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»

Отчёт

по лабораторной работе №2

Название: Алгоритм Винограда

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) И. Е. Афимин (И.О. Фамилия)

Преподаватель _____ (Подпись, дата) Л.Л. Волкова (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.

Задачами данной лабораторной являются:

1) изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;

2) оптимизировать алгоритм Винограда;

3) дать теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда;

4) реализовать три алгоритма умножения матриц на одном из языков программирования;

5) сравнить алгоритмы умножения матриц.

1 Аналитическая часть

В данном разделе рассмотрим понятие произведения матриц, классическое произведение и произведение матриц при помощи алгоритма Винограда

1.1 Умножение матриц

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 [3].

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.

1.2 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 (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Равенство (1.1) можно переписать в виде (1.2)

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

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

В данном разделе будут рассмотрены требования к функциональности ПО, схемы алгоритмов и определены способы тестирования.

2.1 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность консольного приложения.

1) возможность подать на вход две матрицы;
2) при матрицах неправилыных размеров программа не должна аварийно завершаться;
3) корректное умножение двух матриц.

2.2 Тесты

Тестирование ПО будет проводиться методом чёрного ящика. Необходимо проверить работу системы на тривиальных случаях (размеры матриц не являются правильными, размер 1x1) и несколько нетривальных случаев.

2.3 Схемы алгоритмов

Ниже будут представлены схемы алгоритмов нахождениея произведения матриц:

1) классический алгоритм (рисунок 2.1);
2) алгоритм Винограда (рисунок 2.2);
3) алгоритм Винограда с оптимизациями (рисунок 2.3).

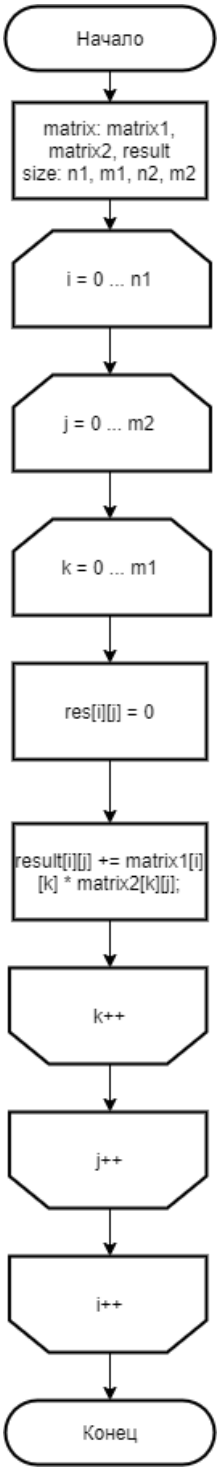

Рисунок 2.1 — Классический алгоритм умножения матриц

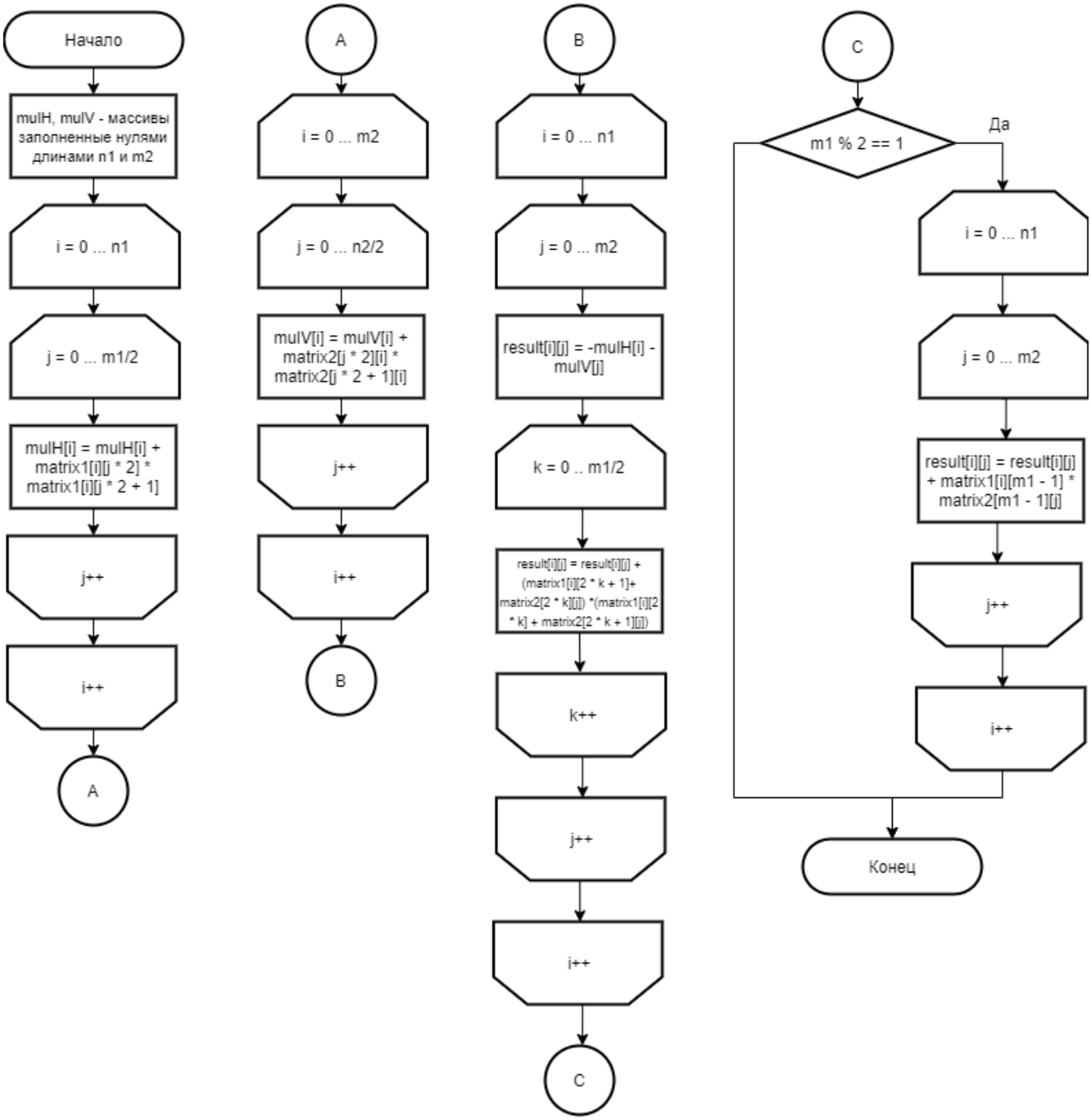

Рисунок 2.2 — Алгоритм Винограда

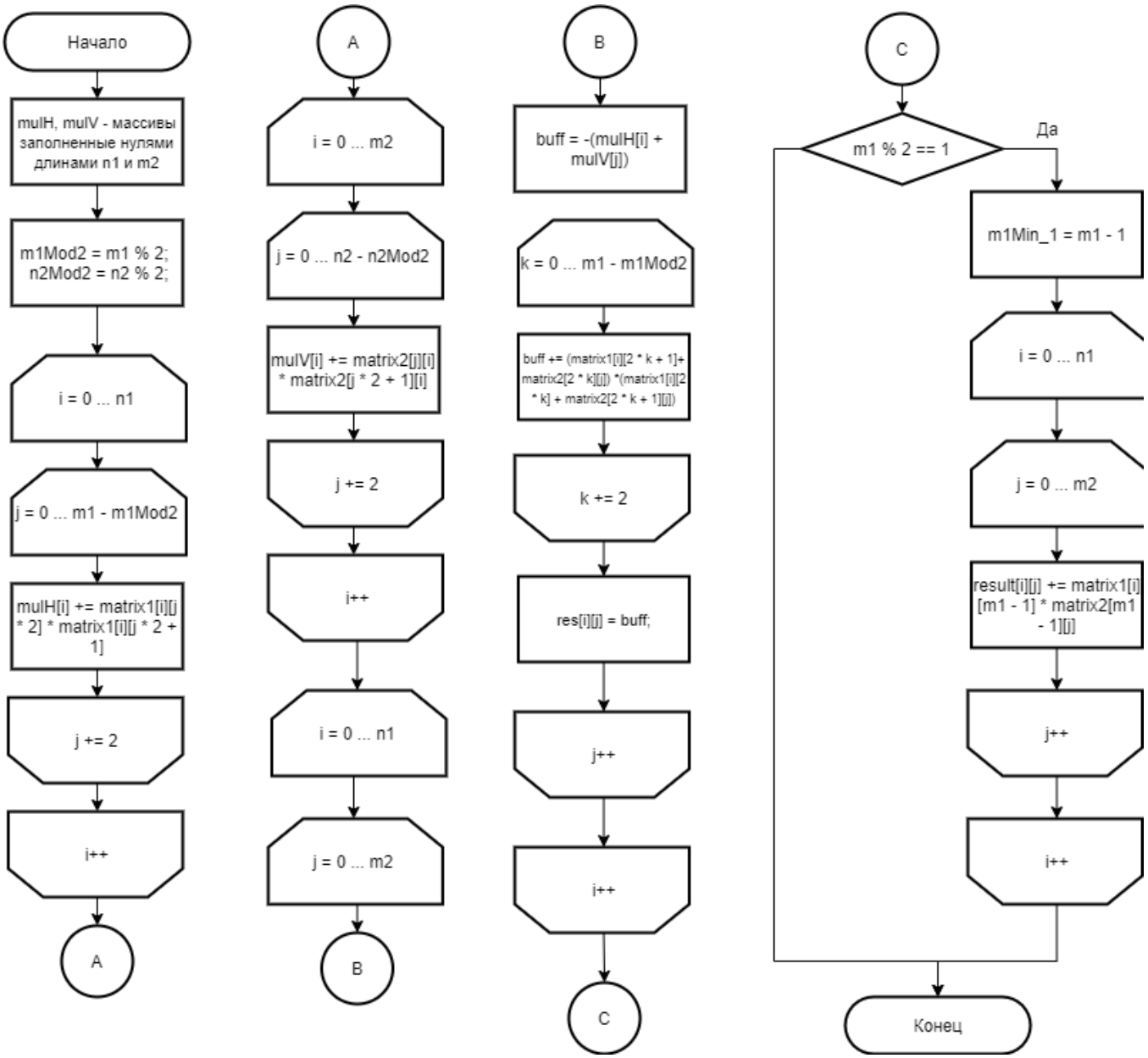

Рисунок 2.3 — Алгоритм Винограда с оптимизациями

2.4 Трудоёмкость алгоритма

Трудоёмкость – количество работы, которую алгоритм затрачивает на обработку данных. Является функцией от длины входов алгоритма и позволяет оценить количество работы.

Введём модель вычисления трудоёмкости.

2.4.1 Базовые операции

Ниже представлены базовые операции, стоимость которых единична:

1) $=, +, + =, -, - =, *, * =, /, / =, ++, --, \%$,
2) $<, \leqslant, ==, \neq, \geqslant, >$,
3) [ ].

2.4.2 Условный оператор

```
if (условие) {
// тело A
}
else {
// тело B
}
```

Пусть трудоёмкость тела A равна f_A, а тела B f_B, тогда стоимость условного оператора можно найти по формуле (2.1):

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B) - \text{лучший случай}, \\ max(f_A, f_B) - \text{худший случай} \end{cases} \tag{2.1}$$

2.4.3 Цикл со счётчиком

```
for (int i = 0; i < n; i++) {
// тело цикла
}
```

Начальная инициализация цикла (int i = 0) выполняется один раз. Условие i < n проверяется перед каждой итерацией цикла и при входе в цикл – n + 1 операций. Тело цикла выполняется ровно n раз. Счётчик (i++) выполняется на каждой итерации, перед проверкой условия, т.е. n раз. Тогда, если трудоёмкость тела цикла равна f, трудоёмкость всего цикла определяется формулой (2.2)

$$f_{\text{цикла}} = 2 + n(2 + f) \tag{2.2}$$

2.4.4 Стандартный алгоритм

Найдём трудоёмкость стандартного алгоритма.

$f_{\text{первый цикл}} = 2 + N1(2 + f_{\text{второй цикл}})$

$f_{\text{второй цикл}} = 2 + M2(2 + f_{\text{третий цикл}} + 3)$

$f_{\text{третий цикл}} = 2 + N1(2 + 8)$

$f_{\text{Стандартный}} = 10N1M2N1 + 7N1M2 + 4N1 + 2 \approx 10N1M1N1$

2.4.5 Алгоритм Винограда

Найдём трудоёмкость алгоритма Винограда.

$f_{\text{первый цикл}} = 2 + N1(2 + \frac{M1}{2}(2 + 12)) = 7N1M1 + 2N1 + 2$

$f_{\text{второй цикл}} = 7N2M2 + 2N1 + 2$

$f_{\text{третий цикл}} = 2 + N1(2 + M2(2 + 7 + \frac{M1}{2}(2 + 23))) = 12N1M1M2 + 9N1M2 + 2N1 + 2$

Условный переход $f_{if} = 2 + \begin{cases} 0 - \text{лучший случай}, \\ 15N1M2 + 2N1 + 2 - \text{худший случай} \end{cases}$

Итого:

$$f_{\text{Винограда}} = 12N1M1N1 + 16N1M2 + 7M1N1 + 6N1 + 6 + \begin{cases} 0 - \text{л.с.}, \\ 15N1M2 + 2N1 + 2 - \text{х.с.} \end{cases} \quad (2.3)$$

$f_{\text{Винограда}} \approx 12MNQ$

2.4.6 Оптимизированный алгоритм Винограда

Найдём трудоёмкость оптимизированного алгоритма Винограда.

$f_{\text{первый цикл}} = \frac{11}{2}(N1M1 + 4N1 + 2)$

$f_{\text{второй цикл}} = \frac{11}{2}M2N2 + 4M2 + 2$

$f_{\text{третий цикл}} = \frac{17}{2}N1M2M1 + 9N1M2 + 4N1 + 2$

Условный переход $f_{if} = 2 + \begin{cases} 0 - \text{лучший случай}, \\ 10N1M2 + 4N1 + 2 - \text{худший случай} \end{cases}$

Итого:

$$f_{\text{Винограда}} = \frac{17}{2}N1M2M1 + \frac{11}{2}N1M1 + \frac{11}{2}M2N2 + 9N1M2 + 8N1 + 4M2 + 6 + \begin{cases} 0 - \text{л.с.}, \\ 10N1M2 + 4N1 + 2 - \text{х.с.} \end{cases} \quad (2.4)$$

2.5 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель оценки трудоемкости алгоритма, были расчитаны трудоемкости алгоритмов в соответсвии с этой моделью. При схожем коэффициенте при старшем слагаемом в трудоемкости алгоритма Винограда и стандартного, доля долгих операций умножения в алгоритме Винограда меньше.

3 Технологическая часть

В данном разделе будут выбраны средства реализации ПО, представлен листинг кода и будет показан метод тестирования.

3.1 Средства реализации

Для реализации программ я выбрал язык программирования C++, так как имею большой опыт работы с ним. Для замера процессорного времени была использована ассемблерная вставка [1].

3.2 Сведения о модулях программы

Программа состоит из:

— main.cpp - главный файл программы, в котором располагается точка входа в программу.
— main_test.cpp - файл с функциями тестирования и замера времени.
— util.cpp - файл с функциями для выделения/заполнения/удаления матрицы.
— matrix_multiplication.cs - файл с реализациями 3-х алгоритмов умножения матриц.

3.3 Листинг программы

Ниже представлены листинги кода поиска растояния Левенштейна:

1) классический алгоритм (листинг 3.1);
2) алгоритм Винограда (листинг 3.2);
3) алгоритм Винограда с оптимизациями (листинг 3.3).

Листинг 3.1 — Классический алгоритма умножения матриц

```
1  IMatr_t classic_mult_matrix(IMatr_t matr1, int n1, int m1, IMatr_t matr2, int n2,
       int m2)
2  {
3      IMatr_t res = alloc_matrix(n1, m2);
4
5      for (int i = 0; i < n1; i++)
6      {
7          for (int j = 0; j < m2; j++)
8          {
9              res[i][j] = 0;
10             for (int k = 0; k < n1; k++)
11                 res[i][j] += matr1[i][k] * matr2[k][j];
12         }
13     }
14
15     return res;
16 }
```

Листинг 3.2 — Алгоритма Винограда

```
IMatr_t vinograd_mult_matrix(IMatr_t matr1, int n1, int m1, IMatr_t matr2, int n2,
    int m2)
{
    IMatr_t res = alloc_matrix(n1, m2);

    IArr_t mulH = (IArr_t)calloc(n1, sizeof(int));

    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];
        }
    }

    IArr_t mulV = (IArr_t)calloc(m2, sizeof(int));

    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] + (matr1[i][2 * k + 1] + matr2[2 * k][j]) *
                    (matr1[i][2 * k] + matr2[2 * k + 1][j]);
            }
        }
    }

    if (m1 \% 2 == 1)
    {
        for (int i = 0; i < n1; i++)
        {
            for (int j = 0; j < m2; j++)
            {
                res[i][j] = res[i][j] + matr1[i][m1 - 1] * matr2[m1 - 1][j];
```

```
            }
        }
    }

    return res;
}
```

Листинг 3.3 — Оптимизированный алгоритм Винограда

```
IMatr_t vinograd_optimized_mult_matrix(IMatr_t matr1, int n1, int m1, IMatr_t matr2,
    int n2, int m2)
{

    if ((n1 < 1 || m1 < 1 || n2 < 1 || m2 < 1) || m1 != n2)
    {
        return nullptr;
    }

    IMatr_t res = alloc_matrix(n1, m2);

    IArr_t mulH = (IArr_t)calloc(n1, sizeof(int));

    int m1Mod2 = m1 % 2;

    for (int i = 0; i < n1; i++)
    {
        for (int j = 0; j < (m1 - m1Mod2); j += 2)
        {
            mulH[i] += matr1[i][j] * matr1[i][j + 1];
        }
    }

    IArr_t mulV = (IArr_t)calloc(m2, sizeof(int));

    int n2Mod2 = n2 % 2;

    for (int i = 0; i < m2; i++)
    {
        for (int j = 0; j < (n2 - n2Mod2); j += 2)
        {
            mulV[i] += matr2[j][i] * matr2[j + 1][i];
        }
    }

    for (int i = 0; i < n1; i++)
    {
        for (int j = 0; j < m2; j++)
```

```
38          {
39              int buff = -(mulH[i] + mulV[j]);
40              for (int k = 0; k < (m1 - m1Mod2); k += 2)
41              {
42                  buff += (matr1[i][k + 1] + matr2[k][j]) * (matr1[i][k] + matr2[k +
                        1][j]);
43              }
44              res[i][j] = buff;
45          }
46      }
47
48      if (m1Mod2 == 1)
49      {
50          int m1Min_1 = m1 - 1;
51          for (int i = 0; i < n1; i++)
52          {
53              for (int j = 0; j < m2; j++)
54              {
55                  res[i][j] += matr1[i][m1Min_1] * matr2[m1Min_1][j];
56              }
57          }
58      }
59
60      return res;
61  }
```

3.3.1 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 3 оптимизации:

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

Листинг 3.4 — Оптимизации алгоритма Винограда №1 и №2

```
1       int m1Mod2 = m1 % 2;
2
3       for (int i = 0; i < n1; i++)
4       {
5           for (int j = 0; j < (m1 - m1Mod2); j += 2)
6           {
7               mulH[i] += matrix1[i][j] * matrix1[i][j + 1];
8           }
9       }
10
11      int n2Mod2 = n2 \% 2;
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]) * (matrix1[i][k] + matrix2[k
                   + 1][j]);
9          }
10         result[i][j] = buff;
11     }
12 }
```

3.4 Тестирование

В таблице 3.1 отображён возможный набор тестов для тестирования методом чёрного ящика. В столбцах Классический, Виноград, Виноград с опт-ей представлены соответсвенно результаты умножения "Матрица 1"и "Матрица 2 полученные классическим алгоритмом, алгоритмом Винограда и оптимизированным алгоритмом Винограда.

Таблица 3.1 — Таблица тестовых данных

№	Матрица 1	Матрица 2	Классический	Виноград	Виноград с опт-ей
1			NULL	NULL	NULL
2	2	2	4	4	4
3	0 1 1 2	0 1 1 2	1 2 2 5	1 2 2 5	1 2 2 5
4	0 1 2 1 2 3 2 3 4	0 1 2 1 2 3 2 3 4	5 8 11 8 14 20 11 20 29	5 8 11 8 14 20 11 20 29	5 8 11 8 14 20 11 20 29

3.5 Тестирование

В данном разделе выбраны средства реализации ПО, представлены листинги кодов 3-х алгоритмов. По итогам тестирования было показано, что алгоритмы работают корректно.

4 Исследовательская часть

В данном разделе будут проведены эксперименты для проведения сравнительного анализа алгоритмов по затрачиваемому процессорному времени [2].

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждой реализации на матрицах разных рамеров. В таблице 4.1 приняты следующие обозначения:

1) Classic – классический алгоритм;
2) Vinograd – алгоритм Винограда;
3) VinogradOpt – алгоритм Винограда с оптимизацией.

Графики по таблице изображены на рисунках 4.1, 4.2

Таблица 4.1 — Время работы реализации алгоритмов (в тиках)

size	Classic	Vinograd	VinogradOpt
100x100	6981488	5041345	4027075
101x101	7076101	5143886	4074936
200x200	54394516	42259227	32604859
201x201	55843226	41372159	32886685
300x300	222039403	167047884	134266889
301x301	204466455	152396429	123611178
400x400	535629631	399587985	318084044
401x401	528259830	411047549	318868217
500x500	1082411370	793781145	628826685
501x501	1082811261	800560090	642382560

4.2 Вывод

Самым медленным алгоритмом оказался алгоритм классического умножения матриц, а самым быстрым — оптимизированный алгоритм Винограда. Причем при нечетных размерах матрицы оба алгоритма Винограда работают чуть медленнее, чем при четных размерах.

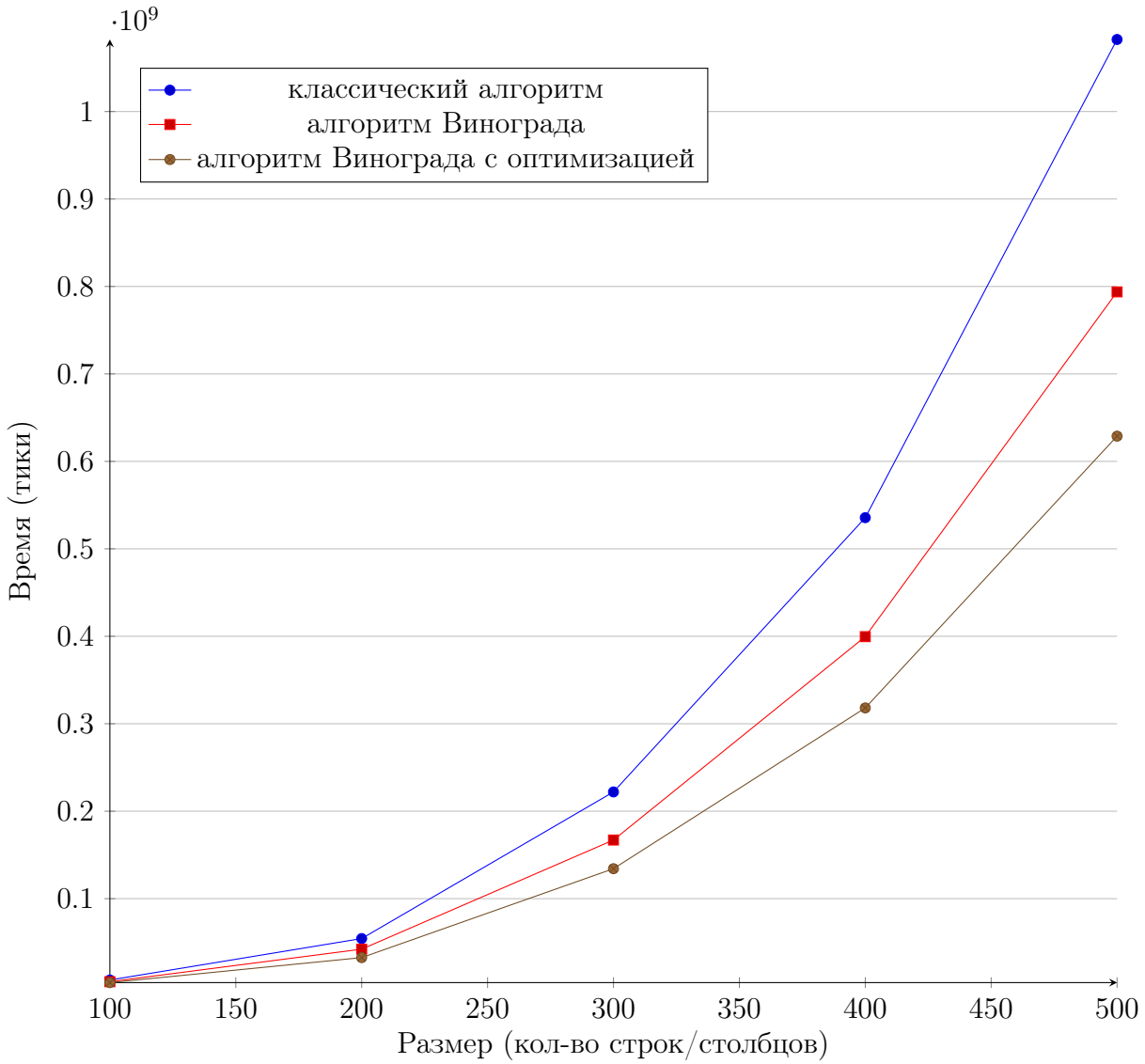

Рисунок 4.1 — Сравнение времени работы алгоритмов при четном размере матрицы

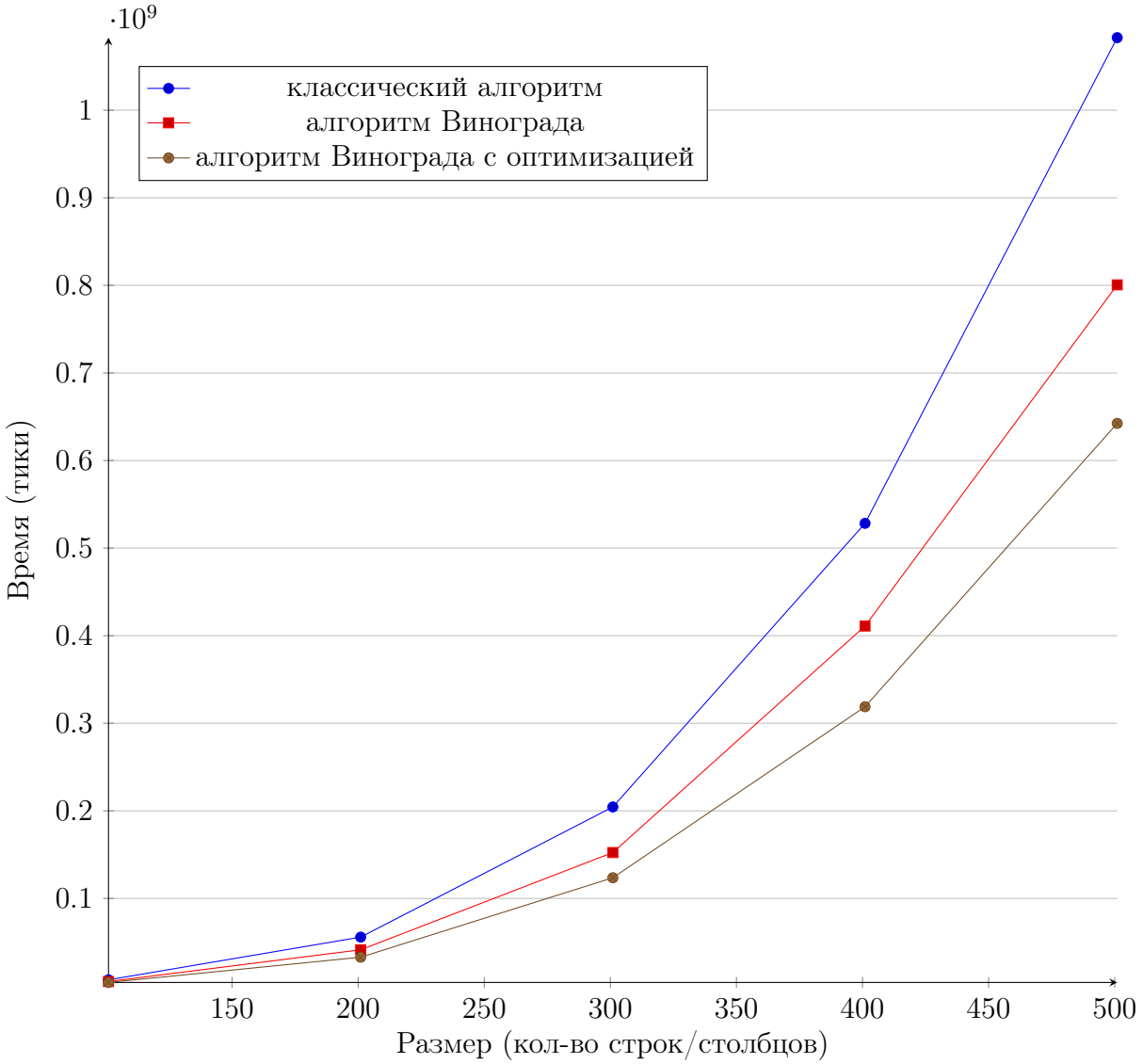

Рисунок 4.2 — Сравнение времени работы алгоритмов при нечетном размере матрицы

Заключение

В ходе лабораторной работы были изучены алгоритмы умножения матриц: стандартный, Винограда; составлен оптимизированный алгоритм Винограда

Экспериментально было подтверждено различие во временной эффективности реализаций алгоритма Винограда и классического алгоритма.

В результате исследований я пришел к выводу, что алгоритм Винограда может заметно увеличить скорость нахождения произведения матриц, следовательно более применим в реальных проектах.

Список использованных источников

1. Ассемблерные вставки в AVR-GCC. // [Электронный ресурс]. Режим доступа: http://we.easyelectronics.ru/AVR/assemblernye-vstavki-v-avr-gcc.html, (дата обращения: 03.10.2020).

2. C/C++: как измерять процессорное время. // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/282301/, (дата обращения: 03.10.2020).

3. Matrix multiplication. // [Электронный ресурс]. Режим доступа: *https* : *//en.wikipedia.org/wiki/Matrix*$_m$*ultiplication*, (дата обращения: 03.10.2020).