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»

на тему: «Алгоритмы умножения матриц»

Студент группы ИУ7-54Б _____ (Подпись, дата) Спирин М.П. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Строганов Ю. В.. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будут рассмотрены алгоритмы умножения матриц. В программировании, как и в математике, на практике часто приходится прибегать к использованию матриц [1]. Существует огромное количество областей их применения в этих сферах. Матрицы активно используются при выводе различных формул в физике, например:

— градиент;

— дивергенция;

— ротор.

Нельзя обойти стороной и различные операции над матрицами — сложение, возведение в степень, умножение. При различных задачах размеры матрицы могут достигать больших значений, поэтому оптимизация операций работы над матрицами является важной задачей в программировании. В данной лабораторной работе пойдёт речь об оптимизациях операции умножения матриц.

Целью данной лабораторной работы является описание алгоритмов умножения матриц.

Для поставленной цели необходимо выполнить следующие задачи.

1) Рассмотреть три алгоритма умножения матриц.
2) Создать программное обеспечение, реализующее следующие алгоритмы:
   — классический алгоритм умножения матриц;
   — алгоритм Винограда;
   — оптимизированный алгоритм Винограда.
3) Оценить трудоемкость алгоритмов умножения матриц.
4) Провести анализ затрат работы программы по времени.
5) Провести сравнительный анализ алгоритмов.

1 Аналитическая часть

В этом разделе будут рассмотрены классический алгоритм умножения матриц и алгоритм Винограда, а также его оптимизированная версия.

1.1 Матрица

Матрицей называют таблицу чисел a_{ik} вида

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

состоящую из m строк и n столбцов [1]. Числа a_{ik} называются её *элементами*.

Пусть A — матрица, тогда $A_{i,j}$ — элемент этой матрицы, который находится на *i-ой* строке и *j-ом* столбце.

Можно выделить следующие операции над матрицами:

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. В итоговой матрице количество строк будет, как у первой матрицы, а столбцов — как у второй.

Замечание: операция умножения матриц не коммутативна — если A и B — квадратные матрицы, а C — результат их перемножения, то произведение $A \times B$ и $B \times A$ дадут разный результат C.

1.2 Классический алгоритм

Пусть даны две матрицы:

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.2}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.3}$$

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].

1.3 Алгоритм Винограда

Алгоритм Винограда — алгоритм умножения квадратных матриц [2]. В 1987 году Дон Копперсмит и Виноград опубликовали метод, содержащий асимптотическую сложность $O(n^{2,3755})$ и улучшили его в 2011 до $O(n^{2,373})$, где n — размер сторон матрицы. После доработки он стал обладать лучшей асимптотикой среди всех алгоритмов умножения матриц.

Рассмотрим пример, пусть есть два вектора $U = (u_1, u_2, u_3, u_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $U \times W = u_1 \cdot w_1 + u_2 \cdot w_2 + u_3 \cdot w_3 + u_4 \cdot w_4$, что эквивалентно:

$$\begin{aligned} U \cdot W = (u_1 + w_2) \cdot (u_2 + w_1) + (u_3 + w_4) \cdot (u_4 + w_3) - \\ - u_1 \cdot u_2 - u_3 \cdot u_4 - w_1 \cdot w_2 - w_3 \cdot w_4. \end{aligned} \tag{1.5}$$

Пусть матрицы A, B, C, ранее определенных размеров.

Упомянутое скалярное произведение, по замыслу Винограда, можно выполнить по формуле:

$$C_{ij} = \sum_{k=1}^{q/2}(a_{i,2k-1} + b_{2k,j}) \cdot (a_{i,2k} + b_{2k-1,j}) - \\ - \sum_{k=1}^{q/2} a_{i,2k-1} \cdot a_{i,2k} - \sum_{k=1}^{q/2} b_{2k-1,j} \cdot b_{2k,j}. \tag{1.6}$$

Казалось бы, это только увеличит количество арифметических операций по сравнению с классическим методом, однако Виноград предложил находить второе и третье слагаемые в (1.6) предварительном этапе вычислений, заранее для каждой строки матрицы A и столбца B соответственно. Так, вычислив единожды для строки i матрицы A значение выражения $\sum_{k=1}^{q/2} a_{i,2k-1} \cdot a_{i,2k}$ его можно далее использовать m раз при нахождении элементов i-ой строчки матрицы C. Аналогично, вычислив единожды для столбца j матрицы B значение выражения $\sum_{k=1}^{q/2} b_{2k-1,j} \cdot b_{2k,j}$ его можно далее использовать n раз при нахождении элементов j-ого столбца матрицы C [2].

За счёт предварительной обработки данных можно получить прирост производительности: несмотря на то, что полученное выражение требует большего количества умножений и сложений, чем стандартное умножение матриц, выражение в правой части равенства можно вычислить заранее и запомнить для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволит выполнить лишь два умножения и пять сложений для квадратной матрицы размером четыре, при учёте, что потом будет сложено только с двумя предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее обычного алгоритма перемножения матриц.

Стоит упомянуть, что при нечётном значении размера любого из сторон матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.4 Оптимизированный алгоритм Винограда

При программной реализации рассмотренного выше алгоритма Винограда можно сделать следующие оптимизации:

1) значение $\frac{N}{2}$, используемое как ограничения цикла подсчёта предварительных данных, можно кэшировать;

2) операцию умножения на 2 программно эффективнее реализовывать как побитовый сдвиг влево на 1;

3) операции сложения и вычитания с присваиванием следует реализовывать при помощи соответствующего оператора $+=$ или $-=$ (при наличии данных операторов в выбранном языке программирования).

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц, а именно классический алгоритм и алгоритм Винограда, который имеет большую эффективность засчёт предварительных вычислений. Также были рассмотрены оптимизации, которые можно учесть при программной реализации алгоритма Винограда.

Алгоритмы будут получать на вход две матрицы, причём количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы. При вводе пустой матрицы будет выведено сообщение об ошибке. Требуется реализовать программное обеспечение, которое даёт возможность выбрать один из алгоритмов, ввести две матрицы и вывести результат их перемножения. Также необходимо провести замеры времени работы реализаций алгоритмов для чётных и нечётных размеров любых из сторон матриц и сравнить результаты, используя графическое представление.

2 Конструкторская часть

В этом разделе будут представлены описания используемых типов данных, а также схемы алгоритмов перемножения матриц – Винограда и оптимизации алгоритма Винограда

2.1 Требования к ПО

К программе предъявлен ряд функциональных требований:

— входные данные — размеры матрицы, целые числа и матрица, двумерный массив целых чисел, выходные данные — матрица;

— программа должна предоставлять интерфейс для выбора действий;

— программа должна давать возможность замерить процессорное время, затрачиваемое реализациями алгоритмов умножения матриц.

2.2 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма стандартного умножения матриц. На рисунках 2.2 и 2.3 — схема алгоритма Винограда, а на 2.4 и 2.5 — схема оптимизированного алгоритма Винограда.

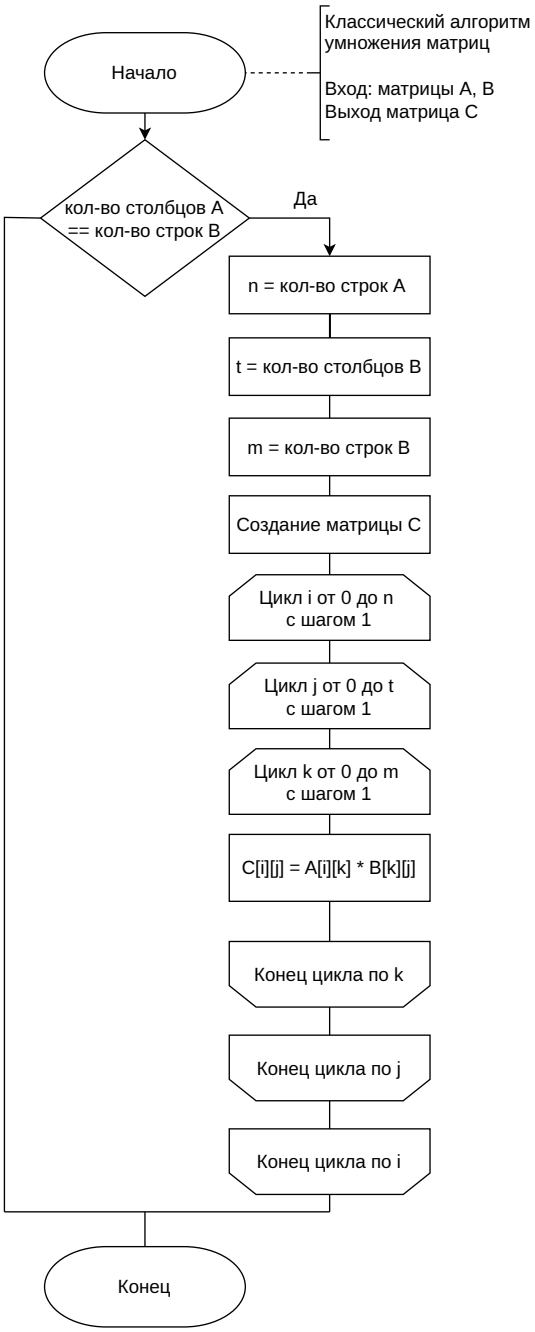

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

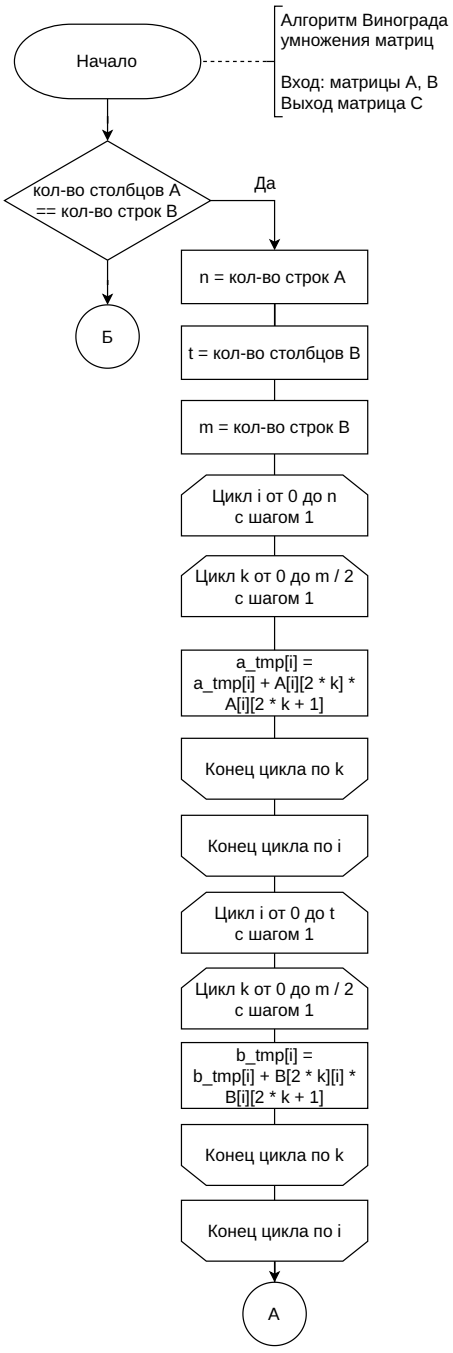

Рисунок 2.2 – Схема умножения матриц по алгоритму Винограда (часть 1)

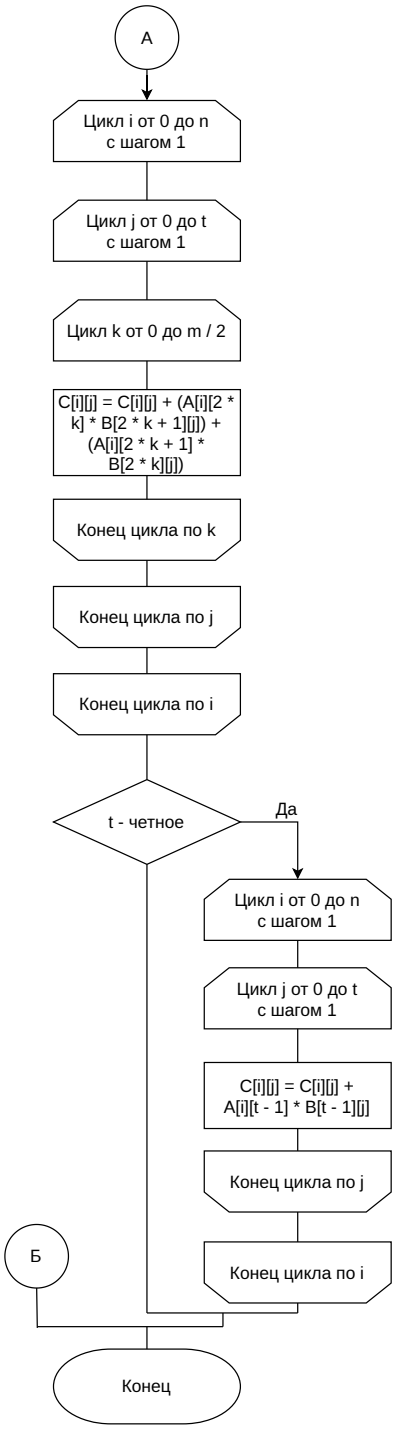

Рисунок 2.3 – Схема умножения матриц по алгоритму Винограда (часть 2)

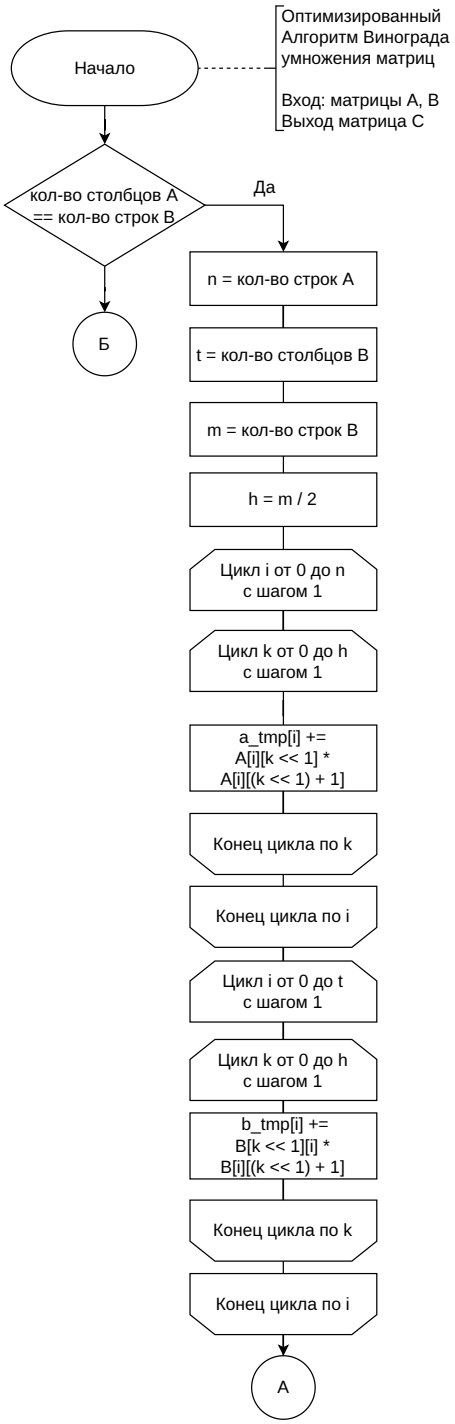

Рисунок 2.4 – Схема умножения матриц по оптимизированному алгоритму Винограда (часть 1)

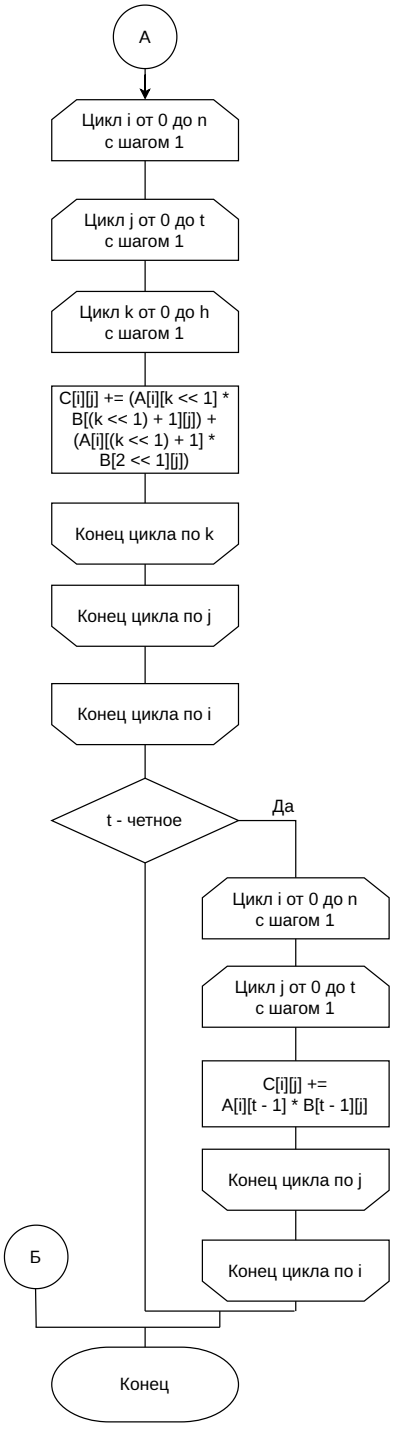

Рисунок 2.5 – Схема умножения матриц по оптимизированному алгоритму Винограда (часть 2)

2.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

— матрица — двумерный список целых чисел;

— вектор — одномерный список целых чисел.

2.4 Модель для оценки трудоемкости

Введем модель вычислений для определения трудоемкости каждого алгоритма умножения матриц.

1) Трудоемкость базовых операций:

   — для операций $+, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, --, \&\&, >>, <<, ||, \&, |$ трудоемкость равна 1;

   — для операций $*, /, \%, *=, /=, \%=$ трудоемкость равна 2.

2) Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай}, \\ max(f_1, f_2), & \text{худший случай}, \end{cases} \tag{2.1}$$

   где $f1$ и $f2$ — трудоемкости соответствующих ветвей условного оператора.

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}). \tag{2.2}$$

4) Трудоемкость передачи параметра в функции и возврат из функции равны 0.

2.5 Трудоемкость алгоритмов

2.5.1 Классический алгоритм

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из следующих компонент:

— внешнего цикла по $i \in [1 \dots N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;

— цикла по $j \in [1 \dots T]$, трудоёмкость которого: $f = 2+2+T \cdot (2+f_{body})$;

— цикла по $k \in [1 \dots M]$, трудоёмкость которого: $f = 2 + 2 + 14M$;

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то она вычисляется по следующей формуле:

$$\begin{aligned} f_{standart} &= 2 + N \cdot (2 + 2 + T \cdot (2 + 2 + M \cdot (2 + 8 + 1 + 1 + 2))) = \\ &= 2 + 4N + 4NT + 14NMT \approx 14NMT = O(N^3). \end{aligned} \tag{2.3}$$

2.5.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, необходимо рассчитать трудоемкость его частей.

1) Создание и инициализация массивов a_{tmp} и b_{tmp}, трудоёмкость которых вычисляется по формуле:

$$f_{init} = N + M. \tag{2.4}$$

2) Заполнение массива a_{tmp}, трудоёмкость которого вычисляется по формуле:

$$\begin{gathered} f_{atmp} = 2 + N \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 4N + \frac{19NM}{2} = 2 + 4N + 9,5NM. \end{gathered} \tag{2.5}$$

3) Заполнение массива b_{tmp}, трудоёмкость которого указана вычисляется по формуле:

$$\begin{gathered} f_{btmp} = 2 + T \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 4T + \frac{19TM}{2} = 2 + 4T + 9,5TM. \end{gathered} \tag{2.6}$$

4) Цикл заполнения для чётных размеров, трудоёмкость которого считается по формуле:

$$\begin{gathered} f_{cycle} = 2 + N \cdot (4 + T \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 28))) = \\ = 2 + 4N + 13NT + \frac{32NTM}{2} = 2 + 4N + 13NT + 16NTM \end{gathered} \tag{2.7}$$

5) Цикл, который дополнительно нужен для подсчёта значений при нечётном размере любой из сторон матрицы, трудоемкость которого высчитывается по формуле:

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная}, \\ 2 + N \cdot (4 + T \cdot (2 + 14)), & \text{иначе}. \end{cases} \tag{2.8}$$

Тогда для худшего случая (нечётный размер любой стороны матриц) имеем формулу:

$$f_{worst} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3). \quad (2.9)$$

Для лучшего случая (чётный размер любой стороны матриц) имеем формулу:

$$f_{best} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3). \quad (2.10)$$

2.5.3 Оптимизированный алгоритм Винограда

Чтобы вычислить трудоемкость оптимизированного алгоритма Винограда, необходимо рассчитать трудоемкость его частей.

1) Кэширование значения $\frac{M}{2}$ в циклах, которое равно 3.

2) Создание и инициализация массивов a_{tmp} и b_{tmp} по формуле (2.4).

3) Заполнение массива a_{tmp}, трудоёмкость которого вычислена в формуле (2.5);

4) Заполнение массива b_{tmp}, трудоёмкость которого вычислена в формуле (2.6);

5) Цикл заполнения для чётных размеров, трудоёмкость которого считается по следующей формуле:

$$\begin{aligned} f_{cycle} &= 2 + N \cdot (4 + T \cdot (4 + 7 + \frac{M}{2} \cdot (2 + 10 + 5 + 2 + 4))) = \\ &= 2 + 4N + 11NT + 11,5 \cdot NTM. \end{aligned} \quad (2.11)$$

6) Условие, нужное для дополнительных вычислений при нечётном размере матрицы, трудоемкость которого считается по следующей формуле:

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная}, \\ 2 + N \cdot (4 + T \cdot (2 + 10)), & \text{иначе}. \end{cases} \quad (2.12)$$

Тогда для худшего случая (нечётный общий размер матриц) имеем формулу:

$$\begin{aligned} f_{worst} &= 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx \\ &\approx 11NMT = O(N^3). \end{aligned} \tag{2.13}$$

Для лучшего случая (чётный общий размер матриц) имеем формулу:

$$f_{best} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3). \tag{2.14}$$

Вывод

В данном разделе были построены схемы алгоритмов умножения матриц, рассматриваемых в лабораторной работе, были описаны классы эквивалентности для функционального тестирования и модули программы. Проведённая теоретическая оценка трудоемкости алгоритмов показала, что трудоёмкость алгоритма Винограда в случае оптимизации в 1.2 раза меньше, чем у простого алгоритма Винограда.

3 Технологическая часть

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

3.1 Средства реализации

Для написания реализации данной лабораторной работы был выбран язык программирования C++ [3]. Данный выбор обусловлен наличием у языка встроенной библиотеки измерения процессорного времени и соответствием с выдвинутыми техническими требованиям.

Для измерения времени использовалось функция *clock()* из модуля *ctime* [4].

3.2 Сведения о файлах программы

Данная программа разбита на следующие файлы:

— `main.cpp` — файл, содержащий точку входа в программу. В не происходит общение с пользователем и вызов алгоритмов;

— `matrix.cpp` — файл содержит алгоритмы умножения матриц, вывод матриц;

— `io.cpp` — файл содержит функции чтение матрицы;

— `cpu_time.cpp` — файл содержит функции, замеряющее процессорное время, затрачиваемое реализациями алгоритмов умножения матриц.

3.3 Реализация алгоритмов

В листингах 3.1 – 3.3 реализованы алгоритмы умножения матриц — классический, алгоритм Винограда и оптимизированный алгоритм Винограда.

Листинг 3.1 – Функция, реализующая классический алгоритм умножения матриц

```
1  Matrix* Matrix::MultClassic(Matrix &other)
2  {
3      if (columns_ != other.rows_)
4          return nullptr;
5
6      Matrix *res = new Matrix(rows_, other.columns_);
7      size_t n = rows_;
8      size_t m = other.rows_;
9      size_t t = other.columns_;
10
11     for (int i = 0; i < n; i++)
12         for (int j = 0; j < t; j++)
13             for (int k = 0; k < m; k++)
14                 res->matrix_[i][j] += matrix_[i][k] *
                       other.matrix_[k][j];
```

```
15
16      return res;
17  }
```

Листинг 3.2 – Функция, реализующая умножение матриц по алгоритму Винограда

```
1  Matrix* Matrix::MultVinograd(Matrix &other)
2  {
3      Matrix *res = new Matrix(rows_, other.columns_);
4      std::vector<int> row_sum, col_sum;
5      row_sum.resize(rows_, 0);
6      col_sum.resize(other.rows_, 0);
7      for (int i = 0; i < rows_; i++) {
8          for (int j = 0; j < other.rows_ / 2; j++) {
9              row_sum[i] = row_sum[i] + matrix_[i][2 * j] *
                   matrix_[i][2 * j + 1];
10         }
11     }
12     for (size_t i = 0; i < other.columns_; i++)
13         for (size_t j = 0; j < other.rows_ / 2; j++)
14             col_sum[i] = col_sum[i] + other.matrix_[2 * j][i] *
15              other.matrix_[2 * j + 1][i];
16     for (size_t i = 0; i < rows_; i++)
17         for (size_t j = 0; j < other.columns_; j++) {
18             res->matrix_[i][j] = -row_sum[i] - col_sum[j];
19             for (size_t k = 0; k < other.rows_ / 2; k++)
20                 res->matrix_[i][j] = res->matrix_[i][j]
21                 + (matrix_[i][2 * k + 1] + other.matrix_[2 *
                       k][j]) *
22                 (matrix_[i][2 * k] + other.matrix_[2 * k +
                       1][j]);
23         }
24     if (other.rows_ % 2 == 1)
25         for(size_t i = 0; i < rows_; i++)
26             for(size_t j = 0; j < other.columns_; j++)
27                 res->matrix_[i][j] = res->matrix_[i][j] +
                       matrix_[i][other.rows_ - 1] *
                       other.matrix_[other.rows_ - 1][j];
28     return res;
29 }
```

Листинг 3.3 – Функция, реализующая умножение матриц по оптимизированному алгоритму Винограда

```
1  Matrix* Matrix::MultOptimized(Matrix &other)
2  {
3      Matrix *res = new Matrix(rows_, other.columns_);
4      int half_rows = other.rows_ / 2;
5      std::vector<int> row_sum, col_sum;
6      row_sum.resize(rows_, 0);
7      col_sum.resize(other.rows_, 0);
8      for (int i = 0; i < rows_; i++) {
9          for (int j = 0; j < half_rows; j++) {
10             row_sum[i] += matrix_[i][j << 1] * matrix_[i][(j <<
                   1) + 1];
11         }
12     }
13     for (size_t i = 0; i < other.columns_; i++)
14         for (size_t j = 0; j < half_rows; j++)
15             col_sum[i] += other.matrix_[j << 1][i] *
16              other.matrix_[(j << 1) + 1][i];
17     for (size_t i = 0; i < rows_; i++)
18         for (size_t j = 0; j < other.columns_; j++) {
19             res->matrix_[i][j] = -row_sum[i] - col_sum[j];
20             for (size_t k = 0; k < half_rows; k++) {
21                 int column = k << 1;
22                 res->matrix_[i][j] += (matrix_[i][column + 1] +
                       other.matrix_[column][j]) *
23                 (matrix_[i][column] + other.matrix_[column +
                       1][j]);
24             }
25         }
26     if (other.rows_ % 2 == 1) {
27         int rows = other.rows_ - 1;
28         for(size_t i = 0; i < rows_; i++)
29             for(size_t j = 0; j < other.columns_; j++)
30                 res->matrix_[i][j] += matrix_[i][rows] *
                       other.matrix_[rows][j];
31     }
32     return res;
33 }
```

3.4 Функциональные тесты

В таблице 3.1 и 3.2 приведены функциональные тесты для разработанных реализаций алгоритмов умножения матриц. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты для классического алгоритма умножения матриц

Входные данные		**Результат для классического алгоритма**	
Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 1 & 5 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 3 & 5 \\ 2 & 1 \\ 9 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 23 & 31 & 39 \\ 6 & 9 & 12 \end{pmatrix}$	$\begin{pmatrix} 23 & 31 & 39 \\ 6 & 9 & 12 \end{pmatrix}$
$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 35 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$

Таблица 3.2 – Функциональные тесты для умножения матриц по алгоритму Винограда

Входные данные		**Результат для алгоритма Винограда**	
Матрица 1	**Матрица 2**	**Ожидаемый результат**	**Фактический результат**
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 1 & 5 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 3 & 5 \\ 2 & 1 \\ 9 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке
$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 35 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$	$\begin{pmatrix} 350 \end{pmatrix}$

Вывод

Были представлены листинги реализаций всех алгоритмов умножения матриц — стандартного, Винограда и оптимизированного Винограда. Также в данном разделе была приведена информация о выбранных средствах для разработки реализаций алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: Intel(R) Core(TM) i7-1165G7 CPU 2.80 ГГц.

— Оперативная память: 15 ГБайт.

— Операционная система: Ubuntu 64-разрядная система версии 22.04.3.

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На рисунке 4.1 представлена демонстрация приложения для работы с алгоритмами умножения матриц на примере умножения матриц с помощью алгоритма Винограда.

```
Выберите команду: 1 - Найти произведение матриц с помощью алгоритма Винограда
2 - Найти произведение матриц с помощью оптимизированного алгоритма Винограда
3 - Найти произведение матриц с помощью классического алгоритма умножения
4 - Найти среднее время на выполнение каждого алгоритма
5 - Остановить выполнение
1
Ввод первой матрицы
Введите кол-во строк в матрице: 2
Введите кол-во столбцов в матрице: 2
Введите каждый элемент матрицы через пробел
1 2
3 4
Ввод второй матрицы
Введите кол-во строк в матрице: 2
Введите кол-во столбцов в матрице: 2
Введите каждый элемент матрицы через пробел
1 2
3 4
Результат:
7 10
15 22
```

Рисунок 4.1 – Демонстрация работы программы

4.3 Результаты замеров времени

В таблице 4.1 приведены результаты замеров времени, затрачиваемого на работу реализациями алгоритмов умножения матриц при четных размеров квадратных матриц от 10 до 80 с шагом 10 на различных входных данных.

Таблица 4.1 – Результаты замеров времени (чётные размеры матриц)

Размер матрицы	**Время, мс**		
	Винограда	**(опт.) Винограда**	**Классическая**
10	11.137	10.944	9.762
20	36.183	35.654	38.072
30	101.165	100.896	108.483
40	240.403	224.245	245.632
50	448.424	415.070	464.139
60	801.632	733.793	832.393
70	1216.046	1094.849	1250.042
80	1773.414	1596.995	1849.741

По таблице 4.1 был построен рисунок 4.2. Исходя из этих данных можно понять, что лучшего всего работает реализация оптимизированного алгоритма Винограда. На разметрах матриц выше 50 он превосходит по времени работы реализацию алгоритма Винограда на 8–10% для матриц, размер которых больше 40, а реализацию классического алгоритма — на 15% для матриц, размер которых больше 30.

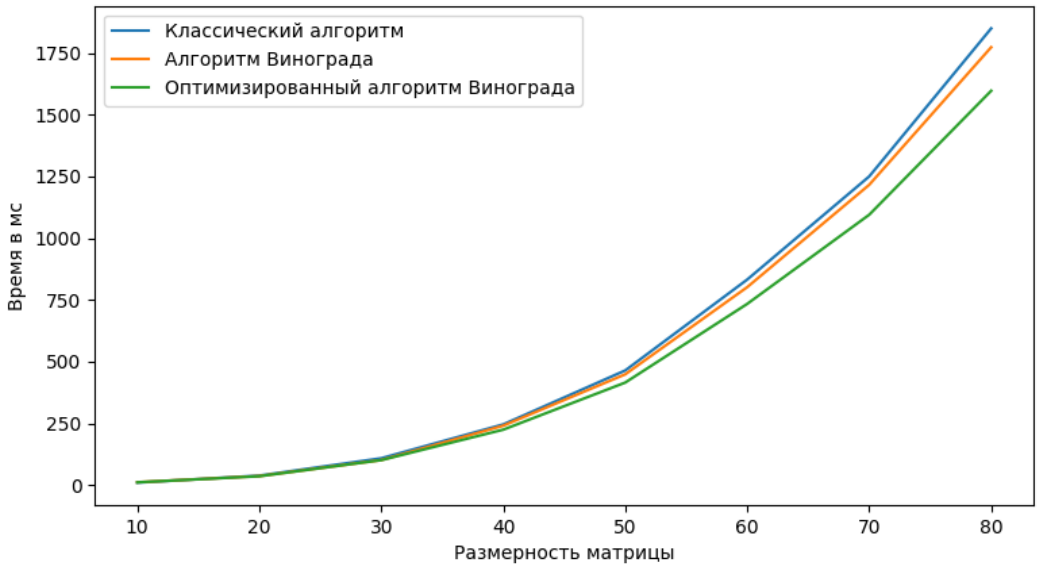

Рисунок 4.2 – Сравнение по времени работы реализаций алгоритмов умножения матриц на чётных размерах матриц

В таблице 4.2 приведены результаты замеров по времени работы реализаций алгоритмов умножения матриц при нечетных размеров квадратных матриц, размеров от 11 до 81 с шагом 10 на различных данных.

Таблица 4.2 – Результаты замеров времени (нечётные размеры матриц)

Размер матрицы	**Время, мс**		
	Винограда	**(опт.) Винограда**	**Классическая**
11	12.880	12.542	11.454
21	39.726	38.877	40.893
31	112.054	110.570	117.586
41	260.520	238.904	265.347
51	503.669	459.966	523.911
61	822.777	740.676	850.514
71	1279.714	1140.724	1309.459
81	1881.056	1693.167	1951.358

По таблице 4.2 был построен рисунок 4.3. Исходя из этих данных можно понять, что лучшего всего работает реализация оптимизированного алгоритма Винограда. На разметрах матриц выше 40 ее время работы меньше времени работы реализации алгоритма Винограда на 8–10 % для матриц, размер которых больше 40, а реализации классического алгоритма — на 16% на матрицах, размер которых больше 30.

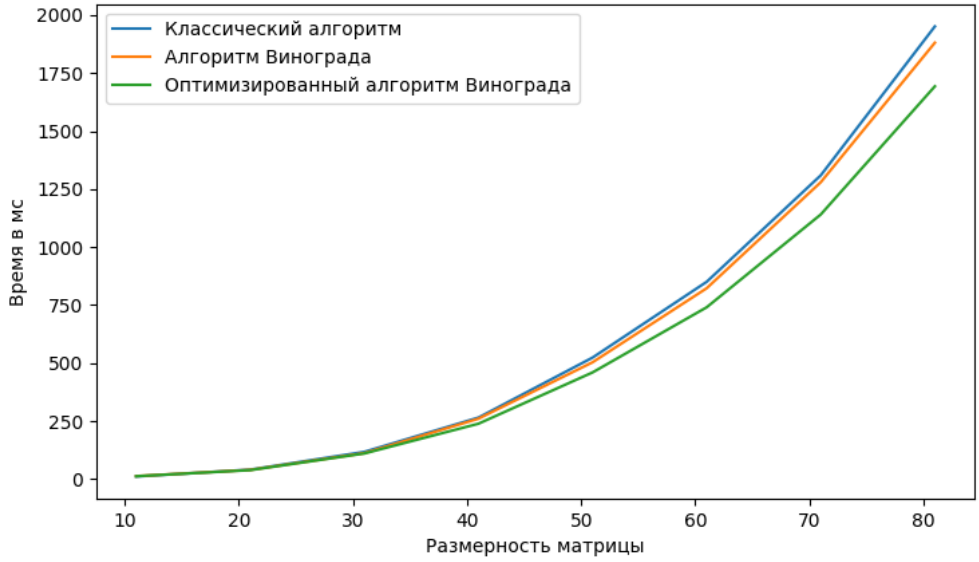

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц на нечётных размерах матриц

4.4 Вывод

В результате эксперимента можно сказать, что при таких данных следует использовать оптимизированный алгоритм Винограда, его реализация примерно на 9% быстрее реализации алгоритма Винограда на размерах матриц свыше 10 и на 12% быстрее реализации классического алгоритма умножения матриц на размерах матриц выше 40. Также при проведении эксперимента было выявлено, что на чётных размерах реализация алгоритма Винограда в 1.1 раза быстрее, чем на нечётных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов.

Заключение

В результате исследования было определено, что реализация классического алгоритма умножения матриц проигрывает по времени реализации алгоритма Винограда примерно на 11% при размерах матриц больше 10 из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций — операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает реализация оптимизированного алгоритма Винограда — он примерно на 10% быстрее реализации алгоритма Винограда на размерах матриц выше 10 из-за замены операций равно и плюс на операцию плюс-равно, а также за счёт замены операции умножения операцией сдвига, что дает проводить часть вычислений быстрее.

При выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда. Также стоит упомянуть, что написанная реализация алгоритма Винограда работает на чётных размерах матриц примерно в 1.1 раза быстрее, чем на нечётных, что связано с тем, что нужно произвести часть дополнительных вычислений для крайних строк и столбцов матриц.

Цель, которая была поставлена в начале лабораторной работы, была достигнута. В ходе выполнения лабораторной работы были решены все задачи.

1) Рассмотрены три алгоритма умножения матриц.
2) Создано программное обеспечение, реализующее следующие алгоритмы:
   - классический алгоритм умножения матриц;
   - алгоритм Винограда;
   - оптимизированный алгоритм Винограда.
3) Оценены трудоемкости алгоритмов умножения матриц.
4) Проведен анализ затрат работы программы по времени.
5) Проведен сравнительный анализ алгоритмов.

Список использованных источников

1. Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. — М.: Гуманит. изд. центр ВЛАДОС, 2003.

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: 25.09.2022).

4. C library function clock() [Электронный ресурс]. — Режим доступа: `https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm` (дата обращения: 20.10.2022).