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: Трудоемкость алгоритмов умножения матриц

Студент Чаушев А.К..

Группа ИУ7-56Б

Оценка (баллы) ____________

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

Москва — 2020 г.

Введение

Матрица [1] – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам.

Произведением матрицы $A_(m \times n)$ на матрицу $B_(n \times k)$ называется матрица $C(m \times k)$ такая, что элемент матрицы С, стоящий в i-ой строке и j-ом столбце, т.е. элемент $C_(i, j)$, равен сумме произведений элементов i-ой строки матрицы А на соответствующие элементы j-ого столбца матрицы В (см. формула 1).

$$C_{ij} = \sum_{p=1}^{m} a_{ip} \cdot b_{pj} \tag{1}$$

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 частности, умножение всегда выполнимо, если оба сомножителя — квадратные матрицы одного и того же порядка.

1 Аналитическая часть

В данной части будут рассмотрены основные теоретические аспекты, связанные с алгоритмами умножения матриц, описания алгоритмов, формулы и оценки сложностей алгоритмов.

1.1 Постановка задачи

Цель данной лабораторной работы: провести сравнительный анализ алгоритмов умножения матриц и получить навык оптимизации алгоритмов. Задачи данной лабораторной работы:

1) дать математическое описание формулы расчета для двух алгоритмов: стандартного и алгоритма Винограда;

2) реализовать стандартный алгоритм умножения матриц и алгоритм Винограда;

3) разработать оптимизированый алгоритм Винограда;

4) дать теоретическую оценку трудоемкости трем алгоритмам;

5) провести замеры процессорного времени работы реализаций всех трех алгоритмов при четных и нечетных размерностях.

1.2 Описание алгоритма

1.2.1 Стандартный алгоритм

Последовательный алгоритм умножения матриц представляется тремя вложенными циклами. Этот алгоритм является итеративным и ориентирован на последовательное вычисление строк матрицы C. Предполагается выполнение n·n·n операций умножения и столько же операций сложения элементов исходных матриц. Количество выполненных операций имеет порядок O(n^3). Поскольку каждый элемент результирующей матрицы есть скалярное произведение строки и столбца исходных матриц, то для вычисления всех элементов матрицы C размером n*n необходимо выполнить (n^2)*(2n-1) скалярных операций и затратить время T1 = (n^2)*(2n-1)*t, где t есть время выполнения одной элементарной скалярной операции.

1.2.2 Умножение 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 (см.формулы 2,3,4). Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора:

$$V = (\nu_1, \nu_2, \nu_3, \nu_4); W = (w_1, w_2, w_3, w_4) \tag{2}$$

Их скалярное произведение равно:

$$V \cdot W = \nu_1 w_1 + \nu_2 w_2 + \nu_3 w_3 + \nu_4 w_4 \tag{3}$$

Это равенство можно переписать в виде:

$$V \cdot W = (\nu_1 + w_2)(\nu_2 + w_1) + (\nu_3 + w_4)(\nu_4 + w_3) - \nu_1 \nu_2 - \nu_3 \nu_4 - w_1 w_2 - w_3 w_4 \tag{4}$$

Вместо четырех умножений (3) в формуле (4) их шесть, а вместо трех сложений - десять. 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. На практике это означает, что над предварительно обработанными элементами можно выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.2.3 Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, их разнца заключается в предварительной обработке скалярного произведения соответсвующих строк и столбцов исходных матриц, а также уменьшение количества операций умножения.

1.3 Вычисление сложности алгоритма

Операции со сложностью "1":

1. Присваивание «=»
2. Сложение «+»
3. Вычитание «−»
4. Унарный плюс «+»
5. Унарный минус «−»
6. Умножение «∗»
7. Деление «/»
8. Взятие остатка от деления «%»
9. Инкемент (постфиксный и префиксный) «++»
10. Декремент (постфиксный и префиксный) «−−>

11. Индексация (обращение к элементу массива) «[]»
12. Присваивание со сложением «+ =»
13. Равенство «==»
14. Неравенство «! =»
15. Больше «>»
16. Меньше «<»
17. Больше или равно «>=»
18. Меньше или равно «<=»
19. Логическое отрицание «!»
20. Логическое умножение «&&»
21. Логическое сложение «||»
22. Побитовая инверсия « »
23. Побитовое И «&»
24. Побитовое ИЛИ «|»
25. Присваивание со сложением «+ =»
26. Присваивание с вычитанием «− =»
27. Присваивание с умножением «∗ =»
28. Присваивание с делением «/ =»
29. Присваивание со взятием остатка от деления «% =»

Условный оператор:

```
if (условие)
{
// тело условия A
}
else
{
// тело условия B
}
```

Пусть стоимость перехода к одной из ветвей решения равна 0, тогда трудоемкость if при $f_{min} = min(f_a, f_b), f_{max} = max(f_a, f_b)$, получим

$$f_{if} = f + \left[\begin{array}{l} D(S_1[1..i], S_2[1..j-1]) + 1 \\ D(S_1[1..i-1], S_2[1..j]) + 1 \end{array} \right. \qquad (5)$$

Цикл со счетчиком:

```
for (int i = 0; i < n; ++i)
{
// тело цикла
}
```

Начальная инициализация int i = 0 выполняется всего один раз - 1 операция Условие $i < n$ проверяется перед каждой итерацией и при входе в цикл – $n + 1$ операций. Тело цикла выполняется n раз – $n * f$ Изменение счетчика $++i$ выполняется на каждой итерации, но перед проверкой условия - n операций. Итого сложность цикла со счетчиком: $2 + n * (2 + f)$, где f - сложность тела цикла.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 1 - 9 представлены схемы алгоритмов умножения матриц: стандартного, Винограда и оптимизированного алгоритма Винограда.

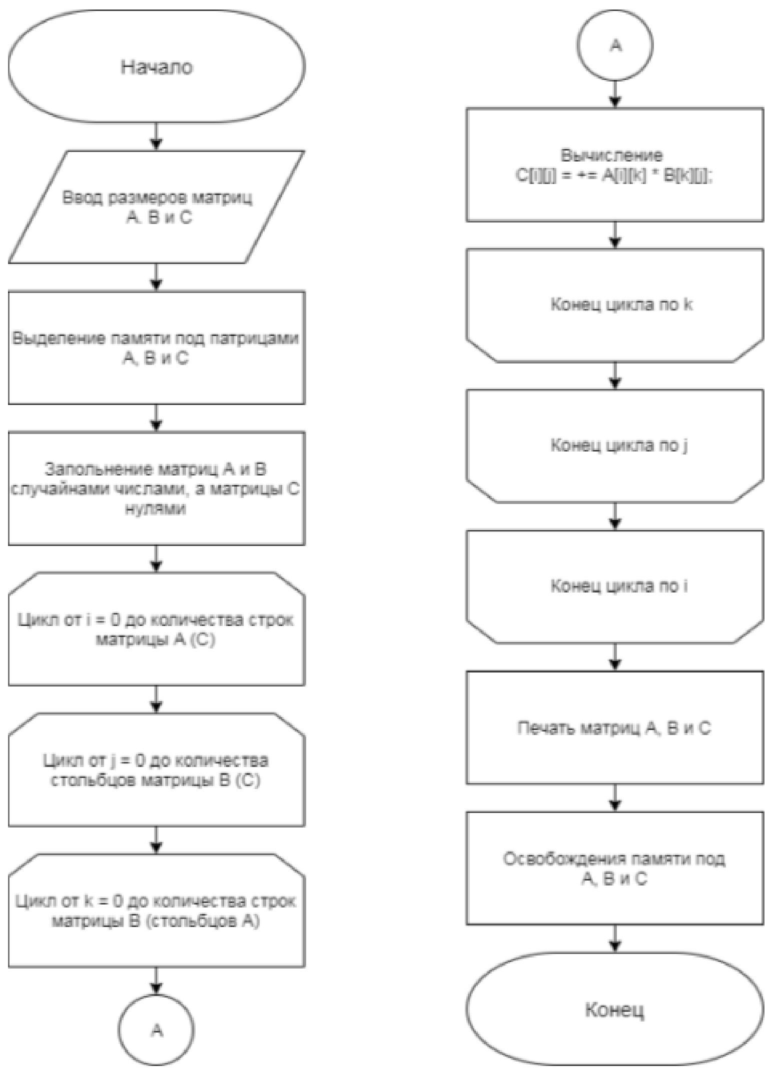

Рисунок 1 – Схема алгоритм нахождения произведения матриц стандартным методом

На рис. 2 - 5 дана схема алгоритма Винограда умножения матриц (без оптимизаций)

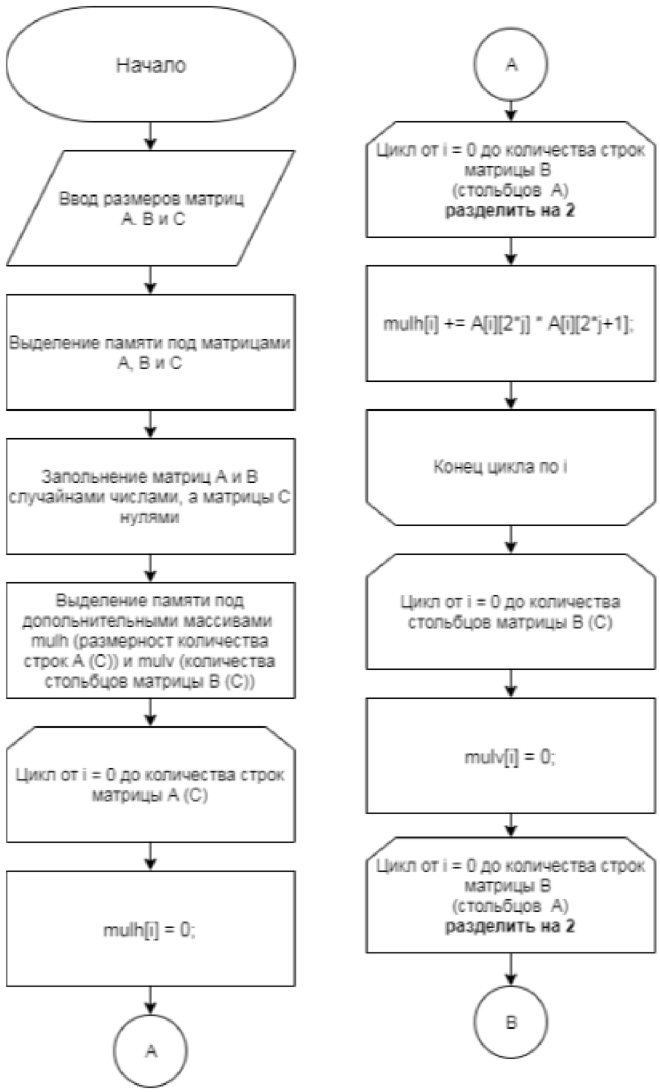

Рисунок 2 – Схема алгоритм нахождения произведения матриц методом Винограда Часть 1

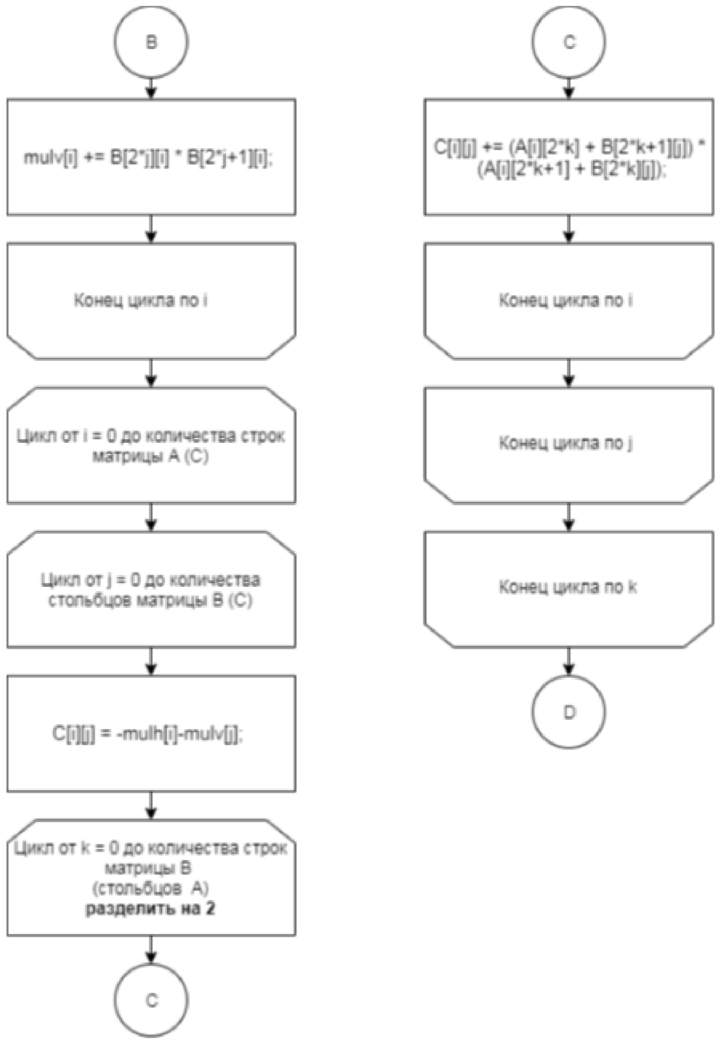

Рисунок 3 – Схема алгоритм нахождения произведения матриц методом Винограда Част 2

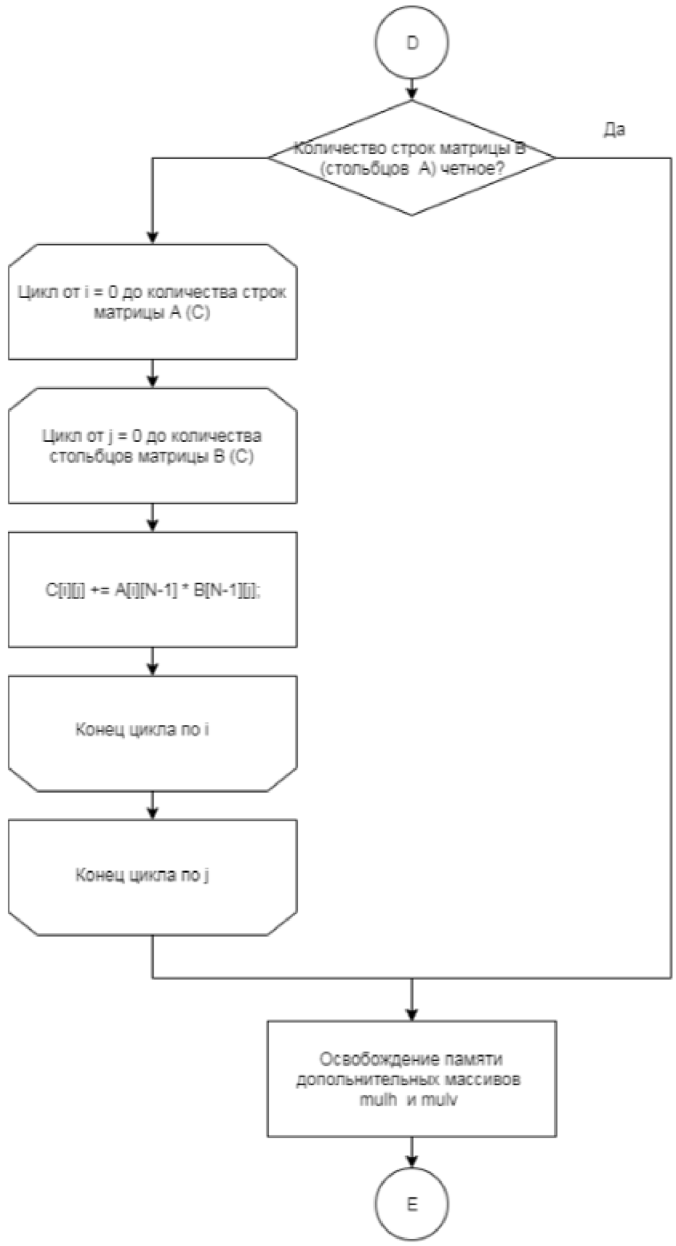

Рисунок 4 – Схема алгоритм нахождения произведения матриц методом Винограда Част 3

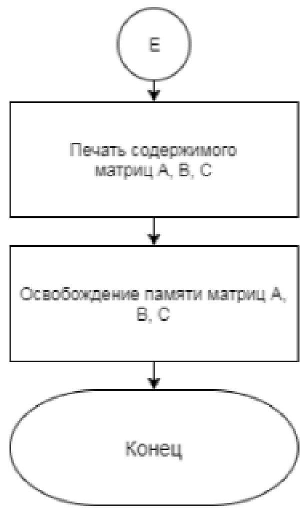

Рисунок 5 – Схема алгоритм нахождения произведения матриц методом Винограда Част 4

На рис. 6 - 9 дана схема алгоритма Винограда умножения матриц (с оптимизациями)

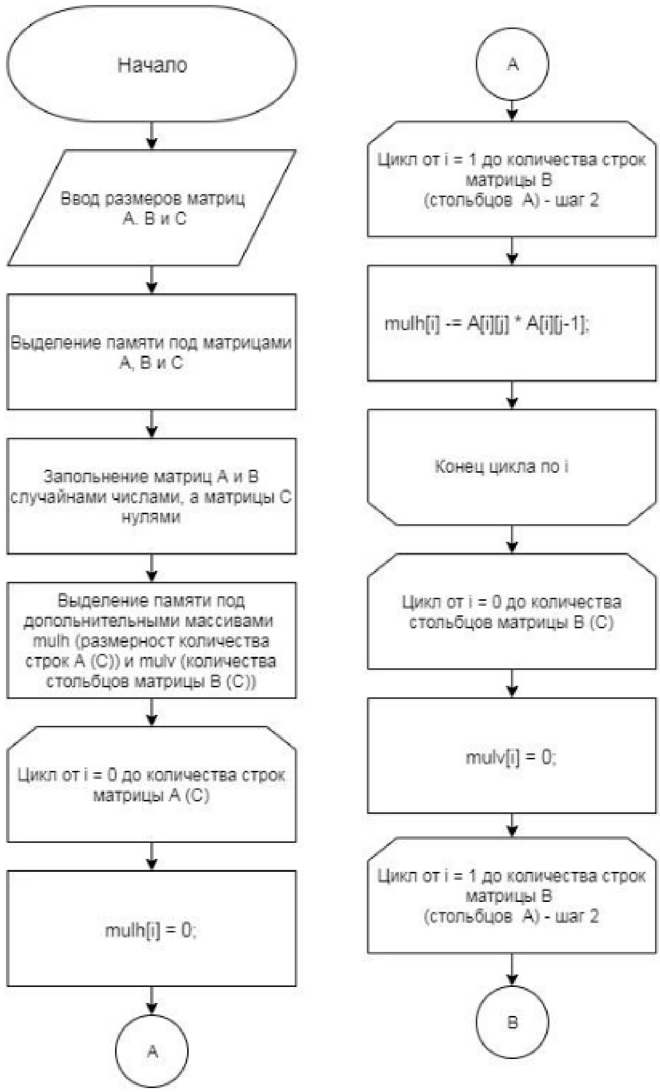

Рисунок 6 – Схема алгоритм нахождения произведения матриц оптимизированным методом Винограда Част 1

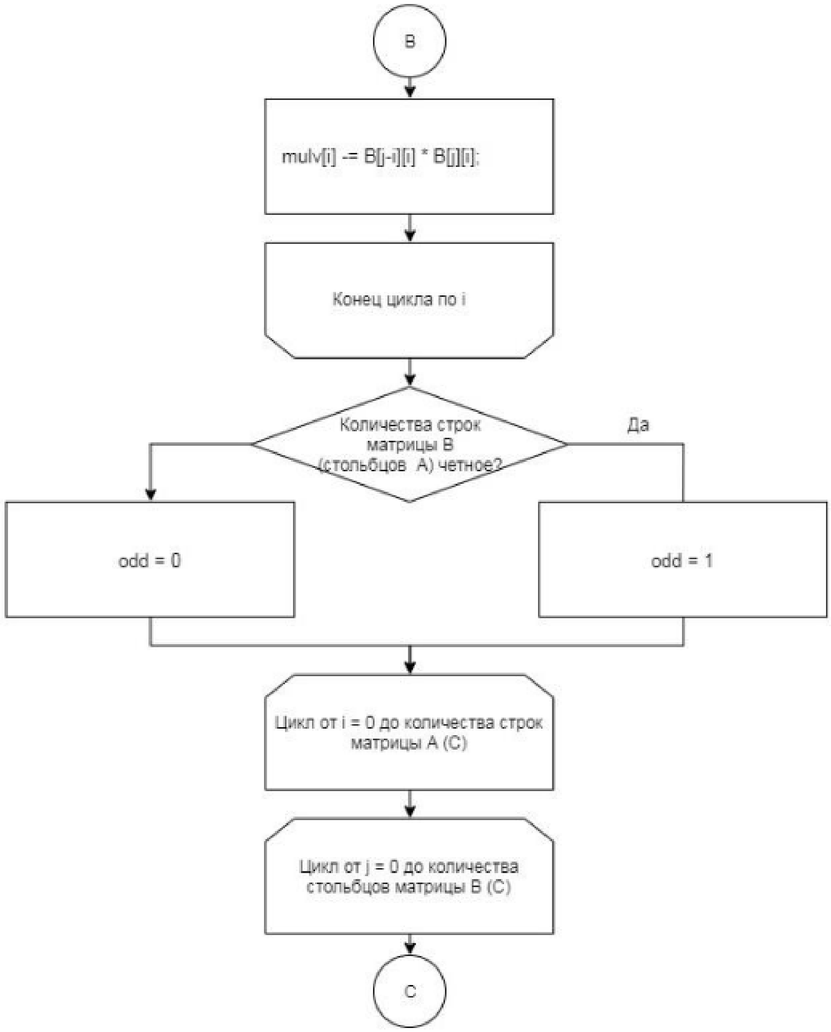

Рисунок 7 – Схема алгоритм нахождения произведения матриц оптимизированным методом Винограда Част 2

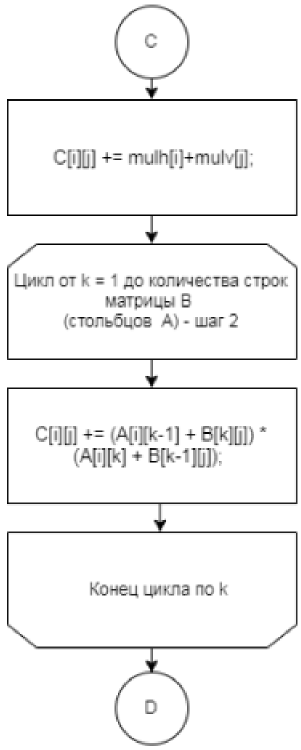

Рисунок 8 – Схема алгоритм нахождения произведения матриц оптимизированным методом Винограда Част 3

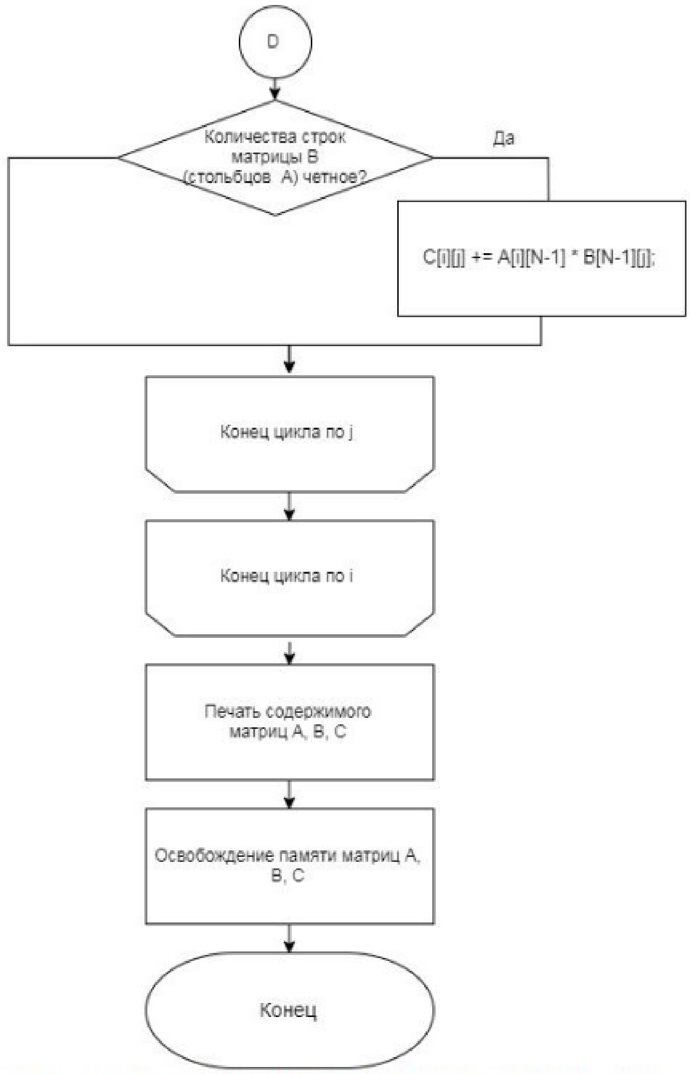

Рисунок 9 – Схема алгоритм нахождения произведения матриц оптимизированным методом Винограда Част 4

3 Технологическая часть

3.1 Выбор языка - Java

Java [3] — строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process, язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. На 2019 год Java — один из самых популярных языков программирования.

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

3.2 Методы замера времени в программе

3.2.1 Время

На листингах 3.2.1 – 3.2.1 показаны методы измерения времени в джаве.

Листинг 1 – Пример System.currentTimeMillis()

```
class Time {
    public static long getCurrentTime() {
        return System.currentTimeMillis();
    }
```

Единственным недостатком этого подхода является то, что "реальное"время doSomething(), которое требуется выполнить, может сильно различаться в зависимости от того, какие другие программы работают в системе и какова его нагрузка. Это делает измерение производительности несколько неточным.

Листинг 2 – Пример ThreadMXBean. [2],

```
public class CPUTime {
    public static long getCPUTime() {
        ThreadMXBean bean = ManagementFactory.getThreadMXBean();
        return bean.isCurrentThreadCpuTimeSupported() ?
                bean.getCurrentThreadCpuTime() : 0L;
    }
}
```

Более точный способ отслеживания времени, затрачиваемого на выполнение кода, при условии, что код является однопоточным, - это смотреть на время процессора, потребляемое потоком во время вызова. Это можно сделать с помощью классов ‘JMX‘; в частности, с ‘ThreadMXBean‘. Нужно получить экземпляр ‘ThreadMXBean‘ из ‘java.lang.management.ManagementFactory‘, и если платформа поддерживает его (большинство из них), использование ‘getCurrentThreadCpuTime‘ вместо ‘System.currentTimeMillis‘ выполнить аналогичный тест. ‘getCurrentThreadCpuTime‘ сообщает время в наносекундах, а не миллисекундах.

3.2.2 Улучшение точности замеров времени

Чтобы получить более плавные результаты каждый тест запускается несколько раз.

4 Экспериментальная часть

4.1 Листинг кода

В листингах 3 - 5 представлена реализация стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

Листинг 3 – Стандартный алгоритм умножения матриц

```
public static void multiply_classic(int[][] a, int[][] b, int[][] res) {
        final int n = a.length;
        final int m = a[0].length;
        final int q = b[0].length;

        for (int i = 0; i < n; i++)
            for (int j = 0; j < q; j++)
                for (int k = 0; k < m; k++)
                    res[i][j] = res[i][j] + a[i][k] * b[k][j];
    }
```

Листинг 4 – Алгоритм Винограда

```
public static void multiply_winograd(int[][] a, int[][] b, int[][] res) {
        final int n = a.length;
        final int m = a[0].length;
        final int q = b[0].length;

        // row factor 'a'
        int[] row_factor = new int[n];
        int[] col_factor = new int[q];

        for (int i = 0; i < n; i++) {
            for  (int j = 0; j < m / 2; j++) {
                row_factor[i] = row_factor[i] + a[i][2 * j] * a[i][2 * j + 1];
            }
        }

        for (int i = 0; i < q; i++) {
            for (int j = 0; j < m / 2; j++) {
                col_factor[i] = col_factor[i] + b[2 * j][i] * b[2 * j + 1][i];
            }
        }


        // mult;
        for (int i = 0; i < n; i++) {
            for  (int j = 0; j < q; j++) {
                res[i][j] = -(row_factor[i] + col_factor[j]);
                for (int k = 0; k < m / 2; k++) {
                    res[i][j] = res[i][j] +
                            (a[i][2 * k] + b[2 * k + 1][j]) *
                            (a[i][2 * k + 1] + b[2 * k][j]);
                }
            }
        }

        // if m is odd
        if (m % 2 != 0) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < q; j++) {
                    res[i][j] = res[i][j] + a[i][m - 1] * b[m - 1][j];
                }
            }
        }
    }
```

Листинг 5 – Оптимизированный алгоритм Винограда

```
public static void multiply_winograd_plus(int[][] a, int[][] b, int[][] res) {
        final int n = a.length;
        final int m = a[0].length;
        final int q = b[0].length;

        final int d = m / 2;
        final boolean isOdd = m % 2 != 0;

        int[] row_factor = new int[n];
        int[] col_factor = new int[q];

        for (int i = 0; i < n; i++) {
            for  (int j = 0; j < d; j++) {
                row_factor[i] += a[i][2 * j] * a[i][2 * j + 1];
            }
        }

        for (int i = 0; i < q; i++) {
            for (int j = 0; j < d; j++) {
                col_factor[i] += b[2 * j][i] * b[2 * j + 1][i];
            }
        }

        int temp;
        // multiplying
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < q; j++) {
                temp = isOdd ? a[i][m - 1] * b[m - 1][j] : 0;
                temp -= row_factor[i] + col_factor[j];

                for (int k = 0; k < d; k += 2) {
                    temp += (a[i][k] + b[k + 1][j]) * (a[i][k + 1] + b[k][j]);
                }

                res[i][j] = temp;
            }
        }
    }
```

4.2 Примеры работы

На таблице 1 показаны примеры работы алгоритмов умножения матриц

Таблица 1 – Подготовленные тестовые данные

<table>
<tr><th>Матрицы</th><th>Стандартный алгоритм</th><th>Алгоритм Винограда</th><th>Алгоритм Винограда (с оптим.)</th></tr>
<tr><td>|2 2| |-3 -3|
|-1 -3| x|-3 -2|</td><td>|-12 -10|
| 12 9|</td><td>|-12 -10|
| 12 9|</td><td>|-12 -10|
| 12 9|</td></tr>
<tr><td>|-3 2 -1 2|
|-2 -3 -3 2|
|-2 1 -1 0|
x
|2 2 -1 1|
|1 -1 2 -1|
|-2 2 1 -3|
|2 -3 0 -1|</td><td>|2 -16 6 -4|
|3 -13 -7 8|
|-1 -7 3 0|</td><td>|2 -16 6 -4|
|3 -13 -7 8|
|-1 -7 3 0|</td><td>|2 -16 6 -4|
|3 -13 -7 8|
|-1 -7 3 0|</td></tr>
<tr><td>|-3 1 -3|
|-3 -3 -2|
x
|2 -3|
|0 -1|
1 2	</td><td>	-9 2
-8 8	</td><td>	-9 2
-8 8	</td><td>	-9 2
-8 8	</td></tr>
<tr><td>	-2 -2 -3 -2
-1 -2 0 -2
2 0 0 2
-3 0 0 -1	 x 	-1 -2 2 0 -1
-3 -1 2 -2 -3
1 1 -1 0 1
1 -2 -3 -2 0	</td><td>	3 7 1 8 5
5 8 0 8 7
0 -8 -2 -4 -2
2 8 -3 2 3	</td><td>	3 7 1 8 5
5 8 0 8 7
0 -8 -2 -4 -2
2 8 -3 2 3	</td><td>	3 7 1 8 5
5 8 0 8 7
0 -8 -2 -4 -2
2 8 -3 2 3	</td></tr>
</table>

4.3 Оценка трудоемкости

1. Стандартный алгоритм
$f = 2{+}M(2{+}2{+}Q(2{+}2{+}N(2{+}8{+}3))) = 13{\cdot}MNQ{+}4MQ{+}M{+}2 \approx 13{\cdot}MNQ$

2. Алгоритм Винограда
На формулах 6 – 14 расчитана трудоемкость алгиртма Винограда

$$f_{v1} = 2+M(2+2+3+\frac{N}{2}(3+6+1+2+3)) = \frac{15}{2}MN+7M+2 \approx \frac{15}{2}MN \quad (6)$$

$$f_{v1+} = 2+M(2+2+2+\frac{N}{2}(2+5+1+1+1)) = 10MN+6M+2 \approx \frac{10}{2}MN = 5MN \quad (7)$$

$$f_{v2} = 2+Q(2+2+3+N2(3+6+1+2+3)) = \frac{15}{2}QN+7Q+2 \approx \frac{15}{2}QN \quad (8)$$

$$f_{v2+} = 2+Q(2+2+\frac{Q}{2}(2+5+1+1+1)) = 10QN+6Q+2 \approx \frac{10}{2}QN = 5QN \quad (9)$$

$$f_{v3} = 2+M(2+2+Q(2+7+3+\frac{N}{2}(3+12+1+5+5)))+ = \frac{26}{2}MNQ+12MQ+4M+2 \approx \frac{26}{2}MNQ = 13MNQ \quad (10)$$

лучший случай 0, худший случай f_{v4}

$$f_{v4} = 3 + M(2+2+Q(2+13)) = 15QM + 4M + 3 \approx 15QM \quad (11)$$

В формуле (12) внесен внутри цикла f_{v4} из формулы (11). Поэтому появляется лвчший и худший случай.

$$f_{v3+} = 3+2+M(2+2+Q(2+6+\begin{bmatrix} \text{л.с. , 0} \\ \text{х.с. , 6 + 2 + 1 + 1} \end{bmatrix}.+2+\frac{N}{2}(2+10+1+2+2+1))) \quad (12)$$

в худшем случаем = 9MNQ+10MQ+4M+5
в лучшем случаем $\approx 8MNQ$

$$f_v = f_{v1} + f_{v2} + f_{v3} + f_4 = \frac{15}{2}MN + \frac{15}{2}QN + \frac{26}{2}MNQ + 15QM \quad (13)$$

$$f_{v+} = f_{v1+} + f_{v2+} + f_{v3+} + f_{4+} = 5MN + 5QN + \begin{bmatrix} \text{л.с. , 9MNQ + 10MQ} \\ \text{х.с. , 9MNQ + 20MQ} \end{bmatrix}. \quad (14)$$

Список оптимизации:

1) замена оперции = на += или -=
2) избавление от деления в условиях цикла
3) Заносим проверку на нечетность кол-ва строк внутрь основных циклов
4) Расчет условия для последнего цикла один раз, а далее использование флага

4.4 Сравнение времени работы

На графиках (рисунки 4.3, 4.4) представлено сравнение времени работы алгоритма на матрицах разных размеров. Для построения графика на рисунке 4.3 генерировались матрицы четной размерности: 50, 90, 130, 170, 210, 250, 290, 330, 370, 410. Для построения графика на рисунке 4.4 генерировались матрицы нечетной размерности: 51, 91, 131, 171, 211, 251, 291, 331, 371, 411.

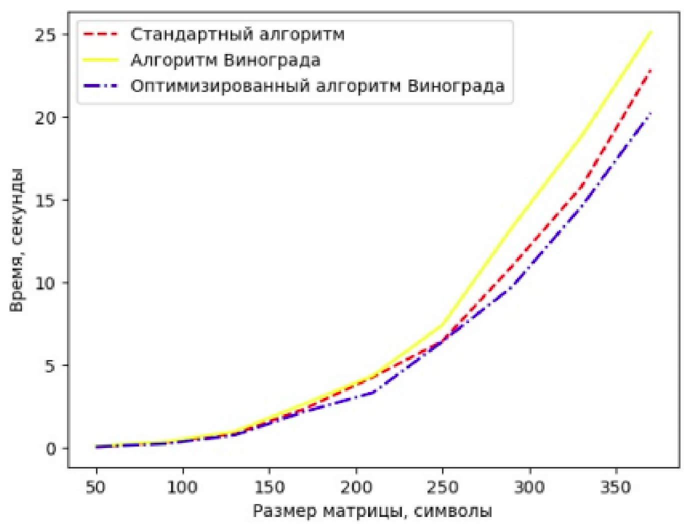

Рисунок 10 – Сравнение реализации алгоритмов нахождения произведения матриц при четных размерностях

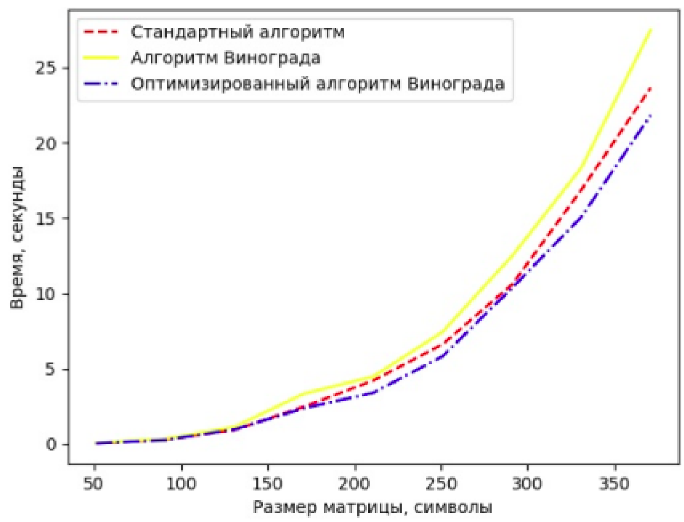

Рисунок 11 – Сравнение реализации алгоритмов нахождения произведения матриц при нечетных размерностях

4.5 Вывод

В результате проведенного эксперимента был получен следующий вывод: оптимизированный алгоритм Винограда работает быстрее классического метода и зачительно быстрее обычного алгоритма Винограда. По результатам тестирования на матрицах с нечетными размерностями видно, что неоптимизированный алгоритм Винограда работает медленнее, чем при работе с четными размерностями

Заключение

В данной лабораторной работе было реализовано и проанализировано три алгоритма умножения матриц:

- стандартный алгоритм умножения матриц;
- алгоритм Винограда;
- оптимизированный алгоритм Винограда.

В данной лабораторной работе была достигнута цель и были выполнены следующие задачи:

1) были описаны формулы расчета для двух алгоритмов: стандартного и алгоритма Винограда;
2) были реализованы стандартный алгоритм умножения матриц и алгоритм Винограда;
3) был реализован оптимизированый алгоритм Винограда;
4) посчитана теоретическая оценка трудоемкости трех алгоритмов;
5) проведены замеры процессорного времени работы реализацийч трех алгоритмов при четных и нечетных размерностях.

Список литературы

[1] Умножение матриц: [ЭЛ. РЕСУРС] Режим доступа: Режим доступа: http://www.algolib.narod.ru/Math/Matrix.html. (дата обращения: 03.10.2020).

[2] Oracle ThreadMXBean [ЭЛ. РЕСУРС] Режим доступа: https://docs.oracle.com/javase/7/docs/api/java/ (дата обращения: 03.10.2020).

[3] Oracle Java [ЭЛ. РЕСУРС] Режим доступа: https://www.oracle.com/java/. (дата обращения: 03.10.2020).