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 дисциплине: Анализ Алгоритмов

Тема: Алгоритмы умножения матриц

Студент Казакова Э.М.

Группа ИУ7-56Б

Оценка (баллы) _____

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

Москва — 2021 г.

Содержание

Введение

Цель данной лабораторной работы: провести сравнительный анализ алгоритмов умножения матриц и получить навык оптимизации алгоритмов.

Задачи данной лабораторной работы:

1. дать математическое описание формулы расчета для двух алгоритмов: стандартного и алгоритма Винограда
2. реализовать стандартный алгоритм умножения матриц и алгоритм Винограда;
3. разработать оптимизированый алгоритм Винограда;
4. дать теоретическую оценку трудоемкости трем алгоритмам;
5. провести замеры процессорного времени работы реализаций всех трех алгоритмов при четных и нечетных размерностях.

1 Аналитическая часть

В данной части будут рассмотрены основные теоретические аспекты, связанные с алгоритмами умножения матриц, описания алгоритмов, формулы и оценки сложностей алгоритмов.

1.1 Описание классического алгоритма умножения матриц

Матрица – это прямоугольная таблица каких-либо элементов. Здесь и далее мы будем рассматривать только матрицы, элементами которых являются числа. Упорядоченная пара чисел (n, m), где n - количество строк в матрице, m - количество столбцов, называется размерностью матрицы, обозначается обычно m x n.
Пусть имеются две матрицы: A и B размерами n x l и l x m соответственно.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & ... & a_{1,l} \\ a_{2,1} & a_{2,2} & ... & a_{2,l} \\ ... & ... & ... & ... \\ a_{n,1} & a_{n,2} & ... & a_{n,l} \end{bmatrix}$$

$$\begin{bmatrix} b_{1,1} & b_{1,2} & ... & b_{1,m} \\ b_{2,1} & b_{2,2} & ... & b_{2,m} \\ ... & ... & ... & ... \\ b_{l,1} & b_{l,2} & ... & b_{l,m} \end{bmatrix}$$

Произведением матриц A и B размерами n x l и l x m соответственно называется матрица C размерами n x m, каждый элемент которой вычисляется по формуле 1:

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

$$\begin{bmatrix} c_{1,1} & b_{1,2} & ... & c_{1,m} \\ c_{2,1} & b_{2,2} & ... & c_{2,m} \\ ... & ... & ... & ... \\ c_{n,1} & c_{n,2} & ... & c_{n,m} \end{bmatrix}$$

Матрица C в стандартном алгоритме находится последовательным вычислением элементов с индексами i, j, $i = \overline{1, n}$, $j = \overline{1, m}$ по формуле 1.[1]

1.2 Описание алгоритма 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. Также некоторые вычисления можно произвести заранее, что ускорит выполнение алгоритма. 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

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \quad (2)$$

Это равенство можно переписать в виде

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \quad (3)$$

В Алгоритме Винограда используется скалярное произведение из формулы 3, в отличие от стандартного алгоритма.[2] Алгоритм Винограда позволяет выполнить предварительную обработку матрицы и запомнить значения для каждой строки/столбца матриц. Над предварительно обработанными элементами нам придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.3 Вычисление сложности алгоритма

В рамках данной работы используется следующая модель вычислений:

1. базовые операции имеют трудоемкость 1 (<, >, =, <=, =>, ==, +, -, *, /, %, &, +=, -=, *=, /=, [ ]);

2. операторы if, else if имеют трудоемкость $F_{if} = F_{body} + F_{chek}$, F_{body} - трудоемкость операций тела оператора, F_{chek} - трудоемкость проверки условия;

3. оператор else имеет трудоемкость F_{body};

4. оператор for имеет трудоемкость $F_{for} = 2 + N \cdot (F_{body} + F_{chek})$, где F_{body} – трудоемкость операций в теле цикла.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов умножения матриц: стандартного, Винограда и оптимизированного алгоритма Винограда.

2.1 Схемы алгоритмов

На рисунке 1 представлена схема классического алгоритма умножения матриц.

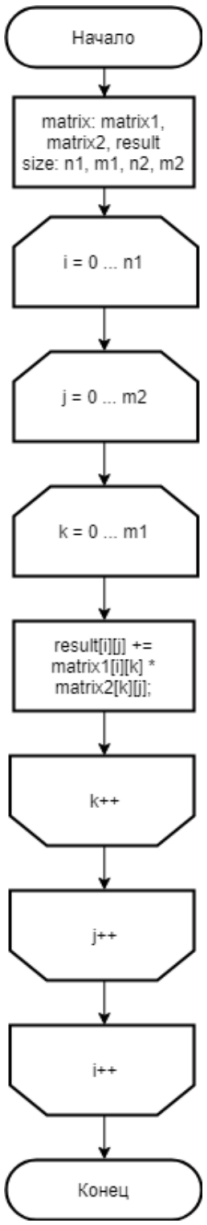

Рис. 1: Схема матричного алгоритма нахождения расстояния Левенштейна.

На рисунке 2 представлена схема алгоритма Винограда.

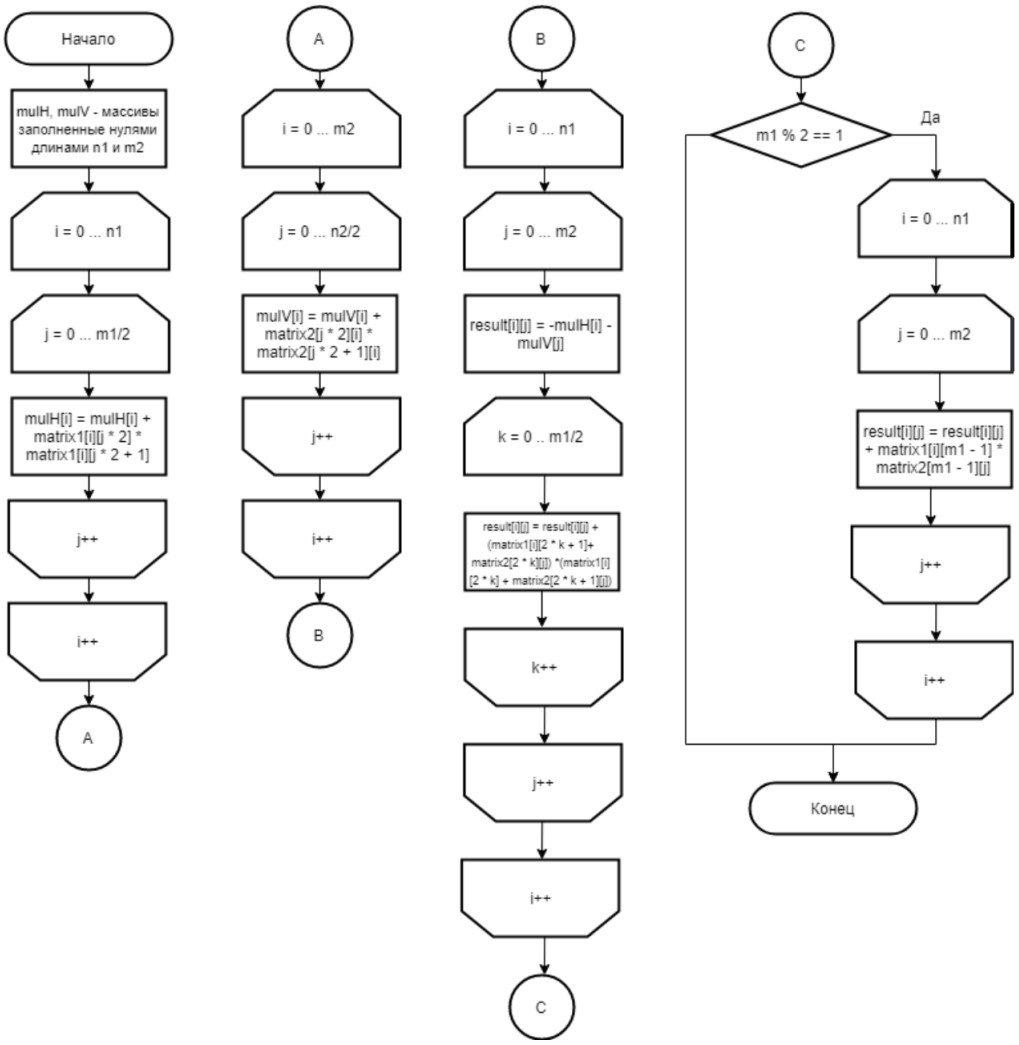

Рис. 2: Схема рекурсивного алгоритма нахождения расстояния Левенштейна.

На рисунке 3 представлена схема оптимизированного алгоритма Винограда.

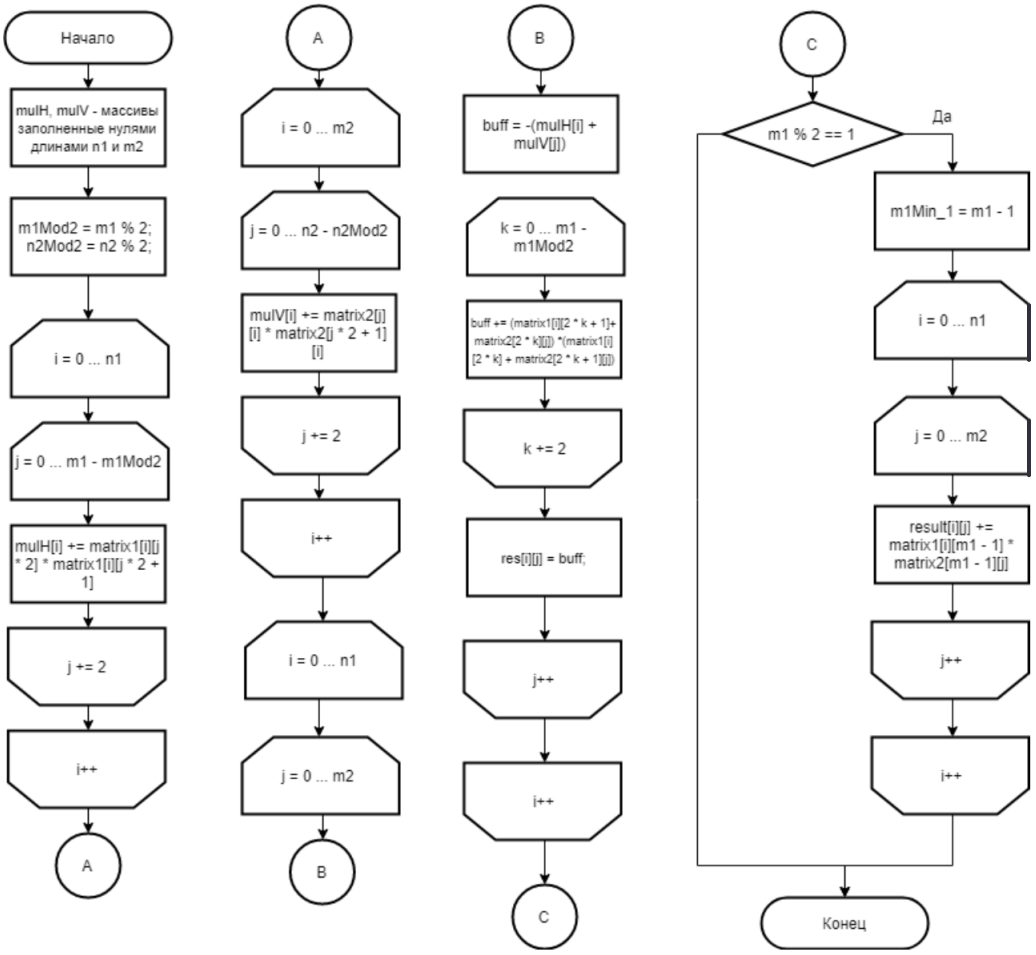

Рис. 3: Схема рекурсивного алгоритма нахождения расстояния ДамерауЛевенштейна.

3 Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации и представлен листинг кода.

3.1 Требования к программному обеспечению

ПО должно предоставлять возможность ввода двух матриц, на выходе пользователь должен получить результат умножения двух матриц, посчитанный тремя алгоритмами. Также ПО должно обеспечить вывод замеров времени работы каждого из алгоритмов.

3.2 Средства реализации

В данной работе используется язык программирования Python, так как ЯП позволяет написать программу за кратчайшее время.В качестве среды разработки выбрана IDLE. Для замеров времени была выбран метод process_time() модуля time[3], он возвращает значение (в долях секунды) системного процессорного времени текущего процесса.

3.3 Листинг кода

В листингах 3.1 - 3.3 представлена реализация стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
1  def standart_multiplication_matrix(m1, m2):
2          n = len(m1)
3          q = len(m2[0])
4          m = len(m1[0])
5          m3 = [[0] * q for i in range(n)]
6          start_time = time.process_time()
7          for i in range(0, n):
8                  for j in range(0, q):
9                          for k in range(0, m):
10                                 m3[i][j] = m3[i][j] + m1[i][k] * m2[k][j]
11         t = time.process_time() - start_time
12         return t
```

Листинг 3.2 – Алгоритм Винограда

```
1  def vinograd_multiplication_matrix(m1, m2):
2          m = len(m1)
3          n = len(m1[0])
4          q = len(m2[0])
5          m3 = [[0] * q for i in range(m)]
6
7          row = [0] * m
8          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]

        start_time = time.process_time()
        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]
        t = time.process_time() - start_time
        return t
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
def vinograd_optimizate_multiplication_matrix(m1, m2):
        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(1, n, 2):
                        row[i] -= m1[i][j] * m1[i][j - 1]

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

        flag = n % 2
        start_time = time.process_time()
        for i in range(0, m):
                for j in range(0, q):
                        m3[i][j] = row[i] + col[j]
                        for k in range(1, n, 2):
                                m3[i][j] += (m1[i][k - 1] + m2[k][j]) *
                                * (m1[i][k] + m2[k - 1][j])
                        if (flag):
```

```
25                                  m3[i][j] += m1[i][n - 1] * m2[n - 1][j]
26          t = time.process_time() - start_time
27          return t
```

3.4 Тестирование

На рисунке 4-5 показаны результаты работы программы при разных входных данных.

```
Введите размерность матрицы 1
m1 = 3
n1 = 2
Введите размерность матрицы 2
m2 = 2
n2 = 4

Матрица 1:
-7 10
-5 1
-2 9

Матрица 2:
-5 8 7 6
0 -5 -9 -10

Стандартный алгоритм:
35 -106 -139 -142
25 -45 -44 -40
10 -61 -95 -102

Алгоритм Винограда:
35 -106 -139 -142
25 -45 -44 -40
10 -61 -95 -102

Оптимизированный алгоритм Винограда:
35 -106 -139 -142
25 -45 -44 -40
10 -61 -95 -102
>>>
```

Рис. 4: Работа программы при умножении матриц с корректными размерами

```
Введите размерность матрицы 1
m1 = 2
n1 = 3
Введите размерность матрицы 2
m2 = 2
n2 = 3
Матрицы не могут быть перемножены
```

Рис. 5: Работа программы при некорректном задании размеров матрицы

3.5 Выводы

В данном разделе была представлена реализация алгоритмов стандартного умножения матриц, Винограда, а также оптимизированного алгоритма Винограда Программа корректно сработала на вводе матриц с размерами, которые не позволяют перемножить матрицы.

4 Экспериментальная часть

В данной части производится экспериментальное сравнение работы трех реализованных алгоритмов (зависимость времени выполнения от размеров матриц и четности/нечетности размеров).

4.1 Постановка эксперимента

В рамках данной лабораторной работы были проведены следующие эксперименты:

1. Сравнение времени работы алгоритмов при четном и нечетном размере матрицы

4.2 Сравнение времени работы

На графиках 6-7 представлено сравнение времени работы алгоритма на матрицах разных размеров.Матрицы заполняются случайно сгенерированными числами в диапазонеот -10 до 10. Для построения графика на рисунке 6 генерировались матрицы четной размерности. Для построения графика на рисунке 7 генерировались матрицы нечетной размерности.

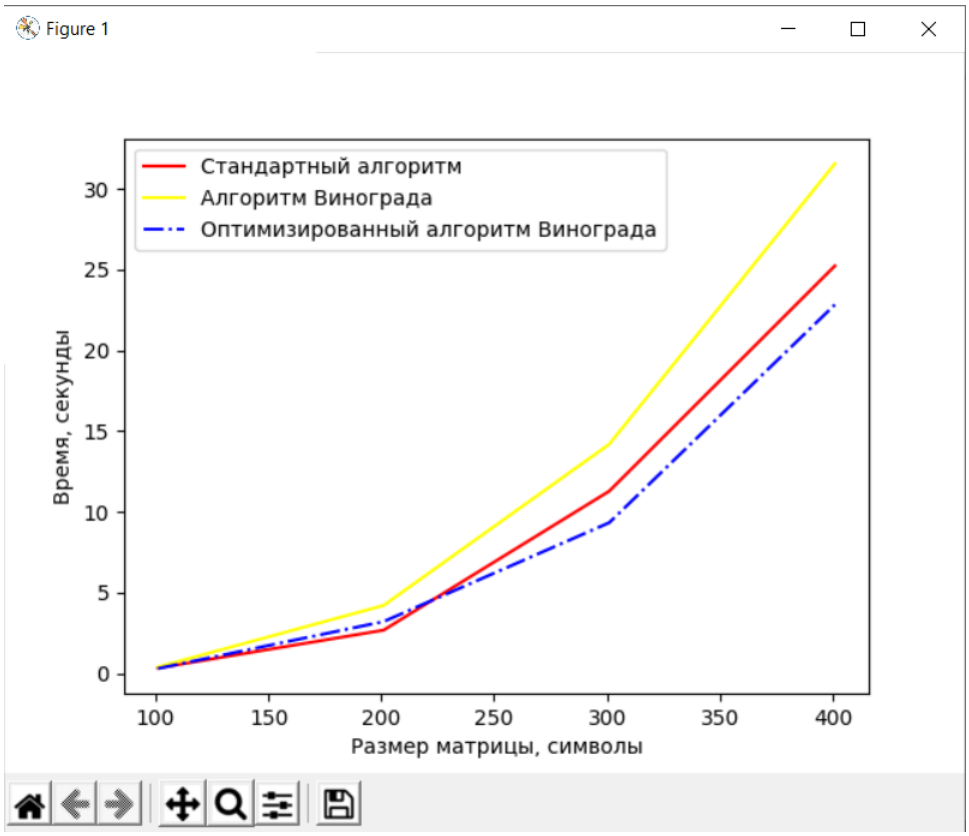

Рис. 6: Сравнение по времени работы реализации алгоритмов нахождения произведения матриц при четных размерностях

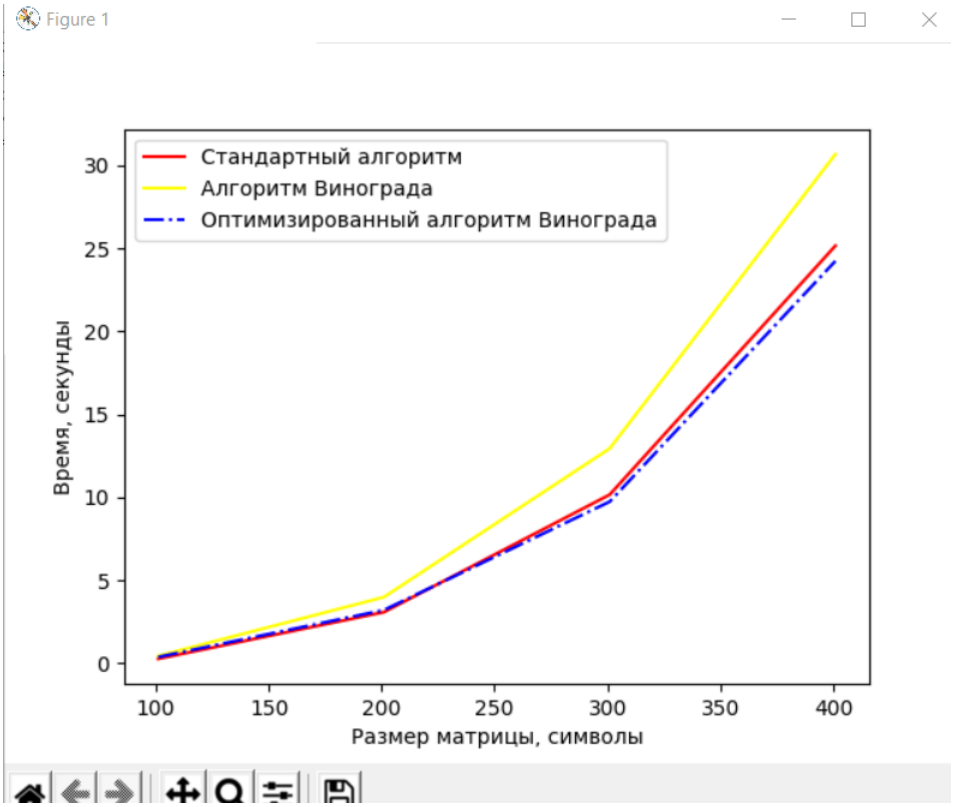

Рис. 7: Сравнение по времени реализации алгоритмов нахождения произведения матриц при нечетных размерностях

4.3 Оценка трудоемкости алгоритмов умножения матриц

Оценим трудоемкость стандартного алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда.

1. Стандартный алгоритм

   $f = 2+M(2+2+Q(2+2+N(2+8+1+1+1))) = 13 \cdot MNQ+4MQ+4M+2 \approx 13 \cdot MNQ$

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: $\begin{bmatrix} 2 & ,(N) \\ 15 \cdot QM + 4 \cdot M + 4 & , \end{bmatrix}$

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

$$f \approx 13 \cdot MNQ$$

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:

$$f = 4{\cdot}NQ{+}4{\cdot}M{+}2{+}4{\cdot}MN{+}4{\cdot}M{+}2{+}9{\cdot}MNQ{+}10{\cdot}MQ{+}4{\cdot}M{+}2{+}\begin{bmatrix} 2 & \text{, л.с} \\ 10 \cdot QM & \text{, х.с} \end{bmatrix} \approx 9{\cdot}MNQ$$

4.4 Выводы

Оптимизированный алгоритм Винограда работает быстрее классического метода и значительно быстрее обычного алгоритма Винограда.

Несмотря на сложность алгоритма Винограда по сравнению со стандартным, доля умножения в алгоритме Винограда меньше. Также, стоит обратить внимание на то, что при работе алгоритма Винограда с матрицами нечетной размерности, необходимо произвести дополнительные действия, в то время как алгоритм стандартный не зависит от четности размерности матриц.

Заключение

В ходе лабораторной работе были исследованы алгоритмы умножения матриц: стандартный, Винограда и оптимизированный алгоритм Винограда. При выполнении лабораторной работе цель была достигнута: был проведен сравнительный анализ алгоритмов умножения матриц и получен навык оптимизации алгоритмов. Также были выполнены следующие задачи:

1. были описаны формулы расчета для двух алгоритмов: стандартного и алгоритма Винограда;
2. были реализованы стандартный алгоритм умножения матриц и алгоритм Винограда;
3. был разработан оптимизированый алгоритм Винограда;
4. посчитана теоретическая оценка трудоемкости трех алгоритмов;
5. проведены замеры процессорного времени работы реализацийч трех алгоритмов при четных и нечетных размерностях;

При сравнении данных алгоритмов пришли к выводу, что классический алгоритм в является более эффективным чем алгоритм Винограда, однако после ряда оптимизаций, алгоритм Винограда становится значительно быстрее стандартного.

Список литературы

[1] Умножение матриц [Электронный ресурс]. - Режим доступа: https://habr.com/ru/post/359272/(дата обращения: 22.01.2021)

[2] Алгоритм Винограда [Электронный ресурс]. - Режим доступа: https://www.gyrnal.ru/statyi/ru/1153/ (дата обращения: 22.01.2021)

[3] Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.