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 Прохорова Л. А.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Содержание

Введение

Термин «матрица» применяется во множестве разных областей: от программирования до кинематографии.

Матрица в математике – это таблица чисел, состоящая из определенного количества строк (m) и столбцов (n).

Мы встречаемся с матрицами каждый день, так как любая числовая информация, занесенная в таблицу, уже в какой-то степени считается матрицей.

Примером могут служить:

- список телефонных номеров;
- различные статистические данные;
- табель успеваемости ученика и многое другое.

1 Аналитическая часть

Целью работы работы является изучение и реализация алгоритмов умножения матриц, вычисление трудоёмкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.

Для достижения цели ставятся следующие задачи.

- Изучить классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
- Реализовать классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
- Дать оценку трудоёмкости алгоритмов.
- Замерить время работы алгоритмов.
- Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

Матрица

Матрица – математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. Две матрицы одинакового размера можно поэлементно сложить или вычесть друг из друга [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. При умножении матрицы размером 3x4 на матрицу размером

4x7 мы получаем матрицу размером 3x7. Умножение матриц некоммутативно: оба произведения AB и BA двух квадратных матриц одинакового размера можно вычислить, однако результаты, вообще говоря, будут отличаться друг от друга [1].

1.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы A (1) и B (2) размерности m на n и n на l соответсвенно:

$$\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,l} \\ \dots & \dots & \dots \\ b_{n,1} & \dots & b_{n,l} \end{bmatrix} \tag{2}$$

В результате получим матрицу C 3 размерности m на l:

$$\begin{bmatrix} c_{1,1} & \dots & c_{1,l} \\ \dots & \dots & \dots \\ c_{m,1} & \dots & c_{m,l} \end{bmatrix} \tag{3}$$

Формула 4 - формула рассчёта элемента, находящегося на i-ой строке j-ого столбца матрицы C.

$$c_{i,j} = \sum_{r=1}^{n} a_{i,r} \cdot b_{r,j} \tag{4}$$

1.2 Алгоритм Винограда

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 [2].

Рассмотрим два вектора $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$. Их скалярное произведение 5 равно:

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{5}$$

Это равенство можно переписать в виде 6:

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{6}$$

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 [2].

1.3 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

- вычисление происходит заранее;
- используется битовый сдвиг, вместо деления на 2;
- последний цикл для нечётных элементов включён в основной цикл, используя дополнительные операции в случае нечётности N.

1.4 Вычисление трудоёмкости алгоритма

Введем модель трудоемкости для оценки алгоритмов.

1. Базовые операции стоимостью 1: +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса.

2. Оценка трудоемкости цикла: f_цикла = f_инициализации + f_сравнения + N * (f_инкремента + f_сравнения + f_тела).

3. Стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся. В условном операторе может возникнуть лучший и худший случаи по трудоёмкости в зависимости от выполнения условия и в зависимости от входных данных алгоритма.

1.5 Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основная разница которого — наличие предварительной обработки, а также уменьшение количества операций умножения.

2 Конструкторская часть

2.1 Требования к вводу

В разрабатываемом ПО предъявляется следующее требование ко вводу а вход подаются две матрицы.

2.2 Требования к выводу

Программа выводит матрицу(результат умножения матриц, поданых на вход).

2.3 Требования к программе

Две пустые матрицы – корректный ввод, программа не должна аварийно завершаться.

2.4 Схемы алгоритмов

На рисунке 1 представлена схема классического умножения матриц.

На рисунках 2, 3, 4, 5 представлена схема алгоритма умножения матриц Винограда.

На рисунках 6, 7, 8, 9 представлена схема оптимизированного алгоритма умножения матриц Винограда.

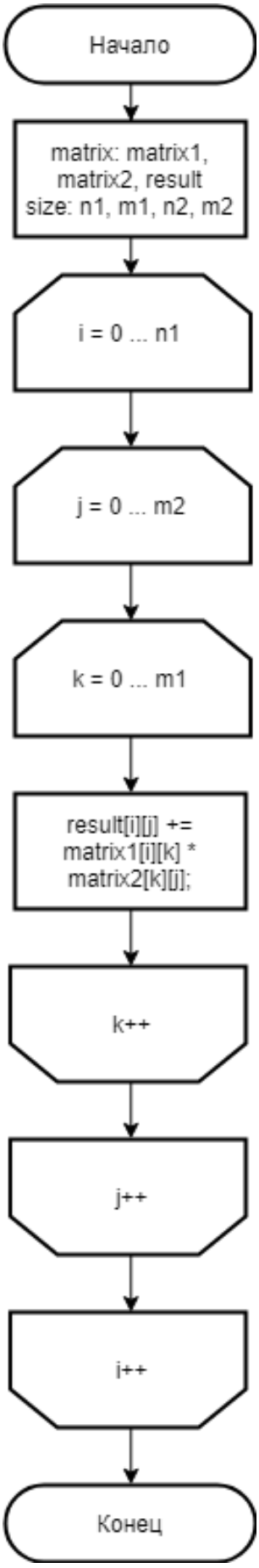

Рисунок 1 – Схема классического алгоритма умножения матриц

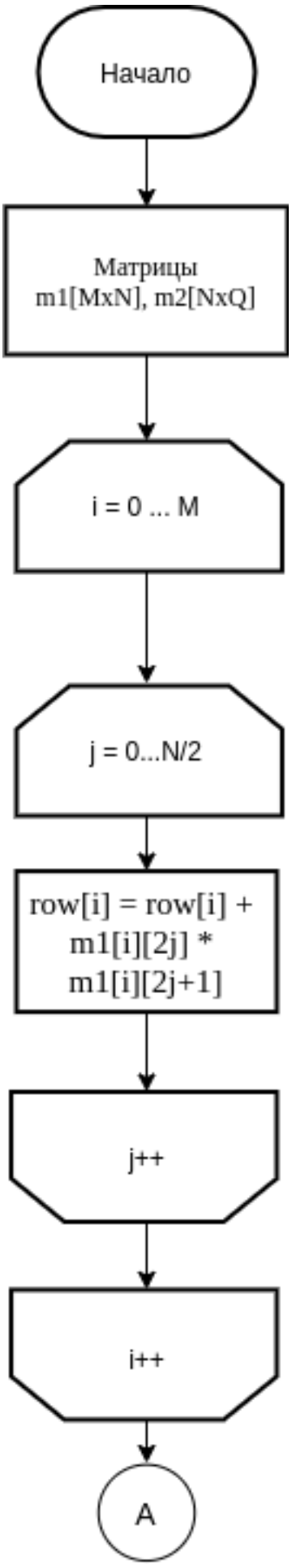

Рисунок 2 – Схема алгоритма Винограда (часть 1)

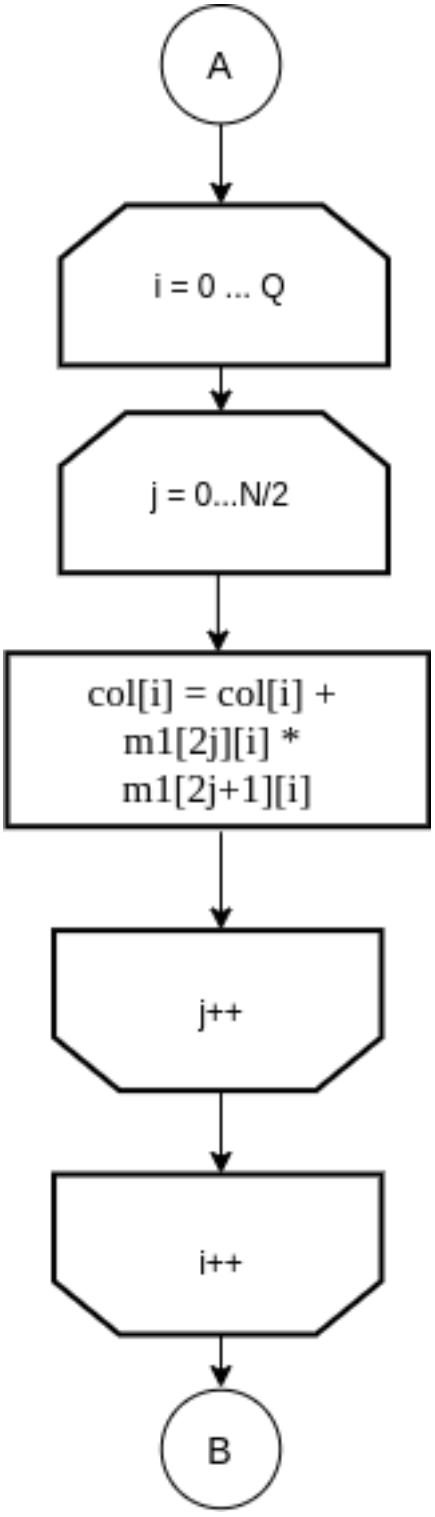

Рисунок 3 – Схема алгоритма Винограда (часть 2)

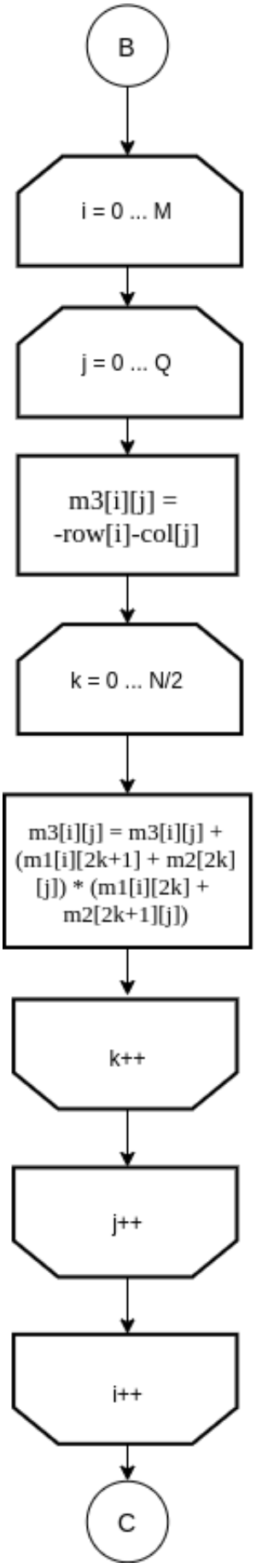

Рисунок 4 – Схема алгоритма Винограда (часть 3)

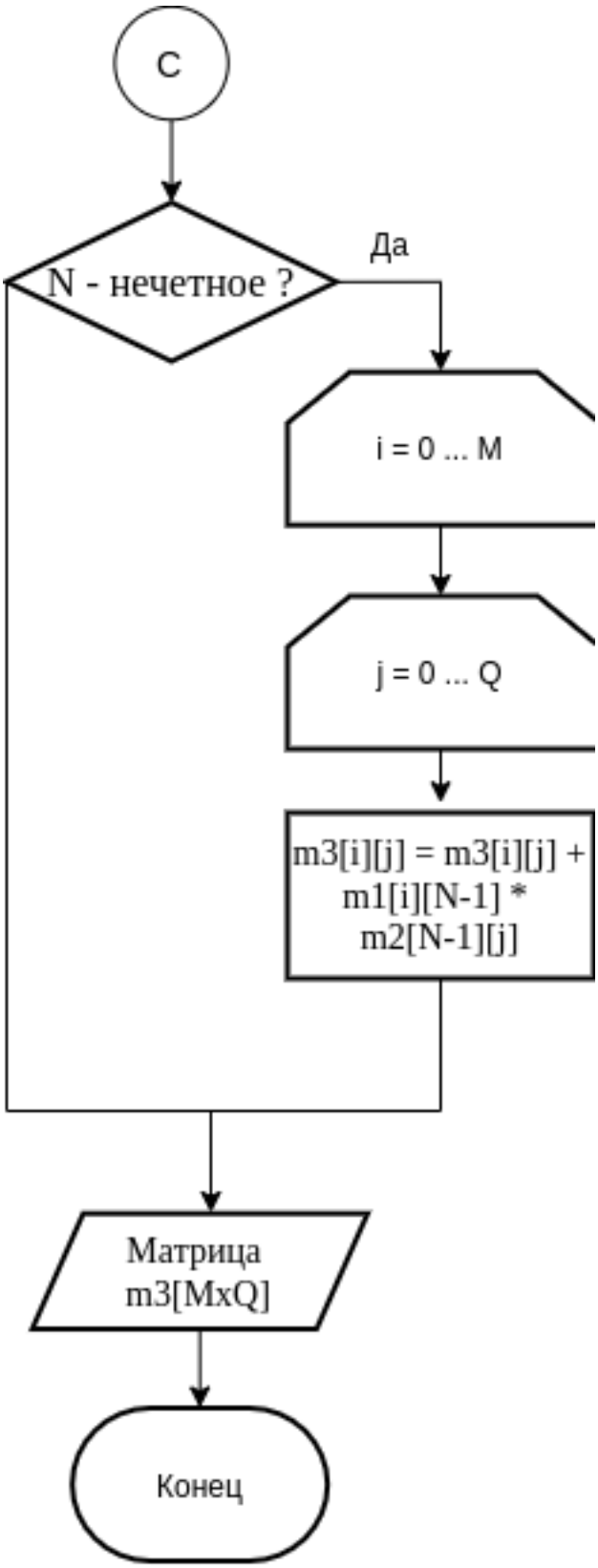

Рисунок 5 – Схема алгоритма Винограда (часть 4)

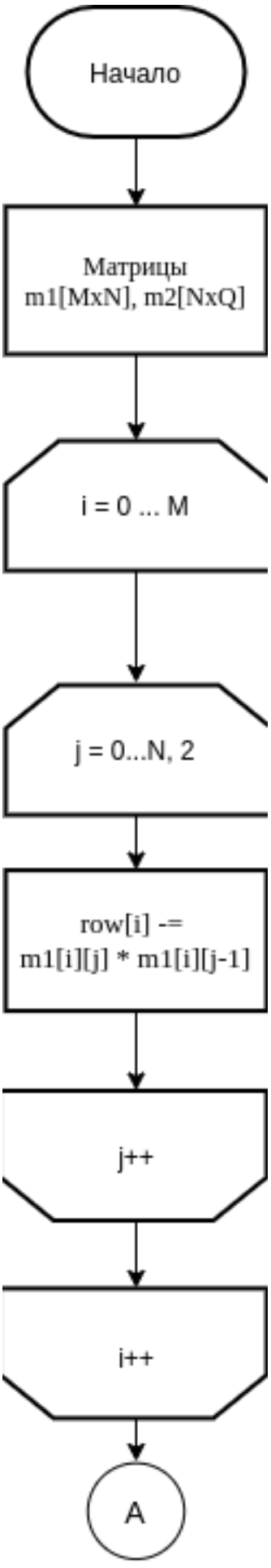

Рисунок 6 – Схема оптимизированного алгоритма Винограда(часть 1)

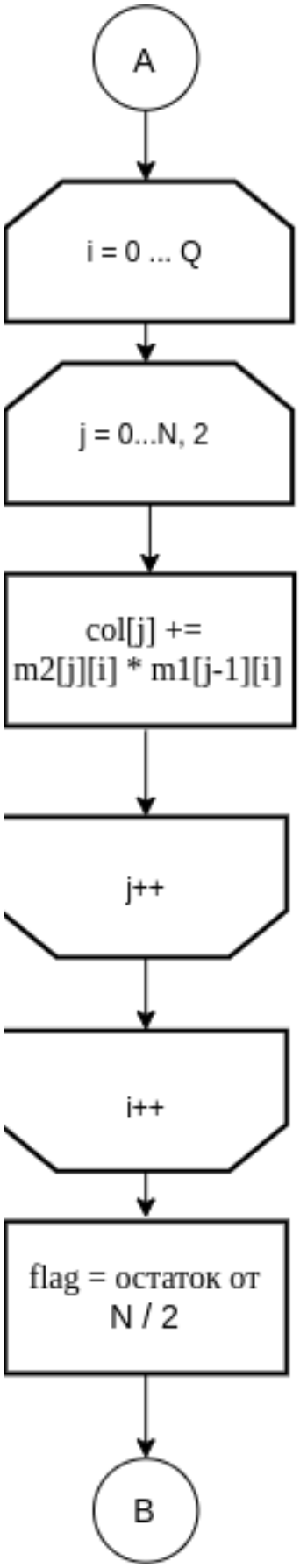

Рисунок 7 – Схема оптимизированного алгоритма Винограда(часть 2)

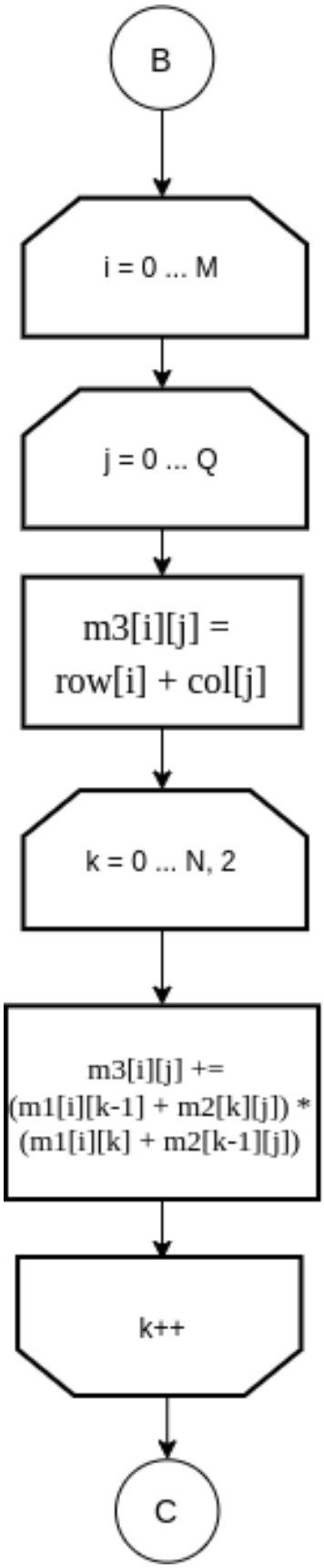

Рисунок 8 – Схема оптимизированного алгоритма Винограда(часть 3)

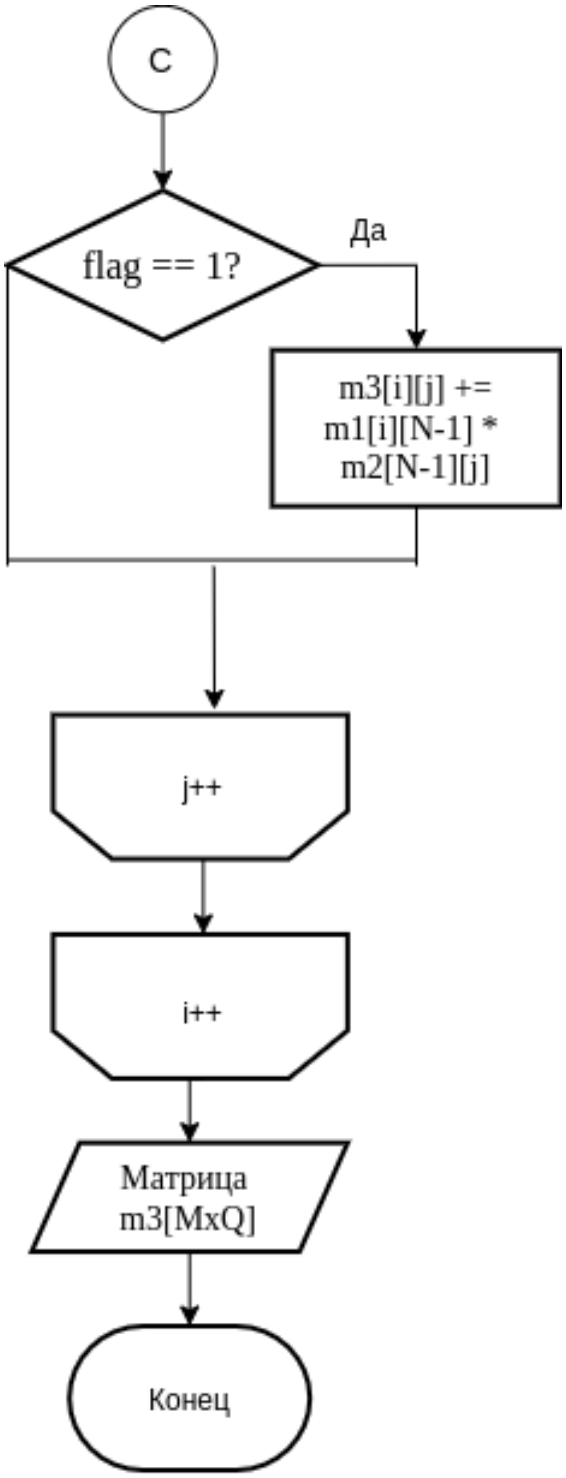

Рисунок 9 – Схема оптимизированного алгоритма Винограда(часть 4)

2.5 Оценка трудоемкости алгоритмов умножения матриц

Оценка трудоемкости будет дана согласно введённой выше модели вычислений.

1. Стандартный алгоритм

$$f = 2 + M(2 + 2 + Q(2 + 2 + N(2 + 8 + 1 + 1 + 1))) = 13\cdot$$

$$\cdot MNQ + 4MQ + 4M + 2 \approx 13 \cdot MNQ$$

2. Алгоритм Винограда

   Трудоемкость алгоритма Винограда:

   Первый цикл: $\frac{15}{2} \cdot NQ + 5 \cdot M + 2$

   Второй цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

   Третий цикл: $13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2$

   Условный переход: $\begin{bmatrix} 2 & \text{, лучший случай (при четном N)} \\ 15 \cdot QM + 4 \cdot M + 4 & \text{, худший случай} \end{bmatrix}$

   Итого: $f = \frac{15}{2} \cdot MN + \frac{15}{2} \cdot QN + 9 \cdot M + 8 + 5 \cdot Q + 13 \cdot MNQ + 12 \cdot MQ + \begin{bmatrix} 2 & \text{, в лучшем случае} \\ 15 \cdot QM + 4 \cdot M + 4 & \text{, в худшем случае} \end{bmatrix}$

$$f \approx 13 \cdot MNQ$$

3. Оптимизированный алгоритм Винограда

   Введем оптимизации:

   (a) замена операции = на += или -=

   (b) избавление от деления в условиях цикла (j < N, j += 2)

   (c) Заносим проверку на нечетность кол-ва строк внутрь основных циклов

   (d) Расчет условия для последнего цикла один раз, а далее использование флага

   Первый цикл: $4 \cdot NQ + 4 \cdot M + 2$

   Второй цикл: $4 \cdot MN + 4 \cdot M + 2$

   Третий цикл: $9 \cdot MNQ + 10 \cdot MQ + 4 \cdot M + 2$

   Условный переход: $\begin{bmatrix} 2 & \text{, лучший случай (при четном N)} \\ 10 \cdot QM & \text{, худший случай} \end{bmatrix}$

   Трудоемкость оптимизированного алгоритма Винограда:

   Итого:

$$f = 4 \cdot NQ + 4 \cdot M + 2 + 4 \cdot MN + 4 \cdot M + 2 + 9 \cdot MNQ + 10 \cdot MQ + 4 \cdot M + 2 +$$

$$+ \begin{bmatrix} 2 & \text{, л.с} \\ 10 \cdot QM & \text{, х.с} \end{bmatrix} \approx 9 \cdot MNQ$$

3 Технологическая часть

3.1 Выбор языка программирования

Я выбрала языком программирования Python.

Время работы алгоритмов было замерено с помощью функции process_time() из библиотеки time.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, в котором располагаются алгоритмы и меню;
- test.py - файл с замерами времени

На листингах 1, 2, 3 представлен код алгоритмов.

Листинг 1 – Классический алгоритм умножения матриц.

```
0  def standart_multiplication_matrix(m1, m2):
1      if len(m2) != len(m1[0]):
2          print("Size error!")
3          return -1
4      else:
5          n = len(m1)
6          q = len(m2[0])
7          m = len(m1[0])
8          m3 = [[0] * q for i in range(m)]
9
10         for i in range(0, n):
11             for j in range(0, q):
12                 for k in range(0, m):
13                     m3[i][j] = m3[i][j] + m1[i][k] * m2[k][j]
14     return m3
15
```

Листинг 2 – Алгоритм Винограда.

```
def vinograd_multiplication_matrix(m1, m2):
  if len(m2) != len(m1[0]):
      print("Size error!")
      return -1
  else:
      m = len(m1)
      n = len(m1[0])
      q = len(m2[0])
      m3 = [[0] * q for i in range(m)]

      row = [0] * m
      for i in range(0, m):
          for j in range(0, n // 2, 1):
              row[i] = row[i] + m1[i][2 * j] * m1[i][2 * j + 1]

      col = [0] * q
      for j in range(0, q):
          for i in range(0, n // 2, 1):
              col[j] = col[j] + m2[2 * i][j] * m2[2 * i + 1][j]

      for i in range(0, m):
          for j in range(0, q):
              m3[i][j] = -row[i] - col[j]
              for k in range(0, n // 2, 1):
                  m3[i][j] = m3[i][j] + (m1[i][2 * k + 1] + m2[2 * k][j
 ]) * (m1[i][2 * k] + m2[2 * k + 1][j])

      if n % 2 == 1:
          for i in range(0, m):
              for j in range(0, q):
                  m3[i][j] = m3[i][j] + m1[i][n - 1] * m2[n - 1][j]

  return m3


```

Листинг 3 – Оптимизированный алгоритм Винограда.

```
def vinograd_optimizate_multiplication_matrix(m1, m2):
  if len(m2) != len(m1[0]):
      print("Size error!")
```

```
3           return -1
4       else:
5           m = len(m1)
6           n = len(m1[0])
7           q = len(m2[0])
8           m3 = [[0] * q for i in range(m)]
9
10          row = [0] * m
11          for i in range(0, m):
12              for j in range(1, n, 2):
13                  row[i] -= m1[i][j] * m1[i][j - 1]
14
15          col = [0] * q
16          for j in range(0, q):
17              for i in range(1, n, 2):
18                  col[j] -= m2[i][j] * m2[i - 1][j]
19
20          flag = n % 2
21          for i in range(0, m):
22              for j in range(0, q):
23                  m3[i][j] = row[i] + col[j]
24                  for k in range(1, n, 2):
25                      m3[i][j] += (m1[i][k - 1] + m2[k][j]) * (m1[i][k] + m2
    [k - 1][j])
26                  if (flag):
27                      m3[i][j] += m1[i][n - 1] * m2[n - 1][j]
28      return m3
29
30
```

3.3 Функциональные тесты

Таблица 3.1 – Функциональные тесты

Матрица1	**Матрица2**	**Ожидаемый результат**
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	Матрицы не могут быть перемножены
$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 9 & 12 & 15 \\ 24 & 33 & 42 \\ 39 & 54 & 69 \end{bmatrix}$

Все реализации алгоритмов на практике показали ожидаемый результат.

Вывод

В этом разделе была представлена реализация алгоритмов классического умножения матриц, алгоритма Винограда, оптимизирвоанного алгоритма Винограда. Тестирование показало, что алгоритмы реализованы правильно и работают корректно.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие.

- Операционная система Ubuntu 18.04 64-bit.
- Память 8 GiB Процессор Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz

4.2 Демонстрация работы программы

На рисунке 10 представлен результат работы программы.

```
Введите матрицу1:
Введите количество строк: 3
Введите количество столбцов: 4
Введите строку: 1 2 3 4
Введите строку: 5 6 7 8
Введите строку: 9 0 1 1
Введите матрицу2:
Введите количество строк: 4
Введите количество столбцов: 3
Введите строку: 5 6 5
Введите строку: 9 8 9
Введите строку: 0 1 0
Введите строку: 9 4 3
Результат, посчитанный классическим алгоритмом:
[59, 41, 35]
[151, 117, 103]
[54, 59, 48]
[0, 0, 0]
Результат, посчитанный алгоритмом Винограда:
[59, 41, 35]
[151, 117, 103]
[54, 59, 48]
Результат, посчитанный оптимизированным алгоритмом Винограда:
[59, 41, 35]
[151, 117, 103]
[54, 59, 48]
```

Рисунок 10 – Результат работы программы.

4.3 Время работы алгоритмов.

Алгоритмы тестировались при помощи функции process_time() [3] из библиотеки time языка Python. time.process_time() всегда возвращает зна-

чение времени типа float в секундах. Возвращает значение (в долях секунды) суммы системного и пользовательского процессорного времени текущего процесса. Не включает время, прошедшее во время сна. Контрольная точка возвращаемого значения не определена, поэтому допустима только разница между результатами последовательных вызовов. Замеры времени для каждой длины слов проводились 100 раз. В качестве результата взято среднее время работы сортировки на каждой длине массива.

На рисунках 11 и 12 изображены графики замеров времени работы алгоритмов на квадратных матрицах, заполненных случайными числами от -100 до 100. Для четного случая линейного размера матриц размеры принимали значение 2, 10, 50, 100, 200, 300, 400, для нечетного случая – значения 3, 11, 55, 101, 203, 305, 403. Время измеряется в нс.

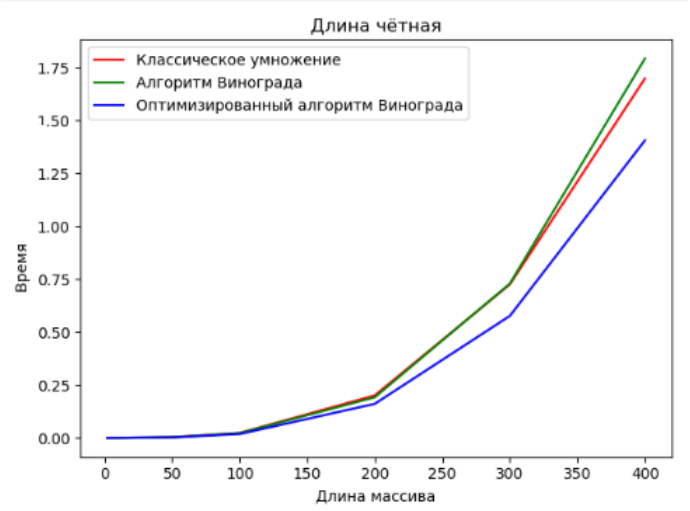

Рисунок 11 – Замеры времени на четном количестве строк и столбцов квадратных матриц.

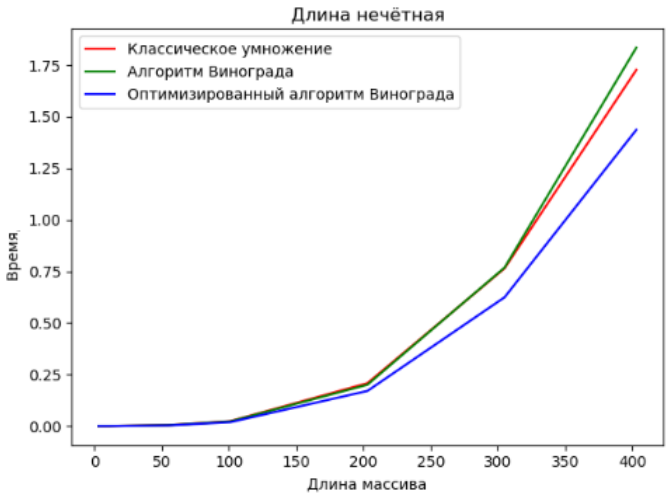

Рисунок 12 – Замеры времени на нечетном количестве строк и столбцов квадратных матриц.

Как видно из графиков, реализации алгоритмов показывают одинаковый характер роста времени работы на чётных и нечётных размерах, что подтверждает полученные оценки трудоёмкости алгоритмов (кубические функции от линейного размера матриц).

Самым быстрым является оптимизированный алгоритм Винограда, на размере матрицы 400 x 400 он работает в 1,3 раза быстрее алгоритма Винограда без оптимизации, и в 1,25 раз быстрее классического алгоритма умножения. Алгоритмы Винограда и классического умножения сопоставимы.

Вывод

В результате проведенного эксперимента был получен следующий вывод: оптимизированный алгоритм Винограда работает быстрее классического метода и зачительно быстрее обычного алгоритма Винограда.

Заключение

Цель лабораторной работы достигнута.

В ходе выполнения работы решены следующие задачи.

- Изучены классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
- Реализованы классический алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда.
- Дана оценка трудоёмкости алгоритмов.
- Замерено время работы алгоритмов.
- Описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

Экспериментально было подтверждено различие по временной эффективности алгоритмов умножения матриц на материале замеров процессорного времени выполнения реализации на варьирующихся размерах матриц. Так, самым быстрым является оптимизированный алгоритм Винограда, на размере матрицы 400 x 400 он работает в 1,3 раза быстрее алгоритма Винограда без оптимизации, и в 1,25 раз быстрее классического алгоритма умножения. Алгоритмы Винограда и классического умножения сопоставимы.

На основании сравнения данных алгоритмов был сделан вывод, что классический алгоритм является более эффективным, чем алгоритм Винограда, однако после ряда оптимизаций, алгоритм Винограда становится значительно быстрее классического.

Список литературы

[1] Умножение матриц.[Электронный ресурс].Режим доступа: https://life-prog.ru/2_90314_umnozhenie-matrits.html (дата обращения: 25.09.2020)

[2] Умножение матриц по Винограду [электронный ресурс]. Режим доступа:

http://algolib.narod.ru/Math/Matrix.html (дата обращения: 25.09.2020).

[3] Python documentation [Электронный ресурс]. Режим доступа:

https://www.python.org/doc/ (дата обращения: 26.09.2020).