**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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

# ОТЧЕТ

по лабораторной работе № 2

по курсу «Анализ алгоритмов»

на тему: «Умножение матриц (сложность)»

Студент ИУ7-55Б _____ И. Д. Половинкин
(Подпись, дата)

Преподаватель _____ Л. Л. Волкова
(Подпись, дата)

*2024 г.*

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Матричное умножение лежит в основе нейронных сетей. Большинство операций при обучении нейронной сети требуют перемножение матриц. Для этого требуется высокая скорость вычислений.

Стандартный алгоритм умножения матриц на больших данных, исчисляемых миллиардами, выполняет вычисления не самым быстрым способом. Существуют различные оптимизации. Одной из них является алгоритм Винограда, который позволяет сократить время вычислений [2]. На практике алгоритм Копперсмита—Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для тех матриц, размер которых превышает память современных компьютеров.

Целью лабораторной работы является изучение и реализация алгоритмов умножения матриц. Для её достижения необходимо выполнить следующие задачи:

- выполнить анализ алгоритмов умножения матриц стандартным способом, по Винограду и Штрассена;
- формально описать данные алгоритмы;
- выполнить оптимизацию алгоритма Винограда;
- реализовать алгоритмы умножения матриц;
- выполнить тестирование реализации алгоритмов методом черного ящика;
- провести сравнительный анализ этих алгоритмов по процессорному выполнению времени на основе экспериментальных данных.

# 1 Аналитический раздел

В данном разделе приводится анализ алгоритмов умножения матриц, а также метод оптимизации вычислений.

## 1.1 Определение матрицы

Матрицей размера $m \times n$ называется прямоугольная таблица элементов некоторого множества (например, чисел или функций), имеющая $m$ строк и $n$ столбцов [1]. Элементы $a_{ij}$ , из которых составлена матрица, называются элементами матрицы. Условимся, что первый индекс $i$ элемента $a_{ij}$ а соответствует номеру строки, второй индекс $j$ – номеру столбца, в котором расположен элемент $a_{ij}$. Матрица $A$ может быть записана по формуле (1.1):

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \tag{1.1}$$

## 1.2 Стандартный алгоритм умножения матриц

Произведением матрицы $A = (a_{ij})$, имеющей $m$ строк и $n$ столбцов, на матрицу $B = (b_{ij})$, имеющую $n$ строк и $p$ столбцов, называется матрица $c_{ij}$, имеющая $m$ строк и $p$ столбцов, у которой элемент $C = (c_{ij})$ определяется по формуле (1.2):

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{ri} \quad (i = 1, 2, \dots n; j = 1, 2, \dots p) \ . \tag{1.2}$$

## 1.3 Алгоритм умножения матриц по Винограду

Обозначим $i$ строку матрицы как $\overline{u}$, $j$ столбец матрицы как $\overline{v}$ [2]. Тогда элемент $c_{ij}$ определяется по формуле (1.3):

$$c_{ij} = \overline{u} \times \overline{v} = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4. \tag{1.3}$$

Эту формулу можно представить в следующем виде (1.4):

$$(u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4 \qquad (1.4)$$

На первый взгляд может показаться, что выражение (1.4) задает больше работы, чем первое: вместо четырех умножений насчитывается их шесть, а вместо трех сложений – десять. Выражение в правой части формулы можно вычислить заранее и затем повторно использовать. На практике это означает, что над предварительно обработанными элементами придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

## 1.4 Алгоритм Штрассена

Алгоритм Штрассена основан на принципе «разделяй и властвуй», который позволяет уменьшить время выполнения умножения матриц за счет рекурсивного разделения на матрицы меньшего размера [3]. Пусть $A$ и $B$ – две матрицы размера $n \times n$, где $n$ – степень числа 2. Каждую матрицу $A$ и $B$ можно разбить на четыре матрицы размером $n/2 \times n/2$ и через них выразить произведение матриц $A$ и $B$ (1.5):

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \qquad (1.5)$$

где элементы матрицы $C$ и их компоненты вычисляются как (1.6):

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21}, \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22}, \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21}, \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22}. \end{aligned} \qquad (1.6)$$

Такой метод требует 8 умножений для вычисления $C_{ij}$ в классическом произведении. Алгоритм Штрассена заключается в наборе из семи новых матриц $M_i$, $i = \overline{1;7}$, которые используются для выражения $C_{ij}$ с помощью 7 умножений (1.7):

$$
\begin{aligned}
M_1 &= (A_{11} + A_{12})(B_{11} + B_{12}),\\
M_2 &= (A_{21} + A_{22})B_{11},\\
M_3 &= A_{11}(B_{12} - B_{22}),\\
M_4 &= A_{22}(A_{21} - B_{11}), \qquad ,\\
M_5 &= (A_{11} + A_{12})B_{22},\\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}),\\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}).
\end{aligned}
\tag{1.7}
$$

Тогда выражения $C_{ij}$ вычисляются как (1.8):

$$
\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7,\\
C_{12} &= M_3 + M_5,\\
C_{21} &= M_2 + M_4,\\
C_{22} &= M_1 - M_2 + M_3 + M_6.
\end{aligned}
\tag{1.8}
$$

## 1.5 Оптимизации алгоритмов

При реализации алгоритмов умножения матрицы возможны следующие методы оптимизации:

1. Предвычисление некоторых слагаемых для алгоритма.
2. Замена операции $x = x + k$ на $x \mathrel{+}= k$.
3. Замена умножения на 2 на побитовый сдвиг.

## Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц стандартным образом, по Винограду, по Штрассену, а также приведены соответствующие математические расчеты. Предложены методы оптимизации расчетов.

# 2 Конструкторский раздел

В данном разделе приводятся схемы алгоритмов умножения матриц стандартным способом, Винограда и Штрассена, приводится расчет трудоемкости каждого из них, а также оптимизированной версии алгоритма Винограда.

## 2.1 Алгоритмы умножения матриц

### 2.1.1 Стандартный алгоритм

На рисунке 2.2 представлен алгоритм стандартного умножения матриц.

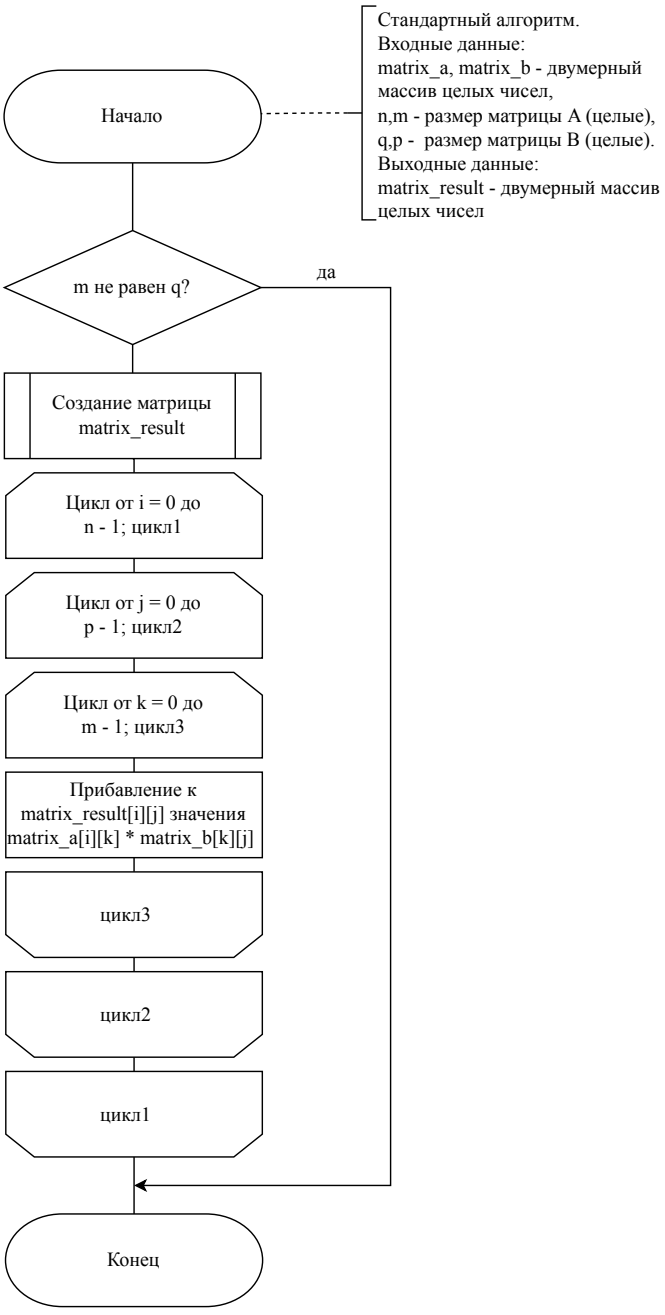


Рисунок 2.1 – Алгоритм стандартного умножения матриц

### 2.1.2 Стандартный алгоритм

На рисунке 2.2 представлен алгоритм стандартного умножения матриц.

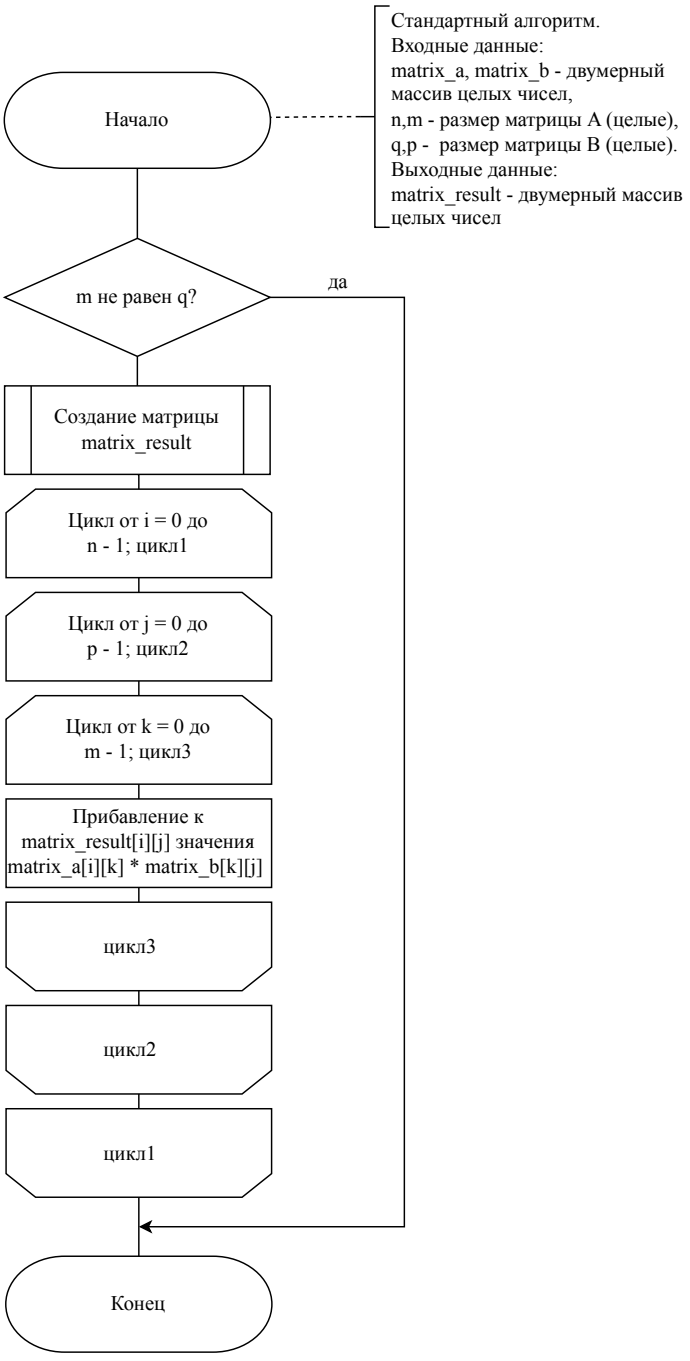


Рисунок 2.2 – Алгоритм стандартного умножения матриц

### 2.1.3 Алгоритм Винограда

На рисунке 2.3 и 2.4 представлен алгоритм стандартного умножения матриц.

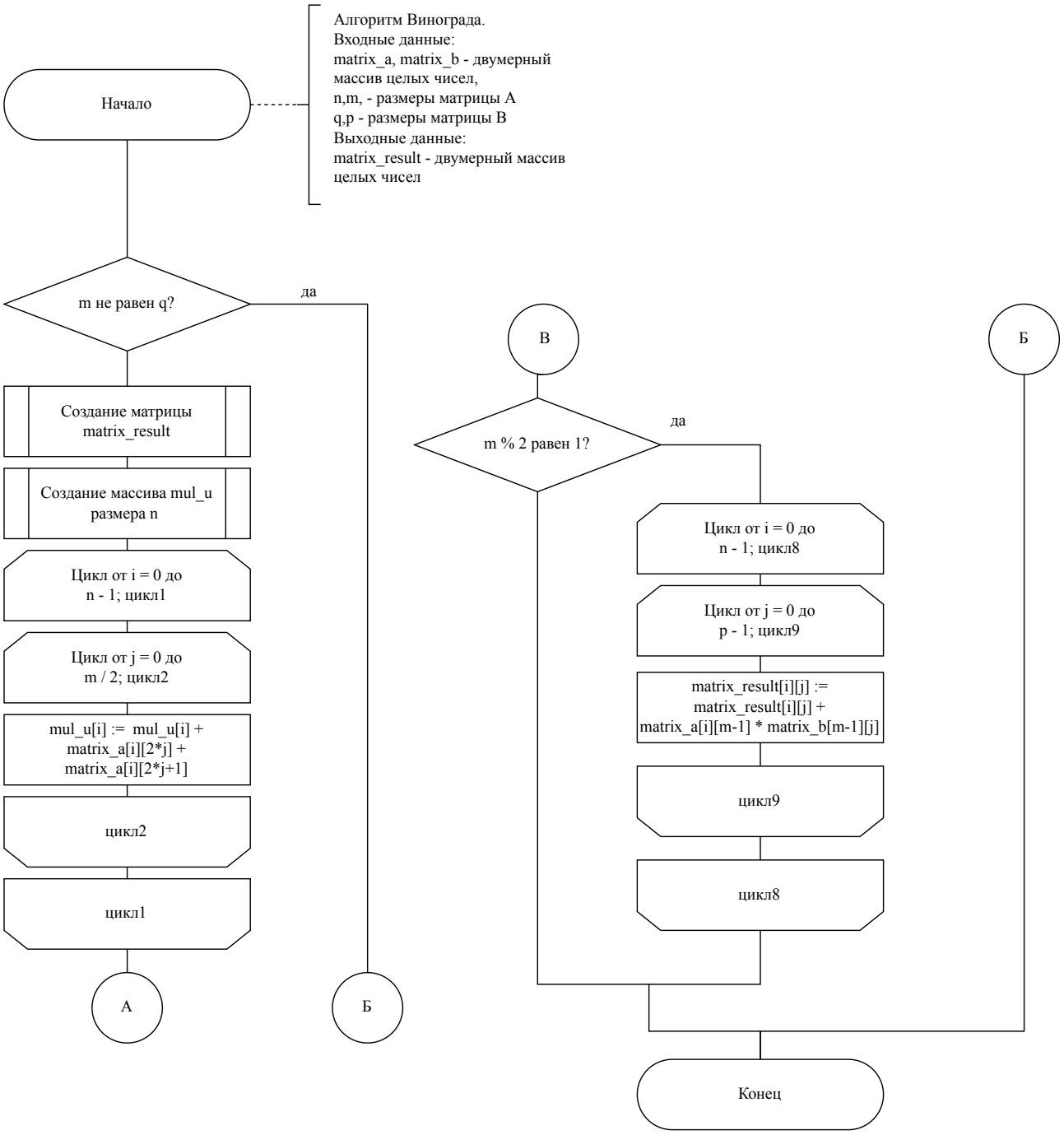


Рисунок 2.3 – Алгоритм умножения матриц по Винограду. Часть 1

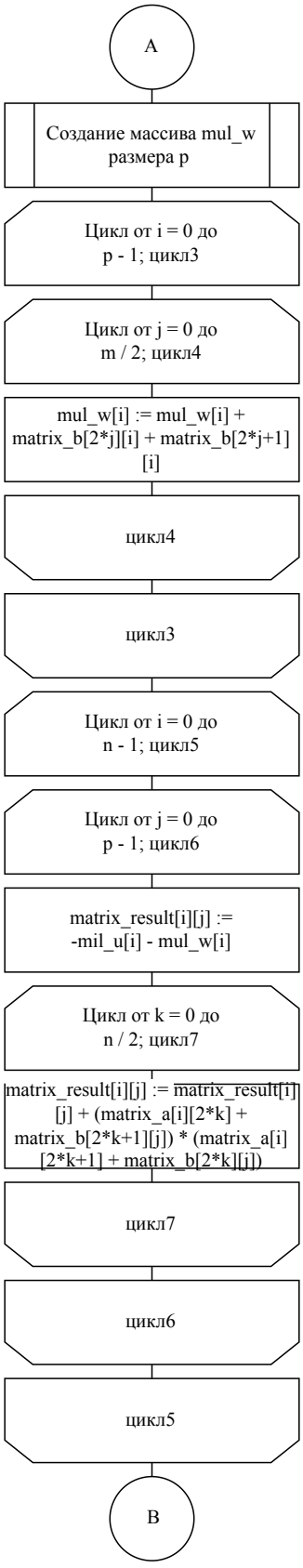


Рисунок 2.4 – Алгоритм умножения матриц по Винограду. Часть 2

### 2.1.4 Алгоритм Штрассена

На рисунке 2.5 и 2.6 представлен алгоритм умножения матриц Штрассена.

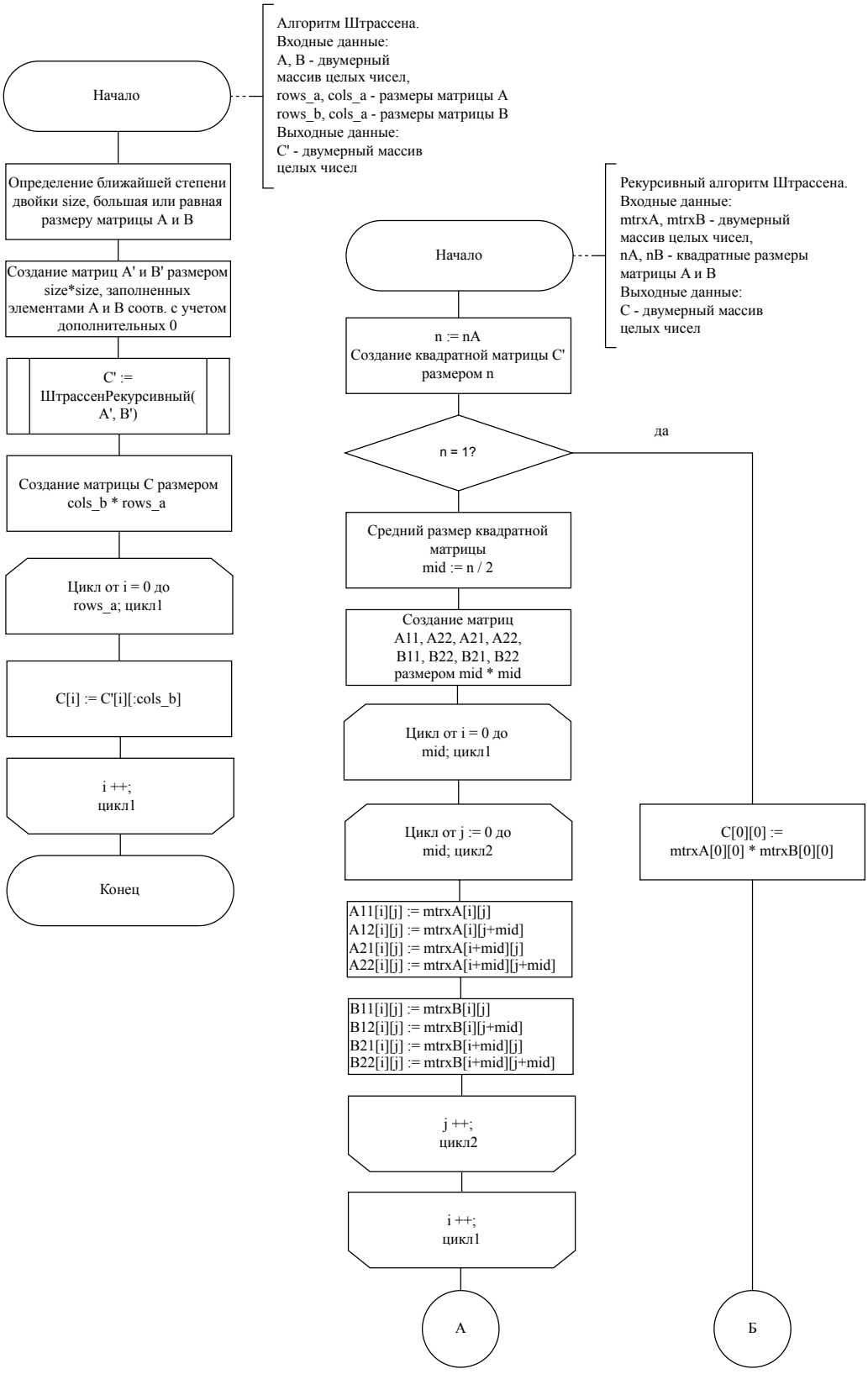


Рисунок 2.5 – Алгоритм умножения матриц Штрассена. Часть 1

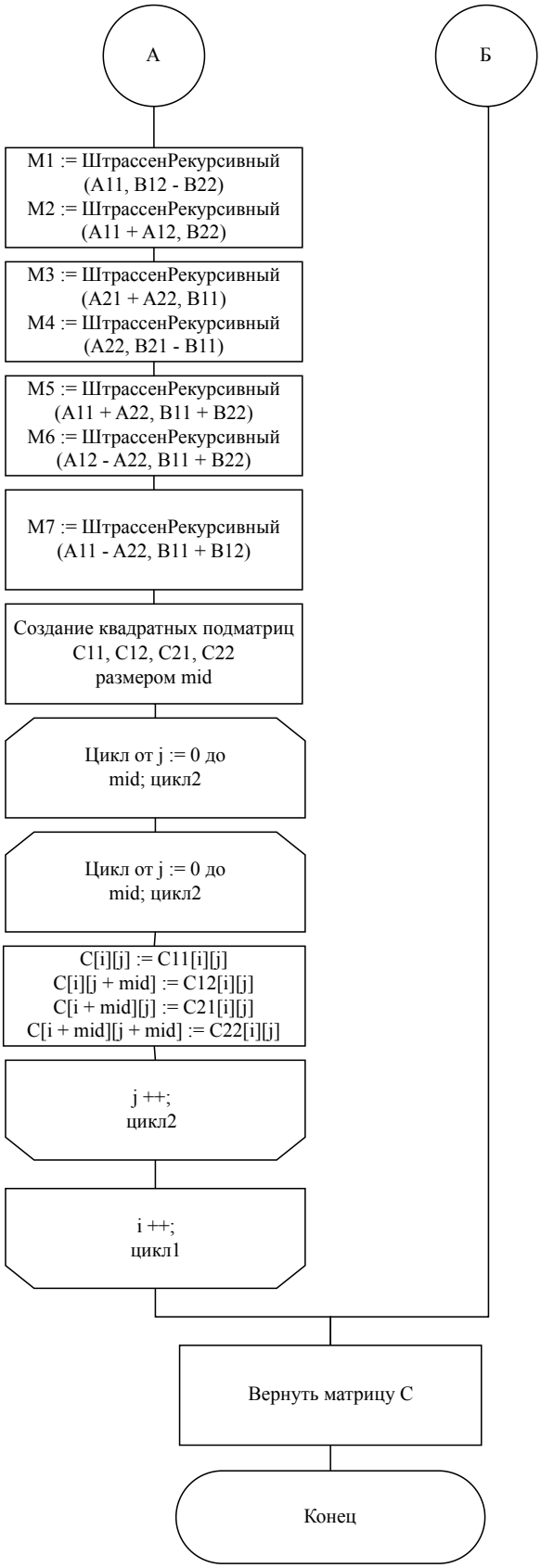


Рисунок 2.6 – Алгоритм умножения матриц Штрассена. Часть 2

## 2.2 Модель оценки трудоемкости алгоритмов

Введем модель оценки трудоемкости.

1. Трудоемкость базовых операций.

   Пусть трудоемкость следующих операций равной 2:

   $$*, /, \ //, \%, *=, /= .$$

   Примем трудоемкость следующих операций равной 1:

   $$=, +, -, + =, - =, ==, !=, <, >, \leq, \geq, |, \&\&, , ||, [\,], <<, >> .$$

2. Трудоемкость цикла.

   Пусть трудоемкость цикла определяется по формуле (2.1).

   $$f = f_{init} + f_{comp} + N_{iter} * (f_{in} + f_{inc} + f_{comp}), \tag{2.1}$$

   где:

   - $f_{init}$: трудоемкость инициализации переменной-счетчика;
   - $f_{comp}$: трудоемкость сравнения;
   - $N_{iter}$: номер выполняемой итерации;
   - $f_{in}$: трудоемкость команд из тела цикла;
   - $f_{inc}$: трудоемкость инкремента;
   - $f_{comp}$: трудоемкость сравнения.

3. Трудоемкость условного оператора.
   Пусть трудоемкость самого условного перехода равна 0 в лучшем случае, когда условие не выполняется, иначе – трудоемкости операций, относящихся к условному оператору $f_{if}$ (2.2):

   $$f_{if} = f_{comp\_if} + \begin{cases} 0, & \text{лучший} \\ f_b, & \text{худший} \end{cases}. \tag{2.2}$$

## 2.3 Вычисление трудоемкости алгоритмов

Пусть во всех дальнейших вычислениях размер матрицы $A$ имеет $M \times N$, для матрицы $B$ – $N \times Q$.

### 2.3.1 Трудоемкость алгоритма умножения матриц в стандартном случае

Трудоемкость $f_{stand}$ алгоритма стандартного умножения матриц вычисляется по формуле (2.3):

$$f_{stand} = \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + N(\underset{++}{2} + \underset{[\,]}{8} + \underset{*}{2} + \underset{=}{1} + \underset{+}{1}))) = \\ = 14MNQ + 4MQ + 4M + 2. \quad (2.3)$$

### 2.3.2 Трудоемкость алгоритма умножения матриц по Винограду

Трудоемкость этого алгоритма состоит из следующих компонентов, определяемых по формуле (2.4):

$$f_{vin} = f_{init} + f_{precomp} + f_{fill} + f_{even}, \quad (2.4)$$

где:

– $f_{init}$ - трудоемкость инициализации массивов для предварительного вычисления (2.5):

$$f_{init} = \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{[\,]}{1} + \underset{=}{1}) + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[\,]}{1} + \underset{=}{1}) = \\ = 2 + 4M + 2 + 4Q = 4 + 4M + 4Q; \quad (2.5)$$

– $f_{precomp}$ - трудоемкость предварительного заполнения строк матрицы A и

столбцов матрицы B (2.6):

$$f_{precomp} = f_{rows} + f_{columns} = \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + \frac{n}{2}(\underset{++}{2} + \underset{[\,]}{6} + \underset{=}{1} + \underset{+}{2} + \underset{*}{6})) +$$
$$+ \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + \frac{N}{2}(\underset{++}{2} + \underset{[\,]}{6} + \underset{=}{1} + \underset{+}{2} + \underset{*}{6})) =$$
$$= 2 + M(4 + \frac{N}{2} * 17) + 2 + Q(4 + \frac{N}{2} * 17) =$$
$$= 4 + 4M + 4Q + \frac{17NM}{2} + \frac{17NQ}{2}; \quad (2.6)$$

– $f_{even}$ - трудоемкость заполнения результирующей матрицы (2.7):

$$f_{fill} = \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[\,]}{4} + \underset{=}{1} + \underset{-}{2} + \underset{=}{1} + \underset{<}{1} +$$
$$+ \frac{N}{2}(\underset{++}{2} + \underset{[\,]}{12} + \underset{=}{1} + \underset{+}{5} + \underset{*}{10} + \underset{/}{2})) = 2 + M(4 + Q(11 + 16N)) =$$
$$= 2 + 4M + 11MQ + 16MNQ; \quad (2.7)$$

– $f_{fill}$ - трудоемкость для дополнения умножения в случае нечетной размерности матрицы. (2.8):

$$f_{fill} = \underset{\%}{2} + \underset{==}{1} + \begin{cases} \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[\,]}{8} + \\ +\underset{=}{1} + \underset{+}{1} + \underset{-}{2})), \\ 0 \end{cases} =$$
$$= 3 + \begin{cases} 2 + 4M + 14MQ, \\ 0 \end{cases} . \quad (2.8)$$

Результирующая трудоемкость алгоритма Винограда составляет $f_{vin} \approx 16MNQ$.

### 2.3.3 Трудоемкость оптимизированного алгоритма умножения матриц по Винограду

Трудоемкость этого алгоритма определяется из следующих компонентов по формуле (2.9):

$$f_{optim} = f_{init} + f_{precomp} + f_{fill}. \tag{2.9}$$

где:

– $f_{init}$ - определяется по формуле (2.5) в добавок с другими компонентами (2.10);

$$f_{init} = 4 + 4M + 4Q + \underset{=}{2} + \underset{\%}{2} + \underset{-}{1} + \underset{==}{1} + \begin{cases} \underset{-=}{1}, \\ 0 \end{cases} = 7 + 4M + 4Q + \begin{cases} \underset{-=}{1}, \\ 0 \end{cases}. \tag{2.10}$$

– $f_{precomp}$ - трудоемкость предварительного заполнения строк матрицы и столбцов матрицы $B$ (2.11):

$$\begin{aligned} f_{precomp} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + N(\underset{+=}{1} + \underset{<<}{1} + \underset{+}{1} + \underset{[\,]}{4}) + \underset{[\,]}{1} + \underset{=}{1}) + \\ &+ \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + N(\underset{+=}{1} + \underset{[\,]}{4} + \underset{<<}{1} + \underset{+}{1}) + \underset{=}{1}) = \\ &= 2 + 7M + 7MN + 2 + 5Q + 8NQ = 9MN + 9NQ + 7M + 5Q + 4; \end{aligned} \tag{2.11}$$

– $f_{fill}$ - трудоемкость для заполнения матрицы (2.12):

$$\begin{aligned} f_{fill} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{2} + \underset{-}{1} + \underset{+}{1} + \underset{[\,]}{2} + \underset{<}{1} + \\ &+ \frac{N}{2}(\underset{+=}{2} + \underset{[\,]}{8} + \underset{+}{4} + \underset{<<}{1}) + \underset{==}{1} + \begin{cases} \underset{+=}{1} + \underset{[\,]}{4} + \underset{<<}{1} \\ 0 \end{cases} + \underset{[\,]}{2} + \underset{=}{1})) = \\ &= 2 + M(4 + Q(\frac{15}{2}) + 13 + \begin{cases} 6 \\ 0 \end{cases})) = \\ &8.5MNQ + 14MQ + 4M + 2 + \begin{cases} 6 \\ 0 \end{cases} MQ; \end{aligned} \tag{2.12}$$

Результирующая трудоемкость оптимизированного алгоритма Винограда для лучшего и худшего случая составляет (2.13):

$$f_{fill} \approx 8.5MNQ. \tag{2.13}$$

### 2.3.4 Алгоритм Штрассена

Введем размер $S$ – ближайшая степень двойки, которая больше или равна размерам матриц $A$ и $B$.

Для алгоритма умножения матриц Штрассена трудоемкость будет складываться из следующих шагов:

1. Создания и инициализации матриц $a_{new}$ и $b_{new}$, трудоемкость которого (2.14):
$$f_{init} = f_{a\_new} + f_{b\_new} = 2 * S. \tag{2.14}$$
2. Заполнения матрицы a_new, трудоемкость которого (2.15):
$$f_{fill_a} = 2 + M(4 + K \cdot 5). \tag{2.15}$$
3. Заполнения матрицы b_new, трудоемкость которого (2.16):
$$f_{fill_b} = 2 + K(4 + N \cdot 5). \tag{2.16}$$
4. Вызова рекурсивной функции Штрассена, трудоемкость которого 0 и трудоемкости этой функции, которую посчитаем отдельно;
5. Заполнения матрицы C, трудоемкость которого (2.17):
$$f_{fill_c} = 2 + M \cdot 3. \tag{2.17}$$

Вычислим трудоемкость рекурсивной функции Штрассена. Пусть $n$ – размер матриц, которые передаются в эту функцию, тогда трудоемкость состоит из:

1. Создания и инициализации матриц $a_{11}, a_{12}, a_{21}, a_{22}, b_{11}, b_{12}, b_{21}, b_{22}$, трудоемкость (2.18):
$$f_{init} = 8 \cdot (\frac{n}{2} + \frac{n}{2}) = 8n. \tag{2.18}$$

2. Заполнения этих матриц, трудоемкость (2.19):
$$f_{fill} = 16 + 20n. \tag{2.19}$$

3. Вызова функции Штрассена 7 раз, трудоемкость которых 0, при этом трудоемкость вычисления передаваемых параметров (2.20):
$$f_{calc\_param} = 10 \cdot (2 + n + 2n^2). \tag{2.20}$$

4. Вычисления подматриц результата, трудоемкость (2.21):
$$f_{calc\_res} = 8 \cdot (2 + n + 2n^2). \tag{2.21}$$

5. Составления результирующей матрицы, трудоемкость (2.22):
$$f_{res} = 4 + 4n. \tag{2.22}$$

Таким образом, общая трудоемкость алгоритма Штрассена (2.23):
$$f_{sht} = 6 + 2S + 7M + 5MK + 5NK + \sum_{n=1}^{n/2} 52 + 28n + 36n^2 \tag{2.23}$$

Сумма происходит для $n$ по степеням двойки от 1 до $n/2$.

## 2.4 Структура программы

Программа состоит из следующих модулей:

– main.py: основной файл программы, в котором вызываются алгоритмы умножения матриц;

– matrix.py: файл, содержащий код всех представленных алгоритмов умножения матриц;

– test_time.py: замер времени выполнения каждого из алгоритмов;

– graph.py: файл отображение результатов замеров зависимости времени работы алгоритмов умножения матриц от размера матрицы;

– menu.py: перечень команд для взаимодействия с программой.

## Вывод

В данном разделе были приведены схемы алгоритмов умножения матриц стандартным образом, Винограда, Штрассена, проведена теоретическая оценка трудоемкости алгоритмов. Стандартный алгоритм умножения матриц имеет трудоемкость $14MNQ$, стандартный по Винограду $16MNQ$, оптимизированный по Винограду $8.5MNQ$.

# 3 Технологический раздел

В данном разделе приводятся требования к программному обеспечению, выбор средств реализации и модули программы.

## 3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

– на вход программе подаются два массива сгенерированных целых чисел, корректный размер которых задает пользователь;

– осуществляется выбор алгоритма умножения матриц из меню;

– на вход программе подаются только корректные данные;

– на выходе программа выдает результат – матрицу, полученную в результате умножения двух входных.

## 3.2 Выбор средств реализации

Для реализации алгоритмов был выбран язык программирования Python [4]. При замере процессорного времени используется модуль time с функцией process_time [5].

## 3.3 Модули программы

### 3.3.1 Стандартное умножение матриц

Реализация умножения матриц стандартным образом приведена на листинге 3.1.

Листинг 3.1 – Стандартное умножение матриц

```
def multiply_matrixes_ordinary(matrix_a, matrix_b) -> list[list[int]]:
    n, m = matrix_a.get_size(); q, p = matrix_b.get_size()
    if m != q:
        print('Несовпадение размеров матриц')
        return
    else:
        matrix_result = Matrix(n, p)
        for i in range(n):
            for j in range(p):
                for k in range(m):
                    matrix_result[i][j] =
                        matrix_result[i][j] + matrix_a[i][k] * matrix_b[k][j]
        return matrix_result
```

### 3.3.2 Умножение матриц по Винограду стандартный

Реализация умножения матриц по Винограду приведена на листинге 3.2.

Листинг 3.2 – Умножение матриц по Винограду

```
def multiply_matrixes_vinograd(matrix_a, matrix_b) -> list[list[int]]:
    n, m = matrix_a.get_size()
    q, p = matrix_b.get_size()
    if m != q:
        print('Несовпадение размеров матриц')
        return
    else:
        matrix_result = Matrix(n, p)
        d = int(m / 2)
        mul_u = [0] * n
        for i in range(n):
            for j in range(d):
                mul_u[i] = mul_u[i] +
                    matrix_a[i][2*j] * matrix_a[i][2*j + 1]

        mul_w = [0] * p
        for i in range(p):
            for j in range(d):
                mul_w[i] = mul_w[i] +
                    matrix_b[2*j][i] * matrix_b[2*j + 1][i]

        for i in range(n):
            for j in range(p):
                matrix_result[i][j] = -mul_u[i] - mul_w[j]
                for k in range(d):
                    matrix_result[i][j] =
                        matrix_result[i][j] +
                            (matrix_a[i][2*k] + matrix_b[2*k+1][j]) * \
                                (matrix_a[i][2*k+1] + matrix_b[2*k][j])

        if m % 2 == 1:
            for i in range(n):
                for j in range(p):
                    matrix_result[i][j] = matrix_result[i][j] +
                        matrix_a[i][m-1] * matrix_b[m-1][j]

        return matrix_result
```

### 3.3.3 Умножение матриц по Винограду оптимизированный

Реализация оптимизированного умножения матриц по Винограду приведена на листинге 3.3.

Листинг 3.3 – Умножение матриц по Винограду (оптимизированный)

```
    def multiply_matrixes_vinograd_optimized(matrix_a, matrix_b) -> list[list[int]]:
        rows_a, cols_a = matrix_a.get_size()
        rows_b, cols_b = matrix_b.get_size()

        if cols_a != rows_b:
            return "Умножение невозможно."

        result = Matrix(rows_a, cols_b)

        mul_row = [0] * rows_a
        mul_col = [0] * cols_b

        for i in range(rows_a):
            for j in range(cols_a >> 1):
                mul_row[i] += matrix_a[i][j << 1] * matrix_a[i][(j << 1) + 1]

        for i in range(cols_b):
            for j in range(rows_b >> 1):
                mul_col[i] += matrix_b[j << 1][i] * matrix_b[(j << 1) + 1][i]

        flag = cols_a % 2

        for i in range(rows_a):
            for j in range(cols_b):
                result[i][j] = -mul_row[i] - mul_col[j]

                for k in range(1, cols_a, 2):
                    result[i][j] +=
                        (matrix_a[i][k - 1] + matrix_b[k][j]) *
                            (matrix_a[i][k] + matrix_b[k - 1][j])

                if flag:
                    result[i][j] +=
                        matrix_a[i][cols_a - 1] * matrix_b[rows_b - 1][j]

        return result
```

### 3.3.4 Умножение матриц Штрассена

Реализация умножения матриц Штрассена приведена на листинге 3.4.

Листинг 3.4 – Умножение матриц Штрассена

```
def multiply_matrixes_strassen(matrix_a, matrix_b):
    rows_a, cols_a = matrix_a.get_size()
    rows_b, cols_b = matrix_b.get_size()

    if cols_a != rows_b:
        return "Умножение невозможно"

    size = max(rows_a, cols_a, rows_b, cols_b)
    size = 2 ** (size - 1).bit_length()

    A_padded = [[0] * size for _ in range(size)]
    B_padded = [[0] * size for _ in range(size)]

    for i in range(rows_a):
        for j in range(cols_a):
            A_padded[i][j] = matrix_a[i][j]

    for i in range(rows_b):
        for j in range(cols_b):
            B_padded[i][j] = matrix_b[i][j]

    C_padded = strassen_recursive(A_padded, B_padded)

    C = []
    for i in range(rows_a):
        C.append(C_padded[i][:cols_b])

    return C
```

Вызов рекурсивной функции Штрассена приведена на листинге 3.5 и 3.6.

Листинг 3.5 – Вызов рекурсивной функции Штрассена. Часть 1

```
def strassen_recursive(matrix_a, matrix_b):
    n = len(matrix_a)

    if n == 1:
        return [[matrix_a[0][0] * matrix_b[0][0]]]

    mid = n // 2
    A11 = [matrix_a[i][:mid] for i in range(mid)]
    A12 = [matrix_a[i][mid:] for i in range(mid)]
    A21 = [matrix_a[i][:mid] for i in range(mid, n)]
    A22 = [matrix_a[i][mid:] for i in range(mid, n)]
    B11 = [matrix_b[i][:mid] for i in range(mid)]
    B12 = [matrix_b[i][mid:] for i in range(mid)]
    B21 = [matrix_b[i][:mid] for i in range(mid, n)]
    B22 = [matrix_b[i][mid:] for i in range(mid, n)]

    M1 = strassen_recursive(matrix_add(A11, A22), matrix_add(B11, B22))
    M2 = strassen_recursive(matrix_add(A21, A22), B11)
    M3 = strassen_recursive(A11, matrix_sub(B12, B22))
    M4 = strassen_recursive(A22, matrix_sub(B21, B11))
    M5 = strassen_recursive(matrix_add(A11, A12), B22)
    M6 = strassen_recursive(matrix_sub(A11, A21), matrix_add(B11, B12))
    M7 = strassen_recursive(matrix_sub(A12, A22), matrix_add(B21, B22))
```

Листинг 3.6 – Вызов рекурсивной функции Штрассена. Часть 2

```
        C11 = matrix_add(matrix_sub(matrix_add(M1, M4), M5), M7)
        C12 = matrix_add(M3, M5)
        C21 = matrix_add(M2, M4)
        C22 = matrix_sub(matrix_sub(matrix_add(M1, M3), M2), M6)


        C = []
        for i in range(mid):
            C.append(C11[i] + C12[i])
        for i in range(mid, n):
            C.append(C21[i-mid] + C22[i-mid])

        return C
```

Матричное сложение и вычитание приведены на листинге 3.7.

Листинг 3.7 – Матричное сложение и вычитание

```
    def matrix_add(mat_a, mat_b):
        return [[mat_a[i][j] + mat_b[i][j] for j in range(len(mat_a[i]))] for i in range(len(mat_a))]

    def matrix_sub(mat_a, mat_b):
        return [[mat_a[i][j] - mat_b[i][j] for j in range(len(mat_a[i]))] for i in range(len(mat_a))]
```

## 3.4 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов.

Таблица 3.1 – Таблица тестов

| № | Описание теста | Матрица 1 | Матрица 2 | Ожидаемый результат |
|---|---|---|---|---|
| 1 | Квадратный размер | $\begin{pmatrix} 6 & 9 & 8 \\ 0 & 3 & 6 \\ 4 & 9 & 5 \end{pmatrix}$ | $\begin{pmatrix} 9 & 6 & 0 \\ 2 & 3 & 5 \\ 6 & 8 & 7 \end{pmatrix}$ | $\begin{pmatrix} 120 & 127 & 101 \\ 42 & 57 & 57 \\ 84 & 91 & 80 \end{pmatrix}$ |
| 2 | Разный размер | $\begin{pmatrix} 2 & 3 & 2 \\ 3 & 9 & 2 \end{pmatrix}$ | $\begin{pmatrix} 2 & 4 \\ 1 & 7 \\ 4 & 1 \end{pmatrix}$ | $\begin{pmatrix} 15 & 31 \\ 23 & 77 \end{pmatrix}$ |
| 3 | Разный размер | $\begin{pmatrix} 5 & 1 \\ 0 & 4 \\ 6 & 4 \end{pmatrix}$ | $\begin{pmatrix} 2 & 1 & 7 \\ 4 & 6 & 1 \end{pmatrix}$ | $\begin{pmatrix} 14 & 11 & 36 \\ 16 & 24 & 4 \\ 28 & 30 & 46 \end{pmatrix}$ |
| 4 | Неподходящий размер | $\begin{pmatrix} 5 & 1 \\ 0 & 4 \end{pmatrix}$ | $\begin{pmatrix} 5 & 1 \\ 0 & 4 \end{pmatrix}$ | Несоответствие размеров. |

Для алгоритмов умножения матриц стандартным образом, по Виногра-

ду и оптимизированный по Винограду, а также Штрассена данные тесты были пройдены успешно.

## Вывод

В данном разделе был выбран язык программирования, средства замера процессорного времени. Реализованы функции, описанные в аналитическом разделе, и проведено их тестирование методом черного ящика по таблице 3.1.

# 4 Исследовательский раздел

В данном разделе приводятся технические характеристики устройства, анализ зависимости времени выполнения алгоритмов от четного и нечетного размера квадратных матриц.

## 4.1 Технические характеристики

Технические характеристии устройства, на котором выполнялось тестирование:

- операционная система: Windows 10 Pro;
- память: 8 GiB;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

## 4.2 Временные характеристики выполнения

Проведем анализ зависимости времени работы алгоритмов умножения матриц от размера исходных матриц. Рассмотрим вариант для лучшего случая, когда размер матрицы имеет четные значения, и для худшего при нечетном размере матриц. Исходными данными является квадратная матрица целых чисел. Единичные замеры выдадут крайне маленький результат, поэтому проведем работу каждого алгоритма n = 5 раз и поделим на число n. Получим среднее значение работы каждого из алгоритмов.

Выполним анализ для случая, когда размер квадратных матриц целых чисел имеет четное значение {100, 200,..., 1000}. Результат зависимости времени выполнения умножения матриц четного размера приведен на рисунке 4.1.

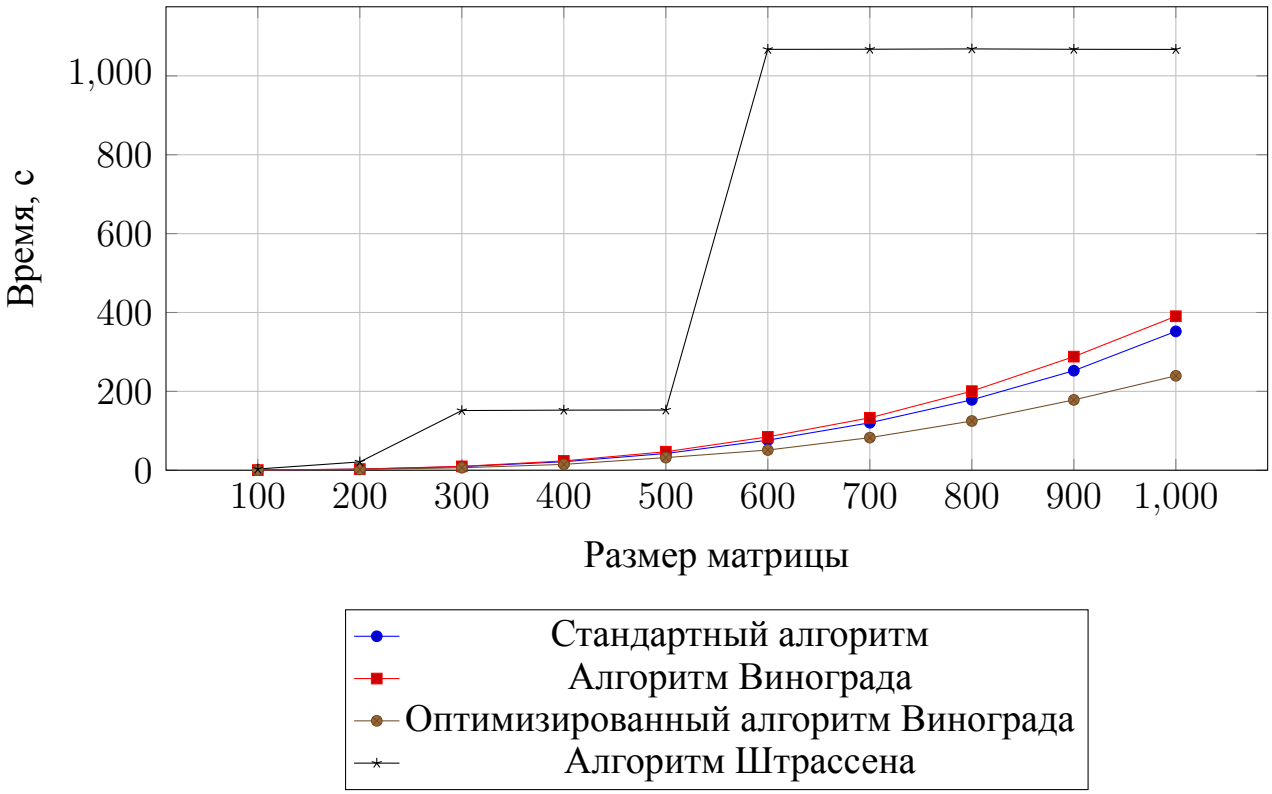


Рисунок 4.1 – Зависимости времени работы алгоритмов при чётных размерностях матриц

Алгоритм умножения матриц по Винограду работает медленнее стандартного приблизительно на 9%. Оптимизированная версия алгоритма Винограда выполняет вычисления быстрее стандартного на размерах {700, ..., 1000} в среднем на 40% и выигрывает по скорости у обычного алгоритма умножения по Винограду на 51% на размерах {700, ..., 1000}. Алгоритм Штрассена медленнее других способов умножения матриц в среднем в 15-17 раз до размера матриц 500; с 500 выполняется резкое увеличение времени выполнения алгоритма Штрассена. Разница с другими алгоритмами составляет в среднем 8-10 раз.

Выполним анализ для случая, когда размер матриц целых чисел имеет нечетный размер {101, 201,..., 1001}. Результат приведен на рисунке 4.2.

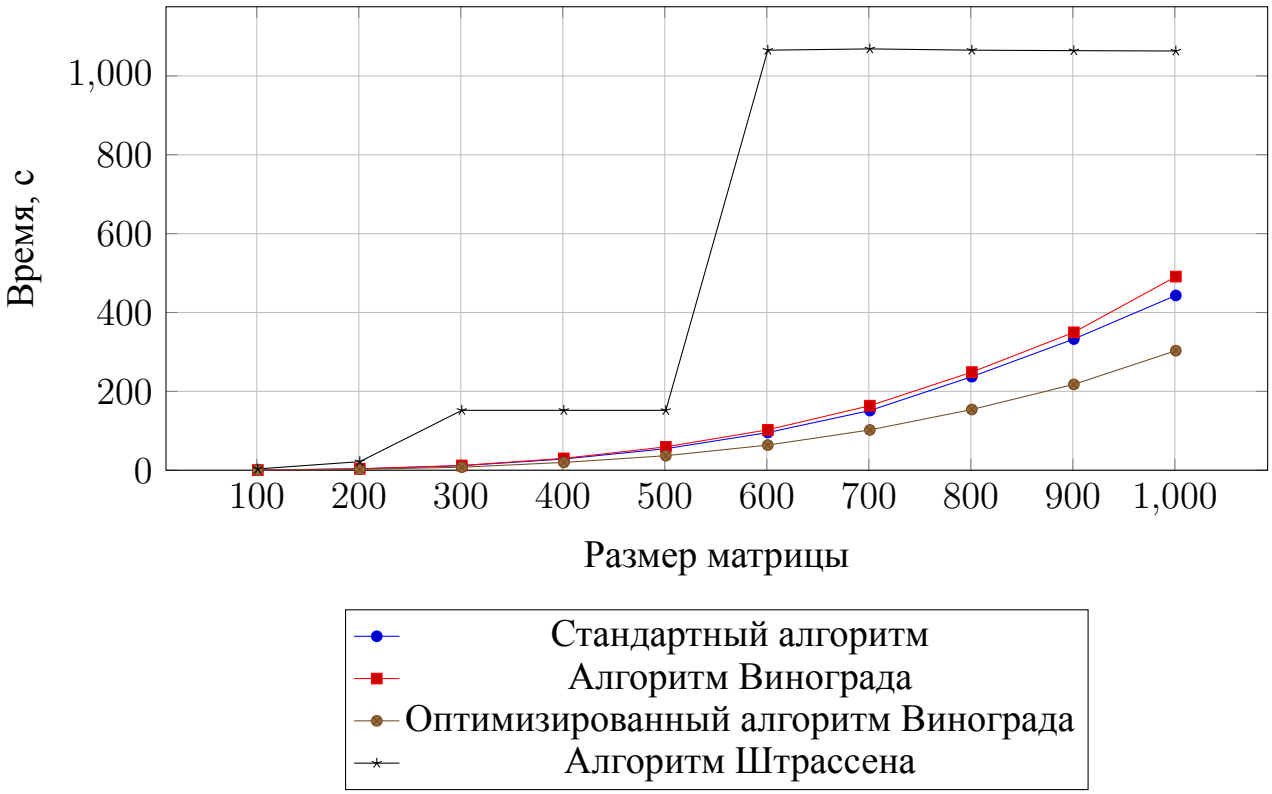


Рисунок 4.2 – График зависимости времени работы алгоритмов при нечётных размерностях матриц

Алгоритм умножения матриц по Винограду работает медленнее стандартного приблизительно на 12% на размерах {701, ..., 1001}. Оптимизированная версия алгоритма Винограда выполняет вычисления быстрее стандартного примерно на 47% и выигрывает по скорости у обычного алгоритма умножения по Винограду в среднем на 63% на размерах {701, ..., 1001}.

## Вывод

Экспериментально была подтвержена трудоемкость алгоритмов умножения матриц, описанная в разделах 2.3.1, 2.3.2, 2.3.3, 2.3.4. Время выполнения, описанное в разделе 4.2., каждого из этих методов при нечетных размерностях матриц больше времения выполнения тех же методов в случае, когда размерность матриц четная. Это связано с дополнительными операциями обработки, указанных в схемах алгоритма раздела 2.1.

## Заключение

В ходе выполнения лабораторной работы были рассмотрены алгоритмы умножения матриц стандартным способом, по Винограду, Штрассена. Проведена оптимизация алгоритма Винограда. Выполнено описание каждого из этих алгоритмов, приведены соотвествующие математические расчёты трудоемкости каждого из них.

При анализе временных характеристик каждого из этих алгоритмов можно сделать следующие выводы: при помощи оптимизации алгоритма Винограда удалось уменьшить трудоемкость стандартного способа умножения матриц в среднем на 47% при нечетных размерах, и в среднем на 40% при четных. Это решение позволит выполнять вычисления на количестве данных порядка тысячи и выше быстрее стандартного способа.

Умножение матриц при помощи алгоритма Штрассена увеличивает время выполнения операции в среднем в 8-10 раз стандартного способа и алгоритма Винограда.

Из набора алгоритмов умножения матриц «стандартный», «по Винограду», «по Винограду оптимизированный», «Штрассена» необходимо использовать оптимизированную версию Винограда.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Канатников, А. Н. Аналитическая геометрия : учебник / А. Н. Канатников, А. П. Крищенко ; под редакцией В. С. Зарубина, А. П. Крищенко. — 9-е изд. — Москва : МГТУ им. Баумана, 2019. — 376 с. — ISBN 978-5-7038-4904-0.

2. Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions //Proceedings of the nineteenth annual ACM symposium on Theory of computing. – 1987. – С. 1-6.

3. Томас Х. Алгоритмы: построение и анализ, / Лейзерсон Ч., Чарльз И., Ривест Р., Рональд Л., Штайн К. 2-е издание. : Пер. с англ. — М. : Издательский дом “Вильямс”, 2011. — 1296 с. : ил. — Парал. тит. англ. ISBN 978-5–8459–0857– (рус.)

4. Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/ (дата обращения: 12.02.2024)

5. Time access and conversions — Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/library/time.html (дата обращения: 12.02.2024)

6. The Python Profilers — Python 3.12.1 documentation [Электронный ресурс]. – Режим доступа, URL: https://docs.python.org/3/library/profile.html (дата обращения: 05.02.2024)