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: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент	ИУ7-55Б		Хетагуров П.К
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподователь			Л.Л. Волкова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Цель и задачи работы 4

1.2 Классическое умножение матриц 4

1.3 Алгоритм Винограда 4

1.4 Вывод 5

2 Конструкторская часть **6**

2.1 Требования к ПО 6

2.2 Оптимизация алгоритма Винограда 6

2.3 Схемы алгоритмов 6

2.4 Оценка трудоемкости 9

2.4.1 Трудоемкость стандартного алгоритма 9

2.4.2 Трудоемкость алгоритма Винограда 10

2.4.3 Трудоемкость оптимизированого алгоритма Винограда 10

2.5 Вывод 11

3 Технологическая часть **12**

3.1 Средства реализации 12

3.2 Реализации алгоритмов 12

3.3 Вывод 16

4 Экспериментальная часть **17**

4.1 Пример работы программы 17

4.2 Сравнительный анализ алгоритмов по времени 17

4.3 Вывод 19

Заключение **20**

Список литературы **21**

Введение

В данной лабораторной работе будут рассмотренны и проанализированы такие алгоритмы умножения матриц как:

1. стандартный;
2. Винограда;
3. Винограда оптимизированный;

1 Аналитическая часть

В данном разделе будут поставлены цели и задачи работы, будут рассмотренны основные теоритические сведения связанные с алгоритмами умножения матриц.

1.1 Цель и задачи работы

Цель работы:

Реализовать и сравнить по трудоемкости алгоритмы умножения матриц.

Задачи работы:

1. дать математическое описание умножения матриц;
2. разработать алгоритмы умножения матриц;
3. реализовать построенные алгоритмы;
4. провести эксперименты по замеру времени работы разработанных алгоритмов;
5. провести сравнения алгоритмов по затраченному времени;
6. дать оценку трудоемкости алгоритмов.

1.2 Классическое умножение матриц

Операция умножения двух матриц выполнима тогда и только тогда, когда число столбцов в первом сомножителе равно числу строк во втором.

Произведением матрицы A[m×n] на матрицу B[n×k] называется матрица C[m×k] такая, что элемент матрицы C, стоящий в i-ой строке и j-ом столбце, т. е. элемент $c_{i,j}$, равен сумме произведений элементов i-ой строки матрицы A на соответствующие элементы j-ого столбца матрицы B. Т.е определяется формулой [1]:

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

1.3 Алгоритм Винограда

Видно, что каждый элемент в результате умножения матриц представляет собой скалярное произведение соответствующих строки и столбца матриц. В алгоритме Винограда происходит некоторая предварительная обработка, позволяющая вычислить часть данных заранее. Заметим, что скалярное произведение двух векторов V и W, например, размерностью 4, можно переписать как [2]:

$$V * W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{2}$$

Видно, что выражение в правой части допускает предварительную обработку, его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй.[1]

1.4 Вывод

В данной части были поставлены задачи и цель работы, рассмотренны математическое описания классического алгоритма умножения матриц и алгоритма Винограда.

2 Конструкторская часть

В данном разделе будут рассмотренны схемы алгоритмов, требования к функциональности ПО, проведена оптимизация алгоритма Винограда и проведена оценка трудоемкости алгоритмов.

2.1 Требования к ПО

ПО должно иметь два режима работы, выбираемые из меню

1. Режим демонстрации. В этом режиме должен осуществляться ввод двух матриц и демонстрация работы на них всех реализованных алгоритмов.

2. Режим тестирования. В этом режиме должны проводится замеры времени выполнения реализованных алгоритмов. Должен осуществляться вывод затраченного процессорного времени на случайным образом сгенерированных данных.

2.2 Оптимизация алгоритма Винограда

Оптимизации:

1. выделены две новые переменные, хранящие m1%2, n2%2;

2. до каждого цикла, содержащего вычисление предела итерирования, предел итерирования вычисляется и записывается в переменную;

3. в циклах формирования массивов mulH, mulV и главном цикле умножение итерируемой переменной на два в теле цикла при каждом вычислении заменено на итерирование прибавлением двойки;

4. в главном цикле выделен буфер, предотвращающий многочисленное обращение к таблице;

5. заменены конструкции вида X = X + Y на X += Y;

2.3 Схемы алгоритмов

На рисунке [1] изображена схема стандартного алгоритма умножения матриц.

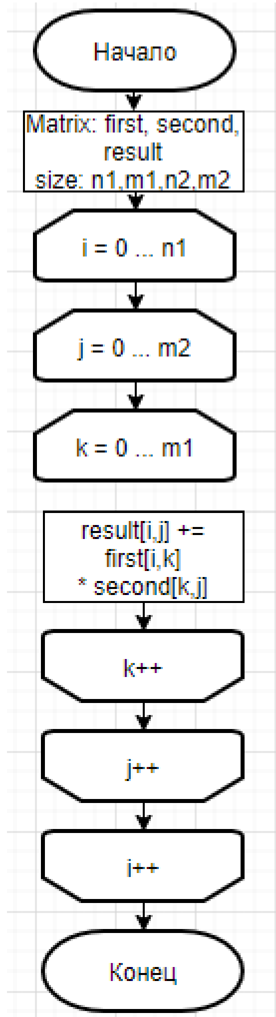

Рис. 1: Схема стандартного алгоритма умножения матриц

На [2] изображена схема алгоритма Винограда.

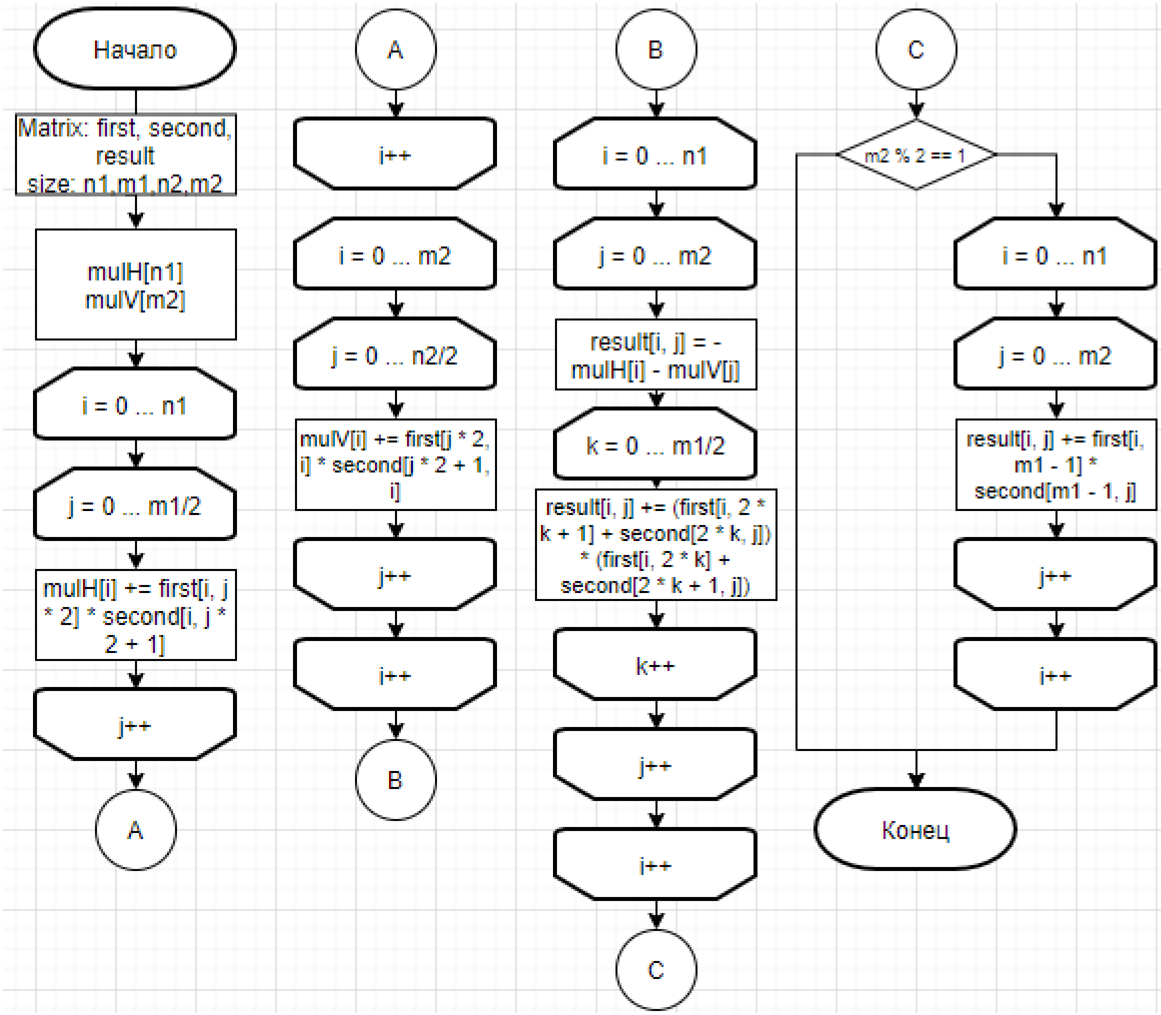

Рис. 2: Схема алгоритма Винограда

На рисунке [3] изображена схема модифицированного алгоритма Винограда.

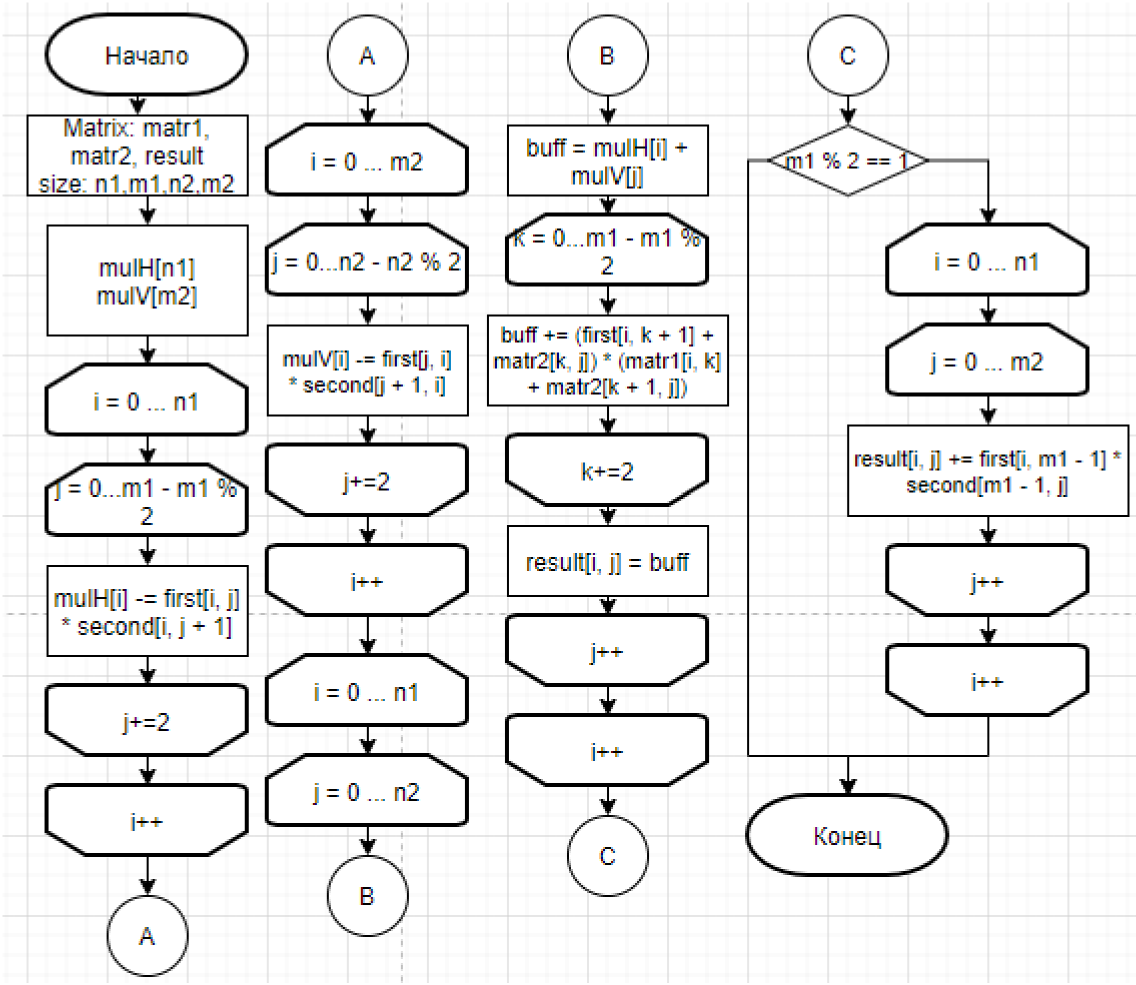

Рис. 3: Схема модифицированного алгоритма Винограда

2.4 Оценка трудоемкости

Модель трудоемкости для оценки алгоритмов:

Стоимость базовых операций единица:

$=, +, *, <, >, <=, >=, ==, ! =, [], + =, - =, * =, / =, ++, --$

2.4.1 Трудоемкость стандартного алгоритма

Трудоемкость внутреннего цикла N1= (8 + 2) * n2

Трудоемкость второго цикла N2 = (2 + 1 + N1) * m2 = (10*n2 + 3) * m2

Трудоемкость ввнешнего цикла N3 = (2 + 1 + N2) * n1 = (3 + (10*n2 + 3) * m2) * n1

Трудоемкость алгоритма = N3 + 1 = (3 + (10*n2 + 3) * m2) * n1 + 1 = 10*n2*m2*n1 + 3*m2*n1 + 3*n1 + 1

2.4.2 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 оператор:

$$\begin{cases} 2, \text{невыполнение} \\ 15n_1m_2 + 4n_1 + 2, \text{выполнение} \end{cases} \tag{3}$$

Результат: $13n_1m_2m_1 + \frac{15}{2}n_1m_1 + \frac{15}{2}m_2n_2 + 12n_1m_2 + 5n_1 + 5m_2 + 4n_1 + 6+$

$$\begin{cases} 2, \text{невыполнение} \\ 15n_1m_2 + 4n_1 + 2, \text{выполнение} \end{cases} \tag{4}$$

2.4.3 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 оператор:

$$\begin{cases} 2, \text{невыполнение} \\ 10n_1m_2 + 4n_1 + 2, \text{выполнение} \end{cases} \tag{5}$$

Результат: $\frac{17}{2}n_1m_2m_1 + \frac{11}{2}n_1m_1 + \frac{11}{2}m_2n_2 + 9n_1m_2 + 8n_1 + 4m_2 + 6+$

$$\begin{cases} 2, \text{невыполнение} \\ 10n_1m_2 + 4n_1 + 2, \text{выполнение} \end{cases} \tag{6}$$

2.5 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, была рассчитана трудоемкость алгоритмов и обозначены требования к ПО.

3 Технологическая часть

Ниже будут представлены средствы реализации и листинги реализованной программы.

3.1 Средства реализации

Выбранный язык программирования - C++, так как требований по конкретнему языку не выдвигалось и он был изучен во время обучения. Среда разработки - Visual Studio Code.[2]

Функции вычисления процессорного времени - QueryPerfomanceCounter из библиотеки WinAPI.[3]

3.2 Реализации алгоритмов

Ниже представлены листинги реализаций алгоритмов. На листинге [1] представлен стандартный алгоритм умножения матриц.

Листинг 1: Стандартный алгоритм умножения матриц

```
vector<vector<int>> multiplyClassic(vector< vector<int>> first, vector< vector<int>>
    second)
{
    int nFirst = first.size();
    int nSecond = second.size();

    vector<vector<int>> result;
    if (nFirst > 0 && nSecond > 0 && first[0].size() == nSecond && second[0].size() != 0)
    {
        int mSecond = second[0].size();

        result = createZeroMatrix(nFirst, mSecond);
        for (int i = 0; i < nFirst; i++)
        {
            for (int j = 0; j < mSecond; j++)
            {
                for (int k = 0; k < nSecond; k++)
                {
                    result[i][j] += first[i][k] * second[k][j];
                }
            }
        }
    }

    return result;
}
```

На листинге [2] Представлен алгоритм Винограда.

Листинг 2: Алгоритм Винограда

```
vector<vector<int>> multiplyVinograd(vector< vector<int>> first, vector< vector<int>>
    second)
{
    int nFirst = first.size();
    int nSecond = second.size();

    vector<vector<int>> result;

    if (nFirst > 0 && nSecond > 0 && first[0].size() == nSecond && second[0].size() != 0)
    {
        int mSecond = second[0].size();
        int mFirst = nSecond;

        result = createZeroMatrix(nFirst, mSecond);

        vector<int> mulH, mulV;
        mulH.reserve(nFirst);
        mulV.reserve(mSecond);

        for (int i = 0; i < nFirst; i++)
        {
            mulH[i] = 0;
            for (int j = 0; j < mFirst / 2; j++)
            {
                mulH[i] += first[i][j * 2] * first[i][j * 2 + 1];
            }
        }

        for (int i = 0; i < mSecond; i++)
        {
            mulV[i] = 0;
            for (int j = 0; j < nSecond / 2; j++)
            {
                mulV[i] += second[j * 2][i] * second[j * 2 + 1][i];
            }
        }

        for (int i = 0; i < nFirst; i++)
        {
```

```
            for (int j = 0; j < mSecond; j++)
            {
                result[i][j] = - mulH[i] - mulV[j];
                for (int k = 0; k < mFirst / 2; k++)
                {
                    result[i][j] += (first[i][2 * k + 1] + second[2 * k][j]) * (first[i][2
                        * k] + second[2 * k + 1][j]);
                }
            }
        }

        if (mFirst % 2 == 1)
        {
            for (int i = 0; i < nFirst; i++)
            {
                for (int j = 0; j < mSecond; j++)
                {
                    result[i][j] += first[i][mFirst - 1] * second[mFirst - 1][j];
                }
            }
        }
    }

    return result;
}
```

На листинге [3] Представлен алгоритм Винограда оптимизированный.

Листинг 3: Алгоритм Винограда оптимизированный

```
vector<vector<int>> multiplyVinogradOpt(vector< vector<int>> first, vector< vector<int>>
    second)
{
    int nFirst = first.size();
    int nSecond = second.size();

    vector<vector<int>> result;

    if (nFirst > 0 && nSecond > 0 && first[0].size() == nSecond && second[0].size() != 0)
    {
        int mSecond = second[0].size();
        int mFirst = nSecond;

```

```
        result = createZeroMatrix(nFirst, mSecond);

        vector<int> mulH, mulV;
        mulH.reserve(nFirst);
        mulV.reserve(mSecond);

        int m1Mod2 = mFirst % 2;
        int n2Mod2 = nSecond % 2;

        int temp = nSecond - n2Mod2;
        for (int i = 0; i < mSecond; i++)
        {
            mulV[i] = 0;
            for (int j = 0; j < temp; j += 2)
            {
                mulV[i] += second[j][i] * second[j + 1][i];
            }
        }

        temp = mFirst - m1Mod2;
        for (int i = 0; i < nFirst; i++)
        {
            mulH[i] = 0;
            for (int j = 0; j < temp; j += 2)
            {
                mulH[i] += first[i][j] * first[i][j + 1];
            }
        }

        for (int i = 0; i < nFirst; i++)
        {
            for (int j = 0; j < mSecond; j++)
            {
                int buff = -(mulH[i] + mulV[j]);
                for (int k = 0; k < temp; k += 2)
                {
                    buff += (first[i][k + 1] + second[k][j]) * (first[i][k] + second[k +
                        1][j]);
                }
                result[i][j] = buff;
            }
```

```
        }

        if (m1Mod2 == 1)
        {
            temp = mFirst - 1;
            for (int i = 0; i < nFirst; i++)
            {
                for (int j = 0; j < mSecond; j++)
                {
                    result[i][j] += first[i][temp] * second[temp][j];
                }
            }
        }
    }

    return result;
}
```

3.3 Вывод

В данном разделе были описаны средства реализации, были представлены листинги реализации стандартного алгоритма умножения матриц, обычного и оптимизированного алгоритма Винограда.

4 Экспериментальная часть

В данной главе будут представлен пример работы программы, результат экспериментов по замеру времени и произведен сравнительный анализ алгоритмов по затрачиваемому времени.

4.1 Пример работы программы

Пример работы программы представлен на рисунке [4]

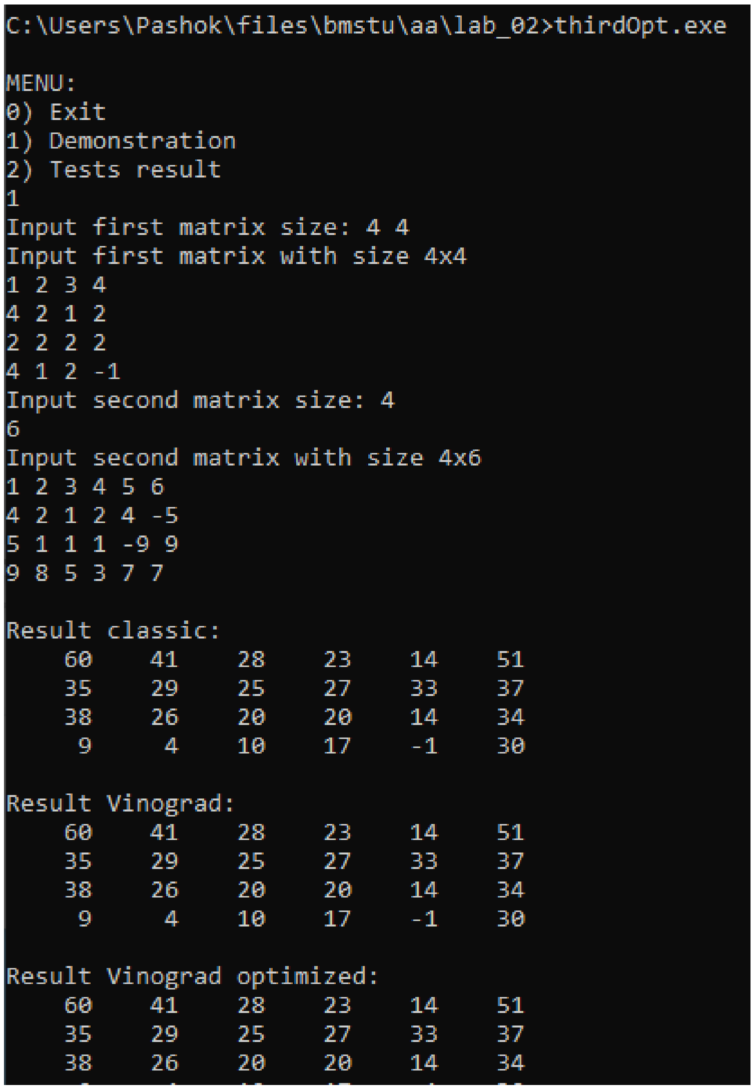

Рис. 4: Пример работы программы

4.2 Сравнительный анализ алгоритмов по времени

Эксперименты проводятся на квадратных матрицах со стороной длиной от 2 до 252 с шагом 50 (результаты на рисунке [5]

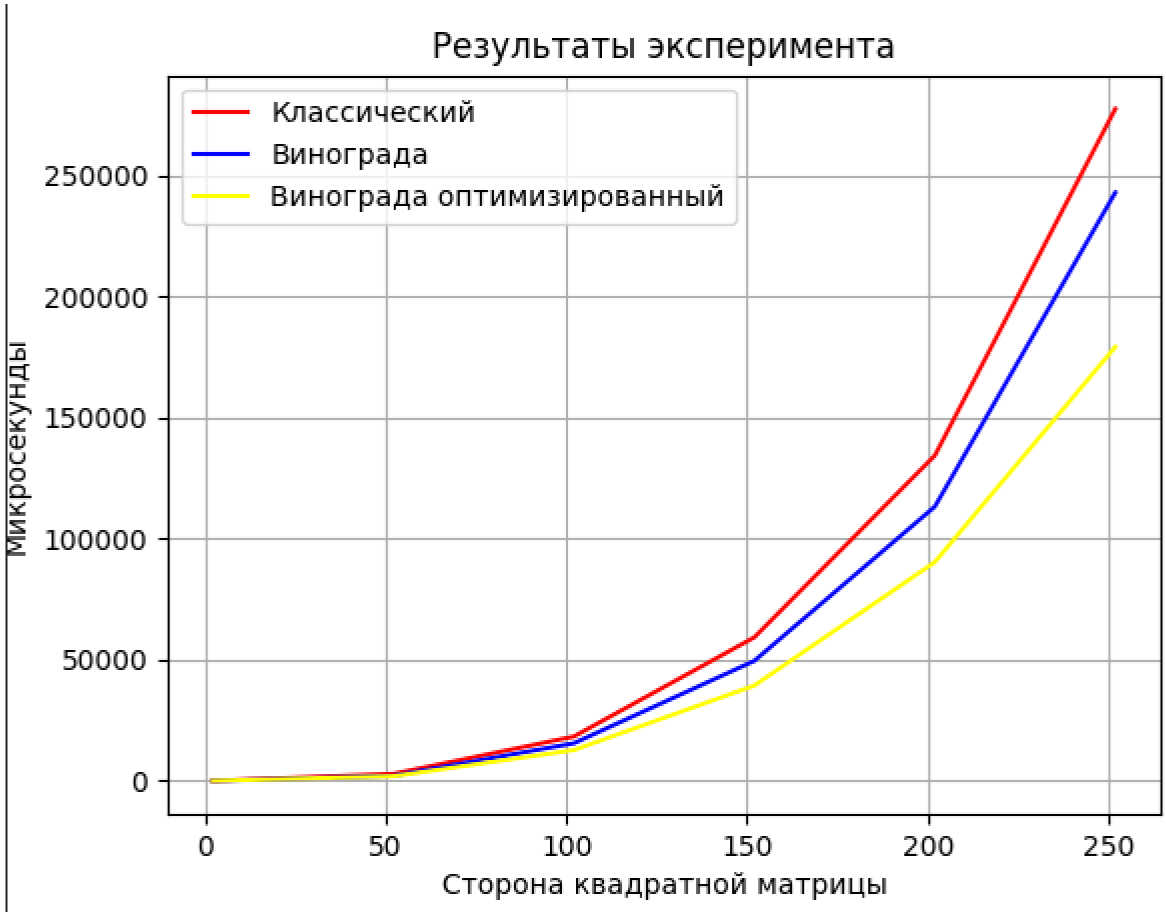

Рис. 5: Результаты на матрицах со стороной от 2 до 252

И на строках длины от 302 до 502 с шагом 50 (результаты на рисунке [6]

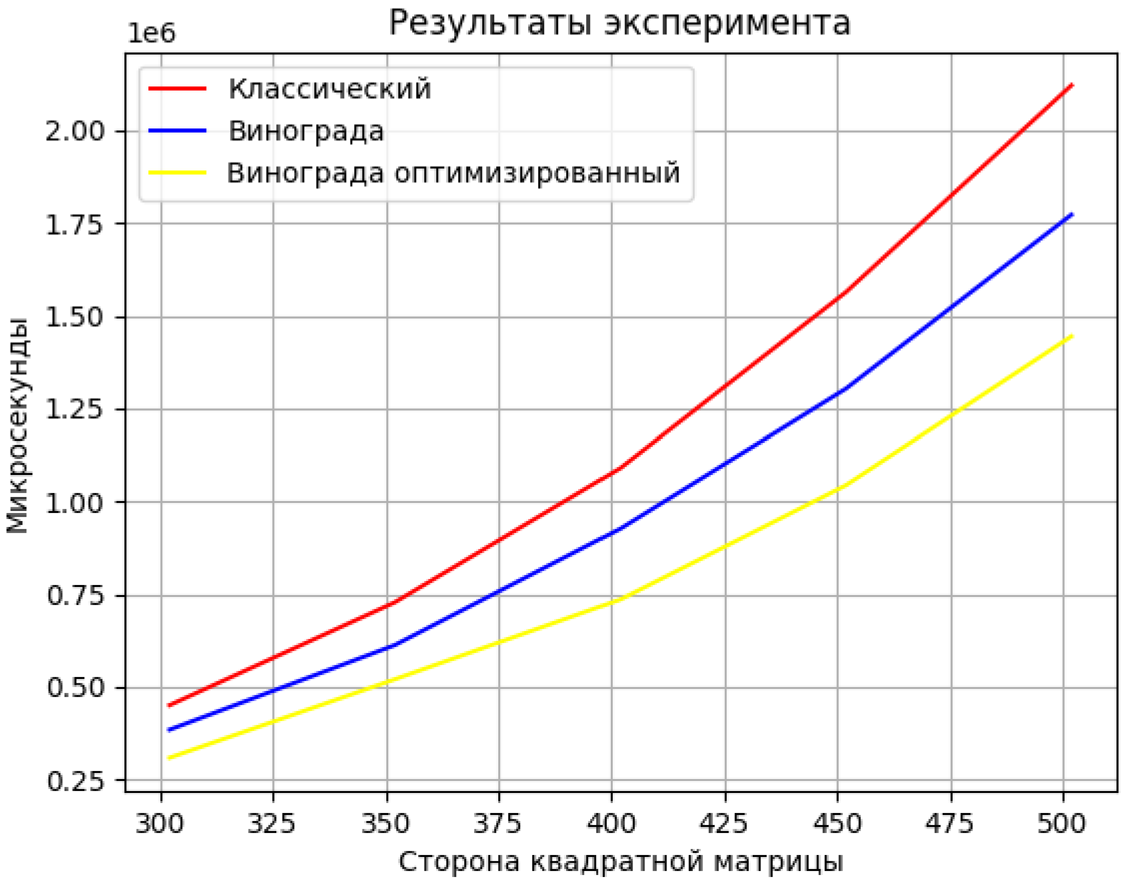

Рис. 6: Результаты на матрицах со стороной от 302 до 502

4.3 Вывод

Как видно из графиков, самым долгим алгоритмом является стандартный алгоритм. Самым быстрым является оптимизированный алгоритм Винограда. Простой алгоритм Винограда алгоритм быстрее стандартного, но уступает своей оптимизированной версииы.

Заключение

В данной лабораторной работе были изложены теоретические основы стандартного умножения матриц и алгоритма Винограда, они были разработаны и реализованы, были проведены эксперименты по замеру времени работы разработанных алгоритмов и проведены сравнения алгоритмов по результатам эксперимента. Также была дана оценка трудоемкости.

Список литературы

[1] Умножение матриц [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL: http://www.algolib.narod.ru/Math/Matrix.html

[2] Visual Studio Code [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL: code.visualstudio.com

[3] WinAPI. Функция QueryPerformanceCounter [Электронный ресурс]. Режим доступа: (дата обращения - 02.10.2020) Свободный. URL: https://docs.microsoft.com/en-us/windows/win32/api/profileapi/nf-profileapi-queryperformancecounter