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"

Тема Алгоритмы умножения матриц

Студент Беляев Н.А.

Группа ИУ7-51Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Операция умножения матриц используется во многих отраслях компьютерных наук, например, в компьютерной графике и машинном обучении. Необходимо оптимизировать выполнение данной операции, чтобы алгоритмы на её основе выполнялись эффективно.

Цель работы — описать, реализовать и сравнить следующие алгоритмы:

- стандартный алгоритм умножения матриц;
- алгоритм Винограда;
- оптимизированный алгоритм Винограда.

Для достижения цели необходимо выполнить следующие задачи:

- описать математические основания алгоритмов;
- спроектировать и реализовать алгоритмы;
- ввести модель вычислений и оценить трудоемкость алгоритма Винограда и его оптимизированной версии;
- провести замер процессорного времени работы алгоритмов.

1 Аналитический раздел

В разделе дано определение матрицы и операции умножения двух матриц. Описаны математические основания алгоритмов умножения матриц.

1.1 Матрица как математический объект

Матрица $M_{m \times n}$ — таблица из m строк и n столбцов, содержащая элементы некоторого множества. В общем случае элементы матрицы могут быть любыми, но для определённости будем рассматривать числовые матрицы. Элемент матрицы a_{ij} находится в i-й строке и j-м столбце. Строки и столбцы матрицы можно рассматривать как векторы.

1.2 Операция умножения матриц

Пусть даны матрицы $A_{m \times n}$ и $B_{n \times q}$. Результатом выполнения операции умножения $A \cdot B$ является матрица $C_{m \times q}$, элементы которой определяются согласно формуле (1.1):

$$c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj} \tag{1.1}$$

Стоит отметить два обстоятельства:

– операция умножения матриц некоммутативна;

– количество столбцов первой матрицы должно совпадать с количеством строк второй матрицы.

Операция скалярного произведения векторов $V = (v_1, \ldots, v_n)$ и $W = (w_1, \ldots, w_n)$ описывается формулой (1.2):

$$V \cdot W = v_1 \cdot w_1 + \ldots + v_n \cdot w_n \tag{1.2}$$

Тогда под элементом c_{ij} в формуле (1.1) можно понимать результат скалярного произведения i-й вектора-строки первой матрицы A на вектор-столбец второй матрицы B под индексом j.

1.3 Стандартный алгоритм умножения матриц

Стандартный алгоритм умножения матриц целиком базируется на математическом определении данной операции (1.2).

1.4 Алгоритм Винограда

Проанализируем стандартный алгоритм (1.3) на наличие повторных вычислений. Даны векторы $V = (v_1, \ldots, v_4)$ и $W = (w_1, \ldots, w_4)$. Их скалярное произведение определено формулой (1.2), но его можно переписать несколько иначе:

$$\begin{aligned} V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) \\ -(v_1 \cdot v_2) - (v_3 \cdot v_4) - (w_1 \cdot w_2) - (w_3 \cdot w_4) \end{aligned} \tag{1.3}$$

Сгруппированные в скобки выражения можно вычислить заранее, таким образом при расчете значения очередного элемента потребуется сделать две операции умножения и пять операций сложения. В стандартном алгоритме для аналогичного расчета применяется четыре операции умножения и три операции сложения. Аппаратно операция сложения эффективнее умножения, потому данный алгоритм должен показывать производительность лучше, нежели стандартный алгоритм.

В случае нечетного размера матрицы следует произвести дополнительную операцию – добавление произведений последних элементов соответствующих строк и столбцов.

1.5 Оптимизированный алгоритм Винограда

Существует множество программных оптимизаций алгоритма Винограда (1.4). В данной работе будут применены следующие изменения:

– инкрементировать счётчик наиболее вложенного цикла на два;

– для инкремента использовать оператор $+=$;

– при вычислении вспомогательных массивов использовать декремент.

Вывод

В разделе было определено понятие матрицы и операции умножения матриц. Описаны алгоритмы умножения матриц.

2 Конструкторский раздел

В данном разделе приведены схемы алгоритмов умножения матриц, рассмотренных в предыдущем разделе. Также для каждого алгоритма проведена оценка трудоемкости в соответствии с определённой моделью вычислений.

2.1 Схемы алгоритмов

2.1.1 Стандартный алгоритм умножения матриц

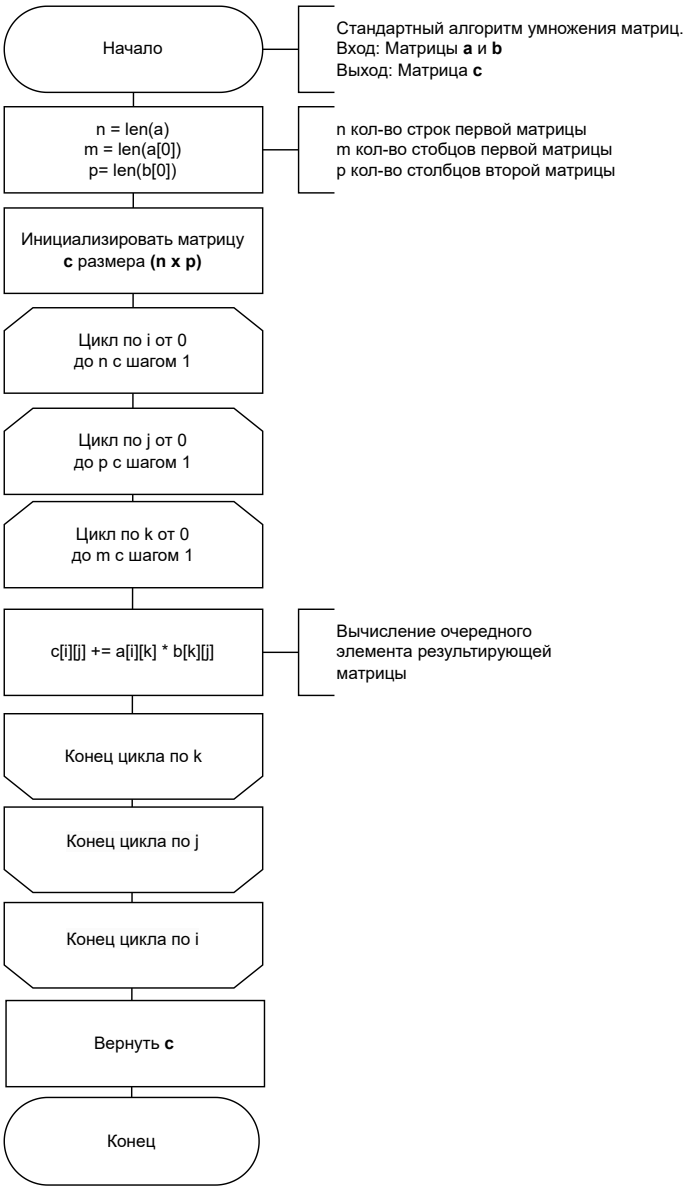

Рисунок 2.1 – Схема алгоритма стандартного умножения матриц

2.1.2 Алгоритм Винограда

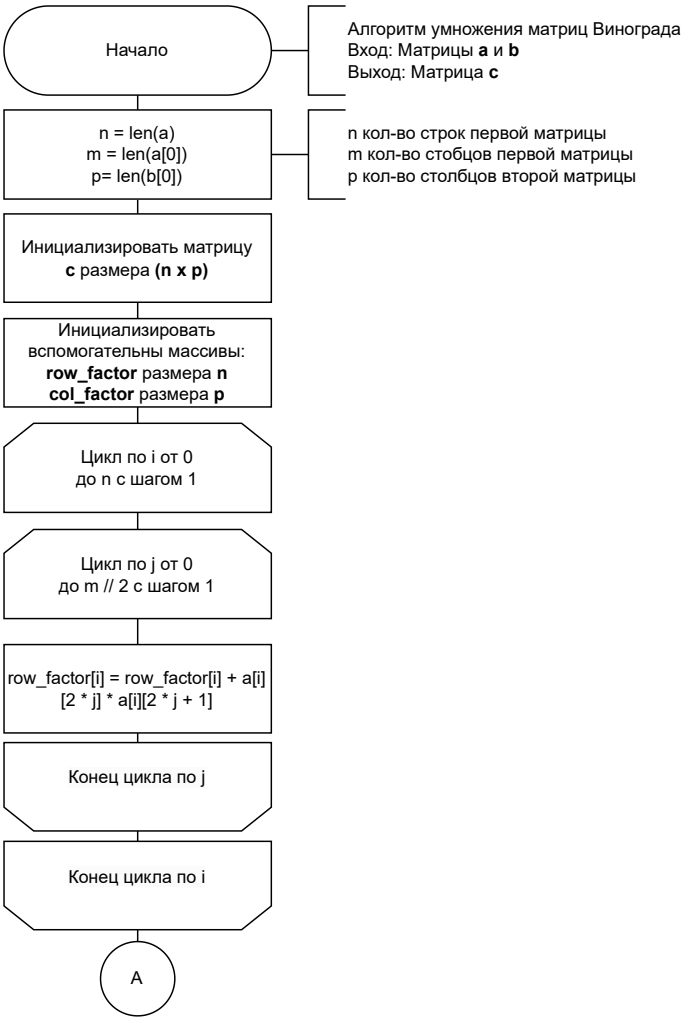

Рисунок 2.2 – Схема алгоритма Винограда. Первая часть

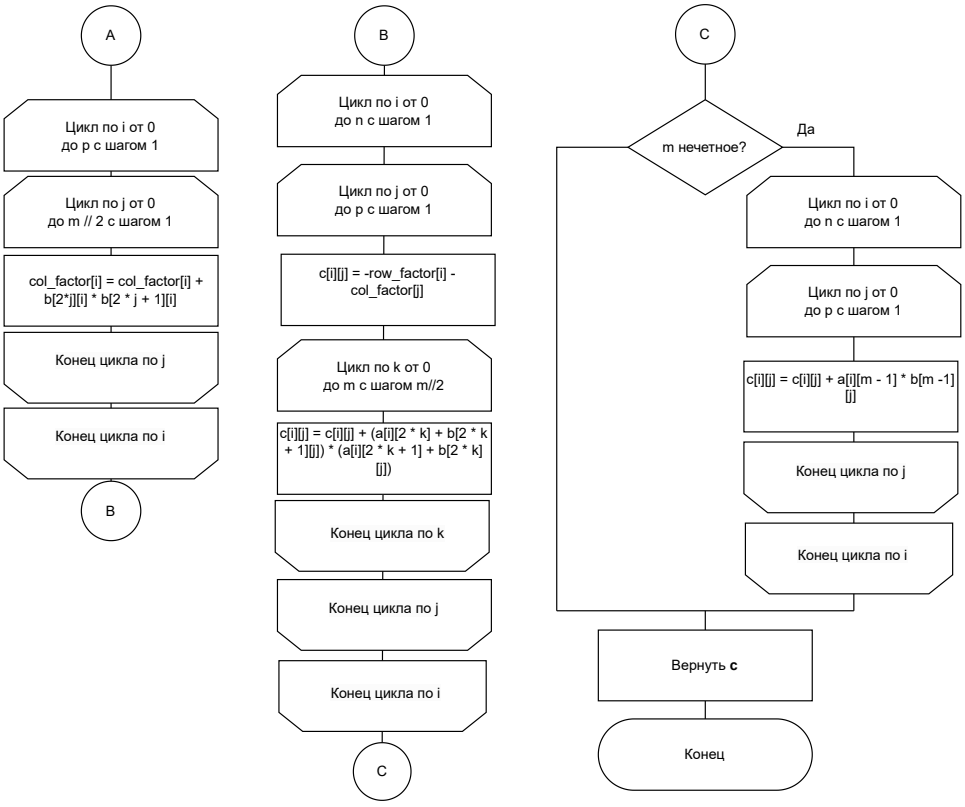

Рисунок 2.3 – Схема алгоритма Винограда. Вторая часть

2.1.3 Оптимизированный алгоритм Винограда

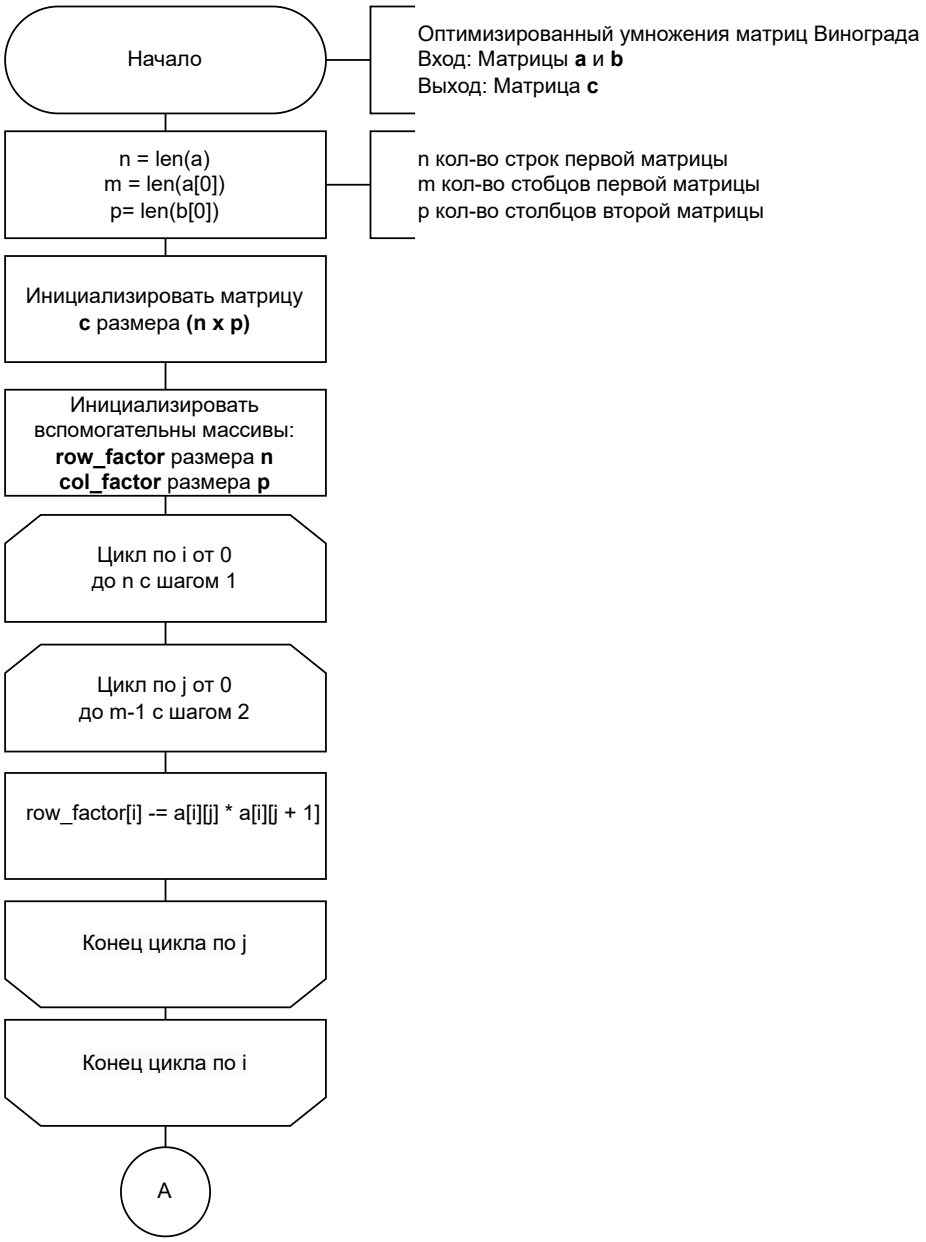

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда. Первая часть

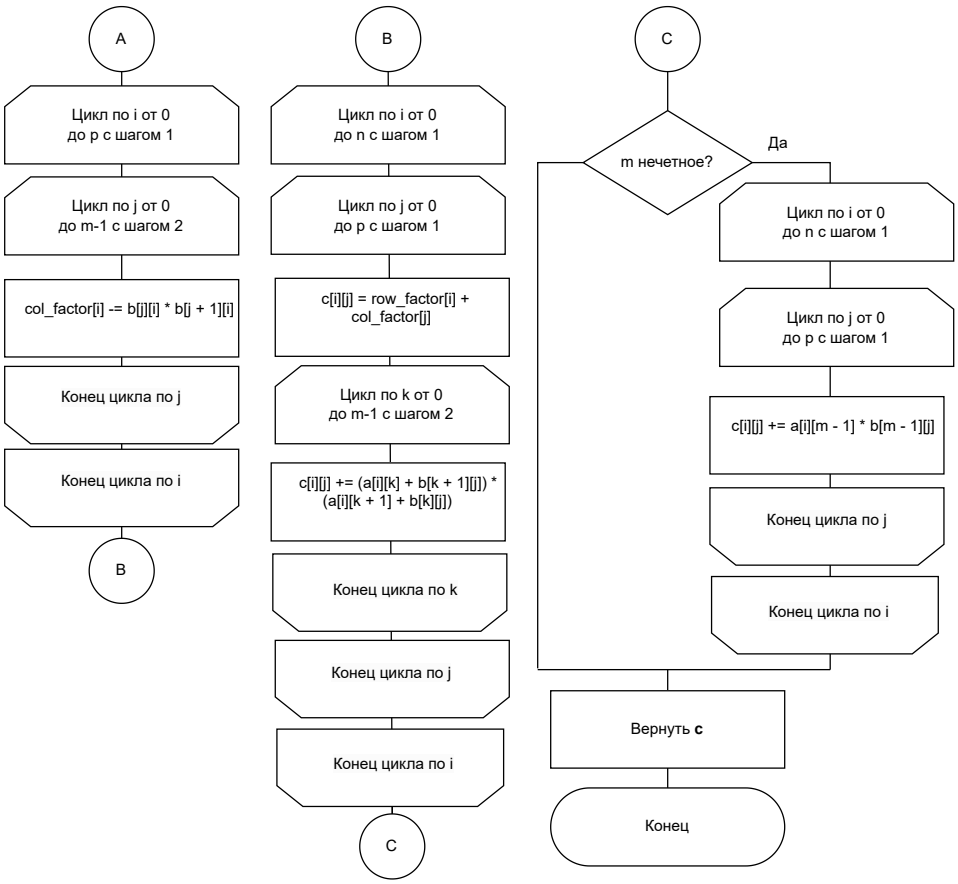

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда. Вторая часть

2.2 Оценка трудоемкости

2.2.1 Модель вычислений

Операции, имеющие единичную стоимость:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

Операции, имеющие двойную стоимость:

$$*, /, \%, * =, / =, \% =, \text{len}() \tag{2.2}$$

Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} \min(f_1, f_2), & \text{лучший случай} \\ \max(f_1, f_2), & \text{худший случай} \end{cases} \tag{2.3}$$

Трудоемкость цикла:

$$f_{loop} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \times (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

Дальнейший расчет трудоемкости будет производиться с использованием данной модели.

2.2.2 Трудоемкость классического алгоритма

Стоимость начальной инициализации рассчитывается по формуле (2.5):

$$f_{init} = (2 + 1) + (2 + 1 + 1) + (2 + 1 + 1) \\ + 2 + n \times (2 + p) \tag{2.5}$$

Стоимость основного цикла рассчитывается по формуле (2.6):

$$f_{main} = 2 + n \times (2 + p \times (2 + m \times (2 + 9))) \tag{2.6}$$

Итоговая стоимость трудоемкости рассчитывается по формуле (2.7):

$$f = f_{main} + f_{init} = 15 + 4 \times n + 3 \times n \times p + 11 \times m \times n \times p \approx 11 \times m \times n \times p = O(N^3) \tag{2.7}$$

2.2.3 Трудоемкость алгоритма Винограда

Стоимость начальной инициализации рассчитывается по формуле (2.8):

$$f_{init} = (2+1) + (2+1+1) + (2+1+1) + 2 + n \times (2+p) = 13 + n \times (2+p) \tag{2.8}$$

Стоимость инициализации вспомогательных массивов вычисляется по формулам (2.9) и (2.10):

$$f_{row} = 2 + 2 \times n + 8.5 \times m \times n \tag{2.9}$$

$$f_{col} = 2 + 2 \times n + 8.5 \times m \times p \tag{2.10}$$

Стоимость основного цикла рассчитывается по формуле (2.11):

$$f_{main} = 2 + 2 \times n + n \times p \times (9 + 15 \times m) = 2 + 2 \times n + 9 \times n \times p + 15 \times m \times n \times p \tag{2.11}$$

Добавочная трудоемкость цикла рассчитывается по формуле (2.20):

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + 2n + 15 \times n \times p, & \text{иначе} \end{cases} \tag{2.12}$$

Итоговая стоимость трудоемкости рассчитывается по формуле (2.13):

$$f = f_{init} + f_{row} + f_{col} + f_{main} + f_{check} \tag{2.13}$$

В лучшем случае трудоемкость алгоритма составляет:

$$f = 13 + n \times (2 + p) + 4 + 4 \times n + 8.5 \times m \times n + 8.5 \times m \times p + \\ 2 + 2n + 9 \times n \times p + 15 \times m \times n \times p \approx 15 \times m \times n \times p = O(N^3) \tag{2.14}$$

В худшем случае трудоемкость алгоритма составляет:

$$f = 13 + n \times (2 + p) + 4 + 4 \times n + 8.5 \times m \times n + 8.5 \times m \times p + \\ 2 + 2 \times n + 9 \times n \times p + 15 \times m \times n \\ \times p + 2 + 2 \times n + 15 \times n \times p \approx 15 \\ \times m \times n \times p = O(N^3) \tag{2.15}$$

2.2.4 Трудоемкость оптимизированного алгоритма Винограда

Стоимость начальной инициализации рассчитывается по формуле (2.16):

$$f_{init} = (2 + 1) + (2 + 1 + 1) + (2 + 1 + 1) \\ +2 + n \times (2 + p) = 13 + n \times (2 + p) \tag{2.16}$$

Стоимость инициализации вспомогательных массивов вычисляется по формулам (2.17) и (2.18):

$$f_{row} = 2 - 2.5 \times n + 4.5 \times m \times n \tag{2.17}$$

$$f_{col} = 2 - 2.5 \times n + 4.5 \times m \times p \tag{2.18}$$

Стоимость основного цикла рассчитывается по формуле (2.19):

$$f_{main} = 2 + 2 \times n + 16.5 \times p \times n + 8.5 \times m \times n \times p \tag{2.19}$$

Добавочная трудоемкость цикла рассчитывается по формуле (2.20):

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + 2 \times n + 15 \times n \times p, & \text{иначе} \end{cases} \tag{2.20}$$

Итоговая стоимость трудоемкости рассчитывается по формуле (2.21):

$$f = f_{init} + f_{row} + f_{col} + f_{main} + f_{check} \tag{2.21}$$

В лучшем случае трудоемкость алгоритма составляет:

$$f = 4 - 5 \times n + 4.5 \times m \times n + 4.5 \times m \times p + 2 + 2 \times n + 16.5 \times p \times n + 8.5 \times m \times n \times p \approx 8.5 \times m \times n \times p = O(n^3) \tag{2.22}$$

В худшем случае трудоемкость алгоритма составляет:

$$\begin{gathered} f = 4 - 5 \times n + 4.5 \times m \times n + 4.5 \times m \times \\ p + 2 + 2 \times n + 16.5 \times p \times n + 8.5 \times m \times n \times p + \\ 2 + 2 \times n + 15 \times p \times n \approx 8.5 \times m \times n \times p = O(n^3) \end{gathered} \tag{2.23}$$

Вывод

В разделе были приведены схемы алгоритмов умножения матриц, а также рассчитана их трудоемкость. В данной модели вычислений все три алгоритма имеют кубическую сложность, а самым маленьким коэффициентом обладает оптимизированный алгоритм Винограда.

3 Технологический раздел

Раздел содержит описание средств реализации программы, листинги кода алгоритмов и функциональные тесты.

3.1 Средства реализации

Для реализации программы выбран язык программирования *Python* [1]. Данный язык позволяет замерить процессорное время работы алгоритма с помощью функции *process_time* модуля *time* [2].

3.2 Реализация алгоритмов

Листинги 3.1 - 3.3 содержат реализации рассматриваемых алгоритмов умножения матриц:

Листинг 3.1 – Стандартный алгоритм

```
def dotprod_std(a: Matrix, b: Matrix) -> Matrix:
    n = len(a)
    m = len(a[0])
    p = len(b[0])
    c = [[0] * p for _ in range(n)]

    for i in range(n):
        for j in range(p):
            for k in range(m):
                c[i][j] += a[i][k] * b[k][j] #

    return c
```

Листинг 3.2 – Алгоритм Винограда

```
def dotprod_winograd(a: Matrix, b: Matrix) -> Matrix:
    n = len(a)
    m = len(a[0])
    p = len(b[0])

    row_factor = [0] * n
    col_factor = [0] * p

    for i in range(n):
        for j in range(0, m // 2):
            row_factor[i] = row_factor[i] + a[i][2 * j] * a[i][2
               * j + 1]

    for i in range(p):
        for j in range(0, m // 2):
            col_factor[i] = col_factor[i] + b[2 * j][i] * b[2 *
               j + 1][i]

    c = [[0] * p for _ in range(n)]

    for i in range(n):
        for j in range(p):
            c[i][j] = -row_factor[i] - col_factor[j] # 7
            for k in range(0, m // 2):
                c[i][j] = c[i][j] + (a[i][2 * k] + b[2 * k +
                   1][j]) * (a[i][2 * k + 1] + b[2 * k][j])

    if m % 2 == 1:
        for i in range(n):
            for j in range(p):
                c[i][j] = c[i][j] + a[i][m - 1] * b[m - 1][j]

    return c
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
def dotprod_winograd_optimized(a: Matrix, b: Matrix) -> Matrix:
    n = len(a)
    m = len(a[0])
    p = len(b[0])

    row_factor = [0] * n
    col_factor = [0] * p

    for i in range(n):
        for j in range(0, m - 1, 2):
            row_factor[i] -= a[i][j] * a[i][j + 1]
    for i in range(p):
        for j in range(0, m - 1, 2):
            col_factor[i] -= b[j][i] * b[j + 1][i]

    c = [[0] * p for _ in range(n)]

    for i in range(n):
        for j in range(p):
            c[i][j] = row_factor[i] + col_factor[j]
            for k in range(0, m - 1, 2):
                c[i][j] += (a[i][k] + b[k + 1][j]) * (a[i][k +
                   1] + b[k][j])

    if m % 2 == 1:
        for i in range(n):
            for j in range(p):
                c[i][j] += a[i][m - 1] * b[m - 1][j]

    return c
```

3.3 Функциональное тестирование

Во время функционального тестирования реализаций алгоритмов были рассмотрены следующие случаи:

– перемножение матриц размера 1×1;

– перемножение квадратных матриц;

– перемножение неквадратных матриц.

Тест 1. Перемножение матриц размера 1×1.

Входные данные.

$$A = \begin{bmatrix} 2 \end{bmatrix}, \quad B = \begin{bmatrix} 3 \end{bmatrix}$$

Результат.

$$A \times B = \begin{bmatrix} 6 \end{bmatrix}$$

Тест 2. Перемножение квадратных матриц.

Входные данные.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Результат:

$$A \times B = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 7 & 1 \cdot 6 + 2 \cdot 8 \\ 3 \cdot 5 + 4 \cdot 7 & 3 \cdot 6 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Тест 3. Перемножение неквадратных матриц.

Входные данные.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

Результат.

$$A \times B = \begin{bmatrix} 1 \cdot 7 + 2 \cdot 9 + 3 \cdot 11 & 1 \cdot 8 + 2 \cdot 10 + 3 \cdot 12 \\ 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11 & 4 \cdot 8 + 5 \cdot 10 + 6 \cdot 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

Все тесты пройдены успешно.

Вывод

В разделе были описаны средства реализации алгоритмов, приведены листинги кода и описание функционального тестирования.

4 Исследовательский раздел

Раздел содержит описание замера процессорного времени работы рассматриваемых алгоритмов умножения матриц.

4.1 Замер процессорного времени

Замер процессорного времени выполнен на ноутбуке *HUAWEI MateBook 14 Core Ultra*. Его характеристики:

– процессор *Intel® Core™ Ultra 5 processor 125H*;

– объем оперативной памяти 16 ГБ;

– ОС *Ubuntu 22.03* [3].

Замер проведен с помощью функции *process_time* модуля *time*. Исследовалась зависимость времени работы алгоритма умножения от линейного размера входных квадратных матриц. Замер для каждого размера производился $N = 10$ раз. В качестве результата выбиралось среднее арифметическое.

В таблице 4.1 приведены результаты замера процессорного времени работы алгоритмов.

Таблица 4.1 – Замер времени для матриц размером от 1 до 301

Размер матрицы	**Стандартный**	**Виноград**	**Оптимизированный Виноград**
1	0.000008	0.000016	0.000011
11	0.000161	0.000197	0.000157
21	0.000976	0.001040	0.000943
31	0.003113	0.003058	0.002724
41	0.006913	0.007010	0.006168
51	0.013487	0.015133	0.011492
61	0.024077	0.027390	0.025145
71	0.043437	0.041829	0.040956
81	0.058066	0.062298	0.049888
91	0.112635	0.081608	0.068125
101	0.116776	0.122409	0.091142
111	0.148105	0.147729	0.118995
121	0.178187	0.208828	0.160765
131	0.242555	0.259179	0.226213
141	0.309648	0.293353	0.257039
151	0.390584	0.367361	0.330666
161	0.430135	0.462702	0.379085
171	0.557330	0.540833	0.534774
181	0.630058	0.676642	0.529143
191	0.774023	0.756067	0.651742
201	0.890761	0.949499	0.810395
211	1.069306	1.017448	0.883063
221	1.189870	1.168395	1.066466
231	1.339761	1.373136	1.262913
241	1.525564	1.586061	1.369485
251	1.704104	1.763292	1.499202
261	2.056993	1.901680	1.722462
271	2.195457	2.331817	1.945634
281	2.475677	2.594344	2.210080
291	2.760152	2.716592	2.394943
301	3.134695	3.099551	2.664289

На рисунке 4.1 табличные данные отображены графически.

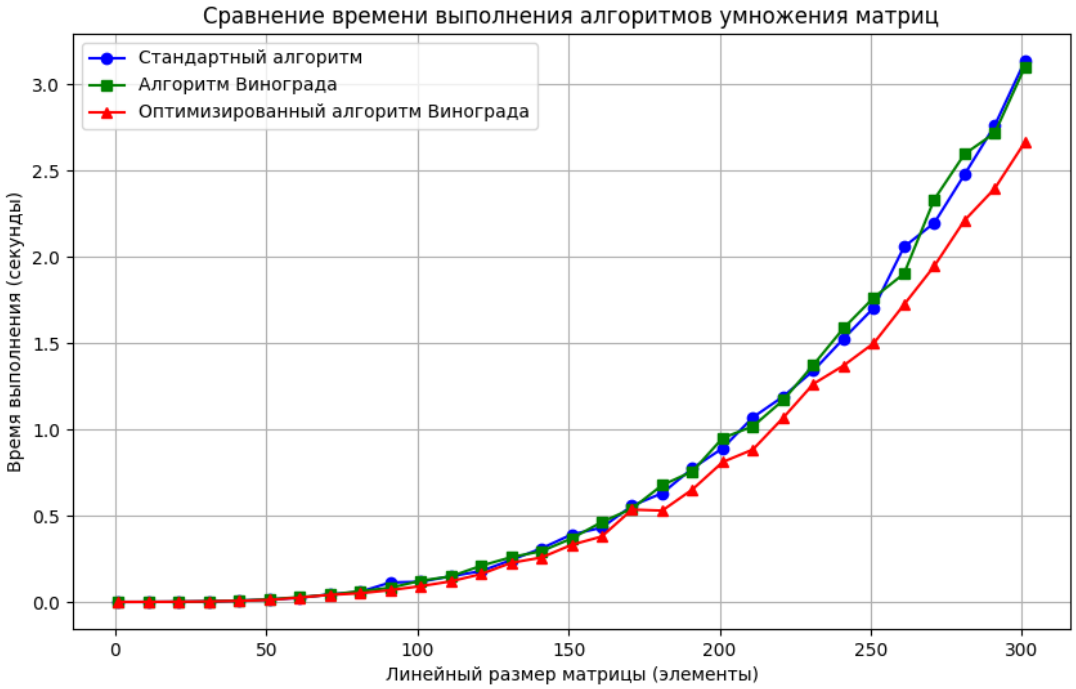

Рисунок 4.1 – Зависимость процессорного времени работы алгоритма от линейного размера матриц

Вывод

Замер процессорного времени подтвердил теоретический расчет трудоемкости. Все рассмотренные алгоритмы имеют схожую эффективность. На линейных размерах матриц до 80 элементов разницей во времени работы алгоритмов можно пренебречь. На размерах больше 80 алгоритм Винограда демонстрирует выигрыш в 7–10 % по сравнению с другими алгоритмами. Стандартный алгоритм и алгоритм Винограда показывают идентичные результаты на рассматриваемых размерах входных данных.

ЗАКЛЮЧЕНИЕ

Задачи лабораторной работы выполнены:

- описаны математические основания алгоритмов;
- алгоритмы спроектированы и реализованы;
- введена модель вычислений. На её основе проведена оценка трудоемкости алгоритмов;
- проведен замер процессорного времени работы алгоритмов.

Цель лабораторной работы достигнута: алгоритмы описаны и реализованы. Выполнено сравнение времени работы алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Python Documentation [Электронный ресурс]. — Режим доступа: https://www.python.org (дата обращения: 20.10.2024).
2. Time access and conversions [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 20.10.2024).
3. Huawei MateBook Core Ultra Specification [Электронный ресурс]. — Режим доступа: https://consumer.huawei.com/en/laptops/matebook-14-2024/specs/ (дата обращения: 20.10.2024).