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: «Умножение матриц»

Вариант № 8703

Студент ИУ7-56Б (Группа) _____ (Подпись, дата) Александрова А. Д. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Строганов Д. В. (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является исследование алгоритмов умножения матриц.

Задачи лабораторной работы:

1) реализация алгоритмов умножения матриц — классического, алгоритма Винограда и оптимизированного алгоритма Винограда;

2) исследование ресурсной эффективности реализованных алгоритмов, сравнительный анализ результатов;

3) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитический раздел

1.1 Определение матрицы

Матрицей [1] называют таблицу чисел a_{ik} вида

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

состоящую из m строк и n столбцов.

Пусть A – матрица, тогда $A_{i,j}$ – элемент этой матрицы, который находится на *i-ой* строке и *j-ом* столбце.

1.2 Классический алгоритм умножения матриц

Пусть даны 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

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

будет называться произведением матриц A и B.

Классический алгоритм умножения матриц реализует эту формулу.

1.3 Алгоритм Винограда

Алгоритм Винограда [2] - алгоритм умножения матриц, основанный на приведенных ниже рассуждениях.

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

Их скалярное произведение равно (1.5)

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \tag{1.5}$$

Равенство (1.5) можно переписать в виде (1.6)

$$V \cdot W = (v_1 + w_2) \cdot (v_2 + w_1) + (v_3 + w_4) \cdot (v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \tag{1.6}$$

Выражение в правой части формулы (1.6) допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. При нечетном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов к результату.

1.4 Оптимизации алгоритма Винограда

В рассмотренный выше алгоритм Винограда при программной реализации можно внести следующие оптимизации:

1) инкремент счётчика наиболее вложенного цикла на 2;

2) объединение III и IV частей алгоритма Винограда;

3) введение декремента при вычислении вспомогательных массивов.

2 Конструкторский раздел

2.1 Проектирование алгоритмов

Были разработаны алгоритмы умножения матриц, представленные на рисунках 2.1 — 2.3

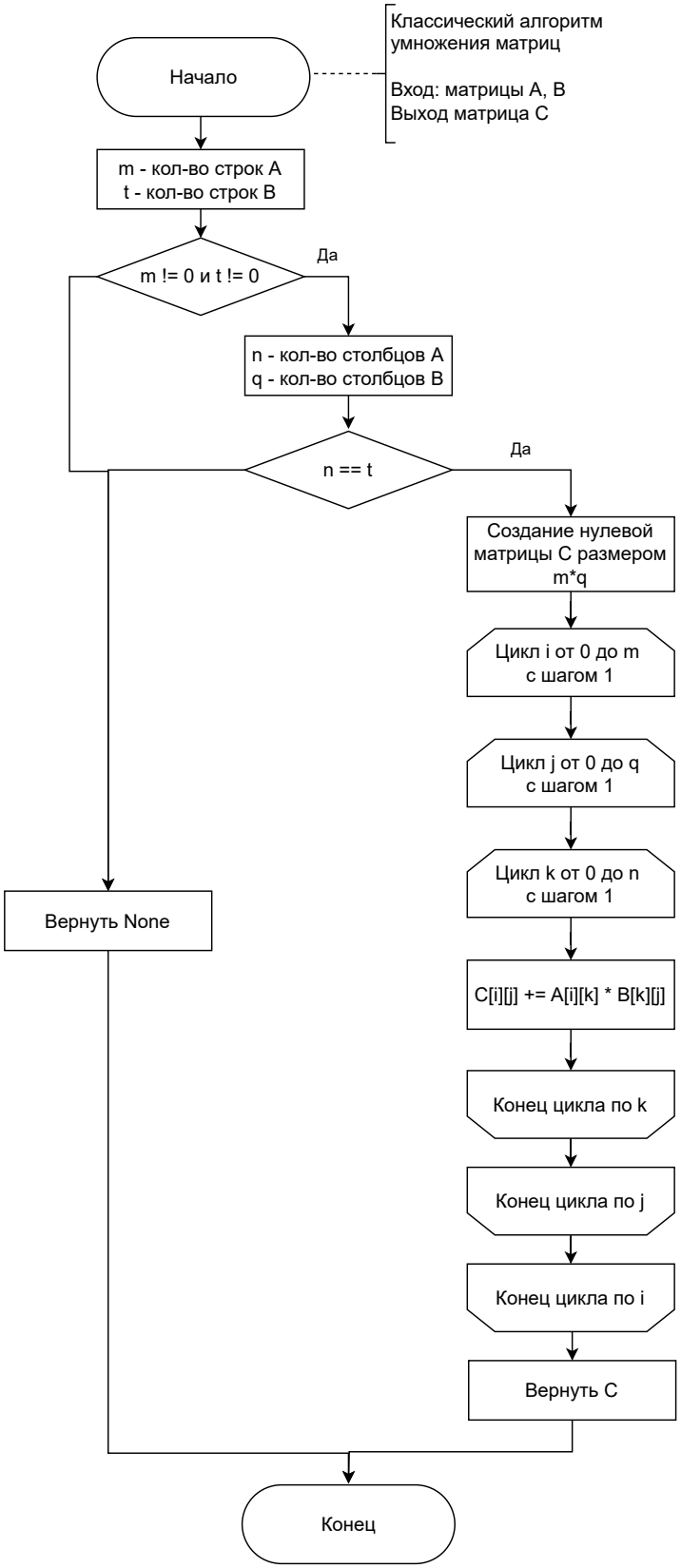

Рисунок 2.1 – Стандартный алгоритм умножения матриц

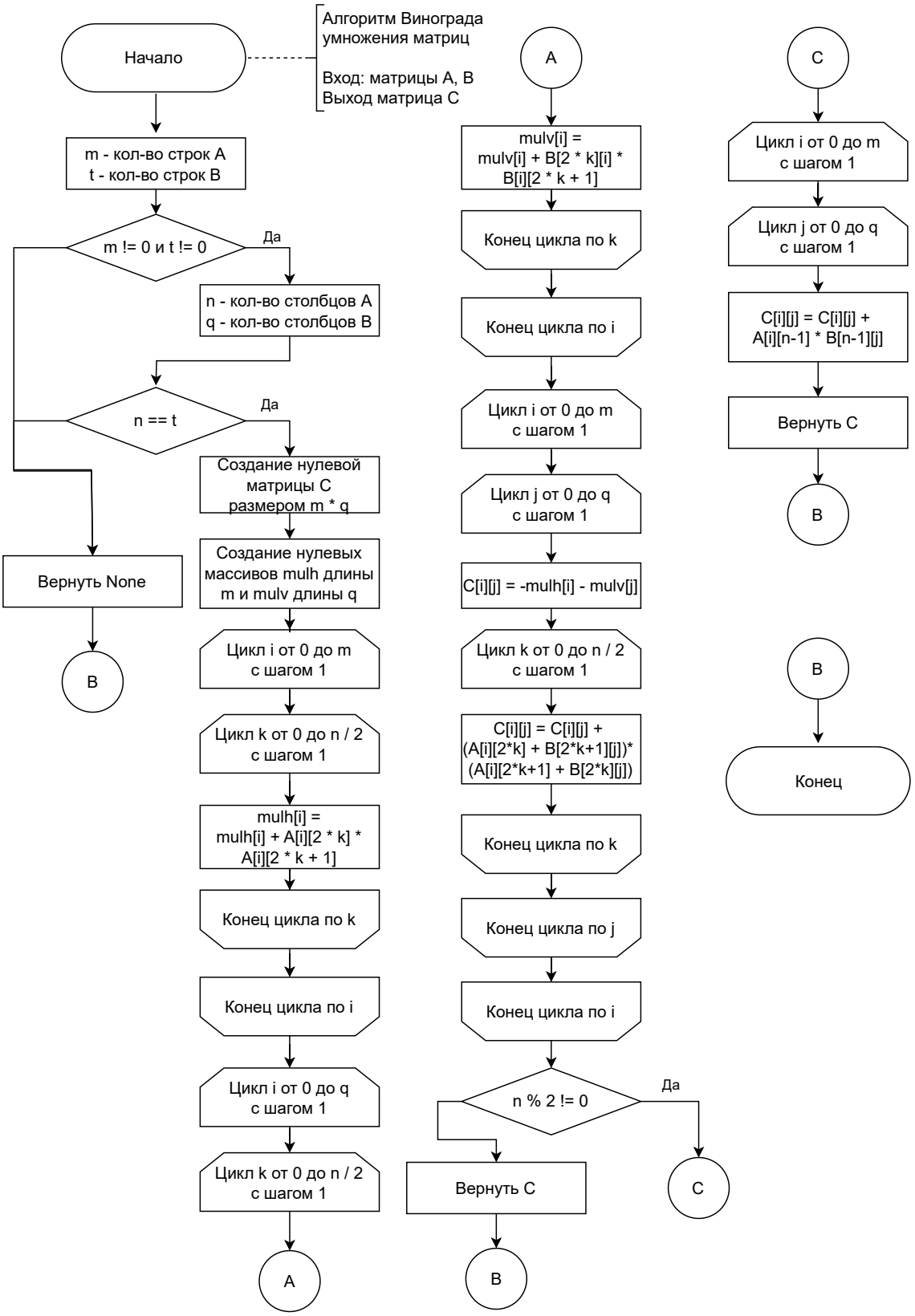

Рисунок 2.2 – Алгоритм Винограда

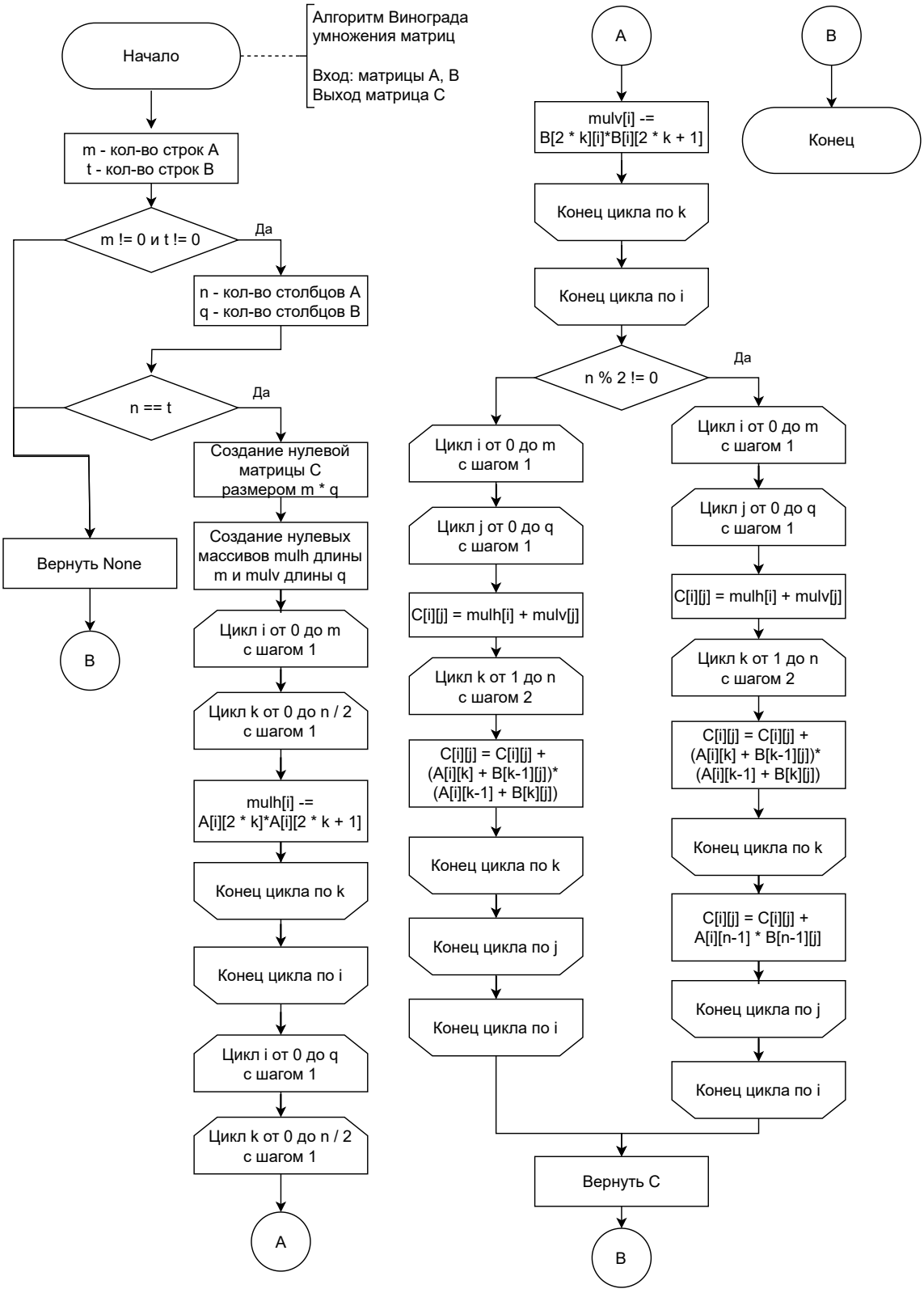

Рисунок 2.3 – Оптимизированный алгоритм Винограда

2.2 Модель вычислений для проведения оценки трудоёмкости

Ниже описана модель вычислений, которая потребуется для определения трудоёмкости каждого отдельного взятого алгоритма сортировки.

1. Трудоёмкость базовых операций:

   — равна 1 для операций в списке (2.1);

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

   — равна 2 для операций в списке (2.2).

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

2. Трудоёмкость условного оператора описана в формуле (2.3).

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

3. Трудоёмкость цикла описана в формуле (2.4).

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

4. Трудоёмкость передачи параметра в функции и возврат из функции равны 0.

2.3 Трудоёмкость классического алгоритма умножения матриц

Для стандартного алгоритма умножения матриц для матрицы A размером $N \cdot M$ и матрицы B размером $M \cdot T$ трудоёмкость будет слагаться из:

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

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

— трудоёмкости цикла по $k \in [1 \dots M]$, которая равна: $f = 2 + 2 + 14M$.

Поскольку трудоёмкость стандартного алгоритма равна трудоёмкости 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 Трудоёмкость алгоритма Винограда

Для алгоритма Винограда для матрицы A размером $N \cdot M$ и матрицы B размером $M \cdot T$ трудоёмкость будет слагаться из:

— трудоёмкости создания и инициализации массивов $mulh$ и $mulv$, которая указана в формуле (2.6);

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

— трудоёмкости заполнения массива $mulh$, которая указана в формуле (2.7);

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

— трудоёмкости заполнения массива $mulv$, которая указана в формуле (2.8);

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

— трудоёмкости цикла заполнения для чётных размеров, которая указана

в формуле (2.9);

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

— трудоёмкости цикла, который дополнительно нужен для подсчёта значений при нечётном размере матрицы, которая указана в формуле (2.10);

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

Тогда для худшего случая (нечётный общий размер матриц):

$$f_{worst} = f_{init} + f_{mulh} + f_{mulv} + f_{cycle} + f_{parity} \approx 16NMT = O(N^3) \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц):

$$f_{best} = f_{init} + f_{mulh} + f_{mulv} + f_{cycle} + f_{parity} \approx 16NMT = O(N^3) \tag{2.12}$$

2.5 Трудоёмкость оптимизированного алгоритма Винограда

Для оптимизированного алгоритма Винограда для матрицы A размером $N \cdot M$ и матрицы B размером $M \cdot T$ трудоёмкость будет слагаться из:

— трудоёмкости создания и инициализации массивов $mulh$ и $mulv$, которая указана в формуле (2.6);

— трудоёмкости заполнения массива $mulh$ с использованием декремента, которая указана в формуле (2.13);

$$f_{mulh} = 2 + N \cdot (4 + \frac{M}{2} \cdot (4 + 5 + 1 + 1 + 3 \cdot 2)) = \\ = 2 + 4N + \frac{17NM}{2} = 2 + 4N + 8,5NM \tag{2.13}$$

— трудоёмкости заполнения массива $mulv$ с использованием декремента,

которая указана в формуле (2.14);

$$f_{mulv} = 2 + T \cdot (4 + \frac{M}{2} \cdot (4 + 5 + 1 + 1 + 3 \cdot 2)) = \\ = 2 + 4T + \frac{17TM}{2} = 2 + 4T + 8,5TM \tag{2.14}$$

— трудоёмкости проверки четности и циклов заполнения для чётных и нечетных размеров с инкрементом на 2, которые указаны в формулах (2.15), (2.16) и (2.17) соответственно;

$$f_{parity} = 3 + \begin{cases} f_{even\_cycle}, & \text{чётная} \\ f_{odd\_cycle}, & \text{иначе} \end{cases} \tag{2.15}$$

$$f_{even\_cycle} = 2 + N \cdot (2 + T \cdot (2 + 6 + 2 + \frac{M}{2} \cdot (2 + 12 + 5 + 1 + 2))) = \\ = 2 + 2N + 10NT + \frac{22NTM}{2} = 2 + 2N + 10NT + 11NTM \tag{2.16}$$

$$f_{odd\_cycle} = 2 + N \cdot (2 + T \cdot (2 + 6 + 2 + \frac{M}{2} \cdot (2 + 12 + 5 + 1 + 2) + 8+ \\ +1 + 3 + 2)) = 2 + 2N + 24NT + \frac{22NTM}{2} = 2 + 2N + 24NT + 11NTM \tag{2.17}$$

Тогда для худшего случая (нечётный общий размер матриц):

$$f_{worst} = f_{init} + f_{mulh} + f_{mulv} + f_{parity} = \\ = f_{init} + f_{mulh} + f_{mulv} + f_{cycle} + f_{odd\_cycle} \approx 11NMT = O(N^3) \tag{2.18}$$

Для лучшего случая (чётный общий размер матриц):

$$f_{best} = f_{init} + f_{mulh} + f_{mulv} + f_{parity} = \\ = f_{init} + f_{mulh} + f_{mulv} + f_{cycle} + f_{even\_cycle} \approx 11NMT = O(N^3) \tag{2.19}$$

3 Технологический раздел

3.1 Требования к программе

К программе предъявлены следующие требования:

— на вход подаются две матрицы и их размеры;

— результатом работы программы является матрица;

— необходимо наличие пользовательского интерфейса для выбора действий и алгоритма работы;

— необходимо наличие замера процессорного времени работы реализованных алгоритмов умножения матриц.

3.2 Средства реализации

Для реализации данной лабораторной был выбран язык Python [3], так как он содержит все необходимые для реализаций алгоритмов инструменты и имеет модуль utime() [4] для замеров времени. Для замеров времени используется функция ticks_us() из модуля utime().

3.3 Программные модули

Программа разбита на следующие модули:

— **main.py** — модуль, реализующий пользовательский интерфейс;

— **funcs.py** — модуль содержит функции умножения, ввода и вывода матриц;

— **tests.py** — модуль для проведения функционального тестирования;

— **time_mes.py** — модуль, реализующий замеры времени;

— **graph.py** — модуль для графического отображения полученных замеров времени.

3.4 Реализация алгоритмов

Реализации алгоритмов представлены в листингах 3.1 — 3.3

Листинг 3.1 – Классический алгоритм умножения матриц

```
def standart_mtx_mul(A, B):
    m, t = len(A), len(B)
    if t == 0 or m == 0:
        return None
    n, q = len(A[0]), len(B[0])
    if n != t:
        return None
    C = [[0] * q for i in range(m)]
    for i in range(m):
        for j in range(q):
            for k in range(n):
                C[i][j] = C[i][j] + A[i][k] * B[k][j]
    return C
```

Листинг 3.2 – Алгоритм Винограда

```
def winograd_mtx_mul(A, B):
    m, t = len(A), len(B)
    if t == 0 or m == 0:
        return None
    n, q = len(A[0]), len(B[0])
    if n != t:
        return None
    C = [[0] * q for i in range(m)]
    mulh = [0] * m
    mulv = [0] * q
    for i in range(m):
        for k in range(n // 2):
            mulh[i] = mulh[i] + A[i][2 * k] * A[i][2 * k + 1]
    for i in range(q):
        for k in range(n // 2):
            mulv[i] = mulv[i] + B[2 * k][i] * B[2 * k + 1][i]
    for i in range(m):
        for j in range(q):
            C[i][j] = - mulh[i] - mulv[j]
            for k in range(n // 2):
                C[i][j] = (C[i][j] + (A[i][2 * k] + B[2 * k +
                   1][j])
                           * (A[i][2 * k + 1] + B[2 * k][j]))
    if n % 2 != 0:
        for i in range(m):
            for j in range(q):
                C[i][j] = C[i][j] + A[i][n - 1] * B[n - 1][j]
    return C
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
def opt_winograd_mtx_mul(A, B):
    m, t = len(A), len(B)
    if t == 0 or m == 0:
        return None
    n, q = len(A[0]), len(B[0])
    if n != t:
        return None
    C = [[0] * q for i in range(m)]
    mulh = [0] * m
    mulv = [0] * q
    for i in range(m):
        for k in range(n // 2):
            mulh[i] -= A[i][2 * k] * A[i][2 * k + 1]
    for i in range(q):
        for k in range(n // 2):
            mulv[i] -= B[2 * k][i] * B[2 * k + 1][i]
    if n % 2 != 0:
        for i in range(m):
            for j in range(q):
                C[i][j] = mulh[i] + mulv[j]
                for k in range(1, n, 2):
                    C[i][j] = (C[i][j] + (A[i][k] + B[k - 1][j])
                               * (A[i][k - 1] + B[k][j]))
                C[i][j] = C[i][j] + A[i][n - 1] * B[n - 1][j]
    else:
        for i in range(m):
            for j in range(q):
                C[i][j] = mulh[i] + mulv[j]
                for k in range(1, n, 2):
                    C[i][j] = (C[i][j] + (A[i][k] + B[k - 1][j])
                               * (A[i][k - 1] + B[k][j]))
    return C
```

3.5 Функциональные тесты

Функциональные тесты приведены в таблице 3.1 и были пройдены успешно всеми алгоритмами.

Таблица 3.1 – Функциональные тесты

Входные данные		**Ожидаемый результат**
Матрица A	**Матрица B**	
$\begin{pmatrix}2\end{pmatrix}$	$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}6\end{pmatrix}$
$\begin{pmatrix}1 & 2\\3 & 4\end{pmatrix}$	$\begin{pmatrix}5 & 6\\7 & 8\end{pmatrix}$	$\begin{pmatrix}19 & 22\\43 & 50\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}0 & 6\\-9 & 5\\-8 & 7\end{pmatrix}$	$\begin{pmatrix}-42 & 37\end{pmatrix}$
$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}0 & 6\\-9 & 5\end{pmatrix}$	None
$\begin{pmatrix}1 & 2 & 3\end{pmatrix}$	$\begin{pmatrix}\end{pmatrix}$	None

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

— Процессор: ARM Cortex-M7 216 MHz.

— Оперативная память: 512 КБайт SRAM.

При замерах времени устройство было нагружено только системными приложениями.

4.2 Временные характеристики

Замеры времени работы алгоритмов проводились на квадратных матрицах размером от 1 до 15. Для каждого размера матрицы замеры были выполнены 100 раз и получено среднее значение, занесенное в результат.

Результаты представлены в таблице 4.1.

По приведенным результатам можно увидеть, что стандартный алгоритм Винограда работает дольше классического алгоритма умножения матриц на всех размерах матриц до 15. При этом оптимизированная версия алгоритма Винограда становится быстрее классического алгоритма при размерах матрицы больших 7. Согласно проведенным замерам, оптимизации алгоритма Винограда ускорили его работу в среднем на 12%.

Таблица 4.1 – Результаты замеров времени работы алгоритмов

Размер матрицы	**Время, мкс**		
	Классический алгоритм	**Алгоритм Винограда**	
		Стандартный	**Оптимизир.**
1	131.200	160.700	155.070
2	196.740	269.950	256.430
3	368.270	479.370	429.700
4	691.670	890.290	800.070
5	1185.040	1457.400	1274.790
6	2029.390	2399.020	2101.490
7	2969.120	3459.850	2989.680
8	4308.480	4807.350	4198.350
9	5992.060	6649.870	5716.530
10	8142.750	8756.130	7548.770
11	10684.820	11456.950	9846.470
12	13791.540	14367.550	12382.840
13	17401.430	18169.560	15630.420
14	21641.010	22128.970	18966.630
15	26539.700	27199.770	23343.650

ЗАКЛЮЧЕНИЕ

В результате исследования сделан вывод, что оптимизированный алгоритм Винограда являются наиболее быстрым из всех рассмотренных, как и можно было предположить по предварительной оценке трудоёмкости, которая у данного алгоритма оказалась ниже прочих.

Таким образом, в результате выполнения лабораторной работы были исследованы и реализованы алгоритмы умножения матриц — классический, алгоритм Винограда и оптимизированный алгоритм Винограда, и проведена оценка их ресурсной эффективности, то есть выполнены все поставленные задачи, и цель выполнения работы — исследование алгоритмов умножения матриц — можно считать достигнутой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Белоусов И. В.* Матрицы и определители, учебное пособие по линейной алгебре //. — ИНСТИТУТ ПРИКЛАДНОЙ ФИЗИКИ, АКАДЕМИИ НАУК РЕСПУБЛИКИ МОЛДОВА, 2006. — С. 1—16.

2. *Погорелов Д. А.* Оптимизация классического алгоритма Винограда для перемножения матриц. / ПОГОРЕЛОВ Д.А., ТАРАЗАНОВ А.М., ВОЛКОВА Л.Л. // ОБРАЗОВАНИЕ И НАУКА В РОССИИ И ЗА РУБЕЖОМ – Москва: Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), 2019. – С. 246 – 249.

3. Welcome to Python [Электронный ресурс]. — — Режим доступа: `https://www.python.org` (Дата обращения: 12.10.2024).

4. Документация модуля utime() [Электронный ресурс]. — — Режим доступа: `https://docs.micropython.org/en/kookaberry/library/utime.html` (Дата обращения: 12.10.2024).