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

по курсу «Анализ Алгоритмов»

на тему: «Алгоритмы умножения матриц»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) Миронов Г. А. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (И. О. Фамилия)

2021 г.

СОДЕРЖАНИЕ

Введение **3**

1 Аналитическая часть **4**

1.1 Некоторые теоретические сведения 4

1.2 Стандартный алгоритм умножения матриц 4

1.3 Умножение матриц по Винограду 5

1.4 Вывод 6

2 Конструкторская часть **7**

2.1 Схемы 7

2.2 Модель вычислений 10

2.3 Трудоёмкость алгоритмов 10

2.3.1 Стандартный алгоритм умножения матриц 10

2.3.2 Алгоритм Винограда 11

2.3.3 Оптимизированный алгоритм Винограда 12

2.4 Вывод 13

3 Технологическая часть **14**

3.1 Выбор языка программирования 14

3.2 Требования к программному обеспечению 14

3.3 Сведения о модулях программы 14

3.4 Вывод 19

4 Экспериментальная часть **20**

4.1 Технические характеристики 20

4.2 Тестирование 20

4.3 Временные характеристики 21

4.4 Вывод 23

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **24**

Введение

В данной лабораторной работе будут рассмотрены алгоритмы умножения матриц.

Матрицы A и B могут быть перемножены, если число столбцов матрицы A равно числу строк B.

Алгоритм Винограда – алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. 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.

На практике алгоритм Винограда не используется [1], так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров.

В настоящее время умножение матриц активно используется в компьютерной графике, криптографии.

Целью данной работы является изучение, программная реализация, а также экспериментальное сравнение следующих алгоритмов:

- стандартный алгоритм умножения матриц;
- алгоритм Винограда;
- оптимизированный алгоритм Винограда.

Для достижения данной цели необходимо решить следующие задачи:

- изучить и реализовать стандартный алгоритм умножения матриц;
- изучить и реализовать алгоритм Винограда умножения матриц;
- оптимизировать алгоритм Винограда умножения матриц;
- оценить трудоемкость реализаций алгоритмов умножения матриц теоретически;
- сравнить временные характеристики вышеизложенных алгоритмов экспериментально.

1 Аналитическая часть

1.1 Некоторые теоретические сведения

Для начала нужно ввести собственно понятие матрицы.

Матрица – объект, записываемый в виде прямоугольной таблицы элементов, которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы (формула 1.1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \tag{1.1}$$

Произведение матриц AB состоит из всех возможных комбинаций скалярных произведений вектор-строк матрицы A и вектор-столбцов матрицы B (рис. 1.1).

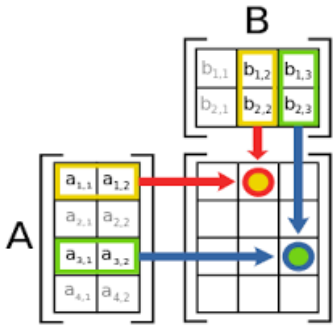

Рисунок 1.1 – Произведение матриц

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первой матрице равно числу строк во второй.

1.2 Стандартный алгоритм умножения матриц

Пусть даны матрицы A (формула 1.1) размерностью $n \times m$ и B (формула 1.2) $m \times q$.

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mq} \end{pmatrix} \tag{1.2}$$

Матрица $C = AB$ будет размерностью $n \times q$. Тогда каждый элемент матрицы C выражается формулой (1.3).

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \quad (i = 1, 2, \ldots n; j = 1, 2, \ldots q) \tag{1.3}$$

1.3 Умножение матриц по Винограду

Каждый элемент в матрице C, которая является результатом умножения двух матриц, представляет собой скалярное произведение соответствующих строки и столбца исходных матриц.

В алгоритме умножение матриц по Винограду предложено сделать предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора V (формула 1.4) и W (формула 1.5).

$$V = (v_1, v_2, v_3, v_4) \tag{1.4}$$

$$W = (w_1, w_2, w_3, w_4) \tag{1.5}$$

Их скалярное произведение вычисляется по формуле 1.6.

$$V * W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4 \tag{1.6}$$

Равенство 1.6 можно записать в виде 1.7.

$$\begin{gathered} V * W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - \\ v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \end{gathered} \tag{1.7}$$

Несмотря на то, что второе выражение 1.7 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, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного

1.4 Вывод

Были рассмотрены основополагающие материалы, которые в дальнейшем потребуются при реализации алгоритмов умножения матриц.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов умножения матриц и модель вычислений.

2.1 Схемы

На рисунке 2.1 представлена схема стандартного алгоритма умножения матриц.

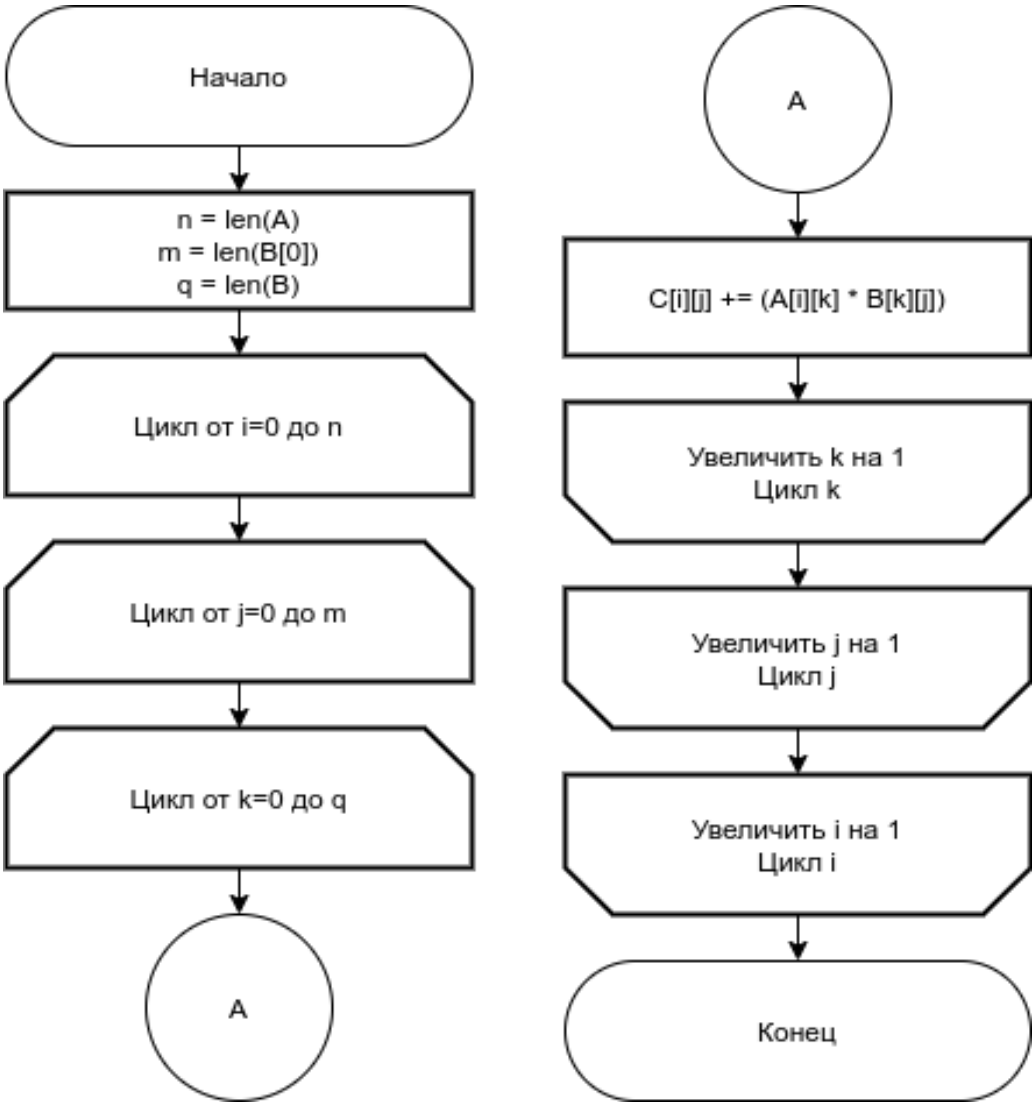

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

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

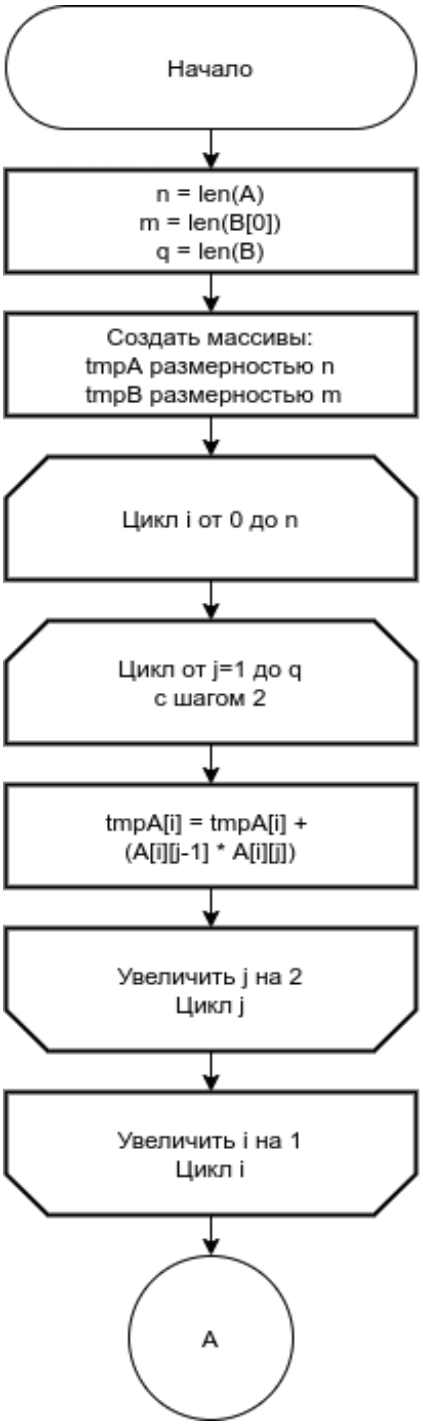

Рисунок 2.2 – Схема алгоритма Винограда умножения матриц. Часть 1

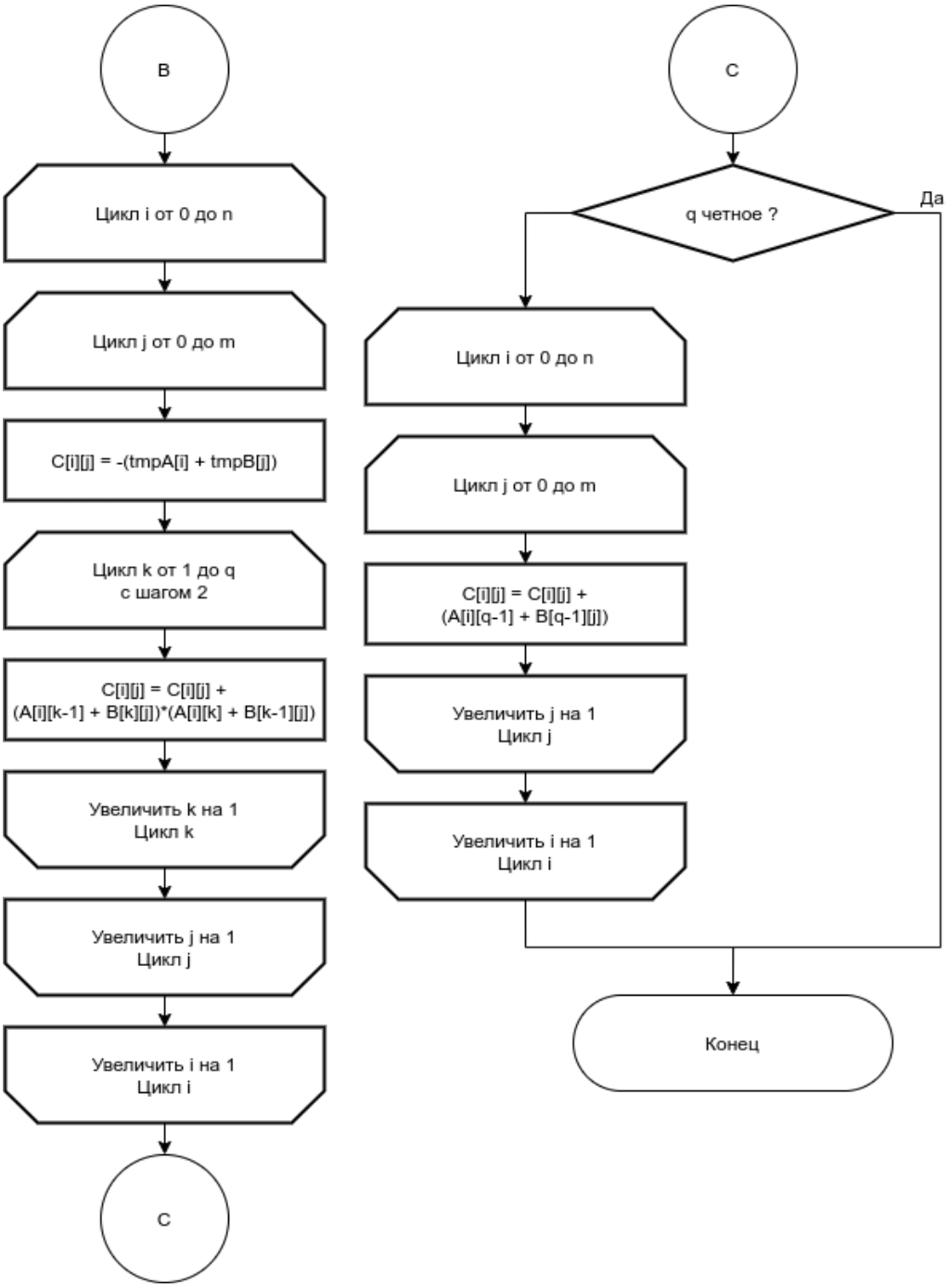

Рисунок 2.3 – Схема алгоритма Винограда умножения матриц. Часть 2

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

- операции из списка (2.1) имеют трудоемкость 1;

$$+, -, =, +=, -=, ==, !=, <, >, <=, >=, >>, <<, [], ++, --\quad (2.1)$$

- операции из списка (2.1) имеют трудоемкость 2;

$$*, /, \%, /=, *=\quad (2.2)$$

- трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.3);

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

- трудоемкость цикла рассчитывается, как (2.4).

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

2.3 Трудоёмкость алгоритмов

2.3.1 Стандартный алгоритм умножения матриц

Трудоёмкость стандартного алгоритма умножения матриц состоит из:

- внешнего цикла по $i \in [1..n]$, трудоёмкость которого:
  $f = 2 + n \cdot (2 + f_{body})$;
- цикла по $j \in [1..m]$, трудоёмкость которого: $f = 2 + m \cdot (2 + f_{body})$;
- скалярного умножения двух векторов - цикл по $k \in [1..q]$, трудоёмкость которого: $f = 2 + 14q$;

Трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела (2.5):

$$f_{base} = 2 + n \cdot (4 + m \cdot (4 + 14q)) = 2 + 4 \cdot n + 4 \cdot nm + 14 \cdot nmq \approx 14 \cdot nmq \quad (2.5)$$

2.3.2 Алгоритм Винограда

Трудоёмкость алгоритма Винограда состоит из:

1. формирования массива сумм произведений пар соседних элементов строк матрицы A:

$$f_1 = 2 + n \cdot (2 + 4 + q/2 \cdot (4 + 1 + 6 + 3 \cdot 2 + 2)) = 19/2 \cdot nm + 6 \cdot n + 2; \quad (2.6)$$

2. формирования массива сумм произведений пар соседних элементов строк матрицы B:

$$f_2 = 19/2 \cdot mq + 6 \cdot m + 2; \quad (2.7)$$

3. цикла заполнения ячеек матрицы C для чётных размеров:

$$f_3 = 2 + n \cdot (2 + 2 + m \cdot (2 + 7 + 4 + q/2 \cdot (4 + 1 + 12 + 5 + 5 \cdot 2))) = 32/2 \cdot nqm + 13 \cdot nm + 4 \cdot n + 2; \quad (2.8)$$

4. цикла для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный (2.9):

$$f_4 = 3 + \begin{cases} 0, & \text{чётная,} \\ 2 + 4 \cdot n + 13 \cdot nm), & \text{иначе.} \end{cases} \quad (2.9)$$

Итого, для худшего (нечётный размер матриц) и лучшего (чётный размер матриц) случаев:

$$f_{winograd} = 16 \cdot nqm. \quad (2.10)$$

2.3.3 Оптимизированный алгоритм Винограда

Из рисунков 2.2 – 2.3 можно заметить, что для алгоритма Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, так как отпадает необходимость в последнем цикле. Данный алгоритм можно оптимизировать:

1. заменив выражения вида `a = a + ...` на `a += ...`;
2. введя промежуточный буфер для подсчета промежуточных значений;
3. сделав в циклах по k шаг 2, избавившись тем самым от двух операций умножения на каждую итерацию.

Трудоёмкость оптимизированного алгоритма Винограда состоит из:

1. формирования массива сумм произведений пар соседних элементов строк матрицы A:
$$f_1 = 11/2 \cdot nq + 4 \cdot n + 2; \tag{2.11}$$
2. формирования массива сумм произведений пар соседних элементов строк матрицы B:
$$f_2 = 11/2 \cdot mq + 4 \cdot m + 2; \tag{2.12}$$
3. цикла заполнения ячеек матрицы C для чётных размеров:
$$f_3 = 2 + n \cdot (2 + 2 + m \cdot (2 + 6 + 2 + q/2 \cdot (2 + 1 + 10 + 4 + 1 \cdot 2))) = 17/2 \cdot nqm + 13 \cdot nm + 4 \cdot n + 2; \tag{2.13}$$
4. цикла для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный (2.14):
$$f_4 = 2 + \begin{cases} 0, & \text{чётная,} \\ 2 + 4 \cdot n + 13 \cdot nm), & \text{иначе.} \end{cases} \tag{2.14}$$

Итого, для худшего (нечётный размер матриц) и лучшего (чётный размер матриц) случаев:

$$f_{winograd} = 17/2 \cdot nqm \approx 8.5 \cdot nqm. \tag{2.15}$$

2.4 Вывод

В данном разделе были рассмотрены схемы (рис. 2.1 – 2.3) алгоритмов умножения матриц. Оценены их трудоемкости в лучшем и худшем случаях.

3 Технологическая часть

3.1 Выбор языка программирования

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Golang [2]. Данный выбор обусловлен тем, что я имею некоторый опыт разработки на нем, а так же наличием у языка встроенных высокоточных средств тестирования и анализа разработанного ПО.

3.2 Требования к программному обеспечению

Входными данными являются две матрицы A и B. Количество столбцов матрицы A должно быть равно количеству строк матрицы B.

На выходе получается результат умножения введенных пользователем матриц.

3.3 Сведения о модулях программы

Данная программа разбита на модули:

- `main.go` - файл, содержащий точку входа в программу. В нем происходит общение с пользователем и вызов алгоритмов;
- `simple_mult.go` – Файл содержит реализацию простого алгоритма умножения матриц.
- `winograd.go` – Файл содержит реализацию алгоритма умножения матриц по Винограду.
- `winograd_imp.go` – Файл содержит улучшенную реализацию алгоритма умножения матриц по Винограду.
- `utils.go` – Файл содержит различные функции для вычислений. матриц.

На листингах 3.1 – 3.7 представлен код программы.

Листинг 3.1 – Основной файл программы main

```
func main() {
    fmt.Println(aurora.Magenta("Matrix multiplication\n"))

    fmt.Println(aurora.Cyan("Matrix A"))
    fmt.Print(aurora.Cyan("Input rows count: "))
    an := matrix.ReadNum()
    fmt.Print(aurora.Cyan("Input cols count: "))
    am := matrix.ReadNum()
    fmt.Println(aurora.Cyan("Input matrix data:"))
    A := matrix.ReadMatrix(an, am)

    fmt.Println(aurora.Cyan("\nMatrix B"))
    fmt.Print(aurora.Cyan("Input rows count: "))
    bn := matrix.ReadNum()
    if bn != am {
        fmt.Println(aurora.Red("Multiplication is imposible. " +
            "Matrix B rows count is not equal " +
            " to Matrix A cols count."))
        return
    }

    fmt.Print(aurora.Cyan("Input cols count: "))
    bm := matrix.ReadNum()
    fmt.Println(aurora.Cyan("Input matrix data:"))
    B := matrix.ReadMatrix(bn, bm)

    fmt.Println(aurora.Yellow("\nResult"))
    fmt.Println(aurora.Cyan("Simple multiplication:"))
    matrix.SimpleMult(A, B).PrintMatrix()
    fmt.Println(aurora.Cyan("Winograd multiplication:"))
    matrix.WinogradMult(A, B).PrintMatrix()
    fmt.Println(aurora.Cyan("Winograd multiplication (imporved):"
       ))
    matrix.WinogradMultImp(A, B).PrintMatrix()
}
```

Листинг 3.2 – Различные функции для вычислений

```
func ReadMatrix(n, m int) MInt {
    mat := formResMat(n, m)

    for i := 0; i < mat.n; i++ {
        for j := 0; j < mat.m; j++ {
            fmt.Scanf("%d", &mat.mat[i][j])
        }
    }

    return mat
}

func ReadNum() int {
    var num int
    fmt.Scanln(&num)
    return num
}

func randomFill(mat MInt, max int) {
    for i := 0; i < mat.n; i++ {
        for j := 0; j < mat.m; j++ {
            mat.mat[i][j] = rand.Intn(max)
        }
    }
}

func formResMat(n, m int) MInt {
    var rmat MInt
    rmat.n, rmat.m = n, m
    rmat.mat = make([][]int, rmat.n)
    for i := range rmat.mat {
        rmat.mat[i] = make([]int, rmat.m)
    }
    return rmat
}
```

Листинг 3.3 – Простое умножение

```
func SimpleMult(amat, bmat MInt) MInt {
    rmat := formResMat(amat.n, bmat.m)

    for i := 0; i < rmat.n; i++ {
        for j := 0; j < rmat.m; j++ {
            for k := 0; k < amat.m; k++ {
                rmat.mat[i][j] += amat.mat[i][k] * bmat.mat[k][j]
            }
        }
    }

    return rmat
}
```

Листинг 3.4 – Алгоритм Винограда. Часть 1

```
func WinogradMult(A, B MInt) MInt {
    n, m, q := A.n, B.m, B.n
    C := formResMat(n, m)
    tmpA, tmpB := precomputeRows(A), precomputeCols(B)
    for i := 0; i < n; i++ {
        for j := 0; j < m; j++ {
            C.mat[i][j] = -(tmpA[i] + tmpB[j])
            for k := 0; k < q/2; k++ {
                C.mat[i][j] = C.mat[i][j] +
                    (A.mat[i][k*2]+B.mat[k*2+1][j])*
                        (A.mat[i][k*2+1]+B.mat[k*2][j])
            }
        }
    }
    if q%2 != 0 {
        for i := 0; i < n; i++ {
            for j := 0; j < m; j++ {
                C.mat[i][j] = C.mat[i][j] +
                    A.mat[i][q-1]*B.mat[q-1][j]
            }
        }
    }
    return C
}
```

Листинг 3.5 – Алгоритм Винограда. Часть 2

```
func precomputeRows(M MInt) []int {
    res := make([]int, M.n)
    for i := 0; i < M.n; i++ {
        for j := 0; j < M.m/2; j++ {
            res[i] = res[i] + M.mat[i][j*2]*M.mat[i][j*2+1]
        }
    }
    return res
}

func precomputeCols(M MInt) []int {
    res := make([]int, M.m)
    for i := 0; i < M.m; i++ {
        for j := 0; j < M.n/2; j++ {
            res[i] = res[i] + M.mat[j*2][i]*M.mat[j*2+1][i]
        }
    }
    return res
}
```

Листинг 3.6 – Оптимизированный алгоритм Винограда. Часть 1

```
func WinogradMultImp(A, B MInt) MInt {
    n, m, q := A.n, B.m, B.n
    C := formResMat(n, m)
    tmpA, tmpB := precomputeRowsImp(A), precomputeColsImp(B)
    for i := 0; i < n; i++ {
        for j := 0; j < m; j++ {
            temp := -(tmpA[i] + tmpB[j])
            for k := 1; k < q; k += 2 {
                temp += (A.mat[i][k-1] + B.mat[k][j]) *
                    (A.mat[i][k] + B.mat[k-1][j])
            }
            C.mat[i][j] = temp
        }
    }
    if q&1 != 0 {
        for i := 0; i < n; i++ {
            for j := 0; j < m; j++ {
                C.mat[i][j] += A.mat[i][q-1] * B.mat[q-1][j]
            }
        }
    }
    return C
}
```

Листинг 3.7 – Оптимизированный алгоритм Винограда. Часть 2

```
func precomputeRowsImp(M MInt) []int {
    res := make([]int, M.n)
    for i := 0; i < M.n; i++ {
        for j := 1; j < M.m; j += 2 {
            res[i] += M.mat[i][j-1] * M.mat[i][j]
        }
    }
    return res
}

func precomputeColsImp(M MInt) []int {
    res := make([]int, M.m)
    for i := 0; i < M.m; i++ {
        for j := 1; j < M.n; j += 2 {
            res[i] += M.mat[j-1][i] * M.mat[j][i]
        }
    }
    return res
}
```

3.4 Вывод

В данном разделе были реализованны вышеописанные алголритмы.

Было разработано программное обеспечение, удовлетворяющее предъявляемым требованиям. Так же были представлены соответствующие листинги 3.1 – 3.7 с кодом программы.

4 Экспериментальная часть

В данном разделе будет проведено функциональное тестирование разработанного программного обеспечения. Так же будет произведено измерение временных характеристик каждого из реализованных алгоритмов.

Для проведения подобных экспериментов на языке программирования `Golang` [2], используется специальный пакет `testing` [3]. Данный пакет предоставляет инструменты для измерения процессорного времени и объема памяти, использованных конкретным алгоритмом в ходе проведения эксперимента.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- Процессор: Intel Core™ i5-8250U [4] CPU @ 1.60GHz.
- Память: 32 GiB.
- Операционная система: Ubuntu [5] Linux [6] 20.04 64-bit.

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

4.2 Тестирование

В данном разделе будет приведена таблица с тестами (таблица 4.1).

Таблица 4.1 – Таблица тестов

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}4\end{pmatrix}$
$\begin{pmatrix}2\\1\end{pmatrix}$	$\begin{pmatrix}1 & 2\end{pmatrix}$	$\begin{pmatrix}2 & 4\\1 & 2\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 2\\1 & 2 & 2\end{pmatrix}$	$\begin{pmatrix}1 & 2\\1 & 2\\1 & 2\end{pmatrix}$	$\begin{pmatrix}5 & 10\\5 & 10\end{pmatrix}$
$\begin{pmatrix}1 & -2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}-1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}0 & 4 & 6\\4 & 12 & 18\\4 & 12 & 18\end{pmatrix}$

При проведении функционального тестирования, полученные результаты работы программы совпали с ожидаемыми. Таким образом, функциональное тестирование пройдено успешно.

4.3 Временные характеристики

Для сравнения возьмем квадратные матрицы размерностью [100, 200, 300, . . . , 800].

Результаты замеров по результатам экспериментов приведены в Таблице 4.2.

Таблица 4.2 – Замер времени для матриц, размером от 100 до 800 элементов

	Время, нс		
Размерность матрицы, эл.	**Простой**	**Виноград**	**Оптимизированный Виноград**
100	2169144	2424132	1811016
200	17680401	22032658	15485909
300	65838464	76854395	55530661
400	210892129	207347115	162146803
500	364922866	350837776	327348293
600	666673788	655599427	631239232
700	1369979155	1332273841	1207834761
800	3205123980	3108582931	2921219674

Отдельно сравним временные характеристики при нечетных размерностях матриц [101, 201, 301, . . . , 801].

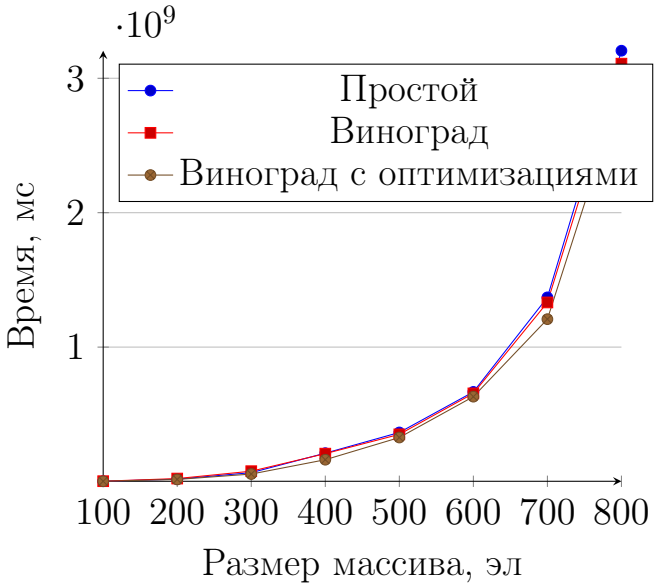

Рисунок 4.1 – Временные характеристики на четных размерах матриц

Результаты замеров по результатам экспериментов приведены в Таблице 4.3.

Таблица 4.3 – Замер времени для матриц, размером от 101 до 801 элементов

	Время, нс		
Размерность матрицы, эл.	**Простой**	**Виноград**	**Оптимизированный Виноград**
101	2259672	2796926	2018691
201	20071583	21688038	18481951
301	74128394	77705821	61183913
401	221489215	218382194	158158291
501	371283268	372238416	361183214
601	672286913	671599427	648239232
701	1382138491	1384273841	1324143862
801	3291482918	3389582931	3217248194

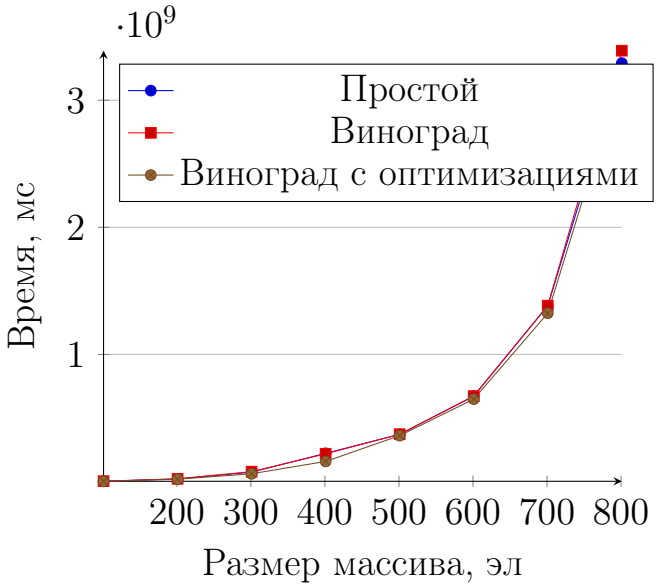

Рисунок 4.2 – Временные характеристики на нечетных размерах матриц

4.4 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. Наименее затратным по времени оказался оптимизированный алгоритм Винограда. Но при этом ему дополнительно требуется $n + m$ памяти под результат (т.е. $n + m$ прибавляется к тому количеству памяти, которое выделяется под результат в реализации стандарного алгоритма умножения матриц - $n \cdot m$).

Время работы реализации алгоритма Винограда незначительно меньше времени работы реализации простого алгоритма умножения, однако при размере 800800 время вычислений реализации алгоритма Винограда на 0.3 секунды меньше, нежели у реализации простого алгоритма (что составляет в данном случае порядка 10%).

Такой результат совпадает с теоретически полученными оценками трудоемкости алгоритмов

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Group-theoretic Algorithms for Matrix Multiplication / H. Cohn [и др.] // Proceedings of the 46th Annual Symposium on Foundations of Computer Science. — 2005. — С. 379—388.
2. The Go Programming Language [Электронный ресурс]. — Режим доступа: `https://golang.org/` (дата обращения: 14.09.2021).
3. testing – The Go Programming Language [Электронный ресурс]. — Режим доступа: `https://golang.org/pkg/testing/` (дата обращения: 14.09.2021).
4. Процессор Intel® Core™ i5-8250U [Электронный ресурс]. — Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html` (дата обращения: 14.09.2021).
5. Ubuntu: Enterprise Open Source and Linux [Электронный ресурс]. — Режим доступа: `https://ubuntu.com/` (дата обращения: 14.09.2021).
6. LINUX.ORG.RU – Русская информация об ОС Linux [Электронный ресурс]. — Режим доступа: `https://www.linux.org.ru/` (дата обращения: 14.09.2021).