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”

на тему:

«Алгоритмы умножения матриц»

Студент **Голикова С. М.**

Группа **ИУ7-55Б**

Оценка (баллы)

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

Москва, 2023г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

1 Аналитический раздел **4**

1.1 Стандартный алгоритм умножения матриц 4

1.2 Алгоритм умножения матриц Винограда 4

2 Конструкторский раздел **6**

2.1 Разработка алгоритмов 6

2.2 Оценка трудоемкости 10

3 Технологический раздел **13**

3.1 Требования к программному обеспечению 13

3.2 Средства реализации 13

3.3 Реализации алгоритмов 13

3.4 Тестирование 17

4 Исследовательский раздел **18**

4.1 Технические характеристики 18

4.2 Демонстрация работы программы 18

4.3 Сравнение времени выполнения реализаций алгоритмов . . . 19

ЗАКЛЮЧЕНИЕ **22**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **23**

ВВЕДЕНИЕ

Матрица – математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов матрицы задают размер матрицы [1].

Матричные вычисления – основа многих алгоритмов. Матрицы получили широкое распространение в компьютерной графике, моделировании физических экспериментов, цифровой обработке изображений, моделировании графов. Зачастую над матрицами приходится производить некоторые операции, одной из которых является умножение.

Умножение матриц – затратная операция, поэтому возникает необходимость в подборе алгоритмов, позволяющих оптимизировать это действие.

Целью работы является изучение способов оптимизации алгоритмов на примере алгоритмов умножения матриц.

В рамках выполнения работы необходимо решить следующие задачи:

— изучить алгоритмы умножения матриц;
— описать алгоритмы умножения матриц — стандартный и Винограда;
— разработать алгоритм Винограда с оптимизациями;
— реализовать три алгоритма умножения матриц — стандартный, Винограда и Винограда с оптимизациями;
— оценить трудоемкость алгоритмов;
— провести сравнительный анализ процессорного времени выполнения реализаций алгоритмов.

1 Аналитический раздел

1.1 Стандартный алгоритм умножения матриц

Пусть даны две прямоугольные матрицы $A[M \times N]$ и $B[N \times Q]$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1q} \\ b_{21} & b_{22} & \cdots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nq} \end{bmatrix}$$

Тогда матрица $C[M \times Q]$ – произведение матриц A и B:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1q} \\ c_{21} & c_{22} & \cdots & c_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mq} \end{bmatrix},$$

в которой каждый элемент вычисляется по формуле:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \ (i = 1, 2, \ldots m; j = 1, 2, \ldots q) \tag{1.1.1}$$

Стандартный алгоритм реализует формулу умножения матриц в прямом виде. Для каждого элемента подсчитывается значение независимо от других вычислений.

Преимущество этого алгоритма состоит в простоте реализации, а также в отсутствии дополнительных затрат на память. Платой за столь короткую реализацию алгоритма является высокое время выполнения. Для матриц размером 500x500 время выполнения уже выше трех секунд. Одним из способов уменьшения времени выполнения является использование более эффективного алгоритма умножения матриц – алгоритма Винограда [2].

1.2 Алгоритм умножения матриц Винограда

Заметим, что каждый элемент в результате умножения двух матриц представляет собой скалярное произведение соответствующих строки и столб-

ца исходных матриц. Кроме того, такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно следующей записи:

$$V \cdot W = (v_1+w_2)(v_2+w_1)+(v_3+w_4)(v_4+w_3)-v_1v_2-v_3v_4-w_1w_2-w_3w_4 \quad (1.2.1)$$

Кажется, что второе выражение задает больше работы, чем первое: вместо четырех умножений мы насчитываем их шесть, а вместо 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 столбца 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 [3].

В случае нечетного размера n начальной матрицы следует произвести еще одну операцию — добавление произведения последних элементов соответствующих строк и столбцов.

Вывод

В данном разделе были рассмотрены идеи, лежащие в основе алгоритмов умножения матриц: стандартного и алгоритма Винограда.

2 Конструкторский раздел

2.1 Разработка алгоритмов

На рисунках 1, 2 и 3 представлены схемы стандартного алгоритма, алгоритма Винограда и алгоритма Винограда с оптимизациями соответственно.

На схеме на рисунке 2 видно, что для алгоритма Винограда худшим случаем являются матрицы с нечётным размером, а лучшим – с чётным, так как в этой ситуации отпадает необходимость в последнем цикле.

Будем использовать следующие оптимизации алгоритма Винограда:

1) замену операции $x = x + k$ на $x+ = k$;
2) замену умножения на 2 на побитовый сдвиг;
3) предвычисление слагаемых для алгоритма.

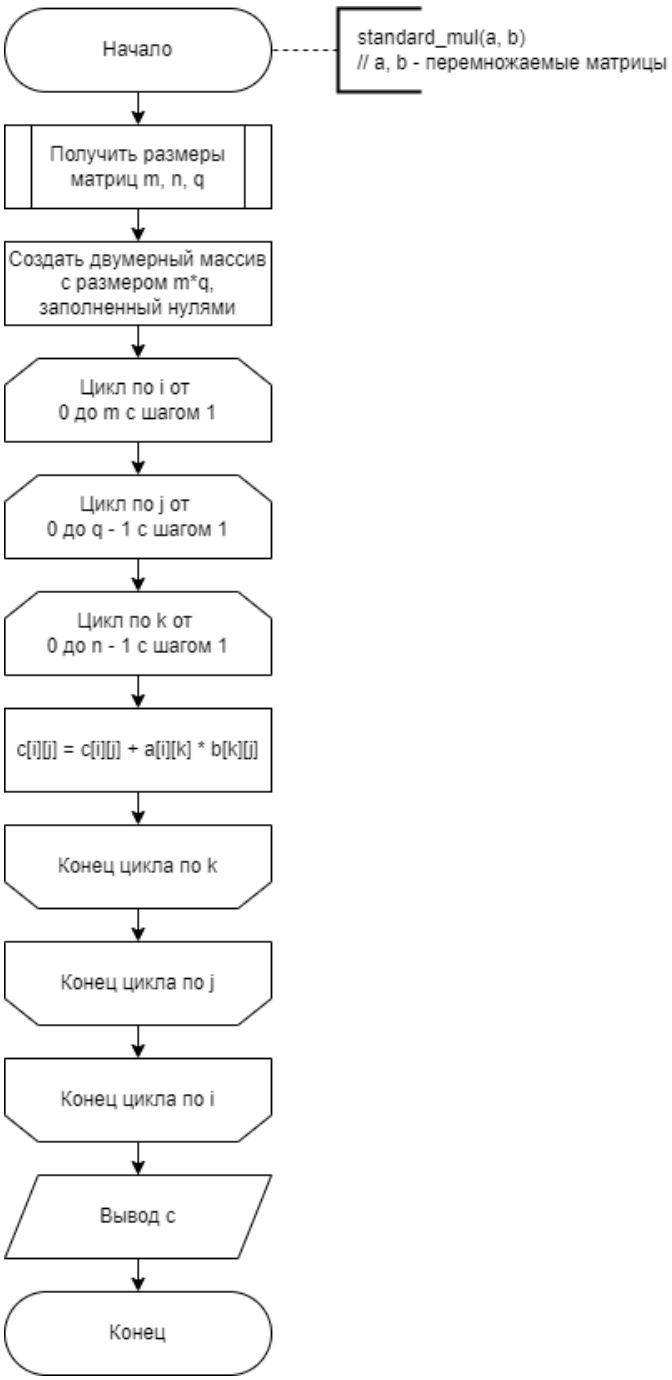

Рисунок 1 – Схема стандартного алгоритма умножения матриц

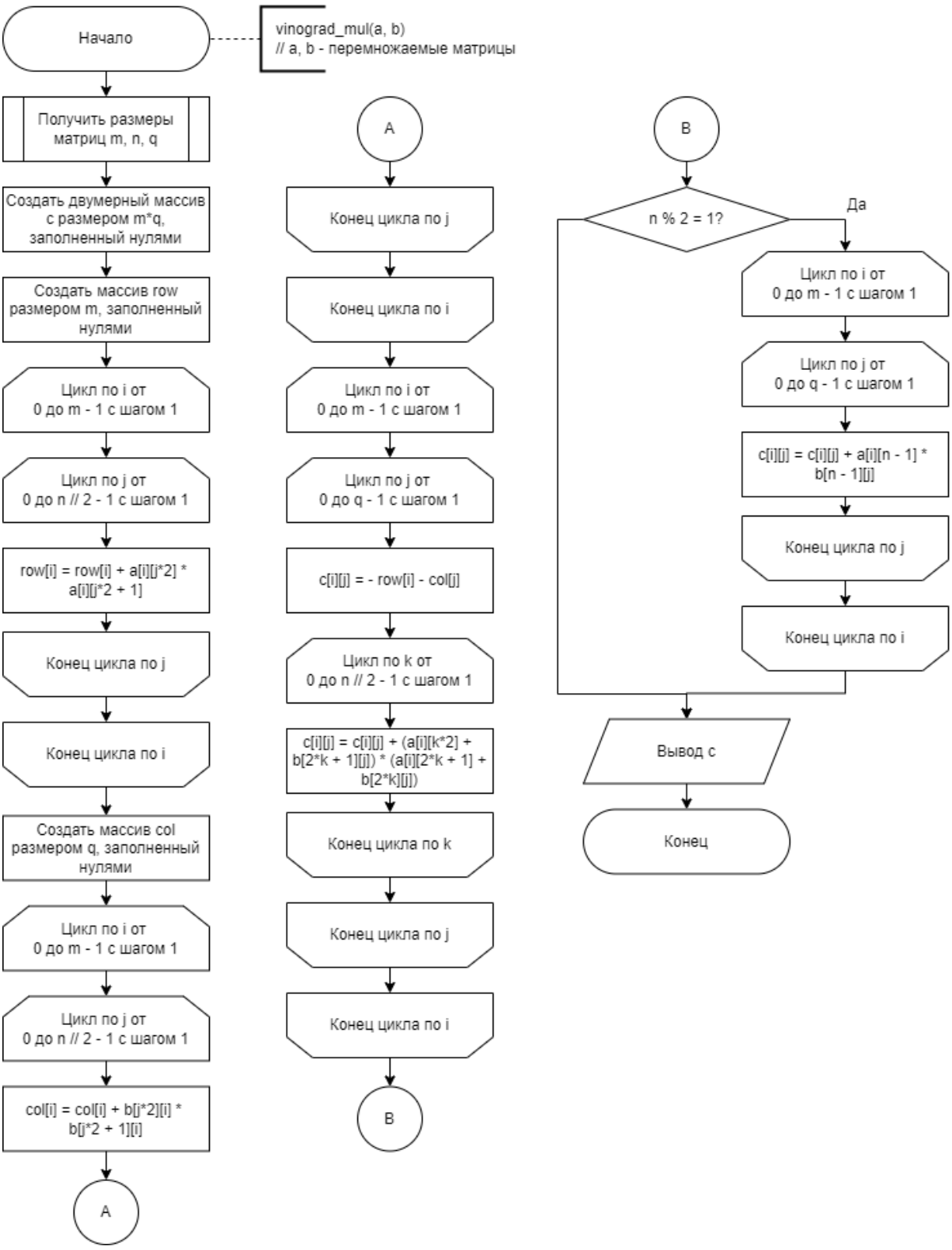

Рисунок 2 – Схема алгоритма Винограда умножения матриц

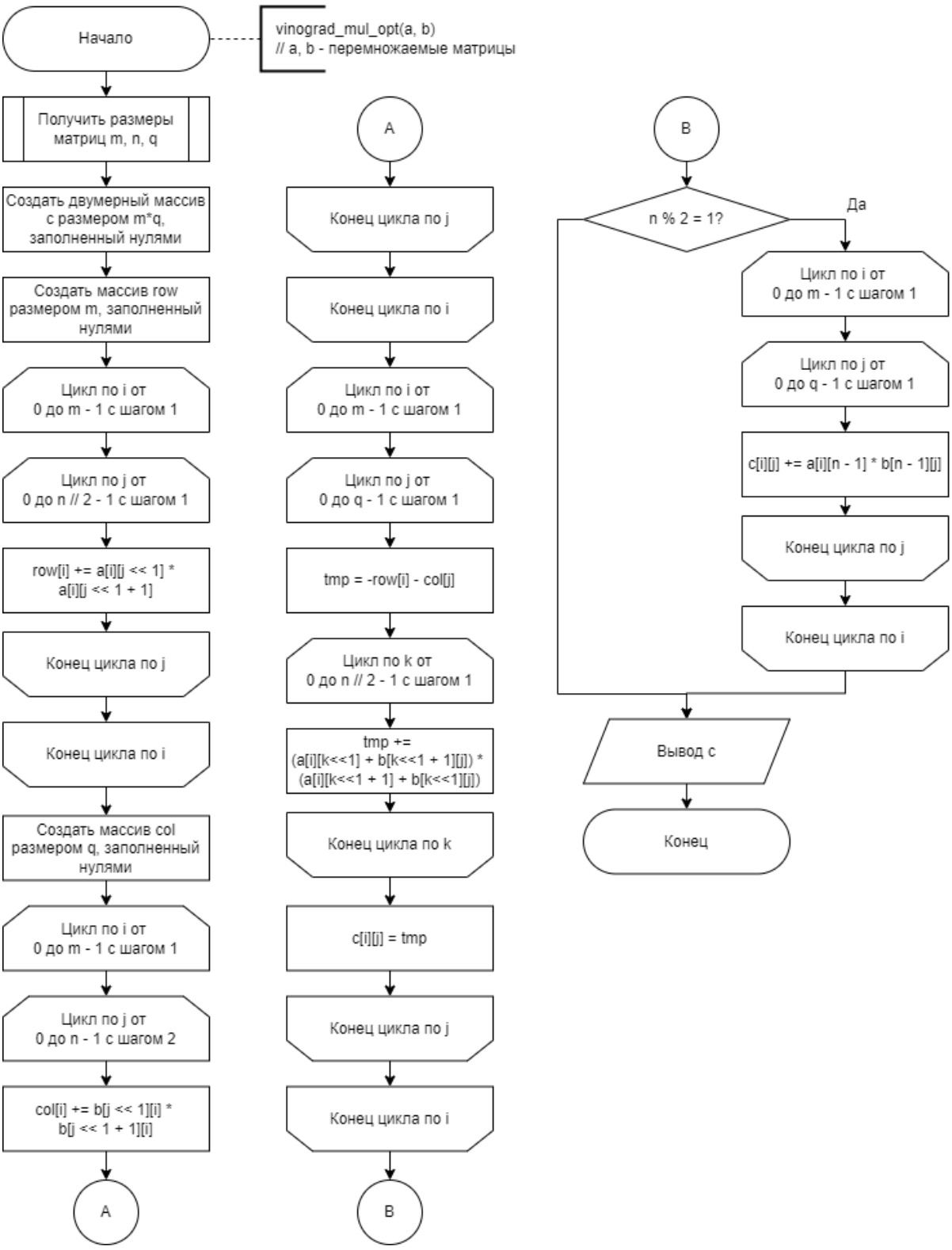

Рисунок 3 – Схема оптимизированного алгоритма Винограда

2.2 Оценка трудоемкости

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1) Трудоемкость следующих базовых операций единична: +, -, =, +=, -=, ==, !=, <, >, <=, >=, [], ++, –, «, ».

   Операции *, %, / имеют трудоемкость 2.

2) Трудоемкость цикла for(k = 0; k < N; k++) {тело цикла} рассчитывается как:

$$f_{for} = f_{\text{инициал.}} + f_{\text{сравн.}} + N(f_{\text{тела}} + f_{\text{инкр.}} + f_{\text{сравн.}}) \tag{2.2.1}$$

3) Трудоемкость условного оператора if (условие) then A else B рассчитывается как:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B), & \text{л.с.} \\ max(f_A, f_B), & \text{х.с.} \end{cases} \tag{2.2.2}$$

Произведем теоретическую оценку трудоемкости алгоритмов умножения матриц.

1) Трудоемкость стандартного алгоритма рассчитывается по формуле

$$f = 2 + m(4 + q(4 + 14n)) = 14mnq + 4mq + 4m + 2 \tag{2.2.3}$$

2) Трудоемкость базового алгоритма Винограда рассчитывается по формуле (2.2.8), представляющей собой сумму трудоемкостей четырех циклов (формулы 2.2.4 – 2.2.7).

$$f1 = 2 + m(2 + 4 + \frac{n}{2} \cdot 19) = \frac{19}{2}mn + 6m + 2 \tag{2.2.4}$$

$$f2 = 2 + q(6 + \frac{n}{2} \cdot 19) = \frac{19}{2}qn + 6q + 2 \tag{2.2.5}$$

$$f3 = 2 + m(4 + q(13 + \frac{n}{2} \cdot 32)) = 16mnq + 13mq + 4m + 2 \quad (2.2.6)$$

$$f4 = 3 + \begin{cases} 0, & \text{л.с.} \\ 2 + m(4 + 16q) = 16mq + 4m + 2, & \text{х.с.} \end{cases} \quad (2.2.7)$$

$$f = 16mnq + 13mq + \frac{19}{2}mn + \frac{19}{2}qn + 6m + 6q + 6 + \\ + \begin{cases} 0, & \text{л.с.} \\ 16mq + 4m + 2, & \text{х.с.} \end{cases} \quad (2.2.8)$$

3) Трудоемкость оптимизированного алгоритма Винограда рассчитывается по формуле 2.2.13, представляющей собой сумму трудоемкостей четырех циклов (формулы 2.2.9 – 2.2.12).

$$f1 = 2 + m(6 + \frac{n}{2} \cdot 15) = \frac{15}{2}mn + 6m + 2 \quad (2.2.9)$$

$$f2 = 2 + q(6 + \frac{n}{2} \cdot 15) = \frac{15}{2}qn + 6q + 2 \quad (2.2.10)$$

$$f3 = 2 + m(4 + q(11 + \frac{n}{2} \cdot 18 + 3)) = 9mnq + 14mq + 7m + 2 \quad (2.2.11)$$

$$f4 = 3 + \begin{cases} 0, & \text{л.с.} \\ 2 + m(4 + 13q) = 13mq + 4m + 2, & \text{х.с.} \end{cases} \quad (2.2.12)$$

$$f = 9mnq + 14mq + \frac{15}{2}mn + \frac{15}{2}qn + 6m + 6q + 6 + \\ + \begin{cases} 0, & \text{л.с.} \\ 13mq + 4m + 2, & \text{х.с.} \end{cases} \quad (2.2.13)$$

Лучший случай алгоритма Винограда и оптимизированного алгоритма Винограда — четная размерность n матрицы, т.к. в этом случае не производится заход в последний цикл.

Вывод

Были разработаны схемы алгоритмов, позволяющих с помощью различных подходов находить произведение матриц, а также была дана оценка трудоемкости для рассмотренных алгоритмов.

3 Технологический раздел

3.1 Требования к программному обеспечению

На вход программе подаются две матрицы, а на выходе должно быть получено искомое произведение матриц, вычисленное с помощью каждого реализованного алгоритма: стандартного, Винограда и Винограда с оптимизациями.

3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран язык Python [4]. Данный язык предоставляет возможности работы с массивами и матрицами.

Для замера процессорного времени использовалась функция process_time библиотеки time [5].

3.3 Реализации алгоритмов

В листингах 1, 2, 3 представлены реализации алгоритмов умножения матриц: стандартного, Винограда, Винограда с оптимизациями.

Листинг 1 – Стандартный алгоритм умножения матриц

```
def standard_matrix_mult(matr1, matr2):
    m, n, q = get_correct_matrices_sizes(matr1, matr2)
    result_matr = [[0] * q for _ in range(m)]

    for i in range(m):
        for j in range(q):
            for k in range(n):
                result_matr[i][j] = result_matr[i][j] + \
                                    matr1[i][k] * matr2[k][j]

    return result_matr
```

Листинг 2 – Алгоритм Винограда умножения матриц

```
def vinograd_matrix_mult(matr1, matr2):
    m, n, q = get_correct_matrices_sizes(matr1, matr2)
    result_matr = [[0] * q for _ in range(m)]

    row = [0] * m
    for i in range(m):
        for j in range(n // 2):
            row[i] = row[i] + matr1[i][j*2] * matr1[i][j*2+1]

    col = [0] * q
    for i in range(q):
        for j in range(n // 2):
            col[i] = col[i] + matr2[j*2][i] * matr2[j*2+1][i]

    for i in range(m):
        for j in range(q):
            result_matr[i][j] = -row[i] - col[j]
            for k in range(n // 2):
                result_matr[i][j] = result_matr[i][j] + \
                            (matr1[i][k*2] + matr2[2*k+1][j]) * \
                            (matr1[i][2*k+1] + matr2[2*k][j])

    if n % 2 == 1:
        for i in range(m):
            for j in range(q):
                result_matr[i][j] = result_matr[i][j] + \
                                    matr1[i][n-1] * matr2[n-1][j]

    return result_matr
```

Листинг 3 – Оптимизированный алгоритм Винограда умножения матриц

```
def vinograd_matrix_mult_optimized(matr1, matr2):
    m, n, q = get_correct_matrices_sizes(matr1, matr2)
    result_matr = [[0] * q for _ in range(m)]

    row = [0] * m
    for i in range(m):
        for j in range(n // 2):
            row[i] += matr1[i][j << 1] * matr1[i][j << 1 + 1]

    col = [0] * q
    for i in range(q):
        for j in range(n // 2):
            col[i] += matr2[j << 1][i] * matr2[j << 1 + 1][i]

    for i in range(m):
        for j in range(q):
            tmp = -row[i] - col[j]
            for k in range(n // 2):
                tmp += (matr1[i][k << 1] + matr2[k << 1 + 1][j])
                    *\
                        (matr1[i][k << 1 + 1] + matr2[k << 1][j])
            result_matr[i][j] = tmp

    if n % 2 == 1:
        for i in range(m):
            for j in range(q):
                result_matr[i][j] += matr1[i][n-1] * matr2[n-1][j
                    ]

    return result_matr
```

3.4 Тестирование

В таблице 1 приведены функциональные тесты для функций, реализующих алгоритмы умножения матриц.

Все тесты пройдены успешно каждой реализацией алгоритма.

Таблица 1 – Тестирование функций

Матрица A	Матрица B	Ожидаемый результат C
$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 \\ 1 & -1 \\ 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 0 & -1 & 1 & 2 \\ 0 & 1 & 1 & 3 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 2 & 5 \\ 0 & -2 & 0 & -1 \\ 0 & 0 & 4 & 10 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 12 & 15 & 18 \\ 12 & 15 & 18 \\ 12 & 15 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 \end{pmatrix}$	Не могут быть перемножены

Вывод

Были представлены листинги реализованных алгоритмов и тесты, успешно пройденные программой.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени:

— операционная система Windows 10;

— память 8 ГБ;

— процессор Intel® Core™ i5-6260U @ 1.80 ГГц.

Замеры времени выполнения реализаций алгоритмов проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно разработанным приложением.

4.2 Демонстрация работы программы

На рисунке 4 представлен пример работы программы. Пользователь вводит размеры и значения каждой из двух матриц. Программа выводит на экран результат умножения, вычисленный с помощью каждого из рассматриваемых алгоритмов.

```
Введите размеры первой матрицы
m1: 2
n1: 2
Введите первую матрицу построчно, через пробелы:
1 1
1 1

Введите размеры второй матрицы
m2: 2
n2: 2
Введите вторую матрицу построчно, через пробелы
1 2
3 4



Стандартный алгоритм
Результат:
[4, 6]
[4, 6]

Алгоритм Винограда
Результат:
[4, 6]
[4, 6]

Оптимизированный алгоритм Винограда
Результат:
[4, 6]
[4, 6]
```

Рисунок 4 – Пример работы программы

4.3 Сравнение времени выполнения реализаций алгоритмов

Все реализации алгоритмов сравнивались на случайно сгенерированных квадратных матрицах размерностями n·n, где n изменялось от 100 до 1000 с шагом 100 (лучший случай) и от 101 до 1001 с шагом 100 (худший случай).

На рисунке 5 приведены результаты сравнения времени работы всех реализаций на лучшем случае (четная размерность), а на рисунке 6 — на худшем (нечетная).

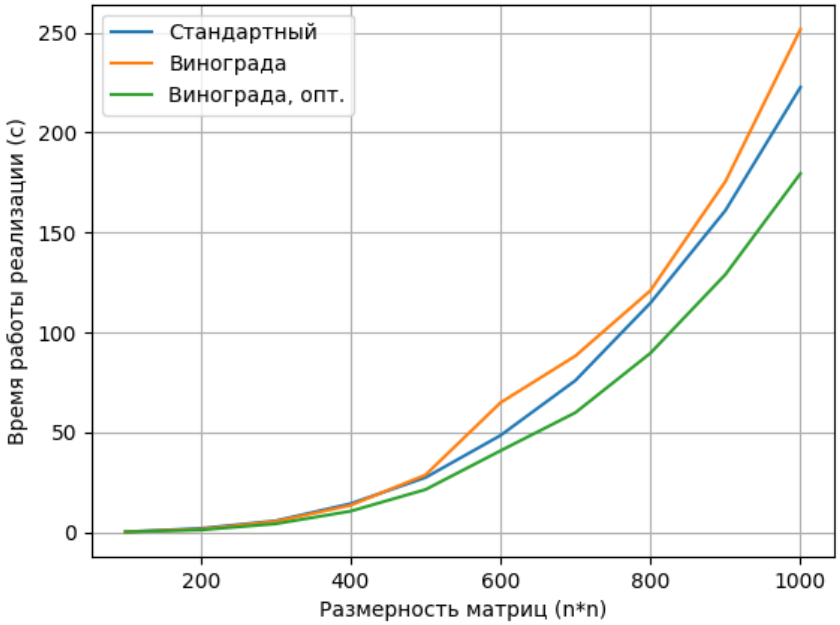

Рисунок 5 – Сравнение времени работы реализаций алгоритмов (л.с.)

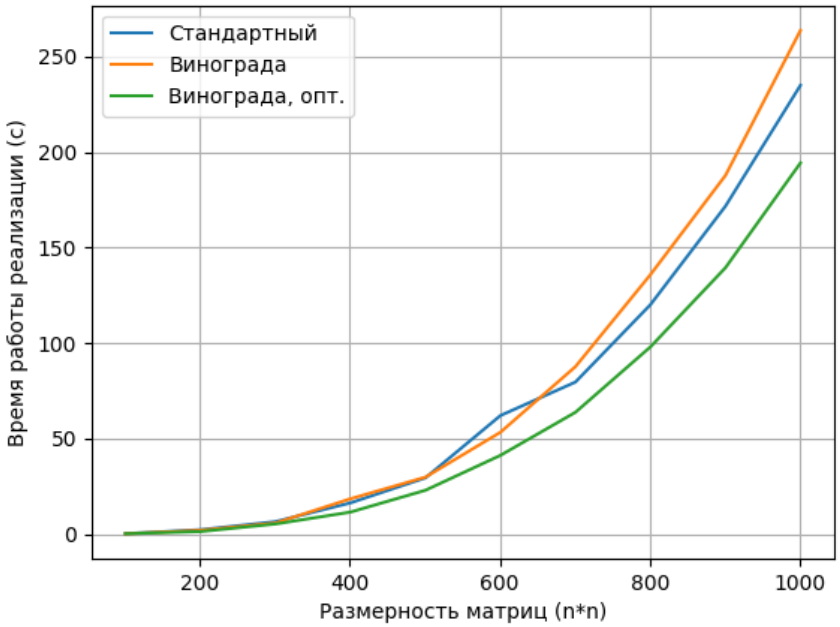

Рисунок 6 – Сравнение времени работы реализаций алгоритмов (х.с.)

Вывод

Были подтверждены теоретические расчеты: все алгоритмы кубически зависят от размерностей матриц. При этом реализация алгоритма Винограда работает дольше реализации стандартного алгоритма, а реализация алгоритма Винограда с оптимизациями – быстрее.

Таким образом, несмотря на сложность алгоритма Винограда по сравнению со стандартным, меньшая доля умножений в нем при применении оптимизаций позволяет получить меньшую трудоемкость.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были решены следующие задачи:

— изучены алгоритмы умножения матриц;
— описаны алгоритмы умножения матриц — стандартный и Винограда;
— разработан алгоритм Винограда с оптимизациями;
— реализованы три алгоритма умножения матриц — стандартный, Винограда и Винограда с оптимизациями;
— произведена оценка трудоемкости алгоритмов;
— проведен сравнительный анализ процессорного времени выполнения реализаций алгоритмов.

Поставленная цель была достигнута: были изучены способы оптимизации алгоритмов на примере алгоритмов умножения матриц.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Матрица, ее история и применение [Электронный ресурс]. Режим доступа: https://urok.1sept.ru/articles/637896 (дата обращения: 03.10.2022).

2. Реализация алгоритма умножения матриц по Винограду на языке Haskell [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell (дата обращения: 03.10.2022).

3. Умножение матриц по Винограду [Электронный ресурс]. Режим доступа: http://algolib.narod.ru/Math/Matrix.html (дата обращения: 03.10.2022).

4. Изучаем Python, том 1, 5-е изд / Лутц, Марк. — Пер. с англ. – СПб.: ООО “Диалектика”, 2019. — 832 с.

5. time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 22.09.2022).