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»

Лабораторная работа №2
по дисциплине «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Тузов Даниил Александрович

Группа ИУ7-52Б

Преподаватель Кормановский Михаил Владимирович

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитическая часть **4**

1.1 Умножение матриц стандартным алгоритмом 4

1.2 Алгоритм Винограда 4

1.3 Вывод 5

2 Конструкторская часть **6**

2.1 Описание алгоритмов 6

2.2 Модель вычислений 11

2.3 Вывод 12

3 Технологическая часть **13**

3.1 Средства реализации 13

3.2 Реализация алгоритмов 13

3.3 Функциональные тесты 16

3.4 Оценка трудоемкости 16

3.4.1 Стандартный алгоритм 17

3.4.2 Неоптимизированный алгоритм Винограда 17

3.4.3 Оптимизированный алгоритм Винограда 18

3.5 Вывод 18

4 Исследовательская часть **19**

4.1 Технические характеристики ЭВМ 19

4.2 Сравнение по времени 19

4.3 Вывод 20

ЗАКЛЮЧЕНИЕ **21**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **22**

ВВЕДЕНИЕ

Во второй лабораторной работе по анализу алгоритмов рассматриваются алгоритмы умножения матриц: стандартный и алгоритм Винограда.

Целью лабораторной работы является оценка ресурсной эффективности алгоритмов умножения матриц. Для достижения поставленной цели небходимо решить следующие задачи:

- описать математическую основу стандартного алгоритма умножения матриц и алгоритма Винограда
- описать модель вычислений
- разработать алгоритмы умножения матриц: стандартный, Винограда и оптимизированный алгоритм Винограда в соответствии со своим вариантом
- выполнить оценку трудоемкости разработанных алгоритмов
- реализовать разработанные алгоритмы в программном обеспечении с двумя режимами работы – одиночного рассчета и массированного замера
- выполнить замеры процессорного времени выполнения реализации разработанных алгоритмов в зависимости от варьируемого линейного размера матриц
- выполнить сравнительный анализ рассчитанных трудоемкостей и результатов замера процессорного времени выполнения реализаций трех алгоритмов с учетом лучшего и худшего случаев по трудоемкости

1 Аналитическая часть

В этой части рассматриваются теоретические аспекты понятия умножения матриц стандартным алгоритмом и с помощью алгоритма Винограда.

1.1 Умножение матриц стандартным алгоритмом

Пусть даны две матрицы A и B размерности $N \times M$ и $M \times K$ соответственно:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mk} \end{pmatrix}, \tag{1}$$

тогда произведением матриц A и B называется матрица C размерностью $N \times K$:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nk} \end{pmatrix}, \tag{2}$$

такая, что

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \quad (i = \overline{1, N}; j = \overline{1, K}) \tag{3}$$

1.2 Алгоритм Винограда

В стандартном алгоритме для вычисления j-ого значения в i-ой строке результирующей матрицы необходимо поссчитать скалярное произведение i-ой строки $U = (u_1, u_2, u_3, ..., u_n)$ первой матрицы на j-ый столбец $V = (v_1, v_2, v_3, ..., v_n)$ второй матрицы, что равно $U \cdot V = u_1v_1 + u_2v_2 + u_3v_3 + ... + u_nv_n$.

Виноград в своем алгоритме предлагает другой способ вычисления скалярного произведения. Этот способ основан на снижении доли операций умножения. Пусть есть два вектора $U = (u_1, u_2, u_3, u_4)$ и $V = (v_1, v_2, v_3, v_4)$, тогда их

скалярное произведение $U \cdot V = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4$, что эквивалентно (4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \quad (4)$$

Произведения v_1v_2, v_3v_4, w_1w_2 и w_3w_4 можно поссчитать заранее, а затем использовать их при вычислении значений в результирующей матрице.

1.3 Вывод

В этой части были рассмотрены теоретические аспекты алгоритмов умножения матриц. Основное отличие алгоритмов заключается в способе вычисления скалярного произведения. В алгоритме Винограда выполняется предварительная обработка, которая в дальнейшем помогает снизить долю операций умножения.

2 Конструкторская часть

В этой части представлены описания алгоритмов, а также модель вычисления трудоемкостей алгоритмов.

2.1 Описание алгоритмов

На вход алгоритмам подаются две матрицы A и B. На выходе результат умножения матриц A и B. В случае, если количество строк первой матрицы не совпадает с количеством столбцов второй матрицы, программа вывводит сообщение о том, что ответ не был получен из-за неправильных размеров матриц.

На рисунках 1 - 5 приведены описания алгоритмов.

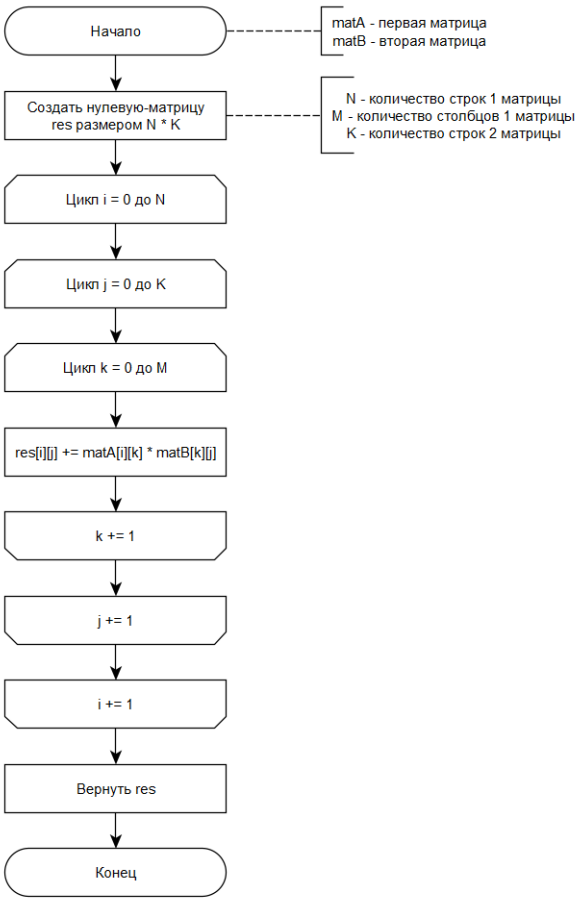

Рисунок 1 – Описание стандартного алгоритма умножения матриц

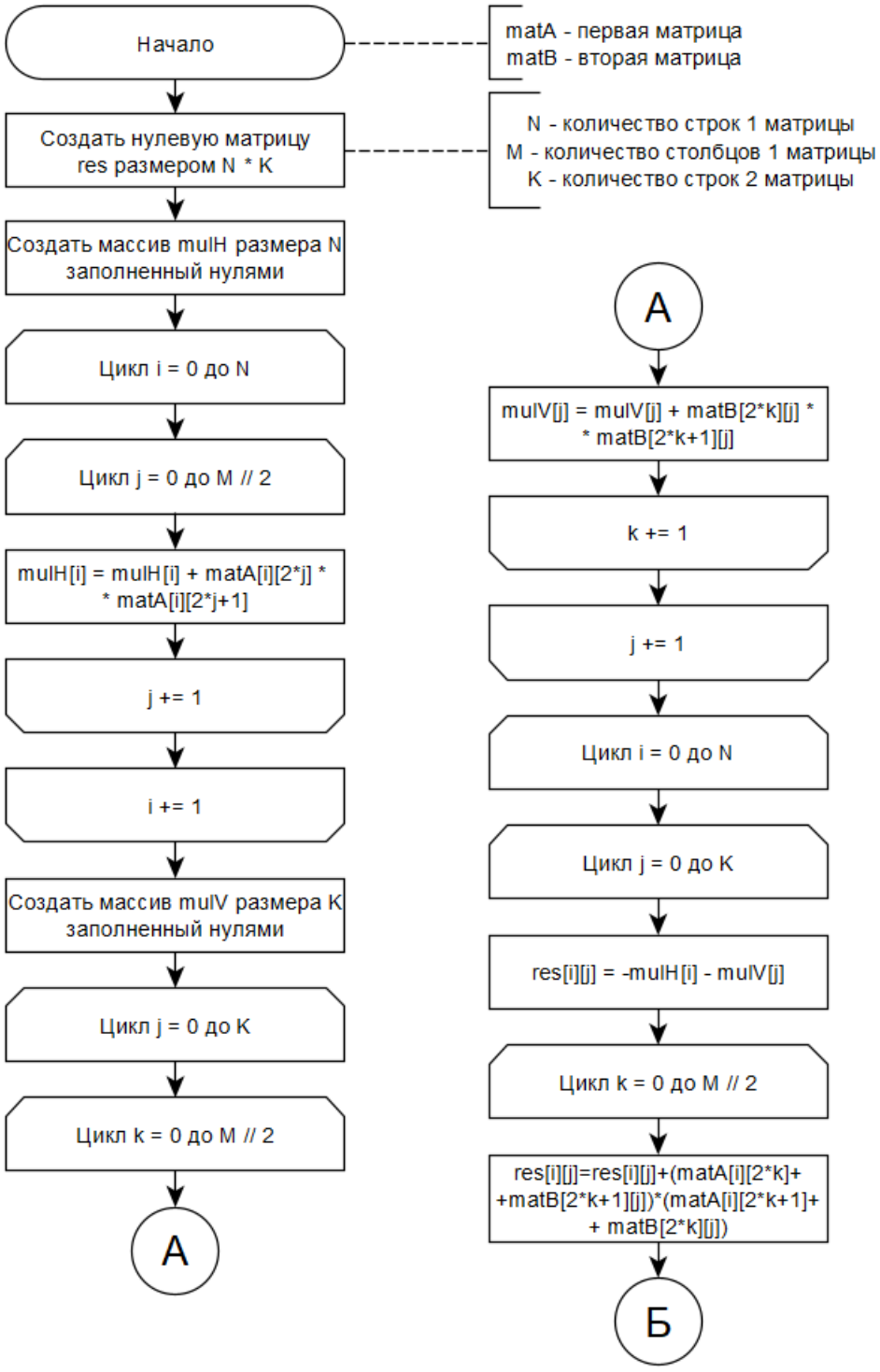

Рисунок 2 – Описание неоптимизированного алгоритма Винограда

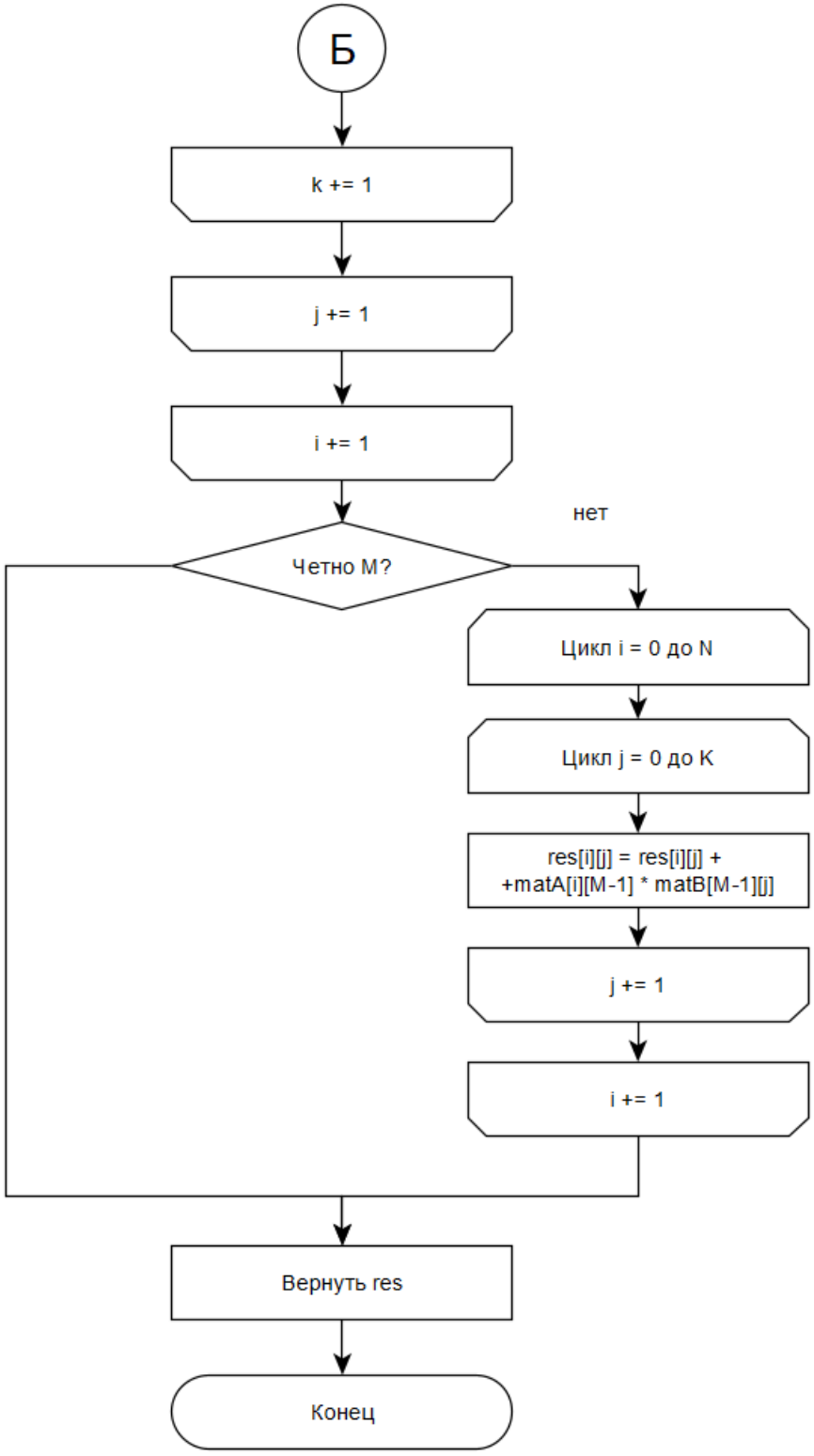

Рисунок 3 – Описание неоптимизированного алгоритма Винограда (продолжение)

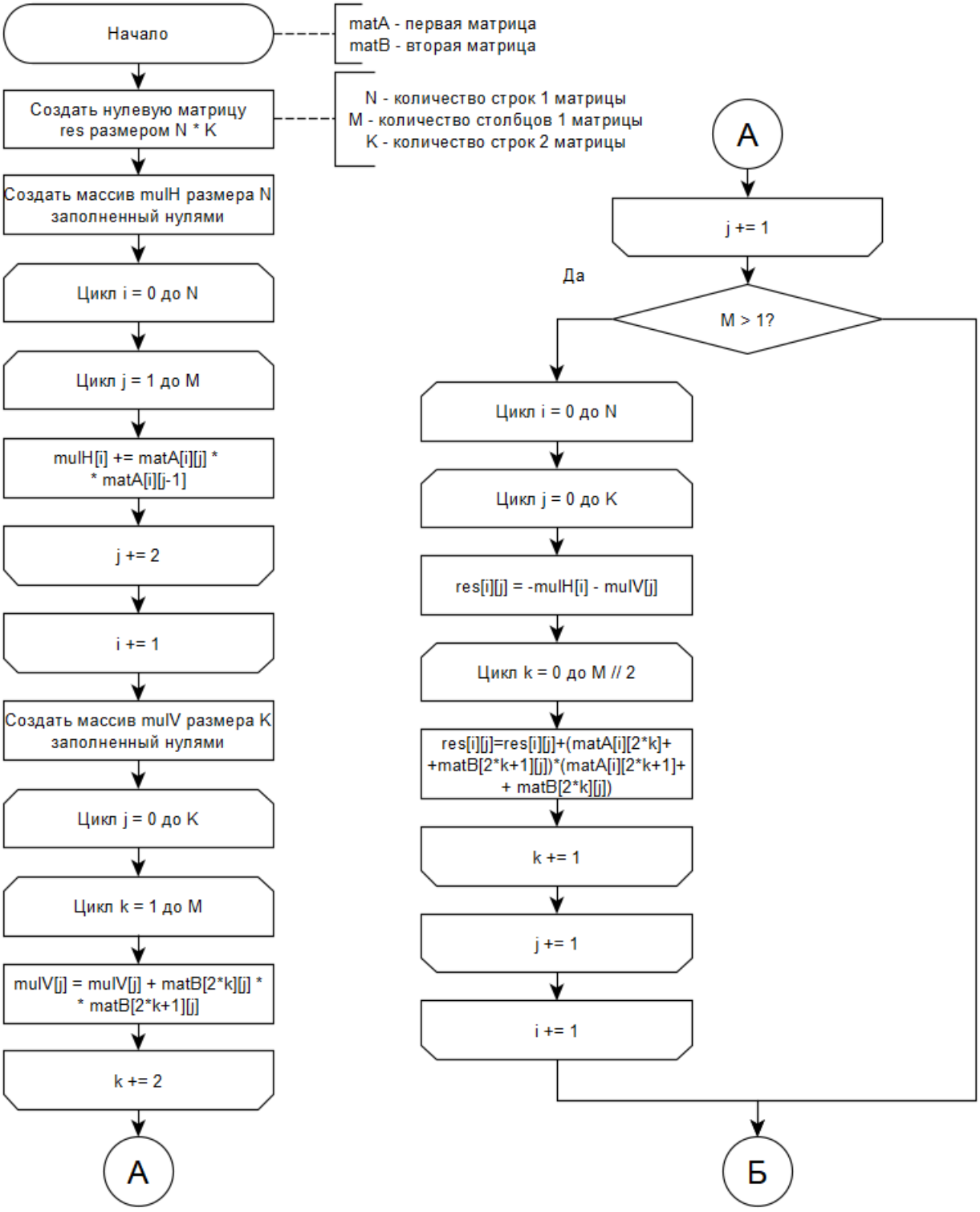

Рисунок 4 – Описание оптимизированного алгоритма Винограда

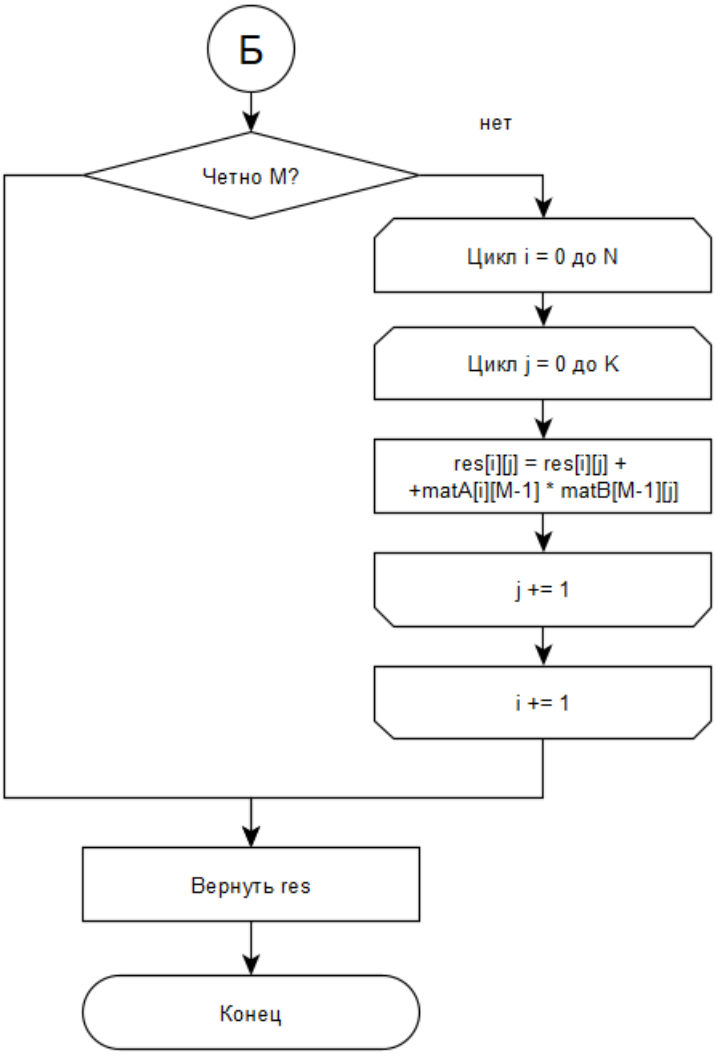

Рисунок 5 – Описание оптимизированного алгоритма Винограда (продолжение)

2.2 Модель вычислений

Для вычисления трудоемкостей алгоритмов необходимо ввести модель вычислений:

1. операции $+, -, [], ++, --, ==, !=, <, >, <=, >=$ имеют трудоемкость 1;
2. операции $*, /, \%, /=, *=, \%=$ имеют трудоемкость 2;
3. трудоемкость условного оператора `if <condition> then A else B` рассчитывается, как (5);

$$f_{if} = f_{\text{condition}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{5}$$

4. трудоемкость цикла `for (init; condition; increment) { body }` рассчитывается, как (6)

$$f_{for} = f_{\text{init}} + f_{\text{condition}} + N \cdot (f_{\text{body}} + f_{\text{condition}} + f_{\text{increment}}) \tag{6}$$

   где N – количество итераций цикла;
5. трудоемкость вызова функции равна 0.

2.3 Вывод

В этом разделе на основе теоретических аспектов были представлены описания стандартного алгоритма умножения матриц, неоптимизированного алгоритма Винограда и оптимизированного алгоритма Винограда, который включает в себя следующие оптимизации:

1. инкремент счётчика наиболее вложенного цикла на 2;
2. использование инкремента (+=);
3. вынос начальной итерации из каждого внешнего цикла.

Так же была описана модель вычислений для дальнейшей оценки трудоемкости алгоритмов.

3 Технологическая часть

В этом разделе обоснованы средства реализации, а так же представлены реализации алгоритмов, функциональные тесты и оценки трудоемкостей алгоритмов.

3.1 Средства реализации

Для реализации алгоритмов в этой лабораторной работы был выбран язык *Python*[2], потому что в нем нет автоматического сборщика мусора. Время работы было замерено на микроконтроллерах *STM32*[5], с помощью функции *ticks_ms*[1] из библиотеки *time* утилиты *MicroPython*[3]. Запуск программы проводился утилитой *ampy*[4].

3.2 Реализация алгоритмов

В листингах 1 - 3 представлены коды написанных алгоритмов.

Листинг 1 – Стандратный алгоритм умножения матриц

```
1  def standard_mult(mat_a, mat_b):
2      if len(mat_a) == 0 or len(mat_b) == 0:
3          return []
4      na = len(mat_a)
5      ma = len(mat_a[0])
6      nb = len(mat_b)
7      mb = len(mat_b[0])
8      if ma != nb:
9          return []
10
11     mat_c = [[0] * mb for i in range(na)]
12     for i in range(na):
13         for j in range(mb):
14             for k in range(ma):
15                 mat_c[i][j] = mat_c[i][j] + mat_a[i][k] * mat_b[k][j]
16
17     return mat_c
```

Листинг 2 – Неоптимизированный алгоритм Винограда

```
def vinograd_alg(mat_a, mat_b):
    if len(mat_a) == 0 or len(mat_b) == 0:
        return []
    na = len(mat_a)
    ma = len(mat_a[0])
    nb = len(mat_b)
    mb = len(mat_b[0])
    if ma != nb:
        return []

    mat_c = [[0] * mb for i in range(na)]

    mul_h = [0] * na
    for i in range(na):
        for j in range(ma // 2):
            mul_h[i] = mul_h[i] + mat_a[i][2 * j] * mat_a[i][2 * j + 1]

    mul_v = [0] * mb
    for j in range(mb):
        for k in range(nb // 2):
            mul_v[j] = mul_v[j] + mat_b[2 * k][j] * mat_b[2 * k + 1][j]

    for i in range(na):
        for j in range(mb):
            mat_c[i][j] = -mul_h[i] - mul_v[j]
            for k in range(ma // 2):
                mat_c[i][j] = mat_c[i][j] + (mat_a[i][2 * k] + mat_b[2 * k +
                    1][j]) * (mat_a[i][2 * k + 1] + mat_b[2 * k][j])

    if ma % 2 == 1:
        for i in range(na):
            for j in range(mb):
                mat_c[i][j] = mat_c[i][j] + mat_a[i][ma - 1] * mat_b[nb - 1][j]

    return mat_c
```

Листинг 3 – Оптимизированный алгоритм Винограда

```
def optimized_vinograd_alg(mat_a, mat_b):
    if len(mat_a) == 0 or len(mat_b) == 0:
        return []
    na = len(mat_a)
    ma = len(mat_a[0])
    nb = len(mat_b)
    mb = len(mat_b[0])
    if ma != nb:
        return []

    mat_c = [[0] * mb for i in range(na)]

    mul_h = [0] * na
    for i in range(na):
        for j in range(1, ma, 2):
            mul_h[i] += mat_a[i][j] * mat_a[i][j - 1]

    mul_v = [0] * mb
    for j in range(mb):
        for k in range(1, nb, 2):
            mul_v[j] += mat_b[k][j] * mat_b[k - 1][j]

    if ma > 1:
        for i in range(na):
            for j in range(mb):
                mat_c[i][j] = (mat_a[i][0] + mat_b[1][j]) * (mat_a[i][1] +
                    mat_b[0][j]) - mul_h[i] - mul_v[j]
                for k in range(2, ma - 1, 2):
                    mat_c[i][j] += (mat_a[i][k] + mat_b[k + 1][j]) * (mat_a[i][
                        k + 1] + mat_b[k][j])

    if ma % 2 == 1:
        for i in range(na):
            for j in range(mb):
                mat_c[i][j] += mat_a[i][ma - 1] * mat_b[nb - 1][j]

    return mat_c
```

3.3 Функциональные тесты

В таблице 1 приведены функциональные тесты, на которых тестировалась программа.

Таблица 1 – Функциональные тесты, на которых были протестированы алгоритмы

Первая матрица	Вторая матрица	Результат
Пустая матрица	Пустая матрица	Сообщение об ошибке
$\begin{pmatrix}1\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$
$\begin{pmatrix}1 & 2\end{pmatrix}$	$\begin{pmatrix}2\\1\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$
$\begin{pmatrix}1 & 2\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix}1\\2\end{pmatrix}$	$\begin{pmatrix}2 & 1\end{pmatrix}$	$\begin{pmatrix}2 & 1\\4 & 2\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 3\\3 & 2 & 1\end{pmatrix}$	$\begin{pmatrix}2 & 1\\3 & 0\\4 & 2\end{pmatrix}$	$\begin{pmatrix}20 & 7\\16 & 5\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	$\begin{pmatrix}1\\2\\3\end{pmatrix}$	$\begin{pmatrix}14\\32\end{pmatrix}$
$\begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 1\\1 & 1\end{pmatrix}$	$\begin{pmatrix}2 & 2\\2 & 2\end{pmatrix}$
$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	$\begin{pmatrix}-1 & 0\\0 & -1\end{pmatrix}$	$\begin{pmatrix}-1 & 0\\0 & -1\end{pmatrix}$
$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$	$\begin{pmatrix}0 & 0\\0 & 0\end{pmatrix}$

3.4 Оценка трудоемкости

В этом разделе приведены оценки трудоемкостей алгоритмов согласно приведенным реализациям. При оценке алгоритма в дальнейшем не будет учитываться проверка входных данных и инициализация результирующей матрицы, так как эти действия являются общими во всех алгоритмах. Для рассчета трудоемкости примем $N = na, M = nb = ma, K = mb$.

3.4.1 Стандартный алгоритм

Трудоемкость стандартного алгоритма рассчитывается, как:

1. цикла по i (12 строка), трудоемкость $f_{standard} = 2 + N \cdot (2 + f_{body_i})$;
2. цикла по j (13 строка), трудоемкость $f_{body_i} = 2 + K \cdot (2 + f_{body_j})$;
3. цикла по k (14 строка), трудоемкость $f_{body_j} = 2 + M \cdot (2 + f_{body_k}$;
4. тело самого вложенного цикла (15 строка), трудоемкость $f_{body_k} = 12$

тогда $f_{standard} = 2 + N \cdot (4 + K \cdot (4 + 14M)) = 14MNK + 4NK + 4N + 2$

$$f_{standard} = 14MNK + 4NK + 4N + 2 \approx 10MNK \tag{7}$$

3.4.2 Неоптимизированный алгоритм Винограда

Трудоемкость неоптимизированного алгоритма Винограда рассчитывается, как:

1. инициализация нулями массивов mul_h и mul_v (13 и 18 строки), трудоемкость $f_{init} = N + K$;
2. заполнение массива mul_h (14-16 строки) $f_{inith} = 2 + N \cdot (2 + 4 + \frac{M}{2} \cdot (4 + 15) = \frac{19}{2}MN + 6N + 2$;
3. заполнение массива mul_v (19-21 строки) $f_{initv} = 2 + K \cdot (2 + 4 + \frac{M}{2} \cdot (4 + 15) = \frac{19}{2}MK + 6K + 2$;
4. заполнение результирующей матрицы (23-27 строки) $f_{mult} = 2 + N \cdot (2 + 2 + K \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 28) = 16MNK + 13NK + 4N + 2$
5. проверка M на нечетность (29-32 строки) $f_{is} = \begin{cases} 3, & \text{четно}, \\ 16NK + 4N + 5, & \text{иначе}. \end{cases}$

$$f_{vinograd} = \begin{cases} 16MNK + 29NK + \frac{19}{2}MK + \frac{19}{2}MN + 15N + 7K + 11, & \text{х.с.}, \\ 16MNK + 13NK + \frac{19}{2}MK + \frac{19}{2}MN + 11N + 7K + 9, & \text{л.с.} \end{cases} \tag{8}$$

3.4.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда рассчитывается, как:

1. инициализация нулями массивов mul_h и mul_v (13 и 18 строки), трудоемкость $f_{init} = N + K$;

2. заполнение массива mul_h (14-16 строки) $f_{inith} = 2+N\cdot(2+2+\frac{M}{2}\cdot(2+9) = \frac{11}{2}MN + 4N + 2$;

3. заполнение массива mul_v (19-21 строки) $f_{initv} = 2+K\cdot(2+2+\frac{M}{2}\cdot(2+9) = \frac{11}{2}MK + 4K + 2$;

4. заполнение результирующей матрицы (23-28 строки) $f_{mult} = 3 + N\cdot(2+2+K\cdot(2+6+2+\frac{M}{2}\cdot(2+17) = \frac{19}{2}MNK + 10NK + 4N + 3$

5. проверка M на нечетность (30-33 строки) $f_{is} = \begin{cases} 3, & \text{четно,} \\ 13NK + 4N + 5, & \text{иначе.} \end{cases}$

$$f_{vinograd} = \begin{cases} \frac{19}{2}MNK + 23NK + \frac{11}{2}MK + \frac{11}{2}MN + 13N + 5K + 12, & \text{х.с.,} \\ \frac{19}{2}MNK + 10NK + \frac{11}{2}MK + \frac{11}{2}MN + 9N + 5K + 10, & \text{л.с.} \end{cases} \quad (9)$$

3.5 Вывод

В этом разделе на основе описаний алгоритмов были написаны и представлены коды алгоритмов. Помимо кодов представлены функциональные тесты, на которых был протестирован каждый алгоритм. Так же приведена оценка трудоемкости по модели вычислений, приведенной в предыдущей части. Согласно этой оценке наименьшая сложность у оптимизированного алгоритма Винограда, затем у стандартного и только потом у неоптимизированного алгоритма Винограда.

4 Исследовательская часть

В этом разделе приведено сравнение по времени и по памяти написанных алгоритмов.

4.1 Технические характеристики ЭВМ

Все замеры проводились с помощью микроконтроллера STM32 на ЭВМ, характеристики которой приведены ниже:

- Процессор – 12th Gen Intel(R) Core(TM) i5-12450H 2.00 ГГц
- Оперативная память – 16,0 ГБ
- Тип системы – 64-разрядная операционная система, процессор x64
- Операционная система – Windows 11
- Версия ОС – 23H2

4.2 Сравнение по времени

Время выполнения работы алгоритмов представлено в секундах. Проводилось 100 замеров.

Таблица 2 – Сравнение алгоритмов по времени выполнения

Размер квадратной матрицы	Стандартный	Виноград	Оптимизированный
20	0.0003	0.0001	0.0003
40	0.002	0.002	0.002
60	0.008	0.009	0.007
80	0.019	0.019	0.014
100	0.037	0.035	0.027

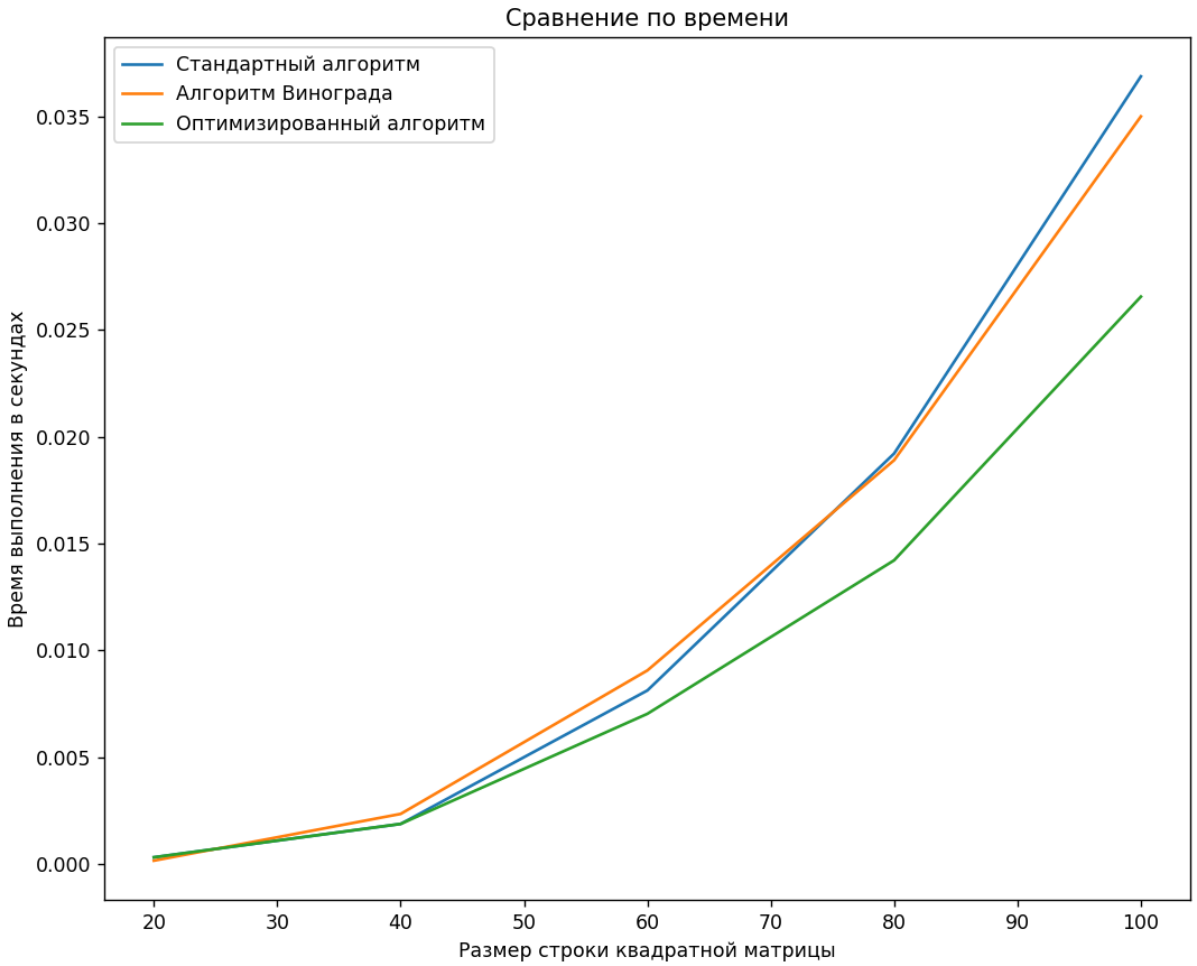

Рисунок 6 – Сравнение алгоритмов по времени выполнения

4.3 Вывод

Неоптимизированный алгоритм Винограда, несмотря на большую теоретическую сложность, работает быстрее стандартного алгоритма. Алгоритм Винограда с оптимизацией работает быстрее всех алгоритмов примерно на 37%.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы поставленная цель была достигнута, а также были решены следующие задачи:

- описана математическая основа стандартного алгоритма умножения матриц и алгоритма Винограда;
- описана модель вычислений трудоемкости алгоритмов;
- реализованы алгоритмы умножения матриц;
- выполнена оценка трудоемкости алгоритмов, согласно которой самый простой алгоритм – оптимизированный алгоритм Винограда;
- проведено сравнение алгоритмов по времени:
  - несмотря на свою большую сложность алгоритм Винограда работает быстрее стандартного алгоритма;
  - самым быстрым оказался оптимизированный алгоритм Винограда;
- описаны и обоснованы полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Функция ticks_ms – https://docs.micropython.org/en/latest/library/time.html
2. Welcome to Python – https://www.python.org
3. MicroPython – https://docs.micropython.org
4. Утилита ampy – https://ampy.readthedocs.io/
5. STM32 – https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/documentation.html