Государственное образовательное учреждение высшего профессионального образования
"Московский государственный технический университет имени Н.Э.Баумана"

Дисциплина: Анализ алгоритмов

Лабораторная работа 2

Умножение матриц с помощью стандартного алгоритма и алгоритма Винограда.

Студент группы ИУ7-55,
Аминов Тимур Саидович

2019 г.

Содержание

Введение

Умножение матриц - одна из самых используемых матричных операций. Самый простой с точки разработки написания алгоритмом является стандартный, однако он не самый эффективный по процессорному времени, он уступает в этом отношении алгоритму Винограда.
В данной лабораторной работе будут изучены стандартный алгоритм умножения матриц и алгоритм Винограда. Цели раоты:

1. изучение трудоемкости стандартного алгоритма умножения матриц и алгоритма Винограда;
2. получение навыка оптимизации алгоритма с целью снижения трудоемкости его выполнения на примере решения задачи умножения матриц;
3. экспериментальное подтверждение оценок трудоемкости.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмов и приведена модель вычислений для оценок трудоемкости.

1.1 Теоретические сведения об умножении матриц

Матрица – это прямоугольная таблица каких-либо элементов. Здесь и далее мы будем рассматривать только матрицы, элементами которых являются числа. Упорядоченная пара чисел (n, m), где n - количество строк в матрице, m - количество столбцов, называется размерностью матрицы, обозначается обычно m x n. Пусть имеются две матрицы: A и B размерами n x l и l x m соответственно.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,l} \\ a_{2,1} & a_{2,2} & \dots & a_{2,l} \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,l} \end{bmatrix}$$

$$\begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,m} \\ b_{2,1} & b_{2,2} & \dots & b_{2,m} \\ \dots & \dots & \dots & \dots \\ b_{l,1} & b_{l,2} & \dots & b_{l,m} \end{bmatrix}$$

Произведением матриц A и B размерами n x l и l x m соответственно называется матрица C размерами n x m, каждый элемент которой вычисляется по формуле 1:

$$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j} \tag{1}$$

$$\begin{bmatrix} c_{1,1} & b_{1,2} & \dots & c_{1,m} \\ c_{2,1} & b_{2,2} & \dots & c_{2,m} \\ \dots & \dots & \dots & \dots \\ c_{n,1} & c_{n,2} & \dots & c_{n,m} \end{bmatrix}$$

1.2 Стандартный алгоритм умножения матриц

Матрица C в стандартном алгоритме находится последовательным вычислением элементов с индексами i, j, $i = \overline{1,n}$, $j = \overline{1,m}$ по формуле 1. Кажется, что этот алгоритм минимален по требуемому процессорному времени, но это не так.

1.3 Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Также некоторые вычисления можно произвести заранее, что ускорит выполнение алгоритма. 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

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

В Алгоритме Винограда используется скалярное произведение из формулы 2, в отличие от стандартного алгоритма. Алгоритм Винограда позволяет выполнить предварительную обработку матрицы и запомнить значения для каждой строки/столбца матриц. Над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.4 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

1. базовые операции имеют трудоемкость 1 ($<$, $>$, $=$, $<=$, $=>$, $==$, $+$, -, *, $/$, %, &, $+=$, -$=$, *$=$, $/=$, [ ]);

2. операторы if, else if имеют трудоемкость $F_{if} = F_{body} + F_{chek}$, F_{body} - трудоемкость операций тела оператора, F_{chek} - трудоемкость проверки условия;

3. оператор else имеет трудоемкость F_{body};

4. оператор for имеет трудоемкость $F_{for} = 2 + N \cdot (F_{body} + F_{chek})$, где F_{body} – трудоемкость операций в теле цикла.

1.5 Вывод

Были рассмотрены стандартный алгоритм умножения матриц и алгоритм Винограда, основные отличия которого - наличие предварительных вычислений и сокращние количества умножений. Также была дана модель вычислений, которая будет использоваться для сравнения трудоемкости алгоритмов в дальнейшем.

2 Конструкторская часть

В данной части будут рассмотрены схемы всех алгоритмов, рассматриваемых в данной лобораторной работе.

2.1 Схемы алгоритмов

На рисунках 1-3 приведены схемы стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда. Модификации для алгоритма Винограда: замена конструкций вида a = a + b на a += b, избавление от деления в условии циклов и накопление результата в буфер. Модификации алгоритма Винограда рассматриваются подробно в разделе 3.4.

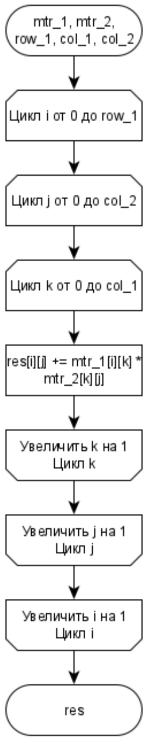

Рис. 1: Стандартный алгоритм

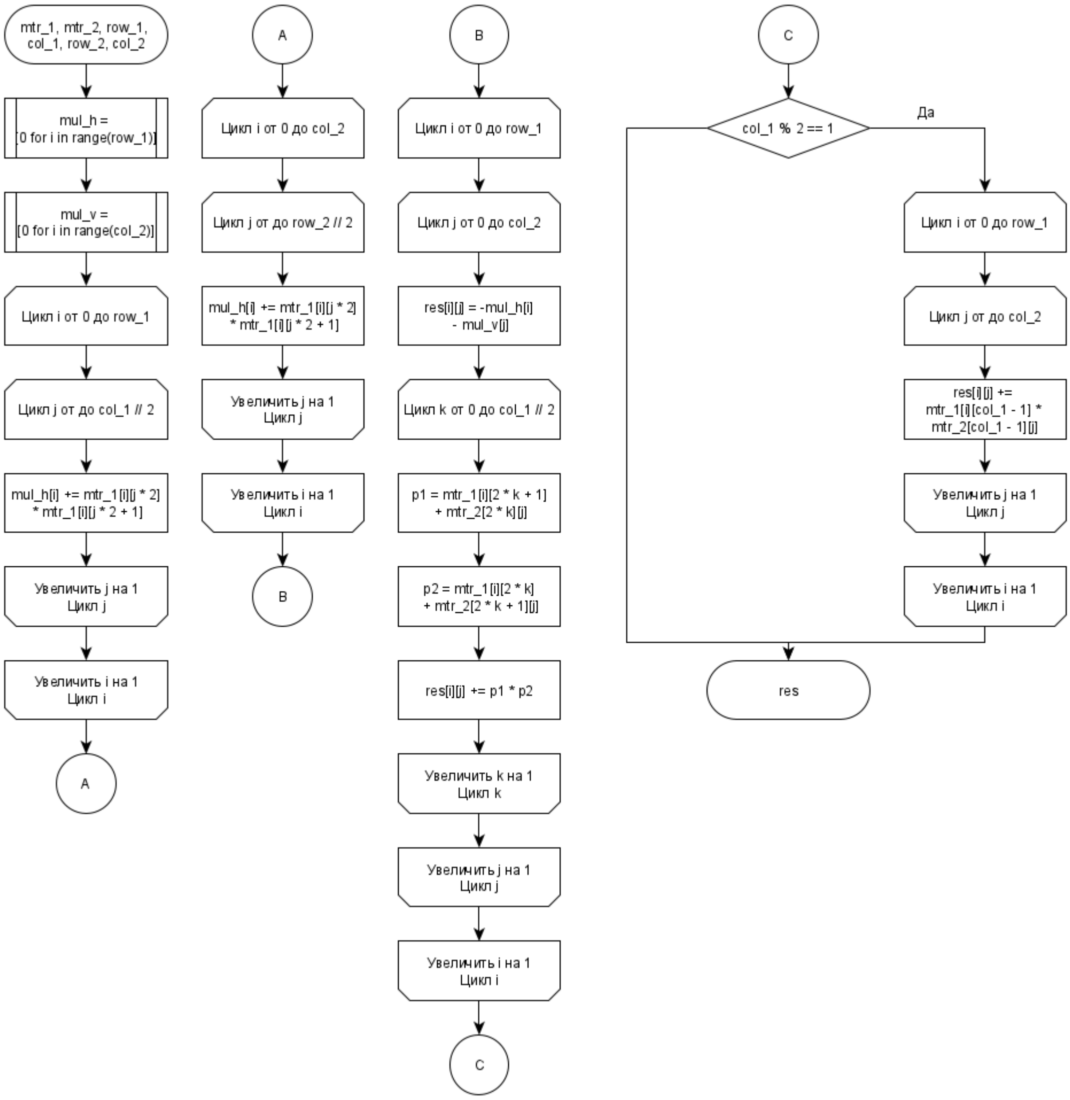

Рис. 2: Алгоритм Винограда

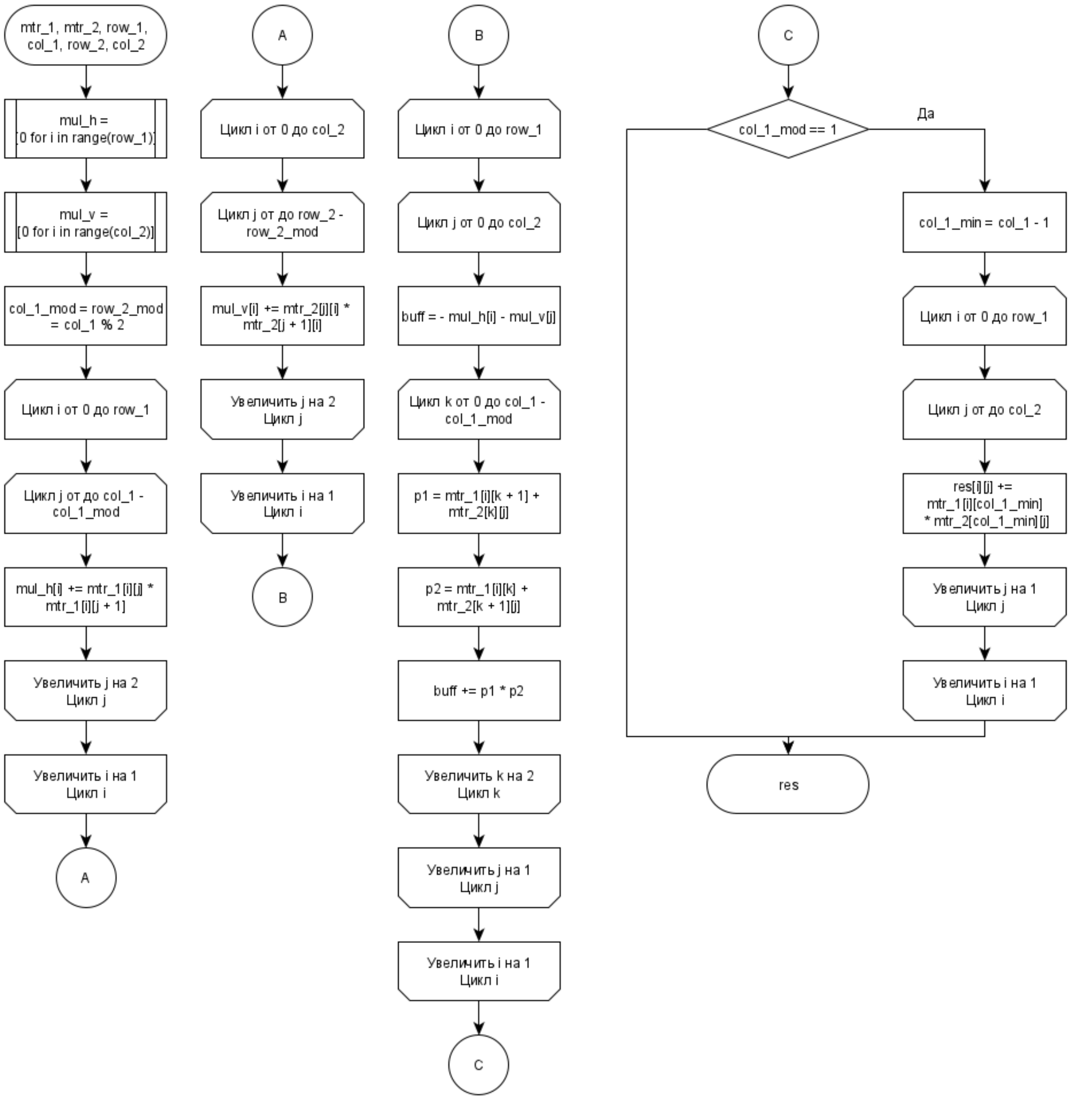

Рис. 3: Алгоритм Винограда(оптимизированный)

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритмов, таких как: стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

3 Технологическая часть

В данном разделе будут приведены листинги алгоритмов на языке python, оптимизации для алгоритма Винограда, оценена трудоемкости каждого алгоритма.

3.1 Требования к программному обеспечению

Входные данные - матрица1, матрица2, их размеры. Выходные данные - произведение матриц.

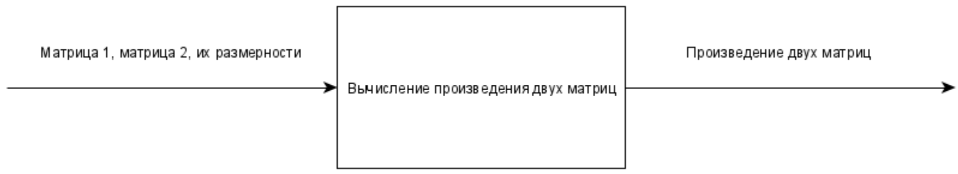

Рис. 4: IDEF0-диаграмма, описывающая алгоритм умножения матриц

3.2 Средства реализации

Программа была написана на языке python в среде разработки PyCharm. Программа корректно работает с пустыми и неправильно введенными данными. Для замеров была использована библиотека timeit.

3.3 Листинг кода

В листингах 1-3 приведены все рассматриваемые в рамках данной лабораторной работы алгоритмы, написанные на языке python.

Листинг 1: Стандартный алгоритм

```
def std_mpl(mtr_1, mtr_2):
    row_1 = len(mtr_1)
    col_1 = row_2 = len(mtr_1[0])
    col_2 = len(mtr_2[0])

    res = [[0 for j in range(col_2)] for i in range(row_1)]

    for i in range(row_1):
        for j in range(col_2):
            for k in range(col_1):
                res[i][j] += mtr_1[i][k] * mtr_2[k][j]

    return res
```

Листинг 2: Алгоритм Винограда

```
def winograd(mtr_1, mtr_2):
    row_1 = len(mtr_1)
    col_1 = row_2 = len(mtr_1[0])
    col_2 = len(mtr_2[0])

    res = [[0 for j in range(col_2)] for i in range(row_1)]

    mul_h = [0 for i in range(row_1)]
    mul_v = [0 for i in range(col_2)]

    for i in range(row_1):
        for j in range(col_1 // 2):
            mul_h[i] += mtr_1[i][j * 2] * mtr_1[i][j * 2 + 1]

    for i in range(col_2):
        for j in range(row_2 // 2):
            mul_v[i] += mtr_2[j * 2][i] * mtr_2[j * 2 + 1][i]

    for i in range(row_1):
        for j in range(col_2):
            res[i][j] = -mul_h[i] - mul_v[j]
            for k in range(col_1 // 2):
                res[i][j] += (mtr_1[i][2 * k + 1] + mtr_2[2 * k][j]) * \
                        (mtr_1[i][2 * k] + mtr_2[2 * k + 1][j])

    if col_1 % 2 == 1:
        for i in range(row_1):
            for j in range(col_2):
                res[i][j] += mtr_1[i][col_1 - 1] * mtr_2[col_1 - 1][j]

    return res
```

Листинг 3: Оптимизированный алгоритм Винограда

```
def winograd_opt(mtr_1, mtr_2):
    row_1 = len(mtr_1)
    col_1 = row_2 = len(mtr_1[0])
    col_2 = len(mtr_2[0])

    col_1_mod = row_2_mod = col_1 % 2

    res = [[0 for j in range(col_2)] for i in range(row_1)]

    mul_h = [0 for i in range(row_1)]
    mul_v = [0 for i in range(col_2)]

    for i in range(row_1):
        for j in range(0, col_1 - col_1_mod, 2):
            mul_h[i] += mtr_1[i][j] * mtr_1[i][j + 1]

    for i in range(col_2):
        for j in range(0, row_2 - row_2_mod, 2):
            mul_v[i] += mtr_2[j][i] * mtr_2[j + 1][i]

    for i in range(row_1):
        for j in range(col_2):
            buff = - mul_h[i] - mul_v[j]
            for k in range(0, col_1 - col_1_mod, 2):
                buff += (mtr_1[i][k + 1] + mtr_2[k][j]) * \
                    (mtr_1[i][k] + mtr_2[k + 1][j])
            res[i][j] = buff

    if col_1_mod == 1:
        col_1_min = col_1 - 1
        for i in range(row_1):
            for j in range(col_2):
                res[i][j] += mtr_1[i][col_1_min] * mtr_2[col_1_min][j]

    return res
```

3.4 Оптимизация алгоритма Винограда

Для оптимизации алгоритма Винограда были использованы следующие модификации:

1. все конструкции вида $a = a + b$ были заменены на $a += b$, пример показан на листинге 4;

Листинг 4: Оптимизация 1 и 2

```
for i in range(row_1):
        for j in range(0, col_1 - col_1_mod, 2):
            mul_h[i] += mtr_1[i][j] * mtr_1[i][j + 1]
```

2. Избавление от деления в условии циклов. Пример показан на листинге 4;

3. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти. Сброс буфера в ячейку матрицы после цикла. Пример показан на листинге 5;

Листинг 5: Оптимизация 3

```
for i in range(row_1):
        for j in range(col_2):
            buff = - mul_h[i] - mul_v[j]
            for k in range(0, col_1 - col_1_mod, 2):
                buff += (mtr_1[i][k + 1] + mtr_2[k][j]) * \
                        (mtr_1[i][k] + mtr_2[k + 1][j])
            res[i][j] = buff
```

3.5 Оценка трудоемкости

Таблица 1. Трудоемкость стандартного алгоритма умножения матриц.

Трудоемкость	Оценка
Fтела	8
Fстанд	2 + n(4 + m(4 + l(4 + Fтела))
Fстанд	**10mln + 4mn + 4n + 2**

Таблица 2. Трудоемкость алгоритма Винограда.

Трудоемкость	Оценка
Fтела1	12
Fтела2	12
Fтела3	23
Fтела4	9 + l/2 *(3 + Fтела3)
Fтела5	13
Fусловия	3 + n(4 + m(4 + Fтела5))
Fвиноград(лучший случай)	**7 + 9n + 7nl + 5m + 7ml + 11nm + 13nml**
Fвиноград(худший случай)	**9 + 13n + 7nl + 5m + 7ml + 28nm + 13nml**

Таблица 3. Трудоемкость оптимизированного алгоритма Винограда.

Трудоемкость	Оценка
Fтела1	8
Fтела2	8
Fтела3	16
Fтела4	7 + l/2 * (3 + Fтела3)
Fтела5	8
Fусловия	n(4 + m(4 + Fтела5))
Fво(лучший случай)	**11 + 9n + 5nl + 5m + 5ml + 9nm + 9.5nml**
Fво(худший случай)	**12 + 13n + 5nl + 5m + 5ml + 21nm + 9.5nml**

Трудоемкость оценивается по самому быстрорастущему слагаемому, то есть mln(куб). Из таблиц 1-3 мы видим, что у стандартного алгоритма коэффициент при этом слагаемом 10, у алгоритма Винограда - 13, а у оптимизированного алгоритма Винограда - 9.5, значит оптимизированный Виноград менее затратен по времени в заданной в аналитическом разделе модели вычислений.

3.6 Вывод

В данном разделе были приведены листинги стандартного алгоритма, алгоритма Винограда и оптимизированного Винограда, также были даны модификации для оптимизации алгоритма Винограда и был произведен анализ трудоемкости всех трех алгоритмов.

4 Экспериментальная часть

В данном разделе будут сравнены все три рассматриваемые в Лабораторной работе алгоритма на предмет затрачиваемого процессорного времени а также проверена правильность работы каждого из алгоритмов на нескольких примерах.

4.1 Примеры работы программы

Листинг 6: Пример работы 1

```
Input filename of first matrix: mtr/in_1.txt
Input filename of second matrix: mtr/in_1.txt
First matrix:
1 2 3
4 5 6
7 8 9

Second matrix:
1 2 3
4 5 6
7 8 9

Result using standart method:
30 36 42
66 81 96
102 126 150

Result using Winograd:
30 36 42
66 81 96
102 126 150

Result using optimized Winograd:
30 36 42
66 81 96
102 126 150
```

В данном примере все алгоритмы дали верный результат.

Листинг 7: Пример работы 2

```
Input filename of first matrix: in_0.txt
Matrix 1:

Empty matrix

Input filename of second matrix: in_0.txt
Matrix 2:

Empty matrix

Result matrix(standart):

Empty matrix

Result matrix(vinograd):

Empty matrix

Result matrix(vinograd optimized):

Empty matrix
```

В данном примере все алгоритмы дали верный результат.

Листинг 8: Пример работы 3

```
Input filename of first matrix: in_1.txt
Matrix 1:
```

```

1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000


Input filename of second matrix: in_2.txt
Matrix 2:

1.000000 1.000000 1.000000
1.000000 1.000000 1.000000
1.000000 1.000000 1.000000


Result matrix(standart):

6.000000 6.000000 6.000000
15.000000 15.000000 15.000000
24.000000 24.000000 24.000000

Result matrix(vinograd):

6.000000 6.000000 6.000000
15.000000 15.000000 15.000000
24.000000 24.000000 24.000000

Result matrix(vinograd optimized):

6.000000 6.000000 6.000000
15.000000 15.000000 15.000000
24.000000 24.000000 24.000000
```

В данном примере все алгоритмы дали верный результат.
В примерах, приведенных на листингах 7-9, все алгоритмы дали правильный результат.

4.2 Постановка эксперимента

Требуется сравнить затрачиваемое время всеми тремя алгоритмами при матрицах с четными и нечетными размерами(так как в алгоритме Винограда и оптимизированном алгоритме Винограда худший случай возникает именно при нечетных размерах матрицы). размеры были выбраны такие: 100x100, 200x200, 300x300, 400x400, 500x500, 600x600, 101x101, 201x201, 301x301, 401x401, 501x501, 601x601.

4.3 Результаты эксперимента

На рисунке 5 приведено время, затрачиваемое алгоритмами при матрицах рамерностей 100x100, 200x200, 300x300, 400x400, 500x500, 600x600, на рисунке 6 - 101x101, 201x201, 301x301, 401x401, 501x501, 601x601. На обоих графиках на оси абсцисс отложена размерность матриц, на оси ординат-затрачиваемое время в секундах.

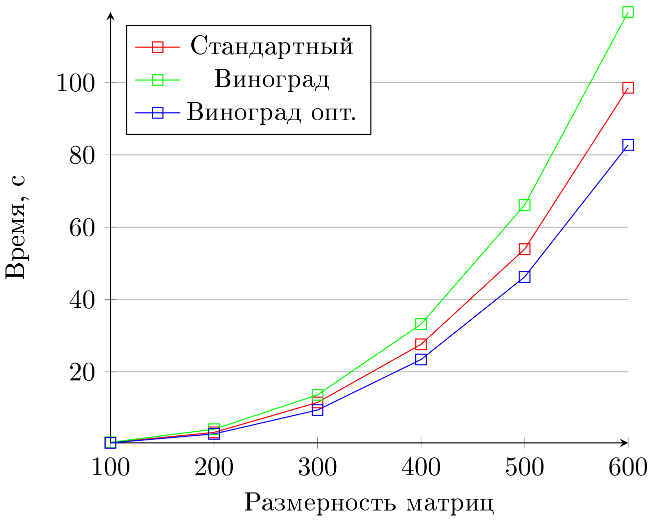

Рис. 5: сравнение времени на умножение матриц стандартным алгоритмом, алгоритмом Винограда, и оптимизированным алгоритмом Винограда при четных размерностях матриц

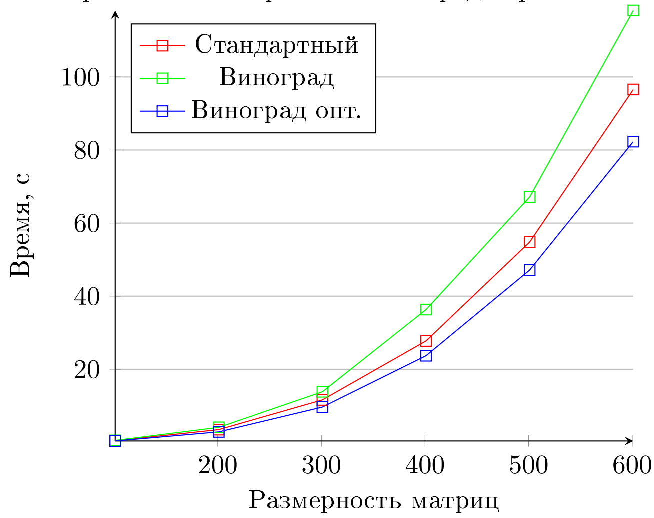

Рис. 6: сравнение времени на умножение матриц стандартным алгоритмом, алгоритмом Винограда, и оптимизированным алгоритмом Винограда при нечетных размерностях матриц

На рисунках 5 и 6 мы видим, что при алгоритм Винограда - самый медленный из трех. А наиболее быстрым является оптимизированный алгоритм Винограда.

4.4 Вывод

В данном разделе алгоритмы были рассмотрены на предмет правильности работы, что было показано на примерах из листингов 7-9. Все алгоритмы оказались верны. Также был произведен анализ по затрачиваемому времени на каждый из алгоритмов.

Заключение

В ходе лабораторной работы были исследованы алгоритмы умножения матриц: стандартный, Винограда, и оптимизированный алгоритм Винограда. Для каждого алгоритма была посчитана трудоемкость в выбранной модели вычислений. Помимо этого, экспериментально были произведены замеры времени работы каждого из рассматриваемых алгоритмов.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.