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 курсу "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Чепиго Д.С.

Группа ИУ7-54Б

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

Москва — 2022 г.

Содержание

Введение **3**

1 Аналитическая часть **4**

1.1 Стандартный алгоритм 4

1.2 Алгоритм Винограда 5

2 Конструкторская часть **6**

2.1 Разработка стандартного алгоритма умножения матриц 6

2.2 Разработка алгоритма Винограда умножения матриц 6

2.3 Разработка оптимизированного алгоритма Винограда умножения матриц 8

2.4 Модель вычислений 9

2.5 Трудоемкость алгоритмов 9

2.5.1 Стандартный алгоритм умножения матриц 9

2.5.2 Алгоритм Винограда умножения матриц 10

2.5.3 Оптимизированный алгоритм Винограда умножения матриц 11

3 Технологическая часть **13**

3.1 Требования к ПО 13

3.2 Средства реализации 13

3.3 Реализация алгоритмов 13

4 Исследовательская часть **17**

4.1 Технические характеристики 17

4.2 Пример работы программы 17

4.3 Время выполнения реализованных алгоритмов 19

Заключение **22**

Список использованных источников **23**

Введение

Матрица – математический объект, который активно применяется почти во всех отрасли человеческой деятельности. Они используются в математике, в физике, в технике, в экономике, в теории управления, статистики и других областях науки и знаний.

Цель лабораторной работы – изучить и исследовать трудоемкость алгоритмов умножения матриц.

Задачи лабораторной работы:

- изучить и реализовать 3 алгоритма умножения матриц: стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда;
- выбрать инструменты для замера процессорного времени выполнения реализаций алгоритмов;
- провести анализ затрат работы алгоритмов по времени;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

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 [1].

Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. Размерность получившийся матрице будет равна количеству строк первой матрицы и количеству столбцов второй матрицы [2].

Умножение матриц некоммутативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга.

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы размерности $l \times m$ и $m \times n$

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

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.

1.2 Алгоритм Винограда

Алгоритм Винограда – алгоритм умножения квадратных матриц, предложенный в 1987 году Ш. Виноградом [2]. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц [3].

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:

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

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: 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 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.

В случае нечетного значений размера изначальной матрицы n, следует произвести еще одну операцию – добавление произведения последних элементов соответствующих строк и столбцов.

2 Конструкторская часть

2.1 Разработка стандартного алгоритма умножения матриц

На рисунке 2.1 приведена схема стандартного алгоритма умножения матриц.

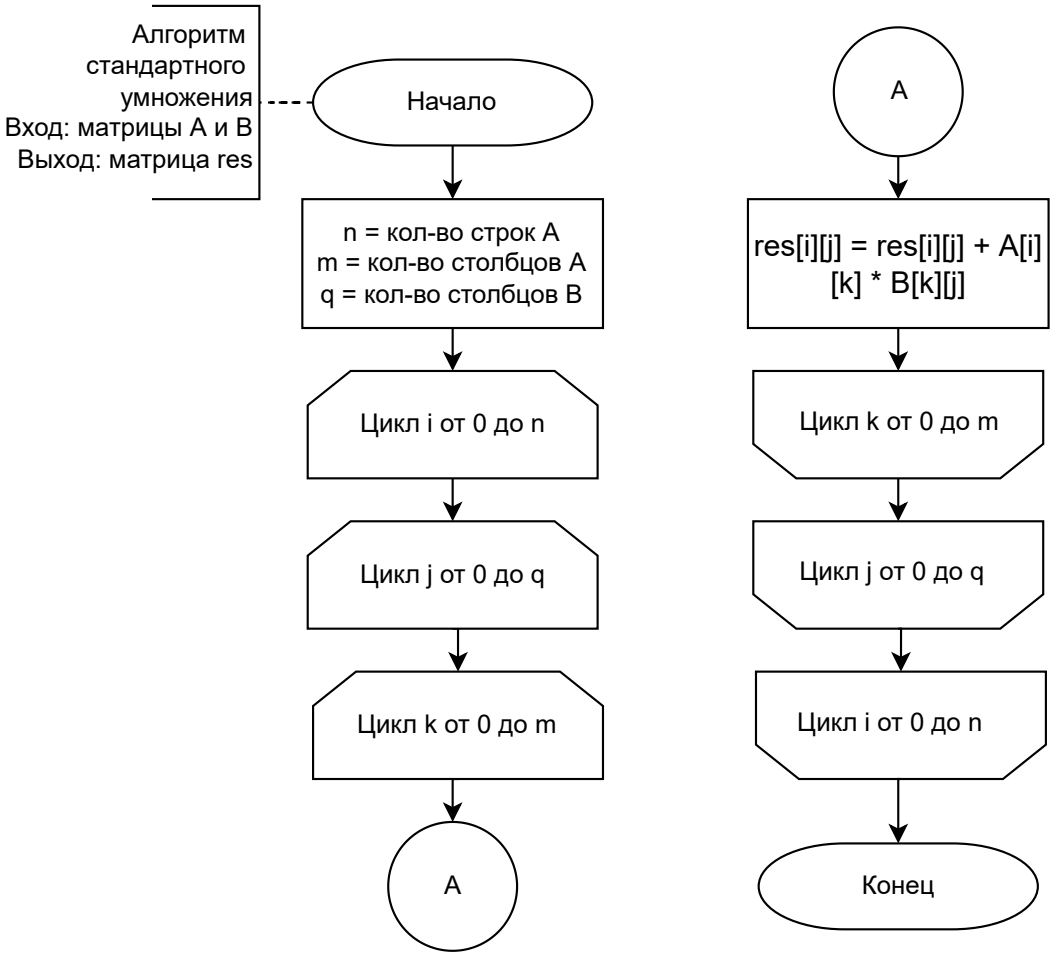

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

2.2 Разработка алгоритма Винограда умножения матриц

На рисунке 2.2 приведена схема алгоритма Винограда умножения матриц.

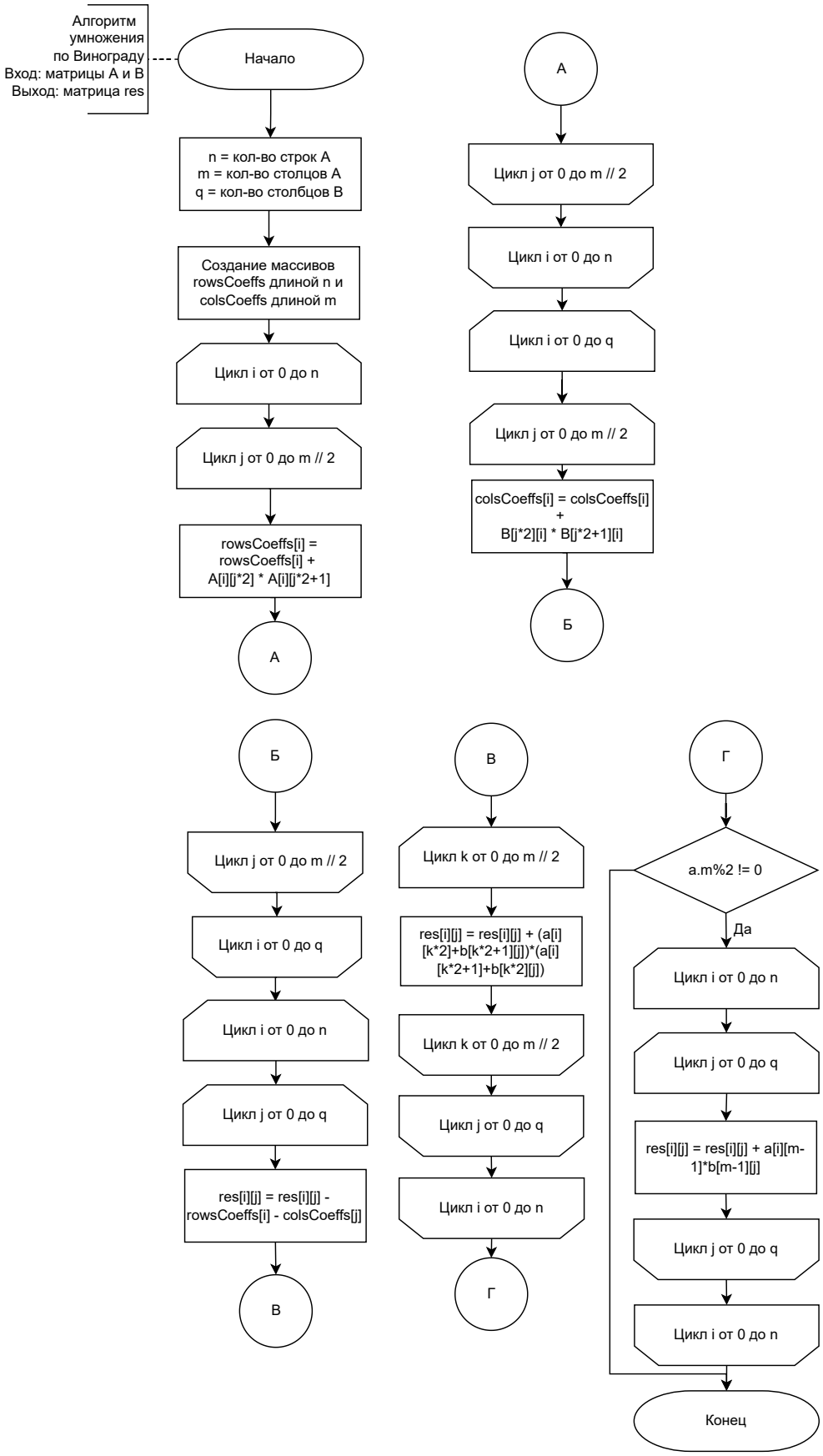

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц

2.3 Разработка оптимизированного алгоритма Винограда умножения матриц

На рисунке 2.3 приведена схема оптимизированного алгоритма Винограда умножения матриц.

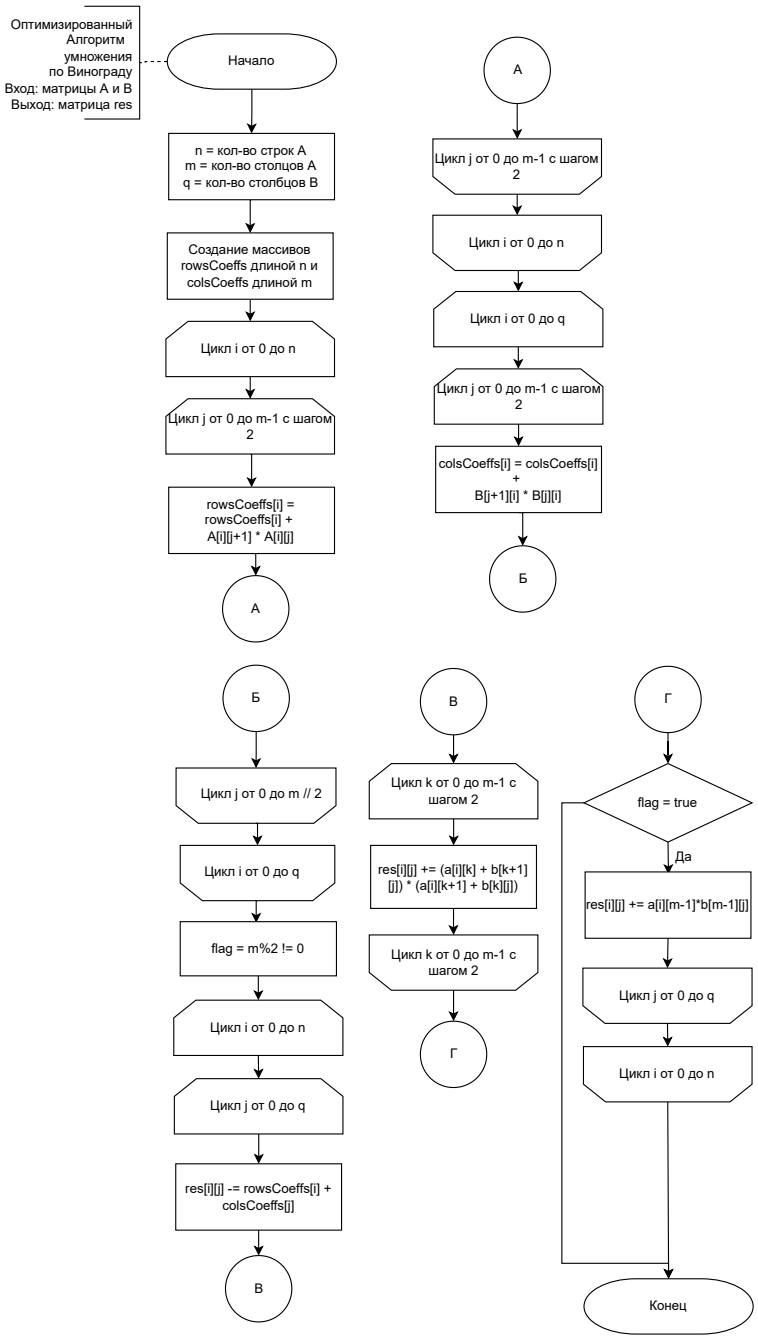

Рисунок 2.3 – Схема оптимизированного алгоритма Винограда умножения матриц

2.4 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

Операции из списка (2.1) имеют трудоемкость 1.

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

Операции из списка (2.2) имеют трудоемкость 2.

$$*, /, \%\tag{2.2}$$

Трудоемкость оператора выбора `if условие then A else B` рассчитывается как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}\tag{2.3}$$

Трудоемкость цикла рассчитывается как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}})\tag{2.4}$$

Трудоемкость вызова функции равна 0.

2.5 Трудоемкость алгоритмов

Во всех алгоритмах создается дополнительная матрица, куда записывается результат. Трудоемкость инициализации не была учтена, так как это действие не является трудоемким.

2.5.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из следующих составляющих.

Трудоёмкость внешнего цикла по $i \in [0..M-1]$: $f = 2 + M \cdot (2 + f_{body})$.

Трудоёмкость цикла по $j \in [0..N-1]$: $f = 2 + N \cdot (2 + f_{body})$.

Трудоёмкость цикла по $k \in [0..K-1]$: $f = 2 + 11K$.

Учитывая, что трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела:

$$f_{standard} = 2 + M \cdot (4 + N \cdot (4 + 11K)) \approx 11MNK \tag{2.5}$$

2.5.2 Алгоритм Винограда умножения матриц

Трудоёмкость алгоритма Винограда состоит из следующих составляющих.

Трудоёмкость создания и инициализации массивов MH и MV:

$$f_{init} = M + N \tag{2.6}$$

Трудоёмкость заполнения массива MH:

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 11) \tag{2.7}$$

Трудоёмкость заполнения массива MV:

$$f_{MV} = 2 + K(2 + \frac{N}{2} \cdot 11) \tag{2.8}$$

Трудоёмкость цикла заполнения для чётных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)) \tag{2.9}$$

Трудоёмкость цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный:

$$f_{last} = \begin{cases} 2, & \text{размер чётный,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \tag{2.10}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11{,}5 \cdot MNK \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 11{,}5 \cdot MNK \tag{2.12}$$

2.5.3 Оптимизированный алгоритм Винограда умножения матриц

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- вычисление происходит заранее;
- вместо деления на 2 в условии циклов, используется увеличение счётчика циклов на 2;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

Трудоёмкость улучшенного алгоритма Винограда состоит из следующих составляющих.

Трудоёмкость создания и инициализации массивов MH и MV:

$$f_{init} = M + N \tag{2.13}$$

Трудоёмкость заполнения массива MH:

$$f_{MH} = 2 + K(2 + \frac{M}{2} \cdot 8) \tag{2.14}$$

Трудоёмкость заполнения массива MV:

$$f_{MV} = 2 + K(2 + \frac{M}{2} \cdot 8) \tag{2.15}$$

Трудоёмкость цикла заполнения для чётных размеров:

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)) \tag{2.16}$$

Трудоёмкость условия для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный:

$$f_{last} = \begin{cases} 1, & \text{размер чётный,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \tag{2.17}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.18}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.19}$$

3 Технологическая часть

3.1 Требования к ПО

К программе предъявляется ряд требований:

- входными данными являются две матрицы A и B (сначала вводятся их размерности, затем элементы);
- элементы матрицы – целые числа;
- на выходе получается матрица res – результат умножения введенных пользователем матриц.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран C++ – компилируемый, статически типизированный язык программирования общего назначения [4].

Данный выбор обусловлен поддержкой языком парадигмы объектно – ориентированного программирования и наличием методов для замера процессорного времени.

Время работы реализованных алгоритмов было замерено с помощью библиотеки chrono [5].

3.3 Реализация алгоритмов

В листинге 3.1 приведена реализация стандартного алгоритма умножения матриц.

Листинг 3.1 – Реализация стандартного алгоритма умножения матриц

```
1 Matrix MultiplicatoinMatrix::standartMultiplication()
2 {
3     vector<vector<int>> tempArray;
4     tempArray.resize(firstRows);
5
```

```
    for (int i = 0; i < firstRows; i++)
        tempArray[i].resize(secondColumns);

    for (int i = 0; i < firstRows; i++)
        for (int j = 0; j < firstColumns; j++)
            for (int k = 0; k < secondColumns; k++)
                tempArray[i][k] = tempArray[i][k] + first[i][j] * second[j][k];

    Matrix resultMatrix(firstMatrix.getRows(), secondMatrix.getColumns(),
        tempArray);

    return resultMatrix;
}
```

В листинге 3.2 приведена реализация алгоритма Винограда умножения матриц.

Листинг 3.2 – Реализация алгоритма Винограда умножения матриц

```
Matrix MultiplicatoinMatrix::winogradMultiplication()
{
    vector<int> vectorH(firstRows, 0);
    vector<int> vectorV(secondColumns, 0);

    for (int i = 0; i < firstRows; i++)
        for (int j = 0; j < firstColumns / 2; j++)
            vectorH[i] = vectorH[i] + first[i][2 * j] * first[i][2 * j + 1];

    for (int i = 0; i < secondColumns; i++)
        for (int j = 0; j < secondRows / 2; j++)
            vectorV[i] = vectorV[i] + second[2 * j][i] * second[2 * j + 1][i];

    vector<vector<int>> tempArray;
    tempArray.resize(firstRows);

    for (int i = 0; i < firstRows; i++)
        tempArray[i].resize(secondColumns);

    for (int i = 0; i < firstRows; i++)
        for (int k = 0; k < secondColumns; k++)
        {
            tempArray[i][k] = tempArray[i][k] - vectorH[i] - vectorV[k];

            for (int j = 0; j < firstColumns / 2; j++)
                tempArray[i][k] = tempArray[i][k] + (first[i][2 * j] + second[2
                    * j + 1][k]) *
                (first[i][2 * j + 1] + second[2 * j][k]);
        }

```

```
30
31      if (firstColumns % 2 == 1)
32          for (int i = 0; i < firstRows; i++)
33              for (int j = 0; j < secondColumns; j++)
34                  tempArray[i][j] = tempArray[i][j] + first[i][firstColumns - 1]
                        * second[secondRows - 1][j];
35
36      Matrix resultMatrix(firstMatrix.getRows(), secondMatrix.getColumns(),
            tempArray);
37
38      return resultMatrix;
39  }
```

В листинге 3.3 приведена реализация оптимизированного алгоритма Винограда умножения матриц.

Листинг 3.3 – Реализация оптимизированного алгоритма Винограда умножения матриц

```
1
2   Matrix MultiplicatoinMatrix::winogradMultiplicationPro()
3   {
4       vector<int> vectorH(firstRows, 0);
5       vector<int> vectorV(secondColumns, 0);
6
7       for (int i = 0; i < firstRows; i++)
8           for (int j = 0; j < firstColumns / 2; j++)
9               vectorH[i] += first[i][j << 1] * first[i][(j << 1) + 1];
10
11      for (int i = 0; i < secondColumns; i++)
12          for (int j = 0; j < secondRows / 2; j++)
13              vectorV[i] += second[j << 1][i] * second[(j << 1) + 1][i];
14
15      vector<vector<int>> tempArray;
16      tempArray.resize(firstRows);
17
18      for (int i = 0; i < firstRows; i++)
19          tempArray[i].resize(secondColumns);
20
21      int flag =  0;
22
23      if (firstColumns % 2 == 1)
24          flag = 1;
25
26      for (int i = 0; i < firstRows; i++)
27          for (int k = 0; k < secondColumns; k++)
28          {
29              if (flag)
30                  tempArray[i][k] += first[i][firstColumns - 1] *
```

```
                    second[secondRows - 1][k];
31              tempArray[i][k] -= (vectorH[i] + vectorV[k]);
32
33              for (int j = 0; j < firstColumns / 2; j++)
34                  tempArray[i][k] += (first[i][j << 1] + second[(j << 1) + 1][k])
                        *
35                  (first[i][(j << 1) + 1] + second[j << 1][k]);
36          }
37
38      Matrix resultMatrix(firstMatrix.getRows(), secondMatrix.getColumns(),
            tempArray);
39
40      return resultMatrix;
41  }
```

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система Ubuntu 22.04.1 LTS Linux x86_64 [6];
- память 8 ГБ;
- процессор Intel® Core™ i3-7130U.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Пример работы программы

На рисунке 4.1 представлен пример работы программы. Вводится размерность первой матрицы и ее элементы, аналогично вводится вторая матрица. Количество столбцов первой матрицы совпадает с количеством строк второй матрицы, значит умножение возможно. Далее выводится результирующая матрица и время выполнения каждого реализованного алгоритма умножения матриц. Все результирующие матрицы совпадают и найдены верно.

```
dashori@fossa ~/P/A/b/l/code (main)> ./a.out
Ввод первой матрицы.
Введите количество строк: 3
Введите количество столбцов: 4
Введите элементы матрицы:
1 2 3 4
0 0 0 0
-1 2 -3 4
Ввод второй матрицы.
Введите количество строк: 4
Введите количество столбцов: 5
Введите элементы матрицы:
1 2 3 4 5
0 0 0 0 0
-1 2 -3 4 -5
-1 -1 -1 -1 -1
Стандартное умножение.
Время: 27 (микросекунды)
Матрица:
-6 4 -10 12 -14
0 0 0 0 0
-2 -12 2 -20 6

Алгоритм Винограда.
Время: 33 (микросекунды)
Матрица:
-6 4 -10 12 -14
0 0 0 0 0
-2 -12 2 -20 6

Оптимизированный алгоритм Винограда.
Время: 26 (микросекунды)
Матрица:
-6 4 -10 12 -14
0 0 0 0 0
-2 -12 2 -20 6
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения реализованных алгоритмов

Замеры времени работы реализованных алгоритмов для квадратных матриц каждой размерности проводились 1000 раз. Каждый раз элементы матриц генерировались случайно – числа из диапазона (-INT_MAX, INT_MAX) [7]. В качестве результата, представленного в таблице 4.1, взято среднее время на каждой размерности матрицы.

Таблица 4.1 – Результаты замеров времени реализованных алгоритмов умножения матриц в микросекундах

Размерность квадратной матрицы	Стандартный	Алгоритм Винограда	Оптимизированный алгоритм Винограда
2	2	3	3
51	3662	2913	2445
52	3549	2792	2408
101	21560	16506	13810
102	22220	16893	14266
201	167384	128039	107315
202	171351	131540	109524
301	561385	446087	371439
302	588042	466535	388604
401	1341353	1159066	930428
402	1338481	1151987	924616
501	2626258	2321114	1845890
502	2625640	2311815	1828466

На рисунке 4.2 представлена зависимость времени работы реализованных алгоритмов умножения матриц, размерность которых является четной. На рисунке 4.3 представлена зависимость времени работы реализованных алгоритмов умножения матриц, размерность которых является нечетной.

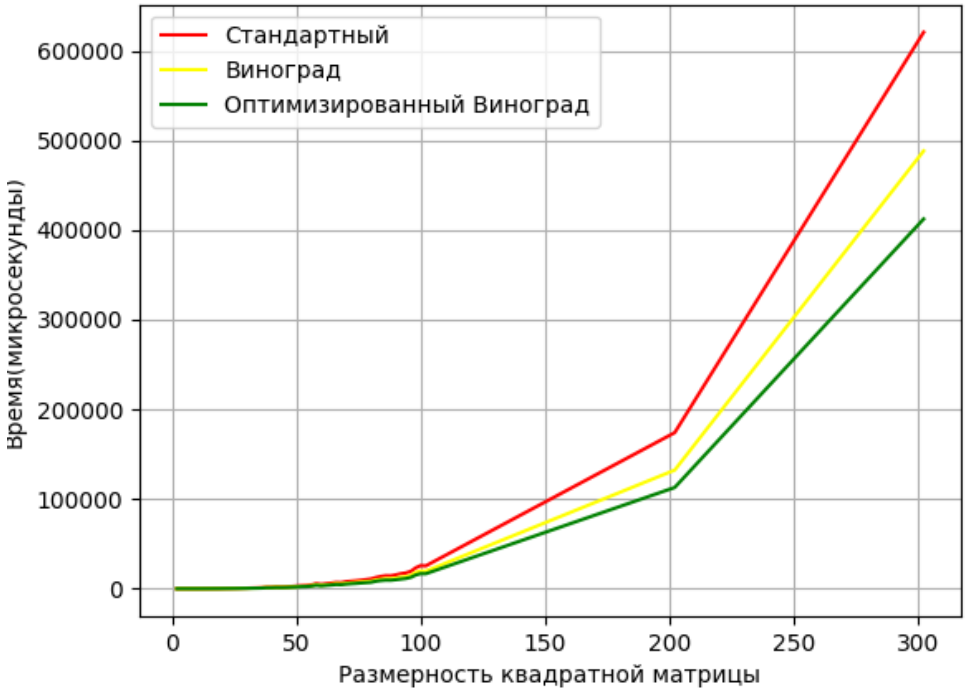

Рисунок 4.2 – Зависимость времени работы реализованных алгоритмов умножения матриц, размерность которых является четной

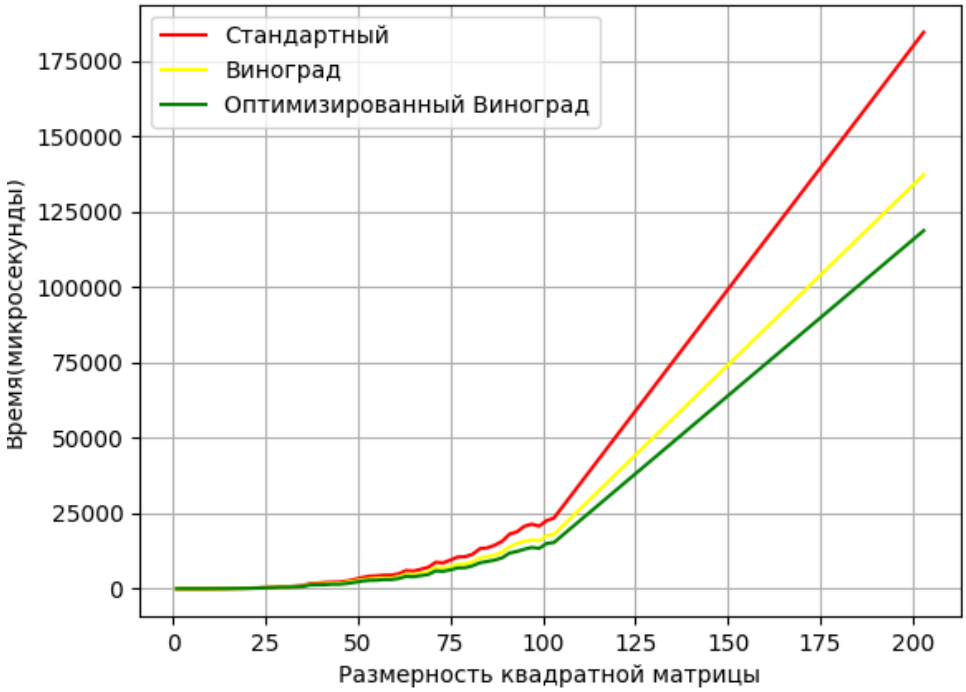

Рисунок 4.3 – Зависимость времени работы реализованных алгоритмов умножения матриц, размерность которых является нечетной

Стандартный алгоритм умножения матриц работает дольше, чем алгоритм Винограда и оптимизированный алгоритм Винограда. Например, на рамерах матриц, равных 502, стандартный алгоритм умножения матриц работает дольше в 1,13 раз чем алгоритм Винограда и в 1,4 раз чем оптимизированный алгоритм Винограда.

Также, сравнивая значения времени в таблице 4.1 можно сделать вывод, что алгоритм Винограда и оптимизированный алгоритм Винограда работают медленнее при нечётных размерах матриц.

Заключение

В ходе выполнения лабораторной работы поставленная цель была достигнута: были изучены и исследованы трудоемкости алгоритмов умножения матриц.

В ходе выполнения лабораторной работы были решены все задачи:

- изучены и реализованы 3 алгоритма умножения матриц: стандартный алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда;
- проведен анализ затрат работы реализованных алгоритмов по времени;
- на основе полученных в ходе экспериментов данных были сделаны выводы по поводу эффективности всех реализованных алгоритмов;
- был подготовлен отчет по лабораторной работе.

Результат замерных экспериментов реализованных алгоритмов показал, что самым быстрым алгоритмом умножения матриц является оптимизированный алгоритм Винограда. Например, на размерах матриц равных 502 оптимизированный алгоритм Винограда превосходит в 1.4 раза стандартный алгоритм и в 1.26 алгоритм Винограда.

Список использованных источников

[1] И. В. Белоусов. Матрицы и определители [Электронный ресурс]. Режим доступа: https://eqworld.ipmnet.ru/ru/library/books/Belousov2006ru.pdf(дата обращения: 20.10.2022).

[2] Умножение матриц [Электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 20.10.2022).

[3] Group-theoretic Algorithms for Matrix Multiplication / H. Cohn, R. Kleinberg, B. Szegedy et al [Электронный ресурс]. Режим доступа: https://arxiv.org/pdf/math/0511460.pdf(дата обращения: 20.10.2022).

[4] Язык программирования C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/(дата обращения: 20.10.2022)

[5] Стандарт языка C++ [Электронный ресурс]. Режим доступа: https://isocpp.org/files/papers/N4860.pdf(дата обращения: 20.10.2022)

[6] Операционная система Ubuntu 22.04 LTS [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/jammy/(дата обращения: 20.10.2022)

[7] Стандарт языка Си [Электронный ресурс]. Режим доступа: https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1256.pdf(дата обращения: 20.10.2022)