



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»  
(ИУ7)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

## О Т Ч Е Т

по лабораторной работе № 2

Название: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент

ИУ7-54Б

(Группа)

Елгин И.Ю.

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Волкова Л.Л.

(Подпись, дата)

(И.О. Фамилия)

Москва, 2021

# Оглавление

|   |           |
|---|-----------|
| <b>Введение</b>                                     | <b>3</b>  |
| <b>1 Аналитическая часть</b>                        | <b>4</b>  |
| 1.1 Стандартный алгоритм умножения матриц . . . . . | 4         |
| 1.2 Алгоритм Винограда . . . . .                    | 5         |
| 1.3 Вывод . . . . .                                 | 6         |
| <b>2 Конструкторская часть</b>                      | <b>7</b>  |
| 2.1 Требования к программе . . . . .                | 7         |
| 2.2 Схемы алгоритмов . . . . .                      | 7         |
| 2.3 Подсчёт трудоёмкости алгоритмов . . . . .       | 7         |
| 2.4 Вывод . . . . .                                 | 8         |
| <b>3 Технологическая часть</b>                      | <b>14</b> |
| 3.1 Выбор языка программирования . . . . .          | 14        |
| 3.2 Реализации алгоритмов . . . . .                 | 14        |
| 3.3 Оптимизация алгоритма Винограда . . . . .       | 17        |
| 3.4 Оценка затрачиваемого времени . . . . .         | 19        |
| 3.5 Вывод . . . . .                                 | 21        |
| <b>4 Исследовательская часть</b>                    | <b>22</b> |
| 4.1 Результаты экспериментов . . . . .              | 22        |
| 4.2 Вывод . . . . .                                 | 22        |
| <b>Заключение</b>                                   | <b>24</b> |
| <b>Список литературы</b>                            | <b>25</b> |

## Введение

**Умножение матриц** - это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения. Многие реализации прямого и обратного распространения сигнала в сверточных слоях перонной сети базируются на этой операции. Для перемножения двух матриц необходимо, чтобы количество столбцов в первой матрице совпадало с количеством строк во второй. У результирующей матрицы будет столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй.

Сложность вычисления произведения матриц по определению составляет  $O(n^3)$ , однако существуют более эффективные алгоритмы, которые применяются для больших матриц. Вопрос о предельной скорости умножения больших матриц, также как и вопрос о построении наиболее быстрых и устойчивых практических алгоритмов умножения больших матриц остаётся одной из нерешённых проблем линейной алгебры.

# 1. Аналитическая часть

Цель данной лабораторной работы заключается в изучении алгоритмов умножения матриц. Рассматриваются стандартный алгоритм умножения матриц, а также алгоритм Винограда и модифицированный алгоритм Винограда. Требуется рассчитать и изучить затрачиваемое каждым алгоритмом время.

В данной лабораторной работе выделено несколько задач:

- изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда;
- модифицировать алгоритм Винограда;
- дать теоретическую оценку базового алгоритма умножения матриц, алгоритму Винограда и модифицированному алгоритму Винограда;
- реализовать три алгоритма умножения матриц на одном из языков программирования;
- сравнить алгоритмы умножения матриц.

## 1.1. Стандартный алгоритм умножения матриц

Пусть даны две матрицы  $A$  и  $B$  с размерностями  $m \times n$  и  $n \times l$  соответственно (1.1) и (1.2):

$$\begin{bmatrix} a_{1,1} & \dots & a_{1,n} \\ \dots & \dots & \dots \\ a_{m,1} & \dots & a_{m,n} \end{bmatrix} \quad (1.1)$$

$$\begin{bmatrix} b_{1,1} & \dots & b_{1,l} \\ \dots & \dots & \dots \\ b_{n,1} & \dots & b_{n,l} \end{bmatrix} \quad (1.2)$$

В результате умножения, получим матрицу  $C$  размерностью  $m \times l$  (1.3):

$$\begin{bmatrix} c_{1,1} & \dots & c_{1,l} \\ \dots & \dots & \dots \\ c_{m,1} & \dots & c_{m,l} \end{bmatrix} \quad (1.3)$$

$c_{i,j} = \sum_{r=1}^n a_{i,r} \cdot b_{r,j}$  называется произведением матриц  $A$  и  $B$ .

## 1.2. Алгоритм Винограда

Если посмотреть на результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, что такое умножение допускает предварительную обработку, позволяющую часть работы выполнить заранее.

Рассмотрим два вектора  $V = (v_1, v_2, v_3, v_4)$  и  $W = (w_1, w_2, w_3, w_4)$ . Их скалярное произведение (1.4).

$$V \cdot W = v_1 \cdot w_1 + v_2 \cdot w_2 + v_3 \cdot w_3 + v_4 \cdot w_4 \quad (1.4)$$

Это равенство можно переписать в виде (1.5))

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 \cdot v_2 - v_3 \cdot v_4 - w_1 \cdot w_2 - w_3 \cdot w_4 \quad (1.5)$$

Кажется, что формула (1.5) задает больше работы, чем первое: вместо четырех умножений мы насчитываем их шесть, а вместо трех сложений - десять. Менее очевидно, что 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. 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.

### 1.3. Вывод

Алгоритм Винограда предлагает подсчитывать значения заранее перед основными вычислениями, что может повысить производительность при перемножении достаточно больших матриц. Однако с размерами матриц не превосходящих 300, он может справляться хуже чем другие, но алгоритм можно оптимизировать и добиться более высокой скорости подсчёта.

## 2. Конструкторская часть

### 2.1. Требования к программе

Для дальнейшего тестирования программы необходимо обеспечить консольный ввод размерностей двух матриц и их содержимого, а также обеспечить выбор алгоритма поиска. На выходе должны получить результирующую матрицу. Также необходимо реализовать функцию подсчёта процессорного времени, которое могут затрачивать функции.

### 2.2. Схемы алгоритмов

На рисунках 2.1 - 2.5 приведены схемы алгоритмов умножения матриц.

### 2.3. Подсчёт трудоёмкости алгоритмов

Для начала оценки алгоритмов, можно ввести специальную модель трудоёмкости:

- стоимость базовых операций  $1 - +, -, *, /, =, == \dots$ ;
- оценка цикла  $- f_{for} = f_{init} + N \cdot (f + f_{body} + f_{post}) + f$ , где  $f$  - условие цикла,  $f_{init}$  - предусловие цикла,  $f_{post}$  - постусловие цикла;
- стоимость условного перехода примем за 0, стоимость вычисления условия остаётся.

Для стандартного алгоритма умножения с матрицами  $A$  и  $B$  и размерами  $n \times m$  и  $m \times l$  соответственно:

$$f = 2 + n \cdot (2 + 2 + l \cdot (2 + 2 + m \cdot (2 + 6 + 2))) = 10nlm + 4ln + 4n + 2$$

Для алгоритма Винограда при тех же матрицах и их размерах.

Для более понятного подсчёта, можно составить таблицу (таблица 2.1), а затем подсчитать общую трудоёмкость:

$$f = 13mnl + 7.5mn + 7.5lm + 11ln + 8n + 4l + 14 + \begin{cases} 0, \text{ если } m \text{ чётное} \\ 15 \cdot l \cdot n + 4 \cdot n + 2, \text{ иначе} \end{cases}$$

Таблица 2.1: трудоёмкость алгоритма Винограда

| Часть алгоритма            | Трудоёмкость   |
|----------------------------|--|
| Инициализация mulH и mulV  | $2 \cdot 3$  |
| Заполнение mulH            | $2 + n \cdot (2 + 2 + m/2 \cdot (3 + 6 + 6))$                    |
| Заполнение mulV            | $2 + l \cdot (2 + 2 + m/2 \cdot (3 + 6 + 6))$                    |
| Подсчёт результата         | $2 + n \cdot (2 + 2 + l \cdot (2 + 7 + 2 + m/2 \cdot (3 + 23)))$ |
| Условный оператор нечёт. m | 2  |
| Для матриц с нечёт m       | $2 + n \cdot (2 + 2 + l \cdot (2 + 8 + 5))$                      |

**Для оптимизированного алгоритма Винограда** при тех же матрицах и размерах. Для более понятного подсчёта, можно тоже составить таблицу (таблица 2.2).

$$f = 8mnl + 5mn + 5lm + 12ln + 8n + 4l + 18 + \begin{cases} 0, \text{ если } m \text{ чётное} \\ 10 \cdot l \cdot n + 4 \cdot n + 4, \text{ иначе} \end{cases}$$

Таблица 2.2: трудоёмкость оптимизированного алгоритма Винограда

| Часть алгоритма                  | Трудоёмкость   |
|----------------------------------|--|
| Инициализация mulH и mulV        | $2 \cdot 3$  |
| Инициализация m1Mod2<br>и n2Mod2 | $2 \cdot 2$  |
| Заполнение mulH                  | $2 + n \cdot (2 + 2 + m/2 \cdot (2 + 5 + 3))$                        |
| Заполнение mulV                  | $2 + l \cdot (2 + 2 + m/2 \cdot (2 + 5 + 3))$                        |
| Подсчёт результата               | $2 + n \cdot (2 + 2 + l \cdot (2 + 5 + 3 + 2 + m/2 \cdot (2 + 14)))$ |
| Условный оператор нечёт. m       | 2  |
| Для матриц с нечёт m             | $2 + 2 + n \cdot (2 + 2 + l \cdot (2 + 6 + 2))$                      |

## 2.4. Вывод

Были составлены схемы и подсчитана трудоёмкость для каждого алгоритма. Из последнего можно увидеть, что оптимизированный алгоритм Винограда менее трудоёмкий, чем неоптимизированный.



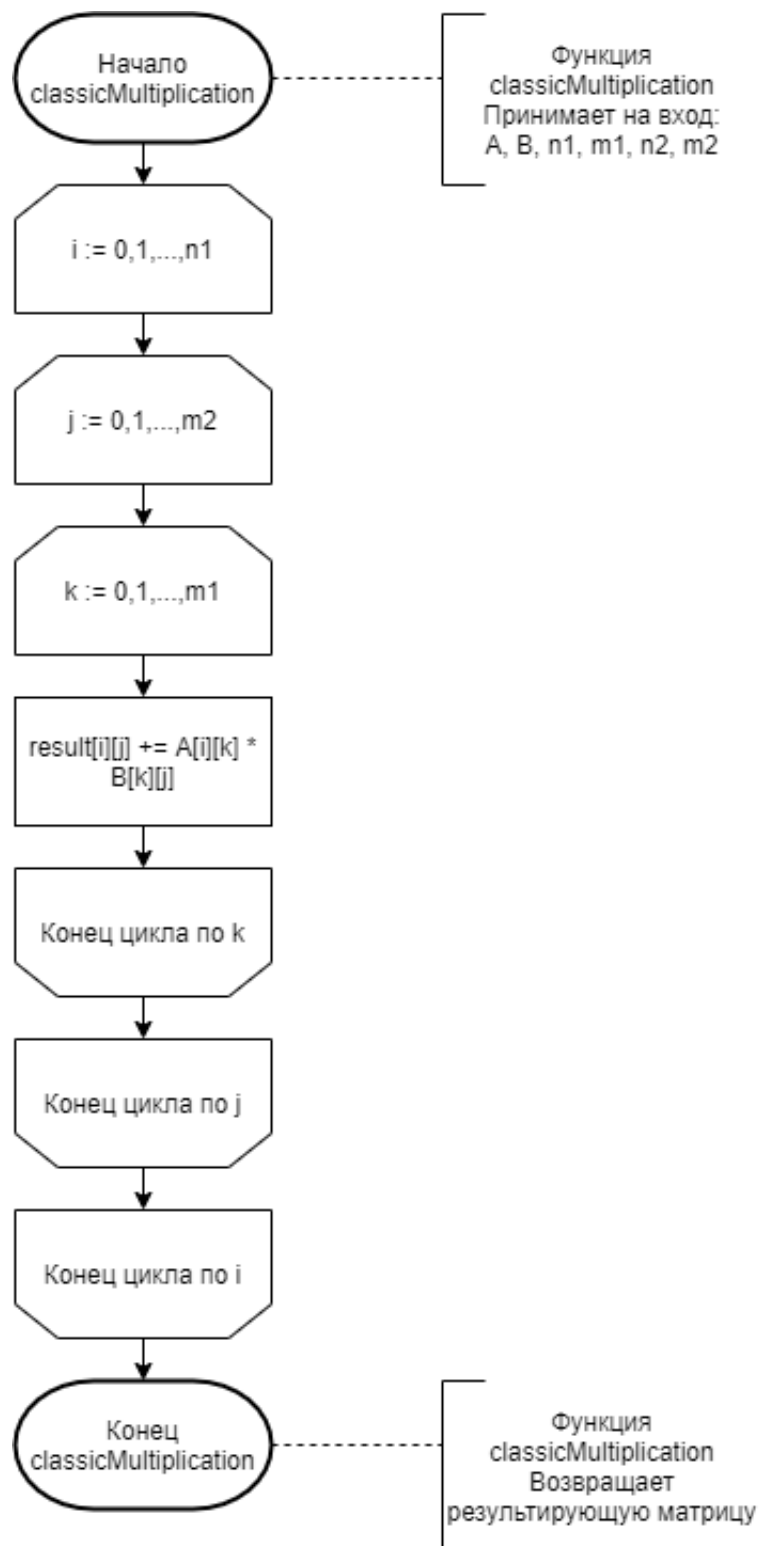


Рис. 2.1: Схема стандартного алгоритма умножения матриц

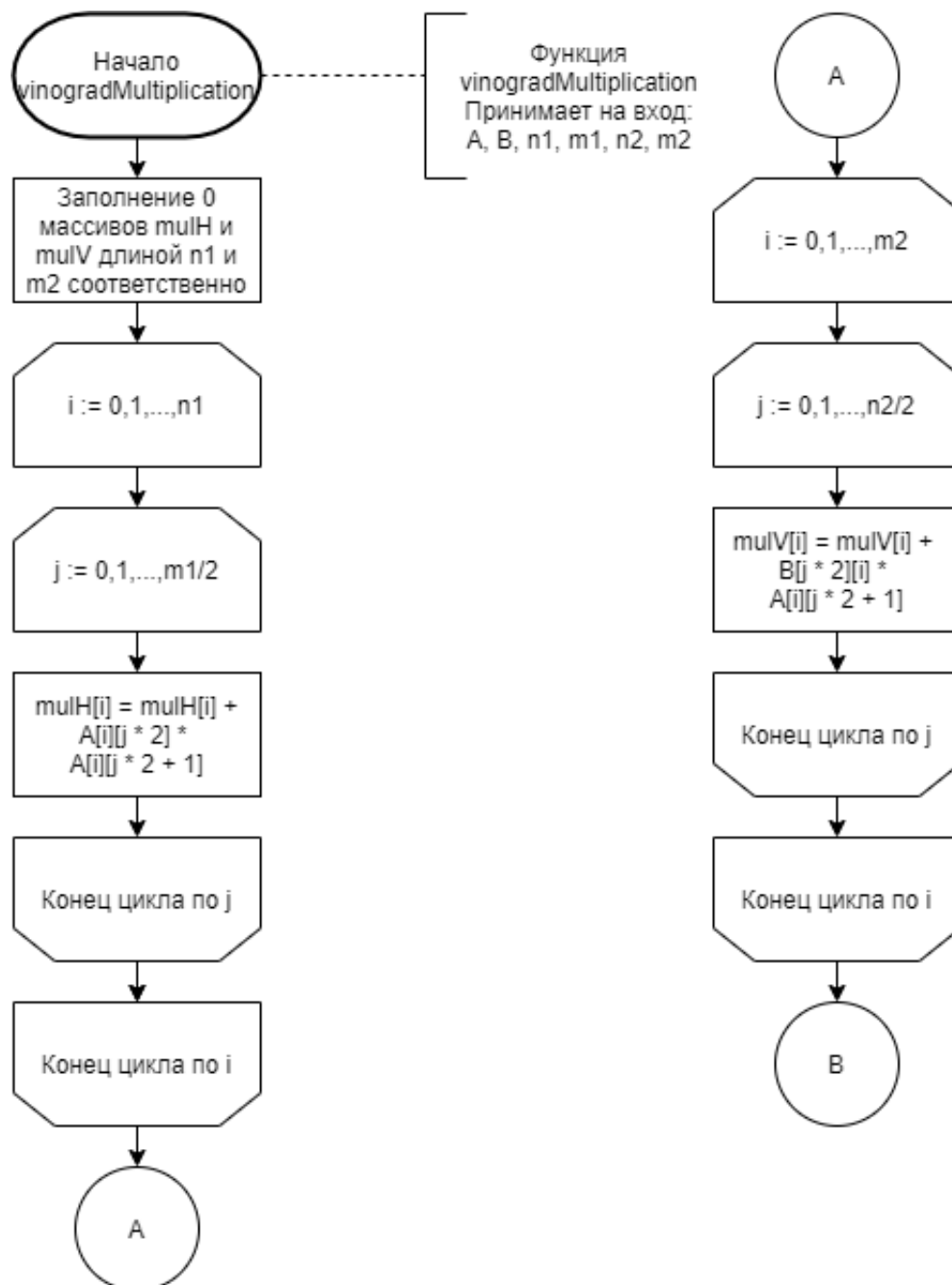


Рис. 2.2: Схема алгоритма Винограда умножения матриц, часть 1

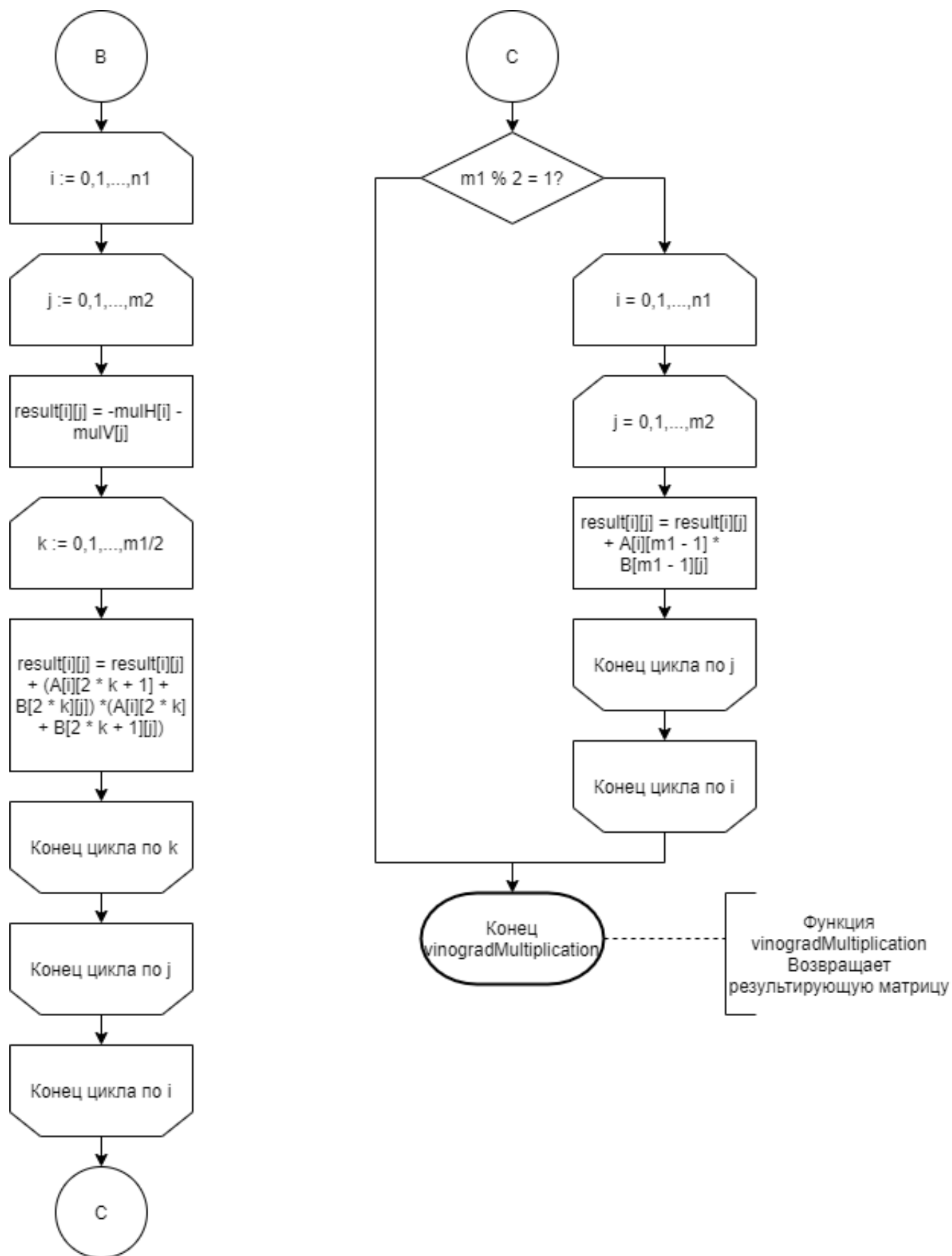


Рис. 2.3: Схема алгоритма Винограда умножения матриц, часть 2

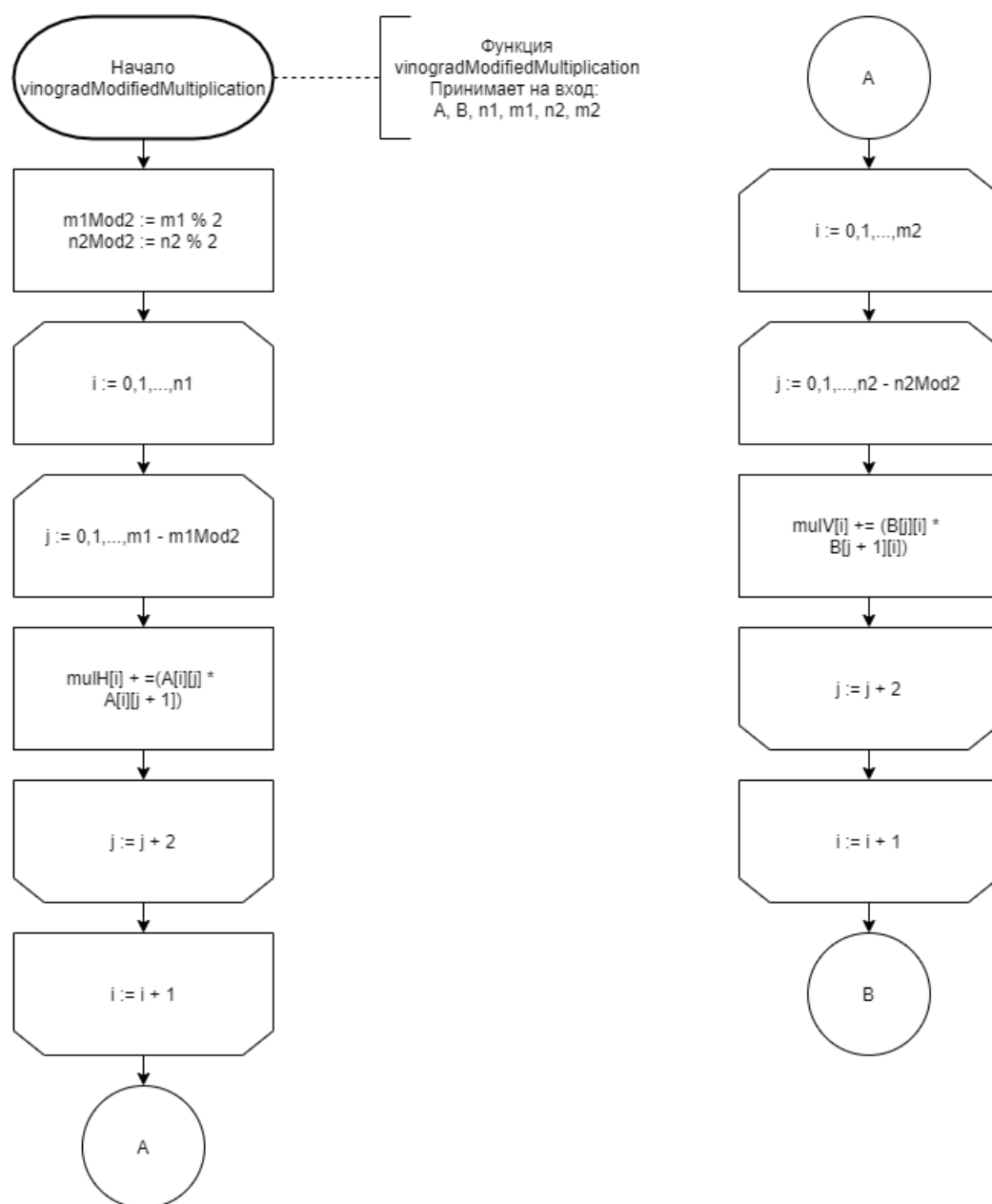


Рис. 2.4: Схема модифицированного алгоритма Винограда умножения матриц, часть 1

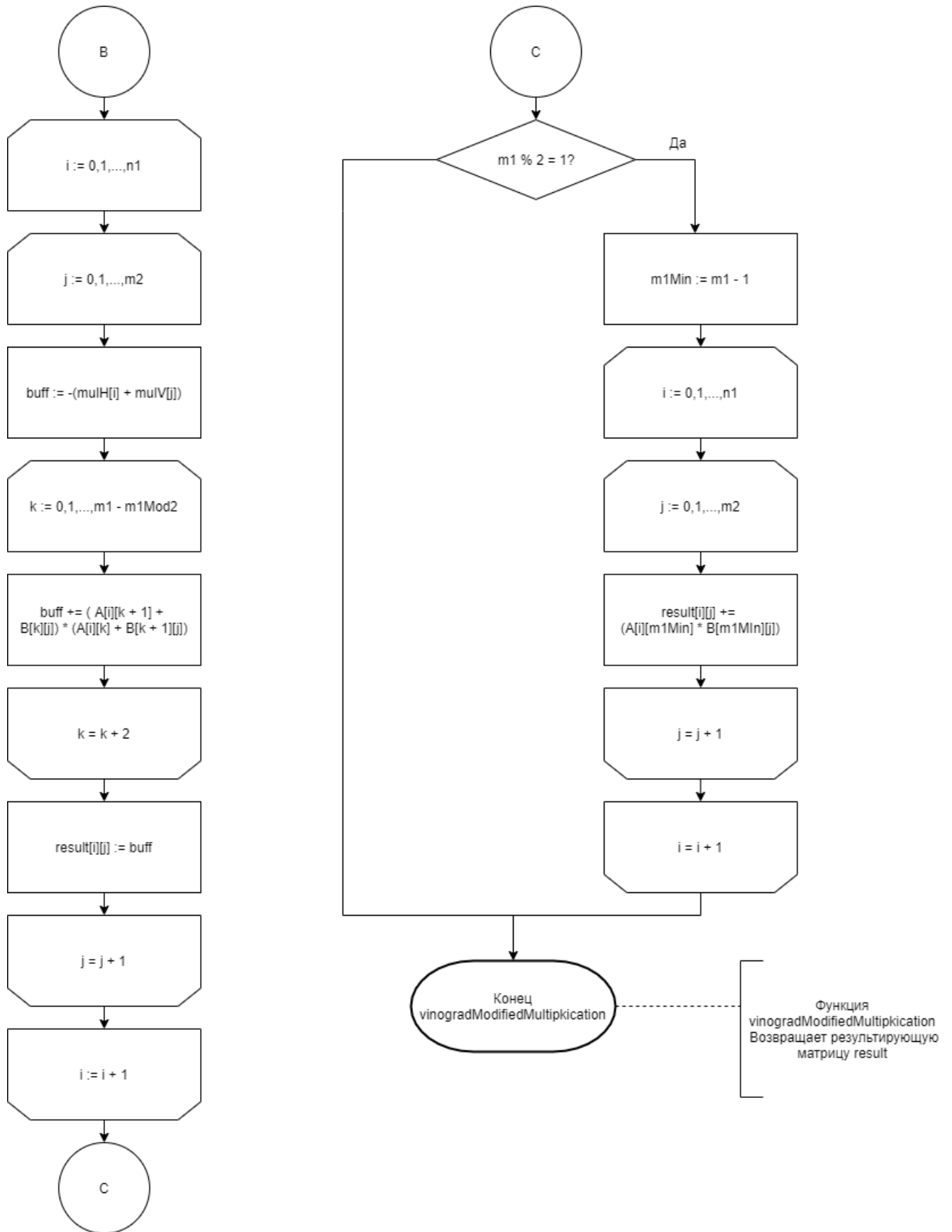


Рис. 2.5: Схема модифицированного алгоритма Винограда умножения матриц, часть 2

## 3. Технологическая часть

### 3.1. Выбор языка программирования

В качестве языка программирования было решено выбрать Python 3, так как уже имеется опыт работы с библиотеками и инструментами языка, которые позволяют реализовать и провести исследования над умножением матриц.

### 3.2. Реализации алгоритмов

В листингах 3.1 - 3.3 приведены реализации алгоритмов умножения на Python.

Листинг 3.1: классический алгоритм умножения матриц

```
1 def classicMultiplication(A, B, isPrint):
2     result = [[0 for j in range(len(A))
3                for i in range(len(B[0]))]
4
5     for i in range(len(A)):
6         for j in range(len(B[0])):
7             for k in range(len(A[i])):
8                 result[i][j] += A[i][k] * B[k][j]
9
10    if (isPrint):
11        print("\n>>> Result with classic method:")
12        printMatrix(result)
```

Листинг 3.2: алгоритм умножения матриц Винограда

```

1 def vinogradMultiplication(A, B, isPrint):
2     n1, m1 = len(A), len(A[0])
3     n2, m2 = len(B), len(B[0])
4     mulH = [0 for _ in range(n1)]
5     mulV = [0 for _ in range(m2)]
6     result = [[0 for j in range(n1)] for i in range(m2)]
7
8     for i in range(n1):
9         for j in range(int(m1 / 2)):
10             mulH[i] = mulH[i] + A[i][j * 2] *
11                         A[i][j * 2 + 1]
12
13     for i in range(m2):
14         for j in range(int(n2 / 2)):
15             mulV[i] = mulV[i] + B[j * 2][i] *
16                         B[j * 2 + 1][i]
17
18     for i in range(n1):
19         for j in range(m2):
20             result[i][j] = -mulH[i] - mulV[j]
21
22             for k in range(int(m1 / 2)):
23                 result[i][j] = result[i][j] +
24                     ((A[i][2 * k + 1] + B[2 * k][j]) *
25                     (A[i][2 * k] + B[2 * k + 1][j]))
26     if (m1 % 2):
27         for i in range(n1):
28             for j in range(m2):
29                 result[i][j] = result[i][j] +
30                     (A[i][m1 - 1] * B[m1 - 1][j])
31     if (isPrint):
32         print("\n>>> Result with vinograd method:")
33         printMatrix(result)

```

Листинг 3.3: оптимизированный алгоритм Винограда

```

1 def vinogradMultiplicationModified(A, B, isPrint):
2     n1, m1 = len(A), len(A[0])
3     n2, m2 = len(B), len(B[0])
4     mulH = [0 for _ in range(n1)]
5     mulV = [0 for _ in range(m2)]
6     result = [[0 for j in range(n1)] for i in range(m2)]
7     m1Mod2 = m1 % 2
8     n2Mod2 = n2 % 2
9
10    for i in range(n1):
11        for j in range(0, m1 - m1Mod2, 2):
12            mulH[i] += A[i][j] * A[i][j + 1]
13
14    for i in range(m2):
15        for j in range(0, n2 - n2Mod2, 2):
16            mulV[i] += B[j][i] * B[j + 1][i]
17
18    for i in range(n1):
19        for j in range(m2):
20            buff = -(mulH[i] + mulV[j])
21
22            for k in range(0, m1 - m1Mod2, 2):
23                buff += ((A[i][k + 1] + B[k][j]) *
24                        (A[i][k] + B[k + 1][j]))
25            result[i][j] = buff
26    if m1Mod2:
27        m1Min = m1 - 1
28        for i in range(n1):
29            for j in range(m2):
30                result[i][j] += A[i][m1Min] * B[m1Min][j]
31    if isPrint:
32        print("\nResult with modified vinograd method:")
33        printMatrix(result)

```



### 3.3. Оптимизация алгоритма Винограда

Для оптимизации алгоритма были внесены некоторые изменения.

Заранее высчитываются значения, для избавления от деления в цикле.

См. листинг 3.4.

Листинг 3.4: избавление от деления в цикле

```
1 def vinogradMultiplicationModified(A, B, isPrint):
2     ...
3
4     m1Mod2 = m1 % 2
5     n2Mod2 = n2 % 2
6
7     for i in range(n1):
8         for j in range(0, m1 - m1Mod2, 2):
9             mulH[i] += A[i][j] * A[i][j + 1]
10
11    for i in range(m2):
12        for j in range(0, n2 - n2Mod2, 2):
13            mulV[i] += B[j][i] * B[j + 1][i]
14
15    ...
```

Заменяется обычное сложение ( $\text{mulV}[i] = \text{mulV}[i] + \dots$ ) на более оптимизированный вариант ( $\text{mulV}[i] += \dots$ , аналогично и в других подобных местах). См. листинг 3.5.

Листинг 3.5: оптимизация сложения

```

1 def vinogradMultiplicationModified(A, B, isPrint):
2     ...
3
4     for i in range(n1):
5         for j in range(0, m1 - m1Mod2, 2):
6             mulH[i] += A[i][j] * A[i][j + 1]
7
8
9     for i in range(m2):
10        for j in range(0, n2 - n2Mod2, 2):
11            mulV[i] += B[j][i] * B[j + 1][i]
12
13    for i in range(n1):
14        for j in range(m2):
15            buff = -(mulH[i] + mulV[j])
16
17            for k in range(0, m1 - m1Mod2, 2):
18                buff += ((A[i][k + 1] + B[k][j]) *
19                        (A[i][k] + B[k + 1][j]))
20
21            result[i][j] = buff
22
23    if m1Mod2:
24        m1Min = m1 - 1
25
26        for i in range(n1):
27            for j in range(m2):
28                result[i][j] += A[i][m1Min] * B[m1Min][j]
29
30    ...

```

Инициализируется специальный буфер для накопления результата умножения, который позже сбрасывается в ячейку матрицы. См. листинг 3.6.

Листинг 3.6: оптимизация, при помощи буфера

```

1 def vinogradMultiplicationModified(A, B, isPrint):
2     ...
3
4     for i in range(n1):
5         for j in range(m2):
6             buff = -(mulH[i] + mulV[j])
7
8             for k in range(0, m1 - m1Mod2, 2):
9                 buff += ((A[i][k + 1] + B[k][j]) *
10                        (A[i][k] + B[k + 1][j]))
11
12             result[i][j] = buff
13
14     ...

```

### 3.4. Оценка затрачиваемого времени

Для замера процессорного времени выполнения алгоритмов используется библиотека `time` [2]. В листинге 3.7 приведена функция наполнения случайными числами матрицы с заданным размером. В листинге 3.8 приведены функции с помощью которых производятся замеры времени.

Листинг 3.7: наполнение матрицы случайными числами

```

1 def generateMatrix(size):
2     return [[random.randint(0, 9) for _ in range(size)]
3             for _ in range(size)]

```

Листинг 3.8: функции для замера времени

```
1 def doTimeTest(method, A, B):
2     t1 = process_time()
3     method(A, B, False)
4     t2 = process_time()
5     return t2 - t1
6
7 def multiplicationTimeTest():
8     sizesMod2 = [100, 150, 200]
9     sizesNotMod2 = [101, 151, 201]
10    for size in sizesMod2:
11        A = generateMatrix(size)
12        B = generateMatrix(size)
13        print(">>> For classic with          len = ", size,
14              "Time:", doTimeTest(classicMultiplication, A, B))
15        print(">>> For Vinograd with          len = ", size,
16              "Time:", doTimeTest(vinogradMultiplication, A, B))
17        print(">>> For Vinograd mod. with len = ", size,
18              "Time:", doTimeTest(vinogradMultiplicationModified,
19                                  A, B))
20
21    for size in sizesNotMod2:
22        A = generateMatrix(size)
23        B = generateMatrix(size)
24        print(">>> For classic with          len = ", size,
25              "Time:", doTimeTest(classicMultiplication, A, B))
26        print(">>> For Vinograd with          len = ", size,
27              "Time:", doTimeTest(vinogradMultiplication, A, B))
28        print(">>> For Vinograd mod. with len = ", size,
29              "Time:", doTimeTest(vinogradMultiplicationModified,
30                                  A, B))
```

### 3.5. Вывод

Были реализованы функции алгоритмов перемножения матриц на языке Python 3, а также функции тестирования и подсчёта процессорного времени.

## 4. Исследовательская часть

Измерения процессорного времени проводятся при одинаковых размерах матриц и при этом тестируются как чётные размеры, так и нечётные: 100, 101, 150, 151, 200, 201.

### 4.1. Результаты экспериментов

Проведя измерения процессорного времени выполнения реализованных алгоритмов, можно составить для чётных размеров таблицу 4.1 и для нечётных таблицу 4.2.

Таблица 4.1: Результаты замеров процессорного времени в секундах, для чётных размеров

| Название метода \ Размер        | 100   | 150   | 200   |
|---------------------------------|-------|-------|-------|
| Стандартный                     | 0.421 | 1.578 | 3.406 |
| алгоритм Винограда              | 0.453 | 1.563 | 3.625 |
| оптимизированный алг. Винограда | 0.328 | 0.969 | 2.422 |

Таблица 4.2: Результаты замеров процессорного времени в секундах, для нечётных размеров

| Название метода \ Размер        | 101   | 151   | 201   |
|---------------------------------|-------|-------|-------|
| Стандартный                     | 0.391 | 1.469 | 3.422 |
| алгоритм Винограда              | 0.516 | 1.813 | 3.813 |
| оптимизированный алг. Винограда | 0.313 | 1.125 | 2.766 |

### 4.2. Вывод

Анализируя результаты замеров затрачиваемого времени и тестируя алгоритмы на разных размерах, можно сказать, что стандартный алгоритм умножения матриц выигрывает по времени у остальных, на размерах менее 300, однако его главным преимуществом является стабильность, то есть алгоритм не зависит от чётности размера матрицы.

Также стоит заметить, что модифицированный алгоритм Винограда показывает наилучшее время на средних размерах, в то время, как неоптимизированный алгоритм Винограда показывает наихудшее. В неоптимизированном алгоритме Винограда приходится неоднократно вычислять одни и те же значения, что может замедлять его.

## Заключение

В ходе работы были изучены алгоритмы умножения матриц. Реализованы 3 алгоритма, приведен программный код реализации алгоритмов по умножению матриц. Была подсчитана трудоемкость каждого из алгоритмов. А также было проведено сравнение алгоритмов по времени и трудоемкости. Показано, что наименее трудоемкий и наименее затратный по времени при больших размерах матриц алгоритм Винограда оптимизированный.

Цель работы достигнута, решены поставленные задачи. Получены практические навыки реализации алгоритмов Винограда и стандартного алгоритма, а также проведена исследовательская работа по оптимизации и вычислении трудоемкости алгоритмов.



## Список литературы

- [1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. – М.: Техносфера, 2017. – 267с.
- [2] Документация на официальном сайте Python про библиотеку time [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html> (дата обращения 11.10.2021)
- [3] Умножение матриц [Электронный ресурс]. Режим доступа: <http://algotlib.narod.ru/Math/Matrix.html> (дата обращения 10.10.2021)
- [4] Умножение матриц [Электронный ресурс]. Режим доступа: <http://algotlib.narod.ru/Math/Matrix.html> (дата обращения 10.10.2021)