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 Умножение матриц (сложность)

Студент Кузин А. А.

Группа ИУ7-51 Б

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

Матрица — это упорядоченный набор чисел, расположенных в виде таблицы. Этот математический объект находит широкое применение во многих областях науки, техники и естественных наук. Матрицы являются одним из основных инструментов в линейной алгебре и находят применение в решении различных математических задач.

Умножение матриц — есть операция вычисления матрицы, каждый элемент которой равен сумме произведений элементов в соответствующей строке первого множителя и столбце второго.

Целью данной лабораторной работы является анализ алгоритмов умножения матриц.

Необходимо выполнить следующие **задачи**:

1. Изучить следующие алгоритмы умножения матриц:
    - классический алгоритм умножения;
    - алгоритм Винограда;
    - алгоритм Штрассена;
    - оптимизированный алогритм Винограда
2. Реализовать алгоритмы умножения матриц;
3. Выполнить замеры затрат реализаций алгоритмов по памяти;
4. Выполнить замеры затрат реализаций алгоритмов по процессорному времени;
5. Провести сравнительных анализ алгоритмов

1 Аналитическая часть

1.1 Классический алгоритм умножения

Умножение матриц (обозначение: AB, реже со знаком умножения $A \times B$) — есть операция вычисления матрицы C, каждый элемент которой равен сумме произведений элементов в соответствующей строке первого множителя и столбце второго.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad (i = \overline{1, m}, j = \overline{1, p}), \tag{1.1}$$

Классический алгоритм реализует формулу 1.1.

1.2 Алгоритм Винограда

Рассмотрим алгоритм Винограда, имеющий асимптотическую сложность $O(n^{2,3755})$ [1].

Даны два вектора:

$$U = (u_1, u_2, u_3, u_4), \tag{1.2}$$

$$V = (v_1, v_2, v_3, v_4). \tag{1.3}$$

Их скалярное произведение равно:

$$U \times V = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4, \tag{1.4}$$

что равносильно

$$U \times V = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3). \tag{1.5}$$

Для упомянутых раннее матриц A, B и C скалярное произведение, по

замыслу Винограда, 1.4 можно свести к следующему выражению:

$$c_{ij} = \sum_{k=1}^{n/2}(a_{i,2k-1} + b_{j,2k})(a_{i,2k} + b_{j,2k-1}) - \\ - \sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k} - \sum_{k=1}^{n/2} b_{2k-1,j}b_{2k,j}. \quad (1.6)$$

Для уменьшения количества арифметических операций Виноград предложил находить второе и третье слагаемое в 1.6 заранее для каждой строки матрицы A и каждого столбца матрицы B.

Так, вычислив для i-ой строки матрицы A значение выражения
$\sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k}$
можно использовать далее n раз для нахождения элементов i-ой строки матрицы C.

Аналогично, вычислив для j-ой столбца матрицы B значение выражения

$\sum_{k=1}^{n/2} b_{2k-1,j}b_{2k,j}$
можно использовать далее n раз для нахождения элементов j-ой столбца матрицы C.

Для примера, приведенного в формуле 1.5, в классическом умножении производится четыре умножения и три сложения; в алоритме Винограда — шесть умножений и девять сложений. Но, несмотря на увеличение количества операций, выражение в правой части можно вычислить заранее и запомнить для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволит выполнить лишь два умножения и пять сложений, складывая затем только лишь с двумя предварительно вычисленными суммами соседних элементов текущих строк и столбцов. Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее классического алгоритма умножения матриц [2].

При условии нечетного размера матрицы необходимо дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.3 Алгоритм Штрассена

Если добавить к матрицам A и B одинаковые нулевые строки и столбцы, их произведение станет равно матрице AB с теми же добавленными строками и столбцами. Поэтому можно рассматривать только матрицы размера $n = 2^k,\ k \in \mathbb{N}$, а другие случаи сводить к этому добавлением нулей, отчего n может увеличиться лишь вдвое.

Пусть A, B – матрицы размера $2^k \times 2^k$. Их можно представить как блочные матрицы размера (2×2) из $(2^{k-1} \times 2^{k-1})$–матриц:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

По принципу блочного умножения, матрица AB выражается через их произведение

$$AB = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix},$$

где в правой части происходит восемь умножений матриц размера $2^{k-1} \times 2^{k-1}$. Поскольку матрицы образуют кольцо, то для вычисления правой части годится любой алгоритм умножения (2×2)-матриц, использующий лишь сложения, вычитания и умножения. Штрассен предложил такой алгоритм с семью умножениями:

$$D = (A_{11} + A_{22})(B_{11} + B_{22}); \tag{1.7}$$

$$D_1 = (A_{12} - A_{22})(B_{21} + B_{22}); \tag{1.8}$$

$$D_2 = (A_{21} - A_{11})(B_{11} + B_{12}); \tag{1.9}$$

$$H_1 = (A_{11} + A_{12})B_{22}; \tag{1.10}$$

$$H_2 = (A_{21} + A_{22})B_{11}; \tag{1.11}$$

$$V_1 = A_{22}(B_{21} - B_{11}); \tag{1.12}$$

$$V_2 = A_{11}(B_{12} - B_{22}); \tag{1.13}$$

$$\begin{aligned} AB &= \begin{pmatrix} D & 0 \\ 0 & D \end{pmatrix} + \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} + \begin{pmatrix} -H_1 & H_1 \\ H_2 & -H_2 \end{pmatrix} + \begin{pmatrix} V_1 & V_2 \\ V_1 & V_2 \end{pmatrix} \\ &= \begin{pmatrix} D + D_1 + V_1 - H_1 & V_2 + H_1 \\ V_1 + H_2 & D + D_2 + V_2 - H_2 \end{pmatrix} \end{aligned} \tag{1.14}$$

Каждое умножение можно совершать рекурсивно по той же процедуре, а сложение – тривиально, складывая $(2^{k-1})^2$ элементов. Тогда время работы алгоритма $T(n)$ оценивается через рекуррентное соотношение:

$T(n) = 7T(n/2) + O(n^2) = O(n^{\log_2 7})$.

1.4 Оптимизированный алогритм Винограда

В исследовательских целях, выполним следующие следующие оптимизации для алгоритма Винограда:

1. сохранять значение $n/2$, используемое в качестве ограничения цикла подсчета предварительных данных;
2. операцию умножения на 2 заменить на побитовый сдвиг влево на 1;
3. заменить операцию x = x + k на x += k

Вывод

В данном разделе были рассмотрены следующие алгоритмы умножения матриц:

- классический алгоритм умножения;
- алгоритм Винограда;
- алгоритм Штрассена;
- оптимизированный алогритм Винограда

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов умножения матриц, описание используемых типов данных и структуры программного обеспечения.

2.1 Требования к программному обеспечению

К программе предъявлен ряд функциональных требований:

- наличие интерфейса для выбора действий;
- возможность ввода чисел, используя текстовые файлы;
- возможность произвести замеры процессорного времени работы;реализованных алгоритмов умножения матриц

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма классического умножения.

На рисунках 2.2–2.2 приведена схема алгоритма Винограда.

На рисунках 2.4–2.5 представлена схема оптимизированного алгоритма Винограда.

На рисунке 2.6 представлена схема алгоритма Штрассена.

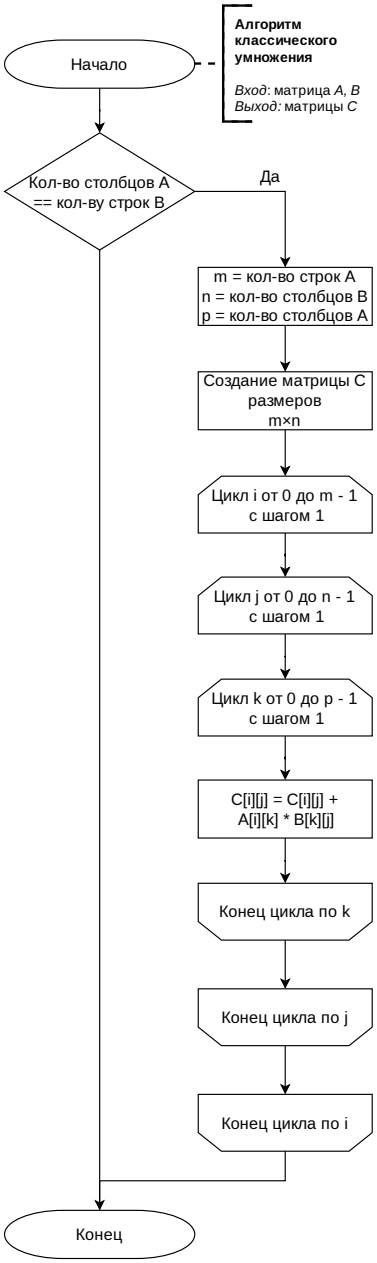

Рисунок 2.1 – Схема классического алгоритма умножения матриц

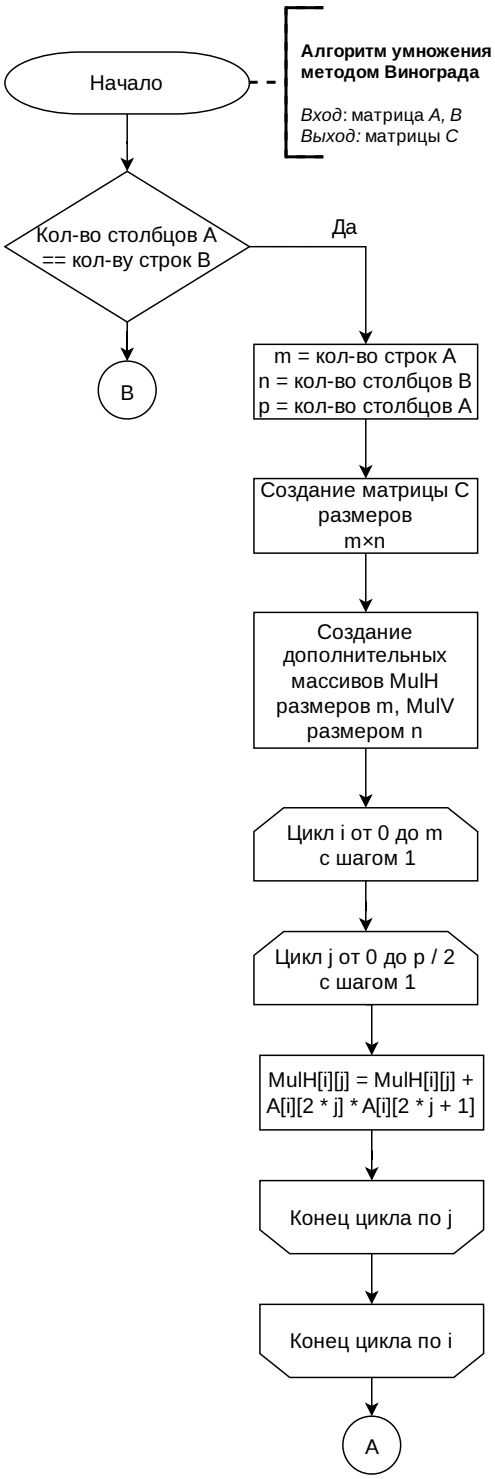

Рисунок 2.2 – Схема алгоритма Винограда (1)

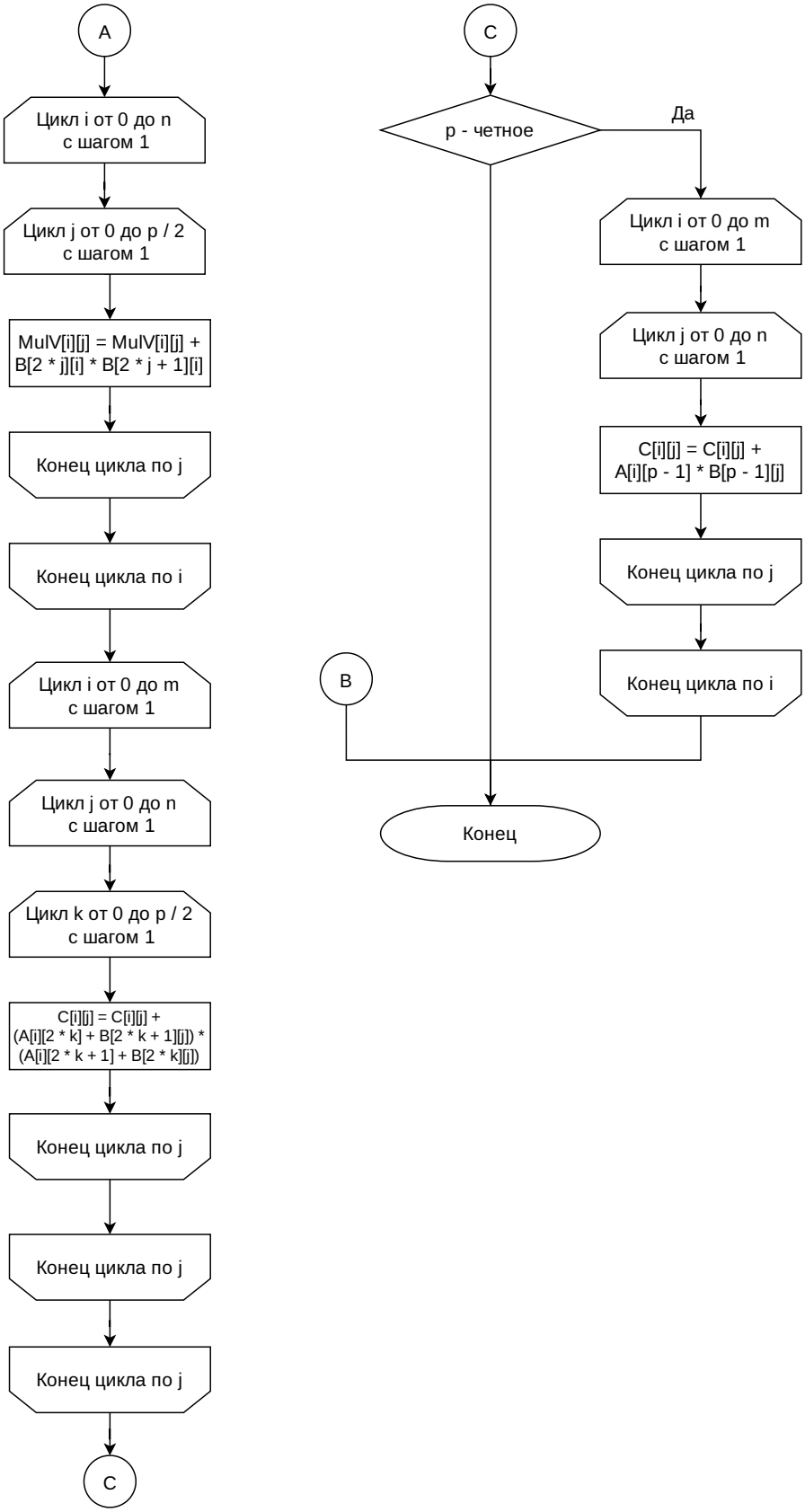

Рисунок 2.3 – Схема алгоритма Винограда (2)

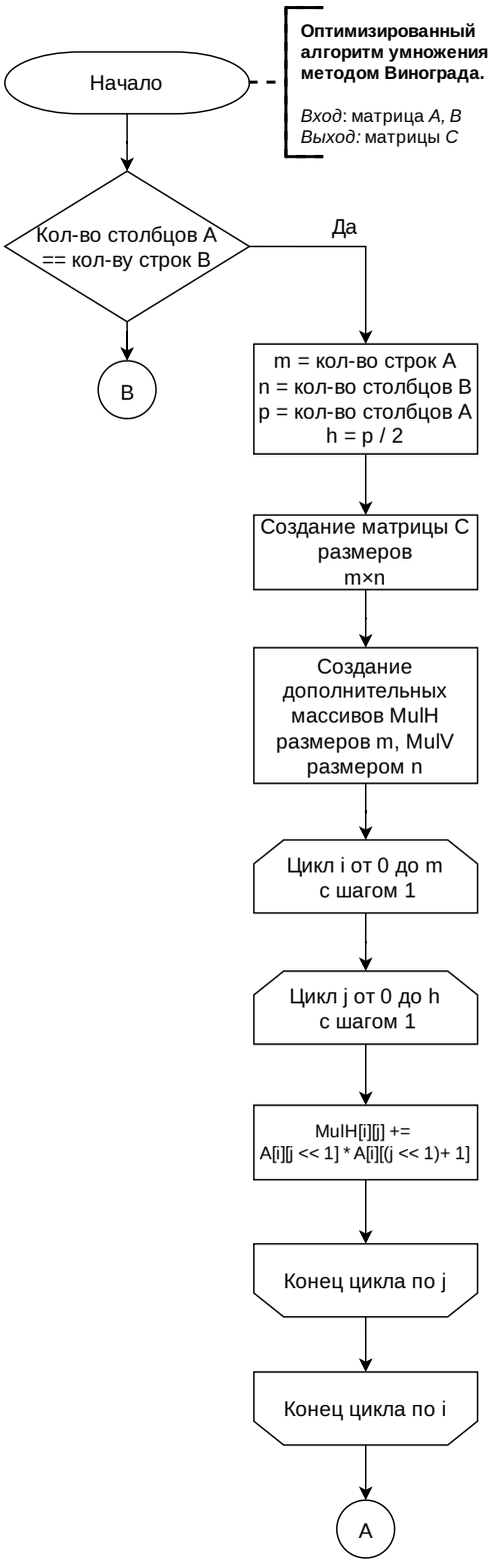

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда (1)

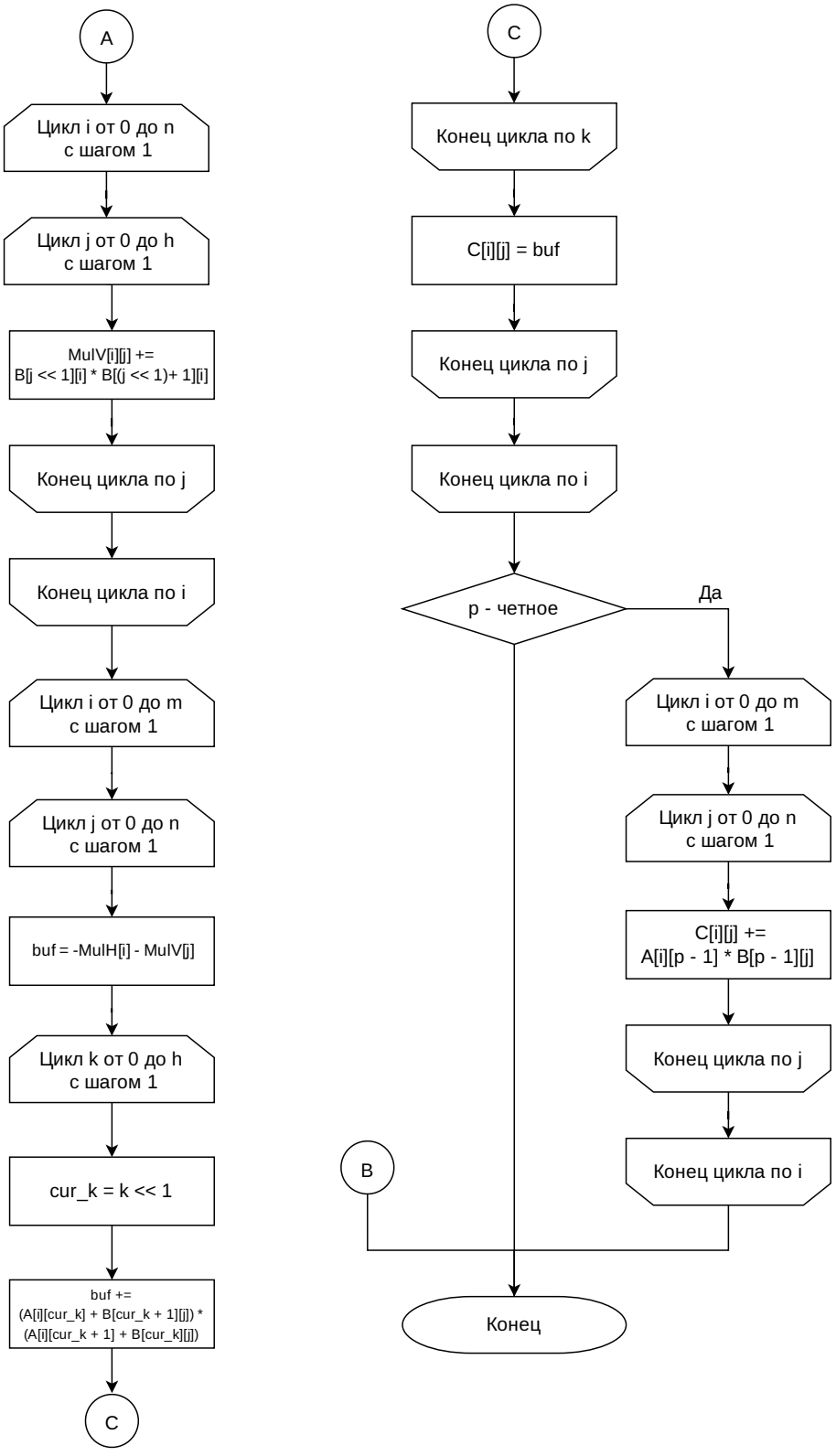

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда (2)

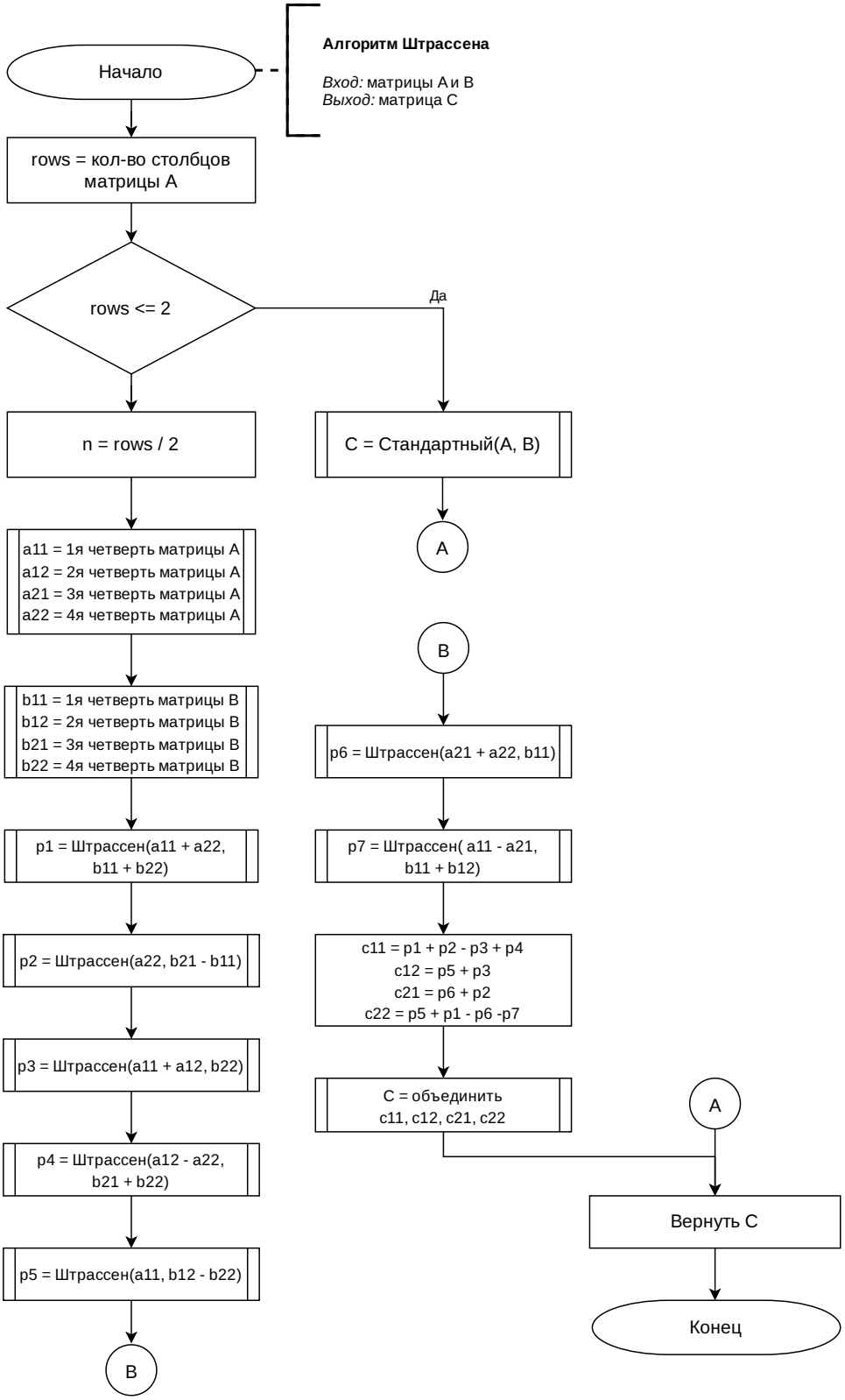

Рисунок 2.6 – Схема алгоритма Штрассена

2.3 Описание используемых типов данных

При реализации алгоритмов будет использована *матрица* — двумерный массив `целочисленного типа`.

2.4 Модель вычислений для проведения оценки трудоемкости алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка 2.1 имеют трудоемкость **1**;

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

2. операции из списка 2.2 имеют трудоемкость **2**;

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

3. трудоемкость условного оператора `if условие then A else B` рассчитывается как 2.3;

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{в случае выполнеия условия,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4. трудоемкость цикла рассчитывается как 2.4

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}); \tag{2.4}$$

5. трудоемкость вызова функции равна 0.

2.5 Трудоемкость алгоритмов

В следующих частях будут приведены рассчеты трудоемкостей алгоритмов для умножения матриц.

Пусть у нас есть 2 матрицы:

1. A размером $M \times P$;
2. B размером $P \times N$.

2.5.1 Стандартный алгоритм

Трудоемкость стандартного алгоритма умножения матриц состоит из:

- внешнего цикла по $i \in [1 \dots M]$, трудоемкость которого: $f = 2 + M \cdot (2 + f_{body})$;
- цикла по $j \in [1 \dots N]$, трудоемкость которого: $f = 2 + N \cdot (2 + f_{body})$;
- цикла по $k \in [1 \dots P]$, трудоемкость которого: $f = 2 + K \cdot (2 + 12)$;

Так как трудоемкость стандартного алгоритма равно трудоемкости внешнего цикла — можно вычислить ее, подставив циклы тела:

$$\begin{aligned} f_{standart} &= 2 + M \cdot (2 + 2 + N \cdot (2 + 2 + P \cdot (2 + 8 + 1 + 1 + 2))) = \\ &= 2 + 4M + 4MN + 14MNP \approx 14MNP = O(N^3). \end{aligned} \tag{2.5}$$

2.5.2 Алгоритм Винограда

При вычислении трудоемкости алгоритма Винограда необходимо учесть следующее:

- трудоемкость создания и инициализации массивов $MulH$ и $MulV$:

$$f_{init} = f_{MulH} + f_{MulV}; \tag{2.6}$$

- трудоемкость заполнения массива $MulH$:

$$f_{MulH} = 2 + M \cdot (2 + 4 + \frac{P}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 6M + \frac{19MP}{2}; \tag{2.7}$$

- трудоемкость заполнение массива $MulV$:

$$f_{MulV} = 2 + N \cdot (2 + 4 + \frac{P}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 6N + \frac{19NP}{2}; \tag{2.8}$$

- трудоемкость цикла заполения для четных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (13 + \frac{P}{2} \cdot 32)) = 2 + 4M + \\ +13MN + \frac{32MNP}{2} = 2 + 4M + 13MN + 16MNP; \tag{2.9}$$

- трудоемкость дополнительного цикла, в случае нечетного размера матрицы:

$$f_{check} = 3 + \begin{cases} 0, & \text{четная} \\ 2 + M \cdot (4 + N \cdot (2 + 14)), & \text{иначе.} \end{cases} \tag{2.10}$$

В итоге, для худшего случая (т. е. когда размер матрицы нечетный) получаем следующую трудоемкость:

$$f_{worst} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx 16MNP = O(N^3). \tag{2.11}$$

Для лучшего случая (т. е. когда размер матрицы четный):

$$f_{best} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx 16MNP = O(N^3). \tag{2.12}$$

2.5.3 Оптимизированный алгоритм Винограда

Итоговая трудоемкость оптимизированного алгоритма Винограда состоит из:

- трудоемкости кеширования значения $\frac{P}{2} - 3$;

- трудоемкость заполнения массива $MulH$:

$$f_{MulH} = 2 + M \cdot (2 + 2 + \frac{P}{2} \cdot (2 + 5 + 1 + 2 + 3)) = \\ = 2 + 2M + \frac{13MP}{2}; \tag{2.13}$$

- трудоемкость заполнение массива $MulV$:

$$f_{MulV} = 2 + N \cdot (2 + 2 + \frac{P}{2} \cdot (2 + 5 + 1 + 2 + 3)) = \\ = 2 + 2N + \frac{13NP}{2}; \tag{2.14}$$

- трудоемкость цикла заполения для четных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (2 + 10 + 2 + \frac{P}{2} \cdot 19)) = \\ = 2 + 4M + 14MN + \frac{19MNP}{2}; \tag{2.15}$$

- трудоемкость дополнительного цикла, в случае нечетного размера матрицы:

$$f_{check} = 3 + \begin{cases} 0, & \text{четная} \\ 2 + M \cdot (4 + N \cdot (2 + 11)), & \text{иначе.} \end{cases} \tag{2.16}$$

В итоге, для худшего случая (т. е. когда размер матрицы нечетный) получаем следующую трудоемкость:

$$f_{worst} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx \frac{19MNP}{2} = O(N^3). \tag{2.17}$$

Для лучшего случая (т. е. когда размер матрицы четный):

$$f_{best} = f_{MulH} + f_{MulV} + f_{cycle} + f_{check} \approx \frac{19MNP}{2} = O(N^3). \tag{2.18}$$

2.5.4 Алгоритм Штрассена

Если $M(n)$ — количество умножений, выполняемых алгоритмом для умножения двух матриц размером $n \times n$ (где n — степень двойки), то получим следующее рекурретное соотношение для $M(n)$:

$$M(n) = 7M(\frac{n}{2}). \tag{2.19}$$

При $n > 1, M(1) = 1$. Поскольку $n = 2^k$,

$$\begin{aligned} M(2^k) = 7M(2^{k-1}) = 7 \cdot [7M(2^{k-2})] = 7^2 M(2^{k-2}) = \cdots = \\ = 7^i M(2^{k-i}) = \cdots = 7^k M(2^{k-k}) = 7^k. \end{aligned} \tag{2.20}$$

Подставляя $k = \log_2 n$, получаем:

$$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}, \tag{2.21}$$

что меньше, чем n_3, необходимое для стандартного алгоритма.

Также необходимо рассмотреть количество сложений $A(n)$, выполняемых алгоритмом. Для умножения двух матриц порядка $n > 1$ алгоритму требуется **7** умножений и **18** сложений матриц размером $\frac{n}{2} \times \frac{n}{2}$. Это сводится к следующему реккурентному уравнению:

$$A(n) = 7 \cdot A \cdot \frac{n}{2} + 18 \cdot (\frac{n}{2})^2. \tag{2.22}$$

При $n > 1, A(1) = 0$.

Итоговую трудоемкость можно рассчитать как:

$$T(n) = A(n) + M(n). \tag{2.23}$$

Вывод

В данном разделе на основе теоретических данных были перечислены требования к ПО, построены схемы реализуемых алгоритмов и представлена модель вычислений на основе данных, полученных на этапе анализа.

3 Технологическая часть

В данном разделе будут приведены средства реализации, листинг кода и функциональные тесты.

3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [3], так как в нем имеется контейнер `std::string`, представляющий собой массив символов `char`, и библиотека `<ctime>` [4], позволяющая производить замеры процессорного времени.

В качестве среды для написания кода был выбран *Visual Studio Code* за счет того, что она предоставляет функционал для проектирования, разработки и отладки ПО.

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

- `main.cpp` — точка входа программы, пользовательское меню;
- `general` — модуль с определением матрицы;
- `algo` — модуль с реализациями алгоритмов умножения матриц;
- `measurement` — модуль с реализаций функции подсчета затрачиваемого времени

3.3 Реализация алгоритмов

Далее будут представлены реализацим следующих алгоритмов умножения матриц:

- `main.cpp` — Классический алгоритм умножения матриц;
- `general` — Алгоритм Винограда;
- `algo` — Оптимизированный алгоритм Винограда;
- `measurement` — Алгоритм Штрассена

Листинг 3.1 – Классический алгоритм умножения матриц

```
1  Matrix mult_classic(const Matrix& matrix1, const Matrix& matrix2)
2  {
3       int rows = matrix1.height,
4           columns = matrix2.width,
5           tmp = matrix1.width;
6
7      Matrix res{rows, columns, 0};
8
9      for (int i = 0; i < rows; ++i) {
10
11         for (int j = 0; j < columns; ++j) {
12
13             for (int k = 0; k < columns; ++k) {
14                 res[i][j] = res[i][j] + matrix1[i][k] *
                       matrix2[k][j];
15             }
16         }
17     }
18
19     return res;
20 }
```

Листинг 3.2 – Алгоритм Винограда

```
1  Matrix mult_vinograd(const Matrix& matrix1, const Matrix& matrix2)
2  {
3      int rows = matrix1.height;
4
5      Matrix res{rows, rows};
6
7      vector<int> mulH(rows, 0);
8      vector<int> mulV(rows, 0);
9
10     for (int i = 0; i < rows; ++i) {
11
12         for (int j = 0; j < rows / 2; ++j)
13             mulH[i] = mulH[i] + matrix1[i][j * 2] * matrix1[i][j
                   * 2 + 1];
14     }
15
16     for (int i = 0; i < rows; ++i) {
17
18         for (int j = 0; j < rows / 2; ++j)
19             mulV[i] = mulV[i] +  matrix2[j * 2][i] * matrix2[j *
                   2 + 1][i];
20     }
21
22     for (int i = 0; i < rows; ++i) {
23
24         for (int j = 0; j < rows; ++j) {
25
26             res[i][j] = -mulH[i] - mulV[j];
27
28             for (int k = 0; k < rows / 2; ++k)
29                 res[i][j] = res[i][j] + (matrix1[i][k* 2] +
                       matrix2[k* 2 + 1][j]) * (matrix1[i][k* 2 + 1]
                       + matrix2[k* 2][j]);
30         }
31     }
32
33     if (rows % 2) {
34
35         for (int i = 0; i < rows; ++i) {
36
```

```
37              for (int j = 0; j < rows; ++j)
38                  res[i][j] = res[i][j] + matrix1[i][rows - 1] *
                        matrix2[rows - 1][j];
39          }
40      }
41
42      return res;
43  }
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
1 int recurs_dam_lev_temp(const char*str1, const char*str2, int
    length1, int length2)
2 Matrix mult_vinograd_opt(const Matrix& matrix1, const Matrix&
    matrix2)
3 {
4     int rows = matrix1.height;
5
6     Matrix res{rows, rows, 0};
7
8     vector<int> mulH(rows, 0);
9     vector<int> mulV(rows, 0);
10
11    int stepHalf = rows / 2;
12
13    for (int i = 0; i < rows; ++i) {
14
15        for (int j = 0; j < stepHalf; ++j)
16            mulH[i] = mulH[i] + matrix1[i][(j << 1)] *
                matrix1[i][(j << 1) + 1];
17    }
18
19    for (int i = 0; i < rows; ++i) {
20
21        for (int j = 0; j < stepHalf; ++j)
22            mulV[i] = mulV[i] +  matrix2[(j << 1)][i] *
                matrix2[(j << 1) + 1][i];
23    }
24
25    for (int i = 0; i < rows; ++i) {
26
27        for (int j = 0; j < rows; ++j) {
28
29            res[i][j] = -mulH[i] - mulV[j];
30
31            for (int k = 0; k < stepHalf; ++k)
32                res[i][j] = res[i][j] + (matrix1[i][(k << 1)] +
                    matrix2[(k << 1) + 1][j]) * (matrix1[i][(k <<
                    1) + 1] + matrix2[(k << 1)][j]);
33        }
34    }
```

```
35
36      if (rows % 2) {
37
38          for (int i = 0; i < rows; ++i) {
39
40              for (int j = 0; j < rows; ++j)
41                  res[i][j] = res[i][j] + matrix1[i][rows - 1] *
                        matrix2[rows - 1][j];
42          }
43      }
44
45      return res;
46  }
```

Листинг 3.4 – Алгоритм Штрассена

```
Matrix mult_strassen(const Matrix& matrix1, const Matrix& matrix2)
{
    int rows = matrix1.height;
    int n = rows / 2;

    if (rows <= 2)
        return mult_classic(matrix1, matrix2);

    auto a11 = matrix1.slice(0, n, 0, n);
    auto a12 = matrix1.slice(0, n, n, rows);
    auto a21 = matrix1.slice(n, rows, 0, n);
    auto a22 = matrix1.slice(n, rows, n, rows);

    auto b11 = matrix2.slice(0, n, 0, n);
    auto b21 = matrix2.slice(n, rows, 0, n);

    auto b12 = matrix2.slice(0, n, n, rows);
    auto b22 = matrix2.slice(n, rows, n, rows);

    auto p1 = mult_strassen(a11 + a22, b11 + b22);
    auto p2 = mult_strassen(a22, b21 - b11);
    auto p3 = mult_strassen(a11 + a12, b22);
    auto p4 = mult_strassen(a12 - a22, b21 + b22);
    auto p5 = mult_strassen(a11, b12 - b22);
    auto p6 = mult_strassen(a21 + a22, b11);
    auto p7 = mult_strassen(a11 - a21, b11 + b12);

    auto c11 = p1 + p2 - p3 + p4;
    auto c12 = p5 + p3;
    auto c21 = p6 + p2;
    auto c22 = p5 + p1 - p6 - p7;


    return matrix_merge(c11, c12, c21, c22);;
}
```

3.4 Функциональные тесты

В данном разделе будут представлены функциональные тесты, проверяющие работу алгоритмов умножения матриц.

В таблице 3.1 приведены тесты для функции, реализующей алгоритм стандартного умножения матриц.

В таблице 3.2 приведены тесты для функции, реализующей алгоритм умножения матриц методом Винограда.

В таблице 3.3 приведены тесты для функции, реализующей алгоритм умножения матриц методом Штрассена.

Таблица 3.1 – Функциональные тесты для стандартного алгоритма умножения матриц

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 3 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{pmatrix}$

Таблица 3.2 – Функциональные тесты для алгоритма умножения матриц методом Винограда

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	*Неверный размер*

Таблица 3.3 – Функциональные тесты для алгоритма умножения матриц методом Штрассена

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 5 & 6 & 7 \\ 4 & 9 & 8 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	*Неверный размер*
$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$
$\begin{pmatrix} 5 & 7 \\ 4 & 8 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 5 & 7 \\ 4 & 8 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 14 & 16 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 14 & 16 \end{pmatrix}$

Вывод

Были реализованы классический алгоритм умножения матриц, алгоритм Винограда и его оптимизированная версия, алгоритм Штрассена. Проведено тестирование реализованных алгортимов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

- Процессор: Intel Core i7 9750H 2.6 ГГц;
- Оперативная память: 16 ГБ;
- Операционная система: Kubuntu 22.04.3 LTS x86_64 Kernel: 6.2.0-36-generic

Во время проведения измерений времени ноутбук был подключен к сети электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 показан пример работы с программой.

```
po4ti@Po4ti-PC:~/Документы/GitHub/Lab_02_AA$ ./app.exe
1 - Классический алгоритм умножения
2 - Алгоритм Винограда
3 - Оптимизированный алгоритм Винограда
4 - Алгоритм Штрассена
0 - Выход
1
Результат:
   5    6    7    1
   4    9    8    1
   3    2    1    1
   1    1    1    1
1 - Классический алгоритм умножения
2 - Алгоритм Винограда
3 - Оптимизированный алгоритм Винограда
4 - Алгоритм Штрассена
0 - Выход
2
Результат:
   5    6    7    1
   4    9    8    1
   3    2    1    1
   1    1    1    1
1 - Классический алгоритм умножения
2 - Алгоритм Винограда
3 - Оптимизированный алгоритм Винограда
4 - Алгоритм Штрассена
0 - Выход
```

Рисунок 4.1 – Демонстрация работы программы.

4.3 Временные характеристики

Исследование временных характеристик реализованных алгоритмов производилось на квадратных матрицах размерами:

- 1 – 128 с шагом 1 без алгоритма штрассена;
- 1 – 128 с увеличением шага в 2 раза для всех алгоритмов

На рисунке 4.2 показаны зависимости времени выполнения всех алгоритмов, кроме алгоритма Штрассена от размера матриц.

На рисунке 4.3 показаны зависимости времени выполнения всех алгоритмов от размера матриц.

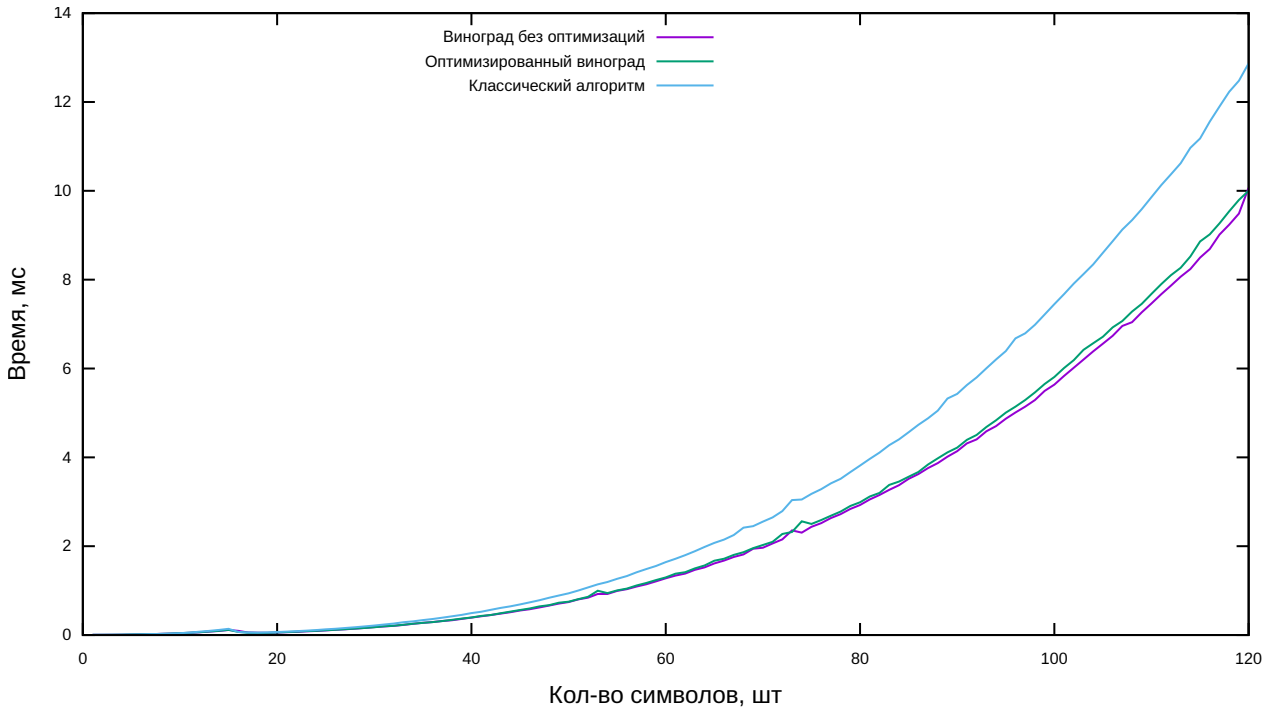

Рисунок 4.2 – Результат измерений времени работы реализуемых алгоритмов на матрицах с шагом размера 1

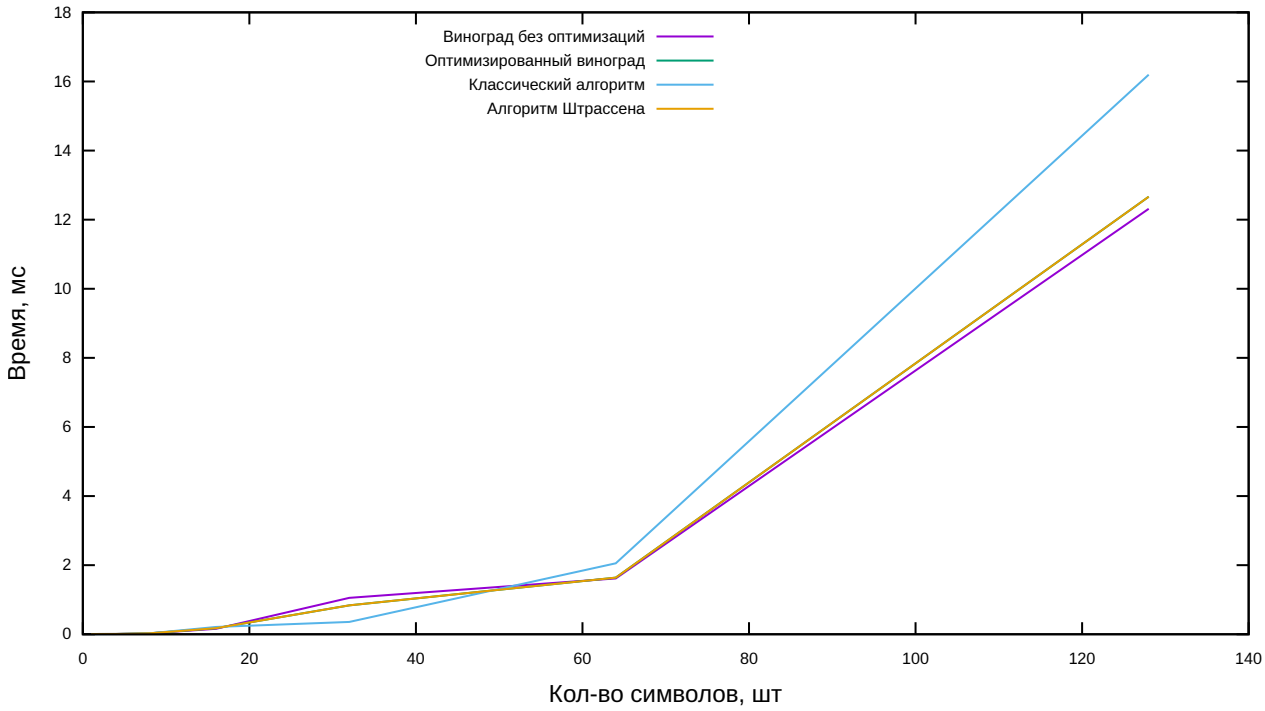

Рисунок 4.3 – Результат измерений времени работы реализуемых алгоритмов на матрицах, размеры которых — степень 2

4.4 Характеристики по памяти

Введем следующие обозначения:

- $size()$ — функция вычисляющая размер в байтах;
- int — целочисленный тип

Приведем теоретический расчет затрат по памяти для умножения двух матриц A и B размером $n \times n$, элементы которых типа int.

Стандартный алгоритм

Затраты по памяти для реализации стандартного алгоритма приведены в формуле 4.1:

$$\begin{aligned} M_{std} &= 3 \cdot size(int) + 3 \cdot size(int) + n \cdot n \cdot size(int) = \\ &= (n^2 + 6) \cdot size(int), \end{aligned} \tag{4.1}$$

где, $3 \cdot size(int)$ — переменные, хранящие размеры матриц;
$3 \cdot size(int)$ — переменные, используемые в циклах;
$n \cdot n \cdot size(int)$ — результирующая матрица.

Алгоритм Винограда

Затраты по памяти для реализации алгоритма умножения методом Винограда приведены в формуле 4.2:

$$M_{vin} = M_{mulH} + M_{mulV} + M_{mul}, \tag{4.2}$$

где M_{mulH}, M_{mulV} — память, используемая для хранения дополнительных массивов;
M_{mul} — память, используемая при самом умножении матриц;

Соответствующие затраты представлены в формулах 4.3–4.4

$$M_{mulH} = M_{mulV} = n \cdot size(int) + 2 * size(int); \tag{4.3}$$

$$M_{mul} = n \cdot n \cdot size(int) + 3 \cdot size(n) + \begin{cases} 0, & \text{четная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \tag{4.4}$$

Итоговые затраты по памяти для реализации алгоритма Винограда:

$$M_{mul} = (n^2 + 2 \cdot n + 7) \cdot size(int) + \begin{cases} 0, & \text{четная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \tag{4.5}$$

Оптимизированнный алгоритм Винограда

Затраты по памяти для оптимизированной реализации алгоритма умножения методом Винограда идентичны формуле 4.2. M_{mulH} и M_{mulV} для данной релизации также совпадают.

Отличия появляются в самом умножении матриц, поскольку в целях оптимизации необходимо было некоторые значения хранить в отдельных пе-

ременных.

$$M_{mul} = n \cdot n \cdot size(int) + 4 \cdot size(n) +$$
$$+n \cdot n \cdot \frac{n}{2} * size(int) + n \cdot n * size(int) \begin{cases} 0, & \text{четная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \quad (4.6)$$

Итоговые затраты по памяти для оптимизированной реализации алгоритма Винограда:

$$M_{mul} = (\frac{n^3}{2} + 2 \cdot n^2 + 6 \cdot n + 4) \cdot size(int) + \begin{cases} 0, & \text{четная} \\ 2 \cdot size(int), & \text{иначе.} \end{cases} \quad (4.7)$$

Алгоритм Штрассена

Рассчитаем затраты по памяти для каждого рекурсивного вызова.

$$M_{mul} = (4 \cdot \frac{n}{2} \cdot \frac{n}{2} + 4 \cdot \frac{n}{2} \cdot \frac{n}{2} +$$
$$+21 \cdot \frac{n}{2} \cdot \frac{n}{2} + n \cdot n) \cdot size(int), \quad (4.8)$$

где $4 \cdot \frac{n}{2} \cdot \frac{n}{2}$ — 4 матрицы, на которые мы разбиваем нашу исходную матрицу;
$13 \cdot \frac{n}{2} \cdot \frac{n}{2}$ — временные матрицы, получаемые в ходе вычислений;
$n \cdot n$ — результирующая матрица.

Итоговые затраты по памяти для реализации алгоритма Штрассена:

$$M_{mul} = \frac{33}{4} \cdot n^2 \cdot size(int). \quad (4.9)$$

Вывод

Исходя из данных, полученных с помощью графиков 4.2 –4.3 , можно сделать вывод, что лучше всего работает отмизированный алгоритм Винограда. Стандартный же алгоритм умножения матриц работает медленнее по

сравнению с двумя реалазиями алгоритма Винограда. Модификации, используемые в оптимизированной реализации алгоритма также повлияли на его скорость работы. Также алгоритм Винограда при четном размере матрицы работает быстрее — это обусловлено проведением дополнительных вычислений для крайних строк и столбцов при нечетном размере. Таким образом, алгоритм Винограда стоит использовать при работе именно с матрицами, размер которых — четный.

Также на рис. 4.3 видно, что реализация алгоритма Штрассена выполняется намного дольше реализации алгоритма Винограда, так как он требует дополнительных операций сложения и вычитаний матриц.

Из теоретических расчетов для потребляемой памяти, можно сделать вывод о том, что стандартное умножение требует наименьшее количество памяти. Наибольшие же затраты требует реализация алгоритма Штрассена, так как при каждом рекурсивном вызове необходимо разбивать исходные матрицы на 4 подматрицы, а также производить дополнительные операции сложения и умножения. Оптимизированная реализация алгоритма также требует больше памяти, по сравнению с неоптимизированной, так как используются дополнительные переменные для хранения некоторых предварительных вычислений.

Заключение

В ходе исследования было определено, что оптимизированный алгоритм Винограда продемонстрировал лучшую производительность. Классический метод умножения матриц работает медленнее по сравнению с остальными рассмотренными алгоритмами. Кроме того, реализация алгоритма Штрассена требует больше времени на выполнение из-за дополнительных операций сложения и вычитания матриц. Среди всех рассмотренных алгоритмов, наименьшую сложность имеет алгоритм штрассена - $O(N^{2.807})$, в то время как остальные - $O(N^3)$.

С учетом расчетов потребляемой памяти можно сделать вывод о том, что стандартное умножение требует наименьшее количество памяти, а алгоритм Штрассена требует наибольших затрат памяти из-за создания дополнительных матриц при каждом рекурсивном вызове, а также дополнительных операций сложения и умножения. Оптимизированная реализация алгоритма Винограда также требует больше памяти по сравнению с неоптимизированной из-за использования дополнительных переменных для хранения некоторых предварительных вычислений.

Цель данной лабораторной работы была достигнута – был проведен анализ алгоритмов умножения матриц.

В результате выполнения лабораторной работы для достижения этой цели были выполнены следующие задачи:

1. Изучены следующие алгоритмы:
    - классический алгоритм умножения;
    - алгоритм Винограда;
    - алгоритм Штрассена;
    - оптимизированный алогритм Винограда
2. Реализованы алгоритмы умножения матриц;
3. Выполнены замеры затрат реализаций алгоритмов по памяти;
4. Выполнены замеры затрат реализаций алгоритмов по процессорному времени;

5. Проведен сравнительных анализ алгоритмов

Список использованных источников

1 С. Анисимов Н., В. Строганов Ю. Реализация умножения матриц по Винограду на языке Haskell. // Новые информационные технологии в автоматизированных системах. 2018.

2 Головашкин Д. Л. Векторные алгоритмы вычислительной линейной алгебры: учеб. пособие. — Самара: Изд-во Самарского университета, 2019. — С. 28–35.

3 Документация по Microsoft C++ [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170&viewFallbackFrom=vs-2017` (дата обращения: 25.09.2023).

4 Standard library header <ctime> [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/header/ctime`.