

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №2

По курсу: "АНАЛИЗ АЛГОРИТМОВ"

## **Алгоритмы умножения матриц**

Работу выполнила: Оберган Татьяна, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

*Москва, 2019*

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Стандартный алгоритм умножения матриц . . . . .	4
1.2 Алгоритм Винограда . . . . .	5
1.3 Вывод . . . . .	5
<b>2 Конструкторская часть</b>	<b>6</b>
2.1 Схемы алгоритмов . . . . .	6
2.2 Трудоемкость алгоритмов . . . . .	10
2.2.1 Трудоемкость первичной проверки . . . . .	10
2.2.2 Классический алгоритм . . . . .	10
2.2.3 Алгоритм Винограда . . . . .	11
2.2.4 Оптимизированный алгоритм Винограда . . . . .	11
<b>3 Технологическая часть</b>	<b>12</b>
3.1 Выбор ЯП . . . . .	12
3.2 Описание структуры ПО . . . . .	12
3.3 Сведения о модулях программы . . . . .	13
3.4 Листинг кода алгоритмов . . . . .	13
3.4.1 Оптимизация алгоритма Винограда . . . . .	17
<b>4 Исследовательская часть</b>	<b>19</b>
4.1 Сравнительный анализ на основе замеров времени работы алгоритмов . . . . .	19
4.2 Тестирование программы . . . . .	20
4.3 Вывод . . . . .	21



# Введение

Цель работы: изучение алгоритмов умножения матриц. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда. Также требуется изучить расчет сложности алгоритмов, получить навыки в улучшении алгоритмов. Эти алгоритмы активно применяются во всех областях, применяющих линейную алгебру, таких как:

- компьютерная графика
- физика
- экономика

и так далее.

В ходе лабораторной работы предстоит:

- Изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда
- Оптимизировать алгоритм Винограда
- Дать теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда
- Реализовать три алгоритма умножения матриц на одном из языков программирования
- Сравнить алгоритмы умножения матриц

# 1 | Аналитическая часть

Матрица - математический объект, эквивалентный двумерному массиву. Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

## 1.1 Стандартный алгоритм умножения матриц

Пусть даны две прямоугольные матрицы  $A[m * n]$  и  $B[n * k]$ :

$$\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix}$$

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,k} \\ \dots & \dots & \dots \\ b_{n,1} & \dots & b_{n,k} \end{bmatrix}$$

В результате получим матрицу  $C[m * k]$ :

$$\begin{bmatrix} c_{1,1} & \dots & c_{1,k} \\ \dots & \dots & \dots \\ c_{m,1} & \dots & c_{m,k} \end{bmatrix}$$

$c_{i,j} = \sum_{r=1}^n a_{i,r} \cdot b_{r,j}$  называется произведением матриц  $A$  и  $B$ .

## 1.2 Алгоритм Винограда

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:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4$$

Это равенство можно переписать в виде:

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

Менее очевидно, что 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. Это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

## 1.3 Вывод

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

## 2 | Конструкторская часть

**Требования к вводу:** На вход подаются две матрицы

**Требования к программе:**

- Корректное умножение двух матриц
- При матрицах неправильных размеров программа не должна аварийно завершаться

### 2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов.

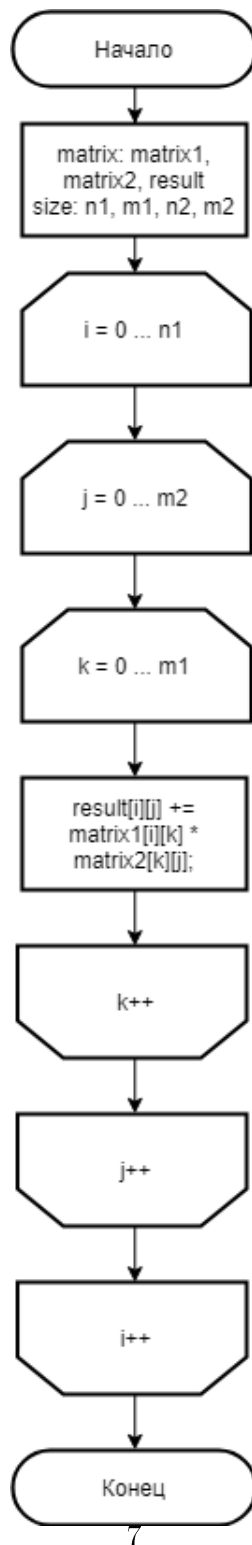


Рис. 2.1: Схема классического алгоритма умножения матриц



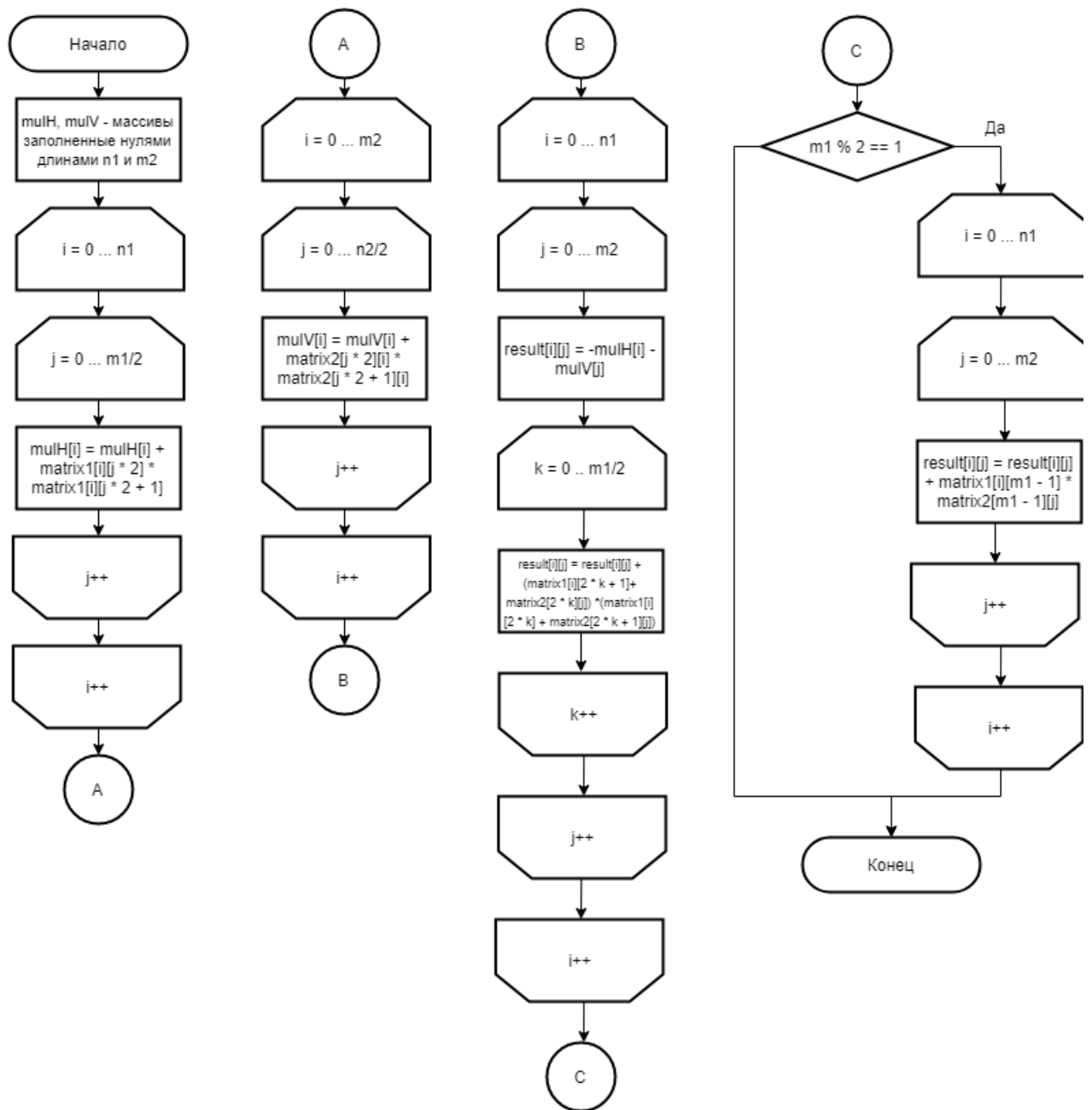


Рис. 2.2: Схема алгоритма Винограда

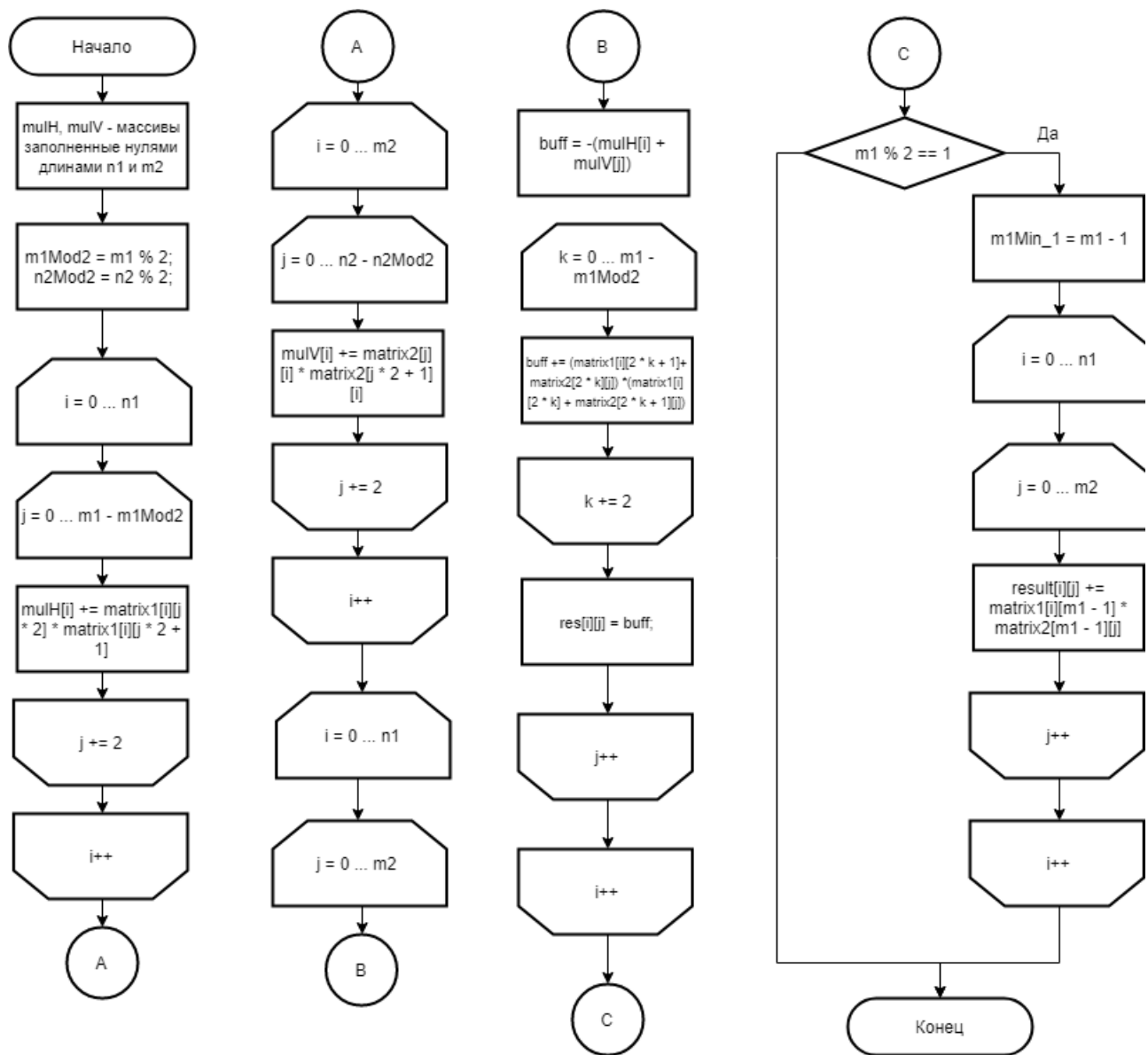


Рис. 2.3: Схема оптимизированного алгоритма Винограда

## 2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, \*, /, =, ==, <=, >=, !=, +=, [], получение полей класса
- оценка трудоемкости цикла:  $F_{\text{ц}} = \text{init} + N * (\text{a} + F_{\text{тела}} + \text{post}) + \text{a}$ , где a - условие цикла, init - предусловие цикла, post - постусловие цикла
- стоимость условного перехода применим за 0, стоимость вычисления условия остаётся

Оценим трудоемкость алгоритмов по коду программы.

### 2.2.1 Трудоемкость первичной проверки

Рассмотрим трудоемкость первичной проверки на возможность умножения матриц.

Табл. 2.1 Построчная оценка веса

Код	Вес
int n1 = mart1.Length;	2
int n2 = matr2.Length;	2
if (n1 == 0    n2 == 0) return null;	3
int m1 = mart1[0].Length;	3
int m2 = matr2[0].Length;	3
if (m1 != n2) return null;	1
Итого	14

### 2.2.2 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма:

Инициализация матрицы результата:  $1 + 1 + n1 * (1 + 2 + 1) + 1 = 4 * n1 + 3$

Подсчет:

$$1 + M * (1 + (1 + Q * (1 + (1 + N * (1 + (8) + 1) + 1) + 1) + 1) + 1) + 1 = \\ M * (Q * (N * 10 + 4) + 4) + 2 = 10NQM + 4 * QM + 4M + 2$$

### 2.2.3 Алгоритм Винограда

Аналогично рассмотрим трудоемкость алгоритма Винограда.

Первый цикл:  $15/2 * MN + 5 * M + 2$

Второй цикл:  $15/2 * MN + 5 * M + 2$

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

Условный переход:  $\begin{bmatrix} 2 & , \text{ в случае невыполнения условия} \\ 15 * QM + 4 * M + 2 & , \text{ в случае выполнения условия} \end{bmatrix}$

Итого:  $13MNQ + 15MN + 12MQ + 14M + 6 + \begin{bmatrix} 2 & , \text{ в случае невыполнения условия} \\ 15QM + 4M + 2 & , \text{ в случае выполнения условия} \end{bmatrix}$

### 2.2.4 Оптимизированный алгоритм Винограда

Аналогично Рассмотрим трудоемкость алгоритма Винограда:

Первый цикл:  $11/2 * MN + 4 * M + 2$

Второй цикл:  $11/2 * MN + 4 * M + 2$

Третий цикл:  $17/2 * MNQ + 9 * MQ + 4 * M + 2$

Условный переход:  $\begin{bmatrix} 1 & , \text{ в случае невыполнения условия} \\ 10 * QM + 4 * M + 2 & , \text{ в случае выполнения условия} \end{bmatrix}$

Итого:  $17/2 * MNQ + 11MN + 9MQ + 12M + 6 + \begin{bmatrix} 1 & , \text{ в случае невыполнения условия} \\ 10 * QM + 4 * M + 2 & , \text{ в случае выполнения условия} \end{bmatrix}$

## 3 | Технологическая часть

### 3.1 Выбор ЯП

В качестве языка программирования был выбран C#, а средой разработки Visual Studio. Время работы алгоритмов было измерено с помощью класса Stopwatch.

### 3.2 Описание структуры ПО

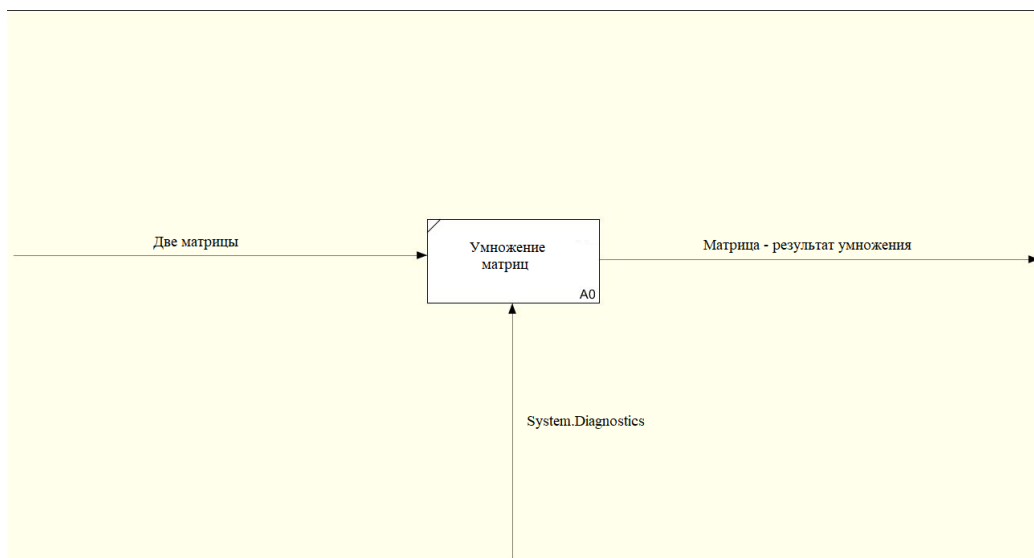


Рис. 3.1: Функциональная схема умножения матриц (IDEF0 диаграмма 1 уровня)

### 3.3 Сведения о модулях программы

Программа состоит из:

- Program.cs - главный файл программы, в котором располагается точка входа в программу и функция замера времени.
- Mult.cs - файл класса Mult, в котором находятся алгоритмы умножения матриц
- TestMult.cs - файл с юнит тестами

### 3.4 Листинг кода алгоритмов

Листинг 3.1: Стандартный алгоритм умножения матриц

```
1 public static int [][] MultStand(int [][] mart1, int [][]  
   matr2)  
2     {  
3         int n1 = mart1.Length;  
4         int n2 = matr2.Length;  
5  
6         if (n1 == 0 || n2 == 0)  
7             return null;  
8  
9         int m1 = mart1[0].Length;  
10        int m2 = matr2[0].Length;  
11  
12        if (m1 != n2)  
13            return null;  
14  
15        int [][] res = new int[n1][];  
16        for (int i = 0; i < n1; i++)  
17            res[i] = new int[m2];  
18  
19        for (int i = 0; i < n1; i++)  
20            for (int j = 0; j < m2; j++)  
21                for (int k = 0; k < m1; k++)  
22                    res[i][j] += mart1[i][k] * matr2[k]  
                        ][j];
```

```

23
24         return res;
25     }

```

Листинг 3.2: Алгоритм Винограда

```

1 public static int [][] MultVin(int [][] matr1, int [][] matr2)
2 {
3     int n1 = matr1.Length;
4     int n2 = matr2.Length;
5
6     if (n1 == 0 || n2 == 0)
7         return null;
8
9     int m1 = matr1[0].Length;
10    int m2 = matr2[0].Length;
11
12    if (m1 != n2)
13        return null;
14
15    int [] mulH = new int[n1];
16    int [] mulV = new int[m2];
17
18    int [][] res = new int[n1][];
19    for (int i = 0; i < n1; i++)
20        res[i] = new int[m2];
21
22    for (int i = 0; i < n1; i++)
23    {
24        for (int j = 0; j < m1 / 2; j++)
25        {
26            mulH[i] = mulH[i] + matr1[i][j * 2] *
27                matr1[i][j * 2 + 1];
28        }
29    }
30
31    for (int i = 0; i < m2; i++)
32    {
33        for (int j = 0; j < n2 / 2; j++)
34        {
35            mulV[i] = mulV[i] + matr2[j * 2][i] *

```

```

35         matr2[j * 2 + 1][i];
36     }
37
38     for (int i = 0; i < n1; i++)
39     {
40         for (int j = 0; j < m2; j++)
41         {
42             res[i][j] = -mulH[i] - mulV[j];
43             for (int k = 0; k < m1 / 2; k++)
44             {
45                 res[i][j] = res[i][j] + (matr1[i][2
                     * k + 1] + matr2[2 * k][j]) * (
                     matr1[i][2 * k] + matr2[2 * k +
                     1][j]);
46             }
47         }
48     }
49
50     if (m1 \% 2 == 1)
51     {
52         for (int i = 0; i < n1; i++)
53         {
54             for (int j = 0; j < m2; j++)
55             {
56                 res[i][j] = res[i][j] + matr1[i][m1
                     - 1] * matr2[m1 - 1][j];
57             }
58         }
59     }
60
61     return res;
62 }

```

Листинг 3.3: Оптимизированный алгоритм Винограда

```

1 public static int [][] MultVinOpt(int [][] matr1, int [][]
   matr2)
2 {
3     int n1 = matr1.Length;
4     int n2 = matr2.Length;

```



```

5
6     if (n1 == 0 || n2 == 0)
7         return null;
8
9     int m1 = matr1[0].Length;
10    int m2 = matr2[0].Length;
11
12    if (m1 != n2)
13        return null;
14
15    int[] mulH = new int[n1];
16    int[] mulV = new int[m2];
17
18    int[][] res = new int[n1][];
19    for (int i = 0; i < n1; i++)
20        res[i] = new int[m2];
21
22    int m1Mod2 = m1 \% 2;
23    int n2Mod2 = n2 \% 2;
24
25    for (int i = 0; i < n1; i++)
26    {
27        for (int j = 0; j < (m1 - m1Mod2); j += 2)
28        {
29            mulH[i] += matr1[i][j] * matr1[i][j +
30                1];
31        }
32    }
33
34    for (int i = 0; i < m2; i++)
35    {
36        for (int j = 0; j < (n2 - n2Mod2); j += 2)
37        {
38            mulV[i] += matr2[j][i] * matr2[j + 1][i
39                ];
40        }
41    }
42
43    for (int i = 0; i < n1; i++)
44    {

```

```

43         for (int j = 0; j < m2; j++)
44         {
45             int buff = -(mulH[i] + mulV[j]);
46             for (int k = 0; k < (m1 - m1Mod2); k +=
47                 2)
48             {
49                 buff += (matr1[i][k + 1] + matr2[k
50                     ][j]) * (matr1[i][k] + matr2[k +
51                         1][j]);
52             }
53             res[i][j] = buff;
54         }
55     }
56     if (m1Mod2 == 1)
57     {
58         int m1Min_1 = m1 - 1;
59         for (int i = 0; i < n1; i++)
60         {
61             for (int j = 0; j < m2; j++)
62             {
63                 res[i][j] += matr1[i][m1Min_1] *
64                     matr2[m1Min_1][j];
65             }
66         }
67     }
68     return res;
69 }

```

### 3.4.1 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 3 оптимизации:

1. Избавление от деления в условии цикла;
2. Замена  $mulH[i] = mulH[i] + \dots$  на  $mulH[i] += \dots$  (аналогично для  $mulV[i]$ );

Листинг 3.4: Оптимизации алгоритма Винограда №1 и №2

```

1  int m1Mod2 = m1 \% 2;
2  int n2Mod2 = n2 \% 2;
3
4  for (int i = 0; i < n1; i++)
5  {
6      for (int j = 0; j < (m1 - m1Mod2); j += 2)
7      {
8          mulH[i] += matrix1[i][j] * matrix1[i][j + 1];
9      }
10 }
11
12 for (int i = 0; i < m2; i++)
13 {
14     for (int j = 0; j < (n2 - n2Mod2); j += 2)
15     {
16         mulV[i] += matrix2[j][i] * matrix2[j + 1][i];
17     }
18 }

```

3. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти. Сброс буфера в ячейку матрицы после цикла.

Листинг 3.5: Оптимизации алгоритма Винограда №3

```

1  for (int i = 0; i < n1; i++)
2  {
3      for (int j = 0; j < m2; j++)
4      {
5          int buff = -(mulH[i] + mulV[j]);
6          for (int k = 0; k < (m1 - m1Mod2); k += 2)
7          {
8              buff += (matrix1[i][k + 1] + matrix2[k][j]) * (
9                  matrix1[i][k] + matrix2[k + 1][j]);
10         }
11         result[i][j] = buff;
12     }
13 }

```

## 4 | Исследовательская часть

### 4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов.

Первый эксперимент производится для лучшего случая на матрицах размером от 100 x 100 до 1000 x 1000 с шагом 100. Сравним результаты для разных алгоритмов:

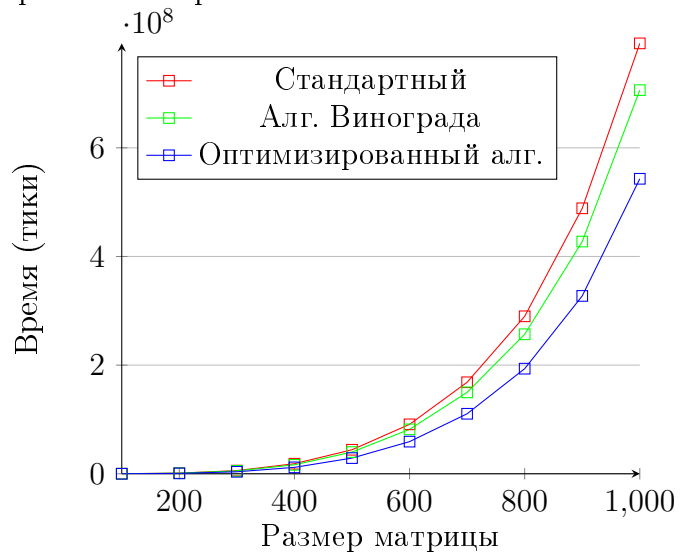


Рис. 4.1: Сравнение времени работы алгоритмов при четном размере матрицы

Второй эксперимент производится для худшего случая, когда поданы матрицы с нечетными размерами от 101 x 101 до 1001 x 1001 с шагом 100. Сравним результаты для разных алгоритмов:

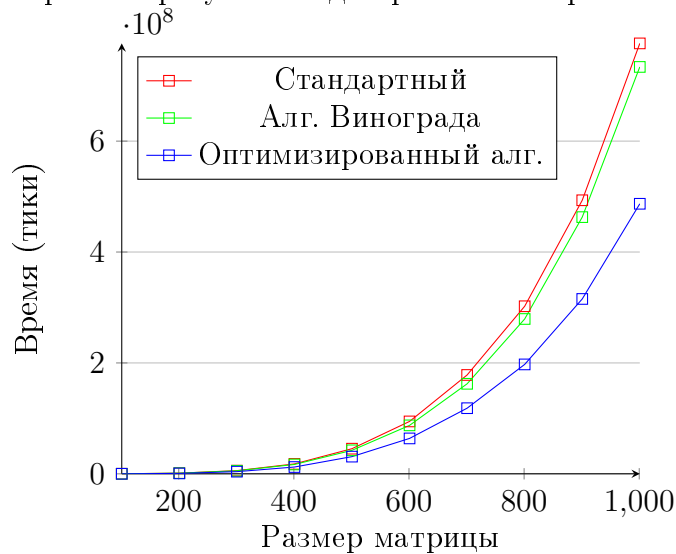


Рис. 4.2: Сравнение времени работы алгоритмов при нечетном размере матрицы

## 4.2 Тестирование программы

Было произведено тестирование реализованных алгоритмов с помощью библиотеки Microsoft.VisualStudio.TestTools.UnitTesting.

Всего было реализованно 7 тестовых случаев:

- Некорректный размер матриц. Алгоритм должен возвращать Null
- Размер матриц равен 1
- Размер матриц равен 2
- Сравнение работы стандартной реализации с Виноградом на случайных значениях

Четный размер

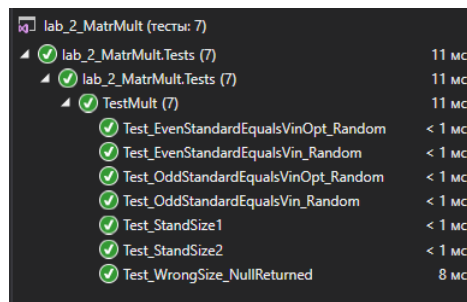
Нечетный размер

- Сравнение работы стандартной реализации с оптимизированным Виноградом на случайных значениях

Четный размер

Нечетный размер

Далее будут предоставлены результаты тестирования программы:



lab_2_MatrMult (тесты: 7)	
lab_2_MatrMult.Tests (7)	11 мс
lab_2_MatrMult.Tests (7)	11 мс
TestMult (7)	11 мс
Test_EvenStandardEqualsVinOpt_Random	< 1 мс
Test_EvenStandardEqualsVin_Random	< 1 мс
Test_OddStandardEqualsVinOpt_Random	< 1 мс
Test_OddStandardEqualsVin_Random	< 1 мс
Test_StandSize1	< 1 мс
Test_StandSize2	< 1 мс
Test_WrongSize_NullReturned	8 мс

Рис. 4.1: Результаты работы тестов

## 4.3 Вывод

По результатам тестирования все рассматриваемые алгоритмы реализованы правильно. Самым медленным алгоритмом оказался алгоритм классического умножения матриц, а самым быстрым — оптимизированный алгоритм Винограда.

## Заключение

В ходе лабораторной работы я изучила алгоритмы умножения матриц: стандартный и Винограда, оптимизировала алгоритм Винограда, дала теоретическую оценку алгоритмов стандартного умножения матриц, Винограда и улучшенного Винограда, реализовала три алгоритма умножения матриц на языке программирования C# и сравнила эти алгоритмы.