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»

Отчет по лабораторной работе №2 по дисциплине "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Андрич К.

Группа ИУ7И-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Термин «матрица» применяется во множестве разных областей: от программирования до кинематографии.

Матрица в математике – это таблица чисел, состоящая из определенного количества строк (m) и столбцов (n).

Мы встречаемся с матрицами каждый день, так как любая числовая информация, занесенная в таблицу, уже в какой-то степени считается матрицей.

Примером могут служить:

- список телефонных номеров;
- различные статистические данные;
- табель успеваемости ученика и многое другое.

Целью работы работы является изучение и реализация алгоритмов умножения матриц, вычисление трудоёмкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Для достижения цели ставятся следующие задачи.

- Изучить классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
- Реализовать классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
- Дать оценку трудоёмкости алгоритмов.
- Замерить время работы алгоритмов.
- Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

Матрица – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Две матрицы одинакового размера можно поэлементно сложить или вычесть друг из друга.

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. При умножении матрицы размером 3x4 на матрицу размером 4x7 мы получаем матрицу размером 3x7. Умножение матриц некоммутативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга.

1.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы A (1.1) и B (1.2) размерности m на n и n на p соответсвенно:

$$\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix} \tag{1.1}$$

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,p} \\ \dots & \dots & \dots \\ b_{n,1} & \dots & b_{n,p} \end{bmatrix} \tag{1.2}$$

В результате получим матрицу C (1.3) размерности m на p:

$$\begin{bmatrix} c_{1,1} & \dots & c_{1,p} \\ \dots & \dots & \dots \\ c_{m,1} & \dots & c_{m,p} \end{bmatrix} \tag{1.3}$$

Формула (1.4) - формула рассчёта элемента, находящегося на i-ой строке j-ого столбца матрицы C.

$$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j} \tag{1.4}$$

1.2 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. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение (1.5) равно:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.5}$$

Это равенство можно переписать в виде (1.6):

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.6}$$

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.

1.3 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- вычисление происходит заранее;
- используется битовый сдвиг, вместо деления на 2;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

1.4 Вычисление трудоёмкости алгоритма

Введем модель трудоемкости для оценки алгоритмов.

1. Базовые операции стоимостью 1: +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса.
2. Оценка трудоемкости цикла: f_цикла = f_инициализации + f_сравнения + N * (f_инкремента + f_сравнения + f_тела).
3. Стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся. В условном операторе может возникнуть лучший и худший случаи по трудоёмкости в зависимости от выполнения условия и в зависимости от входных данных алгоритма.

1.5 Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основная разница которого — наличие предварительной обработки, а также уменьшение количества операций умножения.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема классического умножения матриц.

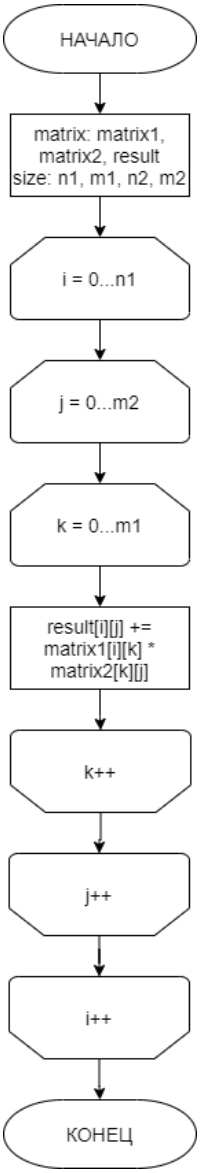

Рис. 2.1: Схема классического алгоритма умножения матриц

На рисунках 2.2, 2.3, 2.4, 2.5 представлена схема алгоритма умножения матриц Винограда.

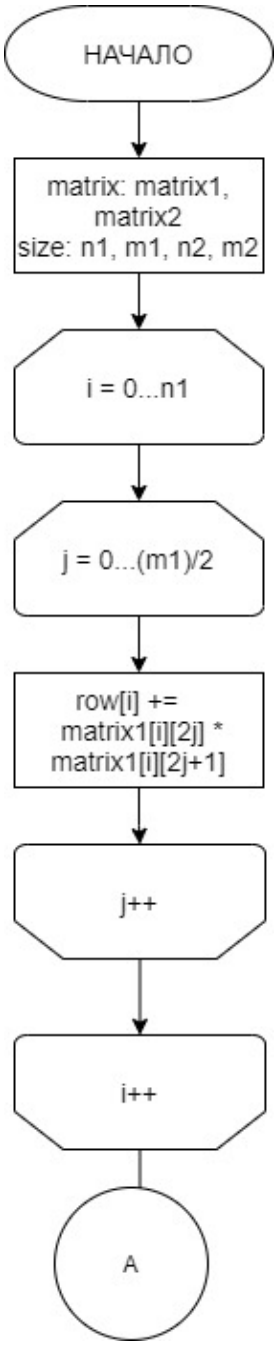

Рис. 2.2: Схема алгоритма Винограда (часть 1)

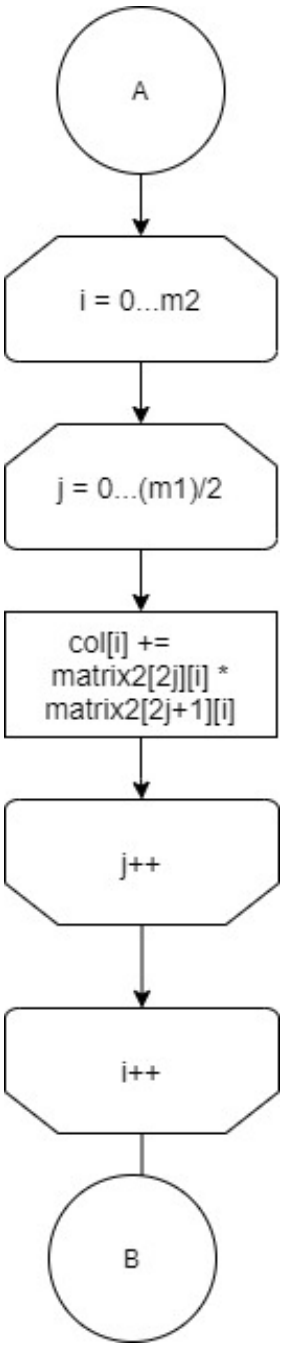

Рис. 2.3: Схема алгоритма Винограда (часть 2)

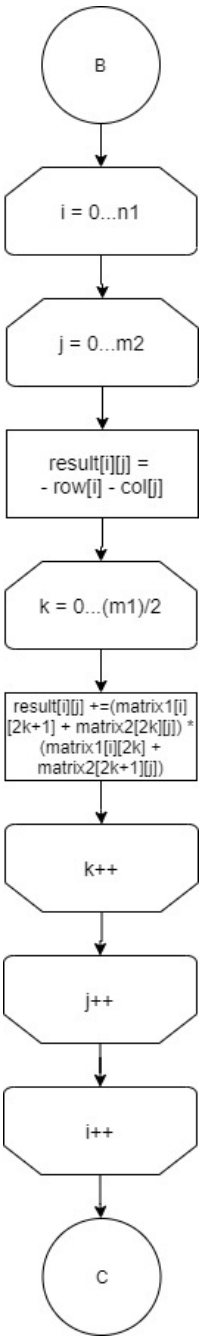

Рис. 2.4: Схема алгоритма Винограда (часть 3)

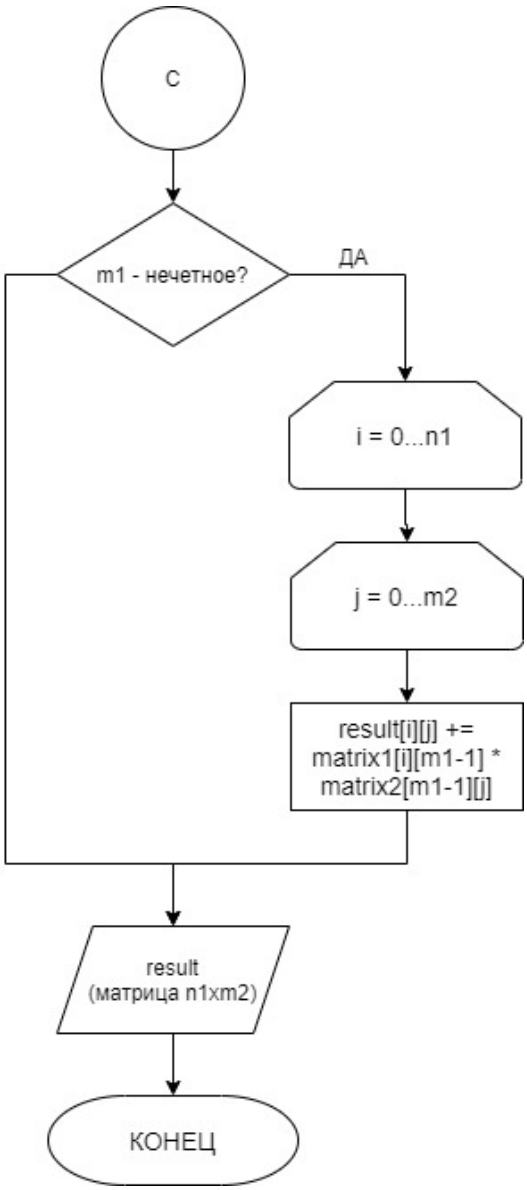

Рис. 2.5: Схема алгоритма Винограда (часть 4)

На рисунках 2.6, 2.7, 2.8, 2.9 представлена схема оптимизированного алгоритма умножения матриц Винограда.

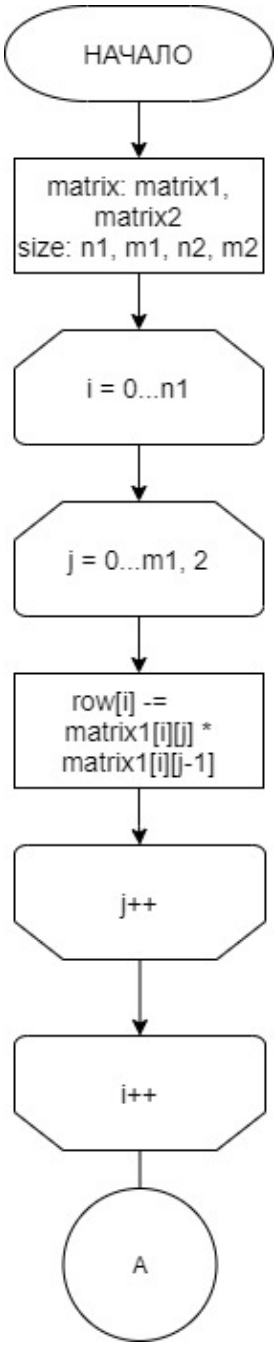

Рис. 2.6: Схема оптимизированного алгоритма Винограда(часть 1)

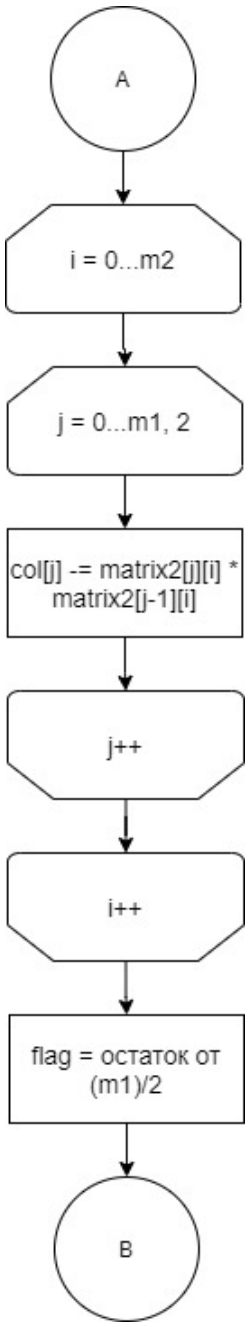

Рис. 2.7: Схема оптимизированного алгоритма Винограда(часть 2)

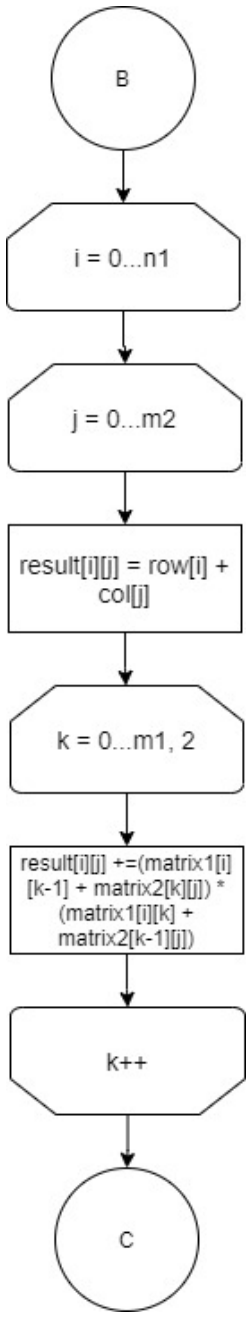

Рис. 2.8: Схема оптимизированного алгоритма Винограда(часть 3)

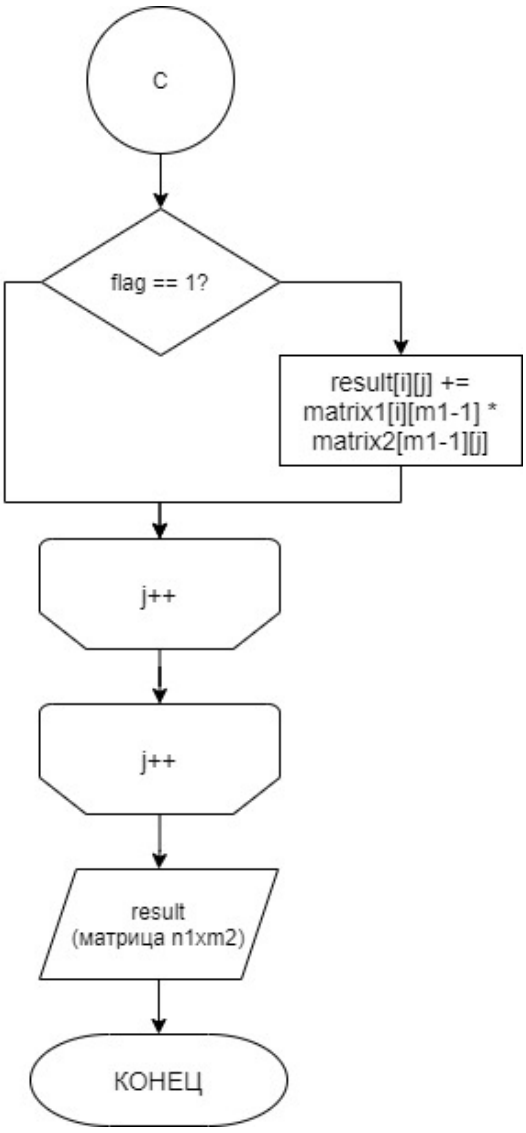

Рис. 2.9: Схема оптимизированного алгоритма Винограда(часть 4)

2.2 Оценка трудоемкости алгоритмов умножения матриц

1. Стандартный алгоритм

$$f = 2 + M(2 + 2 + Q(2 + 2 + N(2 + 8 + 1 + 1 + 1))) = 13\cdot$$

$$\cdot MNQ + 4MQ + 4M + 2 \approx 13 \cdot MNQ$$

2. Алгоритм Винограда

   Трудоемкость алгоритма Винограда:

   Первый цикл: $\frac{15}{2} \cdot NQ + 5 \cdot M + 2$

   Второй цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

   Третий цикл: $13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2$

   Условный переход: $\begin{bmatrix} 2 & \text{, лучший случай (при четном N)} \\ 15 \cdot QM + 4 \cdot M + 4 & \text{, худший случай} \end{bmatrix}$

   Итого: $f = \frac{15}{2}\cdot MN + \frac{15}{2}\cdot QN + 9\cdot M + 8 + 5\cdot Q + 13\cdot MNQ + 12\cdot MQ + \begin{bmatrix} 2 & \text{, в лучшем сл} \\ 15 \cdot QM + 4 \cdot M + 4 & \text{, в худшем сл} \end{bmatrix}$

$$f \approx 13 \cdot MNQ$$

3. Оптимизированный алгоритм Винограда

   Введем оптимизации:

   (a) замена операции = на += или -=

   (b) избавление от деления в условиях цикла (j < N, j += 2)

   (c) Заносим проверку на нечетность кол-ва строк внутрь основных циклов

   (d) Расчет условия для последнего цикла один раз, а далее использование флага

   Первый цикл: $4 \cdot NQ + 4 \cdot M + 2$

   Второй цикл: $4 \cdot MN + 4 \cdot M + 2$

   Третий цикл: $9 \cdot MNQ + 10 \cdot MQ + 4 \cdot M + 2$

   Условный переход: $\begin{bmatrix} 2 & \text{, лучший случай (при четном N)} \\ 10 \cdot QM & \text{, худший случай} \end{bmatrix}$

Трудоемкость оптимизированного алгоритма Винограда:

Итого:

$$f = 4 \cdot NQ + 4 \cdot M + 2 + 4 \cdot MN + 4 \cdot M + 2 + 9 \cdot MNQ + 10 \cdot MQ + 4 \cdot M + 2 +$$

$$+ \begin{bmatrix} 2 & \text{, л.с} \\ 10 \cdot QM & \text{, х.с} \end{bmatrix} \approx 9 \cdot MNQ$$

2.3 Вывод

На основе теоретических данных, полученные в аналитическом разделе были построены схемы иследуеммых алгоритмов.

3 | Технологическая часть

3.1 Выбор ЯП

В данной лабораторной работе использовался язык программирования - python. Данный выбор обусловлен тем, что этот язык наиболее удобен для работы со строками, а также тем, что в нём присутсвует функция для измерения процессорного времени. В качестве среды разработки я использовала Visual Studio Code, так как считаю его достаточно удобным.

3.2 Требование к ПО

Требования к вводу:

1. На вход подаются две пустые матрицы - корректный ввод, программа не должна аварийно завершаться;
2. ПО должно выводить полученную матрицу;
3. ПО должно выводить потраченное время;

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация алгоритмов умножения матриц.

Листинг 3.1: Функция классического умножения матриц

```
def standart_multiplication(mtx1, mtx2):
  if len(mtx2) != len(mtx1[0]):
    print("Size error!")
    return -1
  else:
    n = len(mtx1)
    q = len(mtx2[0])
    m = len(mtx1[0])
    mtx3 = [[0] * q for i in range(n)]

    for i in range(0, n):
      for j in range(0, q):
        for k in range(0, m):
          mtx3[i][j] = mtx3[i][j] + mtx1[i][k] * mtx2[k][j]
  return mtx3
```

Листинг 3.2: Функция умножения матриц с помощью аглоритма Винограда

```
def vinograd_multiplication(mtx1, mtx2):
  if len(mtx2) != len(mtx1[0]):
    print("Size error!")
    return -1
  else:
    m = len(mtx1)
    n = len(mtx1[0])
    q = len(mtx2[0])
    mtx3 = [[0] * q for i in range(m)]

    row = [0] * m
    for i in range(0, m):
      for j in range(0, n // 2, 1):
        row[i] = row[i] + mtx1[i][2 * j] * mtx1[i][2 * j + 1]

    col = [0] * q
    for j in range(0, q):
      for i in range(0, n // 2, 1):
        col[j] = col[j] + mtx2[2 * i][j] * mtx2[2 * i + 1][j]

    for i in range(0, m):
      for j in range(0, q):
        mtx3[i][j] = -row[i] - col[j]
        for k in range(0, n // 2, 1):
          mtx3[i][j] = mtx3[i][j] + (mtx1[i][2 * k + 1] + mtx2[2 * k][j]) *
              (mtx1[i][2 * k] + mtx2[2 * k + 1][j])

    if n % 2 == 1:
      for i in range(0, m):
        for j in range(0, q):
          mtx3[i][j] = mtx3[i][j] + mtx1[i][n - 1] * mtx2[n - 1][j]

  return mtx3
```

Листинг 3.3: Функция умножения матриц с помощью оптимизированного аглоритма Винограда

```
def vinograd_opt_multiplication(mtx1, mtx2):
  if len(mtx2) != len(mtx1[0]):
    print("Size error!")
    return -1
  else:
    m = len(mtx1)
    n = len(mtx1[0])
    q = len(mtx2[0])
    mtx3 = [[0] * q for i in range(m)]

    row = [0] * m
    for i in range(0, m):
      for j in range(1, n, 2):
        row[i] -= mtx1[i][j] * mtx1[i][j - 1]

    col = [0] * q
      for j in range(0, q):
        for i in range(1, n, 2):
          col[j] -= mtx2[i][j] * mtx2[i - 1][j]

    flag = n % 2
    for i in range(0, m):
      for j in range(0, q):
        mtx3[i][j] = row[i] + col[j]
        for k in range(1, n, 2):
          mtx3[i][j] += (mtx1[i][k - 1] + mtx2[k][j]) * (mtx1[i][k] + mtx2[k
              - 1][j])
        if (flag):
          mtx3[i][j] += mtx1[i][n - 1] * mtx2[n - 1][j]
  return mtx3
```

3.4 Тестовые данные

В таблице 3.1 приведены тестовые данные, на которых было протестированно разработанное ПО. Как видно из этой таблицы, все тесты были успешно пройдены, что означает, что программа работает правильно.

Первая матрица	**Вторая матрица**	**Ожидаемый результат**	**Полученный результат**
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	Матрицы не могут быть перемножены	Size error!
$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$

3.5 Вывод

В данном разделе были разработаны исходные коды трех алгоритмов: классическое умножение матриц, алгоритм Винограда, оптимизирвоанный алгоритм Винограда.

4 Исследовательская часть

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

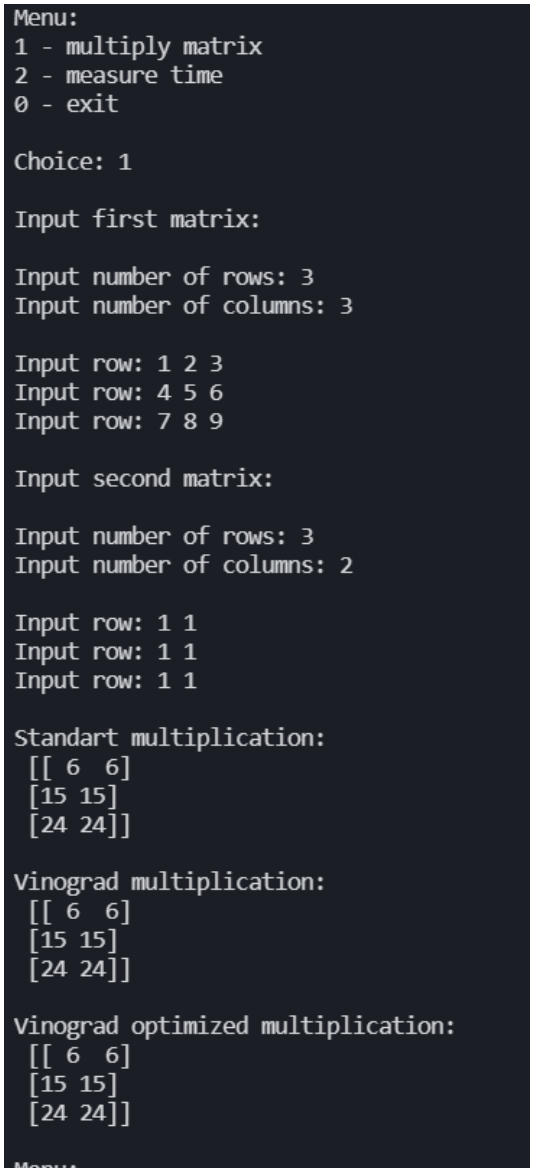

Рис. 4.1: Работа алгоритмов умножения матриц.

4.2 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось с помощью функции process_time модуля time в Python [?]. Данная функция возвращает значение в долях секунды суммы системного и пользовательского процессорного времени текущего процесса.

В таблице 4.1. представлены замеры времени работы для каждого из алгоритмов.

Таблица 4.1: Таблица времени выполнения алгоритмов

Длина строк	Standard	Vinograd	VinogradOpt
100	0.515625	0.593750	0.406250
200	4.000000	4.078125	3.296875
300	13.625000	14.609375	11.093750

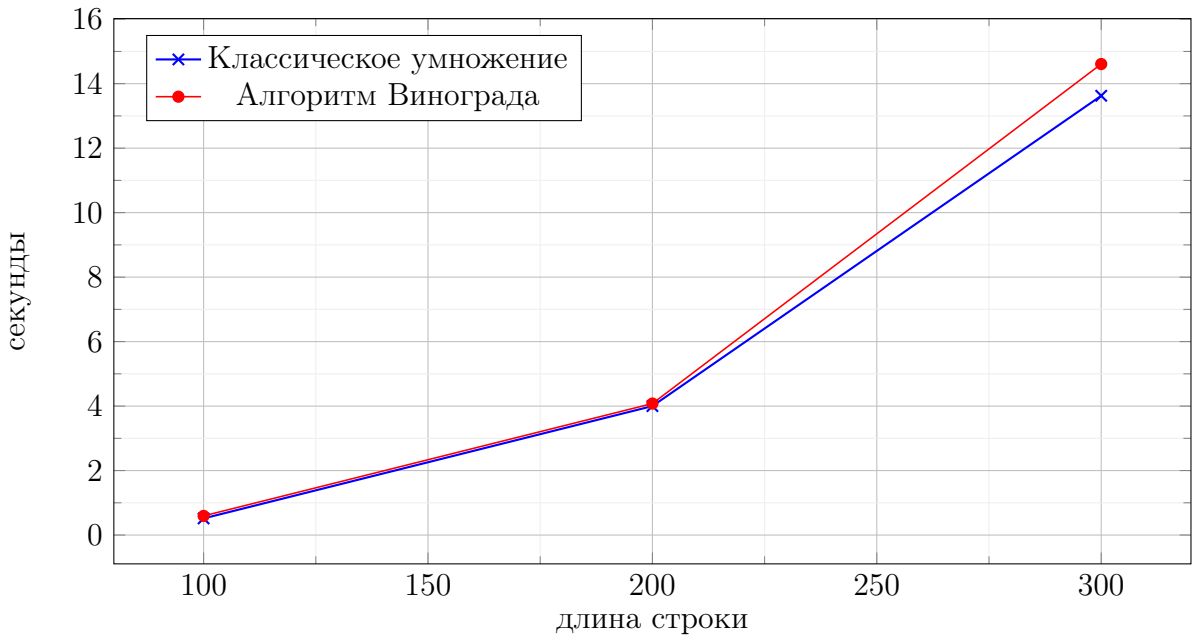

Рис. 4.2: Сравнение классического умножения и умножения с помощью алгортима Винограда

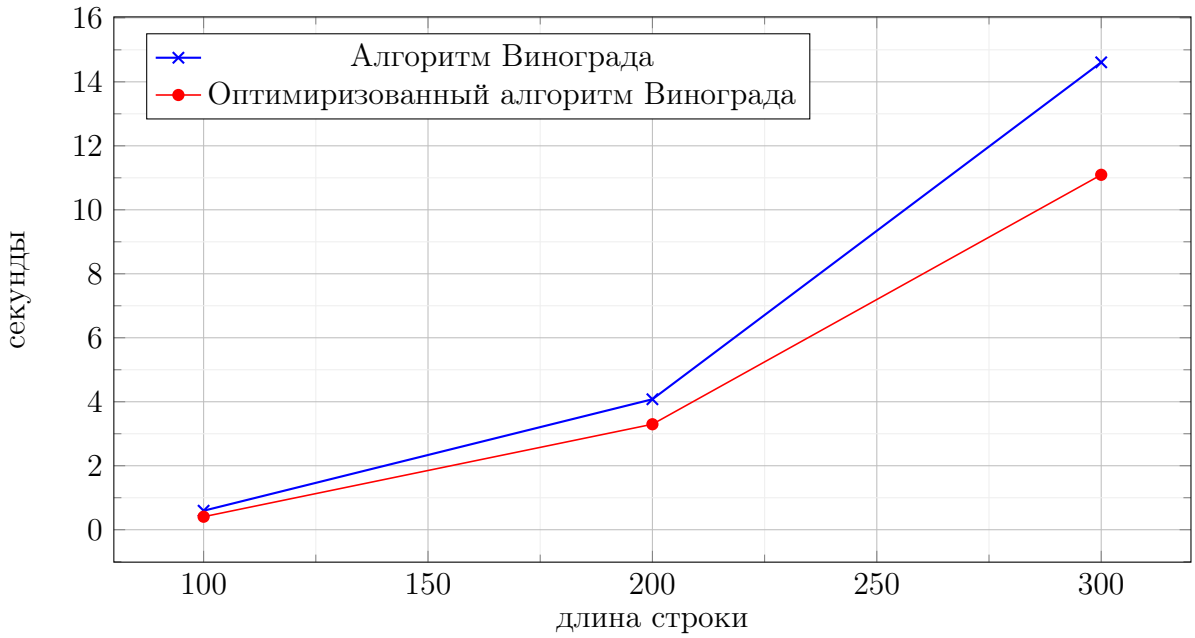

Рис. 4.3: Сравнение умножения с помощю алгортима Винограда и умножения с помощю оптимиризованного алгортима Винограда

4.3 Вывод

В результате проведенного эксперимента был получен следующий вывод: оптимизированный алгоритм Винограда работает быстрее классического метода и зачительно быстрее обычного алгоритма Винограда.

Заключение

В ходе проделанной работы были изучены и реализованы классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Экспериментально было подтверждено различие по временной эффективности алгоритмов умножения матриц на материале замеров процессорного времени выполнения реализации на варьирующихся размерах матриц. Так, самым быстрым является оптимизированный алгоритм Винограда,. Алгоритмы Винограда и классического умножения сопоставимы.

На основании сравнения данных алгоритмов был сделан вывод, что классический алгоритм является более эффективным, чем алгоритм Винограда, однако после ряда оптимизаций, алгоритм Винограда становится значительно быстрее классического.

Литература

[1] Умножение матриц.[Электронный ресурс].Режим доступа: https://life-prog.ru/2_90314_umnozhenie-matrits.html (дата обращения: 25.09.2020)

[2] Умножение матриц по Винограду [электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 25.09.2020).

[3] Python documentation [Электронный ресурс]. Режим доступа: https://www.python.org/doc/ (дата обращения: 26.09.2020).