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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

ОТЧЕТ

по лабораторной работе № 2

Название: Трудоёмкость алгоритмов умножения матриц

Дисциплина: Анализ алгоритмов

Студент	ИУ7-51Б		О.А. Тюрин
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Л.Л. Волкова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Содержание

Введение **3**

1 Аналитическая часть **4**
1.1 Описание алгоритмов 4
1.1.1 Обычный алгоритм умножения матриц 4
1.1.2 Алгоритм Винограда 5
1.1.3 Оптимизированный алгоритм Винограда 5
Вывод 5

2 Конструкторская часть **6**
2.1 Разработка алгоритмов 7
2.1.1 Схема алгоритмов 7
Вывод 9

3 Технологическая часть **10**
3.1 Требования к программному обеспечению 10
3.2 Средства реализации 10
3.3 Листинг кода 10
3.3.1 Обычный алгоритм умножения 10
3.3.2 Алгоритм Винограда 11
3.3.3 Улучшенный алгоритм Винограда 12
3.4 Сравнительный анализ обычного и улучшенного Алгоритма Винограда . . . 13
3.5 Трудоемкость алгоритмов 13
3.5.1 Классический алгоритм 13
3.5.2 Алгоритм Винограда 13
3.5.3 Оптимизированный алгоритм Винограда 14
3.5.4 Описание тестирования 14
Вывод 15

4 Экспериментальная часть **16**
4.1 Сравнительный анализ на основе замеров времени работы алгоритмов . . . 16

Заключение **18**

Введение

Целью данной лабораторной работы является исследование методов умножения матриц, получение практического навыка синтезации алгоритма и рассчёт сложности алгоритмов. Помимо обычного метода будут рассмотрены алгоритм Винограда умножения матриц и оптимизированный алгоритм Винограда умножения матриц.
Применение: умножение матрицы на вектор используется в графике для поворота образа.
Задачи для данной ЛР:

- изучение алгоритмов умножения матриц: стандартный и алгоритм Винограда;
- оптимизировать алгоритм Винограда умножения матриц;
- дать теоретическую оценку всем трём алгоритмам;
- релизовать все три алгоритма на одном из языков программирования;
- эксперементально подтвердить различия во временной эффективности всех трёх алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирущихся размерностях матриц.

1 Аналитическая часть

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов. Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Матрицы A и B могут быть перемножены, если они **совместимы** в том смысле, что число столбцов матрицы A равно числу строк B.

1.1 Описание алгоритмов

В данном разделе будет описан каждый из исследуемых алгоритмов.

1.1.1 Обычный алгоритм умножения матриц

Пусть даны две прямоугольные матрицы A и B размеров $[M * N]$ и $[N * Q]$ соответственно.

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

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & a_{1q} \\ b_{21} & b_{22} & \dots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & a_{n2} & \dots & a_{nq} \end{pmatrix}$$

В результате произведения матриц A и B получим матрицу C размера $[M * Q]$, в которой:

$$c_{i,j} = \sum_{r=1}^{N} a_{i,r} \cdot b_{r,j}. \tag{1.1}$$

1.1.2 Алгоритм Винограда

Основной целью алгоритма Винограда является сокращение доли умножений в самом затратном участке кода.

Рассмотри два вектора:

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \end{pmatrix}$$

$$B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

$$A * B = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 + a_4 * b_4 \tag{1.2}$$

Но в то же время:

$$A * B = (a_1 + b_2) * (a_2 + b_1) + (a_3 + b_4) * (a_4 + b_3) - a_1 * a_2 - a_3 * a_4 - b_1 * b_2 - b_3 * b_4 \tag{1.3}$$

В данном примере значения $b_1 * b_2, b_3 * b_4$ можно вычислить заранее и использовать при каждом умножении на строку матрицы A. Значения $a_1 * a_2, a_3 * a_4$ также можно вычислить заранее и использовать при каждом умножении на столбец матрицы B.

1.1.3 Оптимизированный алгоритм Винограда

Для оптимизации алгоритма Винограда могут использоваться такие стратегии, как:

- предварительные вычисления повторяющихся одинаковых действий;
- использование более быстрых операций при вычислении (такие, как смещение битов вместо умножения или деления на 2);
- уменьшения количества повторных проверок;
- использование аналогичных конструкций, уменьшающих трудоёмкость операций (к примеру, замена сложения с 1 на инкремент).

Вывод

Было рассмотрено теоретическое описание исследуемых алгоритмов: Обычный алгоритм умноженмя и алгоритм Винограда.

2 Конструкторская часть

Требования к вводу: на вход подаются размерности каждой из матриц и сами матрицы.

Требования к программе:

- корректное умножение двух матриц;
- программа должна корректно обрабатывать ситуацию несовместности двух матриц.

2.1 Разработка алгоритмов

В данном разделе представлены схемы реализуемых алгоритмов.

2.1.1 Схема алгоритмов

На рисунке 1.1 представлена схема алгоритма тривиального умножения матриц.

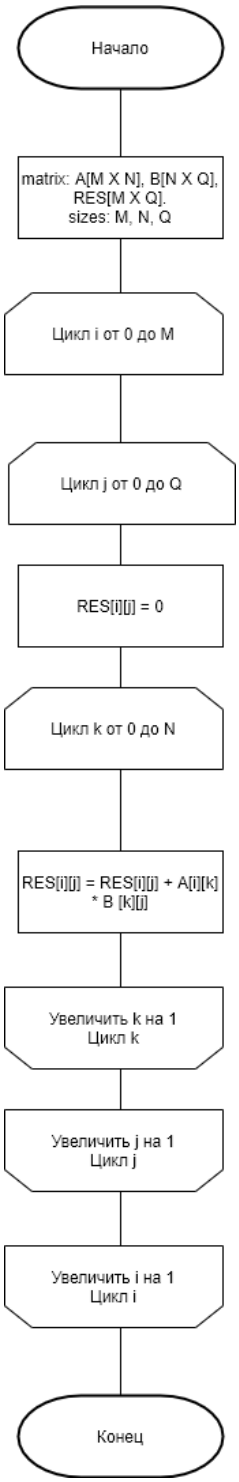


Рис. 1.1: Схема алгоритма тривиального умножения матриц

На рисунке 1.2 представлена схема алгоритма Винограда.

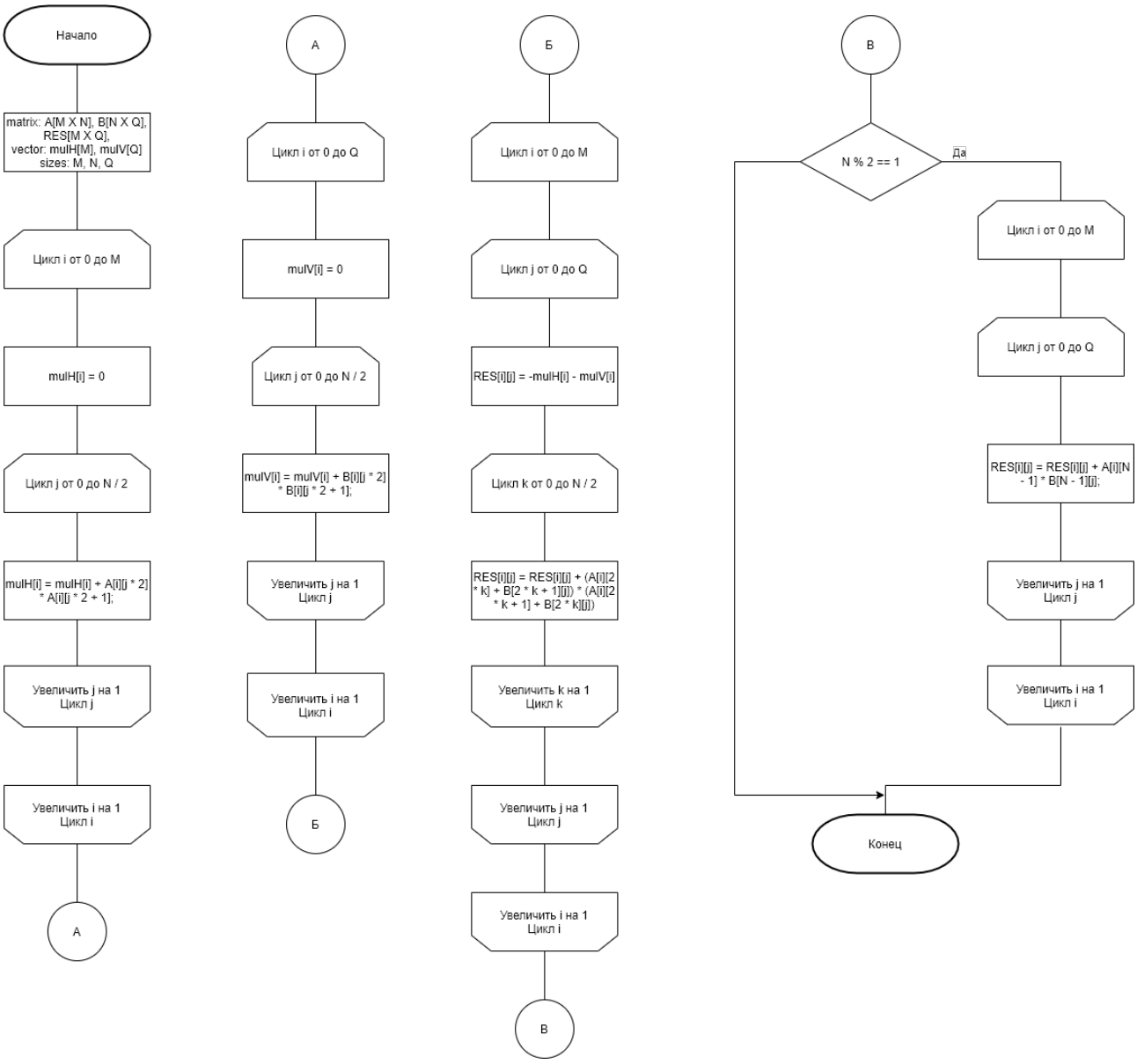


Рис. 1.2: Схема алгоритма Винограда

На рисунке 1.3 представлена схема улучшенного алгоритма Винограда.

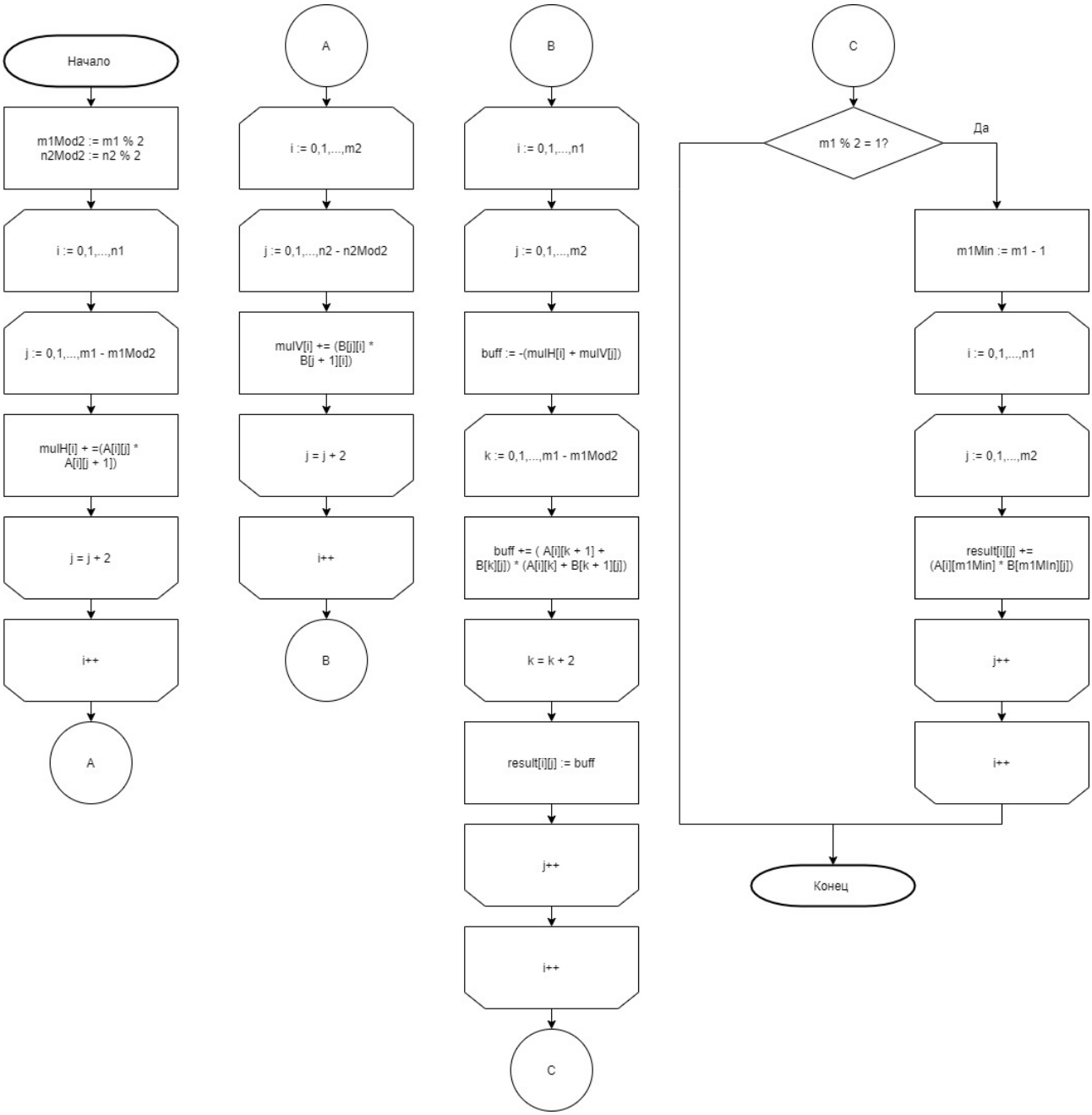


Рис. 1.3: Схема улучшенного алгоритма Винограда

Вывод

В данном разделе были рассмотрены схемы реализуемых алгоритмов.

3 Технологическая часть

В данном разделе будет описана технологическая часть лабораторной работы: требования к ПО, листинг кода, сравнительный анализ всех алгоритмов.

3.1 Требования к программному обеспечению

Входные данные: Размерности двух матриц, сами матрицы.
Выходные данные: результат умножения двух матриц.
Среда выполнения: Windows 10 x64.

3.2 Средства реализации

Для выполнения данной лабораторной работы использовался ЯП Python 3.9.0

3.3 Листинг кода

В данном разделе будет представлен листинг кода разработанных алгоритмов (листинги 3.1 - 3.3).

3.3.1 Обычный алгоритм умножения

Ниже, на листинге 3.1, представлена реализация обычного алгоритма умножения матриц:

Листинг 3.1: Обычный алгоритм умножения матриц

```
1  def default_mult(matrix1, matrix2):
2      if get_columns(matrix1) != get_rows(matrix2):
3          print('Error')
4          return
5
6      result = [[0 for j in range(get_rows(matrix1))]
7          for i in range(get_columns(matrix2))]
8
9      for i in range(len(matrix1)):
10         for j in range(len(matrix2[0])):
11             for k in range(len(matrix1[0])):
12                 result[i][j] += matrix1[i][k] * matrix2[k][j]
13
14     return result
```

3.3.2 Алгоритм Винограда

Ниже, на листинге 3.2, представлена реализация алгоритма Винограда:

Листинг 3.2: Алгоритм Винограда

```
def winograd_multiply(matrix1, matrix2):
    if get_columns(matrix1) != get_rows(matrix2):
        print('Error')
        return

    M = get_rows(matrix1)
    N = get_columns(matrix1)
    Q = get_columns(matrix2)

    result = [[0 for j in range(M)]
        for i in range(Q)]

    mulH = [0 for i in range(M)]
    for i in range(M):
        for j in range(int(N / 2)):
            mulH[i] += matrix1[i][j * 2] * matrix1[i][j * 2 + 1]

    mulV = [0 for i in range(Q)]
    for i in range(Q):
        for j in range(int(N / 2)):
            mulV[i] += matrix2[j * 2][i] * matrix2[j * 2 + 1][i]

    for i in range(M):
        for j in range(Q):
            result[i][j] = -mulH[i] - mulV[j]
            for k in range(int(N / 2)):
                result[i][j] += (matrix1[i][2 * k] +
                    matrix2[2 * k + 1][j]) * (matrix1[i][2 * k + 1] +
                        matrix2[2 * k][j])

    if N % 2 == 1:
        for i in range(M):
            for j in range(Q):
                result[i][j] += matrix1[i][N - 1] *
                    matrix2[N - 1][j]

    return result
```

3.3.3 Улучшенный алгоритм Винограда

Ниже, на листинге 3.3, представлена реализация улучшенного алгоритма Винограда:

Листинг 3.3: Улучшенный алгоритм Винограда

```
def improved_winograd_multiply(matrix1, matrix2):
    if get_columns(matrix1) != L:
        print('Error')
        return

    M = get_rows(matrix1)
    N = get_columns(matrix1)
    Q = get_columns(matrix2)
    L = get_rows(matrix2)
    Lmod2 = L % 2
    Nmod2 = N % 2

    result = [[0 for j in range(M)]
        for i in range(Q)]

    mulH = [0 for i in range(M)]
    mulV = [0 for i in range(Q)]

    for i in range(M):
        for j in range(0, N - Nmod2, 2):
            mulH[i] += matrix1[i][j] * matrix1[i][j + 1]

    for i in range(Q):
        for j in range(0, L - Lmod2, 2):
            mulV[i] += matrix2[j][i] * matrix2[j + 1][i]

    for i in range(M):
        for j in range(Q):
            buff = -mulH[i] - mulV[j]
            for k in range(0, N -
                Nmod2, 2):
                buff += (matrix1[i][k + 1] + matrix2[k][j]) *
                        (matrix1[i][k] + matrix2[k + 1][j])
            result[i][j] = buff

    if Nmod2:
        for i in range(M):
            for j in range(Q):
                result[i][j] += matrix1[i][N - 1] *
                    matrix2[N - 1][j]

    return result
```

3.4 Сравнительный анализ обычного и улучшенного Алгоритма Винограда

Улучшенный вариант алгоритма Винограда отличается от обычного тем, что теперь заранее высчитываются значения, для избавления от деления в цикле. А также инициализируется буфер для накопления результата умножения, который после цикла сбрасывается в ячейку матрицы.

3.5 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1: +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса (в том числе и геттеры);
- оценка трудоемкости цикла: $F_{cycle} = init + N * (statement + iteration + F_{body}) + statement$, где init - инициализация цилка, N - количество итераций цикла, iteration - действие в конце каждой итерации цикла, statement - выражение, описывающее условие цикла;
- стоимость условного перехода применим за 0, стоимость вычисления условия остаётся.

3.5.1 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма с матрицами A и B и размерами $m \times n$ и $n \times k$:

$$2 + M(2 + 2 + K(2 + 2 + N(2 + 6 + 2))) = 10MNK + 4MK + 4M + 2.$$

3.5.2 Алгоритм Винограда

Аналогично рассмотрим трудоемкость алгоритма Винограда при тех же матрицах и размерах.

Часть алгоритма	Трудоёмкость
Инициализация mulH и mulV	$2 \cdot 3$
Заполнение mulH	$2 + m \cdot (2 + 2 + n/2 \cdot (3 + 6 + 6))$
Заполнение mulV	$2 + k \cdot (2 + 2 + n/2 \cdot (3 + 6 + 6))$
Подсчёт результата	$2 + m \cdot (2 + 2 + k \cdot (2 + 7 + 2 + n/2 \cdot (3 + 23)))$
Условный оператор нечёт. n	2
Для матриц с нечёт n	$2 + m \cdot (2 + 2 + k \cdot (2 + 8 + 5))$

Таблица 1: трудоёмкость алгоритма Винограда

$$f = 13NMK + 7.5NM + 7.5KN + 11KM + 8M + 4K + 14 + \begin{cases} 0, \text{если m чётное} \\ 15 \cdot K \cdot M + 4 \cdot M + 2, \text{иначе} \end{cases}$$

3.5.3 Оптимизированный алгоритм Винограда

Теперь рассмотрим трудоемкость оптимизированого алгоритма Винограда при тех же матрицах и размерах.

Часть алгоритма	Трудоёмкость
Заполнение mulH	$2 + m \cdot (2 + 2 + n/2 \cdot (2 + 5 + 3))$
Заполнение mulV	$2 + k \cdot (2 + 2 + n/2 \cdot (2 + 5 + 3))$
Подсчёт результата	$2 + m \cdot (2 + 2 + k \cdot (2 + 5 + 3 + 2 + n/2 \cdot (2 + 14)))$
Условный оператор нечёт. n	2
Для матриц с нечёт n	$2 + 2 + m \cdot (2 + 2 + k \cdot (2 + 6 + 2))$

Таблица 2: трудоёмкость оптимизированного алгоритма Винограда

$$f = 8NMK + 5NM + 5KN + 12KM + 8M + 4K + 18 + \begin{cases} 0, \text{если m чётное} \\ 10 \cdot K \cdot M + 4 \cdot M + 4, \text{иначе} \end{cases}$$

3.5.4 Описание тестирования

Были проведены следующие тривиальные тесты:

Листинг 1: Число строк равно числу стобцов

```
A matrix:
1 2
3 4

B matrix:
5 6
7 8

Default multiplication:
19 22
43 50

Winograd:
19 22
43 50

Improved Winograd:
19 22
43 50
```

Листинг 2: Число строк равно числу стобцов

```
A matrix:
1 2 3
4 5 6
7 8 9

B matrix:
```

```
1 2 3
4 5 6
7 8 9

Default multiplication:
30 36 42
66 81 96
102 126 150

Winograd:
30 36 42
66 81 96
102 126 150

Improved Winograd:
30 36 42
66 81 96
102 126 150
```

Ниже, на листинге 3.4, представлен листинг тестирования корректной работы алгоритмов.

Листинг 3.4: Тестирование корректной работы алгоритмов

```
def matrix_test():
    M = random.randint(1, 30)
    N = random.randint(1, 30)
    Q = random.randint(1, 30)

    matrix1 = [[random.randint(1, 100) for j in range(M)]
                        for i in range(N)]
    matrix2 = [[random.randint(1, 100)
                        for j in range(N)] for i in range(Q)]

    res1 = default_mult(matrix1, matrix2)
    res2 = winograd_multiply(matrix1, matrix2)
    res3 = improved_winograd_multiply(matrix1, matrix2)

    return res1 == res2 == res3
```

Все тесты пройдены успешно.

Вывод

В данном разделе был представлен листинг реализованных алгоритмов, а также описание тестирование корректности их работы.

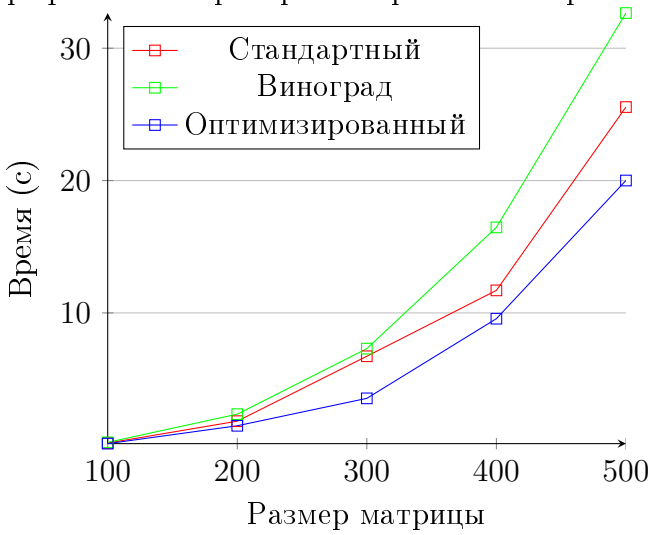
4 Экспериментальная часть

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Эксперименты производятся на квадратных матрицах размером от 100/101 x 100/101 до 500/501 x 500/501 с шагом 100. Эксперимент для каждой размерности матрицы проводится 10 раз, а итоговое время усредняется. Время работы алгоритмов измеряется в секундах.

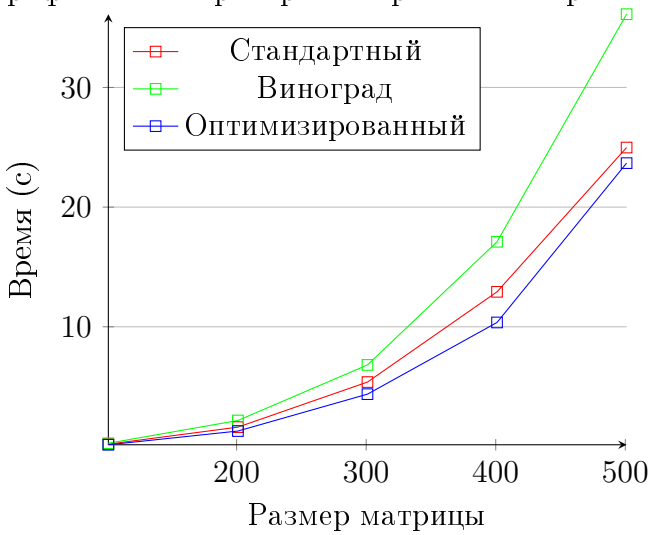
Сравнение времени работы алгоритмов при четном размере матрицы
На графике изображены зависимости времени выполнения программы от четного размера входных квадратных матриц:

График 1: Замеры времени работы алгоритмов



Сравнение времени работы алгоритмов при нечетном размере матрицы
На графике изображены зависимости времени выполнения программы от нечетного размера входных квадратных матриц:

График 2: Замеры времени работы алгоритмов



Вывод

По результатам тестирования все рассматриваемые алгоритмы реализованы так, что возвращают одинаковый результат. Самым медленным алгоритмом оказался алгоритм классического умножения матриц, а самым быстрым — оптимизированный алгоритм Винограда.

Заключение

Цель достигнута и все задачи выполнены.

В ходе работы были изучены и реализованы алгоритмы умножения матриц: классический алгоритм, алгоритм Винограда, улучшенный алгоритм Винограда. Была дана теоретическая оценка всех рассматриваемых алгоритмов, было проведено тестирование и выполнено сравнение всех рассматриваемых алгоритмов. В ходе исследования было установлено, что улучшенный алгоритм Винограда является самым быстрым алгоритмом по трудоемкости и времени выполнения среди рассмотренных алгоритмов умножения матриц. Изучены зависимости времени выполнения алгоритмов от размерности матриц.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. – М.: Техносфера, 2017. – 267с

[2] Кострикин А.И., Манин Ю.И. Линейная алгебра и геометрия. – СПб: Лань, 2018. – 303 с.

[3] Официальный сайт Python, документация [электронный ресурс]. Режим доступа: https://docs.python.org/3.9, свободный – (Дата обращения: 8.10.20)