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: Ле Ни Куанг

Группа: ИУ7и-56Б

Преподаватель: Волкова Л. Л.
Строганов Ю. В.

Москва
2020

Оглавление

Введение

Умножение матриц - одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

Целью работы: изучение алгоритмов умножения матриц. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда. Также требуется изучить рассчет сложности алгоритмов, получить навыки в улучшении алгоритмов.

Задачи работы:

1. изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда, оптимизировать алгоритм Винограда.
2. дать теоретическую оценку алгоритмы (трудоемкость).
3. реализовать три алгоритма умножения матриц.
4. сравнить алгоритмы умножения матриц.

1 Аналитический раздел

В данном разделе будет приведено описание алгоритмов и модель вычислений для оценок трудоемкости.

1.1 Описание алгоритмов

1.1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы A и B размерности l x m и m x n соответственно:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \qquad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}$$

Тогда матрица C размерностью l x m:

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

в которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = 1, 2, ..., l;\ j = 1, 2, ..., n) \tag{1.1}$$

называется их произведением.

1.1.2 Алгоритм Винограда

Рассматривая результат умножения двух матриц очевидно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. 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:

$$V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4 \tag{1.2}$$

Это равенство можно переписать в виде:

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{1.3}$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем первое: 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, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.1.3 Модель вычислений

В данной работы используется следующая модель вычислений:

1. Стоимость базовых операций: $F = 1$
   $(=, *, +, -, /, \%, <, <=, >, >=, ==, != , [], +=, -=, *=, /=)$

2. Стоимость цикла for
   $F_{for} = f_{init} + f_{compare} + N_{loop} \cdot (f_{body} + f_{inc} + f_{compare})$

3. Трудоемкость условного оператора if

$$F_{if} = f_{compare} + f_{body} = f_{compare} + \begin{cases} f_{min}, & \text{лучший случай} \\ f_{max}, & \text{худший случай} \end{cases}$$

1.2 Вывод

Были приведено описание алгоритмов, стандартный и Винограда, также рассмотрено модель вычислений для оценок трудоемкости.

2 Конструкторский раздел

В данном разделе будет приведено описание схем алгоритмов и вычислены их трудоемкости.

2.1 Разработка алгоритмов

На рисунках показаны схемы алгоритмов умножения матриц, стандартный и алгоритм Винограда.

2.1.1 Схема стандартного алгоритма умножения матриц

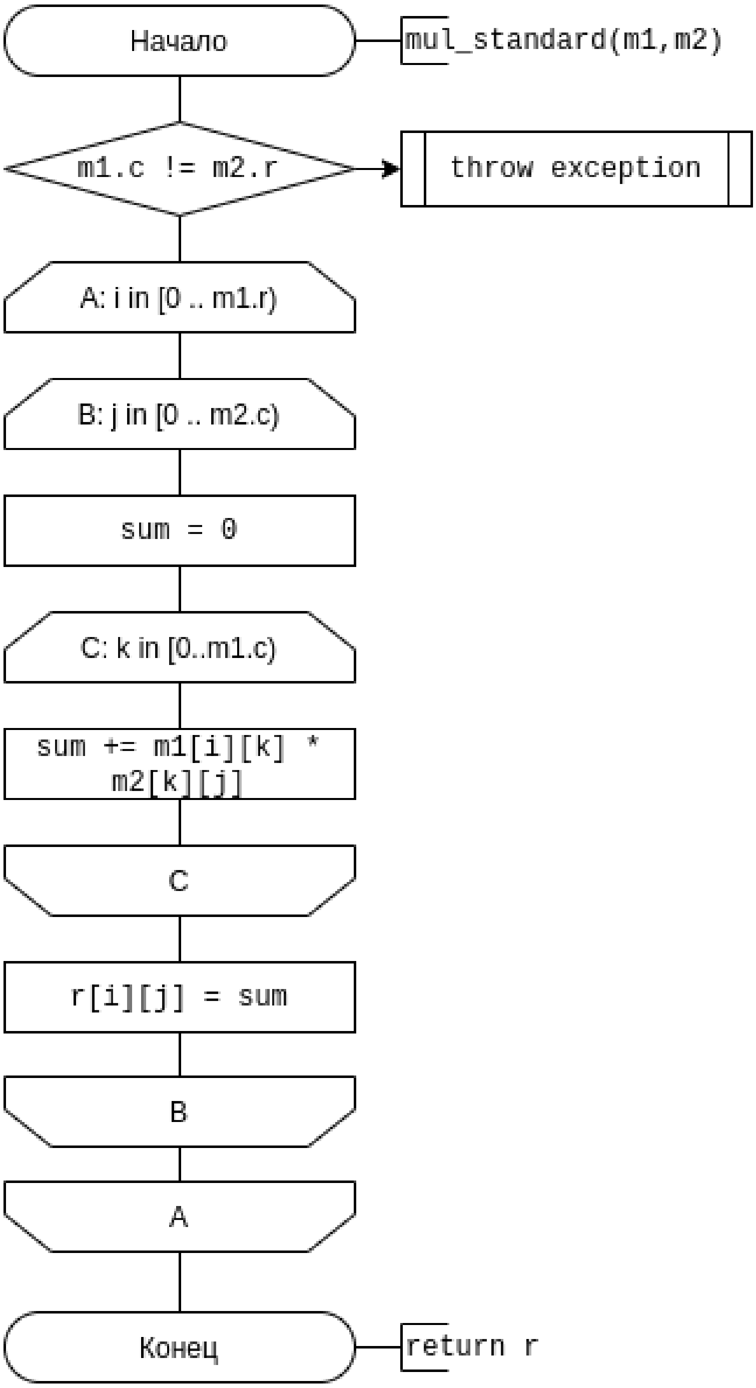

Рис. 2.1: Схема стандартного алгоритма умножения матриц

2.1.2 Схема алгоритма Винограда

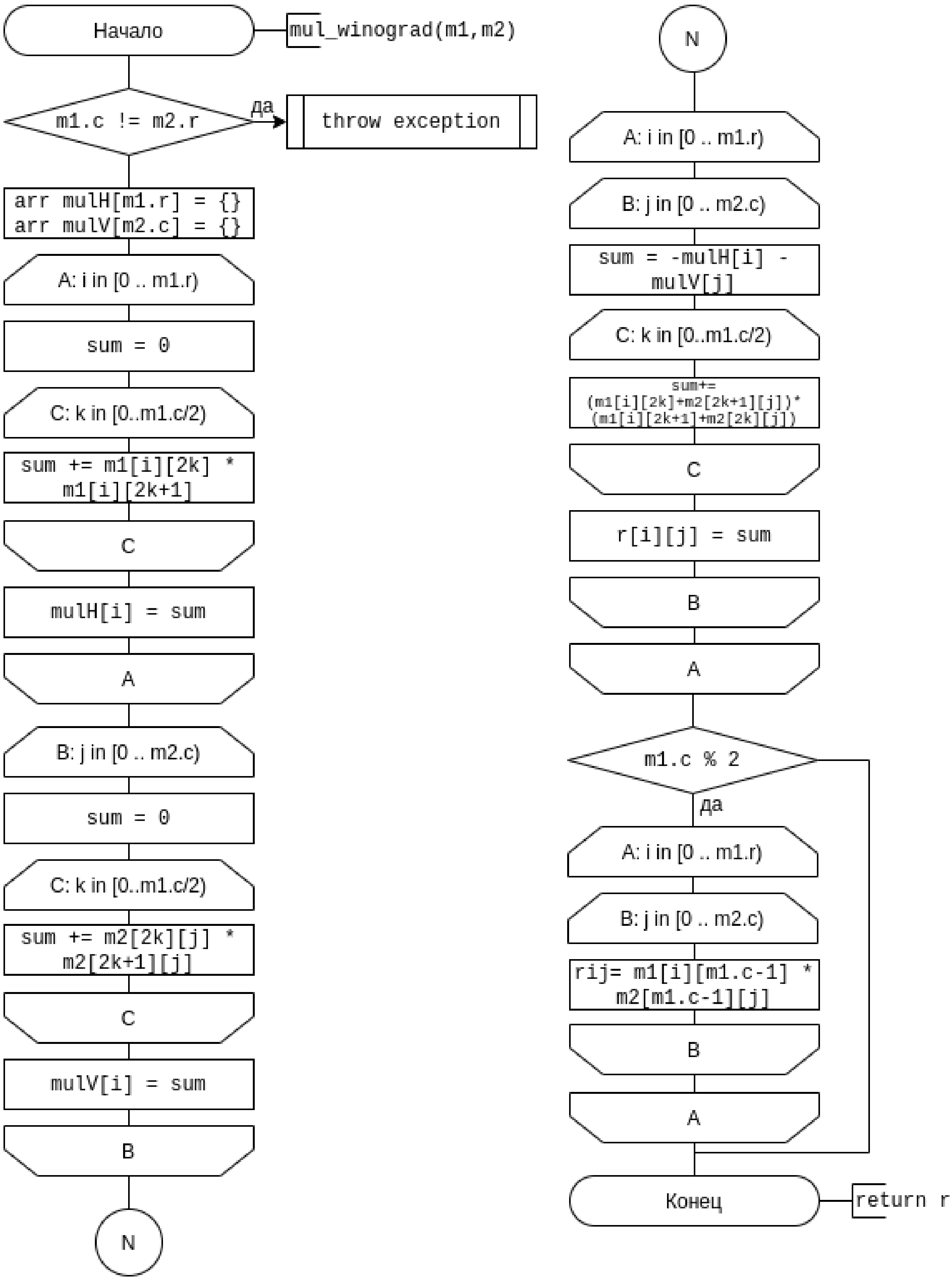

Рис. 2.2: Схема алгоритма Винограда

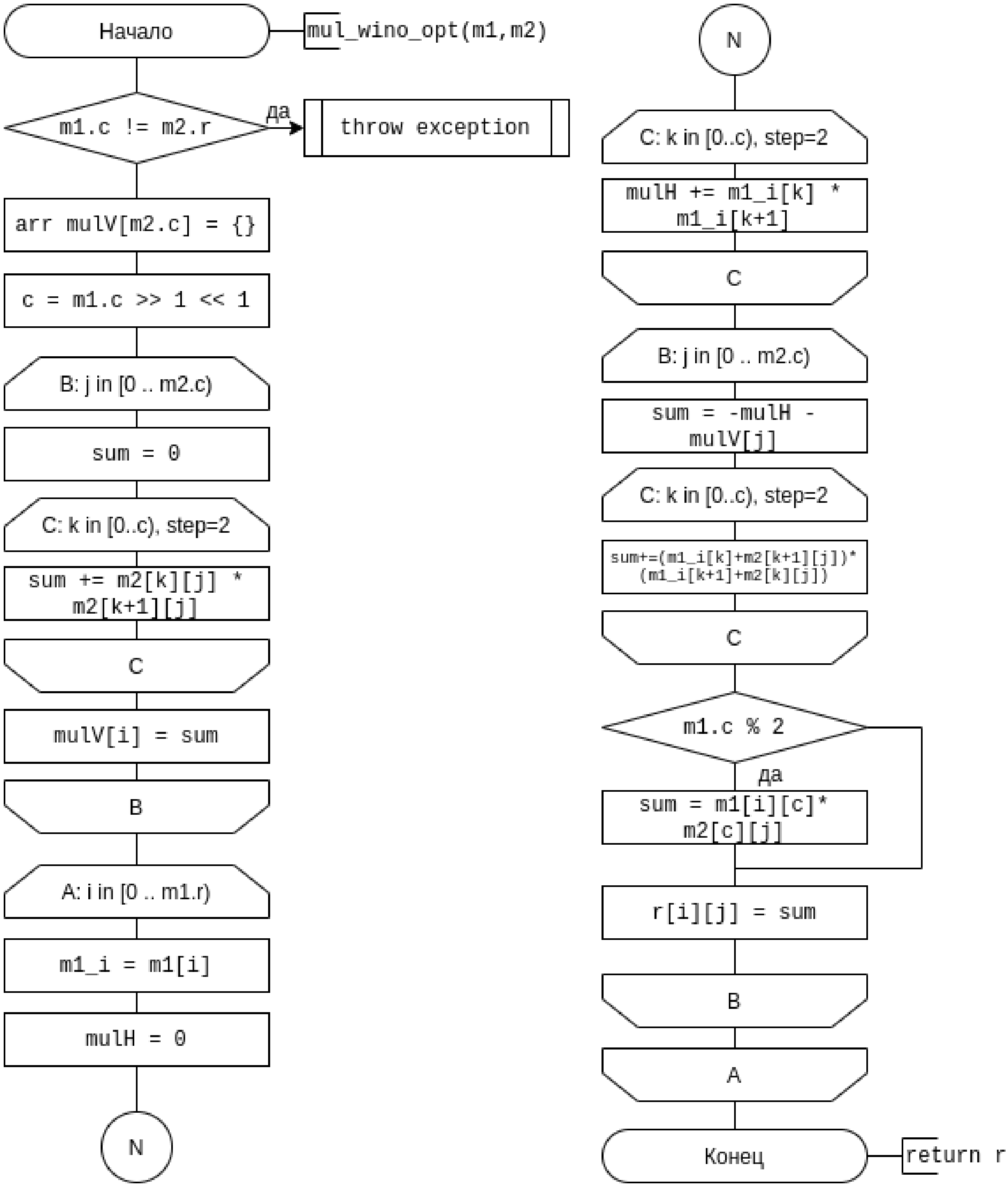

Рис. 2.3: Схема оптимизированного алгоритма Винограда

2.2 Оценка трудоемкости

Примечание: две матрицы можно перемножать $[l, m] * [m, n]$.

2.2.1 Стандартный алгоритм

F_{bodyC}	6
F_{forC}	$2 + m(6 + 2) = 8m + 2$
F_{bodyB}	$8m + 6$
F_{forB}	$2 + n(8m + 6 + 2) = 8mn + 8n + 2$
F_{bodyA}	$8mn + 8n + 2$
F_{forA}	$2 + l(8mn + 8n + 2 + 2) = 8lmn + 8ln + 4l + 2$
$F_{standard}$	$8lmn + 8ln + 4l + 3$

2.2.2 Алгоритм Винограда

F_{bodyC_1}	9
F_{forC_1}	$3 + m/2 \cdot (9 + 3) = 6m + 3$
F_{bodyA_1}	$6m + 6$
F_{forA_1}	$2 + l(6m + 6 + 2) = 6lm + 8l + 2$
F_{bodyC_2}	9
F_{forC_2}	$3 + m/2 \cdot (9 + 3) = 6m + 3$
F_{bodyB_2}	$6m + 6$
F_{forB_2}	$2 + n(6m + 6 + 2) = 6mn + 8n + 2$
F_{bodyC}	18
F_{forC}	$3 + m/2 \cdot (18 + 3) = 10.5m + 3$
F_{bodyB}	$10.5m + 11$
F_{forB}	$2 + n(10.5m + 11 + 2) = 10.5mn + 13n + 2$
F_{bodyA}	$10.5mn + 13n + 2$
F_{forA}	$2 + l(10.5mn + 13n + 2 + 2) = 10.5lmn + 13ln + 4l + 2$
F_{if}	$\begin{cases} 1, & \text{л.с.} \\ 1 + 2 + l(2 + n(2 + 10)), & \text{х.с.} \end{cases}$
$F_{winograd}$	$10.5lmn + 6lm + 6mn + \begin{cases} 13ln, & \text{л.с.} \\ 25ln, & \text{х.с.} \end{cases} + ...$

2.2.3 Оптимизированный алгоритм Винограда

F_{bodyC_1}	7
F_{forC_1}	$2 + m/2 \cdot (7 + 2) = 4.5m + 2$
F_{bodyB_1}	$4.5m + 5$
F_{forB_1}	$2 + n(4.5m + 5 + 2) = 4.5mn + 7l + 2$
F_{bodyC_2}	5
F_{forC_2}	$2 + m/2 \cdot (5 + 2) = 3.5m + 2$
F_{bodyC_3}	12
F_{forC_3}	$2 + m/2 \cdot (12 + 2) = 7m + 2$
F_{bodyB_3}	$\begin{cases} 7m + 10, & \text{л.с.} \\ 7m + 16, & \text{х.с.} \end{cases}$
F_{forB_3}	$\begin{cases} 2 + n(7m + 12), & \text{л.с.} \\ 2 + n(7m + 18), & \text{х.с.} \end{cases}$
F_{bodyA}	$\begin{cases} 7mn + 3.5m + 12n + 7, & \text{л.с.} \\ 7mn + 3.5m + 18n + 7, & \text{х.с.} \end{cases}$
F_{forA}	$\begin{cases} 2 + l(7mn + 3.5m + 12n + 7), & \text{л.с.} \\ 2 + l(7mn + 3.5m + 18n + 7), & \text{х.с.} \end{cases}$
F_{wino_opt}	$7lmn + 3.5lm + 4.5mn + \begin{cases} 12ln, & \text{л.с.} \\ 18ln, & \text{х.с.} \end{cases} + ...$

2.3 Вывод

В данном разделе было приведено описание схем алгоритмов и вычислены их трудоемкости. Трудоемкости алгоритмов соответственно 8, 10.5, 7 (lmn - куб).

3 Технологический раздел

3.1 Средства реализации

Язык программирования: C++

Библиотеки: google test, google benchmark

Редактор: VS Code

Я использую эти инструменты потому, что они мощные, широко используемые и хочу изучить фреймворк для тестирования и тестирования на C ++.

3.2 Листинг кода

Я создал шаблон для матричного типа, сами данные использовал статический двумерный массив. Его интерфейс прост в использовании, но код не очень понятен. Для краткости я перечисляю только шаблон Matrix.

Листинг 3.1: Шаблон для матричного типа

```
template <size_t R, size_t C, typename T = int>
class Matrix : public BaseMatrix
{
private:
    T data[R][C];
    // ...
}
```

Листинг 3.2: Стандартный алгоритм умножения матриц

```
    template <size_t R2, size_t C2>
    Matrix<R,C2,T> operator*(Matrix<R2,C2,T> &m2)
    {
        if (C != R2) throw std::exception();

        Matrix<R,C2,T> r;

        for (int i = 0; i < R; i++)
        {
            for (int j = 0; j < C2; j++)
            {
```

```
                T sum = 0;
                for (int k = 0; k < C; k++)
                    sum += data[i][k] * m2[k][j];

                r[i][j] = sum;
            }
        }

        return r;
    }
```

Листинг 3.3: Алгоритм Винограда для умножения матриц

```
template <size_t R2, size_t C2>
Matrix<R,C2,T> operator^(Matrix<R2,C2,T> &m2)
{
    if (C != R2) throw std::exception();

    Matrix<R,C2,T> r;

    T mulH[R] = {}, mulV[C2] = {};
    T sum;

    for (int i = 0; i < R; i++)
    {
        sum = 0;
        for (int k = 0; k < C/2; k++)
            sum += data[i][2*k] * data[i][2*k+1];
        mulH[i] = sum;
    }

    for (int j = 0; j < C2; j++)
    {
        sum = 0;
        for (int k = 0; k < C/2; k++)
            sum += m2[2*k][j] * m2[2*k+1][j];
        mulV[j] = sum;
    }

    for (int i = 0; i < R; i++)
    {
        for (int j = 0; j < C2; j++)
        {
            sum = -mulH[i] - mulV[j];
            for (int k = 0; k < C/2; k++)
                sum += (data[i][2*k] + m2[2*k+1][j])
                    * (data[i][2*k+1] + m2[2*k][j]);
            r[i][j] = sum;
        }
```

```
        }

        if (C % 2)
        {
            for (int i = 0; i < R; i++)
                for (int j = 0; j < C2; j++)
                    r[i][j] += data[i][C-1] * m2[C-1][j];
        }

        return r;
    }

    template <size_t R2, size_t C2>
```

Листинг 3.4: Алгоритм Винограда для умножения матриц с оптимизацией

```
    {
        if (C != R2) throw std::exception();

        Matrix<R,C2,T> r;

        T mulH, mulV[C2] = {};
        T sum;

        size_t C_ = C >> 1 << 1;


        for (int j = 0; j < C2; j++)
        {
            sum = 0;
            for (int k = 0;  k < C_; k += 2)
                sum += m2[k][j] * m2[k+1][j];
            mulV[j] = sum;
        }

        T* m1_i = data[0];

        for (int i = 0; i < R; i++, m1_i += C)
        {
            mulH = 0;
            for (int k = 0;  k < C_; k += 2)
                mulH += m1_i[k] * m1_i[k+1];

            for (int j = 0; j < C2; j++)
            {
                sum = -mulH - mulV[j];
                for (int k = 0; k < C_; k += 2)
                    sum += (m1_i[k] + m2[k+1][j])
                        * (m1_i[k+1] + m2[k][j]);
```

```

                if (C % 2)
                    sum += m1_i[C_] * m2[C_][j];
                r[i][j] = sum;
            }
        }

        return r;
    }
```

3.3 Описание тестирования

В таблице 3.1 приведен функциональные тесты для алгоритмов умножения матриц.

Матрица 1	**Матрица 2**	**Ожидаемый результат**
$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 2 & 3 \\ 4 & 4 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 2 & 4 & 3 \\ 1 & -3 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & -3 \\ 4 & 4 \\ 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 26 & 19 \\ -6 & -9 \end{pmatrix}$
$\begin{pmatrix} 2 & -3 \\ 4 & 4 \\ 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 2 & 4 & 3 \\ 1 & -3 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 17 & 0 \\ 12 & 4 & 20 \\ 7 & -1 & 12 \end{pmatrix}$
$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \end{pmatrix}$	Exception

Таблица 3.1: Функциональные тесты

3.4 Вывод

В этом разделе было рассмотрено код программы и описание тестирования.

4 Экспериментальный раздел

4.1 Примеры работы

На рисунке 4.1 приведен пример работы программы.

```
============ Program ============
A [2x3]
   0    0    0
   0    0    0

B [2x3]
   1    2    3
   3    4    5

std::exception - Can't multiply matrices A and B
==================
C [2x3]
   2    4    3
   1   -3    2

D [3x2]
   2   -3
   4    4
   2    3

Standard algorithm
CxD [2x2]
  26   19
  -6   -9

DxC [3x3]
   1   17    0
  12    4   20
   7   -1   12

Coppersmith—Winograd algorithm
CxD [2x2]
  26   19
  -6   -9

DxC [3x3]
   1   17    0
  12    4   20
   7   -1   12
```

Рис. 4.1: Примеры работы алгоритмов умножения матриц

4.2 Результаты тестирования

На рисунке 4.2 приведен результат теста с использованием фреймворка google test.

```
============ Testing ============
[==========] Running 7 tests from 3 test suites.
[----------] Global test environment set-up.
[----------] 2 tests from ZeroTest
[ RUN      ] ZeroTest.MulStandard
[       OK ] ZeroTest.MulStandard
[ RUN      ] ZeroTest.MulWinograd
[       OK ] ZeroTest.MulWinograd
[----------] 3 tests from NormalTest
[ RUN      ] NormalTest.MulStandard
[       OK ] NormalTest.MulStandard
[ RUN      ] NormalTest.MulWinograd
[       OK ] NormalTest.MulWinograd
[ RUN      ] NormalTest.MulWinogradOpt
[       OK ] NormalTest.MulWinogradOpt
[----------] 2 tests from ErrorTest
[ RUN      ] ErrorTest.MulStandard
[       OK ] ErrorTest.MulStandard
[ RUN      ] ErrorTest.MulWinograd
[       OK ] ErrorTest.MulWinograd
[----------] Global test environment tear-down
[==========] 7 tests from 3 test suites ran.
[  PASSED  ] 7 tests.
```

Рис. 4.2: Примеры работы алгоритмов умножения матриц

4.3 Сравнение времени работы

В таблице 4.1 приведены замеры времени работы алгоритмов умножения матриц на квадратных матрицах, на основе них построены графики 4.3 и 4.4.

Размер	**Стандартный**	**Винограда**	**Винограда(о)**
100	4.2321e+06	4.3321e+06	3.4289e+06
200	3.5857e+07	3.3469e+07	2.6979e+07
300	1.0255e+08	9.9688e+07	8.6865e+07
400	2.9630e+08	2.8231e+08	2.3073e+08
500	5.0361e+08	4.9825e+08	4.3241e+08
600	8.8321e+08	8.5834e+08	7.4968e+08
700	1.4755e+09	1.4011e+09	1.2223e+09
800	2.5314e+09	2.4905e+09	2.0274e+09
Размер	**Стандартный**	**Винограда**	**Винограда(о)**
101	4.0446e+06	4.1116e+06	3.2180e+06
201	3.2154e+07	3.2622e+07	2.5230e+07
301	1.0256e+08	1.0014e+08	8.9064e+07
401	2.5948e+08	2.5450e+08	2.2657e+08
501	5.1728e+08	5.0715e+08	4.3993e+08
601	9.1651e+08	8.9246e+08	7.7484e+08
701	1.4396e+09	1.4173e+09	1.2808e+09
801	2.1816e+09	2.1910e+09	1.8964e+09

Таблица 4.1: Времени работы (ns)

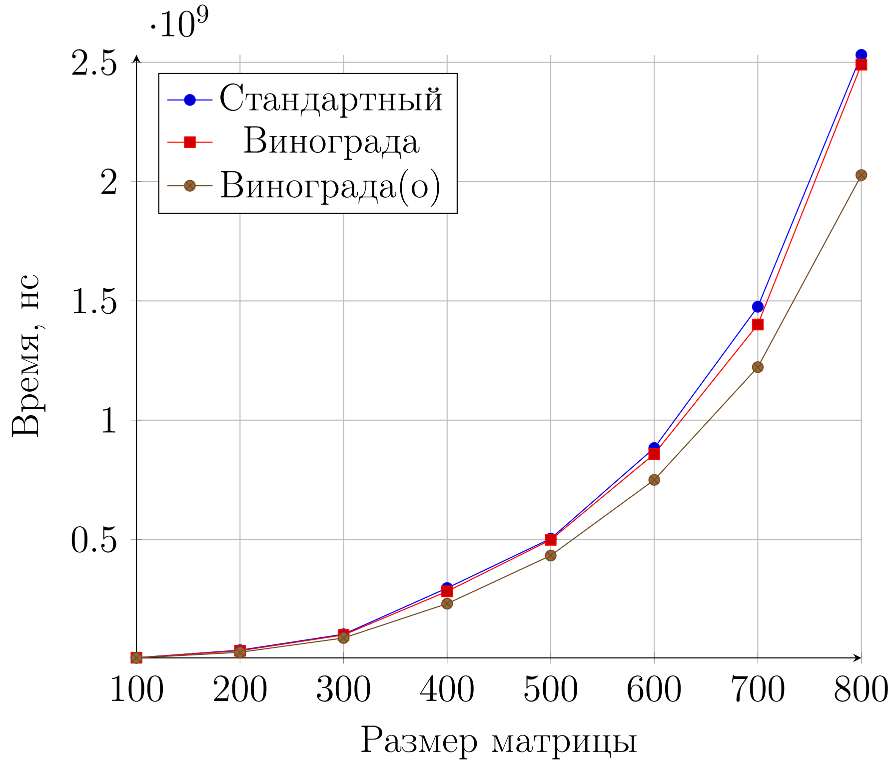

Рис. 4.3: Зависимость времени работы алгоритмов умножения матриц от размеры матрицы (при четном размере)

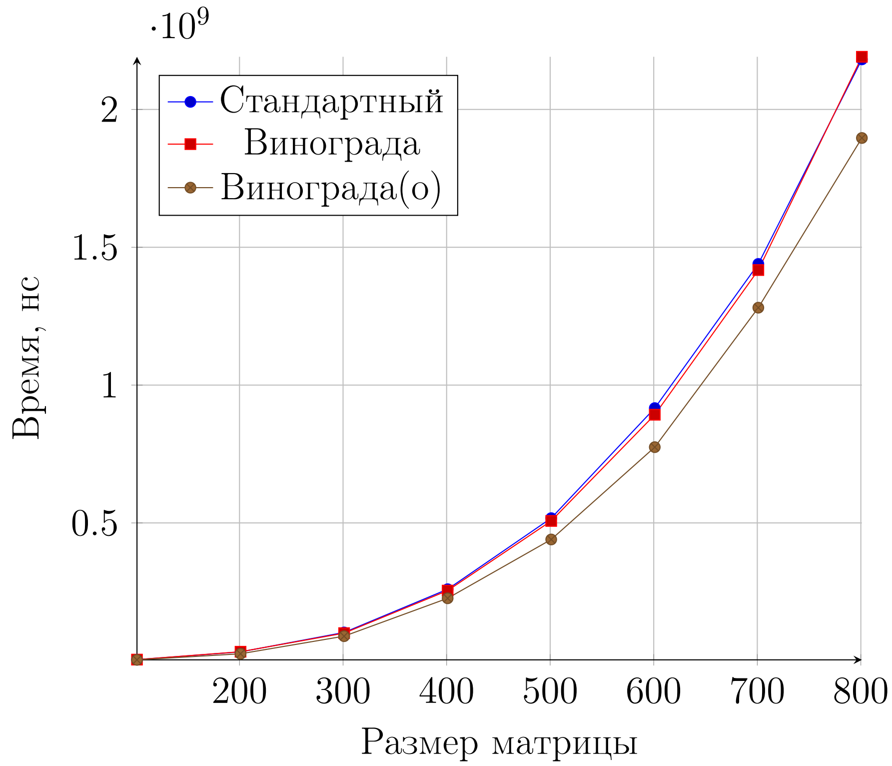

Рис. 4.4: Зависимость времени работы алгоритмов умножения матриц от размеры матрицы (при нечетном размере)

4.4 Вывод

Из графики, очевидно, что алгоритм Винограда с оптимизацией самый быстрый, на матрицах размером 800x800 работает примерно на 20% (15-25% зависит от m четное или нечетное) быстрее стандартный алгоритм.

Заключение

В ходе лабораторной работы было изучено алгоритмов умножения матриц: стандартный алгоритм и алгоритм Винограда. Было проведено рассчет сложности алгоритмов и сделаны следующие выводы:

- алгоритм Винограда быстрее стандартный алгоритм, но не сильно отличаются;
- алгоритм Винограда с оптимизацией самый быстрый, быстрее чем стандартный алгоритм в 20%

Литература

[1] Корн Г., Корн Т. Алгебра матриц и матричное исчисление // Справочник по математике. — 4-е издание. — М: Наука, 1978. — С. 392—394.

[2] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990.

[3] Google Testing Framework
`https://github.com/google/googletest`

[4] Google Benchmark
`https://github.com/google/benchmark`