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 Козлова И.В.

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Оглавление

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 любая числовая информация, занесенная в таблицу, уже в какой-то степени считается матрицей.

Целью работы 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.

Для достижения цели ставятся следующие задачи.

1. Изучить классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

2. Реализовать классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

3. Дать оценку трудоёмкости алгоритмов.

4. Замерить время работы алгоритмов.

5. Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

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. Например, при умножении матрицы размером 3x4 на матрицу размером 4x7 мы получаем матрицу размером 3x7.

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

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

тогда матрица C

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

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 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 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 (1.4):

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

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 толь-

ко 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 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 рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

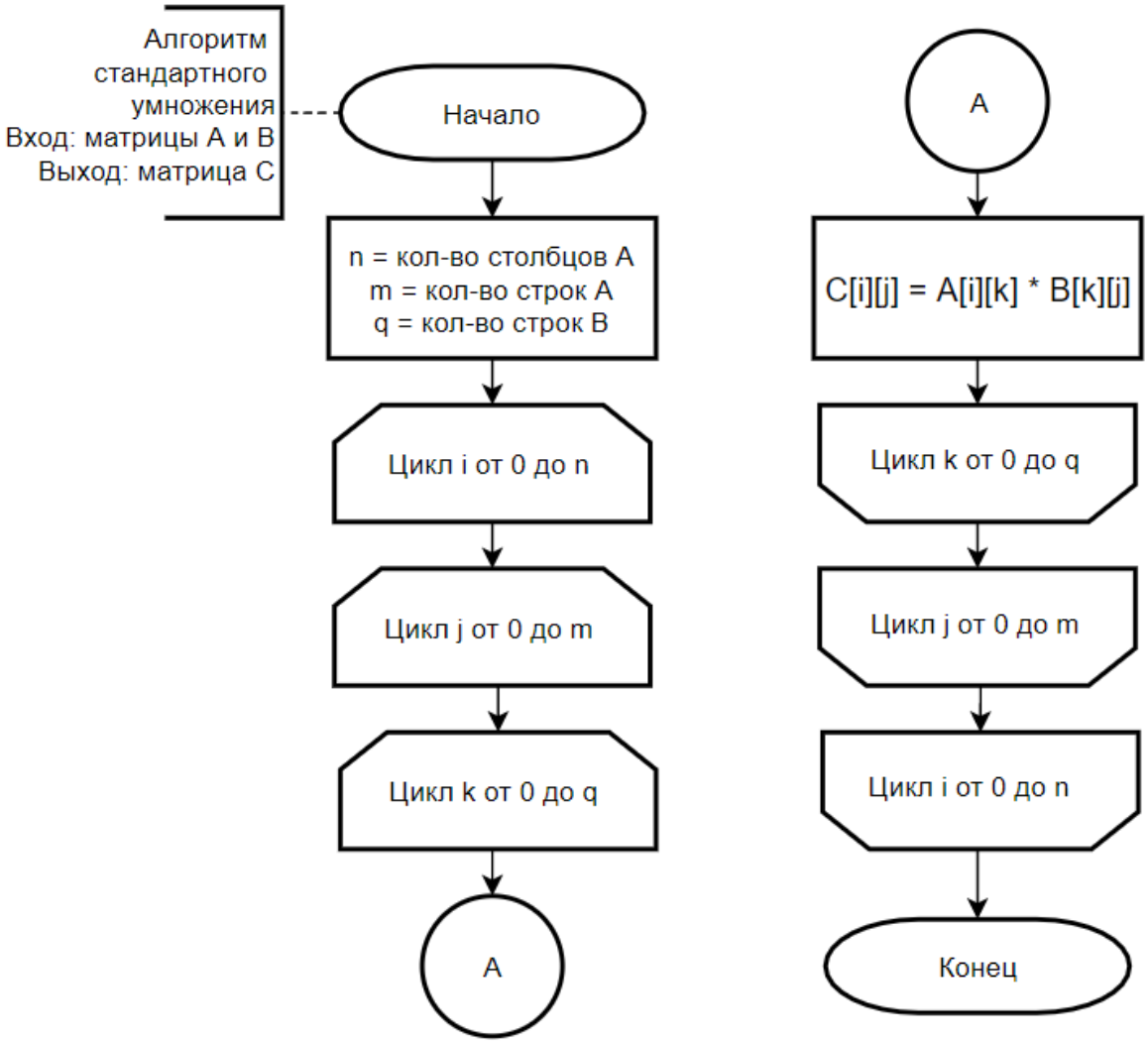

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

На рисунках 2.2 и 2.3 приведена схема алгоритма Винограда умножения матриц.

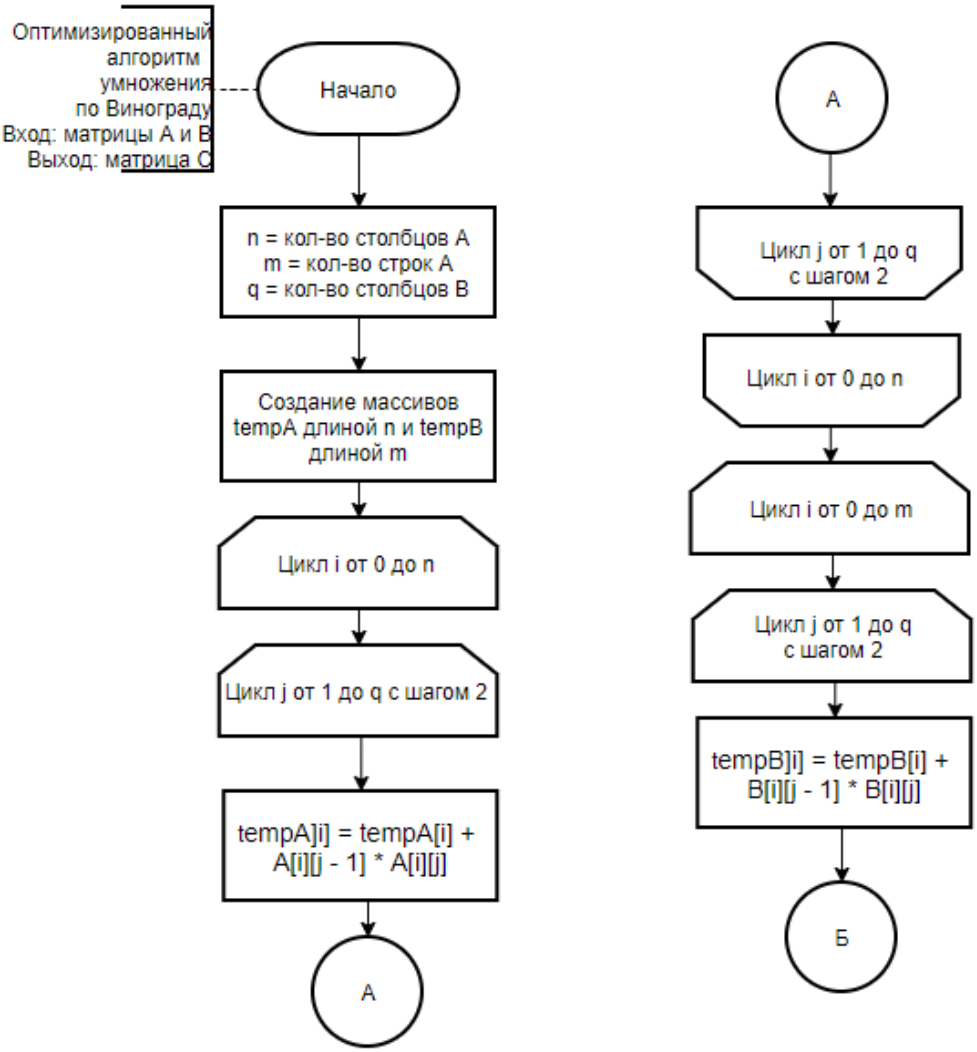

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц

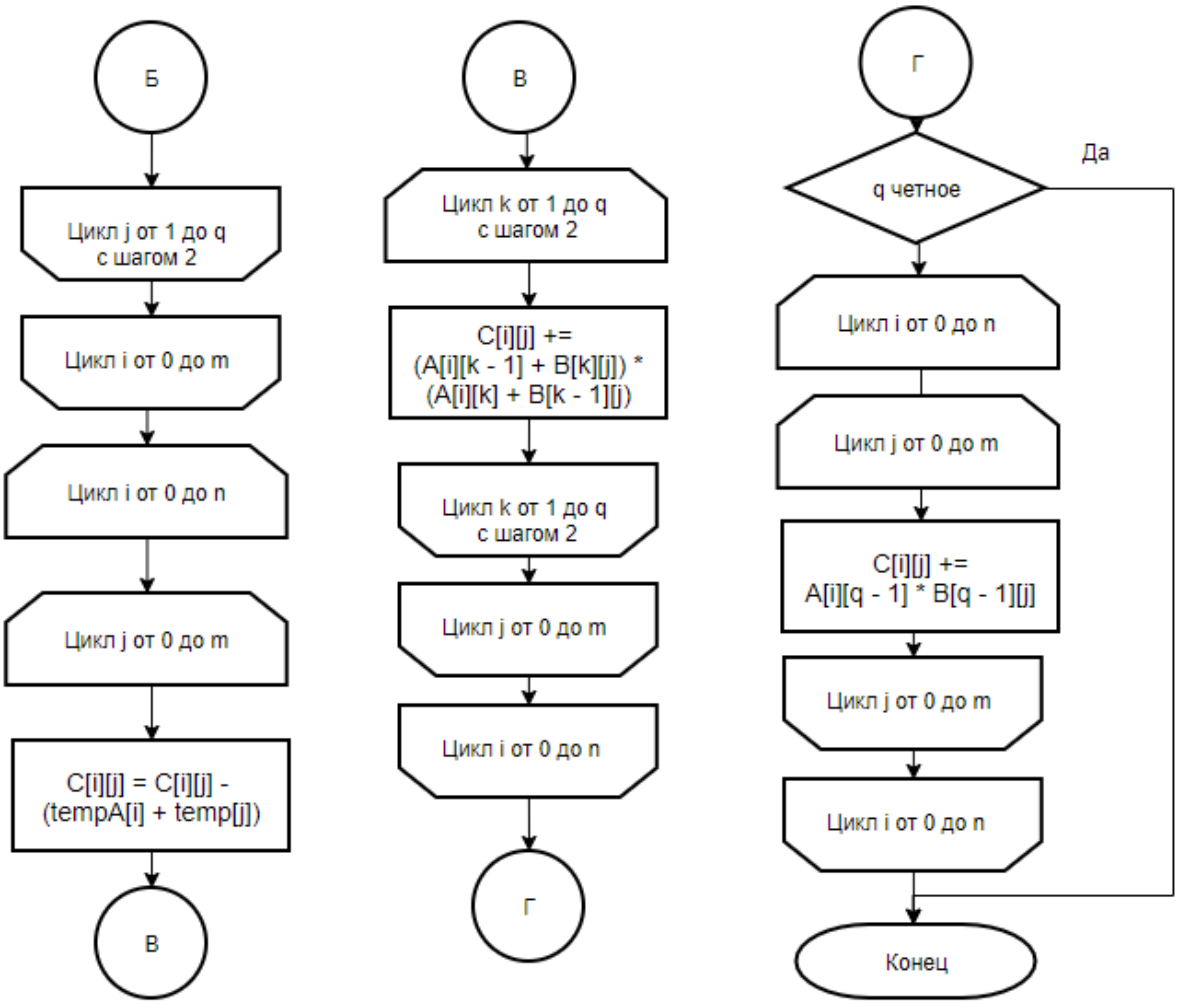

Рисунок 2.3 – Схема алгоритма Винограда умножения матриц(продолжение)

На рисунках 2.4 и 2.5 приведена схема оптимизировванного алгоритма Винограда умножения матриц.

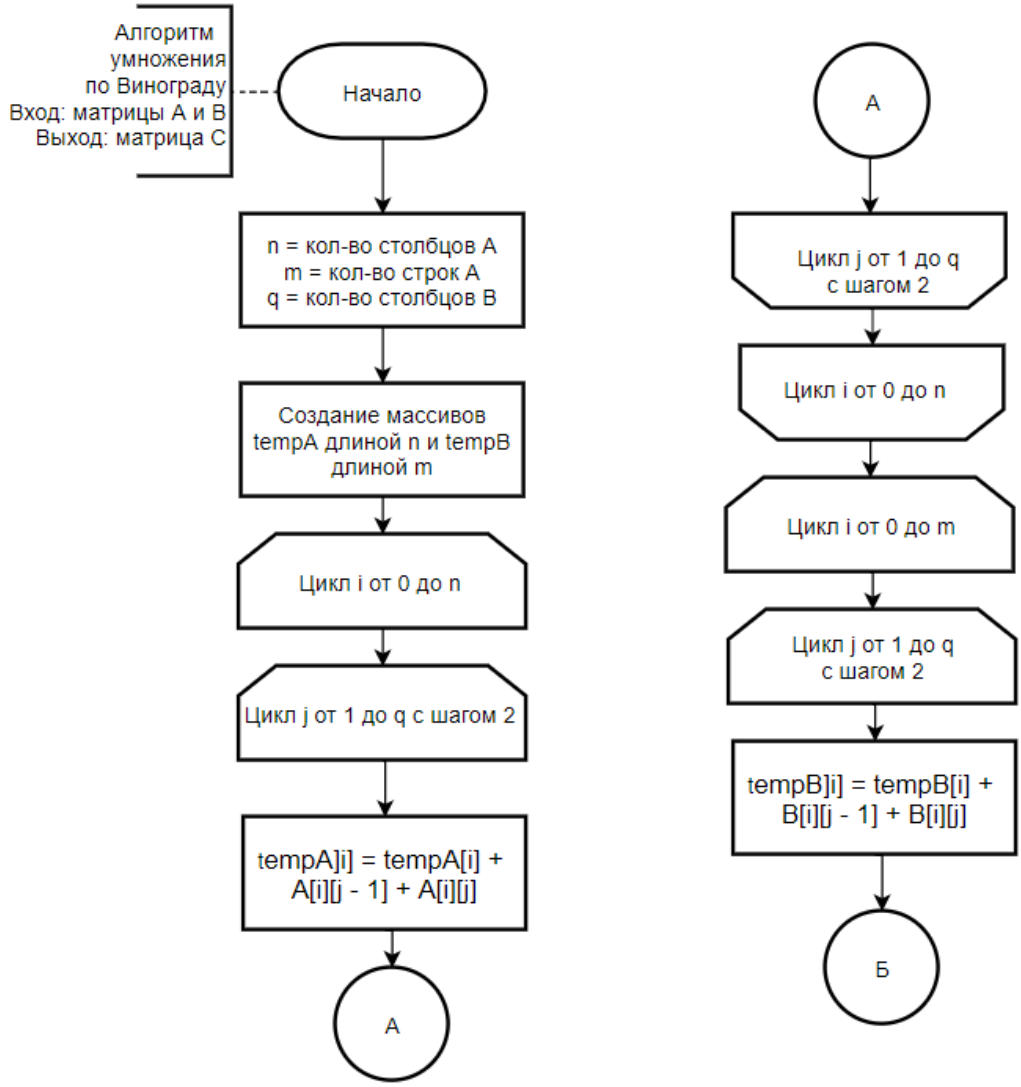

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда умножения матриц

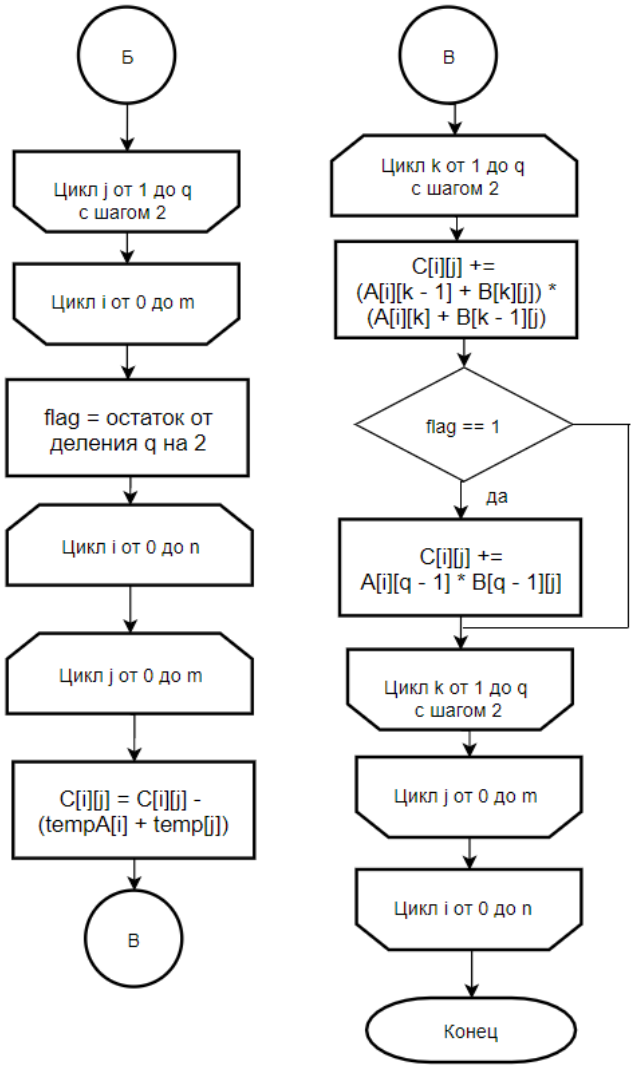

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда умножения матриц(продолжение)

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 будут расчитаны трудоемкости алгоритмов умножения матриц.

2.3.1 Стандартный алгоритм 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.

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):

$$f_{standard} = 2+M\cdot(4+N\cdot(4+10K)) = 2+4M+4MN+10MNK \approx 10MNK \tag{2.4}$$

2.3.2 Алгоритм Копперсмита — Винограда

Трудоёмкость алгоритма Копперсмита — Винограда состоит из:

- создания и инициализации массивов MH и MV, трудоёмкость которого (2.5):
$$f_{init} = M + N; \tag{2.5}$$
- заполнения массива MH, трудоёмкость которого (2.6):
$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 11); \tag{2.6}$$
- заполнения массива MV, трудоёмкость которого (2.7):
$$f_{MV} = 2 + K(2 + \frac{N}{2} \cdot 11); \tag{2.7}$$
- цикла заполнения для чётных размеров, трудоёмкость которого (2.8):
$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)); \tag{2.8}$$
- цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный, трудоемкость которого

(2.9):

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

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

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

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

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

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;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

Трудоёмкость улучшенного алгоритма Копперсмита — Винограда состоит из:

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

$$f_{init} = M + N; \tag{2.12}$$

- заполнения массива MH, трудоёмкость которого (2.13):

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

- заполнения массива MV, трудоёмкость которого (2.14):

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

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

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

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

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

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

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

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

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

Вывод

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

3 Технологическая часть

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

3.1 Требования к ПО

К программе предъявляется ряд требований.

- Входными данными являются две матрицы A и B. Количество столбцов матрицы A долджно 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 ЯП Python [4].

Данный язык достаточно удобен и гибок в использовании.

Время работы алгоритмов было замерено с помощью функции process_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 замерами 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 стандарного умножения матриц

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

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

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

    for i in range(n):
        for j in range(q):
            for k in range(m):
                matrix_res[i][j] = matrix_res[i][j] + \
                    matrix1[i][k] * matrix2[k][j]

    return matrix_res
```

Листинг 3.2 – Реализация алгоритма Копперсмита-Винограда

```
def winograd_matrix_mult(matrix1, matrix2):
    if (len(matrix2) != len(matrix1[0])):
        print("Error size matrix!")
        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
```

```
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                      (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 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] +
                        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 – Тестирование функций

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

Вывод

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

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:

- операционная система: Ubuntu 20.04.3 [6] Linux [7] x86_64;
- память: 8 GiB;
- процессор: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz [8].

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 Демонстрация работы программы

На рисунке 2.1 представлен результат работы программы.

```
Введите матрицу 1:
Введите количество строк: 3
Введите количество столбцов: 3
Введите строку: 1 2 3
Введите строку: 1 2 3
Введите строку: 1 2 3
Введите матрицу 2:
Введите количество строк: 3
Введите количество столбцов: 3
Введите строку: 1 2 3
Введите строку: 1 2 3
Введите строку: 1 2 3
Результат, посчитанный классическим алгоритмом:
[6, 12, 18]
[6, 12, 18]
[6, 12, 18]
Результат, посчитанный алгоритмом Винограда:
[6, 12, 18]
[6, 12, 18]
[6, 12, 18]
Результат, посчитанный оптимизированным алгоритмом Винограда:
[6, 12, 18]
[6, 12, 18]
[6, 12, 18]
```

```
Введите матрицу 1:
Введите количество строк: 1
Введите количество столбцов: 2
Введите строку: 1 2
Введите матрицу 2:
Введите количество строк: 1
Введите количество столбцов: 2
Введите строку: 1 2
Результат, посчитанный классическим алгоритмом:
Error size matrix!
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

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

На рисунках 4.2 и 4.3 приведены графики зависимостей времени работы алгоритмов от размеров матриц.

Время в таблицах и на рисунках приведены в секундах.

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

Размер	AB	CA	OAB
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

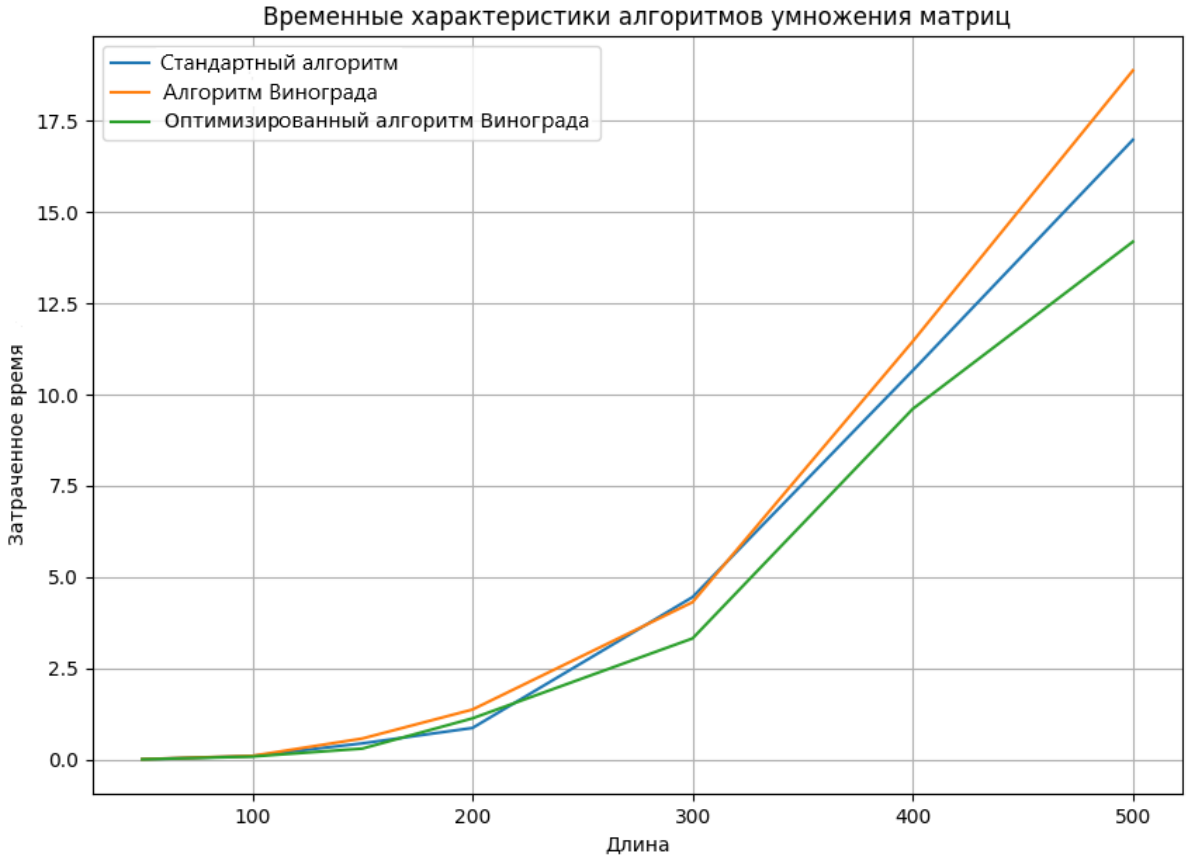

Рисунок 4.2 – Зависимость времени работы алгоритма от четного размера квадратной матрицы (сек)

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

Размер	AB	CA	OAB
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

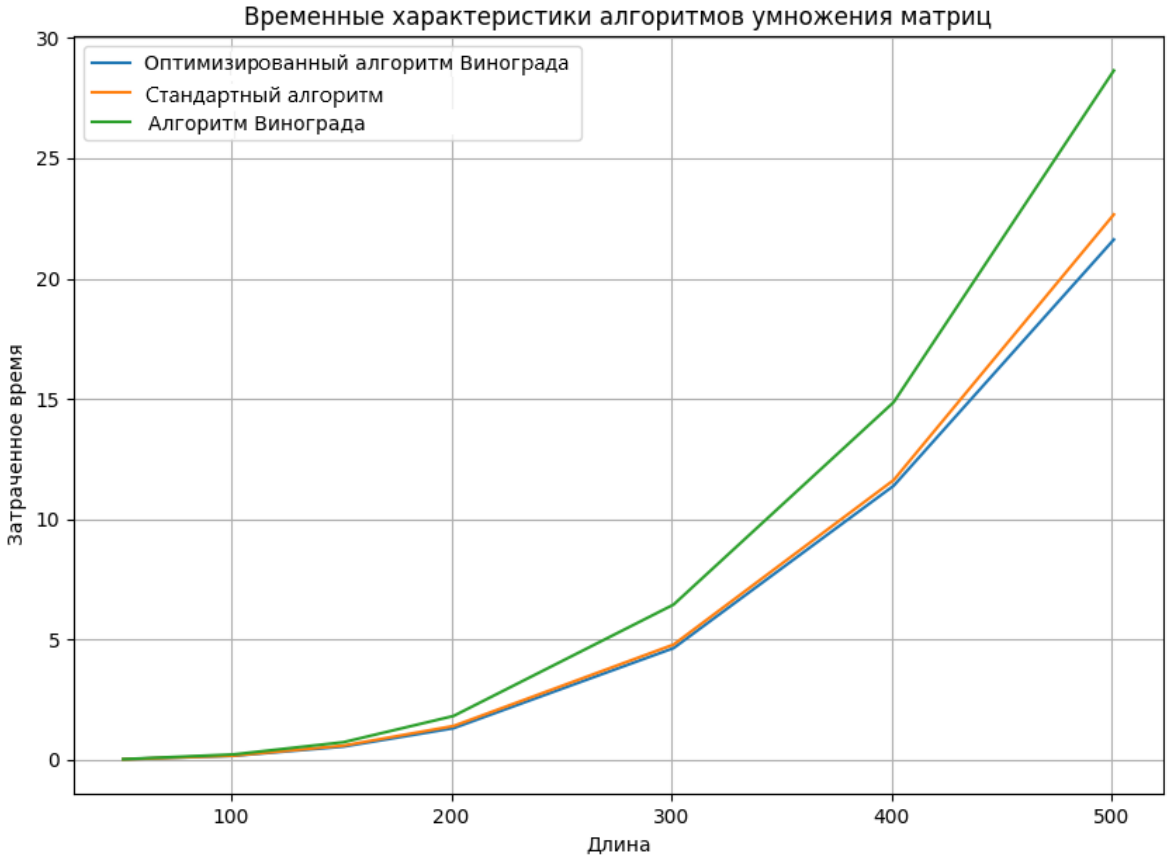

Рисунок 4.3 – Зависимость времени работы алгоритма от нечетного размера квадратной матрицы (сек)

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: обычный, 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;
- подготовлен отчет о лабораторной работе.

Поставленная цель достигнута.

Литература

[1] Матрица [Электронный ресурс]. Режим доступа: https://terme.ru/termin/matrica.html (дата обращения: 13.09.2021).

[2] Умножение матриц [Электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 13.09.2021).

[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).

[7] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 04.09.2021).

[8] Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 04.09.2021).