Государственное образовательное учреждение высшего профессионального образования

"Московский государственный технический университет имени Н.Э.Баумана"

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №2
курса «Анализ алгоритмов»
на тему «Алгоритмы умножения матриц и расчет эффективности алгоритмов»

Студент: Ильясов И.М., группа ИУ7-53Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Содержание

Введение

Целью данной лабораторной работы является изучение расчета трудоемкости алгоритмов и методов оптимизации алгоритмов на примере умножения матриц стандартным алгоритмом и с помощью алгоритма Винограда. Для достижения поставленной задачи необходимо решить следующие задачи:

- изучение алгоритмов умножения матриц стандартным алгоритмом и с помощью алгоритма Винограда;
- получение навыков расчета трудоемкости алгоритмов;
- получение навыков оптимизации алгоритмов с помощью применения не менее трех видов оптимизации к алгоритму Винограда;
- получение практических навыков реализации указанных алгоритмов: двух алгоритмов в версии без оптимизации и одного с применением оптимизации;
- сравнительный анализ неоптимизированной и оптимизированной реализаций алгоритма Винограда по затрачиваемым ресурсам (времени и памяти);
- описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе содержатся теоретическое описание алгоритмов и указание области их применения.

1.1 Описание алгоритмов

Матрица — математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Если число столбцов в первой матрице совпадает с числом строк во второй, то эти матрицы можно перемножить. Произведением матриц $A(M * N)$ и $B(N * Q)$ является такая матрица $C(M * Q)$, что каждый ее элемент рассчитывается по формуле:

$$c_{ij} = \sum_{r=1}^{n} (a_{ir} * b_{rj})$$

Реализация умножения матриц стандартным алгоритмом имеет трудоемкость порядка $O(M * N * Q)$. Существую алгоритмы умножения матриц, снижающие трудоемкость. Одним из таких является алгоритм Винограда. Особенность этого алгоритма заключается в том, что процесс умножения отдельных векторов преобразуется по формуле:

$$\begin{aligned} U * V = u_1 * v_1 + u_2 * v_2 + u_3 * v_3 + u_4 * v_4 = \\ = (u_1 + u_2) * (v_1 + v_2) + (u_3 + u_4) * (v_3 * v_4) - \\ - u_1 * u_2 - u_3 * u_4 - v_1 * v_2 - v_3 * v_4 \end{aligned}$$

При этом для каждых строки и столбца значения $u_i * u_{i+1}, v_i * v_{i+1}$ остаются неизменными, что позволяет вычислить их заранее. Оцененная трудоемкость такого алгоритма составляет порядка $O(N^{2.3727})$ для квадратных матриц [3].

Умножение матриц применяется во множестве задач, например:

- компьютерной графике;
- физике;
- машинном обучении.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов и теоретический расчет их трудоемкости.

2.1 IDEF0-диаграмма

Рисунок 1. IDEF0-диаграмма алгоритма вычисления произведения матриц

2.2 Разработка алгоритмов

Ниже представлены схемы алгоритмов умножение матриц стандартным алгоритмом (рис. 2) и по Винограду (рис. 3)

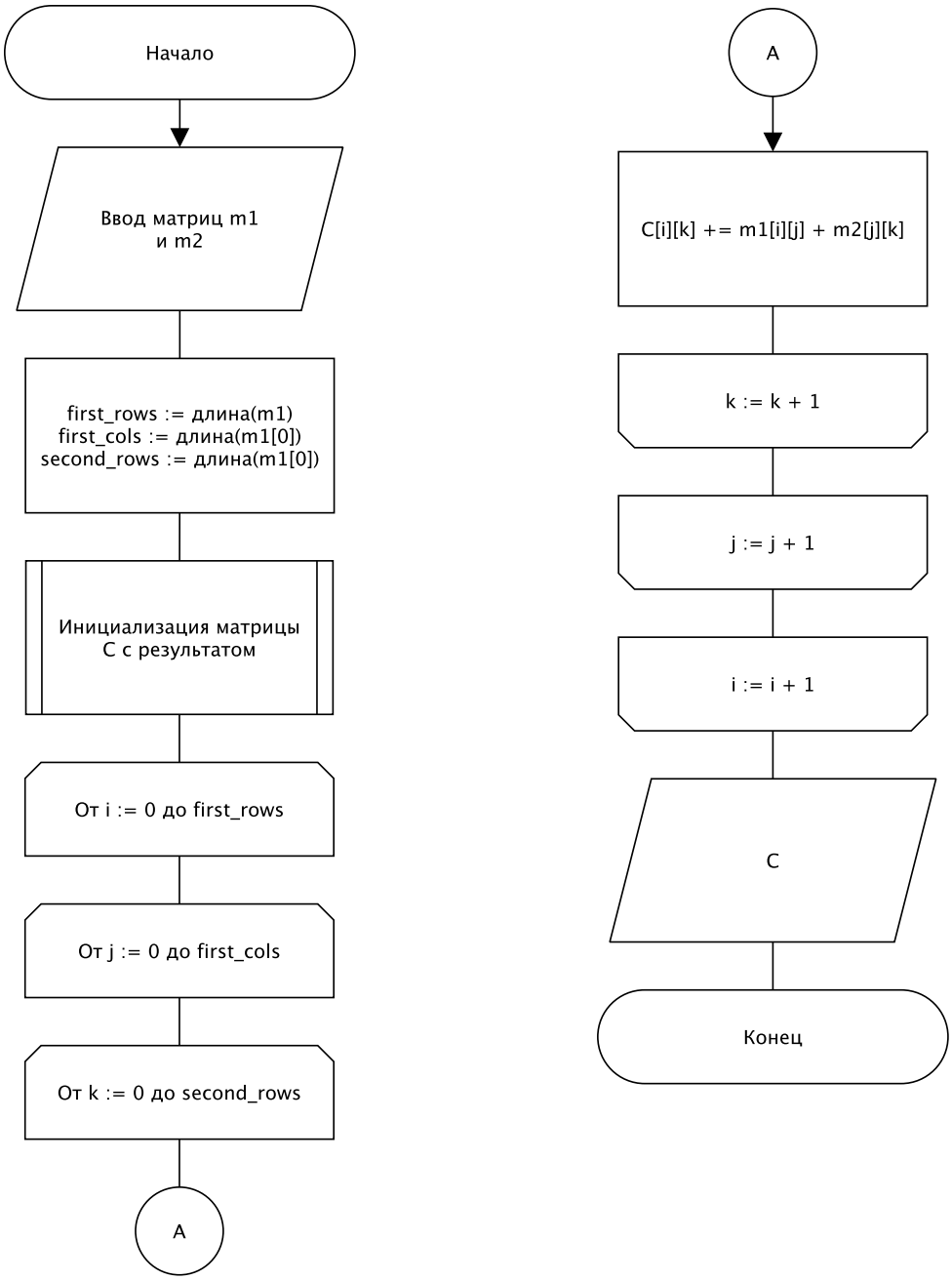

Рисунок 2: схема стандартного алгоритма вычисления произведения матриц

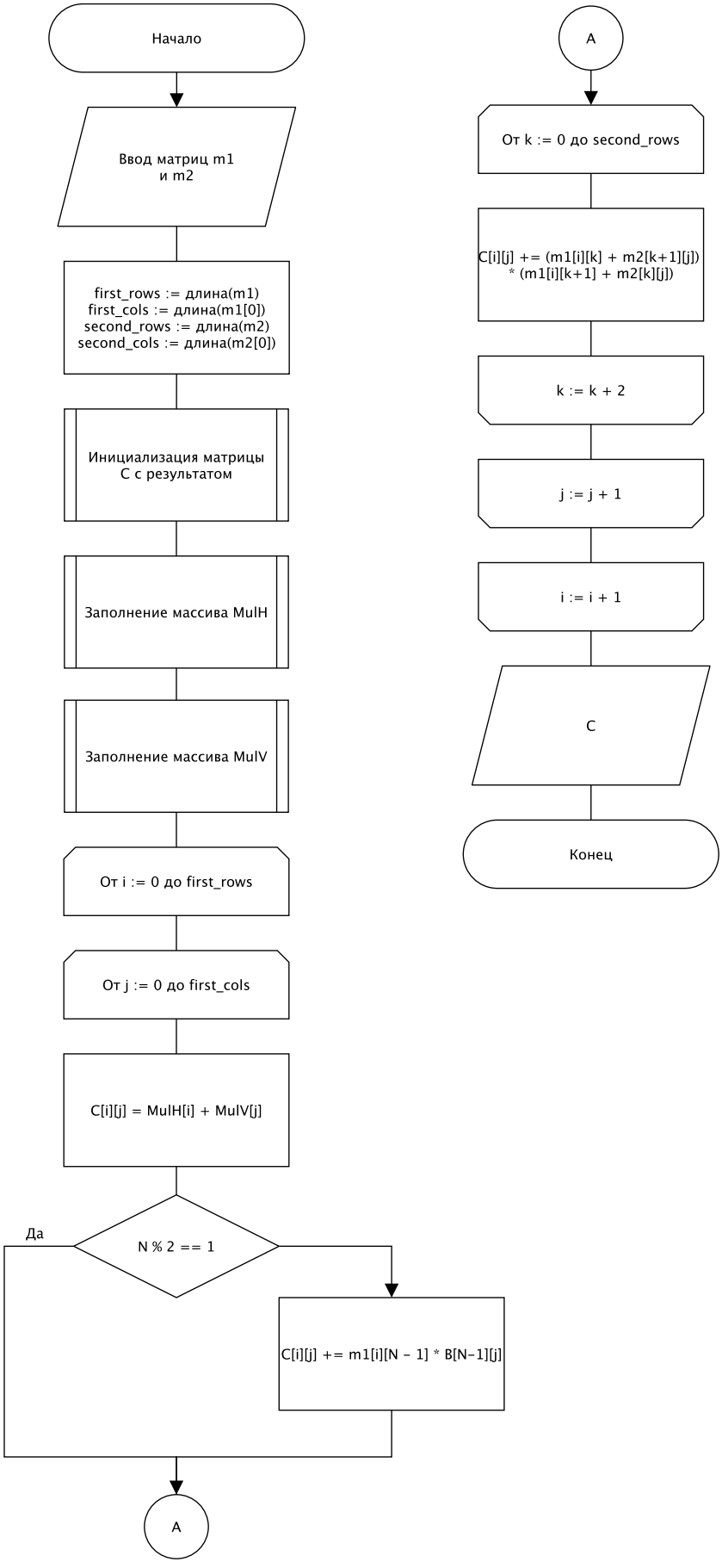

Рисунок 3: схема алгоритма Винограда вычисления произведения матриц

2.3 Расчет трудоемкости

В этом пункте аналитически выполнен расчет трудоемкости исследуемых алгоритмов. В используемой модели расчета трудоемкости за единицу считаются операции $+, -, *, /, =, [], >, <, ==, <=, >=, ++, --, +=, -=$.

2.3.1 Стандартный алгоритм умножения

Стандартный алгоритм выглядит так:

```
for (i = 0; i < m; i++){
    for (j = 0; j < q; j++){
        for (k = 0; k < n; k++){
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

Вычисление трудоёмкости:

$$f = 2 + m * (4 + q * (4 + 10 * n)) = 2 + 4 * m * q + 10 * m * q * n$$

2.3.2 Алгоритм Винограда

Алгоритм разбивается на 4 части. Ниже приведен псевдокод этих частей:

```
for (i = 0; i < m; i++){
    for (j = 0; j < n / 2; j++){
        mulU[i] = mulU[i] + a[i][2 * j] * a[i][2 * j + 1];
    }
}

for (i = 0; i < m; i++){
    for (j = 0; j < n / 2; j++){
        mulV[i] = mulV[i] + b[2 * j][i] * b[2 * j + 1][i];
    }
}

for (i = 0; i < m; i++){
    for (j = 0; j < q; j++){
        c[i][j] = - mulU[i] - mulV[j];
        for (k = 0; k < (n - 1) / 2; k++){
            c[i][j] = c[i][j] + (a[i][2 * k] + b[2 * k + 1][j]) * (a[i][2
               * k + 1] + b[2 * k][j]);
        }
    }
}
```

```
if (n % 2 == 1){
    for (i = 0; i < m; i++){
        for (j = 0; j < q; j++){
            c[i][j] = c[i][j] + a[i][n - 1] * b[n - 1][j];
        }
    }
}
```

Вычисление трудоёмкости отдельных частей:

$$f_1 = 2 + m * (5 + 15 * n/2) = 15/2 * m * n + 5 * m + 2,$$

$$f_2 = 2 + q * (5 + 15 * n/2) = 15/2 * q * n + 5 * q + 2,$$

$$f_3 = 2 + m * (4 + q * (12 + 13 * n)) = 2 + 4 * m + 12 * m * q + 27/2 * m * n * q,$$

$$f_4 = 2 + \begin{cases} 0 \\ 15 * m * q + 4 * m + 4 \end{cases}$$

Суммарная трудоемкость алгоритма Винограда равна:

$$f = f_1 + f_2 + f_3 + f_4 = 27/2 * m * n * q + 15/2 * m * n +$$
$$+ 15/2 * q * n + 12 * m * q + 9 * m + 5 * q +$$
$$+ 8 + \begin{cases} 0 \\ 15 * m * q + 4 * m + 4 \end{cases}$$

Трудоемкость данного алгоритма можно уменьшить, проведя следующие оптимизации:

- ввести операции $+=$, $-=$;
- $j < N, j += 2$ вместо $j < N/2;\ j++$;
- в частях 1 и 2 вычислять значения сразу как отрицательные;
- объединить части 3 и 4 в один цикл.

Согласно листингу трудоемкость составит:

$$f = 9 * m * n * q + 9/2 * m * n + 9/2 * q * n + 13 * m * q +$$
$$+ 8 * m + 4 * q + 6 * m * q * \begin{cases} 0 \\ 10 \end{cases}$$

3 Технологическая часть

В данном разделе представлены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также листинг кода программы.

3.1 Требования к программному обеспечению

Программа должна умножать введенные в два текстовых поля матрицы целых чисел. Ввод осуществляется построчно, числа разделяются переносом строки. программа должна выдавать результат умножения в виде трех таблиц для алгоритмов по определению, Винограда и Винограда с оптимизацией. Про грамма должна выводить в виде таблицы результаты замеров времени.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования Python. Для измерения процессорного времени была использован метод time.process_time().

3.3 Листинг кода

В приведенных ниже листингах представлены реализации стандартного алгоритма (листинг 1), алгоритма Винограда (листинг 2) и оптимизированного алгоритма Винограда (листинг 3).

Листинг 1: Стандартный алгоритм:

```
def multiplication_classic(first_matr, second_matr):
    first_rows, first_cols = len(first_matr), len(first_matr[0])
    second_rows, second_cols = len(second_matr), len(second_matr[0])
    result_matr = [[0 for i in range(second_cols)] for j in range(first_r
ows)]

    for i in range(first_rows):
        for j in range(first_cols):
            for k in range(second_rows):
                result_matr[i][k] += first_matr[i][j] * second_matr[j][k]

    return result_matr
```

Листинг 2: Алгоритм Винограда:

```
def multiplication_winograd(first_matr, second_matr):
    first_rows, first_cols = len(first_matr), len(first_matr[0])
    second_rows, second_cols = len(second_matr), len(second_matr[0])

    rows_factors = [0 for i in range(first_rows)]
    cols_factors = [0 for i in range(first_cols)]

    for i in range(first_rows):
        for j in range(second_rows // 2):
            rows_factors[i] = rows_factors[i] + first_matr[i]
                              [2 * j] * first_matr[i][2 * j + 1]

    for i in range(second_cols):
        for j in range(second_rows // 2):
            cols_factors[i] = cols_factors[i] + second_matr[2 * j]
                              [i] * second_matr[2 * j + 1][i]

    result_matr = [[0 for i in range(second_cols)] for j in range(first_r
ows)]

    for i in range(first_rows):
        for j in range(second_cols):
            result_matr[i][j] = -rows_factors[i] - cols_factors[j]
            for k in range(second_rows // 2):
                result_matr[i][j] = result_matr[i][j] + (first_matr[i]
                                    [2 * k] + second_matr[2 * k + 1][j]) *
                                    (first_matr[i][2 * k + 1] +
                                     second_matr[2 * k][j])

    if (second_rows % 2 == 1):
        for i in range(first_rows):
            for j in range(second_rows):
                result_matr[i][j] = result_matr[i][j] + first_matr[i]
                      [second_rows - 1] * second_matr[second_rows - 1][j]

    return result_matr
```

Листинг 3: Оптимизированный алгоритм Винограда:

```
def multiplication_winograd_modified(first_matr, second_matr):
    first_rows, first_cols = len(first_matr), len(first_matr[0])
    second_rows, second_cols = len(second_matr), len(second_matr[0])

    half = second_rows // 2
    rows_factors = [0 for i in range(first_rows)]
    cols_factors = [0 for i in range(first_cols)]
    is_even = second_rows % 2 == 0

    for i in range(first_rows):
        for j in range(half):
            rows_factors[i] += first_matr[i][2 * j] * first_matr[i]
                               [2 * j + 1]

    for i in range(second_cols):
        for j in range(half):
            cols_factors[i] += second_matr[2 * j]
                           [i] * second_matr[2 * j + 1][i]

    result_matr = [[0 for i in range(second_cols)] for j in range(first_r
ows)]

    for i in range(first_rows):
        for j in range(second_cols):
            result_matr[i][j] = -rows_factors[i] - cols_factors[j]
            for k in range(half):
                result_matr[i][j] += ((first_matr[i]
                                   [2 * k] + second_matr[2 * k + 1][j]) * \
                                   (first_matr[i][2 * k + 1] +
                                    second_matr[2 * k][j]))
            if not is_even:
                result_matr[i][j] += first_matr[i][second_rows -
 1] * second_matr[second_rows - 1][j]

    return result_matr
```

3.4 Вывод

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка Python было разработано программное обеспечение, содержащее реализации выбранных алгоритмов.

4 Экспериментальная часть

В данном разделе будут приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Примеры работы

На приведенных ниже рисунках продемонстрирована работа выбранных алгоритмов.

```
Введите количество строк и столбцов в первой матрице (через пробел): 2 2
1
2
3
4

Введите количество строк и столбцов во второй матрице (через пробел): 2 2
4
3
2
1

Классический алгоритм:
  8   5
 20  13

Алгоритм Винограда:
  8   5
 20  13

Оптимизированный алгоритм Винограда:
  8   5
 20  13
```

Рисунок 4: пример работы алгоритмов.

4.2 Результаты тестирования

Таблица 1:

Результаты замеров времени трех реализаций вычисления произведения двух матриц (четная размерность)

Размерность	Стандартный	Виноград	О. Виноград
100x100	379.85	464.518	465.424
200x200	3037.612	3690.618	3516.794
300x300	10436.786	12605.01	12033.244
400x400	25279.292	30489.528	23036.235
500x500	50142.664	60080.646	58446.04
600x600	87241.76	109012.434	106290.286
700x700	139247.336	175056.638	170466.26
800x800	208601.312	265169.78	258311.41
900x900	302900.482	382402.56	373681.942
1000x1000	407751.364	527054.144	515772.02

Таблица 2:

Результаты замеров времени трех реализаций вычисления произведения двух матриц (нечетная размерность)

Размерность	Стандартный	Виноград	О. Виноград
101x101	415.866	498.698	478.092
201x201	3068.734	3771.998	3599.72
301x301	10517.338	12584.98	12125.68
401x401	25468.874	30544.894	29887.316
501x501	50398.874	60741.752	59757.796
601x601	87715.196	109798.24	105175.302
701x701	140946.786	175653.384	169783.778
801x801	209003.472	265568.1	258919.572
901x901	297690.976	383532.074	375971.84
1001x1001	407892.496	530497.83	517814.592

4.3 Постановка эксперимента по замеру времени

На рис. 5 представлены результаты замеров для варьирующихся размерностей матриц от 100x100 до 1000x1000 с шагом 100 и на рис. 6 — для размерностей матриц от 101x101 до 1001x1001 с шагом 100. Один эксперимент ставился 5 раз, вычислялось среднее время работы.

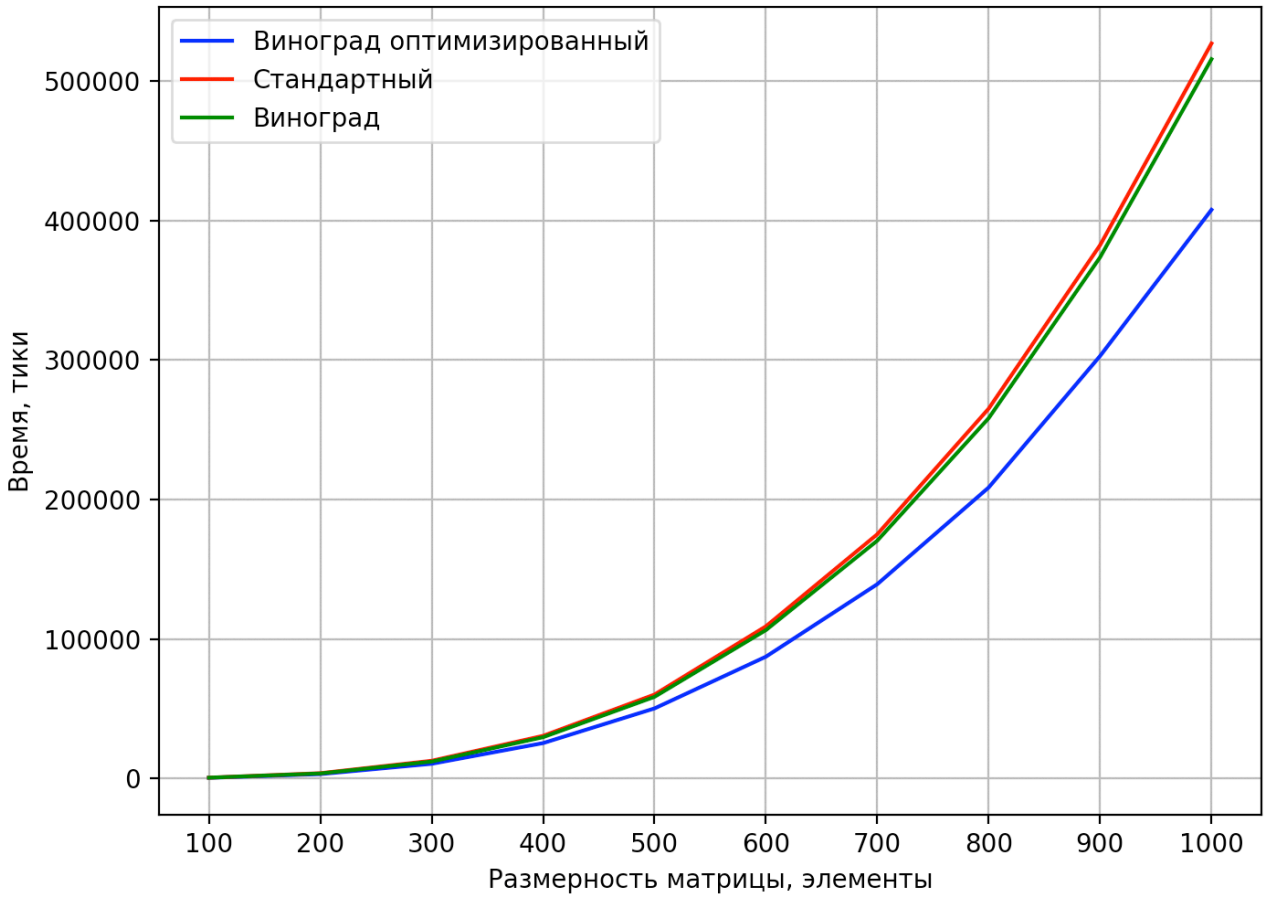

Рисунок 5: график зависимости времени работы стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда от четной размерности матриц.

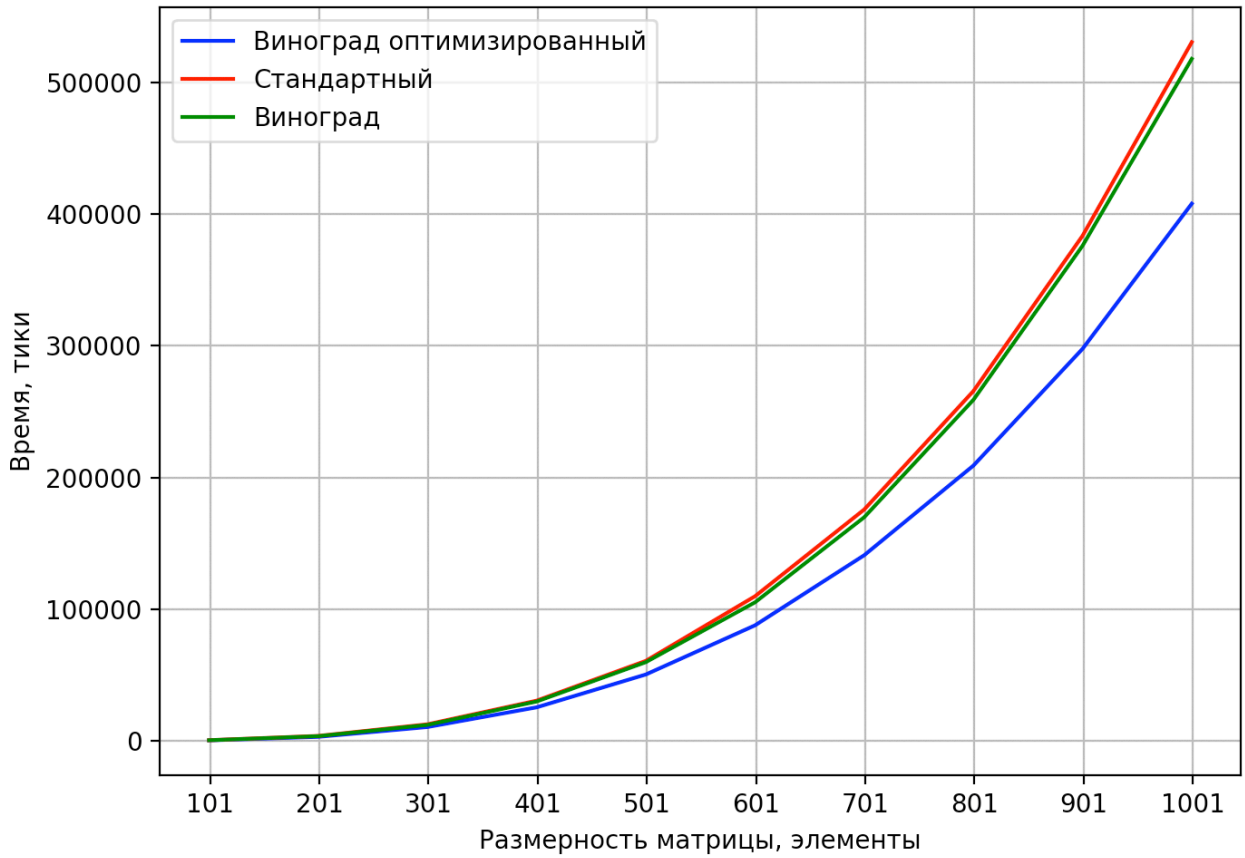

Рисунок 6: график зависимости времени работы стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда от нечетной размерности матриц.

4.4 Вывод

По графикам рис. 5, 6 видно, что умножение матриц стандартным алгоритмом проигрывает в скорости алгоритму Винограда. Это означает, что алгоритм Винограда эффективнее умножения стандартным алгоритмом. Однако данный результат проявляется только на больших размерах матриц. Также видно, что оптимизация алгоритма Винограда дает небольшой прирост эффективности алгоритма, но этот прирост может быть более заметен при использовании более низкоуровневых языков программирования.

Заключение

В процессе выполнения лабораторной работы было проведено исследование стандартного алгоритма умножения матриц и алгоритма Винограда. Во время разработки программного обеспечения в соответствии с поставленными требованиями были получены практические навыки реализации указанных алгоритмов. При помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся размерах матриц были экспериментально подтверждены различия во временной эффективности стандартного алгоритма вычисления произведения матриц, алгоритма Винограда и оптимизированной версии этого алгоритма.

Список литературы

1. Корн Г., Корн Т. - "Алгебра матриц и матричное исчисление // Справочник по математике. — 4-е издание. — М: Наука, 1978. — С. 392—394"
2. Williams, Virginia - "Breaking the Coppersmith-Winograd barrier"