

Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: АНАЛИЗ АЛГОРИТМОВ

ЛАБОРАТОРНАЯ РАБОТА № 2

Трудоемкость алгоритмов умножения матриц

Студент:

Сиденко Анастасия Геннадьевна

Группа: ИУ7-53Б

Преподаватели:

Строганов Юрий Владимирович

Волкова Лилия Леонидовна

2019 г.

Содержание

Введение	2
1 Аналитическая часть	3
1.1 Описание задачи	3
1.2 Пути решения	3
1.3 Выводы	4
2 Конструкторская часть	5
2.1 Функциональная модель	5
2.2 Схемы алгоритмов	5
2.3 Выводы	9
3 Технологическая часть	10
3.1 Требования к программному обеспечению	10
3.2 Средства реализации	10
3.3 Листинг кода	10
3.4 Тестирование	11
3.5 Выводы	11
4 Экспериментальная часть	12
4.1 Примеры работы	12
4.2 Результаты тестирования	12
4.3 Замеры времени	13
4.4 Выводы	14
Заключение	15

Введение

Матрицы упоминались ещё в древнем Китае, называясь тогда «волшебным квадратом». Основным применением матриц было решение линейных уравнений. Также волшебные квадраты были известны чуть позднее у арабских математиков, примерно тогда появился принцип сложения матриц. Сама теория матриц начала своё существование в середине XIX века. Термин «матрица» ввел Джеймс Сильвестр в 1850 г. Сегодня матрицы применяются уже не только при решении линейных уравнений. [1]

Умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения. Многие реализации прямого и обратного распространения сигнала в сверточных слоях нейронной сети базируются на этой операции. [2] В физике и других прикладных науках матрицы — являются средством записи данных и их преобразования. [4] В программировании — в написании программ, массивы. Широкое применение в компьютерной графике: любая картинка на экране — это двумерная матрица, элементами которой являются цвета точек, а также матрицы используются для преобразования фигур. [5]

В психологии понимание термина сходно с данным термином в математике, но взамен математических объектов подразумеваются некие "психологические объекты"— например, тесты. [6]

Кроме того, умножение матриц имеет широкое применение в экономике[7], биологии[8], химии[9].

Также существует абстрактная модель — теорию бракосочетаний в первобытном обществе, где с помощью матриц были показаны разрешенные варианты браков для представителей и даже потомков того или иного племени. [3]

В данной лабораторной работе ставятся следующие задачи.

1. Изучение алгоритмов умножения матриц: стандартный, Винограда и Винограда с оптимизациями.
2. Оценка трудоемкости алгоритмов умножения матриц. Теоретическая оценка лучших и худших случаев с условием их наступления.
3. Получение практических навыков реализации данных алгоритмов на одном из языков программирования.
4. Сравнительный анализ алгоритмов по затрачиваемым ресурсам (зависимость времени от длины строки).
5. Экспериментальное подтверждение различий в трудоемкости алгоритмов с указанием лучшего и худшего случаев.

1 Аналитическая часть

Умножение матриц – это одна из основных вычислительных операций. Вычислительная сложность стандартного алгоритма умножения матриц порядка N составляет $O(N^3)$. Но существуют более сложные алгоритмы, которые дают лучший результат. Целью данной работы является сравнение стандартного алгоритма и алгоритма Винограда.

1.1 Описание задачи

Пусть даны две прямоугольные матрицы $A[M \times N]$ и $B[N \times Q]$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nq} \end{bmatrix}$$

Тогда матрица $C[M \times Q]$ – произведение матриц:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{bmatrix},$$

в которой каждый элемент вычисляется по формуле 1:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}, \quad (i = 1, 2, \dots, l; j = 1, 2, \dots, n) \quad (1)$$

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 данной работе необходимо оценить и подтвердить трудоемкость алгоритмов, обратимся к определению.

Трудоемкость алгоритма – это зависимость количества операций от объема обрабатываемых данных. Модель вычислений:

1. Цена единичных операций. Пусть у следующих операций трудоемкость равна 1:

$$+, -, *, /, \%, =, ==, !=, <, <=, >, >=, [], + =$$

2. Трудоемкость условного перехода примем за единицу, при этом условие вычисляется по пункту 1.
3. Трудоемкость циклов, например цикла for:

$$f_{for} = f_{init} + f_{comp} + N * (f_{body} + f_{inc} + f_{comp})$$

1.2 Пути решения

Сложность вычисления произведения матриц порядка N по определению составляет $O(N^3)$, однако существуют более эффективные алгоритмы, применяющиеся для перемножения матриц.

Первый алгоритм быстрого умножения больших матриц был разработан Фолькером Штрассеном[10] в 1969. На основе данного алгоритма Штрассена разработаны другие, которые улучшают его трудоемкость, однако в силу простоты именно алгоритм Штрассена остаётся одним из практических алгоритмов умножения больших матриц.

В дальнейшем было разработано еще множество различных алгоритмов. Однако эти алгоритмы носили теоретический, в основном приближенный характер. В силу неустойчивости алгоритмов приближенного умножения в настоящее время они не используются на практике.

В 1990 Копперсмит и Виноград[11] опубликовали алгоритм, и на сегодняшний день алгоритм Винограда является наиболее быстрым. Именно этот алгоритм и его оптимизация выбраны для дальнейшего исследования в данной работе.

1.3 Выводы

В данной работе стоит задача реализации 3 алгоритмов умножения матриц. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

Реализуем данные алгоритмы и проверим наши предположения экспериментально.

2 Конструкторская часть

В данной работе для нахождения произведения матриц используются алгоритмы стандартный, Винограда и Винограда с оптимизациями. Необходимо рассмотреть и изучить данные варианты реализации.

2.1 Функциональная модель

На рисунке 1 представлена функциональная модель нашей задачи.

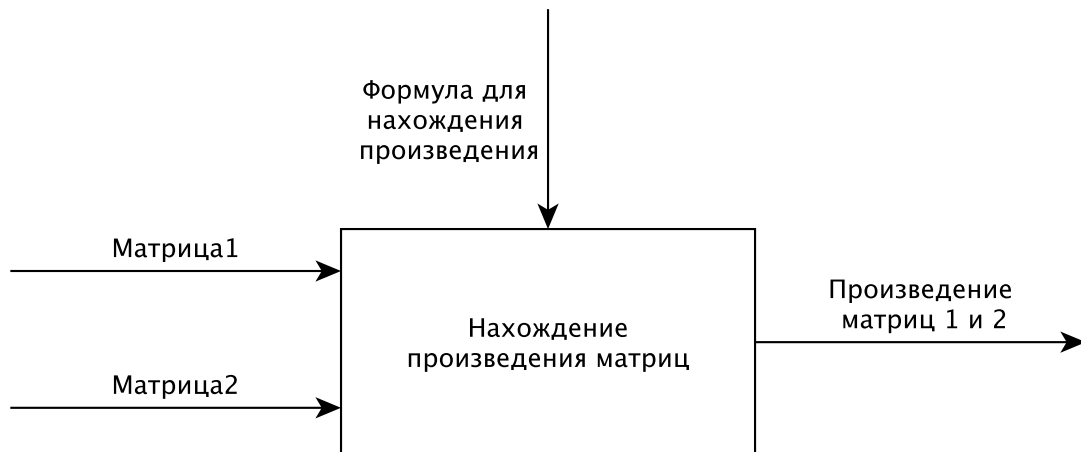


Рис. 1 - Функциональная модель алгоритма нахождения произведения матриц.

2.2 Схемы алгоритмов

Приведем схемы 3 исследуемых алгоритмов (см. рисунки 2-4).

Стандартный алгоритм

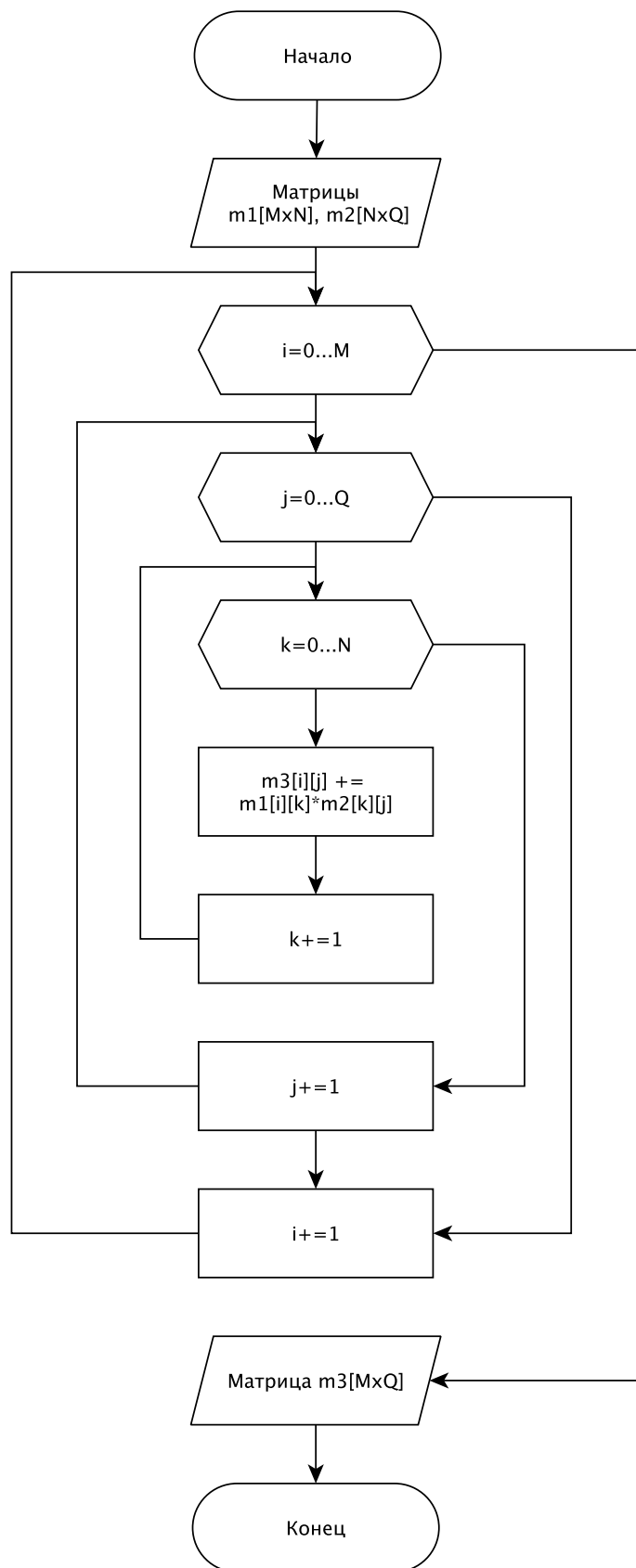


Рис. 2 - Алгоритм нахождения произведения матриц стандартным методом

Алгоритм Винограда

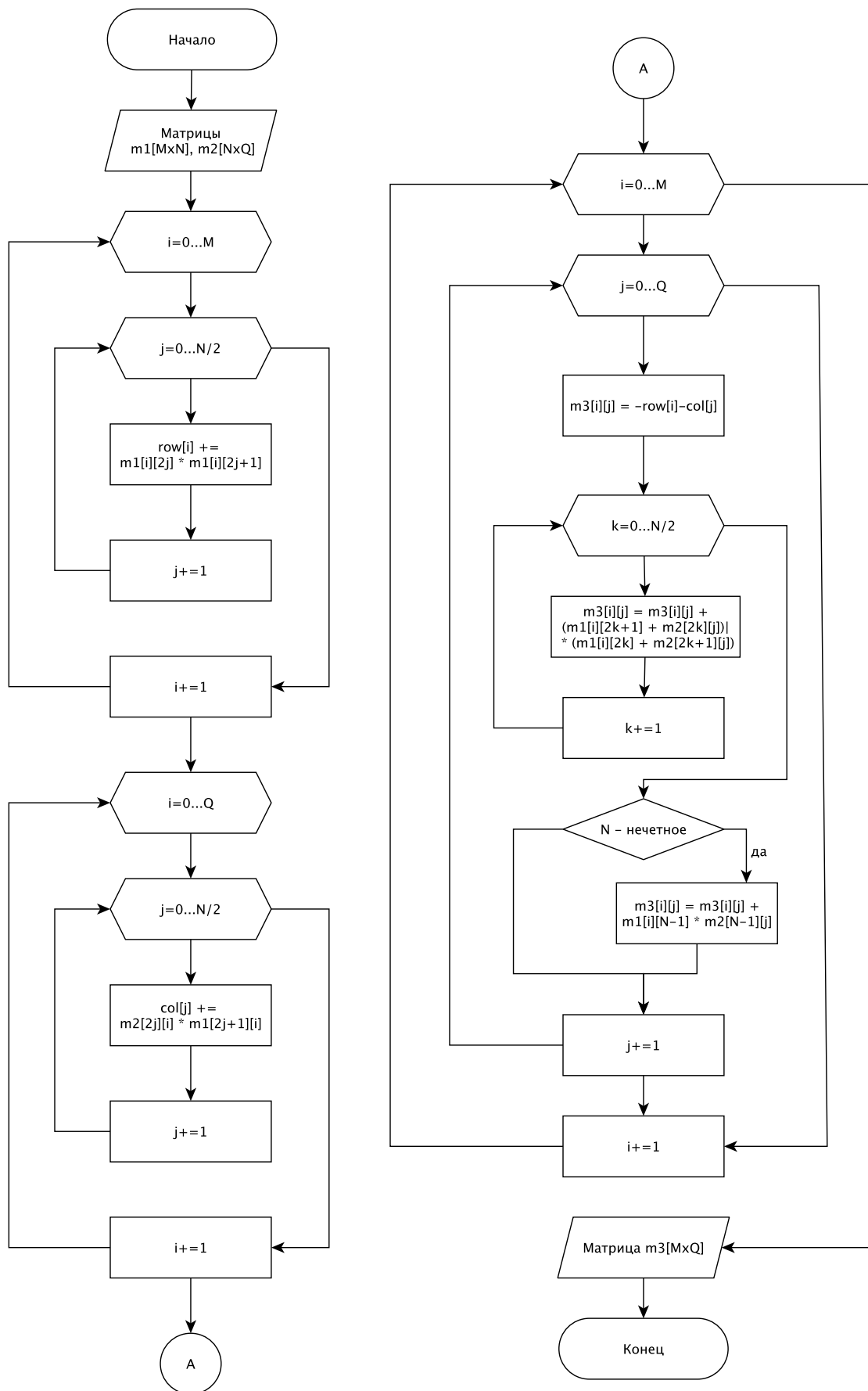


Рис. 3 - Алгоритм нахождения произведения матриц методом Винограда

Алгоритм Винограда с оптимизациями

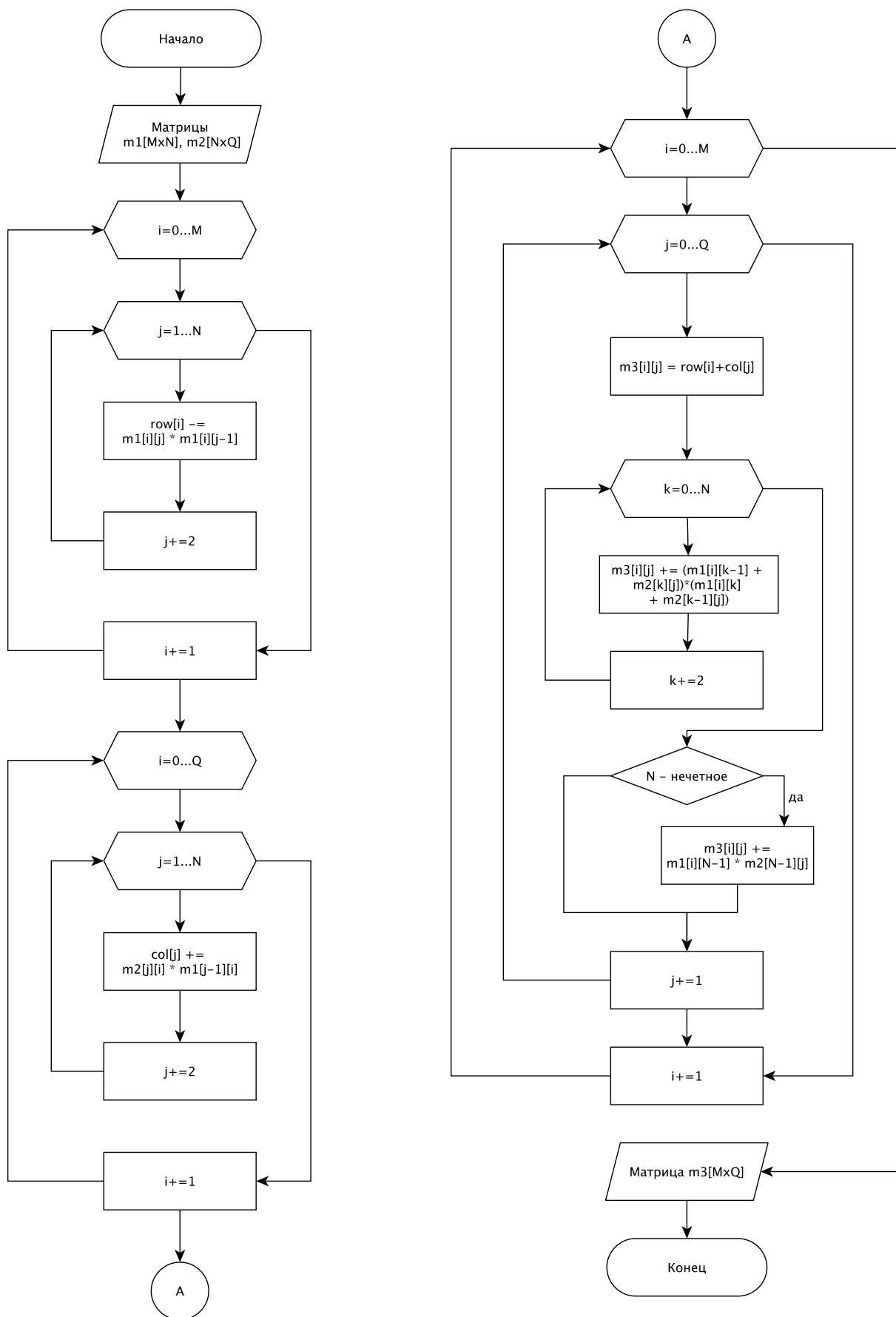


Рис. 4 - Алгоритм нахождения произведения матриц методом Винограда с оптимизациями

Произведем теоретическую оценку трудоемкости алгоритмов умножения матриц

1. Стандартный алгоритм

$$f = 2 + M(2 + 2 + Q(2 + 2 + N(2 + 8 + 1 + 1 + 1))) = 13MNQ + 4MQ + 4M + 2$$

Оценка трудоемкости приближается к наиболее растущим слагаемым, здесь куб линейного размера матриц.

2. Алгоритм Винограда

Предназначен для снижения доли умножения

$$c_{ij} = \underbrace{u_1 u_2 u_3 u_4}_{\vec{u}} * \underbrace{\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}}_{\vec{v}} = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4 = (u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - \underbrace{u_1 u_2 - u_3 u_4 - v_1 v_2 - v_3 v_4}_{\text{Вычисляется заранее для строк}}$$

Таким образом, трудоемкость:

$$f = 2 + M(2 + 2 + Q(2 + 4 + 3 + 3 + N/2(3 + 12 + 11))) = 13MNQ + 12MQ + 4M + 2$$

Доля умножения меньше чем в стандартном.

3. Алгоритм Винограда с соответствующими оптимизациями

Введем оптимизации:

- (a) +=
- (b) j < N, j += 2
- (c) Считаём суммы уже отрицательными

Тогда трудоемкость:

$$f = 2 + M(2 + 2 + Q(2 + 6 + 2 + N/2(2 + 3 + 7 + 6))) = 9MNQ + 10MQ + 4M + 2$$

Доля умножения меньше чем в стандартном.

2.3 Выводы

Несмотря на сложность алгоритма Винограда по сравнению со стандартным, доля умножения в алгоритме Винограда меньше и по расчетам из аналитической части, трудоемкость меньше.

Также, необходимо обратить внимание на то, что при работе алгоритма Винограда с матрицами нечетной размерности, необходимо произвести дополнительные действия, в то время как алгоритм стандартный не зависит от четности размерности матриц.

Необходимо разработать данные алгоритмы и убедиться в корректности наших предположений.

3 Технологическая часть

Стоит задача разработки и сравнительного анализа алгоритмов, вычисляющих произведения матриц.

В реализациях в целях увеличения точности подсчета времени вывод матрицы был вынесен за пределы функций-алгоритмов. В целях наглядности были опущены части программ, не относящиеся к работе алгоритмов.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность замеров процессорного времени выполнения реализации каждого алгоритма. Требуется провести замеры для варьирующихся размеров матриц: от 100 до 1000 и от 101 до 1001. Один эксперимент ставится не менее 100 раз, результат одного эксперимента рассчитывается как среднее значение результатов проведенных испытаний с одинаковыми входными данными.

3.2 Средства реализации

В качестве языка программирования был выбран Python[8], так как я знакома с этим языком программирования.

Для замеров времени была выбран метод `process_time()`, возвращает текущее время процессора как число с плавающей запятой, выраженное в секундах в Unix.

Для генерации случайных матриц, заданного размера использовался метод `randint()`.

3.3 Листинг кода

Стандартный алгоритм:

```
1 for i in range(0, m):
2     for j in range(0, q):
3         for k in range(0, n):
4             m3[i][j] = m3[i][j] + m1[i][k] * m2[k][j]
```

Алгоритм Винограда:

```
1 row = [0] * m
2 for i in range(0, m):
3     for j in range(0, n // 2, 1):
4         row[i] = row[i] + m1[i][2 * j] * m1[i][2 * j + 1]
5
6 col = [0] * q
7 for j in range(0, q):
8     for i in range(0, n // 2, 1):
9         col[j] = col[j] + m2[2 * i][j] * m2[2 * i + 1][j]
10
11 start_time = time.process_time()
12 for i in range(0, m):
13     for j in range(0, q):
14         m3[i][j] = -row[i] - col[j]
15         for k in range(0, n // 2, 1):
16             m3[i][j] = m3[i][j] + (m1[i][2 * k + 1] + m2[2 * k][j])
17             * (m1[i][2 * k] + m2[2 * k + 1][j])
18         if 1 == n % 2:
19             m3[i][j] = m3[i][j] + m1[i][n - 1] * m2[n - 1][j]
```

Алгоритм Винограда с оптимизациями:

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

```

4         row[i] -= m1[i][j] * m1[i][j - 1]
5
6     col = [0] * q
7     for j in range(0, q):
8         for i in range(1, n, 2):
9             col[j] -= m2[i][j] * m2[i - 1][j]
10
11     start_time = time.process_time()
12     for i in range(0, m):
13         for j in range(0, q):
14             m3[i][j] = row[i] + col[j]
15             for k in range(1, n, 2):
16                 m3[i][j] += (m1[i][k - 1] + m2[k][j]) * (m1[i][k] + m2[k - 1][j])
17             if 1 == n % 2:
18                 m3[i][j] += m1[i][n - 1] * m2[n - 1][j]

```

3.4 Тестирование

В таблице 1 представлена заготовка данных для тестирования наших алгоритмов.

Матрица1	Матрица2	Ожидаемый результат		
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	30	36	42
		66	81	96
		102	126	150
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	Матрицы не могут быть перемножены		
$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	9	12	15
		24	33	42
		39	54	69

Таблица 1. Подготовленные тестовые данные.

3.5 Выводы

Реализовано 3 алгоритма, подготовлены тесты для оценки качества их работы.

Получены практические навыки реализации алгоритмов матричного умножения: стандартного и Винограда.

4 Экспериментальная часть

4.1 Примеры работы

На рисунке 5 представлены примеры работы программы на разных входных данных.

```
x + 3_course/Algorithm_analysis/lab2 lab2+ python3 code.py
Перемножение 2 матриц - 1, графики - 2?
1
Введите размерность матрицы 1
2
Введите размерность матрицы 2
3
-1 8 -9
6 -1 1

8 0 7
0 1 4
-6 1 -1
5 0 -9

Матрицы не могут быть перемножены
Матрицы не могут быть перемножены
Матрицы не могут быть перемножены

x + 3_course/Algorithm_analysis/lab2 lab2+ python3 code.py
Перемножение 2 матриц - 1, графики - 2?
1
Введите размерность матрицы 1
3
Введите размерность матрицы 2
3
-5 4 6
-10 7 -1
10 7 7

-1 -4 10
4 -3 1
-2 2 10

9 20 14
40 17 -103
4 -47 177

9 20 14
40 17 -103
4 -47 177

9 20 14
40 17 -103
4 -47 177

x + 3_course/Algorithm_analysis/lab2 lab2+ python3 code.py
Перемножение 2 матриц - 1, графики - 2?
1
Введите размерность матрицы 1
3
Введите размерность матрицы 2
2
-5 -8
-1 1
9 -3

-5 -6 -1
-2 4 10

41 -2 -75
3 10 11
-39 -66 -39

41 -2 -75
3 10 11
-39 -66 -39

41 -2 -75
3 10 11
-39 -66 -39
```

Рис. 5 - Примеры работы

4.2 Результаты тестирования

Проверяем нашу программу на тестах из таблицы 1. Полученные результаты представлены в таблице 2.

Матрица1	Матрица2	Стандартный	Виноград	Виноград с оптимизациями
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	Матрицы не могут быть перемножены	Матрицы не могут быть перемножены	Матрицы не могут быть перемножены
$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$

Таблица 2. Тестирование программы.

Тесты пройдены

4.3 Замеры времени

На графиках 6-7 представлено сравнение алгоритмов умножения матриц.

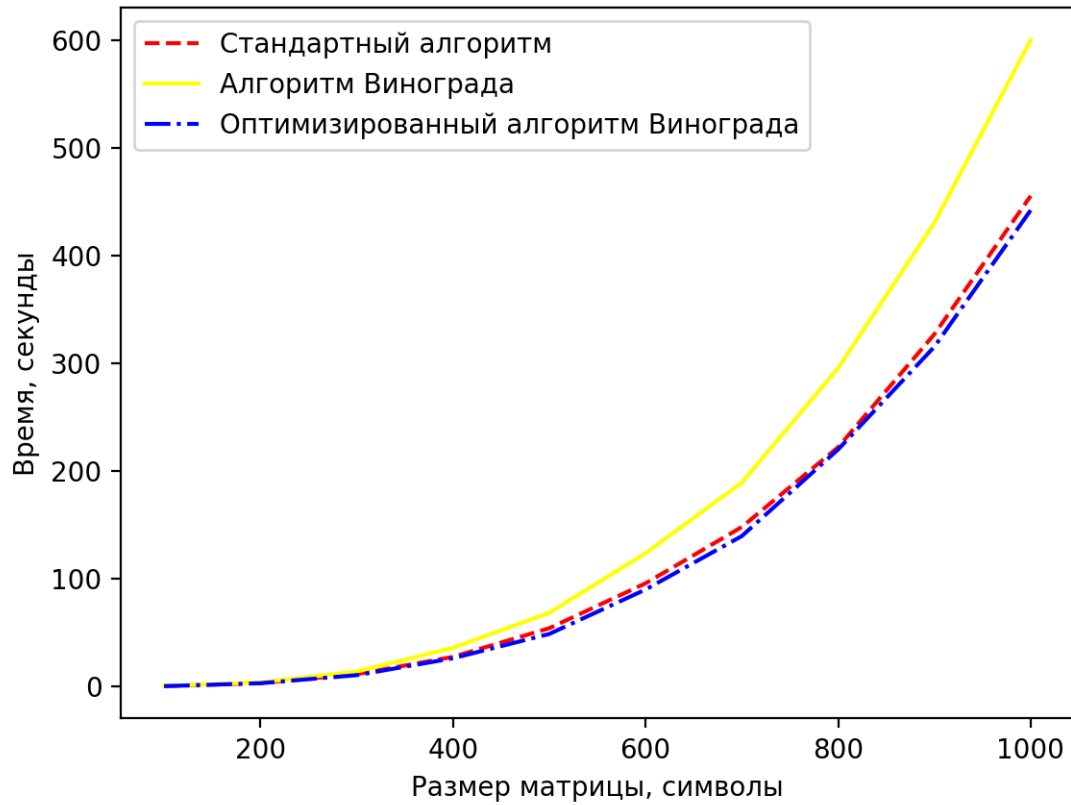


Рис. 6 - Сравнение реализации алгоритмов нахождения произведения матриц при четных размерностях

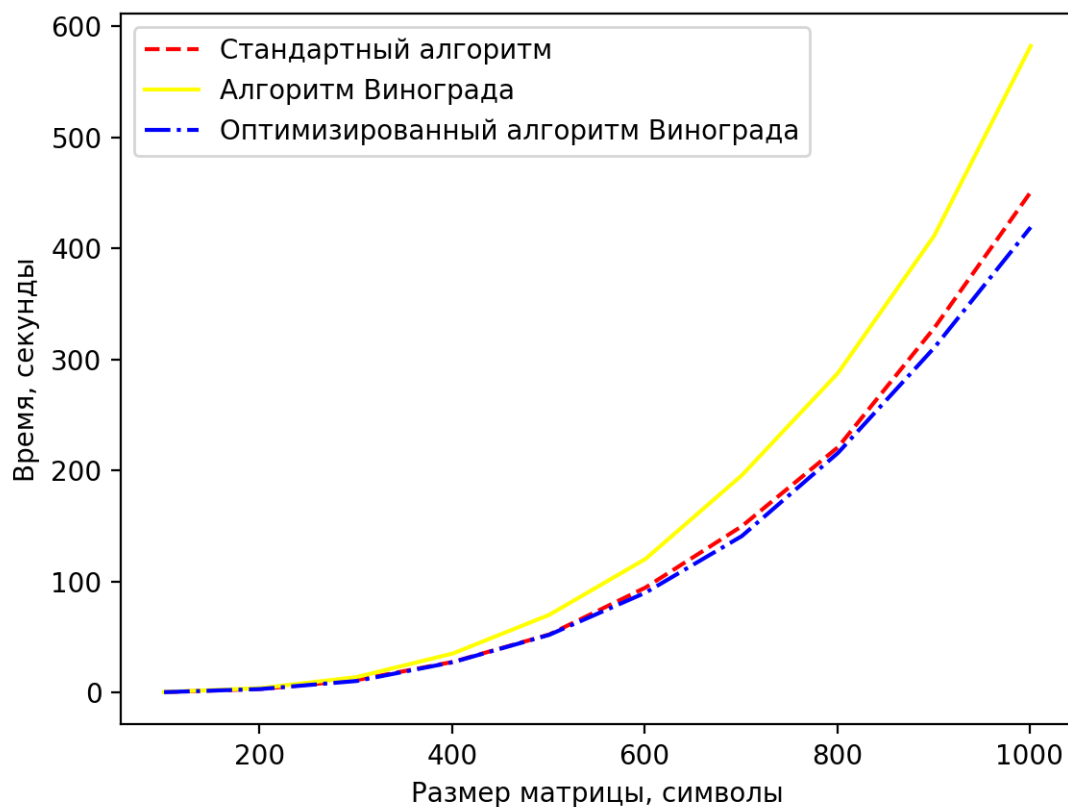


Рис. 7 - Сравнение реализации алгоритмов нахождения произведения матриц при нечетных размерностях

4.4 Выводы

Как мы видим из графиков предположения о более быстрой работе Стандартного алгоритма по сравнению с Виноградом подтвердились, время вычислений меньше на 25%. Но в плюсах использования алгоритма Винограда, это сокращение операции умножения.

Однако алгоритм Винограда с оптимизациями работает примерно также, как и стандартный, с этим связана оценка трудоемкости операции $+=$, оказывается в языке программирования Python эта операция выполняется гораздо дольше обычного сложения и присваивания.

Также при сравнении работы для лучших случаев у Винограда (лучшие при четном размере матрицы) и худших (при нечетном) разница не значительна.

Пример сравнения скорости работы для цикла с 10000000 итераций представлен на рисунке 8. Из него видно, что трудоемкость операции $+=$ выше на 50 %.

```
+ 3_course/Algorithm_analysis/lab2 lab2+ python3 test.py
Время выполнения a = a + 1 для 10 миллионов итераций: 0.8554270267486572
Время выполнения b += 1 для 10 миллионов итераций: 1.725102186203003
```

Рис. 8 - Сравнение работы операции $+=$ и обычного присваивания со сложением

Заключение

В данной лабораторной работе было реализовано и проанализировано 3 алгоритма нахождения произведения 2 матриц:

1. стандартный алгоритм
2. алгоритм Винограда
3. алгоритм Винограда с модификациями

Матрицы и матричное умножение активно применяется:

1. в сверточных слоях нейронных сетей для реализации прямого и обратного распространения сигнала
2. в физике и математике для записи данных и их преобразования
3. в компьютерной графике, для отображения изображений и их преобразований
4. в психологии, для анализа результатов тестов
5. в экономике, биологии, химии и других науках.

При сравнении данных алгоритмов пришли к следующим выводам:

1. Самым быстрым является алгоритм винограда, опережающий стандартный на 25%.
2. Оптимизированный алгоритм Винограда проигрывает из-за трудоемкости операции $+=$ в Python.

В данной лабораторной работе выполнены следующие задачи.

1. Изучены алгоритмов умножения матриц: стандартный, Винограда и Винограда с оптимизациями.
2. Произведена оценка трудоемкости алгоритмов умножения матриц, лучших и худших случаев с условием их наступления.
3. Получены практические навыки реализации данных алгоритмов на одном из языков программирования.
4. Проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (зависимость времени от длины строки).
5. Экспериментально подтверждено различие в трудоемкости алгоритмов с указанием лучшего и худшего случаев.

Список литературы

- [1] <http://poivs.tspu.ru/ru/Math/Algebra/LinearAlgebra/Matrices>
- [2] <https://habr.com/ru/post/359272/>
- [3] <https://urok.1sept.ru/статьи/637896/>
- [4] <http://we.easyelectronics.ru/Theory/cifrovye-rekursivnye-filtry-chast-1.html>
- [5] <http://window.edu.ru/resource/898/72898/files/stup559.pdf>
- [6] <http://vekkv.ru/holotropnoe-dyhanie-transpersonalnaya-psihologiya/1859/>
- [7] <https://www.eduherald.ru/ru/article/view?id=14118>
- [8] <https://cyberleninka.ru/article/n/matrichnyy-printsip-v-biologii-prognoze-nastoyashee-budushee>
- [9] <http://www.chemicals-el.ru/chemicals-2400-1.html>
- [10] Strassen V. Gaussian Elimination is not Optimal // Numer. Math — Springer Science+Business Media, 1969. — Vol. 13, Iss. 4. — P. 354–356. — ISSN 0029-599X; 0945-3245 — doi:10.1007/BF02165411
- [11] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990.