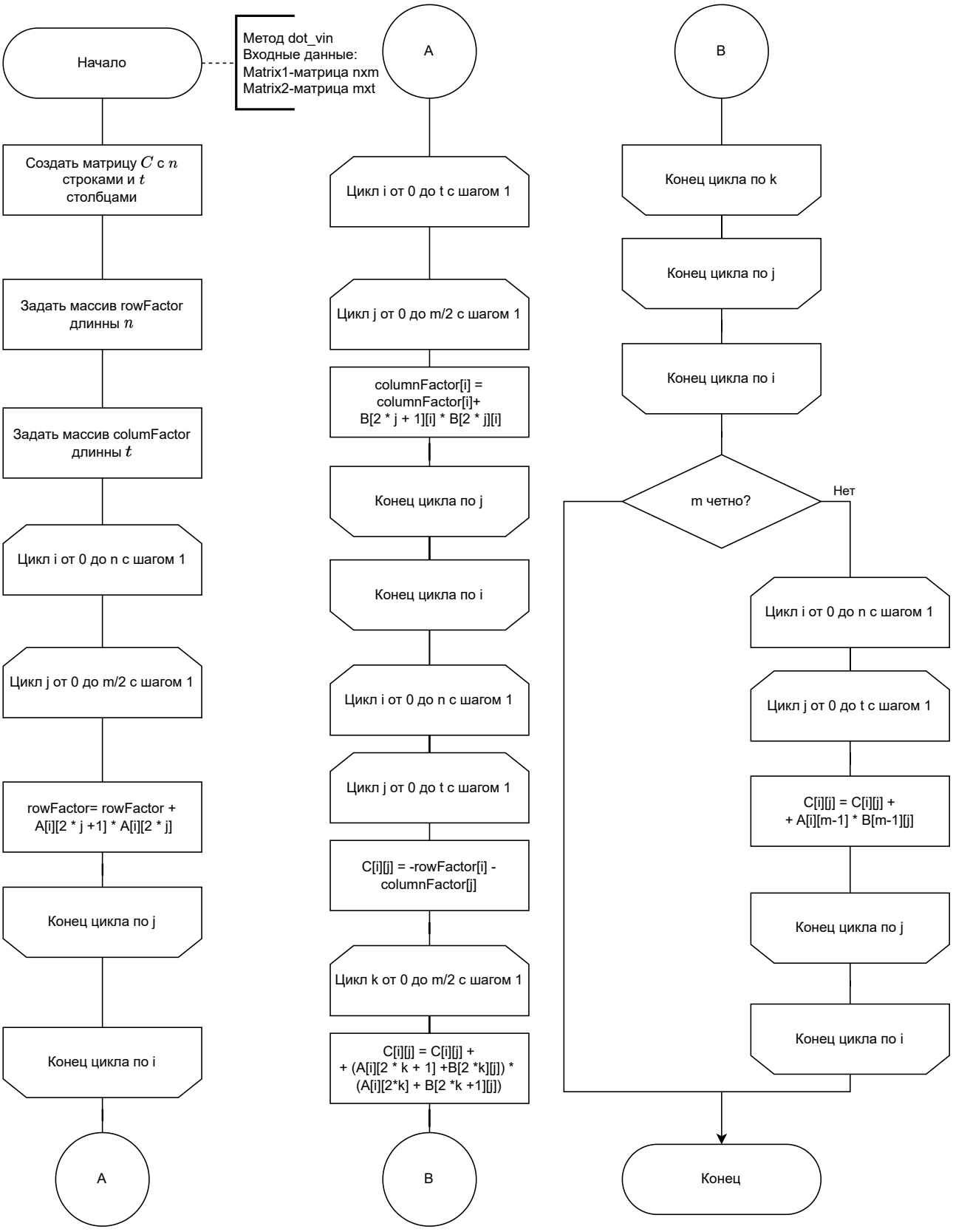

Рисунок 2.2 – Алгоритм Винограда

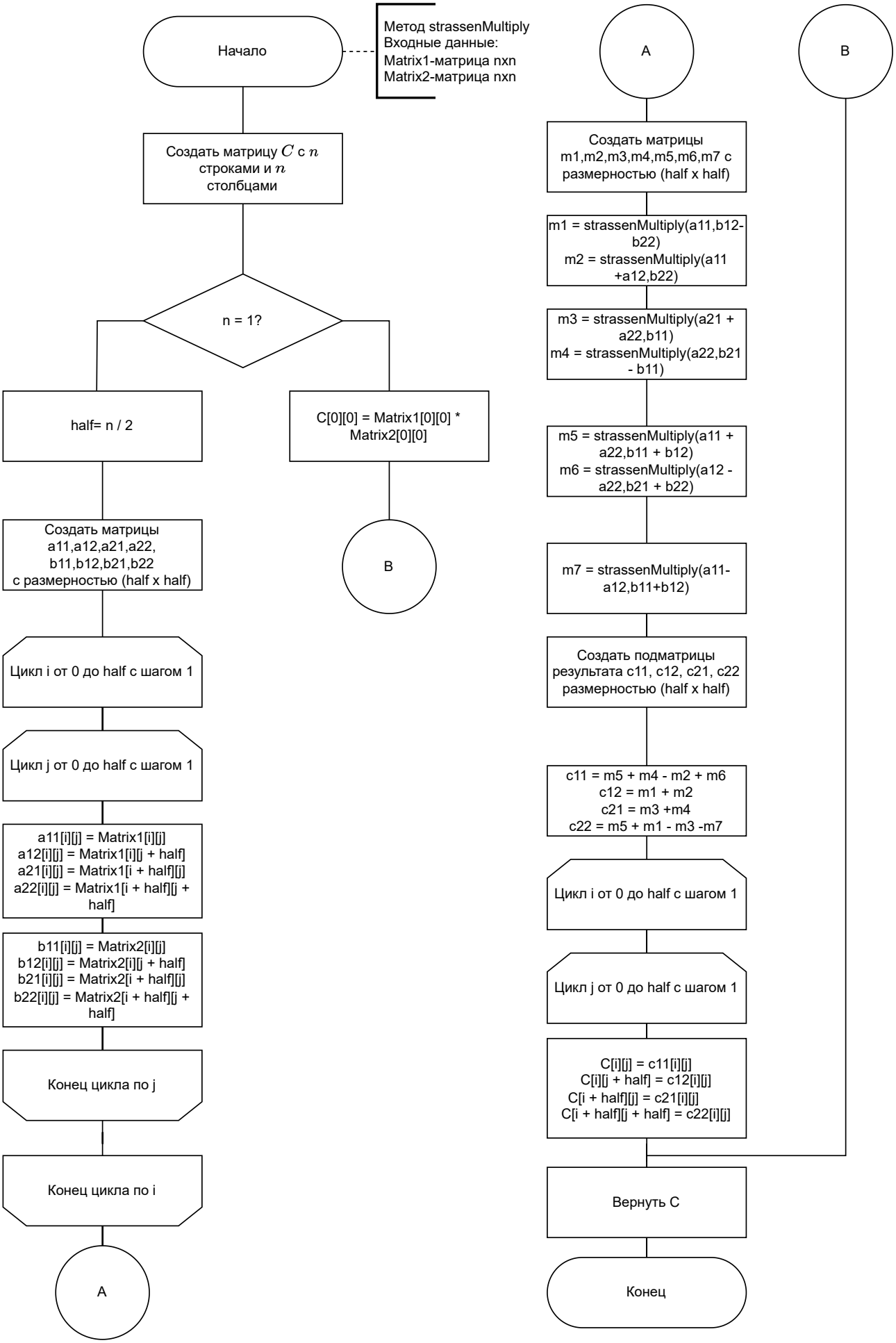

Рисунок 2.3 – Алгоритм Штрассена

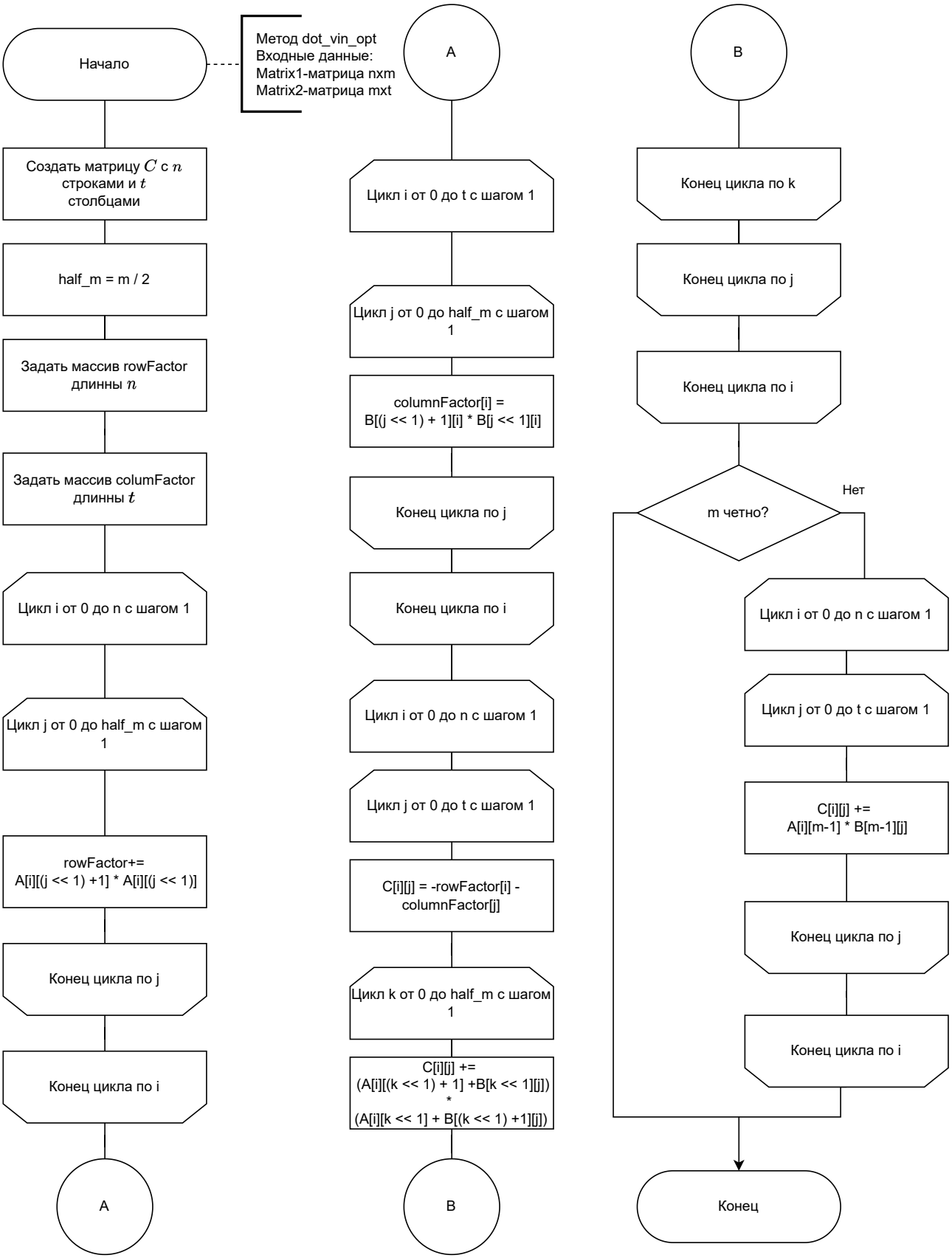

Рисунок 2.4 – Оптимизированный алгоритм Винограда

3 Технологический раздел

В данной части работы будут описаны средства реализации программы, а также листинги, модульные и функциональные тесты.

3.1 Средства реализации

Алгоритмы для данной лабораторной работы были реализованы на языке C++, при использовании компилятора gcc версии 10.5.0, так как в стандартной библиотеке приведенного языка присутствует функция clock_gettime, которая (при использовании макропеременной CLOCK_THREAD_CPUTIME_ID) позволяет рассчитать процессорное время конкретного потока [4].

3.2 Реализация алгоритмов

Стоит отметить, что все используемые выше алгоритмы реализованы как метода класса $Matrix$, рекурсивные части алгоритмов были вынесены в отдельные функции. Листинги исходных кодов программ А.1–А.4 приведены в приложении.

3.3 Тестирование

Функциональные тесты рассмотрены в таблицах 3.1 – 3.3.

Все тесты были успешно пройдены.

Вывод

В данной части работы были рассмотрены средства реализации, проведено тестирование, а также были реализованы рассмотренные ранее алгоритмы.

Таблица 3.1 – Функциональные тесты для классического алгоритма умножения матриц

Входные данные		Результат для классического алгоритма	
Матрица 1	Матрица 2	Ожидаемый результат	Фактический результат
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 \\ 22 & 7 & 22 \\ 39 & 14 & 43 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 \\ 22 & 7 & 22 \\ 39 & 14 & 43 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \\ 1 & 4 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 0 & 7 & 8 \\ 0 & 1 & 4 & 0 & 5 & 6 \\ 5 & 0 & 1 & 0 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 & 0 & 26 & 32 \\ 22 & 7 & 22 & 0 & 41 & 50 \\ 39 & 14 & 43 & 0 & 79 & 96 \\ -9 & 6 & 17 & 0 & 21 & 24 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 & 0 & 26 & 32 \\ 22 & 7 & 22 & 0 & 41 & 50 \\ 39 & 14 & 43 & 0 & 79 & 96 \\ -9 & 6 & 17 & 0 & 21 & 24 \end{pmatrix}$
$\begin{pmatrix} 3 & 5 \\ 2 & 1 \\ 9 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 35 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$

Таблица 3.2 – Функциональные тесты для умножения матриц по алгоритму Винограда

Входные данные		Результат для классического алгоритма	
Матрица 1	Матрица 2	Ожидаемый результат	Фактический результат
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 \\ 22 & 7 & 22 \\ 39 & 14 & 43 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 \\ 22 & 7 & 22 \\ 39 & 14 & 43 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \\ 1 & 4 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 0 & 7 & 8 \\ 0 & 1 & 4 & 0 & 5 & 6 \\ 5 & 0 & 1 & 0 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 & 0 & 26 & 32 \\ 22 & 7 & 22 & 0 & 41 & 50 \\ 39 & 14 & 43 & 0 & 79 & 96 \\ -9 & 6 & 17 & 0 & 21 & 24 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 & 0 & 26 & 32 \\ 22 & 7 & 22 & 0 & 41 & 50 \\ 39 & 14 & 43 & 0 & 79 & 96 \\ -9 & 6 & 17 & 0 & 21 & 24 \end{pmatrix}$
$\begin{pmatrix} 3 & 5 \\ 2 & 1 \\ 9 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 35 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$

Таблица 3.3 – Функциональные тесты для реализации алгоритма умножения матриц Штрассена

Входные данные		Результат для классического алгоритма	
Матрица 1	Матрица 2	Ожидаемый результат	Фактический результат
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 \\ 22 & 7 & 22 \\ 39 & 14 & 43 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 \\ 22 & 7 & 22 \\ 39 & 14 & 43 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 6 & 7 \\ 1 & 4 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 0 & 7 & 8 \\ 0 & 1 & 4 & 0 & 5 & 6 \\ 5 & 0 & 1 & 0 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 & 0 & 26 & 32 \\ 22 & 7 & 22 & 0 & 41 & 50 \\ 39 & 14 & 43 & 0 & 79 & 96 \\ -9 & 6 & 17 & 0 & 21 & 24 \end{pmatrix}$	$\begin{pmatrix} 16 & 4 & 14 & 0 & 26 & 32 \\ 22 & 7 & 22 & 0 & 41 & 50 \\ 39 & 14 & 43 & 0 & 79 & 96 \\ -9 & 6 & 17 & 0 & 21 & 24 \end{pmatrix}$
$\begin{pmatrix} 3 & 5 \\ 2 & 1 \\ 9 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 35 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$

4 Исследовательский раздел

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

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

1) Процессор Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 2592 МГц, ядер: 6, логических процессоров: 12;
2) Оперативная память: 16 ГБайт;
3) Операционная система: Майкрософт Windows 10 Pro [5];
4) Использованная подсистема: WSL2 [6].

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

4.2 Пример работы программы

На рисунке 4.1, представлен пример работы программы. Были введены размерности матриц и выбран метод их умножения, после чего был выведен результат умножения.

```
Меню:
0)Выход
1)Умножение матриц стандартным методом
2)Умножение алгоритмом Винограда
3)Умножение оптимизированным алгоритмом Винограда
4)Умножение алгоритмом Штрассена
5)Провести замеры по времени

Введите пункт из меню: 3

Введите n и m (высоту и ширину первой матрицы)  2 3
Введите первую матрицу
1 2 3
3 4 5
Введите n и m (высоту и ширину второй матрицы) 3 1
Введите вторую матрицу
1
2
3
Введенная первая матрица:
1 2 3
3 4 5
Введенная вторая матрица:
1
2
3
Результирующая матрица:
14
26
```

Рисунок 4.1 – Пример работы программы

4.3 Временные характеристики

Результаты исследования замеров по времени приведены в таблице 4.1. Введем следующие обозначения для чтения таблиц:

1) n — размерность умножаемых матриц;

2) СМ — реализация стандартного алгоритма умножения матриц;

3) ВМ — реализация алгоритма умножения матриц Винограда;

4) ВМО —реализация алгоритма умножения матриц Винограда (оптимизированный);

5) ШМ — реализация алгоритма умножения матриц Штрассена.

Для таблиц 4.1 и 4.2 замеры производились с различным шагом, для каждой пары матриц производилось 100 замеров времени, после чего результаты замеров усреднялись. Все результаты вычислений приведены в миллисекундах.

Таблица 4.1 – Полученная таблица замеров по времени различных реализаций алгоритмов умножения матриц нечетной размерности

n	СМ (мс)	ШМ (мс)	ВМ (мс)	ВМО (мс)
1	0.002159	0.004304	0.001817	0.001758
11	0.025625	7.3367	0.023301	0.020161
21	0.14005	46.559	0.11752	0.1004
31	0.43789	46.262	0.34888	0.29771
41	0.99899	326.63	0.78522	0.66993
51	1.9746	332.43	1.5321	1.3118
61	3.3842	329.97	2.5529	2.1928
71	5.1593	2312.2	4.0072	3.4496
81	7.5757	2279.9	5.763	4.9773
91	10.703	2290.7	7.9208	6.878
101	14.504	2300.3	11.025	9.4659
111	19.166	2285.8	14.579	12.439
121	25.184	2287.3	18.741	16.192

Таблица 4.2 – Полученная таблица замеров по времени различных реализаций алгоритмов умножения матриц четной размерности

n	СМ (мс)	ШМ (мс)	ВМ (мс)	ВМО (мс
2	0.002283	0.024897	0.002821	0.003045
12	0.043452	9.2957	0.037144	0.032815
22	0.19104	51.76	0.14542	0.12715
32	0.5097	48.932	0.39907	0.34554
42	1.0893	331.24	0.83759	0.72743
52	1.9912	323.34	1.509	1.3167
62	3.4132	328.08	2.6004	2.2481
72	5.264	2286.4	3.9626	3.4397
82	8.0663	2332.5	5.9899	5.2073
92	11.053	2294	8.3101	7.1779
102	15.142	2286.2	11.272	9.8261
112	19.793	2273.9	14.748	12.853
122	25.798	2268.4	19.095	16.768

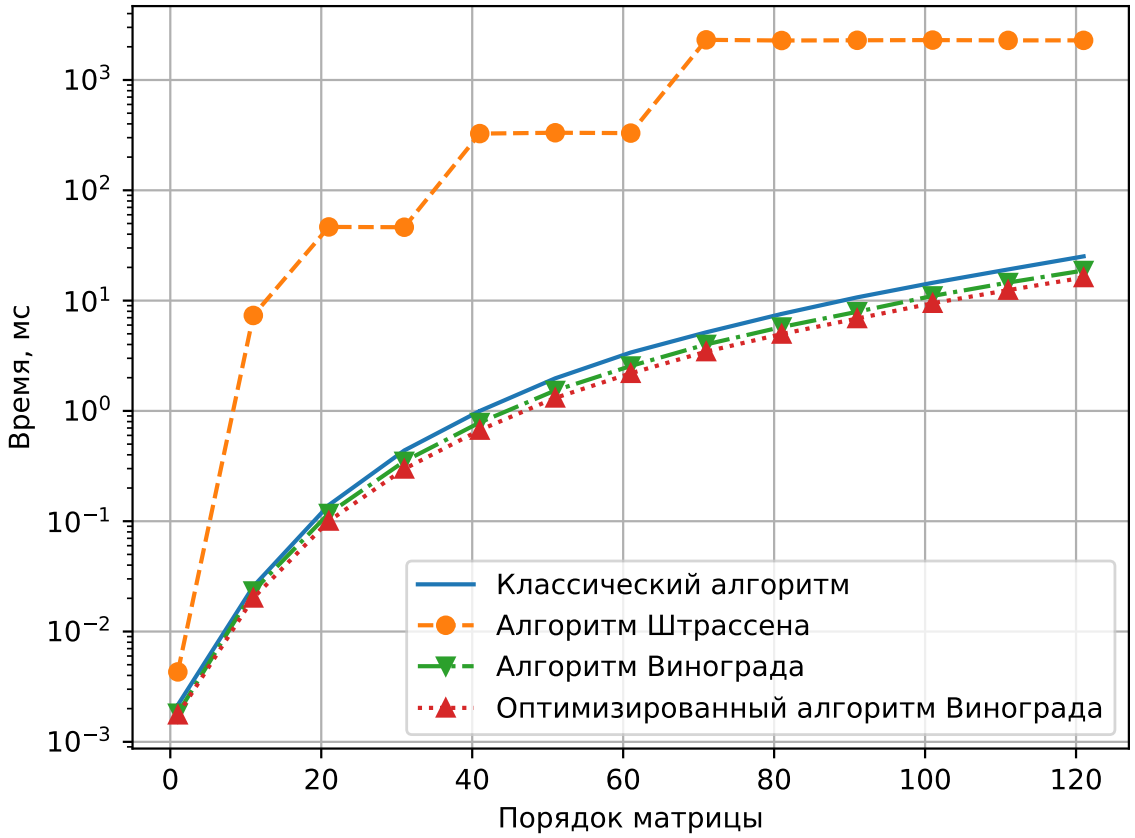

Рисунок 4.2 – Сравнение реализаций исследуемых алгоритмов по временри исполнения с использованием логарифмической шкалы с матрицами нечетных размерностей

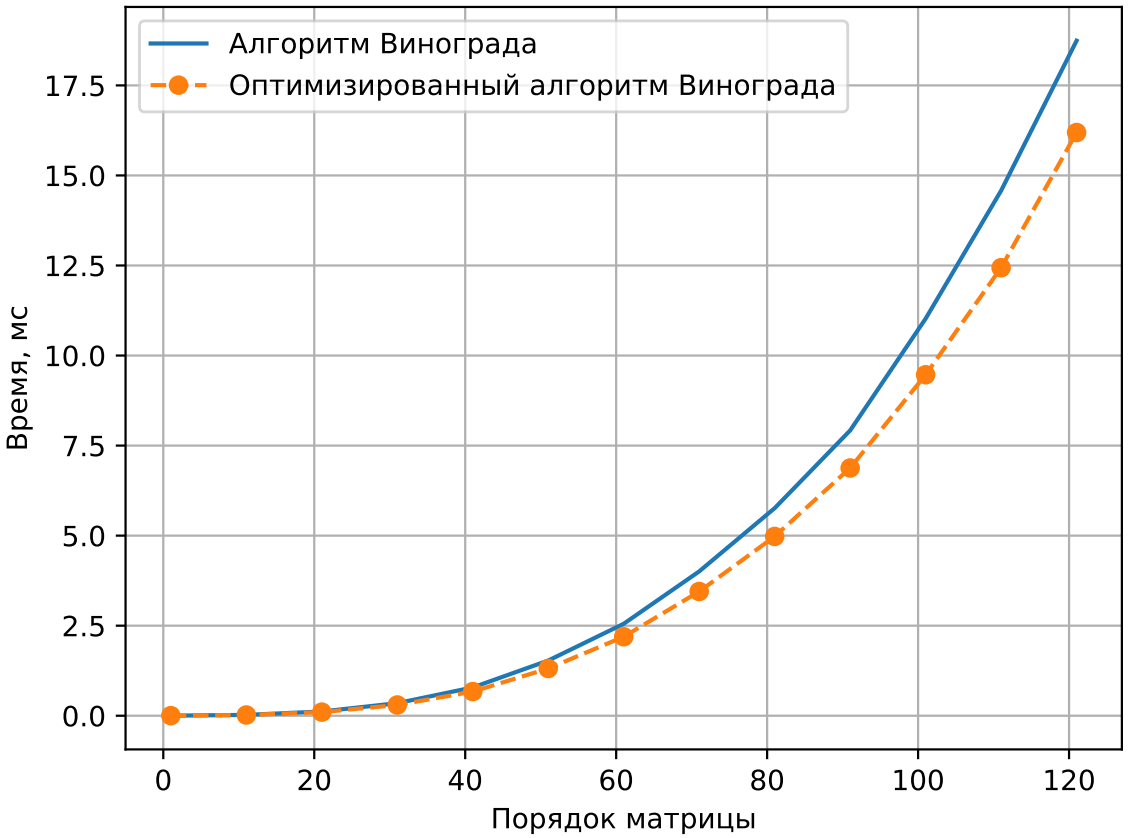

Рисунок 4.3 – Сравнение различных версий реализаций алгоритма Винограда с матрицами нечетных размерностей

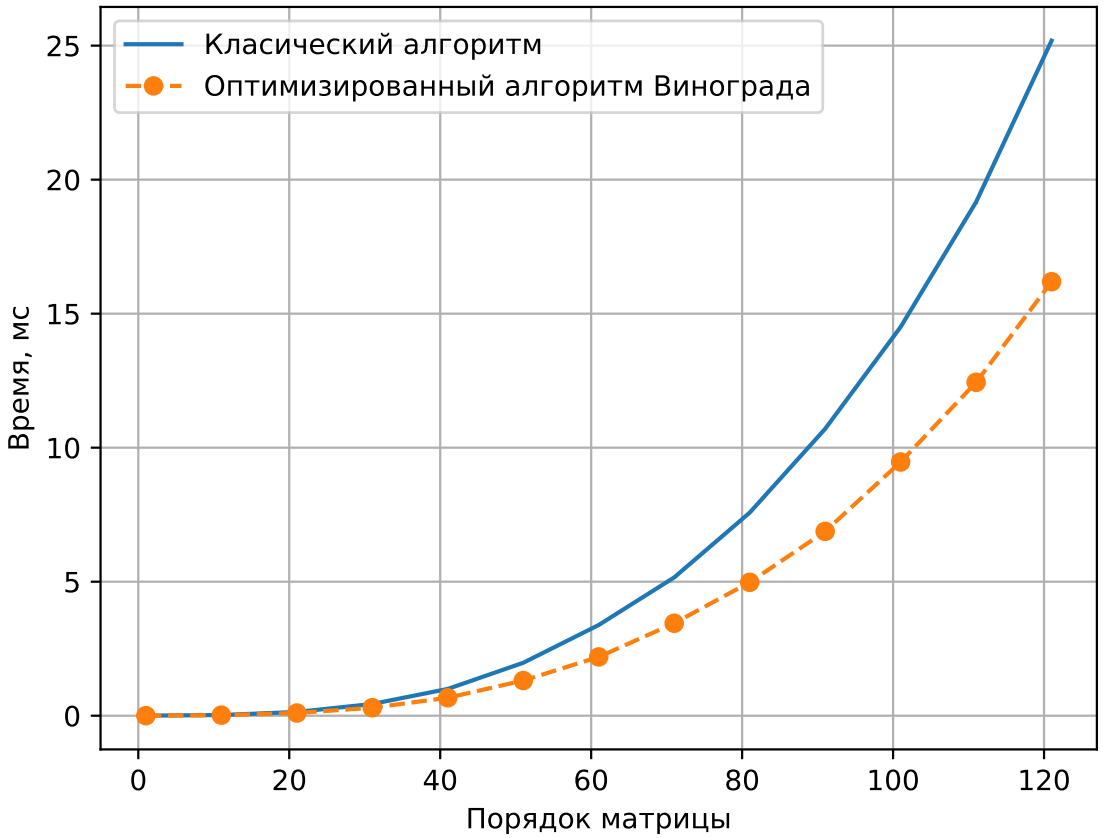

Рисунок 4.4 – Сравнение реализаций алгоритма Винограда и классического алгоритма умножения матриц с нечетными размерностями

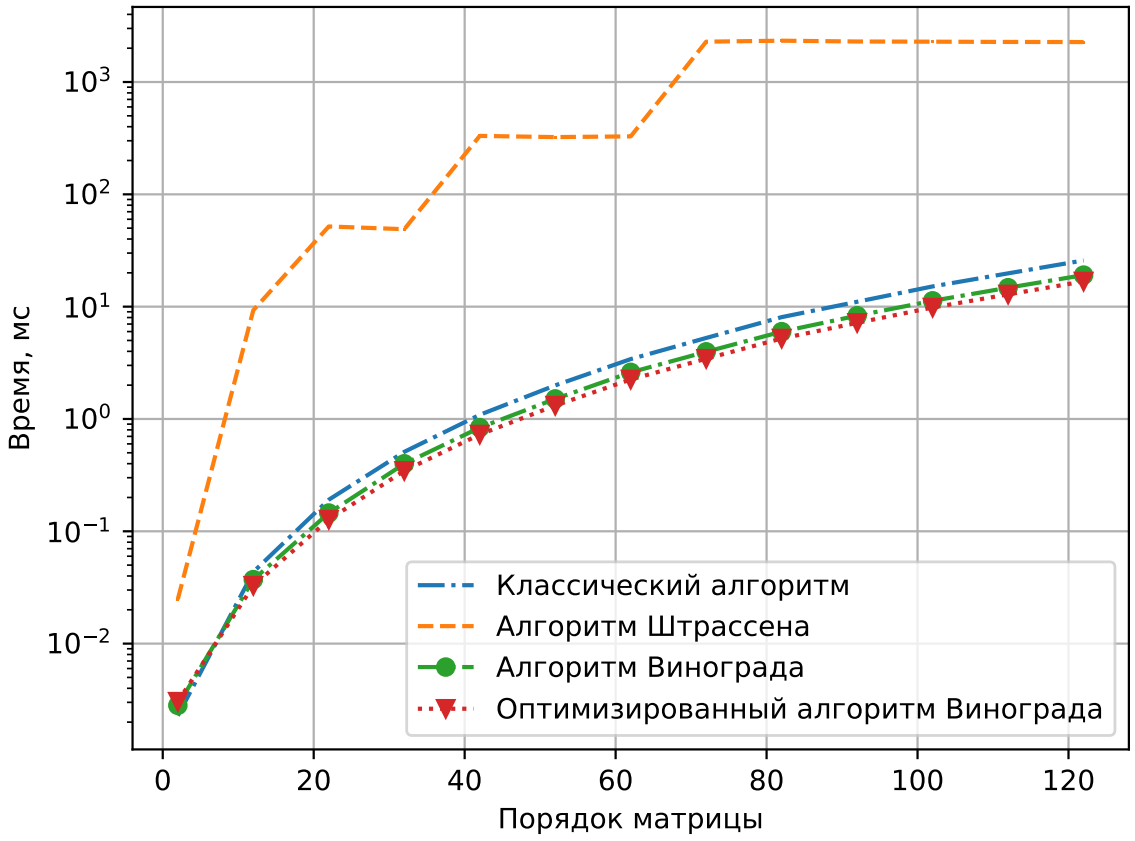

Рисунок 4.5 – Сравнение реализаций исследуемых алгоритмов по времени при исполнения с использованием логарифмической шкалы с четными размерностями матриц

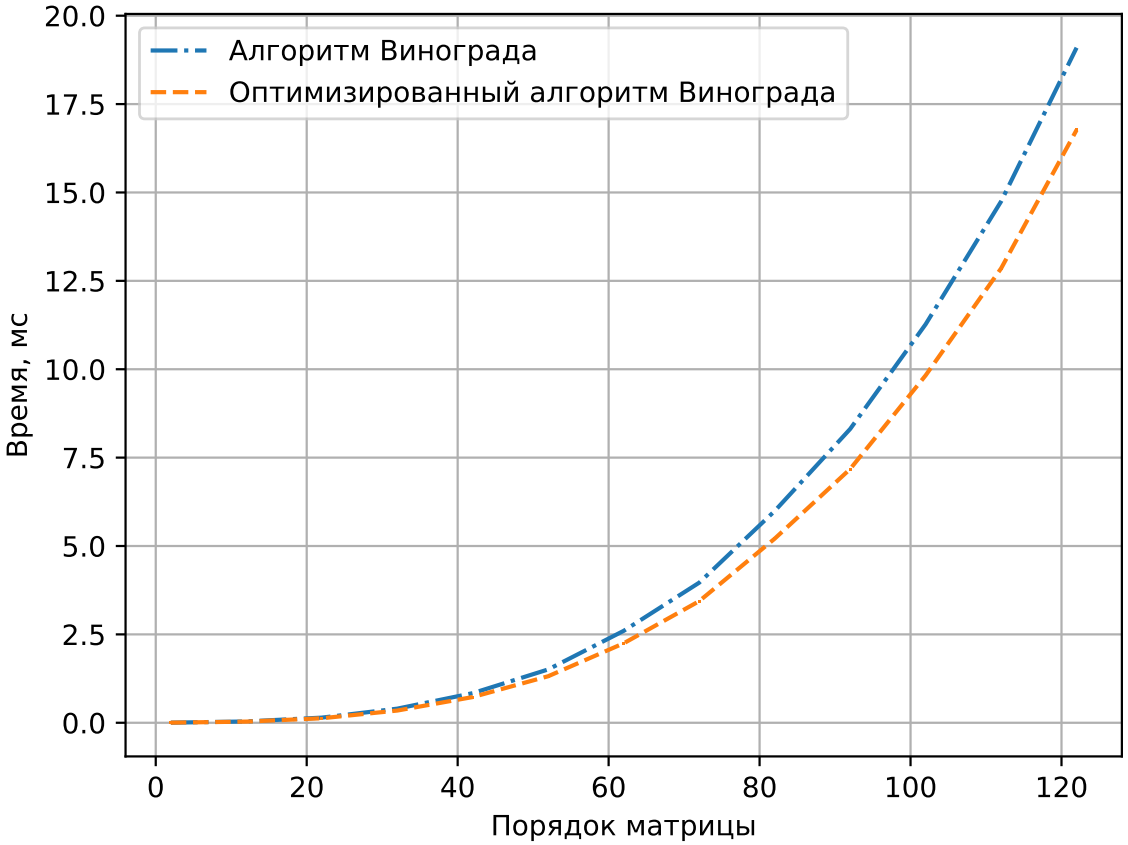

Рисунок 4.6 – Сравнение различных версий реализаций алгоритма Винограда с четными размерностями матриц

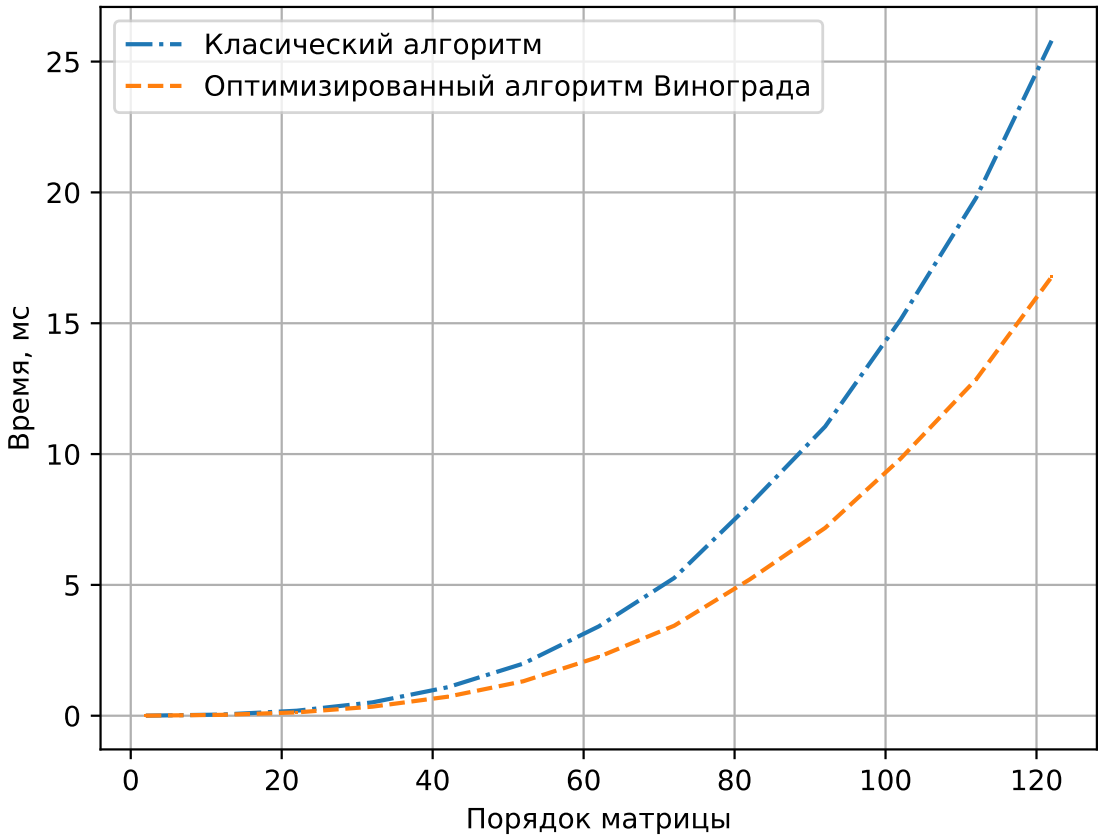

Рисунок 4.7 – Сравнение реализаций алгоритма Винограда и классического алгоритма умножения матриц с четными размерностями матриц

Вывод

В результате анализа таблицы 4.1,4.2, что при больших размерах матриц (свыше 100) нечетной размерности, реализация алгоритма Винограда тратит в 1.38 раза , а реализация оптимизированного алгоритма Винограда в 1.56 раз меньше времени, чем реализация стандартного алгоритма умножения матриц.

Реализация алгоритма Штрассена требует больших временных ресурсов, чем иные исследуемые реализации, при размерностях матриц больше 100 требует в 127 раз больше времени, чем реализация алгоритма Винограда и в 91 раз больше времени, чем стандартная реализация алгоритма умножения матриц, данное обусловлено необходимостью увеличения размерности матриц для использования алгоритма.

Заключение

В результате исследования было определено, что стандартный алгоритм умножения матриц проигрывает по времени алгоритму Винограда примерно в 1.38 раз из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций - операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает оптимизированный алгоритм Винограда – он примерно в 1.58 раза быстрее алгоритма Винограда на размерах матриц свыше 100 из-за замены операций равно и плюс на операцию плюс-равно, а также за счёт замены операции умножения операцией сдвига, что дает проводить часть вычислений быстрее. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда.

Реализация алгоритма Штрассена требует больших временных ресурсов, чем иные исследуемые реализации, при размерностях матриц больше 100 требует в 127 раз больше времени, чем реализация алгоритма Винограда и в 91 раз больше времени, чем стандартная реализация алгоритма умножения матриц. Это обусловлено необходимостью увлечения размерности матрицы, также в реализации данного алгоритма выделяется дополнительная память для разбиения матрицы на подматрицы, что также тратит временные ресурсы.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи.

1) Описаны два алгоритмы умножения матриц;
2) Создано программное обеспечение, реализующее следующие алгоритмы:
   - классический алгоритм умножения матриц;
   - алгоритм Винограда;
   - алгоритм Штрассена;
   - оптимизированный алгоритм Винограда.
3) Оценены трудоемкости реализаций алгоритмов;
4) Проведен анализ затрат работы программы по времени, выяснены влияющие на них характеристики;

5) Проведен сравнительный анализ алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. //. — — М.: Гуманит. изд. центр ВЛАДОС, 2003.
2. Головашкин Д. Л. Векторные алгоритмы вычислительной линейной алгебры: учеб. пособие. //. — — Самара: Изд-во Самарского университета, 2019. — С. 28—35.
3. Конспект лекций СПбГУ [Электронный ресурс]. — — Режим доступа: https://users.math-cs.spbu.ru/%7Eokhotin/teaching/tcs1_2018/okhotin_tcs1alg_2018_l5.pdf (дата обращения: 28.09.2023).
4. C library function clock() [Электронный ресурс]. — — Режим доступа: https://linux.die.net/man/3/clock_gettime (дата обращения: 28.09.2023).
5. Windows 10 Pro 2h21 64-bit [Электронный ресурс]. — — Режим доступа: https://www.microsoft.com/ru-ru/software-download/windows10 (дата обращения: 28.09.2023).
6. Что такое WSL [Электронный ресурс]. — — Режим доступа: https://learn.microsoft.com/ru-ru/windows/wsl/about (дата обращения: 28.09.2023).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация стандартного алгоритма расчета произведения матриц

```
Matrix Matrix::dot(const Matrix& other)
{
    if (this->_m != other._n)
    return Matrix(0, 0);

    Matrix mat_res = Matrix(this->_n, other._m);
    for (size_t i = 0; i < this->_n; i++)
    {
        for (size_t j = 0; j < other._m; j++)
        {
            mat_res._table[i][j] = 0;
            for (size_t k = 0; k < other._n; k++)
            mat_res._table[i][j] = mat_res._table[i][j] +
               this->_table[i][k] * other._table[k][j];
        }
    }
    return mat_res;
}
```

Листинг А.2 – Реализация алгоритма расчета произведения матриц Винограда

```
if (this->_m != other._n)
return Matrix(0, 0);

size_t n = this->_table.size();
size_t m = other._table.size();
size_t t = other._table[0].size();

Matrix mat_res = Matrix(n, t);
std::vector<int> rowFactor(n);
std::vector<int> columnFactor(t);
bool isEvenColumns = (m % 2 == 0);
for (size_t i = 0; i < n; i++)
{
    for (size_t j = 0; j < m / 2; j++)
    rowFactor[i] = rowFactor[i] + this->_table[i][2 * j + 1] *
       this->_table[i][2 * j];
```

```
    }

    for (size_t i = 0; i < t; i++)
    {
        for (size_t j = 0; j < m / 2; j++)
        columnFactor[i] = columnFactor[i] + other._table[2 * j +
            1][i] * other._table[2 * j][i];
    }

    for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < t; j++)
    {
        mat_res._table[i][j] = -rowFactor[i] - columnFactor[j];
        for (size_t k = 0; k < m / 2; k++)
        {
            mat_res._table[i][j] = mat_res._table[i][j] +
                (this->_table[i][2 * k + 1] + other._table[2 * k][j])
            * (this->_table[i][2 * k] + other._table[2 * k + 1][j]);

        }
    }

    if (!isEvenColumns)
    for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < t; j++)
    mat_res._table[i][j] = mat_res._table[i][j] + this->_table[i][m
        - 1] * other._table[m - 1][j];

    return mat_res;
        }
```

Листинг А.3 – Реализация алгоритма расчета произведения матриц Винограда с оптимизациями

```
    Matrix Matrix::dot_vin_opt(const Matrix& other)
    {
        if (this->_m != other._n)
        return Matrix(0, 0);

        size_t n = this->_table.size();
        size_t m = other._table.size();
        size_t t = other._table[0].size();
```

```
size_t half_m = m / 2;

Matrix mat_res = Matrix(n, t);
std::vector<int> rowFactor(n);
std::vector<int> columnFactor(t);
bool isEvenColumns = (m % 2 == 0);
for (size_t i = 0; i < n; i++)
{
    for (size_t j = 0; j < half_m; j++)
    rowFactor[i] += this->_table[i][(j << 1) + 1] *
       this->_table[i][j << 1];
}

for (size_t i = 0; i < t; i++)
{
    for (size_t j = 0; j < half_m; j++)
    columnFactor[i] += other._table[(j << 1) + 1][i] *
       other._table[j << 1][i];
}

for (size_t i = 0; i < n; i++)
for (size_t j = 0; j < t; j++)
{
    mat_res._table[i][j] = -rowFactor[i] - columnFactor[j];
    for (size_t k = 0; k < half_m; k++)
    {
        mat_res._table[i][j] += (this->_table[i][(k << 1) +
           1] + other._table[k << 1][j])
        * (this->_table[i][k << 1] + other._table[(k << 1)
           + 1][j]);

    }
}

if (!isEvenColumns)
for (size_t i = 0; i < n; i++)
for (size_t j = 0; j < t; j++)
mat_res._table[i][j] += this->_table[i][m - 1] *
   other._table[m - 1][j];

return mat_res;
```

```
}
```

Листинг А.4 – Реализация алгоритма расчета произведения матриц Штрассена

```
std::vector<std::vector<int>> strassenMultiply(const
    std::vector<std::vector<int>>& matrix1,
const std::vector<std::vector<int>>& matrix2)
{

    size_t n = matrix1.size();

    if (n == 1)
    {
        std::vector<std::vector<int>> result(1,
            std::vector<int>(1, 0));
        result[0][0] = matrix1[0][0] * matrix2[0][0];
        return result;
    }


    size_t half = n / 2;
    std::vector<std::vector<int>> a11(half,
        std::vector<int>(half, 0));
    std::vector<std::vector<int>> a12(half,
        std::vector<int>(half, 0));
    std::vector<std::vector<int>> a21(half,
        std::vector<int>(half, 0));
    std::vector<std::vector<int>> a22(half,
        std::vector<int>(half, 0));

    std::vector<std::vector<int>> b11(half,
        std::vector<int>(half, 0));
    std::vector<std::vector<int>> b12(half,
        std::vector<int>(half, 0));
    std::vector<std::vector<int>> b21(half,
        std::vector<int>(half, 0));
    std::vector<std::vector<int>> b22(half,
        std::vector<int>(half, 0));

    for (size_t i = 0; i < half; i++)
    {
        for (size_t j = 0; j < half; j++)
        {
```

```
            a11[i][j] = matrix1[i][j];
            a12[i][j] = matrix1[i][j + half];
            a21[i][j] = matrix1[i + half][j];
            a22[i][j] = matrix1[i + half][j + half];

            b11[i][j] = matrix2[i][j];
            b12[i][j] = matrix2[i][j + half];
            b21[i][j] = matrix2[i + half][j];
            b22[i][j] = matrix2[i + half][j + half];
        }
    }


    std::vector<std::vector<int>> m1 = strassenMultiply(a11,
       b12 - b22);
    std::vector<std::vector<int>> m2 = strassenMultiply(a11 +
       a12, b22);
    std::vector<std::vector<int>> m3 = strassenMultiply(a21 +
       a22, b11);
    std::vector<std::vector<int>> m4 = strassenMultiply(a22,
       b21 - b11);
    std::vector<std::vector<int>> m5 = strassenMultiply(a11 +
       a22, b11 + b22);
    std::vector<std::vector<int>> m6 = strassenMultiply(a12 -
       a22, b21 + b22);
    std::vector<std::vector<int>> m7 = strassenMultiply(a11 -
       a21, b11 + b12);


    std::vector<std::vector<int>> c11 = m5 + m4 - m2 + m6;
    std::vector<std::vector<int>> c12 = m1 + m2;
    std::vector<std::vector<int>> c21 = m3 + m4;
    std::vector<std::vector<int>> c22 = m5 + m1 - m3 - m7;


    std::vector<std::vector<int>> result(n, std::vector<int>(n,
       0));
    for (size_t i = 0; i < half; i++)
    {
        for (size_t j = 0; j < half; j++)
        {
```

```
                result[i][j] = c11[i][j];
                result[i][j + half] = c12[i][j];
                result[i + half][j] = c21[i][j];
                result[i + half][j + half] = c22[i][j];
            }
        }

        return result;
    }
```

Листинг А.5 – Метод расчета произведения алгоритмом штрассена с расширением исходной матрицы

```
    Matrix Matrix::dot_shtrassen(const Matrix& other)
    {
        if (this->_m != other._n)
        return Matrix(0, 0);
        size_t max_dim = std::max(std::max(this->_n, this->_m),
           other._m);
        size_t max_dim2d = pow(2, ceil(log2(max_dim)));
        Matrix matrix1_scaled = this->shtrassen_extend(max_dim2d);
        Matrix matrix2_scaled = other.shtrassen_extend(max_dim2d);
        std::vector<std::vector<int>> vals =
           strassenMultiply(matrix1_scaled._table,
           matrix2_scaled._table);
        Matrix mat(this->_n, other._m);
        mat._table = vals;
        return mat;
    }
```

Листинг А.6 – Метод расширения матрицы до квадартной матрицы с заданной размерностью

```
    Matrix Matrix::shtrassen_extend(std::size_t new_dim) const
    {

        Matrix res_matrix = Matrix(new_dim, new_dim);
        for (size_t i = 0; i < this->_n; i++)
        for (size_t j = 0; j < this->_m; j++)
        res_matrix._table[i][j] = this->_table[i][j];

        return res_matrix;
    }
```