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 Анализ алгоритмов умножения матриц

Студент Малышев И. А.

Группа ИУ7-51Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2021 г.

Оглавление

Введение **2**

1 Аналитическая часть **3**
1.1 Стандартный алгоритм . . . 3
1.2 Алгоритм Винограда . . . 3
1.3 Оптимизированный алгоритм Винограда . . . 4
1.4 Вывод . . . 4

2 Конструкторская часть **5**
2.1 Схемы алгоритмов . . . 5
2.2 Модель вычислений . . . 7
2.3 Трудоёмкость алгоритмов . . . 8
2.3.1 Стандартный алгоритм умножения матриц . . . 8
2.3.2 Алгоритм Винограда . . . 8
2.3.3 Оптимизированный алгоритм Винограда . . . 9
2.4 Вывод . . . 9

3 Технологическая часть **10**
3.1 Средства реализации . . . 10
3.2 Реализация алгоритмов . . . 10
3.3 Тестирование . . . 12
3.4 Вывод . . . 13

4 Исследовательская часть **14**
4.1 Технические характеристики . . . 14
4.2 Время выполнения реализаций алгоритмов . . . 14
4.3 Вывод . . . 16

Заключение **17**

Литература **18**

Введение

Матрица – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Две матрицы одинакового размера можно поэлементно сложить или вычесть друг из друга. Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить.

Умножение матриц – это один из базовых алгоритмов, который широко применяется в численных методах, в частности, в машинном обучении, в компьютерной графике для реализации афинных преобразований. Именно поэтому важно, чтобы алгоритм был максимально эффективен по затрачиваемым ресурсам.

Именно поэтому **целью** данной работы является изучение алгоритмов умножения матриц, в частности: обычный алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучение и реализация трёх алгоритмов умножения матриц: обычный, Винограда, оптимизированный Винограда;

2. Сравнительный анализ трудоёмкости алгоритмов на основе расчетов в выбранной модели вычислений;

3. Сравнительный анализ алгоритмов на основе экспериментальных данных.

1 Аналитическая часть

В этом разделе приведён обзор и анализ алгоритмов умножения матриц.

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы:

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

Тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

, где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.3}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.2 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.

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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \tag{1.4}$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырёх умножений - шесть, а вместо трёх 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 позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

1.3 Оптимизированный алгоритм Винограда

Оптимизация алгоритма заключается в следующем:

- в начале происходят предвычисления с помощью двух циклов, которые можно поместить в один, что уменьшит вклад циклов в трудоёмкость;
- в каждом из циклов приходится производить умножение на 2 при индексации, чтобы это исправить можно увеличить шаг с единицы на двойку и границу цикла вдвое.

1.4 Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда. Было выявлено, что алгоритм Винограда отличается от стандартного наличием предварительной обработки строк и столбцов, что позволяет сократить количество умножений, а значит ускорить алгоритм.

2 Конструкторская часть

В этом разделе приводятся схемы алгоритмов умножения матриц, приведённых в аналитической части, и расчёты их трудоёмкости.

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2 и 2.3 показаны схемы алгоритмов сортировки пузырьком, выбором и быстрой сортировки соответственно.

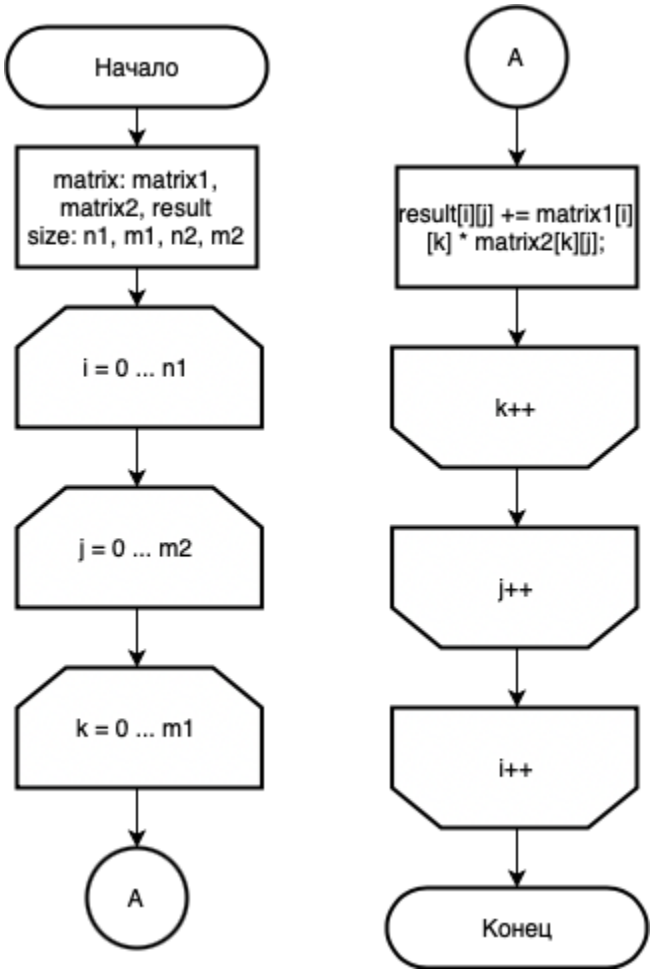

Рис. 2.1: Схема стандартного алгоритма

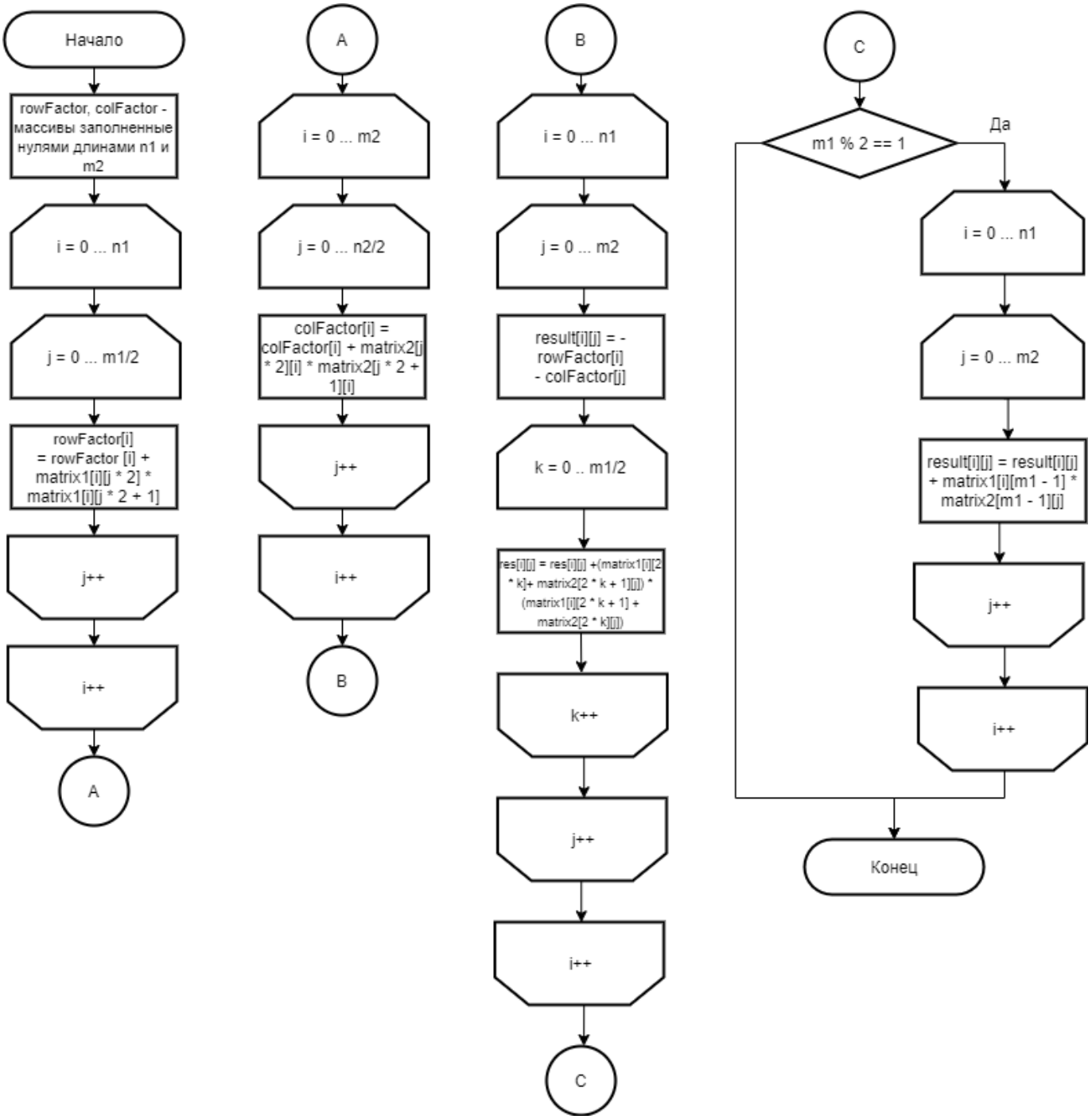

Рис. 2.2: Схема алгоритма Винограда

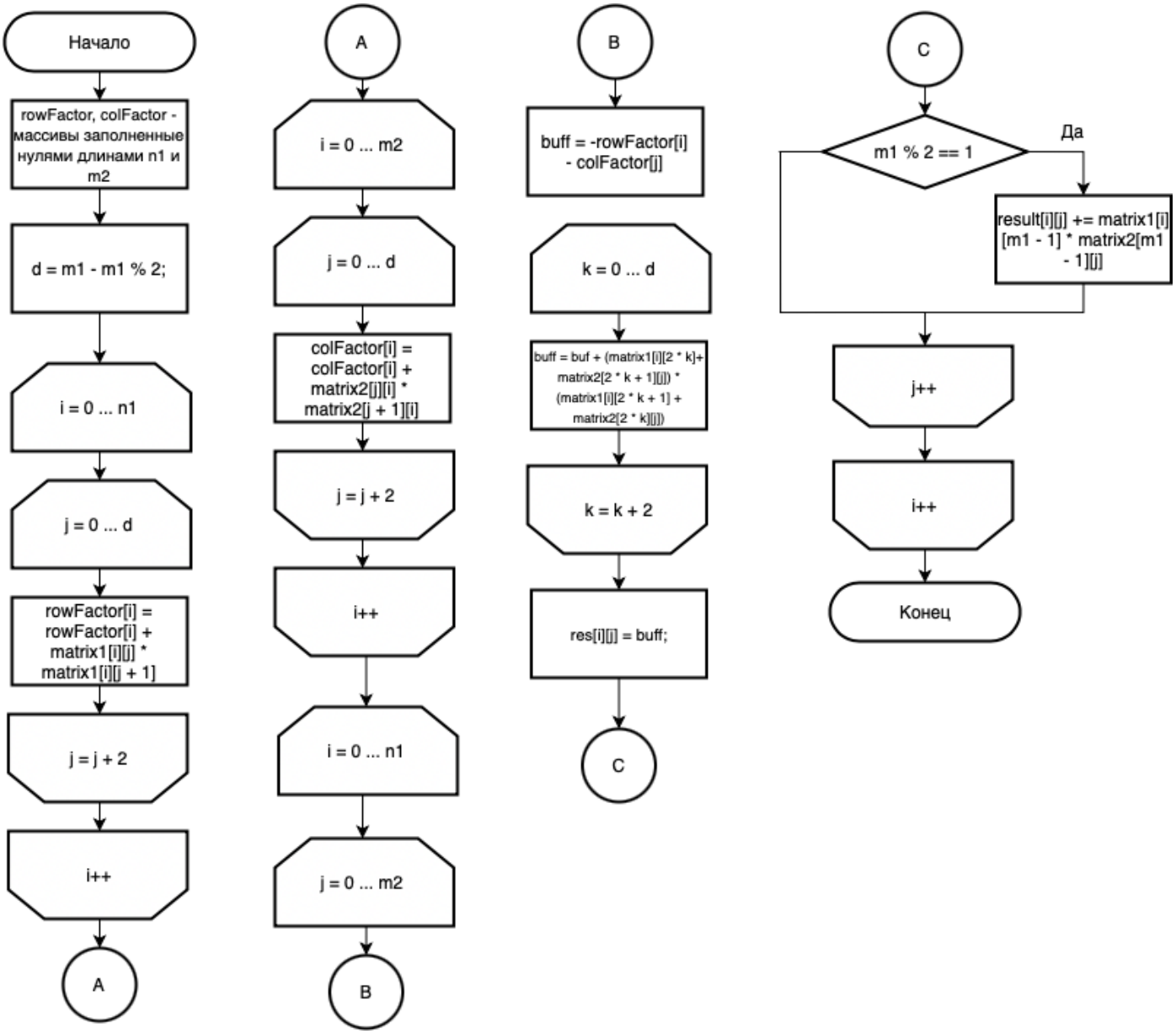

Рис. 2.3: Схема оптимизированного алгоритма Винограда

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

1. Операции из списка (2.1) имеют трудоемкость 1.

$$+, -, /, \%, ==, != , <, >, <=, >=, [], ++, --\tag{2.1}$$

2. Трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}\tag{2.2}$$

3. Трудоемкость цикла рассчитывается, как (2.3).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.3}$$

4. Трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Примем, что размеры первой матрицы (r, s), а второй - (s, c).

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма в выбранной модели вычислений в худшем и лучшем случаях рассчитывается следующим образом:

$$f_{base} = 2 + r(2 + 2 + c(2 + 2 + s(2 + 11))) = 13scr + 4cr + 4r + 2 \tag{2.4}$$

2.3.2 Алгоритм Винограда

Для алгоритма Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, из-за того что отпадает необходимость в последнем цикле.

Трудоёмкость алгоритма Винограда является суммой трудоёмкостей следующих последовательно выполненных действий:

1. Заполнения вектора rowFactor:

$$f_{rowFactor} = 3 + r(2 + 2 + \frac{s}{2}(2 + 11)) = 6.5sr + 4r + 3; \tag{2.5}$$

2. Заполнения вектора colFactor:

$$f_{colFactor} = 2 + c(2 + 2 + \frac{s}{2}(2 + 11)) = 6.5sc + 4r + 2; \tag{2.6}$$

3. Основного цикла заполнения матрицы:

$$f_{cycle} = 2 + r(2 + 2 + c(2 + 2 + 7 + \frac{s}{2}(2 + 23))) = 12.5scr + 11cr + 4r + 2; \tag{2.7}$$

4. Цикла для дополнения умножения, если общий размер нечётный:

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 2 + 2 + r(2 + 2 + c(2 + 13)) = 15cr + 4r + 4, & \text{иначе.} \end{cases} \tag{2.8}$$

Итак, для лучшего случая (чётный размер матриц):

$$f_{vin\_b} = 6.5sr+4r+3+6.5sc+4r+2+12.5scr+11cr+4r+2+2 = 12.5scr+6.5sr+6.5sc+11cr+12r+9; \tag{2.9}$$

Для худшего случая (нечётный общий размер матриц):

$$f_{vin\_w} = 6.5sr + 4r + 3 + 6.5sc + 4r + 2 + 12.5scr + 11cr + 4r + 2 + 15cr + 4r + 4 = \tag{2.10}$$

$$= 12.5scr + 6.5sr + 6.5sc + 26cr + 16r + 11; \tag{2.11}$$

2.3.3 Оптимизированный алгоритм Винограда

Трудоёмкость оптимизированного алгоритма Винограда является суммой трудоёмкостей следующих последовательно выполненных действий:

1. Заполнения вектора rowFactor:

$$f_{rowFactor} = 5 + r(3 + 2 + \frac{s}{2}(3 + 10)) = 6.5sr + 5r + 5; \tag{2.12}$$

2. Заполнения вектора colFactor:

$$f_{colFactor} = 2 + c(2 + 2 + \frac{s}{2}(2 + 11)) = 6.5s + 5r + 2; \tag{2.13}$$

3. Основного цикла заполнения матрицы:

$$f_{cycle} = 3+2+r(2+2+c(2+2+5+\frac{s}{2}(3+15)+f_{last}+3)) = 9scr+12cr+f_{last}cr+4r+5 \tag{2.14}$$

$$f_{last} = \begin{cases} 0, & \text{чётная,} \\ 9, & \text{иначе.} \end{cases} \tag{2.15}$$

Итак, для лучшего случая (чётный размер матриц):

$$f_{vinOpt\_b} = 6.5sr+5r+5+6.5s+5r+2+9scr+12cr+4r+5 = 9scr+6.5sr+6.5sc+12cr+14r+12; \tag{2.16}$$

Для худшего случая (нечётный общий размер матриц):

$$f_{vinOpt\_w} = 6.5sr+5r+5+6.5s+5r+2+9scr+12cr+9cr+4r+5 = 9scr+6.5sr+6.5sc+21cr+14r+12; \tag{2.17}$$

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы алгоритмов умножения матриц, оценены их трудоёмкости в лучшем и худшем случаях.

3 Технологическая часть

В данном разделе приводится реализация алгоритмов, схемы которых были разработаны в конструкторской части. Кроме того, обосновывается выбор технологического стека и проводится тестирование реализованных алгоритмов.

3.1 Средства реализации

В качестве языка программирования был выбран C#, а среду разработки – Visual Studio, т. к. я знаком с данным языком и имею представление о тестировании программ в данном языке. Время работы алгоритмов было замерено с помощью библиотеки System.Diagnostics, класса Stopwatch, который имеет методы для расчёта процессорного времени [1].

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация трёх алгоритмов умножения матриц.

Листинг 3.1: Функция обычного алгоритма умножения матриц

```
1  public Matrix MultiplyStandart(Matrix a)
2  {
3    Matrix result = new Matrix(n, a.M);
4
5    for (int i = 0; i < n; i++)
6    {
7      for (int k = 0; k < a.M; k++)
8      {
9        result[i, k] = 0;
10       for (int j = 0; j < m; j++)
11       result[i, k] += this[i, j] * a[j, k];
12     }
13   }
14
15   return result;
16 }
```

Листинг 3.2: Функция алгоритма умножения матриц Винограда

```
1  public Matrix MultiplyVinograd(Matrix matr)
2  {
3    Matrix result = new Matrix(n, matr.M);
```

```
  double[] rowFactor = new double[n];
  double[] colFactor = new double[matr.M];

  int d = m / 2;
  for (int i = 0; i < n; i++)
  {
    rowFactor[i] = this[i, 0] * this[i, 1];
    for (int j = 1; j < d; j++)
    rowFactor[i] = rowFactor[i] + this[i, 2 * j] * this[i, 2 * j + 1];
  }

  for (int i = 0; i < matr.M; i++)
  {
    colFactor[i] = matr[0, i] * matr[1, i];
    for (int j = 1; j < d; j++)
    colFactor[i] = colFactor[i] + matr[2 * j, i] * matr[2 * j + 1, i];
  }

  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < matr.M; j++)
    {
      result[i, j] = -rowFactor[i] - colFactor[j];
      for (int k = 0; k < d; k++)
      result[i, j] = result[i, j] + (this[i, 2 * k] + matr[2 * k + 1,
         j]) * (this[i, 2 * k + 1] + matr[2 * k, j]);
    }
  }

  if (m % 2 == 1)
  {
    for (int i = 0; i < n; i++)
    for (int j = 0; j < matr.M; j++)
    result[i, j] = result[i, j] + this[i, n - 1] * matr[n - 1, j];
  }

  return result;
}
```

Листинг 3.3: Функция оптимизированного алгоритма умножения матриц Винограда

```
public Matrix MultiplyVinogradOptimised(Matrix matr)
{
  Matrix result = new Matrix(n, matr.M);
  double[] rowFactor = new double[n];
  double[] colFactor = new double[matr.M];

  for (int i = 0; i < n; i++)
  {
    rowFactor[i] = colFactor[i] = 0;
```

```
    for (int j = 0; j < m - 1; j += 2)
    {
      rowFactor[i] += this[i, j] * this[i, j + 1];
      colFactor[i] += matr[j, i] * matr[j + 1, i];
    }
  }

  for (int i = 0; i < n; i++)
  for (int j = 0; j < matr.M; j++)
  {
    double res = -rowFactor[i] - colFactor[j];
    for (int k = 0; k < m - 1; k += 2)
    res += (this[i, k] + matr[k + 1, j]) * (this[i, k + 1] + matr[k, j]);

    result[i, j] = res;
  }

  if (m % 2 != 0)
  for (int i = 0; i < n; i++)
  for (int j = 0; j < matr.M; j++)
  result[i, j] += this[i, m - 1] * matr[matr.N - 1, j];

  return result;
}
```

3.3 Тестирование

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 2 & 3 & 7 \\ 5 & 1 & 10 \\ 6 & -1 & 4 \end{pmatrix}$	$\begin{pmatrix} 30 & 2 & 39 \\ 56 & 8 & 81 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 2 & 3 & 7 \\ 5 & 1 & 10 \end{pmatrix}$	$\begin{pmatrix} 12 & 5 & 27 \\ 26 & 13 & 61 \end{pmatrix}$
(2)	(2)	(4)
$\begin{pmatrix} 1 & -2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & 6 \\ 4 & 12 & 18 \\ 4 & 12 & 18 \end{pmatrix}$

Таблица 3.1: Тестирование функций

Все тесты пройдены успешно.

3.4 Вывод

В данном разделе были реализованы алгоритмы умножения матриц: обычный алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. Кроме того, реализации были успешно протестированы.

4 Исследовательская часть

В данном разделе проводится сравненительный анализ реализованных алгоритмов по процессорному времени.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Windows 10 Home 64-bit;
- Оперативная память: 16 GB;
- Процессор: 4.0 GHz 4-ядерный процессор Intel Core i7-4790K.

4.2 Время выполнения реализаций алгоритмов

Для сравнительного анализа времени выполнения реализаций алгоритмов был проведен эксперимент. Для замеров были сформированы следующие пары квадратных матриц:

1. Порядок матриц чётный и варьируется в пределах от 100 до 1100 с шагом 100;
2. Порядок матриц нечётный и варьируется в пределах от 1001 до 2101 с шагом 100;

В таблицах 4.2 и 4.2 представлены результаты замеров. Время измерялось 10 раз для каждой пары матриц, после усреднялось.

Таблица 4.1: Таблица времени выполнения (в секундах) алгоритмов при чётных размерах

Размер матрицы	Стандартный	Виноград	Оптимизированный
100	0.024	0.014	0.009
200	0.17	0.106	0.075
300	0.527	0.386	0.29
400	1.476	0.961	0.642
500	3.162	2.055	1.296
600	5.463	4.036	2.314
700	9.451	6.788	4.836
800	14.027	9.366	5.86
900	18.884	14.12	11.594
1000	26.829	18.049	13.348

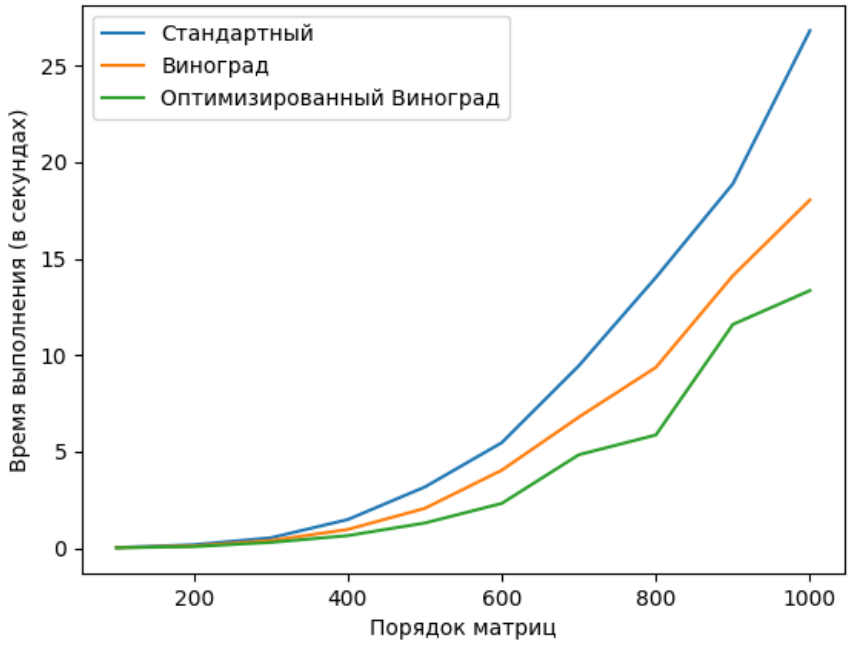

Рис. 4.1: Зависимость времени умножения квадратных матриц чётного порядка от величины порядка

Таблица 4.2: Таблица времени выполнения (в секундах) алгоритмов при нечётных размерах

Размер матрицы	Стандартный	Виноград	Оптимизированный
101	0.020	0.013	0.009
201	0.158	0.108	0.072
301	0.536	0.393	0.246
401	1.337	0.954	0.604
501	2.807	1.835	1.253
601	4.945	3.405	2.124
701	8.353	5.441	3.573
801	13.452	9.147	5.968
901	18.569	14.258	9.55
1001	26.865	21.295	16.106

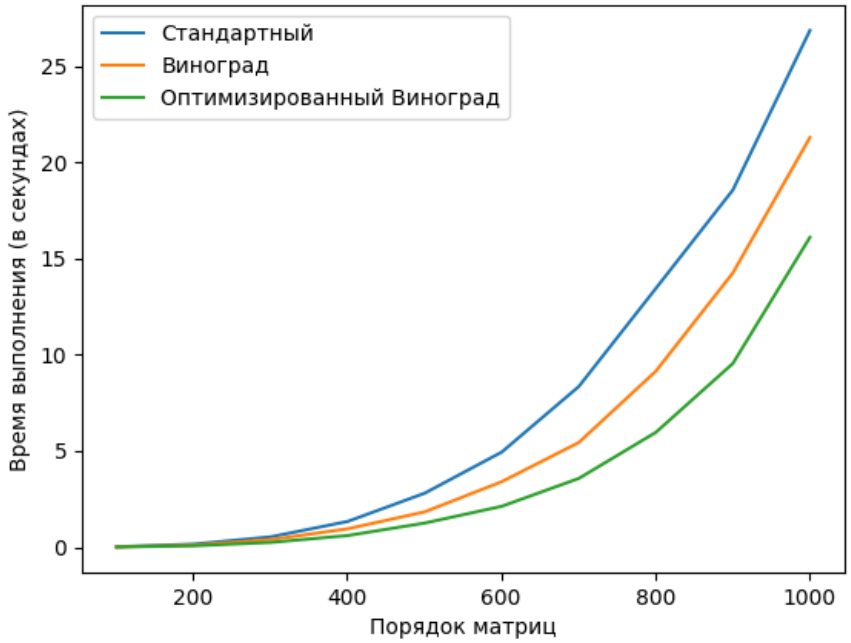

Рис. 4.2: Зависимость времени умножения квадратных матриц нечётного порядка от величины порядка

4.3 Вывод

На основе замеров процессорного времени было получено, что алгоритм Винограда в среднем в 1.4-1.7 раз быстрее, чем обычный алгоритм умножения матриц и немного хуже оптимизированного.

Также выявлено, что матрицы нечётного порядка перемножаются немного медленее, т. к. время тратится ещё и на обработку последнего столбца.

Заключение

В рамках данной лабораторной работы:

1. Были изучены и реализованы 3 алгоритма умножения матриц: обычный, Винограда, оптимизированный Винограда;
2. Был произведён анализ трудоёмкости алгоритмов на основе теоретических расчётов и выбранной модели вычислений;
3. Был сделан сравнительный анализ алгоритмов на основе экспериментальных данных.

На основании анализа трудоёмкости алгоритмов в выбранной модели вычислений было показано, что улучшенный алгоритм Винограда имеет меньшую сложность, нежели простой алгоритм умножения матриц. На основании замеров времени исполнения алгоритмов, был сделан вывод о том, что алгоритм Винограда в среднем в 1.4-1.7 раз быстрее, чем обычный алгоритм умножения матриц и немного хуже оптимизированного. При этом матрицы нечётного порядка перемножаются немного медленее, чем чётного.

Поставленная цель была достигнута.

Литература

1. Свойство Process.UserProcessorTime [Электронный ресурс]. Режим доступа: https://docs.microsoft.c
ru/dotnet/api/system.diagnostics.stopwatch?view=net-5.0. Дата обращения: 02.10.2021