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»

Отчёт по лабораторной работе №2
по дисциплине «Анализ алгоритмов»

Тема: Алгоритмы умножения матриц
Студент: Княжев А. В.
Группа: ИУ7-52Б
Оценка (баллы): _____
Преподаватели: Волкова Л. Л., Строганов Ю. В.

Москва — 2022 г.

Оглавление

Введение

Матрица — это таблица чисел [1]. Размерность матрицы задается такими характеристиками, как количество строк и столбцов таблицы. Основными математическими операциями на матрицах являются:

— умножение;

— сложения;

— вычисление определителя;

— умножение на скаляр.

Задача умножения матриц имеет важное значение для ряда научно-технических направлений, таких как:

— томография;

— компьютерная графика;

— проектирование роботизированных систем;

— классификация бинарных отношений [7].

В связи с этим, существует большое количество подходов, связанных с оптимизацией выполняемых операций.

Цель работы

Получение навыков кодирования программного продукта, тестирования и проведения замерного эксперимента работы программы на различных данных. Все это на примере решения задачи умножения матриц.

Задачи работы

1) изучение алгоритмов умножения матриц:

    — стандартный алгоритм;

- Винограда;
- оптимизированного алгоритма Винограда;

2) анализ трудоемкости данных алгоритмов;

3) кодирование данных алгоритмов;

4) проведение замерного эксперимента для данных алгоритмов, с измерением времени работы и использования памяти;

5) проведение сравнительного анализа алгоритмов на основе полученных данных.

1. Аналитическая часть

В данном разделе рассмотрены теоретические выкладки по алгоритмам умножения матриц, а также общие сведение о данной задаче.

Стоит отметить, что умножение матриц возможно только в том случае, когда количество столбцов первого операнда совпадает с количеством строк второго операнда.

1.1. Стандартный алгоритм умножения матриц

Пусть даны матрицы A $(r_A \times c_A)$ и B $(r_B \times c_B)$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1c_A} \\ a_{21} & a_{22} & \dots & a_{2c_A} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_A1} & a_{r_A2} & \dots & a_{r_Ac_A} \end{pmatrix}, \tag{1.1}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1c_B} \\ b_{21} & b_{22} & \dots & b_{2c_B} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r_B1} & b_{r_B2} & \dots & b_{r_Bc_B} \end{pmatrix}. \tag{1.2}$$

Тогда произведением матриц A и B называется матрица C $(r_C \times c_C)$:

$$r_C = r_A, \tag{1.3}$$

$$c_C = c_B, \tag{1.4}$$

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1c_C} \\ c_{21} & c_{22} & \dots & c_{2c_C} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r_C1} & c_{r_C2} & \dots & c_{r_Cc_C} \end{pmatrix}. \tag{1.5}$$

При этом, каждый элемент матрицы C может быть вычислен по формуле:

$$c_{ij} = \sum_{k=1}^{r_B} a_{ik}b_{kj}, \quad i = \overline{1, r_C}, j = \overline{1, c_C}. \tag{1.6}$$

1.2. Алгоритм Винограда

Алгоритм Винограда является одним из самых эффективных с точки зрения быстродействия алгоритмов [8]. Основной идеей алгоритма является сокращение количества трудоемких операций путем предварительного расчета.

Пусть даны два вектора $D = (d_1, d_2, d_3, d_4)$ и $E = (e_1, e_2, e_3, e_4)$.

Тогда:

$$D \cdot E = d_1 \cdot e_1 + d_2 \cdot e_2 + d_3 \cdot e_3 + d_4 \cdot e_4. \tag{1.7}$$

Путем алгебраических преобразований можно привести формулу 1.7 к следующему виду:

$$D \cdot E = (d_1 + e_2) \cdot (d_2 + e_1) + (d_3 + e_4) \cdot (d_4 + e_3) - d_1 \cdot d_2 - d_3 \cdot d_4 - e_1 \cdot e_2 - e_3 \cdot e_4. \tag{1.8}$$

Несмотря на большую сложность вычисления формулы 1.8, по сравнению с 1.7, преобразованный вариант допускает возможность предварительного расчета слагаемых $d_1 \cdot d_2$, $d_3 \cdot d_4$, $e_1 \cdot e_2$ и $e_3 \cdot e_4$. С учетом предварительно рассчитанных значений, для вычисления формулы 1.8 необходимо совершить 7 операций сложения и 2 операции умножения, против 3 операция сложения и 4 операций умножения в формуле 1.7. В связи с тем, что операция умножения является более трудоемкой [9], вариант с предварительным расчетом является менее тредоемким с точки зрения вычисления.

1.3. Оптимизированный алгоритм Винограда

В данной работе рассмотрены следующий улучшения алгоритма Винограда:

— замена операции $v = v + n$ на $v+ = n$;

— замена операции $v * 2$ на $v << 1$;

— замена операции $v/2$ на $v >> 1$;

— предварительный расчет некоторых элементов, например половинной длины массива, используемой в цикле;

— объединение побочного цикла для крайнего случая внутрь основного цикла;

— замена счетчиков цикла для уменьшения количества выполняемых арифметических операций.

2. Конструкторская часть

В данном разделе представлены схемы алгоритмов умножения матриц и произведена оценка трудоемкости алгоритмов умножения матриц.

2.1. Разработка стандартного алгоритма умножения матриц

На рисунке 2.1 представлена схема стандартного алгоритма умножения матриц.

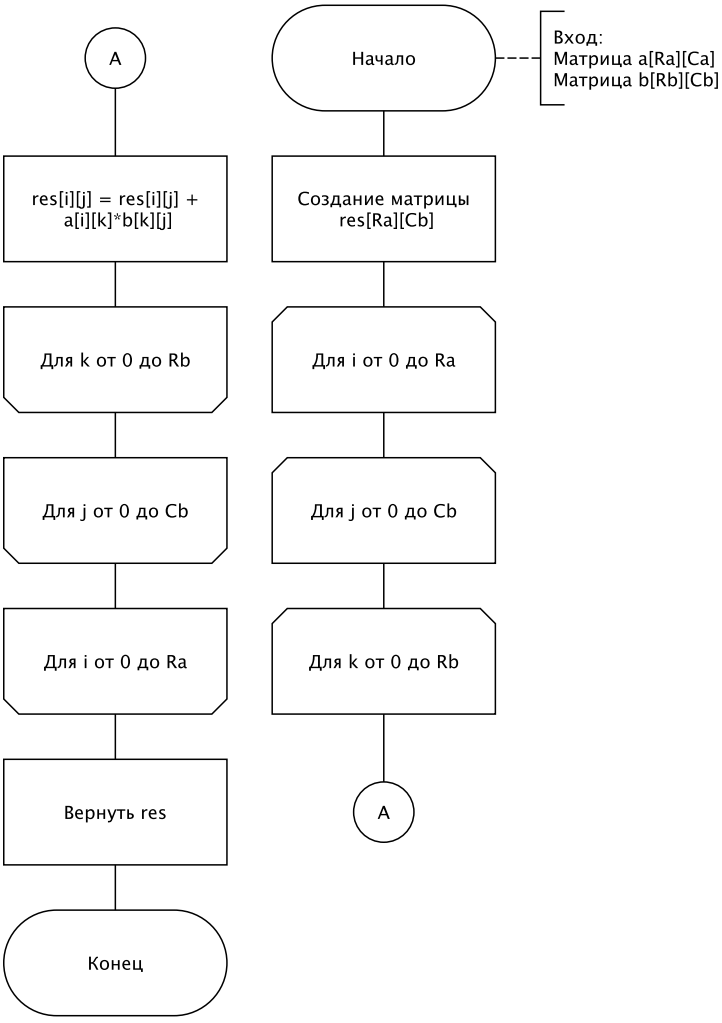

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

2.2. Разработка алгоритма Винограда

На рисунках 2.2 – 2.3 представлена схема алгоритма Винограда для умножения матриц.

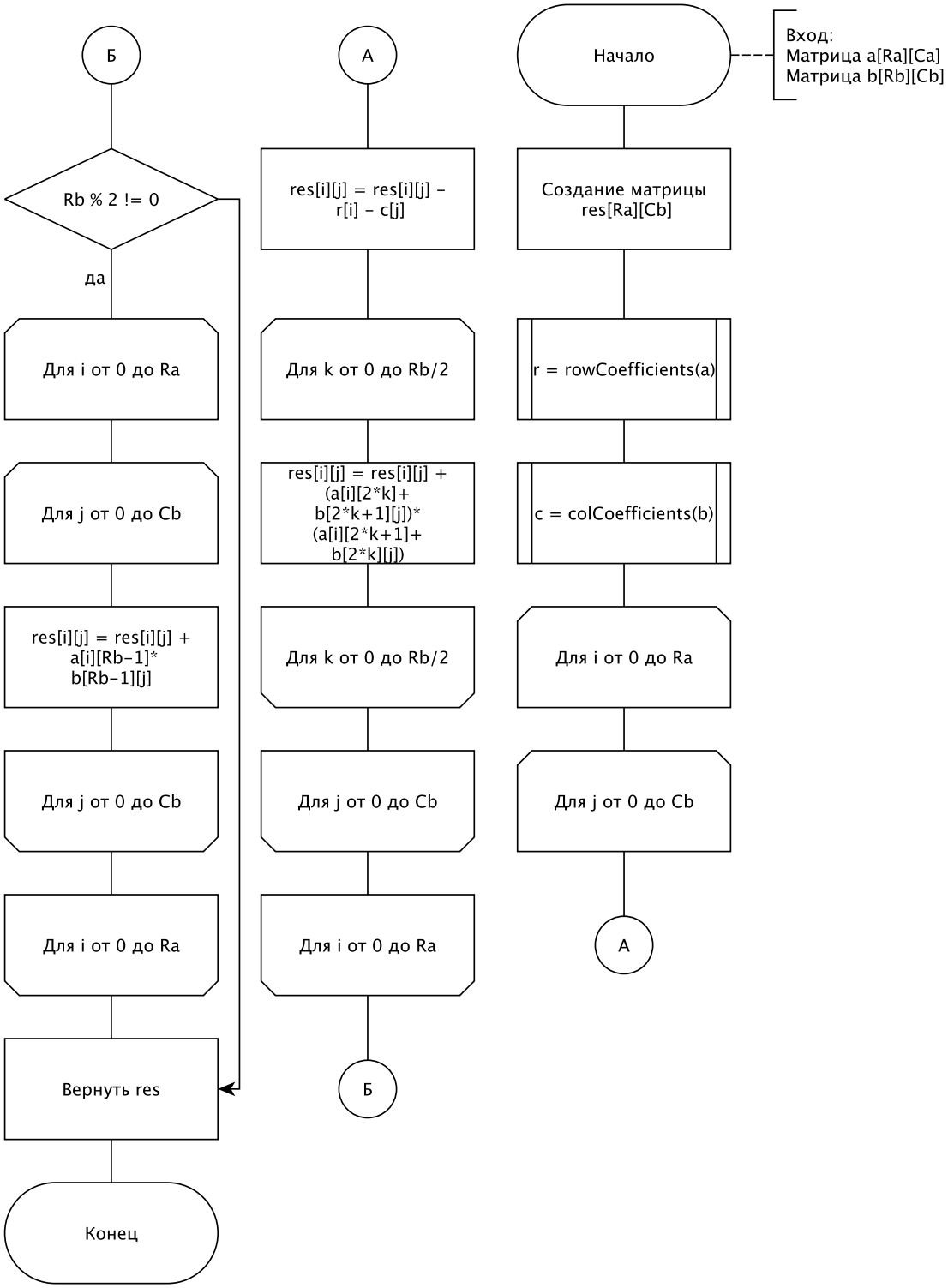

Рисунок 2.2 — Схема алгоритма Винограда для умножения матриц

Рисунок 2.3 — Схема алгоритма Винограда для умножения матриц (функции нахождения произведений соседних элементов строк и столбцов)

2.3. Разработка оптимизированного алгоритма Винограда

На рисунках 2.4 – 2.5 представлена схема оптимизированного алгоритма Винограда.

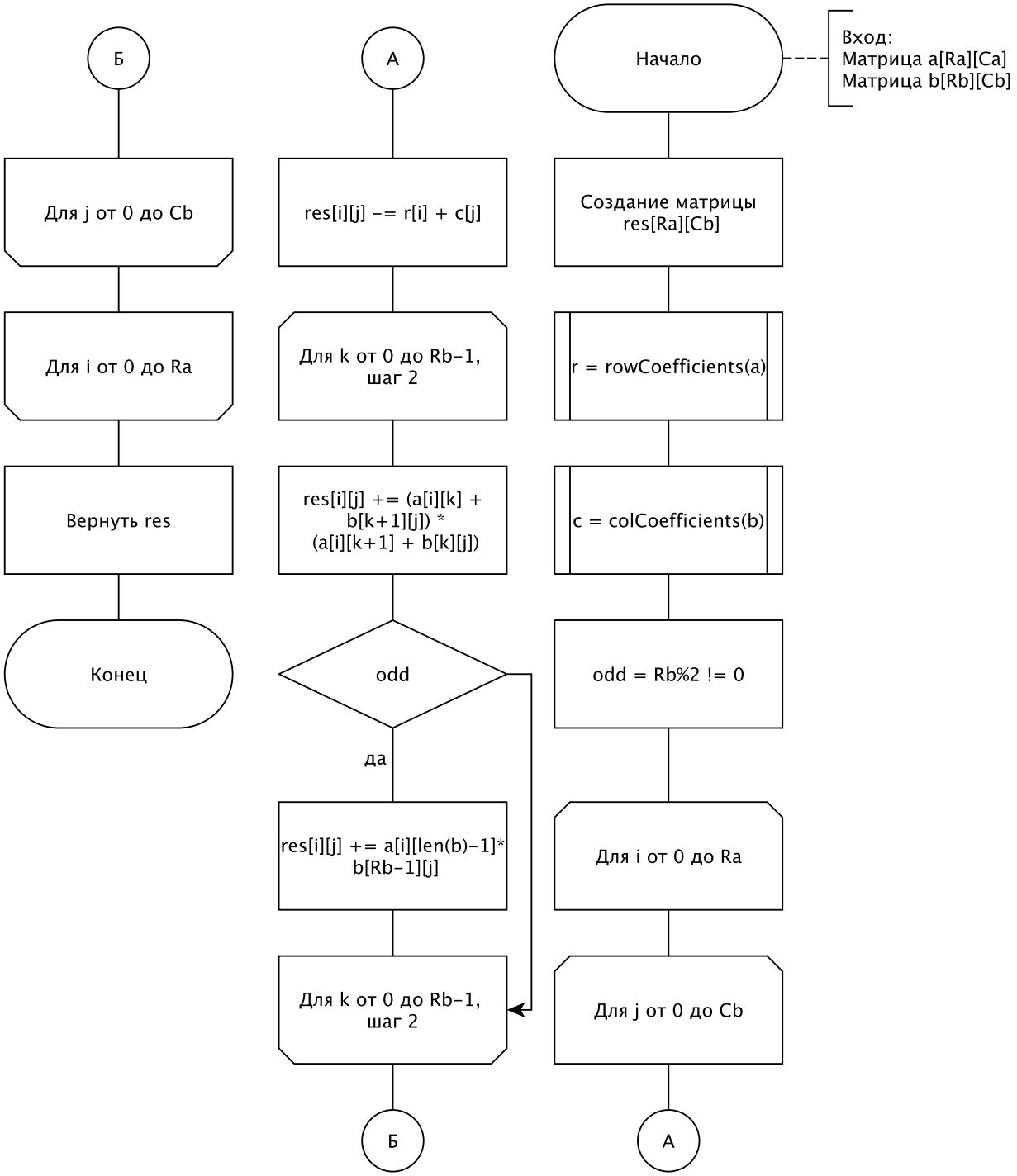

Рисунок 2.4 — Схема оптимизированного алгоритма Винограда

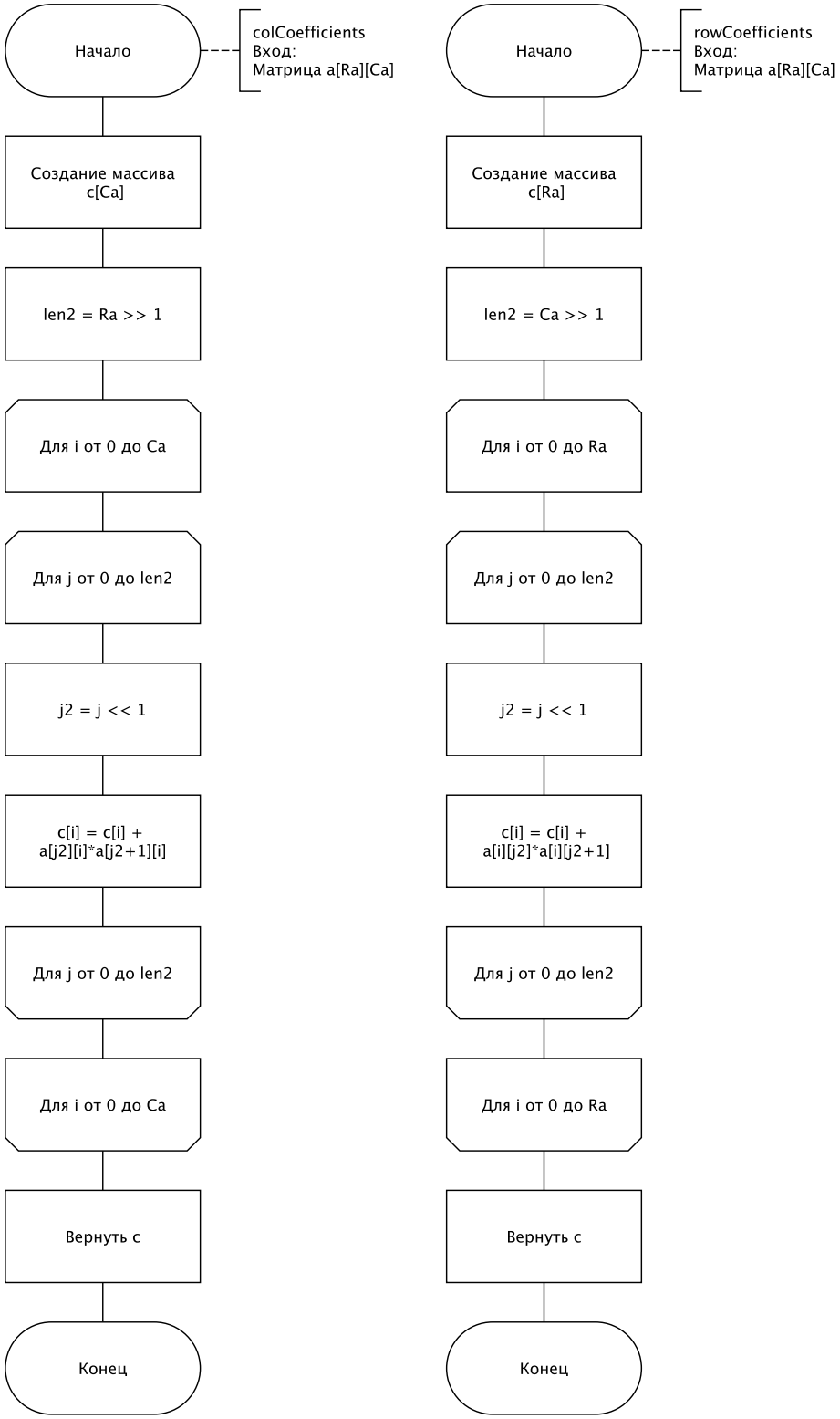

Рисунок 2.5 — Схема алгоритма Винограда для умножения матриц (функции нахождения произведений соседних элементов строк и столбцов)

2.4. Оценка трудоемкости алгоритмов умножения матриц

2.4.1. Модель вычислений

Для вычисления трудоемкости исследуемых алгоритмов необходимо ввести модель вычислений.

Если обозначить трудоемкость некоторой операции a, как f_a, то можно ввести таблицу соответствия значения трудоемкости для базовых операций:

Таблица 2.1 — Таблица значений тредоемкости

Операция	Трудоемкость
:=	1
+ =	1
− =	1
+	1
−	1
<<	1
>>	1
[]	1
++	1
−−	1
>=	1
<=	1
==	1
! =	1
<	1
>	1
*	2
/	2
%	2
вызов функции	0

Трудоемкость условного блока можно ввести следующим образом:

$$f_{if} = f_{cond} + \begin{cases} min(f_{in\_if}, f_{in\_else}) & \text{в лучшем случае,} \\ max(f_{in\_if}, f_{in\_else}) & \text{в худшем случае,} \end{cases} \tag{2.1}$$

где

— f_{if} — трудоемкость условного блока;

— f_{cond} — трудоемкость вычисления условия;

— f_{in_if} — трудоемкость фрагмента после if;

— f_{in_else} — трудоемкость фрагмента после $else$.

Трудоемкость цикла можно ввести следующим образом:

$$f_{loop} = f_{init} + f_{cmp} + n \cdot (f_{body} + f_{cmp} + f_{inc}), \tag{2.2}$$

где

— f_{loop} — трудоемкость цикла;

— f_{init} — трудоемкость инициализирующего выражения;

— f_{cmp} — трудоемкость сравнения цикла;

— f_{body} — трудоемкость тела цикла;

— f_{inc} — трудоемкость инкремента.

2.4.2. Трудоемкость стандартного алгоритма умножения матриц

Здесь и далее будут использоваться следующие обозначения:

— N — количество строк первой матрицы;

— M — количество столбцов первой матрицы (количество строк второй матрицы);

— K — количество столбцов второй матрицы.

Трудоемкость внутреннего цикла по k равна

$$f_1 = 2 + M \cdot (2 + 12) = 2 + M \cdot 14. \tag{2.3}$$

Трудоемкость внутреннего цикла по j равна

$$f_2 = 2 + K \cdot (2 + f_1). \tag{2.4}$$

Трудоемкость алгоритма равна трудоемкости внешнего цикла по i

$$f = 2 + N \cdot (2 + f_2) = 2 + N \cdot (2 + 2 + K \cdot (2 + f_1)), \tag{2.5}$$

$$f = 2 + N \cdot (2 + 2 + K \cdot (2 + 2 + M \cdot 14)), \tag{2.6}$$

$$f = 2 + N \cdot 4 + N \cdot K \cdot 4 + N \cdot K \cdot M \cdot 14 \approx 14 \cdot N \cdot M \cdot K. \tag{2.7}$$

2.4.3. Трудоемкость алгоритма Винограда

Для алгоритма Винограда худшим случаем будет ситуация, в которой количество строк во второй матрице нечетное. Если оно четное, то это лучший случай.

Трудоемкость создания массива сумм произведений соседних элементов строк равна

$$f_1 = 2 + N \cdot (2 + 2 + \frac{M}{2} \cdot 19) = 2 + N \cdot 4 + N \cdot M \cdot 9.5. \tag{2.8}$$

Трудоемкость создания массива сумм произведений соседних элементов столбцов равна

$$f_2 = 2 + K \cdot (2 + 2 + \frac{M}{2} \cdot 19) = 2 + K \cdot 4 + K \cdot M \cdot 9.5. \tag{2.9}$$

Трудоемкость внутреннего цикла по k равна

$$f_3 = 2 + \frac{M}{2} \cdot (2 + 26) = 2 + M \cdot 14. \tag{2.10}$$

Трудоемкость внутреннего цикла по j равна

$$f_4 = 2 + K \cdot (2 + f_3 + 8) = 2 + K \cdot 12 + K \cdot M \cdot 14. \tag{2.11}$$

Трудоемкость внешнего цикла по i равна

$$f_5 = 2 + N \cdot (2 + f_4) = 2 + N \cdot 2 + N \cdot K \cdot 12 + N \cdot K \cdot M \cdot 14. \tag{2.12}$$

Трудоемкость условия после основных циклов равна

$$f_6 = 3 + \begin{cases} 0, & \text{лучший случай,} \\ 2 + N \cdot 2 + N \cdot K \cdot 13, & \text{худший случай.} \end{cases} \tag{2.13}$$

Тогда, трудоемкость алгоритма Винограда равна

$$f = f_1 + f_2 + f_5 + f_6, \tag{2.14}$$

$$f \approx \begin{cases} N \cdot K \cdot M \cdot 14, & \text{лучший случай,} \\ N \cdot K \cdot M \cdot 14, & \text{худший случай.} \end{cases} \tag{2.15}$$

2.4.4. Трудоемкость оптимизированного алгоритма Винограда

Трудоемкость создания массива сумм произведений соседних элементов строк равна

$$f_1 = 2 + N \cdot (2 + 2 + \frac{M}{2} \cdot 12) = 2 + N \cdot 4 + N \cdot M \cdot 6. \tag{2.16}$$

Трудоемкость создания массива сумм произведений соседних элементов столбцов равна

$$f_2 = 2 + K \cdot (2 + 2 + \frac{M}{2} \cdot 12) = 2 + K \cdot 4 + K \cdot M \cdot 6. \tag{2.17}$$

Трудоемкость внутреннего цикла по k равна

$$f_3 = \begin{cases} 4 + M \cdot 8.5, & \text{лучший случай,} \\ 4 + M \cdot 14, & \text{худший случай.} \end{cases} \tag{2.18}$$

Трудоемкость внутреннего цикла по j равна

$$f_4 = 2 + K \cdot 8 + K \cdot f_3) = \begin{cases} 2 + K \cdot 12 + K \cdot M \cdot 8.5, & \text{лучший случай,} \\ 2 + K \cdot 12 + K \cdot M \cdot 14, & \text{худший случай.} \end{cases}$$

(2.19)

Трудоемкость внешнего цикла по i равна

$$f_5 = 2 + N \cdot 2 + N \cdot f_4 = \begin{cases} 2 + N \cdot 4 + N \cdot K \cdot 12 + N \cdot K \cdot M \cdot 8.5, & \text{лучший случай,} \\ 2 + N \cdot 4 + N \cdot K \cdot 12 + N \cdot K \cdot M \cdot 14, & \text{худший случай.} \end{cases} \tag{2.20}$$

Трудоемкость алгоритма равна

$$f = f_1 + f_2 + 4 + f_5 \approx \begin{cases} N \cdot M \cdot K \cdot 8.5, & \text{лучший случай,} \\ N \cdot M \cdot K \cdot 14, & \text{худший случай.} \end{cases} \tag{2.21}$$

3. Технологическая часть

В данном разделе представлены реализации алгоритмов умножения матриц. Кроме того, указаны требования к ПО и средства реализации.

3.1. Требования к ПО

— программа позволяет вводить имена файлов, содержащих информацию о матрицах, с помощью аргументов командной строки;

— программа аварийно завершается в случае ошибок, выводя сообщение о соответствующей ошибке;

— программа выполняет замеры времени работы реализаций алгоритмов;

— программа строит зависимости времени работы реализаций алгоритмов от размеров входных данных;

— программа принимает на вход непустые матрицы, состоящие из целых чисел.

3.2. Средства реализации

Для реализации данной работы выбран язык программирования Go, так как он содержит необходимые для тестирования библиотеки, а также обладает достаточными инструментами для реализации ПО, удовлетворяющего требованиям данной работы [3].

3.3. Реализации алгоритмов

В листингах 3.1 – 3.7 представлены реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна.

Листинг 3.1 — Исходный код реализации стандартного алгоритма умножения матриц

```
func (m *Usual) Multiply(a, b [][]int) ([][]int, error) {
        if len(a[0]) != len(b) {
                return nil, algs.ErrInvalidArgsSize
        }

        res := make([][]int, len(a))
        for i := range res {
                res[i] = make([]int, len(b[0]))
        }

        for i := range res {
                for j := range res[0] {
                        for k := range b {
                                res[i][j] = res[i][j] + a[i][k]*b[k][j]
                        }
                }
        }

        return res, nil
}
```

Листинг 3.2 — Исходный код реализации алгоритма Винограда

```
func (m *Winograd) Multiply(a, b [][]int) ([][]int, error) {
        if len(a[0]) != len(b) {
                return nil, algs.ErrInvalidArgsSize
        }
        res := make([][]int, len(a))
        for i := range res {
                res[i] = make([]int, len(b[0]))
        }
        r := m.rowCoefficients(a)
        c := m.colCoefficients(b)
        for i := 0; i < len(res); i++ {
                for j := 0; j < len(res[0]); j++ {
                        res[i][j] = res[i][j] - r[i] - c[j]
                        for k := 0; k < len(a[0])/2; k++ {
                                res[i][j] = res[i][j] +
                                (a[i][2*k]+b[2*k+1][j])*
                                (a[i][2*k+1]+b[2*k][j])
                        }
                }
        }
        if len(a[0])%2 != 0 {
                for i := 0; i < len(res); i++ {
                        for j := 0; j < len(res[0]); j++ {
                                res[i][j] = res[i][j] +
                                a[i][len(a[0])-1]*b[len(a[0])-1][j]
                        }
                }
        }
        return res, nil
}
```

Листинг 3.3 — Исходный код функции нахождения сумм произведений соседних элементов в строках

```
func (m *Winograd) rowCoefficients(a [][]int) []int {
        c := make([]int, len(a))
        for i := 0; i < len(a); i++ {
                for j := 0; j < len(a[0])/2; j++ {
                        c[i] = c[i] + a[i][2*j]*a[i][2*j+1]
                }
        }
        return c
}
```

Листинг 3.4 — Исходный код функции нахождения сумм произведений соседних элементов в столбцах

```
func (m *Winograd) colCoefficients(a [][]int) []int {
        c := make([]int, len(a[0]))
        for i := 0; i < len(a[0]); i++ {
                for j := 0; j < len(a)/2; j++ {
                        c[i] = c[i] + a[2*j][i]*a[2*j+1][i]
                }
        }
        return c
}
```

Листинг 3.5 — Исходный код реализации оптимизированного алгоритма Винограда

```
func (m *WinogradImproved) Multiply(a, b [][]int) ([][]int, error) {
        if len(a[0]) != len(b) {
                return nil, algs.ErrInvalidArgsSize
        }
        res := make([][]int, len(a))
        for i := range res {
                res[i] = make([]int, len(b[0]))
        }
        r := m.rowCoefficients(a)
        c := m.colCoefficients(b)
        odd := len(a[0])%2 != 0
        len2 := len(res[0])
        for i := 0; i < len(res); i++ {
                for j := 0; j < len2; j++ {
                        res[i][j] -= r[i] + c[j]
                        for k := 0; k < len(a[0])-1; k += 2 {
                                res[i][j] += (a[i][k]+b[k+1][j])*
                                (a[i][k+1]+b[k][j])
                        }
                        if odd {
                                res[i][j] += a[i][len(a[0])-1] *
                                 b[len(a[0])-1][j]
                        }
                }
        }
        return res, nil
}
```

Листинг 3.6 — Исходный код оптимизированной функции нахождения сумм произведений соседних элементов в строках

```
unc (m *WinogradImproved) rowCoefficients(a [][]int) []int {
        c := make([]int, len(a))
        len2 := len(a[0]) >> 1
        for i := 0; i < len(a); i++ {
                for j := 0; j < len2; j++ {
                        j2 := j << 1
                        c[i] += a[i][j2] * a[i][j2+1]
                }
        }
        return c
}
```

Листинг 3.7 — сходный код оптимизированной функции нахождения сумм произведений соседних элементов в столбцах

```
func (m *WinogradImproved) colCoefficients(a [][]int) []int {
        c := make([]int, len(a[0]))
        len1 := len(a[0])
        len2 := len(a) >> 1
        for i := 0; i < len1; i++ {
                for j := 0; j < len2; j++ {
                        j2 := j << 1
                        c[i] += a[j2][i] * a[j2+1][i]
                }
        }
        return c
}
```

3.4. Тестирование

Тестирование проводилось по методологии чёрного ящика. **Тесты пройдены успешно.**

В таблице представлены тестовые данные для реализаций алгоритмов умножения матриц.

Таблица 3.1 — Тестовые данные для алгоритмов умножения матриц

№	A	B	C
1	$\begin{pmatrix}1\end{pmatrix}$	$\begin{pmatrix}1\end{pmatrix}$	$\begin{pmatrix}1\end{pmatrix}$
2	$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}7 & 10\\15 & 22\end{pmatrix}$
3	$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}1 & 0\\0 & 1\end{pmatrix}$	$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$
4	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\end{pmatrix}$	$\begin{pmatrix}1 & 2\\3 & 4\\5 & 6\end{pmatrix}$	$\begin{pmatrix}38 & 32\\101 & 86\end{pmatrix}$
5	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}10\\10\\10\end{pmatrix}$	$\begin{pmatrix}60\\150\\240\end{pmatrix}$
6	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}10\end{pmatrix}$	ошибка
7	$\begin{pmatrix}1 & \text{ап} & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	$\begin{pmatrix}10\end{pmatrix}$	ошибка

4. Экспериментальная часть

В данном разделе описаны замерные эксперименты и представлены результаты исследования.

4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялся эксперимент [5]:

— 8 ГБ оперативной памяти;

— процессор Apple M2 (тактовая частота — до 3.5ГГц);

— операционная система macOS Ventura 13.0.

4.2. Измерение процессорного времени выполнения реализаций алгоритмов

Для измерения процессорного времени выполнения реализаций алгоритмов была использована функция языка C — $clock\_gettime$, которая позволяет получить текущее процессорное время в наносекундах [6].

4.2.1. Худший случай

В таблице 4.1 представлены результаты измерений процессорного времени выполнения в зависимости от размерности квадратной матрицы для худшего случая алгоритма Винограда. На рисунке 4.1 представлена зависимость времени выполнения от размерности квадратной матрицы.

Таблица 4.1 — Результаты замеров процессорного времени для худшего случая (в нс)

Размерность	Обычный	Виноград	Опт. Виноград
1	241	472	442
3	330	559	418
5	544	637	582
7	840	876	768
9	1291	1231	1073
13	3551	2898	2626
15	5585	4230	3891
19	11254	7824	7268
29	41766	27095	24959
39	108670	62862	60685
49	231702	123522	120280
59	427248	218014	226923

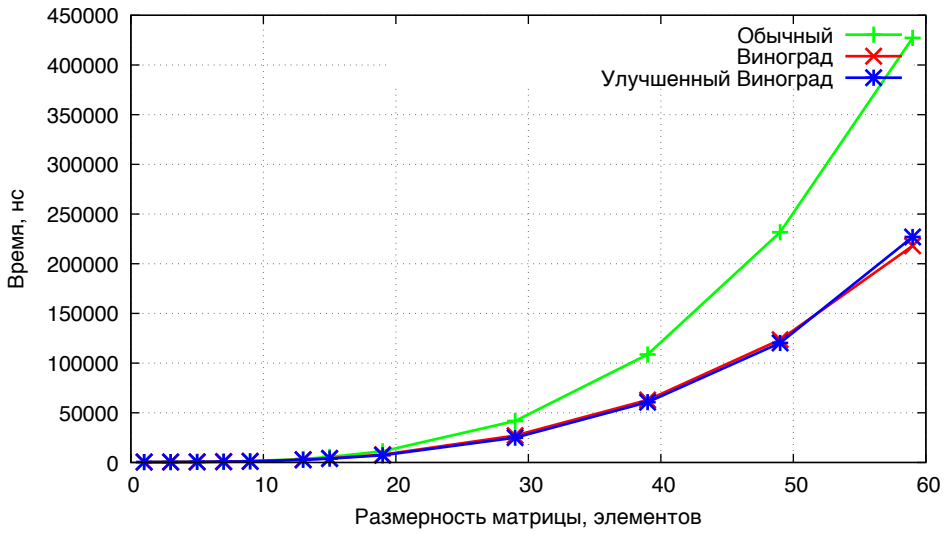

Рисунок 4.1 — Результаты замеров времени для худшего случая

Так как на этом масштабе не видна разница между реализациями алгоритма Винограда и оптимизированного алгоритма Винограда, то на рисунке 4.2 представлена зависимость времени выполнения от размерности квадратной матрицы для данных двух алгоритмов на более показательном масштабе.

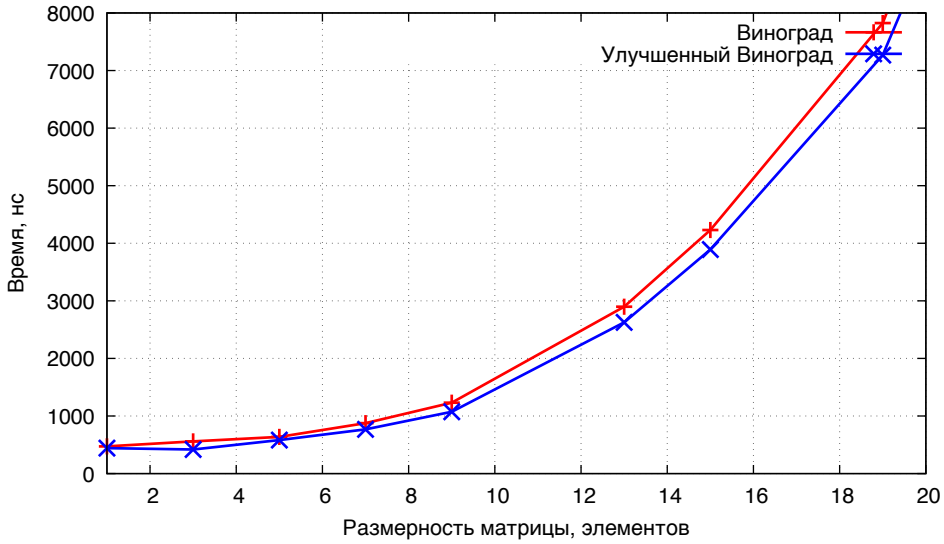

Рисунок 4.2 — Результаты замеров времени для алгоритма Винограда и его оптимизированного варианта

4.2.2. Лучший случай

В таблице 4.2 представлены результаты измерений процессорного времени выполнения в зависимости от размерности квадратной матрицы для худшего случая алгоритма Винограда. На рисунке 4.3 представлена зависимость времени выполнения от размерности квадратной матрицы.

Таблица 4.2 — Результаты замеров процессорного времени для лучшего случая (в нс)

Размерность	Обычный	Виноград	Опт. Виноград
2	389	579	494
4	548	588	501
6	739	737	680
8	1074	1014	920
10	1074	1440	1337
14	4405	3397	3113
16	6783	4890	4508
20	13310	9288	8213
30	48497	31234	33992
40	121926	67162	64612
50	255559	132789	132273
60	453096	228107	225757

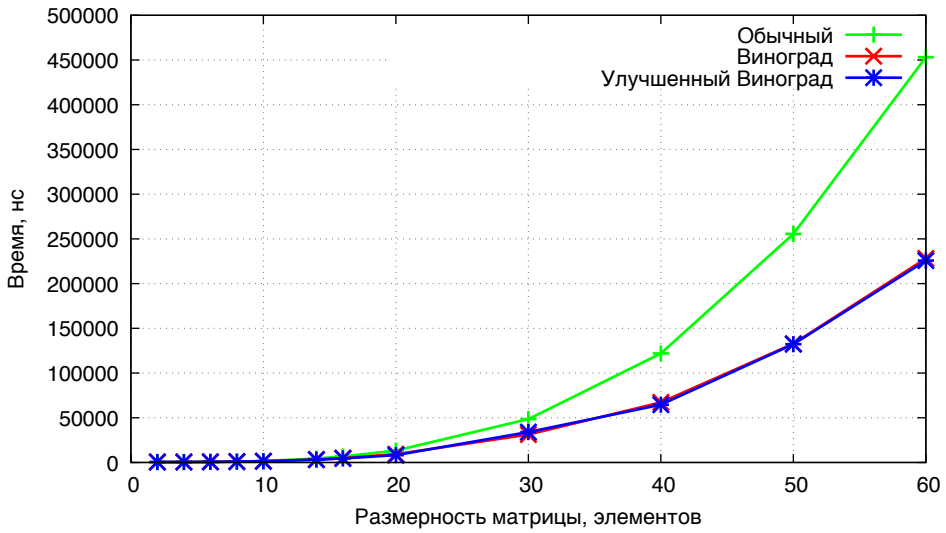

Рисунок 4.3 — Результаты замеров времени для лучшего случая

Так как на этом масштабе не видна разница между реализациями алгоритма Винограда и оптимизированного алгоритма Винограда, то на рисунке 4.4 представлена зависимость времени выполнения от размерности квадратной матрицы для данных двух алгоритмов на более показательном масштабе.

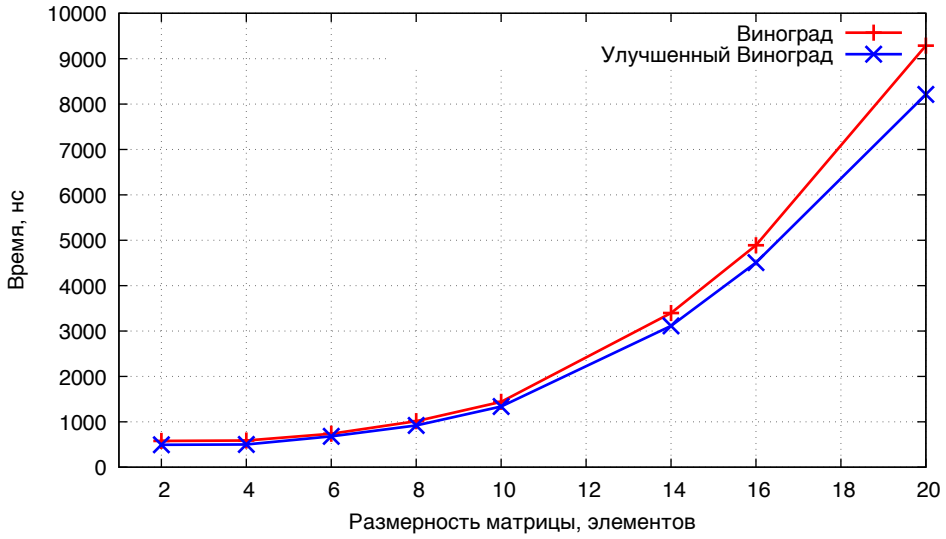

Рисунок 4.4 — Результаты замеров времени для алгоритма Винограда и его оптимизированного варианта

4.3. Измерение объёма потребляемой памяти реализаций алгоритмов

В таблице 4.3 представлены результаты измерения потребляемой памяти в зависимости от размерности квадратной матрицы. На рисунке 4.5 представлена зависимость потребляемой памяти от размерности квадратной матрицы.

Таблица 4.3 — Результаты замеров потребляемой памяти (в байтах)

Размерность	Обычный	Виноград	Опт. Виноград
1	176	336	353
2	224	392	409
3	288	464	481
4	368	552	569
5	464	656	673
6	576	776	793
7	704	912	929
8	848	1064	1081
10	1184	1416	1433
13	1808	2064	2081
16	2576	2856	2873
20	3824	4136	4153
30	8064	8456	8473
40	13904	14376	14393
50	21344	21896	21913
60	30384	31016	31033

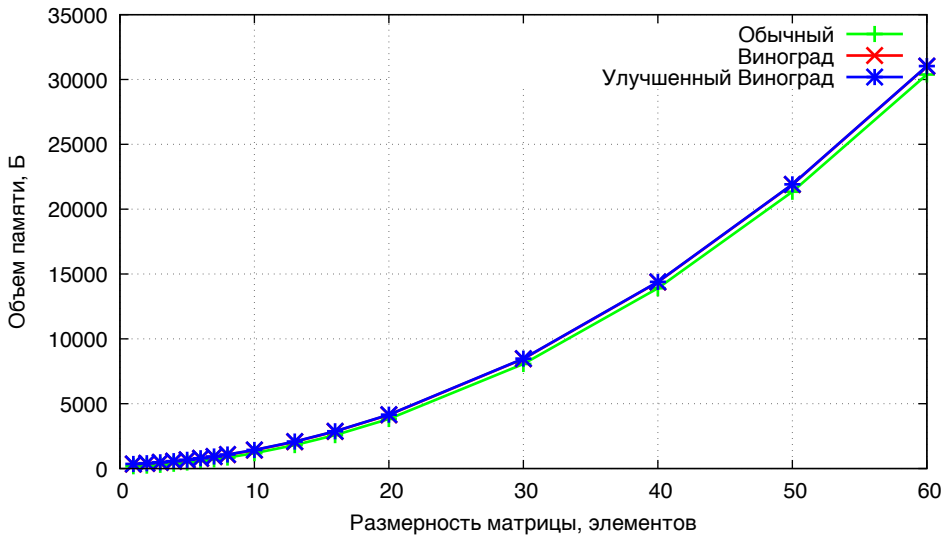

Рисунок 4.5 — Результаты замеров памяти

Так как на этом масштабе не видна разница между реализациями алгоритмов, то на рисунке 4.6 представлена зависимость потребляемой памяти от размерности квадратной матрицы на более показательном масштабе.

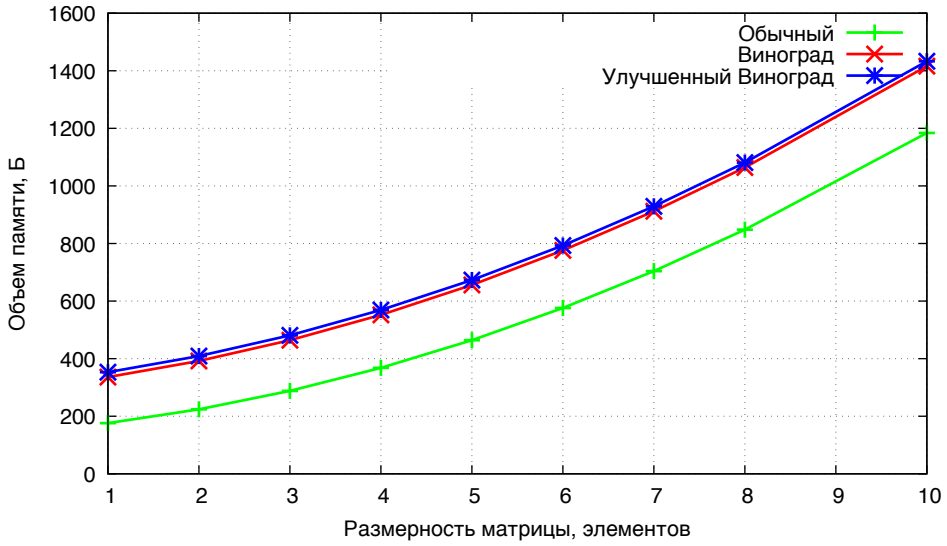

Рисунок 4.6 — Результаты замеров памяти (увеличенный масштаб)

Заключение

Все дальнейшие выводы приведены исходя из анализов данных замеров для матрицы размером 60×60, так как данная размерность позволяет получить достаточно показательные результаты для проведения сравнения.

По итогам замеров можно сказать, что алгоритм Винограда быстрее стандартного алгоритма умножения матриц в 1.98 раз в лучшем случае, и в 1.95 раз в худшем. Оптимизированный алгоритм Винограда быстрее версии без улучшений в 1.01 раз в лучшем случае, и в 1.02 раз в худшем.

Потребление памяти стандартного алгоритма умножения матриц меньше, чем у алгоритма Винограда в 1.02 раз. Алгоритм Винограда без улучшений в 1.0005 раз эффективнее оптимизированного алгоритма в плане потребляемой памяти — это обусловлено наличие дополнительных переменных, используемых для предварительного расчета некоторых параметров в варианте с улучшениями.

Цель работы была достигнута: были изучены алгоритмы умножения матриц. Были выполнены все задачи:

— изучены алгоритмы умножения матриц:

— проанализированы трудоемкости данных алгоритмов;

— кодированы данные алгоритмы;

— проведен замерный эксперимент для данных алгоритмов, с измерением времени работы и использования памяти;

— проведен сравнительный анализа алгоритмов на основе полученных данных.

Список использованных источников

[1] Боревич З. И. Определители и матрицы. – Рипол Классик, 2013.

[2] Зубков С. Assembler. Для DOS, Windows и Unix. – Litres, 2022.

[3] Документация по языку программирования *Go* [Электронный ресурс]. Режим доступа: *https://go.dev/doc* (дата обращения: 07.10.2022).

[4] Документация по пакетам языка программирования *Go* [Электронный ресурс]. Режим доступа: *https://pkg.go.dev* (дата обращения: 07.10.2022).

[5] Техническая спецификация ноутбука *MacBookAir* [Электронный ресурс]. Режим доступа: *https://support.apple.com/kb/SP869* (дата обращения: 08.10.2022).

[6] Документация по функции *clock_gettime* [Электронный ресурс]. Режим доступа: *https://man7.org/linux/man-pages/man3/clock_gettime.3.html* (дата обращения: 25.10.2022).

[7] Ватутин, Э. И. Оценка реальной производительности современных видеокарт с поддержкой технологии CUDA в задаче умножения матриц / Э. И. Ватутин, И. А. Мартынов, В. С. Титов // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. – 2014. – № 2. – С. 8-17. – EDN SKHQXF.

[8] Анисимов Н.С., Строганов Ю.В. Реализация алгоритма умножения матриц по винограду на языке Haskell // Новые информационные технологии в автоматизированных системах. 2018. №21. URL: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell (дата обращения: 29.11.2022).

[9] Pidodnya A. About improving integer multiplication in processors. – 2020.