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»

Отчёт
по лабораторной работе №2

Название: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-54Б _____ Л.Е.Тартыков
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л.Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Содержание

Введение **4**

1 Аналитический раздел **5**

1.1 Определение матрицы 5

1.2 Стандартный алгоритм умножения матриц 5

1.3 Алгоритм умножения матриц по Винограду 5

1.4 Вывод 6

2 Конструкторский раздел **7**

2.1 Модель оценки трудоемкости алгоритмов 7

2.2 Вычисление трудоемкости алгоритмов 8

2.2.1 Трудоемкость алгоритма умножения матриц по Винограду 8

2.2.2 Трудоемкость алгоритма умножения матриц по Винограду 8

2.2.3 Трудоемкость оптимизированного алгоритма умножения матриц по Винограду 10

2.3 Схемы алгоритмов умножения матриц 11

2.4 Вывод 20

3 Технологический раздел **21**

3.1 Требования к программному обеспечению 21

3.2 Выбор средств реализации 21

3.3 Листинги программ 21

3.4 Вспомогательные модули 25

3.5 Тестирование 25

3.6 Вывод 26

4 Исследовательский раздел **27**

4.1 Технические характеристики 27

4.2 Временные харастеристики выполнения 27

4.3 Вывод 29

Заключение **30**

Список литературы **31**

Введение

Матричное умножение лежит в основе нейронных сетей. Большинство операций при обучении нейронной сети требуют некоторой формы умножения 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. 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:

- изучить алгоритм умножения матриц стандартным способом и по Винограду;
- разработать данные алгоритмы;
- оптимизировать алгоритм Винограда;
- выполнить тестирование реализации алгоритмов методом черного ящика;
- провести сравнительный анализ этих алгоритмов по затратам памяти и процессорному выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

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 может быть записана по формуле (1.1).

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

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

$$c_{ij} = \overline{u} \times \overline{v} = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4 \quad (1.3)$$

Эту формулу можно представить в следующем виде (1.4).

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

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 Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц стандартным образом и по Винограду. Были приведены соответствующие математические расчеты.

2 Конструкторский раздел

2.1 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, но она

определяется по формуле (2.2).

$$f_{if} = f_{comp\_if} + \begin{bmatrix} f_a \\ f_b \end{bmatrix} \tag{2.2}$$

2.2 Вычисление трудоемкости алгоритмов

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

2.2.1 Трудоемкость алгоритма умножения матриц по Винограду

Трудоемкость этого алгоритма вычисляется по формуле (2.3).

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

2.2.2 Трудоемкость алгоритма умножения матриц по Винограду

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

$$f_{vin} = f_{init} + f_{precomp} + f_{fill} + f_{even} \tag{2.4}$$

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).

$$f_{fill} = \underset{\%}{2} + \underset{==}{1} + \begin{bmatrix} \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[\,]}{8} + \\ +\underset{=}{1} + \underset{+}{1} + \underset{-}{2})), \\ 0 \end{bmatrix} =$$

$$= 3 + \begin{bmatrix} 2 + 4M + 14MQ, \\ 0 \end{bmatrix} \quad (2.8)$$

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

2.2.3 Трудоемкость оптимизированного алгоритма умножения матриц по Винограду

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

$$f_{optim} = f_{init} + f_{precomp} + f_{fill} \quad (2.9)$$

где:

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

$$f_{init} = 4+4M+4Q+\underset{=}{2}+\underset{\%}{2}+\underset{-}{1}+\underset{==}{1}+\begin{bmatrix} \underset{-=}{1}, \\ 0 \end{bmatrix} = 7+4M+4Q+\begin{bmatrix} \underset{-=}{1}, \\ 0 \end{bmatrix} \quad (2.10)$$

- $f_{precomp}$ - трудоемкость предварительного заполнения строк матри-

цы A и столбцов матрицы B (2.11);

$$f_{precomp} = \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + N(\underset{+=}{2} + \underset{*}{2} + \underset{+}{1} + \underset{[\,]}{4}) + \underset{[\,]}{1} + \underset{=}{1}) +$$
$$+ \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + N(\underset{+=}{2} + \underset{[\,]}{4} + \underset{*}{2} + \underset{+}{1}) + \underset{=}{1}) =$$
$$= 2 + 7M + 9MN + 2 + 6Q + 9NQ = 9MN + 9NQ + 7M + 6Q + 4 \tag{2.11}$$

- f_{fill} - трудоемкость для заполнения матрицы (2.12).

$$f_{fill} = \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{2} + \underset{-}{1} + \underset{+}{1} + \underset{[\,]}{2} + \underset{<}{1} + \frac{N}{2}(\underset{+=}{2} + \underset{[\,]}{8} + \underset{+}{4} + \underset{*}{2}) + \underset{==}{1} +$$
$$+ \begin{bmatrix} \underset{+=}{1} + \underset{[\,]}{4} + \underset{*}{2} \\ 0 \end{bmatrix} + \underset{[\,]}{2} + \underset{=}{1})) =$$
$$= 2 + M(4 + Q(\frac{16}{2}) + 13 + \begin{bmatrix} 6 \\ 0 \end{bmatrix}) =$$
$$8MNQ + 14MQ + 4M + 2 + \begin{bmatrix} 6 \\ 0 \end{bmatrix} MQ \tag{2.12}$$

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

$$f_{fill} \approx 8MNQ \tag{2.13}$$

2.3 Схемы алгоритмов умножения матриц

Ниже представлены следующие схемы алгоритмов:

- рис. 2.1 - схема алгоритма стандартного умножения матриц;

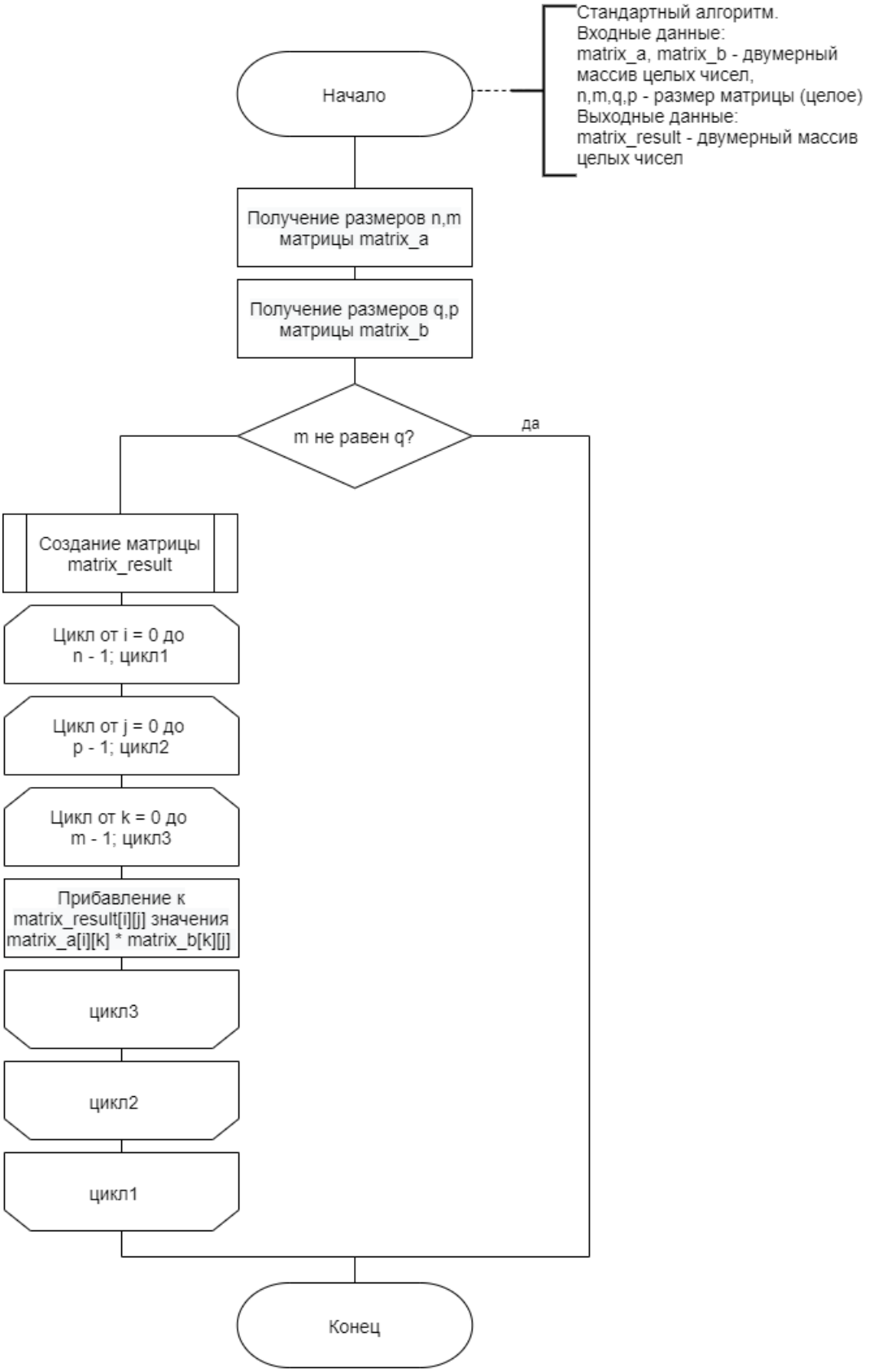

Рисунок 2.1 – Схема алгоритма стандартного умножения матриц.

- рис. 2.2, 2.3, 2.4 - схема алгоритма умножения матриц по Винограду;

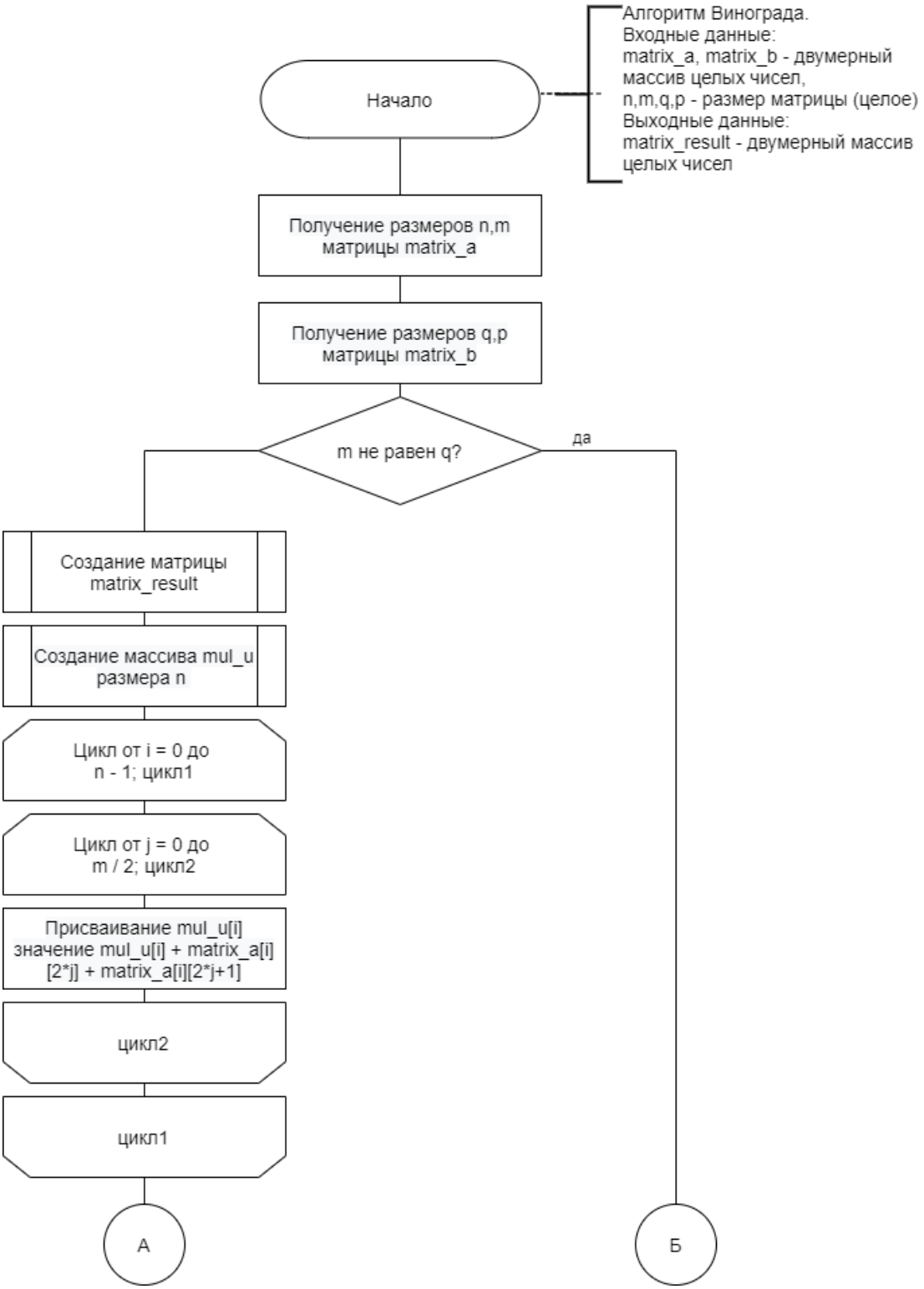

Рисунок 2.2 – Схема алгоритма умножения матриц по Винограду.

Рисунок 2.3 – Схема алгоритма умножения матриц по Винограду.

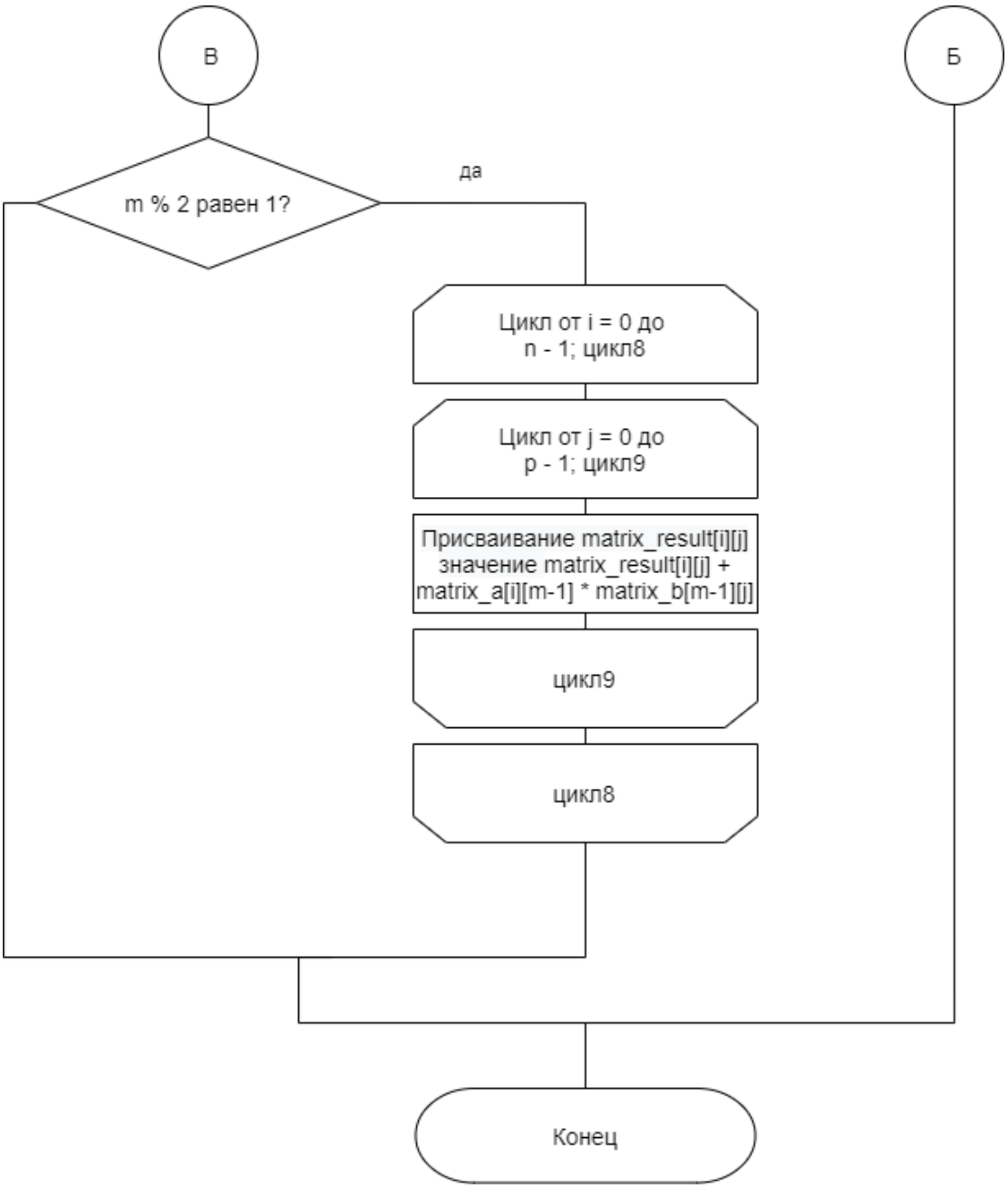

Рисунок 2.4 – Схема алгоритма умножения матриц по Винограду.

- рис. 2.5, 2.6, 2.7, 2.8 - схема алгоритма оптимизации алгоритма Винограда.

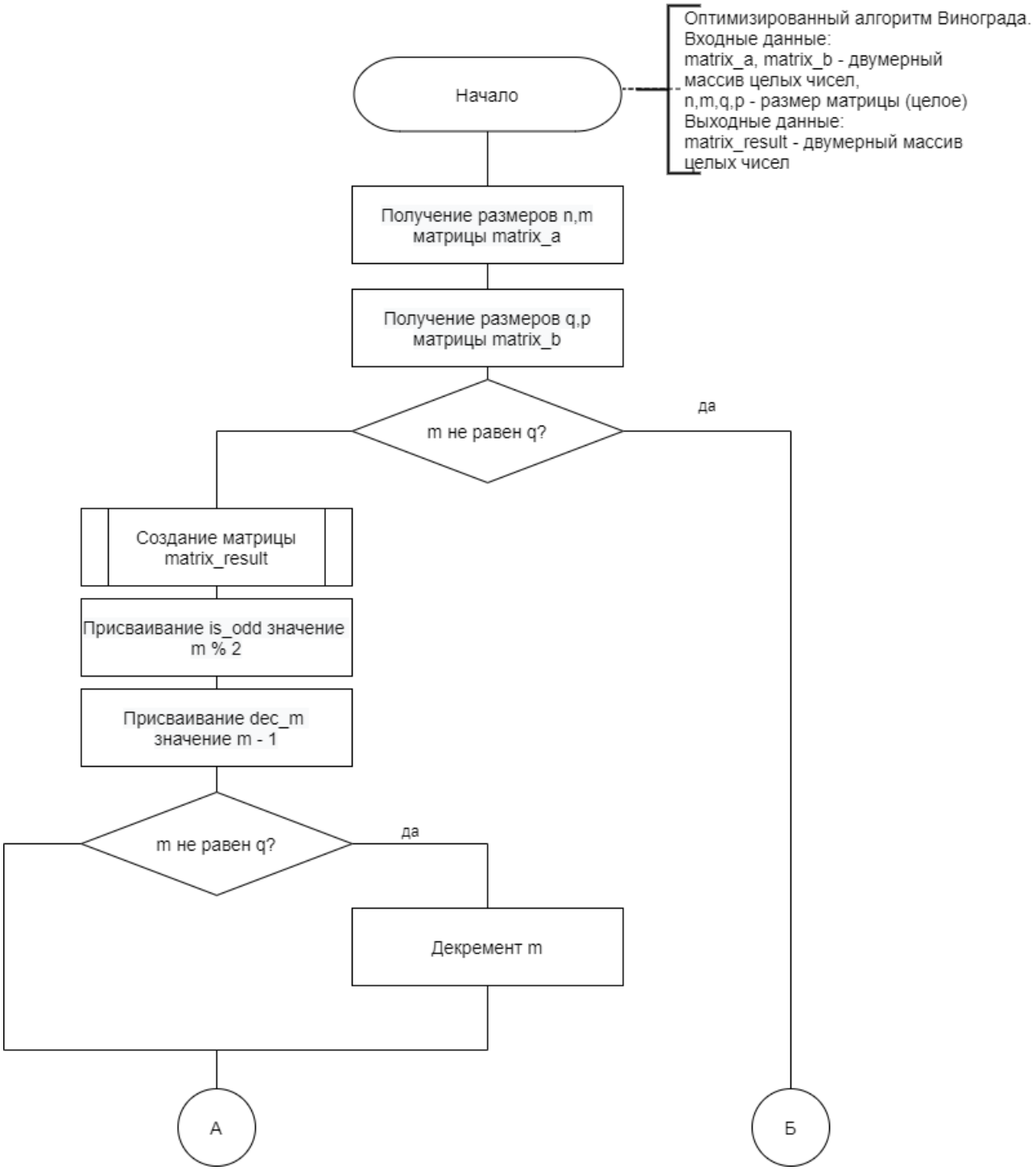

Рисунок 2.5 – Схема алгоритма оптимизации алгоритма Винограда.

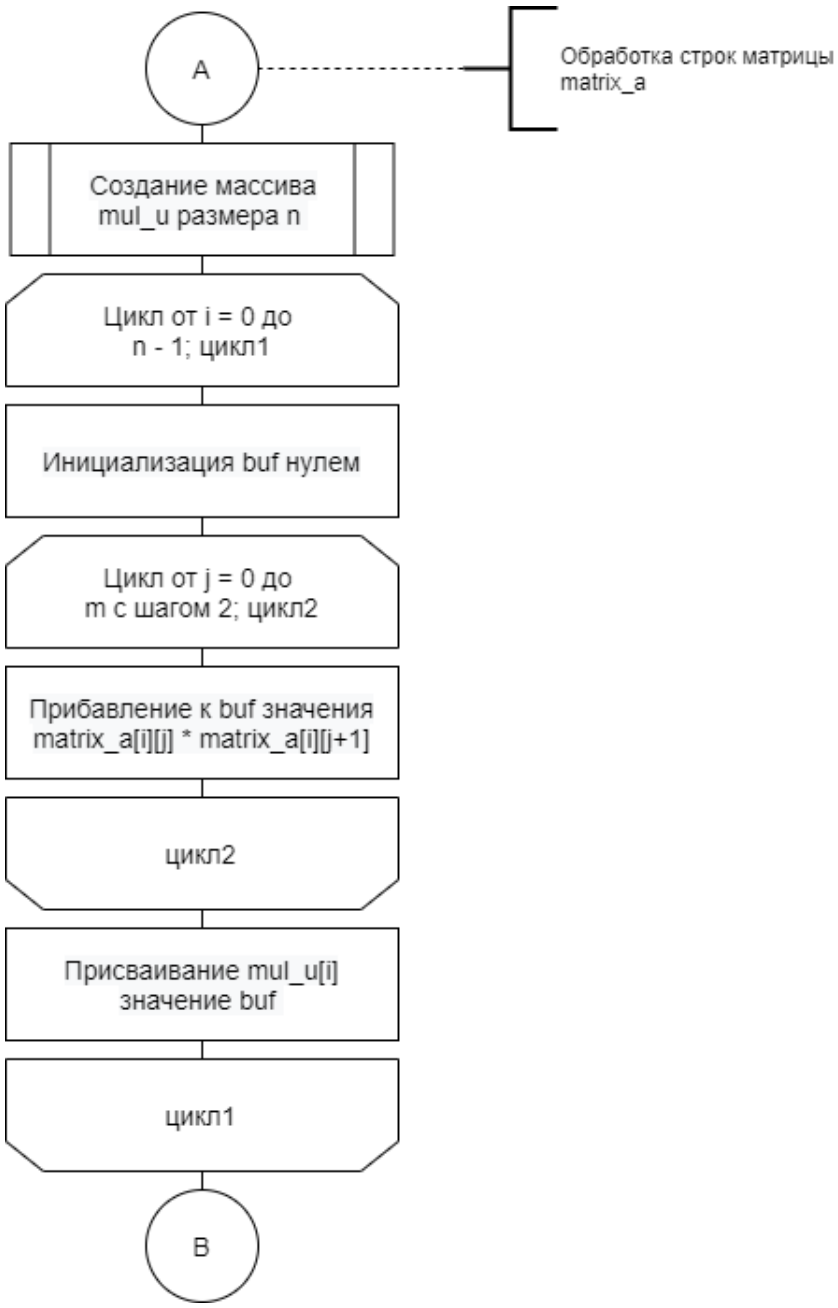

Рисунок 2.6 – Схема алгоритма оптимизации алгоритма Винограда.

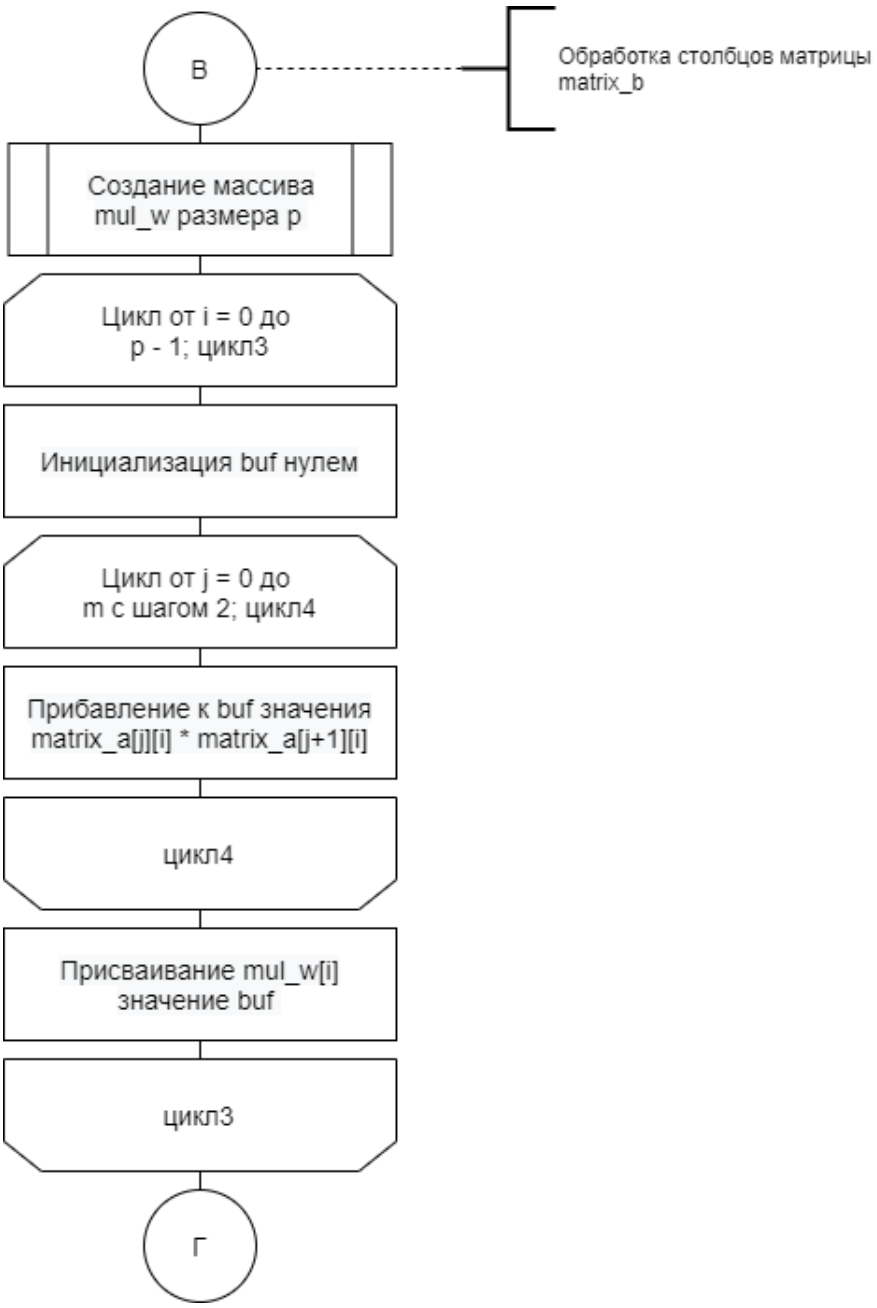

Рисунок 2.7 – Схема алгоритма оптимизации алгоритма Винограда.

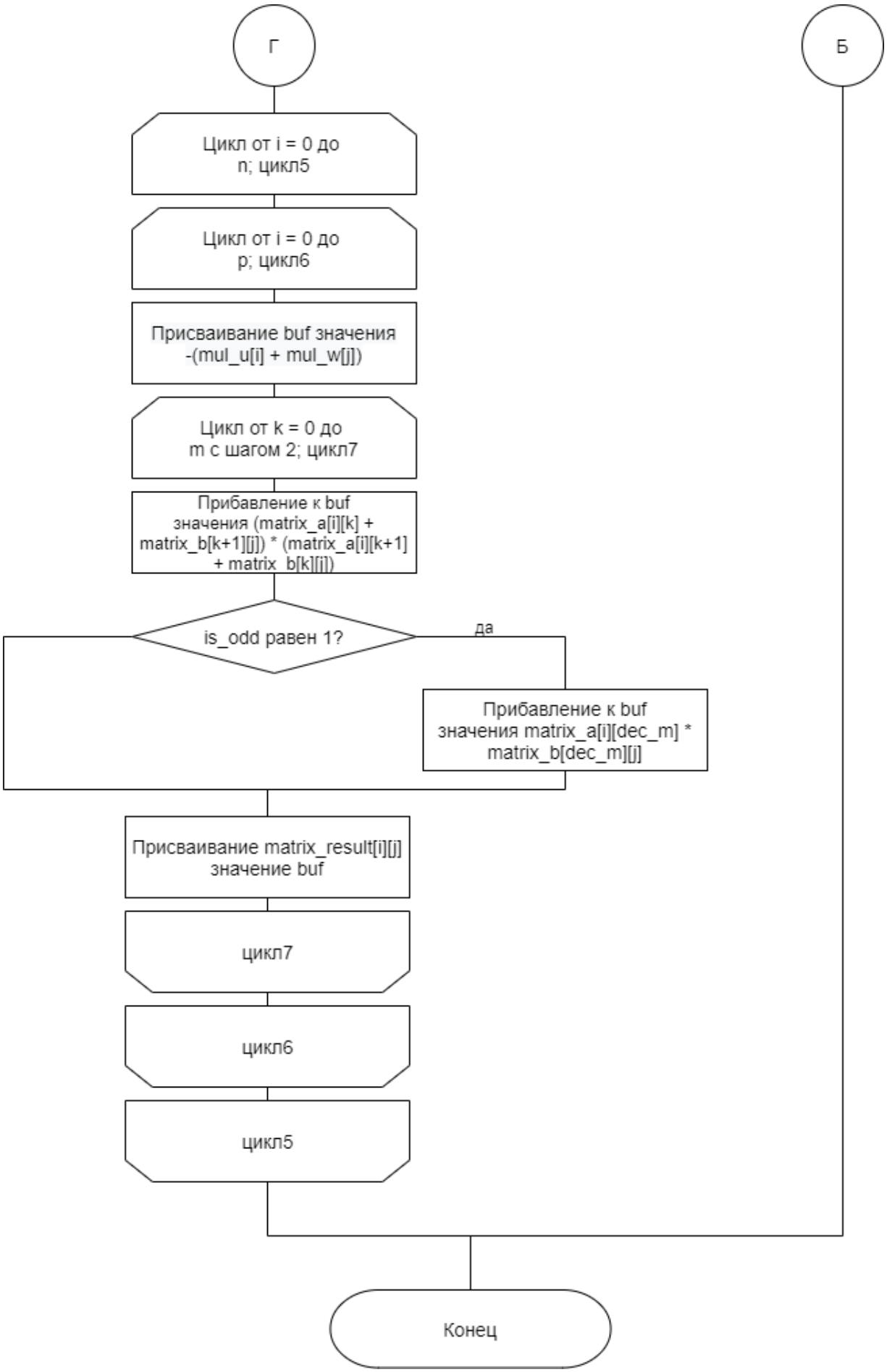

Рисунок 2.8 – Схема алгоритма оптимизации алгоритма Винограда.

2.4 Вывод

На основе теоретических данных, полученных в аналитическом разделе, были построены схемы необходимых алгоритмов. Эти алгоритмы были проанализированы с точки зрения трудоемкость выполнения. Алгоритм умножения матриц по Винограду работает медленнее стандартного алгоритма на MNK. Зато оптимизированный вариант алгоритма Винограда дает выигрыш по сравнению с ним на 6MNK.

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

- на вход программе подаются два массива целых сгенерированных чисел, корректный размер которых задает пользователь;
- осуществляется выбор алгоритма умножения матриц из меню;
- на вход программе подаются только корректные данные данные;
- на выходе программа выдает результат - полученную матрицу.

3.2 Выбор средств реализации

Для реализации алгоритмов в данной лабораторной работе был выбран язык программирования Python 3.9.7[3]. На нем можно выполнять быструю разработку программ. Имеется опыт разработки на этом языке. В качестве среды разработки был использован Visual Studio Code[4], так как в нем присутсвует поддержка практически всех языков программирования. При замере процессорного времени был использован модуль time[5].

3.3 Листинги программ

Ниже представлены листинги разработанных алгоритмов умножения матриц.

Листинг 3.1 – Программный код умножения матриц стандартным способом.

```
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('Dismatch matrix sizes.')
        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
```

Листинг 3.2 – Программный код умножения 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('Dismatch matrix sizes.')
        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
```

Листинг 3.3 – Программный код оптимизированного алгоритма умножения матриц по Винограду.

```
def multiply_matrixes_vinograd_optimized(matrix_a, matrix_b) -> list[
    list[int]]:
     n, m = matrix_a.get_size()
     q, p = matrix_b.get_size()
     if m != q:
         print('Dismatch matrix sizes.')
         return
     else:
         matrix_result = Matrix(n, p)
         is_odd = m % 2
         dec_m = m - 1
         mul_u = [0] * n
         if is_odd:
             m -= 1
         for i in range(n):
             buf = 0
             for j in range(0, m, 2):
                 buf += matrix_a[i][j] * matrix_a[i][j + 1]
             mul_u[i] = buf

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

         for i in range(n):
             for j in range(p):
                 buf = -(mul_u[i] + mul_w[j])
                 for k in range(0, m, 2):
                     buf += (matrix_a[i][k] + matrix_b[k+1][j]) * \
                             (matrix_a[i][k+1] + matrix_b[k][j])

                 if is_odd == 1:
                     buf += matrix_a[i][dec_m] * matrix_b[dec_m][j]
                 matrix_result[i][j] = buf

         return matrix_result
```

3.4 Вспомогательные модули

На листингах представлены программные модули, которые используются в данных функциях:

Листинг 3.4 – Программный код класса по работе с матрицами.

```
class Matrix():
    def __init__(self, n, m):
        self.__n, self.__m = n, m
        self.create_matrix()

    def create_matrix(self):
        self.matrix = [[0] * self.__m for i in range(self.__n)]

    def output_matrix(self):
        for i in range(self.__n):
            for j in range(self.__m):
                print(self.matrix[i][j], end=' ')
            print()

    def __getitem__(self, index):
        return self.matrix[index]

    def __setitem__(self, key, value):
        self.matrix[key] = value

    def get_size(self):
        return self.__n, self.__m

    def fill_matrix_random_values(self):
        for i in range(self.__n):
        for j in range(self.__m):
            self.matrix[i][j] = randint(MIN_RAND, MAX_RAND)
```

3.5 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 Винограду и оптимизированный по Винограду данные тесты были пройдены успешно.

3.6 Вывод

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

4 Исследовательский раздел

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 харастеристики выполнения

Ниже был проведен анализ времени работы алгоритмов. 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 матриц целых чисел имеет нечетный размер {101, 201,..., 1001}. Результат приведен на рис 4.1.

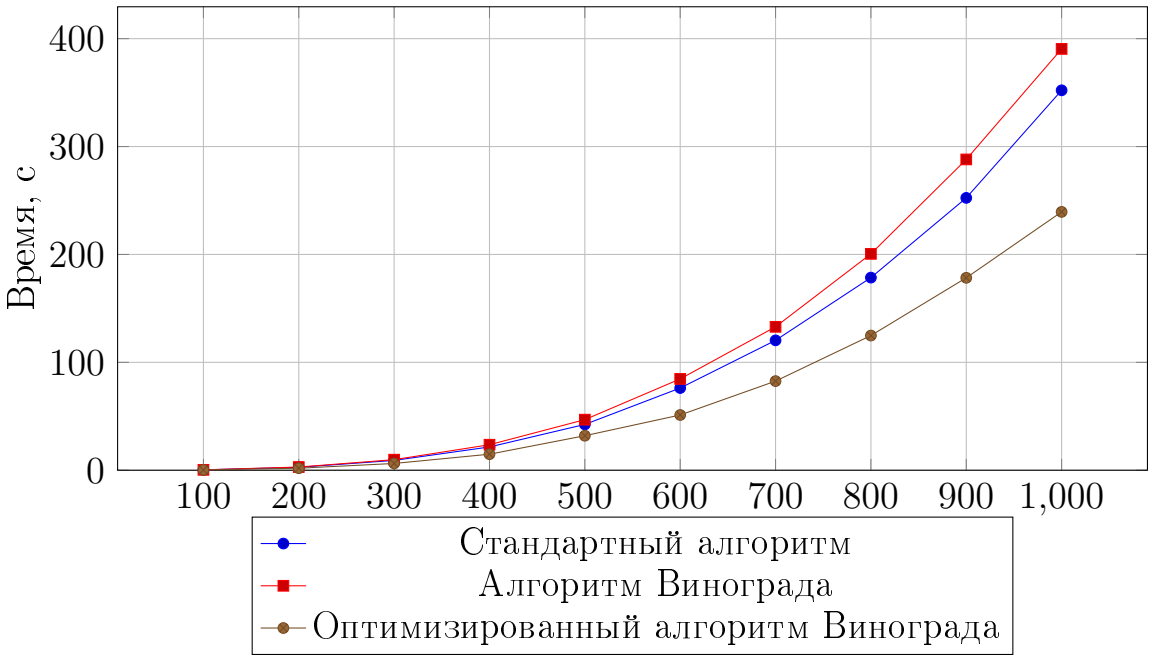

Рисунок 4.1 – График зависимости времени работы алгоритмов при чётных размерностях матриц

Как видно из результатов, алгоритм умножения матриц по Винограду работает медленнее стандартного приблизительно на 12% на размерах {700, ..., 1000}. 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 {700, ..., 1000}.

Выполним анализ для случая, когда размер матриц целых чисел имеет четный размер {100, 200,..., 1000}. Результат приведен на рис 4.2.

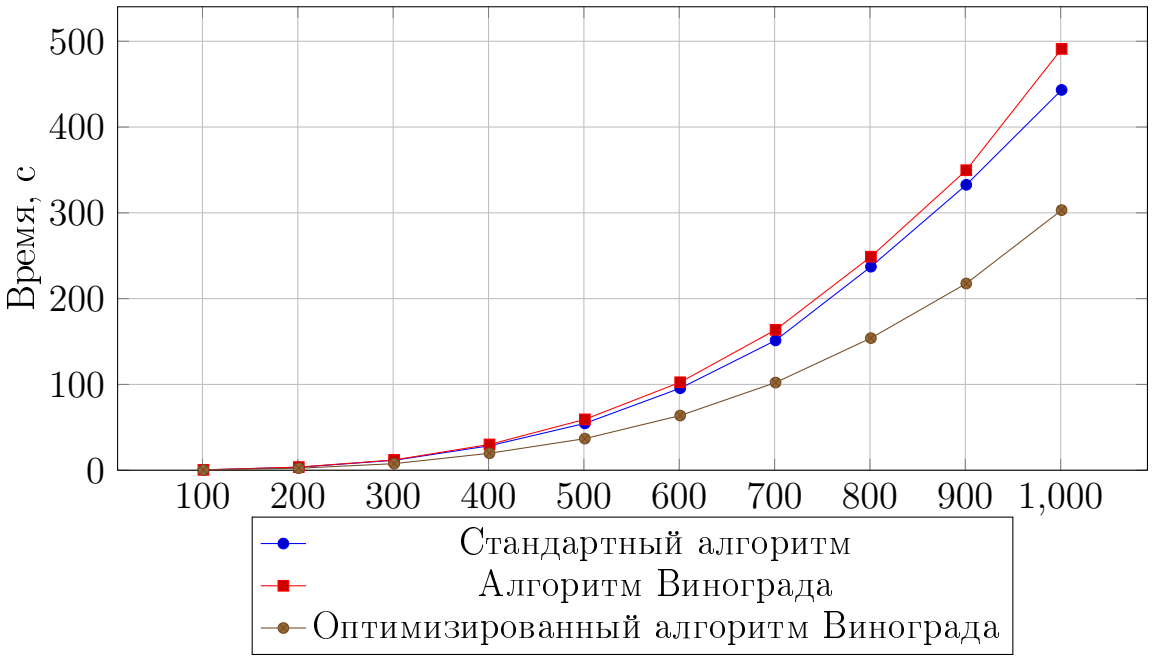

Рисунок 4.2 – График зависимости времени работы алгоритмов при чётных размерностях матриц

Как видно из результатов, все приведенные алгоритмы работают дольше, чем в ситуации, кога размер матриц четный. 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}.

4.3 Вывод

Экспериментально была подтвержена трудоемкость алгоритмов умножения матриц, описанная в разделах 2.2.1, 2.2.2, 2.2.3. 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 2.3.

Заключение

В ходе выполнения лабораторной работы были рассмотрены алгоритмы умножения матриц стандартным способом, по Винограду. Были выполнено описание каждого из этих алгоритмов, приведены соответствующие математические расчёты.

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 большом количестве данных порядка миллиарда и выше быстрее, чем станадартным способом. Эта разница в скорости выполнения операций будет заметна.

Литература

[1] Канатников А. Н. Крищенко А. П. А. Н. Ка-натников, А. П. Крищенко ; под ред. В. С. Зарубина, А. П. Крищенко. – Москва : Издательство МГТУ им. Н. Э. Баумана, 2019. Т. 374. С. 845–848.

[2] Coppersmith Don, Winograd Shmuel. Matrix Multiplication via Arithmetic Progressions // Journal of Symbolic Computation. 1990.

[3] Python. [Электронный ресурс]. Режим доступа: https://docs.python.org/3/ (дата обращения: 27.09.2021).

[4] Documentation for Visual Studio Code. [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs (дата обращения: 27.09.2021).

[5] time — Time access and conversions — Python 3.9.7 documentation. [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 27.09.2021).