



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 ИУ7-55Б
(Группа)

Д.Р.Жигалкин
(Подпись, дата) (И.О. Фамилия)

Преподаватель

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	5
2 Конструкторский раздел	7
2.1 Разработка алгоритмов	7
2.2 Оценка трудоёмкости алгоритмов умножения матриц	7
2.2.1 Стандартный алгоритм	7
2.2.2 Алгоритм Винограда	7
2.2.3 Оптимизированный алгоритм Винограда	7
2.3 Вывод	8
2.4 Требования к функциональности ПО	8
2.5 Требования к тестированию	8
3 Технологический раздел	12
3.1 Средства реализации	12
3.2 Листинг программы	12
3.3 Тестирование	14
4 Экспериментальный раздел	16
4.1 Сравнительный анализ на основе замеров времени работы алгоритмов	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 стандартного 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 $M \times N$ и B размерности $N \times Q$

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

тогда произведением матриц A и B называется матрица C вида:

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

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

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

$$UV = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4 \quad (1.2)$$

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

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

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):

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

Выражение (1.2) требует большего количества операций, чем выражение (1.2) – вместо четырёх умножений – шесть, вместо трёх сложений – десять, 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 алгоритма 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 единична:

- 1) $=, +, + =, -, - =, *, * =, /, / =, ++, --, \%$,
- 2) $<, \leq, ==, \neq, \geq, >$,
- 3) $[]$.

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):

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

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)

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

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и определены 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 условия деления на 2 на цикл с шагом 2
- 2) замена $a = a + \dots$, на $a += \dots$
- 3) вычисление суммы отрицательной при заполнении row и col

2.2 Оценка трудоёмкости алгоритмов умножения матриц

2.2.1 Стандартный алгоритм

Найдём трудоёмкость стандартного алгоритма.

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

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

$$f_{\text{третий цикл}} = 2 + N(2 + 11)$$

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

2.2.2 Алгоритм Винограда

Найдём трудоёмкость алгоритма Винограда.

$$f_{\text{первый цикл}} = 2 + M(2 + 2 + 3 + \frac{N}{2}(3 + 1 + 6 + 2 + 3)) = \frac{15}{2}MN + 7M + 2$$

$$f_{\text{второй цикл}} = \frac{15}{2}QN + 7Q + 2$$

$$f_{\text{третий цикл}} = 2 + M(2 + 2 + Q(2 + 7 + 3 + \frac{N}{2}(3 + 1 + 12 + 5 + 5))) = 13MNQ + 12MQ + 4M + 2$$

$$\text{Условный переход } f_{if} = 2 + \begin{cases} 0 - \text{лучший случай,} \\ 15QM + 4M + 2 - \text{худший случай} \end{cases}$$

Итого:

$$f_{\text{Винограда}} = 13MNQ + 12MQ + \frac{15}{2}(MN + QN) + 7(M + Q) + 4M + 8 + \begin{cases} 0 - \text{л.с.,} \\ 15MQ + 4M + 2 - \text{х.с.} \end{cases} \quad (2.1)$$

$$f_{\text{Винограда}} \approx 13MNQ$$

2.2.3 Оптимизированный алгоритм Винограда

Найдём трудоёмкость оптимизированного алгоритма Винограда.

$$f_{\text{первый цикл}}^* = 2 + M(2 + 2 + 2 + \frac{N}{2}(2 + 1 + 5 + 1 + 1)) = 10MN + 6M + 2$$

$$f_{\text{второй цикл}}^* = 10MN + 6M + 2$$

$$f_{\text{третий цикл}}^* = 2 + M(2 + 2 + Q(2 + 6 + 2 + \frac{N}{2}(2 + 1 + 10 + 4 + 1))) = 9MNQ + 10MQ + 4M + 2$$

$$\text{Условный переход } f_{if}^* = 2 + \begin{cases} 0 - \text{лучший случай,} \\ 12QM + 4M + 2 - \text{худший случай} \end{cases}$$

Итого:

$$f_{\text{Винограда}}^* = 9MNQ + 10MQ + \frac{15}{2}(MN + QN) + 6(M + Q) + 4M + 8 + \begin{cases} 0 - \text{л.с.,} \\ 12MQ + 4M + 2 - \text{х.с.} \end{cases} \quad (2.2)$$

$$f_{\text{Винограда}}^* \approx 9MNQ$$

2.3 Вывод

При одинаковом коэффициенте при старшем слагаемом в трудоемкости алгоритма Винограда и стандартного, доля долгих операций умножения в алгоритме Винограда меньше. Стоит отметить, что алгоритм Винограда имеет худший (матрицы совпадающей нечётной размерности – количество строк матрицы А и столбцов матрицы В) и лучший случаи (матрицы совпадающей чётной размерности – количество строк матрицы А и столбцов матрицы В), в то время как стандартный алгоритм не зависит от чётности совпадающей размерности матриц. Путём оптимизации трудоемкость алгоритма Винограда была снижена.

2.4 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 тремя алгоритмами;
- 2) возможность вывода результатов замера процессорного времени работы реализаций каждого из алгоритмов.

2.5 Требования к тестированию

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

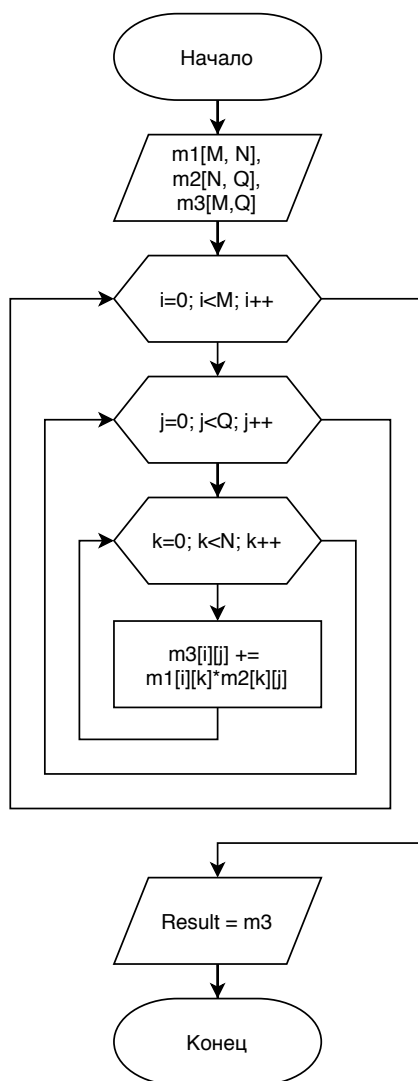


Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

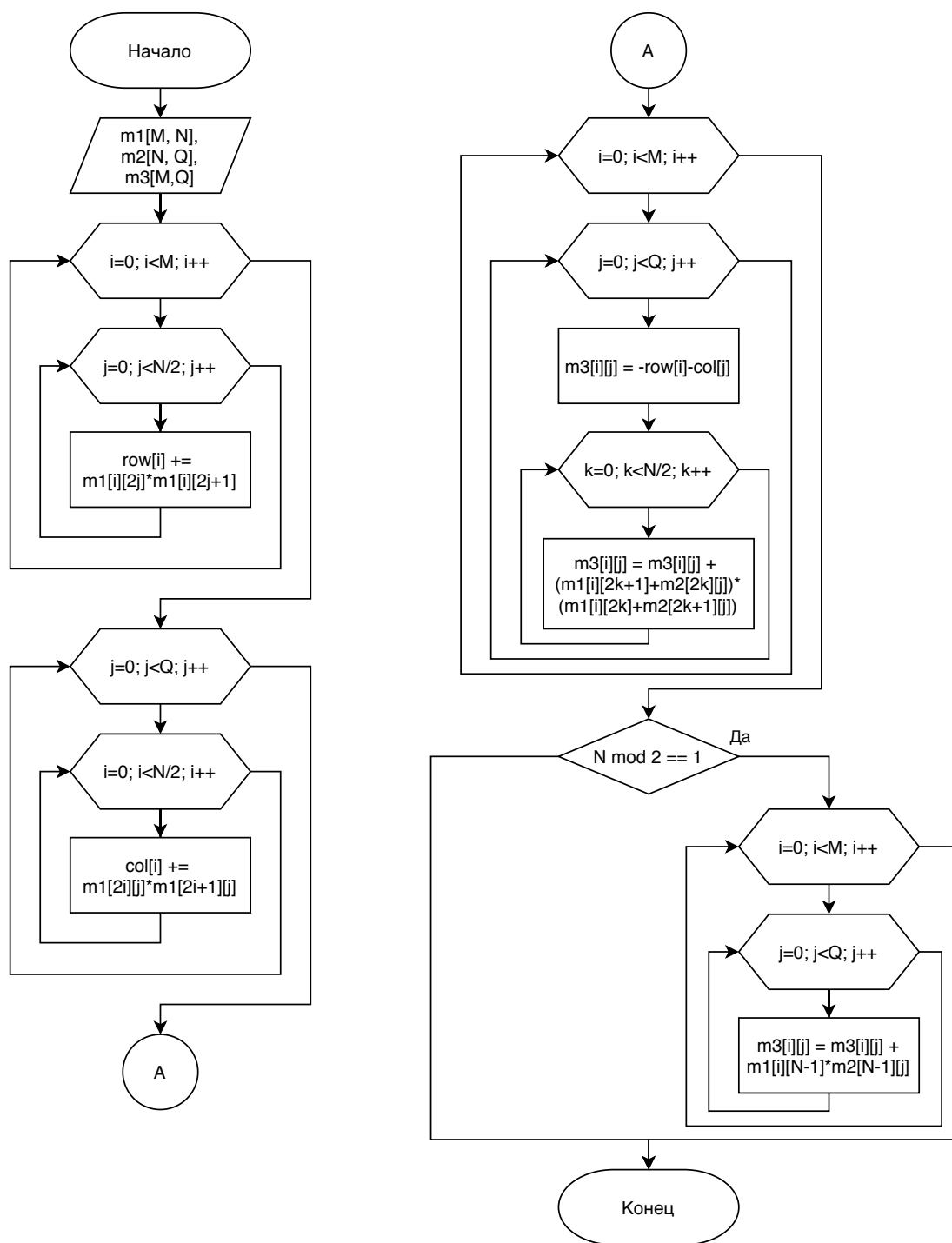


Рисунок 2.2 — Схема алгоритма умножения матриц методом Винограда

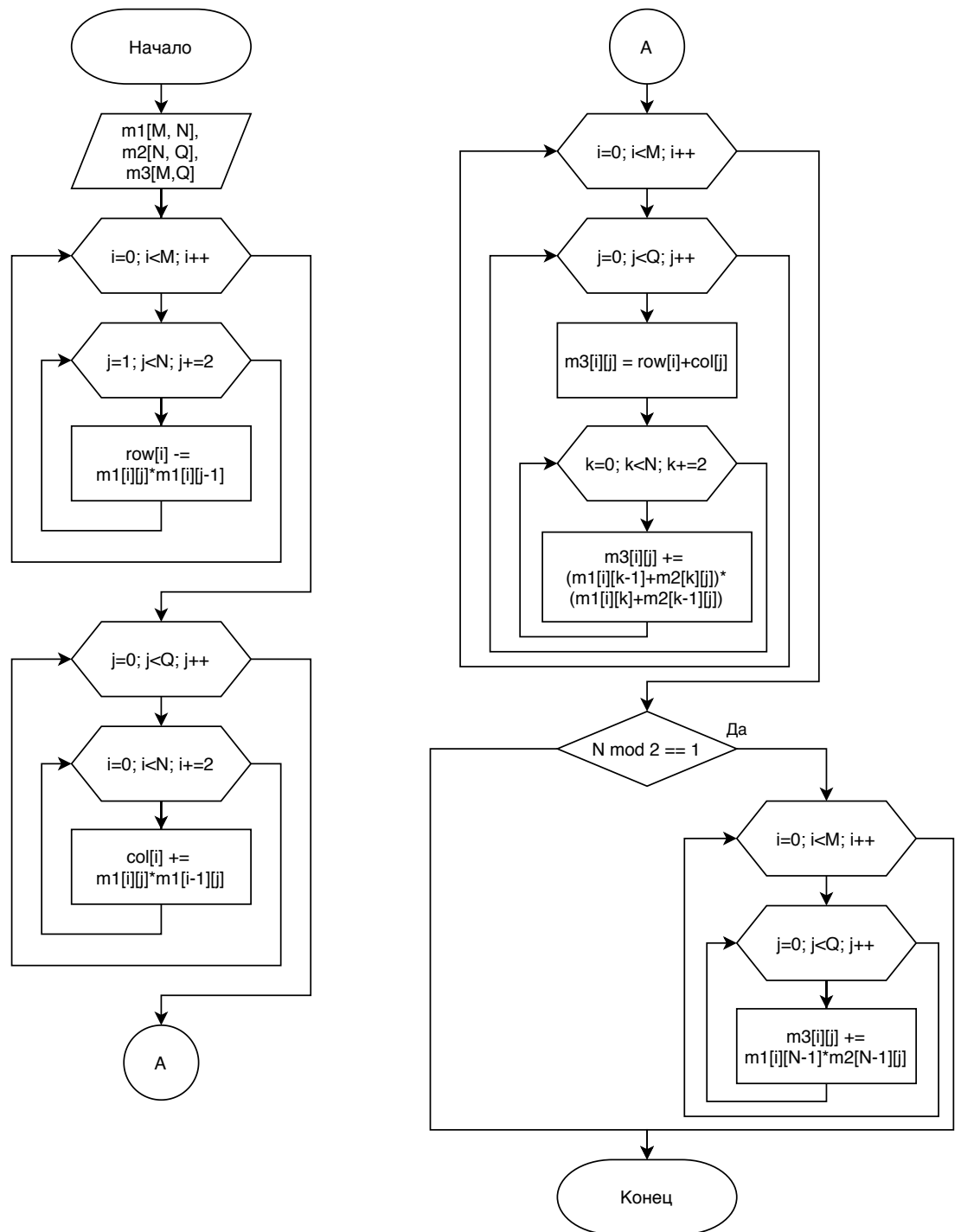


Рисунок 2.3 — Схема оптимизированный алгоритм умножения матриц методом Винограда

3 Технологический раздел

В данном разделе будут выбраны средства репликации 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

```
1 def dotMatrix(matr_a : list , matr_b: list) -> (list , float):
2     if (len(matr_b) != len(matr_a[0])):
3         raise ValueError
4
5     m = len(matr_a)
6     n = len(matr_a[0])
7     q = len(matr_b[0])
8     matr_c = [[0] * q for i in range(m)]
9
10    t_start = process_time()
11    for i in range(m):
12        for j in range(q):
13            for k in range(n):
14                matr_c[i][j] = matr_c[i][j] + matr_a[i][k] * matr_b[k][j]
15    t_end = process_time()
16
17    return matr_c , t_end - t_start
```

Листинг 3.2 — Реализация алгоритма Винограда умножения матриц

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

```

4
5     m = len(matr_a)
6     n = len(matr_a[0])
7
8     q = len(matr_b[0])
9     matr_c = [[0] * q for i in range(m)]
10
11     row = [0] * m
12     col = [0] * q
13
14     t_start = process_time()
15
16     for i in range(m):
17         for j in range(n // 2):
18             row[i] = row[i] + matr_a[i][2*j] * matr_a[i][2*j + 1]
19
20     for j in range(q):
21         for i in range(n // 2):
22             col[j] = col[j] + matr_b[2*i][j] * matr_b[2*i+1][j]
23
24     for i in range(m):
25         for j in range(q):
26             matr_c[i][j] = -row[i] - col[j]
27             for k in range(n//2):
28                 matr_c[i][j] = matr_c[i][j] + (matr_a[i][2*k+1] + matr_b[2*k][j]) *
29                     (matr_a[i][2*k] + matr_b[2*k+1][j])
30
31     if n % 2:
32         for i in range(m):
33             for j in range(q):
34                 matr_c[i][j] = matr_c[i][j] + matr_a[i][n-1] * matr_b[n-1][j]
35
36     t_end = process_time()
37
38     return matr_c, t_end - t_start

```

Листинг 3.3 — Реализация оптимизированного алгоритма Винограда умножения матриц

```

1 def dotMatrixVinogradOptimize(matr_a : list, matr_b: list) -> (list, float):
2     if (len(matr_b) != len(matr_a[0])):
3         raise ValueError
4
5     m = len(matr_a)
6     n = len(matr_a[0])
7     q = len(matr_b[0])
8     matr_c = [[0] * q for i in range(m)]
9
10    row = [0] * m

```

```

11     col = [0] * q
12     t_start = process_time()
13     for i in range(m):
14         for j in range(1, n, 2):
15             row[i] -= matr_a[i][j] * matr_a[i][j - 1]
16
17     for j in range(q):
18         for i in range(1, n, 2):
19             col[j] -= matr_b[i][j] * matr_b[i - 1][j]
20
21     for i in range(m):
22         for j in range(q):
23             matr_c[i][j] = row[i] + col[j]
24             for k in range(1, n, 2):
25                 matr_c[i][j] += (matr_a[i][k - 1] + matr_b[k][j]) * (matr_a[i][k] +
26                                     matr_b[k - 1][j])
27
28     if n % 2:
29         for i in range(m):
30             for j in range(q):
31                 matr_c[i][j] += matr_a[i][n - 1] * matr_b[n - 1][j]
32
33     t_end = process_time()
34
35     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 А	Матрица В	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

Input first matrix: input N: 2 input M: 2 0 0 0 0 Input second matrix: input N: 2 input M: 3 1 1 1 1 1 1 result matrix: 0.0 0.0 0.0 0.0 0.0 0.0 result matrix: 0.0 0.0 0.0 0.0 0.0 0.0 result matrix: 0.0 0.0 0.0 0.0 0.0 0.0	Input first matrix: input N: 2 input M: 2 1 0 0 1 Input second matrix: input N: 2 input M: 3 1 1 1 1 1 1 result matrix: 1.0 1.0 1.0 1.0 1.0 1.0 result matrix: 1.0 1.0 1.0 1.0 1.0 1.0 result matrix: 1.0 1.0 1.0 1.0 1.0 1.0	Input first matrix: input N: 3 input M: 2 1 1 1 1 1 1 Input second matrix: input N: 2 input M: 3 1 1 1 1 1 1 result matrix: 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 result matrix: 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 result matrix: 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0 2.0	Input first matrix: input N: 2 input M: 1 1 1 Input second matrix: input N: 2 input M: 3 1 1 1 1 1 1 the matrices cannot be multiplied the matrices cannot be multiplied 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 неоптимизированного при больших размерностях матриц, в частности, на 43 % и на 14-20 % при размере матрицы 400 стандартного в зависимости от чётности совпадающей размерности матриц. Оптимизированный алгоритм Винограда также быстрее стандартного алгоритма Винограда при больших размерностях матрицы.

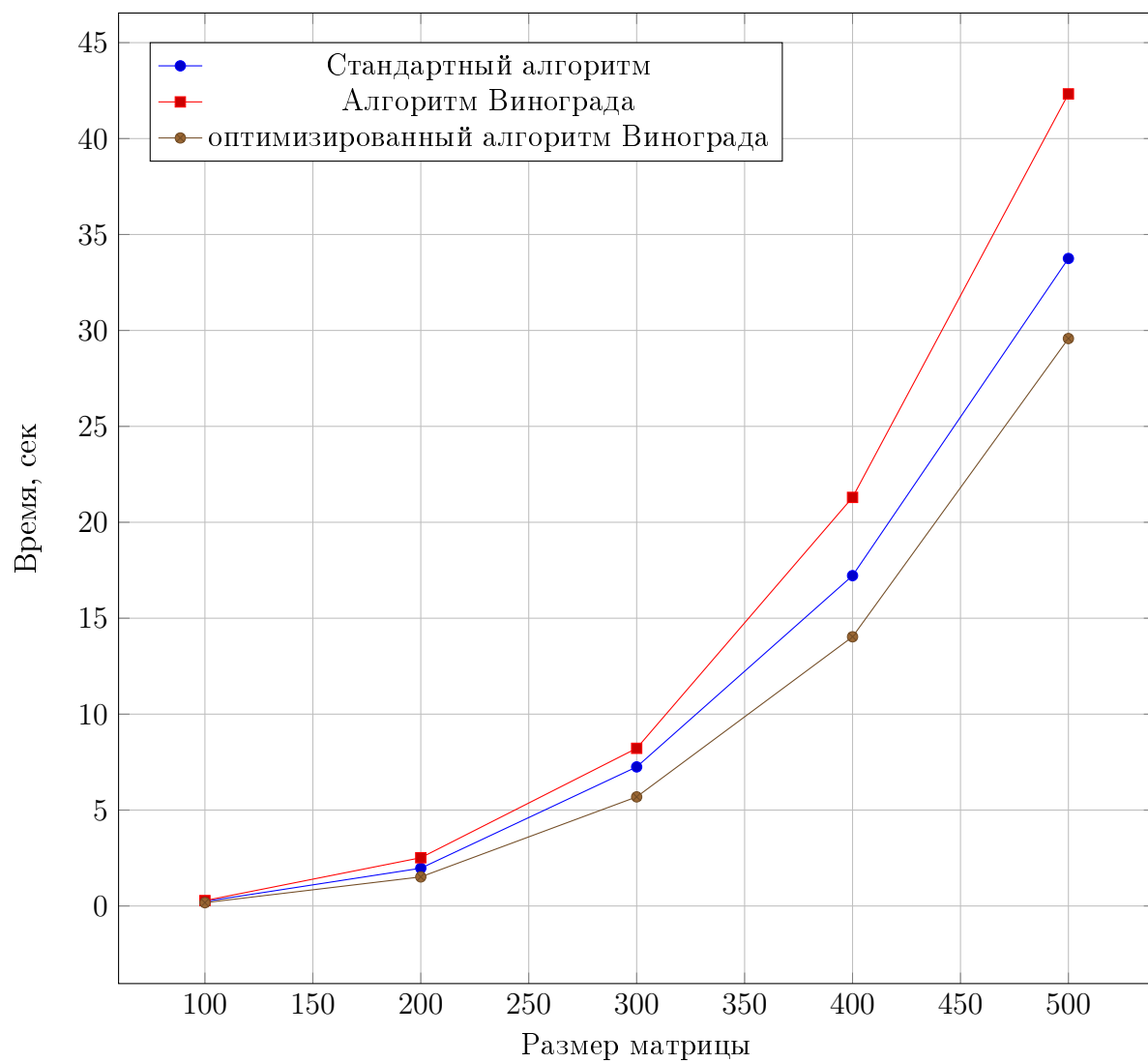


Рисунок 4.1 — График зависимости времени работы алгоритмов при чётных размерностях матриц

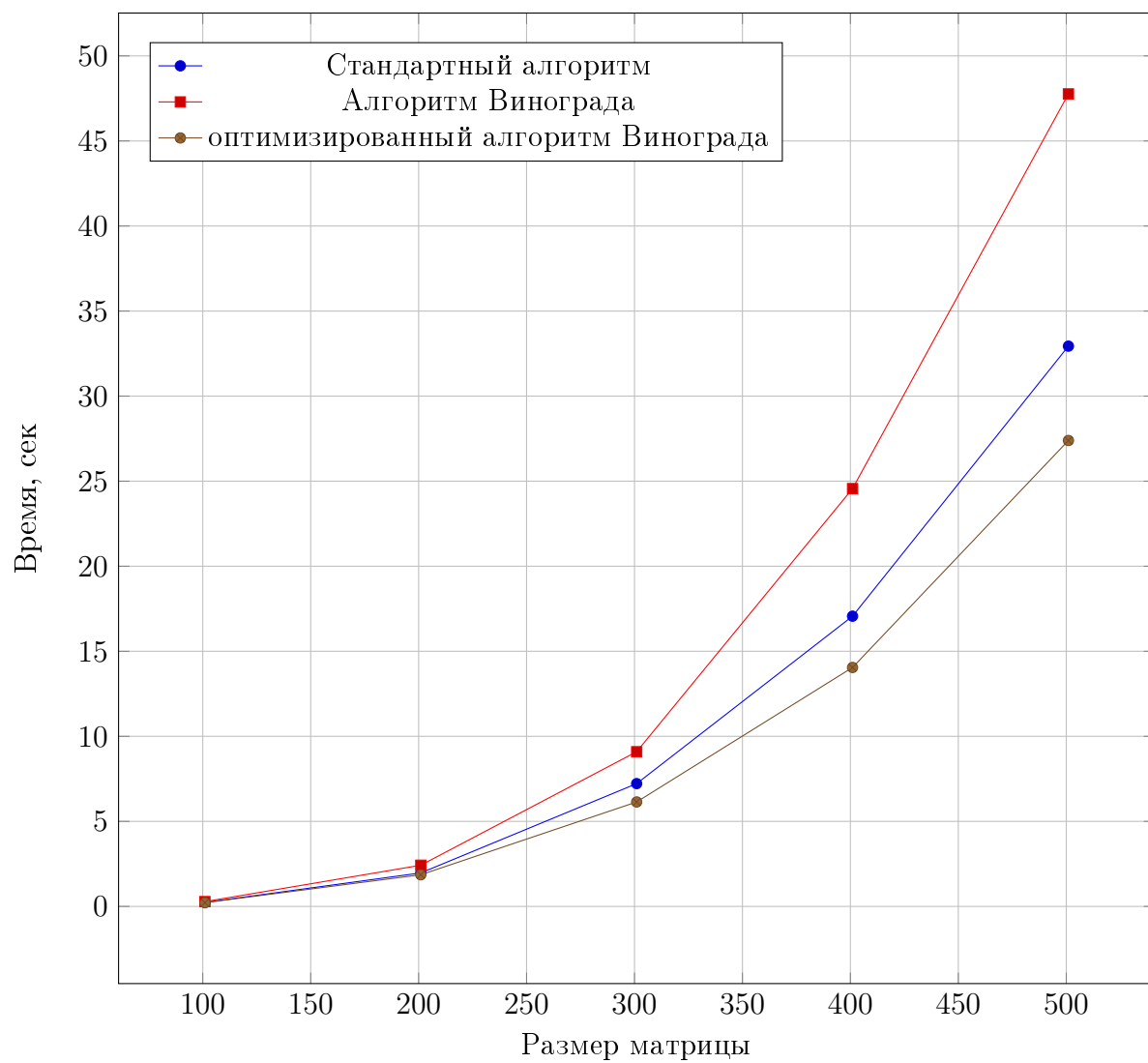


Рисунок 4.2 — График зависимости времени работы алгоритмов при нечётных размерностях матриц

Заключение

В ходе выполнения работы достигнута поставленная цель: проведен сравнительный анализ алгоритмов умножения матриц и получены навыки оптимизации трудоёмкости алгоритмов. Решены все поставленные задачи: 1) дано математическое описание формул расчёта умножения матриц для стандартного алгоритма и Винограда; 2) разработан оптимизированный алгоритм Винограда; 3) реализованы стандартный алгоритм умножения матриц, Винограда и оптимизированного Винограда; 4) дана теоритическая оценка трудоёмкости трёх алгоритмов; 5) проведены замеры процессорного времени работы реализаций трёх алгоритмов в худшем и в лучшем случаях.

В ходе экспериментов по замеру времени работы было установлено, что оптимизированный алгоритм Винограда быстрее неоптимизированного при больших размерностях матриц, в частности, на 43 % и на 14-20 % при размере матрицы 400. Оптимизированный алгоритм Винограда также быстрее стандартного алгоритма Винограда при больших размерностях матрицы.

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).