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 Алгоритмы умножения матриц

Студент Ву Хай Данг

Группа ИУ7и-52Б

Оценка (баллы)

Преподаватель Строганов Д.В., Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

Задача удобного представления и хранения информации возникает в математике, физике, экономике, статистике и программировании. Для решения этой задачи используют матрицу.

Матрица - массив, элементы которого являются массивами [1]. Эти массивы называют строками матрицы. Элементы, стоящие на одной и той же позиции в разных строках, образуют столбец матрицы.

Над матрицами можно проводить следующие операции: сложение, вычитание, транспонирование, возведение в степень. Одной из часто применяющихся операций является умножение матриц. Например, в машинном обучении реализации распространения сигнала в слоях нейронной сети базируются на умножении матриц. Так, актуальной задачей является оптимизация алгоритмов матричного умножения.

Целью данной лабораторной работы является изучение алгоритмов умножения матриц. Для достижения поставленной цели требуется выполнить следующие задачи:

- изучить алгоритмы стандартного умножения матриц, алгоритм Винограда и алгоритм Штрассена;
- рассмотреть оптимизацию указанного алгоритма;
- привести схемы изучаемых алгоритмов;
- провести сравнительный анализ по трудоемкости рассмотренных алгоритмов;
- реализовать разработанные алгоритмы;
- провести сравнительный анализ по времени реализованных алгоритмов;
- подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут описаны алгоритмы умножения матриц: стандартный, Винограда, Штрассена и оптимизация алгоритма Винограда.

1.1 Стандартный алгоритм матричного умножения

Пусть даны две матрицы A и B размерности $N \cdot P$ и $P \cdot M$ соответсвенно:

$$A_{NP} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{PM} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

Тогда матрица C размерностью N*M:

$$C_{NM} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

в которой:

$$c_{ij} = \sum_{r=1}^{P} a_{ir} b_{rj} \quad (i = \overline{1, N}; j = \overline{1, M}) \tag{1.3}$$

называется их произведением.

Для вычисления произведения двух матриц, каждая строка первой матрицы почленно умножается на каждый столбец второй, затем подсчитывается сумма таких произведений и записывается в соответствующий элемент результирующей матрицы [2].

1.2 Алгоритм Винограда умножения матриц

Алгоритм Винограда — это алгоритм умножения матриц, который был разработан в 1960-х годах Виктором Виноградовым. Этот алгоритм позволяет сократить количество умножений при умножении двух матриц, что делает его более эффективным для больших матриц по сравнению с классическим методом умножения.

Основная идея алгоритма Винограда заключается в том, чтобы предварительно вычислить некоторые промежуточные значения и затем использовать их для вычисления конечного результата. Это позволяет уменьшить количество умножений.

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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4):

$$V \cdot W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4. \quad (1.4)$$

Выражение в правой части формулы 1.4 допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. При нечетном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов к результату [2].

1.3 Алгоритм Штрассена умножения матриц

Алгоритм Штрассена работает с квадратными матрицами. На самом деле он настолько эффективен, что иногда разумно расширить матрицы до квадратных, и при этом он все равно дает выигрыш, превышающий расходы на введение дополнительных элементов. Для умножения матриц алгоритм Штрассена использует семь формул. Эти формулы чрезвычай-

но неестественны и, к сожалению, в оригинальной статье Штрассена не объясняется, как он пришел к ним. Замечательно, что как сами формулы, так и их использование не требуют, чтобы умножение элементов матриц было коммутативным. Это означает, в частности, что сами эти элементы могут быть матрицами, а значит, алгоритм Штрассена можно применять рекурсивно. Вот формулы Штрассена:

$$x_1 = (G_{1,1} + G_{2,2}) \cdot (H_{1,1} + H_{2,2}) \tag{1.5}$$

$$x_2 = (G_{2,1} + G_{2,2}) \cdot H_{1,1} \tag{1.6}$$

$$x_3 = G_{1,1} \cdot (H_{1,2} - H_{2,2}) \tag{1.7}$$

$$x_4 = G_{2,2} \cdot (H_{1,2} - H_{2,2}) \tag{1.8}$$

$$x_5 = (G_{1,1} + G_{2,2}) \cdot H_{2,2} \tag{1.9}$$

$$x_6 = (G_{2,1} - G_{1,1}) \cdot (H_{1,1} + H_{1,2}) \tag{1.10}$$

$$x_7 = (G_{2,1} - G_{2,2}) \cdot (H_{2,1} + H_{2,2}) \tag{1.11}$$

Теперь элементы матрицы R могут вычисляться по формулам:

$$R_{1,1} = x_1 + x_4 - x_5 + x_7 \tag{1.12}$$

$$R_{1,2} = x_3 + x_5 \tag{1.13}$$

$$R_{2,1} = x_2 + x_4 \tag{1.14}$$

$$R_{2,2} = x_1 + x_3 - x_2 + x_6 \tag{1.15}$$

На практике алгоритм Штрассена применяется редко: его использование требует аккуратного отслеживания рекурсии.[3].

1.4 Оптимизация алгоритма Винограда умножения матриц

Оптимизированная версия алгоритма Винограда [2] отличается:

- замены операцию x = x + k на x += k;

- замены умножение на 2 на побитовый сдвиг;
- предвычисление некоторые слагаемые для алгоритма..

1.5 Вывод

Были изучены способы умножения матриц при помощи классического алгоритма, алгоритма Винограда и алгоритма Штрассена. Также была рассмотрена оптимизация алгоритма Винограда.

2 Конструкторская часть

В данном разделе будут представлены требования к программе, схемы алгоритмов умножения матриц: стандартного алгоритма, алгоритма Винограда, алгоритма Штрассена и оптимизированного алгоритма Винограда.

2.1 Требования к программе

К программе предъявляются следующие требования.

- Программа должна предоставлять 2 режима работы: режим умножения матриц между введёнными пользователем двумя строками и режим замера процессорного времени выполнения умножения матриц.
- В начале работы программы пользователю нужно ввести целое число — это выбор пункта меню.

К первому режиму работы программы предъявляются следующие требования:

- если пункт меню — число от 1 до 4, то умножить матриц, для этого надо ввести две матрицы;
- программа должна вывести результирующую матрицу.

Ко второму режиму работы программы предъявляются следующие требования:

- если пункт меню — число от 5 до 7, то провести замеры времени и памяти умножения матриц каждого алгоритма;
- матрицы генерируются автоматически.

2.2 Разработка алгоритмов

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц. Схема умножения матриц по алгоритму Винограда приведена на

рисунках 2.2-2.3, Схема умножения матриц по алгоритму Штрассена приведена на рисунка 2.4-2.6, схема оптимизированной версии приведена на рисунках 2.7-2.8.

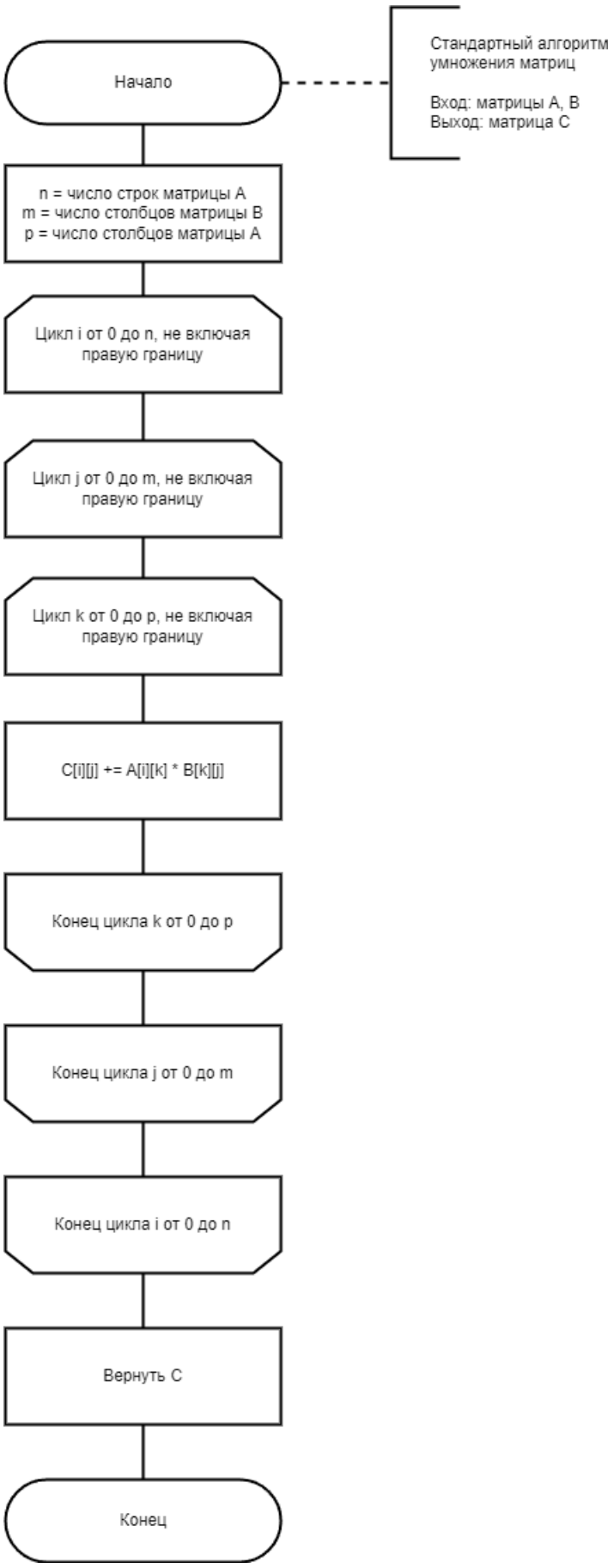

Рисунок 2.1 – Стандартный алгоритм умножения матриц

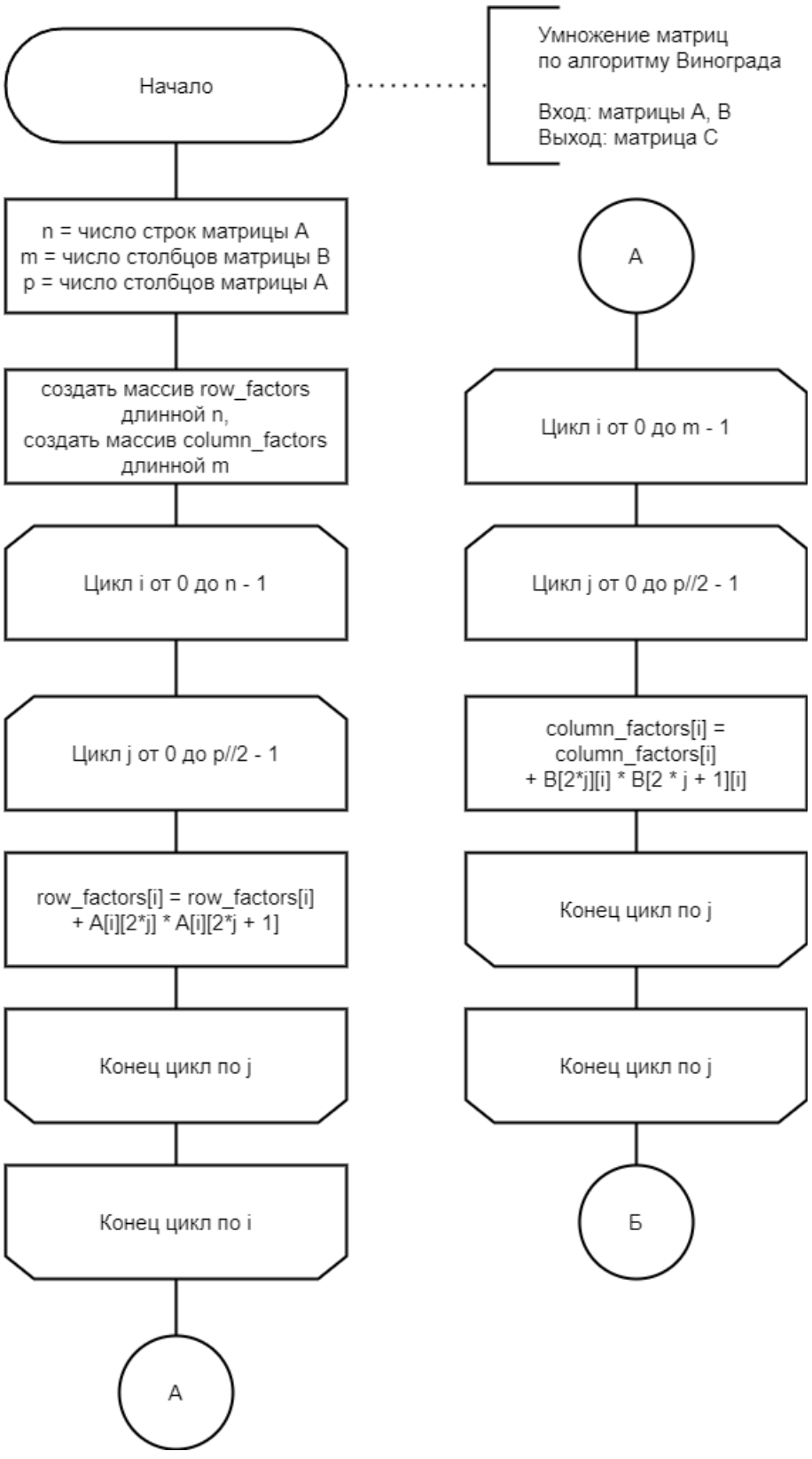

Рисунок 2.2 – Умножение матриц по алгоритму Винограда

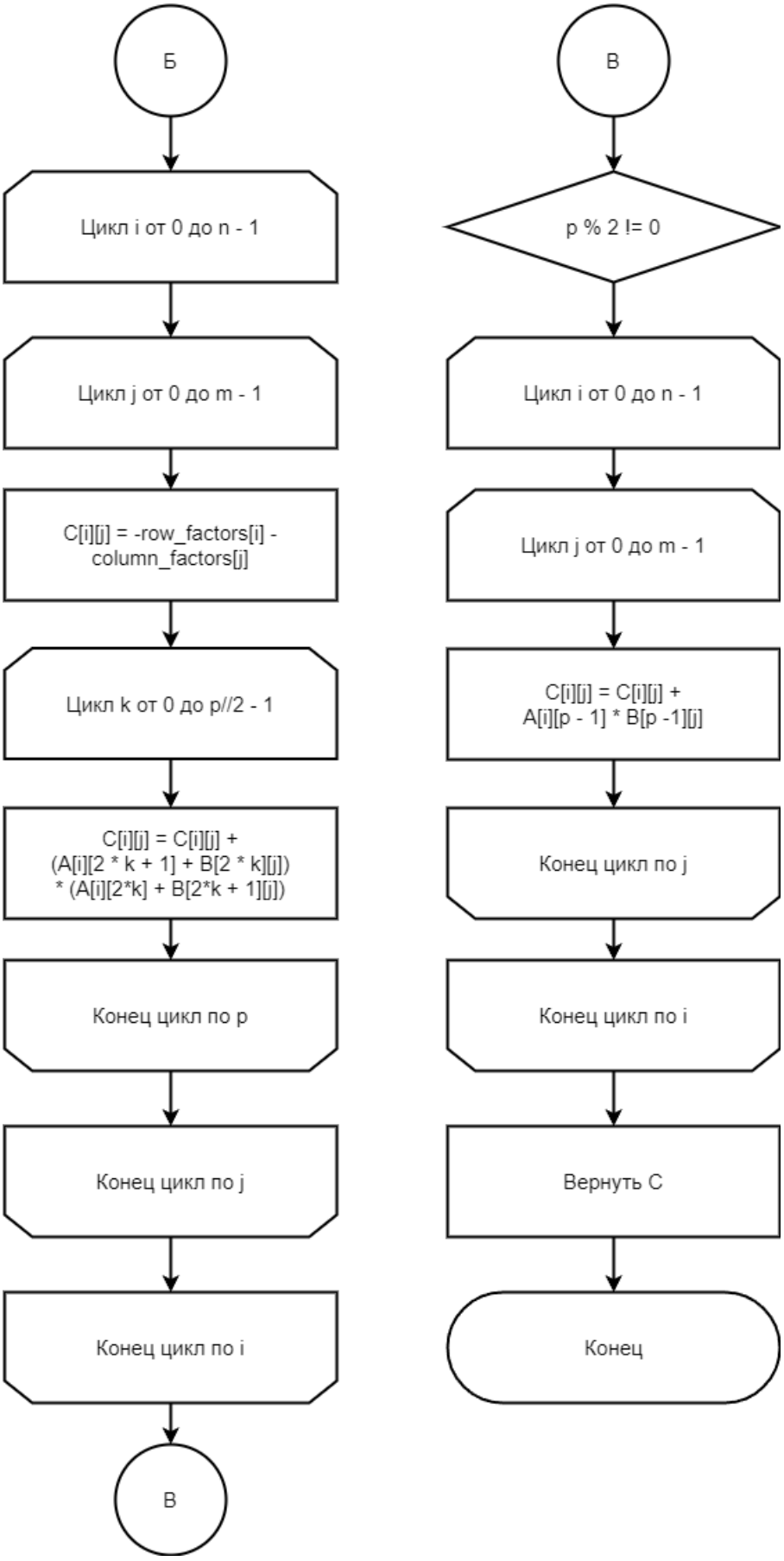

Рисунок 2.3 – Умножение матриц по алгоритму Винограда (продолжение)

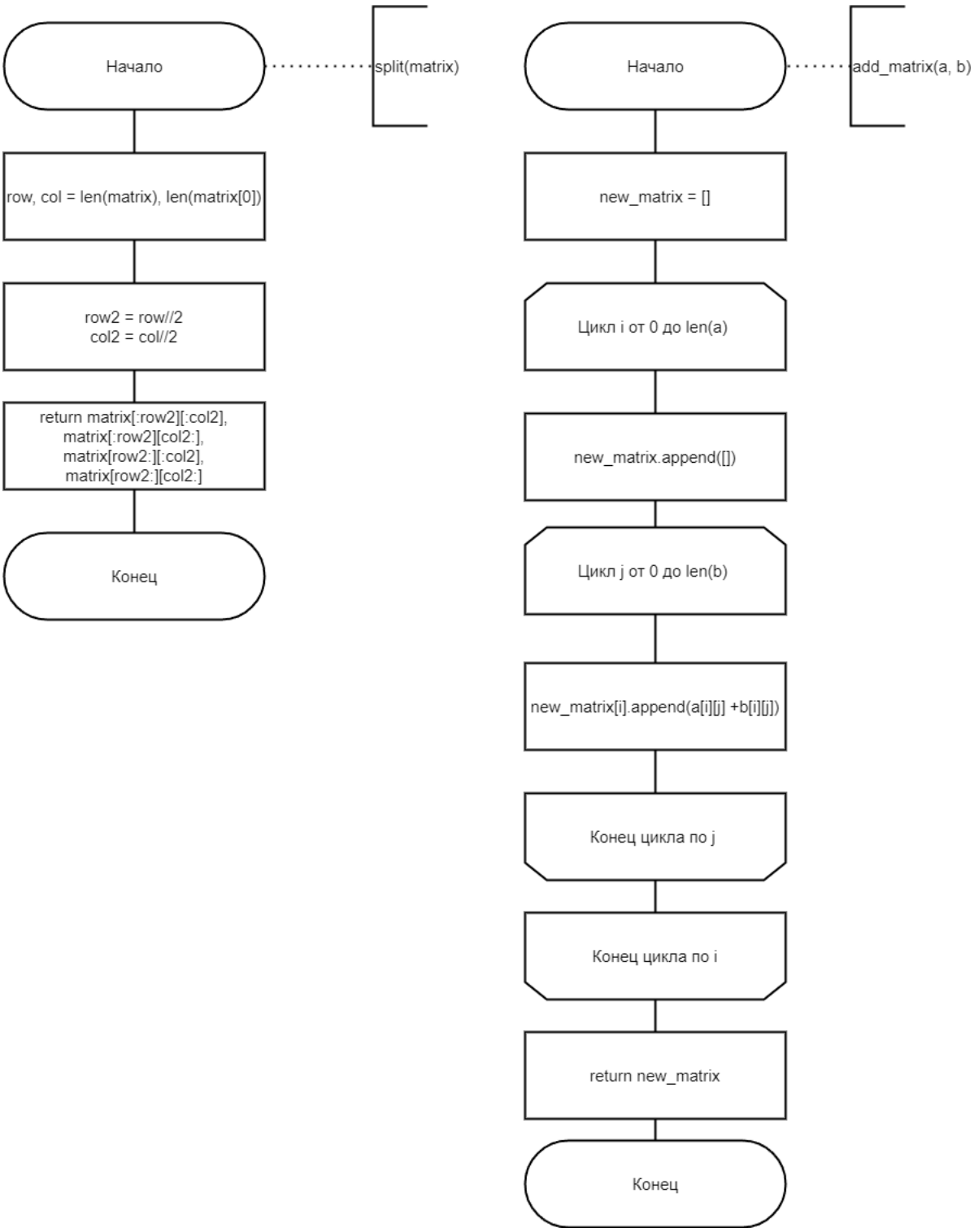

Рисунок 2.4 – Функции разделения матрицы и сложение матриц

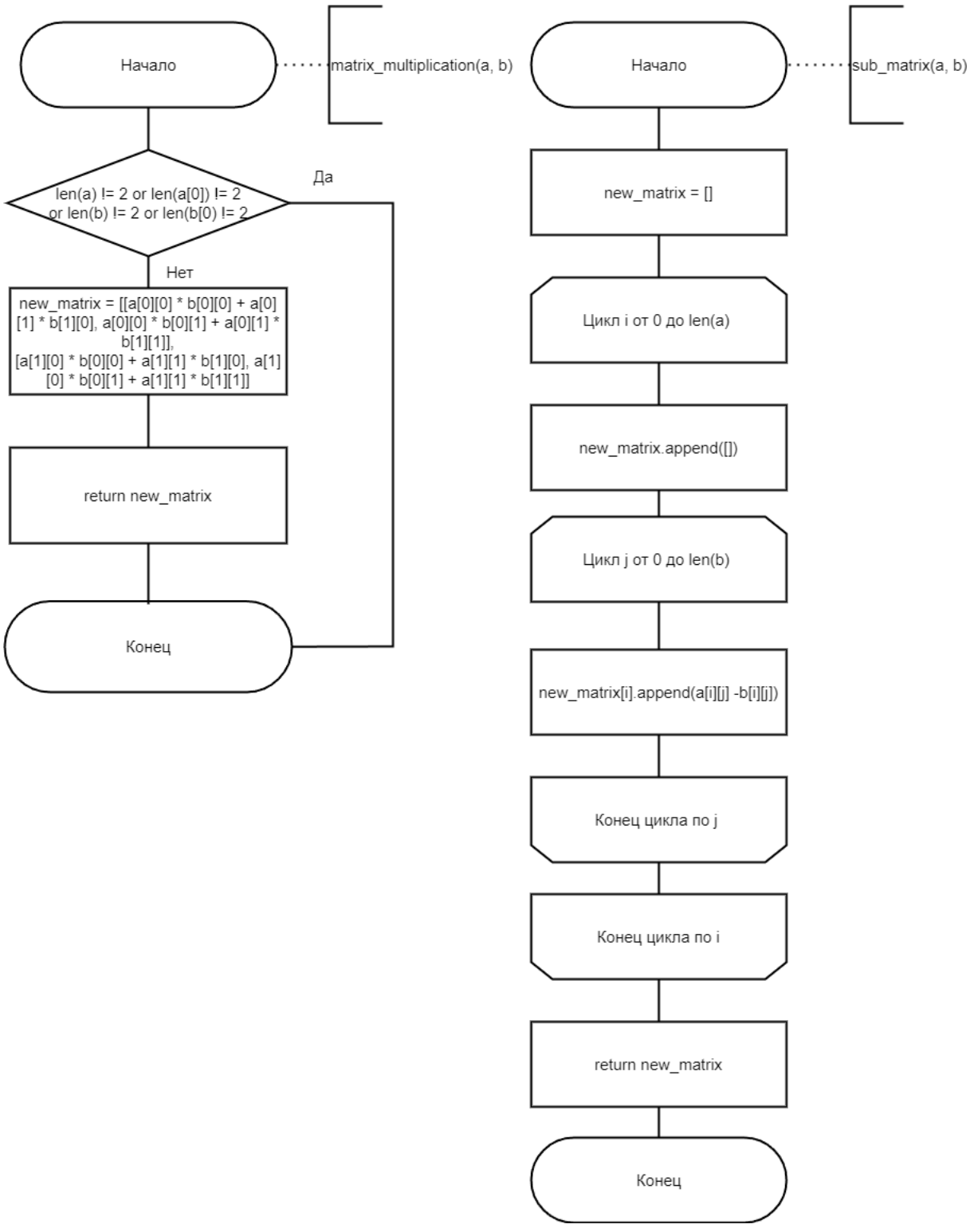

Рисунок 2.5 – Функция умножение матрица размер 2x2 и функция вычитание матриц

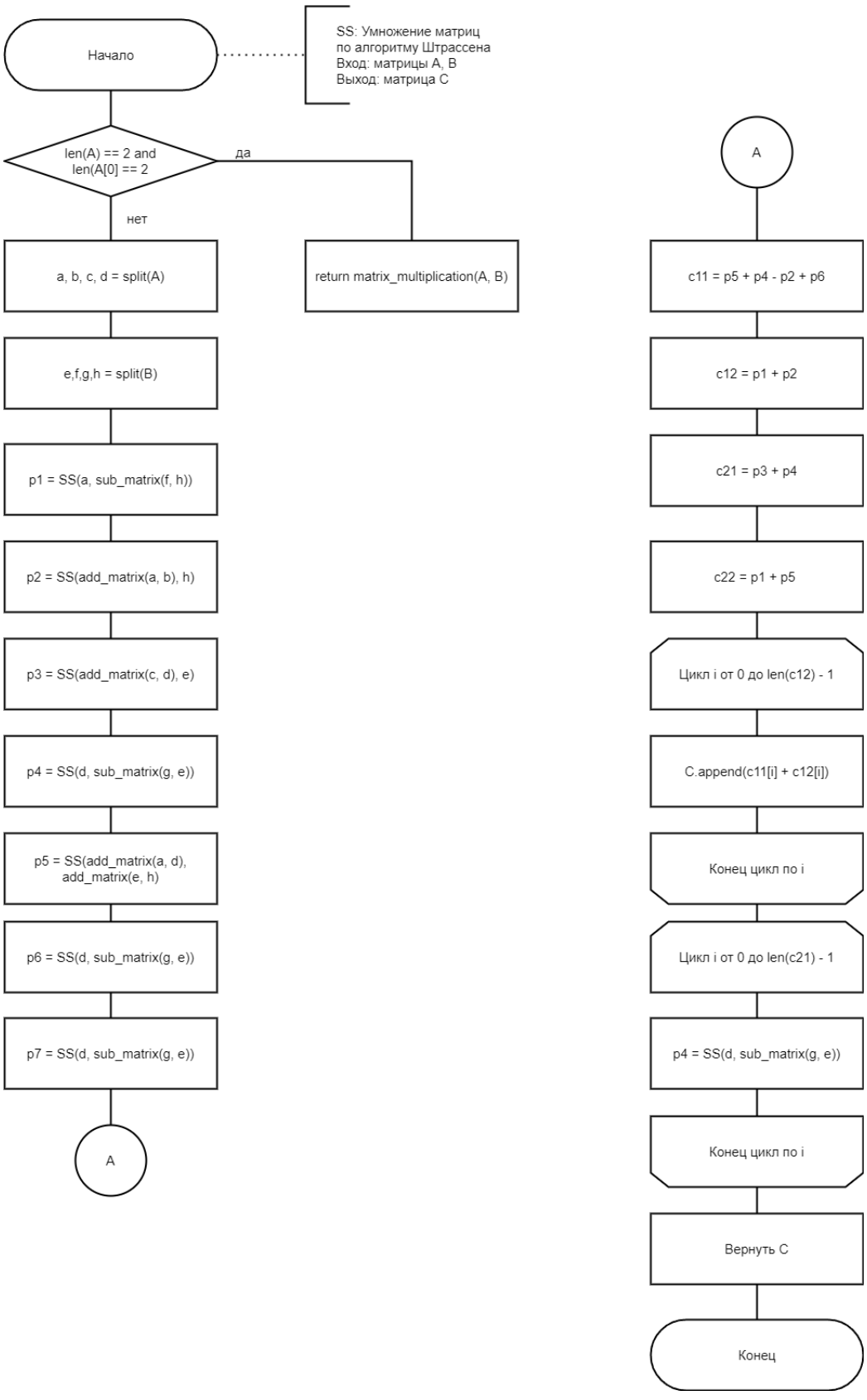

Рисунок 2.6 – Вычисление результата умножения матриц по алгоритму Штрассена

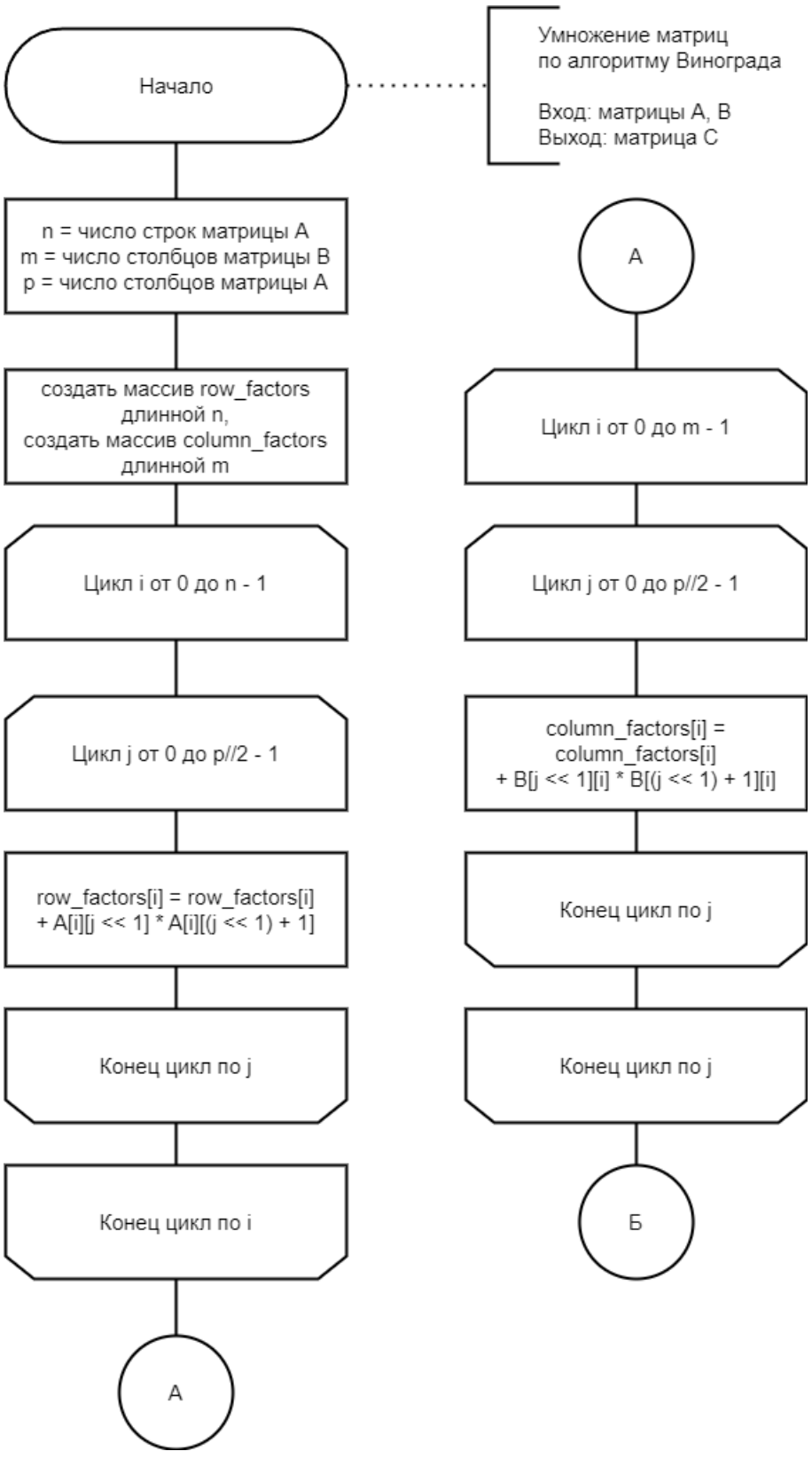

Рисунок 2.7 – Умножение матриц по оптимизированному алгоритму Винограда

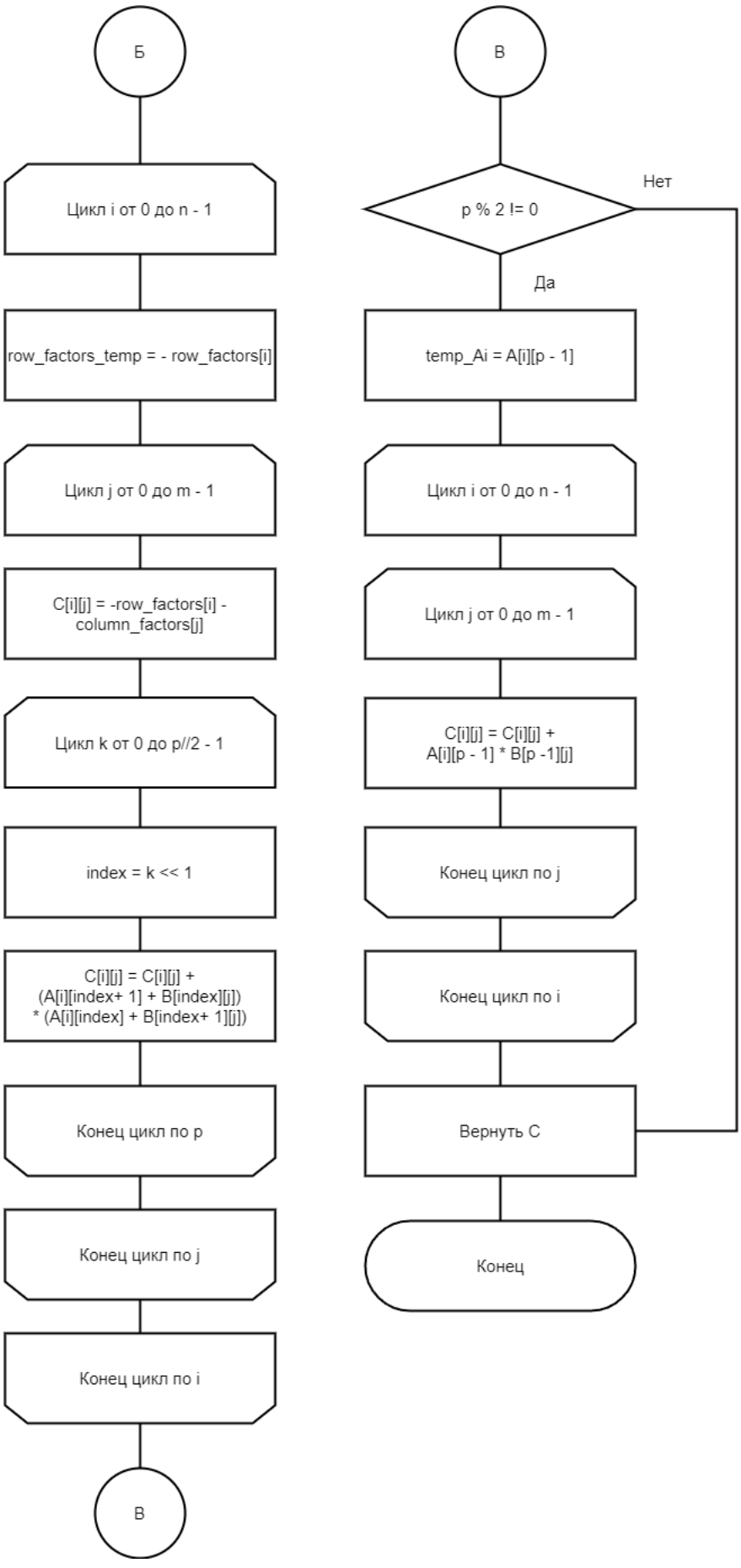

Рисунок 2.8 – Умножение матриц по оптимизированному алгоритму Винограда (продолжение)

2.3 Модель вычислений для оценки трудоемкости алгоритмов

Для определения трудоемкости алгоритмов необходимо ввести модель вычислений [4]:

1. операции из списка (2.1) имеют трудоемкость равную 2;

$$*, /, \%, * =, / =, \% = \tag{2.1}$$

2. операции из списка (2.2) имеют трудоемкость равную 1;

$$+, -, + =, - =, =, ==, ! =, <, >, <=, >=, [], ++, -- \tag{2.2}$$

3. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.3);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4. трудоемкость цикла рассчитывается, как (2.4);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

5. трудоемкость вызова функции равна 0.

2.4 Трудоемкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости для умножения матрицы A[N][P] и матрицы B[P][M].

2.4.1 Стандартный алгоритм умножения матриц

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости цикла по $k \in [0..P-1]$, равной $2+P(2+1+8+1+2) = 2+14P$;
- трудоекмости цикла по $j \in [0..M-1]$, равной $2+M(2+f_{body}) = 2+M(4+14P) = 2+4M+14PM$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $2+N(2+f_{body}) = 2+N(2+2+4M+14MP) = 2+4N+4NM+14NMP$.

Таким образом, трудоемкость стандартного алгоритма умножения равна $f_{standard} = 2+4N+4NM+14NMP \approx 14NMP$.

2.4.2 Алгоритм умножения матриц по Винограду

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости создания и заполнения нулями векторов row_factors и column_factors, равной $N+M$;
- трудоемкости предварительных вычислений для строк, равной $2+N(4+2+P/2(3+12)) = 2+6N+7.5NP$;
- трудоемкости предварительных вычислений для столбцов, равной $2+M(4+2+P/2(3+12)) = 2+6M+7.5MP$;
- трудоемкости цикла по $k \in [0..P/2-1]$, равной $4+P/2 \cdot 30 = 4+15P$;
- трудоекмости цикла по $j \in [0..M-1]$, равной $2+M(2+7+f_{body}) = 2+M(2+7+4+15P) = 2+15M+15PM$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $2+N(2+f_{body}) = 2+N(2+2+15M+14MP) = 2+4N+15NM+15NMP$;
- трудоемкости условия, равной 2;

- трудоемкости двойного цикла добавления в случае нечетного размера матрицы, равной $2 + N(2 + 2 + M(2 + 14)) = 2 + 4N + 16NM$.

Таким образом, трудоемкость алгоритма умножения по Винограду в худшем случае (нечетный размер матрицы) равна $f_{worst} = N + M + 2(2 + 6N + 7.5M) + 2 + 4N + 15NM + 15NMP + 2 + 2 + 4N + 16NM = 10 + 21N + 15M + 21NM + 15NMP \approx 15NMP$.

Трудоемкость алгоритма умножения по Винограду в лучшем случае (четный размер матрицы) равна $f_{best} = N + M + 2(2 + 6N + 7.5M) + 2 + 4N + 15NM + 15NMP = 6 + 17N + 16M + 15NM + 15NMP \approx 15NMP$.

2.4.3 Оптимизированный алгоритм умножения матриц по Винограду

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости создания и заполнения нулями векторов row_factors и column_factors, равной $N + M$;
- трудоемкости предварительных вычислений для строк, равной $2 + N(4 + 2 + P/2(2 + 11)) = 2 + 6N + 6.5NP$;
- трудоемкости предварительных вычислений для столбцов, равной $2 + M(4 + 2 + P/2(2 + 11)) = 2 + 6M + 6.5MP$;
- трудоемкости цикла по $k \in [0..P/2-1]$, равной $4+P/2(20) = 4+10P$;
- трудоекмости цикла по $j \in [0..M - 1]$, равной $2 + M(2 + 4 + f_{body})$;
- трудоемкости цикла по $i \in [0..N - 1]$, равной $2 + N(4 + f_{body})$;
- трудоемкости условия, равной 2;
- трудоемкости двойного цикла добавления в случае нечетного размера матрицы, равной $2 + N(6 + 2 + M(2 + 8)) = 2 + 8N + 10NM$.

Таким образом, трудоемкость оптимизированного алгоритма умножения по Винограду в худшем случае (нечетный размер матрицы) равна $f_{worst} = N + M + 2(1 + 4N + 6.5M) + 2 + N(4 + 2 + M(6 + 10P + 4)) = 4 + 13N + 13M + 20MN + 10MNP \approx 9.5NMP$.

Трудоемкость оптимизированного алгоритма умножения по Винограду в лучшем случае (четный размер матрицы) равна $f_{best} = N + M + 2(1 + 4N + 6.5M) + 2 + N(4 + 2 + M(6 + 10P + 4)) + 2 + 2 + 8N + 10MN = 8 + 21N + 13M + 30MN + 10MNP \approx 9.5NMP$.

2.5 Вывод

Были представлены требования к программе, схемы алгоритмов умножения матриц.

3 Технологическая часть

В данном разделе будут указаны средства реализации, будут представлены листинги кода, а также функциональные тесты.

3.1 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования Python. Выбор ЯП обусловлен простотой синтаксиса, большим числом библиотек и эффективностью визуализации данных.

Замеры времени проводились при помощи функции process_time из библиотеки time.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
- matrix.py - файл, содержащий функции работы с матрицами;
- time_test.py - файл, содержащий функции замеров времени работы указанных алгоритмов;
- graph_result.py - файл, содержащий функции визуализации временных характеристик описанных алгоритмов;
- memory.py - файл, содержащий функции замеров памяти для реализвции алгоритмов умножения матриц.

3.3 Листинги кода

Стандартный алгоритм, алгоритм Винограда, алгоритм Штрассена и оптимизированный алгоритм Винограда умножения матриц приведены в

листингах 3.1-3.4.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
def standard_mult(A, B):
n = len(A)
    m = len(B[0])
    p = len(A[0])

    C = [[0] * m for _ in range(n)]

    for i in range(n):
        for j in range(m):
            for k in range(p):
                C[i][j] = C[i][j] + A[i][k] * B[k][j]

    return C
```

Листинг 3.2 – Алгоритм Винограда умножения матриц

```
def winograd_mult(A, B):
    n = len(A)
    m = len(B[0])
    p = len(A[0])

    C = [[0] * m for _ in range(n)]

    row_factors = [0] * n

    for i in range(n):
        for j in range(p // 2):
            row_factors[i] = row_factors[i] + \
            A[i][2 * j] * A[i][2 * j + 1]

    column_factors = [0] * m

    for i in range(m):
        for j in range(p // 2):
            column_factors[i] = column_factors[i] + \
            B[2 * j][i] * B[2 * j + 1][i]

    for i in range(n):
        for j in range(m):
            C[i][j] = -row_factors[i] - column_factors[j]
            for k in range(p // 2):
                C[i][j] = C[i][j] + (A[i][2 * k + 1] + B[2 *
                   k][j]) * \
                (A[i][2 * k] + B[2 * k + 1][j])

    if p % 2 != 0:
        for i in range(n):
            for j in range(m):
                C[i][j] = C[i][j] + A[i][p - 1] * B[p - 1][j]

    return C
```

Листинг 3.3 – Алгоритм Штрассена умножения матриц

```
def split(matrix):
    row, col = len(matrix), len(matrix[0])
    row2, col2 = row//2, col//2
    matr1 = [matrix[i][:col2] for i in range(row2)]
    matr2 = [matrix[i][col2:] for i in range(row2)]
    matr3 = [matrix[i][:col2] for i in range(row2, row)]
    matr4 = [matrix[i][col2:] for i in range(row2, row)]

    return matr1, matr2, matr3, matr4

def add_matrix(a, b):
    new_matrix = []
    for i in range(len(a)):
        new_matrix.append([])
    for j in range(len(a[0])):
        new_matrix[i].append(a[i][j] + b[i][j])
    return new_matrix

def sub_matrix(a, b):
    new_matrix = []
    for i in range(len(a)):
        new_matrix.append([])
    for j in range(len(a[0])):
        new_matrix[i].append(a[i][j] - b[i][j])
    return new_matrix

def default_matrix_multiplication(a, b):
    if len(a) != 2 or len(a[0]) != 2 or len(b) != 2 or len(b[0])
       != 2:
        raise Exception("Matrices are not 2x2")
    new_matrix = [
    [a[0][0] * b[0][0] + a[0][1] * b[1][0], a[0][0] * b[0][1] +
       a[0][1] * b[1][1]],
    [a[1][0] * b[0][0] + a[1][1] * b[1][0], a[1][0] * b[0][1] +
       a[1][1] * b[1][1]],
    ]
    return new_matrix

def strassen_mult_sq(x, y):
    if len(x) == 2 and len(x[0]) == 2:
```

```
38          return default_matrix_multiplication(x, y)
39
40      a, b, c, d = split(x)
41      e, f, g, h = split(y)
42      p1 = strassen_mult_sq(a, sub_matrix(f, h))
43      p2 = strassen_mult_sq(add_matrix(a, b), h)
44      p3 = strassen_mult_sq(add_matrix(c, d), e)
45      p4 = strassen_mult_sq(d, sub_matrix(g, e))
46      p5 = strassen_mult_sq(add_matrix(a, d), add_matrix(e, h))
47      p6 = strassen_mult_sq(sub_matrix(b, d), add_matrix(g, h))
48      p7 = strassen_mult_sq(sub_matrix(a, c), add_matrix(e, f))
49
50      c11 = add_matrix(sub_matrix(add_matrix(p5, p4), p2), p6)
51      c12 = add_matrix(p1, p2)
52      c21 = add_matrix(p3, p4)
53      c22 = sub_matrix(sub_matrix(add_matrix(p1,p5), p3), p7)
54
55      C = []
56      for i in range(len(c12)):
57          C.append(c11[i] + c12[i])
58
59      for i in range(len(c21)):
60          C.append(c21[i] + c22[i])
61      return C
```

Листинг 3.4 – Оптимизированный алгоритм Винограда умножения матриц

```
def optimized_winograd_mult(A, B):
    n = len(A)
    m = len(B[0])
    p = len(A[0])

    C = [[0] * m for _ in range(n)]

    row_factors = [0] * n

    for i in range(n):
        for j in range(p // 2):
            row_factors[i] += A[i][j << 1] * A[i][(j << 1) + 1]

    column_factors = [0] * m

    for i in range(m):
        for j in range(p // 2):
            column_factors[i] += B[j << 1][i] * B[(j << 1) + 1][i]

    for i in range(n):
        row_factors_temp = -row_factors[i]
        for j in range(m):
            C[i][j] = row_factors_temp - column_factors[j]
            for k in range(p // 2):
                index = k << 1
                C[i][j] += (A[i][index + 1] + B[index][j]) *
                   (A[i][index] + B[index + 1][j])

    if p % 2 != 0:
        for i in range(n):
            temp_Ai = A[i][p - 1]
            for j in range(m):
                C[i][j] += temp_Ai * B[p - 1][j]

    return C
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы умножения матриц. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица A	Матрица B	Ожидаемый результат
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & -1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{pmatrix}$	$\begin{pmatrix} -30 & -36 & -42 \\ -66 & -81 & -96 \\ -102 & -126 & -150 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$

3.5 Вывод

Были реализованы функции алгоритмов умножения матриц. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и проведены замеры процессорного времени и предоставлена информация о технических характеристиках устройства..

4.1 Технические характеристики

Ниже представлены характеристики компьютера, на котором проводились замеры времени работы реализации алгоритмов:

- операционная система Windows 10 Домашняя 21H2;
- оперативная память 12 Гб;
- процессор Intel(R) Core(TM) i7-9750H CPU @ 2.6Гц.

Загруженность компонентов:

- процессор - 10%;
- оперативная память - 53%

4.2 Демонстрация работы программы

На рисунке 4.1-4.3 приведен пример работы программы.

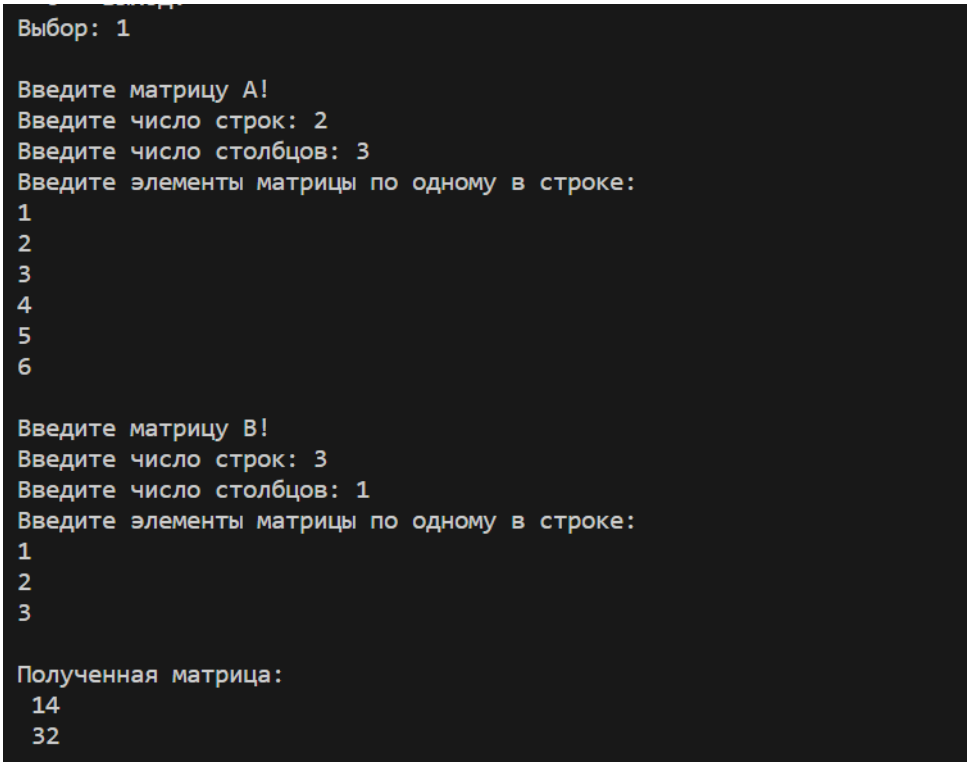

Рисунок 4.1 – Пример работы стандарного алгоритма

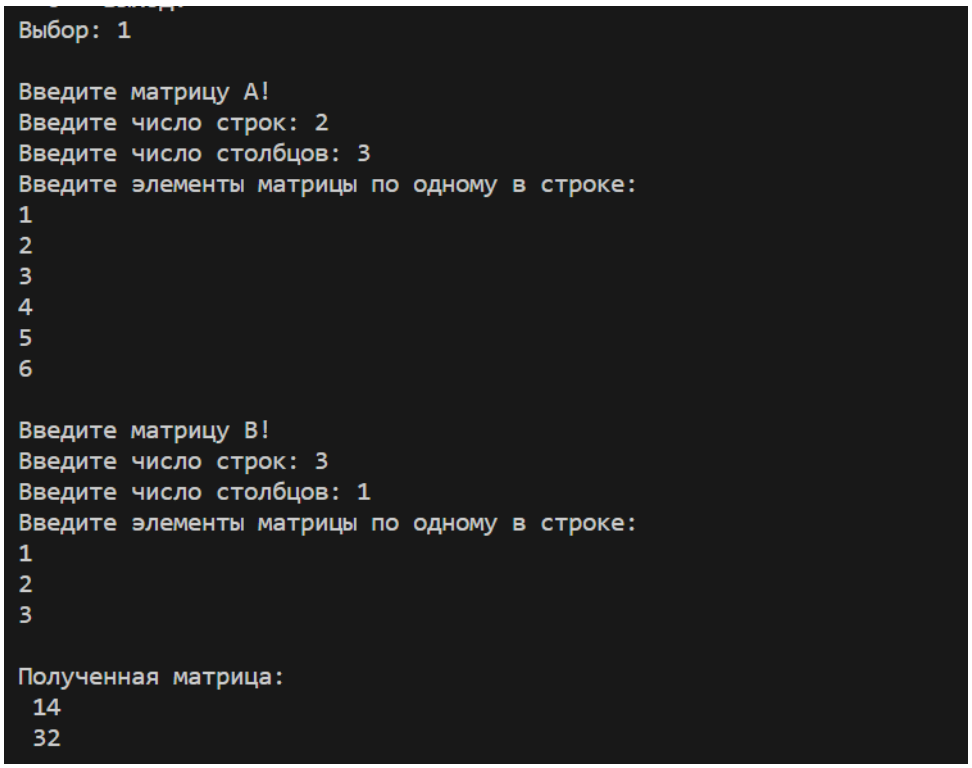

Рисунок 4.2 – Пример работы алгоритма Винограда

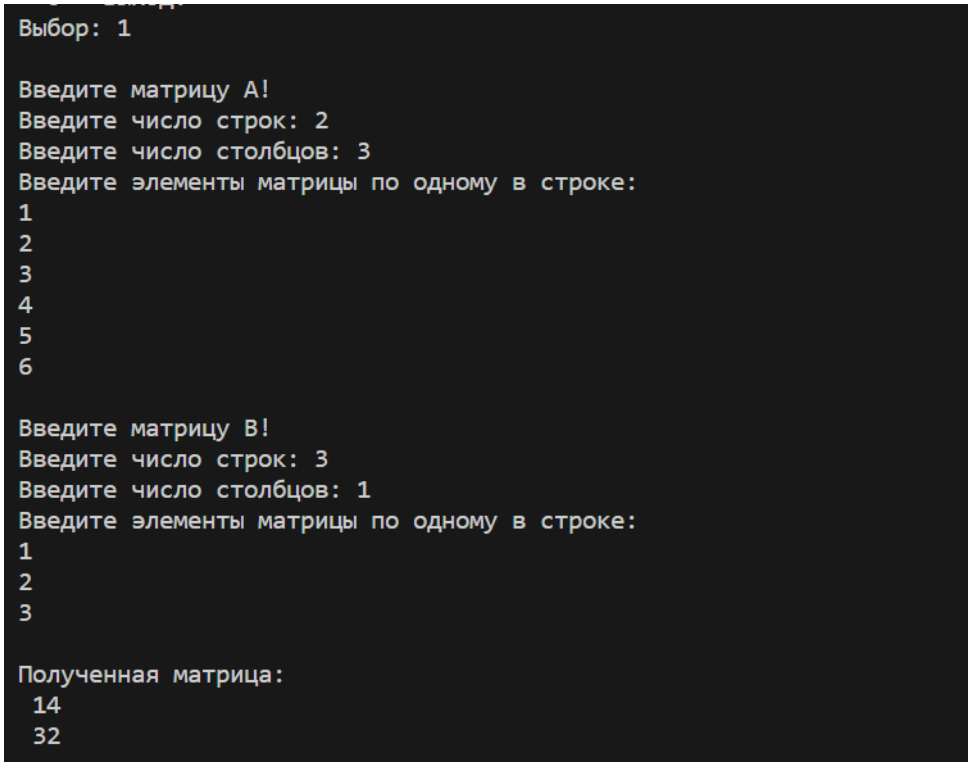

Рисунок 4.3 – Пример работы алгоритма Штрассена

4.3 Время выполнения алгоритмов

Функция process_time из библиотеки time ЯП Python возвращает процессорное время в секундах - значение типа float.

Для замера времени:

- получить значение времени до начала выполнения алгоритма, затем после её окончания. Чтобы получить результат, необходимо вычесть из второго значения первое;
- первый шаг необходимо повторить iters раз (в программе iters равно 100), суммируя полученные значения, а затем усреднить результат.

Замеры проводились для квадратных матриц целых чисел, заполненных случайным образом, размером от 10 до 100 и от 11 до 101. Результаты измерения времени для четного размера матриц приведены в таблице 4.1 (в мс).

Таблица 4.1 – Результаты замеров времени

Размер	**Выходные данные по алгоритму**			
	Стандартный	Винограда	Опт-ый Винограда	Штрассена
10	0	0	0	1.5625
20	1.5625	1.5625	0	14.0625
30	3.125	3.125	3.125	14.0625
40	6.25	6.25	7.8125	104.6875
50	13.9375	14.0625	14.0625	100
60	21.3125	21.875	20.875	101.5625
70	32.8125	34.375	32.225	734.375
80	56.4375	56.25	54.6875	700
90	70.3125	73.4375	70.3125	704.6875
100	100	106.25	98.4375	710.9375

На рисунке 4.4 - 4.5 приведены графические результаты сравнения временных характеристик для четного размера матриц.

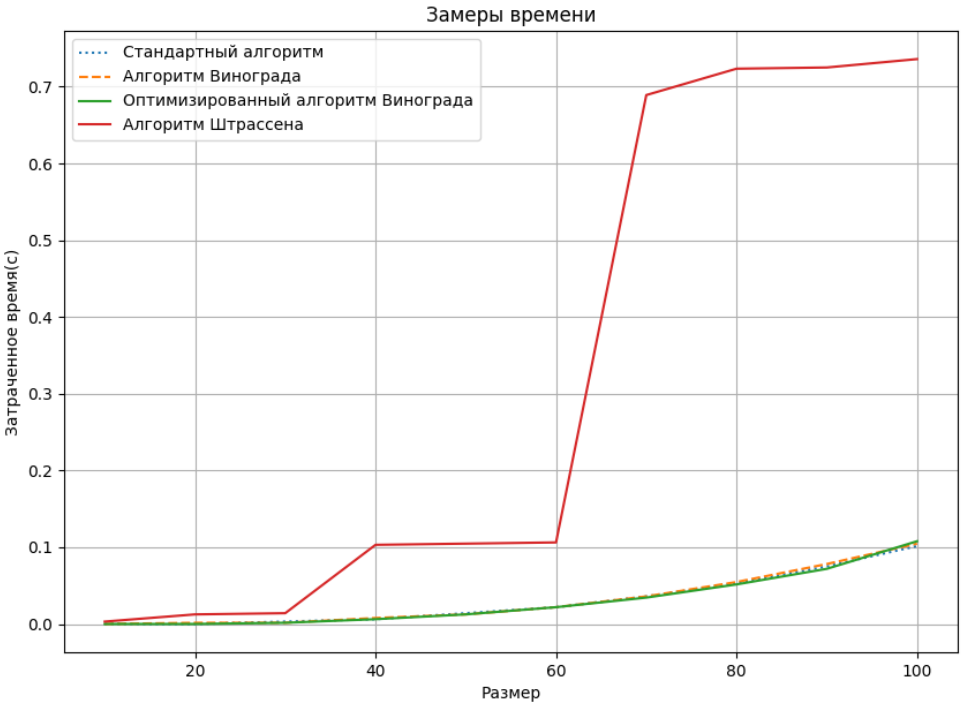

Рисунок 4.4 – Сравнение по времени алгоритмов умножения матриц четного размера

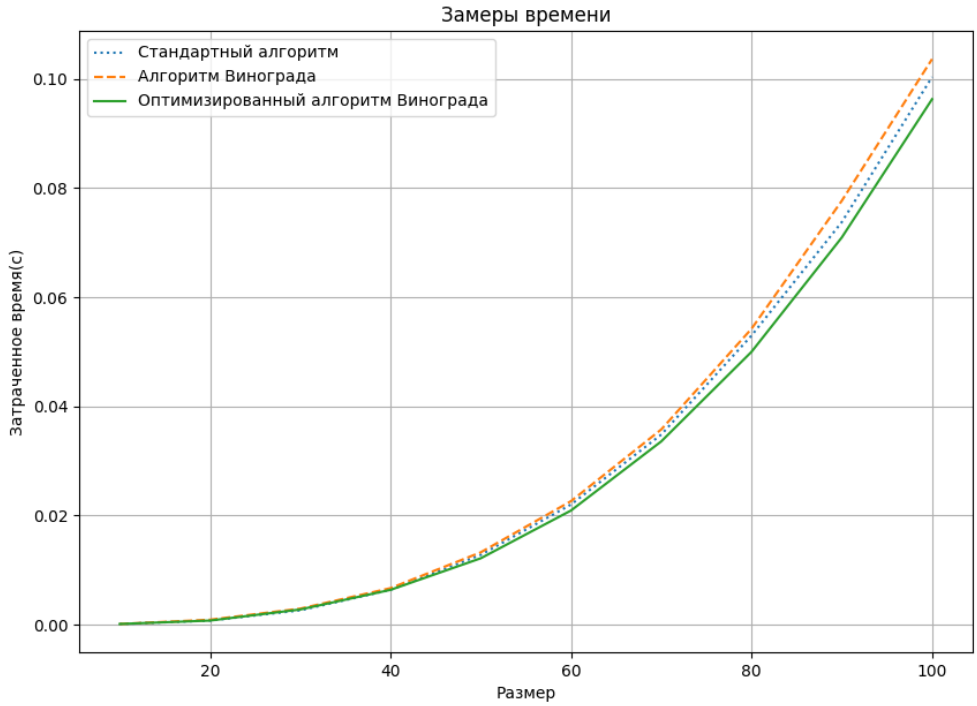

Рисунок 4.5 – Сравнение по времени алгоритмов умножения матриц четного размера

Результаты измерения времени для четного размера матриц приведены в таблице 4.2 (в мс).

Таблица 4.2 – Результаты замеров времени

Размер	**Выходные данные по алгоритму**			
	Стандартный	Винограда	Опт-ый Винограда	Штрассена
11	0.1562	0.1562	0.1562	3.125
21	1.5625	1.5625	1.5625	14.0625
31	3.125	3.125	3.125	15.625
41	6.25	7.8125	6.25	100
51	12.5	15.625	12.0625	103.125
61	21.875	25	21.875	103.125
71	35.9375	37.5	34.375	700
81	54.6875	56.25	48.4375	701.5625
91	88.0625	89.021	81.875	737.5
101	102.25	104.6875	101.5625	746.875

На рисунке 4.6-4.7 приведены графические результаты сравнения временных характеристик для нечетного размера матриц.

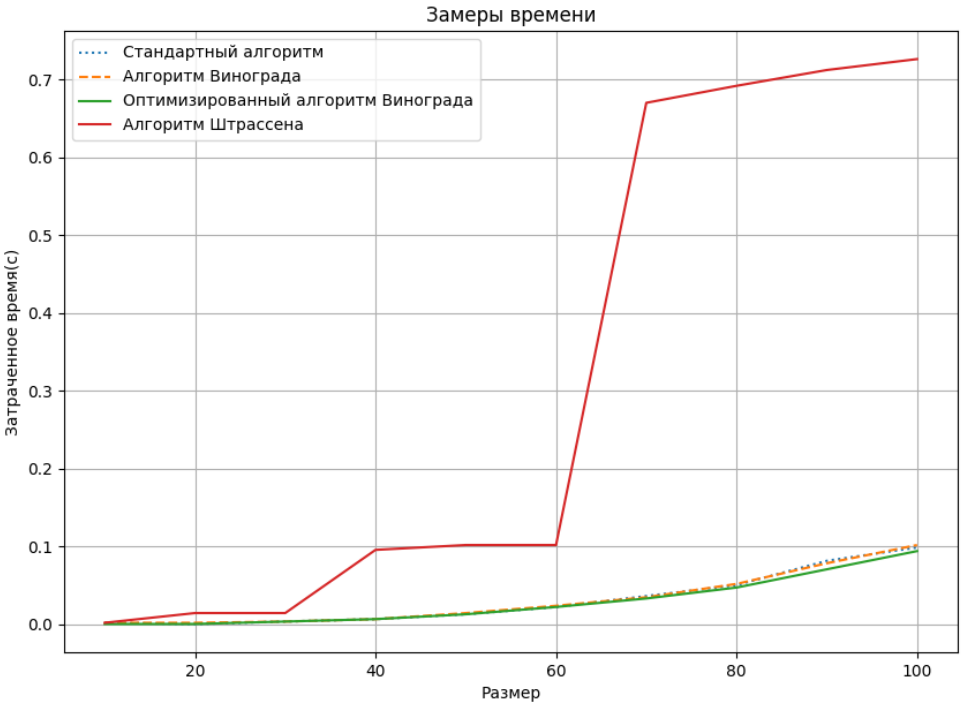

Рисунок 4.6 – Сравнение по времени алгоритмов умножения матриц нечетного размера

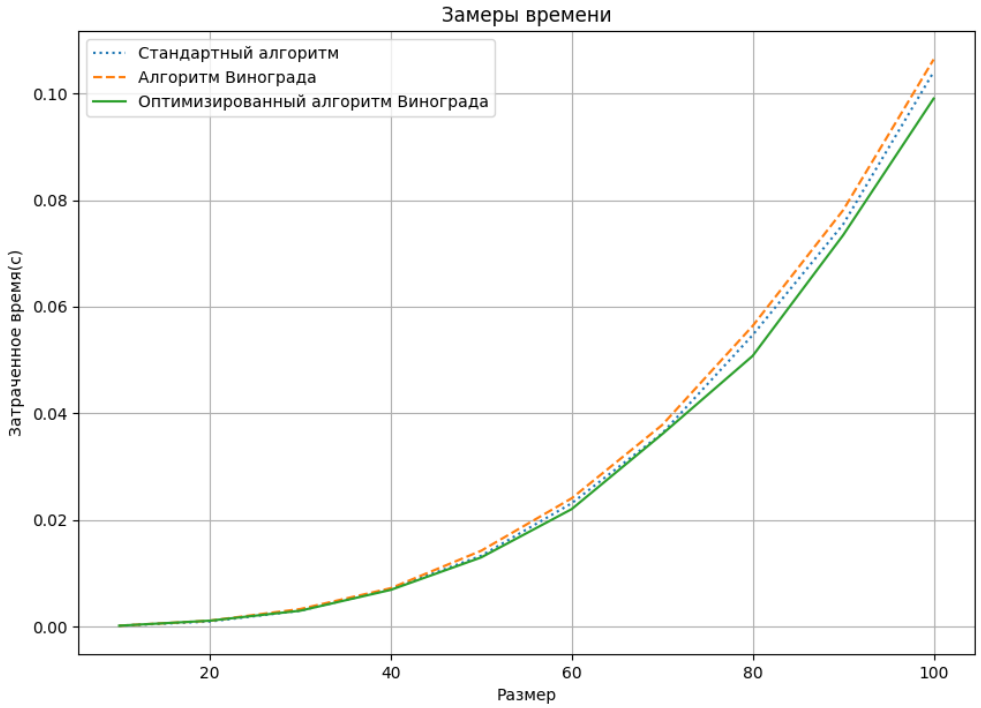

Рисунок 4.7 – Сравнение по времени алгоритмов умножения матриц нечетного размера

4.4 Использование памяти

Затраты по памяти для реализации стандарного алгоритма умножения матриц:

- матрица A: $n \cdot p \cdot \text{sizeof(int)}$;
- матрица B: $p \cdot m \cdot \text{sizeof(int)}$;
- результирующая матрица: $n \cdot m \cdot \text{sizeof(int)}$;
- размер матриц n, m, p: $3 \cdot \text{sizeof(int)}$;
- дополнительные переменные (i, j, k): $3 \cdot \text{sizeof(int)}$;
- адрес возврата.

Итого:

$$(n \cdot p + p \cdot m + n \cdot m + 6) \cdot \text{sizeof(int)} \tag{4.1}$$

Затраты по памяти для реализации алгоритма Винограда умножения матриц:

- матрица A: $n \cdot p \cdot$ sizeof(int);
- матрица B: $p \cdot m \cdot$ sizeof(int);
- результирующая матрица: $n \cdot m \cdot$ sizeof(int);
- размер матриц n, m, p: $3 \cdot$ sizeof(int);
- дополнительные переменные (i, j, k): $3 \cdot$ sizeof(int);
- дополнительные массивы row_factors, column_factors: $(n+m) \cdot$ sizeof(int);
- адрес возврата.

Итого:

$$(n \cdot p + p \cdot m + n \cdot m + 6 + (m + n)) \cdot \text{sizeof(int)} \tag{4.2}$$

Затраты по памяти для реализации оптимизированного алгоритма Винограда умножения матриц:

- матрица A: $n \cdot p \cdot$ sizeof(int);
- матрица B: $p \cdot m \cdot$ sizeof(int);
- результирующая матрица: $n \cdot m \cdot$ sizeof(int);
- размер матриц n, m, p: $3 \cdot$ sizeof(int);
- дополнительные переменные (i, j, k, row_factors_temp, temp_Ai, index): $5 \cdot$ sizeof(int);
- дополнительные переменные row_factors, column_factors: $(n + m) \cdot$ sizeof(int);
- адрес возврата.

Итого:

$$(n \cdot p + p \cdot m + n \cdot m + 8 + m + n) \cdot \text{sizeof(int)} \tag{4.3}$$

Затраты по памяти для реализации алгоритма Штрассена умножения матриц. Для работы алгоритма то матрица должен быть квадратная и размер в виде $n = 2^k$:

- матрица A: $n \cdot n \cdot \text{sizeof(int)}$;
- матрица B: $n \cdot n \cdot \text{sizeof(int)}$;
- результирующая матрица: $n \cdot n \cdot \text{sizeof(int)}$;
- размер матриц n, m, p: $3 \cdot \text{sizeof(int)}$;
- дополнительные переменные: $1 \cdot \text{sizeof(int)}$;
- дополнительные матрицы a, b, c, d, e, f, g, h, p1, p2, p3, p4, p5, p6, p7, c11, c12, c21, c22: $19 * \frac{n \cdot m}{h^2} \cdot \text{sizeof(int)}$;
- адрес возврата.

K - количество вызовов рекурсии:

$$K = log_2(N) \tag{4.4}$$

h - 4 * глубина рекурсии

$$h = 4 * k (1 <= k <= K) \tag{4.5}$$

Итого:

$$(3 \cdot n \cdot n + 4 + 19 * \frac{n \cdot n}{h^2}) \cdot \text{sizeof(int)} \tag{4.6}$$

На рисунках 4.8 также приведены результаты замеров памяти.

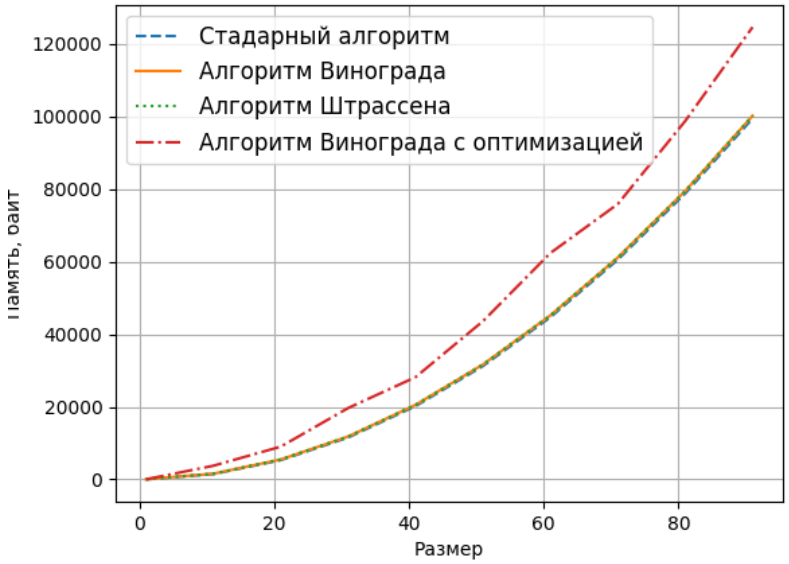

Рисунок 4.8 – Замеры памяти, которую затрачивают реализации алогритмов

4.5 Вывод

В результате эксперимента было получено, что при размере матриц большем 50, оптимизированный алгоритм Винограда работает работает быстрее стандартного алгоритма в 1.1 раза. При этом стандартный алгоритм бытсрее алгоритма Винограда в 1.25 раза. Алгоритм Штрассена работает медленее останых хотя при размере матриц большее 50 уже работает медленее 7 раза. Тогда, для размера матриц, начиная с 50 элементов, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду.

Также в результате эксперимента было установлено, что при четном размере матриц, алгоритм Винограда работает быстрее, чем на матрицах с нечетным размером в 1.2 раза в связи с проведением дополнительных вычислений для крайних строк и столбцов. Можно сделать вывод, что алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров.

Заключение

В результате исследования было получено, что при размере матриц, большем 50, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду, так как данный алгоритм работает быстрее стандартного алгоритма в 1.1 раза. При этом стандартный алгоритм быстрее алгоритма Винограда в 1.2 раза.

Кроме того алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров, так как указанный алгоритм работает в 1.2 раза быстрее, чем на матрицах с нечетным размером. Это связано с проведением дополнительных вычислений для крайних строк и столбцов.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены следующие задачи:

- были изучены классический алгоритм, алгоритм Винограда, его оптимизированная версия умножения матриц и алгоритм Штрассена;
- были разработаны изученные алгоритмы;
- был проведен сравнительный анализ реализованных алгоритмов;
- был подготовлен отчет о выполненной лабораторной работе.

Список литературы

[1] Н.Вирт Алгоритмы и структуры данных. 1989.

[2] Д. А. Погорелов, А. М. Таразанов Оптимизация классического алгоритма Винограда для перемножения матриц. 2019.

[3] Умножение матриц. [Электронный ресурс]. Режим доступа: `http://www.algolib.narod.ru/Math/Matrix.html` (дата обращения: 10.10.2023).

[4] М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. 2007.