Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Алгоритмы умножения матриц

Работу выполнил: Власенко Артём, ИУ7-54Б

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

Москва, 2020

Оглавление

Введение

Цель работы - изучение алгоритмов умножения матриц. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда. Также требуется провести рассчет сложности алгоритмов, получить навыки в оптимизации алгоритмов. Использованные алгоритмы активно применяются во всех областях, применяющих линейную алгебру, таких как:

- компьютерная графика;
- физика;
- экономика;
- прочее.

В ходе лабораторной работы предстоит:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- улучшить 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

Матрица - математический объект, эквивалентный двумерному массиву. Числа располагаются в матрице по строкам и столбцам. 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. Постановка задачи перемножения матриц описана в [1].

1.1 Классический алгоритм умножения матриц

Пусть даны две прямоугольные матрицы A и B размерности m на n и n на l соответсвенно:

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

$$\begin{bmatrix} b_{1,1} & ... & b_{1,l} \\ ... & ... & ... \\ b_{n,1} & ... & b_{n,l} \end{bmatrix}$$

В результате получим матрицу C размерности m на l:

$$\begin{bmatrix} c_{1,1} & ... & c_{1,l} \\ ... & ... & ... \\ c_{m,1} & ... & c_{m,l} \end{bmatrix}$$

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

1.2 Алгоритм Винограда

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

$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4$

Это равенство можно переписать в виде:

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

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. Подробое описание алгоритма Винограда можно найти в [2].

1.2.1 Вывод

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

2 Конструкторская часть

Требования к вводу:

- на вход подаются две матрицы и их размерности.

Требования к выводу:

- корректное произведение введённых матриц или сообщение об ошибке в случае неккоректного ввода.

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов. Схема классического алгоритма умножения матриц показана на рисунке 2.1, схема алгоритма Винограда - на рисунке 2.2, схема оптимизированного алгоритма Винограда - на рисунке 2.3.

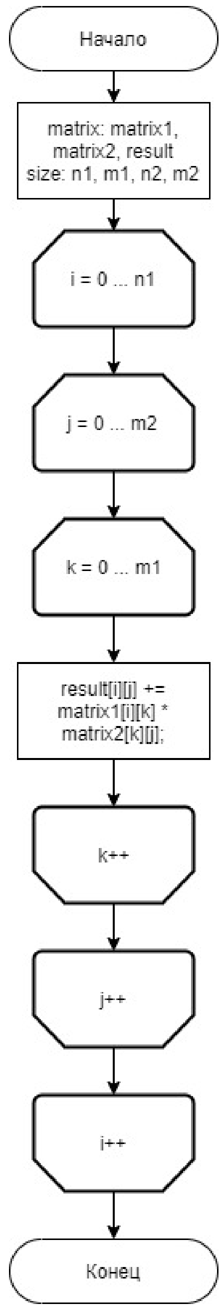

Рис. 2.1: Схема классического алгоритма умножения матриц

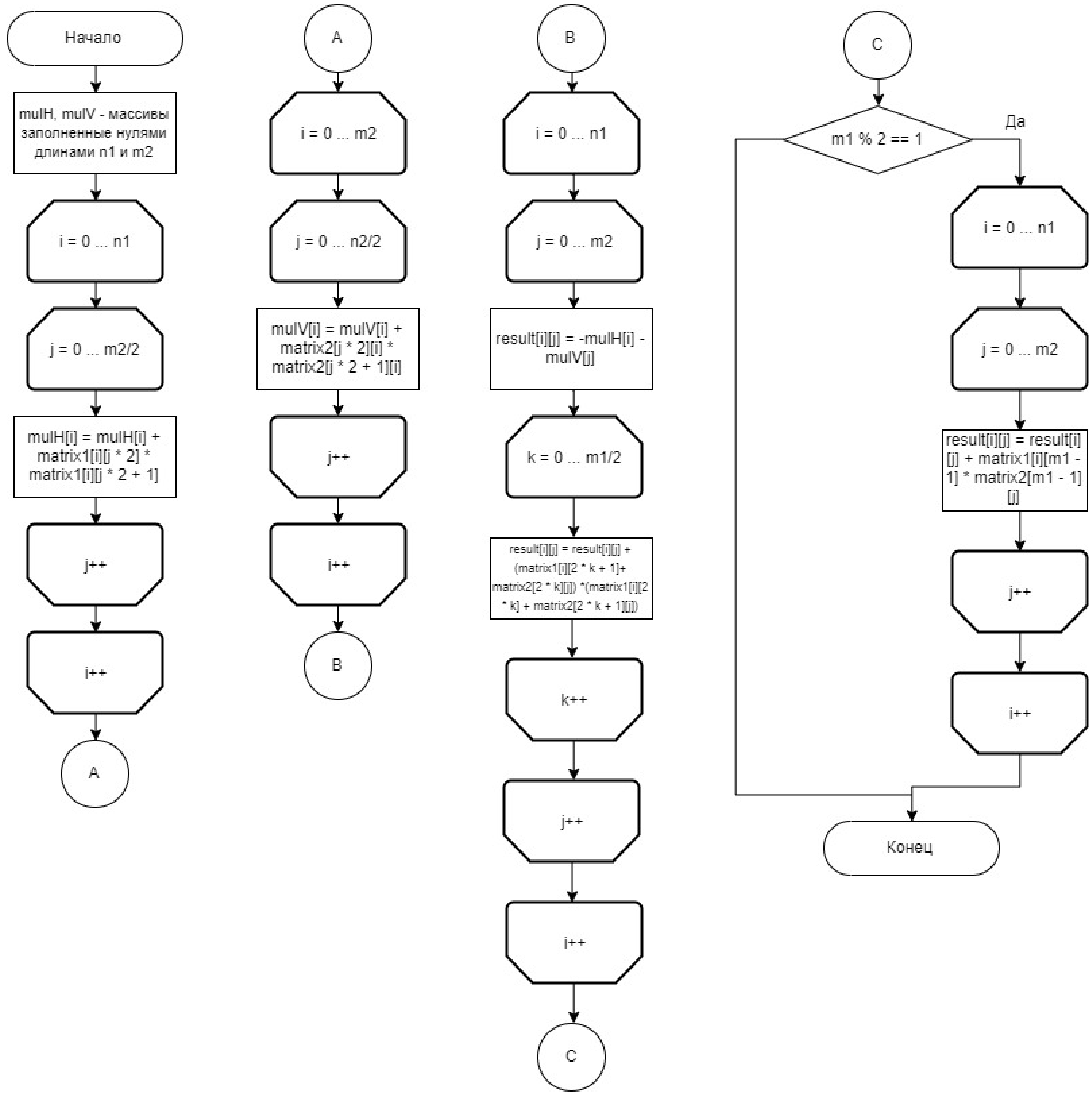

Рис. 2.2: Схема алгоритма Винограда

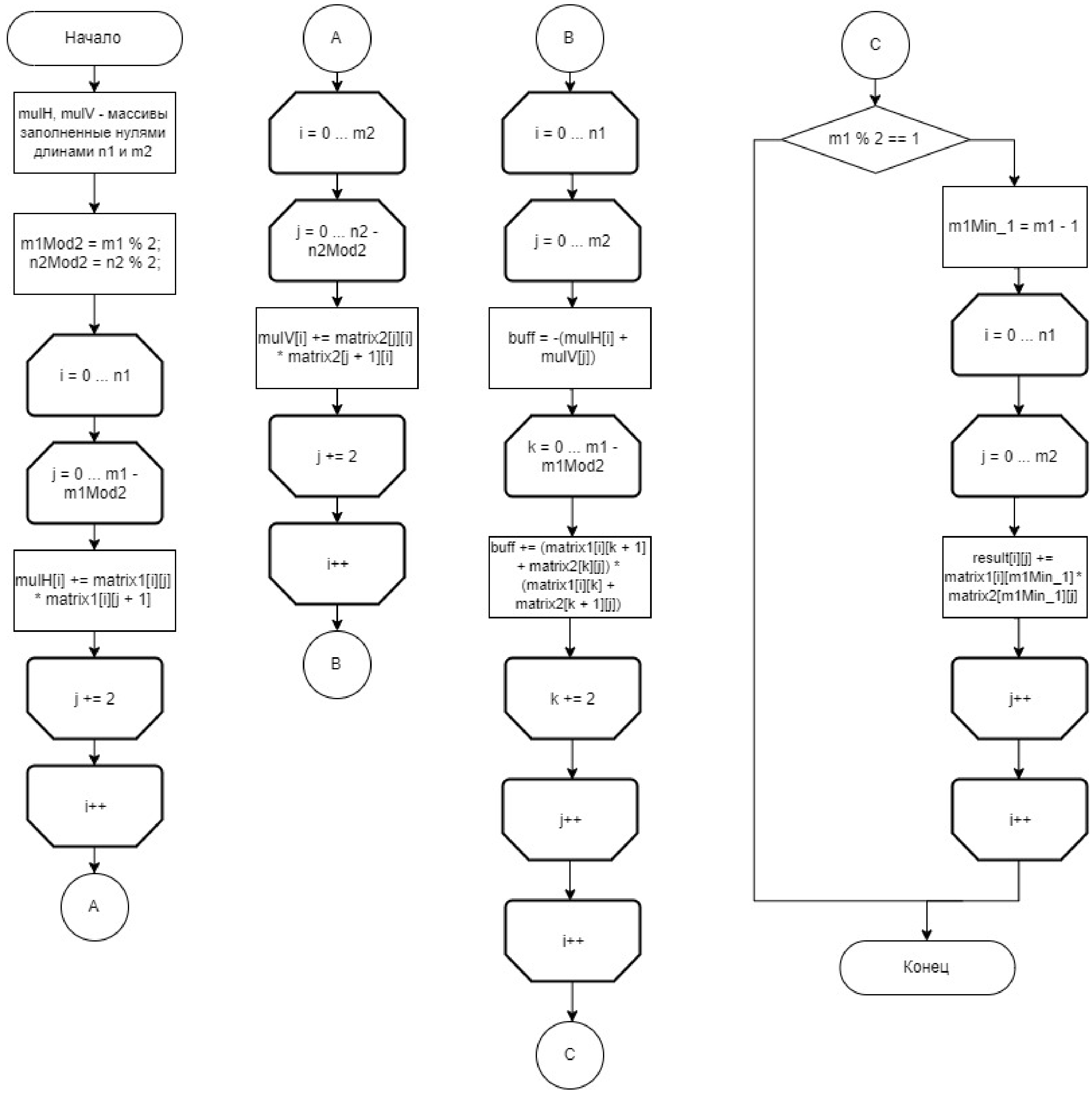

Рис. 2.3: Схема оптимизированного алгоритма Винограда

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [];
- оценка трудоемкости цикла for от 0 до N с шагом 1 $F_{for} = 2 + N \cdot (2 + F_{body})$, где F_{body} - тело цикла;
- стоимость условного перехода применим за 0, стоимость вычисления условия остаётся.

Оценим трудоемкость алгоритмов по коду программы.

2.2.1 Классический алгоритм

Рассмотрим трудоемкость классического алгоритма:

$10MNQ + 4MQ + 4M + 2$

2.2.2 Алгоритм Винограда

Рассмотрим трудоемкость алгоритма Винограда:

Трудоемкость алгоритма Винограда:

Первый цикл: $15/2 \cdot MN + 5 \cdot M + 2$

Второй цикл: $15/2 \cdot MN + 5 \cdot M + 2$

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

Условный переход: $\begin{bmatrix} 2 & \text{, в случае невыполнения условия} \\ 15 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

Итого: $15/2 \cdot MN + 5 \cdot M + 2 + 15/2 \cdot MN + 5 \cdot M + 2 + 13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2 + \begin{bmatrix} 2 & \text{, в случае невыполнения условия} \\ 15 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

2.2.3 Оптимизированный алгоритм Винограда

Рассмотрим трудоемкость алгоритма Винограда:

Трудоемкость алгоритма Винограда:

Первый цикл: $11/2 \cdot MN + 4 \cdot M + 2$

Второй цикл: $11/2 \cdot MN + 4 \cdot M + 2$

Третий цикл: $17/2 \cdot MNQ + 9 \cdot MQ + 4 \cdot M + 2$

Условный переход: $\begin{bmatrix} 1 & \text{, в случае невыполнения условия} \\ 10 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

Итого: $11/2 \cdot MN + 4 \cdot M + 2 + 11/2 \cdot MN + 4 \cdot M + 2 + 15/2 \cdot MNQ + 9 \cdot MQ + 4 \cdot M + 2 + \begin{bmatrix} 1 & \text{, в случае невыполнения условия} \\ 10 \cdot QM + 4 \cdot M + 2 & \text{, в случае выполнения условия} \end{bmatrix}$

3 Технологическая часть

3.1 Выбор ЯП

Для реализации программ был выбран язык программирования C++, ввиду наличия опыта разработки на нём. Среда разработки - Clion.

Для замера процессорного времени используется функция, возвращающая количество тиков.

Листинг 3.1: Функция получения тиков

```
unsigned long long getTicks(void)
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc" : "=A" (d) );
    return d;
}
```

3.2 Реализация алгоритма

Листинг 3.2: Функция классического умножения матриц

```
def MatrMult(m1, m2):
  r2 = len(m2)
  c1 = len(m1[0])

  if r2 != c1:
    return

```

```
  r1 = len(m1)
  c2 = len(m2[0])

  res = [[0 for i in range(c2)] for j in range(r1)]

  for i in range(r1):
    for j in range(c2):
      for k in range(c1):
        res[i][j] += m1[i][k] * m2[k][j]
  return res
```

Листинг 3.3: Алгоритм Винограда

```
def MatrMultVin(mx1, mx2):
  n1 = len(mx1)
  n2 = len(mx2)

  if (n1 == 0 or n2 == 0):
    return

  m1 = len(mx1[0])
  m2 = len(mx2[0])
  if (m1 == 0 or m2 == 0):
    return

  mulH = [0 for i in range(n1)]
  mulV = [0 for i in range(m2)]

  res = [[0 for i in range(m2)] for i in range(n1)]

  for i in range(n1):
    for j in range(int(m1 // 2)):
      mulH[i] = mulH[i] + mx1[i][j * 2] * mx1[i][j * 2 + 1]

  for i in range(m2):
    for j in range(int(m1 // 2)):
      mulV[i] = mulV[i] + mx2[j * 2][i] * mx2[j * 2 + 1][i]

  for i in range(n1):
    for j in range(m2):
      res[i][j] = -mulH[i] - mulV[j]
```

```
  for k in range(int(m1/2)):
    res[i][j] = res[i][j] + (mx1[i][2 * k + 1] + mx2[2 * k
        ][j]) * (mx1[i][2*k] + mx2[2 * k + 1][j])

  if m1 % 2 == 1:
    for i in range(n1):
      for j in range(m2):
        res[i][j] = res[i][j] + mx1[i][m1 - 2] * mx2[m1 -
            1][j]

  return(res)
```

Листинг 3.4: Оптимизированный алгоритм Винограда

```
def Vinograd_optim(matrix1, matrix2):
  n1 = len(matrix1)
  n2 = len(matrix2)

  if (n1 == 0 or n2 == 0):
    return

  m1 = len(matrix1[0])
  m2 = len(matrix2[0])
  if (m1 == 0 or m2 == 0):
    return

  mulH = [0 for i in range(n1)]
  mulV = [0 for i in range(m2)]

  result = [[0 for i in range(m2)] for i in range(n1)]

  m1Mod2 = m1 % 2;
  n2Mod2 = n2 % 2;

  for i in range(n1):
    for j in range (0, m1 - m1Mod2, 2):
      mulH[i] += matrix1[i][j] * matrix1[i][j + 1]


  for i in range(m2):
    for j in range(0, n2 - n2Mod2, 2):
```

```
      mulV[i] += matrix2[j][i] * matrix2[j + 1][i];

  for i in range(n1):
    for j in range(m2):
      buff = -(mulH[i] + mulV[j])
    for k in range(0, m1 - m1Mod2, 2):
      buff += (matrix1[i][k + 1] + matrix2[k][j]) * (
          matrix1[i][k] + matrix2[k + 1][j]);
    result[i][j] = buff;

  if (m1Mod2):
    m1Min_1 = m1 - 1
  for i in range(n1):
    for j in range(m2):
      result[i][j] += matrix1[i][m1Min_1] * matrix2[m1Min_1
          ][j]
  return result
```

3.2.1 Оптимизация алгоритма Винограда

В качестве оптимизации алгоритма Винограда можно выделить следующие пункты:

1. избавиться от деления в цикле;
2. замена $mulH[i] = mulH[i] + \ldots$ на $mulH[i] += \ldots$ (аналогично для $mulV[i]$);

Листинг 3.5: Оптимизации алгоритма Винограда №1 и №2

```
m1Mod2 = m1 % 2;
n2Mod2 = n2 % 2;

for i in range(n1):
  for j in range (0, m1 - m1Mod2, 2):
    mulH[i] += matrix1[i][j] * matrix1[i][j + 1]


for i in range(m2):
  for j in range(0, n2 - n2Mod2, 2):
    mulV[i] += matrix2[j][i] * matrix2[j + 1][i];
```

3. накопление результата в буфер, а вне цикла сброс буфера в ячейку матрицы;

Листинг 3.6: Оптимизации алгоритма Винограда №3

```
for i in range(n1):
  for j in range(m2):
    buff = -(mulH[i] + mulV[j])
    for k in range(0, m1 - m1Mod2, 2):
      buff += (matrix1[i][k + 1] + matrix2[k][j]) * (
          matrix1[i][k] + matrix2[k + 1][j]);
    result[i][j] = buff;
```

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 матрицах с размерами от 100 x 100 до 1000 x 1000 с шагом 100.

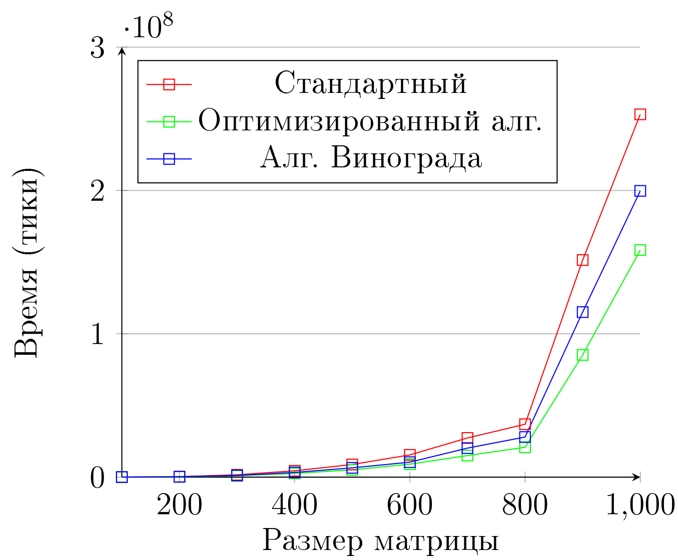

Рисунок 4.1. График времени работы алгоритмов на матрицах четной размерности

Второй эксперимент производится для худшего случая, когда поданы матрицы с нечетными размерами от 101 x 101 до 1001 x 1001 с шагом 100.

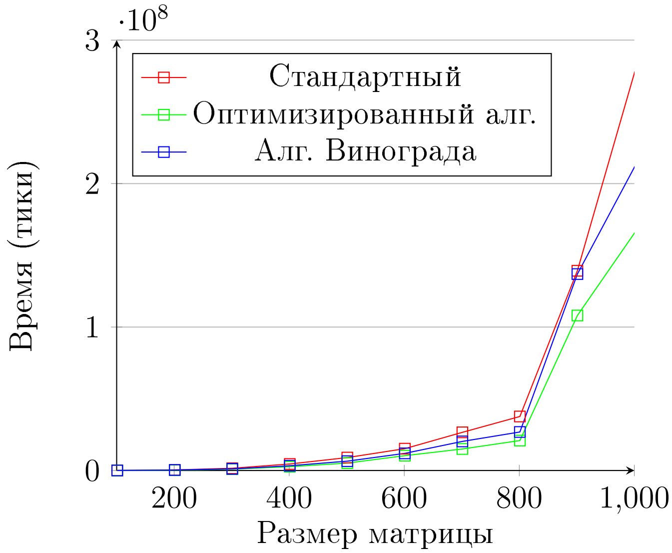

Рисунок 4.2. График времени работы алгоритмов на матрицах нечетной размерности

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

4.2 Тестовые данные

Проверка работы алгоритмов на примерах:

- матрицы размерностью 1 на 1
- матрицы размерностью 2 на 2
- матрицы размерностью 3 на 3

```
M1:
 6

M2:
 5

Результат работы обычного алгоритма умножения матриц:
30

Результат работы алгоритма винограда умножения матриц:
30

Результат работы алгоритма оптимизированного винограда умножения матриц:
30
```

Рис. 4.1: Пример работы алгоритма с матрицами размерностью 1 на 1

```
M1:
 6  1
 6  7

M2:
 7  5
 5  5

Результат работы обычного алгоритма умножения матриц:
47 35
77 65

Результат работы алгоритма винограда умножения матриц:
47 35
77 65

Результат работы алгоритма оптимизированного винограда умножения матриц:
47 35
77 65
```

Рис. 4.2: Пример работы алгоритма с матрицами размерностью 2 на 2

```
M1:
 5  9  5
 6  9  6
 7  8  1

M2:
 9  9  7
 7  5  7
 1  1  2

Результат работы обычного алгоритма умножения матриц:
113 95 108
123 105 117
120 104 107

Результат работы алгоритма винограда умножения матриц:
117 99 116
126 108 123
127 111 121

Результат работы алгоритма оптимизированного винограда умножения матриц:
113 95 108
123 105 117
120 104 107
```

Рис. 4.3: Пример работы алгоритма с матрицами размерностью 3 на 3

Заключение

В ходе лабораторной работы были изучены алгоритмы умножения матриц: стандартный и алгоритм Винограда, оптимизирован алгоритм Винограда, дана теоретическая оценка базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда, реализованы три алгоритма умножения матриц.

Список использованных источников

1. Алгоритм Копперсмита - Винограда [Электронный ресурс]. – Режим доступа: https://math.wikia.org/ru/wiki/Алгоритм-Копперсмита-Винограда. – Дата доступа: 16.10.2020.

2. Алгоритм Штрассена - Винограда [Электронный ресурс]. – Режим доступа: http://wikiredia.ru/wiki/Алгоритм-Винограда-Штрассена. – Дата доступа: 16.10.2020.

3. Умножение матриц. Эффективная реализация шаг за шагом [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/359272/. – Дата доступа: 16.10.2020.