МГТУ им. Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Алгоритмы умножения матриц

Работу выполнила: Саркисов Артём, ИУ7-53Б

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

Москва, 2019

Оглавление

Введение **3**

1 Аналитическая часть **4**
1.1 Алгоритм Винограда 4
1.2 Вывод 5

2 Конструкторская часть **6**
2.1 Схемы алгоритмов 6
2.2 Трудоемкость алгоритмов 10
2.2.1 Трудоемкость первичной проверки 10
2.2.2 Классический алгоритм 10
2.2.3 Алгоритм Винограда 11
2.2.4 Оптимизированный алгоритм Винограда 11
2.3 Вывод 11

3 Технологическая часть **12**
3.1 Выбор ЯП 12
3.2 Описание структуры ПО 12
3.3 Сведения о модулях программы 13
3.4 Листинг кода алгоритмов 13
3.4.1 Оптимизация алгоритма Винограда 17
3.5 Вывод 18

4 Исследовательская часть **19**
4.1 Сравнительный анализ на основе замеров времени работы алгоритмов 19
4.2 Вывод 20

Заключение **21**

Список литературы **21**

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 рассчет сложности алгоритмов, получить навыки в улучшении алгоритмов.

В ходе лабораторной работы предстоит:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- оптимизировать алгоритм Винограда;
- дать теоретическую оценку базового 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 $[m * n]$ 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 $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

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

1.1 Алгоритм Винограда

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

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$.

Их скалярное произведение равно (1.1)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.1}$$

Равенство (1.1) можно переписать в виде (1.2)

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

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.

1.2 Вывод

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

2 | Конструкторская часть

Требования к вводу: На вход подаются две матрицы
Требования к программе:

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

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов.

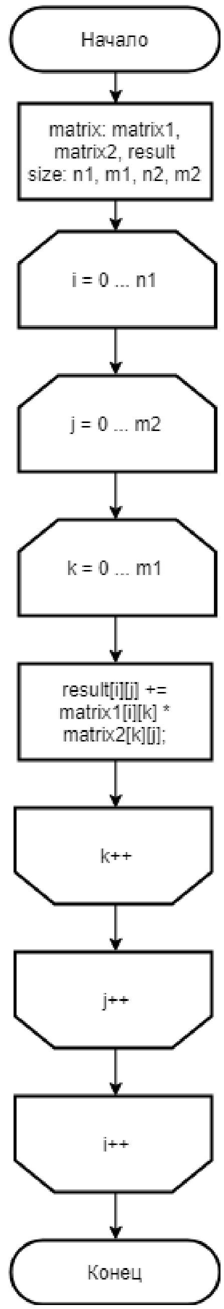

Рис. 2.1: Схема классического алгоритма умножения матриц

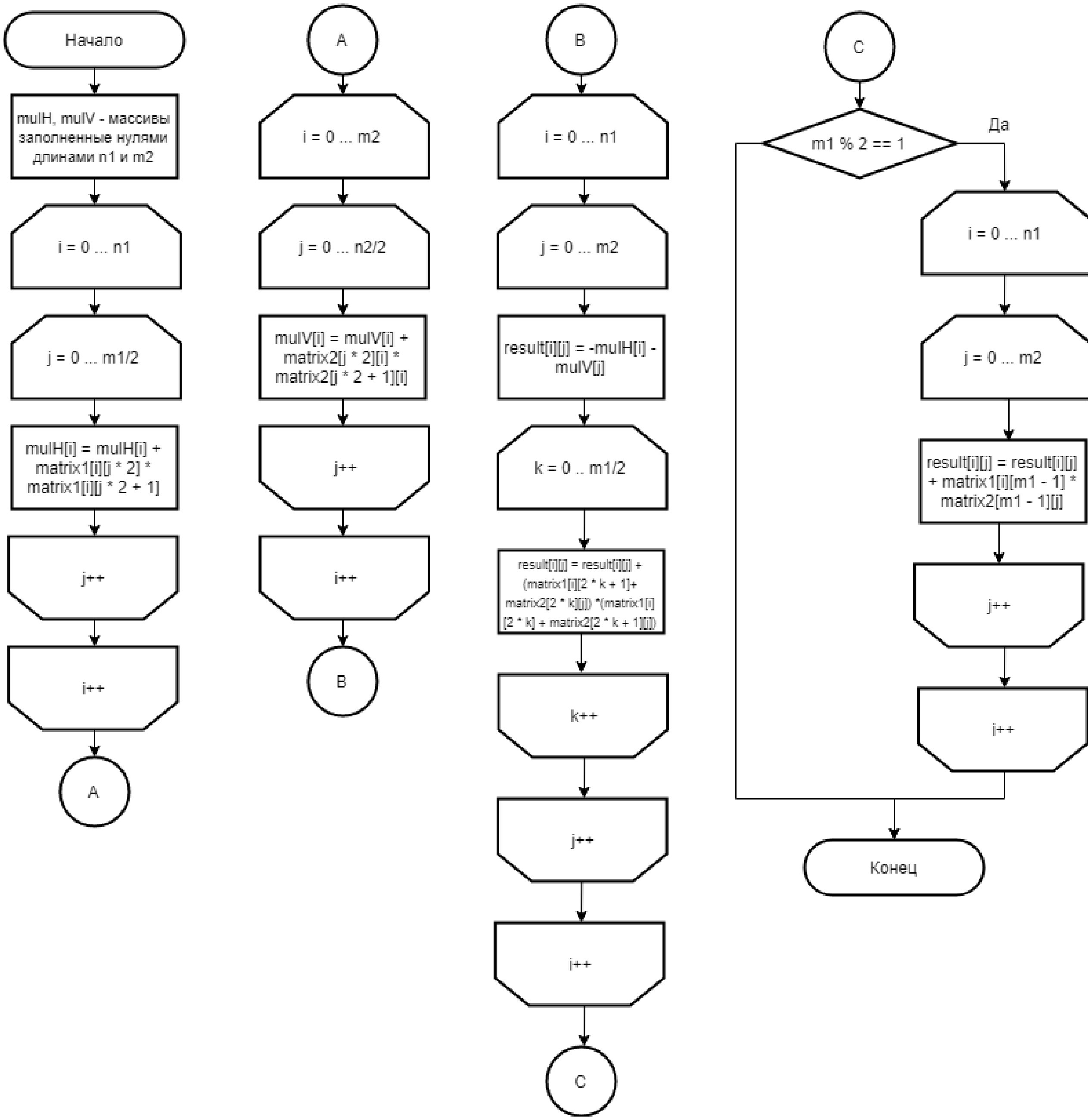

Рис. 2.2: Схема алгоритма Винограда

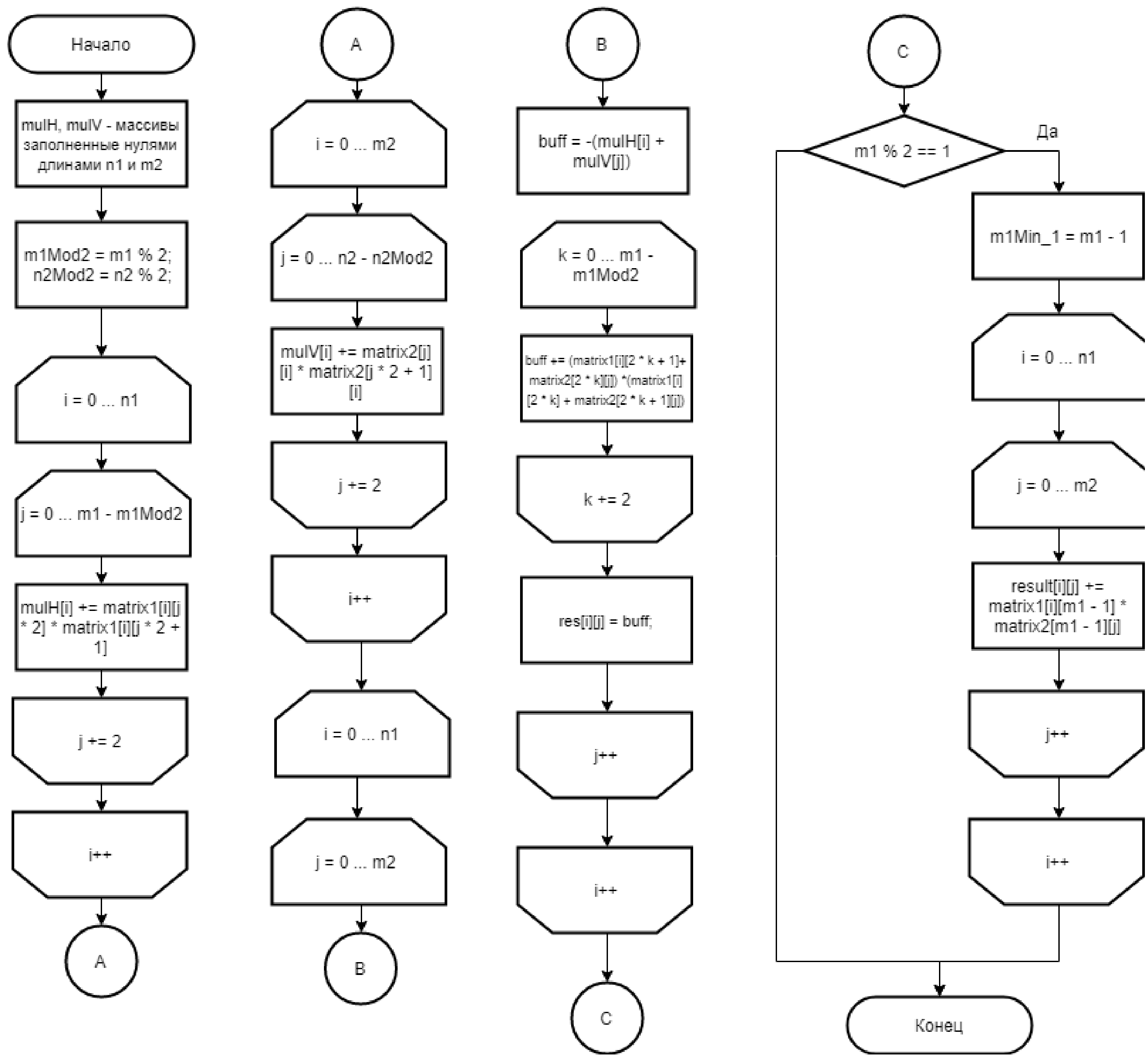

Рис. 2.3: Схема оптимизированного алгоритма Винограда

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса;
- оценка трудоемкости цикла: Fц = init + N*(a + Fтела + post) + a, где a - условие цикла, init - предусловие цикла, post - постусловие цикла;
- стоимость условного перехода применим за 0, стоимость вычисления условия остаётся.

Оценим трудоемкость алгоритмов по коду программы.

2.2.1 Трудоемкость первичной проверки

Рассмотрим трудоемкость первичной проверки на возможность умножения матриц.

Табл. 2.1 Построчная оценка веса

Код	Вес
int n1 = mart1.Length;	2
int n2 = matr2.Length;	2
if (n1 == 0 || n2 == 0) return null;	3
int m1 = mart1[0].Length;	3
int m2 = matr2[0].Length;	3
if (m1 != n2) return null;	1
Итого	14

2.2.2 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма:

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

Подсчет:

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

2.2.3 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: $13n_1m_2m_1 + \frac{15}{2}n_1m_1 + \frac{15}{2}m_2n_2 + 12n_1m_2 + 5n_1 + 5m_2 + 4n_1 + 6 + \begin{bmatrix} 2 & \text{, невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

2.2.4 Оптимизированный алгоритм Винограда

Аналогично Рассмотрим трудоемкость оптимизированого 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: $\begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

Итого: $\frac{17}{2}n_1m_2m_1 + \frac{11}{2}n_1m_1 + \frac{11}{2}m_2n_2 + 9n_1m_2 + 8n_1 + 4m_2 + 6 + \begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель оценки трудоемкости алгоритма, были расчитаны трудоемкости алгоритмов в соответсвии с этой моделью.

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран Golang [3], а средой разработки Visual Studio. Время работы алгоритмов было замерено с помощью класса встроенного в Golang модуля Benchmark.

3.2 Описание структуры ПО

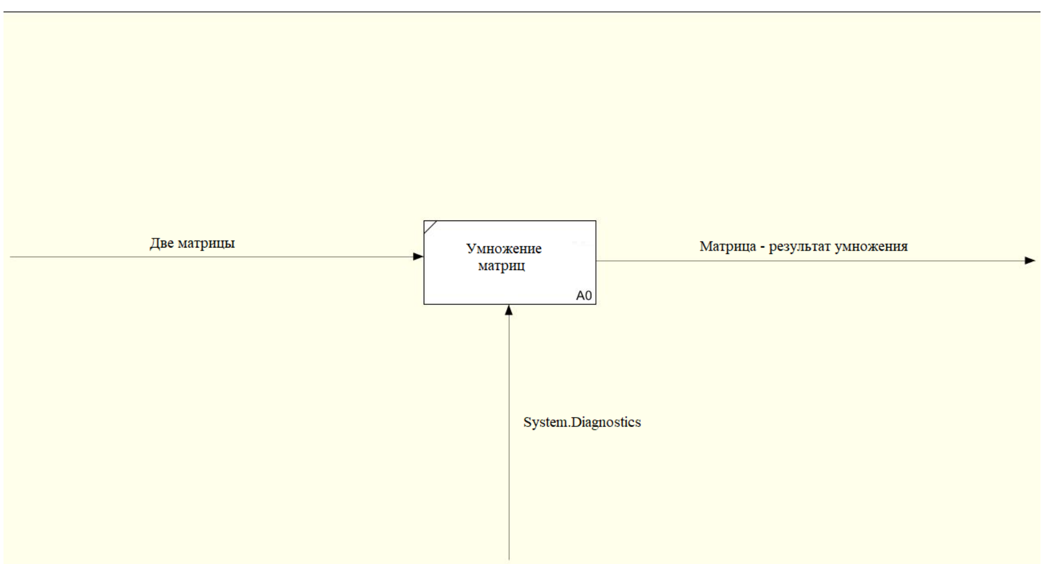

Рис. 3.1: Функциональная схема умножения матриц (IDEF0 диаграмма 1 уровня)

3.3 Сведения о модулях программы

Программа состоит из:

- main.go - главный файл программы, в котором располагается точка входа в программу.
- matrixoper.go - файл, в которм находятся алгоритмы умножения матриц.
- multest.go - файл с встроенным Benchmark для замера процесорного времени выполнения алгоритмов.

3.4 Листинг кода алгоритмов

Листинг 3.1: Стандартный алгоритм умножения матриц

```
func MatrixMultStd(matrix1 [][]int, matrix2 [][]int) [][]
    int {
  var n1 int = len(matrix1)
  var n2 int = len(matrix2)

  if n1 == 0 || n2 == 0 {
    return nil
  }

  var m1 int = len(matrix1[0])
  var m2 int = len(matrix2[0])

  if m1 != n2 {
    return nil
  }

  result := CreateMatrix(n1, m2)

  for i := 0; i < n1; i++ {
    for j := 0; j < m2; j++ {
      for k := 0; k < m1; k++ {
        result[i][j] += matrix1[i][k] * matrix2[k][j]
      }
```

```
    }
  }

  return result
}
```

Листинг 3.2: Алгоритм Винограда

```
func MatrixMultVin(matrix1 [][]int, matrix2 [][]int) [][]
    int {
  var n1 int = len(matrix1)
  var n2 int = len(matrix2)

  if n1 == 0 || n2 == 0 {
    return nil
  }

  var m1 int = len(matrix1[0])
  var m2 int = len(matrix2[0])

  if m1 != n2 {
    return nil
  }

  mulH := make([]int, n1)
  mulV := make([]int, m2)
  result := CreateMatrix(n1, m2)

  for i := 0; i < n1; i++ {
    for j := 0; j < m1/2; j++ {
      mulH[i] += matrix1[i][j*2] * matrix1[i][j*2+1]
    }
  }

  for i := 0; i < m2; i++ {
    for j := 0; j < n2/2; j++ {
      mulV[i] += matrix2[j*2][i] * matrix2[j*2+1][i]
    }
  }

  for i := 0; i < n1; i++ {
```

```
    for j := 0; j < m2; j++ {
      result[i][j] = -mulH[i] - mulV[j]
      for k := 0; k < m1/2; k++ {
        result[i][j] += (matrix1[i][2*k+1] + matrix2[2*k][j]) * (matrix1[i][2*k] + matrix2[2*k+1][j])
      }
    }
  }

  if m1%2 == 1 {
    for i := 0; i < n1; i++ {
      for j := 0; j < m2; j++ {
        result[i][j] += matrix1[i][m1-1] * matrix2[m1-1][j]
      }
    }
  }

  return result
}
```

Листинг 3.3: Оптимизированный алгоритм Винограда

```
func MatrixMultVinOptim(matrix1 [][]int, matrix2 [][]int) [][]int {
  var n1 int = len(matrix1)
  var n2 int = len(matrix2)

  if n1 == 0 || n2 == 0 {
    return nil
  }

  var m1 int = len(matrix1[0])
  var m2 int = len(matrix2[0])

  if m1 != n2 {
    return nil
  }

  mulH := make([]int, n1)
  mulV := make([]int, m2)
  result := CreateMatrix(n1, m2)
```

```

  var m1Mod2 int = m1 % 2
  var n2Mod2 int = n2 % 2

  for i := 0; i < n1; i++ {
    for j := 0; j < m1-m1Mod2; j += 2 {
      mulH[i] += matrix1[i][j] * matrix1[i][j+1]
    }
  }

  for i := 0; i < m2; i++ {
    for j := 0; j < n2-n2Mod2; j += 2 {
      mulV[i] += matrix2[j][i] * matrix2[j+1][i]
    }
  }

  var buff int
  for i := 0; i < n1; i++ {
    for j := 0; j < m2; j++ {
      buff = -mulH[i] - mulV[j]
      for k := 0; k < m1-m1Mod2; k += 2 {
        buff += (matrix1[i][k+1] + matrix2[k][j]) * (matrix1[i][k] + matrix2[k+1][j])
      }
      result[i][j] = buff
    }
  }

  if m1Mod2 == 1 {
    var m1Min1 int = m1 - 1
    for i := 0; i < n1; i++ {
      for j := 0; j < m2; j++ {
        result[i][j] += matrix1[i][m1Min1] * matrix2[m1Min1][j]
      }
    }
  }

  return result
}
```

3.4.1 Оптимизация алгоритма Винограда

В рамках данной лабораторной работы было предложено 3 оптимизации:

1. Избавление от деления в условии цикла;
2. Замена $mulH[i] = mulH[i] + \dots$ на $mulH[i] += \dots$ (аналогично для $mulV[i]$);

Листинг 3.4: Оптимизации алгоритма Винограда №1 и №2

```
var m1Mod2 int = m1 % 2
var n2Mod2 int = n2 % 2

for i := 0; i < n1; i++ {
  for j := 0; j < m1-m1Mod2; j += 2 {
    mulH[i] += matrix1[i][j] * matrix1[i][j+1]
  }
}

for i := 0; i < m2; i++ {
  for j := 0; j < n2-n2Mod2; j += 2 {
    mulV[i] += matrix2[j][i] * matrix2[j+1][i]
  }
}
```

3. Накопление результата в буфер, чтобы не обращаться каждый раз к одной и той же ячейке памяти. Сброс буфера в ячейку матрицы после цикла.

Листинг 3.5: Оптимизации алгоритма Винограда №3

```
  var buff int
for i := 0; i < n1; i++ {
  for j := 0; j < m2; j++ {
    buff = -mulH[i] - mulV[j]
    for k := 0; k < m1-m1Mod2; k += 2 {
      buff += (matrix1[i][k+1] + matrix2[k][j]) * (
          matrix1[i][k] + matrix2[k+1][j])
    }
    result[i][j] = buff
  }
}
```

3.5 Вывод

В данном разделе была рассмотрена структура ПО и листинги кода программы.

4 Исследовательская часть

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов. Первый эксперимент производится для лучшего случая на квадратных матрицах размером от 100 x 100 до 1000 x 1000 с шагом 100. Сравним результаты для разных алгоритмов:

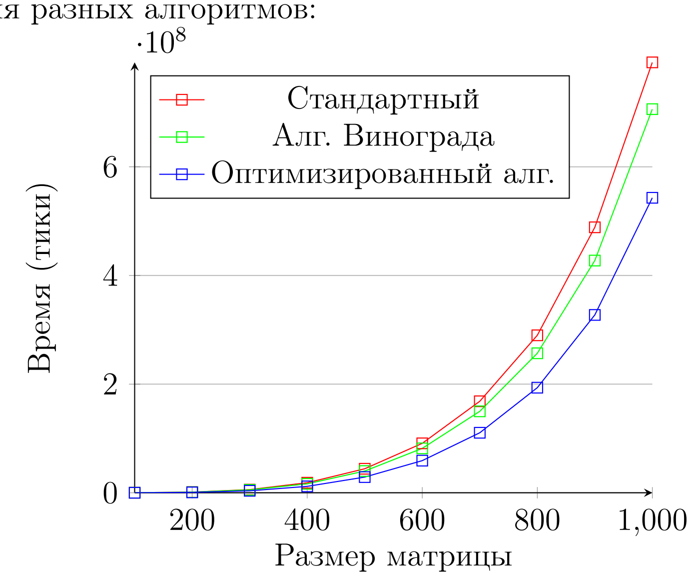

Рис. 4.1: Сравнение времени работы алгоритмов при четном размере матрицы

Второй эксперимент производится для худшего случая, когда поданы квадратные матрицы с нечетными размерами от 101 х 101 до 1001 х 1001 с шагом 100. Сравним результаты для разных алгоритмов:

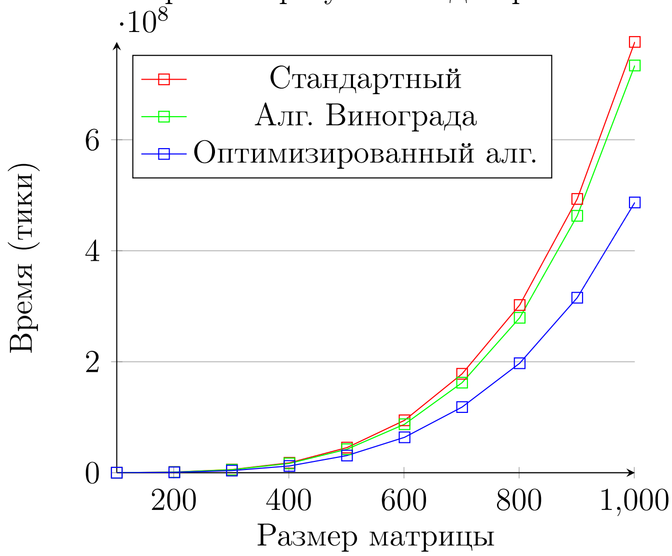

Рис. 4.2: Сравнение времени работы алгоритмов при нечетном размере матрицы

4.2 Вывод

По результатам тестирования все рассматриваемые алгоритмы реализованы правильно. Самым медленным алгоритмом оказался алгоритм классического умножения матриц, а самым быстрым — оптимизированный алгоритм Винограда.

Заключение

В ходе лабораторной работы я изучил алгоритмы умножения матриц: стандартный и Винограда, оптимизировал алгоритм Винограда, дал теоретическую оценку алгоритмов стандартного умножения матриц, Винограда и улучшенного Винограда, реализовал три алгоритма умножения матриц на языке программирования Golang и сравнил эти алгоритмы.

Литература

[1] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[2] Le Gall, F. (2012), "Faster algorithms for rectangular matrix multiplication Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), pp. 514–523

[3] Руководство по языку Golang[Электронный ресурс], - режим доступа: https://golang/