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»

Отчёт по лабораторной работе №2 по курсу «Анализ Алгоритмов»

Студент Паламарчук А.Н.

Группа ИУ7-53Б

Преподаватель Кормановский М.В.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является исследование стандартного алгоритма умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

— разработать алгоритм стандартного умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда;

— разработать программное обеспечение, содержащее алгоритм стандартного умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда;

— провести функциональное тестирование реализованных алгоритмов;

— провести сравнительный анализ алгоритмов.

1 Аналитическая часть

1.1 Матрица

Матрицей A размера $m \times n$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы [1].

Умножение матрицы A на матрицу B определено, лишь когда число столбцов первой матрицы в произведении равно числу строк второй. Тогда произведением матриц $\underset{m\times k}{A}\ \underset{k\times n}{B}$ называется матрица $\underset{m\times n}{C}$, каждый элемент которой c_{ij} равен сумме попарных произведений элементов i–й строки матрицы A на соответствующие элементы j–го столбца матрицы B [1].

1.2 Стандартный алгоритм умножения матриц

Стандартный алгоритм умножения матриц является одним из базовых методов, используемых для вычисления произведения двух матриц. Пусть даны две матрицы A размером $m \times k$ и B размером $k \times n$. Результатом их умножения будет матрица C размером $m \times n$.

Стандартный алгоритм умножения матриц можно описать следующим образом. Инициализация — создается матрица C размером $m \times n$, и все её элементы инициализируются нулями. Это необходимо для того, чтобы избежать случайных значений в результирующей матрице. Вложенные циклы — для вычисления каждого элемента матрицы C используются три вложенных цикла, внешний цикл проходит по строкам матрицы A (индекс i), средний цикл проходит по столбцам матрицы B (индекс j), внутренний цикл проходит по элементам строки матрицы A и столбца матрицы B (индекс k), вычисляя сумму произведений соответствующих элементов.

$$C[i][j] = \sum_{k=0}^{k-1} A[i][k] \times B[k][j] \tag{1.1}$$

1.3 Алгоритм Винограда

Алгоритм Винограда основан на использовании предварительных вычислений для уменьшения количества необходимых операций умножения. В отличие от стандартного алгоритма, который требует $O(m \cdot n \cdot p)$ операций умножения для умножения матриц A размером mp и B размером pn, алгоритм Винограда снижает это количество до $O(m \cdot n + m + n)$, что делает его более эффективным для больших матриц.

Алгоритм начинает с вычисления промежуточных значений, которые позволяют сократить количество операций умножения. Для матриц A и B вычисляются два массива, для строк матрицы A:

$$P[i] = \sum_{k=0}^{k-1} A[i][k] \cdot B[k][j] \tag{1.2}$$

Для столбцов матрицы B:

$$Q[j] = \sum_{k=0}^{k-1} A[i][k] \cdot B[k][j] \tag{1.3}$$

После вычисления промежуточных значений, алгоритм использует их для вычисления элементов результирующей матрицы C. Каждый элемент $C[i][j]$ вычисляется как:

$$C[i][j] = P[i] + Q[j] \tag{1.4}$$

Затем выполняется сложения промежуточных значений, что позволяет избежать повторных вычислений и значительно ускоряет процесс умножения.

1.4 Оптимизированный алгоритм Винограда

Индивидуальный вариант: инкремент счётчика наиболее вложенного цикла на 2; использование инкремента (+=); введение декремента при вычислении вспомогательных массивов;

2 Конструкторская часть

2.1 Описание используемых структур данных

При реализации алгоритмов будут использованы следующие структуры данных:

— матрица — тип массив;

— размер матрицы — целочисленный тип.

2.2 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены соответственно стандартный алгоритм умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда.

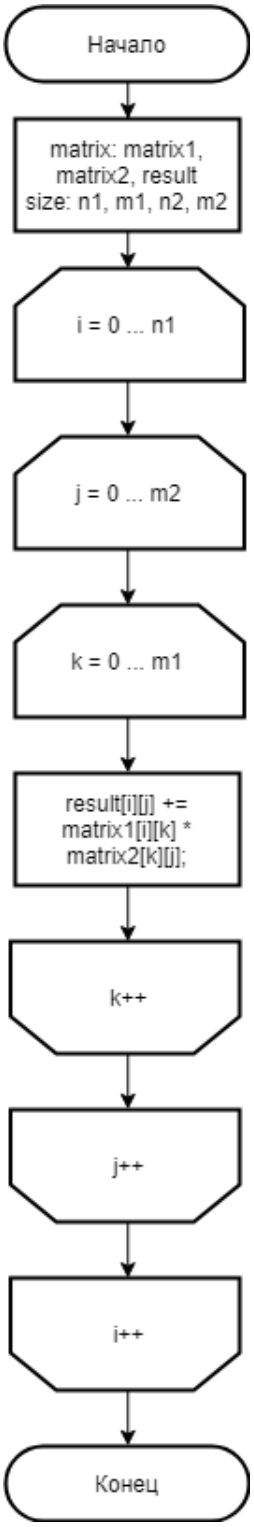

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

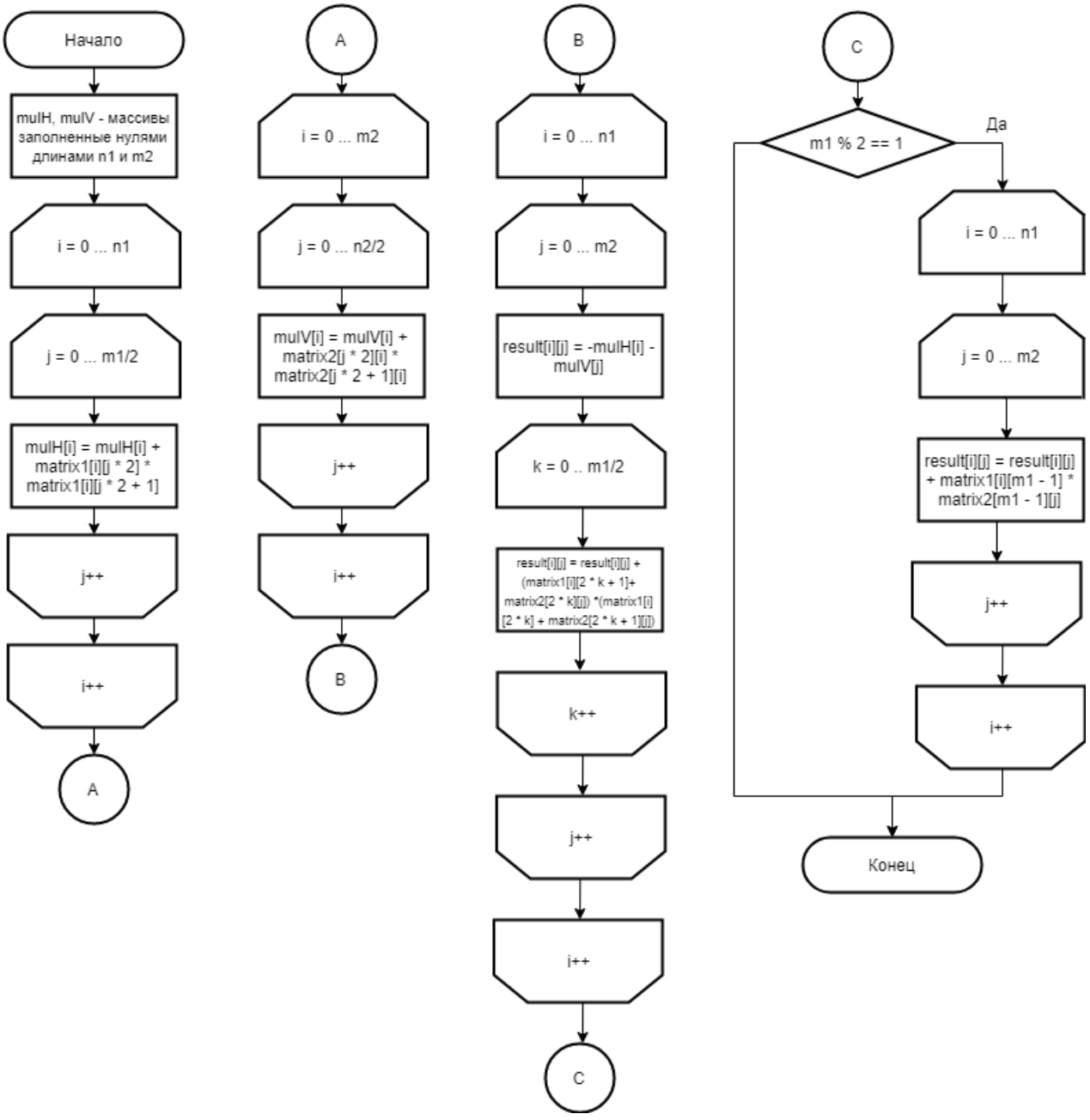

Рисунок 2.2 – Схема алгоритма Винограда

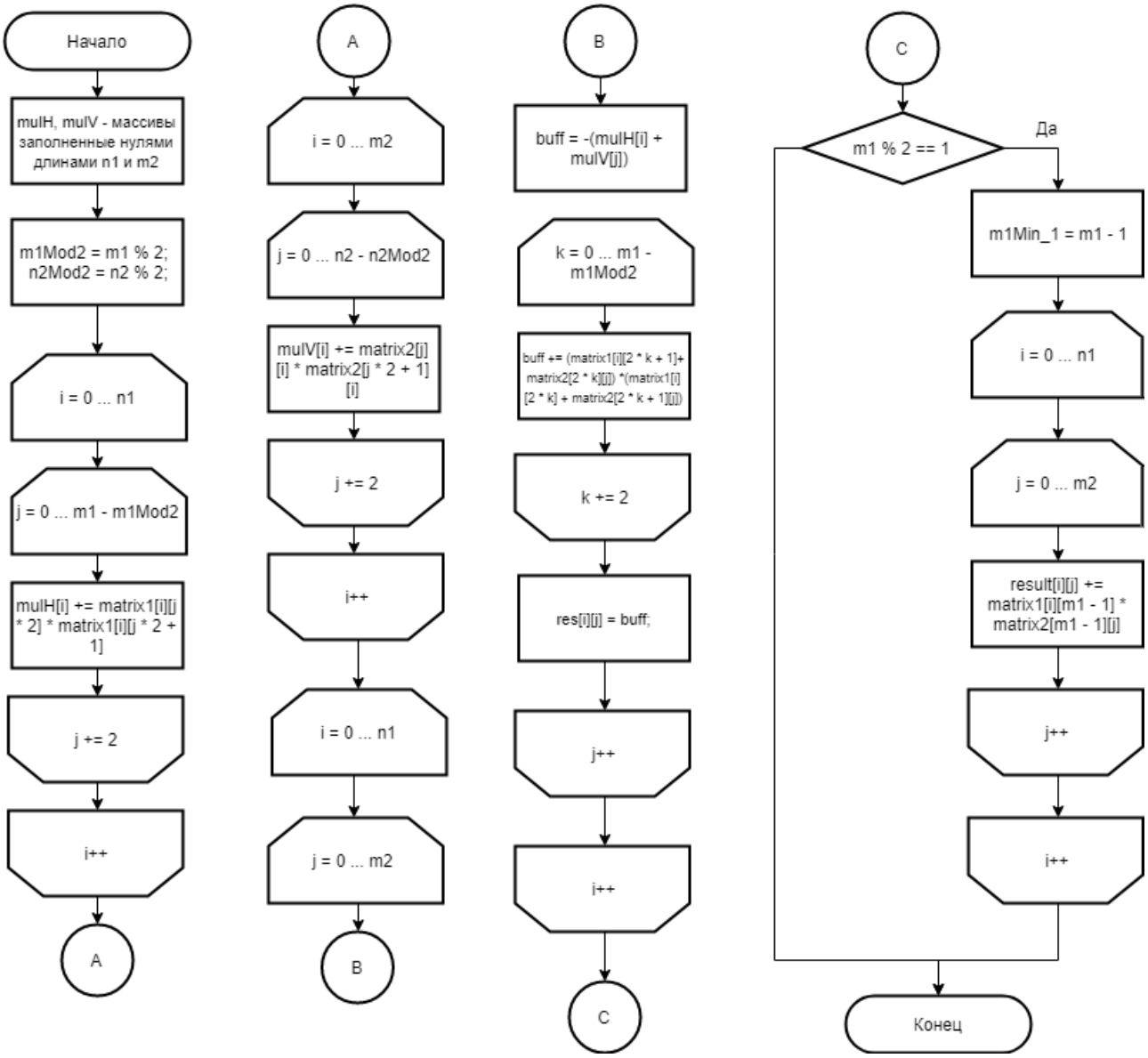

Рисунок 2.3 – Схема оптимизированного алгоритма Винограда

3 Технологическая часть

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python*. Требуется измерить затрачиваемое время и построить графики. Для построения графиков использовалась библиотека *mathplotlib*.

3.2 Реализация алгоритмов

В листингах 3.1, 3.2 и 3.3 представлены реализации алгоритма стандартного умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
1  def mul_std(a: Matrix, b: Matrix):
2      if (a.size != b.size):
3          return ERROR_DIFFERENT_SIZE
4      size = a.size
5      res = Matrix(size, Gen.ZERO_MATRIX)
6      for i in range(size):
7          for j in range(size):
8              for k in range(size):
9                  res.data[i][j] += a.data[i][k] * b.data[k][j]
10     return res
```

Листинг 3.2 – Алгоритм Винограда

```
def mul_grape(a: Matrix, b: Matrix):
    if (a.size != b.size):
        return ERROR_DIFFERENT_SIZE

    size = a.size
    row_a = size
    col_a = size
    col_b = size
    tmp_row = [0] * row_a
    tmp_col = [0] * col_b

    for i in range(row_a):
        for j in range(0, col_a // 2):
            tmp_row[i] = tmp_row[i] + a.data[i][2 * j] *
               a.data[i][2 * j + 1]

    for i in range(col_b):
        for j in range(0, col_a // 2):
            tmp_col[i] = tmp_col[i] + b.data[2 * j][i] * b.data[2 *
               j + 1][i]

    res = Matrix(size, Gen.ZERO_MATRIX)
    for i in range(row_a):
        for j in range(col_b):
            res.data[i][j] = -tmp_row[i] - tmp_col[i]
            for k in range(0, col_a // 2):
                res.data[i][j] = res.data[i][j] + (a.data[i][2 * k
                   + 1] + b.data[2 * k][j]) * (a.data[i][2 * k] +
                   b.data[2 * k + 1][j])

    if (col_a % 2 == 1):
        for i in range(row_a):
            for j in range(col_b):
                res.data[i][j] = res.data[i][j] + a.data[i][col_a -
                   1] * b.data[col_a - 1][j]

    return res
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
1  def mul_grape_opt(a: Matrix, b: Matrix):
2      if (a.size != b.size):
3          return ERROR_DIFFERENT_SIZE
4
5      size = a.size
6      row_a = size
7      col_a = size
8      col_b = size
9      tmp_row = [0] * row_a
10     tmp_col = [0] * col_b
11
12     for i in range(row_a):
13         for j in range(0, col_a // 2):
14             tmp_row[i] = tmp_row[i] + a.data[i][2 * j] *
                   a.data[i][2 * j + 1]
15
16     for i in range(col_b):
17         for j in range(0, col_a // 2):
18             tmp_col[i] = tmp_col[i] + b.data[2 * j][i] * b.data[2 *
                   j + 1][i]
19
20     flag = col_a % 2
21     res = Matrix(size, Gen.ZERO_MATRIX)
22     for i in range(row_a):
23         for j in range(col_b):
24             res.data[i][j] -= tmp_row[i] + tmp_col[i]
25             for k in range(1, col_a, 2):
26                 res.data[i][j] += (a.data[i][k - 1] + b.data[k][j])
                       * (a.data[i][k] + b.data[k - 1][j])
27             if (flag):
28                 res.data[i][j] += a.data[i][col_a - 1] *
                       b.data[col_a - 1][j]
29
30     return res
```

3.3 Функциональные тесты

В таблице 3.1 приведены тесты для функций программы. Тесты для всех функций пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица A	Матрица B	Ожидание	Результат
$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 12 & 15 & 18 \\ 12 & 15 & 18 \\ 12 & 15 & 18 \end{pmatrix}$	$\begin{pmatrix} 12 & 15 & 18 \\ 12 & 15 & 18 \\ 12 & 15 & 18 \end{pmatrix}$
$\begin{pmatrix} 0 & 1 \\ 7 & 9 \end{pmatrix}$	$\begin{pmatrix} 8 & 4 \\ 2 & 4 \end{pmatrix}$	$\begin{pmatrix} 2 & 4 \\ 74 & 64 \end{pmatrix}$	$\begin{pmatrix} 2 & 4 \\ 74 & 64 \end{pmatrix}$
$\begin{pmatrix} -2 & -1 & 0 \\ 3 & 2 & 1 \\ -1 & 2 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 & -1 & 1 \\ 2 & 0 & 3 \\ 1 & 2 & 0 \end{pmatrix}$	$\begin{pmatrix} -4 & 2 & -5 \\ 8 & -1 & 9 \\ 1 & -3 & 5 \end{pmatrix}$	$\begin{pmatrix} -4 & 2 & -5 \\ 8 & -1 & 9 \\ 1 & -3 & 5 \end{pmatrix}$
$\begin{pmatrix} -4 & -1 & -2 \\ -2 & -1 & -3 \\ -2 & -3 & -1 \end{pmatrix}$	$\begin{pmatrix} 3 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 4 & 3 \end{pmatrix}$	$\begin{pmatrix} -15 & -13 & -16 \\ -10 & -15 & -15 \\ -10 & -9 & -13 \end{pmatrix}$	$\begin{pmatrix} -15 & -13 & -16 \\ -10 & -15 & -15 \\ -10 & -9 & -13 \end{pmatrix}$

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства:

— операционная система Manjaro Linux x86_64;

— процессор Ryzen 5500U 6 ядер, тактовая частота 2.1 ГГц;

— оперативная память 16 Гбайт.

При тестировании ноутбук был включён в сеть электропитания. Во время тестирования ноутбук был нагружен только системными приложениями окружения, а также системой тестирования.

4.2 Сравнительный анализ временных затрат

Замеры времени для каждой размерности матрицы проводились 100 раз. Результат замера — среднее арифметическое время работы алгоритма, на вход подавались сгенерированные случайным образом матрицы.

На рисунках 4.1, 4.2 и 4.3 представлено сравнение временных затрат для стандартного алгоритма умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда (общий случай, размерность чётная, размерность нечётная).

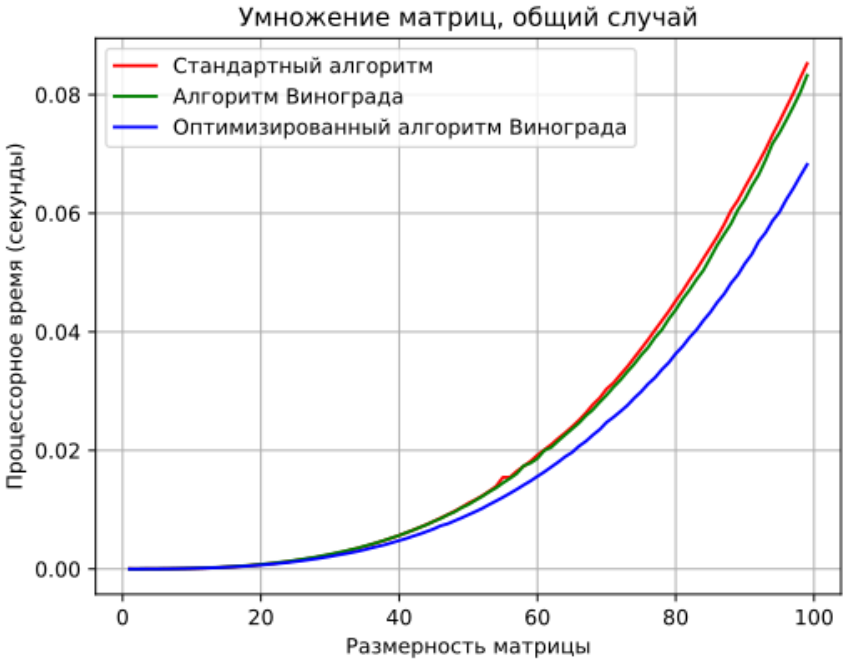

Рисунок 4.1 – Временные затраты, общий случай

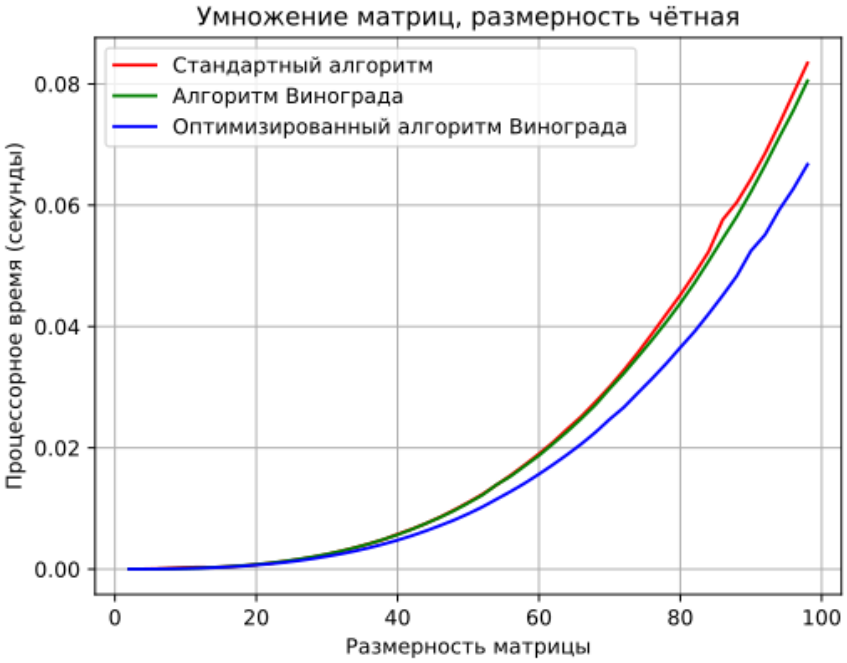

Рисунок 4.2 – Временные затраты, размерность чётная

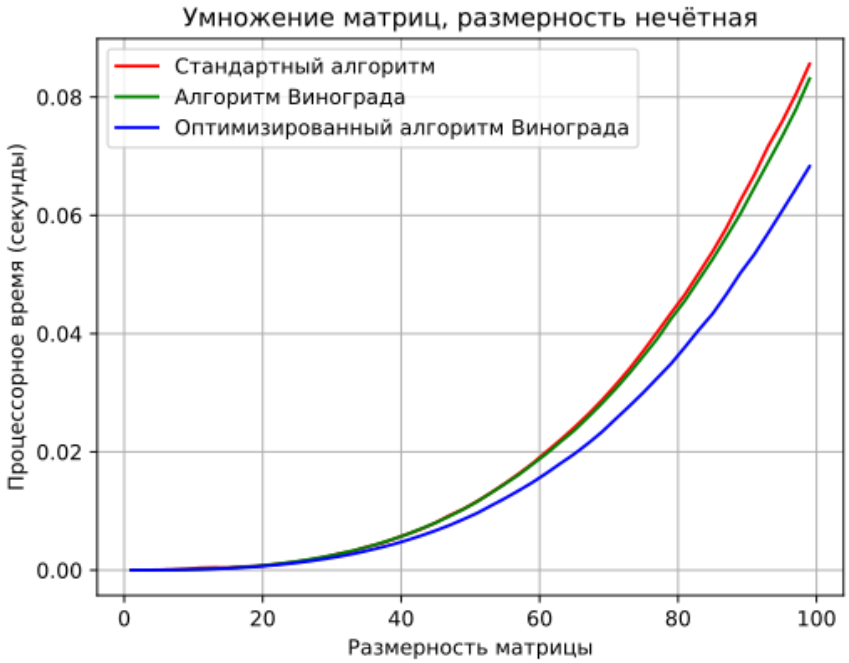

Рисунок 4.3 – Временные затраты, размерность нечётная

4.3 Результаты проведенных исследований

По полученным данным измерений временных затрат был сделан вывод о том, что алгоритм Винограда и оптимизированный алгоритм Винограда, затрачивают меньше времени для обработки по сравнению со стандартным алгоритмом умножения матриц. При увеличении размерности матриц это разрыв становится существеннее.

ЗАКЛЮЧЕНИЕ

В результате исследования было определено, что по временным затратам алгоритм Винограда и оптимизированный алгоритм Винограда эффективнее стандартного алгоритма умножения матриц и с увеличением размерности матриц разрыв во времени работы увеличивается.

Цель достигнута, были исследованы стандартный алгоритм умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда, а также в ходе выполнения лабораторной работы были решены следующие задачи:

— разработаны алгоритм стандартного умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда;

— разработано программное обеспечение, содержащее алгоритм стандартного умножения матриц, алгоритм Винограда, оптимизированный алгоритм Винограда;

— проведено функциональное тестирование реализованных алгоритмов;

— проведен сравнительный анализ алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Белоусов И. В. МАТРИЦЫ и ОПРЕДЕЛИТЕЛИ: учебное пособие по линейной алгебре. второе, исправленное и дополненное изд. Кишинев: Институт прикладной физики, Академии наук Республики Молдова, 2006.