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 «Анализы Алгоритмов»

на тему: «Алгоритмы умножения матриц»

Студент группы ИУ7-56Б _____ (Подпись, дата) Чупахин М. Д. (Фамилия И.О.)

Преподаватель _____ (Подпись, дата) Волкова Л. Л. (Фамилия И.О.)

Москва — 2023 г.

Содержание

Введение

Данная лабораторная работа заслуживает особого внимания, так как она фокусируется на изучении методов перемножения матриц. В области программирования и математики матрицы широко применяются, и их использование находит место в различных областях. Одним из основных применений матриц является их использование при выводе формул в физике, таких как:

— градиент;

— дивергенция;

— ротор.

Кроме того, необходимо учитывать разнообразные операции, выполняемые над матрицами, такие как сложение, возведение в степень и умножение. В зависимости от задачи, размеры матриц могут быть значительными, поэтому оптимизация операций с матрицами является важным аспектом программирования. В данной лабораторной работе будет рассмотрена оптимизация операции умножения матриц.

Главной целью данной лабораторной работы является описание, реализация и исследование алгоритмов умножения матриц.

Ниже приведен набор задач, выполнение которого необходимо для достижения поставленной цели.

1) Провести описание двух алгоритмов умножения матриц.

2) Разработать программное обеспечение, реализующее следующие алгоритмы:

   — классический алгоритм умножения матриц;

   — алгоритм Винограда;

   — оптимизированный алгоритм Винограда.

3) Провести анализ временных характеристик работы программы и выявить факторы, влияющие на эффективность ее выполнения.

4) Сравнить производительность различных алгоритмов.

1 Аналитическая часть

В данном разделе мы рассмотрим два метода умножения матриц: классический алгоритм умножения матриц и алгоритм Винограда, включая его оптимизированную версию.

1.1 Матрица

Матрица [1] представляет собой таблицу чисел a_{ik}, которая имеет вид:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \tag{1.1}$$

где m – количество строк, а n – количество столбцов. Элементы a_{ik} называются элементами матрицы.

Если A – матрица, то $A_{i,j}$ обозначает элемент матрицы, который находится в i-й строке и j-м столбце.

Существуют следующие операции над матрицами:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц, если количество столбцов первой матрицы равно количеству строк второй матрицы. Результатом будет матрица с количеством строк, равным первой матрице, и столбцов, равным второй матрице.

Заметим, что операция умножения матриц не коммутативна: если A и B-квадратные матрицы, и C – их произведение, то AB и BA могут дать разные результаты C.

1.2 Классический алгоритм

Допустим, у нас есть две матрицы:

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.2}$$

тогда матрица C

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.3}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.4}$$

называется произведением матриц A и B [1].

Классический алгоритм реализует эту формулу 1.4.

1.3 Алгоритм Винограда

Алгоритм Винограда [2] – это метод умножения квадратных матриц. В 1987 году Дон Копперсмит и Виноград представили метод, который имеет асимптотическую сложность $O(n^{2,3755})$, и затем улучшили его в 2011 году до $O(n^{2,373})$, где n - размер стороны матрицы. После оптимизации этот метод стал самым эффективным среди всех алгоритмов умножения матриц.

Рассмотрим пример с двумя векторами $U = (u_1, u_2, u_3, u_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно: $U \cdot W = u_1w_1 + u_2w_2 + u_3w_3 + u_4w_4$, что можно переписать как:

$$U \cdot W = (u_1+w_2)(u_2+w_1)+(u_3+w_4)(u_4+w_3)-u_1u_2-u_3u_4-w_1w_2-w_3w_4 \quad (1.5)$$

Теперь представим, что у нас есть матрицы A, B, C определенных размеров. Скалярное произведение, предложенное Виноградом, можно выразить следующим образом:

$$C_{ij} = \sum_{k=1}^{q/2}(a_{i,2k-1}+b_{2k,j})(a_{i,2k}+b_{2k-1,j})-\sum_{k=1}^{q/2} a_{i,2k-1}a_{i,2k}-\sum_{k=1}^{q/2} b_{2k-1,j}b_{2k,j} \quad (1.6)$$

Хотя это выражение требует больше арифметических операций, чем стандартное умножение матриц, Виноград предложил предварительно вычислять второе и третье слагаемые в уравнении 1.6 для каждой строки матрицы A и каждого столбца матрицы B. Это позволяет значительно снизить количество вычислений в последующих шагах. В итоге на практике алгоритм Винограда может работать быстрее, так как операция сложения выполняется быстрее, чем умножение.

Также стоит отметить, что при нечетных размерах матрицы требуется дополнительно учитывать произведение крайних элементов соответствующих строк и столбцов.

1.4 Оптимизированный алгоритм Винограда

При реализации алгоритма Винограда можно внести следующие оптимизации:

1) Значение $\frac{N}{2}$, используемое как ограничение в цикле для вычисления предварительных данных, можно кэшировать;

2) Операцию умножения на 2 можно эффективно заменить побитовым

сдвигом на 1;

3) Операции сложения и вычитания с присваиванием следует реализовывать с использованием соответствующих операторов $+=$ или $-=$.

Вывод

В данном разделе мы рассмотрели два алгоритма умножения матриц: классический и алгоритм Винограда, включая оптимизации для последнего. Мы также обсудили особенности умножения матриц и важность предварительных вычислений для улучшения производительности алгоритма.

2 Конструкторская часть

2.1 Введение

В данном разделе представлены схемы алгоритмов умножения матриц: стандартного, алгоритма Винограда и его оптимизированной версии, а также их теоретическая оценка трудоемкости.

2.2 Разработка алгоритмов

В данном разделе представлены схемы алгоритмов для стандартного умножения матриц (рисунок 2.1), алгоритма Винограда умножения матриц (рисунки 2.2 и 2.3) и оптимизированного алгоритма Винограда (рисунки 2.4 и 2.5).

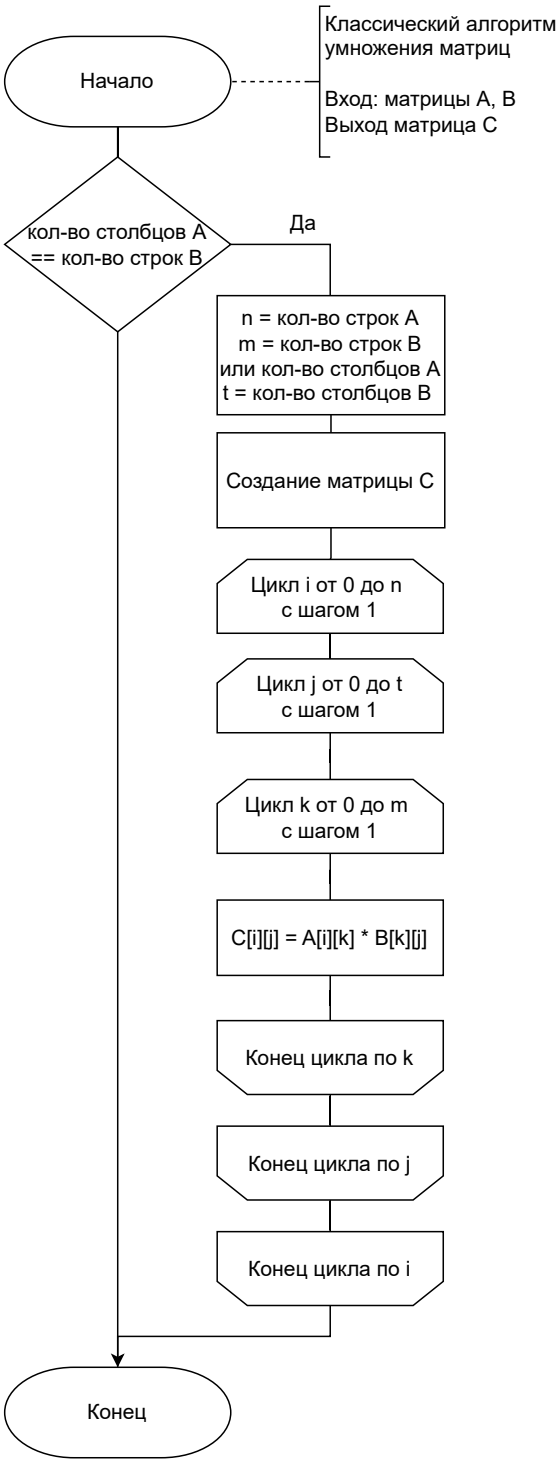

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

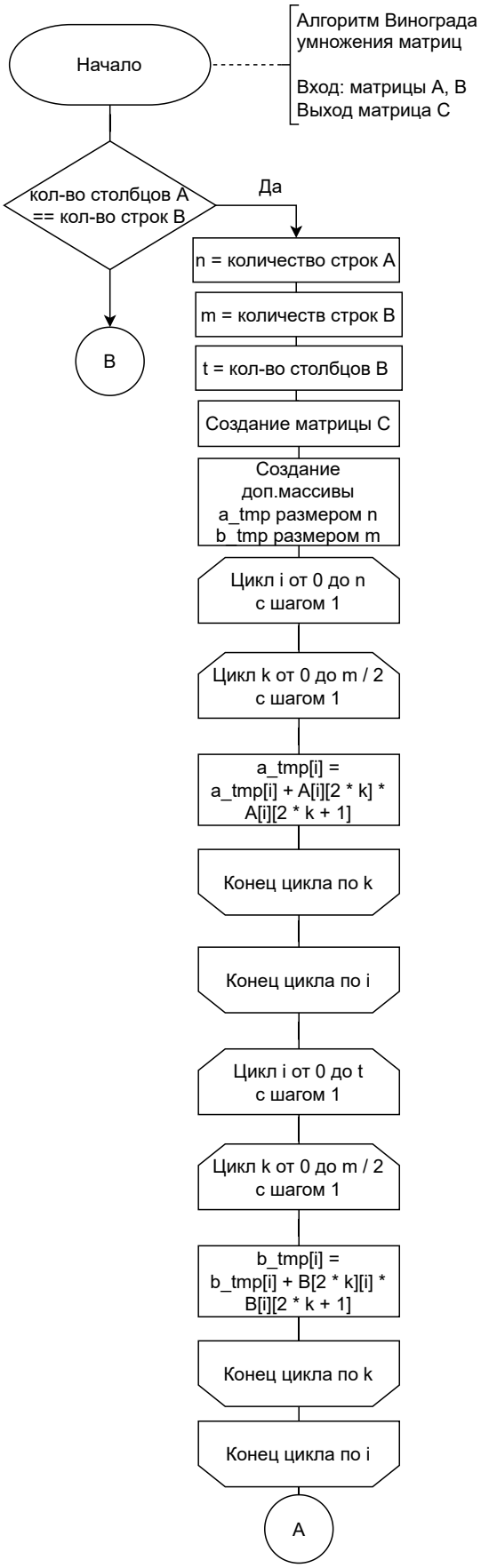

Рисунок 2.2 – Схема умножения матриц по алгоритму Винограда (Часть 1)

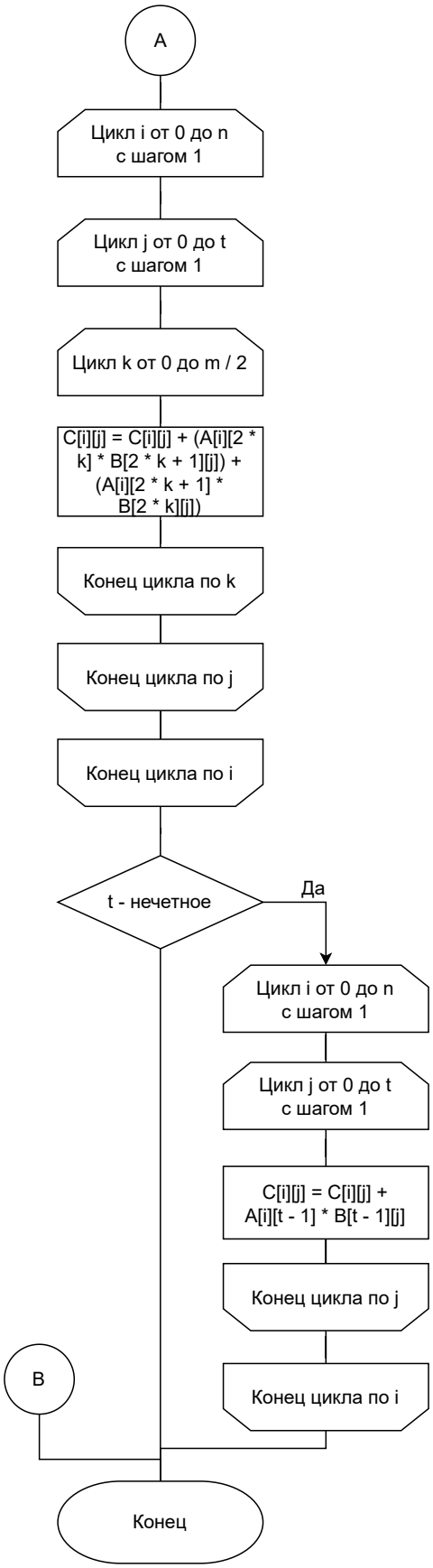

Рисунок 2.3 – Схема умножения матриц по алгоритму Винограда (Часть 2)

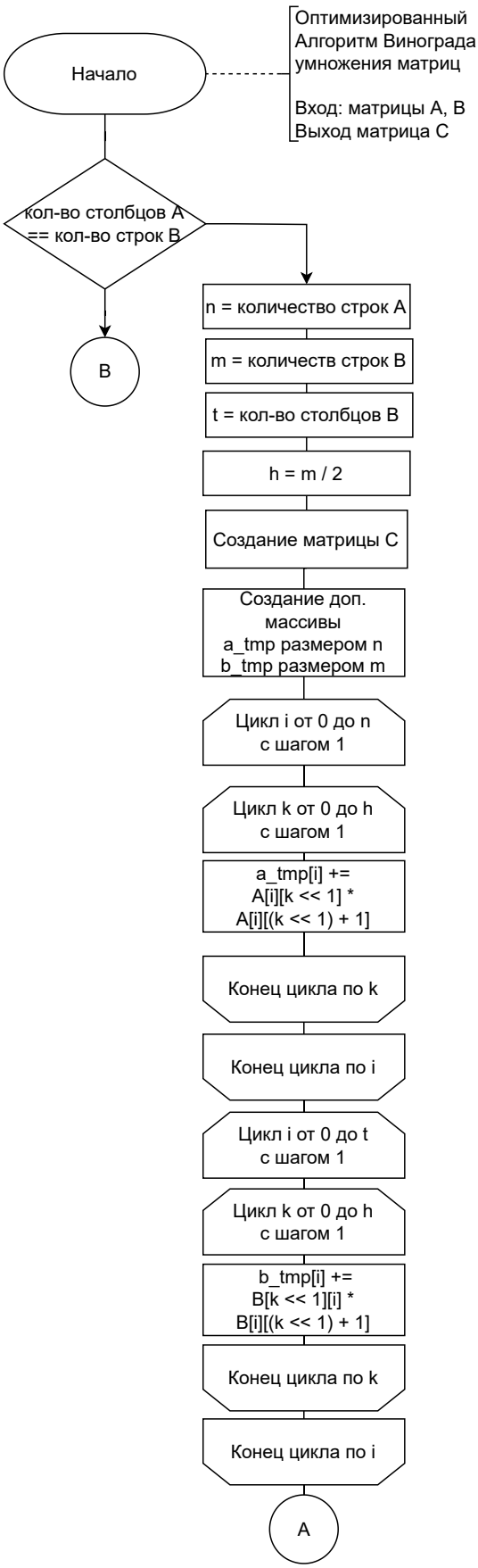

Рисунок 2.4 – Схема умножения матриц по оптимизированному алгоритму Винограда (Часть 1)

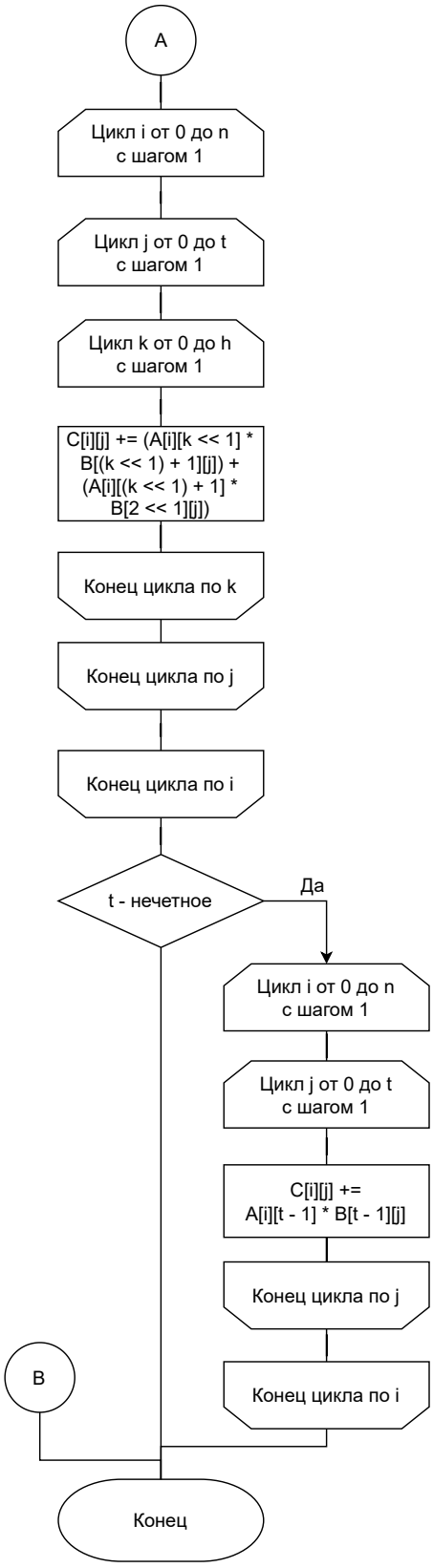

Рисунок 2.5 – Схема умножения матриц по оптимизированному алгоритму Винограда (Часть 2)

2.3 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма умножения матриц.

1) Трудоемкость базовых операций имеет:

   — равную 1:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

   — равную 2:

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

2) Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай} \\ max(f_1, f_2), & \text{худший случай} \end{cases} \tag{2.3}$$

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

4) Трудоемкость передачи параметра в функции и возврат из функции равны 0.

2.4 Трудоемкость алгоритмов

Была рассчитана трудоемкость алгоритмов умножения матриц.

2.4.1 Классический алгоритм

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из:

— внешнего цикла по $i \in [1 \dots N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;

— цикла по $j \in [1 \dots T]$, трудоёмкость которого: $f = 2+2+T \cdot (2+f_{body})$;

— цикла по $k \in [1 \dots M]$, трудоёмкость которого: $f = 2 + 2 + 14M$;

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

$$\begin{aligned} f_{standart} &= 2 + N \cdot (2 + 2 + T \cdot (2 + 2 + M \cdot (2 + 8 + 1 + 1 + 2))) = \\ &= 2 + 4N + 4NT + 14NMT \approx 14NMT = O(N^3) \end{aligned} \tag{2.5}$$

2.4.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, нужно учесть следующее:

— создания и инициализации массивов a_{tmp} и b_{tmp}, трудоёмкость которых указана в формуле (2.6);

$$f_{init} = N + M \tag{2.6}$$

— заполнения массива a_{tmp}, трудоёмкость которого указана в формуле (2.7);

$$\begin{gathered} f_{atmp} = 2 + N \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 4N + \frac{19NM}{2} = 2 + 4N + 9,5NM \end{gathered} \tag{2.7}$$

— заполнения массива b_{tmp}, трудоёмкость которого указана в формуле (2.8);

$$\begin{gathered} f_{btmp} = 2 + T \cdot (4 + \frac{M}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 4T + \frac{19TM}{2} = 2 + 4T + 9,5TM \end{gathered} \tag{2.8}$$

— цикла заполнения для чётных размеров, трудоёмкость которого указана в формуле (2.9);

$$\begin{gathered} f_{cycle} = 2 + N \cdot (4 + T \cdot (2 + 7 + 4 + \frac{M}{2} \cdot (4 + 28))) = \\ = 2 + 4N + 13NT + \frac{32NTM}{2} = 2 + 4N + 13NT + 16NTM \end{gathered} \tag{2.9}$$

— цикла, который дополнительно нужен для подсчёта значений при нечётном размере матрицы, трудоемкость которого указана в фор-

муле (2.10);

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + T \cdot (2 + 14)), & \text{иначе} \end{cases} \quad (2.10)$$

Тогда для худшего случая (нечётный общий размер матриц) имеем:

$$f_{worst} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3) \quad (2.11)$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f_{best} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMT = O(N^3) \quad (2.12)$$

2.4.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

— кэширования значения $\frac{M}{2}$ в циклах, которое равно 3;

— создания и инициализации массивов a_{tmp} и b_{tmp} (2.6);

— заполнения массива a_{tmp}, трудоёмкость которого (2.7);

— заполнения массива b_{tmp}, трудоёмкость которого (2.8);

— цикла заполнения для чётных размеров, трудоёмкость которого указана в формуле (2.13);

$$\begin{aligned} f_{cycle} = 2 + N \cdot (4 + T \cdot (4 + 7 + \frac{M}{2} \cdot (2 + 10 + 5 + 2 + 4))) = \\ = 2 + 4N + 11NT + \frac{23NTM}{2} = \\ = 2 + 4N + 11NT + 11,5 \cdot NTM \end{aligned} \quad (2.13)$$

— условие, которое нужно для дополнительных вычислений при нечёт-

ном размере матрицы, трудоемкость которого указана в формуле (2.14);

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + T \cdot (2 + 10)), & \text{иначе} \end{cases} \tag{2.14}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем:

$$f_{worst} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3) \tag{2.15}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f_{best} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMT = O(N^3) \tag{2.16}$$

Вывод

В данном разделе были представлены схемы алгоритмов умножения матриц и проведена теоретическая оценка трудоемкости алгоритмов. Результаты показывают, что оптимизированный алгоритм Винограда позволяет добиться уменьшения трудоемкости в 1.2 раза.

3 Технологическая часть

В данной главе представлены требования к программному обеспечению, описаны средства реализации, приведены листинги кода и функциональные тесты.

3.1 Требования к программному обеспечению

Программное обеспечение должно удовлетворять следующим функциональным требованиям: на входе – две матрицы, на выходе – произведение этих матриц.

Программное обеспечение также должно соответствовать следующим требованиям:

— наличие пользовательского интерфейса для выбора действий;

— вывод результата умножения матриц

— предоставление функционала для измерения времени выполнения алгоритмов умножения матриц.

3.2 Средства реализации

Для разработки данной лабораторной работы был выбран язык программирования Swift [3]. Этот выбор обусловлен возможностью измерения процессорного времени [4] и соответствием с выдвинутыми техническими требованиям.

Измерение времени выполнения алгоритмов производится с использованием функции *clock_gettime()* [4].

3.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

— Количество строк – целое число;

— Количество столбцов – целое число;

— Матрица – двумерный список целых чисел.

3.4 Сведения о модулях программы

Программа разбита на следующие модули:

— `main.swift` — точка входа в программу, где происходит вызов алгоритмов через интерфейс;

— `Algorithms.swift` — содержит реализации алгоритмов умножения матриц (стандартный, алгоритм Винограда, оптимизированный алгоритм Винограда);

— `CPUTimeMeasure.swift` — измеряет время работы алгоритмов с учетом заданного количества повторений;

— `GraphRenderer.swift` — Строит графики для каждого из алгоритмов с учетом заданного количества повторений для каждого алгоритма;

3.5 Реализация алгоритмов

В листингах 3.1 – 3.4 приведены реализации алгоритмов умножения матриц: стандартный алгоритм, алгоритм Винограда, оптимизированный алгоритм Винограда.

В листингах 3.6 – 3.7 приведены реализации ввода и вывода матрицы.

Листинг 3.1 – Стандартный алгоритм умножения матриц

```
1     private static func standartMultiply(_ matrix1: [[Int]], _
          matrix2: [[Int]]) -> [[Int]] {
2
3         guard !matrix1.isEmpty && !matrix2.isEmpty else {
              return [] }
4         guard matrix1[0].count == matrix2.count else { return
              [] }
5
6         let n = matrix1.count
7         let m = matrix2.count
8         let t = matrix2[0].count
9
10        var res: [[Int]] = Array(repeating: Array(repeating: 0,
              count: t), count: n)
11
12        for i in 0..<n {
13            for j in 0..<t {
14                for k in 0..<m {
15                    res[i][j] += matrix1[i][k] * matrix2[k][j];
16                }
17            }
18        }
19
20        return res;
21    }
```

Листинг 3.2 – Алгоритм Винограда

```
1     private static func vinogradMultiply(_ matrix1: [[Int]], _
          matrix2: [[Int]]) -> [[Int]] {
2
3         guard !matrix1.isEmpty && !matrix2.isEmpty else {
              return [] }
4         guard matrix1[0].count == matrix2.count else { return
              [] }
5
6         let n = matrix1.count
7         let m = matrix2.count
8         let t = matrix2[0].count
9         var res = Array(repeating: Array(repeating: 0, count:
              t), count: n)
10        var rowSum = Array(repeating: 0, count: n)
11        var colSum = Array(repeating: 0, count: t)
12
13        for i in 0..<n {
14            for j in 0..<(m / 2) {
15                rowSum[i] = rowSum[i] + matrix1[i][2 * j] *
                      matrix1[i][2 * j + 1]
16            }
17        }
18
19        for i in 0..<t {
20            for j in 0..<(m / 2) {
21                colSum[i] = colSum[i] + matrix2[2 * j][i] *
                      matrix2[2 * j + 1][i]
22            }
23        }
24
25        for i in 0..<n {
26            for j in 0..<t {
27                res[i][j] = -rowSum[i] - colSum[j]
28                for k in 0..<(m / 2) {
29                    res[i][j] = res[i][j] + (matrix1[i][2 * k +
                          1] + matrix2[2 * k][j]) * (matrix1[i][2
                          * k] + matrix2[2 * k + 1][j])
30                }
31            }
32        }
```

Листинг 3.3 – Продолжение листинга 3.2

```
        if m % 2 == 1 {
            for i in 0..<n {
                for j in 0..<t {
                    res[i][j] = res[i][j] + matrix1[i][m - 1] *
                        matrix2[m - 1][j]
                }
            }
        }

        return res
    }
```

Листинг 3.4 – Оптимизированный алгоритм Винограда

```
1     private static func vinogradOptimizedMultiply(_ matrix1:
          [[Int]], _ matrix2: [[Int]]) -> [[Int]] {
2
3         guard !matrix1.isEmpty && !matrix2.isEmpty else {
              return [] }
4         guard matrix1[0].count == matrix2.count else { return
              [] }
5
6         let n = matrix1.count
7         let m = matrix2.count
8         let t = matrix2[0].count
9         let halfm = m / 2
10        var res = Array(repeating: Array(repeating: 0, count:
              t), count: n)
11        var rowSum = Array(repeating: 0, count: n)
12        var colSum = Array(repeating: 0, count: t)
13
14        for i in 0..<n {
15            for j in 0..<halfm {
16                rowSum[i] += matrix1[i][j << 1] * matrix1[i][j
                      << 1 + 1]
17            }
18        }
19
20        for i in 0..<t {
21            for j in 0..<halfm {
22                colSum[i] += matrix2[j << 1][i] * matrix2[j <<
                      1 + 1][i]
23            }
24        }
25
26        for i in 0..<n {
27            for j in 0..<t {
28                res[i][j] = -rowSum[i] - colSum[j]
29                for k in 0..<halfm {
30                    res[i][j] += (matrix1[i][k << 1 + 1] +
                          matrix2[k << 1][j]) * (matrix1[i][k <<
                          1] + matrix2[k << 1 + 1][j])
31                }
32            }
```

Листинг 3.5 – Продолжение листинга 3.4

```
            }
        }

        if m % 2 == 1 {
            for i in 0..<n {
                for j in 0..<t {
                    res[i][j] += matrix1[i][m - 1] * matrix2[m
                        - 1][j]
                }
            }
        }

        return res
    }
```

Листинг 3.6 – Функция ввода матрицы

```
1     private static func inputMatrix() -> [[Int]]? {
2         print(Welcome.enterNumberOfRows)
3         let numRows = Int(readLine()!)
4         guard let numRows else {
5             print(Errors.invalidNumberInput)
6             return nil
7         }
8         guard numRows > 0 else {
9             print(Errors.belowZero)
10            return nil
11        }
12
13        print(Welcome.enterNumberOfColumns)
14        let numCols = Int(readLine()!)
15        guard let numCols else {
16            print(Errors.invalidNumberInput)
17            return nil
18        }
19        guard numCols > 0 else {
20            print(Errors.belowZero)
21            return nil
22        }
23
24        var matrix = Array(repeating: Array(repeating: 0,
              count: numCols), count: numRows)
25        for i in 0..<numRows {
26            for j in 0..<numCols {
27                print("Enter element at row \(i + 1), column
                      \(j + 1): ")
28                var value = Int(readLine()!)
29                guard let value else {
30                    print(Errors.invalidNumberInput)
31                    return nil
32                }
33                matrix[i][j] = value
34            }
35        }
36
37        return matrix
38    }
```

Листинг 3.7 – Функция вывода матрицы

```
1     private static func printMatrix(_ matrix: [[Int]]) {
2         for row in matrix {
3             for element in row {
4                 let formattedElement = String(format: "%4d",
                      element)
5                 print(formattedElement, terminator: " ")
6             }
7             print()
8         }
9     }
```

3.6 Функциональные тесты

В таблице приведены функциональные тесты для алгоритмов умножения матриц. Все тесты были успешно пройдены.

Таблица 3.1 – Тесты для умножения матриц

Входная матрица A	**Входная матрица B**	**Ожидаемый результат**
$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$	$C = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$
$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$	$B = \begin{bmatrix} 1 & 3 \\ -1 & 2 \end{bmatrix}$	$C = \begin{bmatrix} 2 & 6 \\ -1 & 2 \end{bmatrix}$
$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$	$C = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$
$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$B = \begin{bmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \\ 11 & 12 & 13 \end{bmatrix}$	**Ошибка**
$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 & 3 \end{bmatrix}$	$B = \begin{bmatrix} 1 & 3 \\ -1 & 2 \end{bmatrix}$	**Ошибка**

Вывод

В данной главе были представлены требования к программному обеспечению, описаны средства реализации, приведены листинги кода алгоритмов и функциональные тесты, подтверждающие корректность работы алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени:

— Процессор: Apple M1 Pro [5]

— Оперативная память: 32 ГБайт.

— Операционная система: macOS Ventura 13.5.2. [6]

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

На изображении 4.1 представлена иллюстрация работы разработанного программного продукта. Конкретно, демонстрируются результаты выполнения алгоритмов для вычисления произведения двух матриц.

```
Menu:
1. Multiply matrices
2. Measure CPU time
3. Draw graphs
0. Exit

Choose Menu option: 1
Enter the first matrix:
Enter number of rows
2
Enter number of columns
2
Enter element at row 1, column 1:
1
Enter element at row 1, column 2:
2
Enter element at row 2, column 1:
3
Enter element at row 2, column 2:
4
Enter the second matrix:
Enter number of rows
2
Enter number of columns
2
Enter element at row 1, column 1:
4
Enter element at row 1, column 2:
3
Enter element at row 2, column 1:
2
Enter element at row 2, column 2:
1
Performing multiplication:
Standart Multiply
   8    5
  20   13
Vinograd Multiply
   8    5
  20   13
Vinograd Optimized Multiply
   8    5
  20   13
```

Рисунок 4.1 – Демонстрация работы программы при вычислении произведения матриц

4.3 Анализ временных характеристик

В данном разделе представлены результаты экспериментов, в которых измерялось время выполнения операции умножения матриц. Данные результаты представлены в таблицах 4.1 и 4.2.

Таблица 4.1 содержит результаты замеров времени выполнения алгоритмов умножения матриц для четных размеров квадратных матриц в диапазоне от 4 до 100 с шагом 6, применяя различные наборы входных данных.

Таблица 4.1 – Результаты замеров времени (четные размеры матриц)

Размер матрицы	**Время, мкс**		
	Классический	**Винограда**	**(опт.) Винограда**
4.0	49.18	49.78	49.7
10.0	608.8	452.56	442.16
16.0	2391.6	1596.08	1545.68
22.0	6014.22	3881.72	3750.78
28.0	12252.06	7784.64	7439.32
34.0	21937.68	13191.08	12734.44
40.0	34286.22	21011.48	20346.32
46.0	51940.4	31802.48	31114.14
52.0	75728.78	46198.98	44467.02
58.0	106426.24	63551.26	60865.68
64.0	140228.2	84930.5	81699.24
70.0	186538.56	110360.48	104340.26
76.0	234262.2	140966.54	132992.86
82.0	296745.06	176075.2	166776.0
88.0	365488.5	217110.78	207658.32
94.0	443201.06	263415.24	251650.72
100.0	538818.92	316688.32	305633.76

На основе данных из таблицы 4.1 был построен график (см. рисунок 4.2), который позволяет сделать вывод о том, что оптимизированный алгоритм Винограда работает наилучшим образом, в то время как классический алгоритм умножения матриц отстает примерно на 1.5 раза.

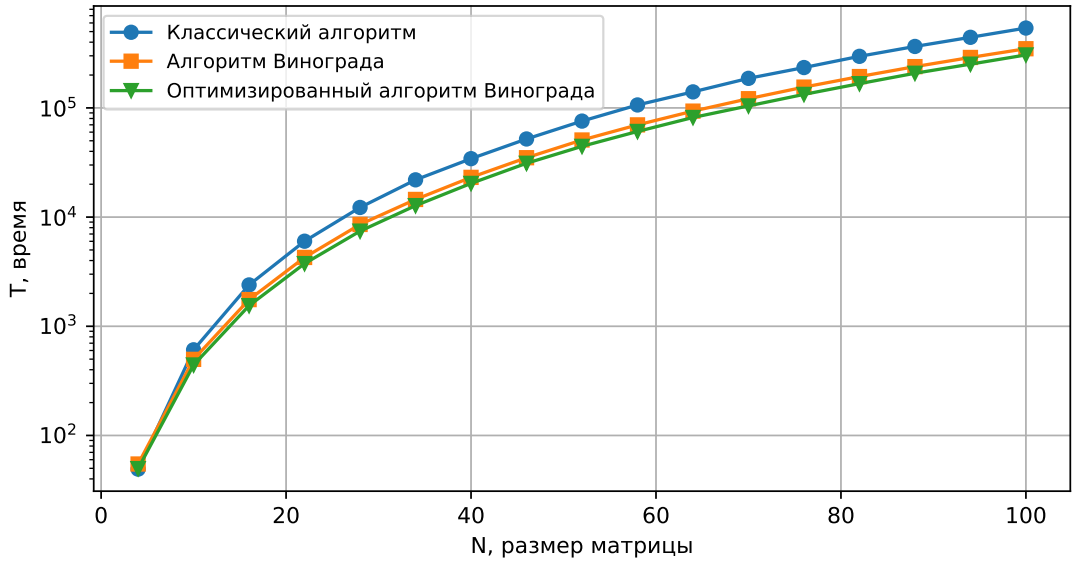

Рисунок 4.2 – Сравнение времени выполнения алгоритмов умножения матриц для четных размеров матриц

Таблица 4.2 содержит результаты замеров времени выполнения алгоритмов умножения матриц для нечетных размеров квадратных матриц в диапазоне от 5 до 101 с шагом 6, также с использованием различных входных данных.

Таблица 4.2 – Результаты замеров времени (нечетные размеры матриц)

Размер матрицы	**Время, мкс**		
	Классический	**Винограда**	**(опт.) Винограда**
5.0	279.84	219.44	185.76
11.0	1119.44	627.26	590.34
17.0	2817.32	1968.2	1910.74
23.0	6912.5	4578.38	4431.9
29.0	13717.74	8795.58	8518.78
35.0	23980.34	15068.12	14312.94
41.0	37826.54	23421.98	22695.06
47.0	56662.14	35014.34	33504.52
53.0	80815.22	49398.68	48168.9
59.0	112571.5	68110.72	65737.5
65.0	151266.12	90702.52	87376.1
71.0	191767.86	115627.1	110964.52
77.0	247003.34	148101.94	143579.58
83.0	308808.26	181623.9	178957.8
89.0	382926.68	225938.14	217564.8
95.0	462403.0	270946.18	265899.66
101.0	555144.74	326918.38	318142.04

На основе данных из таблицы 4.2 были построены графики (см. рисунок 4.3), из которых можно сделать вывод, что оптимизированный алгоритм Винограда также является наилучшим выбором для умножения матриц с нечетными размерами.

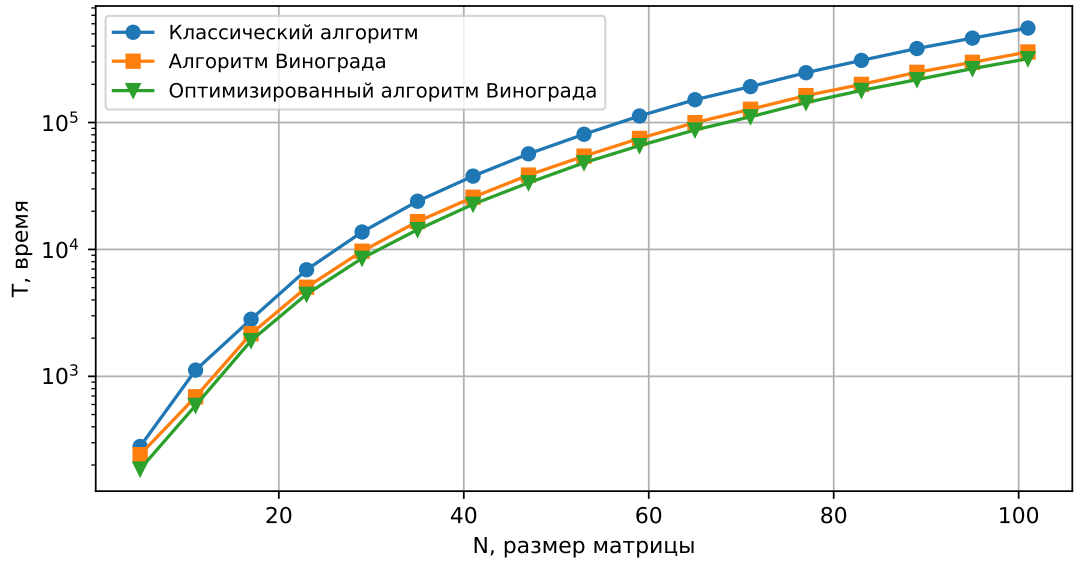

Рисунок 4.3 – Сравнение времени выполнения алгоритмов умножения матриц для нечетных размеров матриц

4.4 Выводы

Результаты экспериментов позволяют сделать вывод о том, что при больших размерах матриц (более 10), алгоритм Винограда работает значительно быстрее, чем классический алгоритм умножения матриц. Оптимизированный алгоритм Винограда в данном контексте демонстрирует лучшие результаты и опережает стандартный алгоритм.

Также было замечено, что алгоритм Винограда на четных размерах матриц работает быстрее по сравнению с нечетными размерами, что объясняется дополнительными вычислениями для крайних строк и столбцов. Следовательно, для матриц с четными размерами рекомендуется использовать алгоритм Винограда.

Заключение

Результаты данного исследования позволяют сделать несколько важных выводов. В частности, было установлено, алгоритм Винограда умножения матриц работает значительно быстрее классического алгоритма. Среди всех реализаций алгоритмов наилучшие показатели по времени демонстрирует оптимизированный алгоритм Винограда. Также стоит отметить, что алгоритмы Винограда работают быстрее на четных размерах матриц, так как выполняют меньше операций.

Основной целью данной лабораторной работы было изучение и описание особенностей алгоритмов умножения матриц, а именно классического алгоритма, алгоритма Винограда и оптимизированного Винограда. Ниже представлены выполненные задачи.

1) Проведено подробное описание алгоритмов для умножения матриц.
2) Разработано программное обеспечение, реализующее следующие алгоритмы:
   - Классический алгоритм умножения матриц.
   - Алгоритм Винограда умножения матриц.
   - Оптимизированный алгоритм Винограда умножения матриц.
3) Выбраны инструменты для измерения процессорного времени выполнения реализаций алгоритмов.
4) Проведен анализ временных затрат программы, чтобы выявить влияющие на них характеристики и факторы.

Цели и задачи исследования были успешно достигнуты, и полученные результаты позволяют лучше понять и оценить эффективность различных алгоритмов умножения матриц.

Список использованных источников

1 Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. — М.: Гуманит. изд. центр ВЛАДОС, 2003.

2 Головашкин Д. Л. Векторные алгоритмы вычислительной линейной алгебры: учеб. пособие. — Самара: Изд-во Самарского университета, 2019. — С. 28–35.

3 Документация к Swift [Электронный ресурс]. — Режим доступа: `https://developer.apple.com/documentation/swift` (дата обращения: 12.09.2023).

4 Измерение процессорного времени [Электронный ресурс]. — Режим доступа: `https://man7.org/linux/man-pages/man3/clock_gettime.3.html` (дата обращения: 14.09.2023).

5 Процессоры M1 [Электронный ресурс]. — Режим доступа: `https://developer.apple.com/documentation/apple-silicon` (дата обращения: 15.09.2023).

6 Операционная система macOS. [Электронный ресурс]. — Режим доступа: `https://developer.apple.com/macos/` (дата обращения: 15.09.2023).