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 квадратных матриц

Студент Прянишников А. Н.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Матричные вычисления - основа многих алгоритмов. Матрицы постоянно используются в компьютерной графике, моделировании физических экспериментов, реализации цифровых фильтров. В алгоритмах по построению рекомендательных систем также используется умножение матриц.

Актуальность работы заключается в том, что умножение матриц - затратная операция, поэтому требуется подбирать алгоритмы, которые позволяют оптимизировать эти действия.

Целью данной работы является разработка программы, которая реализует три алгоритма умножения матриц: стандартный алгоритм, алгоритм Винограда и модифицированный алгоритм Винограда.

Для достижения поставленной цели необходимо выполнить следующее:

- рассмотреть существующие алгоритмы умножения матриц;
- привести схемы реализации рассматриваемых алгоритмов;
- определить средства программной реализации;
- реализовать рассматриваемые алгоритмы;
- протестировать разработанное ПО;
- провести модульное тестирование всех реализаций алгоритмов;
- оценить реализацию алгоритмов по времени и памяти.

1 Аналитическая часть

В этом разделе будут произведена формализация задачи, а также рассмотрены существующие алгоритмы умножения матриц и модель для оценки трудоёмкости алгоритмов.

1.1 Формализация задачи

Матрица - математический объект, записываемый в виде прямоугольной таблицы, который представляет собой совокупность строк и столбцов, на пересечениях которых находятся элементы. Количество строк и столбцов задаёт размер матрицы.

Над матрицами определены несколько основных операций, в том числе умножение. Матрицы A и B могут быть перемножены, если число столбцов матрицы A равняется числу строк B.

Пусть матрица A имеет размеры $L \times M$, а матрица B – $M \times N$. Тогда результирующая матрица C имеет размеры $L \times N$, а каждый элемент матрицы можно подсчитать по формуле 1.1:

$$C_{i,j} = \sum_{r=1}^{M} a_{ir} * b_{rj} (i \in [1, L]; j \in [1, N]) \tag{1.1}$$

1.2 Классический алгоритм умножения матриц

Классический 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. Но недостаток заключается в том, что в алгоритме присутствует три полных цикла, и затраты по времени составляют $O(n^3)$. Реализация становится слишком затратной, поэтому были созданы другие алгоритмы умножения матриц.

1.3 Алгоритм Винограда

Подход алгоритма Винограда является иллюстрацией общей методологии, начатой в 1979 годах на основе билинейных и трилинейных форм, благодаря которым было получено большинство усовершенствований для умножения матриц.

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

Их скалярное произведение равно (1.2)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.2}$$

Равенство (1.2) можно переписать в виде (1.3)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.3}$$

Действий вычисления одного значения стало больше, но 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. В сравнении с классическим алгоритмом, где для вычисления скаляров требовалось четыре умножения и четыре сложения, получается выигрыш как в количестве действий, так и в быстродействии программы, так как сложение выполняется быстрее, чем умножение.

К недостатку алгоритма можно отнести дополнительные затраты на память, так как для каждой строки первой матрицы и для каждого столбца второй матрицы хранится значение в массиве.

1.4 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.

1.5 Модель для расчёта трудоёмкости алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. Для базовых операции: +, -, *, /, =, ==, <=, >=, !=, +=, [], ++ – трудоёмкость равна 1.
2. Трудоёмкость условия *if УСЛОВИЕ then A else B* будет подсчитана по формуле 1.4:

$$F = F_{\text{условия}} + \begin{cases} F_A & \text{, если условие выполняется} \\ F_B & \text{, иначе} \end{cases} \tag{1.4}$$

.

3. Трудоймкость цикла **for** будет подсчитана по формуле 1.5:

$$F_{for} = F_{\text{инициализации}} + F_{\text{сравнения}} + N(F_{\text{тела}} + F_{\text{инициализации}} + F_{\text{сравнения}}) \tag{1.5}$$

4. Трудоёмкость вызова функции равна 0.

1.6 Вывод

Были произведена формализация задачи и рассмотрены основные существующие алгоритмы умножения матрицы.

2 Конструкторская часть

В этом разделе будут приведены требования к ПО, схемы реализации алгоритмов, а также выбранные классы эквивалентности для тестирования ПО.

2.1 Требования к ПО

Ниже будет представлен список требований к разрабатываемому программному обеспечению.

Требования к входным данным:

- элементы матрицы должны быть целыми числами;
- хотя бы один из линейных размеров матрицы может быть нулевым.

Требования к выводу:

- программа должна выводить результирующую матрицу после умножения.

2.2 Схемы алгоритмов

На рисунке 2.1 будет приведена схема реализации алгоритма классического умножения матриц. На рисунках 2.2 будет приведена схема реализации алгоритма Винограда. На рисунке 2.3 будет приведена схема реализации оптимизированного алгоритма Винограда.

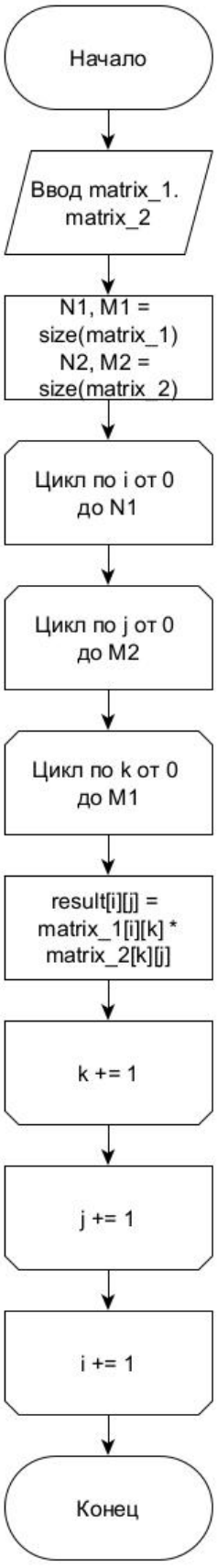

Рисунок. 2.1: Схема алгоритма классического умножения матриц

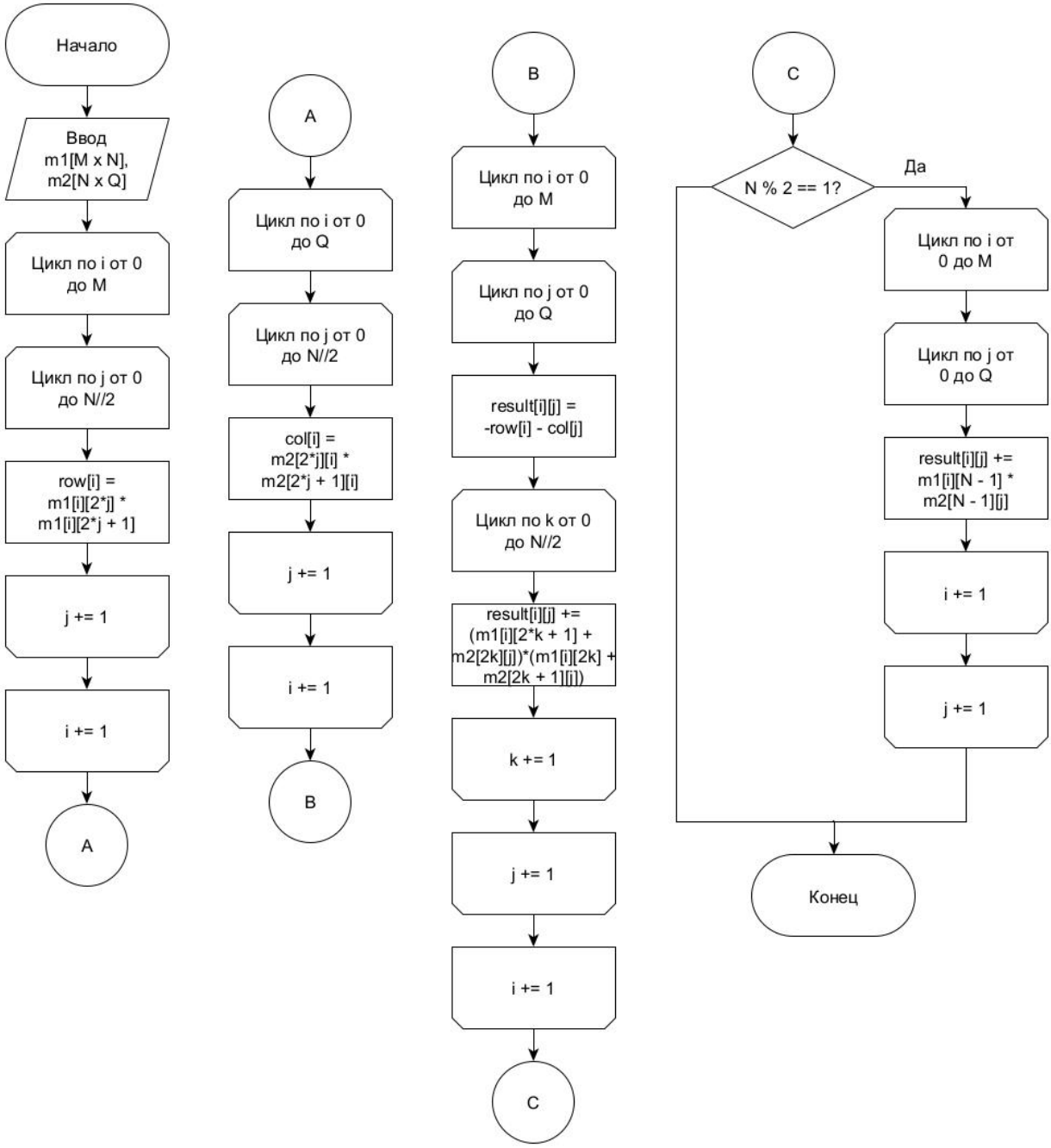

Рисунок. 2.2: Схема алгоритма Винограда

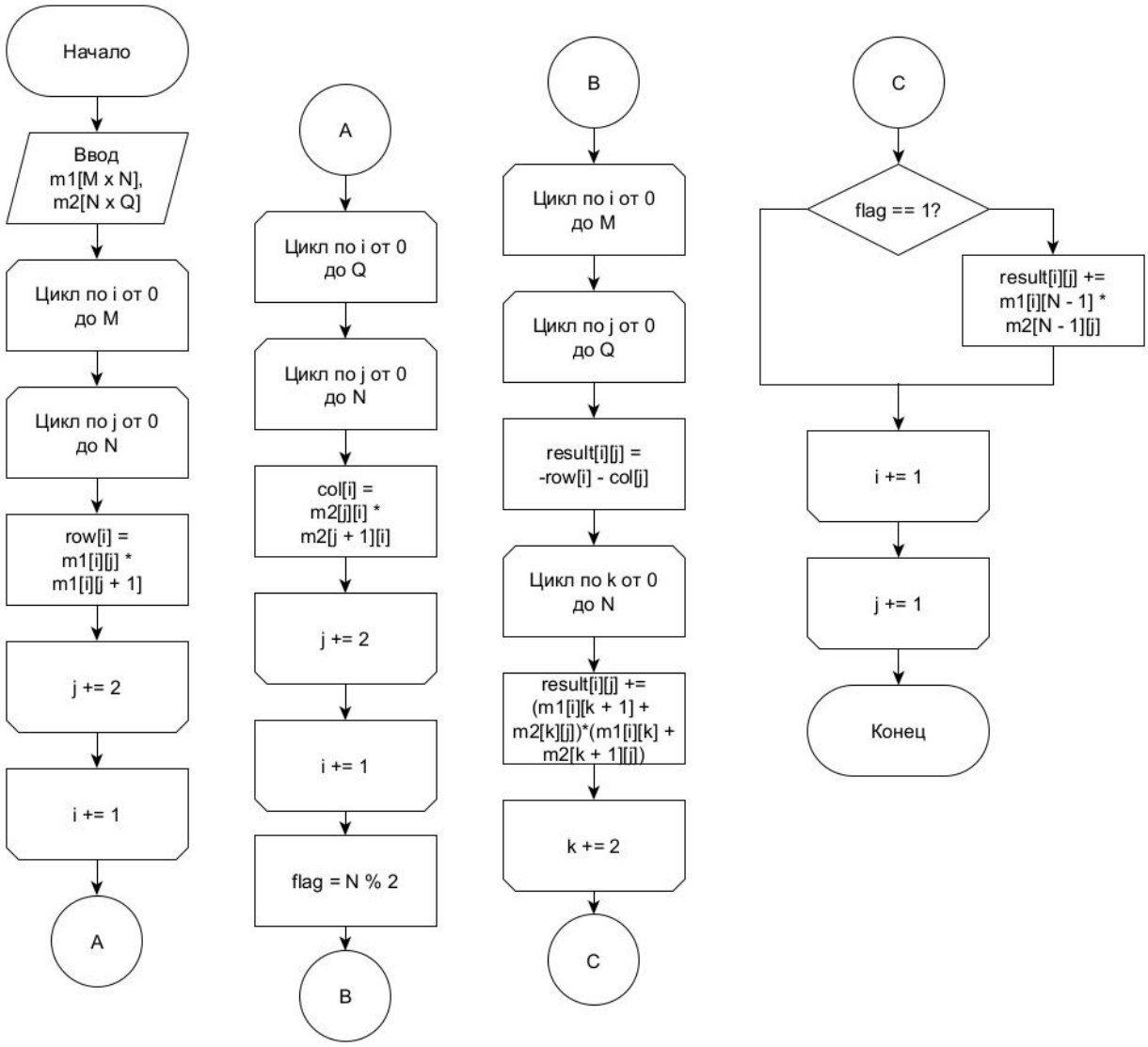

Рисунок. 2.3: Схема оптимизированного алгоритма Винограда

2.3 Типы данных для алгоритмов

Тестирование алгоритмов будет производиться на целых числах, которые могут быть и отрицательными, и равны нулю. Несмотря на это, сами реализации алгоритмов универсальны и предназначены для любых численных типов данных.

Размер матрицы может быть произвольным.

2.4 Способ тестирования

Тестирование программы будет производиться методом чёрного ящика. Такой подход выбран, так как от реализаций алгоритмов требуется в первую очередь правильность работы. Сама по себе реализация не требует тестировки, так как в точности повторяет теоретические принципы, сформированные в аналитическом разделе.

В качестве классов эквивалентности были выбраны следующие сущности:

- одна из матриц нулевого размера;
- матрицы не соответствуют друг другу по размеру;
- матрицы размером в 1×1;
- чётные размеры матрицы;
- нечётные размеры матрицы;
- одна из матриц - единичная;
- одна из матриц - нулевая;

Для корректного сравнения требуется для каждой реализации умножения матриц сравнить полученный на выходе результат с эталонным.

2.5 Вывод

Были приведены требования к ПО, схемы реализации алгоритмов. Были определен способ тестирования алгоритмов.

3 Технологический раздел

В этом разделе будут приведены листинги кода и результаты функционального тестирования. Также будет произведена оценка трудоёмкости алгоритмов.

3.1 Средства реализации программного обеспечения

В качестве языка программирования выбран Python 3.9, так как имеется опыт разработки проектов на этом языке. Для замера процессорного времени используется функция process_time_ns из библиотеки time. В её результат не включается время, когда процессор не выполняет задачу [?].

3.2 Листинг кода

На листингах 4.1, 4.2 и 4.3 приведены реализации различных алгоритмов умножения матриц.

Листинг 3.1: Реализация алгоритма классического умножения матриц

```
def classic(matrix_1, matrix_2):
    N1, M1 = len(matrix_1), len(matrix_1[0])
    N2, M2 = len(matrix_2), len(matrix_2[0])

    result_matrix = []
    check = check_sizes(N1, M1, N2, M2)

    if (check):
        result_matrix = [[0] * N1 for i in range(M2)]
        for i in range(N1):
            for j in range(M2):
                for k in range(M1):
                    result_matrix[i][j] += (matrix_1[i][k] * \
                                            matrix_2[k][j])
    return result_matrix
```

Листинг 3.2: Реализация алгоритма Винограда

```
def vinograd(matrix_1, matrix_2):
    N1, M1 = len(matrix_1), len(matrix_1[0])
    N2, M2 = len(matrix_2), len(matrix_2[0])

    result_matrix = []
    check = check_sizes(N1, M1, N2, M2)

    if (check):
        result_matrix = [[0] * N1 for i in range(M2)]

        M = N1
        N = M1
        Q = M2

        row = [0] * M
        for i in range(M):
            for j in range(N//2):
                row[i] += (matrix_1[i][2 * j] * \
                            matrix_1[i][2 * j + 1])

        col = [0] * Q
        for i in range(Q):
            for j in range(N // 2):
                col[i] += (matrix_2[2 * j][i] * \
                            matrix_2[2 * j + 1][i])

        for i in range(M):
            for j in range(Q):
                result_matrix[i][j] += (-row[i] -col[j])
                for k in range(N//2):
                    result_matrix[i][j] += \
                        (matrix_1[i][2*k + 1] + matrix_2[2*k][j])* \
                        (matrix_1[i][2*k] + matrix_2[2*k + 1][j])

        if (N % 2 == 1):
            for i in range(M):
                for j in range(Q):
                    result_matrix[i][j] += \
                        (matrix_1[i][N - 1] * matrix_2[N - 1][j])
        return result_matrix
```

Листинг 3.3: Реализация оптимизированного алгоритма Винограда

```
def optvinograd(matrix_1, matrix_2):
    N1, M1 = len(matrix_1), len(matrix_1[0])
    N2, M2 = len(matrix_2), len(matrix_2[0])

    result_matrix = []
    check = check_sizes(N1, M1, N2, M2)

    if (check):
        result_matrix = [[0] * N1 for i in range(M2)]

        M = N1
        N = M1
        Q = M2

        row = [0] * M
        for i in range(M):
            for j in range(0, N - 1, 2):
                row[i] += (matrix_1[i][j] * \
                           matrix_1[i][j + 1])

        col = [0] * Q
        for i in range(Q):
            for j in range(0, N - 1, 2):
                col[i] += (matrix_2[j][i] * \
                           matrix_2[j + 1][i])

        flag = N % 2
        for i in range(M):
            for j in range(Q):
                result_matrix[i][j] += (-row[i] -col[j])
                for k in range(0, N - 1, 2):
                    result_matrix[i][j] += \
                        (matrix_1[i][k + 1] + matrix_2[k][j]) *  \
                        (matrix_1[i][k] + matrix_2[k + 1][j])
                if (flag):
                    result_matrix[i][j] += \
                        (matrix_1[i][N - 1] * matrix_2[N - 1][j])
        return result_matrix
```

3.3 Функциональные тесты

В таблице 3.1 приведены результаты функциональных тестов. В первом столбце – первый операнд умножения матриц. Во втором столбце – второй операнд умножения матриц. В третьем столбце – ожидаемый результат.

Таблица 3.1: Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый результат
1 2 3 4 5 6 7 8 9	9 8 7 6 5 4 9 8 7	30 24 18 84 69 54 138 114 90
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	80 70 60 50 240 214 188 162 400 358 316 274 560 502 444 386
1 2 3 4 5 6	6 5 4 3 2 1	20 14 56 41
1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6	9 8 7 6 5 4	-
1 2 3 4 5 6	*	-

* – пустая матрица.

\- – данные введены некорректно

Все тесты алгоритмами были пройдены успешно.

3.4 Вычисление трудоёмкости дял алгоритмов

Для каждого из алгоритмов проведём вычисление трудоёмкости. Для всех случаев размеры первой матрицы – $n_1 \times m_1$, а второй – $n_2 \times m_2$.

3.4.1 Вычисление трудоёмкости классического алгоритма

Инициализация матрицы результата: $1 + 1 + n_1(1 + 2 + 1) + 1 = 4n_1 + 3$

Подсчет:

$1 + n_1(1 + (1 + m_2(1 + (1 + m_1(1 + (8) + 1) + 1) + 1) + 1) + 1) + 1 = n_1(m_2(10m_1 + 4) + 4) + 4) + 2 = 10n_1m_2m_1 + 4n_1m_2 + 4n_1 + 2$

3.4.2 Вычисление трудоёмкости алгоритма Винограда

Первый цикл: $\frac{15}{2}n_1m_1 + 5n_1 + 2$

Второй цикл: $\frac{15}{2}m_2n_2 + 5m_2 + 2$

Третий цикл: $13n_1m_2m_1 + 12n_1m_2 + 4n_1 + 2$

Условный переход: $\begin{bmatrix} 2 & , \text{невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & , \text{выполнение условия} \end{bmatrix}$

Итого: $13n_1m_2m_1 + \frac{15}{2}n_1m_1 + \frac{15}{2}m_2n_2 + 12n_1m_2 + 5n_1 + 5m_2 + 4n_1 + 6 + \begin{bmatrix} 2 & , \text{невыполнение условия} \\ 15n_1m_2 + 4n_1 + 2 & , \text{выполнение условия} \end{bmatrix}$

3.4.3 Вычисление трудоёмкости оптимизированного алгоритма Винограда

Первый цикл: $\frac{11}{2}n_1m_1 + 4n_1 + 2$

Второй цикл: $\frac{11}{2}m_2n_2 + 4m_2 + 2$

Третий цикл: $\frac{17}{2}n_1m_2m_1 + 9n_1m_2 + 4n_1 + 2$

Условный переход: $\begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

Итого: $\frac{17}{2}n_1m_2m_1 + \frac{11}{2}n_1m_1 + \frac{11}{2}m_2n_2 + 9n_1m_2 + 8n_1 + 4m_2 + 6 + \begin{bmatrix} 1 & \text{, невыполнение условия} \\ 10n_1m_2 + 4n_1 + 2 & \text{, выполнение условия} \end{bmatrix}$

3.5 Вывод

Были сформированы требования к ПО, приведены листинги коды. Были вычислены трудоёмкости алгоритмов и проведены функциональные тесты. Все алгоритмы справились с тестированием.