МГТУ им. Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Алгоритмы умножения матриц

Работу выполнил: Мокеев Даниил, ИУ7-54

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

Москва, 2019

Оглавление

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.

В ходе лабораторной работы предстоит:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- оптимизировать алгоритм Винограда;
- дать теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда;
- реализовать три алгоритма умножения матриц на одном из языков программирования;
- сравнить алгоритмы умножения матриц.

1 Аналитическая часть

Матрицей А размера $[m * n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы.[1] 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.

Пусть даны две прямоугольные матрицы А и В размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц А и В получим матрицу С размера $[m * k]$.

$c_{i,j} = \sum\limits_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц А и В [1].

1.1 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 [2].

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)

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

Менее очевидно, что 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.

1.2 Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

Требования к вводу: На вход подаются две матрицы
Требования к программе:

- корректное умножение двух матриц;
- при матрицах неправилыных размеров программа не должна аварийно завершаться.

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов.

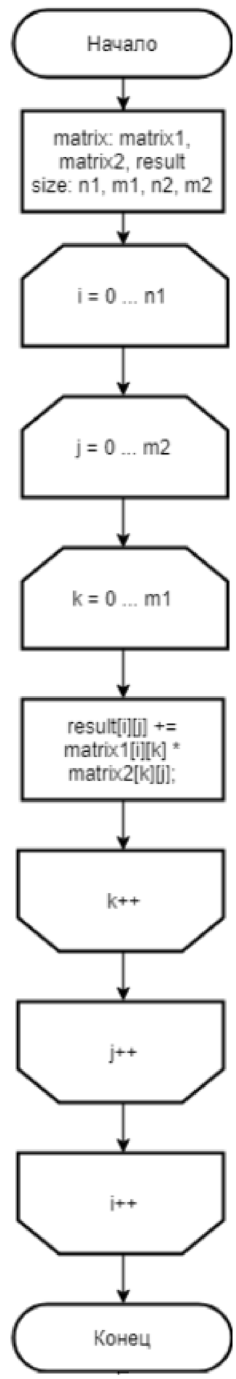

Рис. 2.1: Схема классического алгоритма умножения матриц

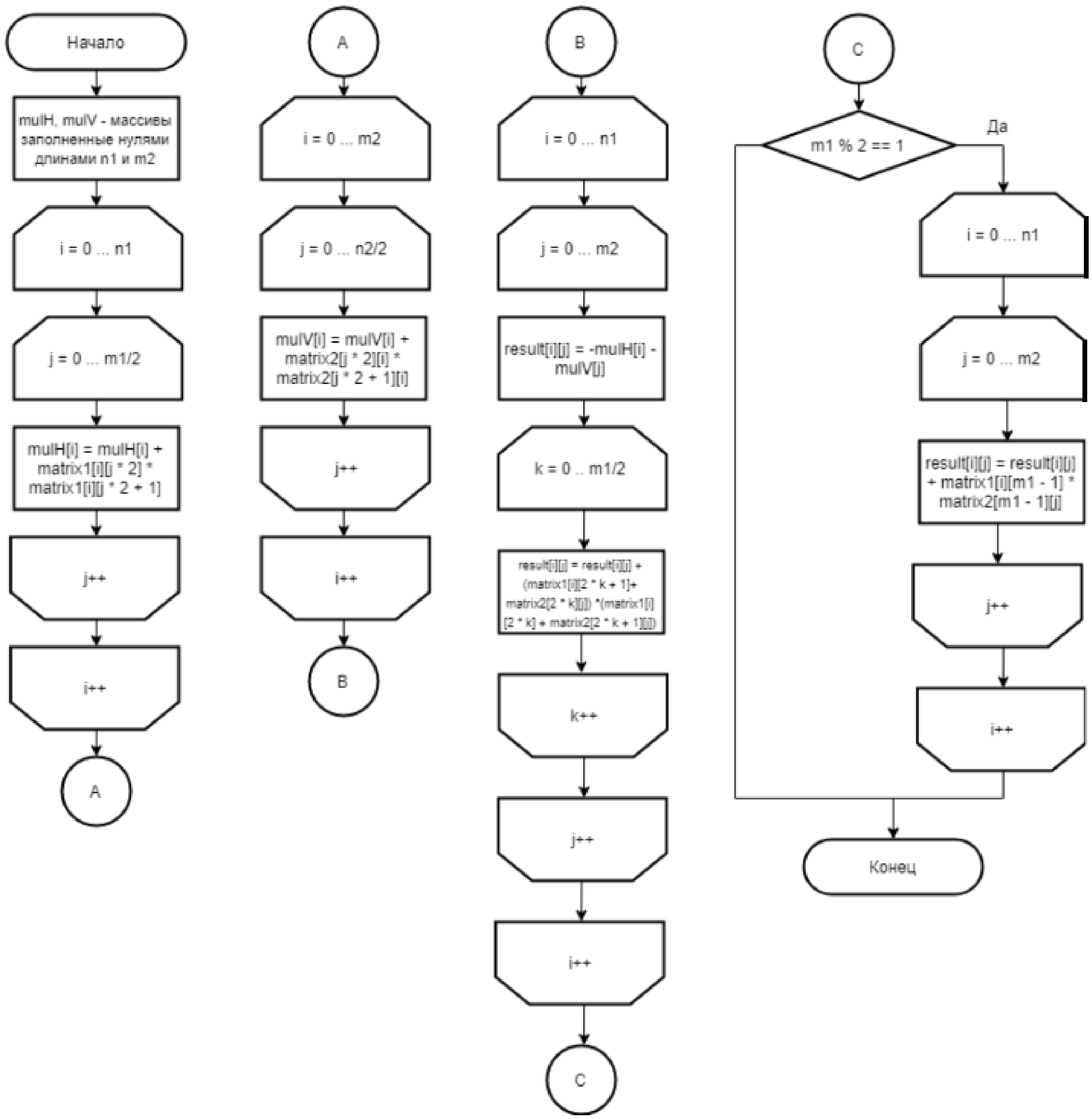

Рис. 2.2: Схема алгоритма Винограда

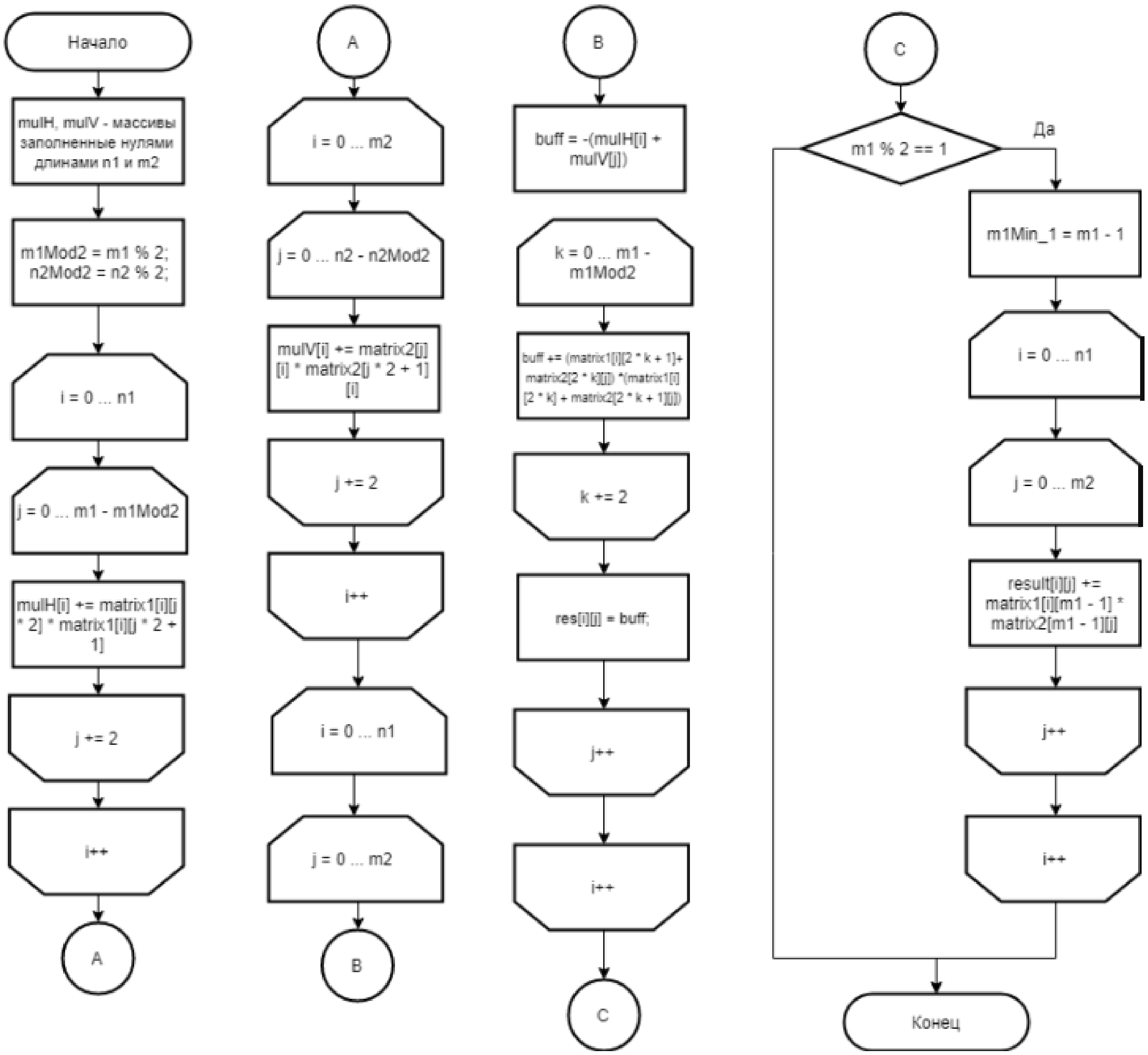

Рис. 2.3: Схема оптимизированного алгоритма Винограда

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса
- оценка трудоемкости цикла: Fц = init + N*(a + Fтела + post) + a, где a - условие цикла, init - предусловие цикла, post - постусловие цикла
- стоимость условного перехода применим за 0, стоимость вычисления условия остаётся

Оценим трудоемкость алгоритмов по коду программы.

2.2.1 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма:

Инициализация матрицы результата: $1+1+n_1(1+2+1)+1 = 4n_1+3$

Подсчет:

$1 + n_1(1 + (1 + m_2(1 + (1 + m_1(1 + (8) + 1) + 1) + 1) + 1) + 1) + 1 = n_1(m_2(10m_1 + 4) + 4) + 4) + 2 = 10n_1m_2m_1 + 4n_1m_2 + 4n_1 + 2$

2.2.2 Алгоритм Винограда

Аналогично рассмотрим 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 переход (в m2 нечетное количество строк):

$$\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$$

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

$$\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$$

2.2.3 Оптимизированный алгоритм Винограда

Аналогично Рассмотрим 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 переход (в m2 нечетное количество строк):

$$\begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$$

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

$$\begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$$

2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель оценки трудоемкости алгоритма, были расчитаны трудоемкости алгоритмов в соответсвии с этой моделью.

3 | Технологическая часть

3.1 Выбор ЯП

Я выбрал в качестве Python языком программирования, потому как он достаточно удобен и гибок.

Время работы алгоритмов было замерено с помощью функции time() из библиотеки time.

3.2 Описание структуры ПО

В данном разделе будет представленна функциональная схема программы (Рис. 3.1.)

Рис. 3.1: Функциональная схема умножения матриц (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- lab02.py - главный файл программы, в котором располагается точка входа в программу и функция замера времени.

3.4 Листинг кода алгоритмов

В данном разделе будет представлен листинги кода стандартного умножения матриц (3.1), алгоритма Винограда (3.2), оптимизированный алгоритм Винограда

Листинг 3.1: Стандартный алгоритм умножения матриц

```
def std(mtr1, mtr2):
  if len(mtr2) != len(mtr1[0]):
    print("Wrong size of matrix")
    return

  row1 = len(mtr1); col1 = len(mtr1[0])
  col2 =  len(mtr2[0])

  res = [[0 for i in range(col2)] for j in range(row1)]
  for i in range(row1):
    for j in range(col1):
      for k in range(col2):
        res[i][k] += mtr1[i][j] * mtr2[j][k]
  return res
```

Листинг 3.2: Алгоритм Винограда

```
def winograd(mtr1, mtr2):
  row1 = len(mtr1)
  row2 = len(mtr2)
  col2 = len(mtr2[0])

  if row2 != len(mtr1[0]):
    print("Different dimension of the matrics")
    return
  d = row2 // 2
```

```
  row_factor = [0 for i in range(row1)]
    col_factor = [0 for i in range(col2)]

  for i in range(row1):
    for j in range(d):
      row_factor[i] += mtr1[i][2 * j] * mtr1[i][2 * j + 1]

  for i in range(col2):
    for j in range(d):
      col_factor[i] += mtr2[2 * j][i] * mtr2[2 * j + 1][i]

  answer = [[0 for i in range(col2)] for j in range(row1)]
  for i in range(row1):
    for j in range(col2):
      answer[i][j] = - row_factor[i] - col_factor[j]
      for k in range(d):
        answer[i][j] += ((mtr1[i][2 * k] + mtr2[2 * k + 1][j]) *\
        (mtr1[i][2 * k + 1] + mtr2[2 * k][j]))

  if row2 % 2:
    for i in range(row1):
      for j in range(col2):
        answer[i][j] += mtr1[i][b - 1] * mtr2[b - 1][j]

  return answer
```

Листинг 3.3: Оптимизированный алгоритм Винограда

```
def imp_winograd(mtr1, mtr2):
  row1 = len(mtr1)
  row2 = len(mtr2)
  col2 = len(mtr2[0])

  if row2 != len(mtr1[0]):
    print("Different dimension of the matrics")
    return

  d = row2 // 2

  row_factor = [0 for i in range(row1)]
```

```
col_factor = [0 for i in range(col2)]

for i in range(row1):
  row_factor[i] = sum(mtr1[i][2 * j] * mtr1[i][2 * j + 1]
      for j in range(d))

for i in range(col2):
  col_factor[i] = sum(mtr2[2 * j][i] * mtr2[2 * j + 1][i]
      for j in range(d))

answer = [[0 for i in range(col2)] for j in range(row1)]
for i in range(row1):
  for j in range(col2):
    answer[i][j] = sum((mtr1[i][2 * k] + mtr2[2 * k + 1][
       j]) * (mtr1[i][2 * k + 1] + mtr2[2 * k][j]) for k
       in range(d))\
    - row_factor[i] - col_factor[j]

if row2 % 2:
  for i in range(row1):
    answer[i][j] = sum(mtr1[i][row2 - 1] * mtr2[row2 -
       1][j] for j in range(col2))

return answer
```

3.4.1 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 3 оптимизации:

1. Избавление от деления в условии цикла;

Листинг 3.4: Оптимизации алгоритма Винограда №1 и №2

```
    for i in range(row1):
    row_factor[i] = sum(mtr1[i][2 * j] * mtr1[i][2 * j
        + 1] for j in range(d))

  for i in range(col2):
    col_factor[i] = sum(mtr2[2 * j][i] * mtr2[2 * j +
        1][i] for j in range(d))
```

2. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти.

Листинг 3.5: Оптимизации алгоритма Винограда №3

```
    for i in range(row1):
    for j in range(col2):
      answer[i][j] = sum((mtr1[i][2 * k] + mtr2[2 * k +
          1][j]) * (mtr1[i][2 * k + 1] + mtr2[2 * k][j
          ]) for k in range(d))\
      - row_factor[i] - col_factor[j]
```

3.5 Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы.

4 Исследовательская часть

Был проведен замер времени работы каждого алгоритма.

4.1 Примеры работы

В данном разделе будет представлен результат работы программы (Рис.4.1
)

```
=== RESTART: C:\Users\danil\Desktop\analysis_of_algorithms\lab02\lab02.py ===
A = [[1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6]],
 B = [[1, 2, 3, 4, 5], [2, 3, 4, 5, 6], [3, 4, 5, 6, 7], [4, 5, 6, 7, 8]]

Стандартный алгоритм умножения A*B [[30, 40, 50, 60, 70], [40, 54, 68, 82, 96], [50, 68, 86, 104, 122]]

Алгоритм Винограда A*B [[30, 40, 50, 60, 70], [40, 54, 68, 82, 96], [50, 68, 86, 104, 122]]

Отпимизированный алгоритм Винограда A*B [[30, 40, 50, 60, 70], [40, 54, 68, 82, 96], [50, 68, 86, 104, 122]]

```

Рис. 4.1: Пример работы программы

4.2 Постановка эксперемента

Проведем сравнение для каждого из алгоритмов. Для замера времени будем использовать функцию time.

4.2.1 Лучший случай

Временные замеры для квадратных матриц с нечетным размером матриц размером size * size (Рис. 4.2)

size	Стандартный	Виноград	Оптимизированный Виноград
100	0.5494	0.69569	0.56495
200	4.15013	4.66497	4.54574
1 300	14.72735	16.41758	14.99365
400	34.76485	40.25307	36.63559
500	0.08998	0.10028	0.16667

Рис. 4.2: Временные замеры для лучшего случая

4.2.2 Худший случай

Временные замеры для квадратных матриц с нечетным размером матриц размером size * size (Рис. 4.3)

size	Стандартный	Виноград	Оптимизированный Виноград
101	0.59867	0.68139	0.61839
201	4.61544	5.01175	4.46456
301	14.79835	16.28355	14.74381
401	35.3947	40.06329	37.95568
501	72.95269	83.25553	72.76217

Рис. 4.3: Временные замеры для худшего случая

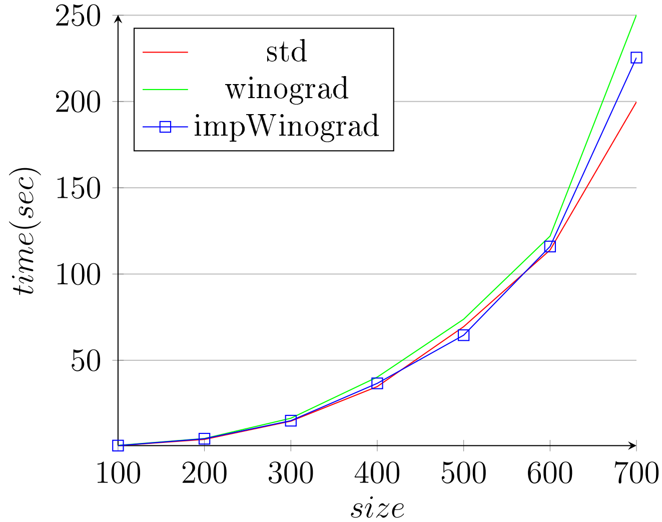

Рис. 4.4: Сравнение алгоритмов на матрицах четных размеров

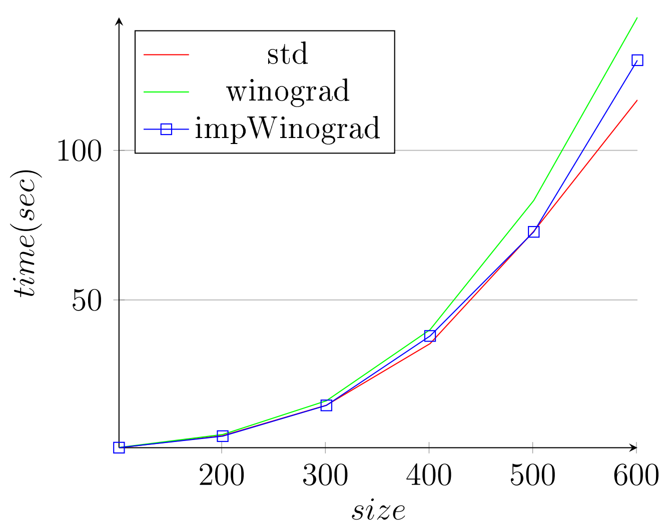

Рис. 4.5: Сравнение алгоритмов на матрицах четных размеров

4.2.3 Выводы экспериментальной части

Были протестированы различные алгоритмы умножения матриц. По результатам эксперимента алгоритм Винограда показывает худшие временные показатели. Стандартный и оптимизированный алгоритм Винограда сравнимы по скорости на небольших размерах матриц.

Заключение

В ходе выполнения данной лабораторной работы были реализованы три алгоритма умножения матриц: стандартный, алгоритм Винограда, оптимизированный алгоритм Винограда. Был проведён анализ каждого алгоритма и измерено время работы алгоритмов для матриц разных размеров. Была оценена трудоёмкость алгоритмов.

Лучшее время работы все сортировки показывают стандартный алгоритм умножения и оптимизированный алгоритм Винограда. Это объясняется тем, что замеры времени были проведены на относительно небольших размерах матриц. При умножении матриц больших размеров, алгоритм Винограда будет эффективнее обычного алгоритма умножения.

Литература

[1] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[2] 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

[3] Руководство по языку C#[Электронный ресурс], - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/