МГТУ им. Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Алгоритмы умножения матриц

Работу выполнила: Овчинникова Анастасия, ИУ7-55Б

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

Москва, 2019

Оглавление

Введение **2**

Аналитическая часть **3**
Классический алгорим умножения матриц 3
Алгоритм Винограда 4
Улучшенный алгоритм Винограда 4
Модель вычислений 4
Вывод 5

Конструкторская часть **6**
Требования к программе 6
Схемы алгоритмов 6

Технологическая часть **10**
Выбор языка программирования 10
Сведения о модулях программы 10
Листинги кода алгоритмов 10
Расчет трудоемкости 13
Классический алгорим умножения матриц 13
Алгоритм Винограда 13
Оптимизированный алгоритм Винограда 14
Тесты 14

Исследовательская часть **16**
Постановка эксперимента 16
Сравнительный анализ на материале экспериментальных данных 16
Выводы 18

Заключение **20**

Введение

Целью данной работы является изучение алгоритмов умножения матриц. Задачи лабораторной работы:

1. реализовать стандартный алгоритм умножения матриц;
2. реализовать алгоритм умножения матриц Винограда;
3. реализовать оптимизированный алгоритм умножения матриц Винограда;
4. дать теоретическую оценку стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда;
5. сравнить время работы перечисленных алгоритмов умножения матриц.

Аналитическая часть

Матрицей называют математический объект, эквивалентный двумерному массиву. Матрица представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Будем говорить исключительно о матрицах прямоугольной формы (в частных случаях - квадратной). Для матриц определена операция умножения. Матрица, получаемая в результате операции умножения, называется произведнием матриц.

Классический алгорим умножения матриц

Пусть даны две прямоугольные матрицы A и B размерности m на n и n на l соответсвенно:

$$\begin{bmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{m1} & ... & a_{mn} \end{bmatrix}$$

$$\begin{bmatrix} b_{11} & ... & b_{1l} \\ ... & ... & ... \\ b_{n1} & ... & b_{nl} \end{bmatrix}$$

Тогда матрица C размерностью m на l:

$$\begin{bmatrix} c_{11} & ... & c_{1l} \\ ... & ... & ... \\ c_{m1} & ... & c_{ml} \end{bmatrix}$$

в которой:
$c_{ij} = \sum_{r=1}^{n} a_{ir} \cdot b_{rj} (i = 0, 1, ..., m-1; j = 0, 1, ..., l-1)$
называется произведением матриц A и B.

Алгоритм Винограда

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.

Алгоритм Винограда считается более эффективным благодаря сокращению количества операций умножения.

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение равно:

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

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.

Улучшенный алгоритм Винограда

Улучшить алгоритм Винограда можно следующим образом.

1. Добавление дополнительного буфера для уменшения количества обращений к результирующей матрице.
2. Уменьшение количества операций умножения при заполнении дополнительных массивов, необходимых для работы алгоритма: mulH и mulV (подробнее см. далее).
3. Замена операций типа $x = x + y$ на операции $x+ = y$.
4. Избавиться от делений в заголовках циклов.

Модель вычислений

Трудоемкость алгоритма измеряется в количестве операций, которые необходимо выполнить.

Введем модель вычислений, используемую при оценке трудоемкости алгоритмов.

1. Базовые операции трудоемкости 1: +, -, *, /, =, ==, <=, >=, !=, >, <, +=, -=, |=, *=, проход по адресу.

2. Трудоемкость цикла вычисляется по формуле $f = 2 + N(2 + f_{body})$, где N - число повтрорений цикла, f_{body} - трудоемкость тела цикла.

3. Трудоемкость условного перехода равна 0, стоимость вычисления условия остается.

4. Вызов метода объекта класса имеет трудоемкость 1.

5. Объявление переменной/массива/структуры без определения имеет трудоемкость 0.

6. Условный оператор без условий внутри имеет трудоемкость 0.

7. Логические операции имеют трудоемкость 1.

Вывод

В данном разделе были рассмотрены идеи классического алгоритма умножения матриц, алгорима Винограда и улучшенного алгоритма Винограда. Более подробно эти алгоритмы (в т.ч. их схемы) будут рассмотрены в следующих разделах.

Конструкторская часть

Требования к программе

Требования к вводу:
На вход программе подаются две матрицы и их размерности.

Требования к программе:
Корректное умножение двух матриц.

Схемы алгоритмов

На рисунке 1 представлена схема классического алгорима умножения матриц. На рисунке 2 представлена схема алгорима Винограда. На рисунке 3 представлена схема оптимизированного алгорима Винограда. В схемах используются обозначения, которые были введены при рассмотрении алгоритмов умножения матриц в предыдущем разделе.

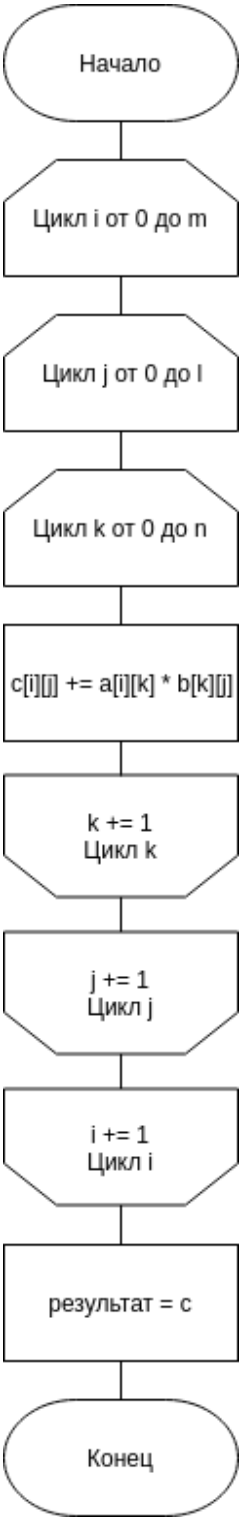

Рис. 1: Схема классического алгорима умножения матриц

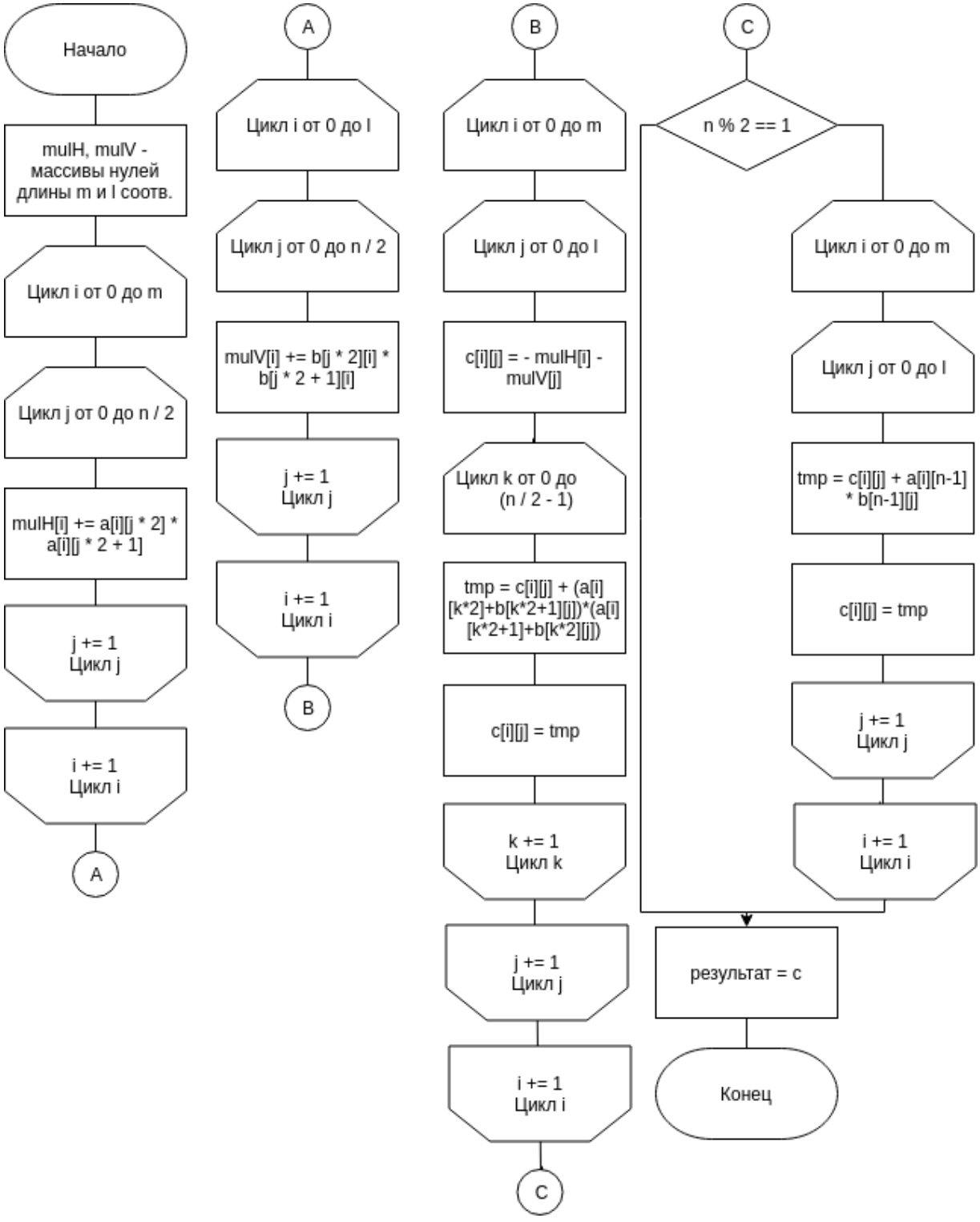

Рис. 2: Схема алгоритма Винограда

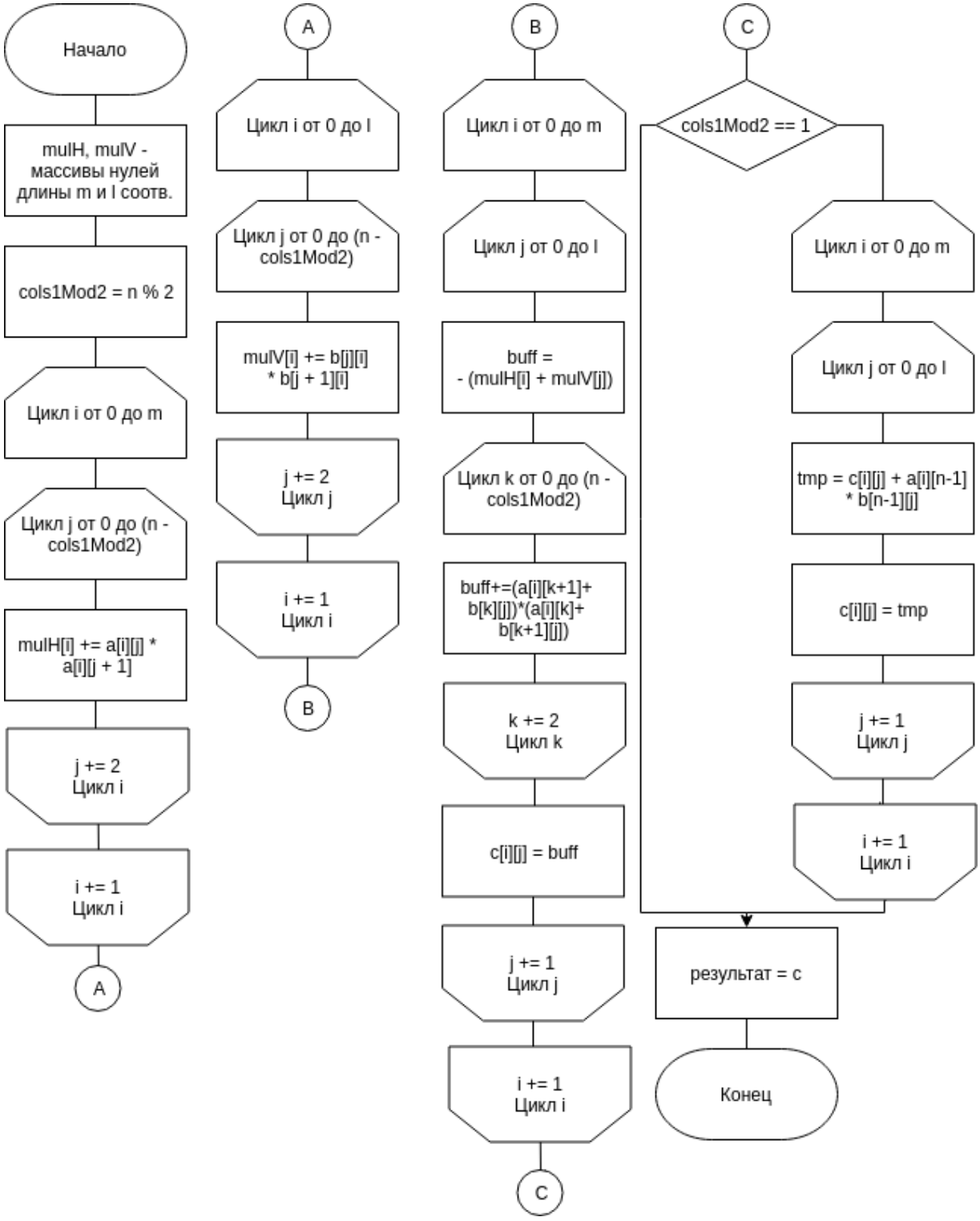

Рис. 3: Схема оптимизированного алгорима Винограда

Технологическая часть

Выбор языка программирования

В качестве языка программирования для реализации программы был выбран язык C++ и фреймворк Qt, потому что:

- язык C++ имеет высокую вычислительную производительность;
- язык C++ поддерживает различные стили программирования;
- в Qt существует удобный инструмент для тестирования - QtTest - который позволяет собирать тесты в группы, собирать результаты выполнения тестов, а также уменьшить дублирование кода при схожих объектах тестирования.

Для замеров времени использовалась функция clock() модуля ctime. Эта функция возвращает количество временных тактов, прошедших с начала запуска программы. С помощью макроса CLOCKS_PER_SEC можно узнать количество пройденных тактов за 1 секунду.

Сведения о модулях программы

Программа состоит из следующих файлов:

- mymatrix.h, mymatrix.cpp - заголовочный файл и файл, в котором расположена реализация алгоритмов сортировки;
- main.cpp - главный файл программы, в котором расположена реализация меню;
- testmymatrix.h, testmymatrix.cpp - файл и заголовочный файл, в котором расположена реализация тестов.

Листинги кода алгоритмов

Код классического алгоритма умножения матриц представлен в листинге 1. Код алгоритма Винограда представлен в листинге 2. Код оптимизированного алгоритма Винограда представлен в листинге 3.

Листинг 1: Классический алгоритм умножения матриц

```
MyMatrix MyMatrix::multiply(const MyMatrix &m)
{
    if (mColumns != m.mRows)
        throw std::logic_error("Attempt to multiply matrices of
           different dimensions.");

    MyMatrix result(mRows, m.mColumns, 0);

    for (int i = 0; i < mRows; ++i)
        for (int j = 0; j < m.mColumns; ++j)
            for (int k = 0; k < mColumns; ++k)
            {
                int resNum = result.at(i, j) + this->at(i, k) * m.at(k
                   , j);
                result.set(i, j, resNum);
            }
    return result;
}
```

Листинг 2: Алгоритм Винограда

```
MyMatrix MyMatrix::multiplyVinograd(const MyMatrix &m)
{
    if (mColumns != m.mRows)
        throw std::logic_error("Attempt to multiply matrices of
           different dimensions.");

    MyMatrix result(mRows, m.mColumns, 0);

    int* mulH = new int[result.mRows];
    int* mulV = new int[result.mColumns];

    for (int i = 0; i < result.mRows; ++i)
        mulH[i] = 0;
    for (int i = 0; i < result.mColumns; ++i)
        mulV[i] = 0;

    for (int i = 0; i < result.mRows; ++i)
        for (int j = 0; j < m.mRows / 2; ++j)
            mulH[i] += this->at(i, j * 2) * this->at(i, j * 2 + 1);

    for (int i = 0; i < result.mColumns; ++i)
        for (int j = 0; j < m.mRows / 2; ++j)
            mulV[i] += m.at(2 * j, i) * m.at(j * 2 + 1, i);

    for (int i = 0; i < result.mRows; ++i)
        for (int j = 0; j < result.mColumns; ++j)
        {
            result.set(i, j, - mulH[i] - mulV[j]);
            for (int k = 0; k < m.mRows / 2; ++k)
            {
                int newRes = result.at(i, j) +
                             (this->at(i, 2 * k) + m.at(2 * k + 1, j))
```

```
                                    *
32                                 (this->at(i, 2 * k + 1) + m.at(2 * k, j))
                                    ;
33                  result.set(i, j, newRes);
34              }
35          }
36
37      if (mColumns % 2)
38      {
39          for (int i = 0; i < mRows; ++i)
40              for (int j = 0; j < m.mColumns; ++j)
41              {
42                  int newRes = result.at(i, j) +
43                               this->at(i, mColumns - 1) *
44                               m.at(mColumns - 1, j);
45                  result.set(i, j, newRes);
46              }
47      }
48      delete [] mulH;
49      delete [] mulV;
50      return result;
51  }
```

Листинг 3: Оптимизированный алгорим Винограда

```
1   MyMatrix MyMatrix::multiplyVinogradOptimized(const MyMatrix &m)
2   {
3       if (mColumns != m.mRows)
4           throw std::logic_error("Attempt to multiply matrices of
                different dimensions.");
5
6       MyMatrix result(mRows, m.mColumns, 0);
7
8       int* mulH = new int[result.mRows];
9       int* mulV = new int[result.mColumns];
10
11      for (int i = 0; i < result.mRows; ++i)
12          mulH[i] = 0;
13      for (int i = 0; i < result.mColumns; ++i)
14          mulV[i] = 0;
15
16      int cols1Mod2 = mColumns % 2;
17
18      for (int i = 0; i < mRows; ++i)
19          for (int j = 0; j < (mColumns - cols1Mod2); j += 2)
20              mulH[i] += this->at(i, j) * this->at(i, j + 1);
21
22      for (int i = 0; i < m.mColumns; ++i)
23          for (int j = 0; j < (m.mRows - cols1Mod2); j += 2)
24              mulV[i] += m.at(j, i) * m.at(j + 1, i);
25
26      for (int i = 0; i < mRows; ++i)
27          for (int j = 0; j < m.mColumns; ++j)
28          {
```

```
29              int buff = -(mulH[i] + mulV[j]);
30              for (int k = 0; k < (mColumns - cols1Mod2); k += 2)
31                  buff += (this->at(i, k + 1) + m.at(k, j)) *
32                          (this->at(i, k) + m.at(k + 1, j));
33              result.set(i, j, buff);
34          }
35
36      if (cols1Mod2)
37      {
38          for (int i = 0; i < result.mRows; ++i)
39              for (int j = 0; j < result.mColumns; ++j)
40              {
41                  int newRes = result.at(i, j) +
42                               this->at(i, mColumns - 1) *
43                               m.at(mColumns - 1, j);
44                  result.set(i, j, newRes);
45              }
46      }
47      delete [] mulH;
48      delete [] mulV;
49      return result;
50  }
```

Расчет трудоемкости

Классический алгорим умножения матриц

Рассмотрим трудоемкость классического алгоритма умножения матриц.

$f = 2+m(2+2+l(2+2+n(2+7))) = 2+m(4+l(4+9n)) = 2+m(4+4l+9ln) = 2 + 4m + 4lm + 9lmn$

Алгоритм Винограда

Рассмотрим трудоемкость алгоритма Винограда.
1 цикл:
$f_1 = 2 + m(4 + 10\frac{n}{2})) = 2 + m(4 + 5n) = 2 + 4m + 5nm$
2 цикл:
$f_2 = 2 + l(4 + 10\frac{n}{2}) = 2 + l(4 + 5n) = 2 + 4l + 5ln$
3 цикл:
$f_3 = 2 + m(4 + l(2 + 5 + 2 + \frac{n}{2}(2 + 15))) = 2 + m(4 + l(9 + 17\frac{n}{2})) = 2 + m(4 + 9l + \frac{17}{2}ln) = 2 + 4m + 9lm + \frac{17}{2}lmn$
Условный переход:
$f_c = \begin{bmatrix} 2, \text{если n mod 2} == 0 \\ 2 + 4m + 10lm, \text{иначе} \end{bmatrix}$
Итого:

$$f = f_1+f_2+f_3+f_c = 6+8m+5nm+4l+5ln+9lm+\frac{17}{2}lmn+\begin{bmatrix}2, \text{если n mod } 2 == 0\\ 2+4m+10lm, \text{иначе}\end{bmatrix}$$

Оптимизированный алгоритм Винограда

Рассмотрим трудоемкость оптимизированного алгоритма Винограда.

1 цикл:
$f_1 = 2 + m(5 + 4n) = 2 + 5m + 4mn$
2 цикл:
$f_2 = 2 + 5l + 4ln$
3 цикл:
$f_3 = 2 + 4m + 7lm + \frac{13}{2}lmn$
Условный переход:
$f_c = \begin{bmatrix}1, \text{если n mod } 2 == 0\\ 2+4m+9lm, \text{иначе}\end{bmatrix}$
Итого:
$f = f_1 + f_2 + f_3 + f_c = 7 + 8m + 4mn + 4l + 4ln + 9lm + \frac{13}{2}lmn$

Тесты

Тестирование проводилось с помощью модуля QtTest. Для этого был написан класс TestMyMatrix. Сначала каждый алгоритм умножения матриц тестировался по отдельности на заранее заготовленном наборе тестовых данных. После этого генерировались пары случайных квадратных матриц целых чисел в диапазоне от -50 до 50 размером от 0 до 200 с шагом 5 (для каждой размерности матрицы генерировалось 10 случайных матриц). Проводилоь умножение этих пар матриц с помощью классического алгоритма умножения матриц, алгорима Винограда и оптимизированного алгорима Винограда и сравнивались результаты работы этих трех алгоритмов. Затем генерировались пары случайных прямоугольных матриц целых чисел в диапазоне от -50 до 50 размером от 0 до 200 с шагом 5 (для каждой размерности матрицы генерировалось 10 случайных матриц). Проводилась умножение этих пар матриц с помощью классического алгоритма умножения матриц, алгорима Винограда и оптимизированного алгорима Винограда и сравнивались результаты работы этих трех алгоритмов.

Все написанные тесты были пройдены.

Использованный набор тестовых данных приведен в таблице 1.

Таблица 1: Набор тестовых данных

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$
$\begin{bmatrix}0\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}0\end{bmatrix}$
$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$	$\begin{bmatrix}1\end{bmatrix}$
$\begin{bmatrix}2\end{bmatrix}$	$\begin{bmatrix}2\end{bmatrix}$	$\begin{bmatrix}4\end{bmatrix}$
$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$
$\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$	$\begin{bmatrix}2 & 2\\2 & 2\end{bmatrix}$	$\begin{bmatrix}8 & 8\\8 & 8\end{bmatrix}$
$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$
$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}$
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}3 & 3\\7 & 7\end{bmatrix}$
$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}4 & 6\\4 & 6\end{bmatrix}$
$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$	$\begin{bmatrix}7 & 10\\15 & 22\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$
$\begin{bmatrix}1 & 1\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}$
$\begin{bmatrix}1 & 2\\3 & 4\\5 & 6\end{bmatrix}$	$\begin{bmatrix}6 & 5 & 4\\3 & 2 & 1\end{bmatrix}$	$\begin{bmatrix}12 & 9 & 6\\30 & 23 & 16\\48 & 37 & 26\end{bmatrix}$
$\begin{bmatrix}1 & 4 & 8\\16 & 32 & 64\end{bmatrix}$	$\begin{bmatrix}1 & 3\\9 & 27\\81 & 243\end{bmatrix}$	$\begin{bmatrix}685 & 2055\\5488 & 16464\end{bmatrix}$
$\begin{bmatrix}1 & 4 & 1\\5 & 9 & 2\\6 & 5 & 3\\5 & 8 & 9\end{bmatrix}$	$\begin{bmatrix}7 & 9 & 3 & 2\\3 & 8 & 4 & 6\\2 & 6 & 4 & 3\end{bmatrix}$	$\begin{bmatrix}21 & 47 & 23 & 29\\66 & 129 & 59 & 70\\63 & 112 & 50 & 51\\77 & 163 & 83 & 85\end{bmatrix}$

Исследовательская часть

Постановка эксперимента

В рамках данной работы были проведены следующие эксперименты.

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

Измерения проводились на компьютере HP Pavilion Notebook на базе Intel Core i5-7200U, 2.50 Гц с 6 Гб оперативной памяти под управлением операционной системы Linux Mint.

Сравнительный анализ на материале экспериментальных данных

В таблице 2 представлены результаты замеров времени работы классического алгоритма умножения матрц (Классич.), алгоритма Винограда (Виноград) и оптимизированного алгорима Винограда (Опт. Виноград) на квадратных матрицах четного размера.

Таблица 2: Результаты замеров времени для матриц четного размера

Размерность	Классич., с	Виноград, с	Опт. Виноград, с
100	0.0249866	0.0170725	0.0117449
200	0.1803776	0.13419775	0.09425999
300	0.6103413	0.464396975	0.325237799
400	1.452259	1.1203795975	0.7838837799
500	2.8890183	2.2427023597	1.580883278
600	5.1179977	3.996203236	2.8101774278
700	8.331629	6.5615820236	4.6541242428
800	12.8242005	10.1323214024	7.0837630243
900	18.5736184	15.2110748402	10.6929463024
1000	27.0825138	21.602753084	15.6942994302

Данные из таблицы 2 изображены на рисунке 4. Условные обозначения:

- classicEven - время работы классического алгоритма умножения матриц на квадратных матрицах четного размера в секундах;
- vinogradEven - время работы алгорима Винограда на квадратных матрицах четного размера в секундах;
- optEven - время работы оптимизированного алгорима Винограда на квадратных матрицах четного размера в секундах.

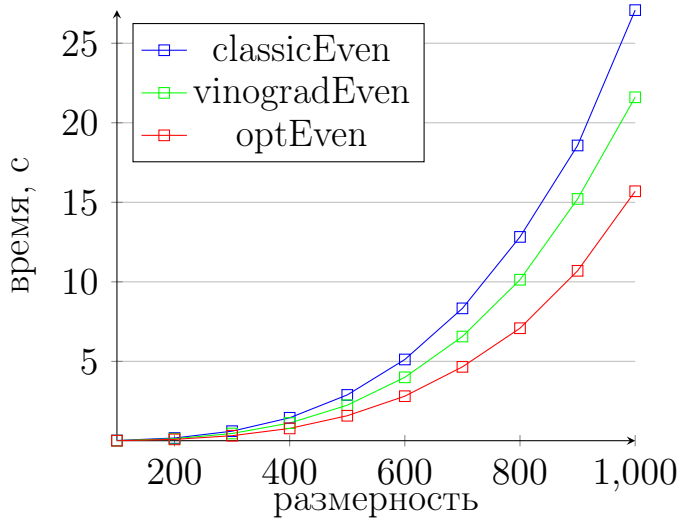

Рис. 4: Результаты замеров времени для матриц четного размера

В таблице 3 представлены результаты замеров времени работы классического алгоритма умножения матрц (Классич.), алгоритма Винограда (Виноград) и оптимизированного алгорима Винограда (Опт. Виноград) на квадратных матрицах нечетного размера.

Таблица 3: Результаты замеров времени для матриц нечетного размера

Размерность	Классич., с	Виноград, с	Опт. Виноград, с
101	0.0239532	0.0172836	0.0121763
201	0.1823425	0.13919326	0.09645523
301	0.6155233	0.4704433260	0.329560323
401	1.4677636	1.1361428326	0.7987197323
501	2.9109864	2.2711334833	1.6029470732
601	5.1334142	4.0243917483	2.8370770073
701	8.3448133	6.5884793748	4.6514172007
801	13.0947333	10.4854465375	7.4338798201
901	19.5909741	16.0785333537	11.624343082
1001	28.395487	23.3195016354	17.0517238082

Данные из таблицы 3 изображены на рисунке 5. Условные обозначения:

- classicOdd - время работы классического алгоритма умножения матриц на квадратных матрицах нечетного размера в секундах;
- vinogradOdd - время работы алгорима Винограда на квадратных матрицах нечетного размера в секундах;
- optOdd - время работы оптимизированного алгорима Винограда на квадратных матрицах нечетного размера в секундах.

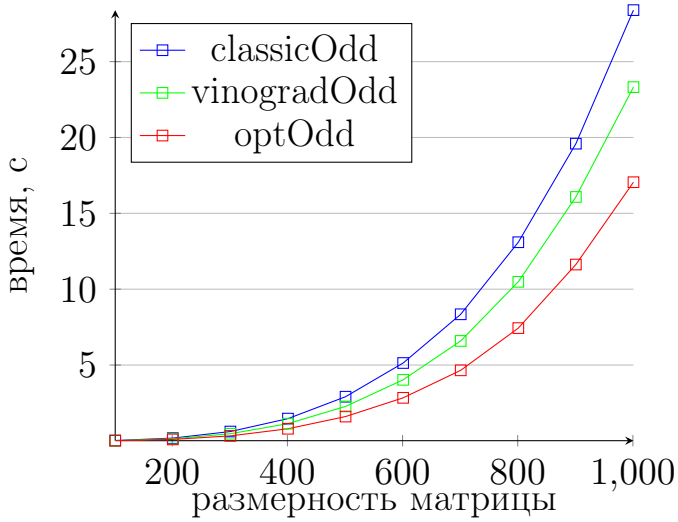

Рис. 5: Результаты замеров времени для матриц нечетного размера

Выводы

В результате проведенных экспериментов было установлено, что:

1. самым быстрым алгоритмом в рамках проведенного эксперимента оказался оптимизированный алгорим Винограда;

2. самым медленным оказался классический алгоритм умножения матриц.

Заключение

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

Был изучен подход к вычислению трудоемкости алгоритмов.

Была дана теоретическая оценка классического алгоритма умножения матриц, алгорима Винограда и оптимизированного алгорима Винограда.

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