



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»

Лабораторная работа № 2

Дисциплина Анализ алгоритмов

Тема Трудоёмкость алгоритмов умножения матриц

Студент Искакова К. М.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Оглавление

Введение	3
1. Аналитическая часть	4
1.1 Описание классического умножения матриц	4
1.2 Описание умножения матриц по Винограду	4
2. Конструкторская часть	6
2.1 Оптимизации алгоритма умножения по Винограду	6
2.2 Схемы алгоритмов	6
3. Технологическая часть	10
3.1 Требования к программному обеспечению	10
3.2 Средства реализации	10
3.3 Листинг кода	11
3.4 Трудоемкость алгоритмов	14
3.4.1 Трудоемкость стандартного алгоритма умножения матриц	14
3.4.2 Трудоемкость алгоритма умножения матриц по Винограду	14
3.4.3 Трудоемкость оптимизированного алгоритма умножения матриц по Винограду	15
3.5 Тестирование	16
4. Экспериментальная часть	17
4.1 Примеры работы	17
4.2 Постановка эксперимента по замеру времени	19
Заключение	21
Список литературы	22

Введение

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].

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц. Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы согласованы. В частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка. Таким образом, из существования произведения AB вовсе не следует существование произведения BA . Алгоритмы умножения матриц активно применяются во всех областях, применяющих линейную алгебру. Примеры использования:

- компьютерная графика;
- физика;
- экономика и так далее.

Цель работы: оценить трудоемкость алгоритмов умножения матриц и получить практический навык оптимизации алгоритмов.

Задачи работы:

- 1) изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- 2) оптимизировать алгоритм Винограда;
- 3) оценить трудоемкость классического алгоритма умножения матриц, стандартного и оптимизированного алгоритма Винограда;
- 4) реализовать данные алгоритмы умножения матриц на одном из языков программирования;
- 5) сравнить алгоритмы умножения матриц.

1 | Аналитическая часть

В данном разделе будет рассмотрено описание умножения матриц.

1.1. Описание классического умножения матриц

Пусть даны две прямоугольные матрицы A и B размерности m на n и n на k соответственно:

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

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,k} \\ \dots & \dots & \dots \\ b_{n,1} & \dots & b_{n,k} \end{bmatrix}$$

В результате получим матрицу C размерности m на k:

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

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

1.2. Описание умножения матриц по Винограду

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 $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4$$

Это равенство можно переписать в виде:

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

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.

В случае нечетной общей размерности происходит прибавление к каждому элементу матрицы произведения последнего элемента столбца первой матрицы на последний элемент строки второй матрицы

2 | Конструкторская часть

В этом разделе содержатся схемы алгоритмов умножения матриц.

2.1. Оптимизации алгоритма умножения по Винограду

В рамках данной лабораторной работы было предложено 3 оптимизации алгоритма умножения по Винограду:

- 1) избавление от деления в условии цикла;
- 2) замена $mulH[i] = mulH[i] + \dots$ на $mulH[i] += \dots$ (аналогично для других);
- 3) накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти, и сброс буфера в ячейку матрицы после цикла.

2.2. Схемы алгоритмов

На рис. 2.1-2.3 приведены схемы следующих алгоритмов:

- 1) стандартный алгоритм умножения матриц;
- 2) алгоритм умножения матриц по Винограду;
- 3) оптимизированный алгоритм умножения матриц по Винограду;

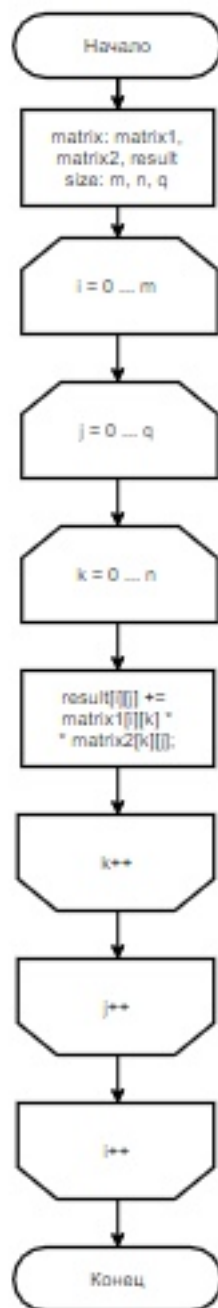


Рис. 2.1. Стандартный алгоритм умножения матриц

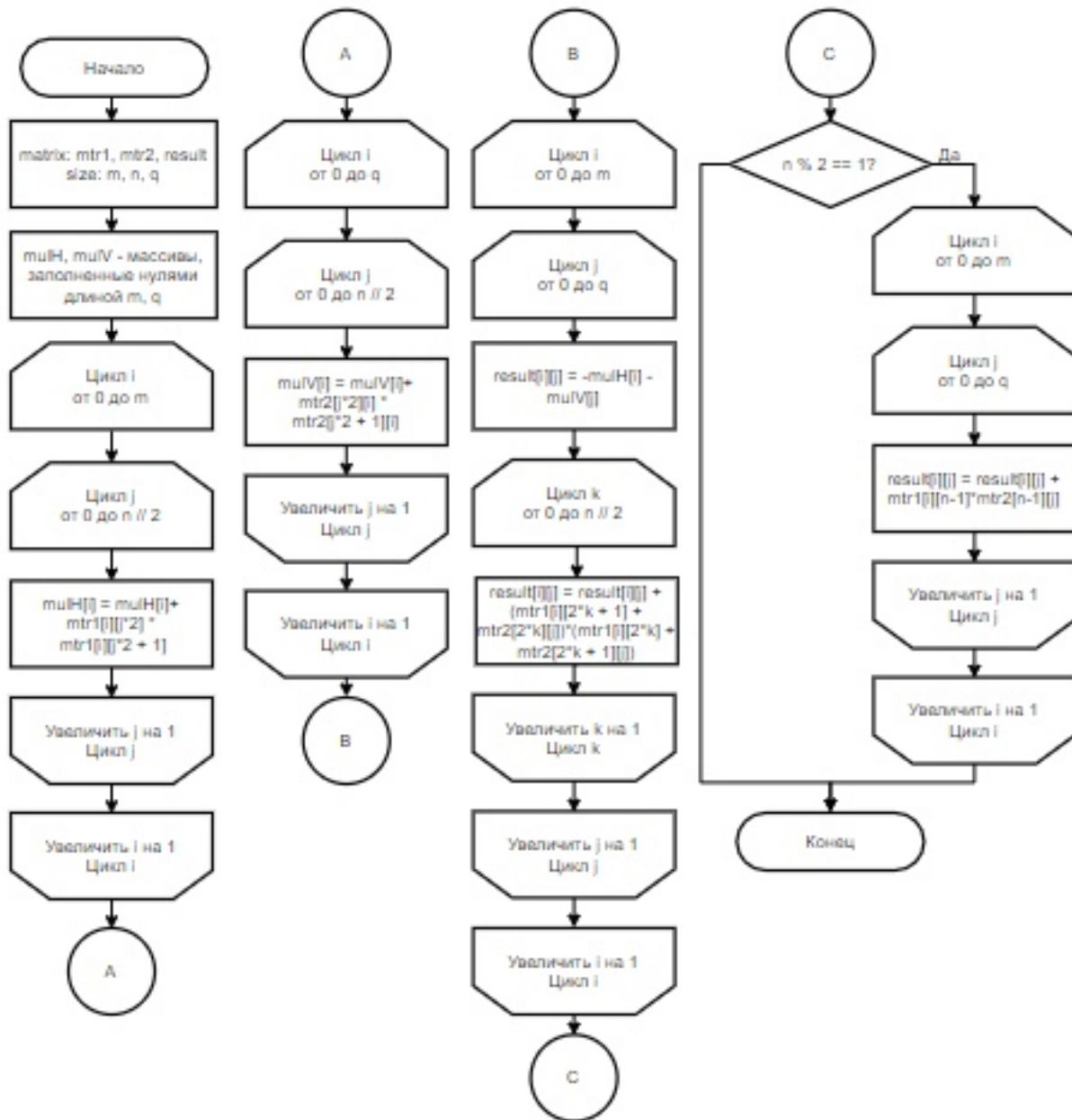


Рис. 2.2. Алгоритм умножения матриц по Винограду

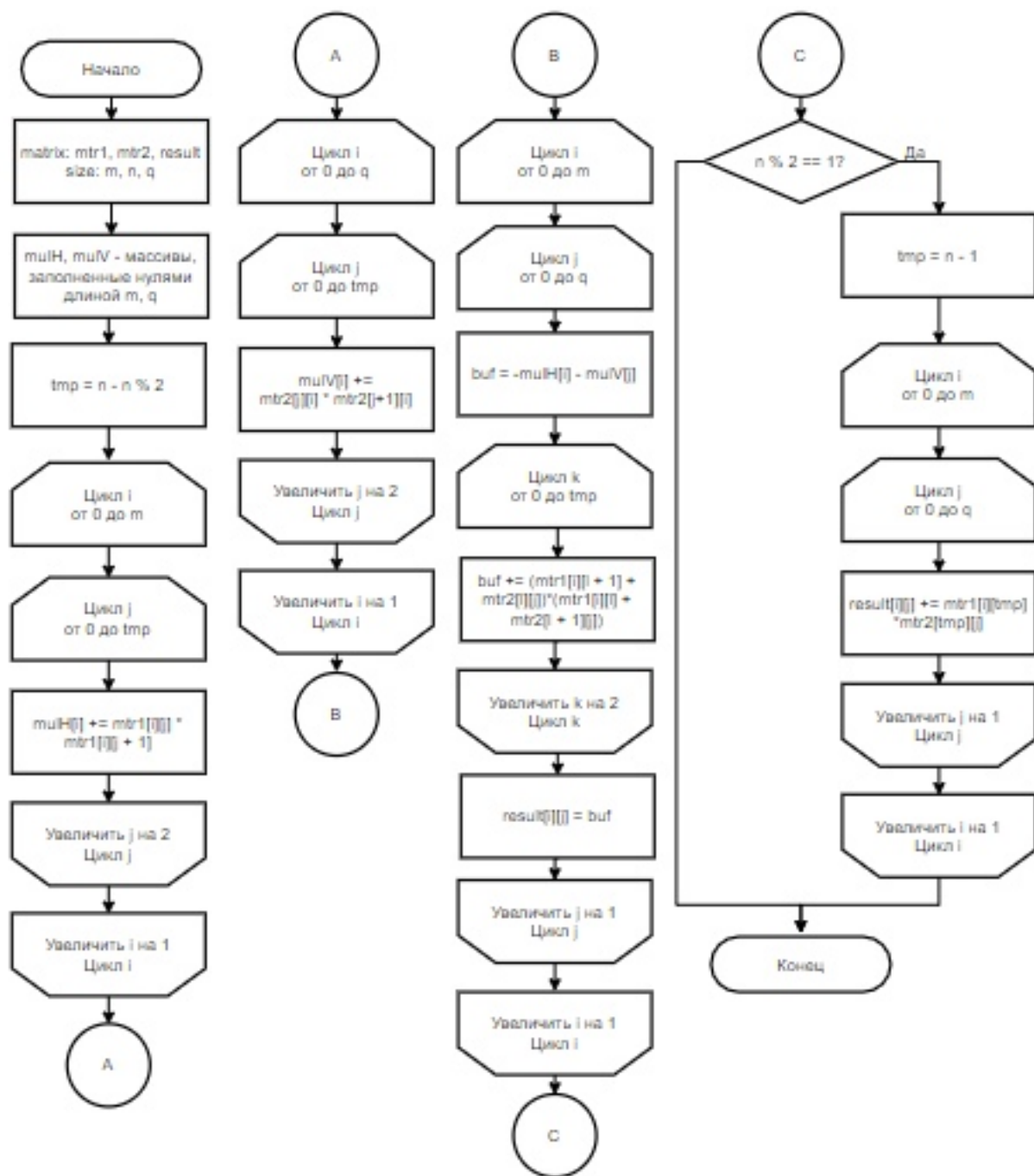


Рис. 2.3. Оптимизированный алгоритм умножения матриц по Винограду

3 | Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации, представлен листинг кода, описание тестирования и трудоемкость алгоритмов.

3.1. Требования к программному обеспечению

Требования к вводу: на вход подаются две матрицы, размеры которых $m \times n$ и $n \times q$ соответственно.

Требования к программе:

- 1) на выходе необходимо получить матрицу, которая является результатом умножения двух матриц;
- 2) требуется замерить время работы каждого из алгоритмов.

3.2. Средства реализации

В качестве языка программирования был выбран Python т.к. я знаком с данным языком, он простой и лаконичный, имеющий немногословный и понятный синтаксис, похожий на псевдокод, обладающий сильной динамической типизацией, которая способствует быстрому написанию кода.

Среда разработки — PyCharm, которая предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, вкуче с автоматическим рефакторингом кода, и богатыми возможностями в навигации.

Время работы алгоритмов было замерено с помощью функции `process_time()` из библиотеки `time` [2].

3.3. Листинг кода

В листингах 3.1-3.3 представлена реализация алгоритмов умножения матриц.

Листинг 3.1. Стандартный алгоритм умножения матриц

```
1 def classic(n, m, k, matr1, matr2, res):
2     i = 0
3     while i < n:
4         j = 0
5         while j < k:
6             l = 0
7             while l < m:
8                 res[i][j] += matr1[i][l] * matr2[l][j]
9                 l += 1
10            j += 1
11        i += 1
12    return res
```

Листинг 3.2. Алгоритм умножения матриц по Винограду

```
1 def vinograd(n, m, k, matr1, matr2, res):
2     mulH = [0] * n
3     mulV = [0] * k
4
5     i = 0
6     while i < n:
7         j = 0
8         while j < m // 2:
9             mulH[i] = mulH[i] + matr1[i][j * 2] * matr1[i][j * 2 + 1]
10            j += 1
11        i += 1
12
13    i = 0
14    while i < k:
15        j = 0
16        while j < m // 2:
17            mulV[i] = mulV[i] + matr2[j * 2][i] * matr2[j * 2 + 1][i]
18            j += 1
19        i += 1
20
21    i = 0
22    while i < n:
23        j = 0
24        while j < k:
25            res[i][j] = -mulH[i] - mulV[j]
26            l = 0
27            while l < m // 2:
```

```

28         res[i][j] = res[i][j] + (matr1[i][2 * l + 1] + matr2[2 * l][j]) * \
29             (matr1[i][2 * l] + matr2[2 * l + 1][j])
30         l += 1
31     j += 1
32     i += 1
33
34     if m % 2 == 1:
35         i = 0
36         while i < n:
37             j = 0
38             while j < k:
39                 res[i][j] = res[i][j] + matr1[i][m - 1] * matr2[m - 1][j]
40                 j += 1
41             i += 1
42     return res

```

Листинг 3.3. Оптимизированный алгоритм умножения матриц по Винограду

```

1 def vinograd_opt(n, m, k, matr1, matr2, res):
2     mulH = [0] * n
3     mulV = [0] * k
4
5     tmp = m - m % 2
6
7     i = 0
8     while i < n:
9         j = 0
10        while j < tmp:
11            mulH[i] += matr1[i][j] * matr1[i][j + 1]
12            j += 2
13        i += 1
14
15    i = 0
16    while i < k:
17        j = 0
18        while j < tmp:
19            mulV[i] += matr2[j][i] * matr2[j + 1][i]
20            j += 2
21        i += 1
22
23    i = 0
24    while i < n:
25        j = 0
26        while j < k:
27            buff = -mulH[i] - mulV[j]
28            l = 0
29            while l < tmp:
30                buff += (matr1[i][l + 1] + matr2[l][j]) * (matr1[i][l] + matr2[l +

```

```

        1][j])
31         l += 2
32         res[i][j] = buff
33         j += 1
34         i += 1
35
36     if m % 2 == 1:
37         i = 0
38         tmp = m - 1
39         while i < n:
40             j = 0
41             while j < k:
42                 res[i][j] += matr1[i][tmp] * matr2[tmp][j]
43                 j += 1
44             i += 1
45     return res

```

3.4. Трудоемкость алгоритмов

Введем модель вычисления трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1: +, -, =, ==, <=, >=, !=, +=, -=, [] ;
- базовые операции стоимостью 2: *, /, %, *=, /= ;
- оценка трудоемкости цикла: $F_{\text{ц}} = \text{init} + N * (a + F_{\text{тела}} + \text{post}) + a$, 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.

3.4.1. Трудоемкость стандартного алгоритма умножения матриц

Рассмотрим трудоемкость стандартного алгоритма умножения матриц.

Подсчет: $2 + m * (2 + 2 + n * (2 + 2 + q * (2 + 6 + 2)))$

Итог: $10 * m * n * q + 4 * m * n + 4 * m + 2$, где m, n, q – константы для матриц A с размерами $m * n$ и B с размерами $n * q$

3.4.2. Трудоемкость алгоритма умножения матриц по Винограду

Рассмотрим трудоемкость алгоритма умножения матриц по Винограду.

Инициализация mulH и mulV : $2 * 3$

Заполнение mulH : $2 + m * (2 + 2 + n / 2 * (3 + 6 + 6))$

Заполнение mulV : $2 + q * (2 + 2 + n / 2 * (3 + 6 + 6))$

Подсчет результата: $2 + m * (2 + 2 + q * (2 + 7 + 2 + n / 2 * (3 + 23)))$

Условие нечетности n :
$$\begin{bmatrix} 2 & , \text{ невыполнение условия} \\ 2 + m * (2 + 2 + q * (2 + 8 + 5)) & , \text{ выполнение условия} \end{bmatrix}$$

Итог: $13 * m * n * q + 7.5 * m * n + 7.5 * n * q + 11 * m * q + 8 * m + 4 * q + 14 +$
 $+ \begin{bmatrix} 2 \\ 15 * m * q + 4 * m + 2 \end{bmatrix}$, где m, n, q – константы для матриц A с размерами $m * n$ и B с размерами $n * q$.

3.4.3. Трудоемкость оптимизированного алгоритма умножения матриц по Винограду

Рассмотрим трудоемкость оптимизированного алгоритма умножения матриц по Винограду.

Инициализация mulH и mulV: $2 * 3$

Переменная tmp: 3

Заполнение mulH: $2 + m * (2 + 2 + n / 2 * (2 + 5 + 3))$

Заполнение mulV: $2 + q * (2 + 2 + n / 2 * (2 + 5 + 3))f$

Подсчет результата: $2 + m * (2 + 2 + q * (2 + 5 + 3 + 2 + n / 2 * (2 + 14)))$

Условие нечетности n:
$$\begin{bmatrix} 2 & , \text{ невыполнение условия} \\ 2 + 2 + m * (2 + 2 + q * (2 + 6 + 2)) & , \text{ выполнение условия} \end{bmatrix}$$

Итог: $8 * m * n * q + 5 * m * n + 5 * n * q + 12 * m * q + 8 * m + 4 * q + 17 +$
 $+ \begin{bmatrix} 2 \\ 10 * m * q + 4 * m + 2 \end{bmatrix}$, где m, n, q – константы для матриц A с размерами $m * n$ и B с размерами $n * q$.

3.5. Тестирование

Реализовано модульное тестирование отдельным файлом test.py с помощью библиотеки unittest [3]. Полученные результаты функций сравниваются с контрольными значениями.

На листингах 3.4-3.5 продемонстрированы функции, использованные для тестирования

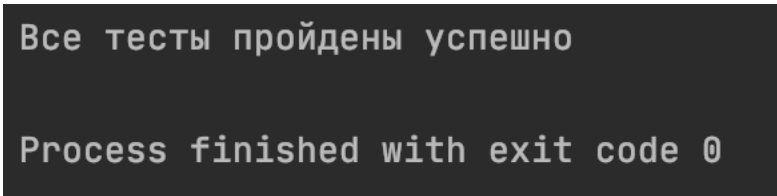
Листинг 3.4. Функция сравнения матриц

```
1 def comp_matr(matr1, matr2):
2     if len(matr1) != len(matr2):
3         return False
4     if len(matr1) == 0 or len(matr1[0]) != len(matr2[0]):
5         return False
6     for i in range(len(matr1)):
7         for j in range(len(matr1[i])):
8             if matr1[i][j] != matr2[i][j]:
9                 return False
10    return True
```

Листинг 3.5. Функция тестирования двух алгоритмов

```
1 iteration = 10
2 for _ in range(iteration):
3     size = [randint(1, iteration), randint(1, iteration), randint(1, iteration
4         )]
5     for _ in range(iteration):
6         if not comp(size[0], size[1], size[2], func1, func2):
7             err += 1
8 if err:
9     print('Errors: ', err)
10 else:
11     print('Success')
```

Программа успешно прошла все тестовые случаи, см. рис. 3.1.



```
Все тесты пройдены успешно
Process finished with exit code 0
```

Рис. 3.1. Тестирование программы

4 | Экспериментальная часть

В данном разделе приведены примеры работы программы и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1. Примеры работы

На рис. 4.1 представлено главное меню программы.

```
Меню:
1) Перемножение случайных матриц заданного размера (Классическое)
2) Перемножение случайных матриц заданного размера (Виноград)
3) Перемножение случайных матриц заданного размера (оптимизированный Виноград)
4) Сравнение по времени
Выберите пункт меню:
```

Рис. 4.1. Главное меню программы

На рис. 4.2-4.3 приведены примеры работы программы при выборе пункта меню 1.

```
Выберите пункт меню: 1
[ Матрица A [N,M] ]
[ Матрица B [M,K] ]

введите N: 5
введите M: 3
введите K: 2
```

Рис. 4.2. Выбор пункта меню 1 и ввод данных для создания матриц

```
Матрица A:
1 3 9
4 8 5
7 5 8
8 3 1
0 0 6
Матрица B:
4 0
7 6
0 4
Результирующая матрица:
25 54
72 68
63 62
53 22
0 24
```

Рис. 4.3. Полученный результат умножения

На рис. 4.4 приведены примеры работы программы при выборе пункта меню 4.

```
Выберите пункт меню: 4
*введите итерации: 100
*введите размер: 10
Размер матриц: 10
Время на создание и заполнение матриц : 0.00031569
Классическое умножение : 0.00073619
Умножение по Винограду : 0.00075060
Умножение по оптимизированному Винограду: 0.00051826
P.S. все результаты получены с учетом времени на создание и инициализации матриц
```

Рис. 4.4. Анализ по времени работы алгоритмов

4.2. Постановка эксперимента по замеру времени

Для произведения замеров времени выполнения реализации алгоритмов будет использована формула:

$$t = \frac{T}{N} \quad (4.1)$$

где t — среднее время выполнения алгоритма, N — количество замеров, T — время выполнения N замеров. Неоднократное измерение времени необходимо для получения более точного результата.

Количество замеров взято равным 100. Эксперимент проводится на квадратных матрицах, заполненных произвольными значениями.

Был проведен замер времени работы каждого из алгоритмов, результат представлен в таблице 4.1.

Таблица 4.1. Сравнение алгоритмов по времени

Алгоритм умножения	Размер матриц	Время работы, сек
Стандартный	2	0.00001297
По Винограду	2	0.00002456
По Винограду оптимизированный	2	0.00001859
Стандартный	3	0.00003906
По Винограду	3	0.00005213
По Винограду оптимизированный	3	0.00004844
Стандартный	4	0.00009531
По Винограду	4	0.00014245
По Винограду оптимизированный	4	0.00010312
Стандартный	5	0.00017656
По Винограду	5	0.00022748
По Винограду оптимизированный	5	0.00018906
Стандартный	6	0.00030469
По Винограду	6	0.00034123
По Винограду оптимизированный	6	0.00030937
Стандартный	7	0.00047813
По Винограду	7	0.00054623
По Винограду оптимизированный	7	0.00046406
Стандартный	10	0.00136250
По Винограду	10	0.00155625
По Винограду оптимизированный	10	0.00119062
Стандартный	25	0.02106250
По Винограду	25	0.02340625
По Винограду оптимизированный	25	0.01775000
Стандартный	50	0.16750000
По Винограду	50	0.17093750
По Винограду оптимизированный	50	0.12218750

Вывод: оптимизированный алгоритм умножения матриц по Винограду показывает наилучшее время на средних размерах, однако неоптимизированный алгоритм умножения матриц по Винограду показывает наихудшее время. Это связано с тем, что обычный алгоритм мало оптимизирован и приходится вычислять одни и те же значения несколько раз. Стандартный алгоритм умножения работает быстрее на малых размерах. Выбирать алгоритм необходимо в зависимости от поставленной задачи. Если требуется быстро перемножать матрицы больших размеров, то оптимизированный алгоритм умножения матриц по Винограду справится с этой задачей быстрее. Если необходимо работать с матрицами малых размеров (< 7), то следует выбирать стандартный алгоритм умножения матриц.

Заключение

В ходе работы были изучены алгоритмы умножения матриц. Реализованы 3 алгоритма, приведен программный код реализации алгоритмов по умножению матриц.

Была подсчитана трудоемкость каждого из алгоритмов. А также было проведено сравнение алгоритмов по времени и трудоемкости.

Цель работы достигнута. Получены практические навыки реализации алгоритмов Винограда и стандартного алгоритма, а также проведена исследовательская работа по оптимизации и вычислению трудоемкости алгоритмов.

Список литературы

1. Курош А. Г. Курс высшей алгебры. — 9-е изд. — М.: Наука, 1968. — 432 с
2. Официальный сайт Python, документация [электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html>, свободный (Дата обращения: 16.09.20)
3. Официальный сайт Python, документация [электронный ресурс] — Режим доступа: <https://docs.python.org/3/library/unittest.html>, свободный — (Дата обращения: 16.09.20)