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 Талышева О.Н.

Группа ИУ7-55Б

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

Москва — 2024 г.

Содержание

ВВЕДЕНИЕ

Цель работы: рассмотреть различные алгоритмы умножения матриц (стандартный алгоритм, алгоритм Винограда и его оптимизированная версия). Также будет изучена оценка сложности алгоритмов и получены навыки по их улучшению.

Для достижения этой цели были поставлены следующие задачи:

- рассмотреть стандартный алгоритм умножения матриц и алгоритм Винограда;
- оптимизировать алгоритм Винограда методом по варианту;
- провести теоретическую оценку сложности стандартного алгоритма умножения матриц, алгоритма Винограда и его оптимизированной версии;
- реализовать три алгоритма умножения матриц на выбранном языке программирования;
- сравнить эффективность этих алгоритмов на практике.

1. Аналитическая часть

1.1. Базовые понятия матриц

Определение 1. Матрицей A размеров $m \times n$ называется совокупность $m \times n$ элементов из некоторого поля, расположенных в виде таблицы из m строк и n столбцов:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} = (a_{ij})_{m \times n},$$

где a_{ij} $(i = 1, 2, \dots, m;\ j = 1, 2, \dots, n)$ — элементы матрицы, а i указывает номер строки, а j — номер столбца.

Определение 2. Матрица называется квадратной, если $m = n$, и число n называется порядком матрицы. При этом число n называется порядком матрицы.

У квадратной матрицы можно выделить главную и побочную диагонали:

- Главная диагональ: $a_{11}, a_{22}, \dots, a_{nn}$;
- Побочная диагональ: $a_{n1}, a_{(n-1)2}, \dots, a_{1n}$.

Определение 3. Суммой $A + B$ двух матриц $A = (a_{ij})_{m \times n}$ и $B = (b_{ij})_{m \times n}$ называется матрица $C = (c_{ij})_{m \times n}$, где $c_{ij} = a_{ij} + b_{ij}$.

Определение 4. Произведением $\lambda \cdot A$ матрицы $A = (a_{ij})_{m \times n}$ на число λ называется матрица $C = (c_{ij})_{m \times n}$, где $c_{ij} = \lambda \cdot a_{ij}$. [3]

1.2. Умножение матриц

Определение 5. Произведением $A \cdot B$ матрицы $A = (a_{ij})_{m \times n}$ на матрицу $B = (b_{ij})_{n \times p}$ называется матрица $C = (c_{ij})_{m \times p}$, где

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}.$$

Элемент c_{ij} равен сумме попарных произведений соответствующих элементов i-й строки матрицы A и j-го столбца матрицы B.
Замечание. Перемножать можно только те матрицы, у которых число столбцов первой матрицы равно числу строк второй.

Свойства умножения матриц:

1) Не является коммутативным: $A \cdot B \neq B \cdot A$ в общем случае;
2) Ассоциативность: $(A \cdot B) \cdot C = A \cdot (B \cdot C)$;
3) Дистрибутивность: $(A + B) \cdot C = A \cdot C + B \cdot C$;
4) Для любой квадратной матрицы A порядка n справедливо: $A \cdot E = E \cdot A = A$,

где E — единичная матрица порядка n. [3]

1.3. Умножение матриц по Винограду

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

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4.$$

Это равенство можно переписать в виде:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4.$$

Эквивалентность двух выражений можно легко проверить. Кажется, что второе выражение задает больше работы, чем первое: вместо четырёх умножений мы видим шесть, а вместо трёх сложений — десять. Менее очевидно, что правое выражение допускает предварительную обработку: его части можно вычислить заранее и сохранить для каждой строки первой матрицы и для каждого столбца второй.

На практике это означает, что после предварительной обработки остаётся выполнить два умножения и семь сложений. [2]

1.4. Анализ алгоритма Винограда

Рассмотрим случай с чётной общей размерностью b. Результаты подсчета сложений и умножений сведены в таблицу 1.

Таблица 1 – Анализ сложности алгоритма Винограда

Этап	Умножения	Сложения
Предварительная обработка матрицы G	ad	$a(d-1)$
Предварительная обработка матрицы H	cd	$c(d-1)$
Вычисление матрицы R	acd	$ac(2d+d+1)$
Всего	$\frac{abc+ab+bc}{2}$	$\frac{a(b-2)+c(b-2)+ac(3b+2)}{2}$

[2]

2. Конструкторская часть

2.1. Описания алгоритмов

На основании теоретических измышлений были разработаны алгоритмы, вычисляющие произведение матриц тремя способами: стандартным, Винограда и оптимизированным Винограда. Блок-схемы этих алгоритмов приведены на рисунках 1 и 2 (стандартный), рисунках 3, 4 и 5 (Винограда) и рисунках 6, 7 и 8 (оптимизированный Винограда). Вариант оптимизации: двоичный сдвиг вместо умножения на 2; объединение III и IV частей алгоритма Винограда; вынос начальной итерации из каждого внешнего цикла.

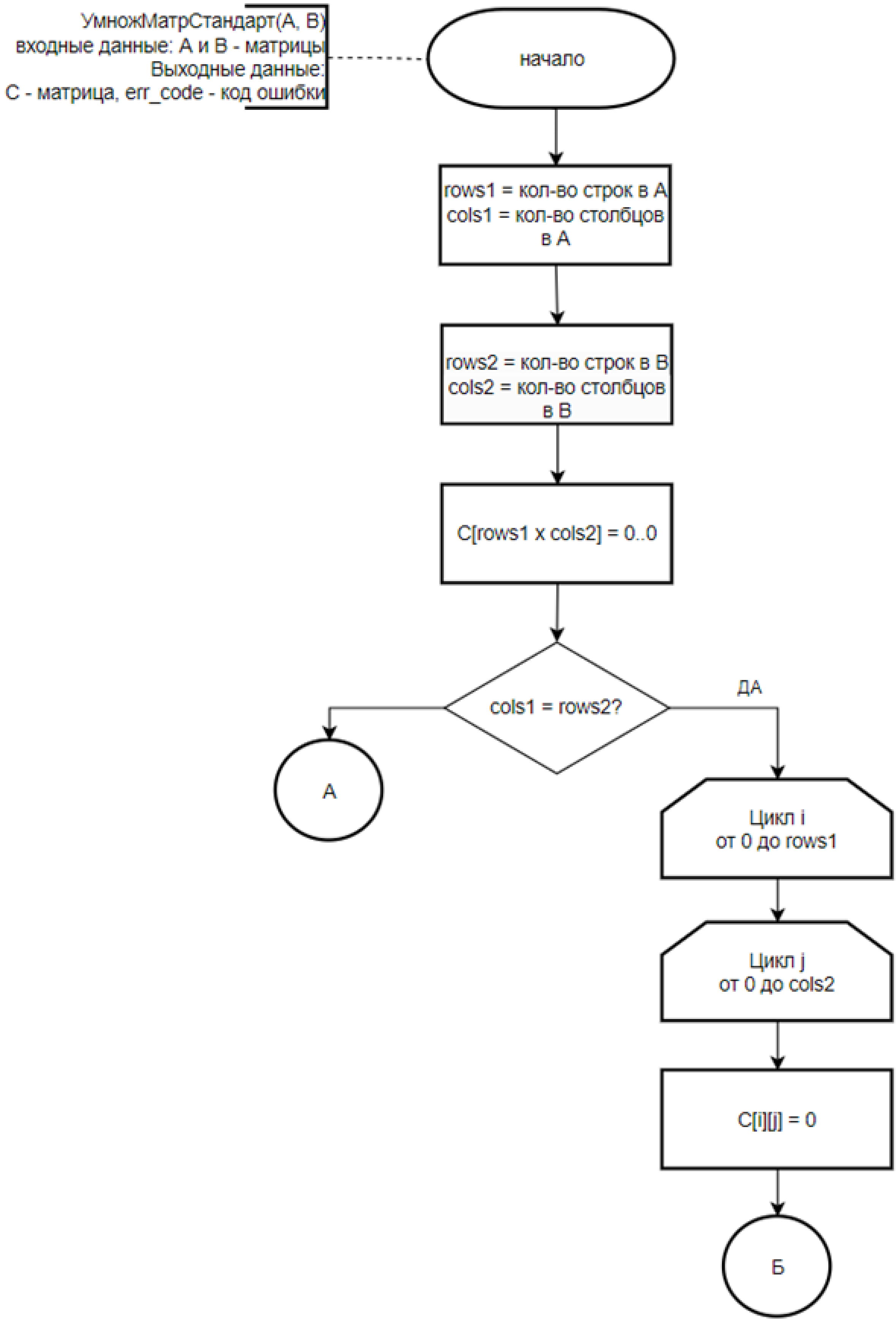

Рисунок 1 – Блок-схема стандартного алгоритма умножения матриц

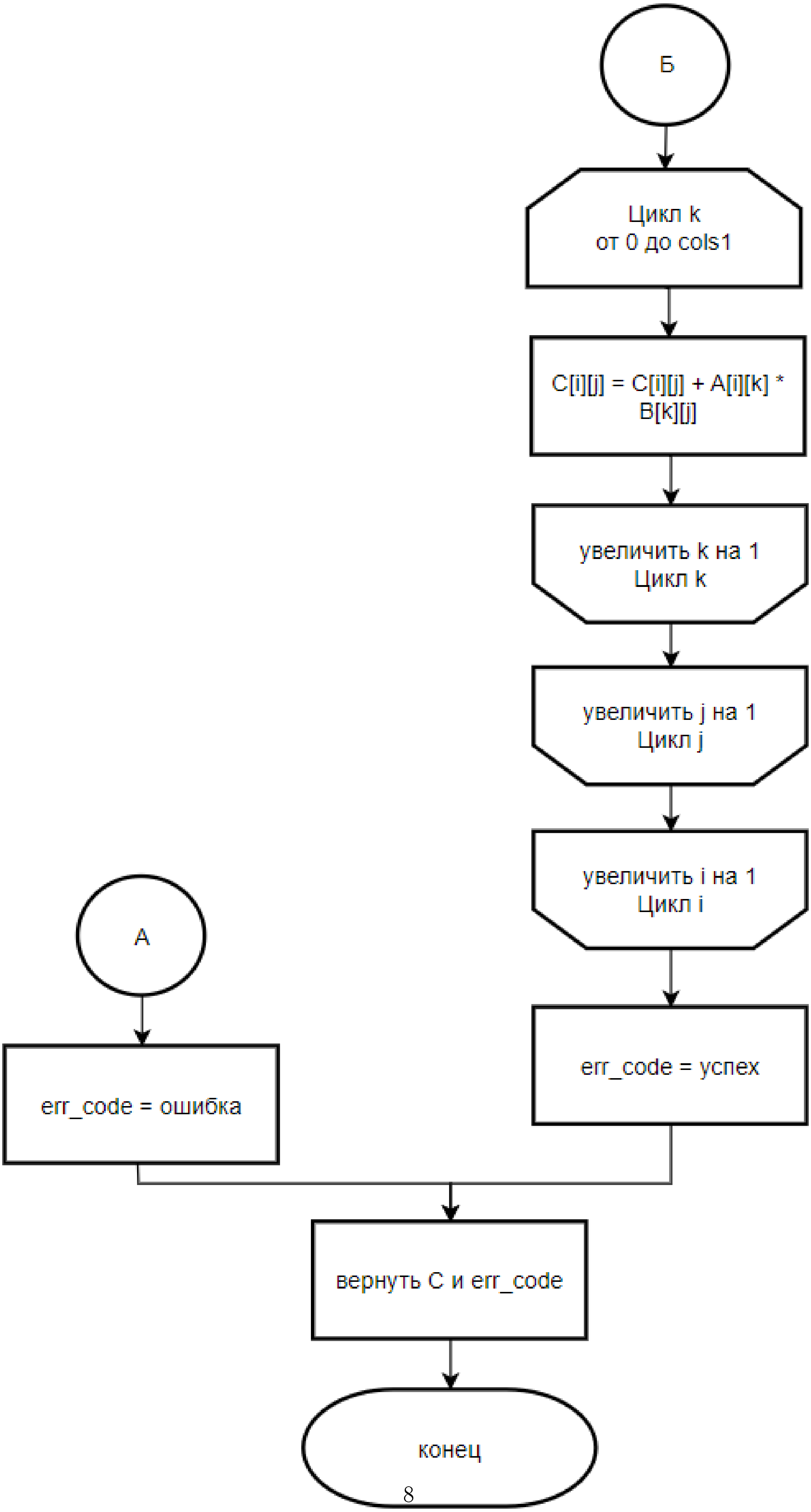

Рисунок 2 – Блок-схема стандартного алгоритма умножения матриц (продолжение)

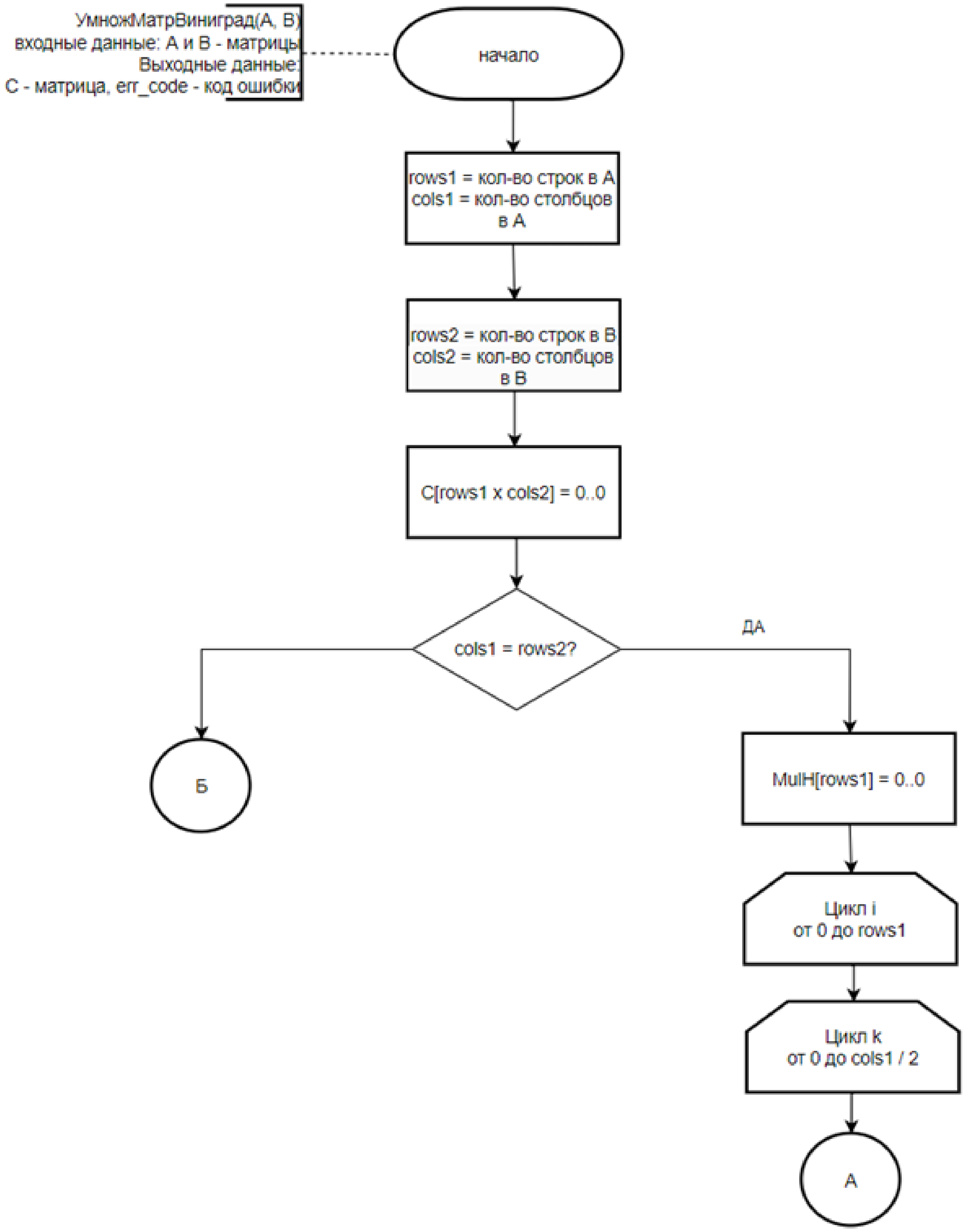

Рисунок 3 – Блок-схема алгоритма Винограда умножения матриц

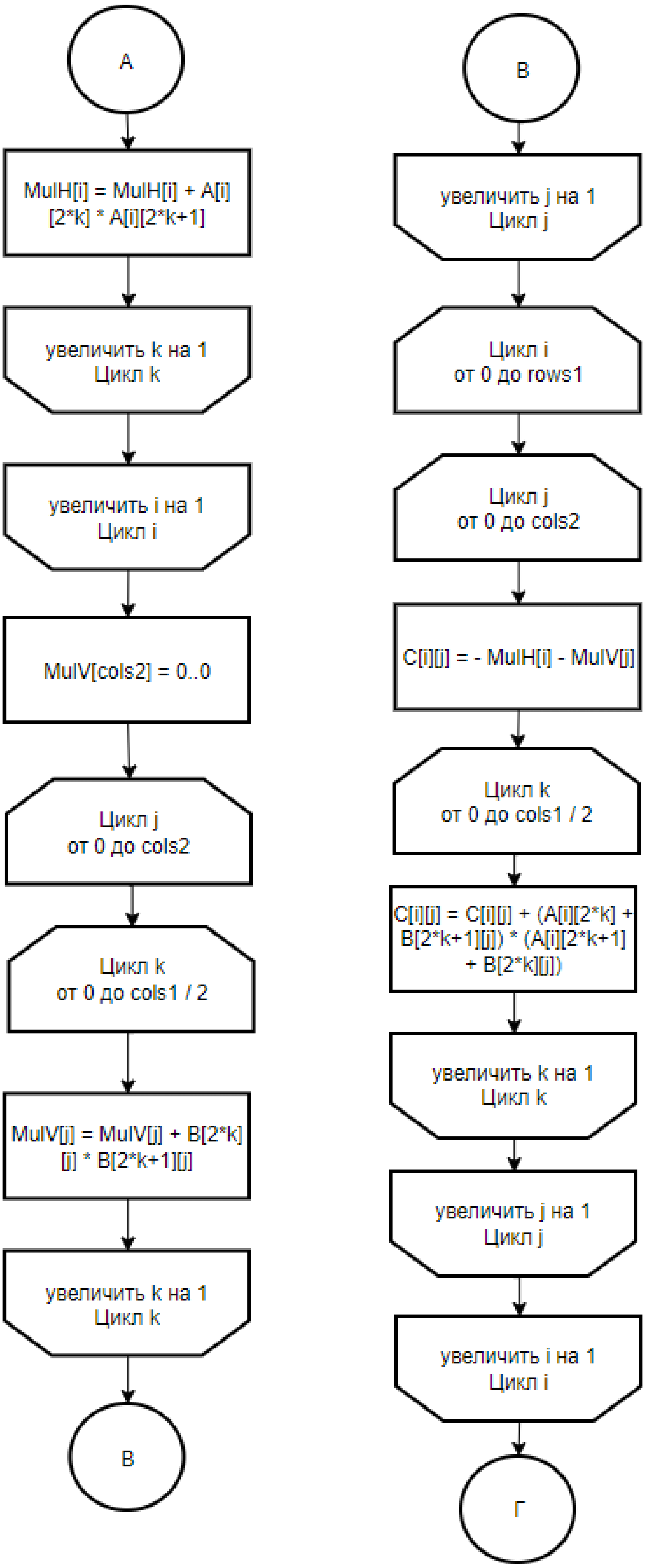

Рисунок 4 – Блок-схема алгоритма Винограда умножения матриц (продолжение)

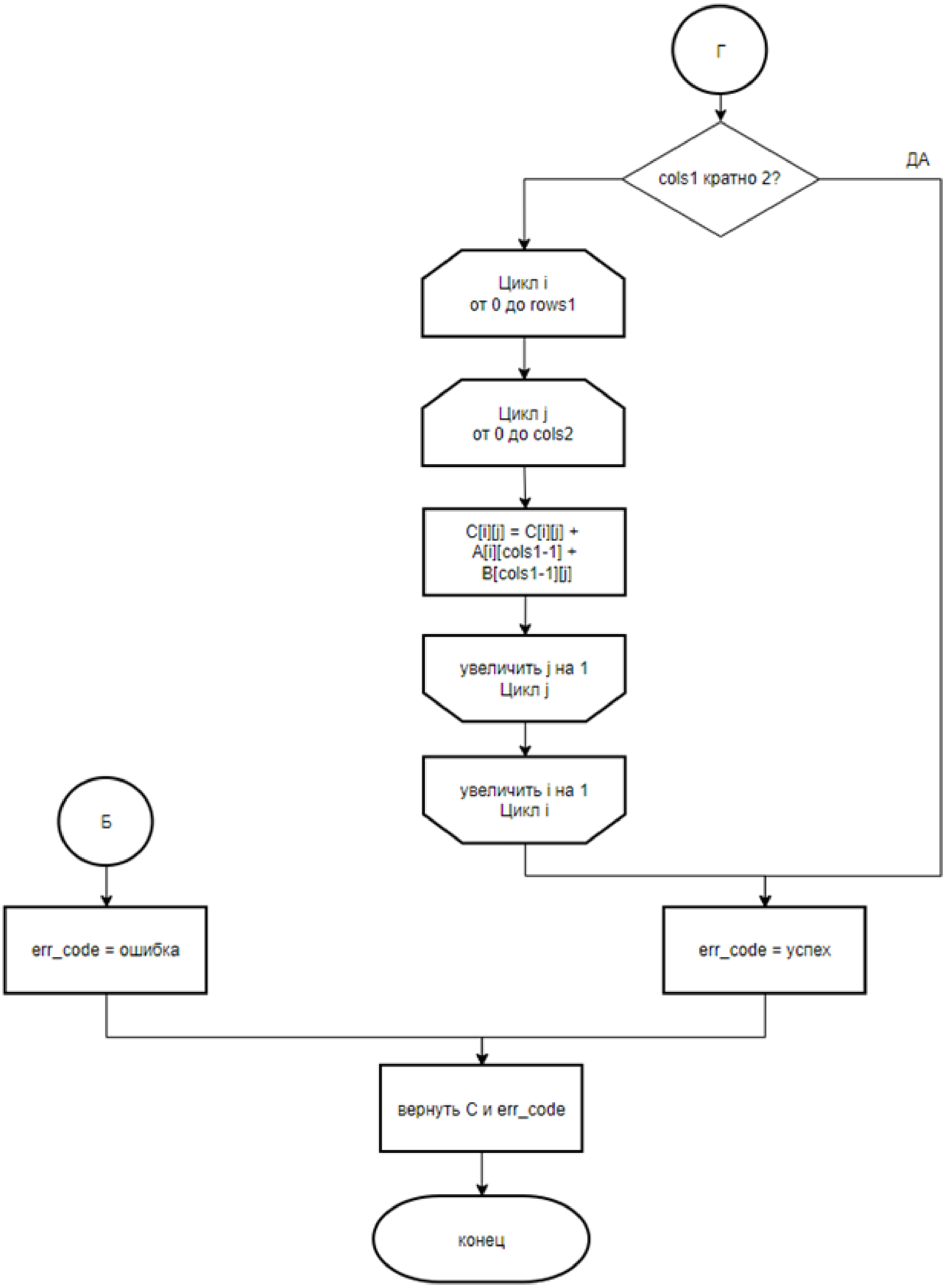

Рисунок 5 – Блок-схема алгоритма Винограда умножения матриц (продолжение 2)

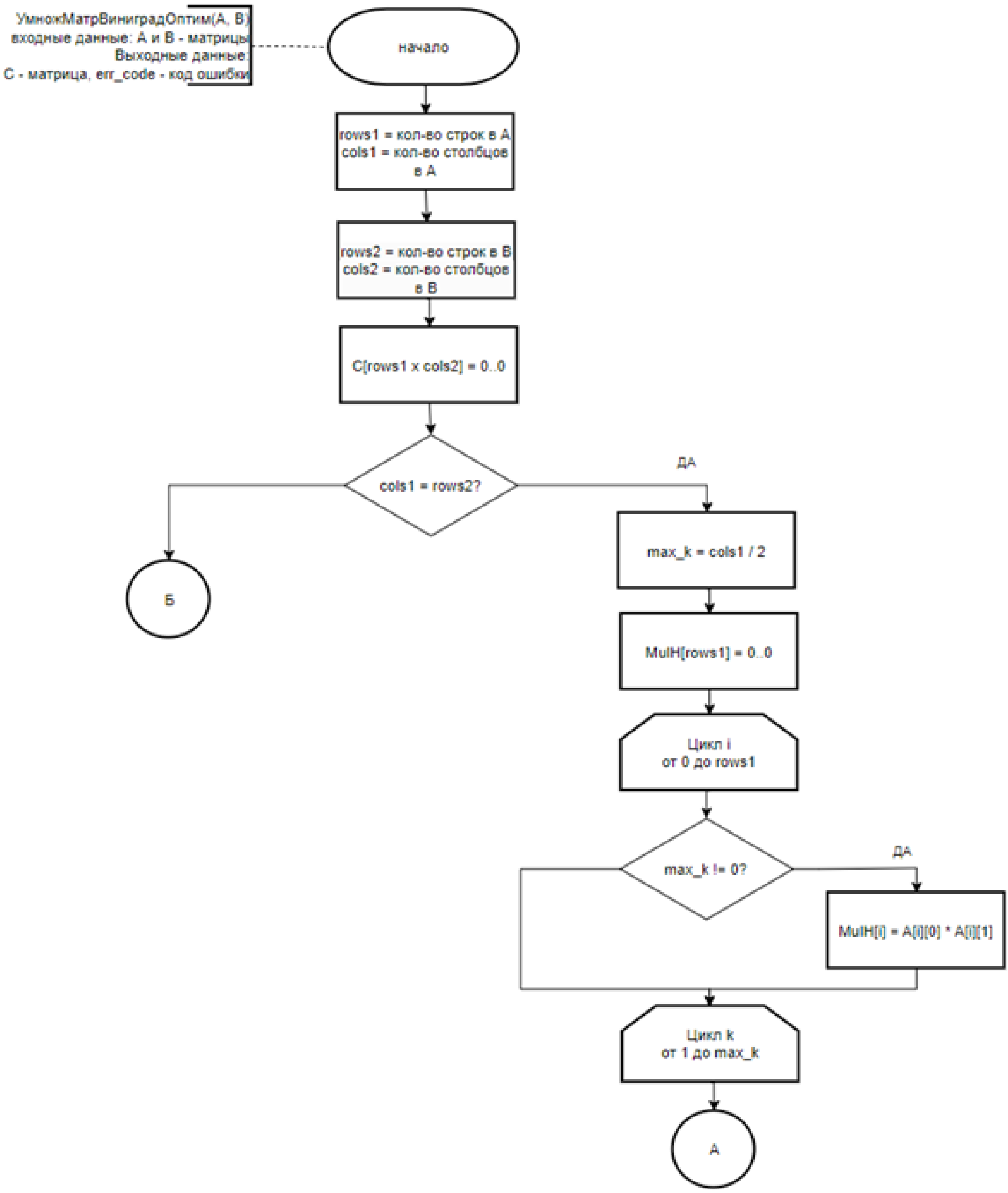

Рисунок 6 – Блок-схема оптимизированного алгоритма Винограда умножения матриц

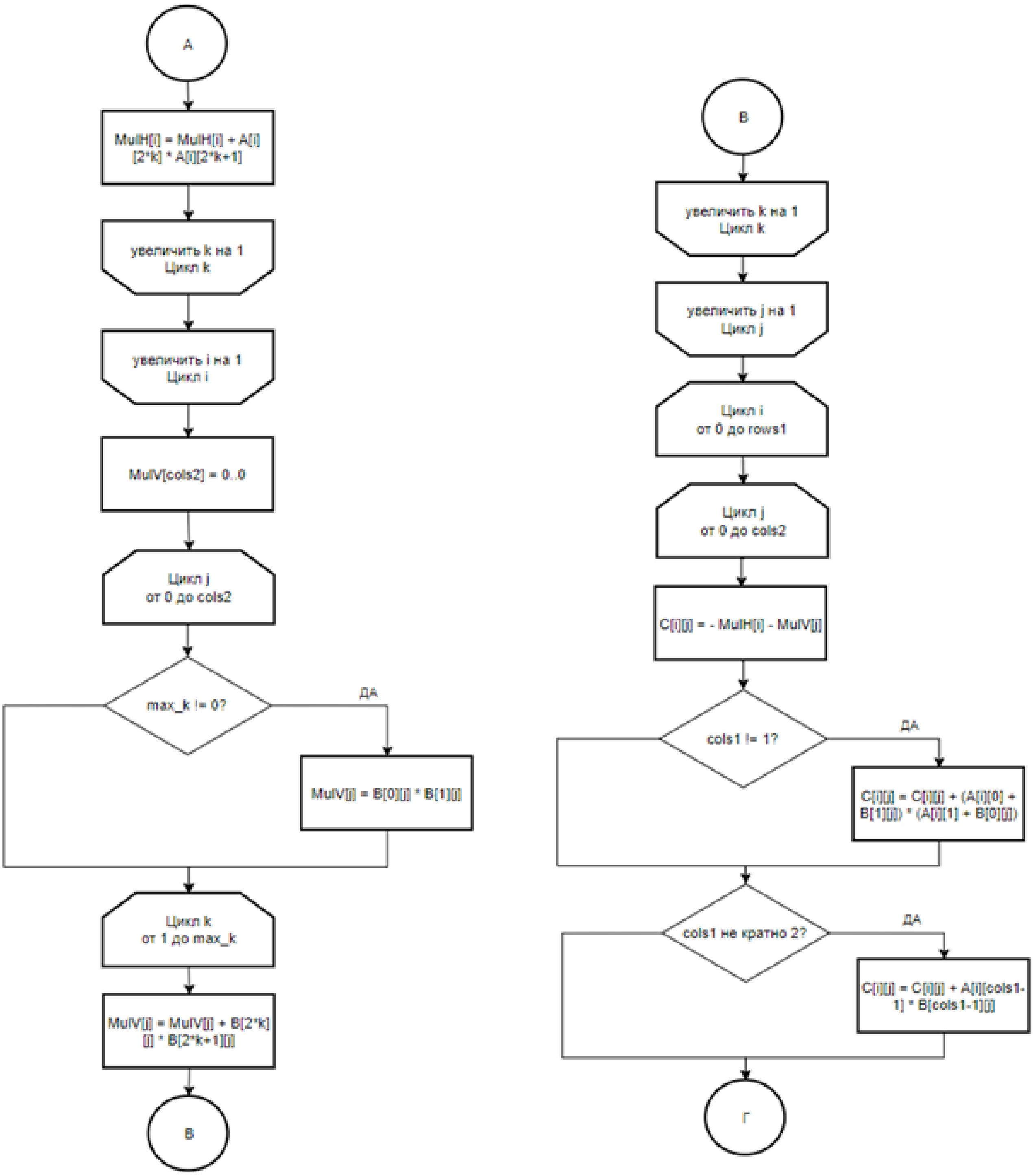

Рисунок 7 – Блок-схема оптимизированного алгоритма Винограда умножения матриц (продолжение)

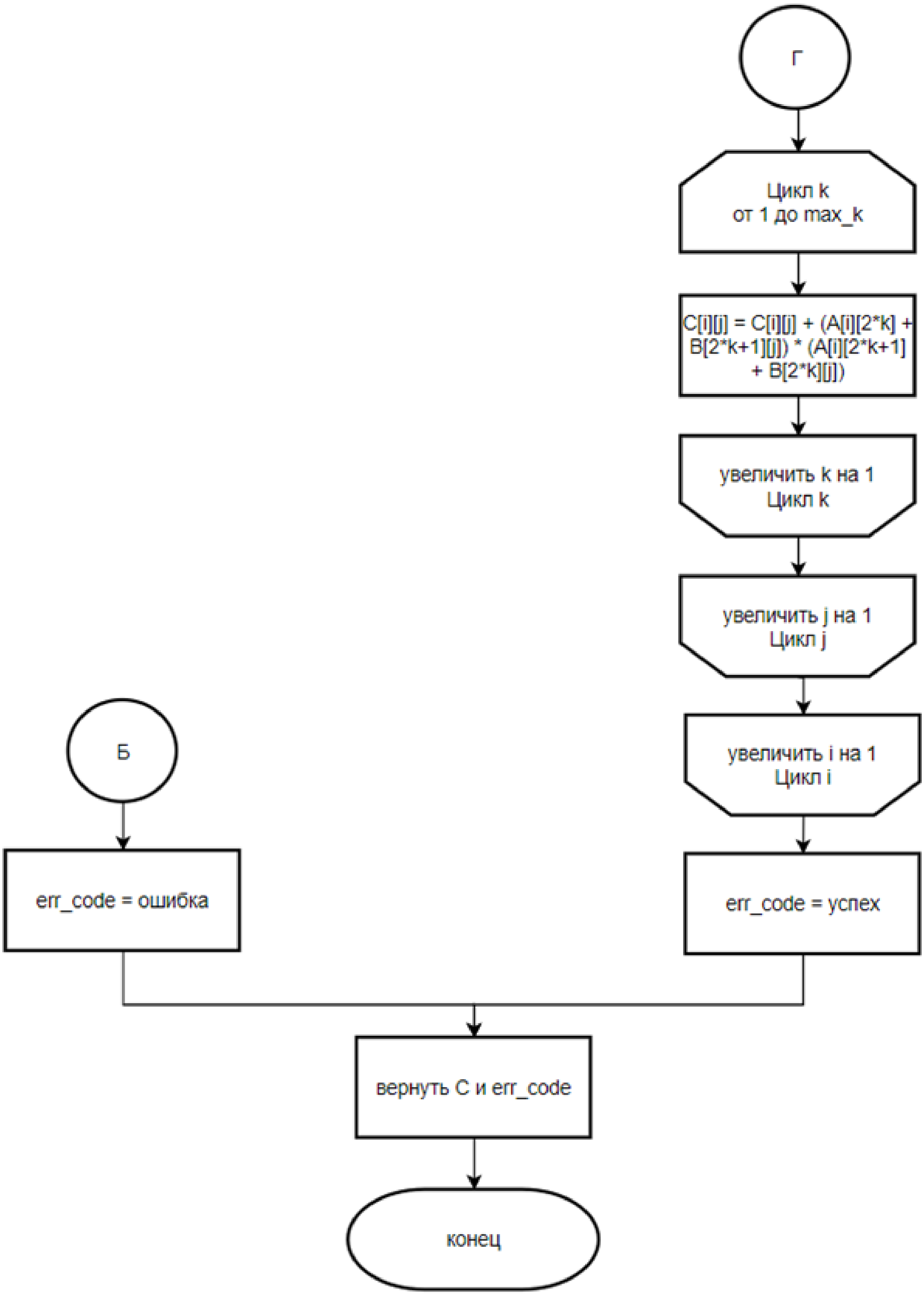

Рисунок 8 – Блок-схема оптимизированного алгоритма Винограда умножения матриц (продолжение 2)

2.2. Модель вычислений

Введём правила, по которым будем оценивать ресурсную эффективность:

1. трудоёмкость базовых операций

   Примем единичной трудоёмкость следующих операций:

   =, +, -, +=, -=, ==, !=, <, >, <=, >=, [], », «, &&, ||, |=, &=

   Трудоёмкость примем за 2:

   /, *, *=, /=,

2. трудоёмкость оператора цикла

   Пусть для цикла вида:

   for (инициализация, сравнение, инкремент)
   { тело }

   известны трудоёмкости соответствующих блоков:

   $$f_{\text{иниц}}, f_{\text{срав}}, f_{\text{инкрем}}, f_{\text{тела}}$$

   Тогда трудоёмкость цикла с "n"итерациями имеет следующий вид:

   $$f_{\text{цикла}} = f_{\text{иниц}} + f_{\text{срав}} + n * (f_{\text{тела}} + f_{\text{инкрем}} + f_{\text{срав}}) \tag{1}$$

3. трудоёмкость условного оператора

   Пусть трудоёмкость условного перехода равна нулю, тогда для условного оператора вида:

   if (условие)
   { блок 1 }
   else
   { блок 2 }

   известны трудоёмкости соответствующих блоков:

   $$f_1, f_2, f_{\text{усл}}$$

   Тогда трудоёмкость будет оценена как:

   $$f_{\text{if}} = f_{\text{усл}} + \begin{cases} \min(f_1, f_2) & \text{лучший случай;} \\ \max(f_1, f_2) & \text{худший случай.} \end{cases} \tag{2}$$

   Примечание: при $f_1 = f_2$ разделения на случаи не будет.

2.3. Расчёт трудоёмкости алгоритмов

Далее будет рассчитана трудоёмкость стандартного алгоритма умножения матриц, Винограда и оптимизированного алгоритма Винограда.

Примечание: выделение памяти под матрицу не будет включено в расчёты трудоёмкости как не самая тяжёлая операция, присутствующая во всех алгоритмах.

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из пяти основных компонент:

1. внешний цикл по i в диапазоне [0, M)

$$f_i = 1 + M * (1 + 1 + f_{\text{тела}}) + 1 \tag{3}$$

2. внутренний цикл по j в диапазоне [0, Q)

$$f_i = 1 + Q * (1 + 1 + f_{\text{тела}}) + 1 \tag{4}$$

3. инициализация текущего элемента нулевым значением с трудоёмкостью равной 3
4. внутренний цикл по k в диапазоне [0, N)

$$f_i = 1 + N * (1 + 1 + f_{\text{тела}}) + 1 \tag{5}$$

5. добавление к текущему элементу произведения чисел из соответствующих строки и столбца с трудоёмкостью равной 12

Подставив в одну формулу, получим итоговую трудоёмкость:

$$f_{\text{стандарт}} = 2 + M * (4 + Q * (4 + 3 + N * (2 + 12))) = 14 * MNQ + 7 * MQ + 4 * M + 2 \tag{6}$$

2.3.2 Алгоритм Винограда умножения матриц

Трудоёмкость алгоритма Винограда умножения матриц состоит из четырёх основных компонент:

1. заполнение массива MulH

$$f_{MulH} = 2 + M * (2 + 4 + N/2 * (4 + 4 + 11)) = 19/2 * MN + 6 * M + 2 \tag{7}$$

2. заполнение массива MulV

$$f_{MulV} = 2 + Q * (2 + 4 + N/2 * (4 + 4 + 11)) = 19/2 * QN + 6 * Q + 2 \tag{8}$$

3. заполнение матрицы C

$$f_C = 2 + M*(2+2+Q*(2+7+4+N/2*(4+6+22))) = 32/2*MNQ + 13*MQ + 4*M + 2 \tag{9}$$

4. поправка на нечётность N

$$f_{odd_N} = 3 + \begin{cases} 0 & \text{л.с.(N чётно);} \\ 2 + M*(2+2+Q*(2+11)) & \text{х.с.(N нечётно).} \end{cases} = 3 + \begin{cases} 0 & \text{л.с.} \\ 13*MQ + NM + 2 & \text{х.с.} \end{cases} \tag{10}$$

Подставив в одну формулу, получим итоговую трудоёмкость:

$$f_{\text{Виноград}} = f_{MulH} + f_{MulV} + f_C + f_{odd_N}$$

$$f_{\text{Виноград}} = \frac{19}{2}MN + 6M + 2 + \frac{19}{2}QN + 6Q + 2 + \frac{32}{2}MNQ + 13MQ + 4M + 2 + 3 + \\ + \begin{cases} 0 & \text{л.с.} \\ 13MQ + NM + 2 & \text{х.с.} \end{cases} \tag{11}$$

$$f_{\text{Виноград}} = \frac{32}{2}MNQ + 13MQ + \frac{19}{2}MN + \frac{19}{2}QN + 10M + 6Q + 9 + \\ + \begin{cases} 0 & \text{л.с.} \\ 13MQ + MN + 2 & \text{х.с.} \end{cases} \tag{12}$$

2.3.3 Оптимизированный алгоритм Винограда умножения матриц

Вариант оптимизации: двоичный сдвиг вместо умножения на 2; объединение III и IV частей алгоритма Винограда; вынос начальной итерации из каждого внешнего цикла.

Трудоёмкость оптимизированного алгоритма Винограда умножения матриц состоит из трёх основных компонент (в связи с оптимизацией — объединением III и IV частей):

1. заполнение массива MulH

$$f_{MulH} = 2 + M*(2 + 1 + \begin{cases} 0 & \text{л.с.(cols1} < 2); \\ 8 & \text{х.с.(cols1} >= 2). \end{cases} + 2 + (N/2 - 1)*(2+4+9))$$

$$f_{MulH} = 2 - 10*M + 15/2*M*N + \begin{cases} 0 & \text{л.с.(cols1} < 2); \\ 8*M & \text{х.с.(cols1} >= 2). \end{cases} \tag{13}$$

2. заполнение массива MulV

$$f_{MulV} = 2 + Q * (2 + 1 + \begin{cases} 0 & \text{л.с.}(\text{cols1} < 2); \\ 8 & \text{х.с.}(\text{cols1} >= 2). \end{cases} + 2 + (N/2 - 1) * (2 + 4 + 9))$$

$$f_{MulV} = 2 - 10 * Q + 15/2 * Q * N + \begin{cases} 0 & \text{л.с.}(\text{cols1} < 2); \\ 8 * Q & \text{х.с.}(\text{cols1} >= 2). \end{cases} \quad (14)$$

3. заполнение матрицы C

$$f_C = 2 + M \cdot (2 + 2 + Q \cdot (2 + 7 + 1 + \begin{cases} 0 & \text{л.с.}(\text{cols1} \mathrel{!}= 1); \\ 18 & \text{х.с.}(\text{cols1} \mathrel{!}= 1). \end{cases} + 3 + \begin{cases} 0 & \text{л.с.}(\text{cols1} \,\vdots\, 2); \\ 14 & \text{х.с.}(\text{cols1} \,\vdots\, 2). \end{cases}$$
$$+ 2 + \left(\frac{N}{2} - 1\right) \cdot (2 + 24)))$$

$$f_C = 2 + 4 \cdot M - 11 \cdot M \cdot Q + 13 \cdot NMQ + \begin{cases} 0 & \text{л.с.}(\text{cols1} \mathrel{!}= 1); \\ 18 \cdot MQ & \text{х.с.}(\text{cols1} \mathrel{!}= 1). \end{cases} +$$
$$+ \begin{cases} 0 & \text{л.с.}(\text{cols1} \,\vdots\, 2); \\ 14 \cdot MQ & \text{х.с.}(\text{cols1} \,\vdots\, 2). \end{cases} \quad (15)$$

Подставив в одну формулу, получим итоговую трудоёмкость:

$$f_{\text{Виноград}} = 2 + f_{MulH} + f_{MulV} + f_C$$

Примечание: 2 добавляется за счёт вычисления заранее cols1 / 2.

$$f_{\text{Виноград}} = 2 + 2 - 10 \cdot M + 15/2 \cdot MN + \begin{cases} 0 & \text{л.с.}(\text{cols1} < 2); \\ 8 \cdot M & \text{х.с.}(\text{cols1} >= 2). \end{cases} + 2 + Q \cdot (2 + 1 +$$
$$+ \begin{cases} 0 & \text{л.с.}(\text{cols1} < 2); \\ 8 & \text{х.с.}(\text{cols1} >= 2). \end{cases} + 2 + (N/2 - 1) \cdot (2 + 4 + 9)) + 2 + 4 \cdot M - 11 \cdot M \cdot Q + 13 \cdot NMQ +$$
$$+ \begin{cases} 0 & \text{л.с.}(\text{cols1} \mathrel{!}= 1); \\ 18 \cdot MQ & \text{х.с.}(\text{cols1} \mathrel{!}= 1). \end{cases} + \begin{cases} 0 & \text{л.с.}(\text{cols1} \,\vdots\, 2); \\ 14 \cdot MQ & \text{х.с.}(\text{cols1} \,\vdots\, 2). \end{cases} \quad (16)$$

3. Технологическая часть

3.1. Требования к программному обеспечению

На вход программе подаются 2 матрицы из целых чисел.

На выход программа выдаёт матрицу — произведение входных данных и код ошибки (на случай невозможности перемножить матрицы, если количество столбцов в первой не равно числу строк во второй). Также в зависимости от выбранного пункта меню программа замеряет время работы алгоритмов и рисует получившиеся графики.

3.2. Средства реализации

Python имеет простой и понятный синтаксис, что облегчает кодирование алгоритмов и их тестирование, поэтому программа была реализована на этом языке программирования. Для замеров времени была использована функция `process_time()` из библиотеки `time`, вычисляющая процессорное время [1].

3.3. Реализации алгоритмов

Ниже приведены реализации алгоритмов умножения матриц стандартным способом (листинг 1), алгоритмом Винограда (листинг 2) и оптимизированным алгоритмом Винограда (листинг 3) на Python.

```
def algo_matrix_mult_standard(A: List[List[int]], B: List[List[int]]) ->
    Tuple[List[List[int]], bool]:
    rows1, cols1 = len(A), len(A[0])
    rows2, cols2 = len(B), len(B[0])
    C = [[0 for _ in range(cols2)] for _ in range(rows1)]
    if (cols1 != rows2):
        err_code = ERROR
    else:
        for i in range(rows1):
            for j in range(cols2):
                C[i][j] = 0
                for k in range(cols1):
                    C[i][j] = C[i][j] + A[i][k] * B[k][j]
        err_code = OK
    return C, err_code
```

Листинг 1 – реализация стандартного алгоритма умножения матриц

```
def algo_matrix_mult_Vinograd(A: List[List[int]], B: List[List[int]]) ->
    Tuple[List[List[int]], bool]:
    rows1, cols1 = len(A), len(A[0])
    rows2, cols2 = len(B), len(B[0])
    C = [[0 for _ in range(cols2)] for _ in range(rows1)]
```

```
    if (cols1 != rows2):
        err_code = ERROR
    else:
        # filling the array MulH
        MulH = [0 for _ in range(rows1)]
        for i in range(rows1):
            for k in range(int(cols1 / 2)):
                MulH[i] = MulH[i] + A[i][2*k] * A[i][2*k+1]
        # filling the array MulV
        MulV = [0 for _ in range(cols2)]
        for j in range(cols2):
            for k in range(int(cols1 / 2)):
                MulV[j] = MulV[j] + B[2*k][j] * B[2*k+1][j]
        # filling in the matrix C itself
        for i in range(rows1):
            for j in range(cols2):
                C[i][j] = - MulH[i] - MulV[j]
                for k in range(int(cols1 / 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 cols1 is odd (correction)
        if cols1 % 2 == 1:
            for i in range(rows1):
                for j in range(cols2):
                    C[i][j] = C[i][j] +  A[i][cols1-1] * B[cols1-1][j]
        err_code = OK
    return C, err_code
```

Листинг 2 – реализация алгоритма Винограда умножения матриц

```
def algo_matrix_mult_Vinograd_better(A: List[List[int]], B: List[List[int
    ]]) -> Tuple[List[List[int]], bool]:
    rows1, cols1 = len(A), len(A[0])
    rows2, cols2 = len(B), len(B[0])
    C = [[0 for _ in range(cols2)] for _ in range(rows1)]
    if (cols1 != rows2):
        err_code = ERROR
    else:
        max_k = int(cols1 / 2)
        # filling the array MulH
        MulH = [0 for _ in range(rows1)]
        for i in range(rows1):
            if max_k != 0:
                MulH[i] = A[i][0] * A[i][1]
            for k in range(1, max_k):
                MulH[i] = MulH[i] + A[i][k << 1] * A[i][(k << 1) + 1]
        # filling the array MulV
        MulV = [0 for _ in range(cols2)]
```

```
        for j in range(cols2):
            if max_k != 0:
                MulV[j] = B[0][j] * B[1][j]
            for k in range(1, max_k):
                MulV[j] = MulV[j] + B[k << 1][j] * B[(k << 1) + 1][j]
        # filling in the matrix C itself
        for i in range(rows1):
            for j in range(cols2):
                C[i][j] = - MulH[i] - MulV[j]
                if cols1 != 1:
                    C[i][j] = C[i][j] + (A[i][0] + B[1][j]) * (A[i][1] + B
                        [0][j])
                # if cols1 is odd (correction)
                if cols1 % 2 == 1:
                    C[i][j] = C[i][j] +  A[i][cols1-1] * B[cols1-1][j]
                for k in range(1, max_k):
                    C[i][j] = C[i][j] + (A[i][k << 1] + B[(k << 1) + 1][j
                        ]) * (A[i][(k << 1) + 1] + B[k << 1][j])
        err_code = OK
    return C, err_code
```

Листинг 3 – реализация оптимизированного алгоритма Винограда умножения матриц

3.4. Тесты

Для тестирования всех алгоритмов умножения матриц были составлены тесты с входными данными (2 матрицы), ожидаемым результатом (матрицей — произведением входных) и полученным результатом от всех способов (см таблицу 2).

Матрица A	Матрица B	Ожидание	Результат (код ошибки)	Результат (матрица)
()	()	()	ERROR	()
()	(1)	()	ERROR	()
$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 3 \end{pmatrix}$	$\begin{pmatrix} 6 \end{pmatrix}$	OK	$\begin{pmatrix} 6 \end{pmatrix}$
$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \end{pmatrix}$	()	ERROR	()
$\begin{pmatrix} 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix}$	()	ERROR	()
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 0 & -1 \\ 5 & -2 \end{pmatrix}$	$\begin{pmatrix} 10 & -5 \\ 20 & -11 \end{pmatrix}$	OK	$\begin{pmatrix} 10 & -5 \\ 20 & -11 \end{pmatrix}$
$\begin{pmatrix} 0 & -1 \\ 5 & -2 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -3 & -4 \\ -1 & 2 \end{pmatrix}$	OK	$\begin{pmatrix} -3 & -4 \\ -1 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	$\begin{pmatrix} -1 & 3 & 1 \\ 7 & 2 & 0 \end{pmatrix}$	$\begin{pmatrix} 13 & 7 & 1 \\ 25 & 17 & 3 \\ 37 & 27 & 5 \end{pmatrix}$	OK	$\begin{pmatrix} 13 & 7 & 1 \\ 25 & 17 & 3 \\ 37 & 27 & 5 \end{pmatrix}$

Таблица 2 – Таблица тестов для алгоритмов умножения матриц

В ходе проведённого тестирования (с помощью pytest) ошибок в алгоритмах не выявлено:

```
pytest
======================= test session starts ========================
platform win32 -- Python 3.11.9, pytest-8.2.2, pluggy-1.5.0
rootdir: L:\sem_5\\Algorithm_analysis\lab_02
plugins: Faker-28.4.1
collected 24 items

test_algo.py ....................................            [100%]

======================= 24 passed in 0.47s =========================
```

Листинг 4 – тестирование алгоритмов с помощью pytest

4. Исследовательская часть

Для сравнения времени реализаций стандартного алгоритма, алгоритма Винограда и его оптимизированной версии программа была запущена на рандомно сгенерированных квадратных матрицах из целых чисел от -10 до 10 на чётных размерностях от 50 до 250 с шагом 50 (рисунок 9 и таблица 3) и на нечётных размерностях от 51 до 251 с шагом 50 (график 10 и таблица 4) (так как при нечётной размерности в алгоритме Винограда добавляется цикл) по 50 замеров каждая матрица, среднее значение было вынесено в таблицу и для наглядности изображено на графике.

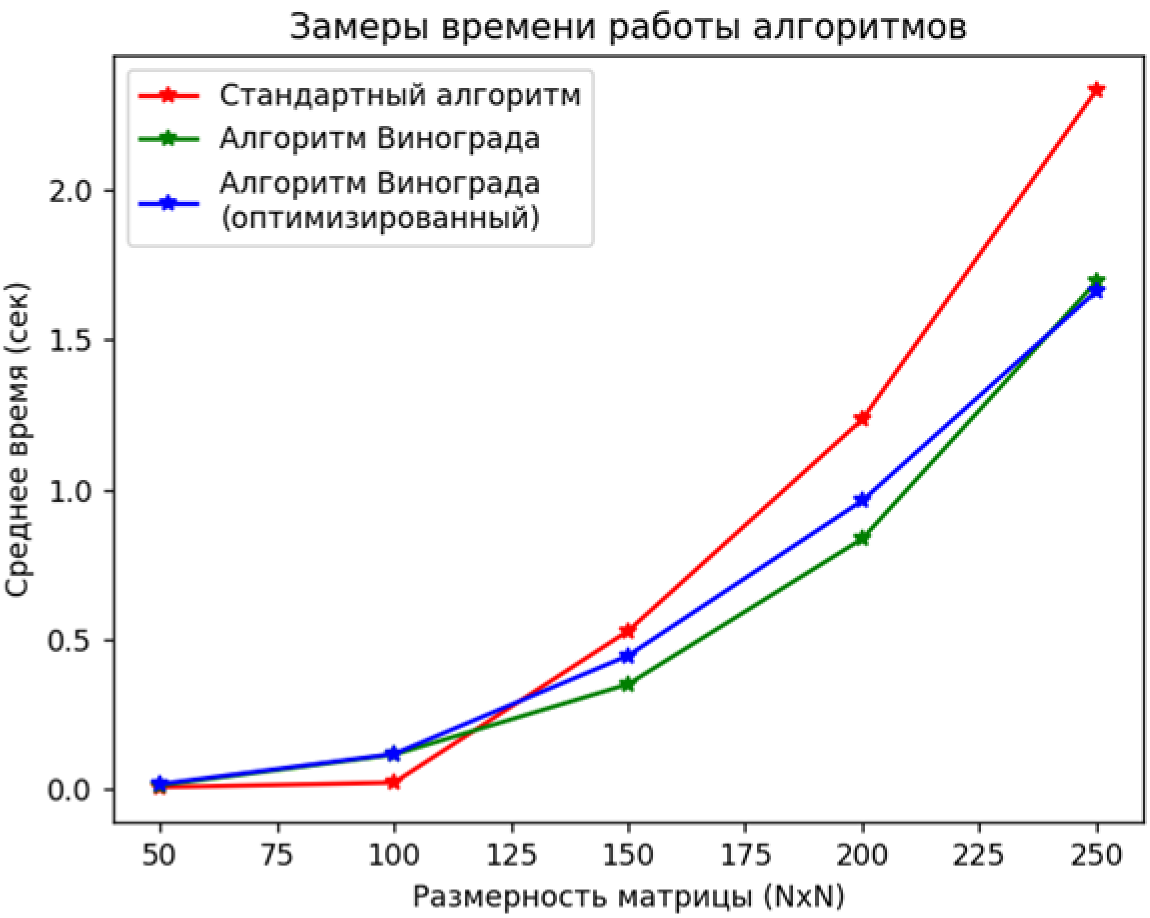

Рисунок 9 – График времени работы алгоритмов в зависимости от чётных размерностей матриц

Алгоритм	50	100	150	200	250
Стандартный	0.0053	0.021	0.53	1.2	2.3
Винограда	0.012	0.11	0.35	0.84	1.7
Винограда (оптимизированный)	0.016	0.12	0.45	0.96	1.7

Таблица 3 – Таблица времени (сек) работы алгоритмов в зависимости от чётных размерностей матриц

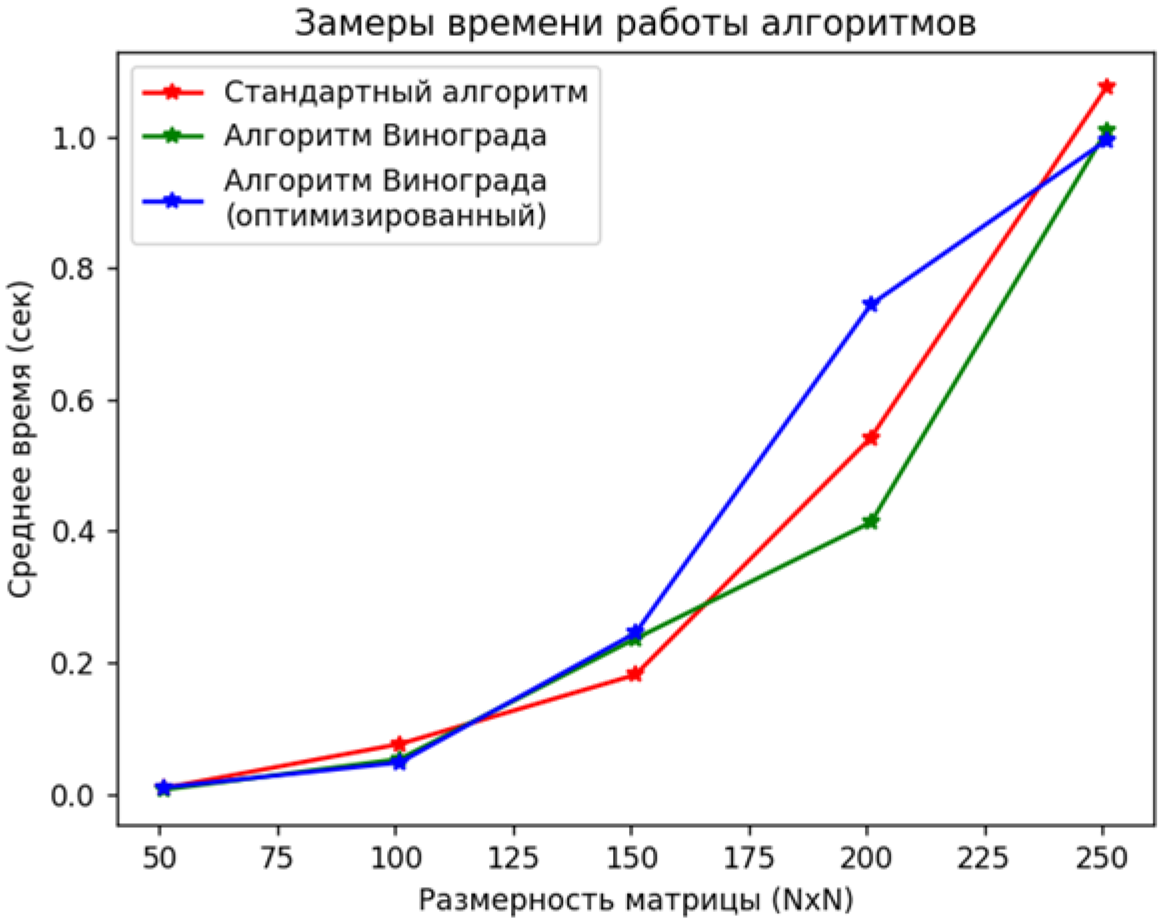

Рисунок 10 – График времени работы алгоритмов в зависимости от нечётных размерностей метриц

Алгоритм	51	101	151	201	251
Стандартный	0.0091	0.076	0.18	0.54	1.1
Винограда	0.0066	0.053	0.24	0.41	1.0
Винограда (оптимизированный)	0.01	0.048	0.24	0.74	0.99

Таблица 4 – Таблица времени (сек) работы алгоритмов в зависимости от нечётных размерностей матриц

Вывод

По проведённым исследованиям была выявлена большая скорость работы алгоритма Винограда над стандартным алгоритмом за счёт уменьшения трудоёмкости вычислений и чем больше размерность матрицы, тем больше видна разница в скорости алгоритмов. Однако на нечётных размерностях такого выигрыша не наблюдается за счёт дополнительного цикла, корректирующего вычисленное значение в алгоритме Винограда.

ЗАКЛЮЧЕНИЕ

В результате выполнения лабораторной работы были рассмотрены различные алгоритмы умножения матриц (стандартный алгоритм, алгоритм Винограда и его оптимизированная версия). Также была изучена оценка сложности алгоритмов и получены навыки по их улучшению.

В частности:

- были рассмотрены стандартный алгоритм умножения матриц и алгоритм Винограда;
- алгоритм Винограда был оптимизирован методом по варианту;
- проведена теоретическая оценка сложности стандартного алгоритма умножения матриц, алгоритма Винограда и его оптимизированной версии;
- реализованы три алгоритма умножения матриц на выбранном языке программирования;
- эффективность этих алгоритмов была сравнена на практике.

В ходе лабораторной работы были рассмотрены, спроетированны и запрограммированы стандартный, Винограда и оптимизированный Винограда алгоритмы умножения матриц.

Были разработаны тесты для всех алгоритмов, учитывающие крайние случаи, ожидаемых результатов которых достигли все реализации.

Сравнения полученных программ показали, что алгоритм Винограда работает быстрее стандартного алгоритма умножения матриц за счёт уменьшения трудоёмности вычислений. Однако на нечётных размерностях он считается медленнее за счёт корректировки вычисленного значения в дополнительном цикле.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Python Documentation. time.process_time() - Документация по стандартной библиотеке Python. Дата обращения: 03 сентября 2024 г. [Электронный ресурс]. Доступно по адресу: https://docs-python.ru/standart-library/modul-time-python/funktsija-process-time-modulja-time/

[2] Algolib: коллекция алгоритмов. Умножение матриц. Дата обращения: 19 сентября 2024 г. [Электронный ресурс]. Доступно по адресу: https://algolib.narod.ru/Math/Matrix.html

[3] Никитенко Е.В. Линейная алгебра и теория матриц. Учебное пособие для студентов всех форм обучения направления «Информатика и вычислительная техника» / Рубцовский индустриальный институт. – Рубцовск, 2022. – 56 с.