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 Пискунов П.

Группа ИУ7-56Б

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

Москва — 2023 г.

Содержание

Введение

В мире математики и информатики матрицы играют фундаментальную роль. Они представляют собой мощный инструмент для организации, хранения и обработки данных. Матрицы используются в различных областях, включая линейную алгебру, статистику, компьютерную графику, машинное обучение и многое другое. Они позволяют представить набор чисел в виде двумерной структуры, облегчая выполнение разнообразных математических операций.

Целью данной лабораторной работы является исследование алгоритмов умножения матриц. Для успешного выполнения лабораторной работы необходимо выполнить следующие задачи:

1) изучить и реализовать 4 алгоритма умножения матриц: стандартный алгоритм умножения матриц, алгоритм Винограда, оптимизированый алгоритм Винограда и алгоритм Штрассена;

2) создать программный продукт, позволяющий протестировать реализованные алгоритмы;

3) выбрать инструменты для замера процессорного времени выполнения реализаций алгоритмов;

4) провести анализ затрат работы алгоритмов по времени и по памяти;

5) подготовить отчет по лабораторной работе.

1 Аналитическая часть

Матрицей[1] размера $m \times n$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы. Числа, функции или алгебраические выражения, образующие матрицу, называются матричными элементами.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \tag{1.1}$$

Основные операции на матрицами это:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц в случае, если количество столбцов первой матрицы равно количеству строк второй матрицы.

Замечание: операция умножения является некоммутативной. Это значит, что, оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, но результаты будут отличаться друг от друга.

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы размерности $n \times m$ и $m \times p$

$$A_{nm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B_{mp} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{pmatrix}, \tag{1.2}$$

тогда матрица C

$$C_{np} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{np} \end{pmatrix}, \tag{1.3}$$

где

$$C_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.4}$$

будет называться произведением матриц A и B.

1.2 Алгоритм Винограда

Алгоритм Винограда, предложенный Ш. Виноградом в 1987 году, представляет собой метод умножения квадратных матриц. Изначально алгоритм имел асимптотическую сложность $O(n^{2,3755})$, где n - это размер стороны матрицы. Благодаря последующим усовершенствованиям, алгоритм Винограда теперь имеет лучшую асимптотику среди известных методов умножения матриц. Это делает его более эффективным и практичным.[2].

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:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \tag{1.5}$$

Второе выражение, несмотря на дополнительные операции (шесть умножений и десять сложений против четырех и трех), предоставляет преимущество в возможности предварительной обработки. Элементы этого выражения могут быть вычислены заранее и сохранены для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволяет выполнять всего лишь два умножения и пять сложений для каждого элемента, при этом используя две заранее вычисленные суммы соседних элементов текущих строк и столбцов. Учитывая, что сложение более эффективно, чем умножение на современных вычислительных системах, этот алгоритм должен работать быст-

рее стандартного.

При нечетном размере матрицы n требуется дополнительная операция: добавление произведения последних элементов строк и столбцов.

1.3 Алгоритм Штрассена

Идея алгоритма Штрассена: произведение матриц $2\times2(C = A \cdot B)$ вычисляется с использованием всего лишь семи умножений, вместо восьми, как в стандартном алгоритме[3]. Действие осуществляется с использованием следующих формул:

$$\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \cdot \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix} \tag{1.6}$$

$$m_1 = (a_{00} + a_{11})(b_{00} + b_{11}) \tag{1.7}$$

$$m_2 = (a_{10} + a_{11}) \cdot b_{00} \tag{1.8}$$

$$m_3 = (b_{01} - b_{11}) \cdot a_{00} \tag{1.9}$$

$$m_4 = a_{11} \cdot (b_{10} - b_{00}) \tag{1.10}$$

$$m_5 = (a_{00} + a_{01}) \cdot b_{11} \tag{1.11}$$

$$m_6 = (a_{10} - a_{00}) \cdot (b_{00} + b_{01}) \tag{1.12}$$

$$m_7 = (a_{01} - a_{11}) \cdot (b_{10} + b_{11}) \tag{1.13}$$

Метод Штрассена работает с квадратными матрицами, порядок которых можно представить в виде числа, равному степени двойки. В случае, когда это не так, матрица дополняется нулевыми элементами до квадратной матрицы ближайшей корректного размера.

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц, а именно классический, алгоритм Штрассена и алгоритм Винограда. Также мы рассмотрели различные оптимизации, которые можно применить при программной реализации алгоритма Винограда.

2 Конструкторская часть

2.1 Разработка стандартного алгоритма умножения матриц

На рисунке 2.1 приведена схема рассматриваемого алгоритма.

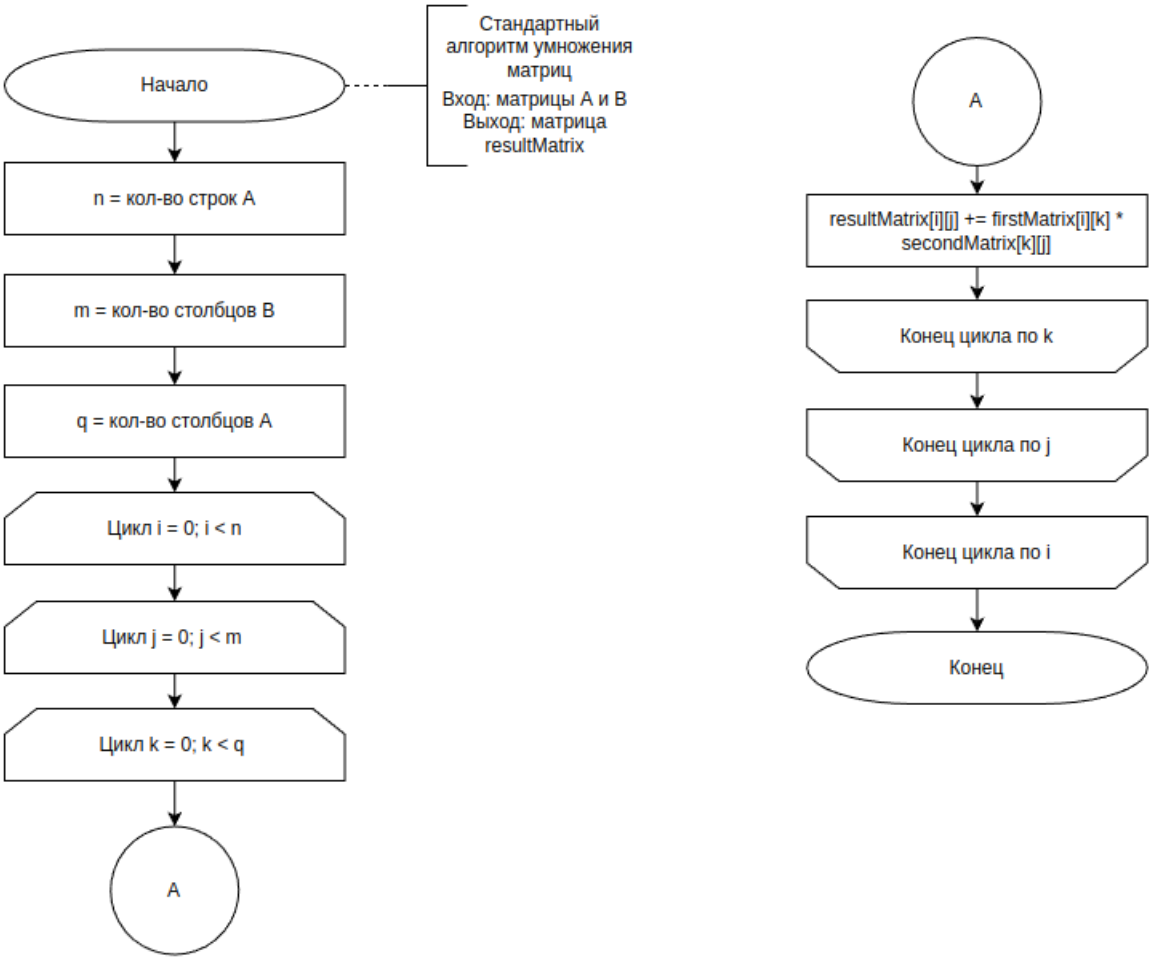

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

2.2 Разработка алгоритма Винограда умножения матриц

На рисунках 2.2 – 2.3 приведена схема рассматриваемого алгоритма.

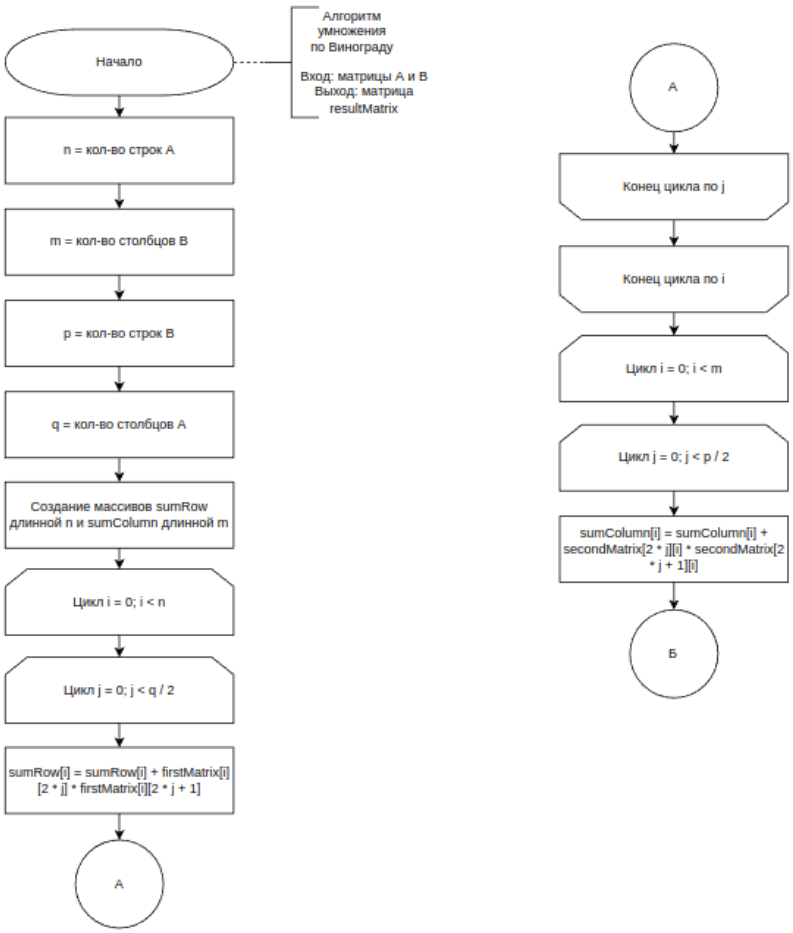

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц(Часть 1)

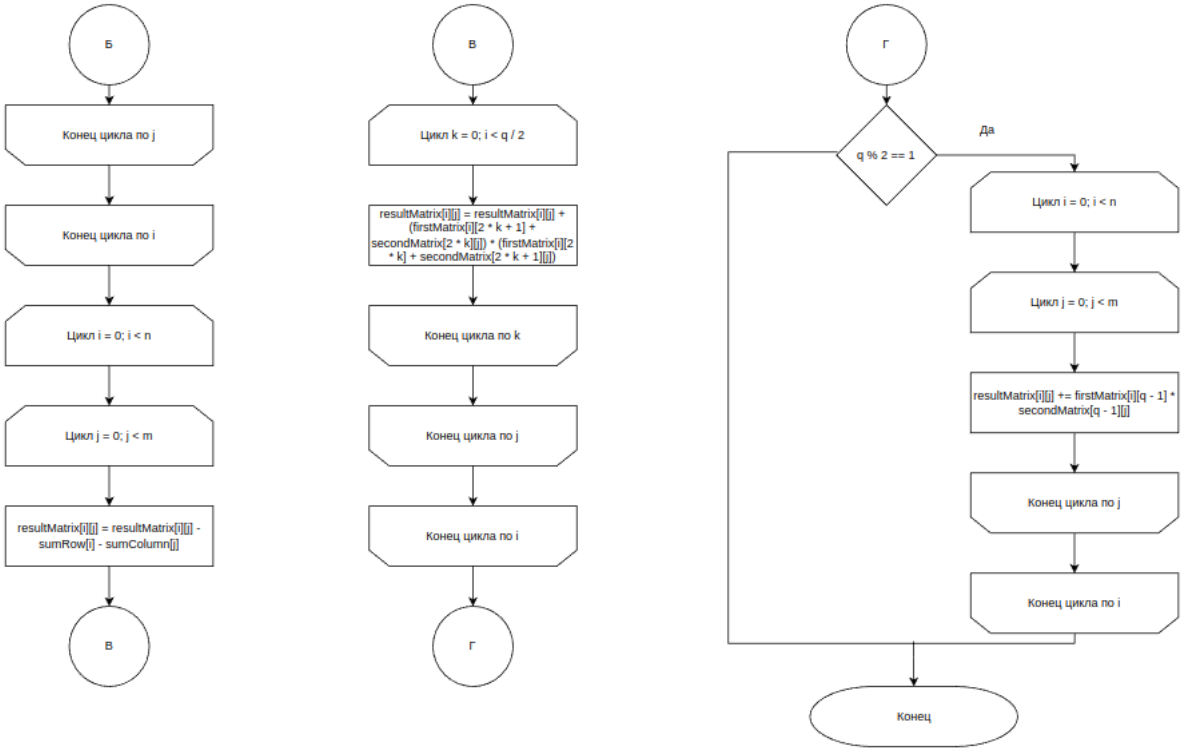

Рисунок 2.3 – Схема алгоритма Винограда умножения матриц(Часть 2)

2.3 Разработка оптимизированного алгоритма Винограда умножения матриц

На рисунках 2.4 – 2.5 приведена схема рассматриваемого алгоритма.

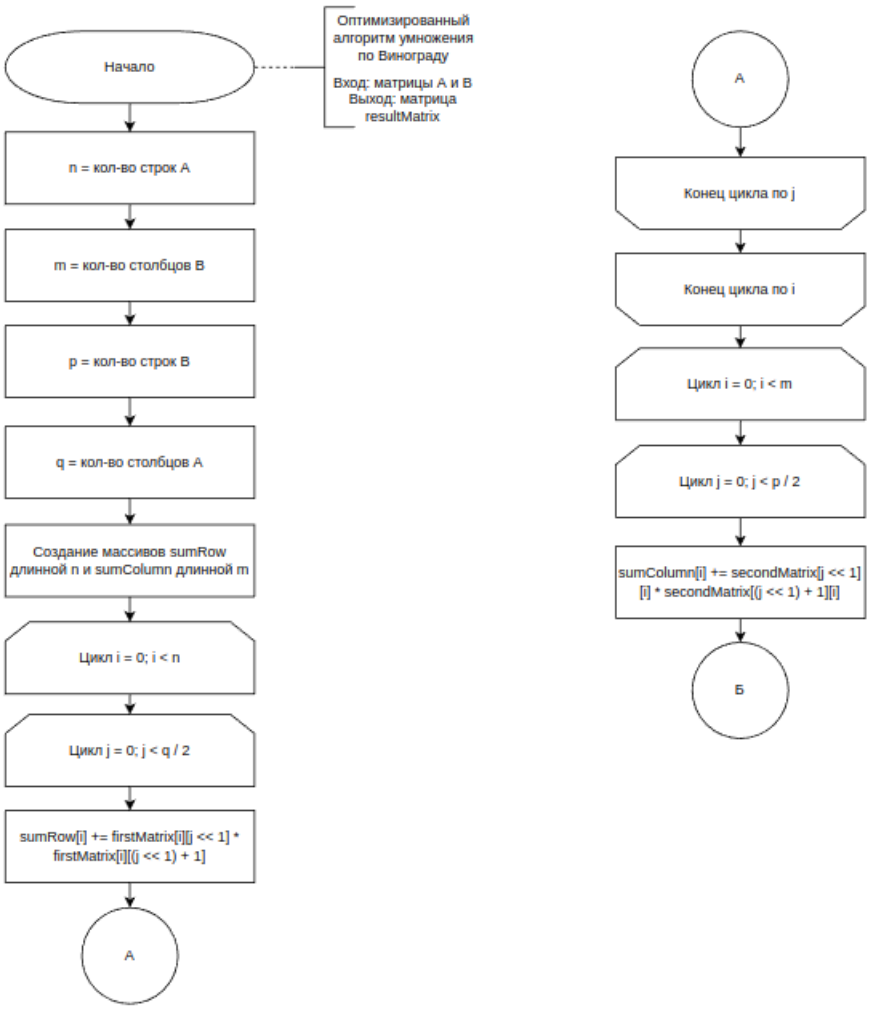

Рисунок 2.4 – Схема оптимизированного алгоритма Винограда умножения матриц(Часть 1)

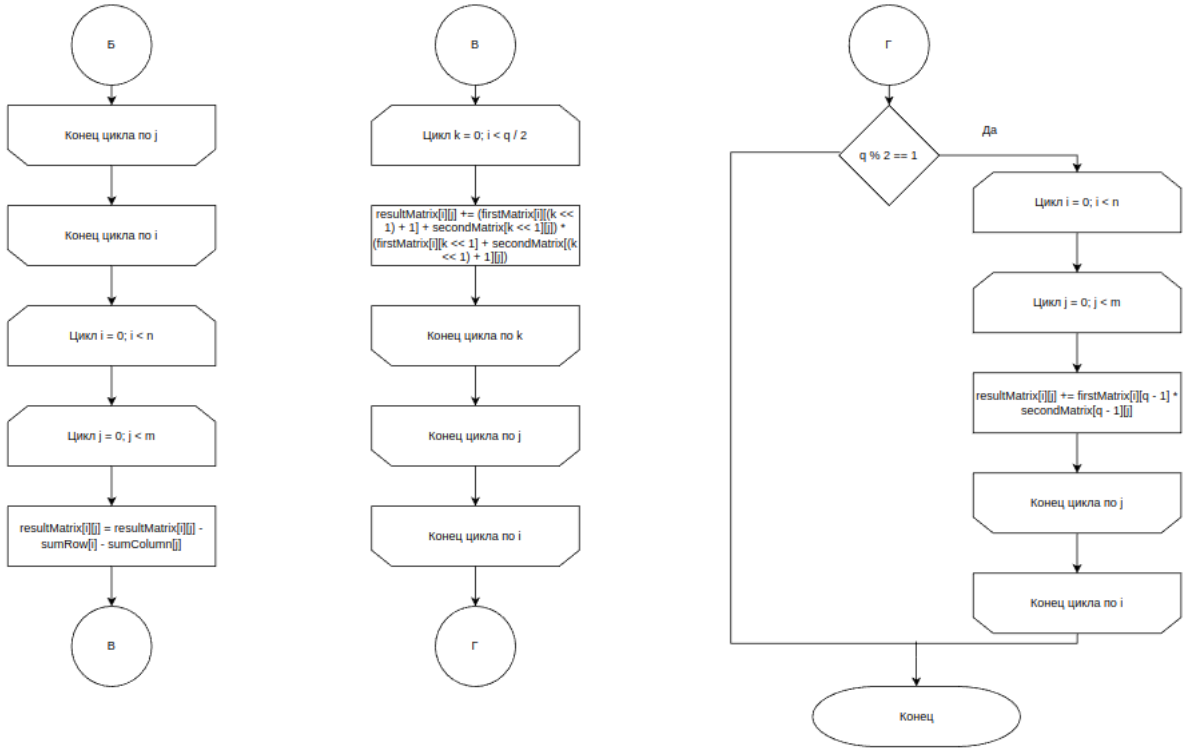

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда умножения матриц(Часть 2)

2.4 Разработка алгоритма Штрассена умножения матриц

На рисунках 2.6 – 2.7 приведена схема рассматриваемого алгоритма.

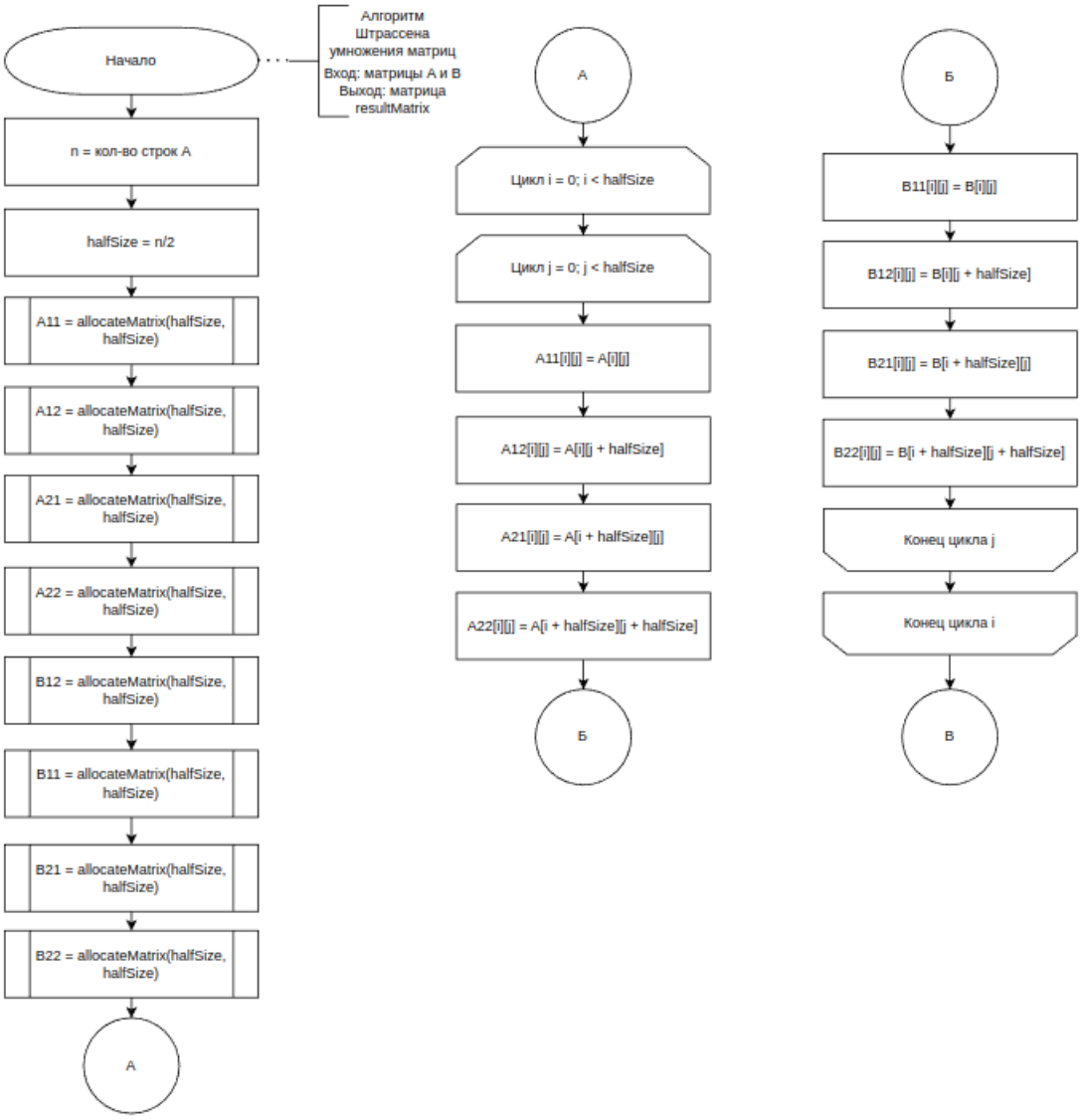

Рисунок 2.6 – Схема алгоритма Штрассена умножения матриц(Часть 1)

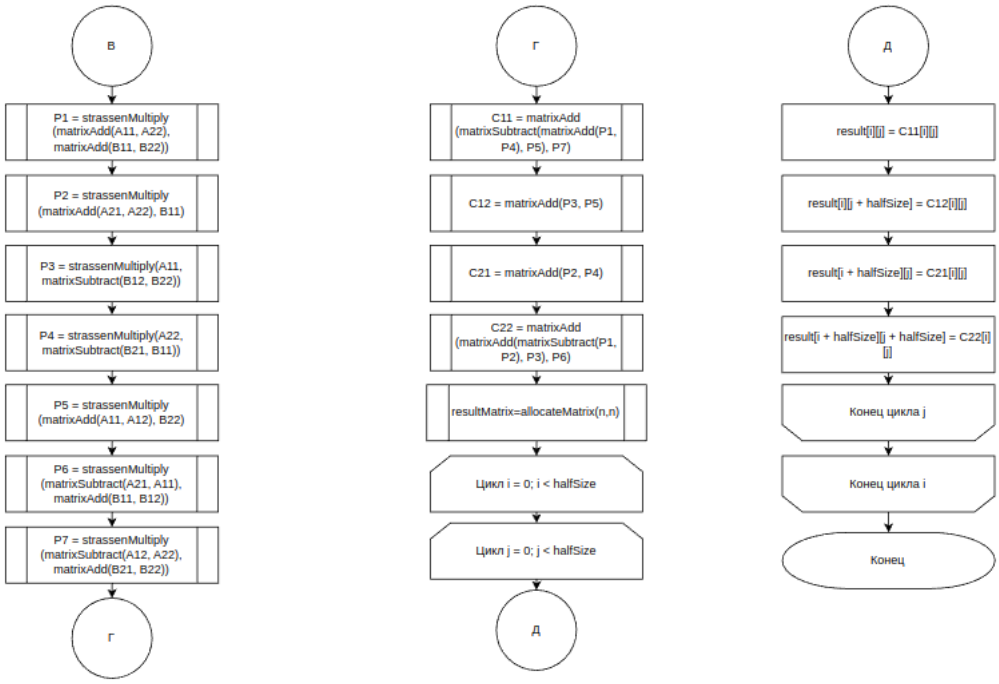

Рисунок 2.7 – Схема алгоритма Штрассена умножения матриц(Часть 2)

2.5 Модель вычислений

Для дальнейшего анализа трудоемкости необходимо ввести модель вычислений.

Операции из списка (2.1) имеют трудоемкость 1.

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

Операции из списка (2.2) имеют трудоемкость 2.

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

Трудоемкость оператора выбора `if условие then A else B` рассчитывается как:

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

Трудоемкость цикла рассчитывается как:

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

Трудоемкость вызова функции равна 0.

2.6 Трудоемкость алгоритмов

В каждом из представленных алгоритмов создается дополнительная матрица, в которую затем записывается результат умножения. Необходимо отметить, что в рассмотрении трудоемкости не уделяется особого внимания этапу инициализации этой дополнительной матрицы. Это действие не является вычислительно сложным и не влияет значительно на общую трудоемкость алгоритмов.

2.6.1 Стандартный алгоритм умножения матриц

Трудоемкость классического алгоритма умножения матриц включает в себя следующие составляющие:

Трудоёмкость внешнего цикла по $i \in [0..M-1]$: $f = 2 + M \cdot (2 + f_{body})$.

Трудоёмкость цикла по $j \in [0..N-1]$: $f = 2 + N \cdot (2 + f_{body})$.

Трудоёмкость цикла по $k \in [0..K-1]$: $f = 2 + 11K$.

Учитывая, что сложность стандартного алгоритма равна сложности внешнего цикла, можно вычислить её, анализируя тело цикла:

$$f_{standard} = 2 + M \cdot (4 + N \cdot (4 + 11K)) \approx 11MNK \tag{2.5}$$

2.6.2 Алгоритм Винограда умножения матриц

Трудоёмкость алгоритма Винограда состоит из следующих составляющих.

Трудоёмкость создания и инициализации массивов MH и MV:

$$f_{init} = M + N \tag{2.6}$$

Трудоёмкость заполнения массива MH:

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 11) \tag{2.7}$$

Трудоёмкость заполнения массива MV:

$$f_{MV} = 2 + K(2 + \frac{N}{2} \cdot 11) \tag{2.8}$$

Трудоёмкость цикла заполнения для чётных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)) \tag{2.9}$$

Трудоёмкость цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный:

$$f_{last} = \begin{cases} 2, & \text{размер чётный,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \tag{2.10}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11{,}5 \cdot MNK \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11{,}5 \cdot MNK \tag{2.12}$$

2.6.3 Оптимизированный алгоритм Винограда умножения матриц

Трудоёмкость улучшенного алгоритма Винограда состоит из следующих составляющих.

Трудоёмкость создания и инициализации массивов MH и MV:

$$f_{init} = M + N \tag{2.13}$$

Трудоёмкость заполнения массива MH:

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 8) \tag{2.14}$$

Трудоёмкость заполнения массива MV:

$$f_{MV} = 2 + K(2 + \frac{M}{2} \cdot 8) \tag{2.15}$$

Трудоёмкость цикла заполнения для чётных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)) \tag{2.16}$$

Трудоёмкость условия для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный:

$$f_{last} = \begin{cases} 1, & \text{размер чётный,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \tag{2.17}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.18}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.19}$$

2.6.4 Алгоритм Штрассена

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из:

$$f_{best}(n) = 7 \cdot T(n/2) + O(n^2) = O(n^{\log_2 7}) \tag{2.20}$$

Вывод

На основе теоретических знаний, полученных в аналитическом разделе, были разработаны схемы алгоритмов. Благодаря им может быть вычислено произведение двух матриц разними способами.

3 Технологическая часть

В этом разделе осуществляется выбор способов реализации, и указываются требования к программному обеспечению (ПО). Также предоставляются листинги выполненных алгоритмов.

3.1 Требования к программному обеспечению

Программе передаются две целочисленые матрицы в качестве входных данных, а на выход получается матрица resultMatrix. resultMatrix – результат умножения введенных пользователем матриц. Кроме того, необходимо сообщить затраченное каждым алгоритмом процессорное время и память.

В создаваемом приложении пользователю должны быть доступны функции ввода двух матриц и выбора желаемого алгоритма. Пользователь должен иметь возможность оценить размер и работу каждого алгоритма.

3.2 Выбор средств реализации

Для выполнения данной лабораторной работы был выбран язык программирования C++. Программное обеспечение, разработанное на C++, позволяет наглядно демонстрировать производительность алгоритмов и упрощает процесс тестирования. Время измерялось с помощью функции *clock()* из библиотеки *time.h* [4]

3.3 Реализация алгоритмов

В листингах 3.1 – 3.4 представлены реализации рассматриваемых алгоритмов.

В Листинге 3.1 показана реализация стандартного алгоритма умножения матриц.

```
1  vector<vector<int>> mulMatrices(const vector<vector<int>> &firstMatrix,
    const vector<vector<int>> &secondMatrix)
2  {
3    size_t n = firstMatrix.size();
4    size_t m = secondMatrix[0].size();
5    size_t q = firstMatrix[0].size();
6    vector<vector<int>> resultMatrix(n, vector<int>(m, 0));
7
8    for (size_t i = 0; i < n; i++)
9      for (size_t j = 0; j < m; j++)
10       for (size_t k = 0; k < q; k++)
11         resultMatrix[i][j] += firstMatrix[i][k] * secondMatrix[k][j];
12
13   return resultMatrix;
14 }
```

Листинг 3.1 – Стандартный алгоритм умножения матриц

В Листинге 3.2 показана реализация алгоритма Винограда умножения матриц.

```
vector<vector<int>> mulVinogradAlg(const vector<vector<int>> &firstMatrix,
  const vector<vector<int>> &secondMatrix)
{
  size_t n = firstMatrix.size();
  size_t m = secondMatrix[0].size();
  size_t p = secondMatrix.size();
  size_t q = firstMatrix[0].size();
  vector<vector<int>> resultMatrix(n, vector<int>(m, 0));
  vector<int> sumRow(n), sumColumn(m);

  for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < q / 2; j++)
      sumRow[i] = sumRow[i] + firstMatrix[i][2 * j] * firstMatrix[i][2 * j
  + 1];

  for (size_t i = 0; i < m; i++)
    for (size_t j = 0; j < p / 2; j++)
      sumColumn[i] = sumColumn[i] + secondMatrix[2 * j][i] * secondMatrix
  [2 * j + 1][i];

  for (size_t i = 0; i < n; i++)
    for (size_t j = 0; j < m; j++)
    {
      resultMatrix[i][j] = resultMatrix[i][j] - sumRow[i] - sumColumn[j];
      for (size_t k = 0; k < q / 2; k++)
        resultMatrix[i][j] = resultMatrix[i][j] + (firstMatrix[i][2 * k +
  1] + secondMatrix[2 * k][j]) * (firstMatrix[i][2 * k] + secondMatrix[2 *
  k + 1][j]);
    }

  if (q % 2 == 1)
  {
    for (size_t i = 0; i < n; i++)
      for (size_t j = 0; j < m; j++)
        resultMatrix[i][j] += firstMatrix[i][q - 1] * secondMatrix[q - 1][
  j];
  }

  return resultMatrix;
}
```

Листинг 3.2 – Алгоритм Винограда умножения матриц

В Листинге 3.3 показана реализация оптимизированного алгоритма Винограда умножения матриц.

```
1  vector<vector<int>> optVinogradAlg(const vector<vector<int>> &firstMatrix,
     const vector<vector<int>> &secondMatrix)
2  {
3    size_t n = firstMatrix.size();
4    size_t m = secondMatrix[0].size();
5    size_t p = secondMatrix.size();
6    size_t q = firstMatrix[0].size();
7    vector<vector<int>> resultMatrix(n, vector<int>(m, 0));
8    vector<int> sumRow(n), sumColumn(m);
9
10   for (size_t i = 0; i < n; i++)
11     for (size_t j = 0; j < q / 2; j++)
12       sumRow[i] += firstMatrix[i][j << 1] * firstMatrix[i][(j << 1) + 1];
13
14   for (size_t i = 0; i < m; i++)
15     for (size_t j = 0; j < p / 2; j++)
16       sumColumn[i] += secondMatrix[j << 1][i] * secondMatrix[(j << 1) +
    1][i];
17
18   for (size_t i = 0; i < n; i++)
19     for (size_t j = 0; j < m; j++)
20     {
21       resultMatrix[i][j] = resultMatrix[i][j] - sumRow[i] - sumColumn[j];
22       for (size_t k = 0; k < q / 2; k++)
23         resultMatrix[i][j] += (firstMatrix[i][(k << 1) + 1] + secondMatrix
    [k << 1][j]) * (firstMatrix[i][k << 1]  + secondMatrix[(k << 1) + 1][j]);
24     }
25
26   if (q % 2 == 1)
27     for (size_t i = 0; i < n; i++)
28       for (size_t j = 0; j < m; j++)
29         resultMatrix[i][j] += firstMatrix[i][q - 1] * secondMatrix[q - 1][
    j];
30
31   return resultMatrix;
32 }
```

Листинг 3.3 – Оптимизированный алгоритм Винограда умножения матриц

В Листинге 3.4 показана реализация алгоритма Штрассена умножения матриц.

```
vector<vector<int>> strassenMultiply(const vector<vector<int>> &A, const
 vector<vector<int>>& B)
{
  size_t n = A.size();

  if (n == 1) {
    vector<vector<int>> result(1, vector<int>(1));
    result[0][0] = A[0][0] * B[0][0];
    return result;
  }

  size_t halfSize = n / 2;
  vector<vector<int>> A11(halfSize, vector<int>(halfSize));
  vector<vector<int>> A12(halfSize, vector<int>(halfSize));
  vector<vector<int>> A21(halfSize, vector<int>(halfSize));
  vector<vector<int>> A22(halfSize, vector<int>(halfSize));
  vector<vector<int>> B11(halfSize, vector<int>(halfSize));
  vector<vector<int>> B12(halfSize, vector<int>(halfSize));
  vector<vector<int>> B21(halfSize, vector<int>(halfSize));
  vector<vector<int>> B22(halfSize, vector<int>(halfSize));

  for (size_t i = 0; i < halfSize; i++) {
    for (size_t j = 0; j < halfSize; j++) {
      A11[i][j] = A[i][j];
      A12[i][j] = A[i][j + halfSize];
      A21[i][j] = A[i + halfSize][j];
      A22[i][j] = A[i + halfSize][j + halfSize];
      B11[i][j] = B[i][j];
      B12[i][j] = B[i][j + halfSize];
      B21[i][j] = B[i + halfSize][j];
      B22[i][j] = B[i + halfSize][j + halfSize];
    }
  }

  vector<vector<int>> P1 = strassenMultiply(matrixAdd(A11, A22), matrixAdd
(B11, B22));
  vector<vector<int>> P2 = strassenMultiply(matrixAdd(A21, A22), B11);
  vector<vector<int>> P3 = strassenMultiply(A11, matrixSubtract(B12, B22))
;
  vector<vector<int>> P4 = strassenMultiply(A22, matrixSubtract(B21, B11))
;
  vector<vector<int>> P5 = strassenMultiply(matrixAdd(A11, A12), B22);
  vector<vector<int>> P6 = strassenMultiply(matrixSubtract(A21, A11),
matrixAdd(B11, B12));
  vector<vector<int>> P7 = strassenMultiply(matrixSubtract(A12, A22),
matrixAdd(B21, B22));
```

```
41
42      vector<vector<int>> C11 = matrixAdd(matrixSubtract(matrixAdd(P1, P4), P5
    ), P7);
43      vector<vector<int>> C12 = matrixAdd(P3, P5);
44      vector<vector<int>> C21 = matrixAdd(P2, P4);
45      vector<vector<int>> C22 = matrixAdd(matrixAdd(matrixSubtract(P1, P2), P3
    ), P6);
46
47      vector<vector<int>> result(n, vector<int>(n));
48
49      for (size_t i = 0; i < halfSize; i++) {
50        for (size_t j = 0; j < halfSize; j++) {
51          result[i][j] = C11[i][j];
52          result[i][j + halfSize] = C12[i][j];
53          result[i + halfSize][j] = C21[i][j];
54          result[i + halfSize][j + halfSize] = C22[i][j];
55        }
56      }
57
58      return result;
59    }
```

Листинг 3.4 – Алгоритм Штрассена умножения матриц

Вывод

Были выбраны инструменты и разработаны алгоритмы умножения матриц: стандартный, алгоритм Винограда, Штрассена и оптимизированный Винограда. Также предоставлены листинги кода на выбранном языке программирования.

4 Исследовательская часть

4.1 Интерфейс приложения

На рисунках 4.1 – 4.2 приведено изображение интерфейса главного экрана приложения.

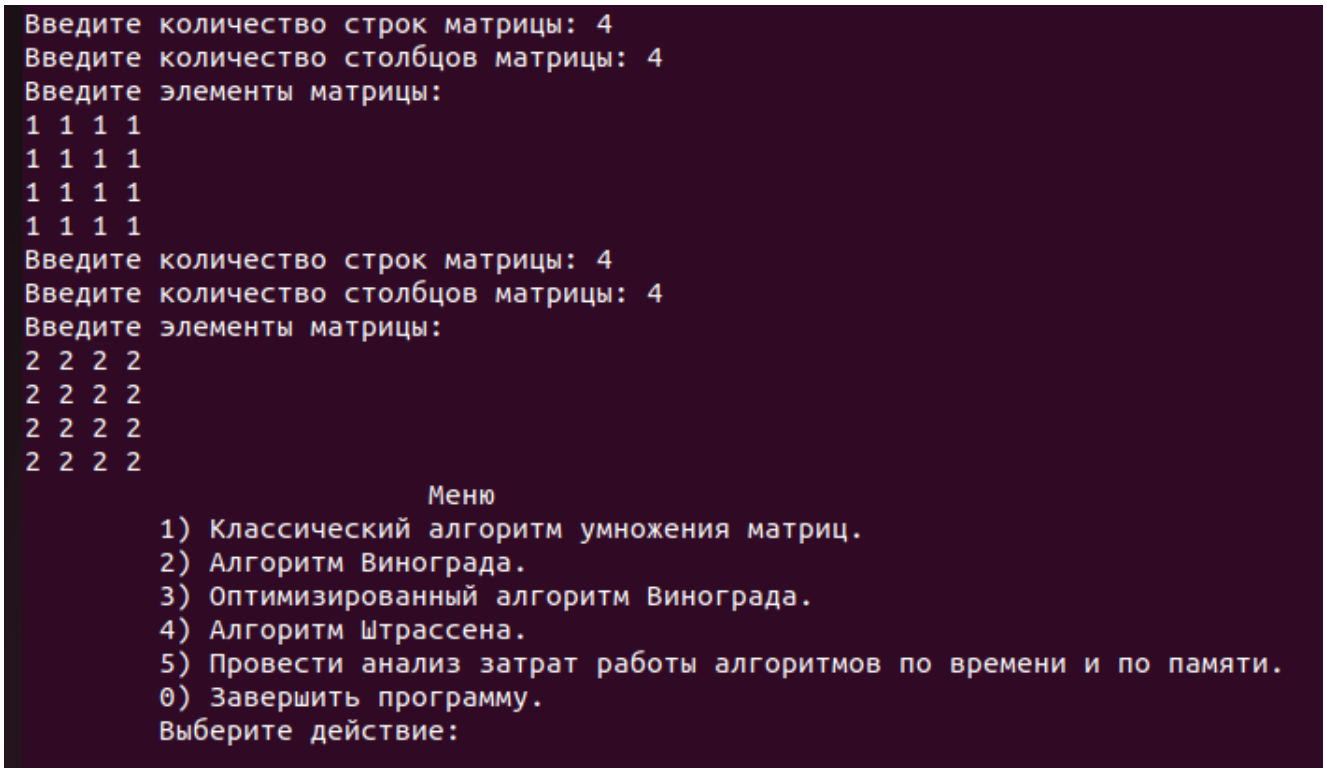

Рисунок 4.1 – Интерфейс

На экране приложения пользователь может ввести две матрицы, для которых необходимо вычислить умножение, а также выбрать метод, который будет использован для этого вычисления. В результате выводится полученное умножение.

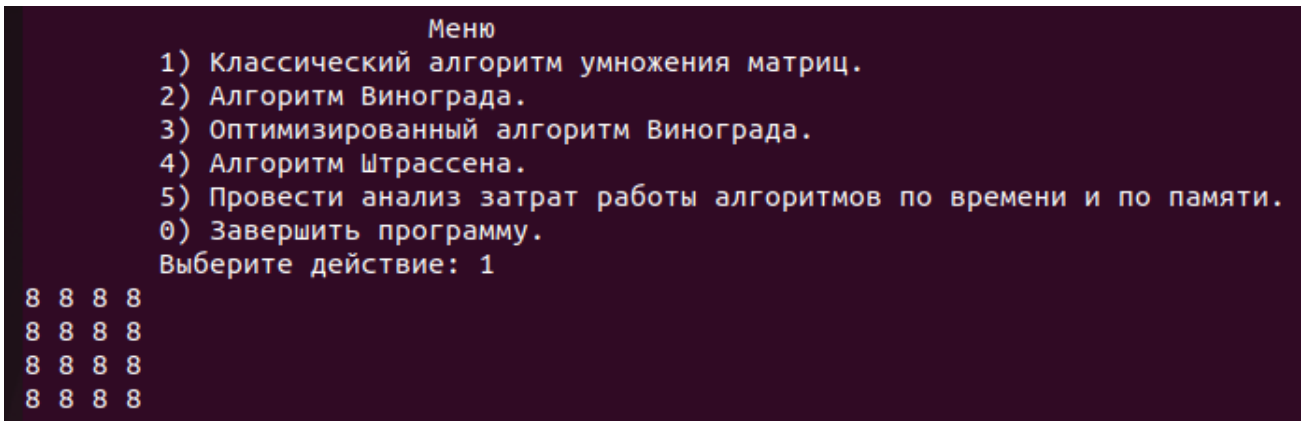

Рисунок 4.2 – Пример работы стандартного алгоритма умножения матриц

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Ubuntu 22.04.3 LTS;
- оперативная память: 12 Гб;
- процессор: AMD® Athlon silver 3050u with radeon graphics × 2;

4.3 Время выполнения реализаций алгоритмов

Замеры времени работы реализованных алгоритмов проводились для квадратных матриц. На рисунке 4.3 приведено сравнение реализации всех алгоритмов умножения матриц, которые содержат 2, 4, 8, 16, 32, 64, 128, 256 элементов.

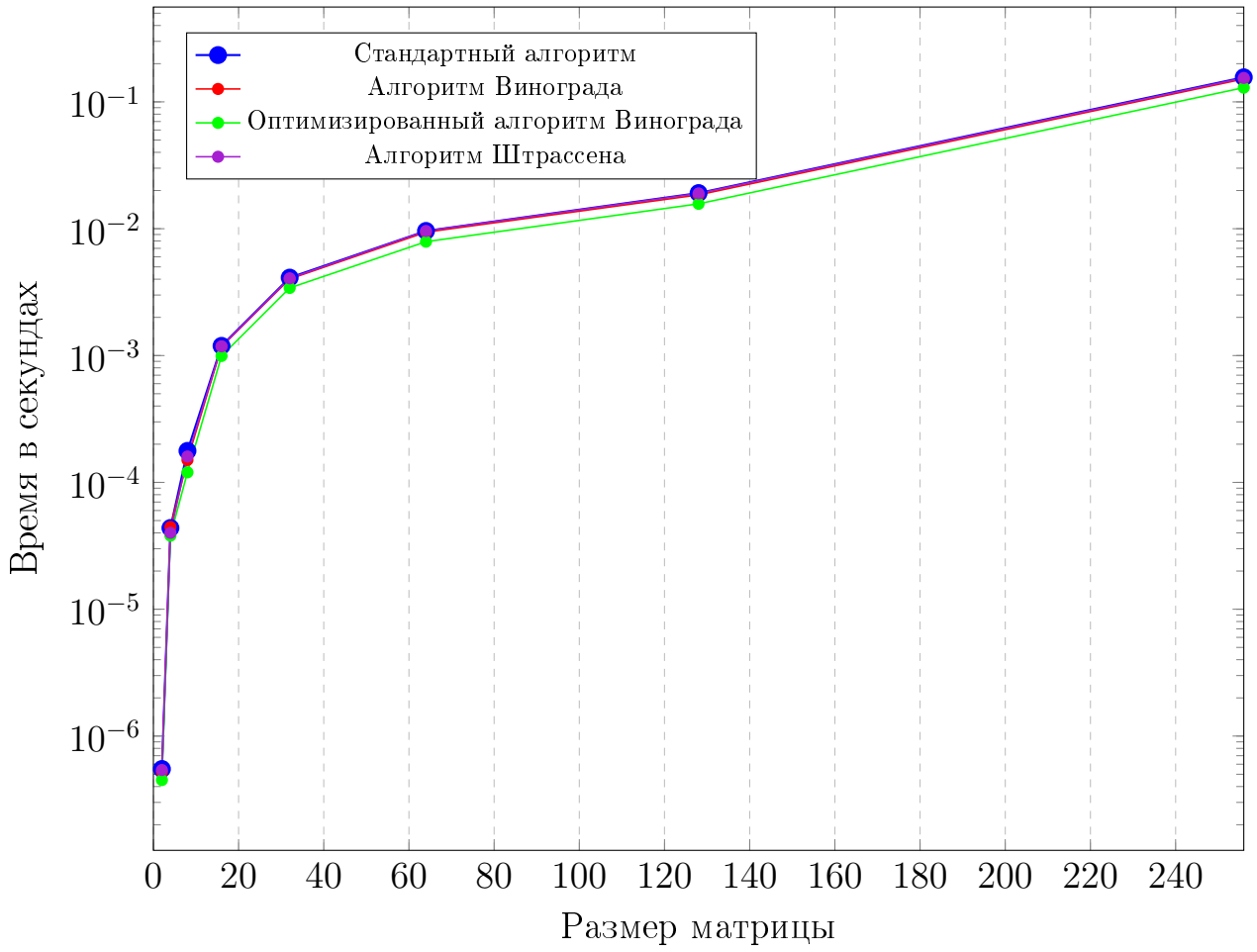

Рисунок 4.3 – Сравнение времени работы реализаций алгоритмов

4.4 Используемая память

Использование памяти при стандартным алгоритмом теоретически равно:

$$3 \cdot sizeof(size\_t) + n_1 \cdot m_1 \cdot sizeof(int) + sizeof(vector < vector < int >>) + \\ + n_2 \cdot m_2 \cdot sizeof(int) + sizeof(vector < vector < int >>) + n_1 \cdot m_2 \cdot \\ \cdot sizeof(int) + sizeof(vector < vector < int >>), \tag{4.1}$$

где:

- $3 \cdot sizeof(size\_t)$ — хранение количество строк и столбцов первой матрицы и количество столбцов второй матрицы;
- $n_1 \cdot m_1 \cdot sizeof(int)$ — хранение количество элементов матрицы;
- $sizeof(vector < vector < int >>$ — структура матрицы.

Для обычного и оптимизированного алгоритма Винограда умножения матриц можно посчитать по следуюшей формуле:

$$4 \cdot sizeof(size\_t) + n_1 \cdot m_1 \cdot sizeof(int) + \\ + sizeof(vector < vector < int >>) + n_2 \cdot m_2 \cdot sizeof(int) + \\ + sizeof(vector < vector < int >>) + n_1 \cdot m_2 \cdot sizeof(int) + \\ + sizeof(vector < vector < int >> + n_1 \cdot sizeof(int) + m_2 \cdot sizeof(int)), \tag{4.2}$$

где

- $4 \cdot sizeof(size\_t)$ – хранение количество строк и столбцов матриц;
- $n_1 \cdot m_1 \cdot sizeof(int)$ — хранение количество элементов матрицы;
- $n_1 \cdot sizeof(int)$ — храниние количество элементов одномерного массива
- $sizeof(vector < vector < int >>$ — структура матрицы.

По расходу памяти алгоритм Винограда и оптимизированный алгоритм Винограда проигрывают от стандартного алгоритма умножения матриц. Это из-за того, что в алгоритме Винограда и оптимизированном алгоритме Винограда хранятся дополнительные два одномерных массива.

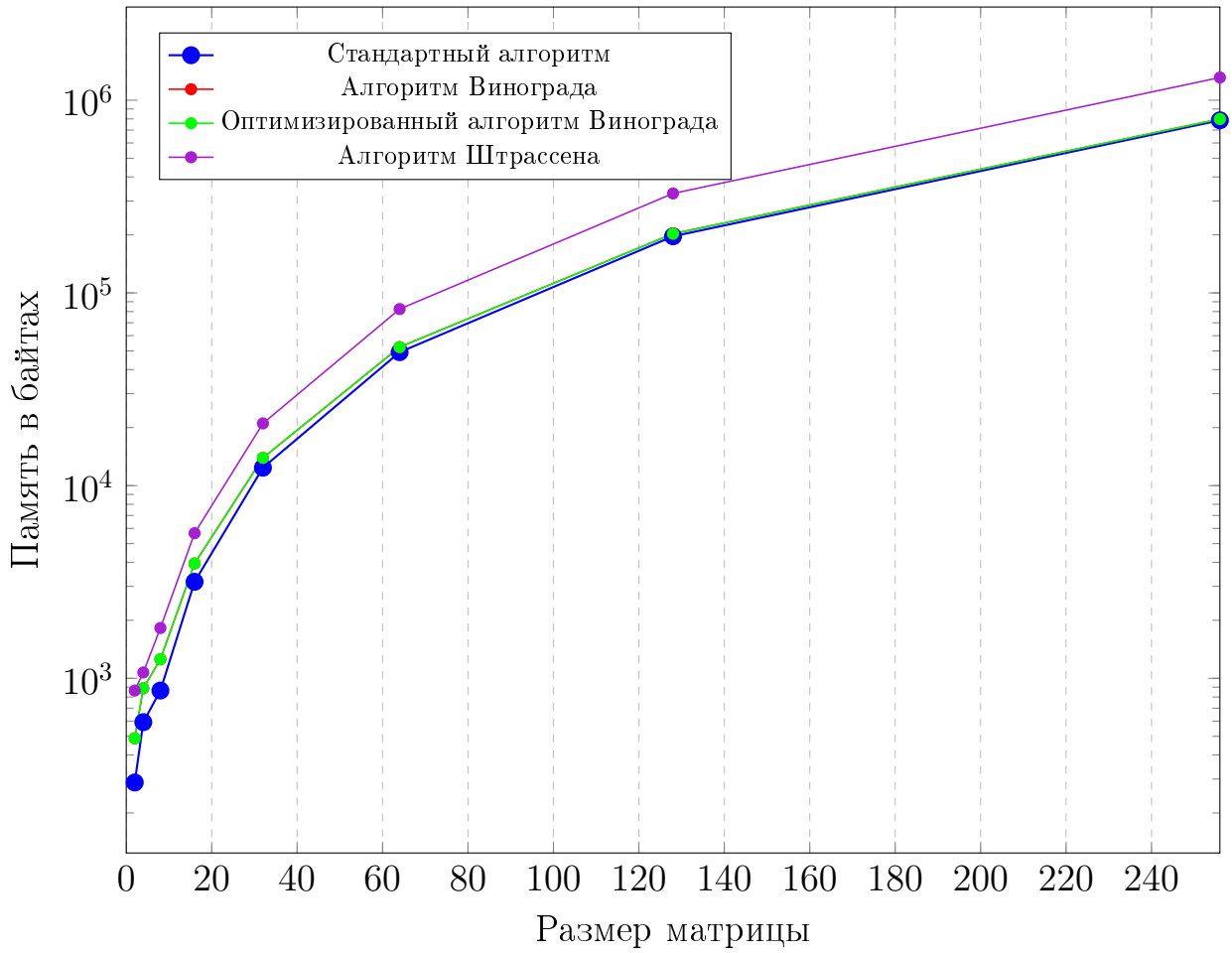

Рисунок 4.4 – Сравнение размеров реализаций алгоритмов в байтах

Вывод

В данном разделе было произведено сравнение количества затраченного времени и памяти алгоритмов умножения матриц. Наименее затратным по времени оказался оптимизированный алгоритм Винограда. Получисля такой результат потому, что у него сложность меньше $O(n^{2,3755})$, пока у стандартного алгоритм сложность $O(n^3)$, а у алгоритма Штрассена $O(n^{\log_2 7})$.

По памяти наименее затратным оказался стандартный алгоритм умножения матриц, так как он содержить наименьшее количество структуры.

Заключение

В результате исследования трудоемкости алгоритмов был получен следующий вывод. Стандартный алгоритм умножения матриц занимает больше времени по сравнению с алгоритмами Винограда и Штрассена. Причина заключается в том, что в алгоритмах Винограда и Штрассена часть вычислений выполняется заранее. Это позволяет снизить сложность операций умножения. Сложность у этих алгоритмах $O(n^{2,37})$ и $O(n^{\log_2 7})$, пока сложность у стандартного алгортма $O(n^3)$. Исходя из этого, алгоритм Винограда является более предпочтительным вариантом.

Однако лучшие результаты по времени показал оптимизированный алгоритм Винограда. Это достигается заменой операций и умножения на операции сдвига, что улучшает эффективность вычислений. Поэтому при выборе наиболее производительного алгоритма рекомендуется отдавать предпочтение оптимизированному алгоритму Винограда.

Несмотря на лучшую производительность по времени, алгоритм Винограда уступает в расходе памяти по сравнению со стандартным алгоритмом. Алгоритм Штрассена, в свою очередь, также требует дополнительной памяти для хранения промежуточных результатов. Это делает его менее эффективным, чем стандартный алгоритм и алгоритм Винограда. Это связано с тем, что в алгоритме Штрассена необходимо хранить дополнительные структуры данных, которые используются для оптимизации вычислений.

Таким образом, при выборе алгоритма умножения матриц, необходимо учитывать как производительность, так и расход памяти. Если необходимо сэкономить память, стандартный метод умножения матриц может оказаться более предпочтительным. Если нам важна скорость работы алгоритма, то оптимизированный алгоритм Винограда можеть оказаться более предпочтительным.

В результате выполнения лабораторной работы были выполнены следующие задачи:

1) разработаны и реализованы алгоритмы умножения матриц;

2) создан программный продукт, позволяющий протестировать реализованные алгоритмы;

3) проведен сравнительный анализ процессорного времени выполнения реализаций данных алгоритмов;

4) проведен сравнительный анализ затрачиваемой алгоритмами памяти;

5) был подготовлен отчет по лабораторной работе.

Список использованных источников

[1] И.В. Белоусов. Матрицы и опредители [Электронный ресурс]. — Режим доступа: https://eqworld.ipmnet.ru/ru/library/books/Belousov2006ru.pdf (дата обращения: 22.10.2023).

[2] Умножение матриц [Электронный ресурс]. — Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 22.10.2023).

[3] А. Ахо Дж. Хопкрофт Дж. Ульман. Построение и анализ вычислительных алгоритмов. Мир, 1979. С. 259–261.

[4] C library function clock() [Электронный ресурс]. — Режим доступа: https://en.cppreference.com/w/c/chrono/clock_t (дата обращения: 26.10.2023).