



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»

на тему: «Алгоритмы умножения матриц»

Студент группы ИУ7-56Б

\_\_\_\_\_  
(Подпись, дата)

Мамврийский И. С.  
(Фамилия И.О.)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Волкова Л. Л.  
(Фамилия И.О.)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Строганов Ю. В..  
(Фамилия И.О.)

Москва — 2023 г.

# Содержание

<b>1</b>	<b>Аналитическая часть</b>	<b>4</b>
1.1	Стандартный алгоритм умножения матриц . . . . .	4
1.2	Алгоритм Виноградова для умножения матриц . . . . .	5
1.3	Оптимизированный алгоритм Винограда . . . . .	6
<b>2</b>	<b>Конструкторская часть</b>	<b>7</b>
2.1	Разработка алгоритмов стандартного умножения матриц и алгоритма Винограда . . . . .	7
2.2	Модель вычислений для проведения оценки трудоемкости .	16
2.3	Оценка трудоемкости алгоритмов . . . . .	17
2.3.1	Стандартный алгоритм . . . . .	17
2.3.2	Алгоритм Виноградова . . . . .	18
2.4	Оптимизация алгоритма Винограда . . . . .	19
2.4.1	Оптимизированный алгоритм Виноградова . . . . .	20
2.5	Описание используемых типов данных . . . . .	21
2.6	Вывод . . . . .	22
<b>3</b>	<b>Технологическая часть</b>	<b>23</b>
3.1	Средства реализации . . . . .	23
3.2	Реализация алгоритмов . . . . .	23
3.3	Вывод . . . . .	26
<b>4</b>	<b>Исследовательская часть</b>	<b>27</b>
4.1	Технические характеристики . . . . .	27
4.2	Пример работы . . . . .	27
4.3	Время выполнения алгоритмов . . . . .	28
4.4	Вывод . . . . .	29
	<b>Заключение</b>	<b>30</b>

# Введение

Умножение матриц является имеет многочисленное применение в физике, математике, программировании. При этом сложность стандартного алгоритма умножения матриц  $N \times N$  составляет  $O(N^3)$ , что послужило причиной разработки новых алгоритмов меньшей сложности. Одним из таких алгоритмов является алгоритм Виноградова со сложностью  $(O(N^{2.3755}))$ .

Целью данной лабораторной работы является изучение алгоритмов умножения матриц таких, как: стандартный и Виноградова. Также будут получены навыки расчета сложности алгоритмов и их оптимизация.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- 1) Описать данные алгоритмы умножения матриц;
- 2) Создать программное обеспечение, реализующее следующие алгоритмы:
  - классический алгоритм умножения матриц;
  - алгоритм Винограда;
  - оптимизированный алгоритм Винограда.

# 1 Аналитическая часть

В данном разделе будут рассмотрены стандартный алгоритм умножения матриц, алгоритм Виноградова и его оптимизация.

## 1.1 Стандартный алгоритм умножения матриц

Стандартный алгоритм умножения матриц является реализацией математического определения произведения матриц, которое формулируется следующим образом: пусть даны две

$$A_{np} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad (1.1)$$

имеющая  $n$  строк и  $p$  столбцов,

$$B_{pm} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \quad (1.2)$$

имеющая  $p$  строк и  $m$  столбцов, тогда матрица  $C$  будет иметь  $n$  строк и  $m$  столбцов

$$C_{nm} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \quad (1.3)$$

где

$$c_{ij} = \sum_{r=1}^m a_{ir} b_{rj} \quad (i = \overline{1, l}; j = \overline{1, n}) \quad (1.4)$$

- называется **произведением матриц**  $A$  и  $B$ .

## 1.2 Алгоритм Виноградова для умножения матриц

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

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 (1.5):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \quad (1.5)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырёх умножений - шесть, а вместо трёх 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.

## 1.3 Оптимизированный алгоритм Винограда

При программной реализации рассмотренного выше алгоритма Винограда можно сделать следующие оптимизации:

- 1) значение  $\frac{N}{2}$ , используемое как ограничения цикла подсчёта предварительных данных, можно кэшировать;
- 2) операцию умножения на 2 программно эффективнее реализовывать как побитовый сдвиг влево на 1;
- 3) операции сложения и вычитания с присваиванием следует реализовывать при помощи соответствующего оператора  $+=$  или  $-=$  (при наличии данных операторов в выбранном языке программирования).
- 4) вместо постоянного обращения к элементам матрицы по индексам, можно использовать буфер, в который будет записан результат выполнения цикла и потом который будет присвоен элементу матрицы.

## Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц – классического и Винограда, который имеет большую эффективность за счёт предварительной обработки, а также количество операций умножения. Также были рассмотрены оптимизации, которые можно учесть при программной реализации алгоритма Винограда.

Алгоритмы будут получать на вход две матрицы, причём количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы.

## 2 Конструкторская часть

В данном подразделе приводятся схемы разработанных алгоритмов, оценка их трудоемкости, на основе которой производится оптимизация алгоритма Винограда с последующим описанием алгоритма в виде схемы.

### 2.1 Разработка алгоритмов стандартного умножения матриц и алгоритма Винограда

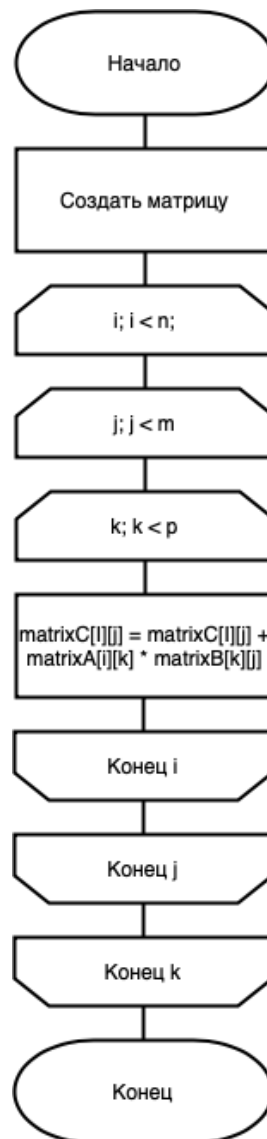


Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

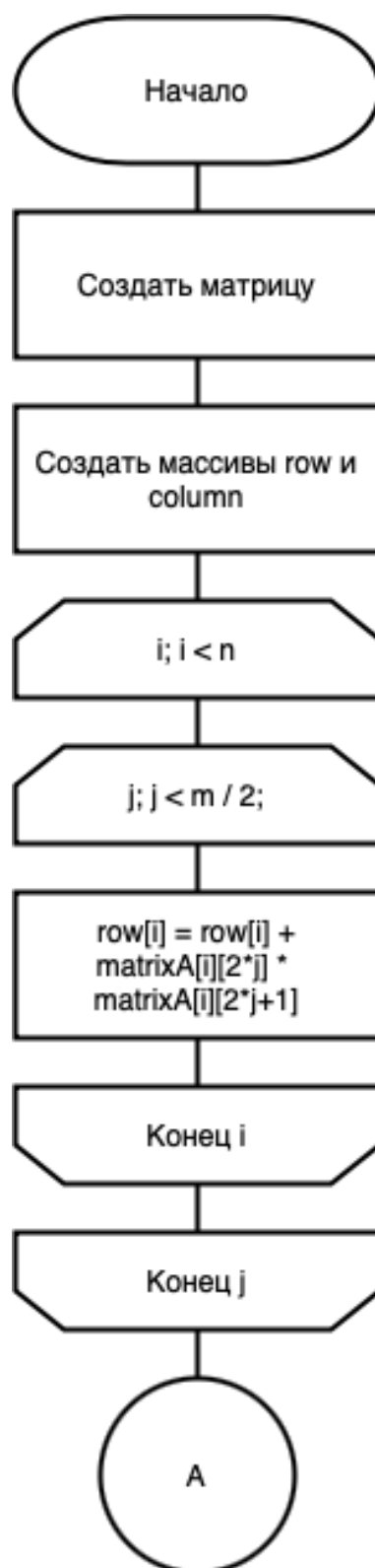


Рисунок 2.2 – Схема алгоритма Виноградова. Вычисление сумм произведений пар соседних элементов строк матрицы



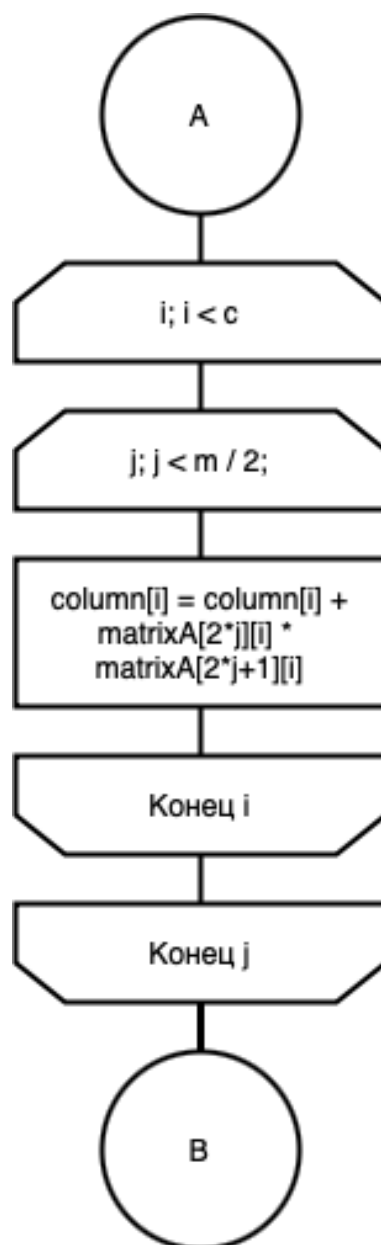


Рисунок 2.3 – Схема алгоритма Виноградова. Вычисление сумм произведений пар соседних элементов столбцов матрицы

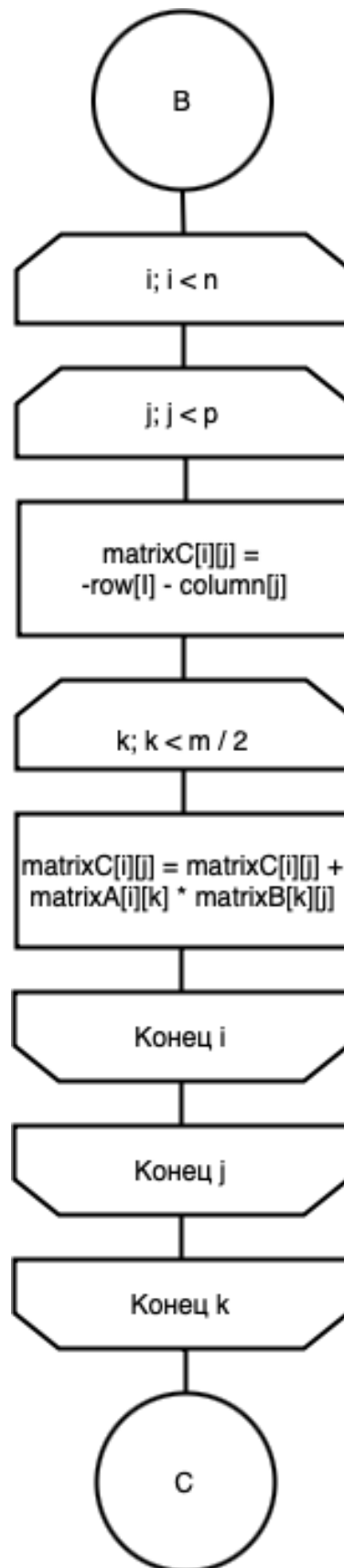


Рисунок 2.4 – Схема алгоритма Виноградова

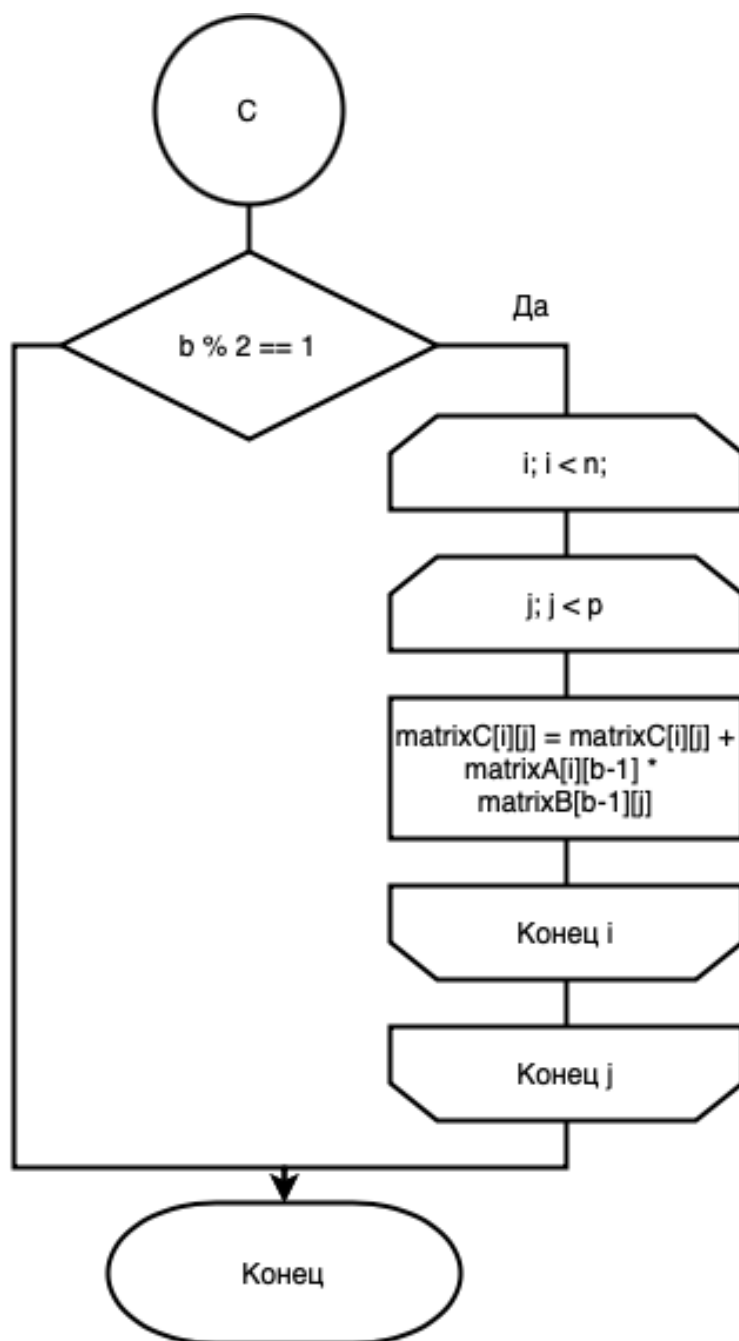


Рисунок 2.5 – Схема алгоритма Виноградова

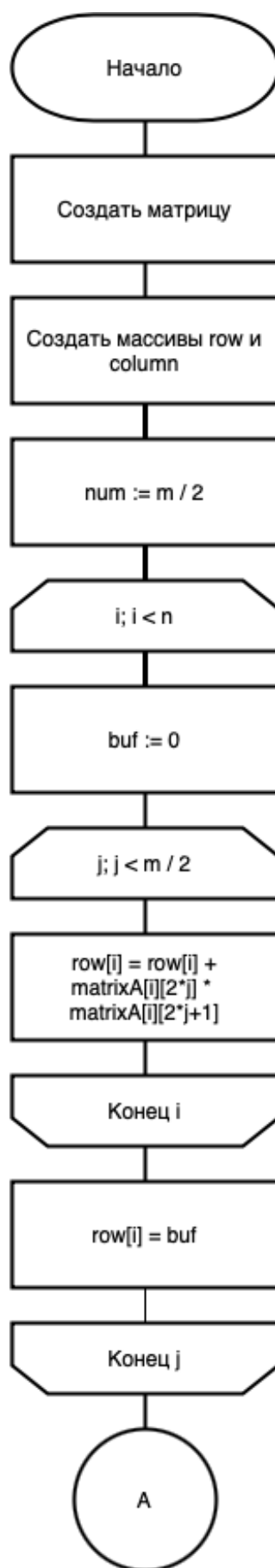


Рисунок 2.6 – Схема оптимизированного алгоритма Виноградова.  
Вычисление сумм произведений пар соседних элементов строк матрицы

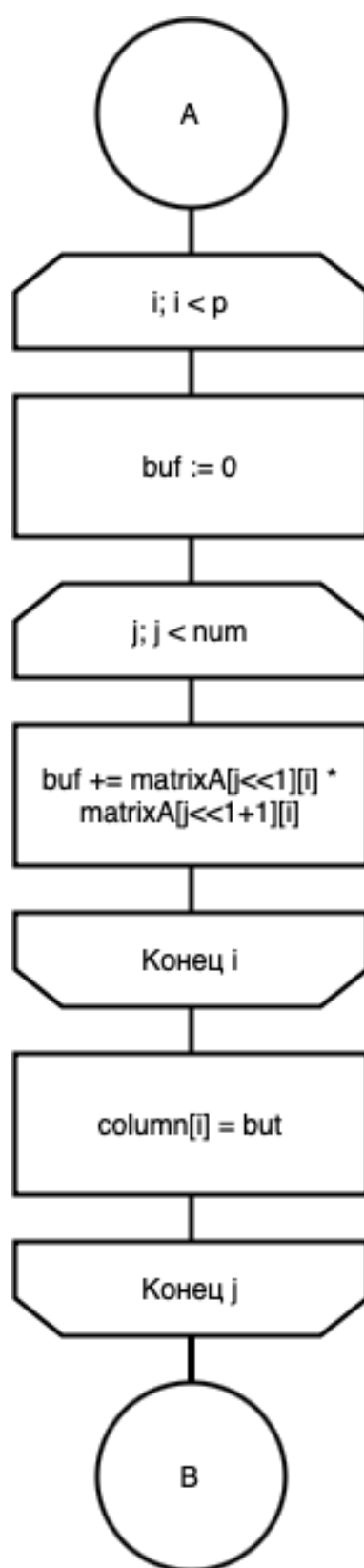


Рисунок 2.7 – Схема оптимизированного алгоритма Виноградова.  
Вычисление сумм произведений пар соседних элементов столбцов матрицы

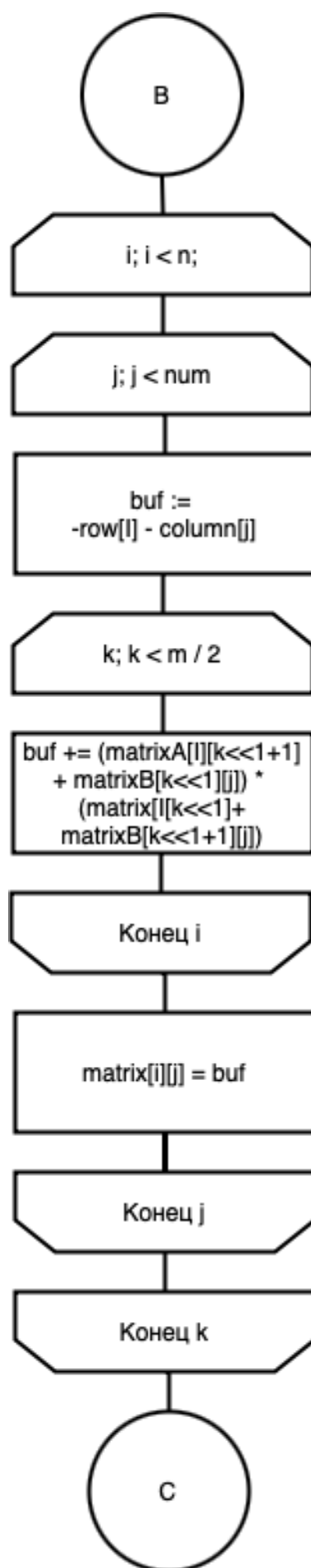


Рисунок 2.8 – Схема оптимизированного алгоритма Виноградова

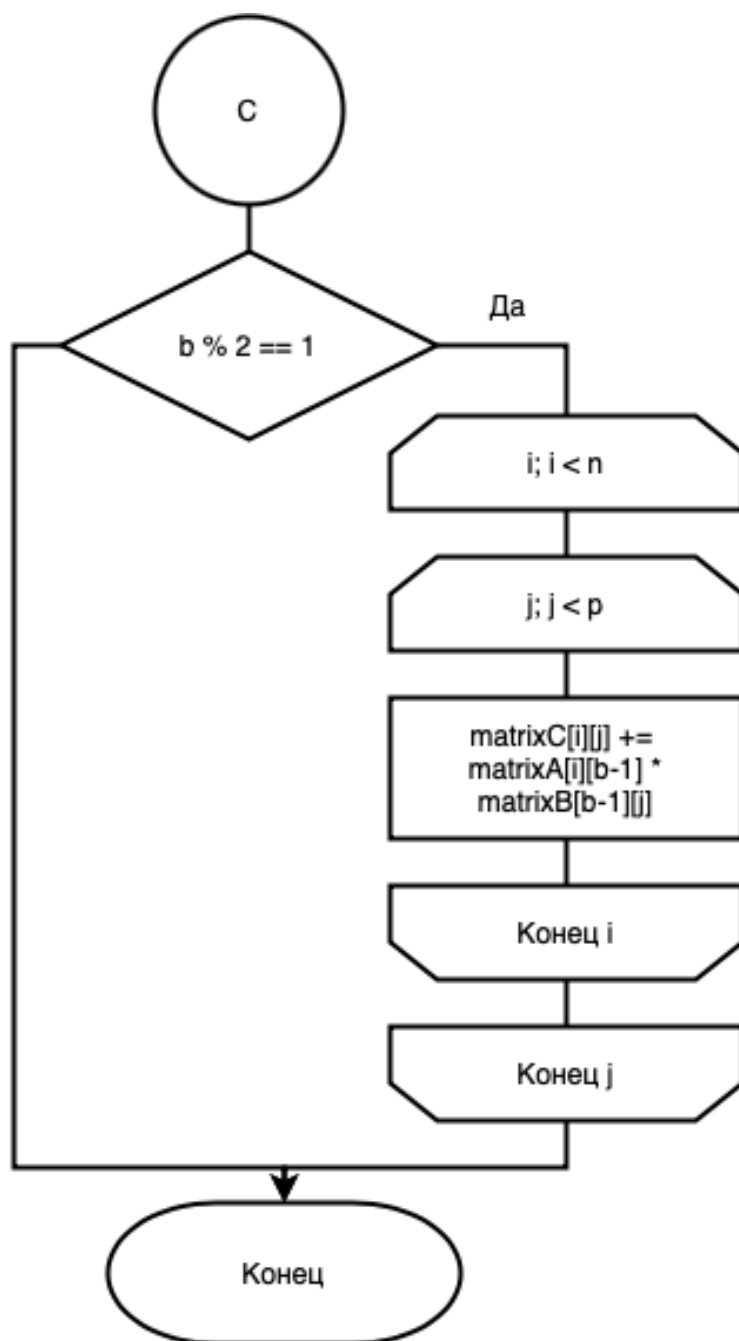


Рисунок 2.9 – Схема оптимизированного алгоритма Виноградова

## 2.2 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма сортиров

1) Трудоемкость базовых операций имеет:

— равную 1:

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

— равную 2:

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

2) Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} \min(f_1, f_2), & \text{лучший случай} \\ \max(f_1, f_2), & \text{худший случай} \end{cases} \quad (2.3)$$

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \quad (2.4)$$

4) Трудоемкость передачи параметра в функции и возврат из функции равны 0.



## 2.3 Оценка трудоемкости алгоритмов

### 2.3.1 Стандартный алгоритм

Трудоемкость стандартного алгоритма умножения матриц считается следующим образом:

- трудоемкость цикла А вычисляется по формуле 2.5;

$$f_A = 1 + 1 + N(f_B + 1 + 1) = 2 + N(2 + f_B) \quad (2.5)$$

- трудоемкость цикла В вычисляется по формуле 2.6;

$$f_B = 1 + 1 + M(f_C + 1 + 1) = 2 + M(2 + f_C) \quad (2.6)$$

- трудоемкость цикла С вычисляется по формуле 2.7;

$$f_C = 1 + 1 + P(9 + 1 + 1) = 2 + 11P \quad (2.7)$$

Кроме циклов в стандартном алгоритме нет действий, поэтому трудоемкость алгоритма равна трудоемкости внешнего цикла А и равна 2.8:

$$f_{standard} = 2 + N(2 + 2 + M(2 + 2 + 11P)) = 11NMP + 4MN + 4N + 2 \quad (2.8)$$

### 2.3.2 Алгоритм Виноградова

Трудоемкость алгоритма умножения матриц Винограда считается следующим образом:

- трудоемкость заполнения массива `row` вычисляется по формуле

$$\begin{aligned} f_{row} = f_A &= 2 + N(2 + f_B) = \\ &= 2 + N(2 + 1 + 3 + \frac{P}{2}(3 + 1 + 15)) = \\ &= 2 + N(6 + \frac{19P}{2}) = 9.5PN + 6N + 2 \quad (2.9) \end{aligned}$$

- трудоемкость заполнения массива `column` вычисляется по формуле

$$\begin{aligned} f_{column} = f_A &= 2 + M(2 + f_B) = \\ &= 2 + M(2 + 1 + 3 + \frac{P}{2}(1 + 3 + 15)) = \\ &= 2 + M(6 + \frac{19P}{2}) = 9.5MP + 6M + 2 \quad (2.10) \end{aligned}$$

- трудоемкость основной части алгоритма равна

$$\begin{aligned} f_{main} = f_A &= 2 + N(2 + f_B) = \\ &= 2 + N(2 + 2 + M(2 + 7 + f_c)) = \\ &= 2 + N(4 + M(9 + 4 + \frac{P}{2}(4 + 28))) = \\ &= 16NMP + 13NM + 4N + 2 \quad (2.11) \end{aligned}$$

— трудоемкость худшего случая при  $P$  - нечетная равна

$$\begin{aligned}
 f_{neg} = f_A &= 3 + 2 + N(2 + f_C) = \\
 &= 5 + N(2 + 2 + M(2 + 14)) = \\
 &= 2 + N(4 + 16M) = \\
 &= 16MN + 4N + 5 \quad (2.12)
 \end{aligned}$$

Таким образом, трудоемкость алгоритма Винограда равна 2.19 для лучшего случая, 2.20 – для худшего.

$$\begin{aligned}
 f_{Winograd}^{\wedge} &= 9.5NP + 6N + 2 + 9.5MP + 6M + 2 + 16NMP + 13NM + 4N + 2 = \\
 &= 16NMP + 13NM + 9.5MP + 9.5NP + 10N + 6M + 6 \quad (2.13)
 \end{aligned}$$

$$\begin{aligned}
 f_{Winograd}^{\vee} &= 9.5NP + 6N + 2 + 9.5MP + 6M + 2 + 16NMP + \\
 &\quad + 13NM + 4N + 2 + 16MN + 4N + 5 = \\
 &= 16NMP + 29NM + 9.5NP + 9.5MP + 14N + 6M + 11 \quad (2.14)
 \end{aligned}$$

## 2.4 Оптимизация алгоритма Винограда

Как видно по вычисленным трудоемкостям алгоритмов (формулы 2.8 и 2.19), трудоемкость алгоритма умножения матриц не уменьшилась, а даже увеличилась, связано это с тем, что с вынесением предварительных вычислений возросло количество таких дорогостоящих операций, как умножение и деление. Чтобы получить выигрыш от предварительных вычислений, необходимо оптимизировать алгоритм, для чего используем следующие типы оптимизации:

- большинство производимых умножений и делений происходит на 2, поэтому заменим их на побитовый сдвиг влево и вправо соответственно;
- заменим выражения вида  $x = x + y$  на выражения вида  $x += y$ , а

выражения вида  $x = x - y$  на выражения вида  $x -= y$ , избавившись худшем случае от одной операции при каждой замене;

- добавим буфер, чтобы постоянно не обращаться к элементам матрицы по индексам;
- $p / 2$  при сравнении и  $b - 1$  при индексации заменим переменными, полученными до циклов.

### 2.4.1 Оптимизированный алгоритм Виноградова

Трудоемкость оптимизированного алгоритма умножения матриц Винограда считается следующим образом:

- трудоемкость заполнения массива `row` вычисляется по формуле

$$\begin{aligned} f_{row} = f_A &= 2 + N(2 + f_B) = \\ &= 2 + N(2 + 1 + 2 + 2 + \frac{P}{2}(2 + 10)) = \\ &= 2 + N(7 + \frac{12P}{2}) = 6NP + 7N + 2 \quad (2.15) \end{aligned}$$

- трудоемкость заполнения массива `row` вычисляется по формуле

$$\begin{aligned} f_{column} = f_A &= 2 + N(2 + f_B) = \\ &= 2 + N(2 + 1 + 2 + 2 + \frac{P}{2}(2 + 10)) = \\ &= 2 + N(7 + \frac{12P}{2}) = 6MP + 7M + 2 \quad (2.16) \end{aligned}$$

- трудоемкость основной части алгоритма равна

$$\begin{aligned} f_{main} = f_A &= 2 + N(2 + f_B) = \\ &= 2 + N(2 + 2 + M(2 + 10 + f_c)) = \\ &= 2 + N(4 + M(12 + \frac{P}{2}(22))) = \\ &= 11NMP + 12NM + 4N + 2 \quad (2.17) \end{aligned}$$

— трудоемкость худшего случая при  $P$  - нечетная равна

$$\begin{aligned}
 f_{neg} = f_A &= 3 + 2 + 2 + N(2 + f_C) = \\
 &= 7 + N(2 + 2 + M(2 + 9)) = \\
 &= 7 + N(4 + 11M) = \\
 &= 11MN + 4N + 7 \quad (2.18)
 \end{aligned}$$

Таким образом, трудоемкость алгоритма Винограда равна 2.19 для лучшего случая, 2.20 – для худшего.

$$\begin{aligned}
 f_{Winograd}^{\wedge} &= 6NP + 7N + 2 + 6MP + 7M + 2 + \\
 &+ 11NMP + 12NM + 4N + 2 = \\
 &= 11NMP + 12NM + 6MP + 6NP + 11N + 7M + 6 \quad (2.19)
 \end{aligned}$$

$$\begin{aligned}
 f_{Winograd}^{\vee} &= 6NP + 7N + 2 + 6MP + 7M + 2 + \\
 &+ 11NMP + 12NM + 4N + 2 + 11MN + 4N + 7 = \\
 &= 11NMP + 23NM + 6MP + 6NP + 11N + 7M + 13 \quad (2.20)
 \end{aligned}$$

## 2.5 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

- количество строк — целое число;
- количество столбцов — целое число;
- матрица — двумерный список целых чисел

## 2.6 Вывод

В данном разделе были разработаны алгоритмы умножения матриц: стандартный и Винограда, – также была произведена оценка трудоемкостей алгоритмов и оптимизация алгоритма Винограда. Для дальнейшей проверки правильности работы программы были выделены классы эквивалентности тестов.

## 3 Технологическая часть

В данном разделе описаны требования к программному обеспечению, средства реализации, приведены листинги кода и данные, на которых будет проводиться тестирование.

Программа должна предоставлять следующие возможности:

- выбор режима работы: для единичного эксперимента и замер времени работы алгоритмов умножения матриц;
- в режиме единичного эксперимента ввод размеров и содержимого каждой матрицы и вывод полученного разными алгоритмами произведений;
- в режиме замера времени происходит замер времени при различных размерах матриц и построение графиков по полученным данным.

### 3.1 Средства реализации

Для реализации данной лабораторной работы Go, так как имеет простой синтаксис, большую библиотеку.

Замеры времени проводились при помощи функции `getThreadCpuTimeNs` написанной на C, подключенной с помощью CGO.

### 3.2 Реализация алгоритмов

В данном подразделе представлены листинги кода ранее описанных алгоритмов:

- стандартный алгоритм умножения матриц (листинг 3.1);
- алгоритм Винограда (листинги 3.2);
- оптимизированный алгоритм Винограда (листинги ??-3.3).

Листинг 3.1 – Реализация стандартного алгоритма умножения матриц

```

1 func MulMatrixA(matrixA , matrixB [][]int , n, m, p int) [][]int {
2     matrixC := make([][]int , n)
3     for i := 0; i < n; i++ {
4         matrixC[i] = make([]int , m)
5         for j := 0; j < m; j++ {
6             for k := 0; k < p; k++ {
7                 matrixC[i][j] = matrixC[i][j] + matrixA[i][k] *
                        matrixB[k][j]
8             }
9         }
10    }
11
12    return matrixC
13 }

```

Листинг 3.2 – Функция нахождения расстояния Левенштейна-Дамерау с помощью рекурсии

```

1 func MulVinogradovA(matrixA , matrixB [][]int , a, b, c int)
   [][]int {
2     row := make([]int , a)
3     column := make([]int , c)
4     matrixC := make([][]int , a)
5
6     for i := 0; i < a; i++ {
7         matrixC[i] = make([]int , c)
8         for j := 0; j < b / 2 ; j++ {
9             row[i] = row[i] + matrixA[i][2 * j] * matrixA[i][2
                    * j + 1]
10        }
11    }
12
13    for i := 0; i < c; i++ {
14        for j := 0; j < b / 2; j++ {
15            column[i] = column[i] + matrixB[2 * j][i]*matrixB[2
                    * j + 1][i]
16        }
17    }
18
19    for i := 0; i < a; i++ {

```



```

20     for j := 0; j < c; j++ {
21         matrixC[i][j] = -row[i] - column[j]
22         for k := 0; k < b / 2; k++ {
23             matrixC[i][j] = matrixC[i][j] + (matrixA[i][2 *
24                 k + 1] + matrixB[2 * k][j]) *
25                 (matrixA[i][2 * k] + matrixB[2 * k + 1][j])
26         }
27     }
28
29     if (b % 2 == 1) {
30         for i := 0; i < a; i++ {
31             for j := 0; j < c; j++ {
32                 matrixC[i][j] = matrixC[i][j] + matrixA[i][b -
33                     1] * matrixB[b - 1][j]
34             }
35         }
36
37     return matrixC
38 }

```

Листинг 3.3 – Функция нахождения расстояния Левенштейна-Дамерау с помощью рекурсии

```

1 func MulVinogradovB(matrixA, matrixB [][]int, a, b, c int)
  [][]int {
2     row := make([]int, a)
3     column := make([]int, c)
4     matrixC := make([][]int, a)
5
6     num := b / 2
7
8     for i := 0; i < a; i++ {
9         matrixC[i] = make([]int, c)
10        buf := 0
11        for j := 0; j < num; j++ {
12            buf += matrixA[i][j << 1] * matrixA[i][j << 1 + 1]
13        }
14        row[i] = buf
15    }
16

```

```

17   for i := 0; i < c; i++ {
18       buf := 0
19       for j := 0; j < num; j++ {
20           buf += matrixB[j << 1][i] * matrixB[j << 1 + 1][i]
21       }
22       column[i] = buf
23   }
24
25   for i := 0; i < a; i++ {
26       for j := 0; j < c; j++ {
27           buf := -row[i] - column[j]
28           for k := 0; k < num; k++ {
29               buf += (matrixA[i][k << 1 + 1] + matrixB[k <<
30                   1][j]) *
31                   (matrixA[i][k << 1] + matrixB[k << 1 +
32                       1][j])
33           }
34           matrixC[i][j] = buf
35       }
36   }
37
38   if (b % 2 == 1) {
39       l := b - 1
40       for i := 0; i < a; i++ {
41           for j := 0; j < c; j++ {
42               matrixC[i][j] += matrixA[i][l] * matrixB[l][j]
43           }
44       }
45   }
46   return matrixC

```

### 3.3 Вывод

В данном разделе были реализованы алгоритмы умножения матриц: стандартный, Винограда и оптимизированный Винограда. Также были написаны тесты для каждого класса эквивалентности, описанного в конструкторском разделе.

## 4 Исследовательская часть

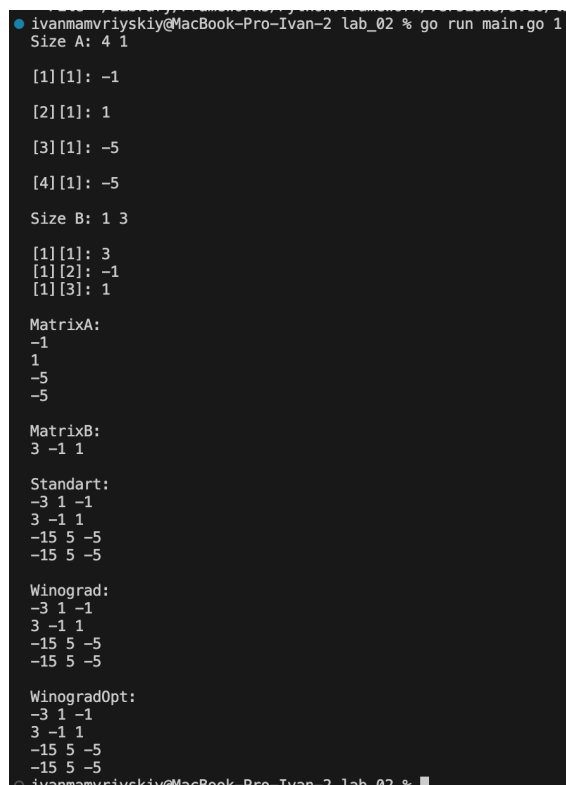
### 4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

- Процессор: 2 GHz 4-ядерный процессор Intel Core i5.
- Оперативная память: 16 ГБайт.
- Операционная система: macOS Ventura 13.5.2.

### 4.2 Прмиер работы

Демонстрация работы программы приведена на рисунке 4.1



```
ivanmamvriyskiy@MacBook-Pro-Ivan-2 lab_02 % go run main.go 1
Size A: 4 1

[1][1]: -1
[2][1]: 1
[3][1]: -5
[4][1]: -5

Size B: 1 3

[1][1]: 3
[1][2]: -1
[1][3]: 1

MatrixA:
-1
1
-5
-5

MatrixB:
3 -1 1

Standart:
-3 1 -1
3 -1 1
-15 5 -5
-15 5 -5

Winograd:
-3 1 -1
3 -1 1
-15 5 -5
-15 5 -5

WinogradOpt:
-3 1 -1
3 -1 1
-15 5 -5
-15 5 -5
ivanmamvriyskiy@MacBook-Pro-Ivan-2 lab_02 %
```

Рисунок 4.1 – Демонстрация работы программы

## 4.3 Время выполнения алгоритмов

LENGHT	STANDART	WINOGRAD	WINOGRADOPT
10	5500	5900	5700
100	2850000	2172900	2014600
200	18733200	17929700	16733500
300	70848800	63572400	58463000
400	156259600	148010600	134163200
500	446163100	359980800	321733900

Рисунок 4.2 – Результаты замера времени

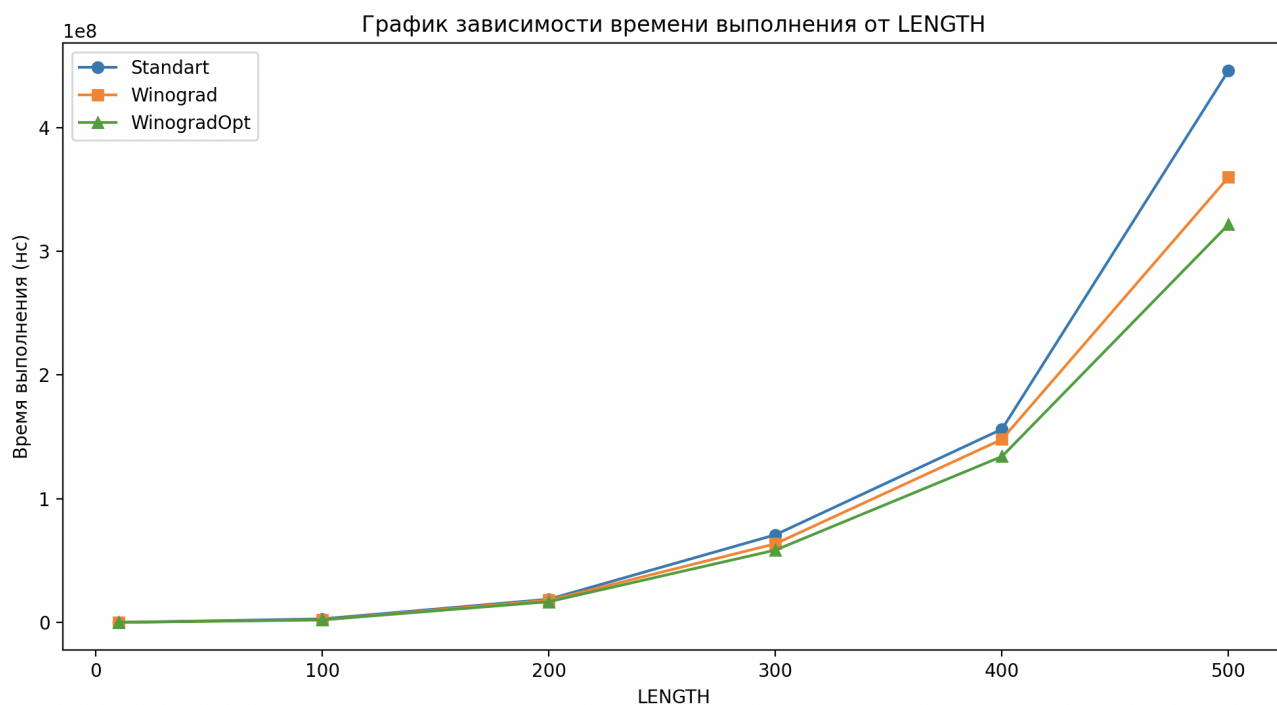


Рисунок 4.3 – График замера времени

## 4.4 Вывод

По результатам эксперимента можно сделать следующие выводы:

- прямая реализация алгоритма Винограда без какой-либо оптимизации работает примерно в 1.2 раза быстрее стандартного алгоритма;
- оптимизированная версия Винограда работает в 1.4 раза быстрее стандартного алгоритма.

Таким образом, для вычислений произведений матриц при необходимости меньшего времени вычислений следует использовать оптимизированный алгоритм Виноградова.

# Заключение

В ходе выполнения лабораторной работы:

- были описаны алгоритмы умножения матриц: стандартный и Винограда;
- была произведена оценка трудоемкости каждого из алгоритмов;
- по схемам алгоритмов и произведенной оценки был сделан вывод о необходимости оптимизации алгоритма Винограда;
- с помощью простых методов оптимизации алгоритмов был получен алгоритм Винограда с меньшей трудоемкостью;
- был реализован каждый из описанных алгоритмов в том числе оптимизированный алгоритм Винограда;
- по экспериментальным данным были сделаны выводы об эффективности по времени каждого из реализованных алгоритмов.