МГТУ им. Баумана

Лабораторная работа №2

По курсу: "Анализ алгоритмов"

Алгоритмы умножения матриц

Работу выполнил: студент группы ИУ7-53Б
Наместник Анастасия

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

Москва, 2020

Оглавление

Введение

Матрица - прямоугольная таблица, заполненная числами. Важнейшие характеристики матрицы – число строк и число столбцов. Сами числа называют элементами матрицы и характеризуют их положением в матрице, задавая номер строки и номер столбца и записывая их в виде двойного индекса, причем вначале записывают номер строки, а затем столбца.

Пусть есть два конечных множества.

1. Номера строк: M = 1, 2,..., n.
2. Номера столбцов: N = 1, 2,..., m, где n и m - натуральные числа.

Назовем матрицей A размерностью m x n (n - строк, m - столбцов) с элементами из некоторого кольца или поля K отображение вида A:N x M -> K. Матрица записывается как (формула 1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \quad (1)$$

Простейшими действиями с матрицами являются следующие операции.

1. Умножение матрицы на число. Для этого необходимо умножить каждый элемент матрицы на данное число.
2. Сложение матриц. Складывать можно только матрицы одинакового размера, то есть имеющие одинаковое число строк и одинаковое

число столбцов. При сложении матриц соответствующие их элементы складываются.

3. Добавить оптимизации к алгоритму Винограда.

4. Транспонирование матрицы. При транспонировании у матрицы строки становятся столбцами и наоборот.

Целью данной лабораторной работы является изучение, реализация и сравнительный анализ алгоритмов стандартного алгоритма умножения матриц, алгоритма Винограда и алгоритма с оптимизациями.

В данной лабораторной работе требуется решить четыре задачи.

1. Изучить и программно реализовать стандартный алгоритм умножения матриц.

2. Изучить и программно реализовать алгоритм Винограда умножения матриц.

3. Добавить оптимизации к алгоритму Винограда.

4. Сделать сравнительный анализ по затрачиваемым ресурсам (времени) компьютера на реализацию каждого рассмотренного алгоритма.

1 | Аналитическая часть

Произведение матриц AB состоит из всех возможных комбинаций скалярных произведений вектор-строк матрицы A и вектор-столбцов матрицы B (рис. 1.1).

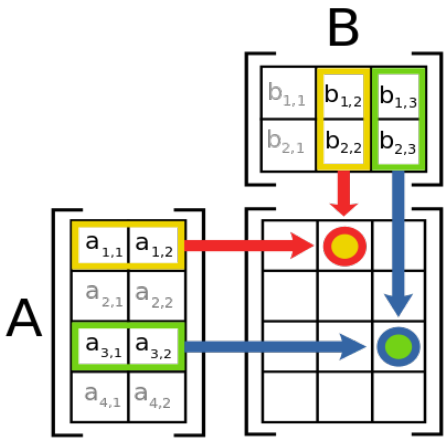

Рис. 1.1: Произведение матриц

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первой матрице равно числу строк во второй.

1.1 Стандартный алгоритм умножения матриц

Допустим имеется матрицы A (формула 1) и B (формула 1.1)

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{pmatrix} \tag{1.1}$$

Матрица C = AB будет размерностью $l \times n$, где матрица A размерностью $l \times m$, а матрица B $m \times n$ Тогда каждый элемент матрицы C выражается формулой (1.2)

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{ri} \quad (i = 1, 2, \dots l; j = 1, 2, \dots n) \tag{1.2}$$

1.2 Умножение матриц по Винограду

Каждый элемент в матрице C, которая является результатом умножения двух матриц, представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. В алгоритме умножение матриц по Винограду предложено сделать предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора V (формула 1.3) и W (формула 1.4).

$$V = (v_1, v_2, v_3, v_4) \tag{1.3}$$

$$W = (w_1, w_2, w_3, w_4) \tag{1.4}$$

Их скалярное произведение равно 1.5.

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

Равенство 1.5 можно записать в виде 1.6.

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

Выражение в правой части равенства 1.6 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. В случае нечетной общей размерности к каждому элементу результирующей матрицы следует добавить произведение элементов первой и второй матрицы, где каждый j-ый элемент первой матрицы и каждый i-ый элемент второй матрицы будут равны количеству столбцов первой матрицы. Оптимизированный алгоритм Винограда умножения матриц допускает следующие модификации.

1. Использование буфера вместо неоднократного обращения к ячейке памяти.
2. Использование битовых (логических) операций.

1.3 Вывод

Были рассмотрены основные теоретические сведения о различных способах реализации произведения матриц, которые в дальнейшем потребуются при разработке программного продукта.

2 | Конструкторская часть

2.1 Схемы алгоритмов

На рисунке 2.1 представлена схема стандартного алгоритма умножения матриц.

На рисунках 2.2 -2.3 представлена схема алгоритма Винограда умножения матриц.

На рисунках 2.4-2.5 представлена схема оптимизированного алгоритма Винограда умножения матриц.

2.2 Вывод

В данном разделе были рассмотрены схемы трех алгоритмов: стандартного алгоритма умножения матриц, алгоритма Винограда умножения матриц и алгоритма Винограда умножения матриц с оптимизациями. Из полученных данных можно отметить, что объем кода растет от стандартного алгоритма к оптимизированному алгоритму Винограда, что объясняется использованием дополнительных переменных и структур данных в алгоритме Винограда умножения матриц.

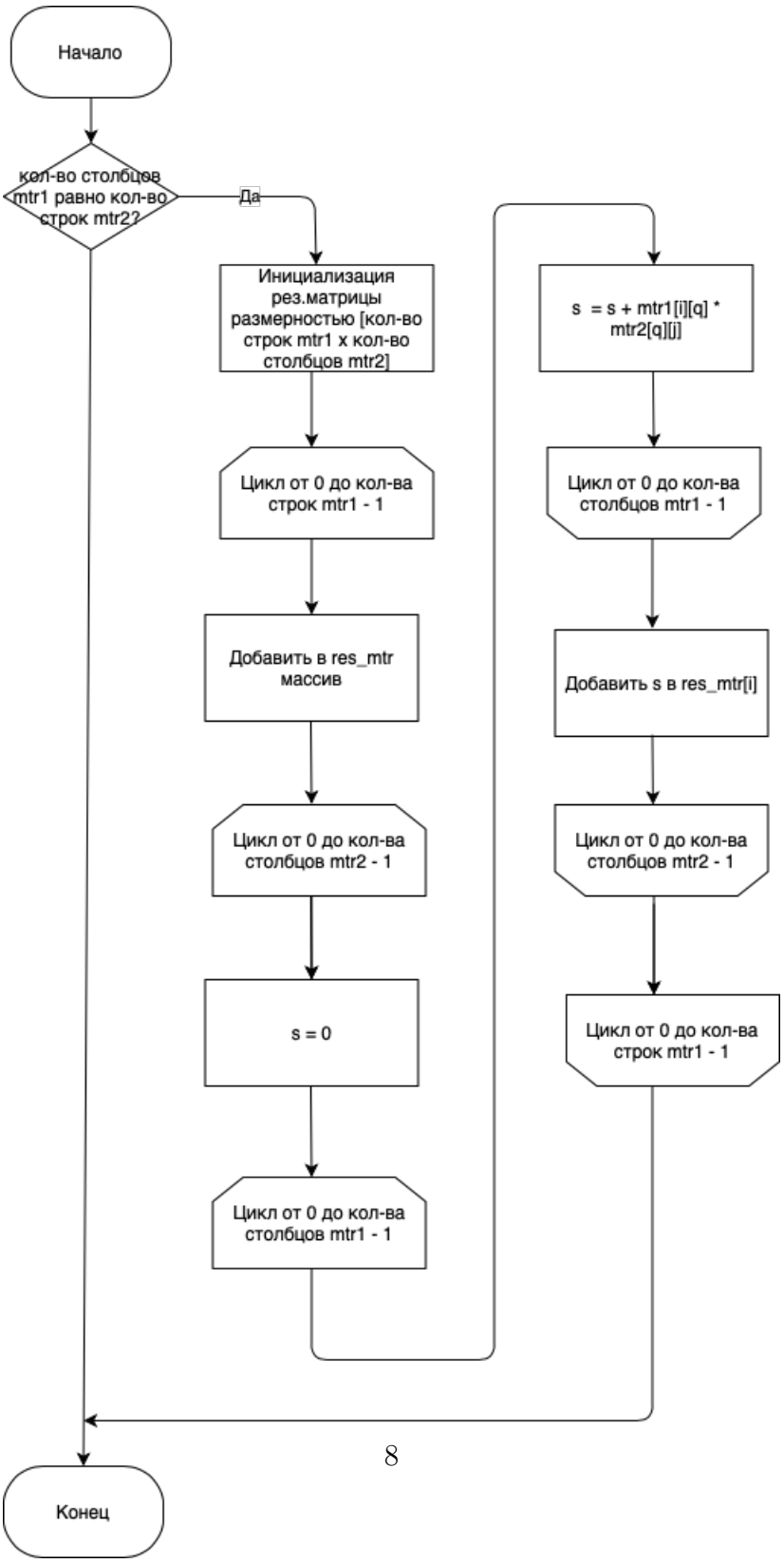

Рис. 2.1: Схема алгоритма стандартного умножения матриц

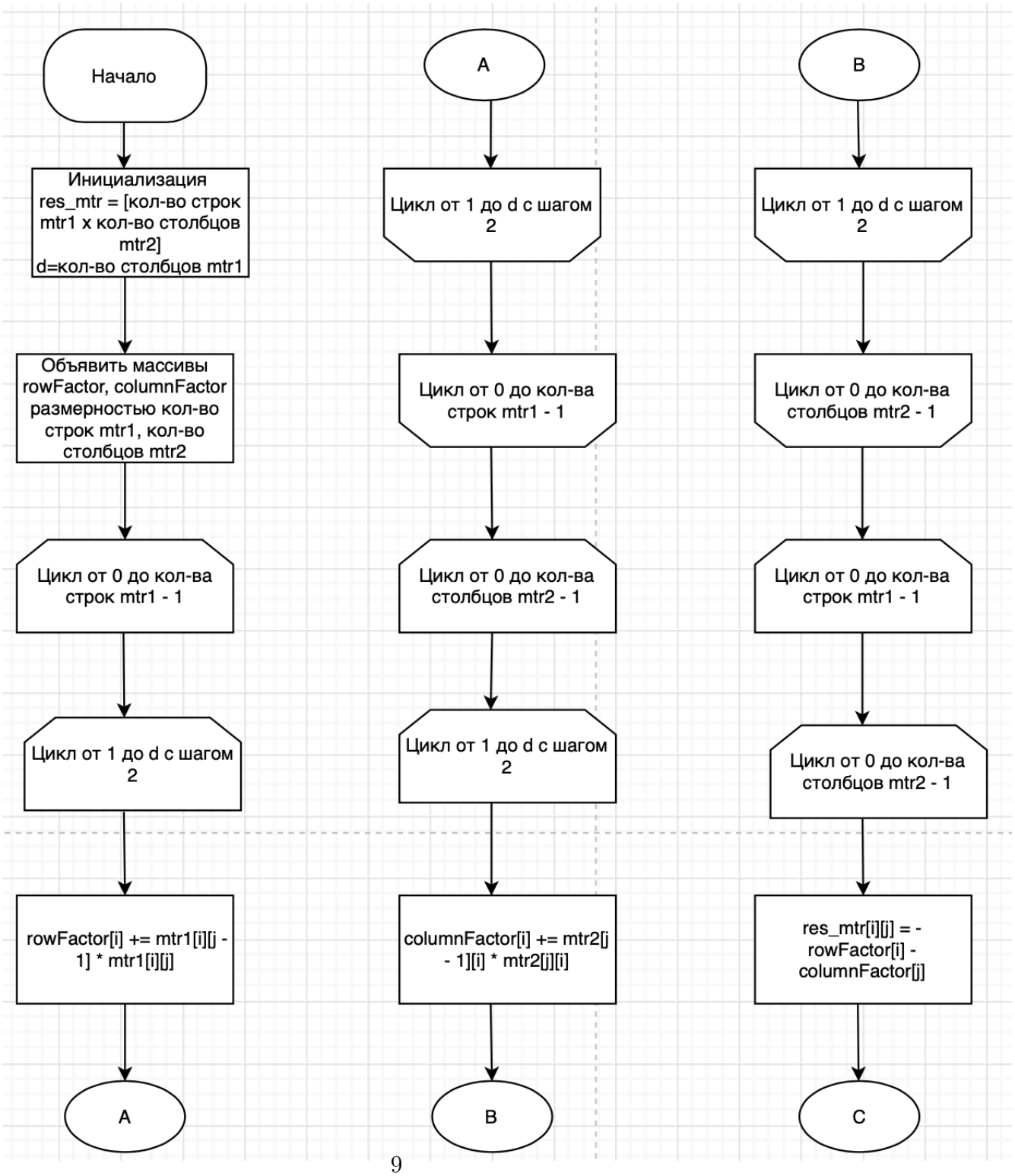

Рис. 2.2: Схема алгоритма Винограда умножения матриц (начало)

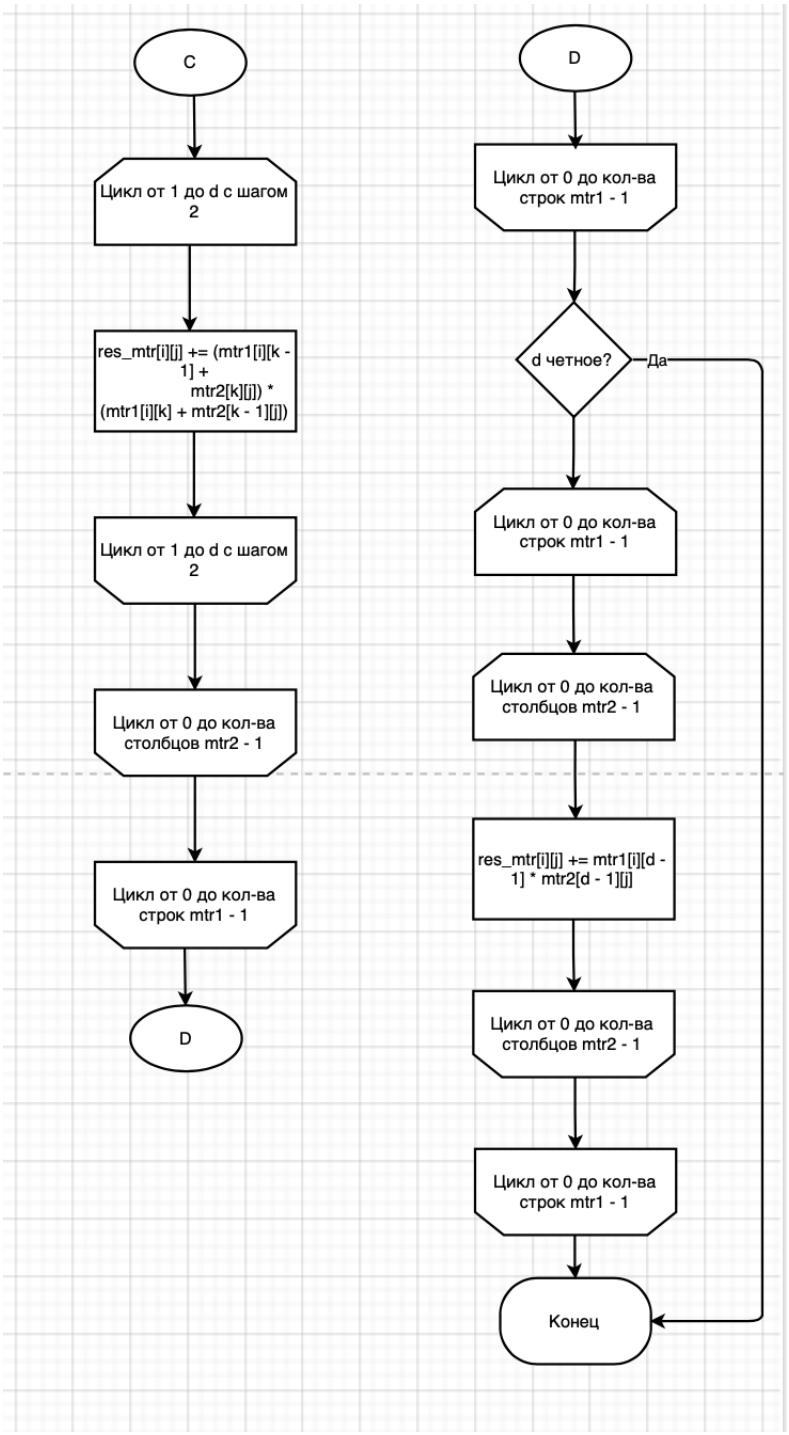

Рис. 2.3: Схема алгоритма Винограда умножения матриц (продолжение)

Рис. 2.4: Схема алгоритма Винограда умножения матриц с оптимизациями (начало)

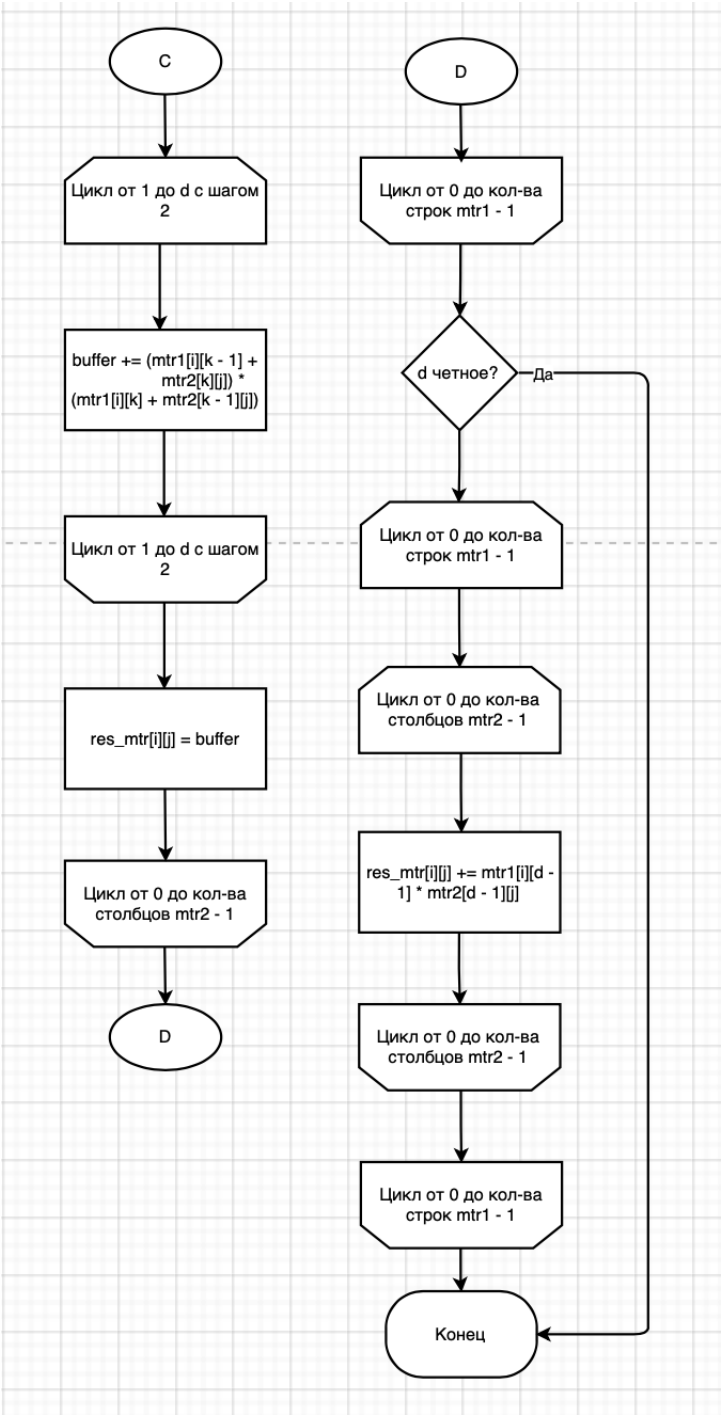

Рис. 2.5: Схема алгоритма Винограда умножения матриц с оптимизациями (продолжение)

3 | Технологическая часть

3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - python (3.8.3) [1] в целях упрощения работы со структурами и визуализацией данных сравнительных анализов, а также из-за наличия опыта работы с данным языком программирования. В качестве интегрированной среды разработки использовался интерпретатор python - IDLE [2]. Для замеров процедурного времени использовалась функция process_time() библиотеки time [3]. Для генерации матрицы использовалась функция randint() модуля random [4].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- matrix.py - главный файл программы;
- time_test.py - файл с замерами временнных характеристик.

Листинг 3.1: Подпрограмма стандартного умножения матриц

```
1 def StandMultMatrix(mtr1, mtr2, n1, m1, n2, m2):
2     if m1 != n2:
3         return False
4     res_mtr = [np.full(m2, 0) for i in range(n1)]
5     for i in range(n1):
6         for j in range(m2):
7             for q in range(m1):
```

```
                res_mtr[i][j] = res_mtr[i][j] + mtr1[i][q]
                    * mtr2[q][j]
    return res_mtr
```

Листинг 3.2: Подпрограмма алгоритма Винограда

```
def WinogradMult(mtr1, mtr2, n1, m1, n2, m2):
    if m1 != n2:
        return False
    res_mtr = [np.full(m2, 0) for i in range(n1)]
    d = m1

    rowFactor = np.full(n1, 0)
    for i in range(n1):
        for j in range(1, d, 2):
            rowFactor[i] += mtr1[i][j - 1] * mtr1[i][j]

    columnFactor = np.full(m2, 0)
    for i in range(m2):
        for j in range(1, d, 2):
            columnFactor[i] += mtr2[j - 1][i] * mtr2[j][i]

    for i in range(n1):
        for j in range(m2):
            res_mtr[i][j] = -rowFactor[i] - columnFactor[j]
            for k in range(1, d, 2):
                res_mtr[i][j] += (mtr1[i][k - 1] +
                mtr2[k][j]) * (mtr1[i][k] + mtr2[k - 1][j])
    if d % 1:
        for i in range(n1):
            for j in range(m2):
                res_mtr[i][j] += mtr1[i][d - 1] * mtr2[d -
                    1][j]

    return res_mtr
```

Листинг 3.3: Подпрограмма алгоритма Винограда с оптимизациями

```
def WinogradOptimization(mtr1, mtr2, n1, m1, n2, m2):
    if m1 != n2:
        return False
```

```
4       res_mtr = [np.full(m2, 0) for i in range(n1)]
5       d = m1
6
7       #Optimization: stop using numpy methods
8       rowFactor = [0 for i in range(n1)]
9       for i in range(n1):
10          for j in range(1, d, 2):
11              rowFactor[i] += mtr1[i][j - 1] * mtr1[i][j]
12
13      columnFactor = [0 for i in range(m2)]
14      for i in range(m2):
15          for j in range(1, d, 2):
16              columnFactor[i] += mtr2[j - 1][i] * mtr2[j][i]
17
18      #Optimization: buffer (stop frequent memory cell access
            request)
19      for i in range(n1):
20          for j in range(m2):
21              buffer = -rowFactor[i] - columnFactor[j]
22              for k in range(1, d, 2):
23                  buffer += (mtr1[i][k - 1] +
24                  mtr2[k][j]) * (mtr1[i][k] + mtr2[k - 1][j])
25              res_mtr[i][j] = buffer
26      #Optimization: bit operation & (and) instead of %
27      if d & 1:
28          for i in range(n1):
29              for j in range(m2):
30                  res_mtr[i][j] += mtr1[i][d - 1] * mtr2[d -
                        1][j]
31
32      return res_mtr
```

Листинг 3.4: Подпрограмма создания матрицы

```
1 def CreateMatrix(n, m):
2     return [[int(j) for j in input().split()] for i in
          range(n)]
```

Листинг 3.5: Подпрограмма создания матрицы с использованием функции randint модуля random [4]

```
def CreateMatrixRandom(n, m):
    return [[randint(0, 100) for j in range(m)] for i in
        range(n)]
```

3.3 Тесты

В данном разделе будет представлена таблица с тестами (таблица 3.1)

Таблица 3.1: Сравнительная таблица времени выполнения всех алгоритмов при четной длине матицы

Матрица 1	Матрица 2	Результат
2 3	3 2	Верный
1 5	5 1	Верный
5 1	1 05	Верный
5 5	5 5	Верный
4 5	3 5	Сообщение об ошибке

Примечание: в сообщении об ошибке указано, что количество строк Матрица1 не совпадает с количеством столбцов Матрица2.

3.4 Вывод

В технологической части были представлены модули программы, листинги кода и тестов к программе, а также обусловлен выбор языка программирования и приведены использовавшиеся в ходе работы инструменты.

4 | Исследовательская часть

4.1 Результат замеров времени выполнения всех алгоритмов

Сравним временные показатели работы каждого из рассматриваемых четырех алгоритмов при длине квадратной матрицы от 20 до 100 чисел. Для этого установим количество итераций (повторений вызова процедуры) iter и воспользуемся библиотекой time для замера времени выполнения каждого алгоритма по iter раз. Возьмем iter = 100, а размерность квадратных матриц равной [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]. Время выполнения алгоритмов произведения матриц довольно мало, поэтому, чтобы оценить его, следует взять среднее арифметическое от времени, за которое процедура отработает установленные iter раз.

На рисунке 4.1 показана работа алгоритмов с матрицами с четным количеством столбцов первой матрицы.

На рисунке 4.2 показана работа алгоритмов с матрицами с нечетным количеством столбцов первой матрицы.

В качестве примера на рисунке 4.1 приведена таблица, представляющая временные характеристики всех алгоритмов при размере квадратной матрицы от 20 до 100 чисел.

4.2 Сравнительный анализ алгоритмов

Введем модель вычислений трудоемкости алгоритма. Пусть трудоемкость 1 у следующих базовых операций: +, -, *, /, =, ==, !=, <, <=, >, >=. Трудоемкость цикла: f цикла = f иниц + f сравн + N итер(f тела + finкрем + fсравн).Трудоемкость условного перехода 1.

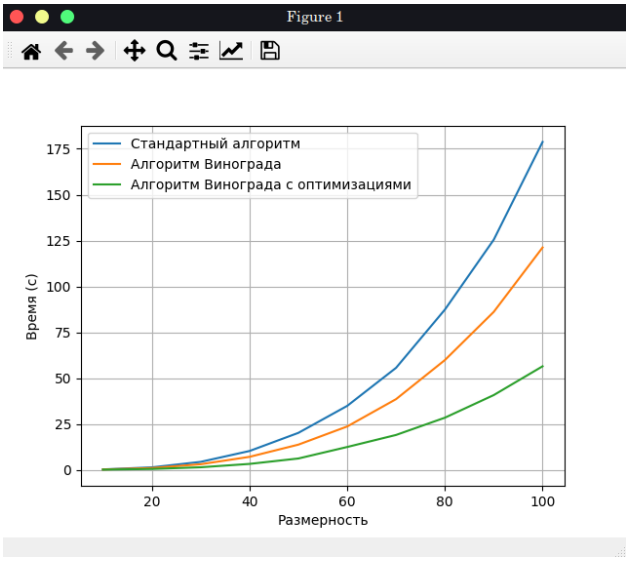

Рис. 4.1: Сравнение времени выполнения алгоритмов

Стандартная реализация алгоритма не эффективна по времени, так как обладает трудоемкостью 13N1M2M1+ 4N1M2+ 4N1 + 2. Оценка трудоемкости данного алгоритма составляет 13N1M2M1. По памяти в стандартном алгоритме умножения матриц требуется m*n памяти под результат.

Теперь рассмотрим алгоритм Винограда умножения матриц. Реализация алгоритма Винограда обладает трудоемкостью 13M1M2 + 7M2 + $\frac{15}{2}$ N1M1 + 7N1 + 13 N1M1M2 + 12N1M2 + 4N1 + 8 + (2, если M1 четное, 15N1M2 + 4N1 + 2, иначе). Оценка трудоемкости данного алгоритма составляет 13N1M2M1 . В алгоритме Винограда умножения матриц требуется дополнительно m+n памяти под результат.

Трудоемкость оптимизированного алгоритма Винограда с с учетом таких оптимизаций, как использование буфера и побитовой операции, будет примерно равна 9N1M1M2.

4.3 Вывод

В данном разделе было произведено сравнение количества затраченного времени вышеизложенных алгоритмов. Самым быстрым оказался опти-

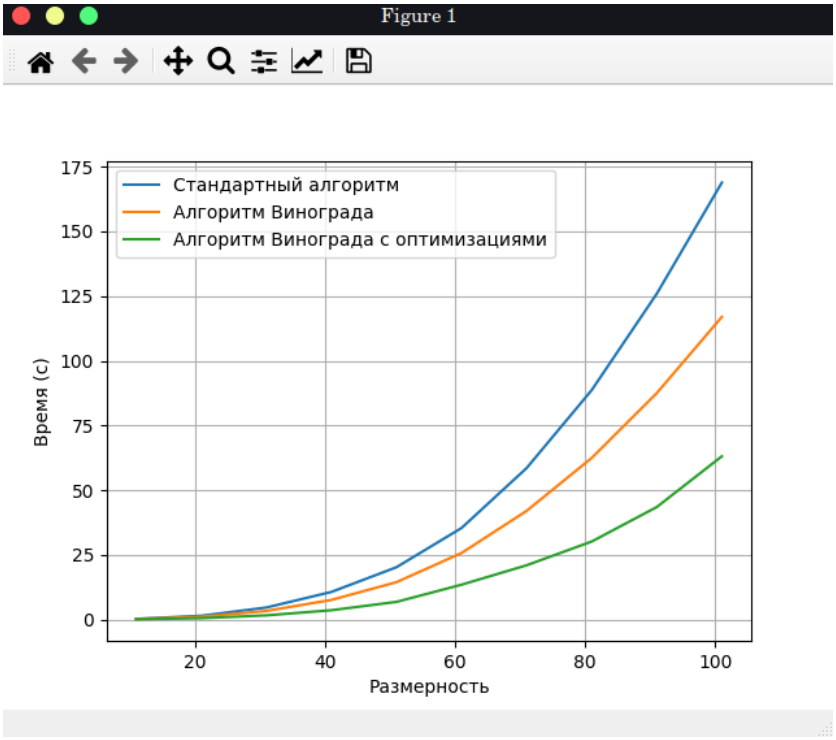

Рис. 4.2: Временные характеристики на нечетных размерах матриц

мизированный алгоритм Винограда. При этом в алгоритме Винограда умножения матриц требуется дополнительно m+n памяти под результат.

Таблица 4.1: Сравнительная таблица времени выполнения (сек.) всех алгоритмов при четной длине матицы

Разм. матриц	Станд.	Виноград	Виноград(опт)
20	0.0000360	0.0000345	0.0000235
30	0.0000751	0.0000617	0.0000386
40	0.0001221	0.0000986	0.0000576
50	0.0001841	0.0001462	0.0000837
60	0.0002648	0.0001988	0.0001119
70	0.0003507	0.0002656	0.0001464
80	0.0004629	0.0003613	0.0001824
90	0.0005592	0.0004178	0.0002259
100	0.0007043	0.0005084	0.0002661

Заключение

В ходе лабораторной работы были реализованы и проанализированы алгоритм стандартного умножения матриц, алгоритм умножения матриц Винограда и алгоритм умножения матриц Винограда с оптимизациями. Было установлено, что благодаря предварительной обработки каждой строки первой матрицы и каждого столбца второй матрицы, результат которой сохранялся в двух дополнительных структурах данных - массивах, алгоритм умножения матриц Винограда значительно экономит время работы программы в сравнении со стандартным алгоритмом. Дополнительные оптимизации этого алгоритма позволили сделать его работу еще более эффективной по времени.

В рамках выполнения работы решены следующие задачи.

1. Изучен и программно реализован стандартный алгоритм умножения матриц.
2. Изучен и программно реализован алгоритм Винограда умножения матриц.
3. Добавлены оптимизации к алгоритму Винограда.
4. Сделан сравнительный анализ по затрачиваемым ресурсам (времени) компьютера на реализацию каждого рассмотренного алгоритма.

Литература

[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] random — Generate pseudo-random numbers. https://docs.python.org/3/library/random.html.