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 «Анализы 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

Введение

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

Целью данной лабораторной работы является исследование стандартного алгоритма умножения матриц и алгоритма Винограда, а также его оптимизация.

Для достижения поставленнной цели необходимо выполнить следующие задачи:

1) Проанализировать данные алгоритмы умножения матриц;
2) Создать программное обеспечение, реализующее следующие алгоритмы:
   - классический алгоритм умножения матриц;
   - алгоритм Винограда;
   - оптимизированный алгоритм Винограда.
3) Оценить трудоемкости алгоритмов умножения матриц;
4) Провести анализ затрат работы программы по процессорному времени;
5) Провести сравнительный анализ между алгоритмами.

1 Аналитическая часть

В данном разделе будут рассмотрены стандартный алгоритм умножения матриц, алгоритм Винограда 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

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

имеющая n строк и p столбцов,

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

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

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

где

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

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

1.2 Алгоритм Винограда для умножения матриц

Из результата умножения двух матриц следует, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. 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 вычисление 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]

1.3 Оптимизированный алгоритм Винограда

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

1) значение $\frac{N}{2}$, используемое как ограничение 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.

1.4 Вывод

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

2 Конструкторская часть

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

2.1 Разработка алгоритмов стандартного и алгоритма Винограда умножения матриц

В данной части будут рассмотрены схемы алгоритмов умножения матриц. На рисунках 2.1 – 2.9 представлены данные алгоритмы.

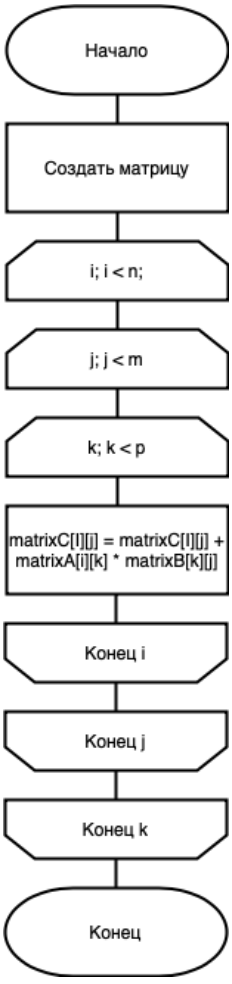

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

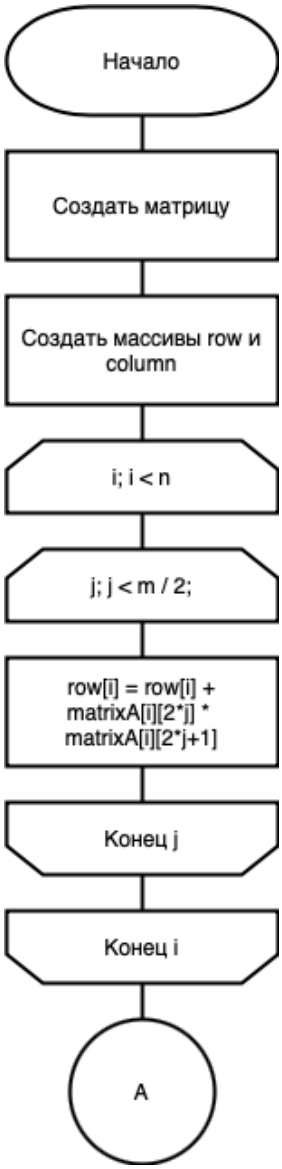

Рисунок 2.2 – Схема алгоритма Винограда. Вычисление сумм произведений пар соседних элементов строк матрицы

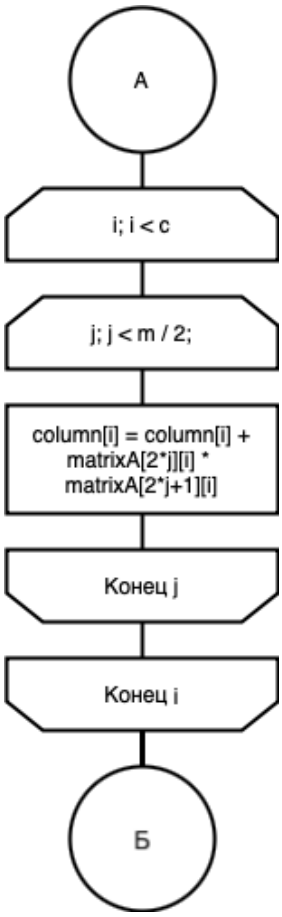

Рисунок 2.3 – Схема алгоритма Винограда. Вычисление сумм произведений пар соседних элементов столбцов матрицы

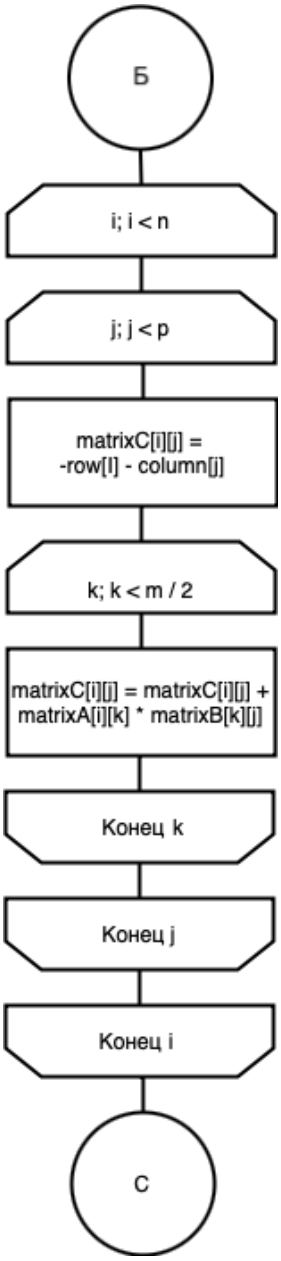

Рисунок 2.4 – Схема алгоритма Винограда

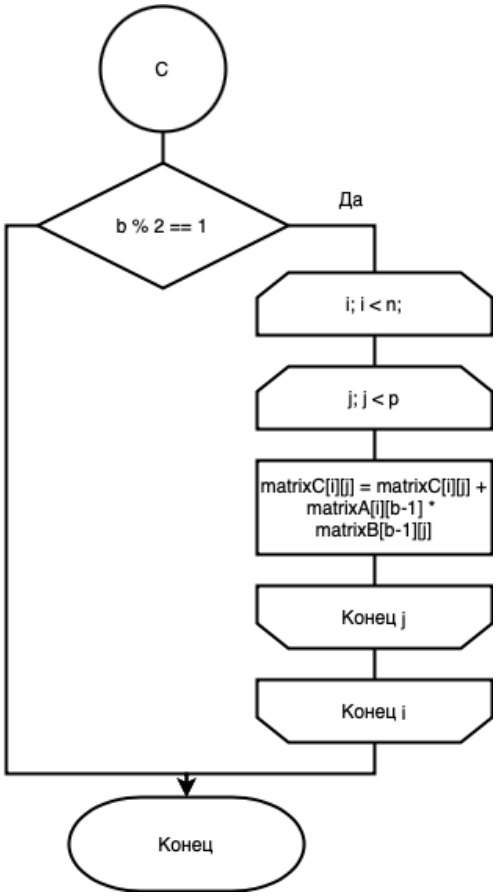

Рисунок 2.5 – Схема алгоритма Винограда

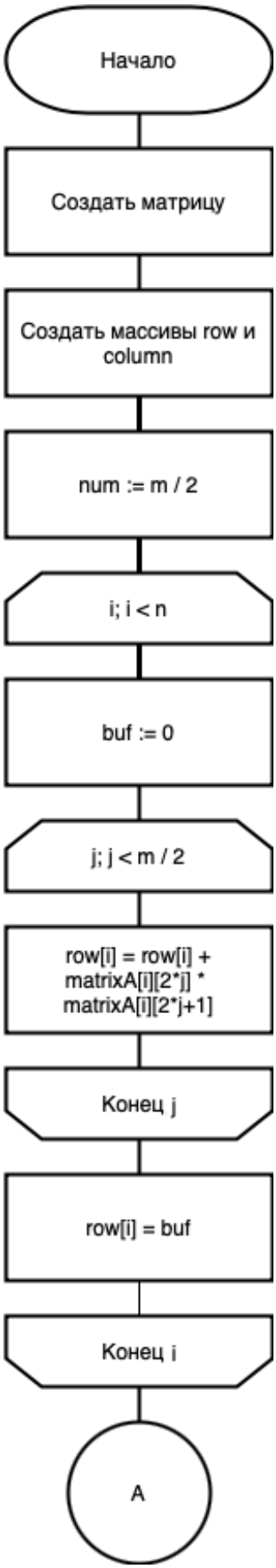

Рисунок 2.6 – Схема оптимизированного алгоритма Винограда. Вычисление сумм произведений пар соседних элементов строк матрицы

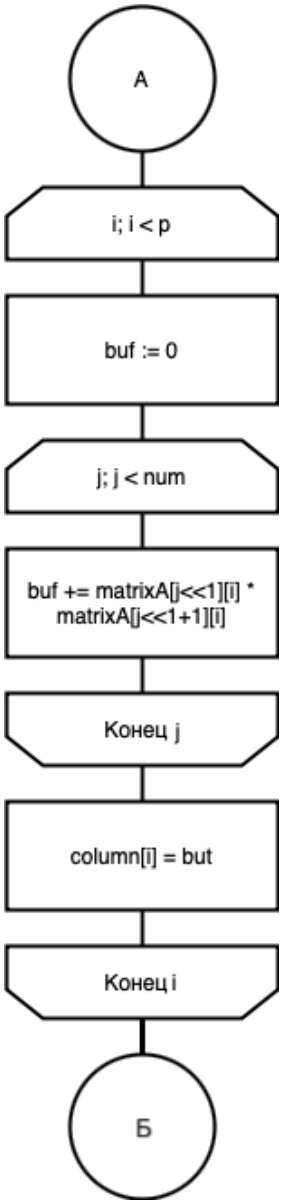

Рисунок 2.7 – Схема оптимизированного алгоритма Винограда. Вычисление сумм произведений пар соседних элементов столбцов матрицы

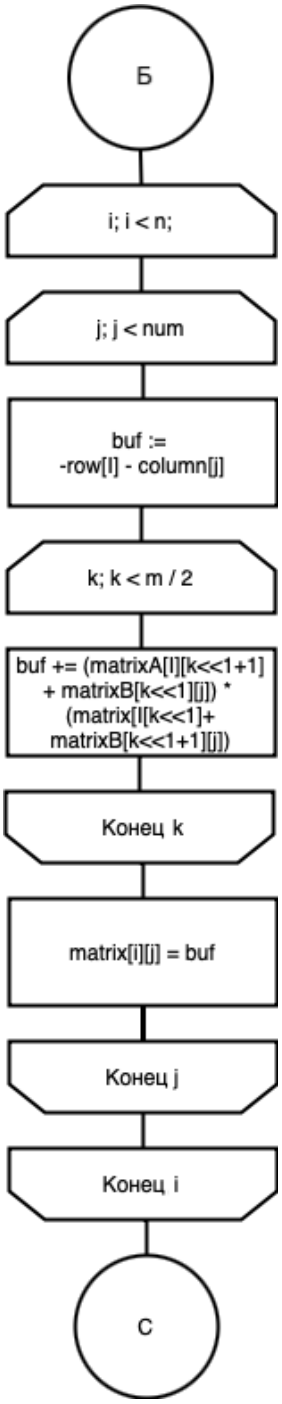

Рисунок 2.8 – Схема оптимизированного алгоритма Винограда

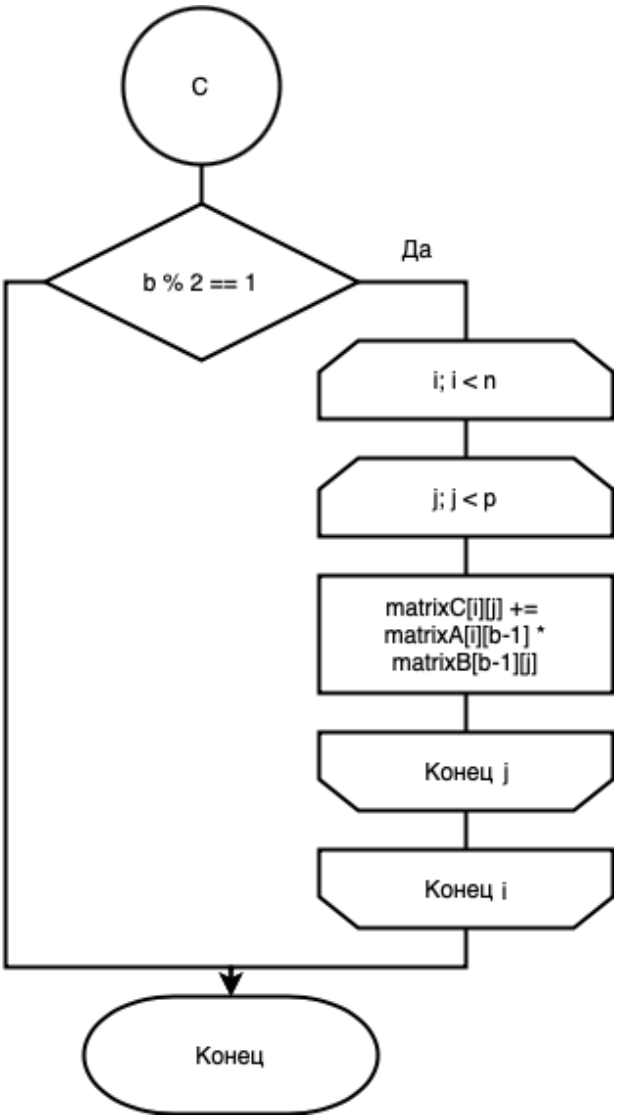

Рисунок 2.9 – Схема оптимизированного алгоритма Винограда

2.2 Модель вычислений для проведения оценки трудоемкости

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

1) Трудоемкость базовых операций имеет:

   — равную 1:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

   — равную 2:

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

2) Трудоемкость условного оператора:

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

3) Трудоемкость цикла:

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

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 A вычисляется по формуле 2.5

$$f_A = 1 + 1 + N(f_B + 1 + 1) = 2 + N(2 + f_B) \tag{2.5}$$

— трудоемкость цикла B вычисляется по формуле 2.6

$$f_B = 1 + 1 + M(f_C + 1 + 1) = 2 + M(2 + f_C) \tag{2.6}$$

— трудоемкость цикла C вычисляется по формуле 2.7

$$f_C = 1 + 1 + P(9 + 1 + 1) = 2 + 11P \tag{2.7}$$

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

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

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма умножения матриц Винограда считается следующим образом:

— трудоемкость заполнения массива row вычисляется по формуле

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

— трудоемкость заполнения массива column вычисляется по формуле

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

— трудоемкость основной части алгоритма равна

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

— трудоемкость худщего случая при P - нечетная равна

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

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

$$f^{\wedge}_{Winograd} = 9.5NP+6N+2+9.5MP+6M+2+16NMP+13NM+4N+2 = \\ = 16NMP + 13NM + 9.5MP + 9.5NP + 10N + 6M + 6 \quad (2.13)$$

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

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма умножения матриц Винограда считается следующим образом:

— трудоемкость заполнения массива row вычисляется по формуле

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

— трудоемкость заполнения массива row вычисляется по формуле

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

— трудоемкость основной части алгоритма равна

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

— трудоемкость худщего случая при P - нечетная равна

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

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

$$f^{\wedge}_{Winograd} = 6NP + 7N + 2 + 6MP + 7M + 2 + \\ + 11NMP + 12NM + 4N + 2 = \\ = 11NMP + 12NM + 6MP + 6NP + 11N + 7M + 6 \quad (2.19)$$

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

2.4 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

2.5 Требования к программному обеспечению

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, вывод результата, полученного разными алгоритмами умножения;

— в режиме замера времени происходит замер времени при различных размерах матриц и построение графиков по полученным данным.

2.6 Вывод

В данном разделе были разработаны алгоритмы умножения матриц: стандартный и Винограда, – также была произведена оценка трудоемкостей алгоритмов и оптимизация алгоритма Винограда. Проведенная оценка трудоемкости показывает, что трудоемкость у оптимизированного алгоритма Винограда в 1.45 раза меньше, чем у простого алгоритма Винограда.

3 Технологическая часть

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

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык GO[2]. Данный выбор обусловлен тем, что данный язык является типизированным, имеет инструменты для тестирования, профилирования и форматирования кода, быстрая скорость компиляции.

Замеры времени проводились при помощи getThreadCpuTimeNs функции, написанной на C, подключенной с помощью CGO.[3]

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 – Реализация алгоритма Винограда умножения матриц

```
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 *
                        k + 1] + matrixB[2 * k][j]) *
24                      (matrixA[i][2 * k] + matrixB[2 * k + 1][j])
25              }
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 -
                        1] * matrixB[b - 1][j]
33              }
34          }
35      }
36
37      return matrixC
38  }
```

Листинг 3.3 – Реализация оптимизированного алгоритма Винограда умножения матриц

```
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 <<
                        1][j]) *
30                      (matrixA[i][k << 1] + matrixB[k << 1 +
                            1][j])
31              }
32              matrixC[i][j] = buf
33          }
34      }
35
36      if (b % 2 == 1) {
37          l := b - 1
38          for i := 0; i < a; i++ {
39              for j := 0; j < c; j++ {
40                  matrixC[i][j] += matrixA[i][l] * matrixB[l][j]
41              }
42          }
43      }
44
45      return matrixC
46  }
```

3.3 Описание тестирования

В таблице 3.1 представлены функциональные тесты для алгоритмов умножения матриц.

Таблица 3.1 – Функциональные тесты

Матрица 1	**Матрица 2**	**Ожидаемый результат**
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & 2 & 3 \\ 1 & -2 & 3 \\ 1 & 2 & -3 \end{pmatrix}$	$\begin{pmatrix} 4 & 4 & 0 \\ 4 & 4 & 0 \\ 4 & 4 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & -3 \\ -2 & -2 \\ -3 & -1 \end{pmatrix}$	$\begin{pmatrix} -14 & -10 \\ -14 & -10 \end{pmatrix}$
$\begin{pmatrix} 3 & -5 \\ 1 & -2 \end{pmatrix}$	$\begin{pmatrix} 2 & -5 \\ 1 & -3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$

Алгоритмы, реализованные в данной лабораторной работе, функциональные тесты прошли успешно.

3.4 Вывод

В данном разделе были реализованы алгоритмы умножения матриц: стандартный, Винограда и оптимизированный Винограда. Также для проверки правильности работы программы были выделены классы эквивалентности тестов.

4 Исследовательская часть

4.1 Технические характеристики

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

— Процессор – 2 ГЦ 4-ядерный процессор Intel Core i5;

— Оперативная память – 16 ГБайт;

— Операционная система – macOS Venura 13.5.2.

4.2 Пример работы

В данном подразделе представлен пример 4.1 работы программы:

Листинг 4.1 – Демонстрация работы алгоритма

```
ivanmamvriyskiy@MacBook-Pro-Ivan-2 main % 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
```

4.3 Время выполнения алгоритмов

В таблице 4.1 представлены замеры времени для алгоритмов умножения матриц при четных размерностях. График зависимостей времени выполнения от размерности показан на графике 4.1.

Таблица 4.1 – Замеры времени выполнения алгоритмов при четных размерностях(нс).

Размерность	Стандартный	Виноград	Виноград Оптимизир
10	4	6	5
100	4 131	2 506	2 169
200	20 429	18 960	19 138
300	72 264	64 371	63 638
400	195 449	189 089	178 892
500	503 049	387 605	329 304
600	684 740	680 307	665 368
700	1 455 149	1 773 952	1 269 319
800	1 615 890	1 418 696	1 517 012
900	3 383 536	3 272 015	2 740 418
1000	6 225 405	5 797 701	5 461 040

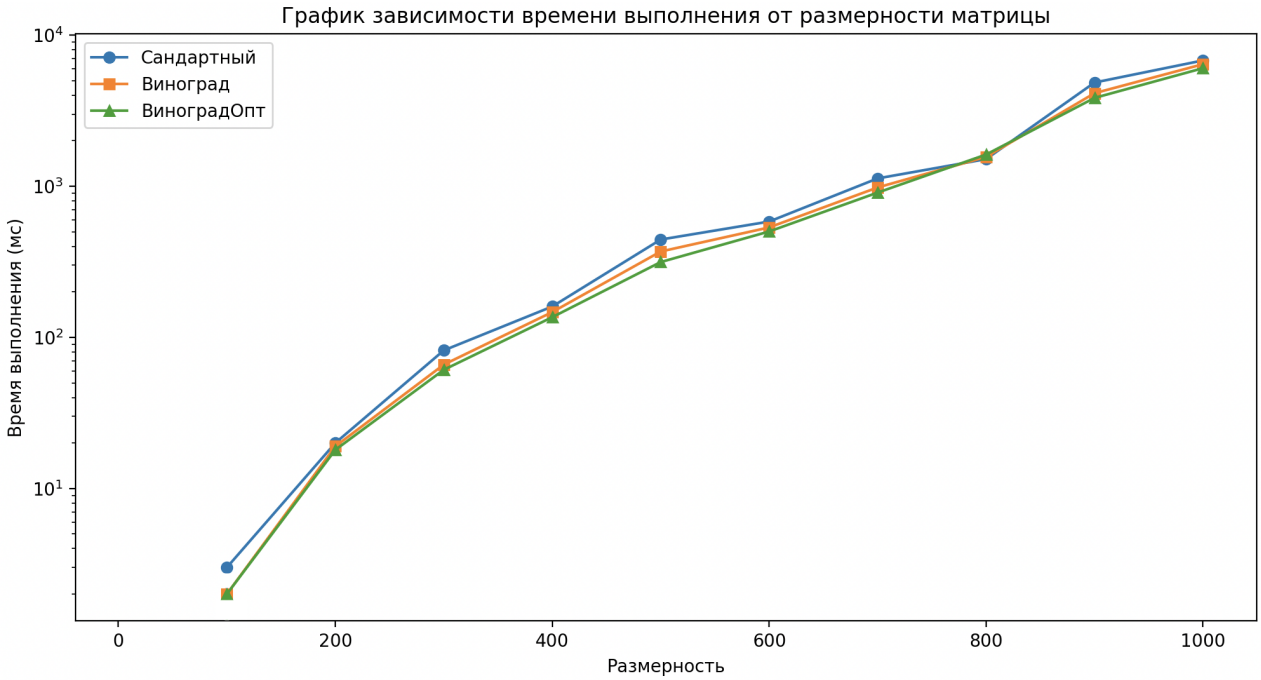

Рисунок 4.1 – График замера времени при четных размерностях

В таблице 4.2 представлены замеры времени для алгоритмов умножения матриц при нечетных размерностях. График зависимостей времени выполнения от размерности показан на графике 4.2.

Таблица 4.2 – Замеры времени выполнения алгоритмов при нечетных размерностях(нс).

Размерность	Стандартный	Виноград	Виноград Оптимизир
11	6	7	6
101	2 819	2 525	2 407
201	20 040	18 278	17 336
301	69 682	62 652	57 036
401	158 474	146 123	135 236
501	439 145	356 716	313 788
601	575 588	527 767	494 750
701	1 128 277	959 669	898 226
801	1 473 640	1 381 828	1 279 278
901	4 678 927	4 444 102	4 153 936
1001	7 007 630	6 864 435	6 789 342

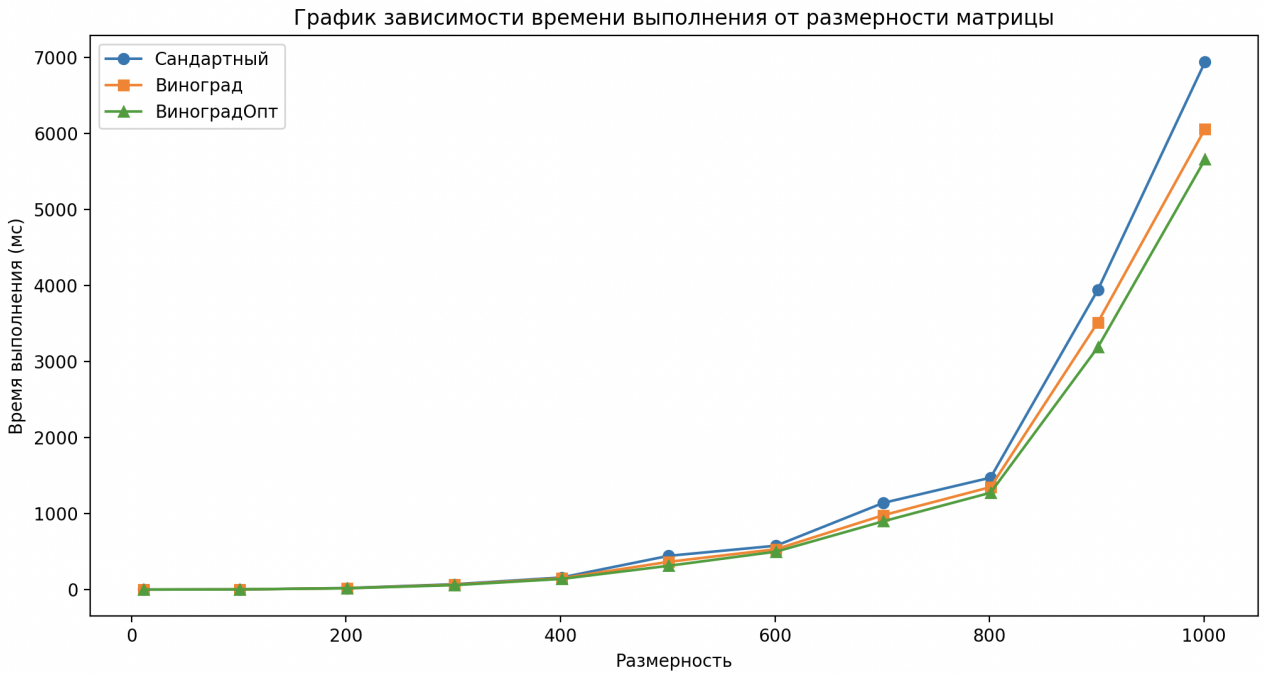

Рисунок 4.2 – График замера времени при нечетных размерностях

4.4 Замеры памяти при выполнении алгоритмов

В таблице 4.3 представлены результаты замера памяти алгоритмов умножения матриц при четной размерности, в таблице 4.4 при нечетной размерности.

Таблица 4.3 – Замеры памяти при выполнении алгоритмов для четных размерностец матриц

Длина	Стандартный	Виноград	Виноград Опт
10	1 040	1 200	1 200
100	92 288	94 080	94 080
200	363 264	366 848	366 848
300	814 592	819 968	819 968
400	1 289 728	1 296 128	1 296 128
500	2 060 288	2 068 480	2 068 576
600	2 937 032	2 944 512	2 944 512
700	4 319 232	4 331 616	4 331 520
800	5 242 880	5 255 936	5 255 936
900	7 394 560	7 411 040	7 410 944
1000	8 216 576	8 232 960	8 232 960

Таблица 4.4 – Замеры памяти при выполнении алгоритмов для нечетных размерностец матриц(байт).

Длина	Стандартный	Виноград	Виноград Опт
11	1 344	1 536	1 536
101	93 184	94 976	94 976
201	365 056	368 640	368 640
301	817 288	822 656	822 664
401	1 395 584	1 402 496	1 402 496
501	2 064 384	2 072 576	2 072 576
601	2 939 648	2 949 376	2 949 376
701	4 325 376	4 337 664	4 337 664
801	5 249 408	5 262 464	5 262 464
901	7 402 752	7 419 136	7 419 136
1001	8 224 768	8 241 152	8 241 152

4.5 Вывод

По результатам эксперимента можно сделать следующие выводы:

— прямая реализация алгоритма Винограда без какой-либо опитимизации работает примерно в 1.2 раза быстрее стандартного алгоритма;

— оптимизированная версия Винограда работает в 1.4 раза быстрее стандартного алгоритма;

— при нечетных размерностях алгоритм Винограда работает дольше, чем при четных размерностях;

— алгоритм Винограда тратит больше памяти, чем стандартный алгоритм, так как происходит предварительная обработка.

Таким образом, для вычисления произведения матриц при необходимости меньшего времени вычисления следует использовать оптимизированный алгоритм Винограда.

Заключение

В результате исследования было определено, что стандартный алгоритм умножения матриц проигрывает по времени алгоритму Винограда примерно в 1.2 раза из-за того, что в алгоримте Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций - операций, умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но наилучшие показатели как по времени, так и по трудоемкости демонстрирует оптимизированный алгоритм Винограда. Он работает примерно в 1.4 раза быстрее и имеет трудоемкость, меньшую примерно в 1.25 раза, благодаря замене операции равно и плюс на операцию плюс-равно, введению буфера, а также замене операции умножения на сдвиг. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда.

Однако по памяти алгоритм Винограда проигрывает стандартному алгоритму, так как происходит выделение памяти под хранение дополнительных данных, полученных при предварительной обработке.

Цель, которая заключается в исследовании стандартного алгоритма умножения матриц и алгоритма Винограда, а также его оптимизация, выполнена, также в ходе выполнения лабораторной работы были решены следующие задачи:

- были описаны алгоритмы умножения матриц: стандартный и Винограда;
- создано программное обеспечение, реализующее следующие алгоритмы:
  - классический алгоритм умножения матриц;
  - алгоритм Винограда;
  - оптимизированный алгоритм Винограда.
- была произведена оценка трудоемкости каждого из алгоритмов;
- по экспериментальным данным были сделаны выводы об эффективности по времени и памяти реализованных алгоритмов.

Список используемых источников

1. Макконелл Дж. Основы современных алгоритмов. 2-е доп. изд. М.: Техносфера, 2004. с. 368.

2. Документация Go [Электронный ресурс]. Режим доступа: `https://go.dev` (дата обращения: 20.09.2023).

3. Документация CGo [Электронный ресурс]. Режим доступа: `https://pkg.go.dev/cmd/cgo` (дата обращения: 20.09.2023).