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

Название: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент	ИУ7-52Б		Д.В. Батраков
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподователь			Л.Л. Волкова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Содержание

Введение . . . 3
1 Аналитический раздел . . . 4
1.1 Алгоритмы умножения матриц . . . 4
1.1.1 Классический алгоритм умножения . . . 4
1.1.2 Алгоритм Винограда . . . 4
1.1.3 Вывод . . . 5
1.2 Трудоёмкость алгоритма . . . 5
1.2.1 Базовые операции . . . 5
1.2.2 Условный оператор . . . 5
1.2.3 Цикл со счётчиком . . . 6
2 Констукторский раздел . . . 7
2.1 Разработка алгоритмов . . . 7
2.2 Требования к функциональности ПО . . . 7
2.3 Тесты . . . 7
3 Технологический раздел . . . 11
3.1 Средства реализации . . . 11
3.2 Листинг программы . . . 11
3.3 Тестирование . . . 13
4 Экспериментальный раздел . . . 15
4.1 Сравнительный анализ на основе замеров времени работы алгоритмов . . . 15
4.2 Оценка трудоёмкости алгоритмов умножения матриц . . . 15
4.2.1 Стандартный алгоритм . . . 15
4.2.2 Алгоритм Винограда . . . 15
4.2.3 Оптимизированный алгоритм Винограда . . . 16
4.3 Вывод . . . 16
Заключение . . . 19
Список использованных источников . . . 20

Введение

Умножение матриц – это одна из самых распространённых операций над матрицами, которая широко применяется в различных численных методах, например, в приложениях для решения системы линейных алгебраических уравнений, в программах для преобразований графических структур данных и многих других задачах.

В данной работе требуется изучить и применить три алгоритма умножения матриц:

1) стандартный алгоритм умножения матриц;
2) алгоритм Винограда;
3) оптимизированный алгоритм Винограда.

Цель лабораторной работы – провести сравнительный анализ алгоритмов умножения матриц и получить навыки оптимизации трудоёмкости алгоритмов.

В лабораторной работе ставятся следующие задачи:

1) дать математическое описание формул расчёта умножения матриц для стандарного алгоритма и Винограда;
2) разработать оптимизированный алгоритм Винограда;
3) реализовать стандартный алгоритм умножения матриц, Винограда и оптимизированного Винограда;
4) дать теоритическую оценку трудоёмкости трёх алгоритмов;
5) провести замеры процессорного времени работы реализаций трёх алгоритмов в худшем и в лучшем случаях.

1 Аналитический раздел

В данном разделе будут рассмотрены основные теоритические понятия алгоритмов умножения матриц.

1.1 Алгоритмы умножения матриц

1.1.1 Классический алгоритм умножения

Пусть даны две прямоугольные матрицы A размерности MxN и B размерности NxQ

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nq} \end{bmatrix}$$

тогда произведением матриц A и B называется матрица C вида:

$$C = AB = \begin{bmatrix} c_{11} & \cdots & c_{1q} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mq} \end{bmatrix}, \tag{1.1}$$

где $_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj}$

Классический алгоритм умножения матриц находит матрицу C по определению.

1.1.2 Алгоритм Винограда

Шмуэль Виноград предложил алгоритм умножения матриц, в котором используется меньше операций умножения в сравнении с классической реализацией, и, следовательно, теоритически быстрее, так как умножение – долгая операция.

Рассмотрим два вектора $U = (u_1, u_2, u_3, u_4)$ и $V = (v_1, v_2, v_3, v_4)^T$. Их произведение равно

$$UV = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4 \tag{1.2}$$

Сократим долю умножения среди всех операций – сгруппируем слагаемые на пары, тогда выражение (1.2) будет иметь вид

$$UV = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4 \tag{1.3}$$

В случаи если длина векторов будет нечётной: $U = (u_1, u_2, u_3)$ и $V = (v_1, v_2, v_3)^T$, выражение (1.3) примет следующий вид (1.4):

$$UV = (u_1 + v_2)(u_2 + v_1) - u_1u_2 - v_1v_2 + v_3u_3 \tag{1.4}$$

Выражение (1.2) требует большего количества операций, чем выражение (1.4) – вместо четырёх умножений – шесть, вместо трёх сложений - пять, оно допускает предварительную обработку.

Правую часть можно вычислить заранее и хранить для каждой строки первой матрицы и для каждого столбца второй матрицы, что позволяет выполнять для каждого элемента лишь два умножения и семь сложений.

1.1.3 Вывод

Алгоритм Винограда отличается от классического алгорима умножения матриц меньшим количеством операций умножения, за счёт предварительной обработки строк и столбцов матриц.

1.2 Трудоёмкость алгоритма

Трудоёмкость – количество работы, которую алгоритм затрачивает на обработку данных. Является функцией от длины входов алгоритма и позволяет оценить количество работы.

Введём модель вычисления трудоёмкости.

1.2.1 Базовые операции

Ниже представлены базовые операции, стоимость которых равна единице:

1) $=, +, +=, -, -=, ++, --,$,
2) $<, \leqslant, ==, \neq, \geqslant, >$,
3) [ ],
4) $<<, >>$.

Ниже представлены базовые операции, стоимость которых равна двум:

1) *, *=, /, /=, %

.

1.2.2 Условный оператор

if (условие) {
// тело A
}
else {
// тело B
}

Пусть трудоёмкость тела A равна f_A, а тела B f_B, тогда стоимость условного оператора можно найти по формуле (1.5):

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B) - \text{лучший случай}, \\ max(f_A, f_B) - \text{худший случай} \end{cases} \tag{1.5}$$

1.2.3 Цикл со счётчиком

```
for (int i = 0; i < n; i++) {
// тело цикла
}
```

Начальная инициализация цикла (int i = 0) выполняется один раз. Условие i < n проверяется перед каждой итерацией цикла и при входе в цикл – n + 1 операций. Тело цикла выполняется ровно n раз. Счётчик (i++) выполняется на каждой итерации, перед проверкой условия, т.е. n раз. Тогда, если трудоёмкость тела цикла равна f, трудоёмкость всего цикла определяется формулой (1.6)

$$f_{\text{цикла}} = 2 + n(2 + f) \tag{1.6}$$

2 Констукторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и опредены способы тестирования.

2.1 Разработка алгоритмов

Ниже будут представлены схемы алгоритмов умножения матриц:

1) классического (рисунок 2.1);
2) Винограда (рисунок 2.2);
3) оптимизированного Винограда (рисунок 2.3).

Для уменьшения трудоёмкости алгоритма Винограда сделаем следующие действия:

1) замена в цикле условии деления на 2 на цикл с шагом 2
2) замена a = a + ..., на a += ...
3) вычисление суммы отрицательной при заполнении row и col

2.2 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность консольного приложения:

1) возможность ввода двух матриц, на выходе результат произведения данных матриц? посчитанный трёмя алгоритмами;

2) возможность вывода результатов замера процессорного времени работы реализаций каждого из алгоритмов.

2.3 Тесты

Тестирование ПО будет проводиться методом чёрного ящика. Необходимо проверить работу системы на тривиальных случаях (одна матрица единичная или нулевая) и несколько нетривальных случаев.

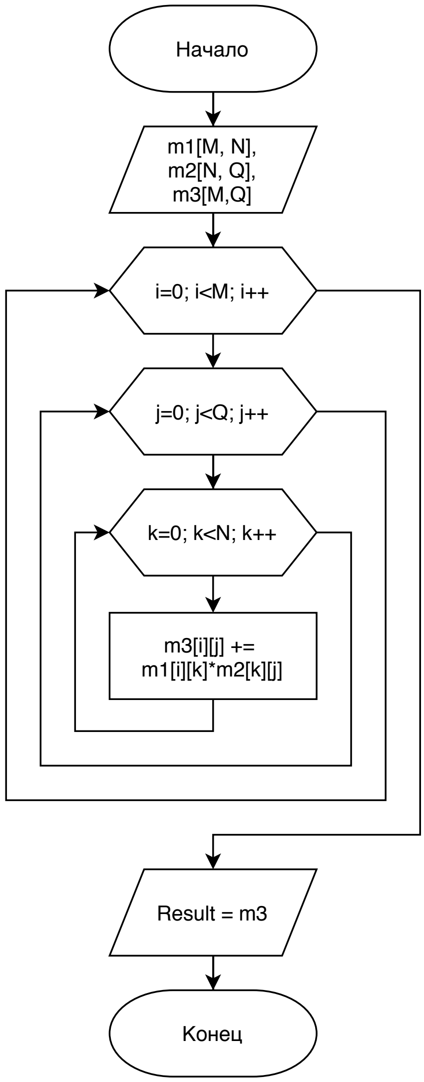

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

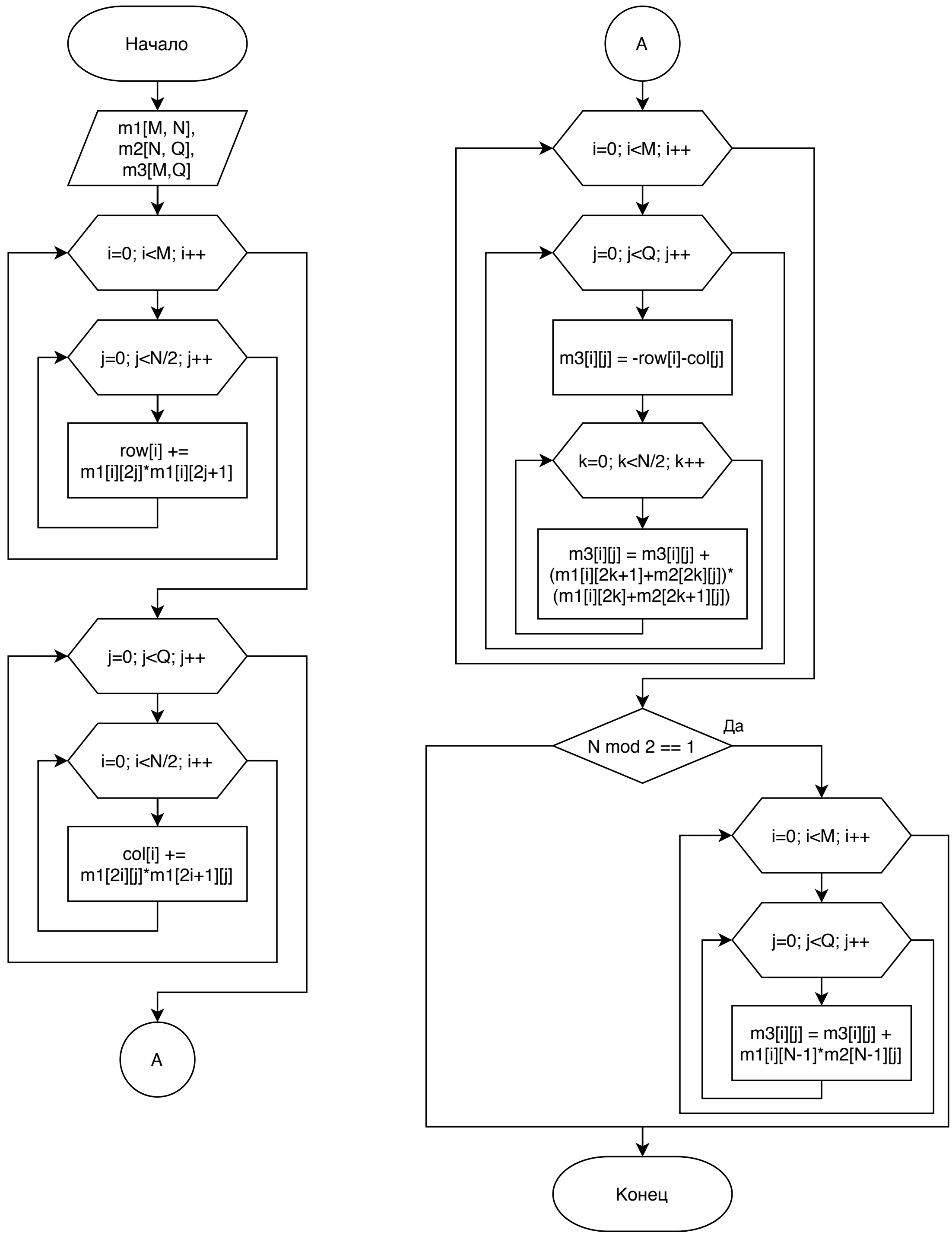

Рисунок 2.2 — Схема алгоритма умножения матриц методом Винограда

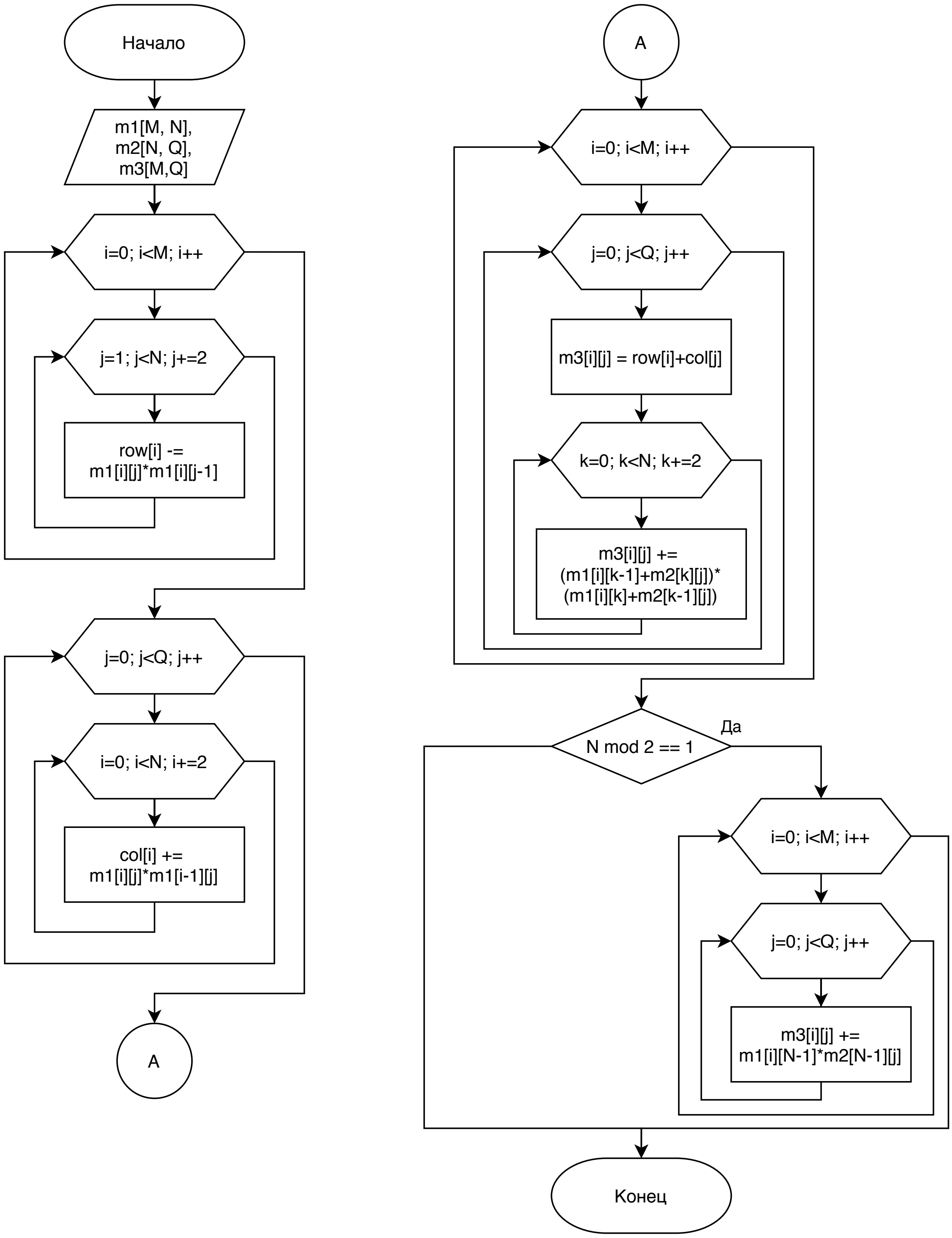

Рисунок 2.3 — Схема оптимизированный алгоритм умножения матриц методом Винограда

3 Технологический раздел

В данном разделе будут выбраны средства реплизации ПО, представлен листинг кода и проведён теоритический анализ максимальной затрачиваемой памяти.

3.1 Средства реализации

В данной работе используется язык программирования python [1], так как он позволяет написать программу в относительно малый срок. В качестве среды разработки использовалась Visual Studio Code [2].

Для замера процессорного времени была использована функция process_time [3] модуля time. Она возвращает значение в долях секунды суммы системного и пользовательского процессорного времени текущего процесса и не включает время, прошедшее во время сна.

3.2 Листинг программы

Ниже представлены листинги кода умножения матриц:

1) стандартной реализации (листинг 3.1);
2) реализация алгоритма Винограда (листинг 3.2);
3) реализация оптимизированного алгоритма Винограда (листинг 3.3).

Листинг 3.1 — Реализация классического алгоритма умножения матриц

```
def dotMatrix(matr_a : list, matr_b: list) -> (list, float):
    if (len(matr_b) != len(matr_a[0])):
        raise ValueError

    m = len(matr_a)
    n = len(matr_a[0])
    q = len(matr_b[0])
    matr_c = [[0] * q for i in range(m)]

    t_start = process_time()
    for i in range(m):
        for j in range(q):
            for k in range(n):
                matr_c[i][j] = matr_c[i][j] + matr_a[i][k] * matr_b[k][j]
    t_end = process_time()

    return matr_c, t_end - t_start
```

Листинг 3.2 — Реализация алгоритма Винограда умножения матриц

```
def dotMatrixVinograd(matr_a : list, matr_b: list) -> (list, float):
    if (len(matr_b) != len(matr_a[0])):
        raise ValueError
```

```

    m = len(matr_a)
    n = len(matr_a[0])

    q = len(matr_b[0])
    matr_c = [[0] * q for i in range(m)]

    row = [0] * m
    col = [0] * q

    t_start = process_time()

    for i in range(m):
        for j in range(n // 2):
            row[i] = row[i] + matr_a[i][2*j] * matr_a[i][2*j + 1]

    for j in range(q):
        for i in range(n // 2):
            col[j] = col[j] + matr_b[2*i][j] * matr_b[2*i+1][j]

    for i in range(m):
        for j in range(q):
            matr_c[i][j] = -row[i] - col[j]
            for k in range(n//2):
                matr_c[i][j] = matr_c[i][j] + (matr_a[i][2*k+1] + matr_b[2*k][j]) * (matr_a[i][2*k] + matr_b[2*k+1][j])

    if n % 2:
        for i in range(m):
            for j in range(q):
                matr_c[i][j] = matr_c[i][j] + matr_a[i][n-1] * matr_b[n-1][j]
    t_end = process_time()

    return matr_c, t_end - t_start
```

Листинг 3.3 — Реализация оптимизированного алгоритма Винограда умножения матриц

```
def dotMatrixVinogradOptimizate(matr_a : list , matr_b: list) -> (list , float):
    if (len(matr_b) != len(matr_a[0])):
        raise ValueError

    m = len(matr_a)
    n = len(matr_a[0])
    q = len(matr_b[0])
    matr_c = [[0] * q for i in range(m)]

    row = [0] * m
```

```
    col = [0] * q
    t_start = process_time()
    for i in range(m):
        for j in range(1, n, 2):
            row[i] -= matr_a[i][j] * matr_a[i][j - 1]

    for j in range(q):
        for i in range(1, n, 2):
            col[j] -= matr_b[i][j] * matr_b[i - 1][j]

    for i in range(m):
        for j in range(q):
            matr_c[i][j] = row[i] + col[j]
            for k in range(1, n, 2):
                matr_c[i][j] += (matr_a[i][k - 1] + matr_b[k][j]) * (matr_a[i][k] + matr_b[k-1][j])

    if n % 2:
        for i in range(m):
            for j in range(q):
                matr_c[i][j] += matr_a[i][n-1] * matr_b[n-1][j]
    t_end = process_time()

    return matr_c, t_end - t_start
```

3.3 Тестирование

В таблице 3.1 отображён возможный набор тестов для тестирования методом чёрного ящика, результаты которого, представленные на рисунке 3.1, подтверждают прохождение программы перечисленных тестов.

Таблица 3.1 — Тесты для проверки корректности программы

Матрица A	Матрица B	Ожидаемый результат
$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	the matrices cannot be multiplied

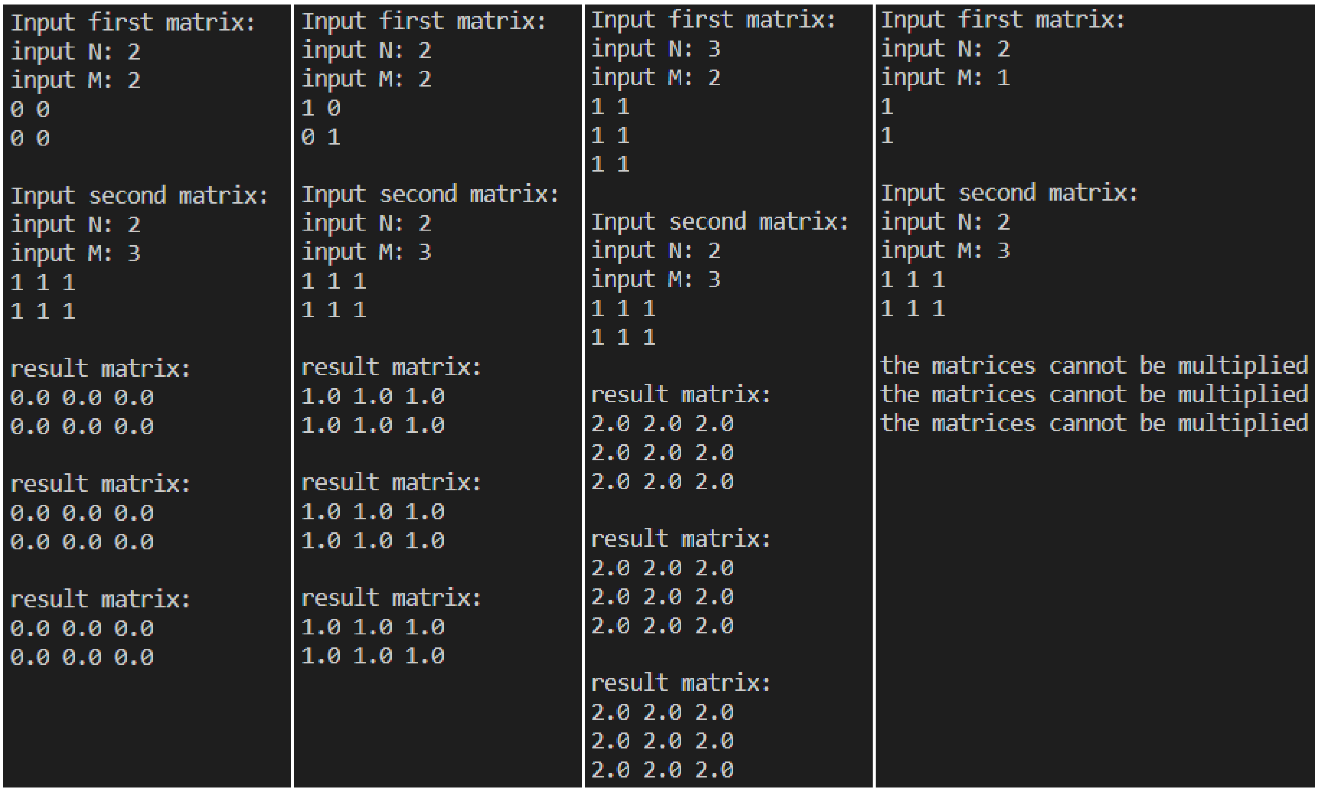

Рисунок 3.1 — Результаты тестирования алгоритмов: стандартного, Винограда и оптимизированного Винограда

4 Экспериментальный раздел

В данном разделе будут проведены эксперименты для проведения сравнительного анализа трёх алгоритмов по затрачиваемому процессорному времени в зависимости от размеров матриц и чётности / нечётности размеров.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данного проекта были проведёны следующие эксперименты:

1) сравнение времени работы алгоритмов на размерностях квадратных матриц 100, 200, 300, 400, 500 (график 4.1);
2) сравнение времени работы алгоритмов на размерностях квадратных матриц 101, 201, 301, 401, 501 (график 4.2).

Матрицы заполнялись случайными числами.

Тестирование проводилось на ноутбуке с процессором Intel(R) Core(TM) i5-7200U CPU 2.50 GHz [4] под управлением Windows 10 с 8 Гб оперативной памяти.

В ходе экспериментов по замеру времени работы было установлено, что оптимизированный алгоритм Винограда быстрее неоптимизированого на 43 % и на 14-20 % стандартного в зависимости от чётности совпадающей размерности матриц.

4.2 Оценка трудоёмкости алгоритмов умножения матриц

4.2.1 Стандартный алгоритм

Найдём трудоёмкость стандартного алгоритма.

$f_{\text{первый цикл}} = 2 + M(2 + f_{\text{второй цикл}})$

$f_{\text{второй цикл}} = 2 + Q(2 + f_{\text{третий цикл}})$

$f_{\text{третий цикл}} = 2 + N(2 + 12)$

$f_{\text{Стандартный}} = 14MNQ + 4MQ + 4M + 2 \approx 13MNQ$

4.2.2 Алгоритм Винограда

Найдём трудоёмкость алгоритма Винограда.

$f_{\text{первый цикл}} = 2 + M(2 + 2 + 2 + \frac{N}{2}(4 + 15)) = \frac{19}{2}MN + 6M + 2$

$f_{\text{второй цикл}} = \frac{19}{2}QN + 6Q + 2$

$f_{\text{третий цикл}} = 2 + M(2 + 2 + Q(2 + 7 + 4 + \frac{N}{2}(4 + 26))) = 15MNQ + 13MQ + 4M + 2$

Условный переход $f_{if} = 2 + \begin{cases} 0 - \text{лучший случай}, \\ 16QM + 4M + 2 - \text{худший случай} \end{cases}$

Итого:

$$f_{\text{Винограда}} = 15MNQ + 13MQ + \frac{19}{2}(MN + QN) + 6(M + Q) + 4M + 8 + \begin{cases} 0 - \text{л.с.}, \\ 16MQ + 4M + 2 - \text{х.с.} \end{cases} \tag{4.1}$$

$f_{\text{Винограда}} \approx 15MNQ$

4.2.3 Оптимизированный алгоритм Винограда

Найдём трудоёмкость оптимизированного алгоритма Винограда.

$f^*_{\text{первый цикл}} = 2 + M(2 + 2 + \frac{N}{2}(2 + 9)) = 6MN + 4M + 2$

$f^*_{\text{второй цикл}} = 6QN + 4M + 2$

$f^*_{\text{третий цикл}} = 2 + M(2 + 2 + Q(2 + 6 + 2 + \frac{N}{2}(2 + 17))) = 10MNQ + 10MQ + 4M + 2$

Условный переход $f^*_{if} = 2 + \begin{cases} 0 - \text{лучший случай}, \\ 13QM + 4M + 2 - \text{худший случай} \end{cases}$

Итого:

$$f^*_{\text{Винограда}} = 10MNQ + 10MQ + \frac{11}{2}(MN + QN) + 6(M + Q) + 4M + 8 + \begin{cases} 0 - \text{л.с.}, \\ 13MQ + 4M + 2 - \text{х.с.} \end{cases} \quad (4.2)$$

$f^*_{\text{Винограда}} \approx 10MNQ$

4.3 Вывод

Несмотря на сложность алгоритма Винограда по сравнению со стандартным, доля операций умножения в нём меньше. Стоит отметить, что алгоритм Винограда имеет худший (матрицы совпадающей нечётной размерности – количество строк матрицы A и столбцов матрицы B) и лучший случаи (матрицы совпадающей чётной размерности – количество строк матрицы A и столбцов матрицы B), в то время как стандартный алгоритм не зависит от чётности совпадающей размерности матриц.

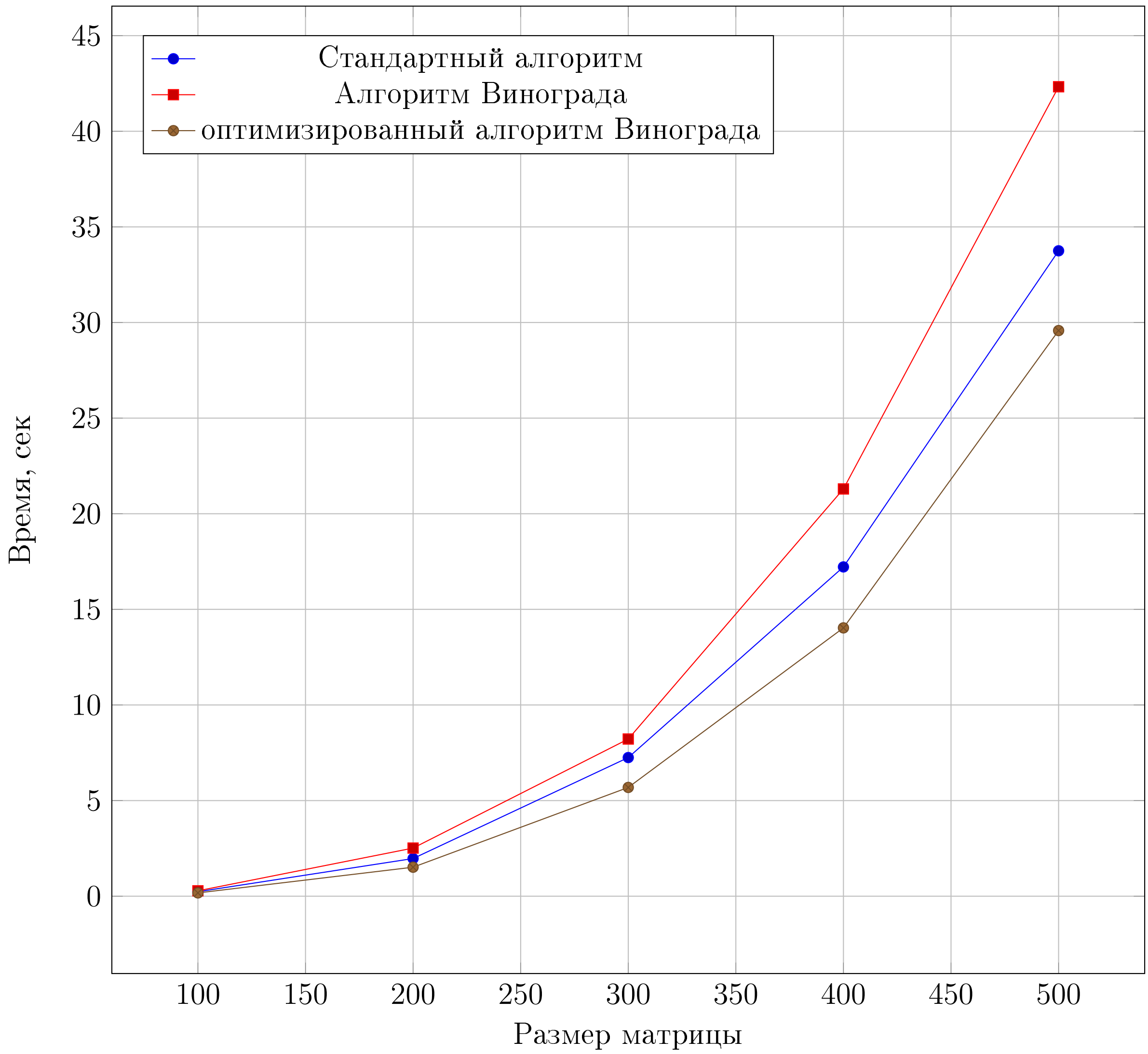

Рисунок 4.1 — График зависимости времени работы алгоритмов при чётных размерностях матриц

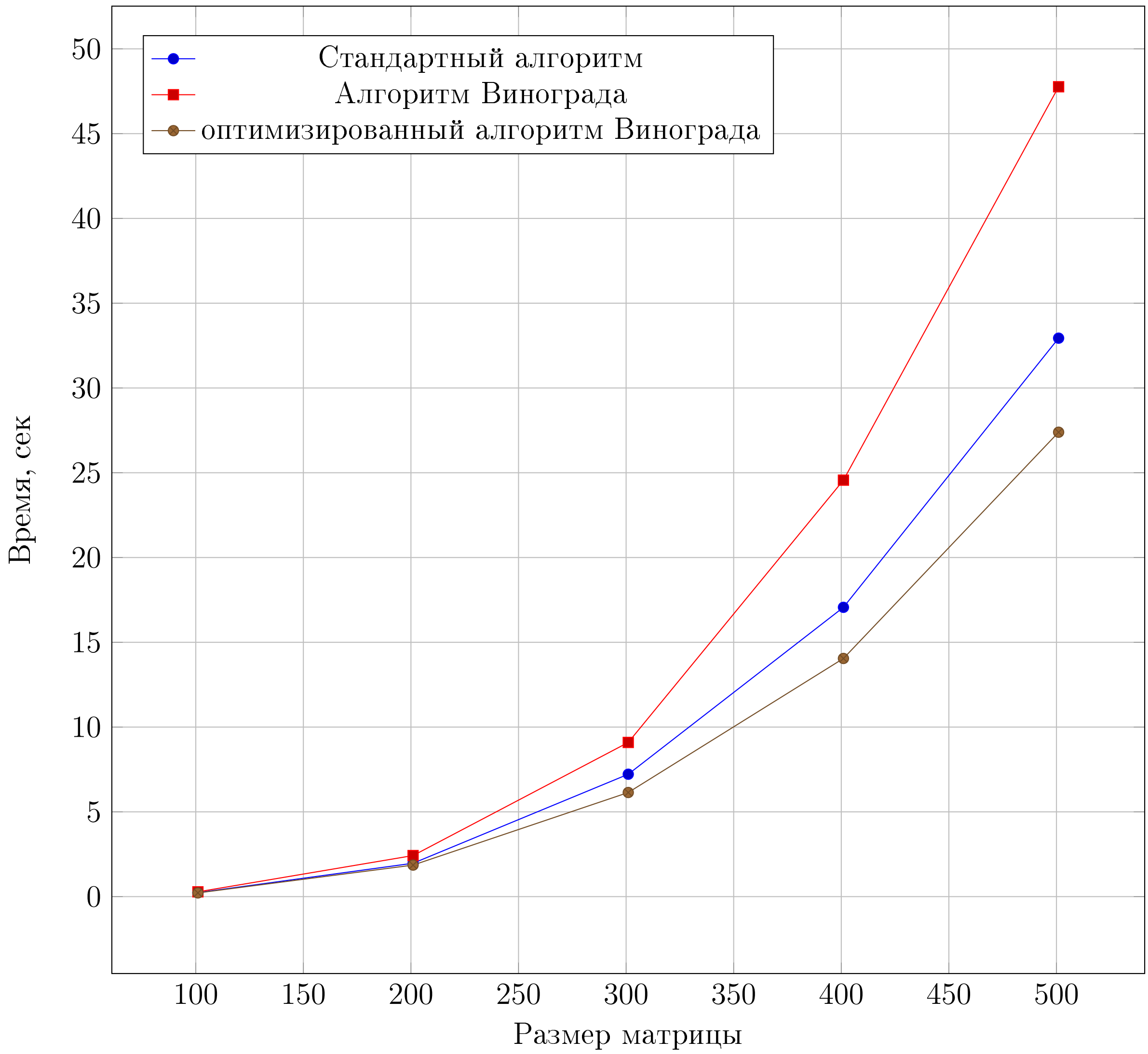

Рисунок 4.2 — График зависимости времени работы алгоритмов при нечётных размерностях матриц

Заключение

В ходе работы были изучены и реализованы алгоритмы умножения матриц (стандартный, Винограда и оптимизированный Винограда), дано математическое описание формул расчёта умножения матриц для стандарного алгоритма и Винограда, и проведена оптимизация алгоритма Винограда.

В ходе экспериментов по замеру времени работы было установлено, что оптимизированный алгоритм Винограда быстрее неоптимизированого на 43 % и на 14-20 % стандартного в зависимости от чётности размерности матриц, что коррелирует с теоритически посчитанной трудоёмкостью алгоритмов.

Список использованных источников

1. Python. // [Электронный ресурс]. Режим доступа: https://www.python.org/, (дата обращения: 01.10.2020).

2. Visual Studio Code - Code Editing. // [Электронный ресурс]. Режим доступа: https://code.visualstudio.com, (дата обращения: 01.10.2020).

3. Process time. // [Электронный ресурс]. Режим доступа: https://docs-python.ru/standart-library/modul-time-python/funktsija-process-time-modulja-time, (дата обращения: 01.10.2020).

4. Intel® Core™ i5-7200U Processor. // [Электронный ресурс]. Режим доступа: https://www.intel.com/content/www/us/en/products/processors/core/i5-processors/i5-7200u.html, (дата обращения: 26.09.2020).