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 курсу "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Мишина Е.В.

Группа ИУ7-54Б

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

Москва — 2020 г.

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 расчет 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 A 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 A и B размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B [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:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Это равенство можно переписать в виде 1.2.

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

Для векторов, размер которых — нечетное число, равенство 1.2 принимает вид 1.3.

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - \\ - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 + v_5 \cdot w_5 \tag{1.3}$$

Эти два равенства — 1.2 и 1.3 — могут быть обобщены на вектора произвольного размера.

Принцип алгоритма Винограда заключается в использовании равенств вида 1.2 и 1.3 в рамках умножения матриц — так, под векторами V и W понимаются строка певрой матрицы и столбец второй матрицы соответственно.

Выражение в правой части равенств 1.2 допускает предварительную обработку: его части $(v_1 \cdot v_2 + v_3 \cdot v_4)$ и $(w_1 \cdot w_2 + w_3 \cdot w_4)$ можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй соответсвенно.

Так, при вычислении $V \cdot W$ с использованием предварительно вычисленных значений нам необходимо выполнить лишь первые два умножения — $(v_1 + w_2) \cdot (v_2 + w_1)$ и $(v_3 + w_4) \cdot (v_4 + w_3)$ — и посчитать линейную комбинацию полученных значения согласно формуле 1.2.

В случае умножения матриц, строка и столбец которых преставляют собой вектора нечетного размера, схема рассчета элементов результирующей матрицы сохраняется. После чего, к каждому элементу c_{ij} результирующей матрицы прибавляется число $v_{im} \cdot u_{mj}$, где v_{im} - последний элемент i-той строки первой матрицы, u_{mj} - последний элемент j-того столбца второй матрицы.

Вывод

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

2 Конструкторская часть

Ввод:

- на вход подаются две числовые матрицы;

Вывод:

- программа выводит результат умножения матриц;
- в режиме замера ресурсов программа выводит средние время, затраченное каждым алгоритмом.

2.1 Схемы алгоритмов

В данном разделе будут рассмотрен стандартный алгоритм умножения матриц (2.1), алгоритм Виноградова (2.2), модифицированный алгоритм Виноградова (2.3).

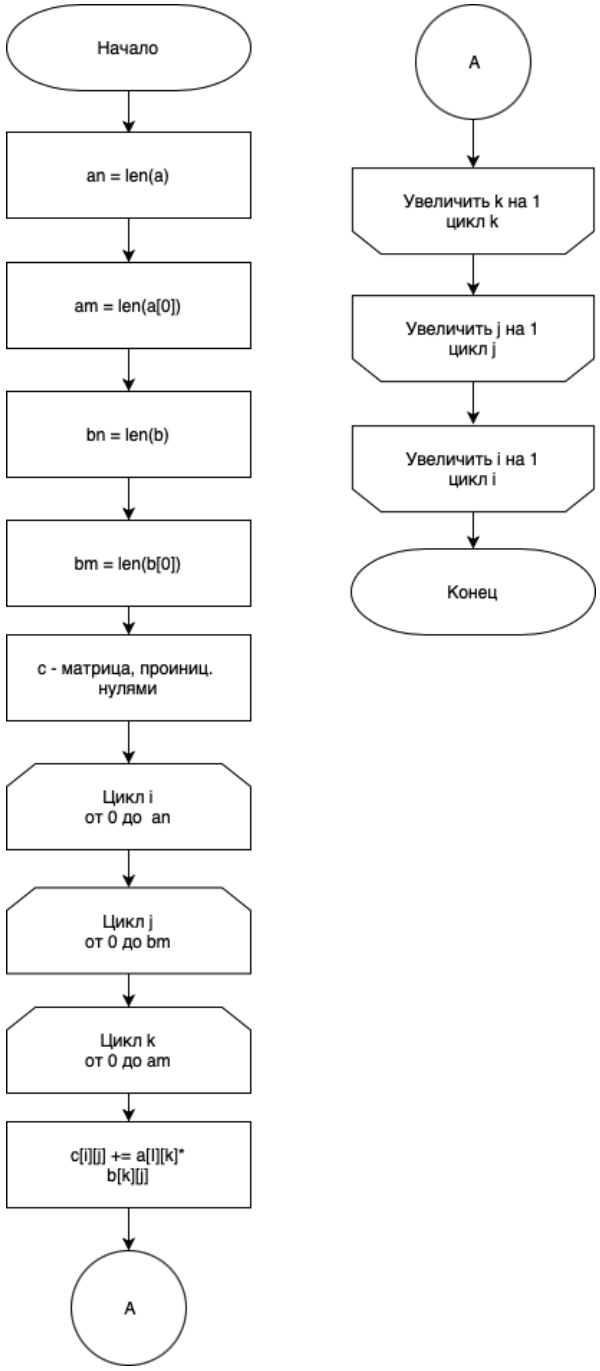

Рис. 2.1: Схема стандартного алгоритма умножения матриц

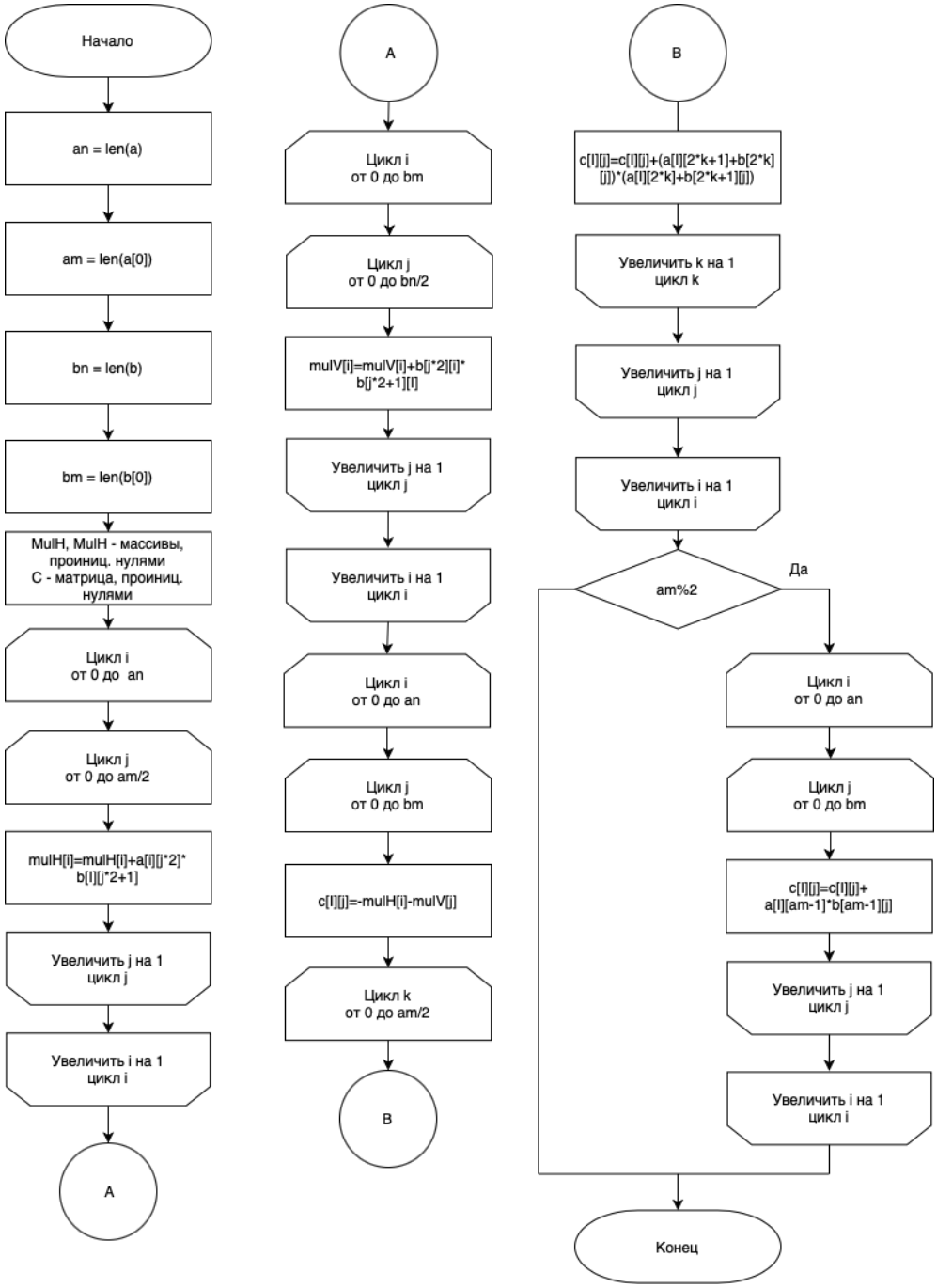

Рис. 2.2: Схема алгоритма Виноградова

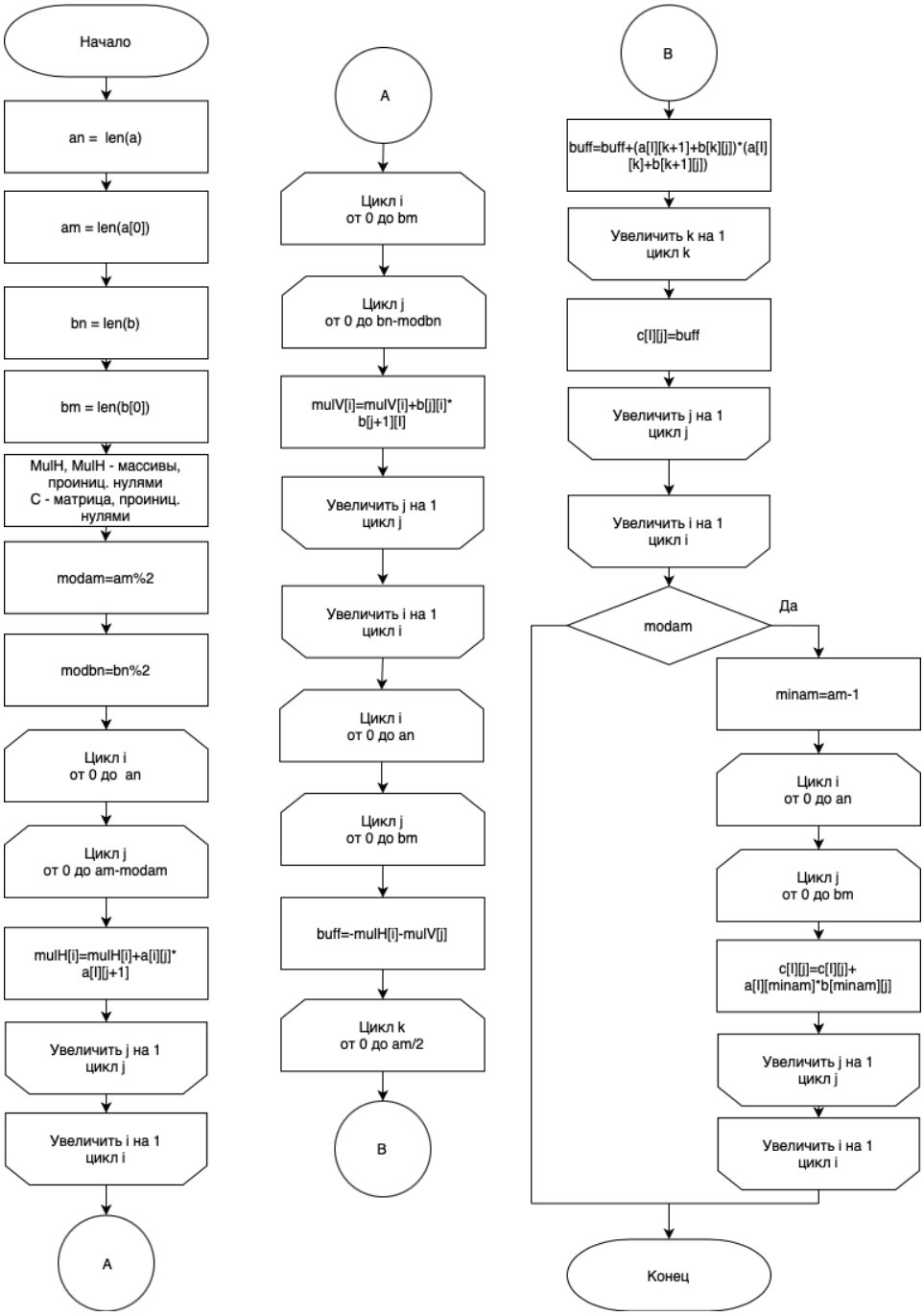

Рис. 2.3: Схема модифицированного алгоритма Виноградова

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], ++, – получение полей класса

2. оценка трудоемкости цикла: Fц = a + N*(a + F(тела)), где a - условие цикла

3. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будет приведены оценки трудоемкости алгоритмов. Построчная оценка трудоемкости стандартного алгоритма умножения матриц (Табл. 2.1), алгоритма Виноградова (Табл. 2.2), модифицированный алгоритм Виноградова (Табл. 2.3).

2.2.1 Стандартный алгоритм умножения матриц

Табл. 2.1 Построчная оценка веса

Код	Вес
an = len(a)	2
am = len(a[0])	3
bn = len(b)	2
bm = len(b[0])	3
c = [[0 for i in range(bm)] for j in range(an)]	2 + n1*(2 + m2*1)
for i in range(an):	2
for j in range(bm):	2
for k in range(am):	2
c[i][j] += a[i][k] * b[k][j]	9

Трудоемкость: $T(n) = 2 + 3 + 2 + 3 + 2 + n1 * (2 + m2 * 1) + 2 + n1 * (2 + m2 * (2 + m1 * 9)) = 14 + n1 * 4 + n1 * m2 * 3 + n1 * m2 * m1 * 9$

Итого: $T(n) = 14 + n1 * 4 + n1 * m2 * 3 + n1 * m2 * m1 * 9$

2.2.2 Алгоритм Виноградова

Табл. 2.2 Построчная оценка веса

Код	Вес
an = len(a)	2
am = len(a[0])	3
bn = len(b)	2
bm = len(b[0])	3
c = [[0 for i in range(bm)] for j in range(an)]	2 + n1*(2 + m2*1)
mulH = [0 for j in range(an)]	2 + n1*1
mulV = [0 for i in range(bm)]	2 + m2*1
for i in range(an):	2
for j in range(int(am / 2)):	3
mulH[i] = mulH[i] + a[i][j * 2] * b[i][j * 2 + 1]	12
for i in range(bm):	2
for j in range(int(bn / 2)):	3
mulV[i] = mulV[i] + b[j * 2][i] * b[j * 2 + 1][i]	12
for i in range(an):	2
for j in range(bm):	2
c[i][j]=-mulH[i] - mulV[j]	7
for k in range(int(am / 2)):	2
c[i][j]=c[i][j] + (a[i][2 * k + 1] + b[2 * k][j]) * (a[i][2 * k] + b[2 * k + 1][j])	23
if am % 2:	2
for i in range(an):	2
for j in range(bm):	2
c[i][j] = c[i][j] + a[i][am - 1] * b[am - 1][j]	13

Трудоемкость: $T(n) = 2+3+2+3+2+n1*(2+m2*1)+2+n1*1+2+m2*1+2+n1*(3+n3*12)+2+m2*(3+m5*12)+2+n1*(2+m2*(7+2+m3*23))+2+2+n1*(2+m2*13) = 22+n1*8+n1*m2*10+m2*4+n1*n3*12+m2*m5*12+n1*m2*m3*23+\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 2+2+n1*2+n1*m2*13) & \text{, выполнение условия} \end{bmatrix}$

Итого: $T(n) = 22+n1*8+n1*m2*10+m2*4+n1*n3*12+m2*m5*12+n1*m2*m3*23+\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 4+n1*2+n1*m2*13 & \text{, выполнение условия} \end{bmatrix}$

2.2.3 Модифицированный алгоритм Виноградова

Табл. 2.3 Построчная оценка веса

Код	Вес
an = len(a)	2
am = len(a[0])	3
bn = len(b)	2
bm = len(b[0])	3
c = [[0 for i in range(bm)] for j in range(an)]	2 + n1*(2 + m2*1)
mulH = [0 for j in range(an)]	2 + n1*1
mulV = [0 for i in range(bm)]	2 + m2*1
modam = am % 2	3
modbn = bn % 2	3
for i in range(an):	2
for j in range(0, am - modam, 2):	3
mulH[i] = mulH[i] + a[i][j] * a[i][j + 1]	10
for i in range(bm):	2
for j in range(0, bn - modbn, 2):	3
mulV[i] = mulV[i] + b[j][i] * b[j + 1][i]	10
for i in range(an):	2
for j in range(bm):	2
buff = -mulH[i] - mulV[j]	5
for k in range(0, am - modam, 2):	2
buff = buff + (a[i][k +1] + b[k][j]) * (a[i][k] + b[k + 1][j])	15
if modam:	1
minam = am - 1	2
for i in range(an):	2
for j in range(bm):	2
c[i][j] = c[i][j] + a[i][minam] * b[minam][j]	11

Трудоемкость: $T(n) = 2+3+2+3+2+n1*(2+m2*1)+2+n1*1+2+m2*1+3+3+2+m4*(3+n5*10)+2+m2*(3+n5*10)+2+n1*(2+m2*(5+2+m3*15))+1+2+2+n1*(2+m2*11) = 28+n1*5+m2*4+n1*m2*6+n4*n5*10+m2*n5*10+n1*m2*2+n1*m2*m3*15+\begin{bmatrix}1 & \text{, невыполнение условия}\\ 5+n1*2+n1*m2*11 & \text{, выполнение условия}\end{bmatrix}$

Итого: $T(n) = 28+n1*5+m2*4+n1*m2*6+n4*n5*10+m2*n5*10+n1*m2*2+n1*m2*m3*15+\begin{bmatrix}1 & \text{, невыполнение условия}\\ 5+n1*2+n1*m2*11 & \text{, выполнение условия}\end{bmatrix}$

2.3 Вывод

Стандартный алгоритм умножения матриц: $T(n) = 14 + n1 * 4 + n1 * m2 * 3 + n1 * m2 * m1 * 9$

Алгоритм Виноградова: $T(n) = 22+n1*8+n1*m2*10+m2*4+n1*n3*12+m2*m5*12+n1*m2*m3*23+\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 4+n1*2+n1*m2*13 & \text{, выполнение условия} \end{bmatrix}$

Модифицированный алгоритм Виноградова: $T(n) = 28 + n1 * 5 + m2 * 4 + n1 * m2 * 6 + n4 * n5 * 10 + m2 * n5 * 10 + n1 * m2 * 2 + n1 * m2 * m3 * 15 + \begin{bmatrix} 1 & \text{, невыполнение условия} \\ 5+n1*2+n1*m2*11 & \text{, выполнение условия} \end{bmatrix}$

В данном разделе была приведена оценка трудоемкости стандартного алгоритма умножения матриц, алгоритма Виноградова, модифицированный алгоритм Виноградова.

3 Технологическая часть

3.1 Выбор ЯП

Для реализации программы был выбран Python[2] из-за наличия опыта разработки на данном языке программирования. Для тестирования были выбрана операционная система - Windows 10. Процессор - 2,3 GHz Quad-Core Intel Core i5. Память - 8 GB 2133 MHz LPDDR3. Графическое ядро - Intel Iris Plus Graphics 655 1536 MB.

3.2 Реализация алгоритма

Листинг 3.1: Функция стандартного алгоритма умножения матриц

```
1
2       def StandardMultiplication(a, b):
3           an = len(a)
4           am = len(a[0])
5
6           bn = len(b)
7           bm = len(b[0])
8
9           c = [[0 for i in range(bm)] for j in range(an)]
10
11          for i in range(an):
12              for j in range(bm):
13                  for k in range(am):
14                      c[i][j] += a[i][k] * b[k][j]
15          return c
```

Листинг 3.2: Функция алгоритма Виноградова

```
1
2       def MultiplicationVinograd(a, b):
3           an = len(a)
4           am = len(a[0])
5           bn = len(b)
6           bm = len(b[0])
7
8           c = [[0 for i in range(bm)] for j in range(an)]
9           mulH = [0 for j in range(an)]
10          mulV = [0 for i in range(bm)]
11          for i in range(an):
12              for j in range(int(am / 2)):
13                  mulH[i] = mulH[i] + a[i][j * 2] * b[i][j * 2 + 1]
14
15          for i in range(bm):
16              for j in range(int(bn / 2)):
17                  mulV[i] = mulV[i] + b[j * 2][i] * b[j * 2 + 1][i]
18
19          for i in range(an):
20              for j in range(bm):
21                  c[i][j] = -mulH[i] - mulV[j]
22                  for k in range(int(am / 2)):
23                      c[i][j] = c[i][j] + (a[i][2 * k + 1] + b[2 *
                            k][j]) * (a[i][2 * k] + b[2 * k + 1][j])
24
25          if am % 2:
26              for i in range(an):
27                  for j in range(bm):
28                      c[i][j] = c[i][j] + a[i][am - 1] * b[am - 1][
                            j]
29
30          return c
```

Листинг 3.3: Функция модифицированного алгоритма Виноградова

```
1
2       def MultiplicationVinogradOptimize(a, b):
3           an = len(a)
4           am = len(a[0])
5           bn = len(b)
6           bm = len(b[0])
7
8           c = [[0 for i in range(bm)] for j in range(an)]
9           mulH = [0 for j in range(an)]
10          mulV = [0 for i in range(bm)]
11          modam = am % 2
12          modbn = bn % 2
13          for i in range(an):
14              for j in range(0, am - modam, 2):
15                  mulH[i] = mulH[i] + a[i][j] * a[i][j + 1]
16
17          for i in range(bm):
18              for j in range(0, bn - modbn, 2):
19                  mulV[i] = mulV[i] + b[j][i] * b[j + 1][i]
20
21          for i in range(an):
22                  for j in range(bm):
23                      buff = -mulH[i] - mulV[j]
24                      for k in range(0, am - modam, 2):
25                          buff = buff + (a[i][k +1] + b[k][j]) * (a
                                [i][k] + b[k + 1][j])
26                      c[i][j] = buff
27          if modam:
28              minam = am - 1
29              for i in range(an):
30                  for j in range(bm):
31                      c[i][j] = c[i][j] + a[i][minam] * b[minam][j]
32
33          return c
```

3.2.1 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 4 оптимизации:

1. Избавление от деления в условии цикла;
2. Замена $mulH[i] = mulH[i] + \ldots$ на $mulH[i] += \ldots$ (аналогично для $mulV[i]$);
3. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти. Сброс буфера в ячейку матрицы после цикла.
4. В последнем цикле на проверку чётности убрано вычетание в цикле

3.3 Вывод

В данном разделе были рассмотрены алгоритм стандартного умножения матриц, алгоритм Виноградова и модифицированного алгоритм Виноградова.

4 Исследовательская часть

4.1 Сравнение на основе замеров времени работы алгоритмов

4.2 Постановка эксперимента

Будем измерять время работы алгоритмов на случайно сгенерированных квадратных матрицах с четной и нечетной размерностью. Для замера времени будем использовать функцию time. Для каждого алгоритма проведем 100 тестов (по каждой размерности), результатом тестирования будет среднее время за 100 проведенных тестов.

4.2.1 Замеры времени с матрицей четной размерности

Стандартный алгоритм умножения матриц

размерность	время, сек.
100	0.0228207111
200	0.1931939125
300	0.6668487310
400	1.4911422968
500	2.9295366049

Алгоритм Виноградова

размерность	время, сек.
100	0.0287097216
200	0.2542057037
300	0.8393203020
400	1.9044293165
500	3.7713220119

Модифицированный алгоритм Виноградова

размерность	время, сек.
100	0.0187155962
200	0.1571273327
300	0.5064072847
400	1.2909508228
500	2.6337560177

4.2.2 Замеры времени с матрицей нечетной размерности

Стандартный алгоритм умножения матриц

размерность	время, сек.
101	0.0209214926
201	0.1652329922
301	0.5921019077
401	1.4821319818
501	2.9179641247

Алгоритм Виноградова

размерность	время, сек.
101	0.0278947115
201	0.2142146111
301	0.7450125933
401	1.8801513195
501	3.7777832985

Модифицированный алгоритм Виноградова

размерность	время, сек.
101	0.0180447102
201	0.1417878151
301	0.5070972204
401	1.2795981169
501	2.6417391062

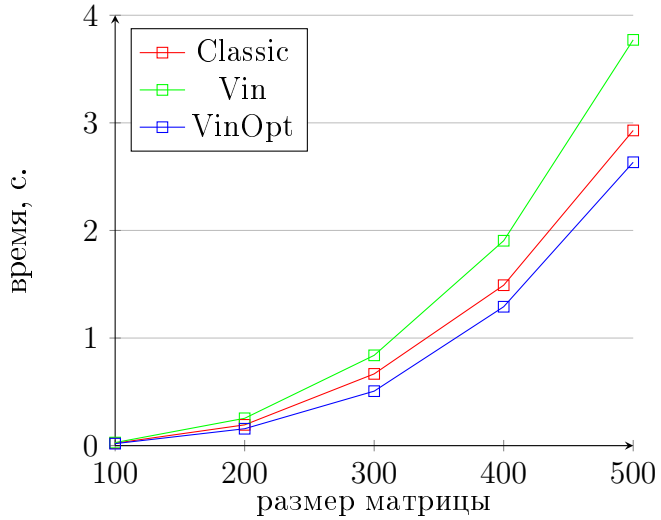

Рис. 4.1: Сравнение алгоритмов умножения с четной размерностью матриц

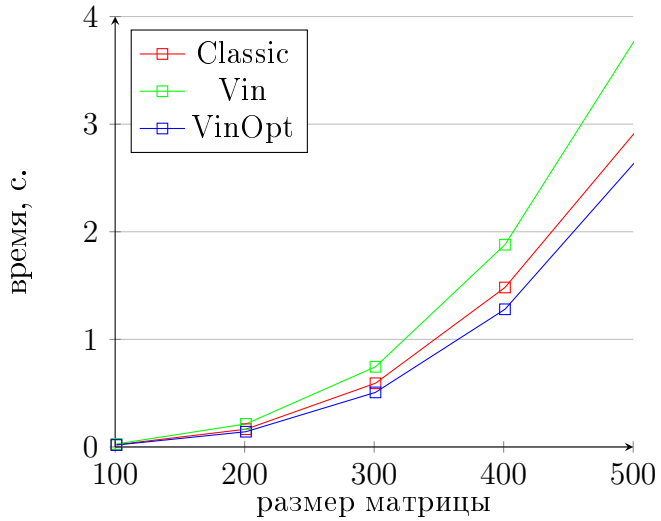

Рис. 4.2: Сравнение алгоритмов умножения с нечетной размерностью матриц

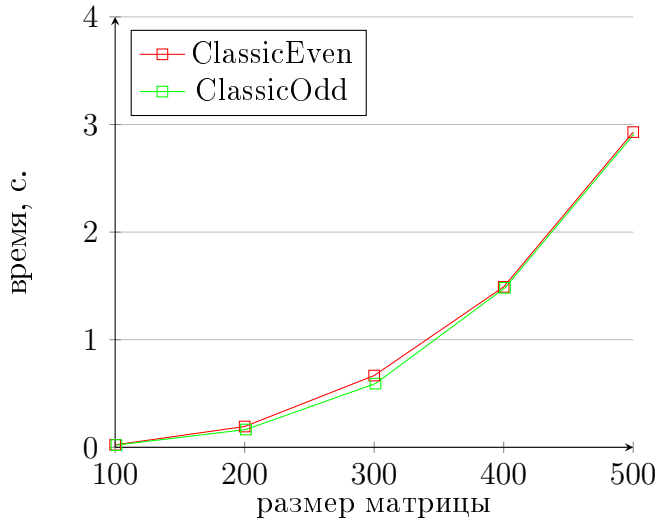

Рис. 4.3: Сравнение стандарного алгоритма умножения с четной и нечетной размерностью матриц

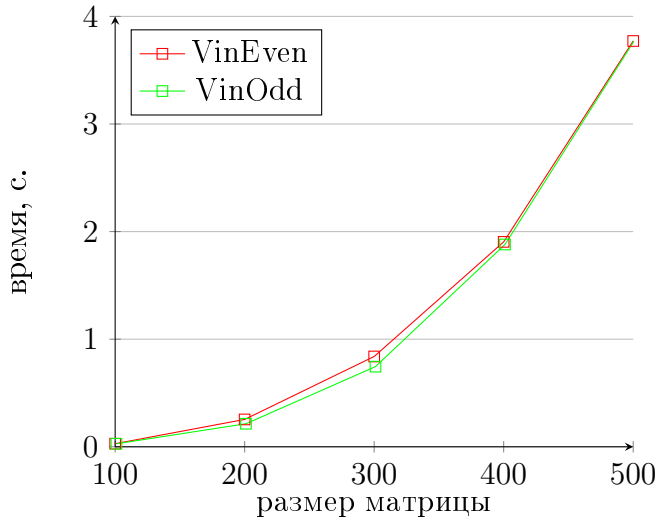

Рис. 4.4: Сравнение алгоритма Виноградова с четной и нечетной размерностью матриц

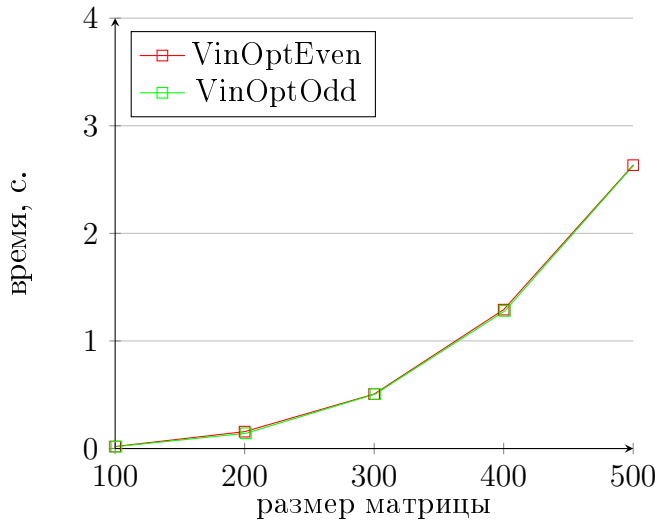

Рис. 4.5: Сравнение алгоритма оптимизированного Виноградова с четной и нечетной размерностью матриц

4.3 Тестовые данные

Проведем тестирование программы. В столбцах "Ожидаемый результат"и "Полученный результат"3 матрицы соответствуют результату стандартного алгоритма умножения матриц, алгоритма Виноградова и модифицированного алгоритма Виноградова.

Таблица 4.1: Таблица тестовых данных

№	матрицы	Ожидаемый результат	Полученный результат
1	[[1,2],[4,5]], [[1,2],[4,5]]	[[7,10],[15,22]]	[[7,10],[15,22]]
2	[[100]], [100]]	[10000]	[10000]
3	[[1,2,3],[4,5,6]], [[1,2],[3,4], [5,6]]	[[22,28],[49,64]]	[22,28],[49,64]]
4	[]	[]	[]

4.4 Вывод

Были протестированы алгоритмы умножения на квадратных матрицах размерами 100...500 с шагом 100 и 101...501 с шагом 100. Бралось среднее время за 100 проведенных тестов каждого алгоритма.

В результате тестирования алгоритмов умножения было получено, что лучшее время показывает улучшенный алгоритм Виноградова. Худшие время умножения показывает стандартный алгоритм умножения.

Сравнивая время умножения четных и нечетных размерностью матриц, видно, что время умножения почти одинаковое.

Эксперименты подтвердили подсчитанную ранее трудоемкость алгоритмов.

Заключение

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения и были рассмотрены их схемы и реализации алгоритмов на языке программирования Python.

Была проведена оценка трудоемкости алгоритмов: Стандартный алгоритм умножения матриц: $T(n) = 14 + n1 * 4 + n1 * m2 * 3 + n1 * m2 * m1 * 9$
Алгоритм Виноградова: $T(n) = 22+n1*8+n1*m2*10+m2*4+n1*n3*12+m2*m5*12+n1*m2*m3*23+\begin{bmatrix} 2 & \text{, невыполнение условия} \\ 4 + n1 * 2 + n1 * m2 * 13 & \text{, выполнение условия} \end{bmatrix}$
Модифицированный алгоритм Виноградова: $T(n) = 28 + n1 * 5 + m2 * 4 + n1 * m2 * 6 + n4 * n5 * 10 + m2 * n5 * 10 + n1 * m2 * 2 + n1 * m2 * m3 * 15 + \begin{bmatrix} 1 & \text{, невыполнение условия} \\ 5 + n1 * 2 + n1 * m2 * 11 & \text{, выполнение условия} \end{bmatrix}$

Были протестированы алгоритмы умножения на квадратных матрицах размерами $100 \ldots 500$ с шагом 100 и $101 \ldots 501$ с шагом 100.

В результате тестирования алгоритмов умножения было получено, что лучшее время показывает улучшенный алгоритм Виноградова. Худшие время умножения показывает стандартный алгоритм умножения.

Сравнивая время умножения четных и нечетных размерностью матриц, видно, что время умножения почти одинаковое.

Эксперименты подтвердили подсчитанную ранее трудоемкость алгоритмов.

Таким образом, задания выполнены и цель достугнута.

Список использованных источников

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. Python [Электронный ресурс]. – Режим доступа: https://www.python.org. – Дата доступа: 01.11.2020.