МГТУ им. Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Матричное перемножение

Работу выполнил: Рязанов М. С., ИУ7-52Б

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

Москва, 2019

Оглавление

Введение

Алгоритм умножения матриц применяется во многих серьезных вычислительных задачах, например, в таких областях как машинное обучение и компьютерная графика. Поиск способа оптимизации такой операции является важным вопросом.

Целью данной лабораторной работы является изучение алгоритмов, умножения матриц. Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать существующие методы решения задачи;
- разработать алгоритмы решения задачи;
- оптимизировать алгоритмы и расчитать их трудоемокость;
- выбрать технологии для последующей реализации и иследования алгоритмов;
- реализовать разработанные алгоритмы;
- произвести тестирование корректности работы реализаций;
- сравнить быстродействие реализаций;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 | Аналитическая часть

В данном разделе анализируется классический подход к решению задачи об умножении матриц, а тажкже алгоритм Винограда.

1.1 Описание алгоритмов

1.1.1 Классический алгоритм умножения матриц

Матрица определяется как математический объект, записываемый в виде прямоуголоьной таблицы чисел, которая представляется собой совокупность строк и столбцов, на пересечении которых находятся элементы. Количество строк и столбцов является размерностью матрицы.

Пусть даны две матрицы A размерностью $m \times q$ и B размерностью $q \times n$.

Тогда результатом умножения матрицы A на B является матрица C раз мерностью $m \times n$, в которой элемент c_{ij} вычисляется следующим образом[2]:

$$c_{ij} = \sum_{k=1}^{q} a_{ik} b_{kj}, \tag{1.1}$$

где
$i = \overline{1, m}$
$j = \overline{1, n}$

Заметим, что операция умножения двух матриц возможна только в том случае, если число столбцов в первом сомножителе равно числу строк во втором.

1.1.2 Умножение матриц по Винограду

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответсвующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.[3]

Рассмотрим два вектор $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \times W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w + 4. \tag{1.2}$$

Это равенство можно переписать в виде:

$$V \times W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{1.3}$$

Кажется, что второе выражение задает больше работы, чем первое: вместо четырех умножений их щесть, а место трех сложений - десять. 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 стобльца второй. На практике это означает, что над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.1.3 Выводы

В аналитическом разделе выполнено следующее:

- Расмотрены основные алгоритмы умножения матриц;
- Указаны идейные различия между ними;
- Обосновано сокращение операций в алгоритме Винограда;
- Для расмотренных алгоритмов получены расчетные соотонешния приведнные в формулах 1.1, 1.3.

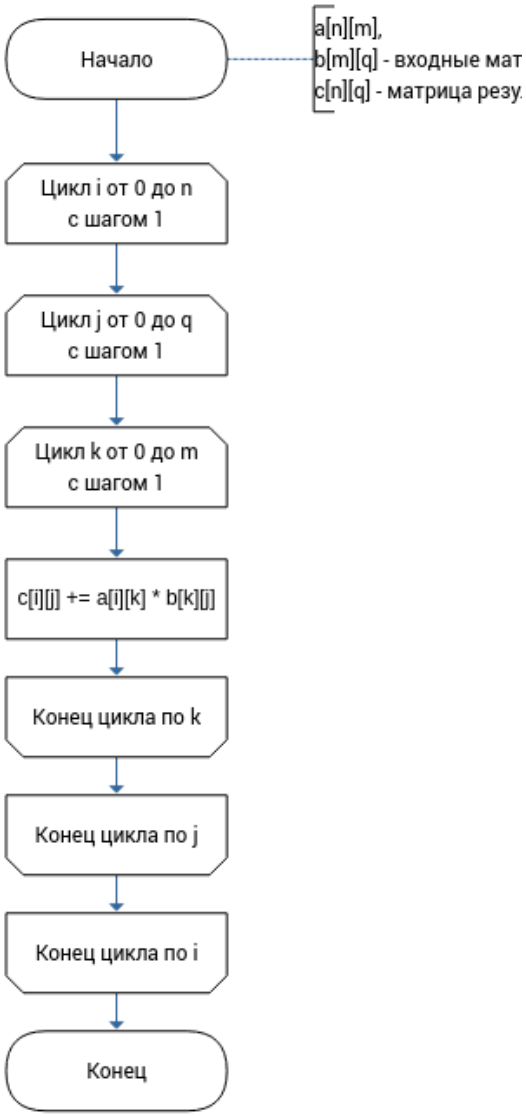
2 | Конструкторская часть

2.1 Разработка алгоритмов

Общая идея обоих алгоритмов сводится к вычислению рассчетных соотношений 1.1, 1.3. За тем исключением, что для алгоритма Винограда заранее вычислить значения для каждой строки и столбца.

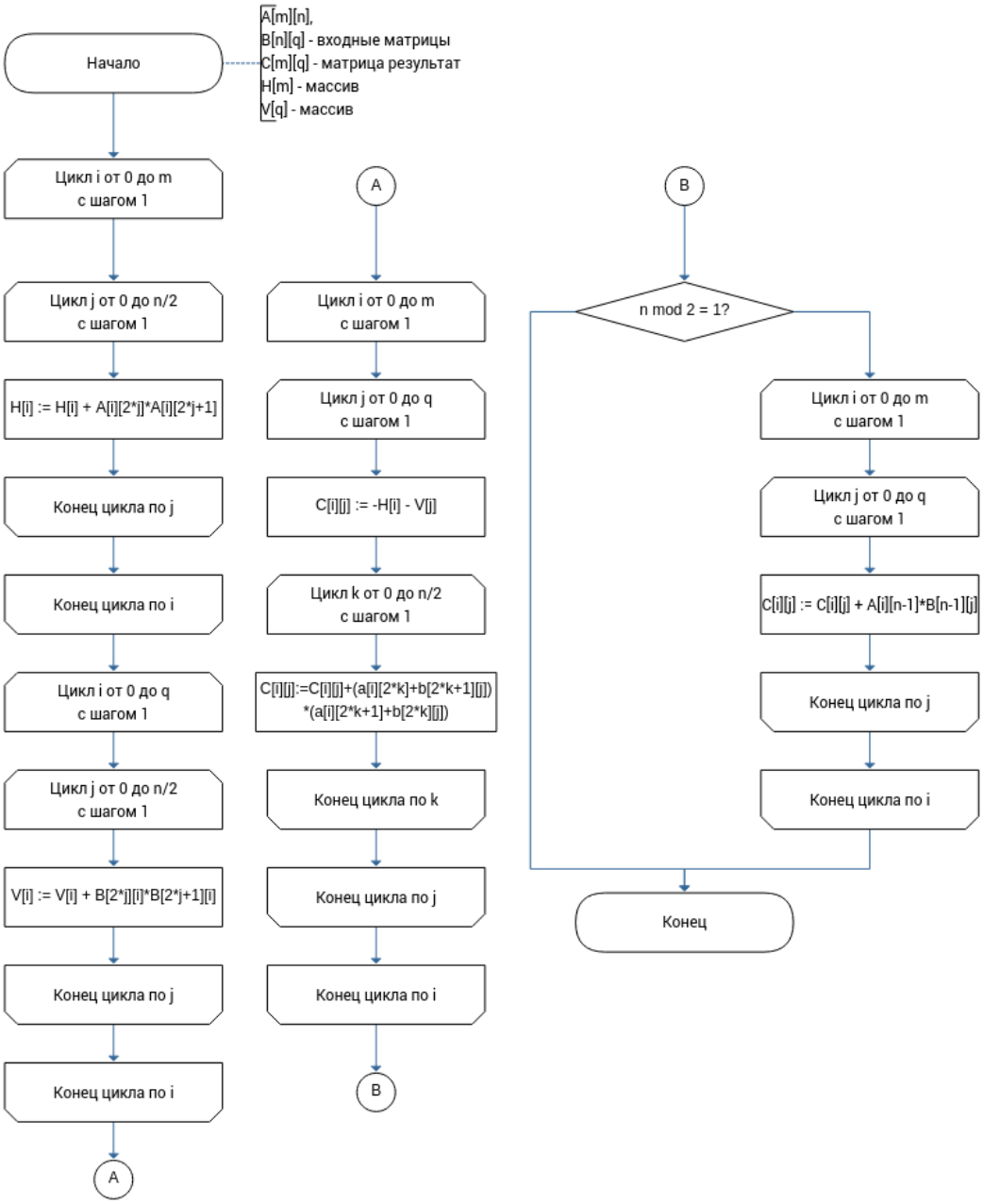
На рисинуке 2.1 приведна схема классического алгоритма умножения матриц

Рис. 2.1: Схема обычного алгоритма умножения матриц



На рисунке 2.2 приведена схема алгоритма Винограда

Рис. 2.2: Схема алгоритма Винограда



К алгоритму Винограда также можно применить следующие оптимизации:

- цикл с шагом 2 вместо условия завершения n / 2;
- использование буфера для подсчета элемента в результирующей матрице;
- корректировка значений для матриц нечетных размеров внутри общего цикла вычисления результата.

На рисунке 2.3 приведена схема оптимизированного алгоритма Винограда

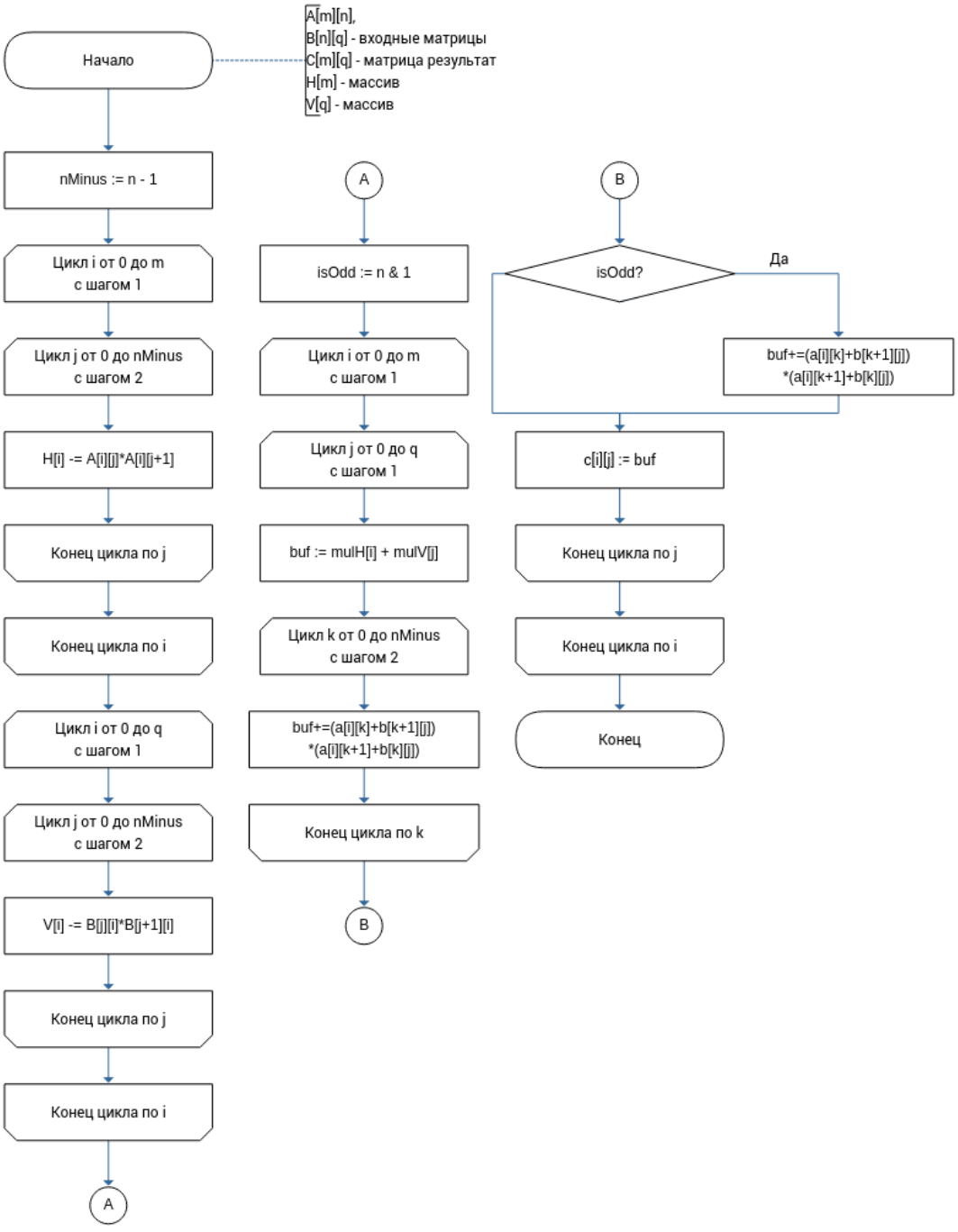


Рис. 2.3: Схема оптимизированного алгоритма Винограда

Выводы: В данном разделе были разработаны следующие алгоритмы умножения матриц

- Классический алгоритм (рисунок 2.1
- Алгоритм Винограда (рисунок 2.2
- Оптимизированный алгоритм Винограда(рисунок 2.3

3 | Технологическая часть

3.1 Выбор ЯП

В качестве языка программирования был выбран go[1] т.к он сочетает в себе эффективность, которая позволит оценить характеристики работы алгоритмов максимально точно, и удобство реализации. Также имеет встроенную библиотеку для проведение модульного тестирования.

Время работы алгоритмов было замерено с помощью функции Now() из пакета time.

3.2 Сведения о модулях программы

Программа состоит из:

- main.go - главный файл программы - точка входа программы, обработка входных данных;
- matrix.go - файл с реализацией алгоритмов;
- utils.go - файл с вспомогательными функциями;
- matrix_test.go - файл с тестами.

Листинг 3.1: Функция классического умножения матриц

```
1 func RegMulInt(a, b [][]int) [][]int {
2   var (
3     n = len(a) // Число строк в первой матрице
4     m = len(b) // Число строк в второй матрице
```

```
    q = len(b[0]) // Число столбцов в второй матрице
  )
  res := InitMatrixInt(n, q) // Инициализация матрицы результата

  for i := 0; i < n; i++ {
    for j := 0; j < q; j++ {
      for k := 0; k < m; k++ {
        res[i][j] += a[i][k] * b[k][j]
      }
    }
  }
  return res
}
```

Листинг 3.2: Функция матричного умножения по Винограду

```
func VinMulInt(a, b [][]int) [][]int {
  var (
    m = len(a) // Число строк в первой матрице
    n = len(b) // Число строк в второй матрице
    q = len(b[0]) // Число столбцов в второй матрице
  )
  res := InitMatrixInt(m, q) // Инициализация матрицы результата

// Предпосчет сум произведений элементов для каждой строки
  mulH := make([]int, m)
  for i := 0; i < m; i++ {
    for j := 0; j < n/2; j++ {
      mulH[i] = mulH[i] + a[i][2*j]*a[i][2*j+1]
    }
  }

// Предпосчет сум произведений элементов для каждого столбца
  mulV := make([]int, q)
  for i := 0; i < q; i++ {
    for j := 0; j < n/2; j++ {
      mulV[i] = mulV[i] + b[2*j][i]*b[2*j+1][i]
    }
  }

```

```
25 // Вычисление матрице результата
26   for i := 0; i < m; i++ {
27     for j := 0; j < q; j++ {
28       res[i][j] = -mulH[i] - mulV[j]
29       for k := 0; k < n/2; k++ {
30         res[i][j] = res[i][j] + (a[i][2*k]+b[2*k+1][j])*(a[
    i][2*k+1]+b[2*k][j])
31       }
32     }
33   }
34
35 // Корректировка значения для матриц с нечетным число строк
36   if n%2 == 1 {
37     for i := 0; i < m; i++ {
38       for j := 0; j < q; j++ {
39         res[i][j] = res[i][j] + a[i][n-1]*b[n-1][j]
40       }
41     }
42   }
43
44   return res
45 }
```

Листинг 3.3: Оптимизированная функция умножения матриц по Винограду

```
1 func OptVinMulInt(a, b [][]int) [][]int {
2   var (
3     m       = len(a) // Число строк в первой матрице
4     n       = len(b) // Число строк в второй матрице
5     q       = len(b[0]) // Число столбцов в второй матрице
6     nMinus = n - 1
7   )
8   res := InitMatrixInt(m, q) // Инициализация матрицы результата
9
10 // Предпосчет сум произведений элементов для каждой строки
11   mulH := make([]int, m)
12   for i := 0; i < m; i++ {
13     for j := 0; j < nMinus; j += 2 {
14       mulH[i] -= a[i][j] * a[i][j+1]
```

```
15      }
16    }
17
18  // Предпосчет сум произведений элементов для каждого столбца
19    mulV := make([]int, q)
20    for i := 0; i < q; i++ {
21      for j := 0; j < nMinus; j += 2 {
22        mulV[i] -= b[j][i] * b[j+1][i]
23      }
24    }
25
26  // Вычисление матрицы результата
27    isOdd := n%2 == 1
28    for i := 0; i < m; i++ {
29      for j := 0; j < q; j++ {
30        buf := mulH[i] + mulV[j]
31        for k := 0; k < nMinus; k += 2 {
32          buf += (a[i][k] + b[k+1][j]) * (a[i][k+1] + b[k][j
     ])
33        }
34      // Корректировка значения для матриц нечетного размера
35        if isOdd {
36          buf += a[i][nMinus] * b[nMinus][j]
37        }
38        res[i][j] = buf
39      }
40    }
41
42    return res
43  }
```

3.3 Тестирование

Тестирование было организовано с помощью пакета **testing**.

Для каждой из функции были составленые следующие наборы тестов:

- квадратные матрицы;

- матрицы разных размеров;
- матрицы четных и нечетных размеров;
- нулевые матрицы;
- единичные матрицы;
- матрицы с отрицательными элементами.

3.4 Интерфейс программы

Для программы был разработан консольный интерфейс позволяющий вводить матрицы различных размеров, а также выбирать функцию выполняющую умножение(классический или Винограда). На рисунке 3.1 приведен интерфейс программы

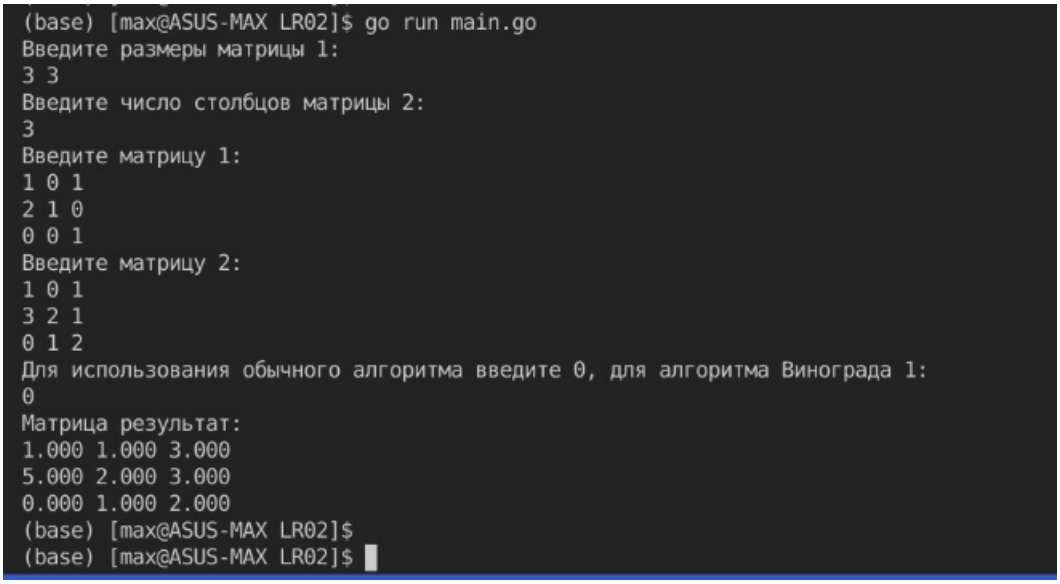


Рис. 3.1: Интерфейс программы

3.5 Сравнительный анализ реализаций

Примем следующую теоретическую модель вычислений для оценки трудуемкости алгоритмов:

- операции единичной стоимости(арифмитические, сравнения, обращение по адресу)
- Циклы определяются формулой:

$$f_{cycle} = f_{init} + f_{cmp} + n \times (f_{inner} + f_{inc} + f_{cmp}), \tag{3.1}$$

  где
  n - количество итераций цикла,
  f_{cycle} - трудоемкость цикла,
  f_{init} - трудоемкость инициализации,
  f_{cmp} - трудоемкость сравнения,
  f_{inner} - трудоемкость тела цикла,
  f_{inc} - трудоемкость инкремента;

- переход по условию в условном операторе равен нулю, но при этом в расчет входит трудоемкость вычисления условия.

Пусть даны две матрицы A размерностью $m \times n$ и B размерностью $q \times n$, результат их умножения - матрица C размерностью $m \times n$.

3.5.1 Трудоемкость стандарного алгоритма

Опираясь на указанные выше правила, подсчитаем трудоемкость стандартного алгоритма умножения матриц:

$$f_{std} = 2+m\cdot(2+2+n\cdot(2+2+q\cdot(2+\underbrace{6}_{[]}+\underbrace{1}_{\times}+\underbrace{1}_{+}))) = 10nmq+4mn+4m+2 \tag{3.2}$$

3.5.2 Трудоемкость алгоритма Винограда

Заполнение массива произведений строчных элементов:

$$f_{row} = 2+m\cdot(2+2+1+\frac{q}{2}\cdot(3+\underbrace{6}_{[]}+\underbrace{1}_{=}+\underbrace{2}_{+}+\underbrace{3}_{\times})) = \frac{15}{2}mq+5m+2 \tag{3.3}$$

Заполнение массива произведений столбцовых элементов производится аналогично, поэтому можно перейти сразу к формуле:

$$f_{col} = \frac{15}{2}qn + 5n + 2. \tag{3.4}$$

Основной цикл вычисления значений элементов результирующей матрицы:

$$f_{inner} = 2+m\cdot(2+2+n\cdot(2+7+3+\frac{q}{2]}\cdot(3+\underbrace{12}_{[]}+\underbrace{1}_{=}+\underbrace{+}_{5}+\underbrace{5}_{\times} = 13mnq+12qn+4m+2. \tag{3.5}$$

Учет условия перехода при значении q:

$$f_{cond} = 2+\left[\begin{array}{c} 0, \text{если } q \text{ четное} \\ 2+m\cdot(2+2+n\cdot(2\underbrace{8}_{[]}\underbrace{1}_{=}+\underbrace{1}_{+}+\underbrace{1}_{\times}+\underbrace{2}_{-}), \text{иначе} \end{array}\right. = \left[\begin{array}{c} 0, \text{ если } q-\text{четное} \\ 15mn+4m+2 \end{array}\right.$$

Таким образом, получаем формулу вычисления произведения матриц алгоритмом Винограда:

$$f_V = 13mnq+\frac{15}{2}mq+\frac{39}{2}qn+9m+5n+8+\left[\begin{array}{cc} 0 & , \text{если } q-\text{четное} \\ 15mn+4m+2, & \text{иначе} \end{array}\right. \tag{3.7}$$

3.5.3 Трудоемкость оптимизированного алгоритма Винограда

Заполнение массива произведений строчных элементов:

$$f_{row} = 2+m\cdot(2+2+\frac{q}{2}\cdot(2+\underbrace{5}_{[]}+\underbrace{1}_{\times}+\underbrace{1}_{+}+\underbrace{1}_{-=})) = 5mq+4m+2. \tag{3.8}$$

Заполнение массива произведений столбцовых элементов производится аналогично, поэтому можно перейти сразу к формуле:

$$f_{col} = 5nq + 4n + 2. \tag{3.9}$$

Основной цикл вычисления значений элементов результирующей матрицы для четных q:

$$f_{inner} = \underbrace{3}_{isOdd} + 2 + m \cdot (2 + 2 + n \cdot (2 + 2 + \underbrace{1}_{if} + \underbrace{4}_{[]} + \underbrace{2}_{=} + \underbrace{1}_{+} +$$
$$+ \frac{q}{2} \cdot (2 + \underbrace{8}_{[]} + \underbrace{1}_{+=} + \underbrace{4}_{+} + \underbrace{1}_{\times}))) = 8mnq + 11mn + 4m + 5 \quad (3.10)$$

Для нечетных q:

$$f_{inner} = \underbrace{3}_{isOdd} + 2 + m \cdot (2 + 2 + n \cdot (2 + 2 + \underbrace{1}_{if} + \underbrace{8}_{[]} + \underbrace{2}_{=} + \underbrace{1}_{+} + \underbrace{1}_{+=} + \underbrace{1}_{\times} +$$
$$+ \frac{q}{2} \cdot (2 + \underbrace{8}_{[]} + \underbrace{1}_{+=} + \underbrace{4}_{+} + \underbrace{1}_{\times})) = 8mnq + 18mn + 4m + 5 \quad (3.11)$$

Общая трудоемкость:

$$f_V = 8mnq + 5mq + 5nq + 8m + 9 + \left[\begin{array}{c} 11mn, \text{ если } q - \text{четное} \\ 18mn, \text{ иначе} \end{array} \right. \quad (3.12)$$

Выводы:

- В качестве языка программирования выбран Golang, так как прост в использовании, а также достаточно эффективен для проведения анализа времени работы разработанных алгоритмов
- Алгоритмы реализованы в виде подпрограмм, код которых приведен в листингах : 3.1, 3.2, 3.3
- Разработаны тестовые случаи для проверки корректности работы функций
- Был выбран консольный интерфейс программы, так как прост в реализации и использовании при небольшом функционале программмы.
- Были расчитаны трудоемкости реализованных алгоритмов

4 | Исследовательская часть

Для оценки эффективности реализованных функций были произведены замеры времени вычислений на квадратных матрицах разного замера. На рисунках 4.1 4.2 приведены графики зависимости времени вычислений от размера матрицы для матриц четных и нечетных размеров.

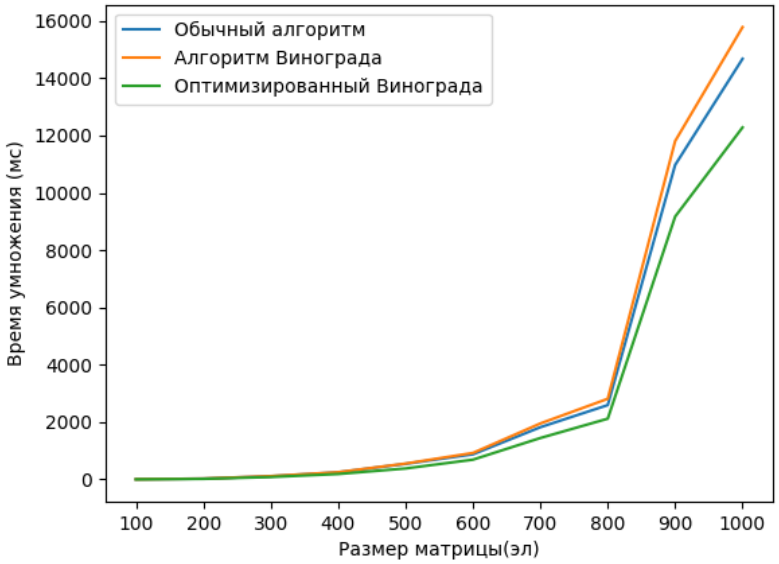


Рис. 4.1: Время выполнения на матрицах четного размера

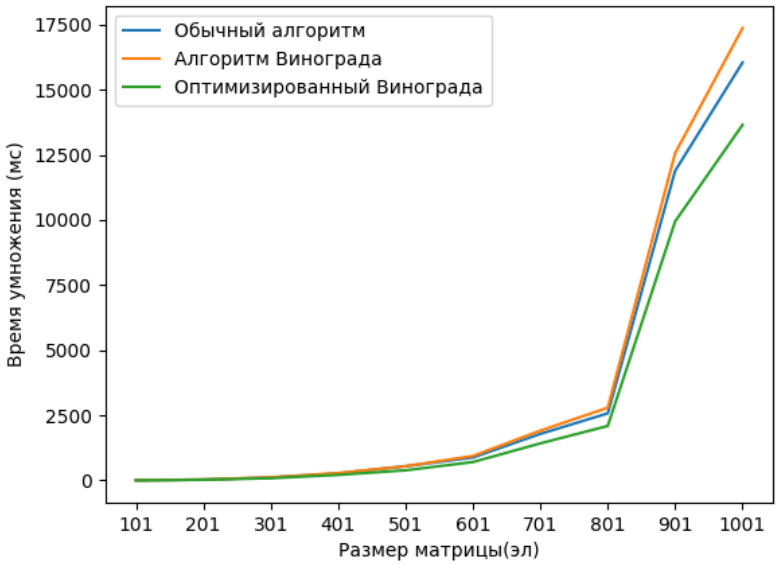


Рис. 4.2: Время выполнения на матрицах нечетного размера

Из приведенных графиков видно, что кривая времени выполения каждого из алгоритма имеет порядка n^3. В тоже время выполнения оптимизированного алгоритма Винограда меньше для размера матриц больше 400.

Выводы:

- Все алгоритмы имеют временную ассимптотику $O(n^3)$
- Оптимизированный алгоритм Винограда имеет меньшую константу при n^3, имеет заметно меньшее время выполнения при размерах матриц более 400

Заключение

Был изучены и реализованы алгоритмы умножения матриц: классический и алгоритм Винограда. Была произведена оптимизация алгоритма Винограда. Для реализованных алгоритмов были расчитаны трудоемкости.

Экспериментально было подтверждено различие во временной эффективности разных алгоритмов, и что алгоритм Винограда дает выигрыш на больших размерах матриц.

В результате исследований пришел к выводу, что использование алгоритма Винограда оправдано при перемножении матриц больших размеров(более 400), в то время как для перемножения матриц меньших размеров выгоднее использовать классический алгоритм.

Список литературы

[1] *Golang documentation*. URL: `https://godoc.org/`. (accessed: 02.10.2019).

[2] Г. Корн. *Алгебра матриц и матричное исчисление*. М: Наука, 1978.

[3] Дж. Макконнелл. *Анализ алгоритмов. Активный обучающий подход*. Информатика и вычислительная биология. М.:Техносфера, 2009.