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: «Алгоритмы умножения матриц»

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

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

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью лабораторной работы: исследовать различные алгоритмы умножения матриц.

Для её достижения были поставлены следующие задачи:

– рассмотреть алгоритмы умножения матриц: стандартный метод и алгоритм Винограда;

– 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

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} 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 кажется, 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.

2 Конструкторский раздел

2.1 Алгоритмы умножения матриц

2.1.1 Стандартный алгоритм умножения матриц

На рисунке 2.1 представлен алгоритм стандартного умножения матриц.

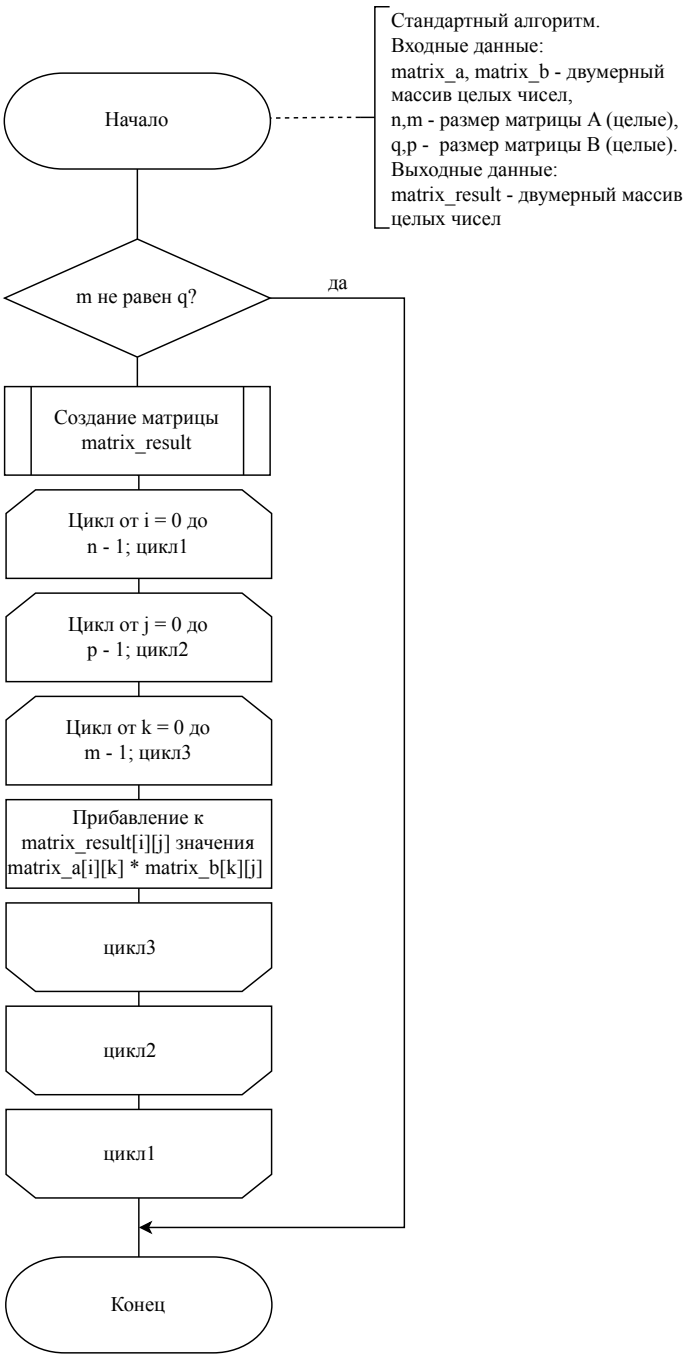

Рисунок 2.1 – Алгоритм стандартного умножения матриц

2.1.2 Алгоритм Винограда умножения матриц

На рисунках 2.2 и 2.3 представлен алгоритм Винограда умножения матриц.

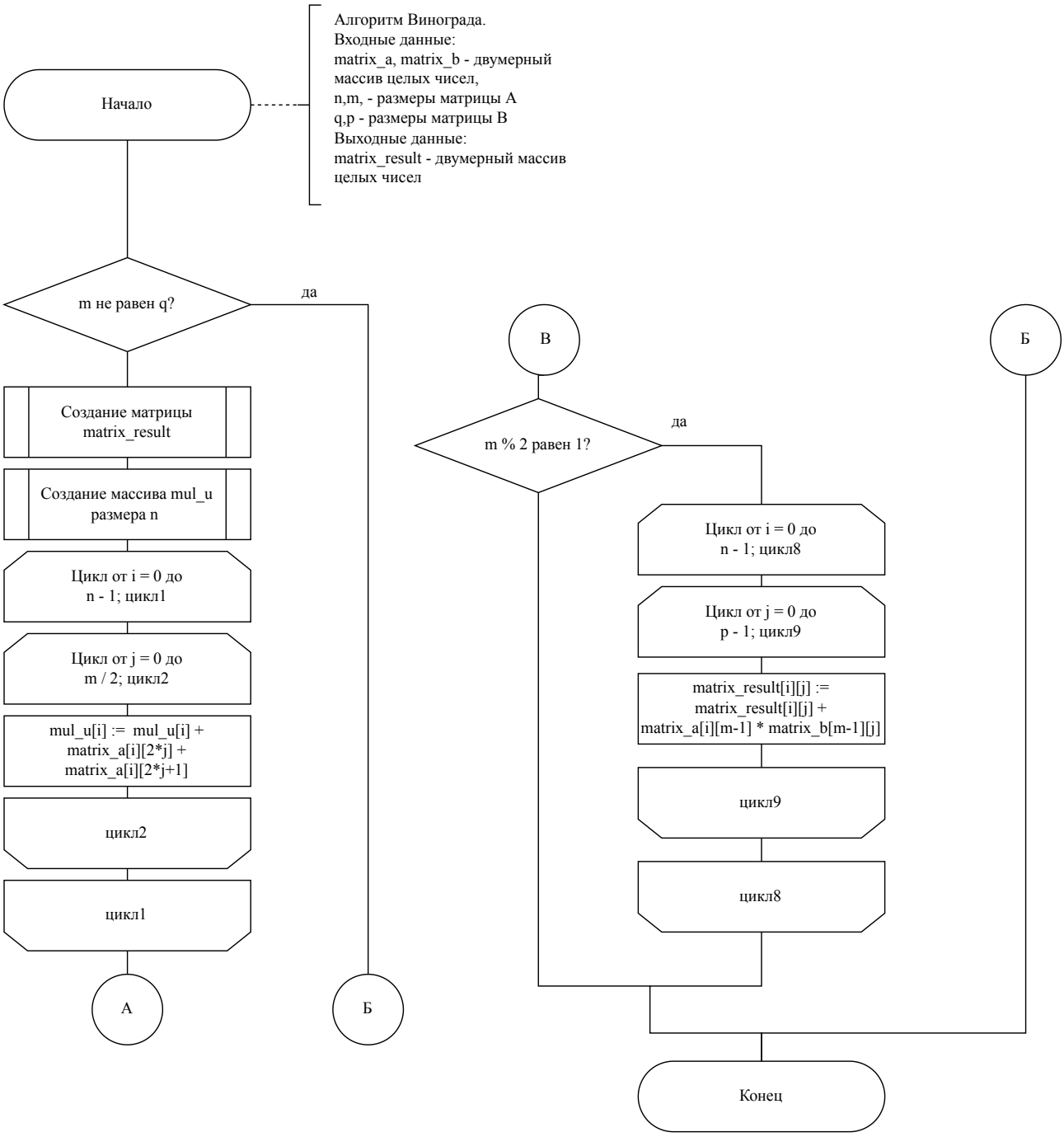

Рисунок 2.2 – Алгоритм Винограда умножения матриц. Часть 1

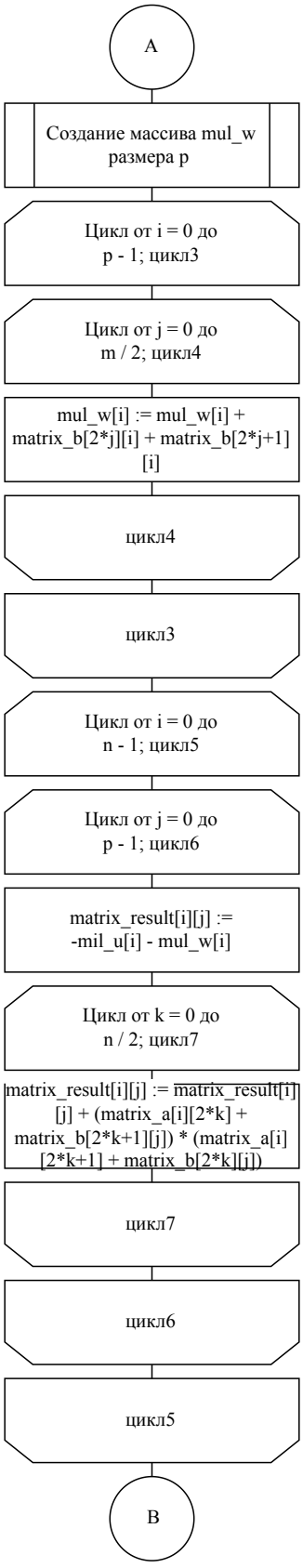

Рисунок 2.3 – Алгоритм Винограда умножения матриц. Часть 2

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 $M \times N$, размер матрицы B имеет $N \times Q$.

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):

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

Вывод

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

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

– на вход программе подаются два массива сгенерированных целых чисел;

– 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 Алгоритм стандартного умножения матриц

Реализация стандартного алгоритма умножения матриц приведена на листинге 3.1.

Листинг 3.1 – Алгоритм стандартного умножения 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 Алгоритм Винограда умножения матриц

Реализация алгоритма Винограда умножения матриц приведена на листинге 3.2.

Листинг 3.2 – Алгоритм Винограда умножения матриц

```
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 Оптимизированный алгоритм Винограда умножения матриц

Реализация оптимизированного алгоритма Винограда умножения матриц приведена на листинге 3.3.

Листинг 3.3 – Оптимизированный алгоритм Винограда умножения матриц

```
    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
```

3.4 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 стандартного алгоритма умножения матриц, алгоритма Винограда умножения матриц и оптимизированного алгоритма Винограда умножения матриц данные тесты были пройдены успешно.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– операционная система: Windows 10 Pro;

– память: 8 Гб;

– процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60 ГГц 1.80 ГГц.

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

Проведен анализ зависимости времени работы алгоритмов умножения матриц от размера исходных матриц. Рассмотрен 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 проведена работа каждого алгоритма n = 5 раз и поделена на число n. Получено среднее значение работы каждого из алгоритмов.

Выполнен 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.

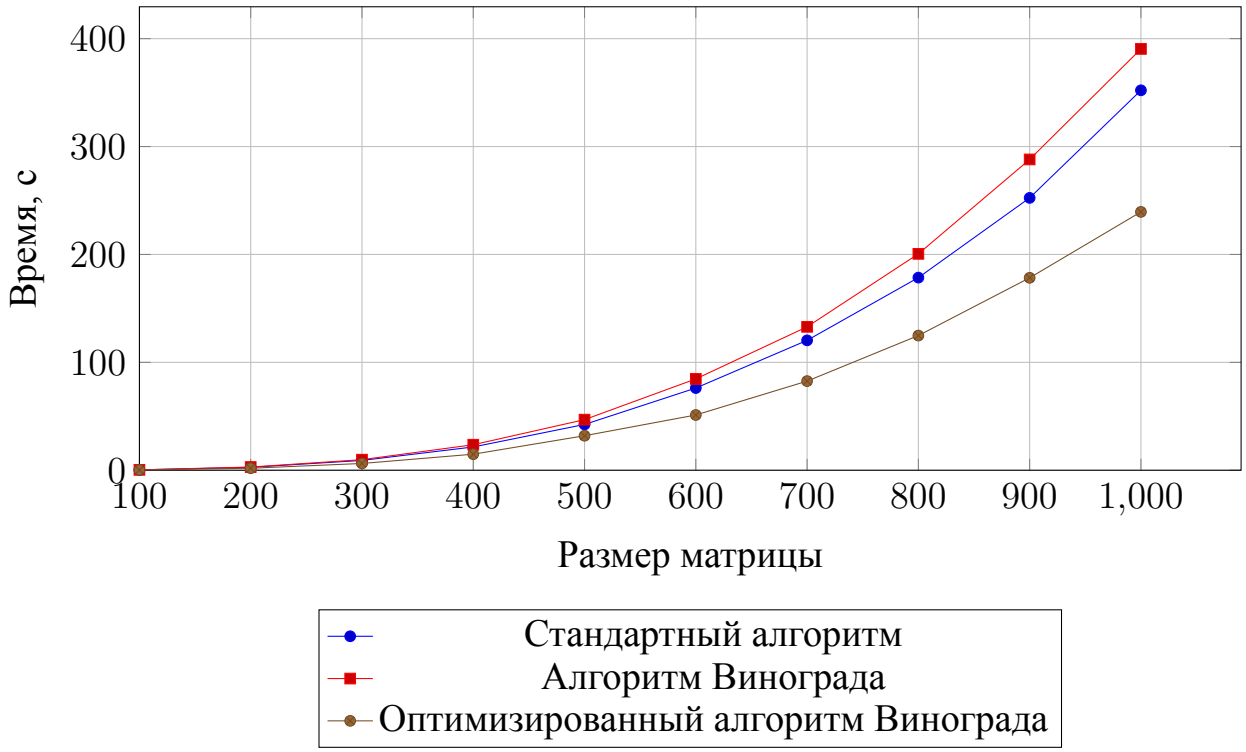

Рисунок 4.1 – Зависимости времени работы алгоритмов при чётных размерностях матриц

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}.

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

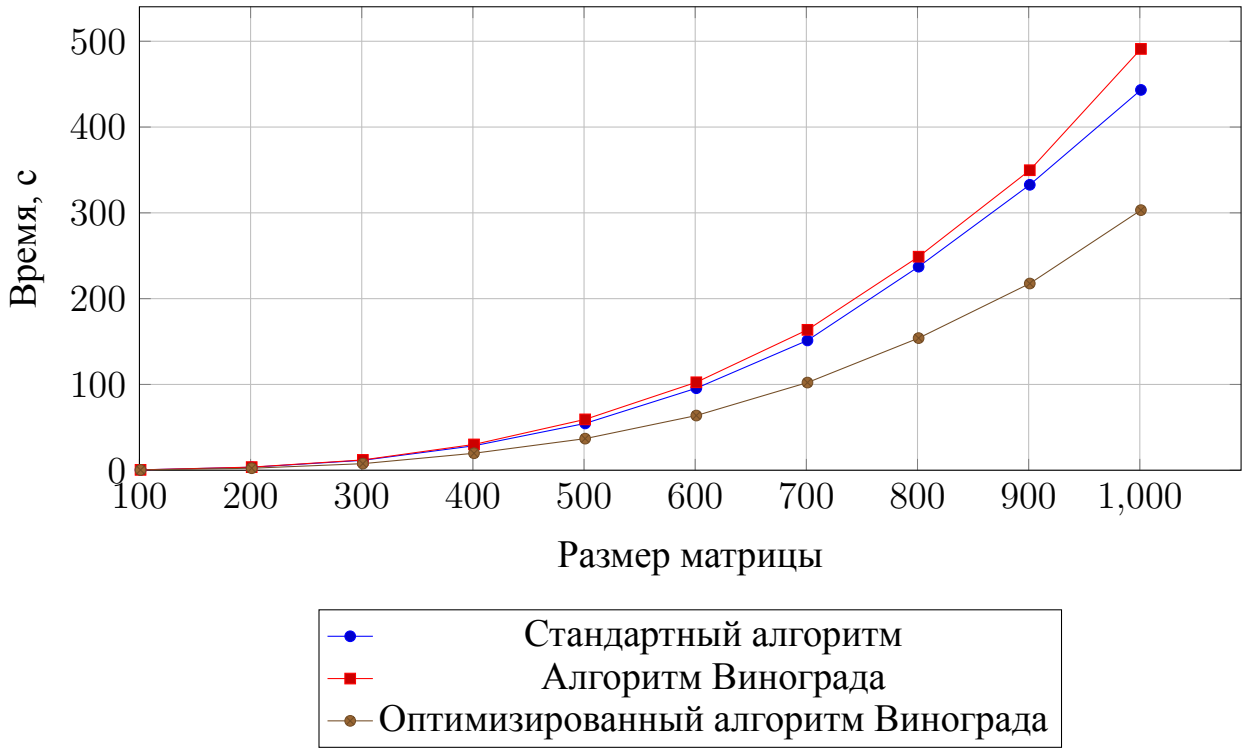

Рисунок 4.2 – График зависимости времени работы алгоритмов при нечётных размерностях матриц

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 подтверждена трудоемкость алгоритмов умножения матриц, описанная в разделах 2.3.1, 2.3.2. Время выполнения, описанное в разделе 4.2., каждого из этих методов при нечетных размерностях матриц больше времени 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 исследованы стандартный алгоритм умножения матриц, алгоритм Винограда умножения матриц. Проведена оптимизация алгоритма Винограда умножения матриц. Были выполнены описание каждого из этих алгоритмов, приведены соответствующие математические расчёты трудоемкости каждого из них. Цель работы достигнута. Поставленные задачи решены.

При анализе временных характеристик каждого из этих алгоритмов сделаны 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.

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)