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 по курсу «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Калашков П. А.

Группа ИУ7-56Б

Оценка (баллы)

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

Москва — 2022 г.

Содержание

Введение

В программировании, как и в математике, часто приходится прибегать к использованию матриц. Существует огромное количество областей их применения в этих сферах. Матрицы активно используются при выводе различных формул в физике, таких, как:

— градиент;

— дивергенция;

— ротор.

Нельзя обойти стороной и различные операции над матрицами – сложение, возведение в степень, умножение. При различных задачах размеры матрицы могут достигать больших значений, поэтому оптимизация операций работы над матрицами является важной задачей в программировании. В данной лабораторной работе пойдёт речь об оптимизациях операции умножения матриц.

Целью данной работы является изучение, реализация и исследование алгоритмов умножения матриц – классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) изучить и реализовать алгоритмы – классический, Винограда и его оптимизацию;

2) протестировать перечисленные алгоритмы по времени и по памяти;

3) сравнить и проанализировать время работы реализаций классического алгоритма и алгоритма Винограда;

4) сравнить и проанализировать время работы реализаций алгоритма Винограда и его оптимизации;

5) описать и обосновать полученные результаты в отчёте о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут рассмотрены классический алгоритм умножения матриц и алгоритм Винограда, а также его оптимизированная версия.

1.1 Матрица

Матрицой [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. Числа a_{ik} называются её *элементами.*

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

Можно выделить следующие операции над матрицами:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц в случае, если количество столбцов первой матрицы равно количеству строк второй матрицы. В итоговой матрице количество строк будет, как у первой матрицы, а столбцов – как у второй.

Замечание: операция умножения матриц не коммутативна – если A и B – квадратные матрицы, а C – результат их перемножения, то произведение AB и BA дадут разный результат C.

1.2 Стандартный алгоритм

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 [1].

Стандартный алгоритм [2] реализует данную формулу.

1.3 Алгоритм Винограда

Алгоритм Винограда [2] – алгоритм умножения квадратных матриц. Начальная версия имела асимптотическую сложность алгоритма примерно $O(n^{2,3755})$, где n – размер стороны матрицы, но после доработки он стал обладать лучшей асимптотикой среди всех алгоритмов умножения матриц.

Рассмотрим два вектора $U = (u_1, u_2, u_3, u_4)$ 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 эквивалентно (1.5).

$$V \cdot W = (u_1+w_2)(u_2+w_1)+(u_3+w_4)(u_4+w_3)-u_1u_2-u_3u_4-w_1w_2-w_3w_4. \tag{1.5}$$

За счёт предварительной обработки данных можно получить прирост про-

изводительности: несмотря на то, что полученное выражение требует большего количества операций, чем стандартное умножение матриц, выражение в правой части равенства можно вычислить заранее и запомнить для каждой строки первой матрицы и каждого столбца второй матрицы. Это позволит выполнить лишь два умножения и пять сложений, при учёте, что потом будет сложено только с двумя предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Операция сложения выполняется быстрее, поэтому на практике алгоритм должен работать быстрее обычного алгоритма перемножения матриц.

Стоит упомянуть, что при нечётном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

1.4 Оптимизация алгоритма Винограда

При программной реализации рассмотренного выше алгоритма Винограда можно сделать следующие оптимизации:

1) значение $\frac{N}{2}$, используемое как ограничения цикла подсчёта предварительных данных, можно кэшировать;

2) операцию умножения на 2 программно эффективнее реализовывать как побитовый сдвиг влево на 1;

3) операции сложения и вычитания с присваиванием следует реализовывать при помощи соответствующего оператора $+=$ или $-=$ (при наличии данных операторов в выбранном языке программирования).

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц – стандартного и Винограда, который имеет большую эффективность за счёт предварительных вычислений. Также были рассмотрены оптимизации, которые можно учесть при программной реализации алгоритма Винограда.

Алгоритмы будут получать на вход две матрицы, причём количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы. При вводе пустой матрицы будет выведено сообщение об ошибке. Требуется реализовать программное обеспечение, которое даёт возможность выбрать один из алгоритмов или все сразу, ввести две матрицы и вывести результат их перемножения. Также необходимо провести замеры времени работы реализаций алгоритмов для чётных и нечётных размеров матриц и сравнить результаты, используя графическое представление.

2 Конструкторская часть

В этом разделе будут представлены описания используемых типов данных, а также схемы алгоритмов перемножения матриц – стандартного, Винограда и оптимизации алгоритма Винограда.

2.1 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие типы данных:

— количество строк – целое число;

— количество столбцов – целое число;

— матрица – двумерный список целых чисел.

2.2 Сведения о модулях программы

Программа состоит из шести модулей:

1) *main.py* — файл, содержащий точку входа;

2) *menu.py* — файл, содержащий код меню программы;

3) *test.py* — файл, содержайший код тестирования алгоритмов;

4) *utils.py* — файл, содержащий служебные алгоритмы;

5) *constants.py* — файл, содержайший константы программы;

6) *algorythms.py* — файл, содержащий код всех алгоритмов.

2.3 Разработка алгоритмов

На рисунке 2.1 представлена схема алгоритма для стандартного умножения матриц. На рисунках 2.2–2.3 схема алгоритма Винограда умножения матриц, а на 2.4–2.5 – схема оптимизированного алгоритма Винограда.

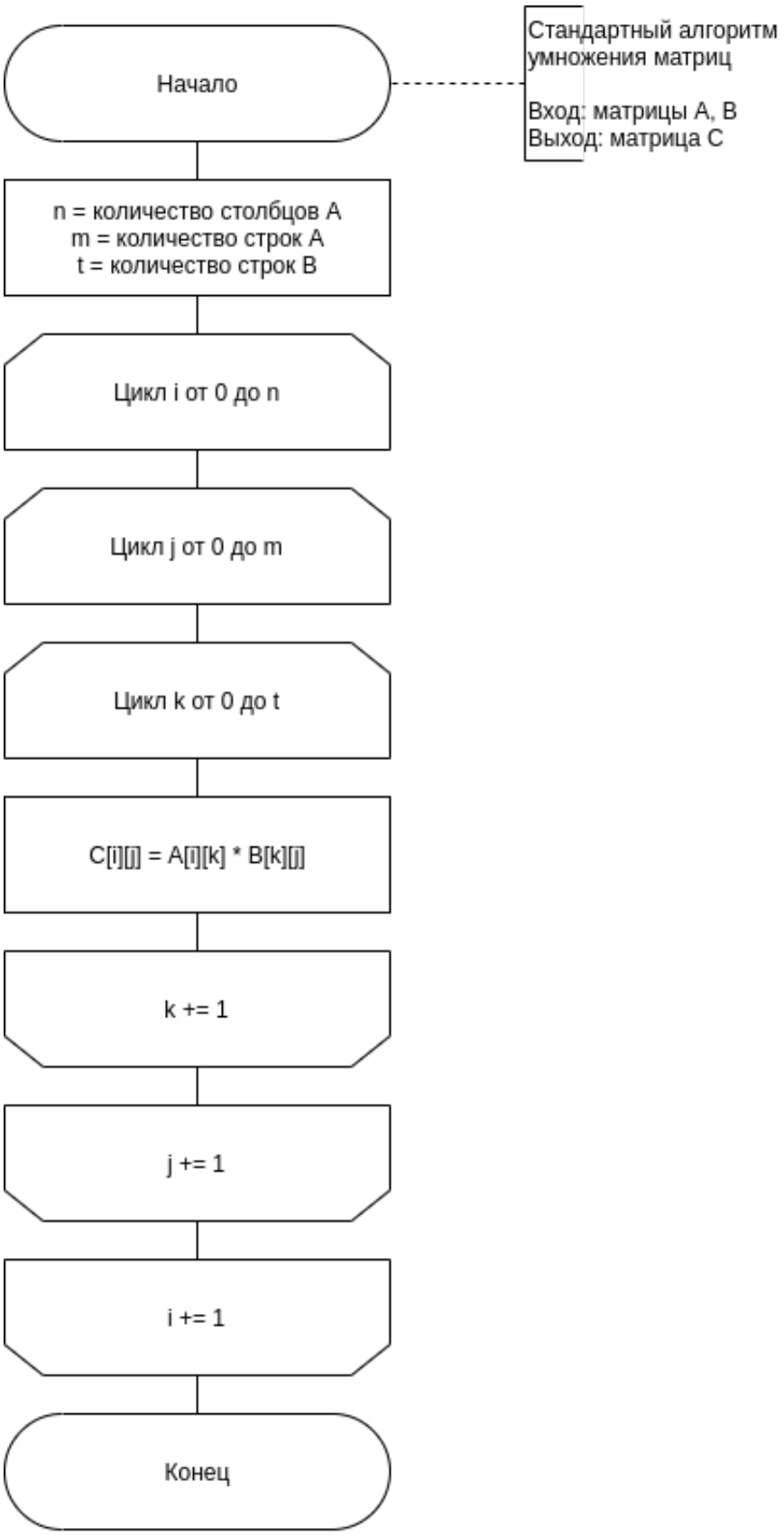

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

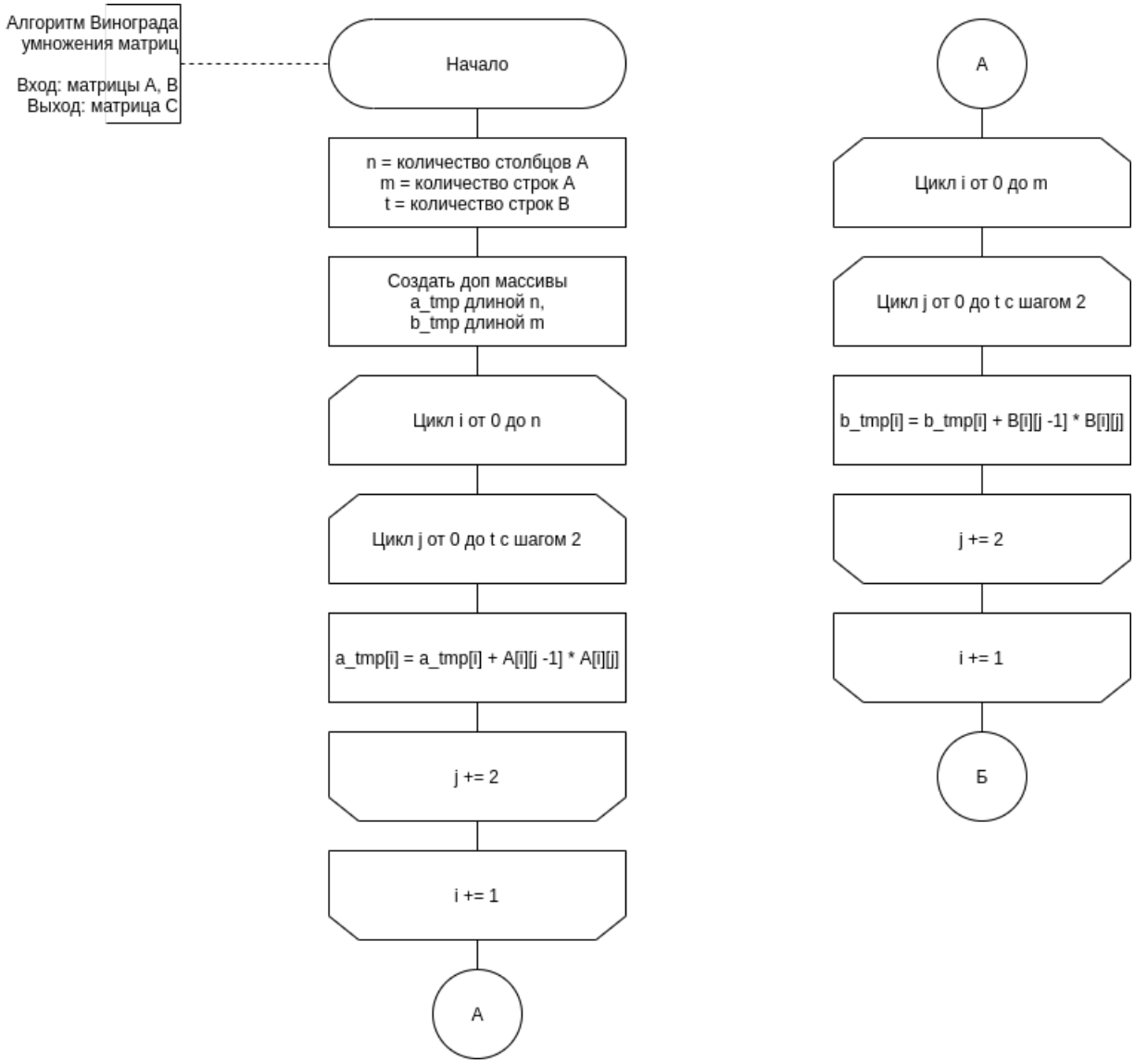

Рисунок 2.2 – Схема умножения матриц по алгоритму Винограда (часть 1)

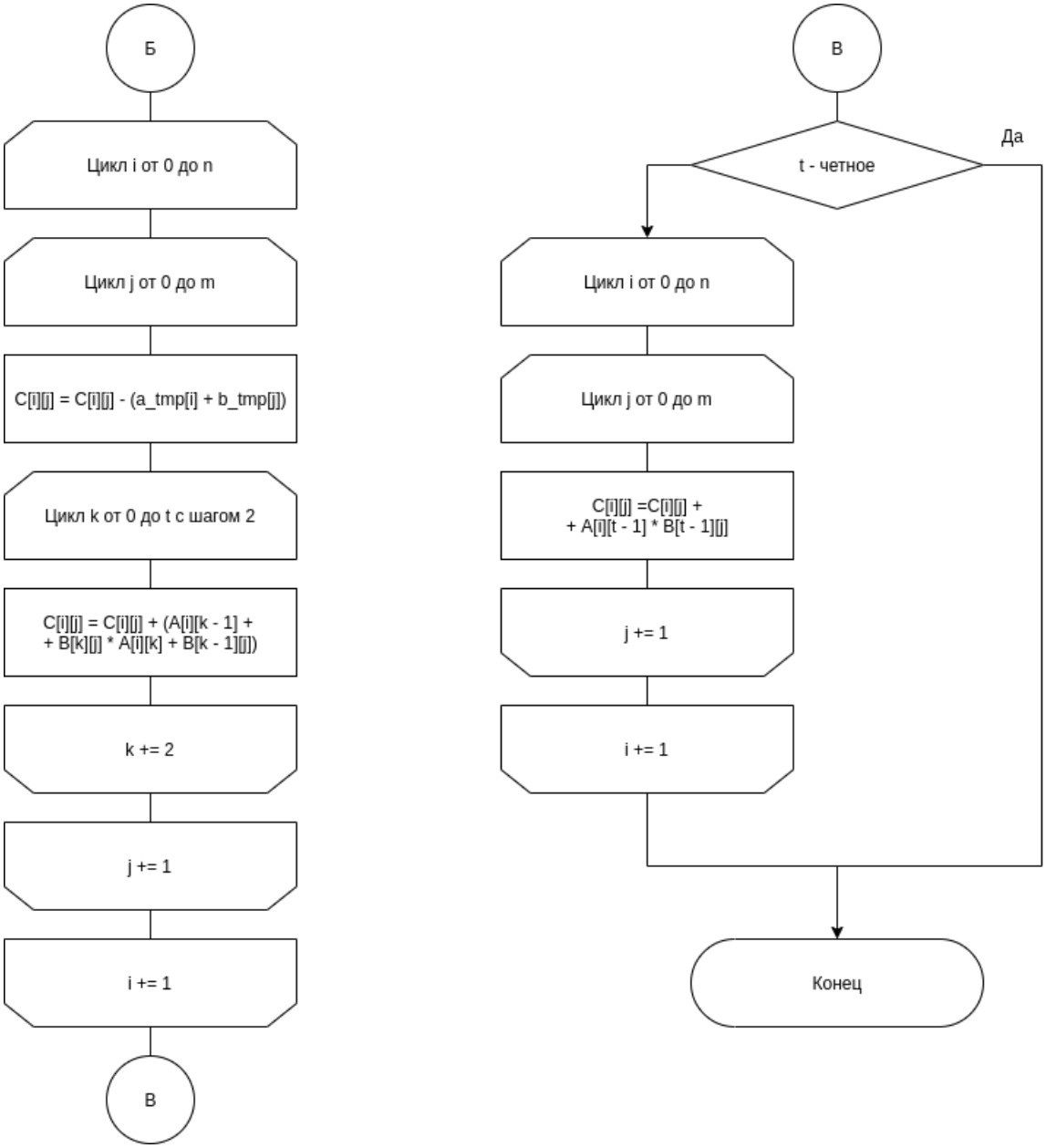

Рисунок 2.3 – Схема умножения матриц по алгоритму Винограда (часть 2)

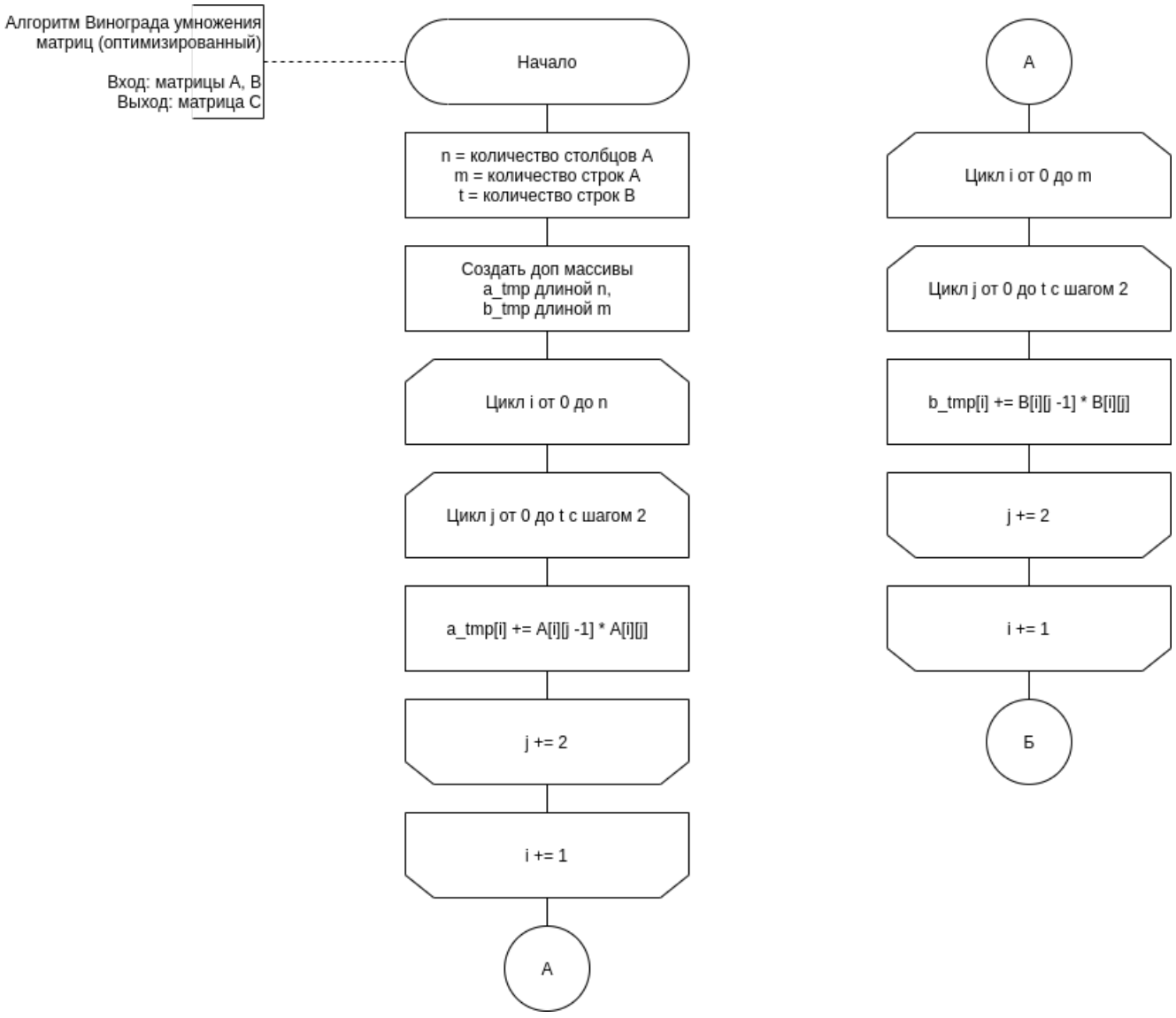

Рисунок 2.4 – Схема умножения матриц по оптимизированному алгоритму Винограда (часть 1)

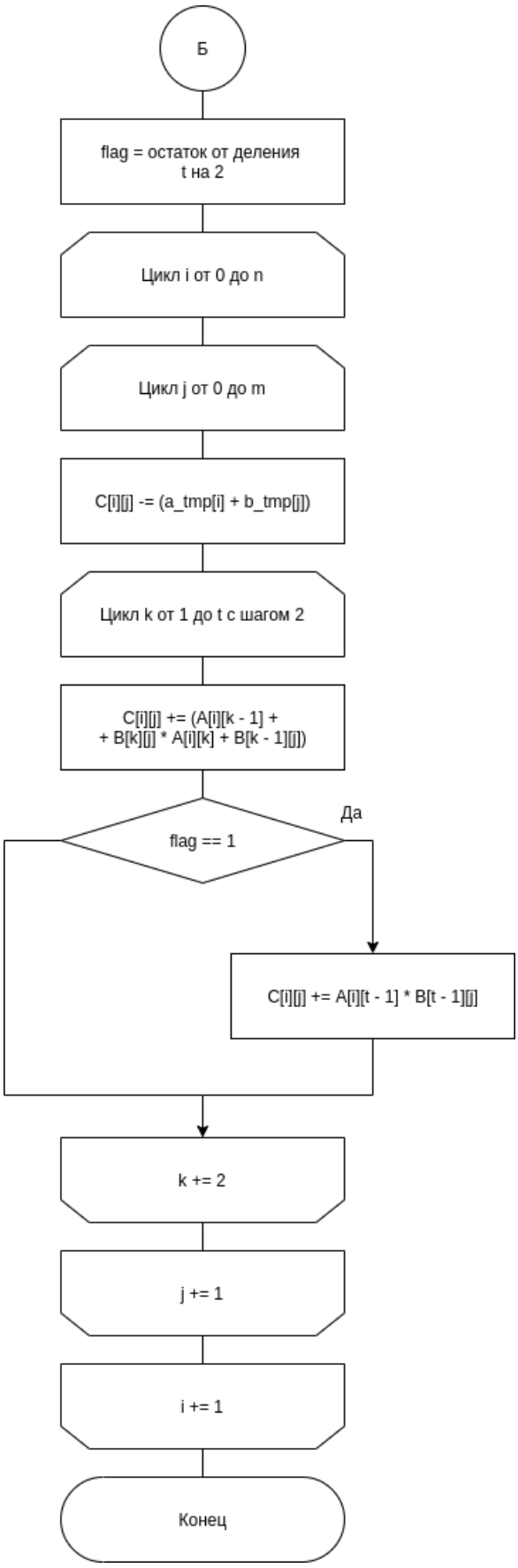

Рисунок 2.5 – Схема умножения матриц по оптимизированному алгоритму Винограда (часть 2)

2.4 Модель вычислений

Чтобы провести вычисление трудоемкости алгоритмов умножения матриц, введем модель вычислений [3]:

1. Операции из списка (2.1) имеют трудоемкость 1;

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

2. Трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3. Трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.3}$$

4. Трудоемкость вызова функции равна 0.

2.5 Трудоемкость алгоритмов

Рассчитаем трудоемкость алгоритмов умножения матриц.

2.5.1 Стандартный алгоритм умножения матриц

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из:

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

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

— цикла по $k \in [1..K]$, трудоёмкость которого: $f = 2 + 2 + 14K$.

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

$$\begin{aligned} f_{standard} &= 2 + M \cdot (4 + N \cdot (4 + 14K)) = 2 + 4M + 4MN + 14MNK \\ &\approx 14MNK \end{aligned} \tag{2.4}$$

2.5.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, нужно учесть следующее:

— создания и инициализации массивов a_{tmp} и b_{tmp}, трудоёмкость которых (2.5);

$$f_{init} = M + N; \tag{2.5}$$

— заполнения массива a_{tmp}, трудоёмкость которого (2.6);

$$f_{a_{tmp}} = 2 + K(2 + \frac{M}{2} \cdot 11); \tag{2.6}$$

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

$$f_{b_{tmp}} = 2 + K(2 + \frac{N}{2} \cdot 11); \tag{2.7}$$

— цикла заполнения для чётных размеров, трудоёмкость которого (2.8);

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 23)); \tag{2.8}$$

— цикла, который дополнительно нужен для подсчёта значений при нечётном размере матрицы, трудоемкость которого (2.9);

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + M \cdot (4 + 14N), & \text{иначе.} \end{cases} \tag{2.9}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем (2.10).

$$f_{worst} = f_{a_{tmp}} + f_{b_{tmp}} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.10}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.11).

$$f_{best} = f_{a_{tmp}} + f_{a_{tmp}} + f_{cycle} + f_{last} \approx 11.5 \cdot MNK \tag{2.11}$$

2.5.3 Оптимизированный алгоритм Винограда

Оптимизация заключается в:

— кэшировании значения $\frac{N}{2}$ в циклах;

— использовании побитового сдвига вместо умножения на 2;

— замены операции сложения и вычитания на операции $+=$ и $-=$ соответственно.

Тогда трудоемкость оптимизированного алгоритма Винограда состоит из:

1) создания и инициализации массивов a_{tmp} и b_{tmp} (2.5);

2) заполнения массива a_{tmp}, трудоёмкость которого (2.6);

3) заполнения массива b_{tmp}, трудоёмкость которого (2.7);

4) цикла заполнения для чётных размеров, трудоёмкость которого (2.12);

$$f_{cycle} = 2 + M \cdot (4 + N \cdot (11 + \frac{K}{2} \cdot 18)); \tag{2.12}$$

5) условие, которое нужно для дополнительных вычислений при нечётном размере матрицы, трудоемкость которого (2.13);

$$f_{last} = \begin{cases} 1, & \text{чётная,} \\ 4 + M \cdot (4 + 10N), & \text{иначе.} \end{cases} \tag{2.13}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем (2.14).

$$f_{worst} = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.14}$$

Для лучшего случая (чётный общий размер матриц) имеем (2.15).

$$f_{best} = f_{MH} + f_{MV} + f_{cycle} + f_{last} \approx 9MNK \tag{2.15}$$

2.6 Классы эквивалентности функционального тестирования

Для функциональные тестирования выделены классы эквивалентности, представленные ниже.

1. Одна из матриц – пустая;
2. Количество столбцов одной матрицы не равно количеству строк второй матрицы;
3. Перемножение квадратных матриц;
4. Перемножение матриц разных размеров (при этом количество столбцов одной матрицы не равно количеству строк второй матрицы).

Вывод

В данном разделе были построены схемы алгоритмов умножения матриц, рассматриваемых в лабораторной работе, были описаны классы эквивалентности для функционального тестирования и модули программы. Проведённая теоретическая оценка трудоемкости алгоритмов показала, что в трудоёмкость выполнения алгоритма Винограда в любом случае в 1.2 раза меньше, чем у стандартного алгоритма умножения матриц.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги реализаций алгоритмов умножения матриц - стандартного, Винограда и оптимизированного алгоритма Винограда.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [4]. В текущей лабораторной работе требуется замерить процессорное время работы выполняемой программы и визуализировать результаты при помощи графиков. Инструменты для этого присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time(...)* из библиотеки *time* [5].

3.2 Реализации алгоритмов

В листингах 3.1-3.3 представлены реализации алгоритмов умножения матриц — стандартного, Винограда и оптимизированного алгоритма Винограда.

Листинг 3.1 – Реализация стандартного алгоритма умножения матриц

```
1  def stdAlg(matA: Matrix, matB: Matrix) -> Matrix:
2      if (len(matA[0]) != len(matB)):
3          return []
4      n = len(matA)
5      m = len(matA[0])
6      t = len(matB[0])
7      result = [[0] * t for _ in range(n)]
8      for i in range(n):
9          for j in range(m):
10             for k in range(t):
11                 result[i][j] += matA[i][k] * matB[k][j]
12
13     return result
```

Листинг 3.2 – Реализация алгоритма Винограда умножения матриц

```
def winAlg(matA: Matrix, matB: Matrix) -> Matrix:
    if (len(matA[0]) != len(matB)):
        return []

    n = len(matA)
    m = len(matA[0])
    t = len(matB[0])

    result = [[0] * t for _ in range(n)]

    rowSum = [0] * n
    colSum = [0] * t

    for i in range(n):
        for j in range(0, m // 2):
            rowSum[i] = rowSum[i] + matA[i][2 * j] * matA[i][2 * j
               + 1]

    for i in range(t):
        for j in range(0, m // 2):
            colSum[i] = colSum[i] + matB[2 * j][i] * matB[2 * j +
               1][i]


    for i in range(n):
        for j in range(t):
            result[i][j] = -rowSum[i] - colSum[j]
            for k in range(0, m // 2):
                result[i][j] = result[i][j] + (matA[i][2 * k + 1] +
                   matB[2 * k][j]) * (matA[i][2 * k] + matB[2 * k +
                   1][j])

    if (m % 2 == 1):
        for i in range(n):
            for j in range(t):
                result[i][j] = result[i][j] + matA[i][m - 1] *
                   matB[m - 1][j]

    return result
```

Листинг 3.3 – Реализация оптимизированного алгоритма Винограда умножения матриц

```
def optWinAlg(matA: Matrix, matB: Matrix) -> Matrix:
    if (not len(matA) or len(matA[0]) != len(matB)):
        return []

    n = len(matA)
    m = len(matA[0])
    t = len(matB[0])

    result = [[0] * t for _ in range(n)]

    rowSum = [0] * n
    colSum = [0] * t

    halfM = m // 2

    for i in range(n):
        for j in range(0, halfM):
            rowSum[i] += matA[i][j << 1] * matA[i][(j << 1) + 1]

    for i in range(t):
        for j in range(0, halfM):
            colSum[i] += matB[j << 1][i] * matB[(j << 1) + 1][i]


    for i in range(n):
        for j in range(t):
            result[i][j] = -rowSum[i] - colSum[j]
            for k in range(0, halfM):
                result[i][j] += (matA[i][(k << 1) + 1] + matB[k <<
                    1][j]) * (matA[i][k << 1] + matB[(k << 1) +
                    1][j])

    if (m % 2 == 1):
        for i in range(n):
            for j in range(t):
                result[i][j] += + matA[i][m - 1] * matB[m - 1][j]

    return result
```

3.3 Функциональные тесты

В таблице 3.1 приведена методология тестирования функций, реализующих алгоритмы умножения матриц, рассматриваемых в данной лабораторной работе. Тесты для всех алгоритмов пройдены *успешно*.

Таблица 3.1 – Функциональные тесты

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$(\ )$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 7 \end{pmatrix}$	$\begin{pmatrix} 8 \end{pmatrix}$	$\begin{pmatrix} 56 \end{pmatrix}$

Вывод

Были представлены листинги реализаций всех алгоритмов умножения матриц – стандартного, Винограда и оптимизированного алгоритма Винограда. Также в данном разделе была приведена информации о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программа, а также проведён сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени работы реализации алгоритма статической раздачи информации, представлены далее:

— операционная система Mac OS Monterey Версия 12.5.1 (21G83) [6] x86_64;

— память 16 ГБ;

— четырёхъядерный процессор Intel Core i7 с тактовой частотой 2,7 ГГц [7].

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен пример работы программы: возможность выбрать алгоритм умножения матриц, ввести матрицы, и посмотреть результат перемножения.

```
Меню
 1. Станадартное умножение матриц
 2. Алгоритм Винограда
 3. Оптимизированный алгоритм Винограда
 4. Все алгоритмы вместе
 5. Замерить время
 0. Выход

 Выбор: 2

Введите количество строк: 3
Введите количество столбцов: 3

Введите матрицу построчно (в одной строке - все числа для данной строки матрицы):
1 2 3
4 5 6
7 8 9

Введите количество строк: 3
Введите количество столбцов: 3

Введите матрицу построчно (в одной строке - все числа для данной строки матрицы):
1 0 0
0 1 0
0 0 1
Результат:
1  2  3
4  5  6
7  8  9
```

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Для замера процессорного времени используется функция process_time(...) из библиотеки time на Python. Функция возвращает пользовательское процессорное время типа float.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для чётных размеров матриц от 10 до 100 по 100 раз на различных входных матрицах, также для нечётных размеров матриц от 11 до 101 по 100 раз на различных данных.

Результаты замеров приведены в таблицах 4.1–4.2 (время в мс).

Таблица 4.1 – Результаты замеров времени (чётные размеры матриц)

Размер	Стандартный	Винограда	Винограда (опт.)
10	0.2522	0.2274	0.2517
20	1.6957	1.4082	1.2897
30	5.5724	4.5085	4.1747
40	12.8172	10.4253	9.5714
50	24.8676	20.1231	18.5403
60	42.9029	34.1578	31.9409
70	67.6836	53.6597	50.7058
80	101.7345	80.3008	75.6634
90	145.0621	114.0665	107.3641
100	199.4262	154.5421	146.0644

Таблица 4.2 – Результаты замеров времени (нечётные размеры матриц)

Размер	Стандартный	Винограда	Винограда (опт.)
11	0.3037	0.2853	0.2954
21	1.9355	1.6225	1.4955
31	6.0607	5.0755	4.5675
41	13.7567	11.2067	10.3181
51	26.3087	21.1598	19.7767
61	44.7732	37.0524	33.6701
71	70.5573	56.5121	52.6599
81	104.9317	84.3000	78.6872
91	151.3442	117.5793	111.0584
101	202.5043	160.0509	154.0139

Также на рисунках 4.2, 4.3 приведены графические результаты замеров.

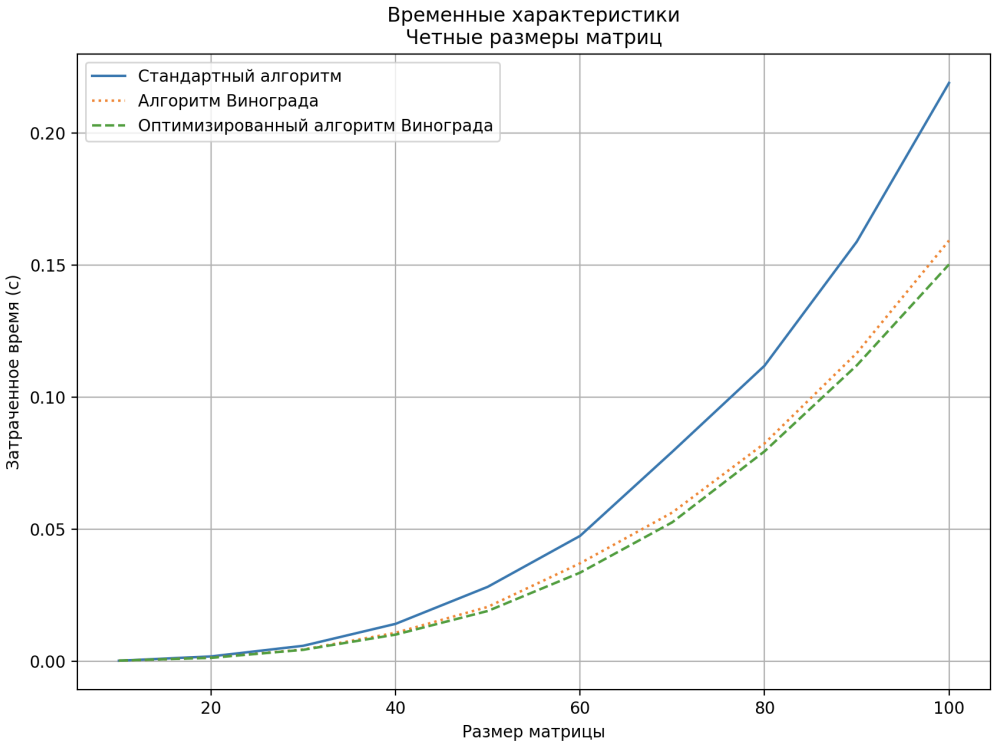

Рисунок 4.2 – Сравнение по времени алгоритмов умножения матриц на чётных размерах матриц

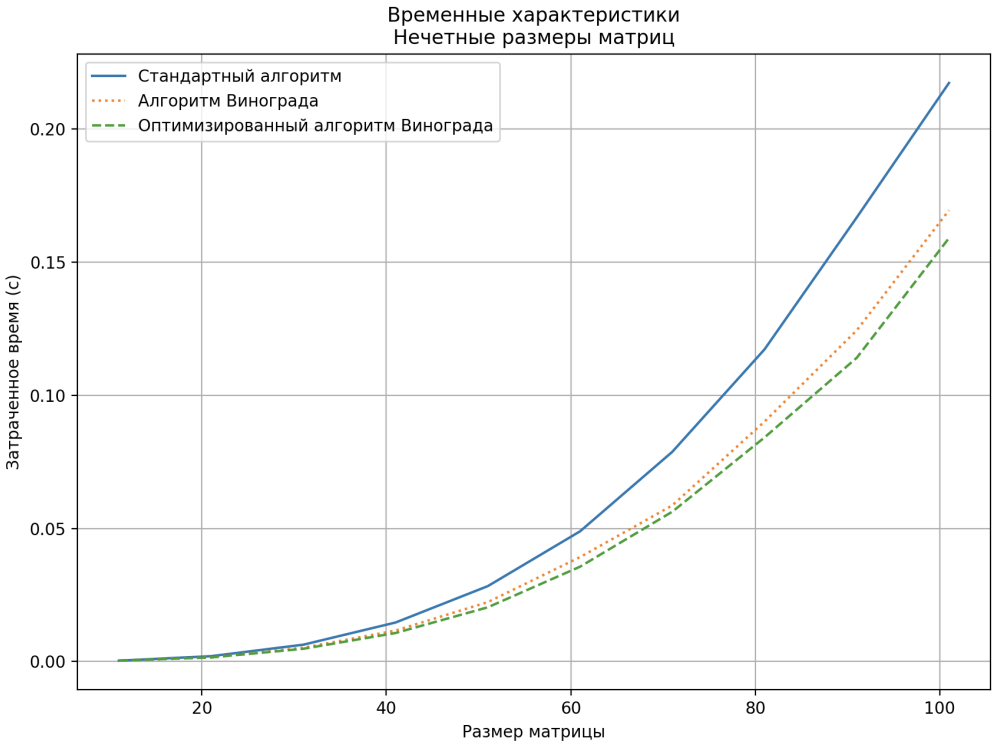

Рисунок 4.3 – Сравнение по времени алгоритмов умножения матриц на нечётных размерах матриц

При размерах матрицы больше 10, алгоритм Винограда работает быстрее стандартного алгоритма более чем в 1.1 раза, а оптимизированный алгоритм Винограда работает быстрее стандартного примерно в 1.2 раза (рисунки 4.2, 4.3). Стоит отметить, что на чётных размерах матриц реализация алгоритма Винограда в 1.1 раз быстрее, чем на нечётных в силу того, что нужно проводить дополнительные вычисления для крайних строк и столбцов.

Вывод

В результате эксперимента было получено, что при больших размерах матриц (свыше 10), алгоритм Винограда быстрее стандартного алгоритма более, чем 1.1 раза, а оптимизированный алгоритм Винограда быстрее стандартного алгоритма в 1.2 раза. В итоге, можно сказать, что при таких данных следует использовать оптимизированный алгоритм Винограда.

Также при проведении эксперимента было выявлено, что на чётных размерах реализация алгоритма Винограда в 1.1 раза быстрее, чем на нечётных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов. Следовательно, стоит использовать алгоритм Винограда для матриц, которые имеют чётные размеры.

Заключение

В результате исследования было определено, что стандартный алгоритм умножения матриц проигрывает по времени алгоритму Винограда примерно в 1.1 раза из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций – операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает оптимизированный алгоритм Винограда – он примерно в 1.2 раза быстрее алгоритма Винограда на размерах матриц свыше 10 из-за замены операций равно и плюс на операцию плюс-равно, а также за счёт замены операции умножения операцией сдвига, что дает проводить часть вычислений быстрее. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда. Также стоит упомянуть, что алгоритм Винограда работает на чётных размерах матриц примерно в 1.1 раза быстрее, чем на нечётных, что связано с тем, что нужно произвести часть дополнительных вычислений для крайних строк и столбцов матриц, поэтому алгоритм Винограда лучше работает чётных размерах матриц.

В результате лабораторной работы были изучены, реализованы и исследованы алгоритмы умножения матриц – классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда.

Были решены следующие задачи:

1) изучены и реализованы алгоритмы умножения матриц - стандартный, Винограда и оптимизированный алгоритм Винограда;

2) проведён сравнительный анализ по времени алгоритмов умножения матриц на чётных размерах матриц;

3) проведён сравнительный анализ по времени алгоритмов умножения матриц на нечётных размерах матриц;

4) проведён сравнительный анализ по времени алгоритмов алгоритмов между собой;

5) подготовлен отчёт о лабораторной работе в виде расчётно-пояснительной записки к ней.

Список использованных источников

1. Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. – М.: Гуманит. изд. центр ВЛАДОС, 2003. С. 45–49.

2. Kakaradov B. Ultra-Fast Matrix Multiplication: An Empirical Analysis of Highly Optimized Vector Algorithms, 2004 [Электронный ресурс]. Режим досступа: `https://cs.stanford.edu/people/boyko/pubs/MatrixMult_SURJ_2004.pdf` (дата обращения: 04.10.2022).

3. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. – М. Наука, Физматлит, 2007. с. 376.

4. Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 04.10.2022).

5. time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 17.09.2022).

6. macOS Monterey [Электронный ресурс]. Режим доступа: `https://www.apple.com/macos/monterey/` (дата обращения: 17.09.2022).

7. Процессор Intel® Core™ i7 [Электронный ресурс]. Режим доступа: `https://www.intel.com/processors/core/i7/docs` (дата обращения: 17.09.2022).