



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 Алгоритм Коперсмита-Винограда

Студент Малышев И.Н.

Группа ИУ7-54Б

Оценка (баллы)

Преподаватель Волкова Л.Л., Строганов Ю.В.

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Матрица	4
1.2 Стандартный алгоритм	5
1.3 Алгоритм Копперсмита — Винограда	5
2 Конструкторская часть	7
2.1 Разработка алгоритмов	7
2.2 Модель вычислений	7
2.3 Трудоемкость алгоритмов	8
2.3.1 Стандартный алгоритм умножения матриц	8
2.3.2 Алгоритм Копперсмита — Винограда	8
2.3.3 Оптимизированный алгоритм Копперсмита — Вино- града	10
3 Технологическая часть	14
3.1 Требования к программе	14
3.2 Средства реализации	14
3.3 Сведения о модулях программы	14
3.4 Листинг кода	15
3.5 Функциональные тесты	17
4 Исследовательская часть	19
4.1 Технические характеристики	19
4.2 Время выполнения алгоритмов	19
Заключение	22
Литература	23

Введение

Термин «матрица» применяется во множестве разных областей: от программирования до кинематографии.

Матрица в математике – это таблица чисел, состоящая из определенного количества строк (m) и столбцов (n).

Мы встречаемся с матрицами каждый день, так как любую числовую информацию, структурированную в виде таблицы, можно рассматривать как матрицу.

Целью работы является изучение и реализация алгоритмов умножения матриц, вычисление трудоёмкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Задачи данной лабораторной:

- 1) Реализовать алгоритмы перемножения матриц:
 - 1.1) классический;
 - 1.2) Винограда;
 - 1.3) Винограда с оптимизациями согласно варианту;
- 2) оценить трудоемкость алгоритмов;
- 3) провести замеры времени работы и объема используемой памяти алгоритмов;
- 4) предложить рекомендации об особенностях применения оптимизаций.

1 Аналитическая часть

В этом разделе будут представлены описания алгоритмов стандартного умножения матриц и алгоритм Копперсмита - Винограда.

1.1 Матрица

Матрица [1] – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Две матрицы одинакового размера можно поэлементно сложить или вычесть друг из друга.

Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Умножение матриц некоммукативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга.

1.2 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \quad (1.1)$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \quad (1.2)$$

где

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

будет называться произведением матриц A и B .

Стандартный алгоритм реализует данную формулу.

1.3 Алгоритм Копперсмита — Винограда

Алгоритм Копперсмита — Винограда [2] — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита — Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц [3].

Рассмотрим векторы $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4$, что экви-

валентно (1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \quad (1.4)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

В случае нечетного значений размера изначальной матрицы (n), следует произвести еще одну операцию - добавление произведения последних элементов соответствующих строк и столбцов.

Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которого от классического алгоритма — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов умножения матриц и расчет их трудоемкости.

2.1 Разработка алгоритмов

Предполагается, что на вход всех алгоритмов поступили матрицы верного размера.

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

На рисунке 2.2 приведена схема алгоритма Винограда умножения матриц.

На рисунке 2.3 приведена схема оптимизированного алгоритма Винограда умножения матриц.

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \quad (2.1)$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.2)$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{инициализации} + f_{сравнения} + N(f_{тела} + f_{инкремента} + f_{сравнения}) \quad (2.3)$$

4. трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут рассчитаны трудоемкости алгоритмов умножения матриц.

Во всех последующих алгоритмах не будем учитывать инициализацию матрицы, в которую записывается результат, потому что данное действие есть во всех алгоритмах и при этом не является самым трудоёмким.

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из:

- внешнего цикла по $i \in [1..M]$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;
- цикла по $j \in [1..N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;
- цикла по $k \in [1..K]$, трудоёмкость которого: $f = 2 + 10K$;

Учитывая, что трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела (2.4):

$$f_s = 2 + M \cdot (4 + N \cdot (4 + 10K)) = 2 + 4M + 4MN + 10MNK \approx 10MNK \quad (2.4)$$

2.3.2 Алгоритм Копперсмита — Винограда

Трудоёмкость алгоритма Копперсмита — Винограда состоит из:

- Создания и инициализации массивов MH и MV , трудоёмкость которого (2.5).

$$f_{init} = M + N; \quad (2.5)$$

- Заполнения массива MH , трудоёмкость которого (2.6).

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 11) \quad (2.6)$$

- Заполнения массива MV , трудоёмкость которого (2.7).

$$f_{MV} = 2 + K(2 + \frac{N}{2} \cdot 11) \quad (2.7)$$

- Цикла заполнения для чётных размеров, трудоёмкость которого (2.8).

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)) \quad (2.8)$$

- Цикла, для дополнения умножения суммой последних нечётных строки и столбца, трудоёмкость которого (2.9).

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \quad (2.9)$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.10).

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \quad (2.10)$$

Для лучшего случая (чётный общий размер матриц) имеем (2.11).

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \quad (2.11)$$

2.3.3 Оптимизированный алгоритм Копперсмита — Винограда

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- вычисление происходит заранее;
- используется битовый сдвиг, вместо деления на 2;
- операция вида $a = a + b$ заменена на $a += b$.

Трудоёмкость улучшенного алгоритма Копперсмита — Винограда состоит из:

- создания и инициализации массивов MH и MV , трудоёмкость которого (2.12):

$$f_{init} = M + N; \quad (2.12)$$

- заполнения массива MH , трудоёмкость которого (2.13):

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 8); \quad (2.13)$$

- заполнения массива MV , трудоёмкость которого (2.14):

$$f_{MV} = 2 + K(2 + \frac{M}{2} \cdot 8); \quad (2.14)$$

- цикла заполнения для чётных размеров, трудоёмкость которого (2.15):

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)); \quad (2.15)$$

- условие, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоёмкость которого (2.16):

$$f_{last} = \begin{cases} 1, & \text{чётная,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \quad (2.16)$$

Итого, для худшего случая (нечётный общий размер матриц) имеем (2.17):

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \quad (2.17)$$

Для лучшего случая (чётный общий размер матриц) имеем (2.18):

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \quad (2.18)$$

Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы обоих алгоритмов умножения матриц. Оценены их трудоёмкости в лучшем и худшем случаях.

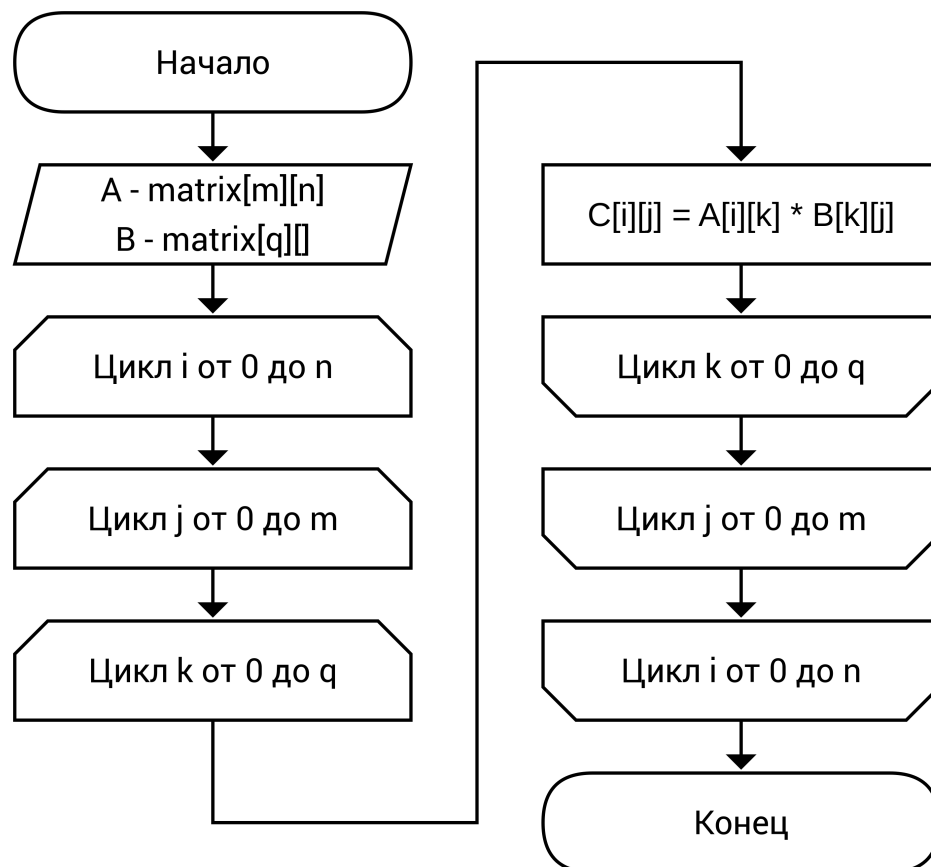


Рисунок 2.1 – Схема стандартного алгоритма умножения матриц



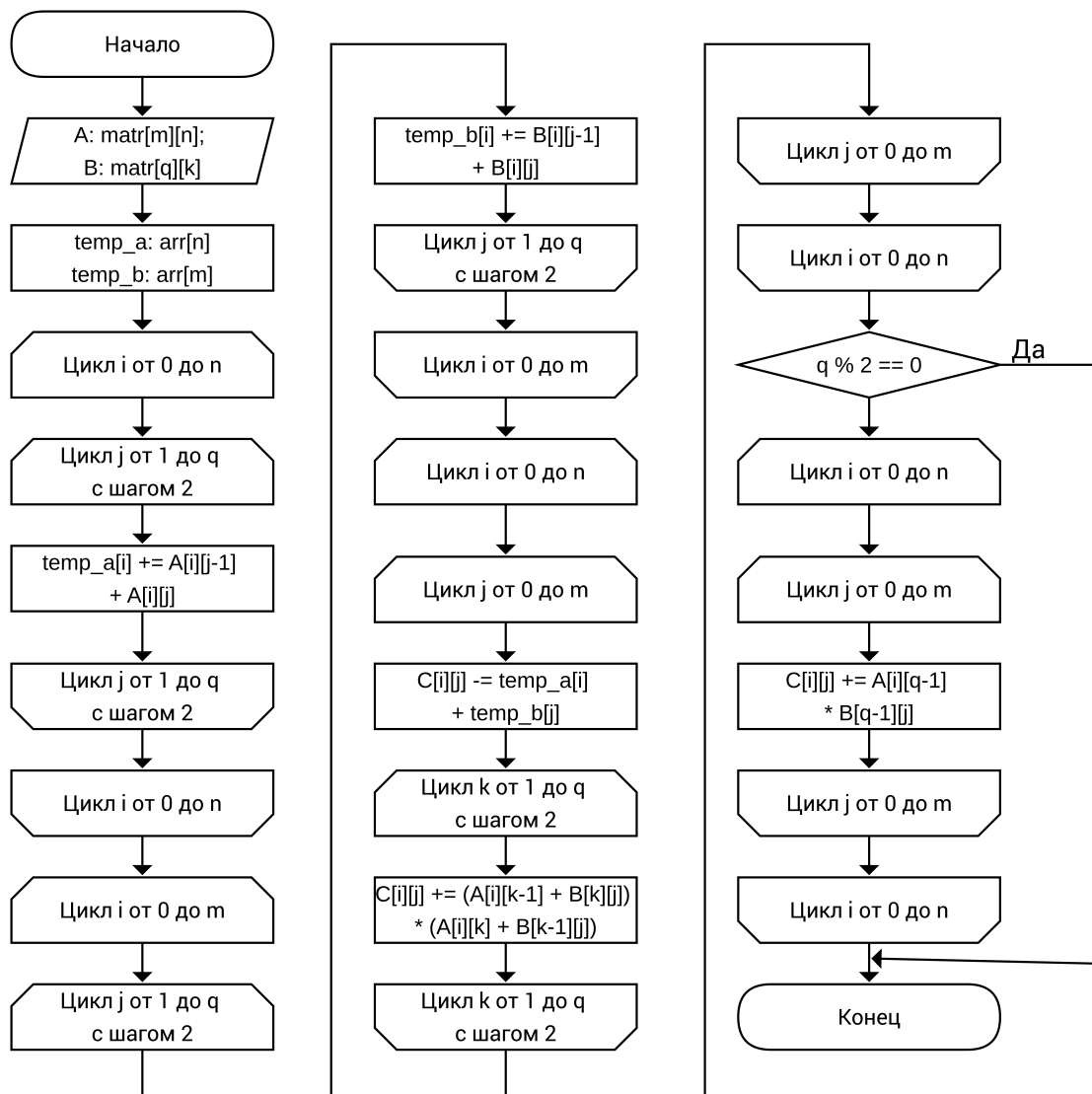


Рисунок 2.3 – Схема оптимизированного алгоритма Винограда умножения матриц

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к программе

К программе предъявляется ряд требований.

- Входными данными являются две матрицы A и B.
- Количество столбцов матрицы A должно быть равно количеству строк матрицы B.
- На выходе получается результат умножения, введенных пользователем, матриц.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования (ЯП) Python [4].

Данный ЯП предоставляет широкий набор библиотек и руководств по их использованию, что значительно облегчает выполнения большинства задач в области программирования.

Время работы алгоритмов было замерено с помощью функции `time()` из библиотеки `time` [5].

3.3 Сведения о модулях программы

Программа состоит из двух модулей.

1. `main.py` - главный файл программы, в котором располагаются коды всех алгоритмов и меню.

2. test.py - файл с замерахми времени для графического изображения результата.

3.4 Листинг кода

В листинге 3.1 приведена реализация алгоритма стандартного алгоритма матриц.

В листингах 3.2, 3.3 приведена реализация алгоритма Копперсмита — Винограда.

Листинг 3.1 – Реализация стандартного умножения матриц

```
1 def simple_matrix_mult(matrix1, matrix2):
2     if (len(matrix1) != len(matrix2[0])):
3         print("Error_size_matrix!")
4         return -1
5
6     n = len(matrix1)
7     m = len(matrix1[0])
8     q = len(matrix2[0])
9
10    matrix_res = [[0] * q for i in range(n)]
11
12    for i in range(n):
13        for j in range(q):
14            for k in range(m):
15                matrix_res[i][j] = matrix_res[i][j] + \
16                    matrix1[i][k] * matrix2[k][j]
17
18    return matrix_res
```

Листинг 3.2 – Реализация алгоритма Копперсмита-Винограда

```
1 def winograd_matrix_mult(matrix1, matrix2):
2     if (len(matrix2) != len(matrix1[0])):
3         print("Error_size_matrix!")
4         return -1
5
6     n = len(matrix1)
7     m = len(matrix1[0])
8     q = len(matrix2[0])
```

```

9
10 matrix_res = [[0] * q for i in range(n)]
11
12 row_factor = [0] * n
13 for i in range(n):
14     for j in range(0, m // 2, 1):
15         row_factor[i] = row_factor[i] + \
16             matrix1[i][2 * j] * matrix1[i][2 * j + 1]
17
18 column_factor = [0] * q
19 for i in range(q):
20     for j in range(0, m // 2, 1):
21         column_factor[i] = column_factor[i] + \
22             matrix2[2 * j][i] * matrix2[2 * j + 1][i]
23
24 for i in range(n):
25     for j in range(q):
26         matrix_res[i][j] = -row_factor[i] - column_factor[j]
27         for k in range(0, m // 2, 1):
28             matrix_res[i][j] = matrix_res[i][j] + \
29                 (matrix1[i][2 * k + 1] + matrix2[2 * k][j]) *
30                 \
31                 (matrix1[i][2 * k] + matrix2[2 * k + 1][j])
32
33 if m % 2 == 1:
34     for i in range(n):
35         for j in range(q):
36             matrix_res[i][j] = matrix_res[i][j] + \
37                 matrix1[i][m - 1] * matrix2[m - 1][j]
38
39 return matrix_res

```

Листинг 3.3 – Реализация алгоритма Копперсмита-Винограда
(оптимизированный)

```

1 def winograd_matrix_mult_opim(matrix1, matrix2):
2     if (len(matrix2) != len(matrix1[0])):
3         print("Error_size_matrix!")
4         return -1
5
6     n = len(matrix1)
7     m = len(matrix1[0])

```



```

8     q = len(matrix2[0])
9
10    matrix_res = [[0] * q for i in range(n)]
11
12    row_factor = [0] * n
13    for i in range(n):
14        for j in range(1, m, 2):
15            row_factor[i] += matrix1[i][j] * matrix1[i][j - 1]
16
17    column_factor = [0] * q
18    for i in range(q):
19        for j in range(1, m, 2):
20            column_factor[i] += matrix2[j][i] * matrix2[j - 1][i]
21
22    flag = n % 2
23    for i in range(n):
24        for j in range(q):
25            matrix_res[i][j] = -(row_factor[i] + column_factor[j])
26            for k in range(1, m, 2):
27                matrix_res[i][j] += (matrix1[i][k - 1] +
28                                     matrix2[k][j]) * \
29                                     (matrix1[i][k] + matrix2[k - 1][j])
30            if (flag):
31                matrix_res[i][j] += matrix1[i][m - 1] \
32                                     * matrix2[m - 1][j]
33
34    return matrix_res

```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Тесты пройдены успешно.

Таблица 3.1 – Тестирование функций

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 1 & 1 \\ 5 & 5 & 5 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 15 \\ 6 \end{pmatrix}$
$(1 \ 1 \ 1)$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$
(2)	(2)	(4)
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$
$(1 \ 2)$	$(1 \ 2)$	Неверный размер

Вывод

В этом разделе была представлена реализация алгоритмов классического умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда. Тестирование программы было пройдено успешно.

4 Исследовательская часть

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 выполнялось тестирование, следующие:

- Операционная система Ubuntu 22.04.1 [6] Linux x86_64.
- Память: 8 ГБ.
- Процессор: AMD® Ryzen 5 3500u.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Алгоритмы тестировались при помощи функции `time()` из библиотеки `time` языка Python. Данная функция возвращает количество секунд, прошедших с начала эпохи.

Контрольная точка возвращаемого значения не определена, поэтому допустима только разница между результатами последовательных вызовов.

Замеры времени для каждой длины входного массива проводились 1000 раз. В качестве результата взято среднее время работы алгоритма на данной длине слова. При каждом запуске алгоритма, на вход подавались случайно сгенерированные строки. Тестовые пакеты создавались до начала замера времени.

Результаты замеров приведены в таблицах 4.1 и 4.2. Некоторые обозначения в таблице: СА – стандартный алгоритм, АВ – алгоритм Винограда, ОАВ – оптимизированный алгоритм Винограда.

Время в таблицах и на рисунках приведены в микросекундах.

Таблица 4.1 – Результаты замеров времени алгоритмов при четных размерах матриц (микросекунды)

Размер	АВ	СА	ОАВ
50	0.0138	0.0122	0.0108
100	0.1054	0.1001	0.0869
150	0.5808	0.4479	0.3051
200	1.3738	0.8694	1.1329
300	4.3151	4.4543	3.3223
400	11.4547	10.6556	9.6063
500	18.8731	16.9742	14.1837

Таблица 4.2 – Результаты замеров времени алгоритмов при нечетных размерах матриц (микросекунды)

Размер	АВ	СА	ОАВ
50	0.0294	0.0237	0.0273
100	0.2271	0.1751	0.1681
150	0.7349	0.5922	0.5510
200	1.8234	1.4086	1.3162
300	6.4540	4.7933	4.6446
400	14.8665	11.6306	11.3941
500	28.6478	22.6672	21.6239

Вывод

В результате проведенного эксперимента был получен следующий вывод: алгоритм Винограда работает дольше за счет большого количества операций. Оптимизированный алгоритм работает быстрее, так как там меньше операций по сравнению с обычным алгоритмом Винограда и стандартным алгоритмом.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- были изучены и реализованы 3 алгоритма перемножения матриц: классический, Копперсмита-Винограда, модифицированный Копперсмита-Винограда;
- был произведен анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был сделан сравнительный анализ алгоритмов на основе экспериментальных данных;
- был подготовлен отчет о лабораторной работе.

Поставленная цель достигнута.

Литература

- [1] Матрица [Электронный ресурс]. Режим доступа: <https://terme.ru/termin/matrica.html> (дата обращения: 14.10.2022).
- [2] Умножение матриц [Электронный ресурс]. Режим доступа: <http://algotlib.narod.ru/Math/Matrix.html> (дата обращения: 14.10.2022).
- [3] Group-theoretic Algorithms for Matrix Multiplication / H. Cohn, R. Kleinberg, B. Szegedy et al. // Proceedings of the 46th Annual Symposium on Foundations of Computer Science. 2005. October. P. 379–388.
- [4] Welcome to Python [Электронный ресурс]. Режим доступа: <https://www.python.org> (дата обращения: 04.09.2021).
- [5] time — Time access and conversions [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html#functions> (дата обращения: 04.09.2021).
- [6] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: <https://releases.ubuntu.com/20.04/> (дата обращения: 04.09.2021).