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 по дисциплине анализ алгоритмов

Тема: Алгоритмы умножения матриц

Студент: Серова М.Н.

Группа: ИУ7-55Б

Оценка (баллы): _____

Преподаватель: Волкова Л.Л.

Москва, 2021

Содержание

Введение

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 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 содержит схемы алгоритмов, реализуемых в работе (классический, Копперсмита-Винограда и оптимизированный Копперсмита-Винограда), а также теоретические вычисления трудоемкости для каждого из них.

2.1 Схемы алгоритмов

В данном пункте раздела представлены схемы реализуемых в работе алгоритмов.

2.1.1 Схема классического алгоритма

На рисунке 2.1 представлена схема обычного алгоритма умножения матриц.

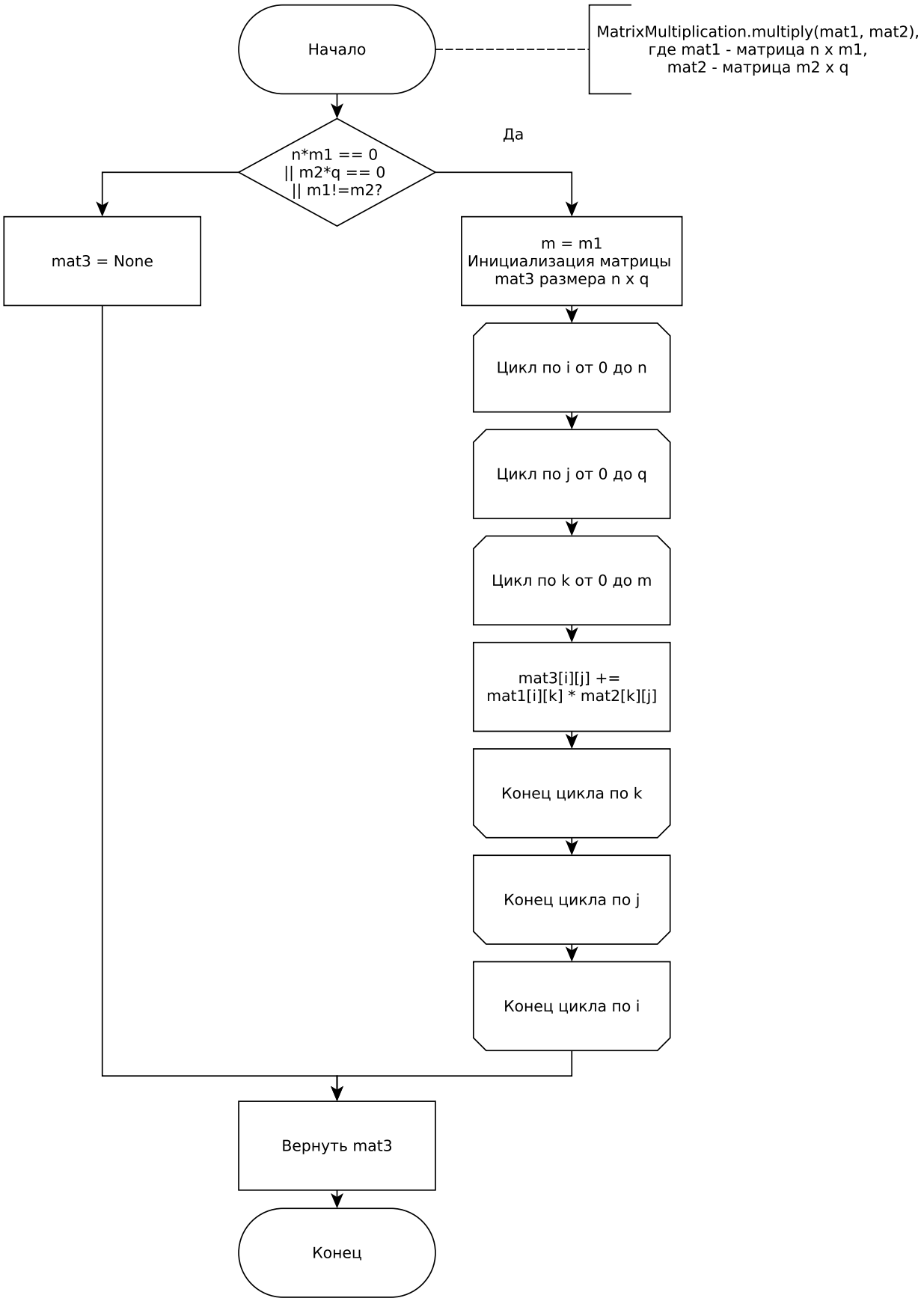

Рис. 2.1: Схема стандартного алгоритма умножения матриц

2.1.2 Схема алгоритма Копперсмита-Винограда

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

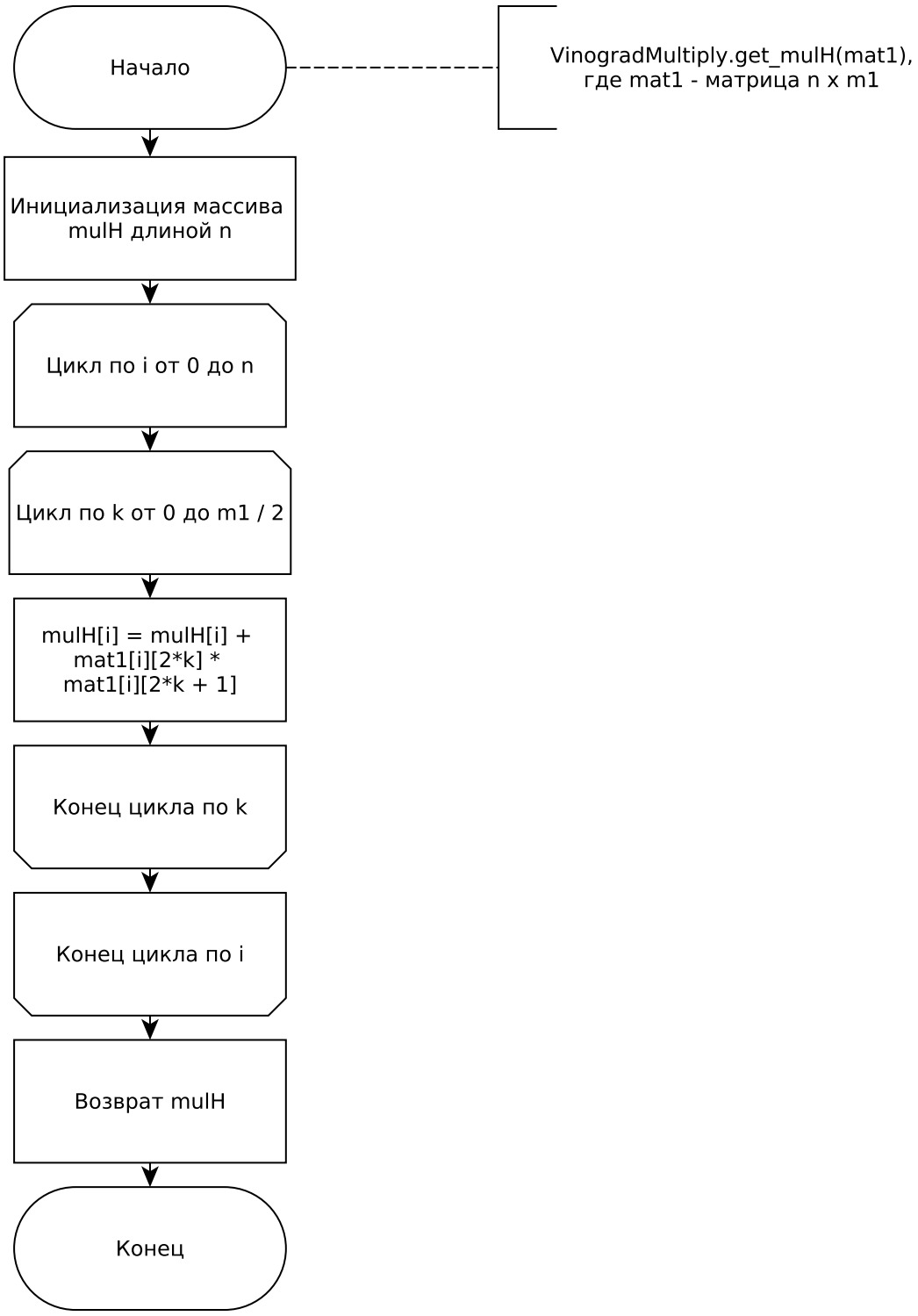

Рис. 2.2: Схема функции предварительной обработки первой матрицы для алгоритма Винограда

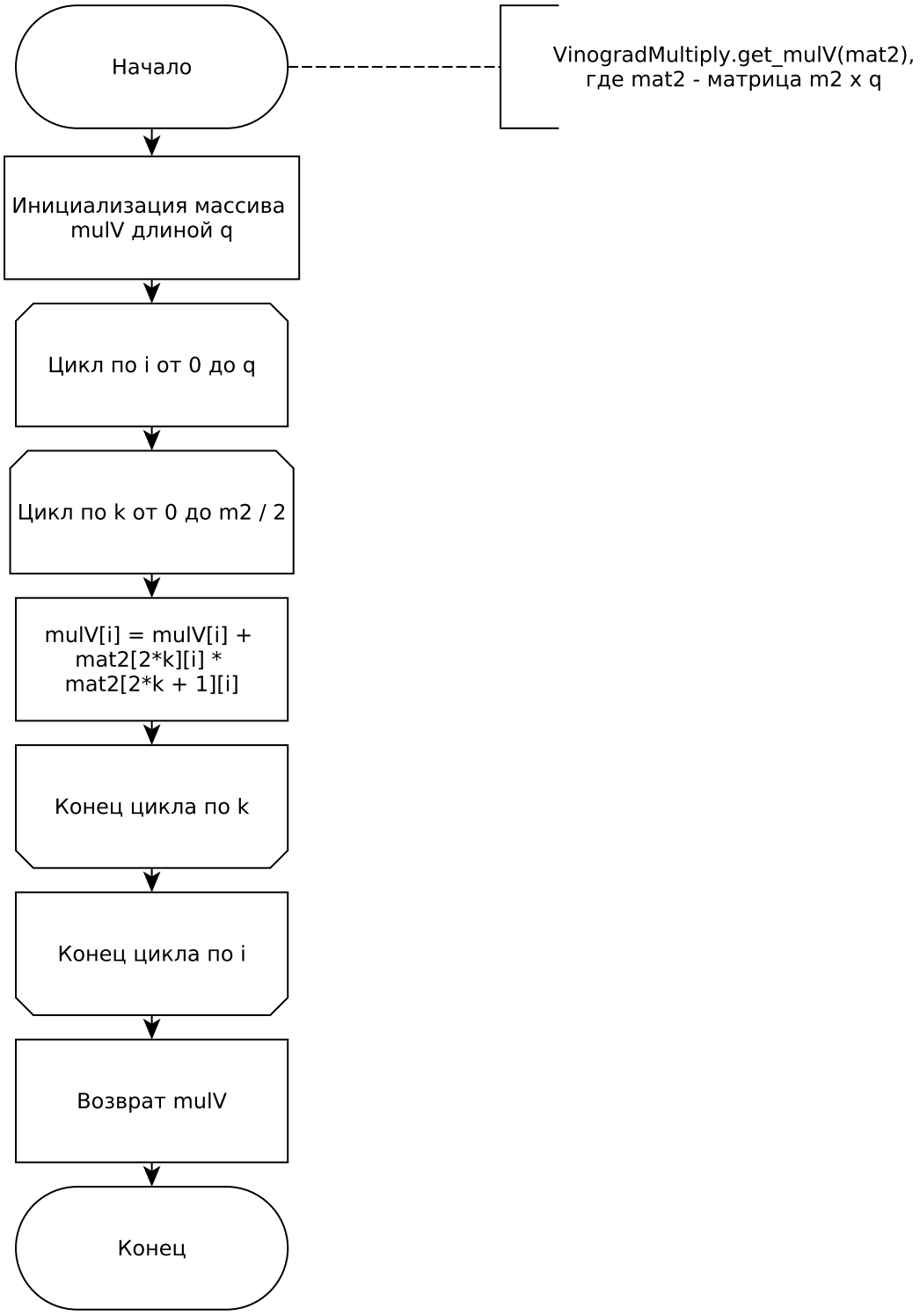

Рис. 2.3: Схема функции предварительной обработки второй матрицы для алгоритма Винограда

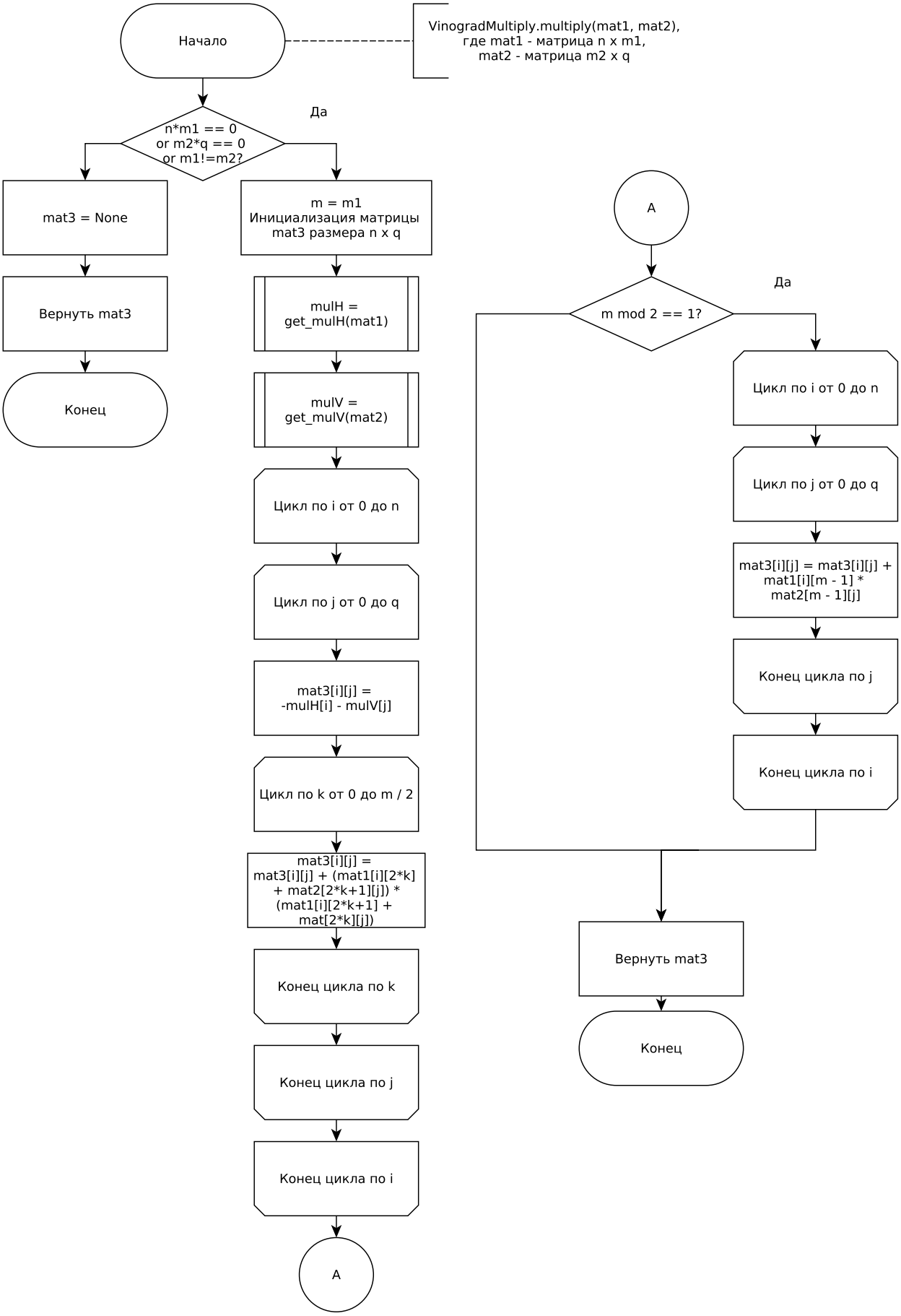

Рис. 2.4: Схема алгоритма Винограда

2.1.3 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.

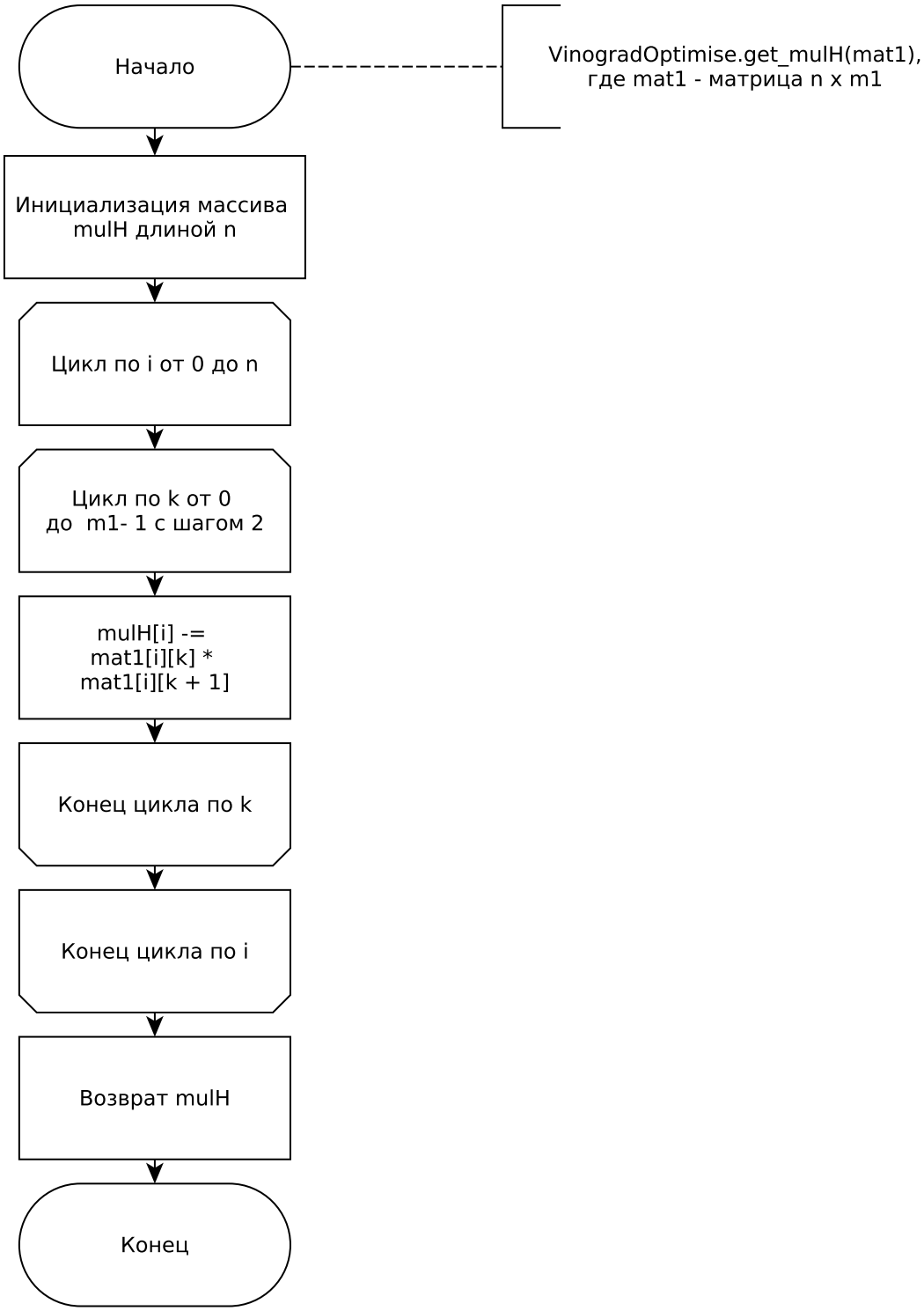

Рис. 2.5: Схема функции предварительной обработки первой матрицы для оптимизированного алгоритма Винограда

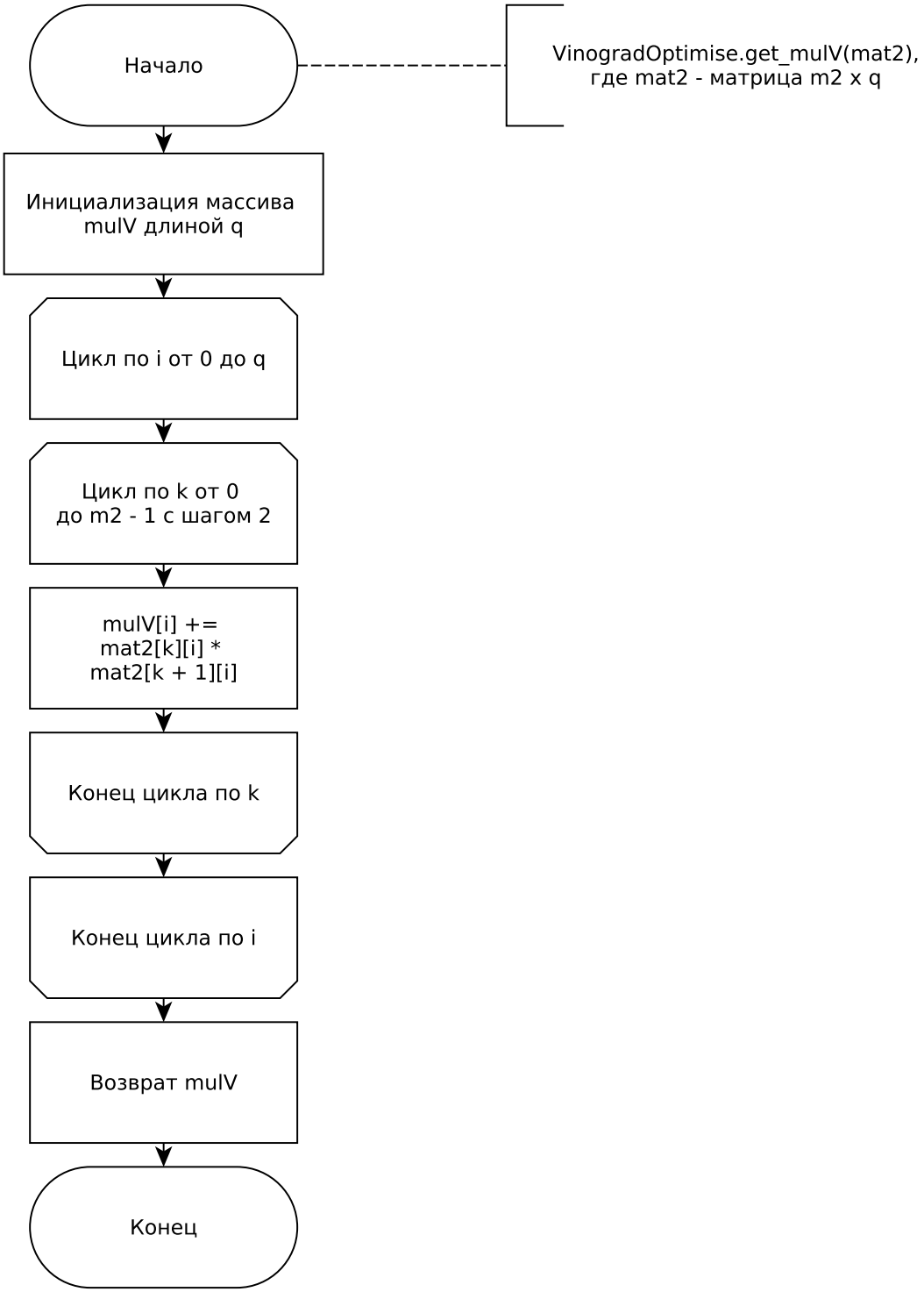

Рис. 2.6: Схема функции предварительной обработки второй матрицы для оптимизированного алгоритма Винограда

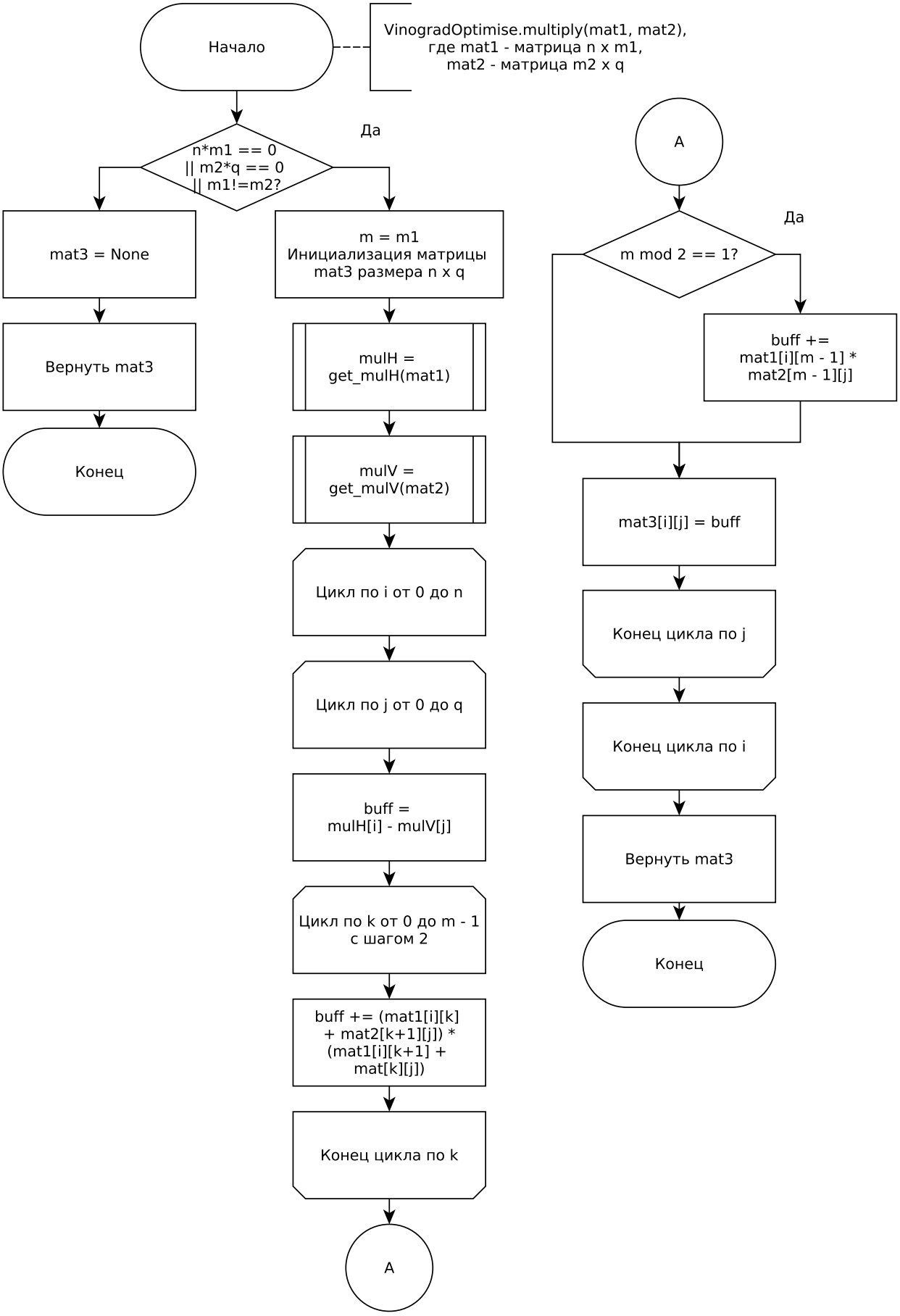

Рис. 2.7: Схема оптимизированного алгоритма Винограда

2.2 Модель вычислений

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

- операции из списка 2.1 имеют трудоемкость 1;

$$+, -, + =, - =, =, ==, ! =, !, \&\&, ||, <, >, <<, >>, <=, >=, [] \quad (2.1)$$

- операции из списка 2.2 имеют трудоемкость 2;

$$*, /, //, \%, * =, / = \quad (2.2)$$

- трудоемкость цикла рассчитывается по формуле 2.3;

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

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

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

2.3 Трудоемкость алгоритмов умножения матриц

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

2.3.1 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.

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

2.3.2 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: $f_{for_j} = 1+1+q(1+1+1\cdot4+1+1\cdot2+f_{for_k}) = 2 + q(9 + f_{for_k})$;
  - 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.

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

2.3.3 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: $f_{for_j} = 1+1+q(1+1+1+2\cdot1+1+f_{for_k}) = 2 + q(6 + f_{for_k} + f_{check} + f_{add} + f_{rem})$;
  - 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.

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

Трудоемкость худшего случая вычисляется по формуле 2.9.

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

2.4 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 Требования к программному обеспечению

Требования, выдвигаемые к разрабатываемому ПО:

- входные данные - размеры двух матриц, их элементы;
- выходные данные - матрица, являющаяся произведением первой входной матрицы на вторую.

3.2 Средства реализации

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

- имеется большой опыт разработки;
- имеет большое количество расширений и библиотек, что облегчает работу с некоторыми типами данных и математическими формулами;
- обладает информативной документацией.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.5 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1: Классический алгоритм

```
class MatrixMultiplication:
  def __init__(self):
    pass

  def multiply(self, mat1, mat2):
    if len(mat1) == 0 or len(mat2) == 0 or mat1.shape[1] != mat2.
       shape[0]:
      return None
    n = mat1.shape[0]
    m = mat1.shape[1]
    q = mat2.shape[1]
    mat3 = np.zeros((n, q), dtype=float)
    for i in range(n):
      for j in range(q):
        for k in range(m):
          mat3[i][j] += mat1[i][k] * mat2[k][j]

    return mat3
```

Листинг 3.2: Алгоритм Копперсмита-Винограда (часть 1)

```
class VinogradMultiply(MatrixMultiplication):
  def __init__(self):
    super(VinogradMultiply, self).__init__()

  def multiply(self, mat1, mat2):
    if len(mat1) == 0 or len(mat2) == 0 or mat1.shape[1] != mat2.
       shape[0]:
      return None
    n = mat1.shape[0]
    m = mat1.shape[1]
    q = mat2.shape[1]
    mat3 = np.zeros((n, q), dtype=float)
    mulH = self.get_mulH(mat1)
    mulV = self.get_mulV(mat2)
```

Листинг 3.3: Алгоритм Копперсмита-Винограда (часть 2)

```
    for i in range(n):
      for j in range(q):
        mat3[i][j] = - mulH[i] - mulV[j]
        for k in range(m // 2):
          mat3[i][j] = mat3[i][j] + (mat1[i][2 * k] + mat2[2 * k +
              1][j]) * \
          (mat1[i][2 * k + 1] + mat2[2 * k][j])

    if m % 2 == 1:
      for i in range(n):
        for j in range(q):
          mat3[i][j] = mat3[i][j] + mat1[i][m - 1] * mat2[m-1][j]
    return mat3

  def get_mulH(self, mat1):
    mulH = np.zeros(mat1.shape[0], dtype=float)
    for i in range(mat1.shape[0]):
      for k in range(mat1.shape[1] // 2):
        mulH[i] = mulH[i] + mat1[i][2 * k] * mat1[i][2 * k + 1]
    return mulH

  def get_mulV(self, mat2):
    mulV = np.zeros(mat2.shape[1], dtype=float)
    for i in range(mat2.shape[1]):
      for k in range(mat2.shape[0] // 2):
        mulV[i] = mulV[i] + mat2[2 * k][i] * mat2[2 * k + 1][i]
    return mulV
```

Листинг 3.4: Оптимизированный алгоритм Копперсмита-Винограда (часть 1)

```
class VinogradOptimise(MatrixMultiplication):
  def __init__(self):
    super(VinogradOptimise, self).__init__()

  def multiply(self, mat1, mat2):
    if len(mat1) == 0 or len(mat2) == 0 or mat1.shape[1] != mat2.
       shape[0]:
      return None
    n = mat1.shape[0]
    m = mat1.shape[1]
    q = mat2.shape[1]
```

Листинг 3.5: Оптимизированный алгоритм Копперсмита-Винограда (часть 2)

```
    mat3 = np.zeros((n, q), dtype=float)
    mulH = self.get_mulH(mat1)
    mulV = self.get_mulV(mat2)

    for i in range(n):
      for j in range(q):
        buff = mulH[i] - mulV[j]
        for k in range(0, m - 1, 2):
          buff += (mat1[i][k] + mat2[k + 1][j]) * \
          (mat1[i][k + 1] + mat2[k][j])
        if m % 2 == 1:
          buff += mat1[i][m - 1] * mat2[m - 1][j]
        mat3[i][j] = buff

    return mat3

  def get_mulH(self, mat1):
    mulH = np.zeros(mat1.shape[0], dtype=float)
    for i in range(mat1.shape[0]):
      for k in range(0, mat1.shape[1] - 1, 2):
        mulH[i] -= mat1[i][k] * mat1[i][k + 1]
    return mulH

  def get_mulV(self, mat2):
    mulV = np.zeros(mat2.shape[1], dtype=float)
    for i in range(mat2.shape[1]):
      for k in range(0, mat2.shape[0] - 1, 2):
        mulV[i] += mat2[k][i] * mat2[k + 1][i]
    return mulV
```

3.4 Тестирование

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

3.5 Выводы

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 4.1-4.1.

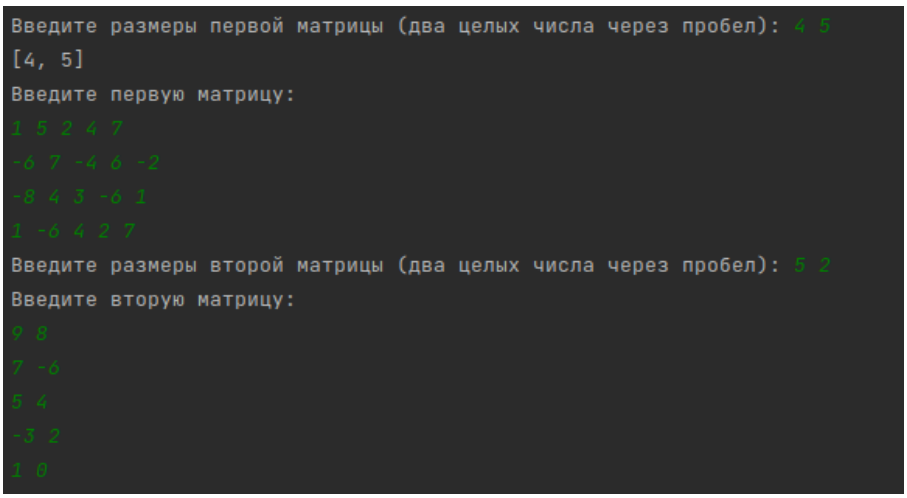

Рис. 4.1: Ввод данных

```
Первая матрица:
 [[ 1.  5.  2.  4.  7.]
 [-6.  7. -4.  6. -2.]
 [-8.  4.  3. -6.  1.]
 [ 1. -6.  4.  2.  7.]]
Вторая матрица:
 [[ 9.  8.]
 [ 7. -6.]
 [ 5.  4.]
 [-3.  2.]
 [ 1.  0.]]
Классический алгоритм умножения матриц
CPU time:  947  mks
Result:
 [[ 49.  -6.]
 [-45. -94.]
 [-10. -88.]
 [-12.  64.]]
Алгоритм Винограда умножения матриц
CPU time:  479  mks
Result:
 [[ 49.  -6.]
 [-45. -94.]
 [-10. -88.]
 [-12.  64.]]
Оптимизированный алгоритм Винограда умножения матриц
CPU time:  382  mks
Result:
 [[ 49.  -6.]
 [-45. -94.]
 [-10. -88.]
 [-12.  64.]]
```

Рис. 4.2: Результат работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Ubuntu 20.01 Linux x86_64 [5];
- оперативная память: 8 Гб;

- процессор: AMD Ryzen5 3500U [6].

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось на автоматически генерируемых квадратных матрицах необходимого размера с использованием функции getrusage библиотеки resources [7]. Усредненные результаты замеров процессорного времени приведены в таблице. Используемые обозначения: "Классич." - классический алгоритм умножения матриц, "Виноград" - алгоритм Копперсмита-Винограда, "Опт.Виноград" - оптимизированный алгоритм Копперсмита-Винограда.

Таблица 4.1: Время обработки строк разной длины в микросекундах

Размер	Классич.	Виноград	Опт.Виноград
10	7626	8432	6334
11	10114	11119	8247
30	198387	200491	144975
31	221763	221463	159228
50	914673	913305	651054
51	990919	975478	695120
70	2568877	2492211	1768562
71	2692695	2625663	1855013
90	5472335	5286995	3717559
91	5634068	5480780	3892148
110	10070443	9613919	6775491
111	10310690	9927475	7034633

Рис. 4.3: Сравнение времени работы алгоритмов умножения квадратных матриц четных размеров

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

4.4 Выводы

По результатам проведенных замеров видно, что оптимизированный алгоритм Копперсмита-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. Алгоритм Копперсмита-Винограда на четных матрицах меньше 50x50 работает медленнее классического алгоритма, на больших - чуть быстрее, однако разрыв составляет не более 1.05 раз. В случае с нечетным размером матрицы алгоритм Винограда на небольших матрицах проигрывает классическому алгоритму, но при увеличении размерности время работы обоих алгоритмов сопоставимо, с небольшим преимуществом алгоритма Винограда.

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.

Список литературы

[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] About Python [Электронный ресурс]. Режим доступа: `https://www.python.org/about/`. Дата обращения: 03.10.2021.

[5] Ubuntu по-русски [Электронный ресурс]. Режим доступа: `https://ubuntu.ru/`. Дата обращения: 03.10.2021.

[6] Мобильный процессор AMD Ryzen™ 5 3500U с графикой Radeon™ Vega 8 [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-5-3500u`. Дата обращения: 03.10.2021.

[7] resource — Resource usage information - Python 3.11.0a0 documentation. Режим доступа: `https://docs.python.org/dev/library/resource.html?highlight=resources`. Дата обращения: 03.10.2021.