

Дисциплина: АНАЛИЗ АЛГОРИТМОВ

ЛАБОРАТОРНАЯ РАБОТА 2

Умножение матриц с помощью стандартного алгоритма и
алгоритма Винограда.

Студент группы ИУ7-55,
Аминов Тимур Саидович

Содержание

Введение	3
1 Аналитическая часть	4
1.1 Теоретические сведения об умножении матриц	4
1.2 Стандартный алгоритм умножения матриц	4
1.3 Алгоритм Винограда	4
1.4 Модель вычислений	5
1.5 Вывод	5
2 Конструкторская часть	6
2.1 Схемы алгоритмов	6
2.2 Вывод	9
3 Технологическая часть	10
3.1 Требования к программному обеспечению	10
3.2 Средства реализации	10
3.3 Листинг кода	10
3.4 Оптимизация алгоритма Винограда	12
3.5 Оценка трудоемкости	12
3.6 Вывод	13
4 Экспериментальная часть	14
4.1 Примеры работы программы	14
4.2 Постановка эксперимента	15
4.3 Результаты эксперимента	15
4.4 Вывод	16
Заключение	17
Список литературы	18

Введение

Умножение матриц - одна из самых используемых матричных операций. Самый простой с точки зрения 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 работы:

- 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 $m \times n$. Пусть имеются две матрицы: A и B размерами $n \times l$ и $l \times m$ 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 $n \times l$ и $l \times m$ соответственно называется матрица C размерами $n \times m$, каждый элемент которой вычисляется по формуле 1:

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

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

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

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

Это равенство можно переписать в виде

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \quad (3)$$

В Алгоритме Винограда используется скалярное произведение из формулы 3, 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 сокращение 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 лабораторной 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.

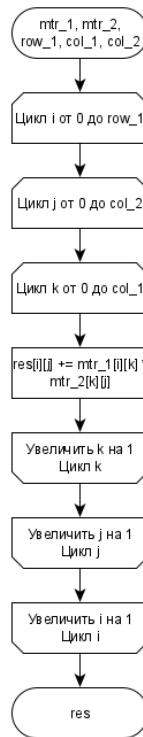


Рис. 1: Стандартный алгоритм

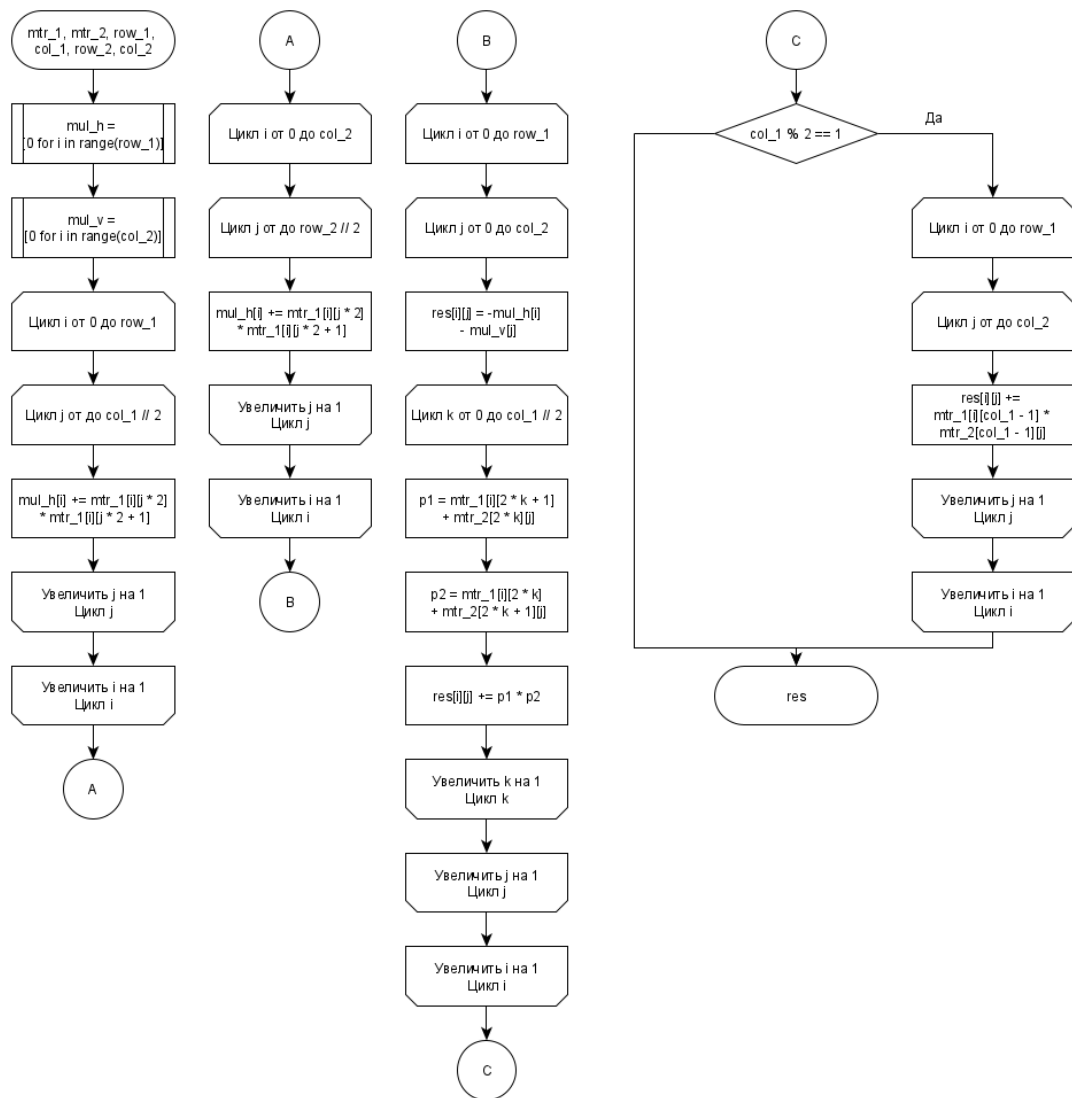


Рис. 2: Алгоритм Винограда

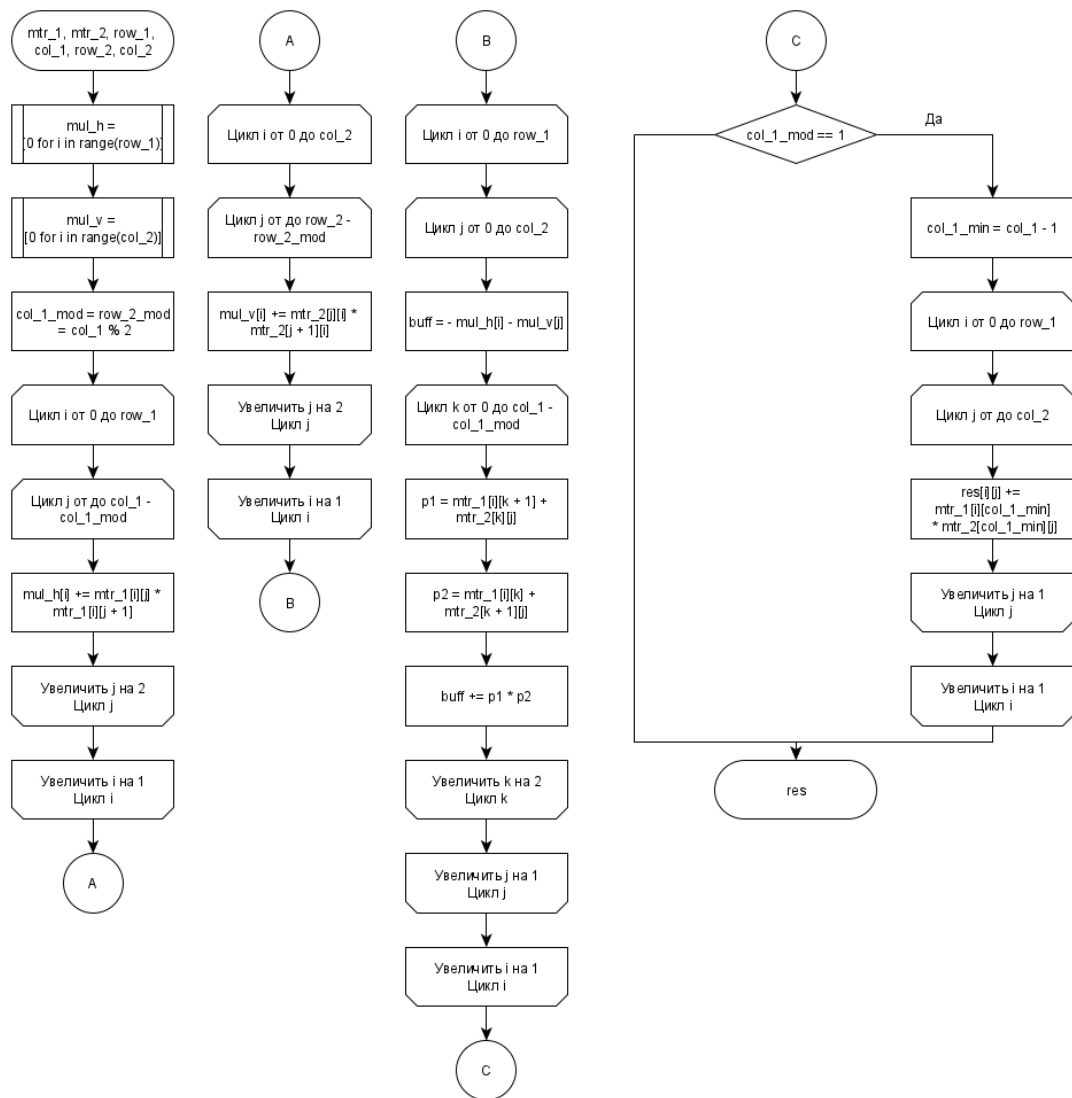


Рис. 3: Алгоритм Винограда(оптимизированный)

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.

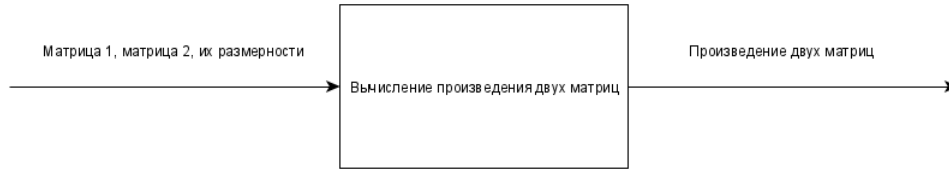


Рис. 4: IDEF0-диаграмма, описывающая алгоритм умножения матриц

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

```

1  def std_mpl(mtr_1, mtr_2):
2      row_1 = len(mtr_1)
3      col_1 = row_2 = len(mtr_1[0])
4      col_2 = len(mtr_2[0])
5
6      res = [[0 for j in range(col_2)] for i in range(row_1)]
7
8      for i in range(row_1):
9          for j in range(col_2):
10             for k in range(col_1):
11                 res[i][j] += mtr_1[i][k] * mtr_2[k][j]
12
13     return res

```

Листинг 2: Алгоритм Винограда

```

1  def winograd(mtr_1, mtr_2):
2      row_1 = len(mtr_1)
3      col_1 = row_2 = len(mtr_1[0])
4      col_2 = len(mtr_2[0])
5
6      res = [[0 for j in range(col_2)] for i in range(row_1)]
7
8      mul_h = [0 for i in range(row_1)]
9      mul_v = [0 for i in range(col_2)]
10
11     for i in range(row_1):
12         for j in range(col_1 // 2):
13             mul_h[i] += mtr_1[i][j * 2] * mtr_1[i][j * 2 + 1]
14
15     for i in range(col_2):
16         for j in range(row_2 // 2):
17             mul_v[i] += mtr_2[j * 2][i] * mtr_2[j * 2 + 1][i]
18
19     for i in range(row_1):
20         for j in range(col_2):
21             res[i][j] = -mul_h[i] - mul_v[j]
22             for k in range(col_1 // 2):
23                 res[i][j] += (mtr_1[i][2 * k + 1] + mtr_2[2 * k][j]) * \
24                     (mtr_1[i][2 * k] + mtr_2[2 * k + 1][j])
25
26     if col_1 % 2 == 1:
27         for i in range(row_1):
28             for j in range(col_2):
29                 res[i][j] += mtr_1[i][col_1 - 1] * mtr_2[col_1 - 1][j]
30
31     return res

```

Листинг 3: Оптимизированный алгоритм Винограда

```

1  def winograd_opt(mtr_1, mtr_2):
2      row_1 = len(mtr_1)
3      col_1 = row_2 = len(mtr_1[0])
4      col_2 = len(mtr_2[0])
5
6      col_1_mod = row_2_mod = col_1 % 2
7
8      res = [[0 for j in range(col_2)] for i in range(row_1)]
9
10     mul_h = [0 for i in range(row_1)]
11     mul_v = [0 for i in range(col_2)]
12
13     for i in range(row_1):
14         for j in range(0, col_1 - col_1_mod, 2):
15             mul_h[i] += mtr_1[i][j] * mtr_1[i][j + 1]
16
17     for i in range(col_2):
18         for j in range(0, row_2 - row_2_mod, 2):
19             mul_v[i] += mtr_2[j][i] * mtr_2[j + 1][i]
20
21     for i in range(row_1):
22         for j in range(col_2):
23             buff = - mul_h[i] - mul_v[j]
24             for k in range(0, col_1 - col_1_mod, 2):
25                 buff += (mtr_1[i][k + 1] + mtr_2[k][j]) * \
26                     (mtr_1[i][k] + mtr_2[k + 1][j])
27             res[i][j] = buff
28
29     if col_1_mod == 1:
30         col_1_min = col_1 - 1
31         for i in range(row_1):
32             for j in range(col_2):
33                 res[i][j] += mtr_1[i][col_1_min] * mtr_2[col_1_min][j]
34
35     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

```

1  for i in range(row_1):
2      for j in range(0, col_1 - col_1_mod, 2):
3          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

```

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

```

3.5 Оценка трудоемкости

Таблица 1. Трудоемкость стандартного алгоритма умножения матриц.

Трудоемкость	Оценка
Гтела	8
Гстанд	$2 + n(4 + m(4 + l(4 + \text{Гтела})))$
Гстанд	$10mln + 4mn + 4n + 2$

Таблица 2. Трудоемкость алгоритма Винограда.

Трудоемкость	Оценка
Гтела1	12
Гтела2	12
Гтела3	23
Гтела4	$9 + 1/2 * (3 + \text{Гтела3})$
Гтела5	13
Гусловия	$3 + n(4 + m(4 + \text{Гтела5}))$
Гвиноград(лучший случай)	$7 + 9n + 7nl + 5m + 7ml + 11nm + 13nml$
Гвиноград(худший случай)	$9 + 13n + 7nl + 5m + 7ml + 28nm + 13nml$

Таблица 3. Трудоемкость оптимизированного алгоритма Винограда.

Трудоемкость	Оценка
Гтела1	8
Гтела2	8
Гтела3	16
Гтела4	$7 + 1/2 * (3 + \text{Гтела3})$
Гтела5	8
Гусловия	$n(4 + m(4 + \text{Гтела5}))$
Гво(лучший случай)	$11 + 9n + 5nl + 5m + 5ml + 9nm + 9.5nml$
Гво(худший случай)	$12 + 13n + 5nl + 5m + 5ml + 21nm + 9.5nml$

Трудоемкость оценивается по самому быстрорастущему слагаемому, то есть $mln(\text{куб})$. 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. Исследования проводились на 64-битной Windows 10, на x64 процессоре Intel Core i7 с 16гб оперативной памяти.

4.1 Примеры работы программы

Листинг 6: Пример работы 1

```
1 Input filename of first matrix: mtr/in_1.txt
2 Input filename of second matrix: mtr/in_1.txt
3 First matrix:
4 1 2 3
5 4 5 6
6 7 8 9
7
8 Second matrix:
9 1 2 3
10 4 5 6
11 7 8 9
12
13 Result using standart method:
14 30 36 42
15 66 81 96
16 102 126 150
17
18 Result using Winograd:
19 30 36 42
20 66 81 96
21 102 126 150
22
23 Result using optimized Winograd:
24 30 36 42
25 66 81 96
26 102 126 150
```

В данном примере все алгоритмы дали верный результат.

Листинг 7: Пример работы 2

```
1 Input filename of first matrix: mtr/in_0.txt
2 Input filename of second matrix: mtr/in_0.txt
3
4 Empty matrix
```

В данном примере все алгоритмы дали верный результат.

Листинг 8: Пример работы 3

```
1 Input filename of first matrix: in_1.txt
2 Matrix 1:
3
4 1 2 3
5 4 5 6
6 7 8 9
7
8
9 Input filename of second matrix: in_2.txt
10 Matrix 2:
11
12 1 1 1
13 1 1 1
14 1 1 1
15
16
17 Result matrix(standart):
18
```

```

19 6 6 6
20 15 15 15
21 24 24 24
22
23 Result matrix(vinograd):
24
25 6 6 6
26 15 15 15
27 24 24 24
28
29 Result matrix(vinograd optimized):
30
31 6 6 6
32 15 15 15
33 24 24 24

```

В данном примере все алгоритмы дали верный результат.

В примерах, приведенных на листингах 6-8, 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.

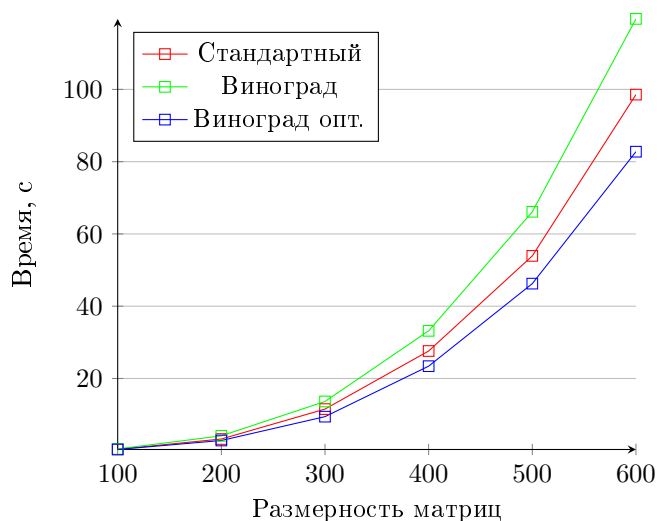


Рис. 5: сравнение времени на умножение матриц стандартным алгоритмом, алгоритмом Винограда, и оптимизированным алгоритмом Винограда при четных размерностях матриц

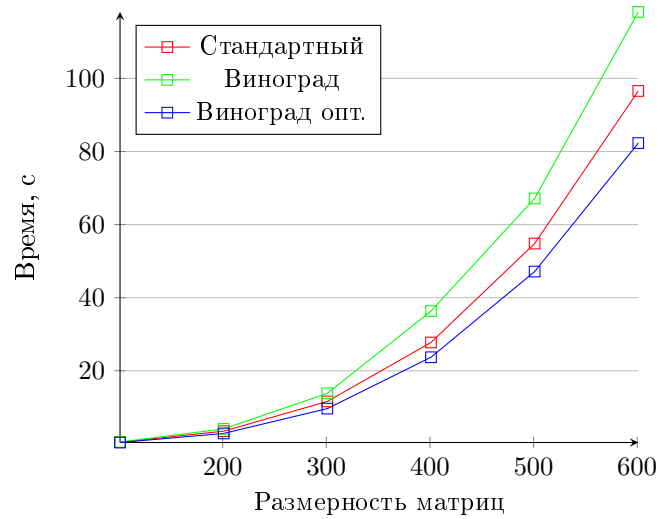


Рис. 6: сравнение времени на умножение матриц стандартным алгоритмом, алгоритмом Винограда, и оптимизированным алгоритмом Винограда при нечетных размерностях матриц

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. Основными отличиями алгоритма Винограда является наличие предварительных вычислений и сокращение количества умножений. Для каждого алгоритма была посчитана трудоемкость в выбранной модели вычислений. Помимо этого, экспериментально были произведены замеры времени работы каждого из рассматриваемых алгоритмов. Алгоритм Винограда оказался самым медленным, затем по скорости работы идет стандартный алгоритм, а самым быстрым является оптимизированный алгоритм Винограда. В среднем алгоритм Винограда медленнее стандартного на 23%, а стандартный медленнее оптимизированного алгоритма Винограда на 18%. Алгоритм Винограда является самым трудоемким, оптимизированный алгоритм Винограда - самым быстрым, а стандартный алгоритм расположился между ними.

Список литературы

- [1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.
- [2] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16
- [3] Le Gall, F. (2012), "Faster algorithms for rectangular matrix multiplication Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 514–523
- [4] Язык программирования python // [Электронный ресурс]. Режим доступа: <https://www.python.org/> (дата обращения: 10.09.19).
- [5] Библиотека для замера времени timeit // [Электронный ресурс]. Режим доступа: <https://docs.python.org/2/library/timeit.html> (дата обращения: 10.09.19).