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 (ИУ7)**

ОТЧЕТ

по лабораторной работе № 2

Название: Алгоритмы умножения матриц

Дисциплина: Анализ Алгоритмов

Студент ИУ7-53Б _____ В.П. Федоров
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Ю.В. Строганов
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Введение

В работе требуется изучить алгоритмы умножения матриц, а также научиться рассчитывать трудоемкость алгоритмов и получить навыки оптимизации алгоритмов.

Алгоритмы умножения матриц для изучения:

1. Классический алгоритм;
2. Алгоритм Винограда;
3. Улучшенный алгоритм Винограда.

1. Аналитическая часть

Рассмотрим алгоритмы умножения матриц.

1.1 Описание алгоритмов

Умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, например, в алгоритмах машинного обучения.
Умножения матриц используется в множестве областей человеческой жизнедеятельности. Например, в физике и компьютерных играх, в экономике и бизнесе.
Давайте разберем каждый из алгоритмов более детально.

1.2 Классический алгоритм

Пусть даны две прямоугольные матрицы A и B размерности l x m и m x n соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}.$$

Тогда матрица C размерностью l x n:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix},$$

в которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i \in (1, 2, \ldots, l); j \in (1, 2, \ldots, n)) \qquad (1.2.1)$$

называется их произведением.

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 A B вовсе не следует существование произведения B A .

1.3 Алгоритм Винограда

Рассматривая результат умножения двух матриц очевидно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора V = (v1, v2, v3, v4) и W = (w1, w2, w3, w4). (1.3.1)

Их скалярное произведение равно:

V • W = v1w1 + v2w2 + v3w3 + v4w4. (1.3.2)

Это равенство можно переписать в виде:

V • W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) — v1v2 — v3v4 — w1w2 — w3w4. (1.3.3)

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем первое: вместо четырех умножений - шесть, а вместо трех сложений - десять, 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, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения[1].

1.4 Оптимизации алгоритма Винограда

В рамках данной работы было добавлено 3 оптимизации:

1. Избавление от деления в условии цикла;
2. Замена mulH[i] = mulH[i] + … на mulH[i] + = … (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.

Вывод

Основная разница между рассмотренными алгоритмами это наличие предварительной обработки, а также уменьшение количества операций умножения.

2. Конструкторская часть

В данном разделе будут рассмотрены требования к программному обеспечению и схемы алгоритмов.

2.1 Требования к программному обеспечению

Требования к программе:

- На вход подаются две матрицы
- На выход произведение двух матриц

Требования к вводу:

- Программа выведет сообщение об ошибке при некорректном размере матриц
- Программа выведет сообщение об ошибке при некорректном вводе

2.2 Схемы алгоритмов

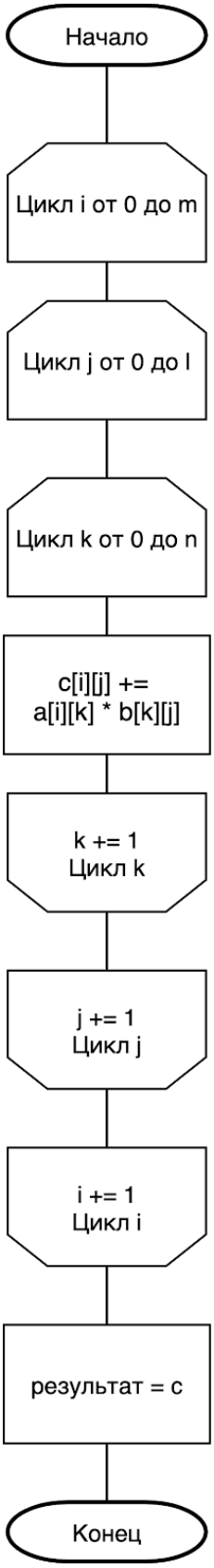


Рис. 2.1.1 Классический алгоритм умножения матриц

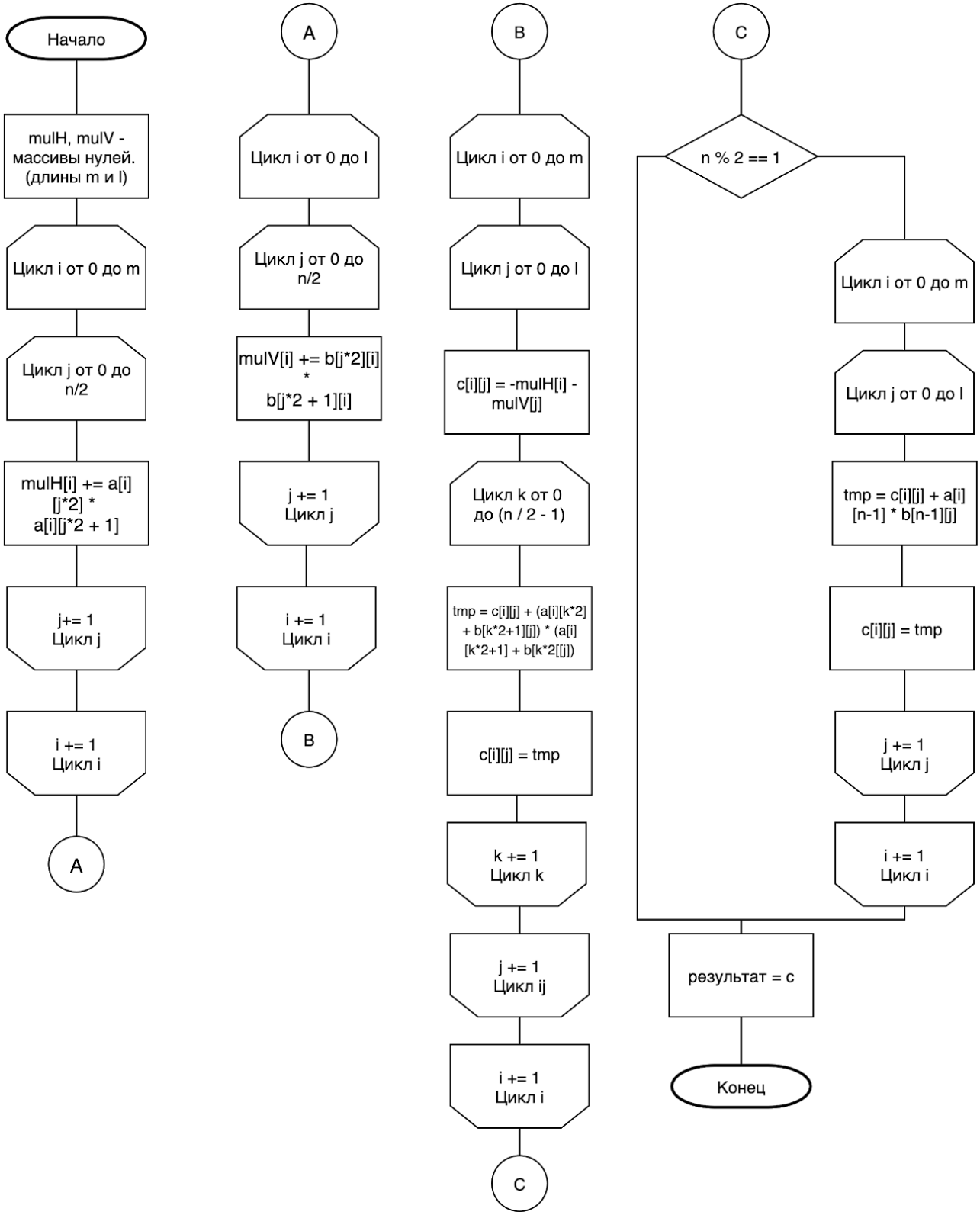


Рис. 2.1.2 Алгоритм Винограда

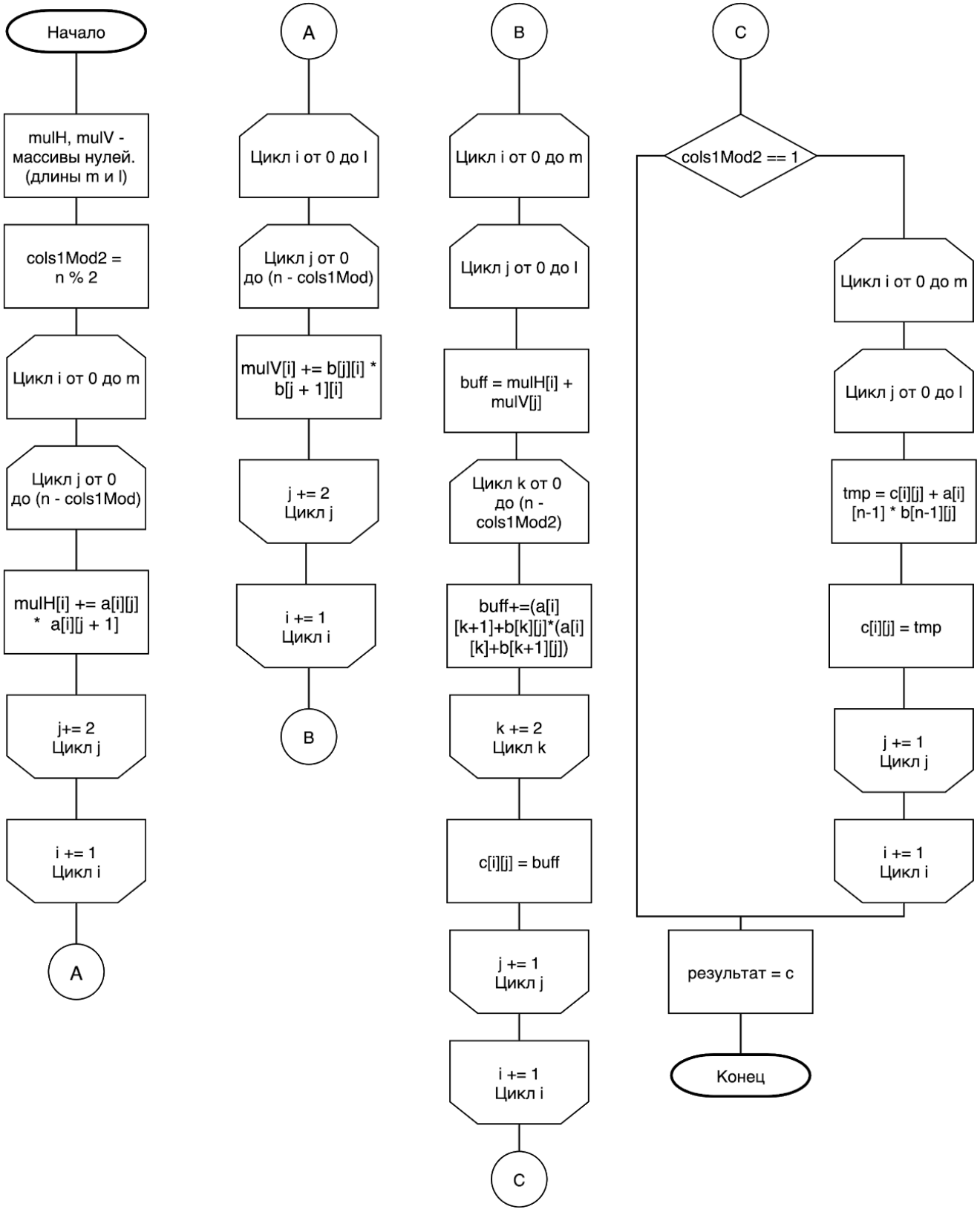


Рис. 2.1.3 Оптимизированный алгоритм Винограда

2.3 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1) Базовые операции стоимостью 1:
   +, -, *, /, =, ==, <=, >=, !=, +=, [];

2) оценка трудоемкости цикла
   for от 0 до N с шагом 1
   $F_{for} = a + N \cdot \left(a + F_{body} \right)$, *где* F_{body} *– тело цикла*, a *– условие*

3) стоимость условного перехода примем за 0, стоимость вычисления условия остается

Оценим трудоемкость алгоритмов по коду программы

2.3.1 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма:

Инициализация матрицы результата: $1 + 1 + n_1 \cdot (1 + 2 + 1) + 1 = 4 \cdot n_1 + 3$

Подсчет:

$$1 + n_1 \cdot \Big(1 + \Big(1 + m_2 \Big(1 + \Big(1 + m_1 (1 + (8) + 1) + 1 \Big) + 1 \Big) + 1 \Big) + 1 \Big) + 1 =$$

$$n_1 \Big(m_2 \big(10 m_1 + 4 \big) + 4 \Big) + 2 = 10 n_1 m_2 m_1 + 4 n_1 m_2 + 4 n_1 + 2$$

2.3.2 Алгоритм Винограда

Аналогично рассмотрим трудоемкость алгоритма Винограда

Первый цикл: $\frac{15}{2} n_1 m_1 + 5 n_1 + 2$

Второй цикл: $\frac{15}{2} m_2 n_2 + 5 m_2 + 2$

Третий цикл: $13 n_1 m_2 m_1 + 12 n_1 m_2 + 4 n_1 + 2$

Условный переход: $\begin{bmatrix} 2 & , \textit{невыпоплнение условия} \\ 15 n_1 m_2 + 4 n_1 + 2 & , \textit{выполнение условия} \end{bmatrix}$

Итого:

$$13 n_1 m_2 m_1 + \frac{15}{2} m_2 n_2 + 12 n_1 m_2 + 5 n_1 + 5 m_2 + 4 n_1 + 6 + \begin{bmatrix} 2 & , \textit{невыпоплнение условия} \\ 15 n_1 m_2 + 4 n_1 + 2 & , \textit{выполнение условия} \end{bmatrix}$$

2.3.3 Оптимизированный алгоритм Винограда

Первый цикл: $\frac{11}{2}n_1m_1 + 4n_1 + 2$

Второй цикл: $\frac{11}{2}m_2n_2 + 4m_2 + 2$

Третий цикл: $\frac{17}{2}n_1m_2m_1 + 9n_1m_2 + 4n_1 + 2$

Условный переход: $\begin{bmatrix} 1 & , \textit{невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & , \textit{выполнение условия} \end{bmatrix}$

Итого: $\frac{17}{2}n_1m_2m_1 + 9n_1m_2 + 4n_1 + 2 + \frac{11}{2}n_1m_1 + 4n_1 + 2 + \frac{11}{2}m_2n_2 + 4m_2 + 2 + \begin{bmatrix} 1 & , \textit{невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & , \textit{выполнение условия} \end{bmatrix}$

Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель оценки трудоемкости алгоритма, были рассчитаны трудоемкости алгоритмов в соотвествии с этой моделью.

3. Технологическая часть

3.1 Выбор средств разработки

Был использован язык программирования Java и среда разработки Intellij IDEA, так как ими я собираюсь пользоваться в дальнейшей работе.
С помощью Python и matplotlib были получены графические иллюстрации для экспериментального раздела.

3.2 Описание структуры ПО

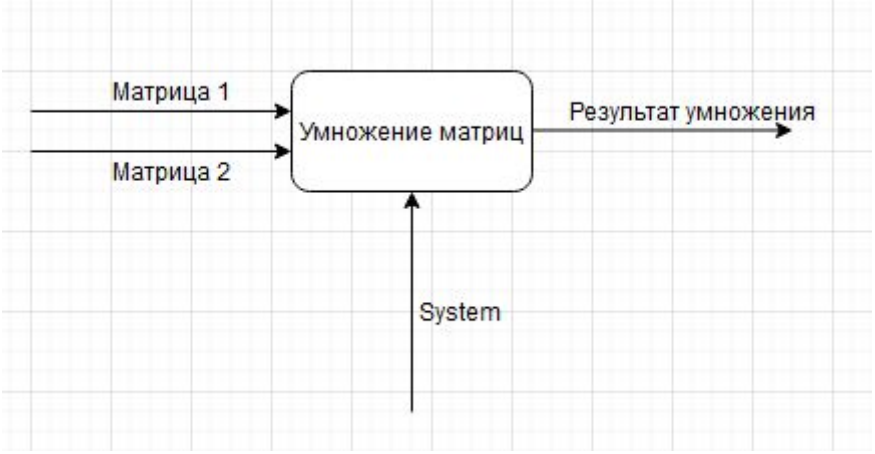


Рис.3.2.1: Функциональная схема умножения матриц

3.3 Сведения о модулях программы

Программа состоит из:

- Main.java - главный файл программы, в котором располагается точка входа в программу и функция замера времени.
- Matrix.java - файл, содержащий вспомогательные функции для работы с матрицами(ввод/вывод матриц, сравнение матриц и т.д.)
- ClassicalMultiplication.java - - файл, содержащий в себе реализацию классического алгоритма умножения матриц
- Vinograd.java - файл, содержащий в себе реализацию алгоритма Винограда
- VingradOptimized.java - файл, содержащий в себе реализацию оптимизированного алгоритма Винограда
- ClassicalMultiplicationTest.java - файл с юнит-тестами для класса ClassicalMultiplication
- VinogradTest.java - файл с юнит-тестами для класса Vinograd
- VingradOptimizedTest.java файл с юнит-тестами для класса VingradOptimized

3.4 Листинги кода

Листинг 3.4.1 Классический алгоритм умножения матриц

```
public class ClassicalMultiplication {
  public static int[][] multiplicateMatrix(int[][] firstMatrix, int[][] secondMatrix) throws
Exception {
      int n1 = firstMatrix.length;
      int n2 = secondMatrix.length;

      if (n1 * n2 == 0) return null;

      int m1 = firstMatrix[0].length;
      int m2 = secondMatrix[0].length;

      if (m1 != n2) return null;

      int[][] res = new int[n1][];
      for (int i = 0; i < n1; i++)
         res[i] =  new int[m2];

      for (int i = 0; i < n1; i++)
         for (int j = 0; j < m2; j++)
            for (int k = 0; k < m1; k++)
               res[i][j] += firstMatrix[i][k] * secondMatrix[k][j];

      return res;

   }

}
```

Листинг 3.4.2 Алгоритм Виноград

```
public class Vinograd {
  public static int[][] vinogradMultiplication(int[][] firstMatrix, int[][] secondMatrix) throws
Exception {
      int n1 = firstMatrix.length;
      int n2 = secondMatrix.length;
```

```
if (n1 * n2 == 0) return null;

int m1 = firstMatrix[0].length;
int m2 = secondMatrix[0].length;

if (m1 != n2) return null;

int[] mulH = new int[n1];
int[] mulV = new int[m2];

int[][] res = new int[n1][];
for (int i = 0; i < n1; i++)
   res[i] = new int[m2];

for (int i = 0; i < n1; i++) {
   for (int j = 0; j < m1 / 2; j++) {
      mulH[i] = mulH[i] + firstMatrix[i][j*2] * firstMatrix[i][j*2 + 1];
   }
}

for (int i = 0; i < m2; i++) {
   for (int j = 0; j < n2 / 2; j++) {
      mulV[i] = mulV[i] + secondMatrix[j*2][i] * secondMatrix[j*2 + 1][i];
   }
}

for (int i = 0; i < n1; i++) {
   for (int j = 0; j < m2; j++) {
      res[i][j] = -mulH[i] - mulV[j];
      for (int k = 0; k < m1 / 2; k++) {
         res[i][j] = res[i][j] + (firstMatrix[i][2 * k + 1] + secondMatrix[2 * k][j]) *
               (firstMatrix[i][2 * k] + secondMatrix[2*k + 1][j]);
      }
   }
}

if (m1 % 2 == 1) {
   for (int i = 0; i < n1; i++) {
      for (int j = 0; j < m2; j++) {
         res[i][j] = res[i][j] + firstMatrix[i][m1 - 1] * secondMatrix[m1-1][j];
      }
   }
}
```

```
        return res;
    }
}
```

Листинг 3.4.3 Оптимизированный алгоритм Виноград

```
public class VingradOptimized {
    public static int[][] vinogradMultiplication(int[][] firstMatrix, int[][] secondMatrix){
        int n1 = firstMatrix.length;
        int n2 = secondMatrix.length;

        if (n1 * n2 == 0) return null;

        int m1 = firstMatrix[0].length;
        int m2 = secondMatrix[0].length;

        if (m1 != n2) return null;

        int[] mulH = new int[n1];
        int[] mulV = new int[m2];

        int[][] res = new int[n1][];
        for (int i = 0; i < n1; i++)
            res[i] = new int[m2];

        int m1Mod2 = m1 % 2;
        int n2Mod2 = n2 % 2;

        for (int i = 0; i < n1; i++) {
            for (int j = 0; j < (m1 - m1Mod2); j += 2) {
                mulH[i] += firstMatrix[i][j] * firstMatrix[i][j+1];
            }
        }

        for (int i = 0; i < m2; i++) {
            for (int j = 0; j < (n2 - n2Mod2); j += 2) {
                mulV[i] += secondMatrix[j][i] * secondMatrix[j+1][i];
            }
```

```
        }

        for (int i = 0; i < n1; i++) {
            for (int j = 0; j < m2; j++) {
                int buff = -(mulH[i] + mulV[j]);
                for (int k = 0; k < (m1 - m1Mod2); k+= 2) {
                    buff += (firstMatrix[i][k+1] + secondMatrix[k][j]) * (firstMatrix[i][k] +
secondMatrix[k+1][j]);
                }
                res[i][j] = buff;
            }
        }

        if (m1Mod2 == 1) {
            int m1Min_1 = m1 - 1;
            for (int i = 0; i < n1; i++) {
                for (int j = 0; j < m2; j++) {
                    res[i][j] += firstMatrix[i][m1Min_1] * secondMatrix[m1Min_1][j];
                }
            }
        }

        return  res;
    }
}
```

3.5 Тесты

Тестирование проводилось с помощью модуля JUnit. Для каждого алгоритма был написан класс. Сначала каждый алгоритм тестировался отдельно на заранее заготовленных тестах (см. таблицу 3.5.1). Далее генерировались пары случайных матриц случайных размеров, производилось умножение матриц с помощью классического алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда. Сравнивались результаты.

Все написанные тесты были пройдены.
Таблица 3.5.1 Набор тестовых данных

Матрица A	Матрица B	Результат
[0]	[0]	[0]

$\begin{bmatrix} 0 \end{bmatrix}$	$\begin{bmatrix} 1 \end{bmatrix}$	$\begin{bmatrix} 0 \end{bmatrix}$
$\begin{bmatrix} 1 \end{bmatrix}$	$\begin{bmatrix} 1 \end{bmatrix}$	$\begin{bmatrix} 1 \end{bmatrix}$
$\begin{bmatrix} 2 \end{bmatrix}$	$\begin{bmatrix} 2 \end{bmatrix}$	$\begin{bmatrix} 4 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$
$\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$	$\begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}$	$\begin{bmatrix} 8 & 8 \\ 8 & 8 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 3 & 3 \\ 7 & 7 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 4 & 6 \\ 4 & 6 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$
$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 12 & 9 & 6 \\ 30 & 23 & 16 \\ 48 & 37 & 26 \end{bmatrix}$
$\begin{bmatrix} 1 & 4 & 8 \\ 16 & 32 & 64 \end{bmatrix}$	$\begin{bmatrix} 1 & 3 \\ 9 & 27 \\ 81 & 243 \end{bmatrix}$	$\begin{bmatrix} 685 & 2055 \\ 5488 & 16464 \end{bmatrix}$

$\begin{bmatrix} 1 & 4 & 1 \\ 5 & 9 & 2 \\ 6 & 5 & 3 \\ 5 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 7 & 9 & 3 & 2 \\ 3 & 8 & 4 & 6 \\ 2 & 6 & 4 & 3 \end{bmatrix}$	$\begin{bmatrix} 21 & 47 & 23 & 29 \\ 66 & 129 & 59 & 70 \\ 63 & 112 & 50 & 51 \\ 77 & 163 & 83 & 85 \end{bmatrix}$

3.6 Вывод

В данном разделе были рассмотрены структура программы, листинги кода, тесты, а также сведения о модулях.

4. Исследовательская часть

4.1 Описание характеристик аппаратуры

Процессор: Intel(R) Core(™) i3-6100U CPU @2.30Ghz 2.30GHZ
Установленная память(ОЗУ): 16,0 ГБ(15.8 ГБ Доступно)
Тип системы: 64-разрядная операционная система, процессор x64
Операционная система: Windows 10

4.2 Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

1. Замеры времени работы классического алгоритма умножения матриц дляквадратных матриц размерностей от 100 x 100 до 1000 x 1000 с шагом 100.
2. Замеры времени работы алгоритма Винограда для квадратных матриц размерности от 100 x 100 до 1000 x 1000 с шагом 100.
3. Замеры времени работы оптимизированного алгоритма Винограда для квадратных матриц размерностей от 100 x 100 до 1000 x 1000 с шагом 100.
4. Замеры времени работы классического алгорима умножения матриц дляквадратных матриц размерностей от 101 x 101 до 1001 x 1001 с шагом 100
5. Замеры времени работы алгоритма Винограда для квадратных матриц размерности от 101 x 101 до 1001 x 1001 с шагом 100
6. Замеры времени работы оптимизированного алгоритма Винограда для квадратных матриц размерностей от 101 x 101 до 1001 x 1001 с шагом 100

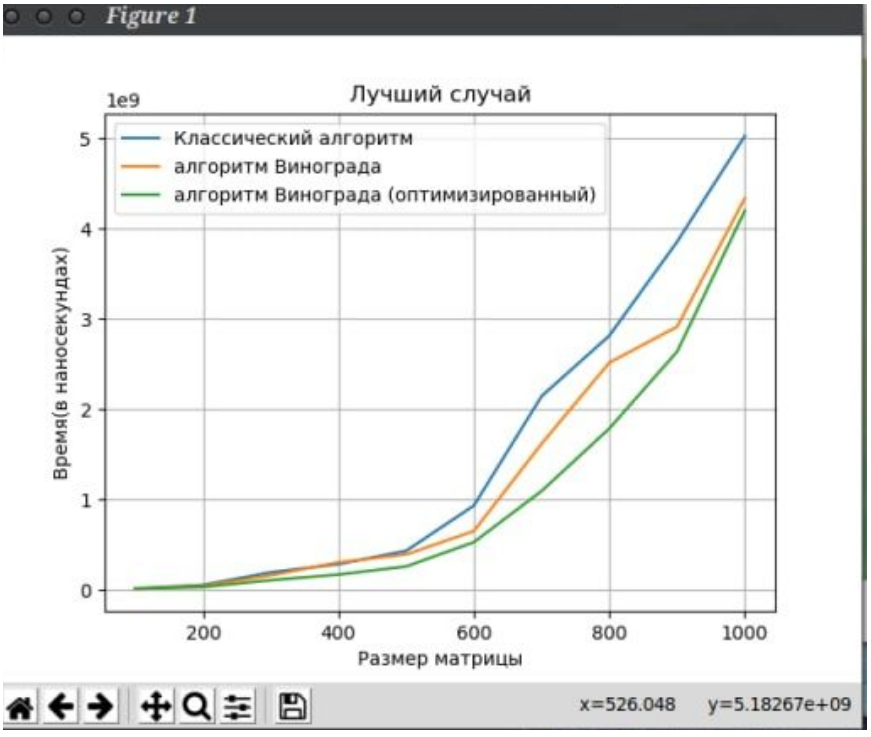
В рамках каждого эксперимента для матриц каждой размерности проводилось по 10 замеров, и вычислялось среднее время работы конкретного алгоритма на матрицах данной размерности.

4.3 Сравнительный анализ на материале экспериментальных данных

Был проведен замер времени работы каждого из алгоритмов.

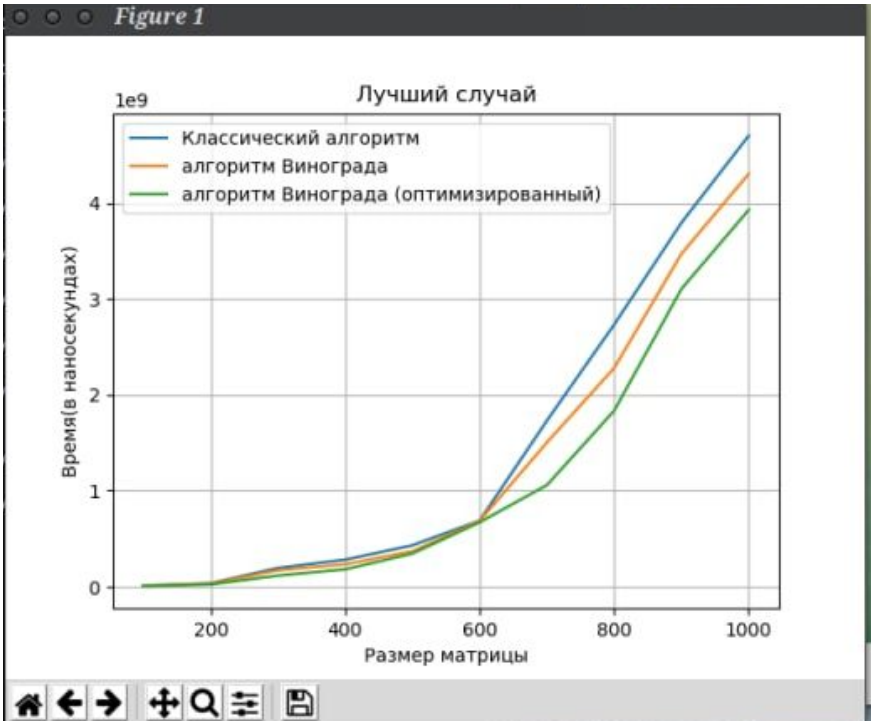
Первый эксперимент проводился для лучшего случая на квадратных матрицах размером от 100x100 до 1000x1000. Сравним результаты для разных алгоритмов.

Рис. 4.3.1 Сравнение времени для лучшего случая (четный размер матрицы)



Второй эксперимент производится для худшего случая когда поданы квадратные матрицы с нечетным размерами от 101x101 до 1001x1001 с шагом 100. Сравним результаты для разных алгоритмов:

Рис. 4.3.2 Сравнение времени для худшего случая (нечетный размер матрицы)



4.4 Вывод

По результатам эксперимента самым быстрым оказался оптимизированный алгоритм Винограда, а самым медленным классический алгоритм.

Заключение

В ходе данной лабораторной работы были изучены и реализованы классический алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

Был изучен подход к вычислению трудоемкости алгоритмов.

Была дана теоретическая оценка классического алгоритма умножения матриц, алгоритма Винограда.

Было произведено сравнение классического алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда по времени их работы.

Литература

1. Матрицы и определители, учебное пособие по линейной алгебре[Текст] И. В. Белоусов, Издательство: Дрофа, Москва, 1999, с. 1 - 16
2. Умножение матриц: эффективная реализация шаг за шагом [Электронный ресурс], режим доступа: https://habr.com/ru/post/359272/