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»

Отчёт

по лабораторной работе №2

Название «Алгоритмы умножения матриц»

Дисциплина «Анализ алгоритмов»

Студент ИУ7и-54Б Шавиш Тарек.
(подпись, дата) (Фамилия И.О.)

Преподователь Волкова Л.Л.
(подпись, дата) (Фамилия И.О.)

Москва, 2024

Содержание

Введение

Матрица - это прямоугольный массив элементов, записанных в виде набора строк и столбцов, количество которых определяют размер матрицы. В данной лабораторной работе мы рассмотрим несколько алгоритмов умножения матриц. Если говорить о том, где используется умножение матриц, то основное применение данная операция находит в алгоритмах компьютерной графики при операциях над объектами в трёхмерном пространстве. Помимо этого матрицы активно используются в математике, физике, химии и прочих науках для различных вычислений, связанных с массивами данных.

Целью данной лабораторной работы является изучение алгоритмов умножения матриц, получения практических навыков при реализации данных алгоритмы, приобритение навыков расчёта трудоёмкости алгоритмов и получение навыков оптимизации реализации алгоритмов. Для того, чтобы достичь поставленной цели нам необходимо выполнить следующие задачи:

1) Провести анализ данных алгоритмы умножения матриц:
   а) Классический алгоритм умножения матриц;
   б) Алгоритм Винограда для умножения матриц;
2) оптимизировать алгоритм Винограда для умножения матриц;
3) описать используемые структуры данных;
4) привести схемы рассматриваемых алгоритмов;
5) вычислить трудоёмкость для указанных выше алгоритмов;
6) программно реализовать данные выше алгоритмы;
7) провести сравнительный анализ каждого алгоритма по затрачиваемому в процессе работы времени.

1 Аналитический раздел

В данном разделе будет рассмотрено описание работы классического алгоритма умножения матриц и алгоритма умножения матриц Винограда.

1.1 Основные сведения об алгоримтах

Умножение матриц - одна из основных операций над матрицами. В результате данной операции мы получаем новую матрицу. Данная операция становится возможна только в том случае, если число столбцов в первом сомножителе равно числу строк во втором. При выполнении этого условия говорится, что матрицы согласованы. Из данного условия следует, что произведение матриц некоммутативно.

1.2 Классический алгоритм

Пусть даны две прямоугольные матрицы A и B, размерности $n \times m$ и $m \times k$ соответственно:

$$A_{n\times m} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \tag{1.1}$$

$$B_{m\times k} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} \tag{1.2}$$

Тогда произведением матриц A, B будет является матрица C размерностью $n \times k$:

$$C_{n\times k} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{bmatrix} \tag{1.3}$$

где:

$$C_{ij} = \sum_{l=1}^{m} a_{il}b_{lj}(i = \overline{1,n}, j = \overline{1,k}) \tag{1.4}$$

Стандартный алгоритм умножения матриц использует эту формулу.

1.3 Алгоритм Винограда

При рассмотрении операции умножения матриц, можно сделать вывод, что каждый элемент в матрице C на самом деле является скалярным произведением векторов строки из матрицы A и столбца из матрицы B. При помощи предварительной обработки мы можем выполнить часть работы заранее, что позволит снизить количество операций.

Рассмотрим два вектора V и W:

$$V = (v_1, v_2, v_3, v_4) \tag{1.5}$$

$$W = (\omega_1, \omega_2, \omega_3, \omega_4) \tag{1.6}$$

Тогда скалярное произведение этих векторов может быть записано следующим образом.

$$V \times W = v_1\omega_1 + v_2\omega_2 + v_3\omega_3 + v_4\omega_4 \tag{1.7}$$

Это равенство может быть преобразовано следующим образом:

$$V \times W = (v_1 + \omega_2)(v_2 + \omega_1) + (v_3 + \omega_4)(v_4 + \omega_3) - v_1v_2 - v_3v_4 + \omega_1\omega_2 + \omega_3\omega_4 \tag{1.8}$$

Данное выражение позволяет сэкономить количество операций при вычислении благодаря тому, что правую часть можно вычислить заранее отдельно для каждой матрицы и использовать уже посчитанный результат при вычислении. При выполнении программы над предварительно обработанными элементами выполняются только первые два умножения и последующие пять сложений, а таже два сложения в скобках. Так как сложение быстрее умножения, алгоритм должен работать быстрее стандартного.

1.4 Модель вычислений

Для того, чтобы вычислить трудоёмкость алгоритма, введём следующую модель вычислений:

1) операторы, трудоёмкость которых равна 1: +, -, /, %, ==, !=, <, >, >=, <=, [], ++;
2) трудоёмкость оператора выбора if (условие) then A else B рассчитывается как $f_{\text{условия}} + f_A$ или f_B в зависимости то того, какое условие выполняется;
3) трудоёмкость цикла рассчитывается как $f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}})$;
4) трудоёмкость вызова функции равна 0.

1.5 Вывод

В данном разделе были рассмотрены основные теоретические сведения о классическом алгоритме умножения матриц и об алгоритме Винограда умножения матриц. В результате чего был сделаны выводы о том, что на вход алгоритмов подаются матрицы и их размерности, а на выходе возвращается матрица с результатом. Алгоритмы работают на матрицах с согласованными размерностями от 0 до физически возмого предела для используемой машины. В качестве критерием для сравнения будет использоваться время работы алгоритмов. Также был проведён анализ модели вычислений, которая будет использоваться для вычисления трудоёмкости алгоритмов.

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы, структуры данных, способы тестирования, описания памяти для следующих алгоритмов:

1) классический алгоритм умножения матриц;
2) алгоритм Винограда для умножения матриц;
3) оптимизированный алгоритм Винограда для умножения матриц.

2.1 Тестирование алгоритмов

Описание классов эквивалентности:

1) проверка на общем случае матриц;
2) проверка на векторах (частный случай матрицы);
3) проверка на единичной матрице;
4) проверка на нулевой матрице.

Описание тестов:

1) тест на двух матрицах размером n x k и k x m, где $n > 0$, $k > 0$, $m > 0$;
2) тест на двух матрицах размером n x k и k x m, где $n = 1$, $k > 0$, $m > 0$;
3) тест на двух матрицах размером n x k и k x m, где $n > 0$, $k > 0$, $m = 1$;
4) тест на двух матрицах размером n x k и k x m, где $n > 0$, $k > 0$, $m > 0$, и первая матрица - нулевая;
5) тест на двух матрицах размером n x k и k x m, где $n > 0$, $k > 0$, $m > 0$, и первая матрица - единичная.

2.2 Классический алгоритм умножения матриц

Классический алгоритм умножения матриц реализует описанную выше формулу умножения двух матриц.

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов и размерностей матрицы
2) matrix, массив массивов вещественного типа - используется для хранения двух входных матриц и матрицы, хранящей результат умножения

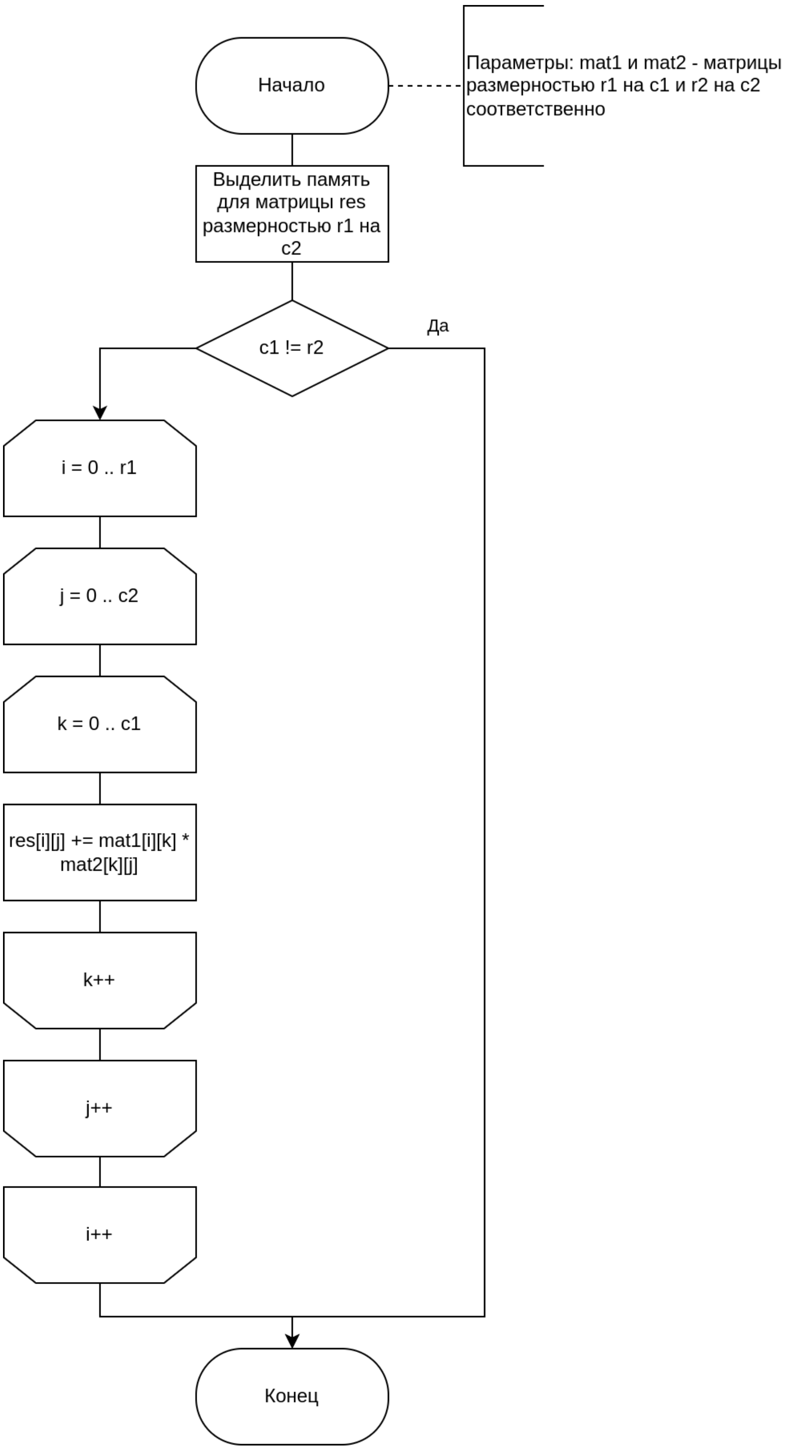

Рисунок 2.1 — Схема классического алгоритма умножения матриц

2.3 Алгоритм Винограда для умножения матриц

В данном алгоритме в конце необходимо сделать проверку на чётность для того, чтобы не потерять элементы при нечётных размерах матрицы.

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов и размерностей матрицы

2) matrix, массив массивов вещественного типа - используется для хранения двух входных матриц и матрицы, хранящей результат умножения

Схема алгоритма Винограда представлена на рисунке:

2.4 Оптимизированный алгоритм Винограда для умножения матриц

Для оптимизации алгоритма Винограда была добавлена операция $+=$, введены буферы, позволяющие не вычислять часть выражения повторно, и использованы битовые операции, имеющие меньшую трудоёмкость, чем обычные.

Используемые типы и структуры данных включают в себя:

1) integer, целое число - используется для хранения индексов и размерностей матрицы
2) matrix, массив массивов вещественного типа - используется для хранения двух входных матриц и матрицы, хранящей результат умножения

Схема оптимизированного алгоритма Винограда представлена на рисунке:

2.5 Трудоёмкость алгоритмов

Проведём оценку трудоёмкости алгоритмов на основе описанной выше схемы. В приведённых ниже вычислениях констранты N, K и N - целые константы, обозначающие размеры матриц A - $N \times K$, B - $K \times M$.

Результирующая трудоёмкость - $F_{\text{классического}} = (10 * N * K * M) + (4 * N * M) + (4 * N) + 2$

Результирующая трудоёмкость - $F_{\text{Винограда}} = 13 * M * N * K + 7.5 * K * N + 7.5 * K * M + 11 * M * N + 8 * N + 4 * M + 14 + \left[\begin{array}{c} 0 \\ 15MN + 4N + 2 \end{array} \right.$

Результирующая трудоёмкость - $F_{\text{Винограда оптимизированного}} = 8 * M * N * K + 5 * K * N + 5 * K * M + 12 * M * N + 8 * N + 4 * M + 17 + \left[\begin{array}{c} 0 \\ 10MN + 4N + 4 \end{array} \right.$

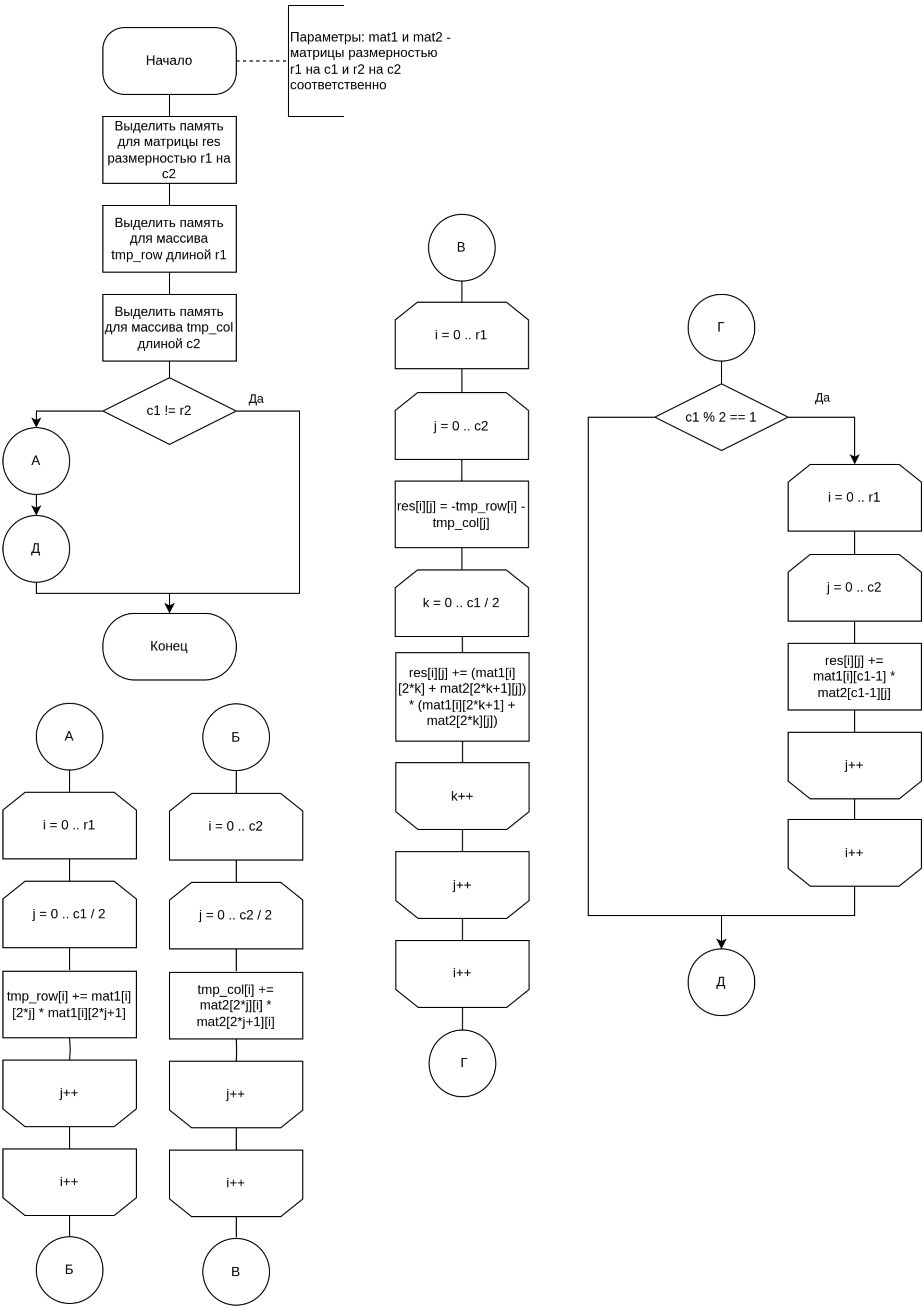

Рисунок 2.2 — Схема алгоритма Винограда

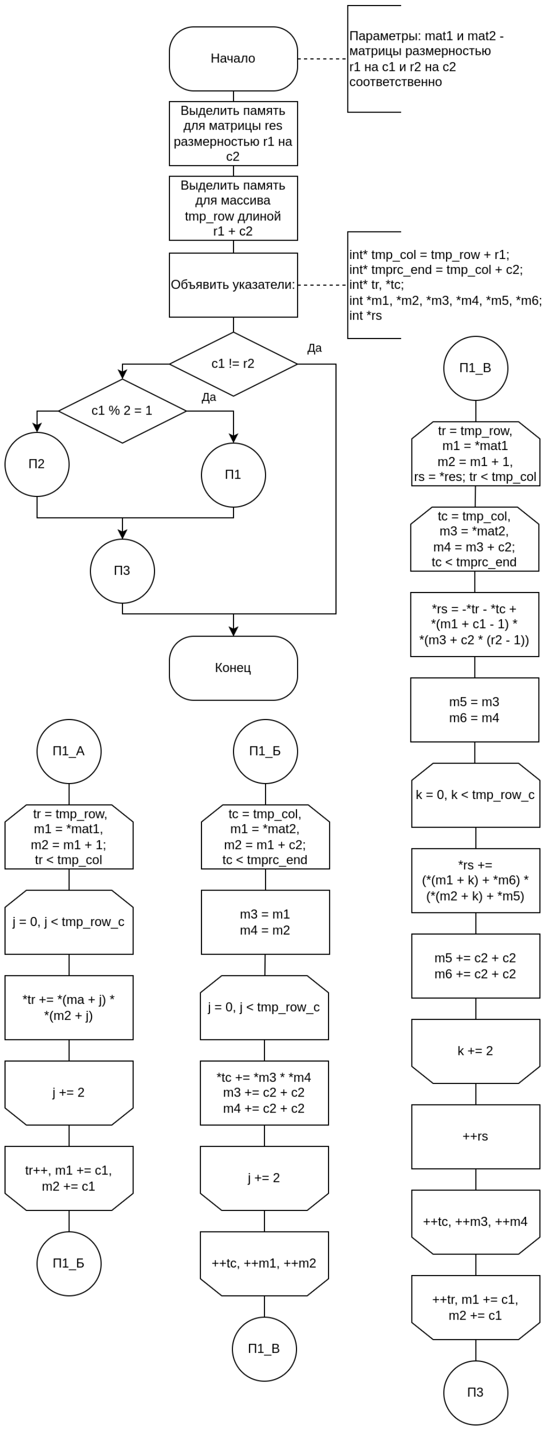

Рисунок 2.3 — Схема оптимизированного алгоритма Винограда

Таблица 2.1 — Оценка трудоёмкости для классического алгоритма

Трудоёмкость	Оценка трудоёмкости
$F_{\text{классического}}$	2 + N(2 + 2 + M(2 + 2 + K(2 + $F_{\text{тела}}$)))
$F_{\text{тела}}$	2 + 2 + 2 * 2

Таблица 2.2 — Оценка трудоёмкости для алгоритма Винограда

Трудоёмкость	Оценка трудоёмкости
$F_{\text{инициализации}}$	2 * 3
$F_{\text{заполнения col\_vec}}$	2 + M * (2 + 2 + K / 2 * (3 + 6 + 6))
$F_{\text{заполнения row\_vec}}$	2 + N * (2 + 2 + K / 2 * (3 + 6 + 6))
$F_{\text{вычисления результата}}$	2 + N * (2 + 2 + M * (2 + 7 + 2 + K / 2 * (3 + 23)))
$F_{\text{условия нечётности}}$	2
$F_{\text{учёта нечётных матриц}}$	2 + N *(2 + 2 + M * (2 + 8 + 5))

Таблица 2.3 — Оценка трудоёмкости для оптимизированного алгоритма Винограда

Трудоёмкость	Оценка трудоёмкости
$F_{\text{инициализации}}$	2 * 3 + 3
$F_{\text{заполнения col\_vec}}$	2 + M * (2 + 2 + K / 2 * (3 + 6 + 6))
$F_{\text{заполнения row\_vec}}$	2 + N * (2 + 2 + K / 2 * (3 + 6 + 6))
$F_{\text{вычисления результата}}$	2 + N * (2 + 2 + M * (2 + 5 + 3 + 2 + K / 2 * (2 + 14)))
$F_{\text{условия нечётности}}$	2
$F_{\text{учёта нечётных матриц}}$	2 + 2 + N * (2 + 2 + M * (2 + 6 + 2))

2.6 Функциональная схема ПО

На изображении ниже представлена функиональная схема разрабатываемого ПО. На вход подаются две матрицы и их размерности и при помощи алгоритмов, реализованных на языке Python мы получаем в результате работы новую матрицу, содерждащую в себе результат операции.

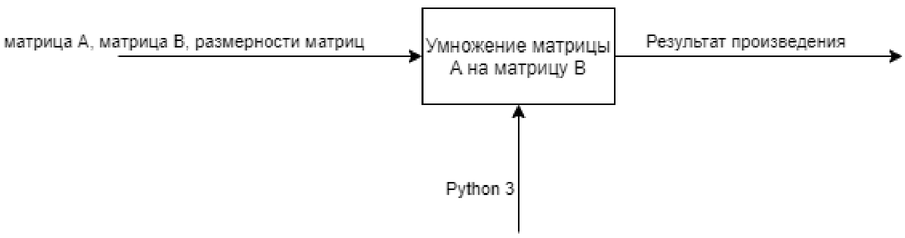

Рисунок 2.4 — IDEF0 диаграмма разрабатываемой программы

2.7 Вывод

В данном разделе были рассмотрены схемы алгоритмов для каждого из способов вычисления произведения матриц, и были определены тесты для каждого алгоритма, были описаны типы и структуры данных, использующихся в алгоритмах. Также была произведена оценка трудоёмкости для изучаемых алгоритмов и приведена функциональная схема разрабатываемого ПО.

3 Технологический раздел

В данном разделе будут рассмотрены подробности реализации описаных выше алгоритмов. Также будут обоснованы выбор языка программирования для реализации, выбор библиотек для проведения экспериментов и представлены важные фрагменты кода написанной в рамках работы программы.

3.1 Выбор языка программирования

В качестве языка программирования для реализации данной лабораторной работы использовался язык программирования python (3.9.7) [1] в целях упрощения работы со структурами данных и визцализацией данных сравнительных анализов и наличием опыта работы с данным языком программирования. В качестве среды разработки использовалась Visual Studio Code [2].

3.2 Сведения о модулях программы

Реализованное ПО состоит из трёх модулей:

1) algos.py - в данном модуле реализованы алгоритмы умножения матриц;
2) time.py- в данном модуле реализованы замеры временных характеристик алгоритмов;
3) tests.py - в данном модуле реализованы тесты программ.

3.3 Реализация алгоритмов умножения матриц

Листинг 3.1 — Реализация классического умножения матриц

```
def classicMatrixMultiplation(result, mat_a, mat_b, n, k, m):
    for i in range(n):  # Iterate over rows of mat_a
        for j in range(m):  # Iterate over columns of mat_b
            result[i][j] = sum(mat_a[i][l] * mat_b[l][j] for l in range(k))  # Sum
    return result
```

Листинг 3.2 — Алгоритм Винограда для умножения матриц

```
def vinogradMatrixMultiplation(result, mat_a, mat_b, n, k, m):
    row_vec = [sum(mat_a[i][j << 1] * mat_a[i][(j << 1) + 1] for j in range(k >> 1))
        for i in range(n)]
    col_vec = [sum(mat_b[j << 1][i] * mat_b[(j << 1) + 1][i] for j in range(k >> 1))
        for i in range(m)]

    for i in range(n):
        for j in range(m):
            result[i][j] = -row_vec[i] - col_vec[j]
            for l in range(k >> 1):  # binary instead of multiplication
                result[i][j] += (mat_a[i][(l << 1) + 1] + mat_b[l << 1][j]) *
                    (mat_a[i][l << 1] + mat_b[(l << 1) + 1][j])
```

```

    if k % 2 == 1:
        for i in range(n):
            for j in range(m):
                result[i][j] += mat_a[i][k - 1] * mat_b[k - 1][j]

    return result
```

Листинг 3.3 — Оптимизированный алгоритм Винограда для умножения матриц

```
def vinogradMatrixOptimized(result, mat_a, mat_b, n, k, m):

    row_vec = [0] * n
    col_vec = [0] * m

    for i in range(n):
        for j in range(0, k - (k % 2), 2):
            row_vec[i] += mat_a[i][j] * mat_a[i][j + 1]

    for i in range(m):
        for j in range(0, k - (k % 2), 2):
            col_vec[i] += mat_b[j][i] * mat_b[j + 1][i]


    k_mod = k - (k % 2)
    for i in range(n):
        for j in range(m):
            temp = -row_vec[i] - col_vec[j]
            for l in range(0, k_mod, 2):
                temp += (mat_a[i][l + 1] + mat_b[l][j]) * (mat_a[i][l] + mat_b[l +
                    1][j])
            result[i][j] = temp


    if k % 2 == 1:
        for i in range(n):
            for j in range(m):
                result[i][j] += mat_a[i][k - 1] * mat_b[k - 1][j]

    return result
```

3.4 Реализация тестирования алгоритмов

Для тестирования алгоритмов было реализованы следующие тесты:

1) тест на двух матрицах размером n x k и k x m, где n > 0, k > 0, m > 0;
2) тест на двух матрицах размером n x k и k x m, где n = 1, k > 0, m > 0;

3) тест на двух матрицах размером n x k и k x m, где n > 0, k > 0, m = 1;

4) тест на двух матрицах размером n x k и k x m, где n > 0, k > 0, m > 0, и первая матрица - нулевая;

5) тест на двух матрицах размером n x k и k x m, где n > 0, k > 0, m > 0, и первая матрица - единичная.

Листинг 3.4 — Реализация функции рандомной генерации матриц

```

def generateMatrix(n, k, m):
    a = [[rd.randint(-100, 100) for _ in range(k)] for _ in range(n)]  # Matrix a (n x k)
    b = [[rd.randint(-100, 100) for _ in range(m)] for _ in range(k)]  # Matrix b (k x m)
    res = [[0] * m for _ in range(n)]  # Result matrix (n x m) initialized with zeros
    return a, b, res
```

Листинг 3.5 — Реализация общей функции тестирования

```
def caseTest(matrix_mult_function, n, k, m):
    # Generate random matrices
    mat_a, mat_b, expected_result = generateMatrix(n, k, m)

    # Perform the matrix multiplication
    result = matrix_mult_function(expected_result, mat_a, mat_b, n, k, m)

    # Calculate the expected result using classic matrix multiplication
    expected_result = classicMatrixMultiplation([[0] * m for _ in range(n)], mat_a, mat_b, n, k, m)

    # Check if the results match
    if result == expected_result:
        print(f"Test passed for {matrix_mult_function.__name__}!")
    else:
        print(f"Test failed for {matrix_mult_function.__name__}!")
        print("Expected result:")
        printMatrix(expected_result)
        print("Actual result:")
        printMatrix(result)
```

Листинг 3.6 — Реализация функции для случая нулевой матрицы

```
def testNullCase(n, k, m, case_name):
    print(case_name)
    print('Parameters: ', n, k, m)
    a = [[0 for j in range(0, k)] for i in range(0, n)]
    b = [[rd.randint(-100, 100) for j in range(0, m)] for i in range(0, k)]
```

```
    res = [[0 for j in range(0, m)] for i in range(0, n)]

    res_classic = classicMatrixMultiplation(res.copy(), a, b, n, k, m)
    res_vinograd = res.copy()
    res_vinograd_opt = res.copy()

    print('Multiply result: ')
    printMatrix(a)
    printMatrix(b)
    printMatrix(res_classic)
    printMatrix(res_vinograd)
    printMatrix(res_vinograd_opt)

    print("Checking the results: ")
    print(matrixCheck(res_classic, res_vinograd) == matrixCheck(res_classic,
        res_vinograd_opt) == matrixCheck(res_vinograd_opt, res_vinograd))
    print('===\n')
```

Листинг 3.7 — Реализация для случая единичной матрицы

```
def testOnesCase(n, k, m, case_name):
    print(case_name)
    print('Parameters: ', n, k, m)
    a = [[rd.randint(-100, 100) for j in range(0, k)] for i in range(0, n)]
    b = [[0 for j in range(0, m)] for i in range(0, k)]
    for i in range(0, len(b)):
        b[i][i] = 1
    res = [[0 for j in range(0, m)] for i in range(0, n)]

    classicResult = classicMatrixMultiplation(res.copy(), a, b, n, k, m)
    vinogradResults = res.copy()
    vinoradOptimizedResults = res.copy()

    print('Multiply results: ')
    printMatrix(a)
    printMatrix(b)
    printMatrix(classicResult)
    printMatrix(vinogradResults)
    printMatrix(vinoradOptimizedResults)

    print("Checking the results: ")
    print(matrixCheck(classicResult, vinogradResults) == matrixCheck(classicResult,
        vinoradOptimizedResults) == matrixCheck(vinoradOptimizedResults,
        vinogradResults))
    print('===\n')
```

3.5 Вывод

В данном разделе была представлена реализация классического умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Были разработаны алгоритмы тестирования разработанных методов по методу чёрного ящика.

4 Экспериментальный раздел

В данном разделе будут измерены временные характеристики алгоритмов умножения матриц и сделаны выводы об их временной эффективности.

Таблица 4.1 — Время работы классического алгоритма умножения матриц

Размерность матрицы	Время умножения
10	0.0
25	0.0
50	0.015625
75	0.078125
100	0.15625
125	0.328125
150	0.515625

Таблица 4.2 — Время работы алгоритма Винограда

Размерность матрицы	Время умножения
10	0.0
25	0.0
50	0.015625
75	0.0625
100	0.171875
125	0.375
150	0.609375

Таблица 4.3 — Время работы классического алгоритма оптимизированного алгоритма Винограда

Размерность матрицы	Время умножения
10	0.0
25	0.015625
50	0.015625
75	0.0625
100	0.125
125	0.234375
150	0.375

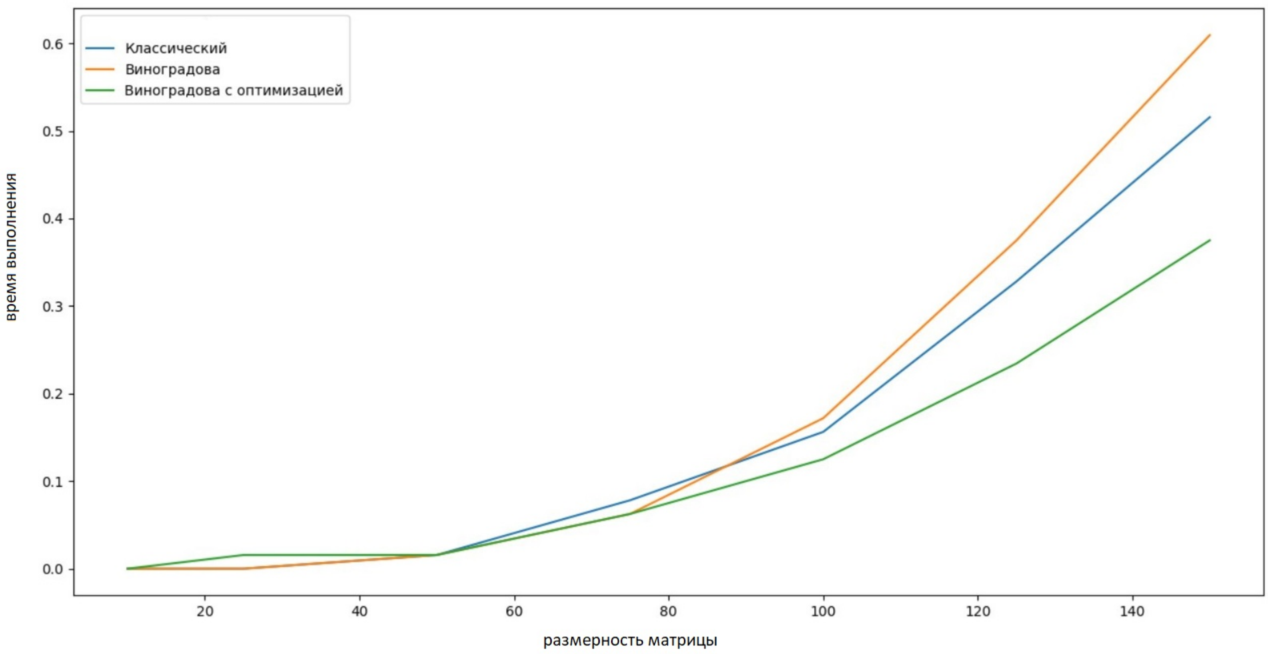

Рисунок 4.1 — График зависимости времени выполнения от размерности матрицы

4.1 Вывод

В результате экспериментов было получено, что на квадратных матрицы размерности от 10 до 80 алгоритмы работают в среднем одинаково. Однако после того, как размерность переходит 100, самым быстрым алгоритмом становится алгоритм Винограда оптимизированный, после него идёт классический алгоритм умножения матриц, и самым медленным становится алгоритм Винограда. В результате можно сделать вывод, что для матриц, размерностью больше 100 предпочтительно использовать оптимизированный алгоритм Винограда.

Заключение

В процессе выполнения данной лабораторной работы были изучены классический алгоритм умножения матриц, алгоритм Винограда для умножения матриц и оптимизированный алгоритм Винограда для умножения матриц. Были выполнены анализ алгоритмов и произведены вычисления трудоёмкости для каждого алгоритма. После чего эти алгоритмы были реализованы при помощи языка Python в IDE Visual Studio Code. Помимо этого были произведены эксперименты с целью получить информацию о временной производительности алгоритмов. В результате было получено, что на квадратных матрицы размерности от 10 до 80 алгоритмы работают в среднем одинаково. Однако после того, как размерность переходит 100, самым быстрым алгоритмом становится алгоритм Винограда оптимизированный, после него идёт классический алгоритм умножения матриц, и самым медленным становится алгоритм Винограда. В результате можно сделать вывод, что для матриц, размерностью больше 100 предпочтительно использовать оптимизированный алгоритм Винограда. Целью данной лабораторной работы являлось изучения алгоритмов умножения матриц, что было успешно достигнуто.

Список литературы

[1] Майкл, Доусон. Python Programming for the Absolute Beginner, 3rd Edition / Доусон Майкл. - Прогресс книга, 2019 - P. 416.

[2] Lutz, M. The IDLE User Interface / M Lutz. - O'Reilly Media, 2013.

[3] Time. https://docs.python.org/3/library/time.html.

[4] Беллман Р. Введение в теорию матриц. — Мир, 1969

[5] DonCoppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. Journal of Symbolic Computation, 9:251-280, 1990