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 Копперсмита-Винограда

Студент Динь Вьет Ань

Группа ИУ7И-64Б

Оценка (баллы) _____

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

Москва — 2023 г.

Оглавление

Введение

Матрицы используются во многих сферах деятельности. Например, ими пользуются при решении различных практических задач в математике, биологии, физике, технике, химии, экономике, маркетинге, психологии и других областях науки.

Оптимизация операций работы над матрицами является важной задачей в программировании, так как размеры матриц могут достигать больших значений. Об оптимизации операции умножения пойдет речь в данной лабораторной работе.

Целью данной работы является изучение, реализация и исследование алгоритмов умножения матриц - классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить и реализовать алгоритмы - классический, Винограда и его оптимизацию;
- провести тестирование по времени и по памяти для алгоритмов лабораторной работы;
- провести сравнительный анализ по времени классического алгоритма и алгоритма Винограда;
- провести сравнительный анализ по времени алгоритма Винограда и его оптимизации;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе были представлены описания алгоритма стандартного умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

1.1 Матрица

Матрица [?] - математический объект, который представляет собой двумерный массив, в котором элементы располагаются 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 некоммутативно: 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 (1.1).

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

тогда матрица C (1.2).

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

где

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

будет называться произведением матриц A и B.

Стандартный алгоритм реализует данную формулу (1.3).

1.3 Алгоритм Винограда

Алгоритм Винограда [?] — алгоритм умножения квадратных матриц. Начальная версия, предложенная в 1987 году Д. Копперсмитом и Ш. Виноградом, имела асимптотическую сложность примерно $O(n^{2,3755})$, где n - размер стороны матрицы, но после доработки алгоритм стал обладать лучшей асимптотикой среди всех алгоритмов умножения матриц [?].

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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно:

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

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.

1.4 Оптимизированный алгоритм Винограда

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

- вычисление происходит заранее;
- используется битовый сдвиг, вместо деления на 2;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

Вывод

Были рассмотрены алгоритм классического умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда. Было выяснено, что основные отличия алгоритма Винограда от классического алгоритма — наличие предварительной обработки и количество операций умножения.

2 Конструкторская часть

В этом разделе были приведены требования к вводу и программе, а также схемы алгоритмов умножения матриц.

2.1 Разработка алгоритмов

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

На рисунках 2.1-2.4 приведены схема стандартного алгоритма умножения матриц, схема алгоритма Винограда и схема оптимизированного алгоритма Винограда соответственно.

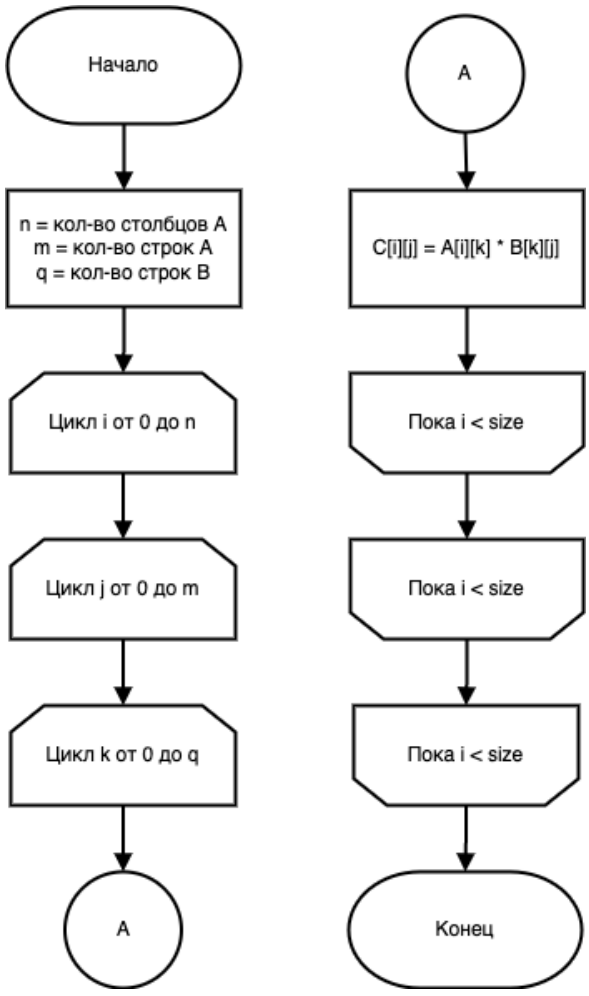

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

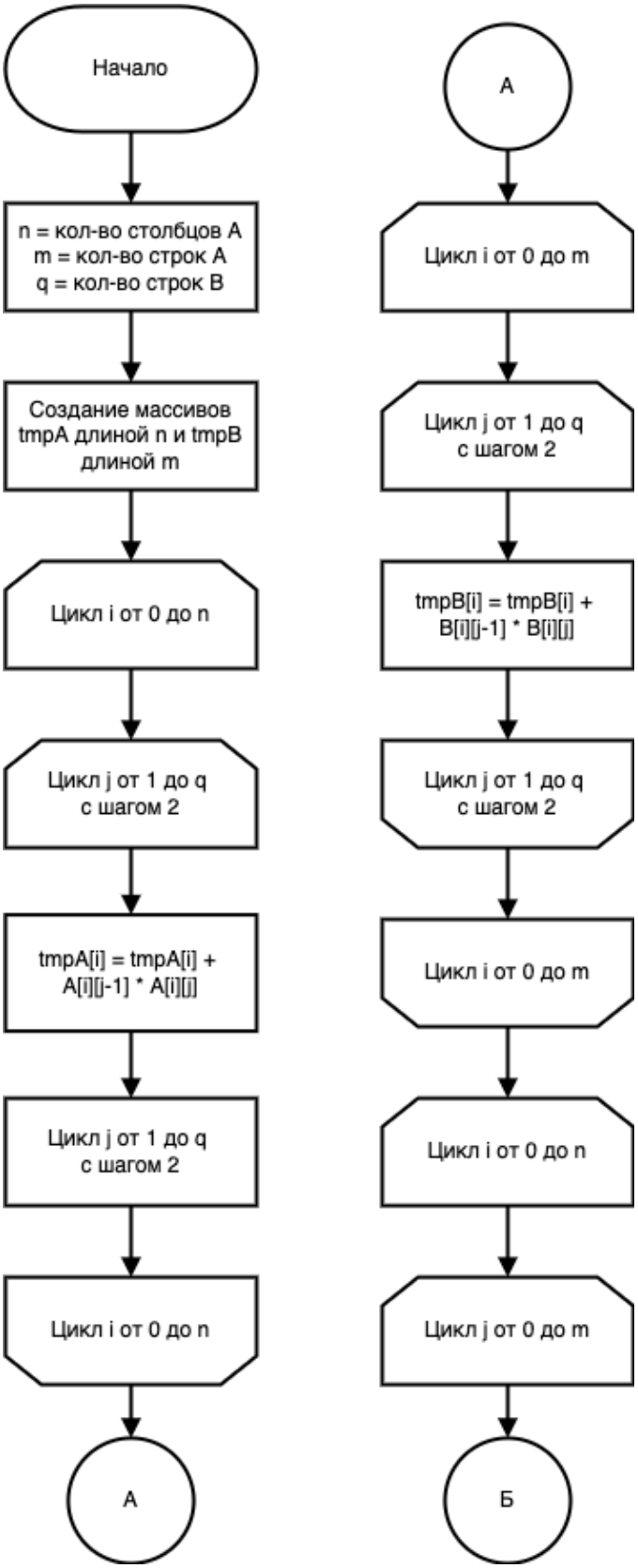

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц

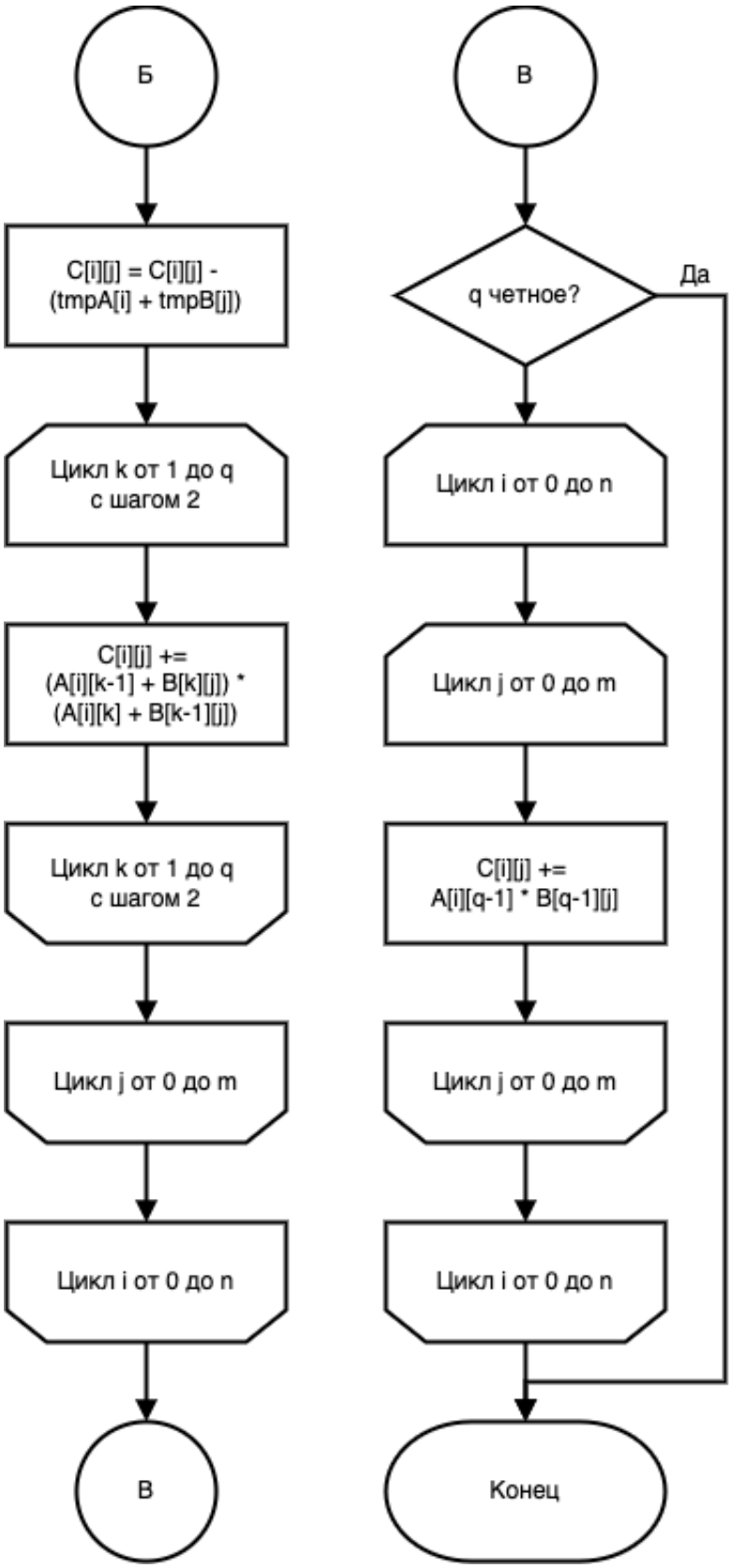

Рисунок 2.3 – Схема алгоритма Винограда умножения матриц (продолжение)

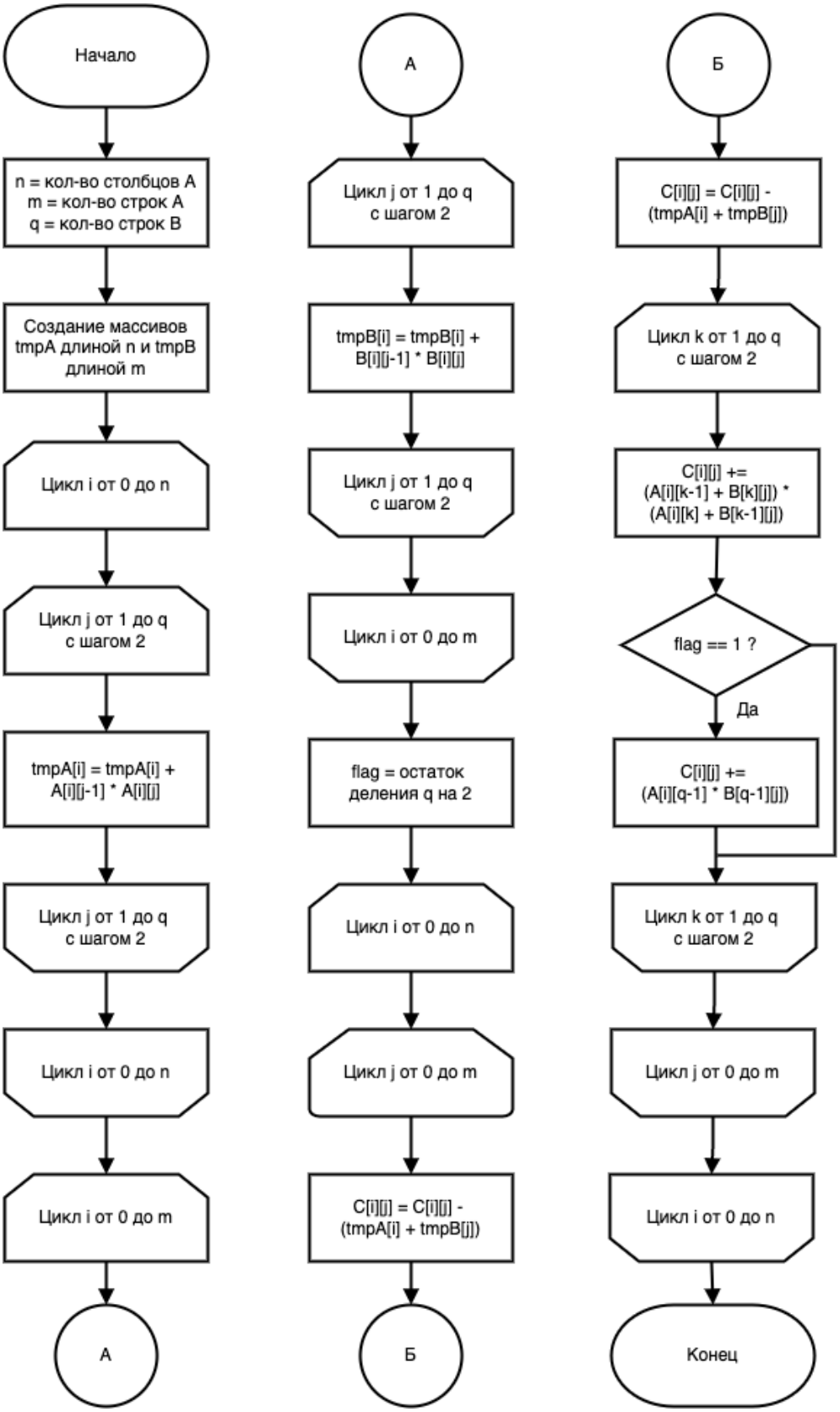

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда умножения матриц

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

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

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

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

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

3. трудоемкость цикла рассчитывается, как (2.3);

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

4. трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях были рассчитаны трудоемкости алгоритмов умножения матриц.

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость внешнего цикла по $i \in [1..M]$:

$$f = 2 + M \cdot (2 + f_{body}) \tag{2.4}$$

Трудоёмкость цикла по $j \in [1..N]$:

$$f = 2 + N \cdot (2 + f_{body}) \tag{2.5}$$

Трудоёмкость цикла по $k \in [1..K]$:

$$f = 2 + 10K \tag{2.6}$$

Учитывая, что трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее по следующей формуле:

$$f_{standard} = 2 + M \cdot (4 + N \cdot (4 + 10K)) \approx 10MNK \tag{2.7}$$

2.3.2 Алгоритм Винограда

Трудоёмкость создания и инициализации массивов MH и MV:

$$f_{init} = M + N \tag{2.8}$$

Трудоёмкость заполнения массива MH:

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 11) \tag{2.9}$$

Трудоёмкость заполнения массива MV:

$$f_{MV} = 2 + K(2 + \frac{N}{2} \cdot 11) \tag{2.10}$$

Трудоёмкость цикла заполнения для чётных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)) \tag{2.11}$$

Трудоёмкость цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный:

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \tag{2.12}$$

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

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.13}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.14}$$

2.3.3 Оптимизированный алгоритм Винограда

Трудоёмкость создания и инициализации массивов MH и MV:

$$f_{init} = M + N \tag{2.15}$$

Трудоёмкость заполнения массива MH:

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 8) \tag{2.16}$$

Трудоёмкость заполнения массива MV:

$$f_{MV} = 2 + K(2 + \frac{M}{2} \cdot 8) \tag{2.17}$$

Трудоёмкость цикла заполнения для чётных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)) \tag{2.18}$$

Трудоёмкость дополнительных вычислений при нечетном размере матрицы:

$$f_{last} = \begin{cases} 1, & \text{чётная,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \tag{2.19}$$

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

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.20}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.21}$$

Вывод

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

3 Технологическая часть

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

3.1 Требования к программному обеспечению

Программа принимает на вход две матрицы A и B. Количество столбцов матрицы A должно быть равно количеству строк матрицы B. На выходе получается результат умножения матриц, введенных пользователем.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python [**?**].

Данный язык имеет все небходимые инструменты для решения поставленной задачи.

Время работы алгоритмов было замерено с помощью функции time() из библиотеки time [**?**]

3.3 Сведения о модулях программы

Программа состоит из четырех модулей:

1. algorithms.py - хранит реализацию алгоритмов сортировок;
2. unit_tests.py - хранит реализацию тестирующей системы и тесты;
3. time_memory.py - хранит реализацию системы замера памяти и времени;
4. tools.py - хранит реализацию вспомогательных функций.

3.4 Реализация алгоритмов

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

Листинг 3.1 – Реализация стандарного умножения матриц

```
def simple_matrix_mult(m1, m2):
    if len(m1[0]) != len(m2):
        print("Error")
        return -1

    n = len(m1)
    m = len(m1[0])
    q = len(m2[0])

    m_res = [[0] * q for _ in range(n)]

    for i in range(n):
        for j in range(q):
            for k in range(m):
                m_res[i][j] = m_res[i][j] + m1[i][k] * m2[k][j]

    return m_res
```

Листинг 3.2 – Реализация алгоритма Копперсмита-Винограда

```
def winograd_matrix_mult(matrix1, matrix2):
    if (len(matrix2) != len(matrix1[0])):
        print("Error")
        return -1

    n = len(matrix1)
    m = len(matrix1[0])
    q = len(matrix2[0])

    matrix_res = [[0] * q for i in range(n)]

    row_factor = [0] * n
    for i in range(n):
        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                      (matrix1[i][2 * k] + matrix2[2 * k + 1][j])
31
32      if m % 2 == 1:
33          for i in range(n):
34              for j in range(q):
35                  matrix_res[i][j] = matrix_res[i][j] +\
36                      matrix1[i][m - 1] * matrix2[m - 1][j]
37
38      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")
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] +
                        matrix2[k][j]) * \
28                              (matrix1[i][k] + matrix2[k - 1][j])
29              if (flag):
30                  matrix_res[i][j] += matrix1[i][m - 1] \
31                      * matrix2[m - 1][j]
32
33      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 – Функциональные тесты

Матрица A	Матрица B	Ожидаемый результат
$\begin{pmatrix}1 & 1 & 1\end{pmatrix}$	$\begin{pmatrix}1\\1\\1\end{pmatrix}$	$\begin{pmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{pmatrix}$
$\begin{pmatrix}5 & -1\end{pmatrix}$	$\begin{pmatrix}-1 & 10\end{pmatrix}$	Неверный размер
$\begin{pmatrix}1 & 0 & 0\\1 & 1 & 0\\0 & 0 & 1\end{pmatrix}$	$\begin{pmatrix}1\\4\\2\end{pmatrix}$	$\begin{pmatrix}1 & 0 & 0\\0 & 4 & 0\\0 & 0 & 2\end{pmatrix}$
$\begin{pmatrix}1 & 2\\2 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}$	$\begin{pmatrix}3 & 3\\3 & 3\end{pmatrix}$
$\begin{pmatrix}1\end{pmatrix}$	$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}3\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}12 & 18 & 24\\30 & 45 & 60\\12 & 18 & 24\end{pmatrix}$

Вывод

В этом разделе была представлена реализация алгоритмов классического умножения матриц, алгоритма Винограда, оптимизированного алгоритма Винограда. Тестирование показало, что алгоритмы реализованы правильно и работают корректно.

4 Исследовательская часть

В данном разделе приводятся результаты замеров алгоритмов по процессорному времени.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система macOS Monterey 12.5.1
- Память 16 Гб.
- Процессор 2,3 ГГц 4-ядерный процессор Intel Core i5.

Во время тестирования устройство было подключено к сети электропитания, нагружено приложениями окружения и самой 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, типа float.

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

Замеры времени для каждого размера матрицы проводились 500 раз. В качестве результата взято среднее время работы алгоритма на данном размере. При каждом запуске алгоритма, на вход подавались случайно сгенерированные матриц. Тестовые пакеты создавались до начала замера времени.

Результаты замеров приведены на рисунках 4.1, 4.2 (в микросекундах). Графики зависимостей времени работы алгоритмов от размеров матриц на рисунках 4.3 и 4.4

N	Стандартный алгоритм	Алгоритм Винограда	Оптимизированный алгоритм Винограда
10	439	426	344
20	2157	1880	1632
50	28458	28605	23954
100	231953	221790	182307
200	1833988	1766213	1501507
300	6356873	6210797	5024939
500	28839648	29937409	26216433

Рисунок 4.1 – Результаты замеров времени алгоритмов при четных размерах матриц (в микросекундах)

N	Стандартный алгоритм	Алгоритм Винограда	Оптимизированный алгоритм Винограда
11	324	350	394
21	2092	2185	1816
51	29903	29011	25171
101	234071	239060	177787
201	1687508	1734066	1527341
301	6372649	6145855	5279598
501	29503069	31222224	25412715

Рисунок 4.2 – Результаты замеров времени алгоритмов при нечетных размерах матриц (в микросекундах)

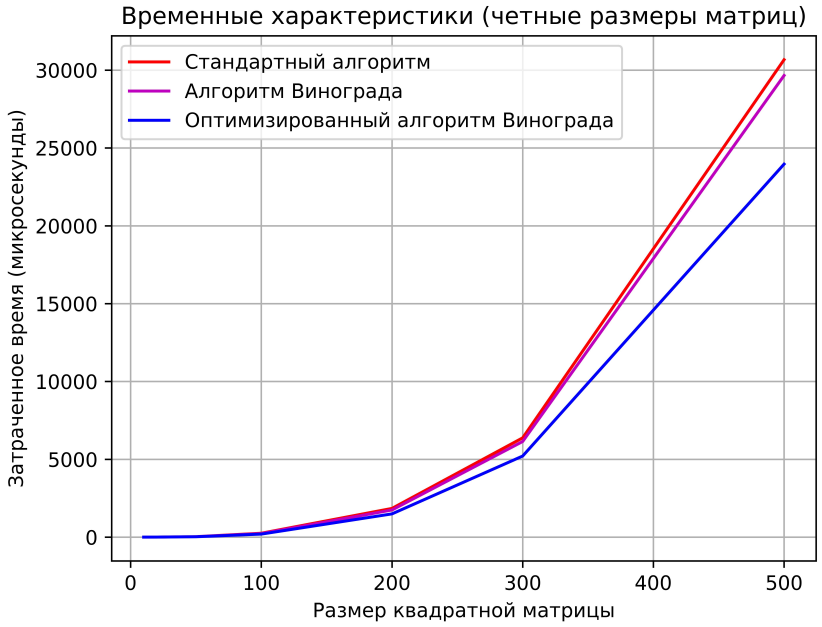

Рисунок 4.3 – Зависимость времени работы алгоритма от четного размера квадратной матрицы (микросекунды)

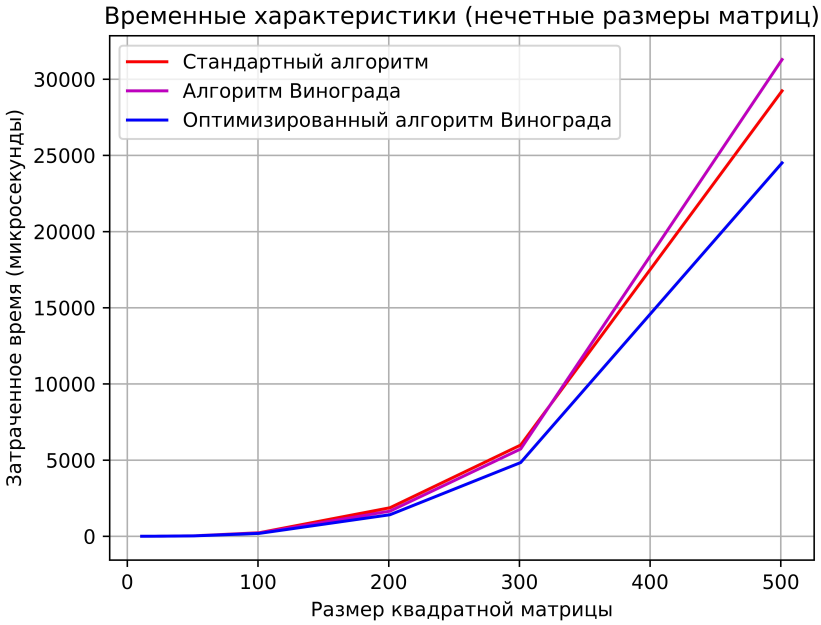

Рисунок 4.4 – Зависимость времени работы алгоритма от нечетного размера квадратной матрицы (микросекунды)

Вывод

В результате эксперимента было получено, что при больших размерах матриц (свыше 10), алгоритм Винограда быстрее стандартного алгоритма более, чем 1.2 раза, а оптимизированный алгоритм Винограда быстрее стандартного алгоритма в 1.3 раза. Также при проведении эксперимента было выявлено, что на четных размерах реализация алгоритма Винограда в 1.2 раза быстрее, чем на нечетных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов.

Заключение

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

- были изучены и реализованы алгоритмы умножения матриц - стандартный, Винограда и оптимизированный алгоритм Винограда;
- проведен сравнительный анализ по времени алгоритмов умножения матриц на четных размерах матриц
- проведен сравнительный анализ по времени алгоритмов умножения матриц на нечетных размерах матриц
- проведен сравнительный анализ по времени алгоритмов алгоритмов между собой
- подготовлен отчет о лабораторной работе.