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 дисциплине "Анализ алгоритмов"

Тема Умножение матриц

Студент Шацкий Р.Е.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2021 г.

Оглавление

Введение

Умножение матриц является центральной операцией во многих численных алгоритмах, потому много усилий было вложено в повышение эффективности алгоритма умножения матриц. Приложения алгоритма умножения матриц в вычислительных задачах найдены во многих областях, включая научные вычисления и распознавания образов.

Цель: сравнить классический алгоритм произведения матриц с алгоритмом Копперсмита-Винограда и его оптимизацией.

Задачи:

1. Изучить алгоритмы классического произведения матриц и Копперсмита-Винограда.
2. Реализовать следующие алгоритмы:
   (a) классического умножения матриц;
   (b) Копперсмита-Винограда;
   (c) оптимизированный Копперсмита-Винограда.
3. Вычислить и сравнить трудоемкость реализуемых алгоритмов на основе выбранной модели вычислений.
4. Провести сравнительный анализ алгоритмов по затрачиваемым ресурсам (процессорному времени работы).

1 Аналитическая часть

Умножение матриц - операция, широко применяющаяся в различных программах и приложениях. Использование данной операции повлекло за собой необходимость её программной реализации. Классический алгоритм умножения матриц является интуитивно понятным, поскольку повторяет действия, производимые человеком для нахождения произведения двух матриц. Однако частое использование операции умножения матриц привело к необходимости поиска более быстрой реализации, ведь несмотря на простоту, алгоритмы, основанные на последовательном переборе элементов, для больших объемов данных работают достаточно долго.

Алгоритм Копперсмита-Винограда — это алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом [1]. Изначально ассимптотическая сложность алгоритма составляла $O(n^{2.3755})$, где n — это размер стороны матрицы. Алгоритм Копперсмита-Винограда, с учётом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита — Винограда не используется, поскольку имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров [2]. Поэтому пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости [3].

1.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы, причем количество столбцов первой совпадает с количеством строк второй

$$A_{nm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B_{mq} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mq} \end{pmatrix} \tag{1.1}$$

Тогда матрица $C = A \times B$, называющаяся произведением матриц A и B выглядит следующим образом:

$$C_{nq} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1q} \\ c_{21} & c_{22} & \dots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nq} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \quad (i = 1,\vec{n}, j = 1,\vec{q}) \tag{1.3}$$

Программная реализация классического алгоритма умножения матриц заключается в прямой реализации формулы 1.3 для вычисления каждого элемента результирующей матрицы.

1.2 Алгоритм Копперсмита-Винограда умножения матриц

Если посмотреть на результат умножения двух матриц, можно заметить, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $\vec{u} = (u_1, u_2, u_3, u_4)$ и $\vec{w} = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $\vec{u} \cdot \vec{w} = u_1 \cdot w_1 + u_2 \cdot w_2 + u_3 \cdot w_3 + u_4 \cdot w_4$, что эквивалентно 1.4.

$$\vec{u} \cdot \vec{w} = (u_1 + w_2)(u_2 + w_1) + (u_3 + w_4)(u_4 + w_3) - u_1 \cdot u_2 - u_3 \cdot u_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.4}$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм (вместо четырех умножений вычисляется шесть, вместо трех сложений - десять), выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с двумя предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

1.3 Вывод

Алгоритм Копперсмита-Винограда отличается от классического алгоритма наличием предварительной обработки матриц и уменьшением общего количества операций умножения, которые выполняются медленнее операций сложения.

2 Конструкторская часть

Данный раздел требования к разрабатываемому программному обеспечению, содержит схемы алгоритмов, реализуемых в работе (классический, Копперсмита-Винограда и оптимизированный Копперсмита-Винограда), а также теоретические вычисления трудоемкости для каждого из них.

2.1 Требования к программному обеспечению

Требования, выдвигаемые к разрабатываемому ПО:

- предусмотреть возможность просмотра результатов тестирования алгоритмов;
- предусмотреть возможность просмотра процессорного времени, использованного алгоритмами;
- входные данные - размеры двух матриц, их элементы;
- выходные данные - матрица, являющаяся произведением первой входной матрицы на вторую.

2.2 Схемы алгоритмов

В данном пункте раздела представлены схемы реализуемых в работе алгоритмов.

2.2.1 Схема классического алгоритма

На рисунке 2.1 представлена схема обычного алгоритма умножения матриц.

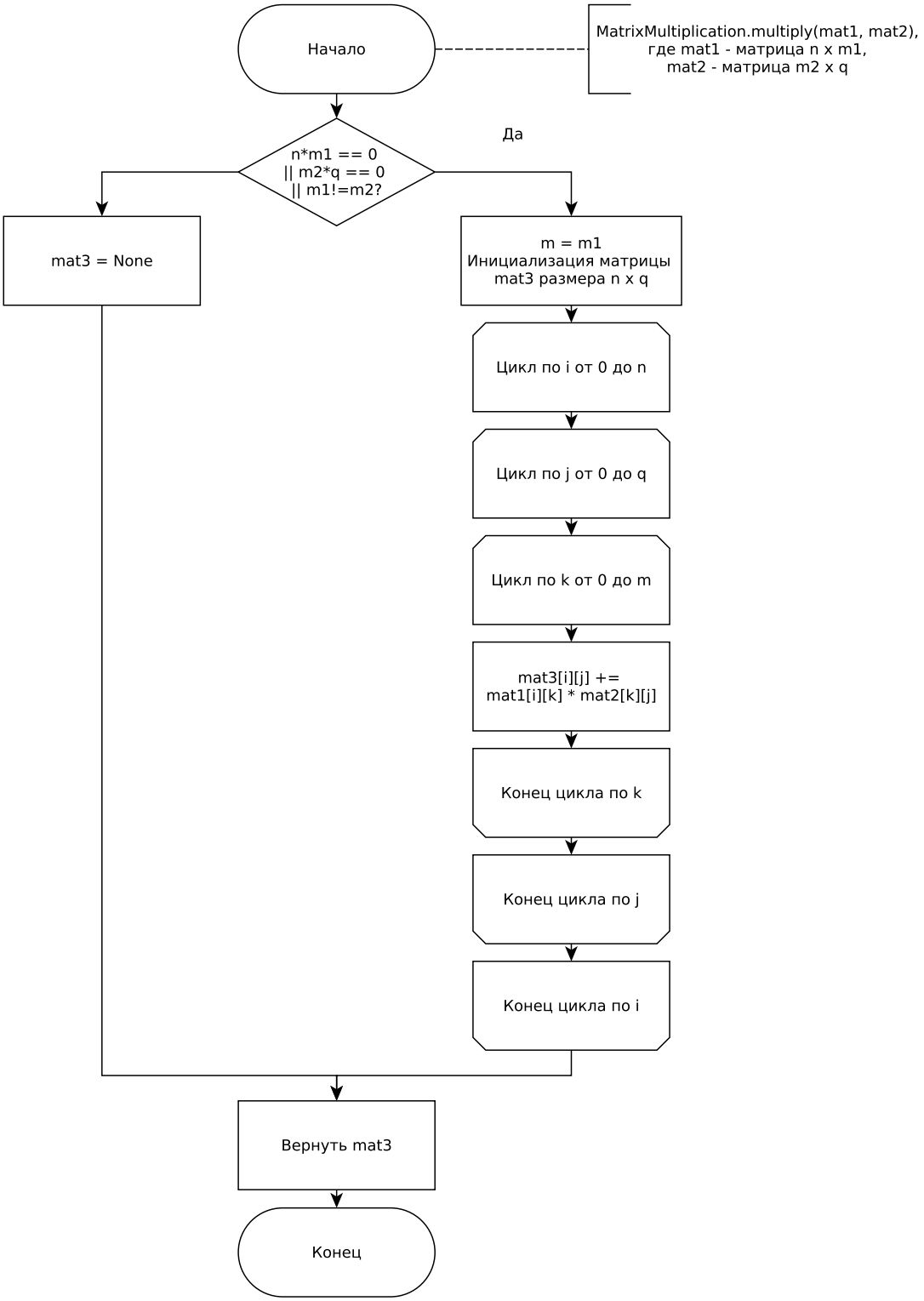

Рис. 2.1: Схема стандартного алгоритма умножения матриц

2.2.2 Схема алгоритма Копперсмита-Винограда

На рисунках 2.2-2.4 представлены схемы алгоритма Копперсмита-Винограда умножения матриц.

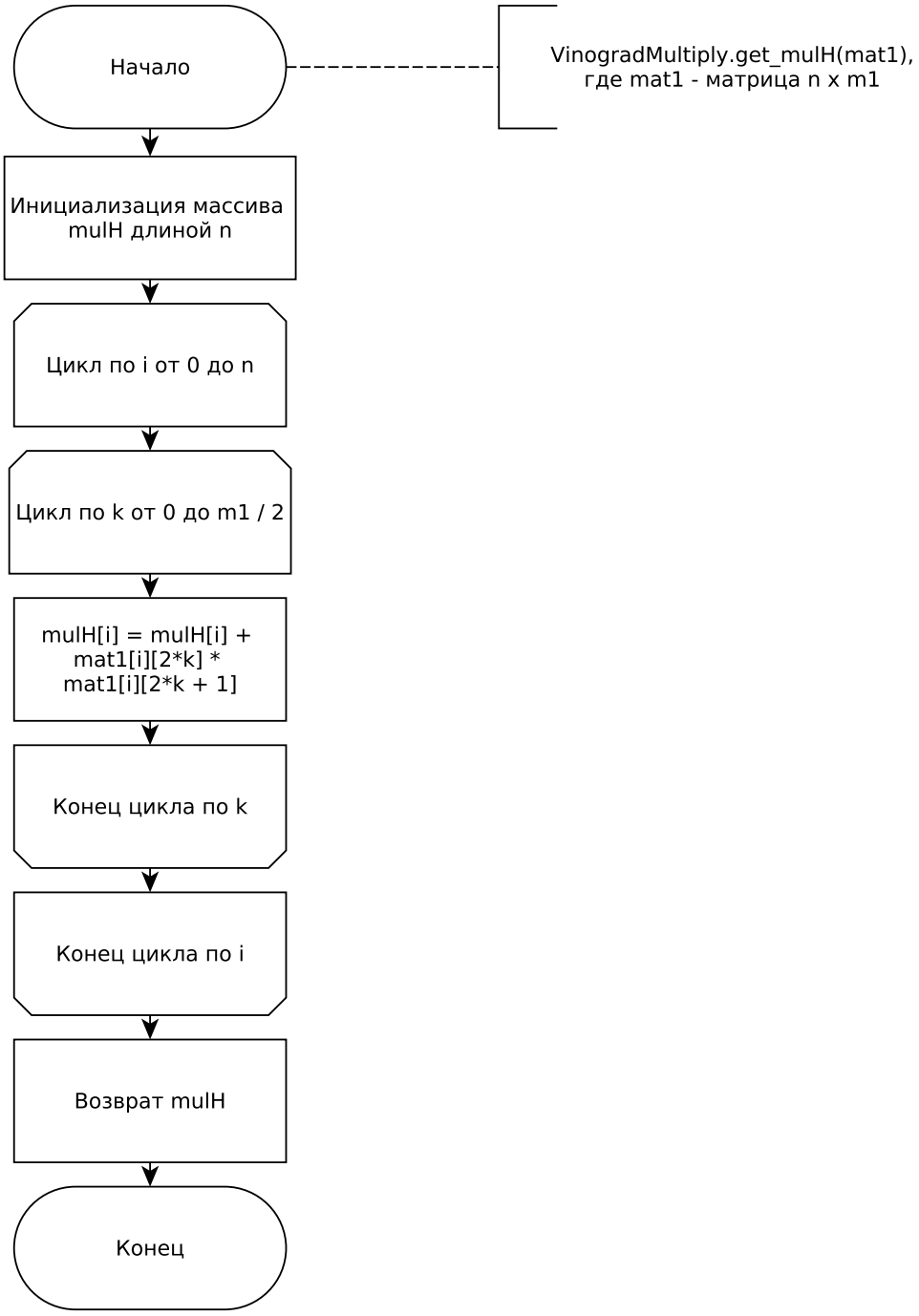

Рис. 2.2: Схема функции предварительной обработки первой матрицы для алгоритма Винограда

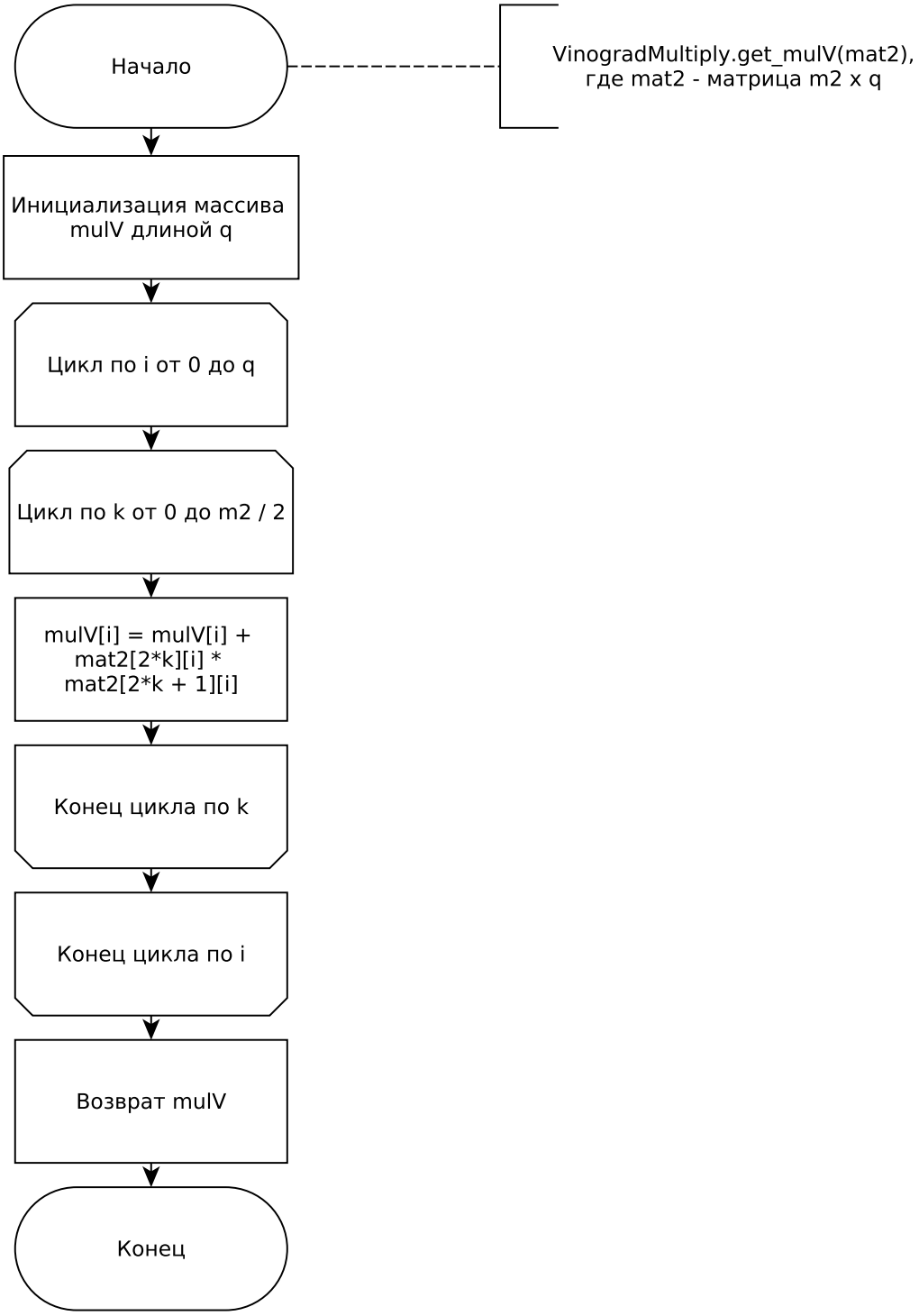

Рис. 2.3: Схема функции предварительной обработки второй матрицы для алгоритма Винограда

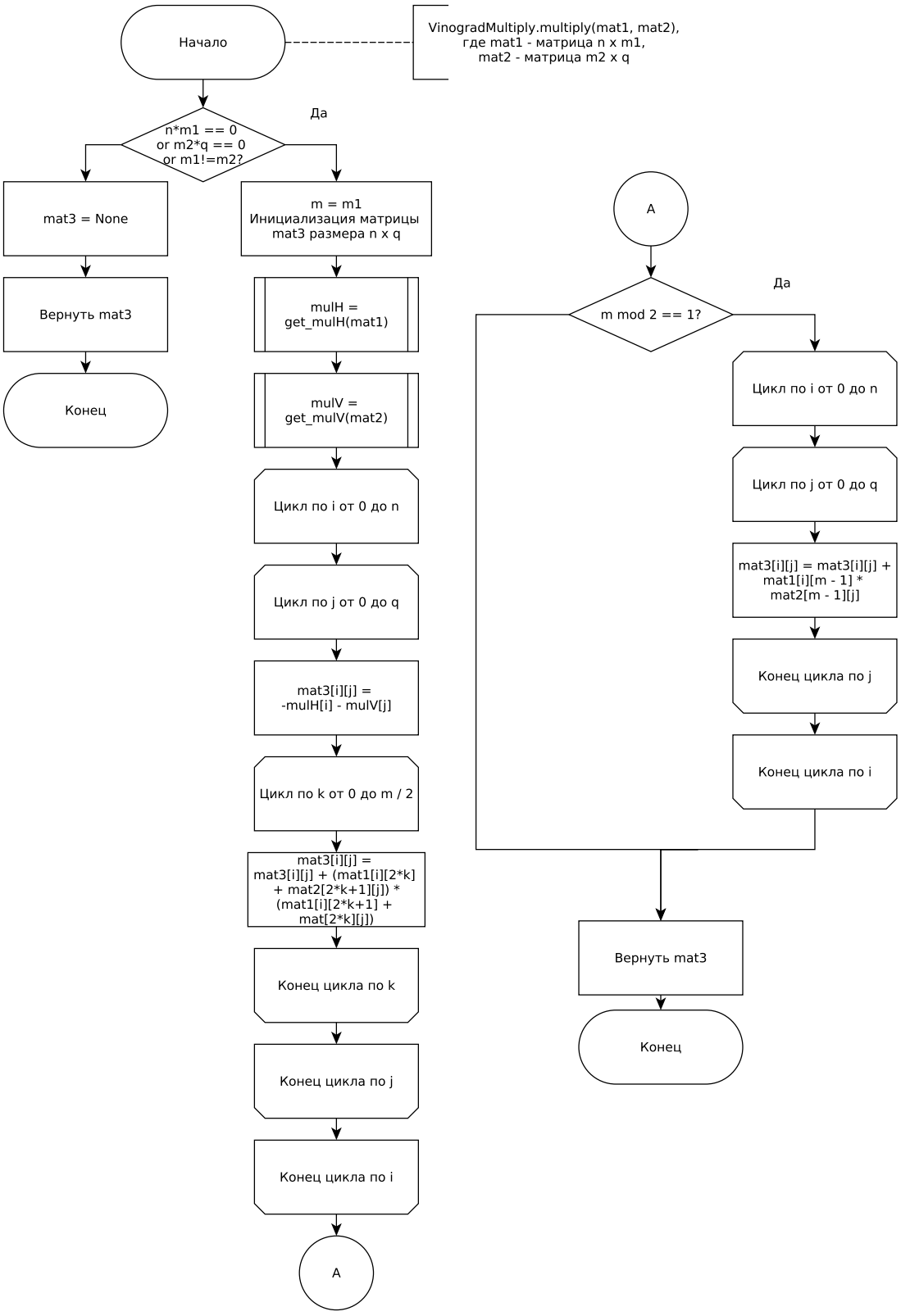

Рис. 2.4: Схема алгоритма Винограда

2.2.3 Схема оптимизированного алгоритма Копперсмита-Винограда

Оптимизации алгоритма Копперсмита-Винограда, используемые в улучшенном варианте:

- замена двух операций $a = a + \ldots$ на одну a $+= \ldots$;
- замена циклов до $n/2$ на циклы до n-1 с шагом 2;
- сокращение затрат на получение значения по индексам за счет ввода аккумулирующей промежуточное значение переменной buff;
- соединение двух циклов и сокращение затрат на содержание второго идентичного цикла для матриц нечетных размерностей.

Схема улучшенного алгоритма представлена на рисунках 2.5 - 2.7.

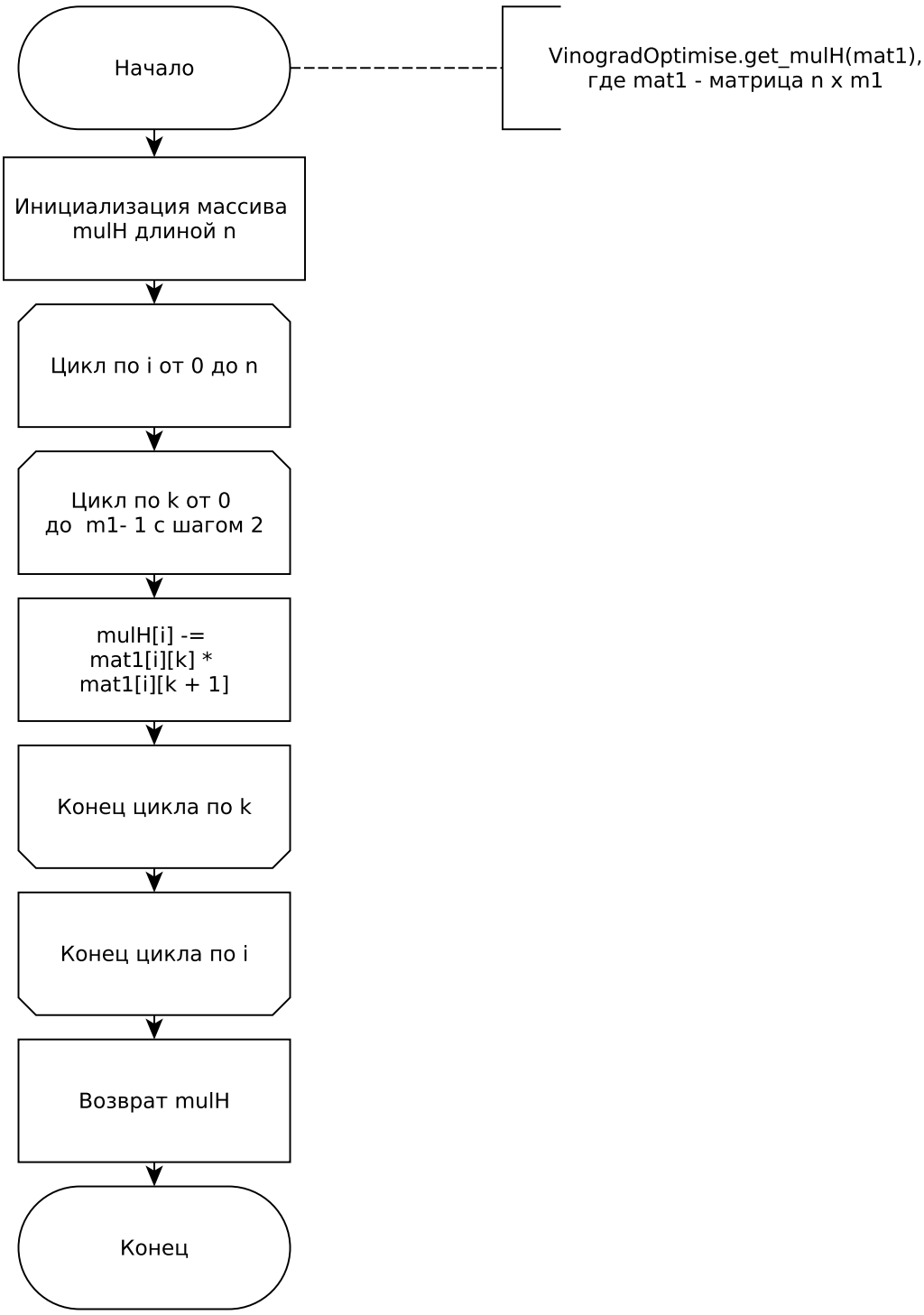

Рис. 2.5: Схема функции предварительной обработки первой матрицы для оптимизированного алгоритма Винограда

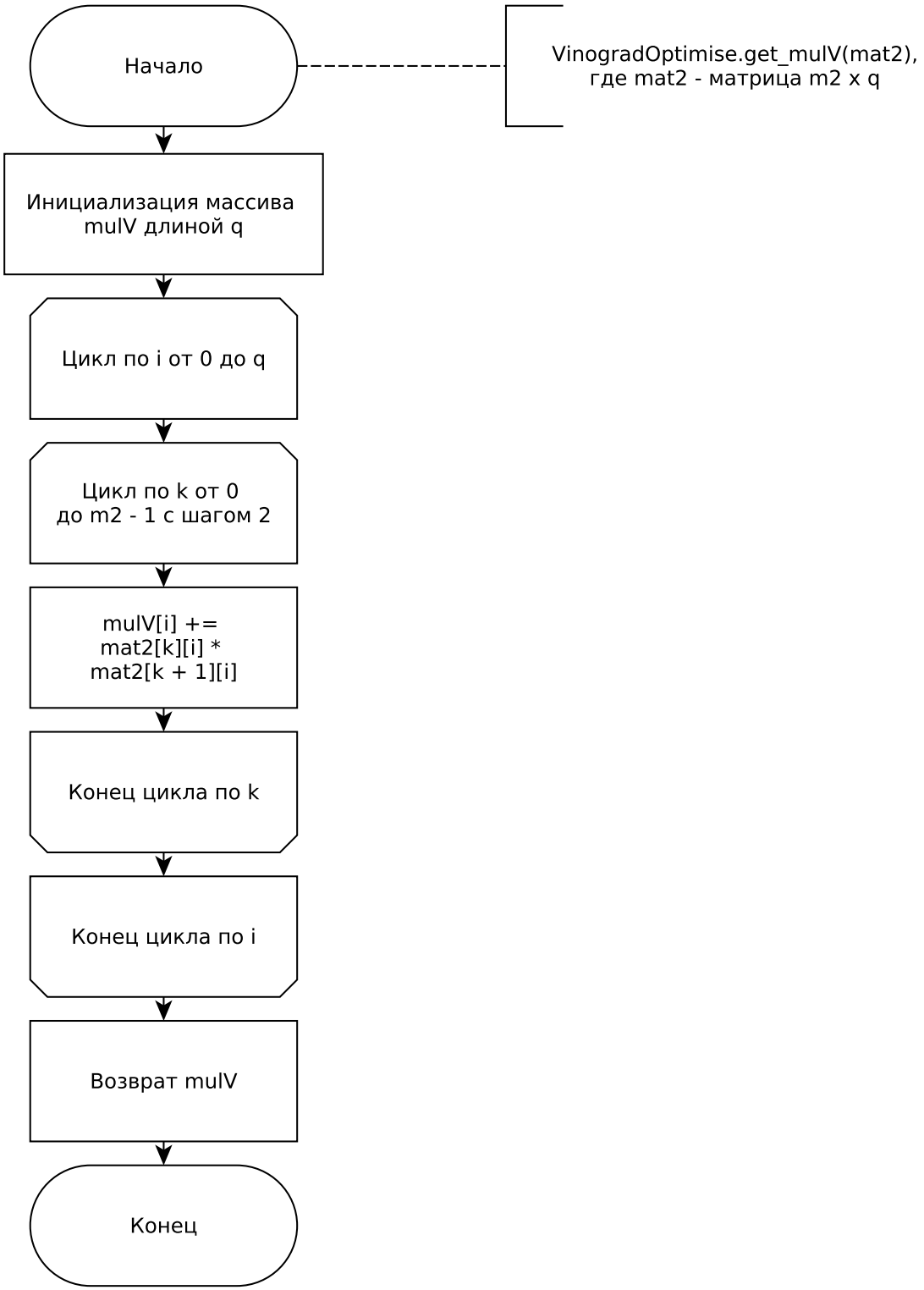

Рис. 2.6: Схема функции предварительной обработки второй матрицы для оптимизированного алгоритма Винограда

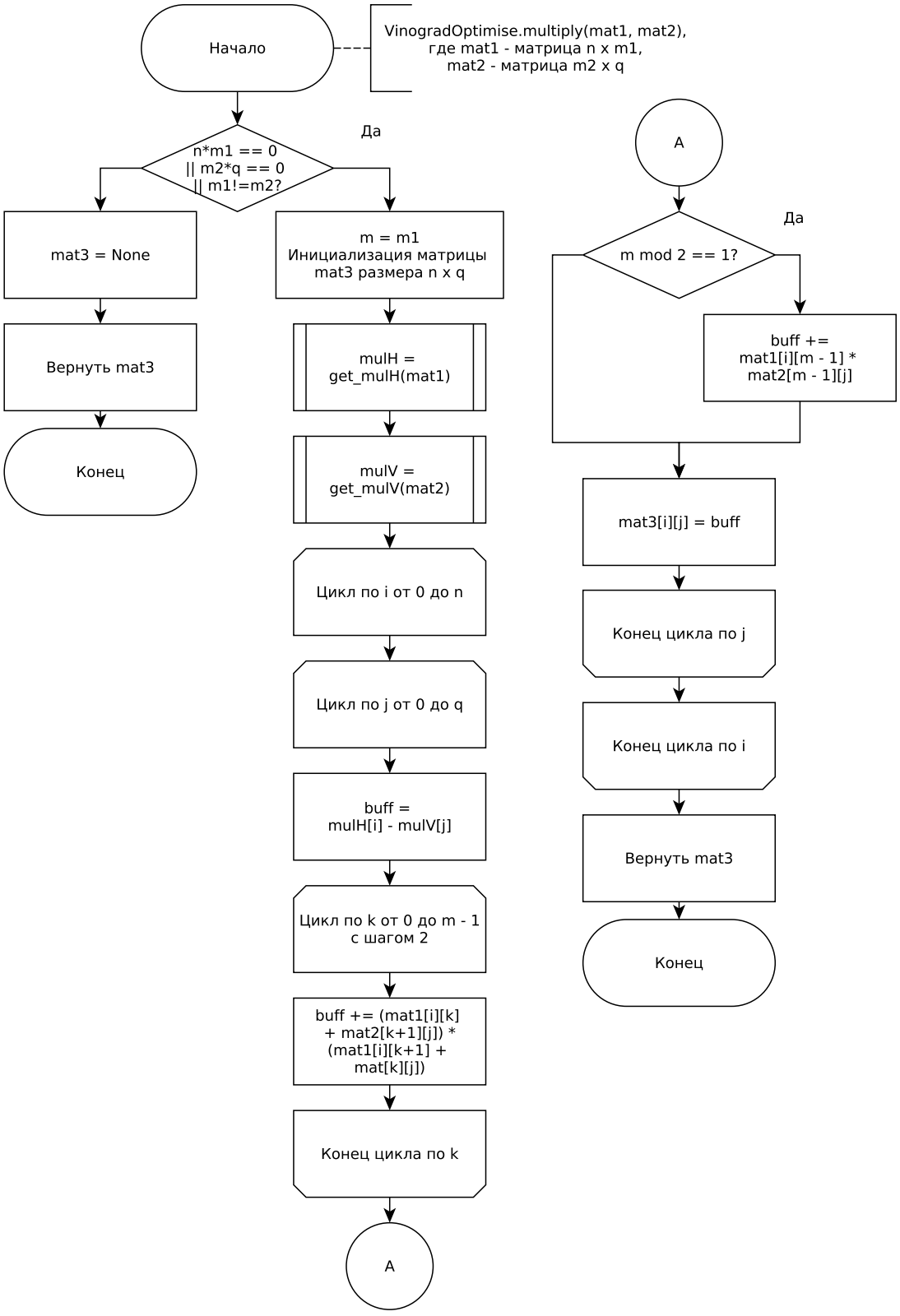

Рис. 2.7: Схема оптимизированного алгоритма Винограда

2.3 Модель вычислений

В данной работе используется следующая модель вычислений для определения трудоемкости алгоритмов:

- операции из списка 2.1 имеют трудоемкость 1;

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

- операции из списка 2.2 имеют трудоемкость 2;

$$*, /, //, \%, *=, /= \tag{2.2}$$

- трудоемкость цикла рассчитывается по формуле 2.3;

$$\begin{aligned} &f_{\text{цикла for}} = f_{\text{инициализации}} + f_{\text{сравнения}} + \\ &N_{\text{итераций}}(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \end{aligned} \tag{2.3}$$

- трудоемкость условного оператора if (if(условие) $then$ $\{A\}$ $else$ $\{B\}$)
  рассчитывается по формуле 2.4 с учетом того, что трудоемкость условного перехода равна 0.

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{лучший случай} \\ max(f_A, f_B), & \text{худший случай} \end{cases} \tag{2.4}$$

2.4 Трудоемкость алгоритмов умножения матриц

В данном подразделе представлены расчеты трудоемкости для алгоритмов за исключением затрат на проверку корректности входных данных и инициализацию результирующей матрицы.

2.4.1 Трудоемкость классического алгоритма

Трудоемкость классического алгоритма умножения матриц складывается из следующих частей:

- внешний цикл от 0 до n: $f_{fori} = 1 + 1 + n(1 + 1 + f_{\text{тела i}}) = 2 + n(2 + f_{\text{тела i}})$;
- вложенный цикл от 0 до q: $f_{for_j} = 1 + 1 + q(1 + 1 + f_{\text{тела j}}) = 2 + q(2 + f_{\text{тела j}})$;
- внутренний цикл от 0 до m: $f_{for_k} = 1 + 1 + m(1 + 1 + f_{\text{тела k}}) = 2 + m(2 + f_{\text{тела k}})$;
- Тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 6 + 1 + 2 = 9$.

Подставив вложенные циклы как тела внешних по отношению к ним, вычислим общую трудоемкость алгоритма, представленную в формуле 2.5.

$$\begin{aligned} &f_{\text{классический}} = 2 + n(2 + 2 + q(2 + 2 + m(2 + 9))) = \\ &11mnq + 4nq + 4n + 2 \approx 11mnq \end{aligned} \tag{2.5}$$

2.4.2 Трудоемкость алгоритма Копперсмита-Винограда

Трудоемкость алгоритма Винограда умножения матриц содержит следующие части:

- создание и инициализация массива mulH:
  - цикл от 0 до n: $f_{for_i} = 1 + 1 + n(1 + 1 + f_{\text{тела i}}) = 2 + n(2 + f_{\text{тела i}})$;
  - цикл от 0 до m/2: $f_{for_k} = 1 + 1 + 2 + \frac{m}{2}(1 + 1 + 2 + f_{\text{тела k}}) = 4 + \frac{m}{2}(4 + f_{\text{тела k}})$;
  - тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 6 + 1 + 1 \cdot 2 + 3 \cdot 2 = 15$;
- создание и инициализация массива mulV:
  - цикл от 0 до n: $f_{for_i} = 1 + 1 + q(1 + 1 + f_{\text{тела i}}) = 2 + q(2 + f_{\text{тела i}})$;
  - цикл от 0 до m/2: $f_{for_k} = 1 + 1 + 2 + \frac{m}{2}(1 + 1 + 2 + f_{\text{тела k}}) = 4 + \frac{m}{2}(4 + f_{\text{тела k}})$;
  - тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 6 + 1 + 1 \cdot 2 + 3 \cdot 2 = 15$;
- заполнение матрицы:
  - цикл от 0 до n: $f_{for_i} = 1 + 1 + n(1 + 1 + f_{\text{тела i}}) = 2 + n(2 + f_{\text{тела i}})$;
  - вложенный цикл от 0 до q: $f_{for_j} = 1+1+q(1+1+1\cdot 4+1+1\cdot 2+f_{for_k}) = 2+q(9+f_{for_k})$;
  - внутренний цикл от 0 до m/2: $f_{for_k} = 1+1+2+\frac{m}{2}(1+1+2+f_{\text{тела k}}) = 4+\frac{m}{2}(4+f_{\text{тела k}})$;
  - тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 12 + 1 + 1 \cdot 5 + 5 \cdot 2 = 28$;
  - проверка нечетности размера: $f_{check} = 3$;
  - цикл для дополнения умножения суммой последних нечетных строки и столбца, если общий размер нечетный (худший случай):
    - цикл от 0 до n: $f_{for_i} = 1 + 1 + n(1 + 1 + f_{\text{тела i}}) = 2 + n(2 + f_{\text{тела i}})$;
    - цикл от 0 до q: $f_{for_j} = 1 + 1 + q(1 + 1 + f_{\text{тела j}}) = 2 + q(2 + f_{\text{тела j}})$;
    - тело внутреннего цикла: $f_{\text{тела j}} = 1 \cdot 8 + 1 + 1 + 1 \cdot 2 + 2 = 14$.

Таким образом, для лучшего случая получим выражение 2.6.

$$\begin{aligned} &f = 2 + n(2 + 4 + \tfrac{m}{2}(4 + 15)) + 2 + q(2 + 4 + \tfrac{m}{2}(4 + 15)) + 2 + \\ &n(2 + 2 + q(9 + 4 + \tfrac{m}{2}(4 + 28))) + 3 = \\ &16mnq + 13nq + \tfrac{19}{2}mn + \tfrac{19}{2}mq + 10n + 6q + 9 \approx 16mnq \end{aligned} \tag{2.6}$$

Для худшего случая получим выражение 2.7.

$$\begin{aligned} &f = 2 + n(2 + 4 + \tfrac{m}{2}(4 + 15)) + 2 + q(2 + 4 + \tfrac{m}{2}(4 + 15)) + 2 + \\ &n(2 + 2 + q(9 + 4 + \tfrac{m}{2}(4 + 28))) + 3 + 2 + n(2 + 2 + q(2 + 14)) = \\ &16mnq + 29nq + \tfrac{19}{2}mn + \tfrac{19}{2}mq + 14n + 6q + 11 \approx 16mnq \end{aligned} \tag{2.7}$$

2.4.3 Трудоемкость оптимизированного алгоритма Копперсмита-Винограда

Трудоемкость оптимизированного алгоритма Винограда умножения матриц включает в себя следующие составляющие:

- создание и инициализация массива mulH:
  - цикл от 0 до n: $f_{for_i} = 1 + 1 + n(1 + 1 + f_{\text{тела i}}) = 2 + n(2 + f_{\text{тела i}})$;
  - цикл от 0 до m - 1 с шагом 2: $f_{for_k} = 1+1+1+\frac{m-1}{2}(1+1+1+f_{\text{тела k}}) = 3+\frac{m-1}{2}(3+f_{\text{тела k}})$;
  - тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 5 + 1 + 2 + 1 = 9$;
- создание и инициализация массива mulV:
  - цикл от 0 до n: $f_{for_i} = 1 + 1 + q(1 + 1 + f_{\text{тела i}}) = 2 + q(2 + f_{\text{тела i}})$;
  - цикл от 0 до m-1 с шагом 2: $f_{for_k} = 1+1+1+\frac{m-1}{2}(1+1+1+f_{\text{тела k}}) = 3+\frac{m-1}{2}(3+f_{\text{тела k}})$;
  - тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 5 + 1 + 2 + 1 = 9$;
- заполнение матрицы:
  - цикл от 0 до n: $f_{for_i} = 1 + 1 + n(1 + 1 + f_{\text{тела i}}) = 2 + n(2 + f_{\text{тела i}})$;
  - вложенный цикл от 0 до q: $f_{for_j} = 1 + 1 + q(1 + 1 + 1 + 2 \cdot 1 + 1 + f_{for_k}) = 2 + q(6 + f_{for_k} + f_{check} + f_{add} + f_{rem})$;
  - внутренний цикл от 0 до m-1 с шагом 2: $f_{for_k} = 1+1+1+\frac{m-1}{2}(1+1+1+f_{\text{тела k}}) = 3+\frac{m-1}{2}(3+f_{\text{тела k}})$;
  - тело внутреннего цикла: $f_{\text{тела k}} = 1 \cdot 8 + 1 + 4 + 2 = 15$;
  - проверка нечетности размера: $f_{check} = 3$;
  - дополнение умножения суммой последних нечетных строки и столбца, если общий размер нечетный (худший случай): $f_{\text{add}} = 1 \cdot 4 + 1 + 1 \cdot 2 + 2 = 9$;
  - запоминание промежуточного значения: $f_{rem} = 1 \cdot 2 + 1 = 3$.

Основываясь на приведенных выше выражениях, трудоемкость лучшего случая оптимизированного алгоритма Копперсмита-Винограда выразим формулой 2.8.

$$\begin{aligned} &f = 2 + n(2 + 3 + \tfrac{m-1}{2}(3 + 9)) + 2 + q(2 + 3 + \tfrac{m-1}{2}(3 + 9)) + 2 + \\ &n(2 + 2 + q(6 + 3 + \tfrac{m-1}{2}(3 + 15) + 3 + 3)) = \\ &9mnq + 6nq + 6mq + 6mn + 3n - q + 6 \approx 9mnq \end{aligned} \tag{2.8}$$

Трудоемкость худшего случая вычисляется по формуле 2.9.

$$\begin{aligned} &f = 2 + n(2 + 3 + \tfrac{m-1}{2}(3 + 9)) + 2 + q(2 + 3 + \tfrac{m-1}{2}(3 + 9)) + 2 + \\ &n(2 + 2 + q(6 + 3 + \tfrac{m-1}{2}(3 + 15) + 3 + 3 + 9)) = \\ &9mnq + 15nq + 6mq + 6mn + 3n - q + 6 \approx 9mnq \end{aligned} \tag{2.9}$$

2.5 Вывод

В данном разделе на основе приведенных в аналитическом разделе теоретических данных были составлены схемы алгоритмов для реализации в технологической части, вычислена трудоемкость лучшего и худшего случаев алгоритмов. Приблизительная трудоемкость классического алгоритма умножения матриц равна $11mnq$, алгоритма Копперсмита-Винограда - $16mnq$, оптимизированного алгоритма Копперсмита-Винограда - $9mnq$.

3 Технологическая часть

Данный раздел содержит обоснование выбора языка и среды разработки, реализацию алгоритмов.

3.1 Средства реализации

Для реализации программы был выбран язык программирования Java [4]. Такой выбор обусловлен следующими причинами:

- удобный и понятный синтаксис;
- обладает информативной документацией.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.2 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Классический алгоритм

```
1  public class MatrixClassicMultiplier implements MatrixMultiplier {
2      @Override
3      public int[][] multiply(int[][] m1, int[][] m2) {
4          if (m1 == null || m2 == null) {
5              return null;
6          }
7
8          if (m1.length != m2[0].length) {
9              return null;
10         }
11
12         int rows1 = m1.length;
13         int rows2 = m2.length;
14         int cols1 = m1[0].length;
15         int cols2 = m2[0].length;
16
17         int[][] result = new int[rows1][cols2];
18
19         for (int i = 0; i < rows1; i++) {
20             for (int k = 0; k < cols2; k++) {
21                 for (int m = 0; m < cols1; m++) {
```

```
                    result[i][k] += m1[i][m] * m2[m][k];
                }
            }
        }

        return result;
    }
}
```

Листинг 3.2: Алгоритм Копперсмита-Винограда

```
    public class MatrixVinogradMultiplier implements MatrixMultiplier {
    @Override
    public int[][] multiply(int[][] m1, int[][] m2) {
        if (m1 == null || m2 == null) {
            return null;
        }

        if (m1.length != m2[0].length) {
            return null;
        }

        int rows1 = m1.length;
        int rows2 = m2.length;
        int cols1 = m1[0].length;
        int cols2 = m2[0].length;

        int[][] result = new int[rows1][cols2];

        int[] rowFactor = new int[rows1];
        int[] colFactor = new int[cols2];

        for (int i = 0; i < rows1; i++) {
            for (int k = 0; k < cols1 / 2; k++) {
                rowFactor[i] = rowFactor[i] + m1[i][2 * k] * m1[i][2 * k +
                    1];
            }
        }

        for (int i = 0; i < cols2; i++) {
            for (int k = 0; k < rows2 / 2; k++) {
                colFactor[i] = colFactor[i] + m2[2 * k][i] * m2[2 * k + 1][i
                    ];
            }
        }

        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                result[i][j] = -rowFactor[i] - colFactor[j];
                for (int k = 0; k < cols1 / 2; k++) {
```

```
38                      result[i][j] = result[i][j] + (m1[i][2 * k] + m2[2 * k +
                            1][j]) *
39                              (m1[i][2 * k + 1] + m2[2 * k][j]);
40                  }
41              }
42          }
43
44          if (cols1 % 2 == 1) {
45              for (int i = 0; i < rows1; i++) {
46                  for (int k = 0; k < cols2; k++) {
47                      result[i][k] = result[i][k] + m1[i][cols1 - 1] * m2[
                            cols1 - 1][k];
48                  }
49              }
50          }
51
52          return result;
53      }
54  }
```

Листинг 3.3: Оптимизированный алгоритм Копперсмита-Винограда

```
1   public class MatrixVinogradOptimisedMultiplier implements MatrixMultiplier {
2       @Override
3       public int[][] multiply(int[][] m1, int[][] m2) {
4           if (m1 == null || m2 == null) {
5               return null;
6           }
7
8           if (m1.length != m2[0].length) {
9               return null;
10          }
11
12          int rows1 = m1.length;
13          int rows2 = m2.length;
14          int cols1 = m1[0].length;
15          int cols2 = m2[0].length;
16
17          int[][] result = new int[rows1][cols2];
18
19          int[] rowFactor = new int[rows1];
20          int[] colFactor = new int[cols2];
21
22          for (int i = 0; i < rows1; i++) {
23              for (int k = 0; k < cols1 - 1; k += 2) {
24                  rowFactor[i] -= m1[i][k] * m1[i][k + 1];
25              }
26          }
27
28          for (int i = 0; i < cols2; i++) {
```

```
29              for (int k = 0; k < rows2 - 1; k += 2) {
30                  colFactor[i] += m2[k][i] * m2[k + 1][i];
31              }
32          }
33
34          for (int i = 0; i < rows1; i++) {
35              for (int j = 0; j < cols2; j++) {
36                  int buffer = rowFactor[i] - colFactor[j];
37
38                  for (int k = 0; k < cols1 - 1; k += 2) {
39                      buffer += (m1[i][2 * k] + m2[2 * k + 1][j]) *
40                              (m1[i][2 * k + 1] + m2[2 * k][j]);
41                  }
42
43                  if (cols1 % 2 == 1) {
44                      buffer += m1[i][cols1 - 1] * m2[cols1 - 1][j];
45                  }
46
47                  result[i][j] = buffer;
48              }
49          }
50
51          return result;
52      }
53  }
```

3.3 Тестирование

В таблице 3.1 представлены использованные для тестирования методом "черного ящика"данные, были рассмотрены все возможные тестовые случаи. Все тесты пройдены успешно.

3.4 Вывод

В данном разделе были реализованы и протестированы алгоритмы умножения матриц: классический, Копперсмита-Винограда и оптимизированный Копперсмита-Винограда.

Таблица 3.1: Проведенные тесты

Матрица 1	Строка 1	Ожидаемый результат
$\begin{pmatrix} 1 & 5 & 2 \\ 1 & 2 & 8 \\ 1 & 3 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 4 & 9 \\ 8 & 8 & 8 \\ 12 & 21 & 13 \end{pmatrix}$	$\begin{pmatrix} 65 & 86 & 75 \\ 113 & 188 & 129 \\ 49 & 70 & 59 \end{pmatrix}$
$\begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \end{pmatrix}$	$\begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 50 \\ 32 \end{pmatrix}$
$\begin{pmatrix} 12 \end{pmatrix}$	$\begin{pmatrix} 17 \end{pmatrix}$	$\begin{pmatrix} 204 \end{pmatrix}$
$\begin{pmatrix} -1 & -2 & 3 \\ 8 & -9 & 7 \\ -4 & -7 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & -9 \end{pmatrix}$	$\begin{pmatrix} 12 & 12 & -42 \\ 21 & 27 & -93 \\ 3 & -3 & -99 \end{pmatrix}$
$\begin{pmatrix} 8 & 7 \end{pmatrix}$	$\begin{pmatrix} 4 & 2 \end{pmatrix}$	Ошибка
$\begin{pmatrix} 1 & -1 & 2 \\ -4 & 7 & -5 \end{pmatrix}$	$\begin{pmatrix} 1 & 8 & 3 \\ 6 & -9 & 7 \\ 1 & 4 & 8 \end{pmatrix}$	$\begin{pmatrix} -3 & 25 & 12 \\ 33 & -115 & -3 \end{pmatrix}$
$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 8 & 6 \\ 3 & 5 & -4 \\ 6 & 6 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 4 \\ 8 & -3 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & -9 & 5 \\ -8 & 7 & 9 & 0 \\ 7 & 9 & -1 & 1 \\ 8 & 5 & 2 & 8 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & -9 & 5 \\ -8 & 7 & 9 & 0 \\ 7 & 9 & -1 & 1 \\ 8 & 5 & 2 & 8 \end{pmatrix}$
$\begin{pmatrix} 7 & 1 & 3 \\ -1 & -1 & -1 \\ 3 & 5 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 7 & 1 & 3 \\ -1 & -1 & -1 \\ 3 & 5 & 2 \end{pmatrix}$

4 Экспериментальная часть

В данном разделе сравниваются реализованные алгоритмы, дается сравнительная оценка затрат на память и время.

4.1 Пример работы программы

Пример работы программы представлен на рисунках 4.1-??.

```
Введите первую матрицу
Количество строк:
4
Количество столбцов:
4
Введите числа из строки №0
1 0 0 0
Введите числа из строки №1
0 1 0 0
Введите числа из строки №2
0 0 1 0
Введите числа из строки №3
0 0 0 3
Введите вторую матрицу
Количество строк:
4
Количество столбцов:
4
Введите числа из строки №0
4 8 -9 5
Введите числа из строки №1
-8 7 9 0
Введите числа из строки №2
7 9 -1 1
Введите числа из строки №3
8 5 2 8
Классический алгоритм
    Ответ: [[4, 8, -9, 5], [-8, 7, 9, 0], [7, 9, -1, 1], [24, 15, 6, 24]]
Алгоритм Винограда
    Ответ: [[4, 8, -9, 5], [-8, 7, 9, 0], [7, 9, -1, 1], [24, 15, 6, 24]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[4, 8, -9, 5], [-8, 7, 9, 0], [7, 9, -1, 1], [24, 15, 6, 24]]

Process finished with exit code 0
```

Рис. 4.1: Ввод данных

```
----------------------------------------
I матрица: [[1, 5, 2], [1, 2, 8], [1, 3, 2]]
II матрица: [[1, 4, 9], [8, 8, 8], [12, 21, 13]]
Классический алгоритм
    Ответ: [[65, 86, 75], [113, 188, 129], [49, 70, 59]]
Алгоритм Винограда
    Ответ: [[65, 86, 75], [113, 188, 129], [49, 70, 59]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[65, 86, 75], [113, 188, 129], [49, 70, 59]]
----------------------------------------
I матрица: [[9, 8, 7], [6, 5, 4]]
II матрица: [[3], [2], [1]]
Классический алгоритм
    Ответ: [[50], [32]]
Алгоритм Винограда
    Ответ: [[50], [32]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[50], [32]]
----------------------------------------
I матрица: [[12]]
II матрица: [[17]]
Классический алгоритм
    Ответ: [[204]]
Алгоритм Винограда
    Ответ: [[204]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[204]]
----------------------------------------
I матрица: [[-1, -2, 3], [8, -9, 7], [-4, -7, 5]]
II матрица: [[1, 2, 3], [4, 5, 6], [7, 8, -9]]
Классический алгоритм
    Ответ: [[12, 12, -42], [21, 27, -93], [3, -3, -99]]
```

Рис. 4.2: Результат работы программы

```
----------------------------------------
I матрица: [[1, 5, 2], [1, 2, 8], [1, 3, 2]]
II матрица: [[1, 4, 9], [8, 8, 8], [12, 21, 13]]
Классический алгоритм
    Ответ: [[65, 86, 75], [113, 188, 129], [49, 70, 59]]
Алгоритм Винограда
    Ответ: [[65, 86, 75], [113, 188, 129], [49, 70, 59]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[65, 86, 75], [113, 188, 129], [49, 70, 59]]
----------------------------------------
I матрица: [[9, 8, 7], [6, 5, 4]]
II матрица: [[3], [2], [1]]
Классический алгоритм
    Ответ: [[50], [32]]
Алгоритм Винограда
    Ответ: [[50], [32]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[50], [32]]
----------------------------------------
I матрица: [[12]]
II матрица: [[17]]
Классический алгоритм
    Ответ: [[204]]
Алгоритм Винограда
    Ответ: [[204]]
Алгоритм Винограда (оптимизированный)
    Ответ: [[204]]
----------------------------------------
I матрица: [[-1, -2, 3], [8, -9, 7], [-4, -7, 5]]
II матрица: [[1, 2, 3], [4, 5, 6], [7, 8, -9]]
Классический алгоритм
    Ответ: [[12, 12, -42], [21, 27, -93], [3, -3, -99]]
```

Рис. 4.3: Результат работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система — Windows [5] 10 64-bit;
- оперативная память — 4 GB (для Java);
- процессор — Intel(R) Core(TM) i5-7600 CPU @ 3.50GHz [6]

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось с помощью Java-интерфейса для управления потоками виртуальной машины Java - ThreadMXBean [7], позволяющего вычислить процессорное время, затраченное на определенный процесс. Процессорное время находилось как среднее арифметическое из нескольких разовых замеров процессорного времени (1000000 замеров).

Усредненные результаты замеров процессорного времени приведены в таблице 4.1. Используемые обозначения: "Классич." - классический алгоритм умножения матриц, "Виноград" - алгоритм Копперсмита-Винограда, "Опт.Виноград" - оптимизированный алгоритм Копперсмита-Винограда.

Таблица 4.1: Время работы алгоритмов

Размер	Классич.	Виноград	Опт.Виноград
10	2265	2515	2234
11	2640	2906	2640
30	25218	28468	25331
31	27015	30343	27390
50	128906	130312	116562
51	129375	139687	108750
70	335625	345000	315781
71	339062	367031	301406

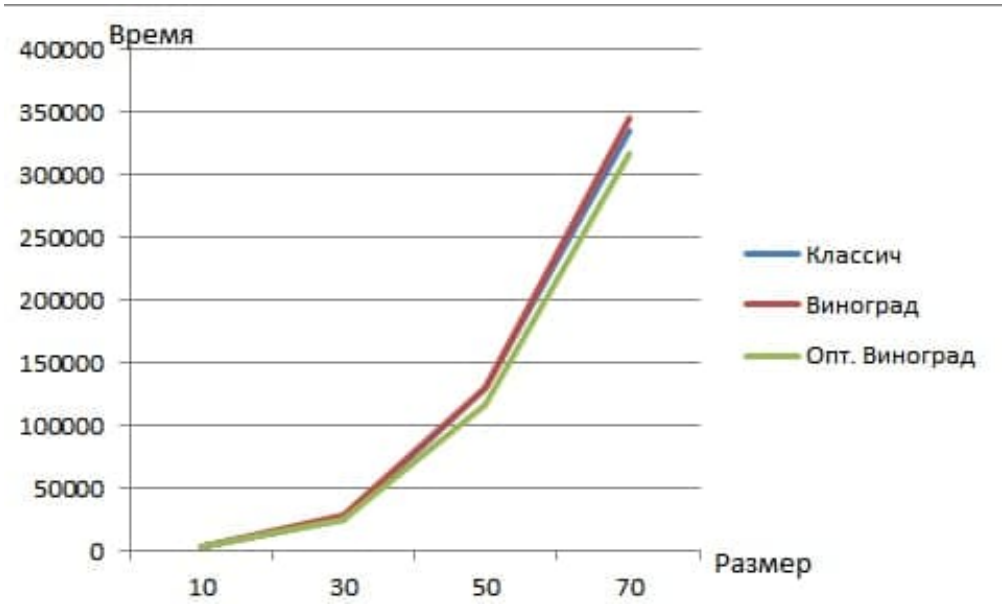

Рис. 4.4: Сравнение времени работы алгоритмов умножения квадратных матриц четных размеров

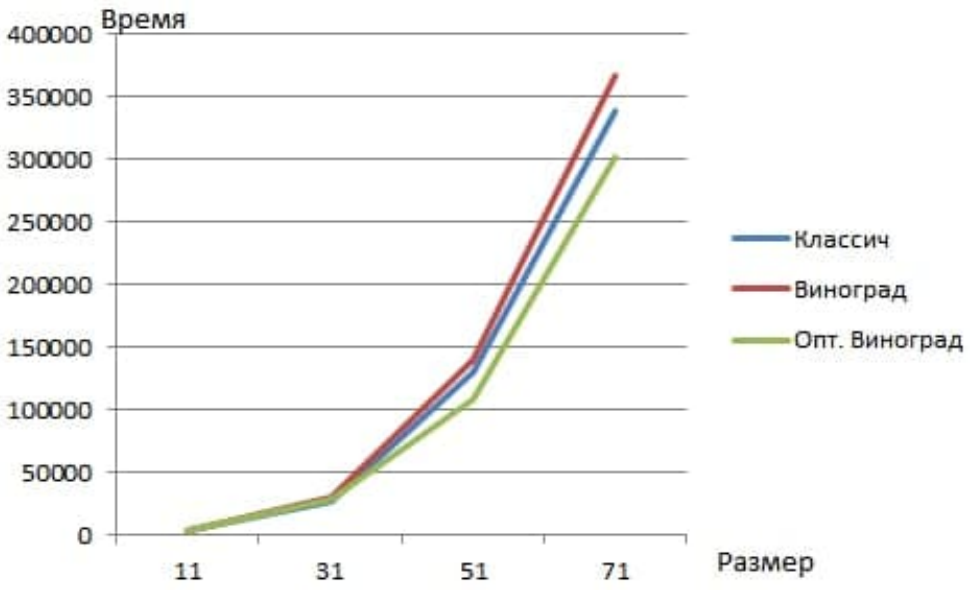

Рис. 4.5: Сравнение времени работы алгоритмов умножения квадратных матриц нечетных размеров

4.4 Вывод

По результатам проведенных замеров видно, что оптимизированный алгоритм Винограда работает в 1.2-1.5 раза быстрее классического алгоритма умножения матриц с увеличением этого коэффициента при увеличении размера матрицы. Важно заметить, что коэффициент незначительно меньше для матриц нечетных размеров, что говорит о более медленной работе оптимизированного алгоритма Винограда для таких матриц, в то время как классический алгоритм зависимости от четности размеров не имеет.

Обычный алгоритм Винограда на четных матрицах меньше 50x50 работает медленнее классического алгоритма, на больших - практически одинаково, разрыв составляет не более 1.05 раз. В случае с нечетным размером матрицы алгоритм Винограда проигрывает классическому алгоритму, но при увеличении размерности время работы обоих алгоритмов сопоставимо.

Заключение

В процессе выполнения лабораторной работы были изучены и реализованы классический алгоритм умножения и алгоритм Копперсмита-Винограда матриц, оптимизирован алгоритм Винограда.

Согласно проведенному анализу трудоемкости алгоритмов в соответствии с выбранной моделью вычислений, трудоемкость классического алгоритма составила приблизительно $11mnq$, алгоритма Копперсмита-Винограда - $16mnq$, оптимизированного алгоритма Копперсмита-Винограда - $9mnq$.

Было исследовано процессорное время выполнения выше обозначенных алгоритмов. В результате было выявлено, что на матрицах с количеством элементов в строках и столбцах, меньших 50, дольше всего работает алгоритм Копперсмита-Винограда, на больших - классический, причем время работы алгоритма Винограда незначительно меньше (разница не превышает 1.04 раз). Быстрее всего работает оптимизированный алгоритм Копперсмита-Винограда (в 1.2-1.5 раз быстрее других алгоритмов с увеличением разницы во времени работы с увеличением размеров матриц), однако заметна небольшая деградация времени работы для матриц с нечетным количеством строк и столбцов (как и у алгоритма Винограда-Копперсмита), тогда как классический алгоритм таким свойством не обладает.

Литература

[1] Coppersmith D., Winograd S. Matrix multiplication via arithmetic progressions // Journal of Symbolic Computation. 1990. no. 9. P. 251–280.

[2] Robinson S. Toward an Optimal Algorithm for Matrix Multiplication // SIAM News. 2005. November. Vol. 38, no. 9.

[3] Strassen V. Gaussian Elimination is not Optimal // Numerische Mathematik. 2005. Vol. 13, no. 9. P. 354–356.

[4] Что такое технология Java и каково ее применение? [Электронный ресурс]. Режим доступа: https://www.java.com/ru/download/help/whatis-java.html. Дата обращения: 21.09.2021.

[5] Сравните выпуски Windows 10 [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/windows/compare-windows-10-home-vs-pro. Дата обращения: 21.09.2021.

[6] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/97150/intel-core-i57600-processor-6m-cache-up-to-4-10-ghz.html. Дата обращения: 21.09.2021.

[7] Interface ThreadMXBean [Электронный ресурс]. Режим доступа: https://docs.oracle.com/javase/7/docs/api/java/lang/management/ThreadMXBean.html. Дата обращения: 21.09.2021.