**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** Алгоритмы умножения матриц

**Студент** Косарев А.А.

**Группа** ИУ7-51Б

**Оценка (баллы)**

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

Москва — 2022 г.

# Содержание

# Введение

В программировании перемножение матриц является одним из базовых алгоритмов, который широко применяется в различных математических задачах, численных методах и машинном обучении. Также существует большое количество вариантов оптимизаций, используемых для уменьшения времени работы реализаций выбранных методов.

В данном отчете будут рассмотрены следующие алгоритмы умножения матриц:

- классический;
- Винограда;
- Винограда с оптимизациями.

# 1 Аналитическая часть

## 1.1 Цель и задачи

**Цель** – изучение и исследование алгоритмов перемножения матриц.

Для достижения поставленной цели следует решить следующие задачи:

- разработать алгоритмы перемножения матриц (классический, Винограда и Винограда с оптимизациями);
- реализовать разработанные алгоритмы;
- вывести оценку трудоемкости алгоритмов;
- выполнить замеры процессорного времени работы реализаций алгоритмов;
- провести сравнительный анализ заданных алгоритмов сортировки по затрачнному времени работы реализаций.

## 1.2 Классический алгоритм умножения матриц

Произведением матриц $A$ и $B$ называется матрица $C$ такая, что число строк и столбцов матрицы $C$ равно количеству строк матрицы $A$ и столбцов матрицы $B$ соответственно [1].

Необходимым условием умножения двух матриц является равнство количества столбцов первой матрицы количеству строк второй матрицы.

Пусть даны матрицы $A$ $[a \times b]$ и $B$ $[c \times d]$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \tag{1.1}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}, \tag{1.2}$$

тогда произведением матриц $A$ и $B$ будет считаться матрица $C$ [$a$ x $d$]:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}, \tag{1.3}$$

где

$$C_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj},\ i = 1, 2, ..., a;\ j = 1, 2, ..., d. \tag{1.4}$$

Классический алгоритм умножения двух матриц работает по формуле 1.4.

## 1.3 Алгоритм Винограда

В матрице, которая является результатом произведения двух других матриц, каждый элемент представляет собой скалярное произведение соответствующих строки и столбца исходных матриц [2]. Рассмотрим два вектора:

$$P = \begin{pmatrix} p_1, & p_2, & p_3, & p_4 \end{pmatrix}, \tag{1.5}$$

$$Q = \begin{pmatrix} q_1, & q_2, & q_3, & q_4 \end{pmatrix}. \tag{1.6}$$

Их скалярное произведение будет равно:

$$P \cdot Q = p_1 q_1 + p_2 q_2 + p_3 q_3 + p_4 q_4; \tag{1.7}$$

$$P \cdot Q = (p_1 + q_2)(p_2 + q_1) + (p_3 + q_4)(p_4 + q_3) - p_1 p_2 - p_3 p_4 - q_1 q_2 - q_3 q_4. \tag{1.8}$$

После внимательного изучения формулы 1.7 можно сказать, что ее сла-

гаемые могут быть вычислены заранее для каждой строки первой матрицы и для каждого столбца второй. В итоге некоторые операции умножения заменяются на операции сложения, которые являются менее времязатратными.

Именно этот прием и используется в алгоритме Винограда.

## 1.4 Алгоритм Винограда с оптимизациями

Для того, чтобы дополнительно уменьшить время работы алгоритма Винограда необходимо применить следующие методы оптимизации:

- заменить операцию $x = x + k$ на $x+ = k$;
- заменить умножение на 2 на побитовый сдвиг;
- предвычислять слагаемые для алгоритма, например те, которые считаются в цикле каждый раз заново, но не зависят от исполняемых итераций.

## Вывод

В данном разделе был теоретически разобран классический алгоритм умножения матриц и алгоритм Винограда.

# 2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов умножения матриц, рассмотренных в аналитической части, выбранные типы данных и вывод оценки трудоемкости алгоритмов.

## 2.1 Алгоритмы умножения матриц

Схемы алгоритмов умножения матриц представлены ниже на рисунках 2.1-2.5.

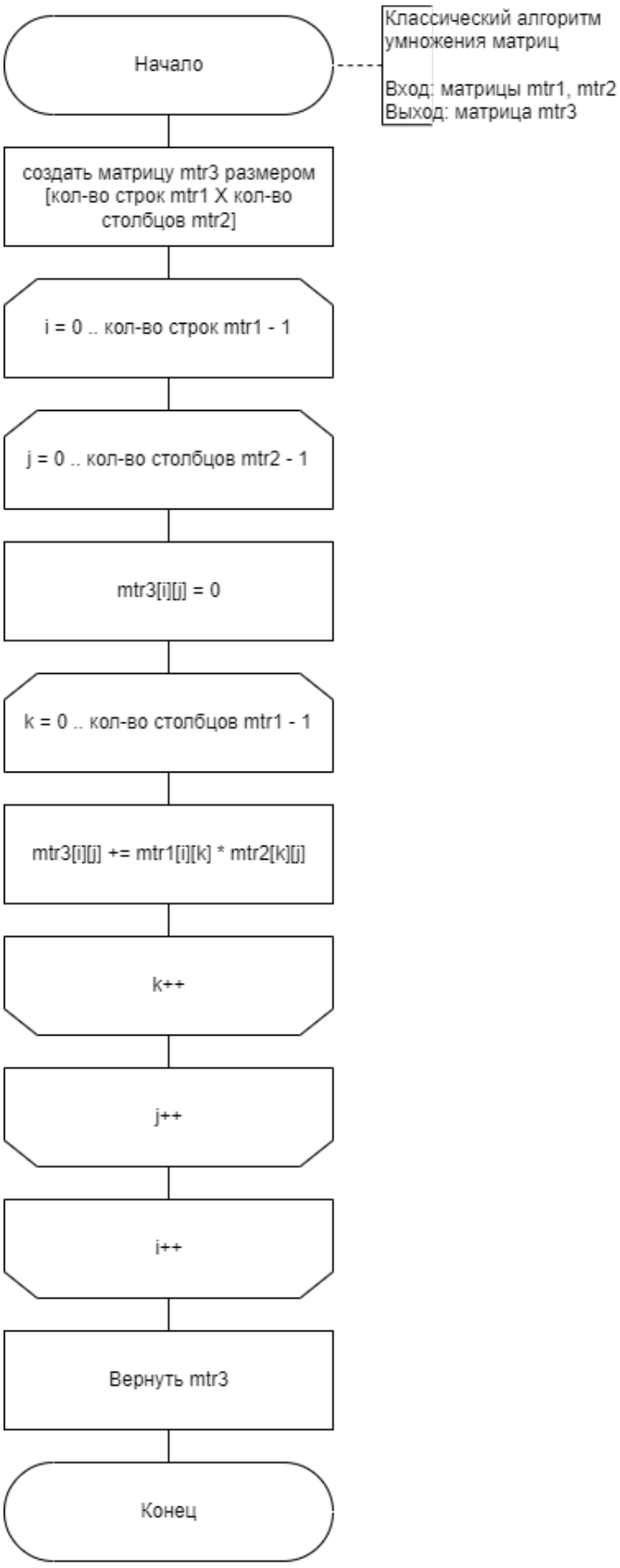


Рисунок 2.1 – Классический алгоритм умножения матриц

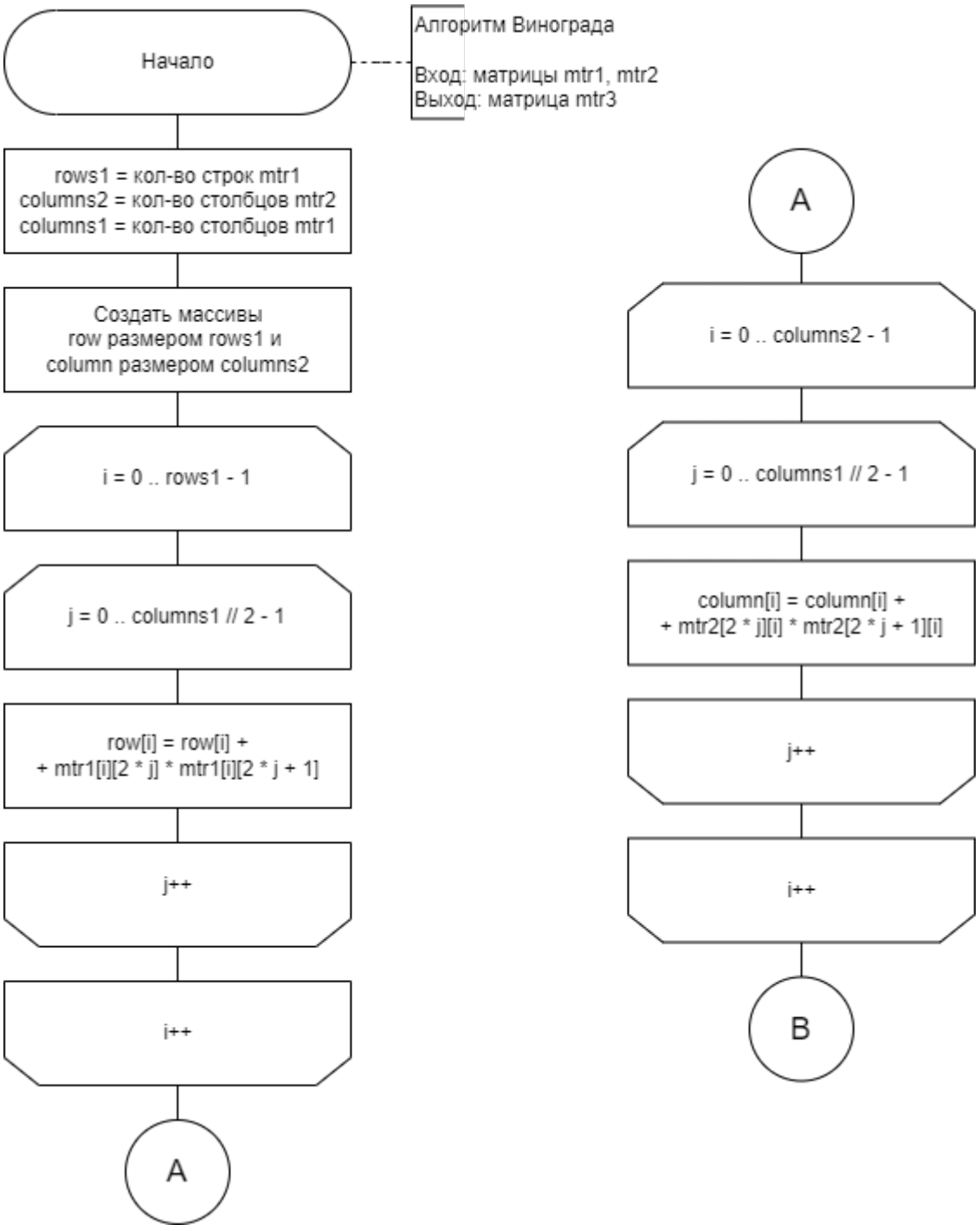


Рисунок 2.2 – Алгоритм Винограда (часть 1)

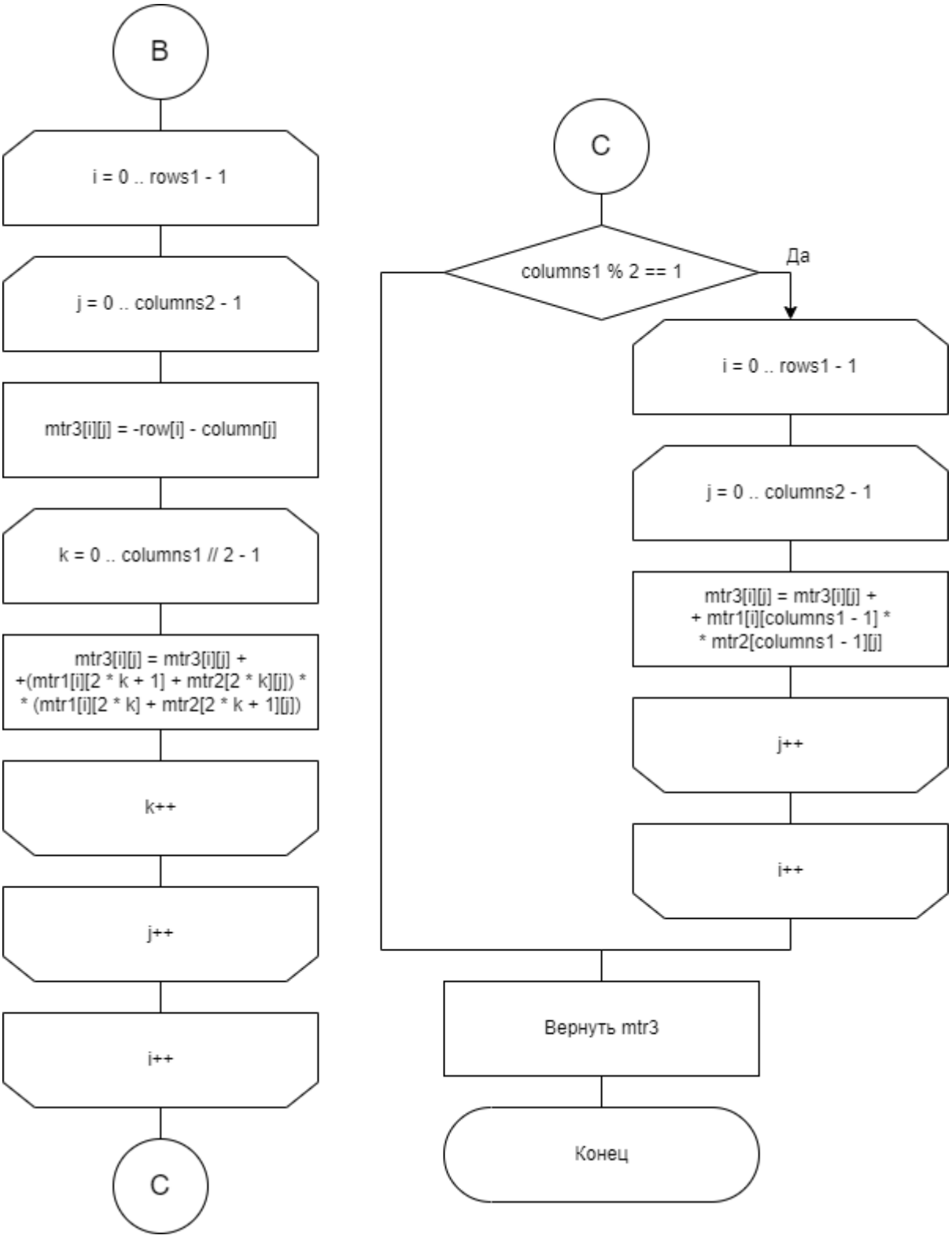


Рисунок 2.3 – Алгоритм Винограда (часть 2)

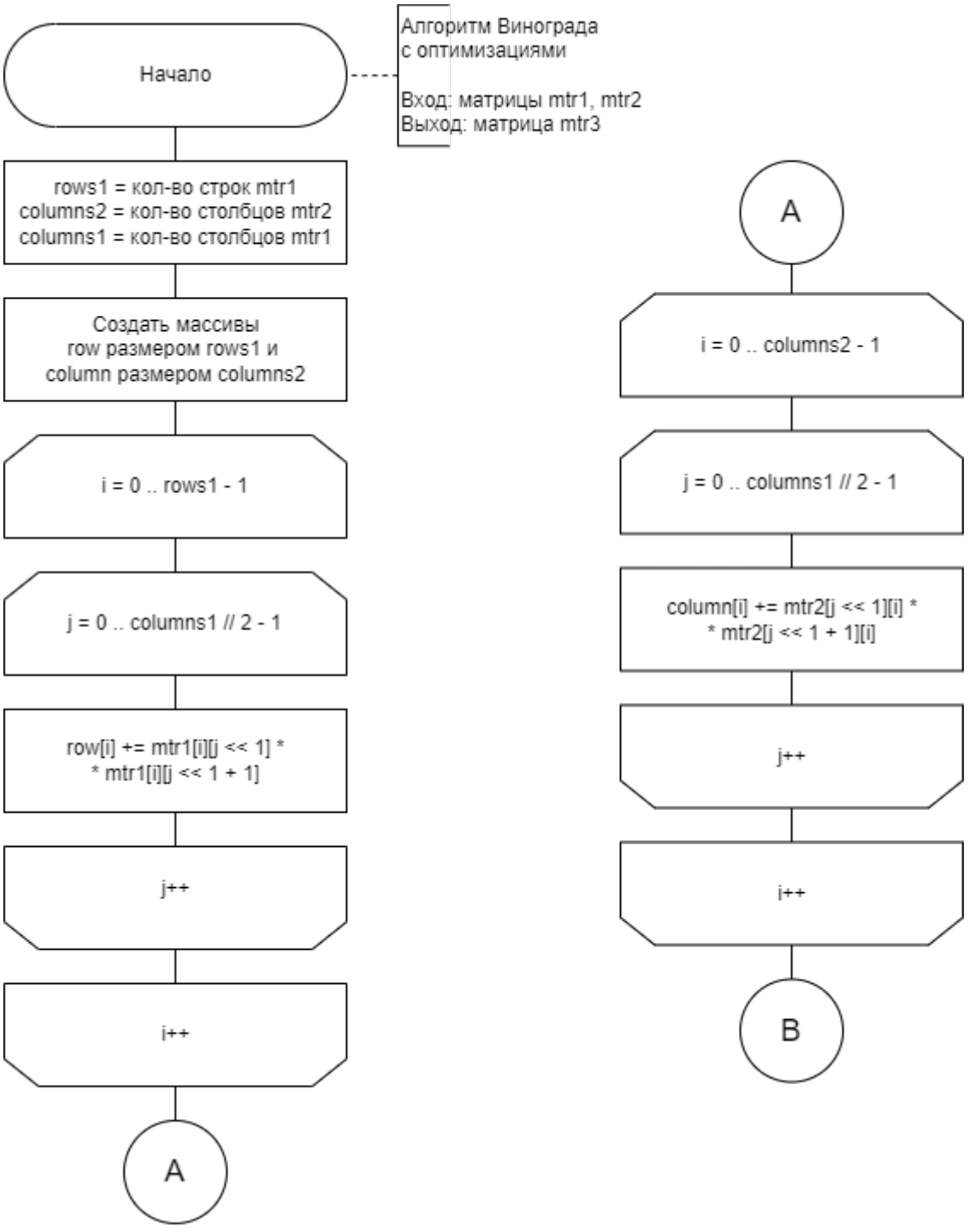


Рисунок 2.4 – Алгоритм Винограда с оптимизациями (часть 1)

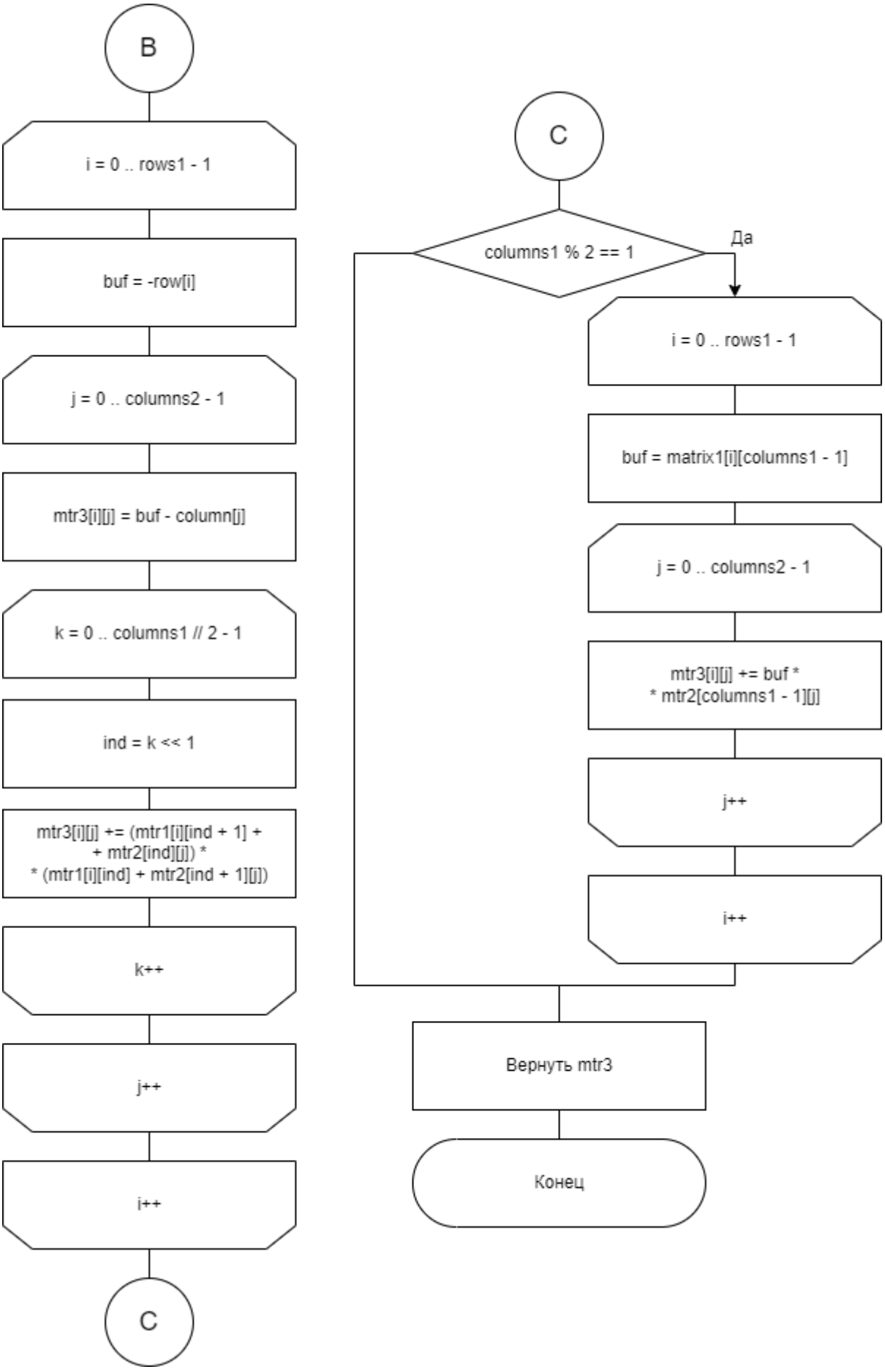


Рисунок 2.5 – Алгоритм Винограда с оптимизациями (часть 2)

## 2.2 Описание типов данных

Выбранные типы данных:

- матрица - массив массивов типа *float*;
- количество строк матрицы - тип *int*;
- количество столбцов матрицы - тип *int*;
- дополнительные массивы типа *float*.

Для матрицы выбран тип *float*, так как данный тип охватывает и целые числа, и вещественные, а это необходима для полноценного тестирования работы алгоритмов.

## 2.3 Оценка трудоемкости алгоритмов

Для вывода оценки трудоемкости алгоритмов существуют определенные правила.

1. Трудоемкость трех операций (/, %, *) равна 2, а для всех остальных (=, +=, -=, ++, −−, ==, !=, <, <=, >, >=, +, -, <<, >>, [ ]) – «штраф» равен 1.

2. Трудоемкость цикла:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \quad (2.1)$$

3. Трудоемкость условного оператора:

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

Вывод оценки трудоемкости алгоритмов заданных сортировок:

- **Классический алгоритм умножения матриц**

  В таблице 2.1 представлена построчная оценка трудоемкости классического алгоритма умножения матриц.

Таблица 2.1 – Построчная оценка трудоемкости классического алгоритма умножения матриц

| Строка кода | Вес |
|---|---|
| for i in range(len(matrix1)): | 3 |
| for j in range(len(matrix2[0])): | 4 |
| matrix_res[i][j] = 0 | 3 |
| for k in range(len(matrix1[0])): | 4 |
| matrix_res[i][j] = matrix_res[i][j] + matrix1[i][k] * matrix2[k][j] | 12 |
| return matrix_res | 0 |

  – Трудоемкость цикла по $i \in [1..N]$:

$$f = 2 + N \cdot (2 + f_{body}) \tag{2.3}$$

  – Трудоемкость цикла по $j \in [1..M]$:

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

  – Трудоемкость цикла по $k \in [1..K]$:

$$f = 2 + K \cdot (2 + 12) \tag{2.5}$$

  Итого трудоемкость:

$$f = 2 + N \cdot (4 + M \cdot (4 + 14K)) \approx 14NMK \tag{2.6}$$

- **Алгоритм Винограда**

  В таблице 2.2 представлена построчная оценка трудоемкости алгоритма Винограда.

Таблица 2.2 – Построчная оценка трудоемкости алгоритма Винограда

| Строка кода | Вес |
|---|---|
| rows1 = len(matrix1) | 1 |
| columns2 = len(matrix2[0]) | 2 |
| columns1 = len(matrix1[0]) | 2 |
| row = [0] * rows1 | $rows1 \cdot 2$ |
| column = [0] * columns2 | $columns2 \cdot 2$ |
| for i in range(rows1): | 3 |
| for j in range(columns1 // 2): | 5 |
| row[i] = row[i] + matrix1[i][2 * j] * matrix1[i][2 * j + 1] | 15 |
| for i in range(columns2): | 3 |
| for j in range(columns1 // 2): | 5 |
| column[i] = column[i] + matrix2[2 * j][i] * matrix2[2 * j + 1][i] | 15 |
| for i in range(rows1): | 3 |
| for j in range(columns2): | 3 |
| matrix_res[i][j] = -row[i] - column[j] | 7 |
| for k in range(columns1 // 2): | 5 |
| matrix_res[i][j] = matrix_res[i][j] + (matrix1[i][2 * k + 1] + matrix2[2 * k][j]) * (matrix1[i][2 * k] + matrix2[2 * k + 1][j]) | 28 |
| if columns1 % 2 == 1: | 3 |
| for i in range(rows1): | 3 |
| for j in range(columns2): | 3 |
| matrix_res[i][j] = matrix_res[i][j] + matrix1[i][columns1 - 1] * matrix2[columns1 - 1][j] | 14 |
| return matrix_res | 0 |

Подсчет трудоемкости.

– Создание вспомогательных массивов:

$$f_{arr} = 2 \cdot (rows1 + columns1) \tag{2.7}$$

– Работа с массивом $row$ (первый цикл):

$$f_{row} = 2 + rows1 \cdot (4 + 17 \cdot \frac{columns1}{2}) \tag{2.8}$$

– Работа с массивом $column$ (второй цикл):

$$f_{column} = 2 + columns2 \cdot (4 + 17 \cdot \frac{columns1}{2}) \tag{2.9}$$

– Цикл заполнения результирующей матрицы:

$$f_{res} = 2 + rows1 \cdot (4 + columns2 \cdot (13 + 15 \cdot columns1)) \tag{2.10}$$

– Дополнительный цикл, если матрица нечетного размера:

$$f_{extra} = 2 + rows1 \cdot (4 + 17 \cdot columns2) \tag{2.11}$$

Трудоемкость для лучшего случая (количество столбцов первой матрицы – четное):

$$f = 2f_{arr} + f_{row} + f_{column} + f_{res} + 2 \approx 15 \cdot rows1 \cdot columns1 \cdot columns2 \tag{2.12}$$

Трудоемкость для худшего случая (количество столбцов первой матрицы – нечетное):

$$\begin{aligned} f = 2f_{arr} + f_{row} + f_{column} + f_{res} + f_{extra} \approx \\ \approx 15 \cdot rows1 \cdot columns1 \cdot columns2 \end{aligned} \tag{2.13}$$

- **Алгоритм Винограда с оптимизациями**

  В таблице 2.3 представлена построчная оценка трудоемкости алгоритма Винограда с оптимизациями.

Таблица 2.3 – Построчная оценка трудоемкости алгоритма Винограда с оптимизациями

| Строка кода | Вес |
|---|---|
| rows1 = len(matrix1) | 1 |
| columns2 = len(matrix2[0]) | 2 |
| columns1 = len(matrix1[0]) | 2 |
| row = [0] * rows1 | $rows1 \cdot 2$ |
| column = [0] * columns2 | $columns2 \cdot 2$ |
| for i in range(rows1): | 3 |
| for j in range(columns1 // 2): | 5 |
| row[i] += matrix1[i][j << 1] * matrix1[i][(j << 1) + 1] | 11 |
| for i in range(columns2): | 3 |
| for j in range(columns1 // 2): | 5 |
| column[i] += matrix2[j << 1][i] * matrix2[(j << 1) + 1][i] | 11 |
| for i in range(rows1): | 3 |
| buf = -row[i] | 3 |
| for j in range(columns2): | 3 |
| matrix_res[i][j] = buf - column[j] | 5 |
| for k in range(columns1 // 2): | 5 |
| ind = k << 1 | 2 |
| matrix_res[i][j] += (matrix1[i][ind + 1] + matrix2[ind][j]) * (matrix1[i][ind] + matrix2[ind + 1][j]) | 17 |
| if columns1 % 2 == 1: | 3 |
| for i in range(rows1): | 3 |
| buf = matrix1[i][columns1 - 1] | 4 |
| for j in range(columns2): | 3 |
| matrix_res[i][j] += buf * matrix2[columns1 - 1][j] | 8 |
| return matrix_res | 0 |

Подсчет трудоемкости.

– Создание вспомогательных массивов:

$$f_{arr} = 2 \cdot (rows1 + columns1) \tag{2.14}$$

– Работа с массивом $row$ (первый цикл):

$$f_{row} = 2 + rows1 \cdot (4 + 13 \cdot \frac{columns1}{2}) \tag{2.15}$$

– Работа с массивом $column$ (второй цикл):

$$f_{column} = 2 + columns2 \cdot (4 + 13 \cdot \frac{columns1}{2}) \tag{2.16}$$

– Цикл заполнения результирующей матрицы:

$$f_{res} = 2 + rows1 \cdot (7 + columns2 \cdot (9 + \frac{21}{2} \cdot columns1)) \tag{2.17}$$

– Дополнительный цикл, если матрица нечетного размера:

$$f_{extra} = 2 + rows1 \cdot (8 + 10 \cdot columns2) \tag{2.18}$$

Трудоемкость для лучшего случая (количество столбцов первой матрицы – четное):

$$\begin{aligned} f = 2f_{arr} + f_{row} + f_{column} + f_{res} + 2 \approx \\ \approx 10.5 \cdot rows1 \cdot columns1 \cdot columns2 \end{aligned} \tag{2.19}$$

Трудоемкость для худшего случая (количество столбцов первой матрицы – нечетное):

$$\begin{aligned} f = 2f_{arr} + f_{row} + f_{column} + f_{res} + f_{extra} \approx \\ \approx 10.5 \cdot rows1 \cdot columns1 \cdot columns2 \end{aligned} \tag{2.20}$$

## 2.4 Исходные файлы программы

Программа состоит из следующих модулей:

- *main.py* – файл с главной функцией, вызывающей меню программы;
- *cdio.py* – файл с функциями, осуществляющими ввод и вывод данных;
- *multi_matrix.py* – файл, содержащий код алгоритмов умножения матриц;
- *proc_time.py* – файл с функциями подсчета процессорного времени алгоритмов.

## Вывод

В этом разделе были представлены схемы классического алгоритма умножения матриц и алгоритма Винограда с оптимизациями и без, вывод оценки их трудоемкости и выбранные типы данных.

Проведя анализ трудоемкости алгоритмов, можно сказать, что между классическим алгоритмом умножения матриц и алгоритмом Винограда не столь большая разница, однако алгоритм Винограда с введенными оптимизациями теоретически работает уже значительно быстрее.

# 3 Технологическая часть

В данном разделе будут рассмотрены средства реализации лабораторной работы, представлены листинги алгоритмов умножения матриц, а также тестовые данные, которые использовались для проверки корректности работы алгоритмов.

## 3.1 Выбор языка программирования и среды разработки

Для реализации алгоритмов был выбран язык программирования *Python*, а в качестве среды разработки – *PyCharm*. Для замеров процессорного времени использовалась функция *process_time()* [3] из библиотеки *time*, а для построения графиков – библиотека *matplotlib* [4].

## 3.2 Реализация алгоритмов

Ниже представлены реализации классического алгоритма умножения матриц и алгоритма Винограда с оптимизациями и без (листинги 3.1–3.3).

Листинг 3.1 – Классический алгоритм умножения матриц

```
1 def matrix_multiply(matrix1, matrix2):
2     matrix_res = [[0] * len(matrix1) for _ in
        range(len(matrix2[0]))]
3     for i in range(len(matrix1)):
4         for j in range(len(matrix2[0])):
5             matrix_res[i][j] = 0
6             for k in range(len(matrix1[0])):
7                 matrix_res[i][j] = matrix_res[i][j] + matrix1[i][k]
                    * matrix2[k][j]
8     return matrix_res
```

Листинг 3.2 – Алгоритм Винограда

```
def vinograd(matrix1, matrix2):
    rows1 = len(matrix1)
    columns2 = len(matrix2[0])
    columns1 = len(matrix1[0])
    matrix_res = [[0] * rows1 for _ in range(columns2)]
    row = [0] * rows1
    column = [0] * columns2

    for i in range(rows1):
        for j in range(columns1 // 2):
            row[i] = row[i] + matrix1[i][2 * j] * matrix1[i][2 * j
               + 1]

    for i in range(columns2):
        for j in range(columns1 // 2):
            column[i] = column[i] + matrix2[2 * j][i] * matrix2[2 *
               j + 1][i]

    for i in range(rows1):
        for j in range(columns2):
            matrix_res[i][j] = -row[i] - column[j]
            for k in range(columns1 // 2):
                matrix_res[i][j] = matrix_res[i][j] + (matrix1[i][2
                   * k + 1] + matrix2[2 * k][j]) * (matrix1[i][2 *
                   k] + matrix2[2 * k + 1][j])

    if columns1 % 2 == 1:
        for i in range(rows1):
            for j in range(columns2):
                matrix_res[i][j] = matrix_res[i][j] +
                   matrix1[i][columns1 - 1] * matrix2[columns1 -
                   1][j]

    return matrix_res
```

Листинг 3.3 – Алгоритм Винограда с оптимизациями

```
def optimized_vinograd(matrix1, matrix2):
    rows1 = len(matrix1)
    columns2 = len(matrix2[0])
```

```
    columns1 = len(matrix1[0])
    matrix_res = [[0] * rows1 for _ in range(columns2)]
    row = [0] * rows1
    column = [0] * columns2

    for i in range(rows1):
        for j in range(columns1 // 2):
            row[i] += matrix1[i][j << 1] * matrix1[i][(j << 1) + 1]

    for i in range(columns2):
        for j in range(columns1 // 2):
            column[i] += matrix2[j << 1][i] * matrix2[(j << 1) +
               1][i]

    for i in range(rows1):
        buf = -row[i]
        for j in range(columns2):
            matrix_res[i][j] = buf - column[j]
            for k in range(columns1 // 2):
                ind = k << 1
                matrix_res[i][j] += (matrix1[i][ind + 1] +
                   matrix2[ind][j]) * (matrix1[i][ind] +
                   matrix2[ind + 1][j])

    if columns1 % 2 == 1:
        for i in range(rows1):
            buf = matrix1[i][columns1 - 1]
            for j in range(columns2):
                matrix_res[i][j] += buf * matrix2[columns1 - 1][j]

    return matrix_res
```

## 3.3 Тестирование

Классы эквивалентности:

- две пустые матрицы;
- первая матрица пустая;

- вторая матрица пустая;
- матрицы, у которых число столбцов первой не равно числу строк второй;
- две нулевые матрицы;
- первая матрица нулевая;
- вторая матрица нулевая;
- две единичные матрицы;
- первая матрица единичная;
- вторая матрица единичная;
- две случайно заполненные матрицы.

В таблице 3.1 представлены модульные тесты. Все тесты пройдены успешно.

Таблица 3.1 – Модульные тесты

| | Входные данные | | Ожидаемый результат | | |
|---|---|---|---|---|---|
| № | mtr1 | mtr2 | Классич. | Винограда | Винограда с опт. |
| 1 | $()$ | $()$ | Error | Error | Error |
| 2 | $()$ | $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ | Error | Error | Error |
| 3 | $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ | $()$ | Error | Error | Error |
| 4 | $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ | $\begin{pmatrix} 5 & 0 \\ 3 & 7 \end{pmatrix}$ | Error | Error | Error |
| 5 | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ |
| 6 | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 3 & 9 \\ 2 & 1 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ |
| 7 | $\begin{pmatrix} 4 & 5 \\ 3 & 2 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ | $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ |
| 8 | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ |
| 9 | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 2 & 8 \\ 9 & -4 \end{pmatrix}$ | $\begin{pmatrix} 2 & 8 \\ 9 & -4 \end{pmatrix}$ | $\begin{pmatrix} 2 & 8 \\ 9 & -4 \end{pmatrix}$ | $\begin{pmatrix} 2 & 8 \\ 9 & -4 \end{pmatrix}$ |
| 10 | $\begin{pmatrix} 7 & 5 \\ 3 & 10 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 7 & 5 \\ 3 & 10 \end{pmatrix}$ | $\begin{pmatrix} 7 & 5 \\ 3 & 10 \end{pmatrix}$ | $\begin{pmatrix} 7 & 5 \\ 3 & 10 \end{pmatrix}$ |
| 11 | $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ | $\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$ | $\begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$ | $\begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$ | $\begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$ |

# Вывод

В этом разделе были рассмотрены средства реализации лабораторной работы, представлены листинги классического алгоритма умножения матриц и алгоритма Винограда с отпимизациями и без, а также модульные тесты.

# 4 Исследовательская часть

В данном разделе будут представлены примеры работы программы, проведены замеры процессорного времени и предоставлена информация о технических характеристиках устройства.

## 4.1 Примеры работы программы

На рисунках 4.1-4.2 представлен результат работы программы. В каждом примере пользователем введены две матрицы и получены результаты их умножения.

```
Меню
1. Умножение матриц:
     - Классический
     - Винограда
     - Винограда с оптимизацией
2. Замеры процессорного времени
0. Выход
>  1
Ввод первой матрицы
Введите количество строк: 3
Введите количество столбцов: 4
Введите элементы матрицы (построчно):
1 6 2 3
9 0 0 2
3 4 -1 2
Ввод второй матрицы
Введите количество строк: 4 2
Введите натуральное число!
Введите количество строк: 4
Введите количество столбцов: 2
Введите элементы матрицы (построчно):
7 2
9 10
-3 0
-4 2

 ================================================================================
Классический алгоритм перемножения матриц
Результат:
  43    68
  55    22
  52    50

 ================================================================================
Алгоритм Винограда
Результат:
  43    68
  55    22
  52    50

 ================================================================================
Алгорит Винограда с оптимизацией
Результат:
  43    68
  55    22
  52    50
```

Рисунок 4.1 – Пример работы программы №1

```
Меню
1. Умножение матриц:
     - Классический
     - Винограда
     - Винограда с оптимизацией
2. Замеры процессорного времени
0. Выход
>  1
Ввод первой матрицы
Введите количество строк: 3
Введите количество столбцов: 3
Введите элементы матрицы (построчно):
3
Неправильное количество чисел в строке! Попробуйте снова!
1 0 0
0 1 0
0 0 1
Ввод второй матрицы
Введите количество строк: 3
Введите количество столбцов: 5
Введите элементы матрицы (построчно):
1 4 3 -10 9
4 3 8 10 9
8 11 0 -19 1

 ================================================================================
Классический алгоритм перемножения матриц
Результат:
   1     4     3   -10     9
   4     3     8    10     9
   8    11     0   -19     1

 ================================================================================
Алгоритм Винограда
Результат:
   1     4     3   -10     9
   4     3     8    10     9
   8    11     0   -19     1

 ================================================================================
Алгорит Винограда с оптимизацией
Результат:
   1     4     3   -10     9
   4     3     8    10     9
   8    11     0   -19     1
```

Рисунок 4.2 – Пример работы программы №2

## 4.2 Время выполнения реализаций алгоритмов

Для замера процессорного времени использовалась функция *process_time()* библиотеки *time*. Возвращаемый результат – время в миллисекундах, число типа *float*.

Чтобы получить достаточно точное значение, производилось усреднение времени. В замерах использовались матрицы размером от $[50 \times 50]$ до $[450 \times 450]$, а сам процесс перемножения для каждых двух матриц запускался по 200 раз.

В таблице 4.1 представлено процессорное время работы алгоритмов умножения матриц.

Таблица 4.1 – Результаты замеров времен

| Размер матрицы | Классический | Винограда | Винограда с опт. |
|---|---|---|---|
| 50 | 0.026328 | 0.023984 | 0.021094 |
| 100 | 0.165547 | 0.174063 | 0.164062 |
| 150 | 0.565937 | 0.456016 | 0.458047 |
| 200 | 1.135781 | 1.183828 | 1.235234 |
| 250 | 2.311016 | 2.591016 | 2.154219 |
| 300 | 4.498516 | 4.429297 | 3.884453 |
| 350 | 6.387656 | 7.152188 | 6.172891 |
| 400 | 9.993828 | 10.394141 | 9.592266 |
| 450 | 13.757500 | 15.271719 | 13.475078 |

На рисунке 4.3 также приведены результаты замеров процессорного времени.

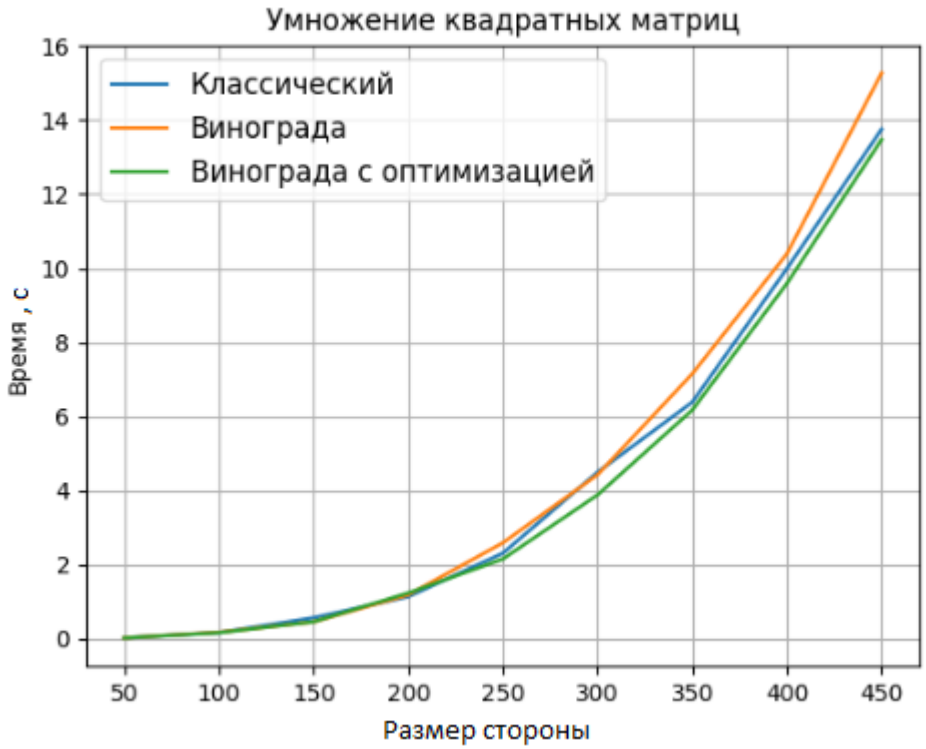


Рисунок 4.3 – Сравнение процессорного времени работы алгоритмов умножения матриц

## 4.3 Технические характеристики устройства

Ниже представлены характеристики компьютера, на котором проводилось тестирование программы:

- операционная система Windows 10 Домашняя 21H2;
- оперативная память 16 Гб;
- процессор Intel(R) Core(TM) i7-10870H CPU @ 2.20 ГГц.

Во время тестирования ноутбук был подключен к сети электропитания. Из программного обеспечения были запущены только среда разработки *PyCharm* и браузер *Chrome*.

Процессор был загружен на 19%, оперативная память – на 50%.

## 4.4 Вывод

В результате замеров процессорного времени выделены следущие аспекты: работа алгоритма Винограда без оптимизаций занимает больше времени, чем два остальных алгоритма; классический алгоритм умножения матриц и алгоритм Винограда с оптимизациями производят вычисления приблизительно за одно время.

# Заключение

Цель, которая была поставлена в начале лабораторной работы, была достигнута: изучены и исследованы алгоритмы умножения матриц.

Решены все поставленные задачи:

- разработаны и реализованы алгоритмы умножения матриц (классический, Винограда с оптимизациями и без);
- выведена оценка трудоемкости алгоритмов;
- выполнены замеры процессорного времени работы реализаций алгоритмов;
- проведен сравнительный анализ заданных алгоритмов умножения матриц по затрачнному времени работы реализаций.

Стоит отметить, что алгоритм Винограда с использованными оптимизациями и классический метод являются наиболее предпочтительными в решении задачи умножении матриц.

Алгоритм Винограда без оптимизаций в среднем тратит на 12.5% больше времени, чем два остальных алгоритма.

# Список использованных источников

[1] Бахвалов Н.С. Численные методы / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельков - М.: Наука, 1987.

[2] Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р.М. Ривест: - МЦНТО, 1999.

[3] time — Time access and conversions [Электронный ресурс]. – URL: `https://docs.python.org/3/library/time.html#time.process_time` (дата обращения: 14.10.2022)

[4] Matplotlib documentation [Электронный ресурс]. – URL: `https://matplotlib.org/stable/index.html` (дата обращения: 14.10.2022)

[5] Windows 10 Home [Электронный ресурс]. – URL: `https://www.microsoft.com/en-us/d/windows-10-home/d76qx4bznwk4?activetab=pivot:overviewtab` (дата обращения: 14.10.2022)

[6] Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz [Электронный ресурс]. – URL: `https://ark.intel.com/content/www/ru/ru/ark/products/208018/intel-core-i710870h-processor-16m-cache-up-to-5-00-ghz.html` (дата обращения: 14.10.2022)