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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

ОТЧЕТ

по лабораторной работе № 2

Название: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-51Б _____ М.С.Матвиенко
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2020

Содержание

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 рассчет сложности алгоритмов, получить навыки в улучшении алгоритмов.
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

Матрицей A размера $[m * n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы. 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.

Пусть даны две прямоугольные матрицы A и B размеров $[m*n]$ и $[n*k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

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

1.1 Описание алгоритмов

1.1.1 Стандартный алгоритм

По определению каждый элемент матрицы C вычисляется следующим образом:
$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$, это значит, что для вычисления всей матрицы необходимо выполнить $m * k$ вычислений, где $(i = 1, ..., k; j = 1, ..., m)$

1.1.2 Алгоритм Винограда

Подход Алгоритма Винограда является иллюстрацией общей методологии, начатой в 1979-х годах на основе билинейных и трилинейных форм, благодаря которым большинство усовершенствований для умножения матриц были получены.

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Их скалярное произведение равно (1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \quad (1)$$

Равенство (1) можно переписать в виде (2)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \quad (2)$$

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.

1.2 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:
1) На вход подаются две матрицы;
Требования к программе::
1) Корректное умножение двух матриц;
2) При матрицах неправилыных размеров программа не должна аварийно завершаться;

2.1 Разработка алгоритмов

В данном разделе представлены схемы реализуемых алгоритмов.

2.1.1 Стандартное умножения матриц

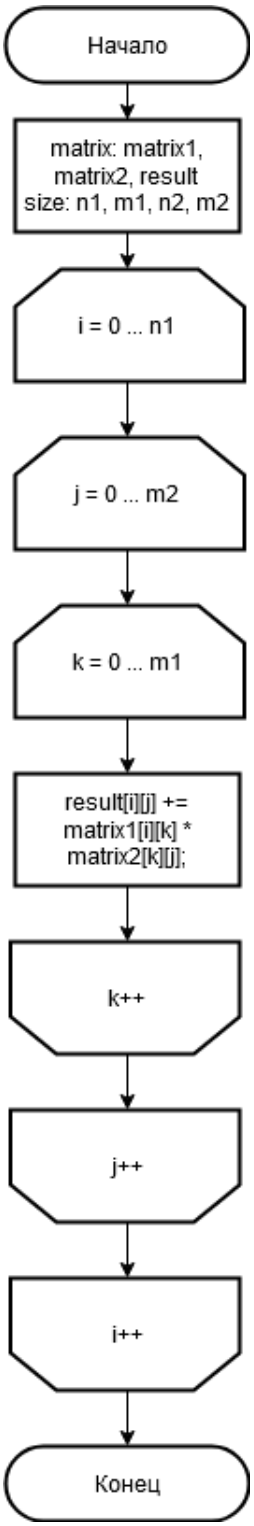

Рис. 1: Схема классического алгоритма умножения матриц

2.1.2 Алгоритм Винограда

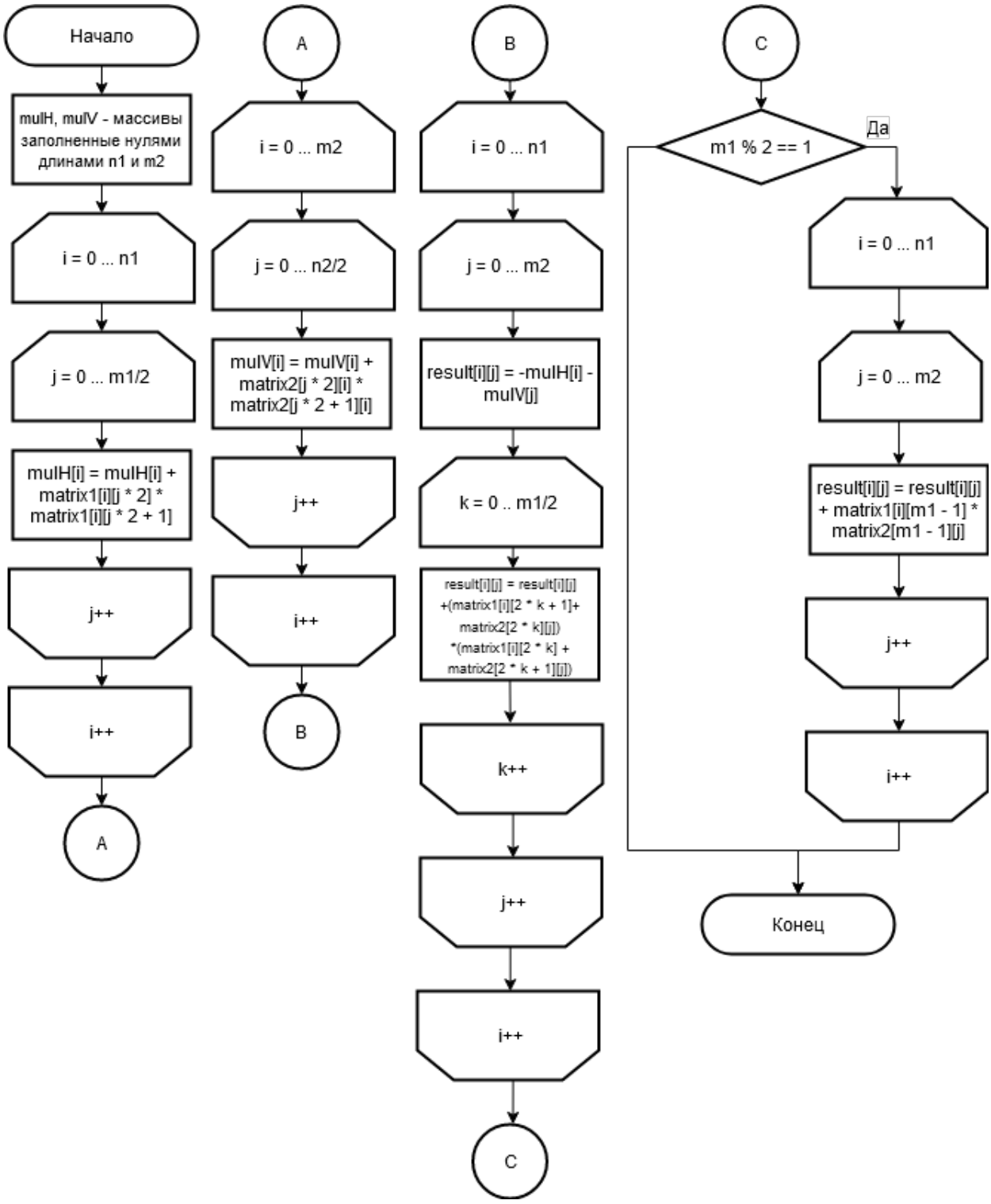

Рис. 2: Схема алгоритма Винограда

2.1.3 Оптимизированный алгоритм Винограда

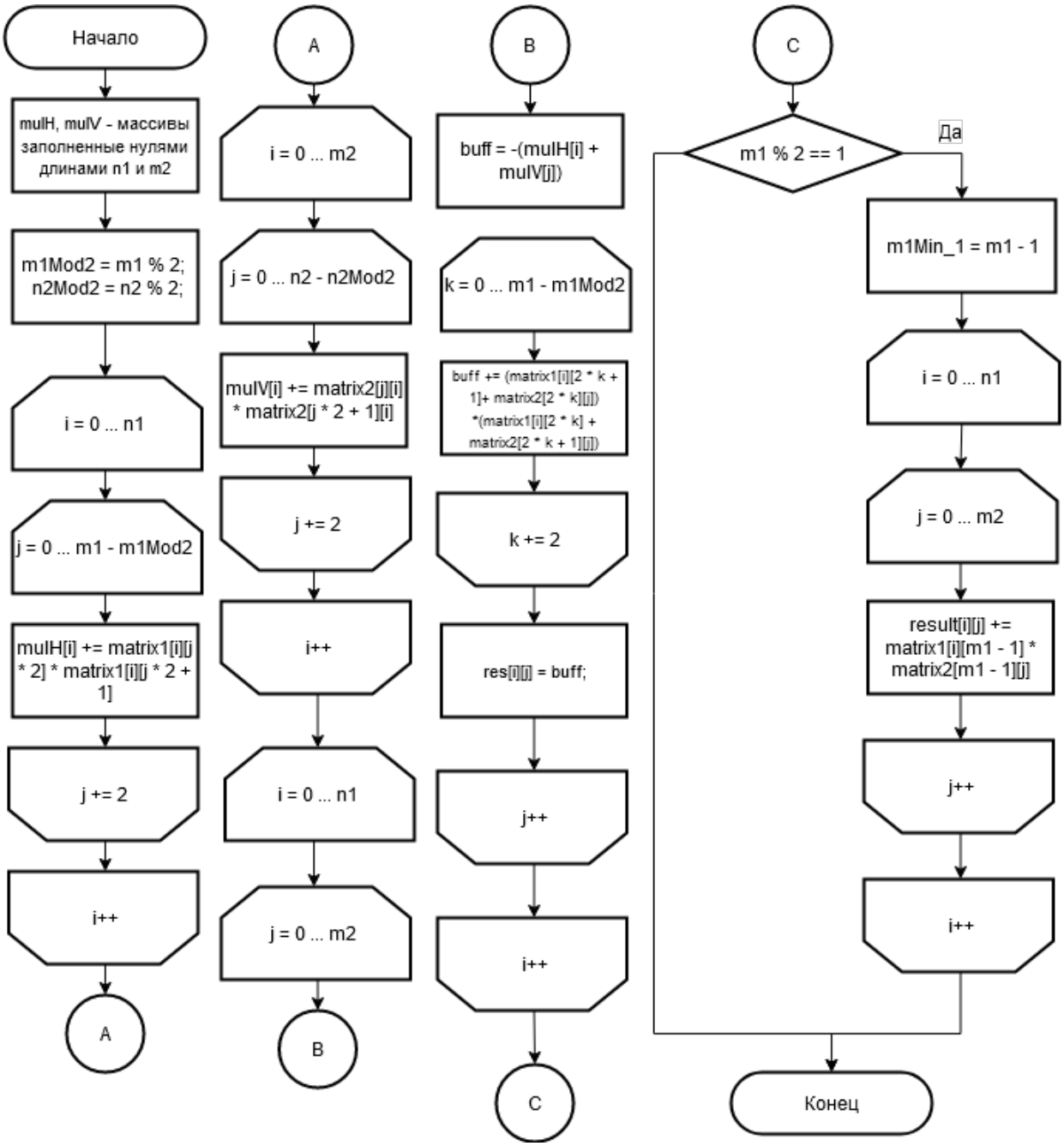

Рис. 3: Схема оптимизированного алгоритма Винограда

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса
- оценка трудоемкости цикла: Fц = init + N*(a + Fтела + post), 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. 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: $2 + 2 + n_1(1 + 3 + 1) = 5n_1 + 4$

Подсчет:

$1+n_1(1+(1+m_2(1+(1+m_1(1+(8)+1)+1)+1)+1)+1)+1 = n_1(m_2(10m_1+4)+4)+4)+2 = 10n_1m_2m_1 + 4n_1m_2 + 4n_1 + 2$

2.2.3 Алгоритм Винограда

Аналогично рассмотрим трудоемкость алгоритма Винограда.

Первый цикл: $\frac{15}{2}n_1m_1 + 5n_1 + 2$

Второй цикл: $\frac{15}{2}m_2n_2 + 5m_2 + 2$

Третий цикл: $13n_1m_2m_1 + 12n_1m_2 + 4n_1 + 2$

Условный переход: $\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

Итого: $13n_1m_2m_1 + \frac{15}{2}n_1m_1 + \frac{15}{2}m_2n_2 + 12n_1m_2 + 5n_1 + 5m_2 + 4n_1 + 6 + \begin{bmatrix} 2 & \text{, невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

2.2.4 Оптимизированный алгоритм Винограда

Аналогично Рассмотрим трудоемкость оптимизированого алгоритма Винограда:

Первый цикл: $\frac{11}{2}n_1m_1 + 4n_1 + 2$
Второй цикл: $\frac{11}{2}m_2n_2 + 4m_2 + 2$
Третий цикл: $\frac{17}{2}n_1m_2m_1 + 9n_1m_2 + 4n_1 + 2$
Условный переход: $\begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

Итого: $\frac{17}{2}n_1m_2m_1 + \frac{11}{2}n_1m_1 + \frac{11}{2}m_2n_2 + 9n_1m_2 + 8n_1 + 4m_2 + 6 + \begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель оценки трудоемкости алгоритма, были расчитаны трудоемкости алгоритмов в соответсвии с этой моделью.

3 Технологическая часть

В данном разделе будет описана технологическая часть лабораторной работы: листинг кода, сравнительный анализ всех алгоритмов.
Среда выполнения: Windows 10 x64

3.1 Средства реализации

Для выполнения данной лабораторной работы использовался ЯП C# и IDE VisualStudio
Время работы алгоритмов было замерено с помощью класса Stopwatch.

3.2 Листинг кода

В данном разделе будет представлен листинг кода разработанных алгоритмов.

3.2.1 Стандартный алгоритм умножения матриц

Листинг 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         }
```

3.2.2 Алгоритм Винограда

Листинг 2: Алгоритм Винограда

```
public static int[][] MultVin(int[][] matr1, int[][] matr2)
        {
            int n1 = matr1.Length;
            int n2 = matr2.Length;

            if (n1 == 0 || n2 == 0)
                return null;

            int m1 = matr1[0].Length;
            int m2 = matr2[0].Length;

            if (m1 != n2)
                return null;

            int[] mulH = new int[n1];
            int[] mulV = new int[m2];

            int[][] res = new int[n1][];
            for (int i = 0; i < n1; i++)
                res[i] = new int[m2];

            for (int i = 0; i < n1; i++)
            {
                for (int j = 0; j < m1 / 2; j++)
                {
                    mulH[i] = mulH[i] + \
                    matr1[i][j * 2] * matr1[i][j * 2 + 1];
                }
            }

            for (int i = 0; i < m2; i++)
            {
                for (int j = 0; j < n2 / 2; j++)
                {
                    mulV[i] = mulV[i] + \
                    matr2[j * 2][i] * matr2[j * 2 + 1][i];
                }
            }

            for (int i = 0; i < n1; i++)
            {
                for (int j = 0; j < m2; j++)
                {
                    res[i][j] = -mulH[i] - mulV[j];
                    for (int k = 0; k < m1 / 2; k++)
                    {
                        res[i][j] = res[i][j] + \
```

```
48                          (matr1[i][2 * k + 1] + \
49                           matr2[2 * k][j]) * \
50                          (matr1[i][2 * k] + \
51                           matr2[2 * k + 1][j]);
52                      }
53                  }
54              }
55
56              if (m1 \% 2 == 1)
57              {
58                  for (int i = 0; i < n1; i++)
59                  {
60                      for (int j = 0; j < m2; j++)
61                      {
62                          res[i][j] = res[i][j] + \
63                          matr1[i][m1 - 1] * matr2[m1 - 1][j];
64                      }
65                  }
66              }
67
68              return res;
69          }
```

3.2.3 Оптимизированный алгоритм Винограда

Листинг 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 + 1];
30                  }
31              }
32
33              for (int i = 0; i < m2; i++)
34              {
35                  for (int j = 0; j < (n2 - n2Mod2); j += 2)
36                  {
37                      mulV[i] += matr2[j][i] * matr2[j + 1][i];
38                  }
39              }
40
41              for (int i = 0; i < n1; i++)
42              {
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 += 2)
47                      {
48                          buff += (matr1[i][k + 1] + matr2[k][j]) * \
49                                  (matr1[i][k] + matr2[k + 1][j]);
50                      }
51                      res[i][j] = buff;
52                  }
53              }
54
55              if (m1Mod2 == 1)
56              {
57                  int m1Min_1 = m1 - 1;
58                  for (int i = 0; i < n1; i++)
59                  {
60                      for (int j = 0; j < m2; j++)
61                      {
62                          res[i][j] += matr1[i][m1Min_1] * \
63                                       matr2[m1Min_1][j];
64                      }
65                  }
66              }
67
68              return res;
69          }
```

3.2.4 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 3 оптимизации:

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

Листинг 4: Оптимизация алгоритма Винограда №1 и №2

```
int m1Mod2 = m1 % 2;
int n2Mod2 = n2 % 2;

for (int i = 0; i < n1; i++)
{
        for (int j = 0; j < (m1 - m1Mod2); j += 2)
        {
                mulH[i] += mtrx1[i][j] * mtrx1[i][j + 1];
        }
}

for (int i = 0; i < m2; i++)
{
        for (int j = 0; j < (n2 - n2Mod2); j += 2)
        {
                mulV[i] += mtrx2[j][i] * mtrx2[j + 1][i];
        }
}
```

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 5: Оптимизации алгоритма Винограда №3

```
for (int i = 0; i < n1; i++)
{
        for (int j = 0; j < m2; j++)
        {
                int buff = -(mulH[i] + mulV[j]);
                for (int k = 0; k < (m1 - m1Mod2); k += 2)
                {
                        buff += (mtrx1[i][k + 1] + mtrx2[k][j])*\
                        (mtrx1[i][k] + mtrx2[k + 1][j]);
                }
                result[i][j] = buff;
        }
}
```

3.3 Вывод

В данном разделе были рассмотрена структура ПО и листинги кода программы.

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 квадратных матрицах размером от 100 x 100 до 1000 x 1000 с шагом (100; 100).
Сравним результаты для разных алгоритмов:

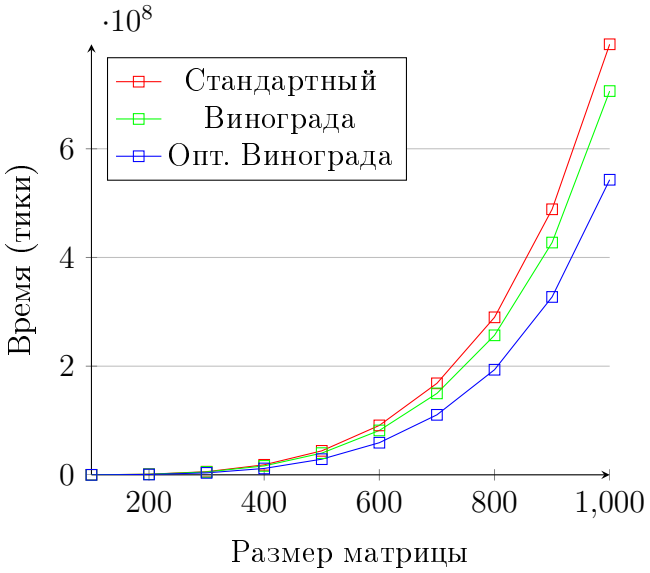

Рис. 4: Сравнение времени работы алгоритмов при четном размере матрицы

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

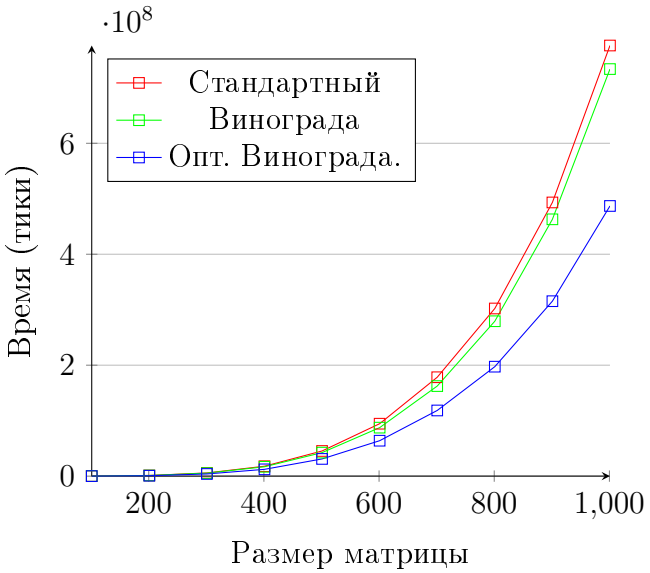

Рис. 5: Сравнение времени работы алгоритмов при нечетном размере матрицы

4.2 Тестирование программы

Было произведено тестирование реализованных алгоритмов с помощью библиотеки Microsoft.Vis
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

На рисунке 4 будут предоставлены результаты тестирования программы. Откуда видно, что тестирование пройдено, алгоритмы реализованы правильно.

Рис. 4: 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 изучил алгоритмы умножения матриц: стандартный и Винограда, оптимизировал алгоритм Винограда, дал теоретическую оценку алгоритмов стандартного умножения матриц, Винограда и улучшенного Винограда, реализовал три алгоритма умножения матриц на языке программирования C# и сравнил эти алгоритмы.