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»

«Трудоемкость алгоритмов умножения матриц»

Студент Маслова Марина Дмитриевна

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Лилия Леонидовна

2021 г.

Содержание

Введение **4**

1 Аналитическая часть **5**

1.1 Стандартный алгоритм умножения матриц 5

1.2 Алгоритм умножения матриц Винограда 5

1.3 Вывод 6

2 Конструкторская часть **7**

2.1 Разработка алгоритмов 7

2.1.1 Схемы стандартного алгоритма умножения матриц и алгоритма Винограда 7

2.2 Оценка трудоемкости алгоритмов 10

2.2.1 Модель вычислений 10

2.2.2 Трудоемкость стандартного алгоритма умножения матриц 10

2.2.3 Трудоемкость алгоритма умножения матриц Винограда 11

2.3 Оптимизация алгоритма Винограда 13

2.3.1 Схема оптимизированного алгоритма Винограда 13

2.3.2 Трудоемкость оптимизированного алгоритма Винограда 13

2.4 Структура разрабатываемого ПО 17

2.5 Классы эквивалентности при тестировании 17

2.6 Вывод 18

3 Технологическая часть **19**

3.1 Требования к программному обеспечению 19

3.2 Средства реализации 19

3.3 Листинги кода 19

3.4 Описание тестирования 22

3.5 Вывод 23

4 Исследовательская часть **24**

4.1 Технические характеристики 24

4.2 Примеры работы программы 24

4.3 Результаты тестирования 25

4.4 Постановка эксперимента по замеру времени 25
4.5 Результаты эксперимента 25
4.6 Вывод 28

Заключение **29**

Список литературы **30**

Введение

Умножение матриц является основным инструментом линейной алгебры и имеет многочисленные применения в математике, физике, программировании [1]. При этом сложность стандартного алгоритма умножения матриц $N \times N$ составляет $O(N^3)$ [2], что послужило причиной разработки новых алгоритмов меньшей сложности. Одним из них является алгоритм Винограда с асимптотической сложностью $O(N^{2.3755})$ [1].

Целью данной работы является изучение алгоритмов умножения матриц: стандартного и Винограда, — а также получение навыков расчета сложности алгоритмов и их оптимизации.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- разработать каждый из алгоритмов;
- дать теоретическую оценку трудоемкости стандартного алгоритма и алгоритма Винограда;
- оптимизировать алгоритм Винограда и дать теоретическую оценку его трудоемкости;
- реализовать каждый из трех алгоритмов;
- провести тестирование реализованных алгоритмов;
- провести сравнительный анализ алгоритмов по процессорному времени работы реализации.

1 Аналитическая часть

В данном разделе представлено теоретическое описание стандартного алгоритма умножения матриц и алгоритма Винограда.

1.1 Стандартный алгоритм умножения матриц

Стандартный алгоритм умножения матриц является реализацией математического определения произведения матриц, которое формулируется следующим образом:

пусть даны матрица $A = (a_{ij})_{n \times p}$, имеющая n строк и p столбцов, и матрица $B = (b_{ij})_{p \times m}$, имеющая p строк и m столбцов, тогда матрица $C = (c_{ij})_{n \times m}$, имеющая n строк и m столбцов, где:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + ... + a_{ip}b_{pj} = \sum_{k=1}^{p} a_{ik}b_{kj}; \quad i = \overline{1,n};\ j = \overline{1,m}; \tag{1.1}$$

— называется **произведением** матриц A и B [3].

1.2 Алгоритм умножения матриц Винограда

Из формулы 1.1 видно, что каждый элемент итоговой матрицы представляет собой скалярное произведение соответсвующих строки и столбца исходных матриц, которое, в свою очередь, допускает предварительную обработку, т. е. часть вычислений можно выполнить заранее.

Пусть даны два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение представлено формулой 1.2:

$$V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4 \tag{1.2}$$

Равенство 1.2 можно представить в виде:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4 \tag{1.3}$$

Хотя в выражении 1.3 больше операций, чем в выражении 1.2: девять

сложений против трех, и шесть умножений против четырех, — последнее выражение допускает предварительную обработку, а именно операции вида $v_i v_{i+1}$ и $w_i w_{i+1}$ для $i \in \overline{0, 2..p}$ могут быть вычислены заранее. Это позволить нам для каждого элемента выполнять только операции для двух первых слагаемых в выражении 1.3, то есть два умножения и пять сложений, а также две операции сложения для учета уже вычисленных значений [4] .

1.3 Вывод

В данном разделе были рассмотрены два алгоритма умножения матриц: стандартный и Винограда. Из представленных описаний можно предъявить ряд требований для разрабатываемого программного опеспечения:

- на вход должны подаваться две матрицы;
- на выходе должна выдаваться результирующая матрица, являющаяся произведением двух исходных;
- так как произведение определено не для всех пар матриц, а только для таких, у которых в паре количество столбцов в первой матрице равно количеству строк во второй, программа должна корректно реагировать на недопустимые входные размеры матриц.

2 Конструкторская часть

В данном разделе разрабатываются алгоритмы умножения матриц, структура программы и способы её тестирования, также проводится оценка трудоемкости каждого из алгоритмов и оптимизация алгоритма Винограда.

2.1 Разработка алгоритмов

В данном подразделе приводятся схемы разработанных алгоритмов, оценка их трудоемкости, на основе которой производится оптимизация алгоритма Винограда с последующим описаним алгоритма в виде схемы.

2.1.1 Схемы стандартного алгоритма умножения матриц и алгоритма Винограда

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

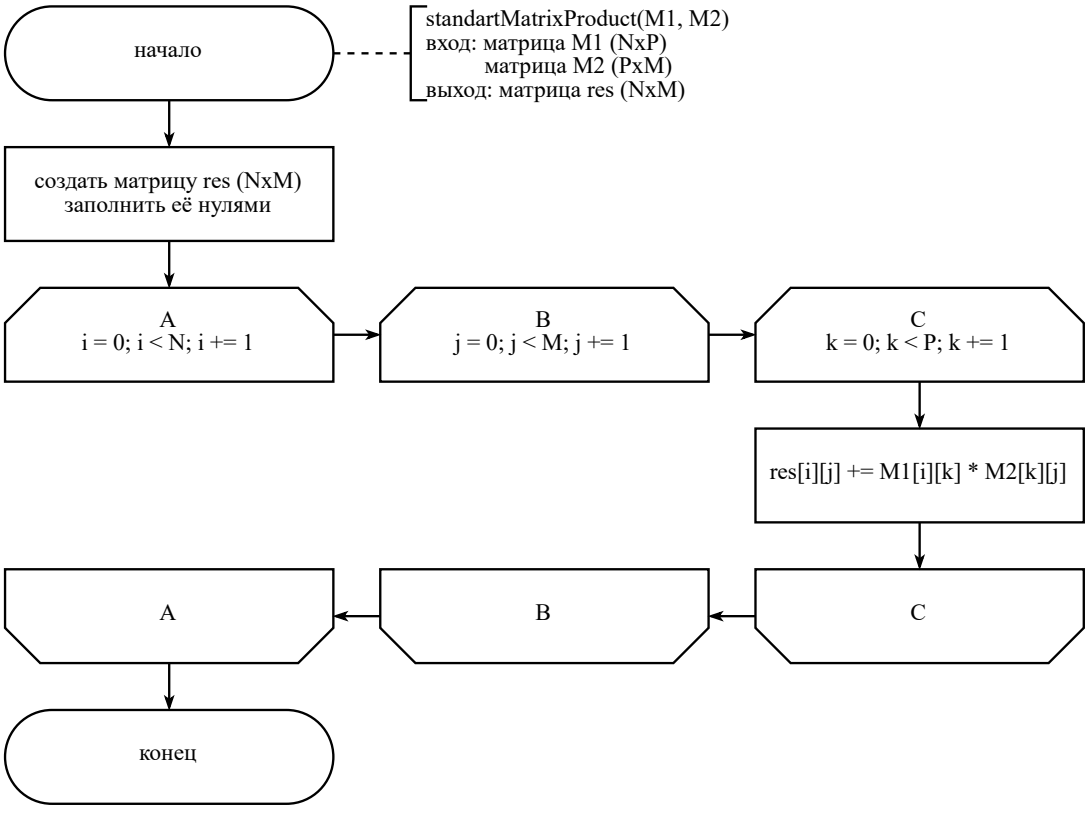

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

На рисунках 2.2-2.4 приведена схема алгоритма Винограда.

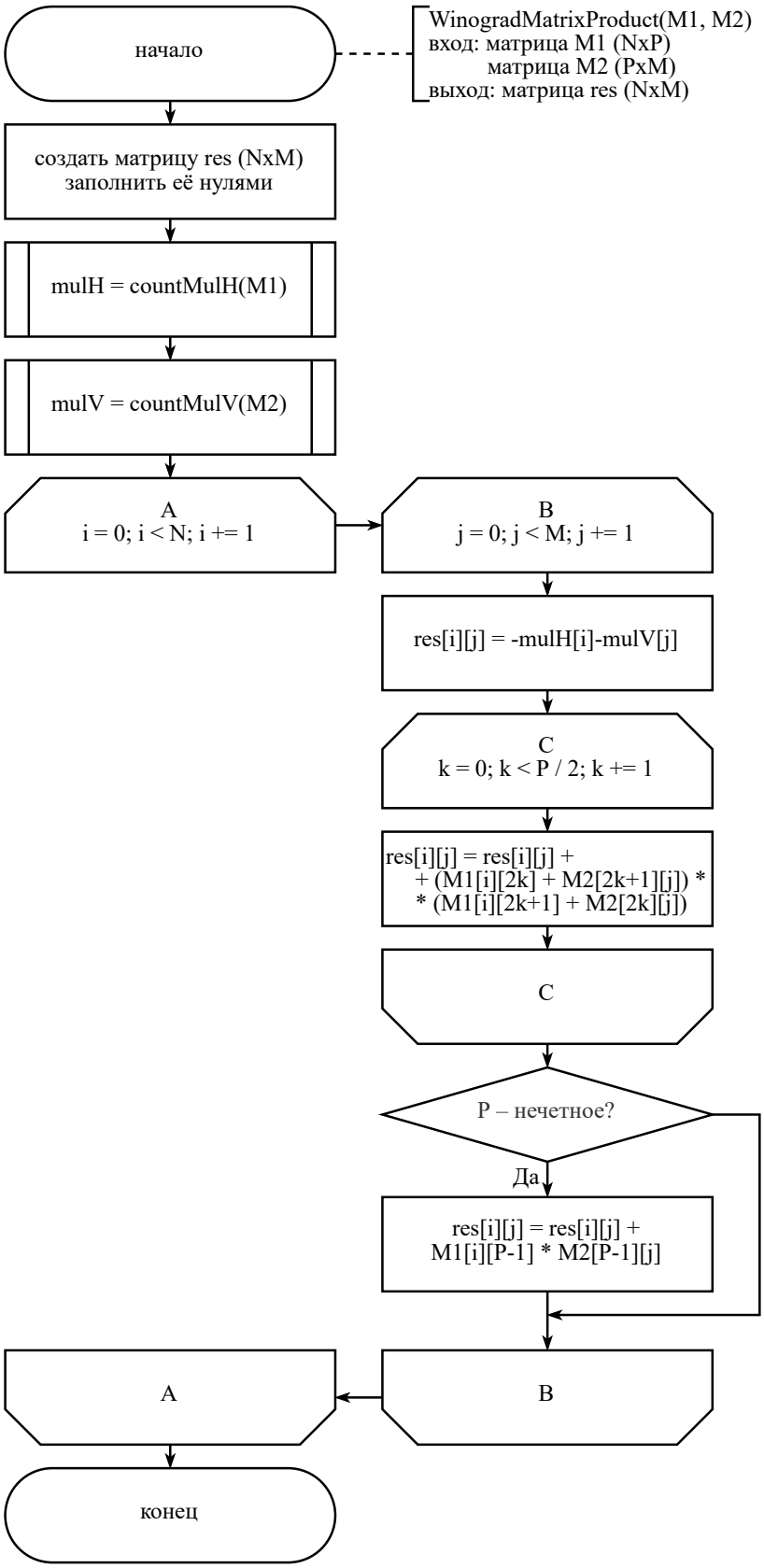

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц

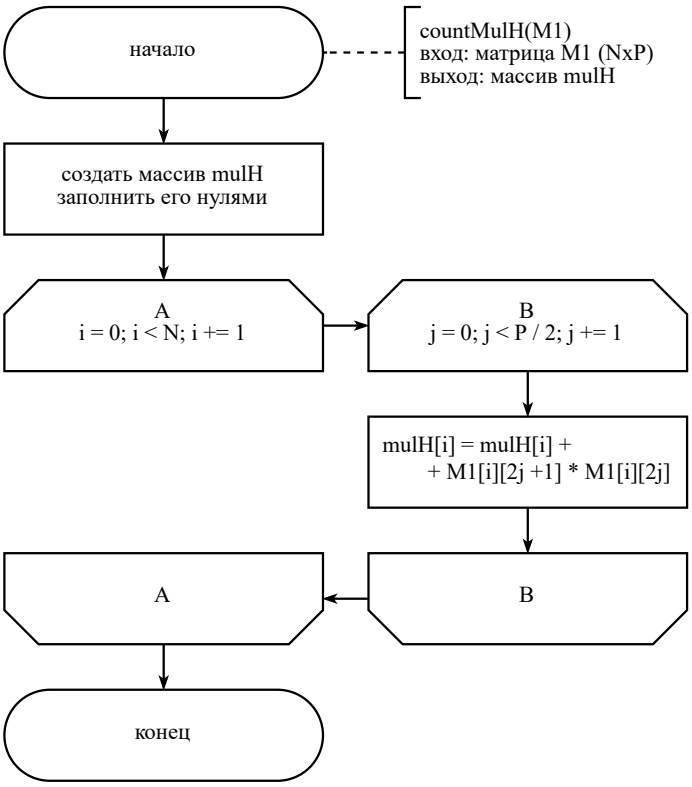

Рисунок 2.3 – Схема алгоритма вычисления сумм произведений пар соседних элементов строк матрицы

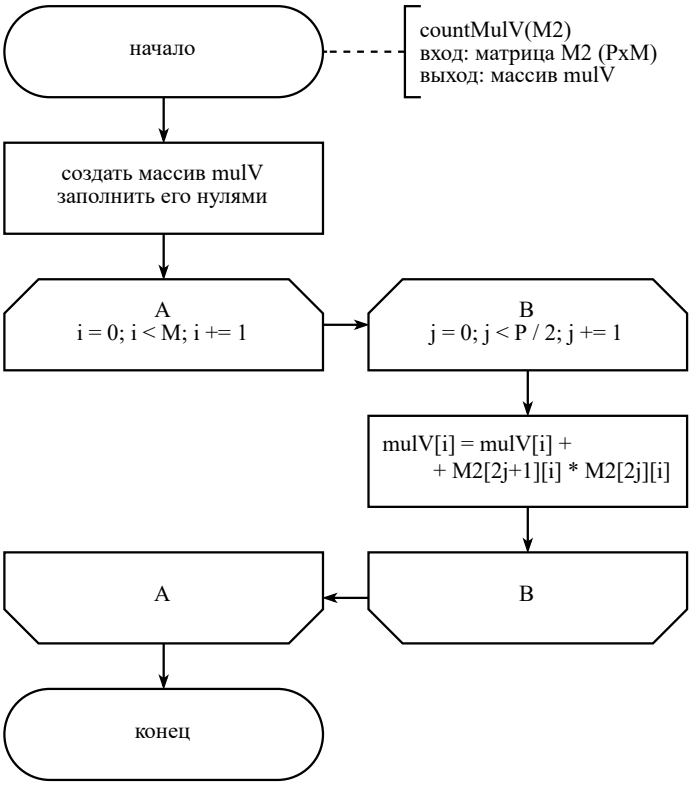

Рисунок 2.4 – Схема алгоритма вычисления сумм произведений пар соседних элементов столбцов матрицы

2.2 Оценка трудоемкости алгоритмов

В данном подразделе производится оценка трудоемкости каждого из алгоритмов.

2.2.1 Модель вычислений

Введем модель вычислений для оценки трудоемкости алгоритмов:

– операции из списка 2.1 имеют трудоемкость 2;

$$*,\ /,\ //,\ \%,\ *=,\ /=,\ //= \tag{2.1}$$

– операции из списка 2.2 имеют трудоемкость 1;

$$=,\ +,\ -,\ +=,\ -=,\ <,\ >,\ ==,\ !=,\ >=,\ <=,\ [],\ <<,\ >>,\ ++,\ -- \tag{2.2}$$

– трудоемкость оператора выбора `if` условие `then` A `else` B рассчитывается по формуле 2.3:

$$f_{if} = f + \begin{cases} f_A, & \text{если условие выполняется;} \\ f_B, & \text{иначе} \end{cases} \tag{2.3}$$

– трудоемкость оператора цикла рассчитывется по формуле 2.4:

$$f_{\text{for}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

– трудоемкость вызова функции равна 0.

Так как во всех следующих алгоритмах присутствует инциализация матрицы, которая не является самым трудоемким действием, при оценке трудоемкости она не учитывается.

2.2.2 Трудоемкость стандартного алгоритма умножения матриц

Трудоемкость стандартного алгоритма умножения матриц считается следующим образом (циклы обозначаются так же, как и на схеме 2.1):

– трудоемкость цикла A вычисляется по формуле 2.5;

$$f_A = 1 + 1 + N(f_B + 1 + 1) = 2 + N(2 + f_B) \tag{2.5}$$

– трудоемкость цикла B вычисляется по формуле 2.6;

$$f_B = 1 + 1 + M(f_C + 1 + 1) = 2 + M(2 + f_C) \tag{2.6}$$

– трудоемкость цикла C вычисляется по формуле 2.7;

$$f_C = 1 + 1 + P(9 + 1 + 1) = 2 + 11P \tag{2.7}$$

Кроме циклов в стандартном алгоритме нет действий, поэтому трудоемкость алгоритма равна трудоемкости внешнего цикла A и равна 2.8:

$$f_{standart} = 2 + N(2 + 2 + M(2 + 2 + 11P)) = 11NMP + 4MN + 4N + 2 \tag{2.8}$$

2.2.3 Трудоемкость алгоритма умножения матриц Винограда

Трудоемкость алгоритма умножения матриц Винограда считается следующим образом:

– трудоемкость инициализации массивов mulH и mulV равна 2.9;

$$f_{init} = N + M \tag{2.9}$$

– трудоемкость заполнения массива mulH вычисляется по формуле 2.10 (обоначение циклов как на схеме 2.3);

$$\begin{aligned} f_{mulH} = f_A = 2 + N(2 + f_B) = \\ = 2 + N(2 + 1 + 3 + \frac{P}{2}(3 + 1 + 15)) = \\ = 2 + N(6 + \frac{19P}{2}) = 9.5PN + 6N + 2 \end{aligned} \tag{2.10}$$

– трудоемкость заполнения массива mulV вычисляется по формуле 2.11

(обоначение циклов как на схеме 2.4);

$$f_{mulV} = f_A = 2 + M(2 + f_B) = \\ = 2 + M(2 + 1 + 3 + \frac{P}{2}(1 + 3 + 15)) = \\ = 2 + M(6 + \frac{19P}{2}) = 9.5MP + 6M + 2 \quad (2.11)$$

– трудоемкость основной части алгоритма для лучшего случая, то есть когда P – четное, равна 2.12 (обознаяение циклов как на схеме 2.2);

$$f^{\wedge}_{main} = f_A = 2 + N(2 + f_B) = \\ = 2 + N(2 + 2 + M(2 + 7 + f_c + 3)) = \\ = 2 + N(4 + M(12 + 4 + \frac{P}{2}(4 + 28))) = \\ = 16NMP + 16NM + 4N + 2 \quad (2.12)$$

– трудоемкость основной части алгоритма для худшего случая, то есть когда P – нечетное, равна 2.13 (обознаяение циклов как на схеме 2.2);

$$f^{\vee}_{main} = f_A = 2 + N(2 + f_B) = \\ = 2 + N(2 + 2 + M(2 + 7 + f_c + 3 + 14)) = \\ = 2 + N(4 + M(26 + 4 + \frac{P}{2}(4 + 28))) = \\ = 16NMP + 30NM + 4N + 2 \quad (2.13)$$

Таким образом, трудоемкость алгоритма Винограда равна 2.14 для лучшего случая, 2.15 – для худшего.

$$f^{\wedge}_{Winograd} = 16NMP + 16NM + 4N + 2 + \\ + 9.5MP + 6M + 2 + 9.5PN + 6N + 2 + N + M = \\ = 16NMP + 16NM + 9.5MP + 9.5PN + 11N + 7M + 6 \quad (2.14)$$

$$f^{\vee}_{Winograd} = 16NMP + 30NM + 4N + 2+ \\ + 9.5MP + 6M + 2 + 9.5PN + 6N + 2 + N + M = \\ = 16NMP + 30NM + 9.5MP + 9.5PN + 11N + 7M + 6 \quad (2.15)$$

2.3 Оптимизация алгоритма Винограда

Как видно по вычисленным трудоемкостям алгоритмов (формулы 2.8 и 2.14), трудоемкость алгоритма умножения матриц не уменьшилась, а даже увеличилась, связано это с тем, что с вынесением предварительных вычислений возросло количество таких дорогостоящих операций, как умножение и деление. Чтобы получить выигрыш от предварительных вычислений, необходимо оптимизировать алгоритм, для чего используем следующие типы оптимизации:

- большинство производимых умножений и делений происходит на 2, поэтому заменим их на побитовый сдвиг влево и вправо соответственно;
- заменим выражения вида `x = x + y` на выражения вида `x += y`, а выражения вида `x = x - y` на выражения вида `x -= y`, избавившись худшем случае от одной операции при каждой замене;
- операции умножения можно убрать, сделав в циклах по `k` шаг 2.

2.3.1 Схема оптимизированного алгоритма Винограда

Применив типы оптимизации, описанные выше, получим схему оптимизированного алгоритма Винограда, представленную на рисунках 2.5 - 2.7.

2.3.2 Трудоемкость оптимизированного алгоритма Винограда

Рассчитаем трудоемкость оптимизированного алгоритма Винограда аналогично неоптимизированному случаю:

- трудоемкость инициализации массивов mulH и mulV равна 2.16;

$$f_{init} = N + M \quad (2.16)$$

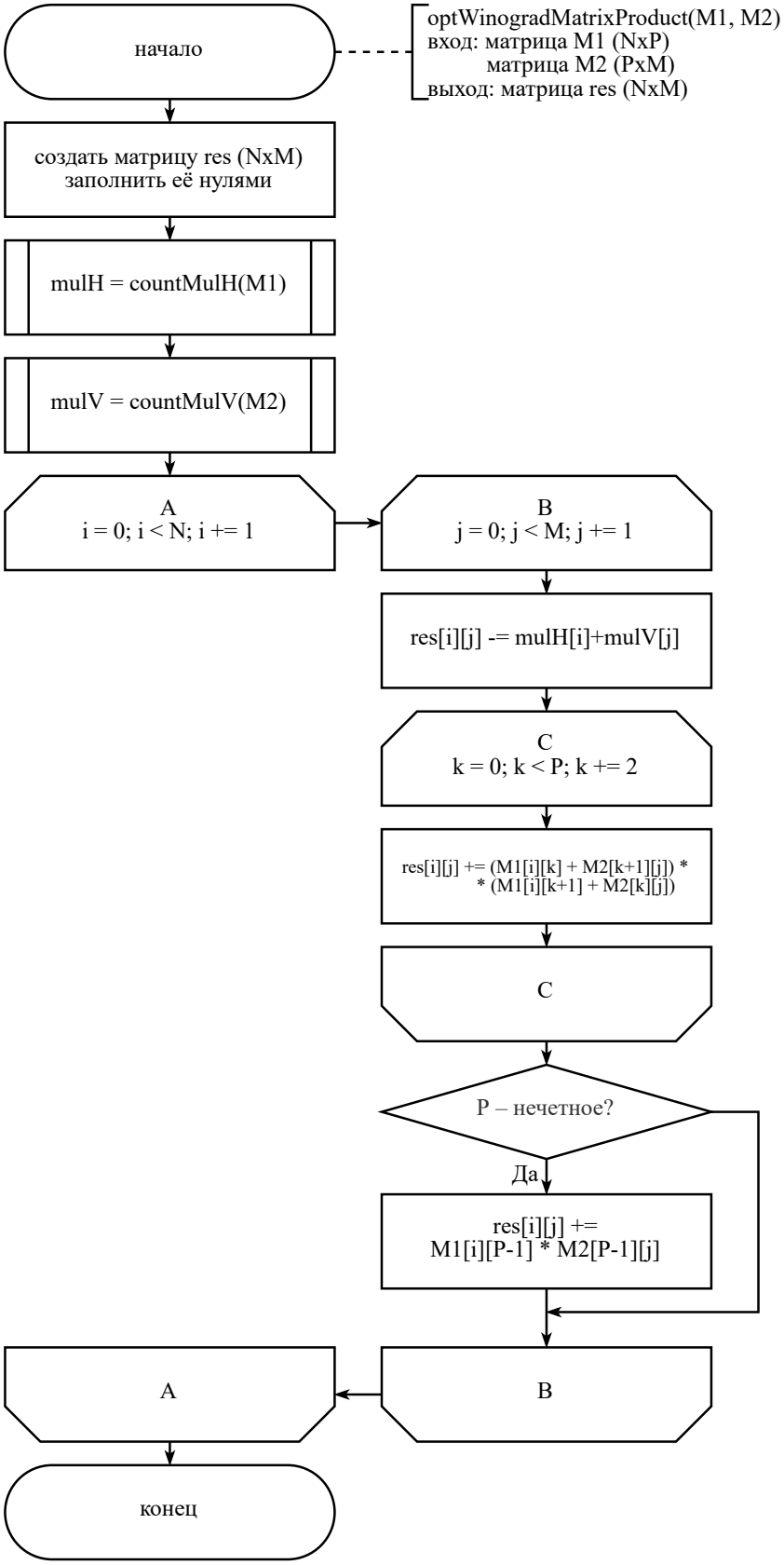

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда умножения матриц

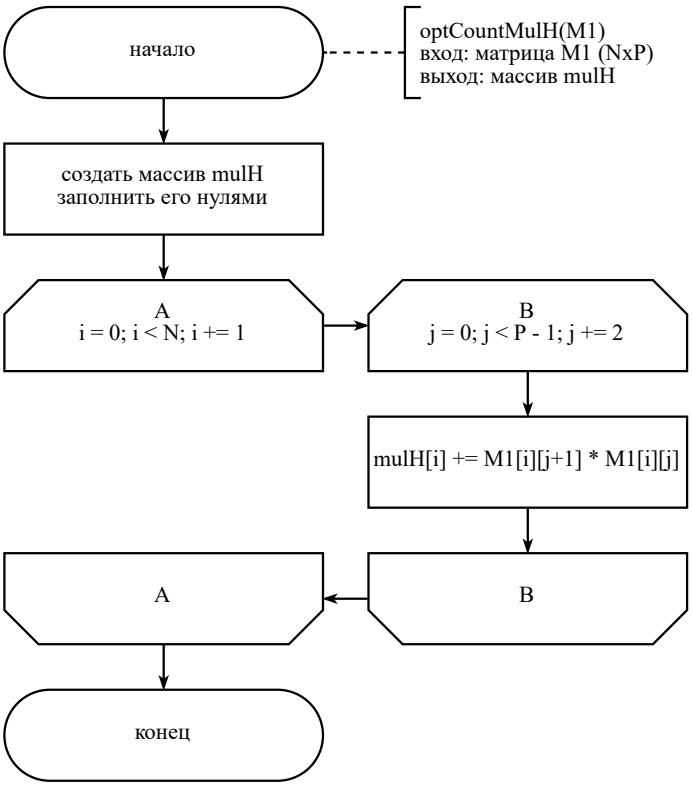

Рисунок 2.6 – Схема оптимизированного алгоритма вычисления сумм произведений пар соседних элементов строк матрицы

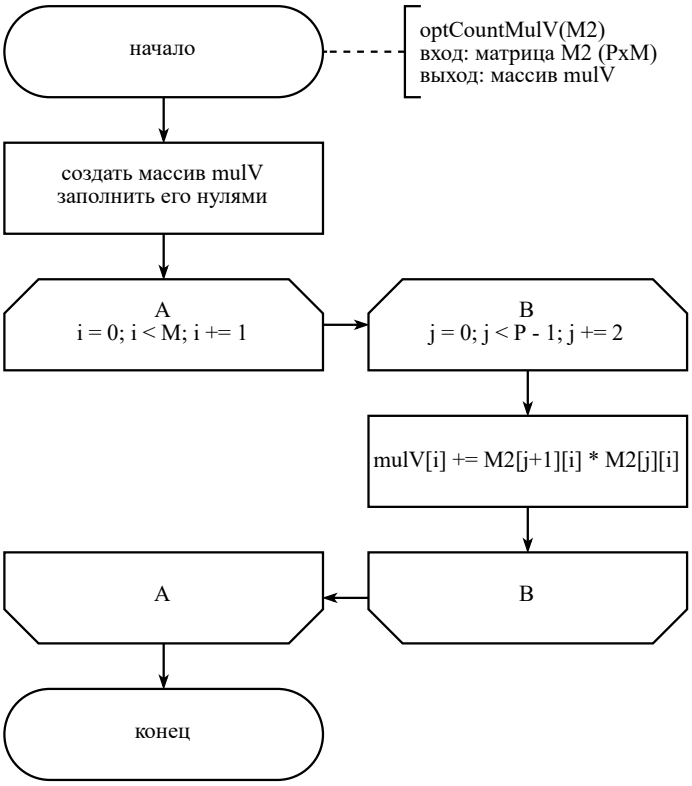

Рисунок 2.7 – Схема оптимизированного алгоритма вычисления сумм произведений пар соседних элементов столбцов матрицы

– трудоемкость заполнения массива mulH вычисляется по формуле 2.17 (обоначение циклов как на схеме 2.6);

$$f_{optMulH} = f_A = 2 + N(2 + f_B) = \\ = 2 + N(2 + 1 + 2 + \frac{P}{2}(3 + 9)) = \\ = 2 + N(5 + \frac{12P}{2}) = 6PN + 5N + 2 \quad (2.17)$$

– трудоемкость заполнения массива mulV вычисляется по формуле 2.18 (обоначение циклов как на схеме 2.7);

$$f_{optMulV} = f_A = 2 + M(2 + f_B) = \\ = 2 + M(2 + 1 + 2 + \frac{P}{2}(3 + 9)) = \\ = 2 + M(5 + \frac{12P}{2}) = 6MP + 5M + 2 \quad (2.18)$$

– трудоемкость основной части алгоритма для лучшего случая, то есть когда P – четное, равна 2.19 (обознаяение циклов как на схеме 2.5);

$$f^{\wedge}_{optMain} = f_A = 2 + N(2 + f_B) = \\ = 2 + N(2 + 2 + M(2 + 6 + f_c + 3)) = \\ = 2 + N(4 + M(11 + 3 + \frac{P}{2}(3 + 17))) = \\ = 10NMP + 14NM + 4N + 2 \quad (2.19)$$

– трудоемкость основной части алгоритма для худшего случая, то есть когда P – нечетное, равна 2.20 (обознаяение циклов как на схеме 2.5);

$$f^{\vee}_{optMain} = f_A = 2 + N(2 + f_B) = \\ = 2 + N(2 + 2 + M(2 + 6 + f_c + 3 + 11)) = \\ = 2 + N(4 + M(22 + 3 + \frac{P}{2}(3 + 17))) = \\ = 10NMP + 25NM + 4N + 2 \quad (2.20)$$

Таким образом, трудоемкость оптимизированного алгоритма Винограда

равна 2.21 для лучшего случая, 2.22 – для худшего.

$$f^{\wedge}_{optWinograd} = 10NMP + 14NM + 4N + 2+ \\ + 6MP + 5M + 2 + 6PN + 5N + 2 + N + M = \\ = 10NMP + 14NM + 6MP + 6PN + 10N + 6M + 6 \quad (2.21)$$

$$f^{\vee}_{optWinograd} = 10NMP + 25NM + 4N + 2+ \\ + 6MP + 5M + 2 + 6PN + 5N + 2 + N + M = \\ = 10NMP + 25NM + 6MP + 6PN + 10N + 6M + 6 \quad (2.22)$$

2.4 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для какждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

2.5 Классы эквивалентности при тестировании

Для тестирования программного обеспечения во множестве тестов будут выделены следующие классы эквивалентности:

- количество столбцов первой матрицы не равно количеству строк второй;
- квадратные матрицы;
- произвольные матрицы с определенной операцией умножения;
- умножение матрицы на обратную;
- умножение на нулевую матрицу;
- умножение на единичную матрицу.

2.6 Вывод

В данном разделе были разработаны алгоритмы умножения матриц: стандартный и Винограда, – также была произведена оценка трудоемкостей алгоритмов и оптимизация алгоритма Винограда. Для дальшейшей проверки правильности работы программы были выделены классы эквавалентности тестов.

3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие возможности:

- выбор режима работы: для единичного эксперимента и для массовых экспериментов;
- в режиме единичного эксперимента ввод размеров и содержимого каждой матрицы и вывод полученного разными алгоритмами произведений;
- в режиме массовых экспериментов измерение времени при различных размерах матриц и построение графиков по полученным данным.

3.2 Средства реализации

Для реализации данной лабораторной работы выбран интерпретируемый язык программирования высокого уровня Python[5], так как он позволяет реализовывать сложные задачи за кратчайшие сроки за счет простоты синтаксиса и наличия большого количества подключаемых библиотек.

В качестве среды разработки выбран текстовый редактор Vim[6] с установленными плагинами автодополнения и поиска ошибок в процессе написания, так как он реализует быстрое перемещение по тексту программы и простое взаимодействие с командной строкой.

Замеры времени проводились при помощи функции process_time_ns из библиотеки time[7].

3.3 Листинги кода

В данном подразделе представлены листинги кода ранее описанных алгоритмов:

- стандартный алгоритм умножения матриц (листинг 3.1);

– алгоритм Винограда (листинги 3.2-3.4);
– оптимизированный алгоритм Винограда (листинги 3.5-3.7).

Листинг 3.1 – Реализация стандартного алгоритма умножения матриц

```
def standartMatrProd(m1, m2):
    N = len(m1)
    P = len(m2)
    M = len(m2[0])

    res = [[0 for j in range(N)] for i in range(M)]

    for i in range(N):
        for j in range(M):
            for k in range(P):
                res[i][j] += m1[i][k] * m2[k][j]

    return res
```

Листинг 3.2 – Реализация подпрограммы алгоритма Винограда для расчета значений массива mulH

```
def countMulH(matrix):
    N = len(matrix)
    P = len(matrix[0])

    mulH = [0] * N

    for i in range(N):
        for j in range(P // 2):
            mulH[i] = mulH[i] + matrix[i][2 * j + 1] * matrix[i][2 * j]

    return mulH
```

Листинг 3.3 – Реализация подпрограммы алгоритма Винограда для расчета значений массива mulV

```
def countMulV(matrix):
    P = len(matrix)
    M = len(matrix[0])

    mulV = [0] * M

    for i in range(M):
        for j in range(P // 2):
            mulV[i] = mulV[i] + matrix[2 * j + 1][i] * matrix[2 * j][i]

    return mulV
```

Листинг 3.4 – Реализация алгоритма Винограда

```
def WinogradMatrProd(m1, m2):
    N = len(m1)
    P = len(m2)
    M = len(m2[0])

    res = [[0 for j in range(N)] for i in range(M)]

    mulH = countMulH(m1)
    mulV = countMulV(m2)

    for i in range(N):
        for j in range(M):
            res[i][j] = - mulH[i] - mulV[j]

            for k in range(P // 2):
                res[i][j] = (res[i][j]
                             + (m1[i][2 * k] + m2[2 * k + 1][j])
                             * (m1[i][2 * k + 1] + m2[2 * k][j]))

            if P % 2:
                res[i][j] = res[i][j] + m1[i][P - 1] * m2[P - 1][j]

    return res
```

Листинг 3.5 – Реализация подпрограммы оптимизированного алгоритма Винограда для расчета значений массива mulH

```
def optCountMulH(matrix):
    N = len(matrix)
    P = len(matrix[0])

    mulH = [0] * N

    for i in range(N):
        for j in range(0, P - 1, 2):
            mulH[i] += matrix[i][j + 1] * matrix[i][j]

    return mulH
```

Листинг 3.6 – Реализация подпрограммы оптимизированного алгоритма Винограда для расчета значений массива mulH

```
def optCountMulV(matrix):
    P = len(matrix)
    M = len(matrix[0])

    mulV = [0] * M

    for i in range(M):
        for j in range(0, P - 1, 2):
            mulV[j] += matrix[j + 1][i] * matrix[j][i]

    return mulV
```

Листинг 3.7 – Реализация оптимизированного алгоритма Винограда

```
def optWinogradMatrProd(m1, m2):
    N = len(m1)
    P = len(m2)
    M = len(m2[0])

    res = [[0 for j in range(N)] for i in range(M)]

    mulH = countMulH(m1)
    mulV = countMulV(m2)

    for i in range(N):
        for j in range(M):
            res[i][j] -= mulH[i] + mulV[j]

            for k in range(0, P - 1, 2):
                res[i][j] += (
                        (m1[i][k] + m2[k + 1][j])
                        * (m1[i][k + 1] + m2[k][j])
                        )

            if P % 2:
                res[i][j] += m1[i][P - 1] * m2[P - 1][j]

    return res
```

3.4 Описание тестирования

В таблице 3.1 приведены функциональные тесты для алгоритмов умножения матриц.

Таблица 3.1 – Функциональные тесты

Матрица 1	**Матрица 2**	**Ожидаемый результат**
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	Не могут быть перемножены
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & -2 & 3 \\ 1 & 2 & -3 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 0 \\ 4 & 4 & 0 \\ 4 & 4 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & -3 \\ -2 & -2 \\ -3 & -1 \end{pmatrix}$	$\begin{pmatrix} -14 & -10 \\ -14 & -10 \end{pmatrix}$
$\begin{pmatrix} 3 & -5 \\ 1 & -2 \end{pmatrix}$	$\begin{pmatrix} 2 & -5 \\ 1 & -3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$

3.5 Вывод

В данном разделе были реализованы алгоритмы умножения матриц: стандартный, Винограда и оптимизированный Винограда. Также были написаны тесты для каждого класса эквивалентности, описанного в конструкторском разделе.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

– Операционная система: Manjaro [8] Linux x86_64.
– Память: 8 GiB.
– Процессор: Intel® Core™ i5-8265U[9].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Примеры работы программы

В данном подразделе представлены примеры работы программы. На рисунке 4.1 представлен ввод данных. На рисунке 4.2 представлены результаты работы программы на введенных данных.

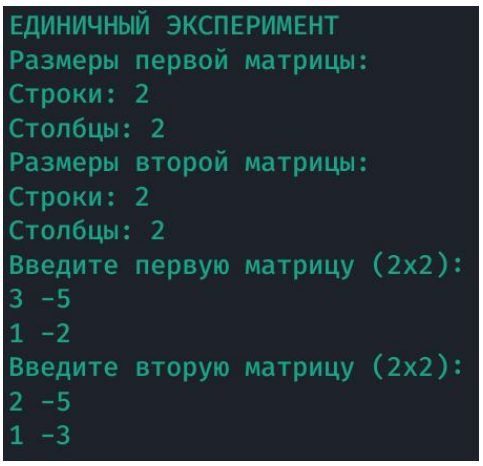

Рисунок 4.1 – Пример ввода данных

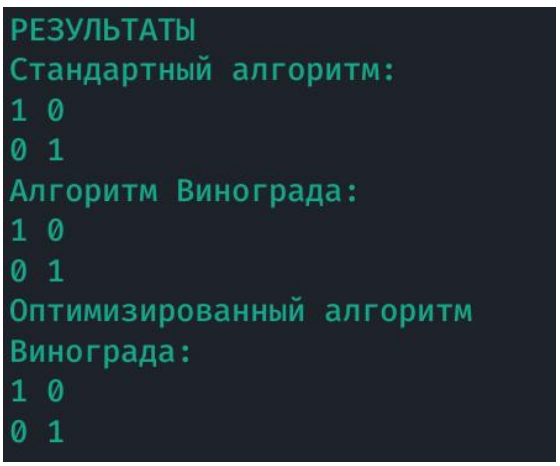

Рисунок 4.2 – Пример результата работы программы

4.3 Результаты тестирования

Программа была протестирования на входных данных, приведенных в таблице 3.1. Полученные результаты работы программы совпали с ожидаемыми результатами.

4.4 Постановка эксперимента по замеру времени

Для оценки времени работы релизаций алгоритмов умножения матриц был проведен эксперимент, в котором определялось влияние размеров матриц на время работы каждого из алгоритмов. Тестирование проводилось на квадратных матрицах размерами от 10 до 200 с шагом 10 в начале последовательности и шагом 20 в конце для исключения ситуации неопределенно долгого ожидания завершения работы программы. Так как от запуска к запуску процессорное время, затрачиваемое на выполнение агоритма менялось в определенном промежутке времени, необходимо было усреднить вычисляемые значения. Для этого каждый алгоритм на каждом значении размера запускался 10 раз, и для полученных 10 значений определялось среднее арифметическое, которое и заносилось в таблицу результатов.

Так как в алгоритме Винограда есть зависимость трудоемкости от четности размера, тестирование также было проведено на последовательности значений размеров, полученной из выше описанной прибавлением к каждому элементу единицы.

Результаты эксперимента были представлены в виде таблиц и графиков, приведенных в следующем подразделе.

4.5 Результаты эксперимента

В таблицах 4.1, 4.2 представлены результаты измерения времени работы алгоритмов на четных и нечетных размерах матриц соответственно. На основе табличных данных построены графики зависимости времени работы каждого алгоритма от размеров матриц (рисунки 4.3, 4.4).

Таблица 4.1 – Время работы алгоритмов умножения матриц на четных размерах матриц

Размеры	**Стандартный**	**Винограда**	**Оптимизированный Винограда**
10x10	539833	384197	340553
20x20	2039553	2377819	1931391
30x30	6263790	7074519	5965006
40x40	14973350	19775718	17716510
50x50	32670128	31954287	27434800
70x70	74718360	85367278	72728343
90x90	164385160	180547314	153299091
110x110	300695477	329993366	281507369
130x130	496679269	526456683	454239335
150x150	712216515	760923701	646114517
170x170	998066313	1101018878	917697414
200x200	1628347463	1770501308	1508471574

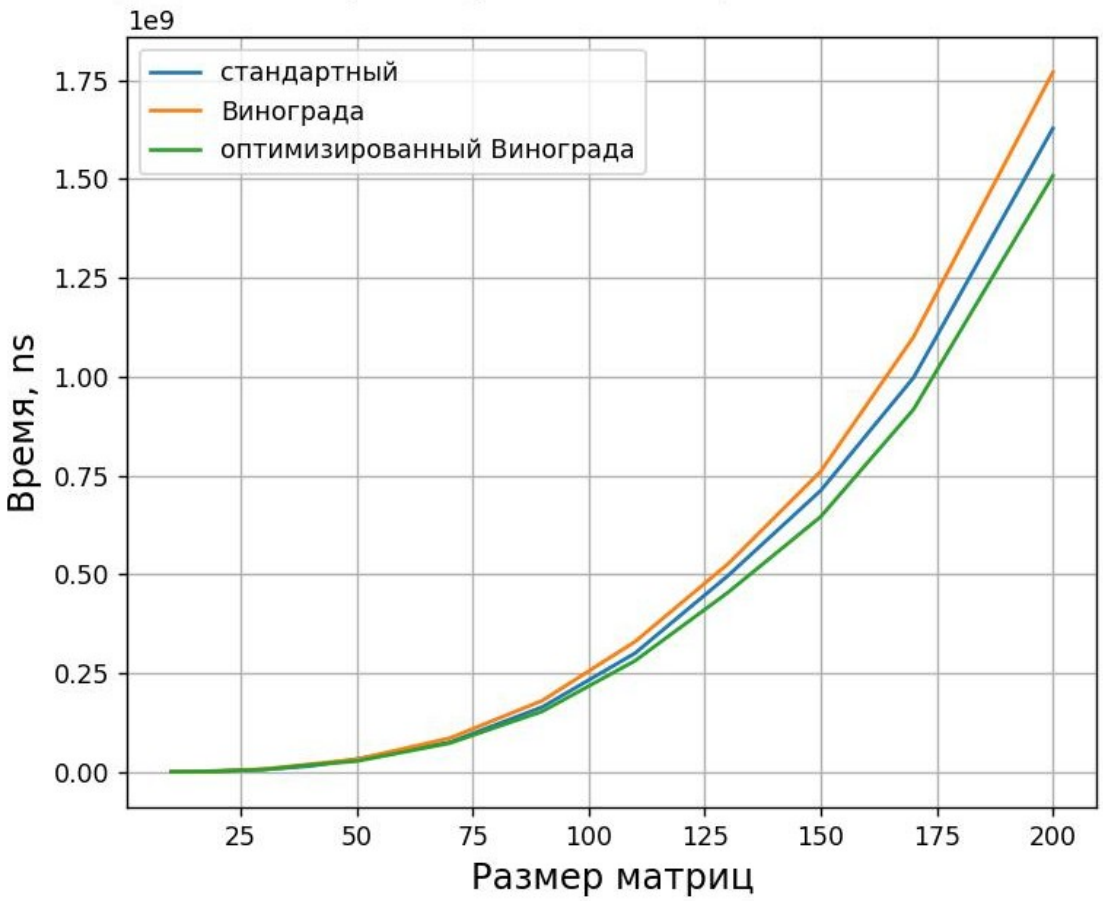

Рисунок 4.3 – Графики зависимости времени выполнения алгоритмов умножения матриц от четных размеров матриц

Таблица 4.2 – Время работы алгоритмов умножения матриц на нечетных размерах матриц

Размеры	Стандартный	Винограда	Оптимизированный Винограда
11x11	297253	382616	351488
21x21	2007013	2481616	2176297
31x31	6786029	7847581	6920048
41x41	16144894	18423024	15848454
51x51	30601890	34622928	29476640
71x71	78644866	90692956	77208153
91x91	171470377	190576740	160945628
111x111	311257471	342684786	297150124
131x131	503892910	531306715	457998213
151x151	735751274	786882716	674992111
171x171	1023125471	1122404305	958485541
201x201	1695932225	1832543500	1526100349

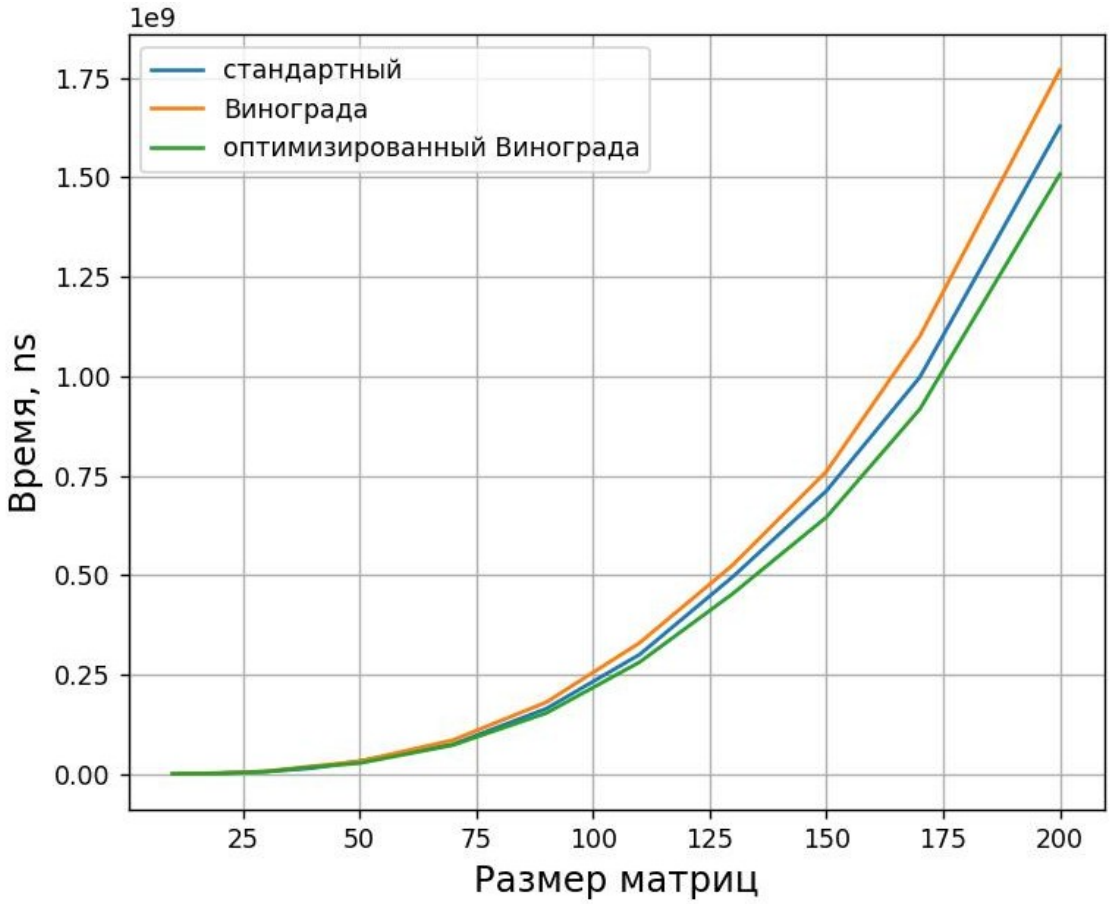

Рисунок 4.4 – Графики зависимости времени выполнения алгоритмов умножения матриц от нечетных размеров матриц

4.6 Вывод

По результатам эксперимента можно сделать следующие выводы:

- прямая реализация алгоритма Винограда без какой-либо опитимизации работает в 1.1 раз медленнее стандартного алгоритма;
- оптимизированная версия Винограда работает в 1.2 раз быстрее стандартного алгоритма;
- при этом оптимизированный алгоритм Винограда работает в 1.05 раза быстрее на четных размерах матриц, чем на нечетных.

Таким образом, для вычислений произведений матриц при их размерах до 200 следует при необходимости меньшего времени вычислений надо использовать оптимизированный алгоритм Винограда, при этом предпочтительны четные размеры матриц, хотя и при нечетных значений оптимизированный алгоритм Винограда дает выигрыш по времени по сравнению со стандартным.

Заключение

В ходе выполения лабораторной работы:

- были описаны алгоритмы умножения матриц: стандартный и Винограда;
- была произведена оценка трудоемкости каждого из алгоритов;
- по схемам алгоритмов и произведенной оценки был сделан вывод о необходимости оптимизации алгоритма Винограда;
- с помощью простый методов оптимизации алгоритмов был получен алгоритм Винограда с меньшей трудоемкостью;
- был реализован каждый из описанных алгоритмов в том числе оптимизированный алгоритм Винограда;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов.

Список литературы

[1] Анисимов Н. С. Строганов Ю. В. Реализация алгоритма умножения матриц по Винограду на языке Haskell // Новые информационные технологии в автоматизированных системах. 2018. № 21. С. 390–395. Режим доступа: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell (дата обращения: 30.10.2021).

[2] Späth Till. Searching for fast matrix multiplication algorithms. Ph.D. thesis. 2019. 10.

[3] Шихобалов Л. С. Матрицы и определители : учеб. пособие. СПб.: СПбГУ, 2015. с. 55.

[4] Макконелл Дж. Основы современных алгоритмов. 2-е доп. изд. М.: Техносфера, 2004. с. 368.

[5] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 12.10.2021).

[6] welcome home : vim online [Электронный ресурс]. Режим доступа: https://www.vim.org/ (дата обращения: 12.10.2021).

[7] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#time.process_time_ns (дата обращения: 04.10.2021).

[8] Manjaro — enjoy the simplicity [Электронный ресурс]. Режим доступа: https://manjaro.org/ (дата обращения: 17.10.2021).

[9] Процессор Intel® Core™ i5-8265U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i5-8265u-processor-6m-cache-up-to-3-90-ghz.html (дата обращения: 17.10.2021).