



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 Богаченко А.Е.

Группа ИУ7-56Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение	3
1 Аналитическая часть	4
1.1 Алгоритм Винограда	4
2 Конструкторская часть	6
2.1 Схемы алгоритмов	6
2.2 Трудоемкость алгоритмов	9
2.2.1 Классический алгоритм	10
2.2.2 Алгоритм Винограда	10
2.2.3 Оптимизированный алгоритм Винограда	10
3 Технологическая часть	12
3.1 Требования к ПО	12
3.2 Средства реализации	12
3.3 Листинг кода	13
4 Исследовательская часть	17
4.1 Пример работы	17
4.2 Технические характеристики	18
4.3 Время выполнения алгоритмов	18
4.4 Производительность алгоритмов	20
Заключение	22
Литература	23

Введение

Цель работы: изучение алгоритмов умножения матриц. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда. Также требуется изучить расчет сложности алгоритмов, получить навыки в улучшении алгоритмов.

Задачи лабораторной работы:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- оптимизировать алгоритм Винограда;
- дать теоретическую оценку трудоемкости классического алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда;
- реализовать три алгоритма умножения матриц на одном из языков программирования;
- сравнить алгоритмы умножения матриц.

1 Аналитическая часть

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.[1] 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 $[m \times k]$.

$c_{i,j} = \sum_{r=1}^n a_{i,r} \cdot b_{r,j}$ называется произведением матриц A и B [1].

1.1 Алгоритм Винограда

Подход алгоритма Винограда является иллюстрацией общей методологии, начатой в 1979 годах на основе билинейных и трилинейных форм, благодаря которым большинство усовершенствований для умножения матриц были получены [2].

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 (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \quad (1.1)$$

Равенство (1.1) можно переписать в виде (1.2)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \quad (1.2)$$

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.

Вывод

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

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 приведена схема классического алгоритма умножения матриц.

На рисунке 2.2 приведена схема алгоритма Винограда.

На рисунке 2.3 приведена схема оптимизированного алгоритма Винограда.

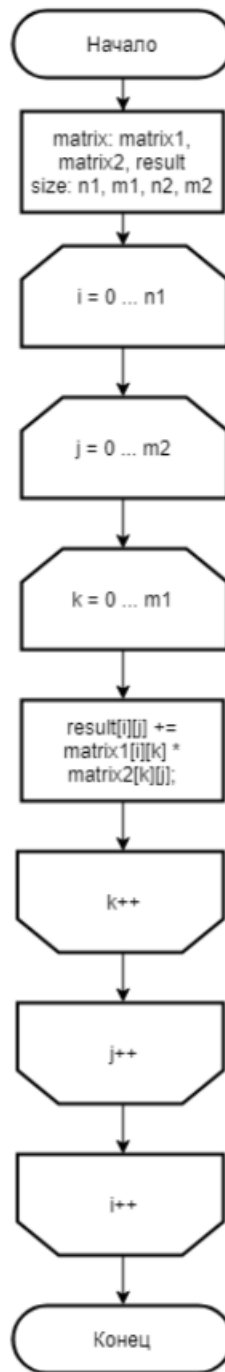


Рисунок 2.1 – Схема классического алгоритма умножения матриц

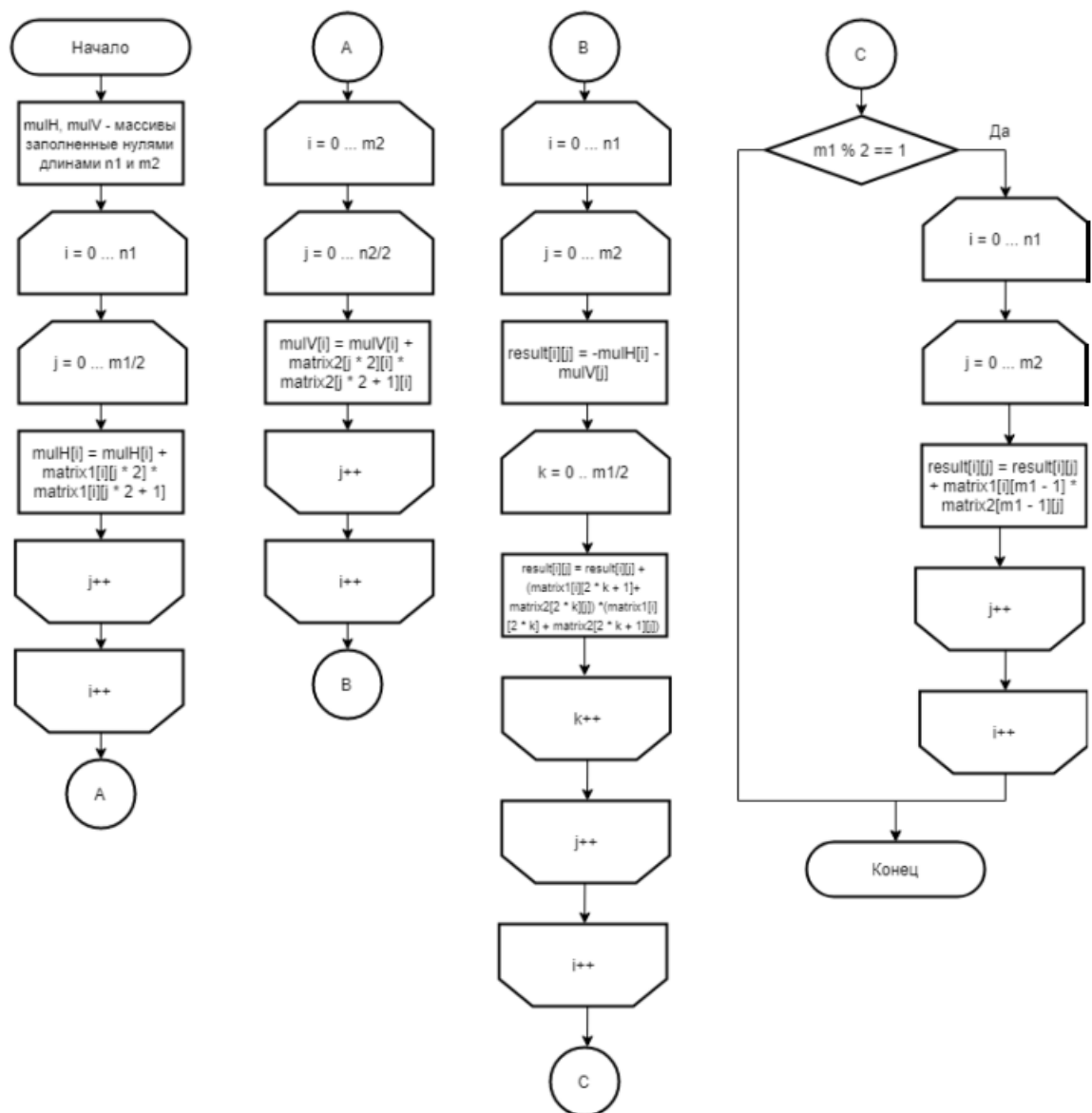


Рисунок 2.2 – Схема алгоритма Винограда

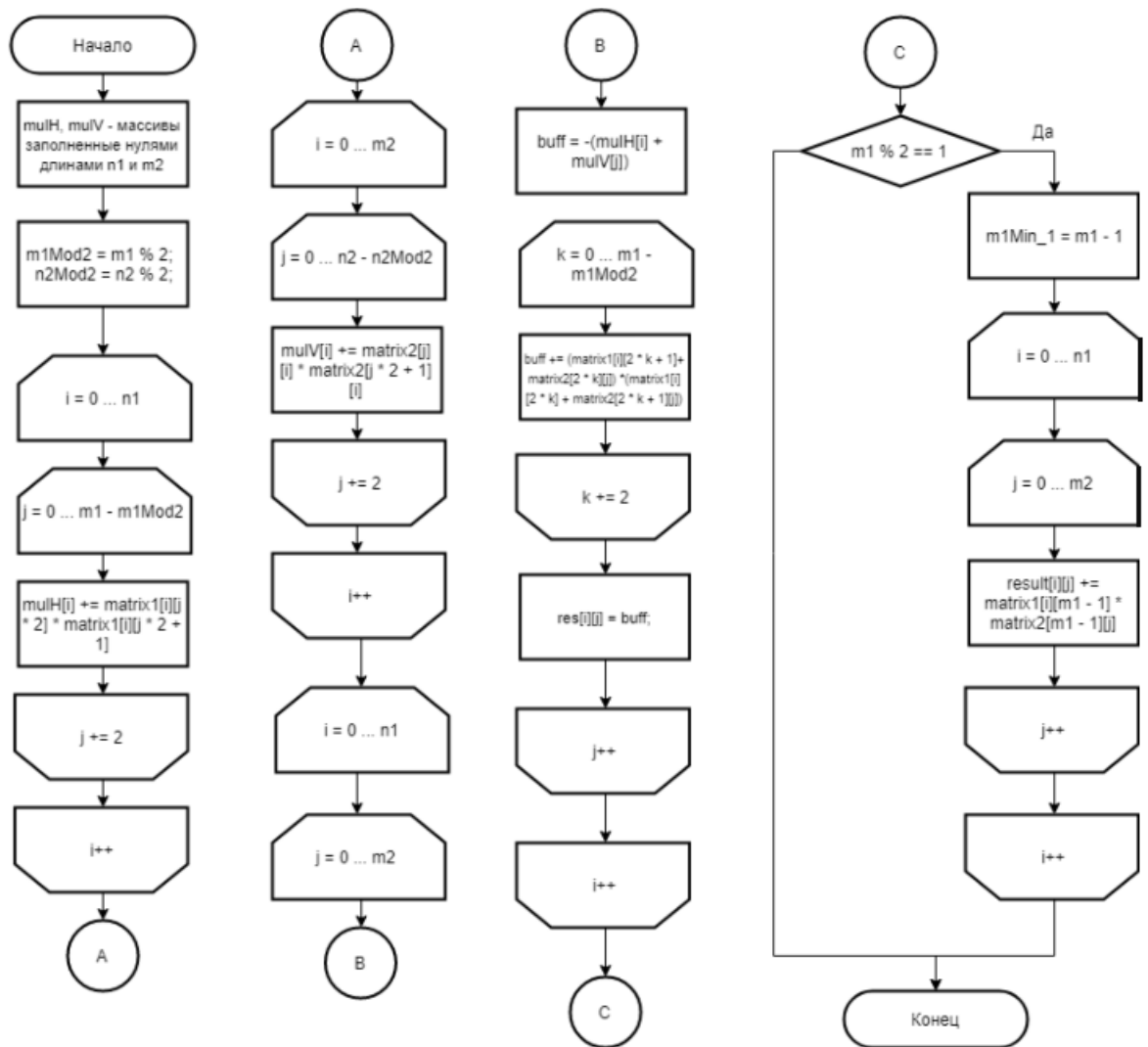


Рисунок 2.3 – Схема оптимизированного алгоритма Винограда

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — $+$, $-$, $*$, $/$, $=$, $==$, $<=$, $>=$, $!=$, $+=$, $||$, $++$, $-$ получение полей класса;
- оценка трудоемкости цикла: $F_{\text{ц}} = a + N \cdot (a + F_{\text{тела}})$, где a - условие цикла;
- стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

2.2.1 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма:

Инициализация матрицы результата: $1 + 1 + n_1(1 + 2 + 1) + 1 = 4n_1 + 3$

Подсчет:

$$1 + n_1(1 + (1 + m_2(1 + (1 + m_1(1 + (8) + 1) + 1) + 1) + 1) + 1) + 1 = \\ n_1(m_2(10m_1 + 4) + 4) + 4 + 2 = 10n_1m_2m_1 + 4n_1m_2 + 4n_1 + 2$$

2.2.2 Алгоритм Винограда

Аналогично рассмотрим трудоемкость алгоритма Винограда.

Первый цикл: $\frac{15}{2}n_1m_1 + 5n_1 + 2$

Второй цикл: $\frac{15}{2}m_2n_2 + 5m_2 + 2$

Третий цикл: $13n_1m_2m_1 + 12n_1m_2 + 4n_1 + 2$

Условный переход: $\begin{bmatrix} 2 & , \text{ невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & , \text{ выполнение условия} \end{bmatrix}$

Итого: $13n_1m_2m_1 + \frac{15}{2}n_1m_1 + \frac{15}{2}m_2n_2 + 12n_1m_2 + 5n_1 + 5m_2 + 4n_1 + 6 +$
 $\begin{bmatrix} 2 & , \text{ невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & , \text{ выполнение условия} \end{bmatrix}$

2.2.3 Оптимизированный алгоритм Винограда

Аналогично Рассмотрим трудоемкость оптимизированного алгоритма Винограда:

Первый цикл: $\frac{11}{2}n_1m_1 + 4n_1 + 2$

Второй цикл: $\frac{11}{2}m_2n_2 + 4m_2 + 2$

Третий цикл: $\frac{17}{2}n_1m_2m_1 + 9n_1m_2 + 4n_1 + 2$

Условный переход: $\begin{bmatrix} 1 & , \text{ невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & , \text{ выполнение условия} \end{bmatrix}$

$$\begin{aligned} & \text{Итого: } \frac{17}{2}n_1m_2m_1 + \frac{11}{2}n_1m_1 + \frac{11}{2}m_2n_2 + 9n_1m_2 + 8n_1 + 4m_2 + 6 + \\ & \left[\begin{array}{ll} 1 & , \text{ невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & , \text{ выполнение условия} \end{array} \right] \end{aligned}$$

Вывод

На основе теоретических данных, полученных из аналитического раздела были построены схемы требуемых алгоритмов. Была введена модель оценки трудоемкости алгоритма, были рассчитаны трудоемкости алгоритмов в соответствии с этой моделью.

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- корректное умножение матриц;
- при матрицах неправильных размеров программа не должна аварийно завершаться.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран многопоточный язык GO [3]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка. Так же данный язык предоставляет средства тестирования разработанного ПО.

3.3 Листинг кода

В листингах 3.1–3.3 приведены реализации алгоритмов умножения матриц.

Листинг 3.1 – Классический

```
1 func Multiply(a, b Matrix) (res Matrix) {
2     rows := a.Rows()
3     columns := b.Cols()
4
5     res = matrix.Zeros(rows, columns)
6
7     for i := 0; i < rows; i++ {
8         for j := 0; j < columns; j++ {
9             for k := 0; k < rows; k++ {
10                 res.Set(i, j, res.Get(i, j)+a.Get(i, k)*b.Get(k, j))
11             }
12         }
13     }
14
15     return
16 }
```

Листинг 3.2 – Виноград

```
1 func Winograd(a, b Matrix) (res Matrix) {
2     row1 := a.Rows()
3     row2 := b.Rows()
4     col2 := b.Cols()
5
6     if row2 != a.Cols() {
7         return
8     }
9
10    rowFactor := make([]int, row1)
11    for i := range rowFactor {
12        rowFactor[i] = 0
13    }
14    colFactor := make([]int, col2)
15    for i := range rowFactor {
16        colFactor[i] = 0
17    }
18
19    d := row2 / 2
20    for i := 0; i < row1; i++ {
21        for j := 0; j < d; j++ {
22            rowFactor[i] += a.Get(i, 2*j) * a.Get(i, 2*j+1)
```

```

23     }
24 }
25
26 for i := 0; i < col2; i++ {
27     for j := 0; j < d; j++ {
28         colFactor[i] += b.Get(2*j, i) * b.Get(2*j+1, i)
29     }
30 }
31
32 res = matrix.Zeros(row1, col2)
33 for i := 0; i < row1; i++ {
34     for j := 0; j < col2; j++ {
35         res.Set(i, j, -rowFactor[i]-colFactor[j])
36         for k := 0; k < d; k++ {
37             res.Set(i, j, res.Get(i, j)+
38                 ((a.Get(i, 2*k)+b.Get(2*k+1, j))*
39                 (a.Get(i, 2*k+1)+b.Get(2*k, j))))
40         }
41     }
42 }
43
44 if row2%2 != 0 {
45     for i := 0; i < row1; i++ {
46         for j := 0; j < col2; j++ {
47             res.Set(i, j, res.Get(i, j)+a.Get(i, row2-1)*b.Get(row2-1, j))
48         }
49     }
50 }
51
52 return
53 }

```

Листинг 3.3 – Оптимизированный Виноград

```

1 func ImpWinograd(a, b Matrix) (res Matrix) {
2     row1 := a.Rows()
3     row2 := b.Rows()
4     col2 := b.Cols()
5
6     if row2 != a.Cols() {
7         return
8     }
9
10    rowFactor := make([]int, row1)
11    for i := range rowFactor {
12        rowFactor[i] = 0
13    }
14    colFactor := make([]int, col2)
15    for i := range rowFactor {

```

```

16     colFactor[i] = 0
17 }
18
19 d := row2 / 2
20
21 for i := 0; i < row1; i++ {
22     val := 0
23     for j := 0; j < d; j++ {
24         val += a.Get(i, 2*j) * a.Get(i, 2*j+1)
25     }
26     rowFactor[i] = val
27 }
28
29 for i := 0; i < col2; i++ {
30     val := 0
31     for j := 0; j < d; j++ {
32         val += b.Get(2*j, i) * b.Get(2*j+1, i)
33     }
34     colFactor[i] = val
35 }
36
37 res = matrix.Zeros(row1, col2)
38
39 for i := 0; i < row1; i++ {
40     for j := 0; j < col2; j++ {
41         val := 0
42         for k := 0; k < d; k++ {
43             val += (a.Get(i, 2*k) + b.Get(2*k+1, j)) *
44                 (a.Get(i, 2*k+1) + b.Get(2*k, j))
45         }
46         res.Set(i, j, val-rowFactor[i]-colFactor[j])
47     }
48 }
49
50
51 if row2%2 != 0 {
52     for i := 0; i < row1; i++ {
53         for j := 0; j < col2; j++ {
54             res.Set(i, j, res.Get(i, j)+a.Get(i, row2-1)*b.Get(row2-1, j))
55         }
56     }
57 }
58 return

```

В таблице 3.1 приведены функциональные тесты для алгоритмов умножения матриц.

Таблица 3.1 – Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{pmatrix}$

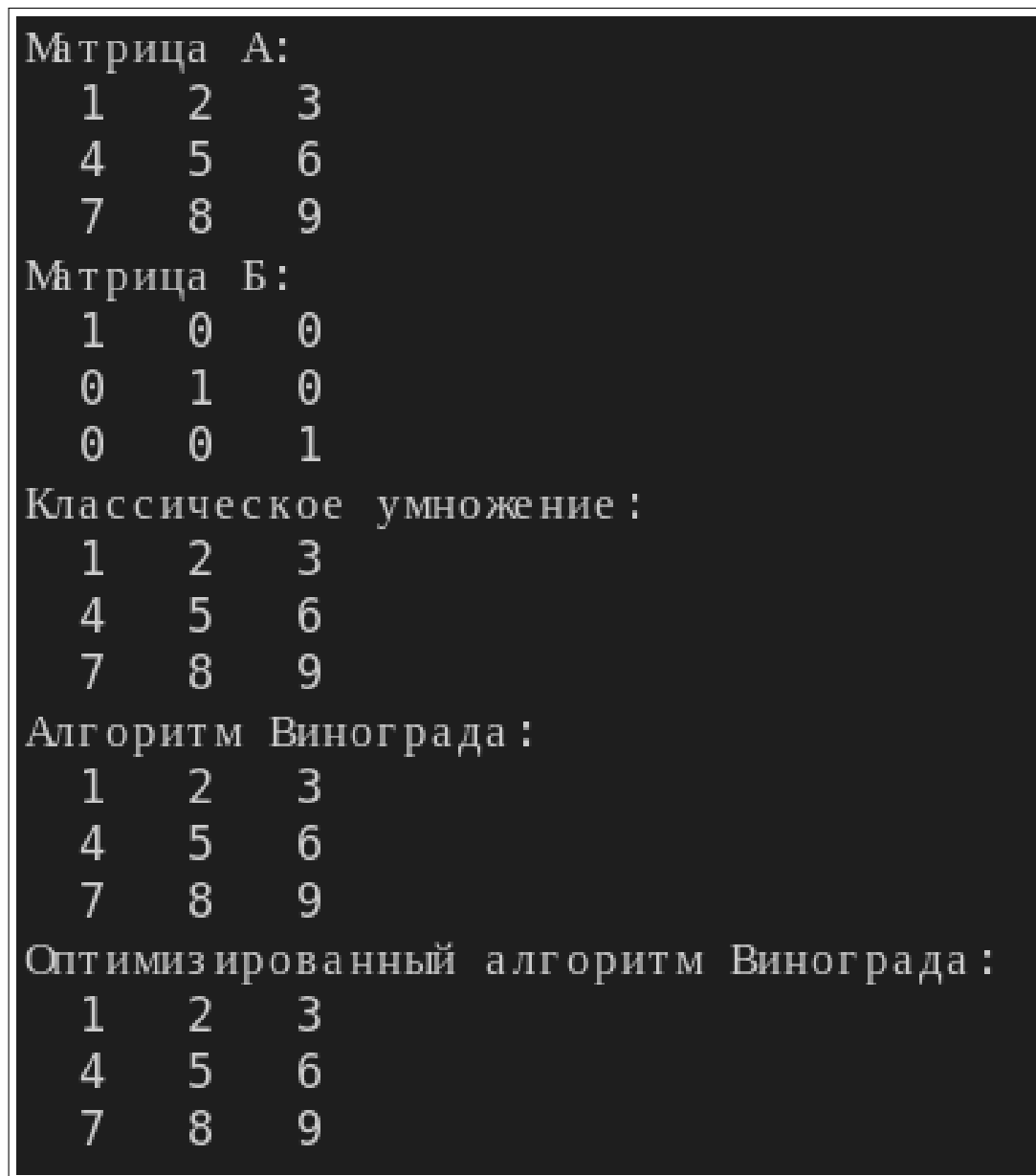
Вывод

Были разработаны и протестированы спроектированные алгоритмы: стандартного умножения матриц, алгоритм Винограда, а также оптимизированный алгоритм Винограда.

4 Исследовательская часть

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.



```
Матрица А:
1 2 3
4 5 6
7 8 9
Матрица Б:
1 0 0
0 1 0
0 0 1
Классическое умножение :
1 2 3
4 5 6
7 8 9
Алгоритм Винограда :
1 2 3
4 5 6
7 8 9
Оптимизированный алгоритм Винограда :
1 2 3
4 5 6
7 8 9
```

Рисунок 4.1 – Демонстрация работы алгоритмов умножения матриц

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Kali [4] Linux [5] 5.8.10-1kali1 64-bit.
- Память: 8 GB.
- Процессор: Intel® Core™ i5-8250U [6] CPU @ 1.60GHz

Тестирование проводилось на ноутбуке при включённом режиме производительности. Во время тестирования ноутбук был нагружен только системными процессами.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи написания «бенчмарков» [7], предоставляемых встроенными в Go средствами. Для такого рода тестирования не нужно самостоятельно указывать количество повторов. В библиотеке для тестирования существует константа N , которая динамически варьируется в зависимости от того, был ли получен стабильный результат или нет.

В листинге 4.1 пример реализации бенчмарка.

Листинг 4.1 – Реализация бенчмарка

```
1 package multiplication
2
3 import (
4     "testing"
5     "../matrix"
6 )
7
8 // Std benchmarks
9
10 func BenchmarkStandart3x3(b *testing.B) {
11     m1 := matrix.Zeros(3, 3)
12     m2 := matrix.Zeros(3, 3)
13 }
```

```

14   for i := 0; i < b.N; i++ {
15       Multiply(m1, m2)
16   }
17 }

```

Результаты замеров приведены в таблице 4.1. На рисунках 4.2 и 4.3 приведены графики зависимостей времени работы алгоритмов от нечетных и четных размеров матриц.

Таблица 4.1 – Замер времени для разных размеров матриц

Размер матрицы	Время, нс		
	Classic	Winograd	ImpWinograd
3x3	677	786	705
9x9	12592	12288	8541
4x4	1293	1316	1164
8x8	8934	7740	6074

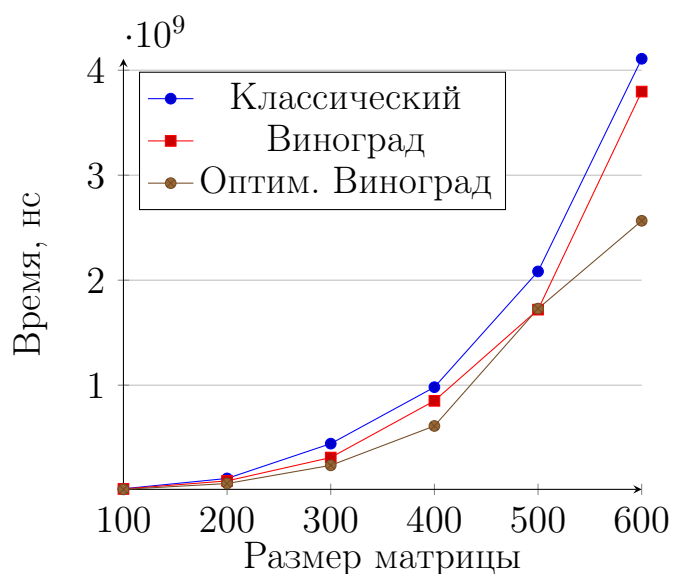


Рисунок 4.2 – Зависимость времени работы алгоритма умножения матриц от размера матрицы. Четные матрицы.

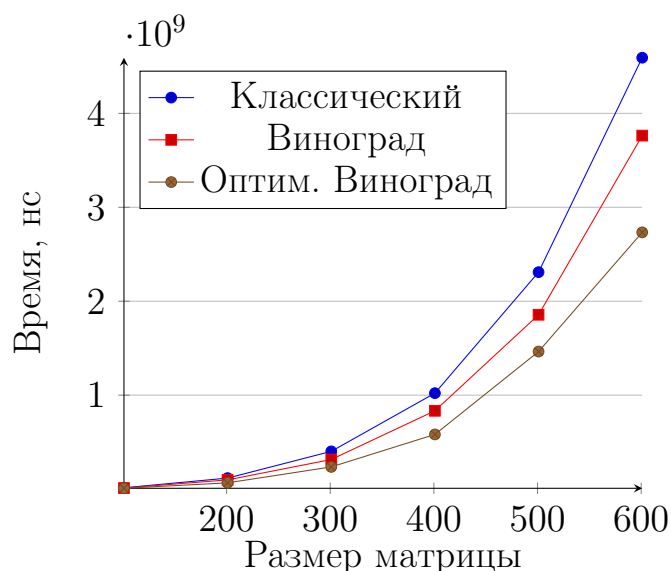


Рисунок 4.3 – Зависимость времени работы алгоритма умножения матриц от размера матрицы. Нечетные матрицы.

4.4 Производительность алгоритмов

Производительность и объем выделенной памяти при работе алгоритмов указаны на рисунке 4.4.

```

goos: linux
goarch: amd64
BenchmarkStandart3x3-8      1725253          677 ns/op      256 B/op       5 allocs/op
BenchmarkStandart9x9-8      91708           12592 ns/op     1504 B/op      5 allocs/op
BenchmarkStandart4x4-8     945428           1293 ns/op      352 B/op      5 allocs/op
BenchmarkStandart8x8-8     150613           8934 ns/op     1120 B/op     5 allocs/op
BenchmarkWinograd3x3-8     1487608           786 ns/op      320 B/op      7 allocs/op
BenchmarkWinograd9x9-8     98996           12288 ns/op     1664 B/op     7 allocs/op
BenchmarkWinograd4x4-8     839816           1316 ns/op      416 B/op      7 allocs/op
BenchmarkWinograd8x8-8     140247           7740 ns/op     1248 B/op     7 allocs/op
BenchmarkImpWinograd3x3-8  1653600           705 ns/op      320 B/op      7 allocs/op
BenchmarkImpWinograd9x9-8  122174           8541 ns/op     1664 B/op     7 allocs/op
BenchmarkImpWinograd4x4-8  1000000           1164 ns/op      416 B/op      7 allocs/op
BenchmarkImpWinograd8x8-8  178983           6074 ns/op     1248 B/op     7 allocs/op
PASS
ok      _/root/bmstu-aa/lab2/src/multiplication 19.845s

```

Рисунок 4.4 – Замеры производительности алгоритмов, выполненные при помощи команды `go test -bench . -benchmem`

Вывод

Были протестированы различные алгоритмы умножения матриц. По результатам эксперимента классический алгоритм показывает худшие вре-

менные показатели. Алгоритм Винограда и оптимизированный алгоритм Винограда работают немногим хуже на нечетных размерах матриц.

Заключение

В ходе выполнения работы были выполнены все поставленные задачи и изучены методы динамического программирования на основе алгоритмов умножения матриц. Был проведён анализ каждого алгоритма и измерено время работы для разных размеров матриц. Была оценена трудоёмкость алгоритмов.

Лучшее время умножения показывает оптимизированный алгоритм Винограда. Было установлено, что на нечетных размерах матриц алгоритмы работают немногим хуже (в 1.036 раза). Классический алгоритм медленнее улучшенного Винограда (в 1.68 раза).

Литература

- [1] Белоусов И. В. Матрицы и определители. Учебное пособие по линейной алгебре. – Институт прикладной физики, г. Кишинёв, 2006. С. 1–16.
- [2] Le Gall F. Faster algorithms for rectangular matrix multiplication. – Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), 2012. P. 514–523.
- [3] The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/> (дата обращения: 09.09.2020).
- [4] Our Most Advanced Penetration Testing Distribution, Ever. [Электронный ресурс]. Режим доступа: <https://kali.org/> (дата обращения: 12.09.2020).
- [5] Linux – Википедия [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux> (дата обращения: 12.09.2020).
- [6] Intel Processors [Электронный ресурс]. Режим доступа: <https://www.intel.com/content/www/us/en/products/processors/core/i5-processors.html> (дата обращения: 12.09.2020).
- [7] testing – The Go Programming Language [Электронный ресурс]. Режим доступа: <https://golang.org/pkg/testing/> (дата обращения: 12.09.2020).