



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:

Пронин А. С.

Группа:

ИУ7-52Б

Преподаватель:

Волкова Л. Л.

Оценка:

Москва

2021

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Умножение матриц	4
1.2 Классический алгоритм умножения матриц	5
1.3 Алгоритм Винограда	5
2 Конструкторский раздел	6
2.1 Требования к ПО	6
2.2 Схемы алгоритмов	6
2.3 Оптимизация алгоритма Винограда	6
2.4 Вывод	10
3 Технологический раздел	11
3.1 Выбор инструментов	11
3.2 Реализация алгоритмов	11
3.3 Тестирование	13
3.4 Вывод	14
4 Исследовательский раздел	15
4.1 Пример работы	15
4.2 Сравнительный анализ времени выполнения алгоритмов . .	15
4.3 Оценка трудоёмкости	17
4.4 Вывод	22
Заключение	24
Список использованных источников	25

Введение

Цель работы – изучение алгоритмов умножения матриц, оценка их трудоёмкости и получение навыков в улучшении алгоритмов.

Задачи работы:

- оптимизировать алгоритм Винограда;
- реализовать три алгоритма умножения матриц – классический, алгоритм Винограда и улучшенный алгоритм Винограда;
- оценить время выполнения реализованных алгоритмов;
- рассчитать трудоемкость каждого алгоритма умножения.

1 Аналитический раздел

Результатом умножения матриц $A_{m \times n}$ и $B_{n \times k}$ будет матрица $C_{m \times k}$ такая, что элемент матрицы C , стоящий в i -той строке и j -том столбце (c_{ij}), равен сумме произведений элементов i -той строки матрицы A на соответствующие элементы j -того столбца матрицы B :

$$c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \dots + a_{in} \cdot b_{nj} [1]$$

В данной лабораторной работе рассматриваются следующие алгоритмы стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

1.1 Умножение матриц

Матрицей A размера $[m \times n]$ называется прямоугольная таблица чисел, функций или алгебраических выражений, содержащая m строк и n столбцов. Числа m и n определяют размер матрицы.[2] Если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить. У произведения будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Пусть даны две прямоугольные матрицы A и B размеров $[m \times n]$ и $[n \times k]$ соответственно. В результате произведения матриц A и B получим матрицу C размера $[m \times k]$. Тогда матрица C (1.1)

$$C_{mk} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mk} \end{pmatrix}, \quad (1.1)$$

где элементы матрицы равны (1.2)

$$c_{ij} = \sum_{r=1}^n a_{ir} b_{rj} \quad (i = \overline{1, m}; j = \overline{1, k}) \quad (1.2)$$

будет называться произведением матриц A и B [2].

1.2 Классический алгоритм умножения матриц

Реализация классического алгоритма умножения двух матриц заключается в реализации вычисления элементов итоговой матрицы по формуле 1.2. То есть по определению.

1.3 Алгоритм Винограда

Подход алгоритма Винограда является иллюстрацией общей методологии, начатой в 1979 году на основе билинейных и трилинейных форм, благодаря которым большинство усовершенствований для умножения матриц были получены [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 (1.3)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \quad (1.3)$$

Равенство (1.3) можно переписать в виде (1.4)

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

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. В случае нечетных размеров векторов, после всех вычислений добавим недостающую сумму элементов $v_5 + w_5$ в цикле по элементам результирующей матрицы.

Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторский раздел

В данном разделе представлены требования к разрабатываемому ПО, схемы алгоритмов умножения матриц, а также оптимизация алгоритма Винограда.

2.1 Требования к ПО

К программе предъявляется ряд требований:

- корректное умножение матриц размером до $[N \times N]$, где $N \in [0 : 2000]$;
- при матрицах неправильных размеров программа не должна аварийно завершаться.

2.2 Схемы алгоритмов

Ниже представлены схемы следующих алгоритмов сортировки:

- схема классического алгоритма умножения матриц (Рисунок ??);
- схема алгоритма Винограда (рисунки ?? - ??);

2.3 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 3 оптимизации:

1. избавление от деления в условии цикла;
2. замена двух операций $+$ и $=$ на одну $+=$ (это нам позволяет сделать язык программирования);
3. внесение обработки нечётных размеров векторов в цикл заполнения ячеек матрицы результата.

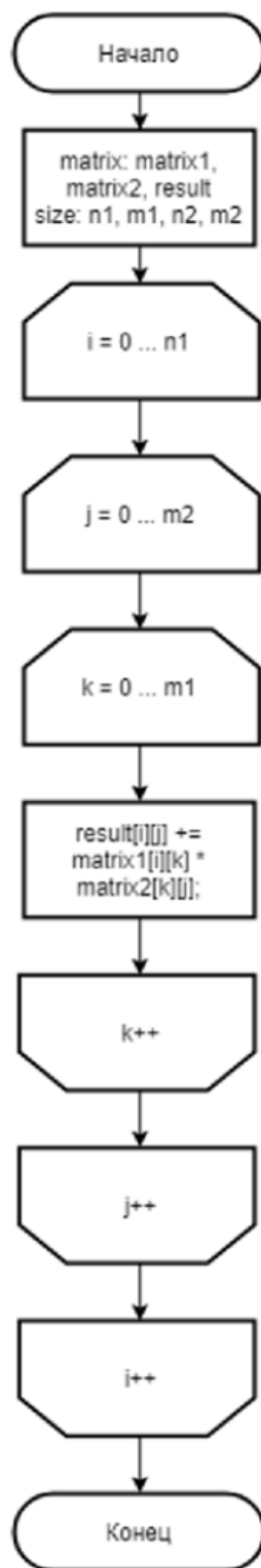


Рисунок 2.1 – Схема классического алгоритма умножения матриц

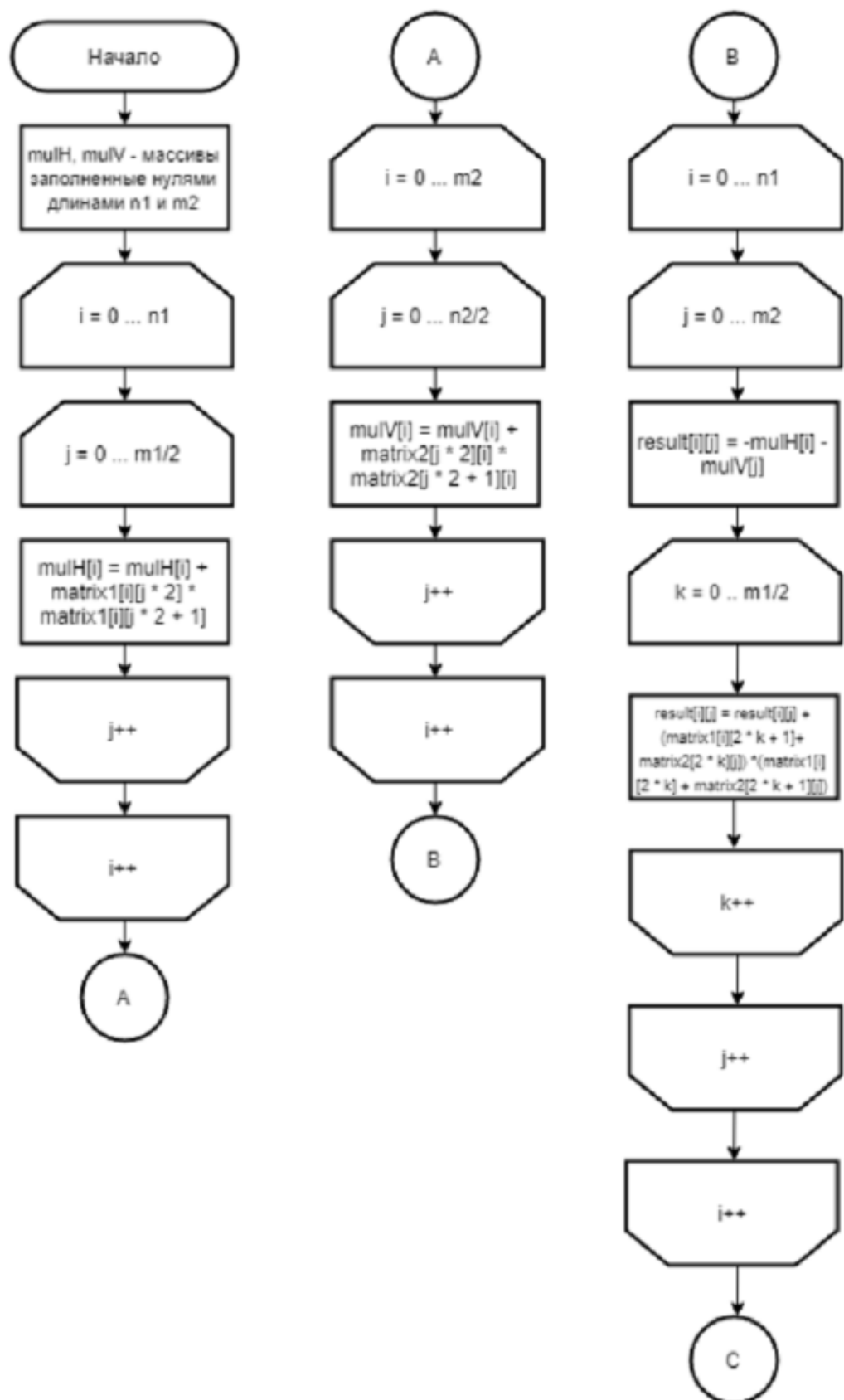


Рисунок 2.2 – Схема алгоритма Винограда

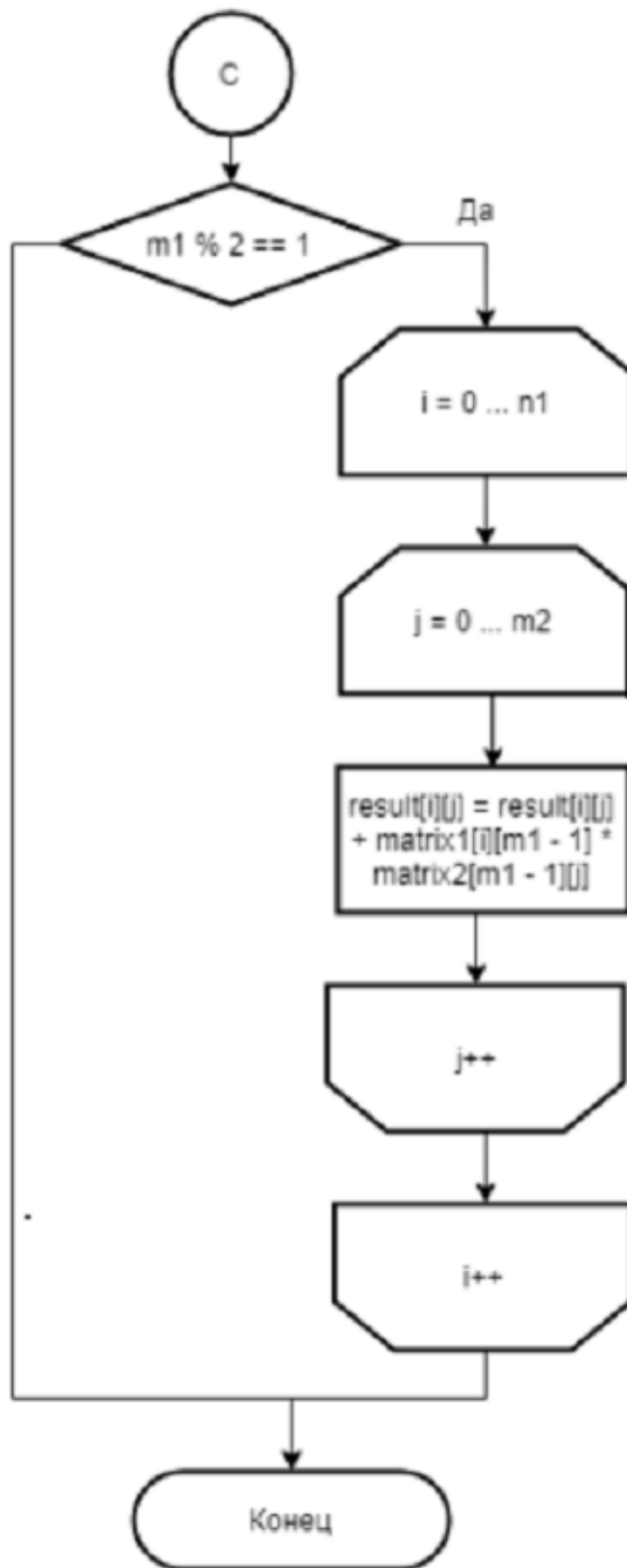


Рисунок 2.3 – Схема алгоритма Винограда (продолжение)

2.4 Вывод

В данном разделе были представлены требования к разрабатываемому ПО, произведена модификация алгоритма Винограда и разработаны схемы алгоритмов умножения матриц.

3 Технологический раздел

В данном разделе представлены выбор инструментов для реализации и оценки алгоритмов, листинги полученного кода, а также функциональное тестирование.

3.1 Выбор инструментов

По-скольку наиболее освоенным языком для разработчика является `c++`, для реализации алгоритмов был выбран именно он, т.к. таким образом работа будет проделана наиболее быстро и качественно.

Соответственно для компиляции кода будет использоваться компилятор `G++`.

Чтобы оценить время выполнения программы будет замеряться процессорное время, т.к. таким образом будут получены данные подходящие для целесообразного сравнения алгоритмов. Для замера процессорного времени программы используется функция `GetProcessTimes()` т.к. программа тестируется на компьютере с установленной ОС Windows. [4]

Кроме этого, необходимо отключить оптимизации компилятора для более честного сравнения алгоритмов. В моём случае это делается с помощью ключа `-O0` т.к. используется компилятор `G++`. [5]

3.2 Реализация алгоритмов

На листингах 3.1–3.3 представлены реализации алгоритмов умножения матриц.

Листинг 3.1 – Классический алгоритм

```

1 int standartMult(mtrx &rez, mtrx mtrx1, int n1, int m1, mtrx mtrx2, int n2, int m2)
2 {
3     if (m1 != n2)
4         return SIZE_ERROR;
5
6     for (int i = 0; i < n1; i++)
7         for (int j = 0; j < m2; j++)
8         {
9             rez[i][j] = 0;
10            for (int k = 0; k < n2; k++)
11                rez[i][j] = rez[i][j] + mtrx1[i][k] * mtrx2[k][j];
12        }
13    return 0;
14 }

```

Листинг 3.2 – Алгоритм Винограда

```

1 int vinograd(mtrx &rez, mtrx mtrx1, int n1, int m1, mtrx mtrx2, int n2, int m2)
2 {
3     if (m1 != n2)
4         return SIZE_ERROR;
5     vector<int> mulH(n1, 0);
6     for (int i = 0; i < n1; i++)
7         for (int j = 0; j < m1 / 2; j++)
8             mulH[i] = mulH[i] + mtrx1[i][j * 2] * mtrx1[i][j * 2 + 1];
9
10    vector<int> mulV(n1, 0);
11    for (int i = 0; i < m2; i++)
12        for (int j = 0; j < n2 / 2; j++)
13            mulV[i] = mulV[i] + mtrx2[j * 2][i] * mtrx2[j * 2 + 1][i];
14
15    for (int i = 0; i < n1; i++)
16        for (int j = 0; j < m2; j++)
17        {
18            rez[i][j] = -mulH[i] - mulV[j];
19            for (int k = 0; k < n2 / 2; k++)
20                rez[i][j] = rez[i][j] + (mtrx1[i][2 * k + 1] + mtrx2[2 * k][j]) *
21                    (mtrx1[i][2 * k] + mtrx2[2 * k + 1][j]);
22        }
23
24    if (n2 % 2 == 1)
25        for (int i = 0; i < n1; i++)
26            for (int j = 0; j < m2; j++)
27                rez[i][j] = rez[i][j] + mtrx1[i][n2 - 1] * mtrx2[n2 - 1][j];
28    return 0;
29 }

```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
1 int optimizedVinograd(mtrx &rez, mtrx mtrx1, int n1, int m1, mtrx mtrrx2, int n2, int m2)
2 {
3     if (m1 != n2)
4         return SIZE_ERROR;
5
6     vector<int> mulH(n1, 0);
7     for (int i = 0; i < n1; i++)
8         for (int j = 0; j < m1 - 1; j += 2)
9             mulH[i] -= mtrx1[i][j] * mtrx1[i][j + 1];
10
11     vector<int> mulV(n1, 0);
12     for (int i = 0; i < m2; i++)
13         for (int j = 0; j < n2 - 1; j += 2)
14             mulV[i] -= mtrrx2[j][i] * mtrrx2[j + 1][i];
15
16     bool flag = false;
17     if (n2 % 2 == 1)
18         flag = true;
19
20     for (int i = 0; i < n1; i++)
21         for (int j = 0; j < m2; j++)
22         {
23             rez[i][j] = mulH[i] + mulV[j];
24             for (int k = 0; k < n2 - 1; k += 2)
25                 rez[i][j] += (mtrx1[i][k + 1] + mtrrx2[k][j]) * (mtrx1[i][k] + mtrrx2[k +
26                     1][j]);
27
28             if (flag)
29                 rez[i][j] += mtrx1[i][n2 - 1] * mtrrx2[n2 - 1][j];
30         }
31
32     return 0;
33 }
```

3.3 Тестирование

В таблице 3.1 приведены функциональные тесты для алгоритмов умножения матриц. Тестирование проводилось методом чёрного ящика. Все тесты пройдены успешно для всех алгоритмов.

Таблица 3.1 – Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый рез.	Фактический рез.
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 4 \\ 0 & 4 & 4 \\ 3 & 3 & 2 \end{pmatrix}$	$\begin{pmatrix} 4 & 0 & 0 \\ 1 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 10 & 4 & 10 \\ 8 & 8 & 12 \\ 17 & 6 & 7 \end{pmatrix}$	$\begin{pmatrix} 10 & 4 & 10 \\ 8 & 8 & 12 \\ 17 & 6 & 7 \end{pmatrix}$

На рисунке 3.1 приведены результаты тестирования.

```
Matrix 2:
4 0 0
1 2 1
1 0 2

Expected result:
10 4 10
8 8 12
17 6 7

standartMult Result Matrix:
10 4 10
8 8 12
17 6 7

vinograd Result Matrix:
10 4 10
8 8 12
17 6 7

optimizedVinograd Result Matrix:
10 4 10
8 8 12
17 6 7

6/6 positive tests
```

Рисунок 3.1 – Результаты функционального тестирования

Как видно по рисунку, функциональные тесты пройдены.

3.4 Вывод

В данном разделе были выбраны инструменты для реализации алгоритмов, представлены листинги их реализации, а также проведено функциональное тестирование.

4 Исследовательский раздел

В данном разделе представлены примеры работы программы, сравнительный анализ реализованных алгоритмов и оценка их трудоёмкости.

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

```
Matrix 1:
1 2 4
0 4 4
3 3 2

Matrix 2:
4 0 0
1 2 1
1 0 2

standartMult Result Matrix:
10 4 10
8 8 12
17 6 7

vinograd Result Matrix:
10 4 10
8 8 12
17 6 7

optimizedVinograd Result Matrix:
10 4 10
8 8 12
17 6 7
```

Рисунок 4.1 – Пример работы программы

4.2 Сравнительный анализ времени выполнения алгоритмов

Чтобы провести сравнительный анализ времени выполнения алгоритмов замерялось процессорное время для квадратных матриц следующих размеров 100x100, 200x200, ... 1000x1000 и 101x101, 201x201, ... 1001x1001. Чтобы оценить время выполнения умножения матриц, они заполнялись числами от 0 до 999, замерялось процессорное время для части кода, ко-

торая умножала матрицы 10 раз, после чего результат делился на кол-во итераций.

Сравнительный анализ проводился на компьютере с процессором Intel Core i7-7700K и установленной операционной системой Windows 10.

Для алгоритма винограда наихудшим случаем является умножение матриц с нечётными размерами. Наилучшим случаем матрицы с чётными размерами. На рисунках 4.2 и 4.3 приведены зависимости времени работы алгоритмов от нечетных и четных размеров матриц.

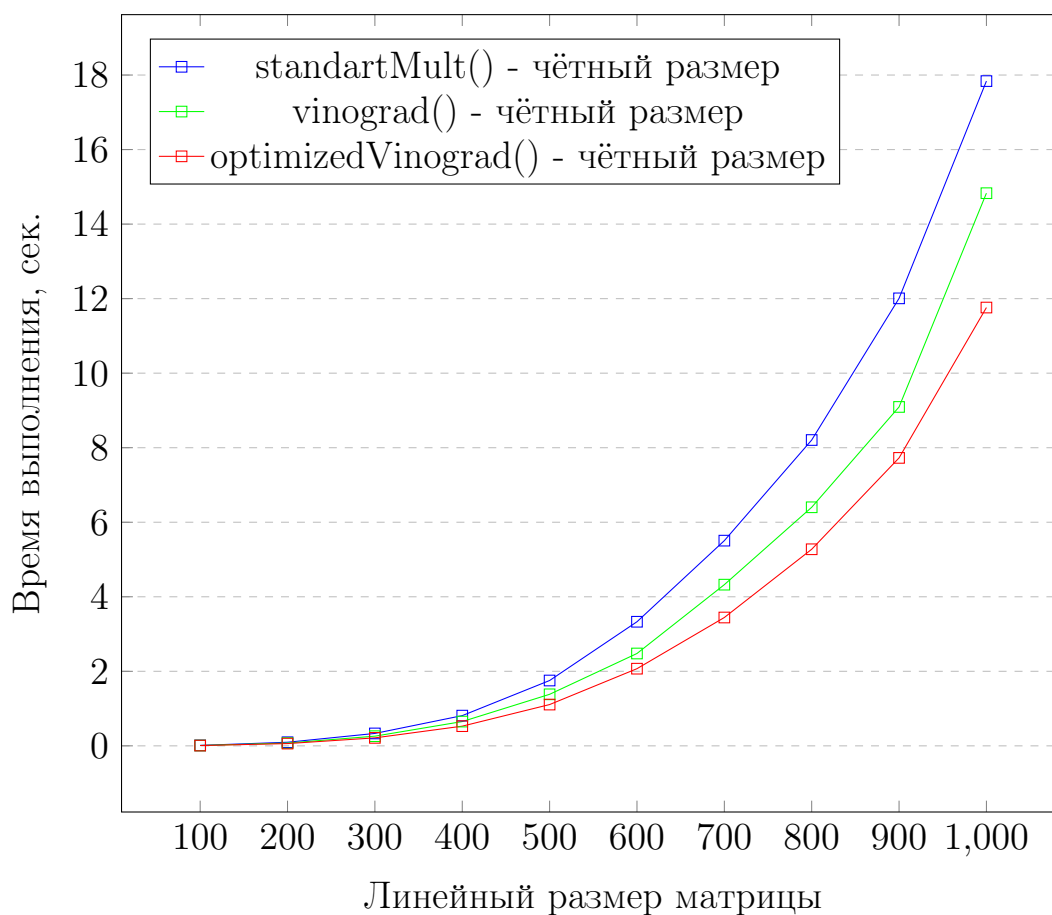


Рисунок 4.2 – Зависимость времени выполнения умножения матриц от их размеров в случае чётных размеров

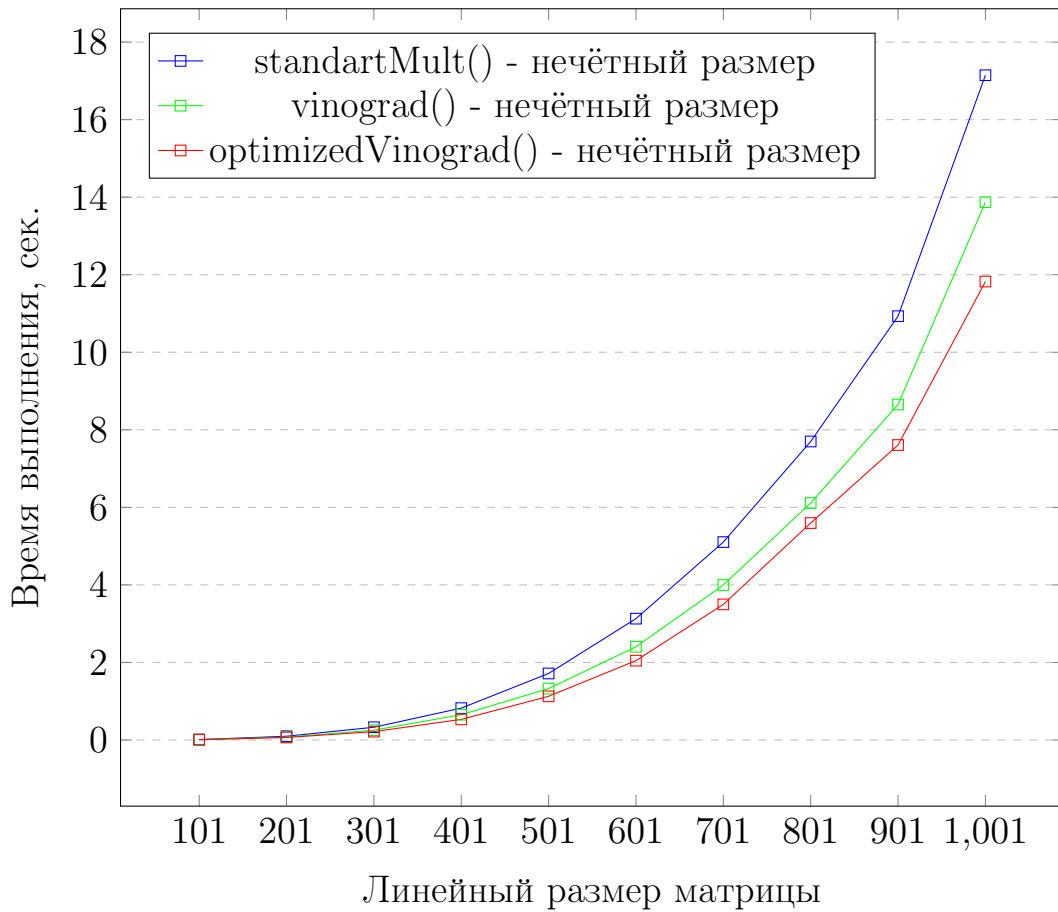


Рисунок 4.3 – Зависимость времени выполнения умножения матриц от их размеров в случае нечётных размеров

4.3 Оценка трудоёмкости

Для оценки трудоёмкости использовалась следующая модель вычислений: [6]

- Трудоёмкость следующих операций единична: $+$, $-$, $=$, $+=$, $-=$, $==$, $!=$, $<$, $>$, $-=$, $>=$, \ll , \gg , \llbracket ;
- Трудоёмкость следующих операций = 2: $*$, $/$, $\%$, $/=$, $*=$.

Трудоёмкость выбранных алгоритмов сортировок рассчитывалась по написанному коду.

На листинге 4.1 представлена программа для вычисления трудоёмкости классического алгоритма умножения матриц.

Листинг 4.1 – Вычисление трудоёмкости классического алгоритма умножения матриц

```

1 int standartMultCalc(mtrx &rez, mtrx mtrx1, int n1, int m1, mtrx mtrx2, int n2, int m2)
2 {
3     int t = 1; // check if !=
4     if (m1 != n2)
5         return SIZE_ERROR;
6
7     t += 1 + 1 + 1; // for i prep
8     for (int i = 0; i < n1; i++)
9     {
10         t += 1 + 1 + 1; // for j prep
11         for (int j = 0; j < m2; j++)
12         {
13             t += 1 + 1 + 1; // [] [] =
14             rez[i][j] = 0;
15             t += 1 + 1 + 1; // for k prep
16             for (int k = 0; k < n2; k++)
17             {
18                 t += 2 + 1 + 2 + 1 + 2 + 2 + 2; // [] [] = [] [] + [] [] * [] []
19                 rez[i][j] = rez[i][j] + mtrx1[i][k] * mtrx2[k][j];
20                 t += 2; //inc + check
21             }
22             t += 2; //inc + check
23         }
24         t += 2; //inc + check
25     }
26     return t;
27 }

```

Соответственно получается следующая формула трудоёмкости:

$F_{standartMult} = 3 + 4N + 7NM + 14NMK$, где N - кол-во строк первой матрицы, M - кол-во столбцов второй матрицы, K - кол-во столбцов первой матрицы

На листингах 4.3-?? представлена программа для вычисления трудоёмкости алгоритма Винограда.

Листинг 4.2 – Вычисление трудоёмкости алгоритма Винограда, часть 1

```

1 int vinogradCalc(mtrx &rez, mtrx mtrx1, int n1, int m1, mtrx mtrx2, int n2, int m2)
2 {
3     int t = 1; // check if !=
4     if (m1 != n2)
5         return SIZE_ERROR;
6
7     vector<int> mulH(n1, 0);
8     t += 2;
9     for (int i = 0; i < n1; i++)
10    {
11        t += 4;
12        for (int j = 0; j < m1 / 2; j++)
13        {
14            t += 15; // [] = [] + [][*] * [][*+]
15            mulH[i] = mulH[i] + mtrx1[i][j * 2] * mtrx1[i][j * 2 + 1];
16            t += 4;
17        }
18        t += 2;
19    }
20
21    vector<int> mulV(n1, 0);
22    t += 2;
23    for (int i = 0; i < m2; i++)
24    {
25        t += 4;
26        for (int j = 0; j < n2 / 2; j++)
27        {
28            t += 15; // [] = [] + [*] [] * [*+] []
29            mulV[i] = mulV[i] + mtrx2[j * 2][i] * mtrx2[j * 2 + 1][i];
30            t += 4;
31        }
32        t += 2;
33    }

```

Листинг 4.3 – Вычисление трудоёмкости алгоритма Винограда, часть 2

```

1  t += 2;
2  for (int i = 0; i < n1; i++)
3  {
4      t += 2;
5      for (int j = 0; j < m2; j++)
6      {
7          t += 6; // [][] = [] - []
8          rez[i][j] = -mulH[i] - mulV[j];
9          t += 4;
10         for (int k = 0; k < n2 / 2; k++)
11         {
12             t += 6 + 10 + 2 + 10; // [][] = [][] + ([][]*[] + []*[]) * ([][]*[] + []*[])
13             rez[i][j] = rez[i][j] + (mtrx1[i][2 * k + 1] + mtrx2[2 * k][j]) *
14                 (mtrx1[i][2 * k] + mtrx2[2 * k + 1][j]);
15             t += 4;
16         }
17         t += 2;
18     }
19     t += 2;
20 }
21 t += 3;
22 if (n2 % 2 == 1)
23 {
24     t += 2;
25     for (int i = 0; i < n1; i++)
26     {
27         t += 2;
28         for (int j = 0; j < m2; j++)
29         {
30             t += 14; // [][] = [][] + [][-] * [-][]
31             rez[i][j] = rez[i][j] + mtrx1[i][n2 - 1] * mtrx2[n2 - 1][j];
32             t += 2;
33         }
34         t += 2;
35     }
36 }
37 return t;
38 }

```

Следовательно формула трудоёмкости будет следующей:

$F_{vinograd} = 10 + 10N + 6M + 19N(K//2) + 19M(K//2) + 12NM + 32NM(K//2)$, при чётном K и такой:

$F_{vinograd} = 12 + 14N + 6M + 19N(K//2) + 19M(K//2) + 28NM + 32NM(K//2)$, при нечётном K.

На листингах 4.4-4.5 представлена программа для вычисления трудоёмкости оптимизированного алгоритма Винограда.

Листинг 4.4 – Вычисление трудоёмкости оптимизированного алгоритма Винограда, часть 1

```
1 int optimizedVinogradCalc(mtrx &rez, mtrx mtrx1, int n1, int m1, mtrx mtrrx2, int n2,
2   int m2)
3 {
4     int t = 1; // check if !=
5     if (m1 != n2)
6         return SIZE_ERROR;
7
8     vector<int> mulH(n1, 0);
9     t += 2;
10    for (int i = 0; i < n1; i++)
11    {
12        t += 3;
13        for (int j = 0; j < m1 - 1; j += 2)
14        {
15            t += 9; // [] -= [] [] * [] [+]
16            mulH[i] -= mtrx1[i][j] * mtrx1[i][j + 1];
17            t += 3;
18        }
19        t += 2;
20    }
21
22    vector<int> mulV(n1, 0);
23    t += 2;
24    for (int i = 0; i < m2; i++)
25    {
26        t += 3;
27        for (int j = 0; j < n2 - 1; j += 2)
28        {
29            t += 9; // [] -= [] [] * [+] []
30            mulV[i] -= mtrrx2[j][i] * mtrrx2[j + 1][i];
31            t += 3;
32        }
33        t += 2;
34    }
35
36    t += 4;
37    bool flag = false;
38    if (n2 % 2 == 1)
39    {
40        t += 1;
41        flag = true;
42    }
```

Листинг 4.5 – Вычисление трудоёмкости оптимизированного алгоритма Винограда, часть 2

```

1  t += 2;
2  for (int i = 0; i < n1; i++)
3  {
4      t += 2;
5      for (int j = 0; j < m2; j++)
6      {
7          t += 6; // [][] = [] + []
8          rez[i][j] = mulH[i] + mulV[j];
9          t += 3;
10         for (int k = 0; k < n2 - 1; k += 2)
11         {
12             t += 3+6+2+6; // [][] += ([][]+[][]) * ([][]+[+][])
13             rez[i][j] += (mtrx1[i][k + 1] + mtrrx2[k][j]) * (mtrx1[i][k] + mtrrx2[k +
14                 1][j]);
15             t += 3;
16         }
17         t += 1;
18         if (flag)
19         {
20             t += 3 + 3 + 2 + 3; // [][] += [][][-] * [-][]
21             rez[i][j] += mtrx1[i][n2 - 1] * mtrrx2[n2 - 1][j];
22         }
23         t += 2;
24     }
25     t += 2;
26 }
27
28 return t;
29 }

```

Формула трудоёмкости:

$F_{\text{OptimizedVinograd}} = 11 + 9N + 5M + 12N(K//2) + 12M(K//2) + 12NM + 20MN(K//2)$, при чётном K ,

$F_{\text{OptimizedVinograd}} = 12 + 9N + 5M + 12N(K//2) + 12M(K//2) + 23NM + 20MN(K//2)$, при нечётном K .

4.4 Вывод

Были протестированы различные алгоритмы умножения матриц. По итогу исследования выяснилось, что алгоритм Винограда начинает заметно выигрывать по времени выполнения у стандартного алгоритма умножения

матриц при линейном размере матрицы ~ 400 и больше. Оптимизированный алгоритм Винограда показывает еще более низкие показатели как и при анализе времени выполнения, так и при оценке трудоёмкости, как и ожидалось.

Заключение

По итогу проделанной работы была достигнута цель - изучены алгоритмы умножения матриц, вычислены их трудоёмкости и получены навыки в улучшении алгоритмов.

Также были решены все поставленные задачи, а именно:

- оптимизирован алгоритм Винограда;
- реализованы три алгоритма умножения матриц – классический, алгоритм Винограда и улучшенный алгоритм Винограда;
- произведена оценка времени выполнения реализованных алгоритмов;
- рассчитана трудоемкость каждого алгоритма умножения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Умножение матриц. [Электронный ресурс]. // URL: <https://ru.onlinemschool.com/math/library/matrix/multiply/>.
- [2] И. В. Белоусов. *Матрицы и определители. Учебное пособие по линейной алгебре*, pages 1–16. – Институт прикладной физики, г. Кишинёв, 2006.
- [3] F Le Gall. *Faster algorithms for rectangular matrix multiplication*, pages 514–523. – Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), 2012.
- [4] Getprocesstimes function (processthreadsapi.h) [Электронный ресурс]. // URL: <https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-getprocesstimes#syntax>.
- [5] Как применить настройки оптимизации gcc в qt? // URL: <http://blog.kislenko.net/show.php?id=1991>.
- [6] Ульянов М.В. *Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ*. Москва: ФИЗМАТЛИТ, 2008. 304 с.