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»

Лабораторная работа №2
по дисциплине "Анализ Алгоритмов"

Тема Умножение матриц

Студент Сабуров С. М.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение 3

1 Аналитическая часть 4
1.1 Стандартный алгоритм 4
1.2 Алгоритм Копперсмита – Винограда 4
1.3 Вывод 5

2 Конструкторская часть 7
2.1 Разработка алгоритмов 7
2.2 Трудоемкость алгоритмов 7
2.2.1 Классический алгоритм 8
2.2.2 Алгоритм Винограда 8
2.3 Оптимизированный алгоритм Винограда 8
2.4 Описание структур данных 9
2.5 Способы тестирования и классы эквивалентности 10
2.6 Вывод 10

3 Технологическая часть 14
3.1 Средства реализации 14
3.2 Листинги кода 14
3.3 Тестирование функций 18
3.4 Вывод 18

4 Исследовательская часть 19
4.1 Технические характеристики 19
4.2 Время выполнения алгоритмов 19
4.3 Вывод 19

Заключение 23

Список литература 24

Введение

Алгоритм Копперсмита - Винограда — алгоритм умножение квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2.3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита – Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита — Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров. Поэтому пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости.

Алгоритм Штрассена предназначен для быстрого умножения матриц. Он был разработан Фолькером Штрассеном в 1969 году и является обобщением метода умножения Карацубы на матрицы.

В отличие от традиционного алгоритма множения матриц, алгоритм Штрассена умножает матрицы за время $\Theta(n^{\log_2 7}) = O(n^{2.81})$

Несмотря на то, что алгоритм Штрассена является асимптотически не самым быстрым из существующих алгоритмов быстрого умножения матриц, он проще программируется и эффективнее при умножении матриц относительно малого размера.

Целью данной работы является реализация и анализ алгоритмов умножения матриц.

Для достижения поставленной цели необходимо выполнить следующие задачи:

— реализовать классический алгоритм умножения матриц;
— реализовать алгоритм Копперсмита — Винограда;
— реализовать улучшенный Алгоритм Копперсмита – Винограда;
— рассчитать их трудоемкость;
— сравнить их временные характеристики экспериментально;
— на основании проделанной работы сделать выводы.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы умножения матриц

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{pmatrix}, \tag{1}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}. \tag{2}$$

Тогда матрица C размерностью $l \times n$

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{pmatrix}, \tag{3}$$

в которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj}, \quad (i = \overline{1, l}; j = \overline{1, n}) \tag{4}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.2 Алгоритм Копперсмита – Винограда

В результате умножения двух матриц, каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $V \cdot W = v_1w_1 + \cdots + v_4w_4$, что эквивалентно

$$V \cdot W = (v_1 + w_1)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \quad (5)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного

1.3 Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения.

- Входные данные : количество строк в первой матрице, количество столбцов в первой матрице, элементы первой матрицы, Количество строк во второй матрице, количество столбцов во второй матрице, элементы второй матрицы.
- Выходные данные : на выходе имеем матрицу - результат умножения двух матриц, являющихся входными данными.
- Ограничения, в рамках которых будет работать программа : размеры матриц должны быть целыми положительными числами, элементы матриц должные быть также числами(допустим вещественный тип).
- Функциональные требования : функции, представленные на листингах 2 - 6 должны вычислять результат умножения двух матриц.

— Требования к программному обеспечению : к программе предъявляется ряд требований:
    — на вход подаются размеры матриц (натуральные числа) и самы матрицы, которые нужно перемножить;
    — на выходе - результаты умножения матриц алгоритмами простого умножения матриц, умножения матриц по Копперсмиту–Винограду и улучшенного умножения матриц по Копперсмиту–Винограду.

2 Конструкторская часть

В данном разделе будут приведены схемы и оценка трудоёмкостей алгоритмов.

2.1 Разработка алгоритмов

На рисунках 1-3 приведены схемы алгоритмов простого умножения матриц, умножения матриц по Копперсмиту–Винограду и улучшенного умножения матриц по Копперсмиту–Винограду соответственно.

Для алгоритма Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, так как отпадает необходимость в последнем цикле.

Данный алгоритм можно оптимизировать:

1. Убрать деления в цикле.
2. Замена выражения $a = a + \cdots$ на $a+ = \cdots$.
3. Увеличить шаг в цикле до 2.

2.2 Трудоемкость алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. $+, -, /, \%, =, \neq, <, >, \leq, \geq, [], *, ++$ — трудоемкость 1.
2. Трудоемкость оператора выбора *if* условие *then A else B* рассчитывается, как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A & \text{если условие выполняется,} \\ f_B & \text{иначе.} \end{cases} \tag{6}$$

3. Трудоемкость цикла расчитывается, как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}}). \tag{7}$$

4. Трудоемкость вызова функции равна 0.

2.2.1 Классический алгоритм

Трудоемкость классического алгоритма:

$10MNQ + 4MQ + 4M + 2$

2.2.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда:

Первый цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

Второй цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

Третий цикл: $13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2$

Условный переход:

$$\begin{cases} 2 & \text{,если размер матрицы нечетный} \\ 15 \cdot QM + 4 \cdot M + 2 & \text{,иначе} \end{cases}$$

Итого:

$$\frac{15}{2} \cdot MN + 5M + 2 + \frac{15}{2} \cdot MN + 5M + 2 + 13MNQ +$$
$$+ 12MQ + 4M + 2 + \begin{cases} 2 & \text{,если размер матрицы нечетный} \\ 15 \cdot MQ + 4M + 2 & \text{,иначе.} \end{cases}$$

2.3 Оптимизированный алгоритм Винограда

Рассмотрим трудоемкость оптимизированного алгоритма Винограда:

Первый цикл: $\frac{11}{2} \cdot MN + 4M + 2$

Второй цикл: $\frac{11}{2} \cdot MN + 4M + 2$

Третий цикл: $\frac{15}{2} \cdot MNQ + 9MQ + 4M + 2$

Условный переход: $\begin{cases} 1 & \text{, если размер матрицы нечетный} \\ 10MQ + 4M + 2 & \text{, иначе.} \end{cases}$

Итого: $\frac{11}{2} \cdot MN + 4M + 2 + \frac{11}{2}MN + 4M + 2 + \frac{15}{2}MNQ + 9MQ + 4M + 2 + \begin{cases} 1 & \text{, если размер матрицы нечетный} \\ 10MQ + 4M + 2 & \text{, иначе.} \end{cases}$

2.4 Описание структур данных

Был реализован класс Matrix, объединивший в себе алгоритмы работы с матрицей и элементы матрицы

Листинг 1: Описание класса Matrix

```
template<class T>
class Matrix
{
private:
        int M;
        int N;
        std::shared_ptr<std::shared_ptr<T[]>[]> data;
public:
        Matrix();
        Matrix(const Matrix<T>&M_);

        Matrix(int M_, int N_);



        Matrix<T> classic_mult(const  Matrix<T> &M2);



        Matrix<T>  Vinograd_mult(const Matrix<T>& M2);

        Matrix<T>  Vinograd_mult_opt(const Matrix<T>& M2);

        ~Matrix<T>();

        Matrix<T>& input_matrix();

        void output_matrix();

};
```

2.5 Способы тестирования и классы эквивалентности

Была выбрана методика тестирования черным ящиком. Классы эквивалентности:

— Матрицы одинкаовых размеров.
— Количество столбцов первой матрицы равно количеству строк матрицы, при этом матрицы не одинаковых размеров.
— Матрицы представляют собой 1 элемент.
— Количество столбцов первой матрицы не равно количеству строк матрицы.

2.6 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов и проведена теоретическая оценка трудоемкости.

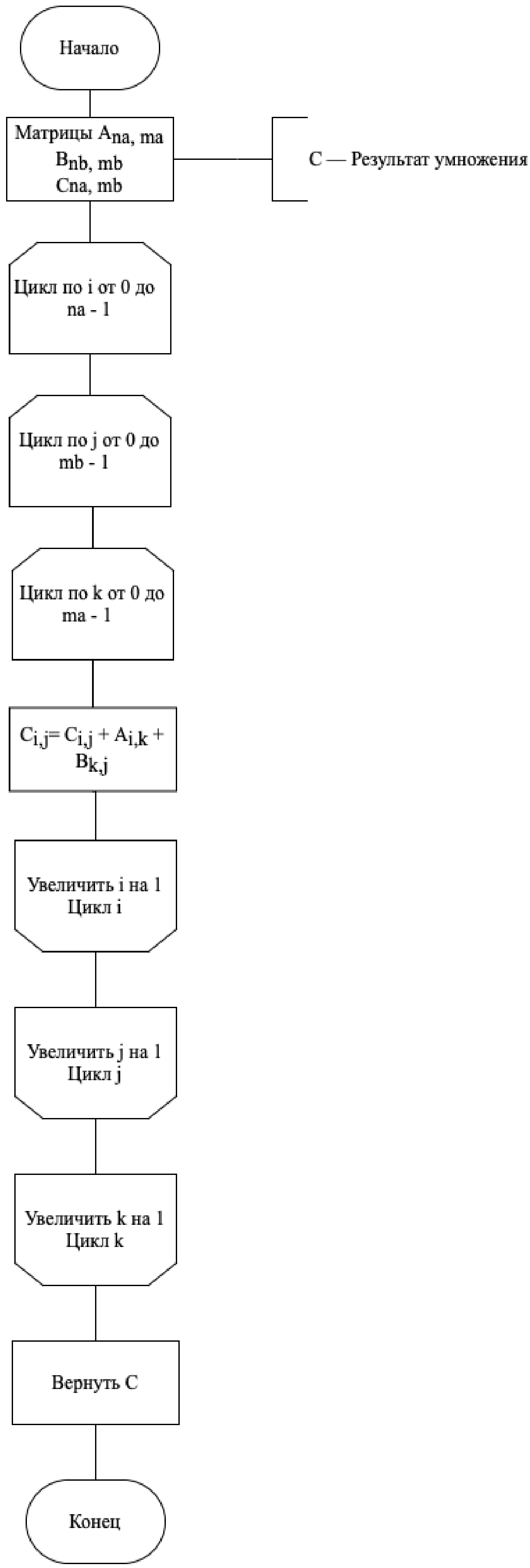

Рис. 1 — Схема алгоритма простого умножения матриц

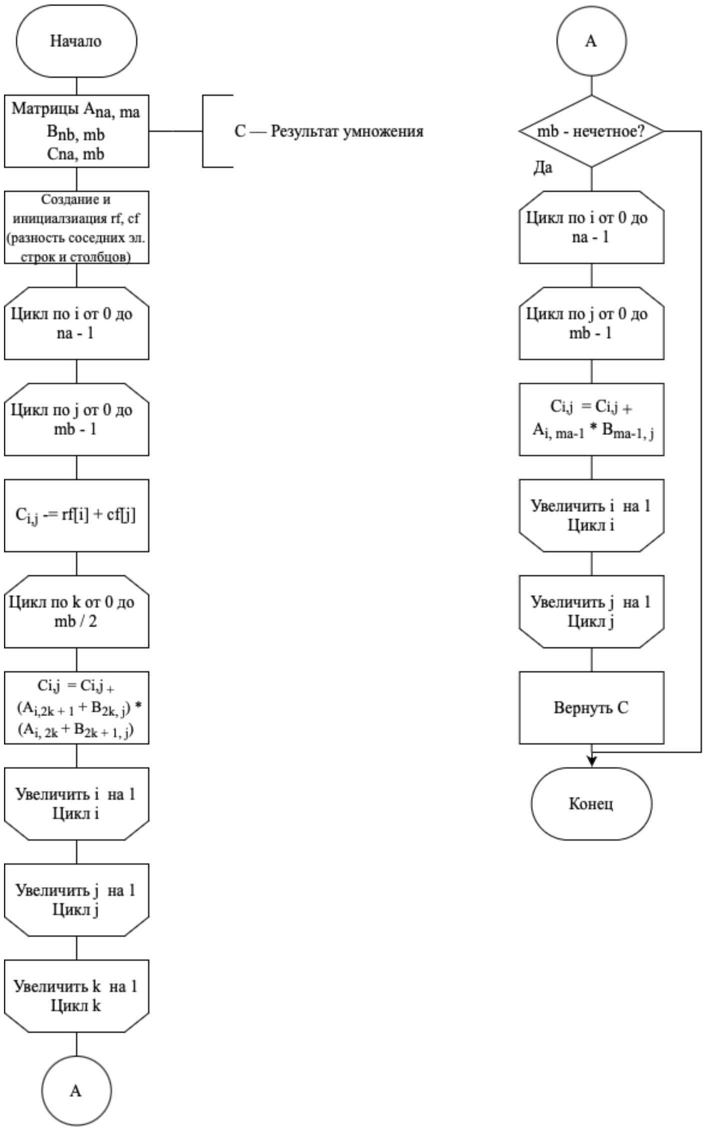

Рис. 2 — Схема алгоритма Винограда умножения матриц

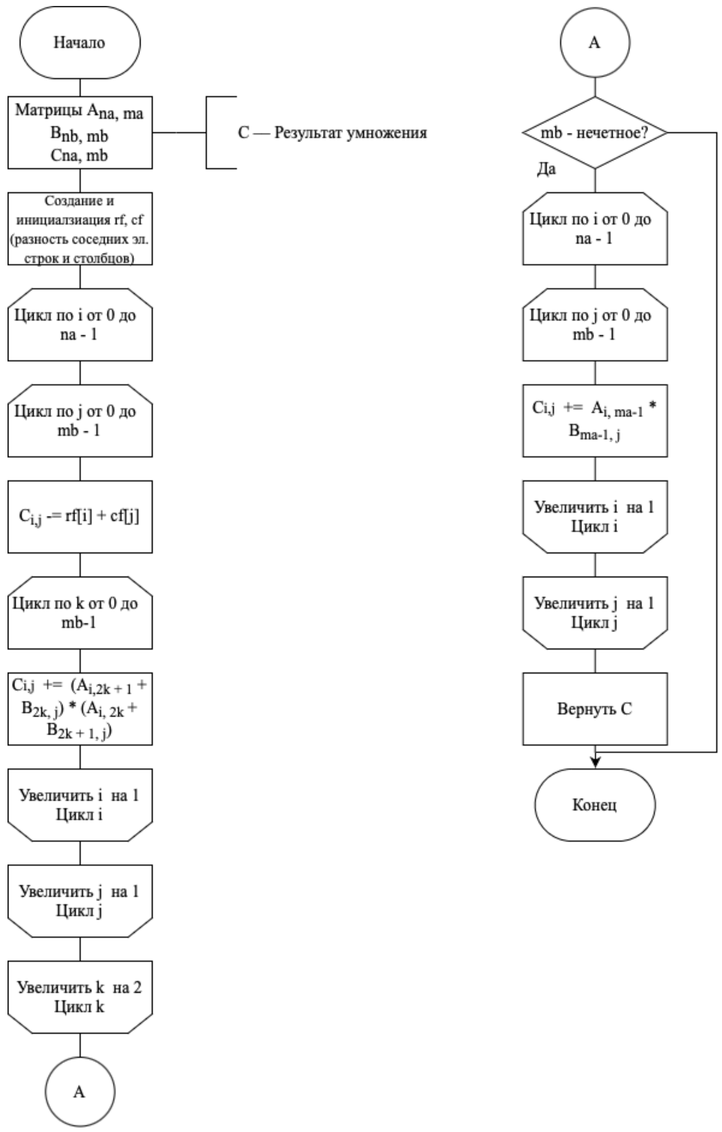

Рис. 3 — Схема оптимизированного алгоритма Винограда умножения матриц

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Средства реализации

Для реализации программ был выбран язык программирования C++ [1]. Данный язык был выбран потому, что в нем присутствует инструментарий для замера процессорного времени и тестирования.

3.2 Листинги кода

Листинг 2: Алгоритм простого умножения матриц

```
template <typename T>
Matrix<T> Matrix<T>::classic_mult( const Matrix<T>& M2)
{
        Matrix<T> res(this->M, M2.N);
        int K = this->N;
        for (int i = 0; i < res.M; i++)
        {
                for (int j = 0; j < res.N; j++)
                {
                        for (int k = 0; k < K; k++)
                        {
                                res.data[i][j] = res.data[i][j] + (this->data[i][k] * M2.data[k][j]);
                        }
                }
        }
        return res;

}
```

Листинг 3: Алгоритм умножения матриц Винограда

```
Matrix<T>  Matrix<T>::Vinograd_mult(const Matrix<T>& M2)
{
```

```
        int C = this->N;
        Matrix<T> c_matr(this->M, M2.N);
        Matrix<T> matr_r = rows_sum_matr(*this, this->M, C);
        Matrix<T> matr_c = cols_sum_matr(M2, C, M2.N);
        for (int i = 0; i < c_matr.M; i++)
        {
                for (int j = 0; j < c_matr.N; j++)
                {
                        for (int k = 0; k < (C / 2); k++)
                        {
                                c_matr.data[i][j] += ((this->
    data[i][k * 2] + M2.data[k * 2 + 1][j]) * (this->data[i][k
     * 2 + 1] + M2.data[k * 2][j]) - matr_r.data[i][k] -
    matr_c.data[k][j]);
                        }
                }
        }
        if (C \% 2 != 0)
        {
                for (int i = 0; i < c_matr.M; i++)
                {
                        for (int j = 0; j < c_matr.N; j++)
                        {
                                c_matr.data[i][j] += this->
    data[i][C - 1] * M2.data[C - 1][j];
                        }
                }
        }
        return c_matr;
}
```

Листинг 4: Вспомогательные процедуры для алгоритма умножения матриц Винограда

```
template <typename T>
Matrix<T> rows_sum_matr(Matrix<T> matr, int M_, int C_)
{
        C_ = C_ / 2;
        Matrix<T> c_matr(M_, C_);
        for (int i = 0; i < M_; i++)
        {
```

```
                for (int j = 0; j < C_; j++)
                {
                        c_matr.data[i][j] = c_matr.data[i][j]
   + matr.data[i][j * 2] * matr.data[i][j * 2 + 1];
                }
        }
        return c_matr;
}template <typename T>
Matrix<T> cols_sum_matr(Matrix<T> matr, int C_, int N_)
{
        C_ = C_ / 2;
        Matrix<T> c_matr(C_, N_);
        for (int i = 0; i < C_; i++)
        {
                for (int j = 0; j < N_; j++)
                {
                        c_matr.data[i][j] += matr.data[i *
   2][j] * matr.data[i * 2 + 1][j];
                }
        }
        return c_matr;
}
```

Листинг 5: Оптимизированный лгоритм умножения матриц Винограда

```
Matrix<T>  Matrix<T>::Vinograd_mult(const Matrix<T>& M2)
{
        int C = this->N;
        Matrix<T> c_matr(this->M, M2.N);
        Matrix<T> matr_r = rows_sum_matr(*this, this->M, C);
        Matrix<T> matr_c = cols_sum_matr(M2, C, M2.N);
        for (int i = 0; i < c_matr.M; i++)
        {
                for (int j = 0; j < c_matr.N; j++)
                {
                        for (int k = 0; k < (C / 2); k++)
                        {
                                c_matr.data[i][j] += ((this->
   data[i][k * 2] + M2.data[k * 2 + 1][j]) * (this->data[i][k
    * 2 + 1] + M2.data[k * 2][j]) - matr_r.data[i][k] -
   matr_c.data[k][j]);
```

```
                        }
                }
        }
        if (C \% 2 != 0)
        {
                for (int i = 0; i < c_matr.M; i++)
                {
                        for (int j = 0; j < c_matr.N; j++)
                        {
                                c_matr.data[i][j] += this->
   data[i][C - 1] * M2.data[C - 1][j];
                        }
                }
        }
        return c_matr;
}
```

Листинг 6: Вспомогательные процедуры для оптимизированного алгоритма умножения матриц Винограда

```
template <typename T>
Matrix<T> rows_sum_matr(Matrix<T> matr, int M_, int C_)
{
        C_ = C_ / 2;
        Matrix<T> c_matr(M_, C_);
        for (int i = 0; i < M_; i++)
        {
                for (int j = 0; j < C_; j++)
                {
                        c_matr.data[i][j] = c_matr.data[i][j]
    + matr.data[i][j * 2] * matr.data[i][j * 2 + 1];
                }
        }
        return c_matr;
}template <typename T>
Matrix<T> cols_sum_matr(Matrix<T> matr, int C_, int N_)
{
        C_ = C_ / 2;
        Matrix<T> c_matr(C_, N_);
        for (int i = 0; i < C_; i++)
        {
```

```
                for (int j = 0; j < N_; j++)
                {
                    c_matr.data[i][j] += matr.data[i *
    2][j] * matr.data[i * 2 + 1][j];
                }
        }
        return c_matr;
}
```

3.3 Тестирование функций

В таблице 1 приведены модульные тесты для функций умножения матриц выше перечисленными методами. Все тесты были пройдены успешно.

Таблица 1 — Тестирование функций умножения матриц

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}30 & 36 & 42\\66 & 81 & 96\\102 & 126 & 150\end{pmatrix}$
$\begin{pmatrix}1 & 2\\4 & 5\end{pmatrix}$	$\begin{pmatrix}1 & 2\\4 & 5\end{pmatrix}$	$\begin{pmatrix}9 & 12\\24 & 33\end{pmatrix}$
$\begin{pmatrix}8\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$	$\begin{pmatrix}32\end{pmatrix}$
$\begin{pmatrix}1 & 2\end{pmatrix}$	$\begin{pmatrix}3 & 4\end{pmatrix}$	Умножение невозможно

3.4 Вывод

Были разработаны и протестированы реализации алгоритмов: простой алгоритм умножения матриц, алгоритм умножения матриц по Копперсмиту – Винограду и улучшенный алгоритм умножения матриц по Копперсмиту – Винограду.

4 Исследовательская часть

В данном разделе будут приведены результаты исследовательской деятельности - замеры процессорного времени работы алгоритмов и тестирование алгоритмов.

4.1 Технические характеристики

Технические характеристики электронно-вычислительнй машины, на которой выполнялось тестирование:

- операционная система: Windows 10 64-bit;
- оперативная память: 8 гигабайт ;
- процессор: Intel i5 7th gen.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Был проведен замер времени работы каждого из алгоритмов с помощью функции GetProcessTimes. Эта функция замеряет процессорное время выполнения функции и усредняет его (проводится 15 замеров). В таблицах 2, 3 содержатся результаты исследований при четном и нечетном размерах матриц.

На рисунках 4, 5 демонстрируется зависимость времени выполнения конкретных реалзиаций алгоритмов умножения матриц от размера стороны квадратной матрицы.

4.3 Вывод

Время работы реализации алгоритма Копперсмита–Винограда быстрее классического алгоритма умножения матриц примерно на 25-30% быстрее. В то же

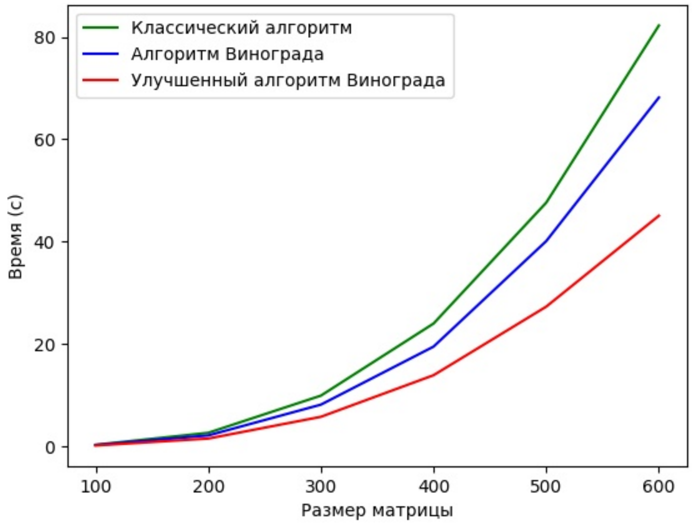

Рис. 4 — Зависимость времени выполнения алгоритмов от четного размера стороны квадратной матрицы

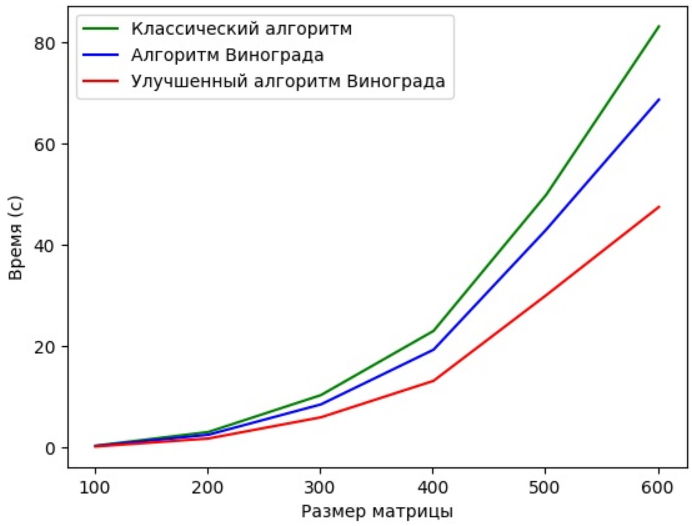

Рис. 5 — Зависимость времени выполнения алгоритмов от нечетного размера стороны квадратной матрицы

Таблица 2 — Время выполнения реализаций алгоритмов (в секундах) при четном размере матрицы

Размер	К	В	ОВ
100	0.34375	0.290625	0.203125
200	2.68125	2.16562	1.55625
300	9.94375	8.18437	5.80625
400	24.025	19.5188	13.9313
500	47.6313	40.1219	27.3219
600	82.2344	68.1656	45.0781

время оптимизированный алгоритм Копперсмита–Винограда быстрее оригинального на 10-15%. Таким образом на матрицах значительного размера (больше 200) следует использовать алгоритм Копперсмита–Винограда, так как он значительно быстрее (таблица 2).

Таблица 3 — Время выполнения реализаций алгоритмов (в секундах) при нечетном размере матрицы

Размер	К	В	ОВ
101	0.3781255	0.31875	0.228125
201	3.08438	2.54375	1.8
301	10.3781	8.54375	5.99375
401	23.0688	19.3531	13.2031
501	49.9688	43.0812	30.1281
601	83.2219	68.7594	47.5594

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

— реализовать классический алгоритм умножения матриц;
— реализовать алгоритм Копперсмита — Винограда;
— реализовать улучшенный Алгоритм Копперсмита — Винограда;
— рассчитать их трудоемкость;
— сравнить их временные характеристики экспериментально;
— на основании проделанной работы сделать выводы.

Экспериментально были установлены различия в производительности различных алгоритмов умножения матриц. Оптимизированный алгоритм Копперсмита–Винограда имеет меньшую сложность, нежели классический алгоритм умножения матриц.

Список литература

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы. Построение и анализ. Издательский дом “Вильямс”, 2011. 823 - 869.
2. Б. Страуструп. Язык программирования С++ . Addison-Wesley, 2000. 142 - 149.
3. Г. Шилдт. С++. Полное руководство. СПб.: Наука и Техника, Издательский дом “Вильямс”, 2006. 621 - 693.
4. Я. Галовиц. С++17 STL. Стандартная библиотека шаблонов. Серийная библиотека программиста, 2018. 91 - 123.
5. Р. Седжвик. Фундаментальные алгоритмы С++. Diasoft, 2001. 42 - 69.