

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Москва, 2021

Содержание

Введение	3
1 Аналитический раздел	4
1.1 Определение матрицы	4
1.2 Стандартный алгоритм умножения матриц	4
1.3 Алгоритм умножения матриц по Винограду	4
1.4 Вывод	5
2 Конструкторский раздел	6
2.1 Модель оценки трудоемкости алгоритмов	6
2.2 Вычисление трудоемкости алгоритмов	7
2.2.1 Трудоемкость алгоритма умножения матриц по Винограду	7
2.2.2 Трудоемкость алгоритма умножения матриц по Винограду	7
2.2.3 Трудоемкость оптимизированного алгоритма умножения матриц по Винограду	9
2.3 Схемы алгоритмов умножения матриц	10
2.4 Вывод	19
3 Технологический раздел	20
3.1 Требования к программному обеспечению	20
3.2 Выбор средств реализации	20
3.3 Листинги программ	20
3.4 Вспомогательные функции	23
3.5 Тестирование	24

Введение

Матричное умножение лежит в основе нейронных сетей. Большинство операций при обучении нейронной сети требуют некоторой формы умножения матриц. Для этого требуется высокая скорость вычислений.

Стандартный алгоритм умножения матриц на больших данных, исчисляемых миллиардами, выполняет вычисления не самым быстрым способом. Существуют различные оптимизации. Одной из них является алгоритм Винограда, который позволяет сократить время вычислений. На практике алгоритм Копперсмита—Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для тех матриц, размер которых превышает память современных компьютеров.

Целью лабораторной работы является изучение и реализация алгоритмов умножения матриц. Для её достижения необходимо выполнить следующие задачи:

- изучить алгоритм умножения матриц стандартным способом и по Винограду;
- разработать данные алгоритмы;
- оптимизировать алгоритм Винограда;
- выполнить тестирование реализации алгоритмов методом черного ящика;
- провести сравнительный анализ этих алгоритмов по затратам памяти и процессорному выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

1.1 Определение матрицы

Матрицей размера $m \times n$ называется прямоугольная таблица элементов некоторого множества (например, чисел или функций), имеющая m строк и n столбцов. Элементы a_{ij} , из которых составлена матрица, называются элементами матрицы. Условимся, что первый индекс i элемента a_{ij} соответствует номеру строки, второй индекс j – номеру столбца, в котором расположен элемент a_{ij} . Матрица может быть записана по формуле (1.1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (1.1)$$

1.2 Стандартный алгоритм умножения матриц

Произведением матрицы $A = (a_{ij})$, имеющей m строк и n столбцов, на матрицу $B = (b_{ij})$, имеющую n строк и p столбцов, называется матрица $C = (c_{ij})$, имеющая m строк и p столбцов, у которой элемент $C = (c_{ij})$ определяется по формуле (1.2).

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

1.3 Алгоритм умножения матриц по Винограду

Обозначим i строку матрицы A как \bar{u} , j столбец матрицы B как \bar{v} . Тогда элемент c_{ij} определяется по формуле (1.3).

$$c_{ij} = \bar{u} \times \bar{v} = \begin{pmatrix} u_1 & u_2 & u_3 & u_4 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix} = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4 \quad (1.3)$$

Эту формулу можно представить в следующем виде (1.4).

$$(u_1 + v_2)(u_2 + v_1) + (u_3 + v_4)(u_4 + v_3) - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4 \quad (1.4)$$

На первый взгляд может показаться, что выражение (1.4) задает больше работы, чем первое: вместо четырех умножений насчитывается их шесть, а вместо трех сложений - десять. Выражение в правой части формулы можно вычислить заранее и затем повторно использовать. На практике это означает, что над предварительно обработанными элементами придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

1.4 Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц стандартным образом и по Винограду. Были приведены соответствующие математические расчеты.

2 Конструкторский раздел

2.1 Модель оценки трудоемкости алгоритмов

Введем модель оценки трудоемкости.

1. Трудоемкость базовых операций.

Пусть трудоемкость следующих операций равной 2:

$$*, /, //, \%, *, =, / =$$

Примем трудоемкость следующих операций равной 1:

$$=, +, -, + =, - =, ==, !=, <, >, \leq, \geq, |, \&\&, , ||, []$$

2. Трудоемкость цикла.

Пусть трудоемкость цикла определяется по формуле (2.1).

$$f = f_{init} + f_{comp} + N_{iter} * (f_{in} + f_{inc} + f_{comp}) \quad (2.1)$$

где:

- f_{init} - трудоемкость инициализации переменной-счетчика;
- f_{comp} - трудоемкость сравнения;
- N_{iter} - номер выполняемой итерации;
- f_{in} - трудоемкость команд из тела цикла;
- f_{inc} - трудоемкость инкремента;
- f_{comp} - трудоемкость сравнения.

3. Трудоемкость условного оператора.

Пусть трудоемкость самого условного перехода равна 0, но она

определяется по формуле (2.2).

$$f_{if} = f_{comp_if} + \begin{bmatrix} f_a \\ f_b \end{bmatrix} \quad (2.2)$$

2.2 Вычисление трудоемкости алгоритмов

Пусть во всех дальнейших вычислениях размер матрицы A имеет $M \times N$, для матрицы B - $N \times Q$.

2.2.1 Трудоемкость алгоритма умножения матриц по Винограду

Трудоемкость этого алгоритма вычисляется по формуле (2.3).

$$\begin{aligned} f &= \underbrace{1}_{=} + \underbrace{1}_{<} + M(\underbrace{2}_{++} + \underbrace{1}_{=} + \underbrace{1}_{<} + Q(\underbrace{2}_{++} + \underbrace{1}_{=} + \underbrace{1}_{<} + N(\underbrace{2}_{++} + \underbrace{8}_{[]} + \underbrace{2}_{*} + \underbrace{1}_{=} + \underbrace{1}_{+}))) = \\ &= 14MNQ + 4MQ + 4M + 2 \quad (2.3) \end{aligned}$$

2.2.2 Трудоемкость алгоритма умножения матриц по Винограду

Трудоемкость этого алгоритма состоит из следующих компонентов, определяемых по формуле (2.4).

$$f_{vin} = f_{init} + f_{precomp} + f_{fill} + f_{even} \quad (2.4)$$

где:

- f_{init} - трудоемкость инициализации массивов для предварительного вычисления (2.5);

$$\begin{aligned} f_{init} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{[]}{1} + \underset{=}{1}) + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[]}{1} + \underset{=}{1}) = \\ &= 2 + 4M + 2 + 4Q = 4 + 4M + 4Q \quad (2.5) \end{aligned}$$

- $f_{precomp}$ - трудоемкость предварительного заполнения строк матрицы A и столбцов матрицы B (2.6);

$$\begin{aligned} f_{precomp} = f_{rows} + f_{columns} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + \frac{n}{2}(\underset{++}{2} + \underset{[]}{6} + \underset{=}{1} + \underset{+}{2} + \underset{*}{6})) + \\ &+ \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + \frac{N}{2}(\underset{++}{2} + \underset{[]}{6} + \underset{=}{1} + \underset{+}{2} + \underset{*}{6})) = \\ &= 2 + M(4 + \frac{N}{2} * 17) + 2 + Q(4 + \frac{N}{2} * 17) = \\ &= 4 + 4M + 4Q + \frac{17NM}{2} + \frac{17NQ}{2} \quad (2.6) \end{aligned}$$

- f_{even} - трудоемкость заполнения результирующей матрицы (2.7);

$$\begin{aligned} f_{fill} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[]}{4} + \underset{=}{1} + \underset{-}{2} + \underset{=}{1} + \underset{<}{1} + \\ &+ \frac{N}{2}(\underset{++}{2} + \underset{[]}{12} + \underset{=}{1} + \underset{+}{5} + \underset{*}{10} + \underset{/}{2})) = 2 + M(4 + Q(11 + 16N)) = \\ &= 2 + 4M + 11MQ + 16MNQ \quad (2.7) \end{aligned}$$

- f_{fill} - трудоемкость для дополнения умножения в случае нечетной

размерности матрицы. (2.8).

$$f_{fill} = \underset{\%}{2} + \underset{==}{1} + \begin{bmatrix} \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{[]}{8} + \\ + \underset{=}{1} + \underset{+}{1} + \underset{-}{2})), \\ 0 \end{bmatrix} =$$

$$= 3 + \begin{bmatrix} 2 + 4M + 14MQ, \\ 0 \end{bmatrix} \quad (2.8)$$

Результирующая трудоемкость алгоритма Винограда составляет $f_{vin} \approx 16MNQ$

2.2.3 Трудоемкость оптимизированного алгоритма умножения матриц по Винограду

Трудоемкость этого алгоритма определяется из следующих компонентов по формуле (2.9).

$$f_{optim} = f_{init} + f_{precomp} + f_{fill} \quad (2.9)$$

где:

- f_{init} - определяется по формуле (2.5) в добавок с другими компонентами (2.10);

$$f_{init} = 4 + 4M + 4Q + \underset{=}{2} + \underset{\%}{2} + \underset{-}{1} + \underset{==}{1} + \begin{bmatrix} \underset{=}{1}, \\ 0 \end{bmatrix} = 7 + 4M + 4Q + \begin{bmatrix} \underset{=}{1}, \\ 0 \end{bmatrix} \quad (2.10)$$

- $f_{precomp}$ - трудоемкость предварительного заполнения строк матри-

цы А и столбцов матрицы В (2.11);

$$\begin{aligned}
f_{precomp} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + N(\underset{+=}{2} + \underset{*}{2} + \underset{+}{1} + \underset{[]}{4}) + \underset{[]}{1} + \underset{=}{1}) + \\
&\quad + \underset{=}{1} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + N(\underset{+=}{2} + \underset{[]}{4} + \underset{*}{2} + \underset{+}{1}) + \underset{=}{1}) = \\
&= 2 + 7M + 9MN + 2 + 6Q + 9NQ = 9MN + 9NQ + 7M + 6Q + 4
\end{aligned} \tag{2.11}$$

- f_{fill} - трудоемкость для заполнения матрицы (2.12).

$$\begin{aligned}
f_{fill} &= \underset{=}{1} + \underset{<}{1} + M(\underset{++}{2} + \underset{=}{2} + \underset{<}{1} + Q(\underset{++}{2} + \underset{=}{2} + \underset{-}{1} + \underset{+}{1} + \underset{[]}{2} + \underset{<}{1} + \frac{N}{2}(\underset{+=}{2} + \underset{[]}{8} + \underset{+}{4} + \underset{*}{2}) + \underset{=}{1} + \\
&\quad + \left[\begin{array}{c} \underset{+=}{1} + \underset{[]}{4} + \underset{*}{2} \\ 0 \end{array} + \underset{[]}{2} + \underset{=}{1} \right)) = \\
&= 2 + M(4 + Q(\frac{16}{2} + 13 + \left[\begin{array}{c} 6 \\ 0 \end{array} \right])) = \\
&\quad 8MNQ + 14MQ + 4M + 2 + \left[\begin{array}{c} 6 \\ 0 \end{array} \right] MQ \tag{2.12}
\end{aligned}$$

Результирующая трудоемкость оптимизированного алгоритма Винограда для лучшего и худшего случая составляет (2.13).

$$f_{fill} \approx 8MNQ \tag{2.13}$$

2.3 Схемы алгоритмов умножения матриц

Ниже представлены следующие схемы алгоритмов:

- рис. 2.1 - схема алгоритма стандартного умножения матриц;

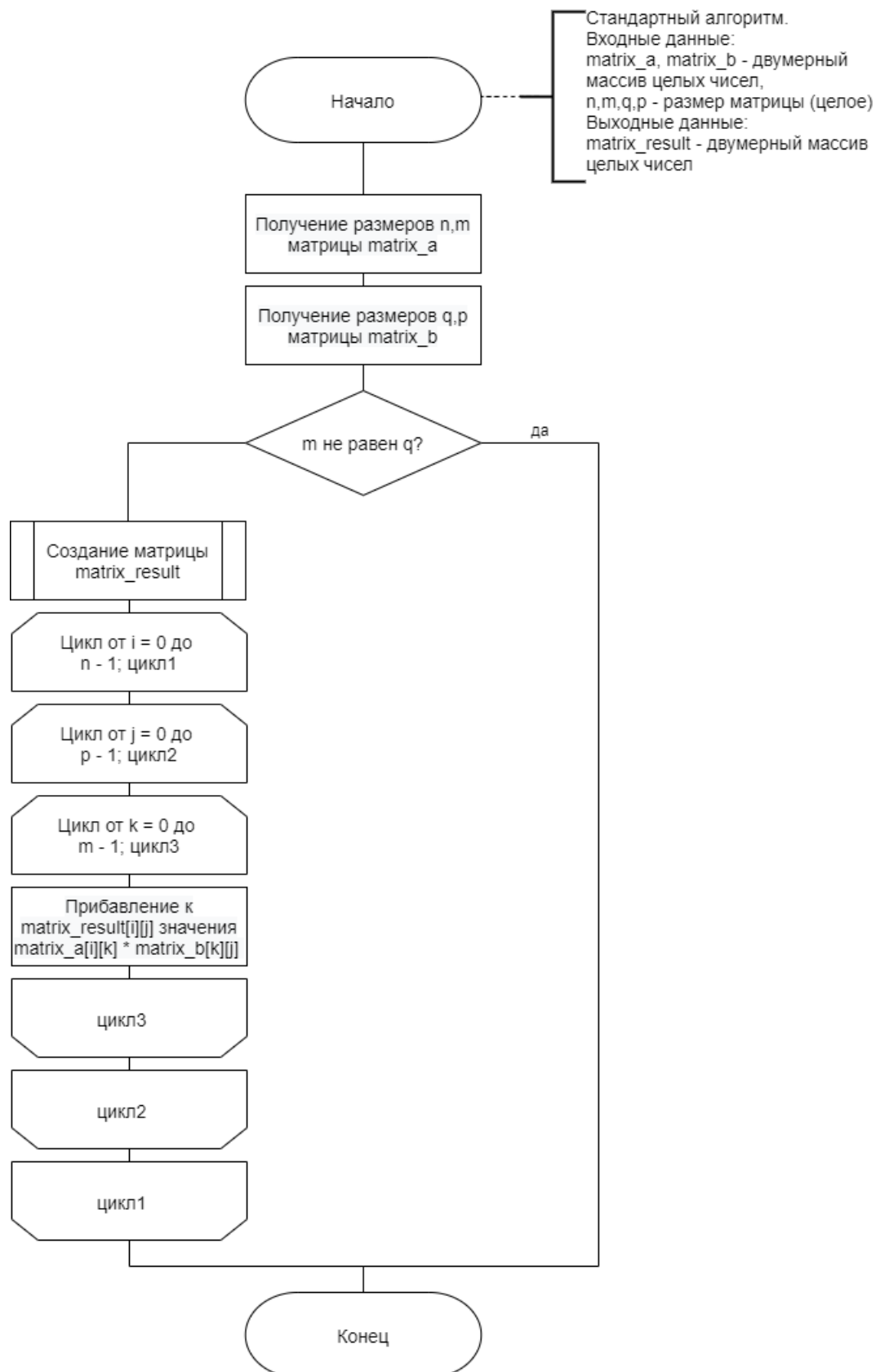


Рисунок 2.1 – Схема алгоритма стандартного умножения матриц.

- рис. 2.2, 2.3, 2.4 - схема алгоритма умножения матриц по Винограду;

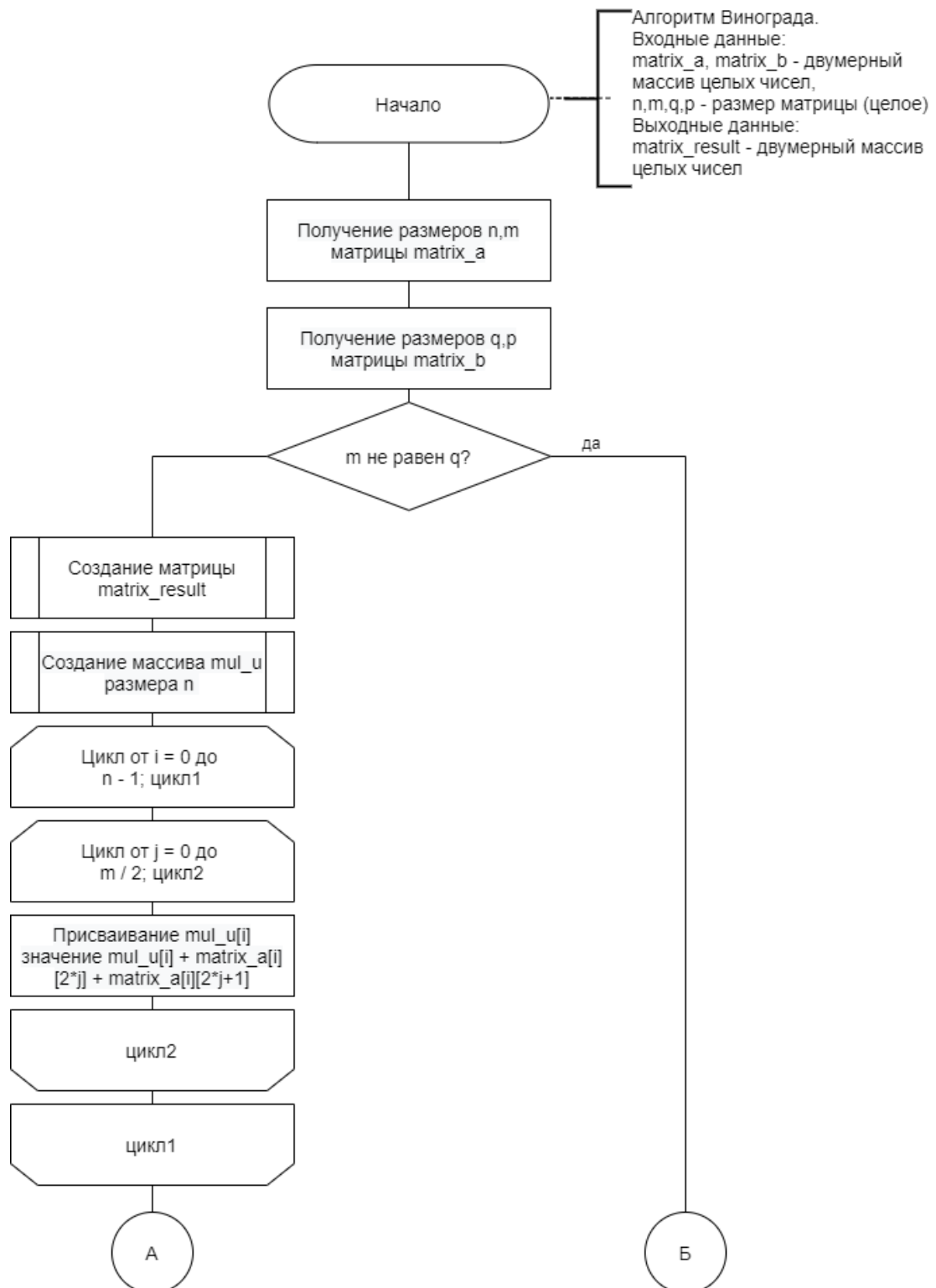


Рисунок 2.2 – Схема алгоритма умножения матриц по Винограду.

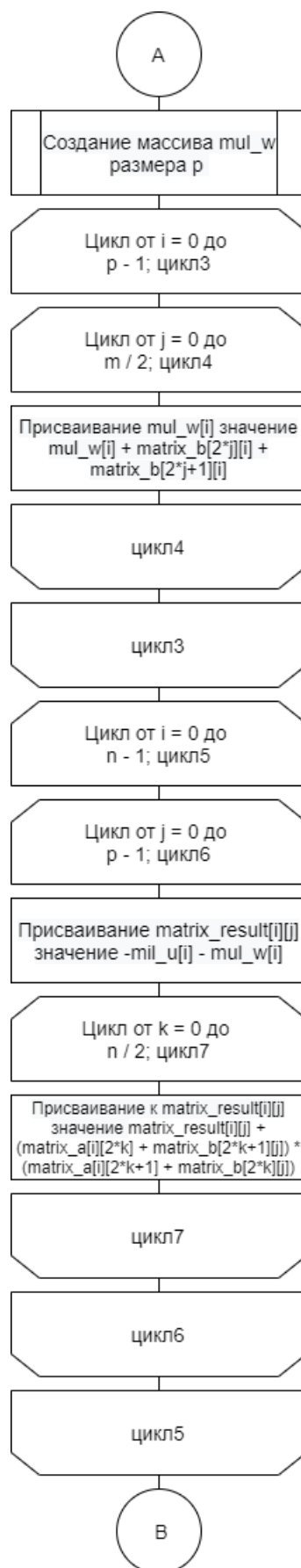


Рисунок 2.3 – Схема алгоритма умножения матриц по Винограду.

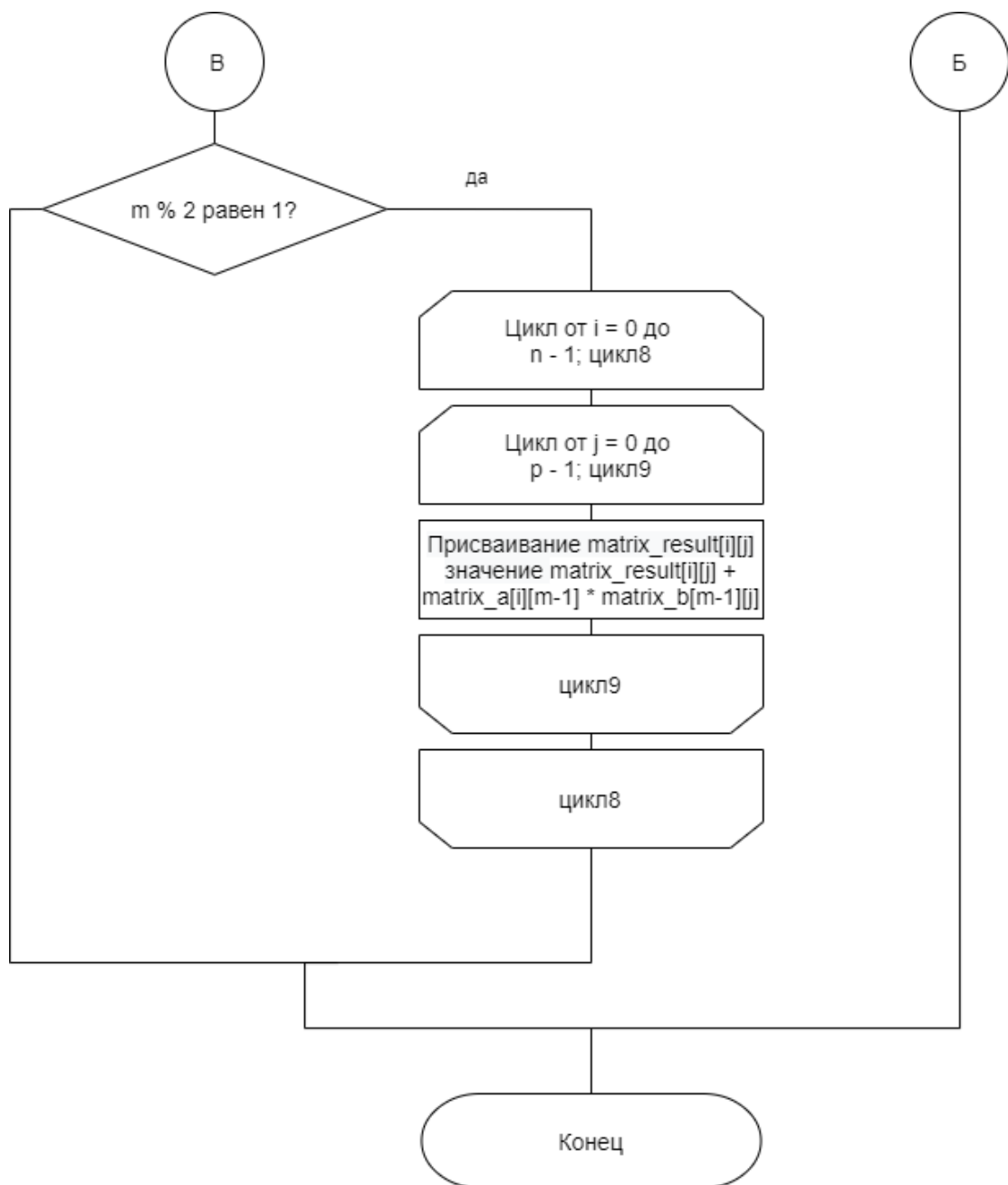


Рисунок 2.4 – Схема алгоритма умножения матриц по Винограду.

- рис. 2.5, 2.6, 2.7, 2.8 - схема алгоритма оптимизации алгоритма Винограда.

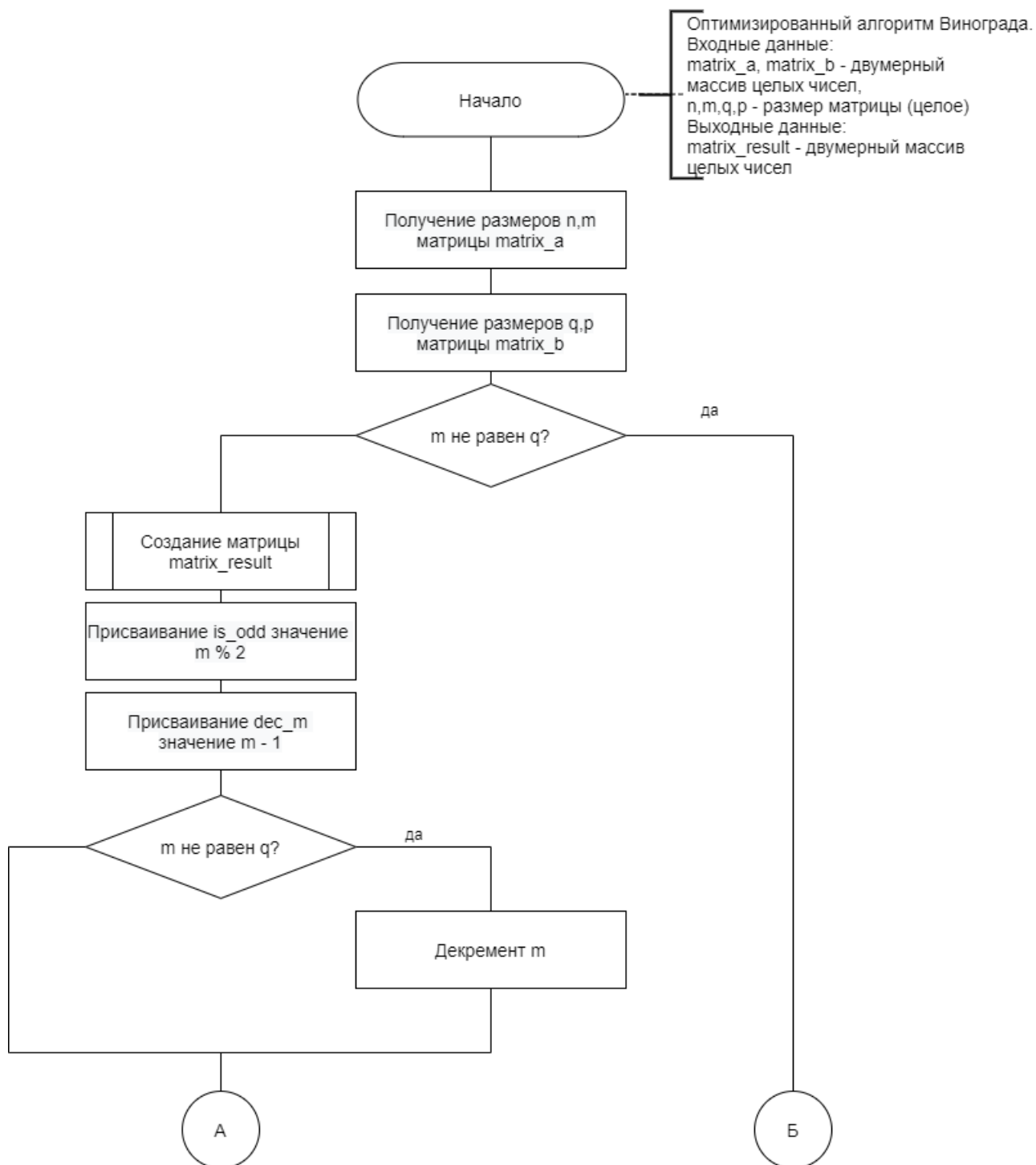


Рисунок 2.5 – Схема алгоритма оптимизации алгоритма Винограда.

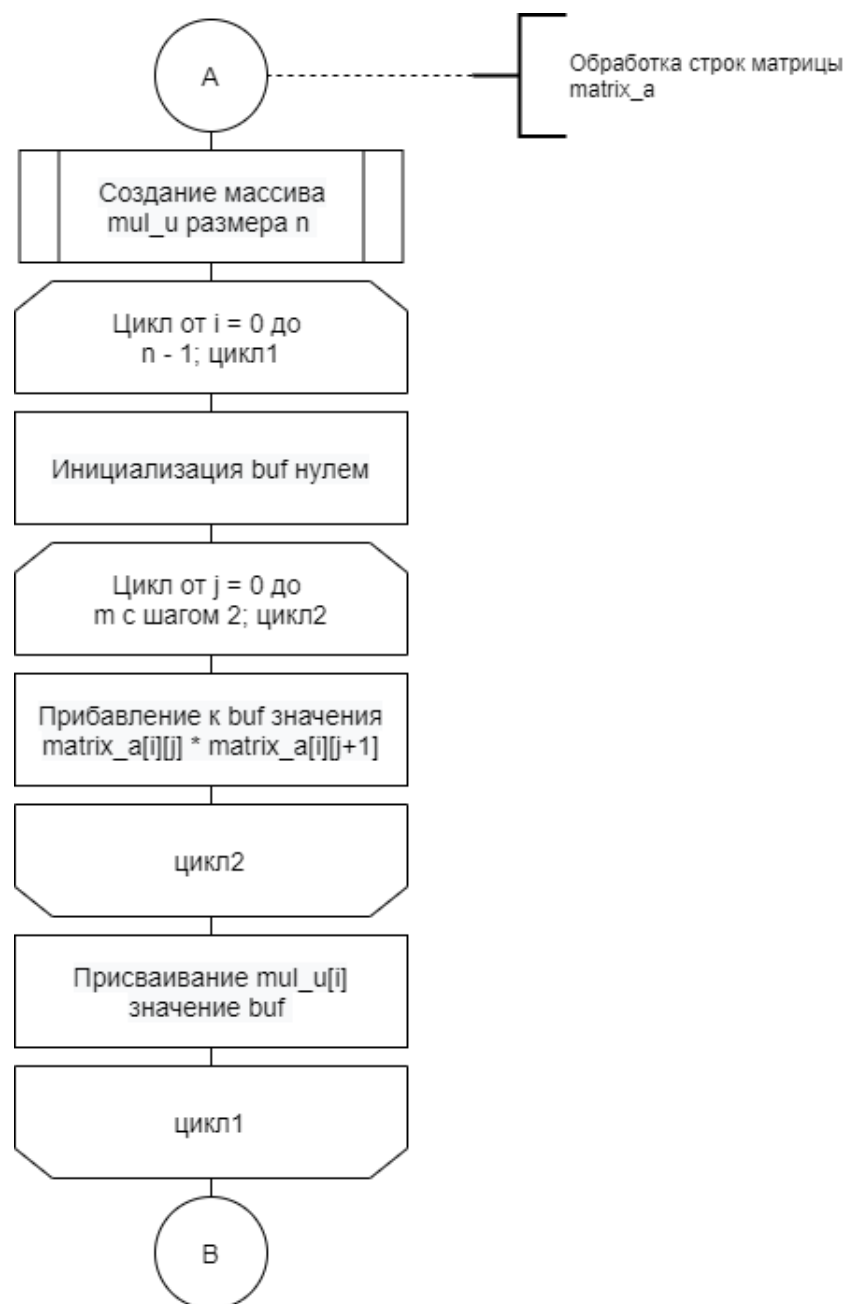


Рисунок 2.6 – Схема алгоритма оптимизации алгоритма Винограда.

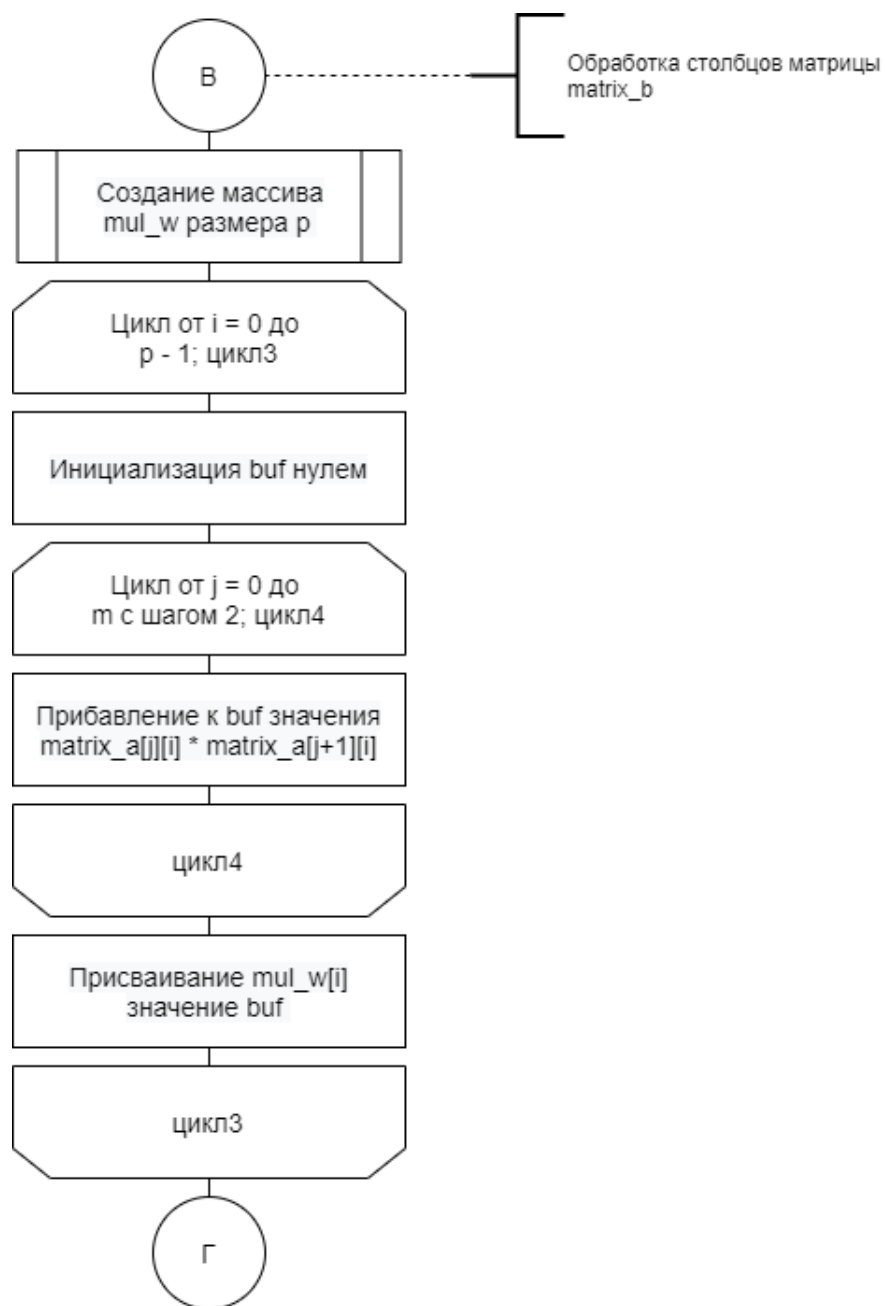


Рисунок 2.7 – Схема алгоритма оптимизации алгоритма Винограда.

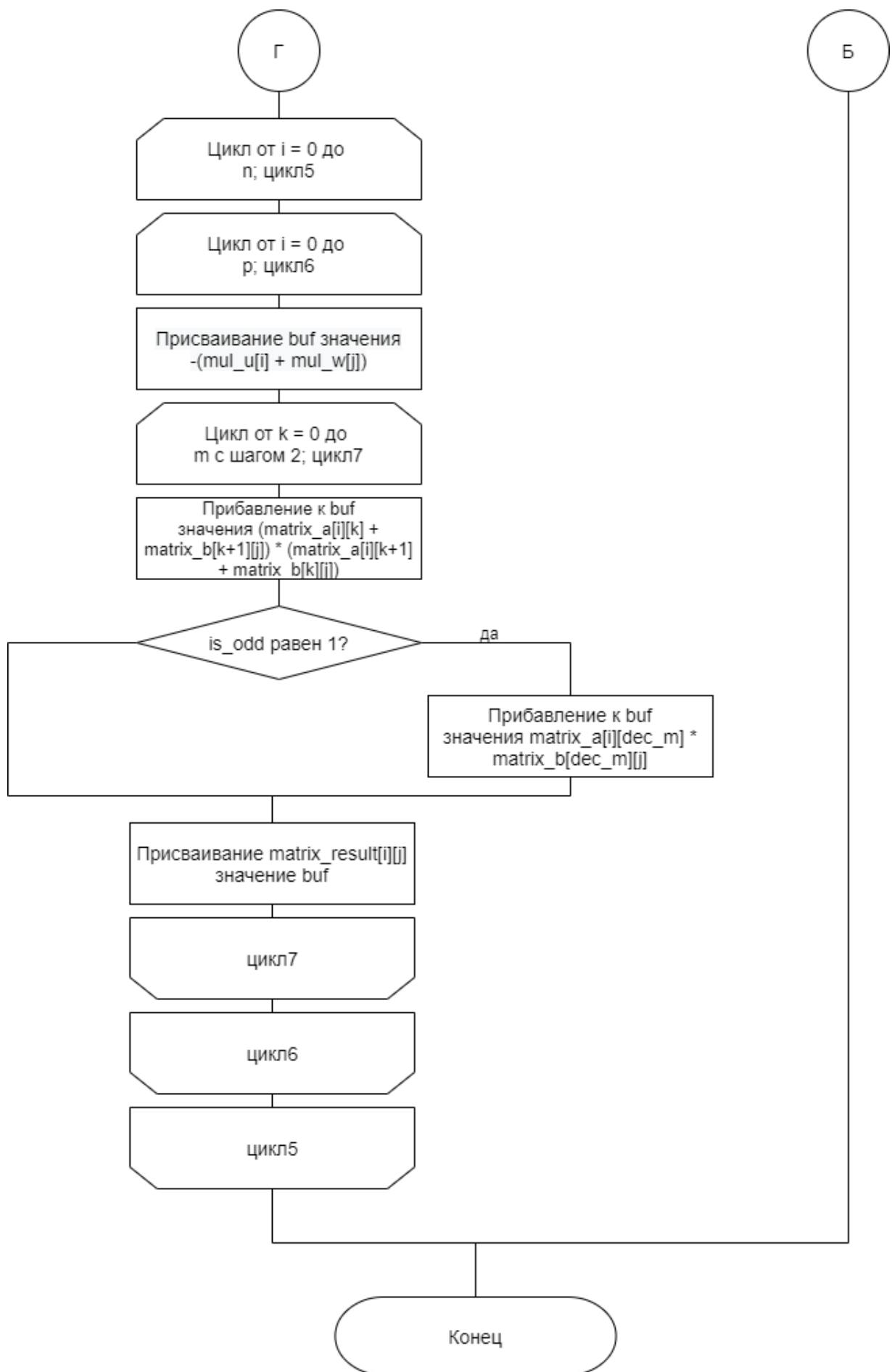


Рисунок 2.8 – Схема алгоритма оптимизации алгоритма Винограда.

2.4 Вывод

На основе теоретических данных, полученных в аналитическом разделе, были построены схемы необходимых алгоритмов. Эти алгоритмы были проанализированы с точки зрения трудоемкость выполнения. Алгоритм умножения матриц по Винограду работает медленнее стандартного алгоритма на MNK . Зато оптимизированный вариант алгоритма Винограда дает выигрыш по сравнению с ним на $6MNK$.

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

- на вход программе подаются два массива целых сгенерированных чисел, размер которых задает пользователь;
- осуществляется выбор алгоритма умножения матриц из меню;
- на вход программе подаются только корректные данные;
- на выходе программа выдает результат - полученную матрицу.

3.2 Выбор средств реализации

Для реализации алгоритмов в данной лабораторной работе был выбран язык программирования Python 3.9.7[?]. На нем можно выполнять быструю разработку программ. Имеется опыт разработки на этом языке. В качестве среды разработки был использован Visual Studio Code[?], так как в нем присутствует поддержка практически всех языков программирования. При замере процессорного времени был использован модуль time[?].

3.3 Листинги программ

Ниже представлены листинги разработанных алгоритмов умножения матриц.

Листинг 3.1 – Программный код умножения матриц стандартным способом

```
1 def multiply_matrixes_ordinary(matrix_a, matrix_b) -> list[list[int]]:  
2     n, m = matrix_a.get_size()  
3     q, p = matrix_b.get_size()
```

```

4     if m != q:
5         print('Dismatch matrix sizes.')
6         return
7     else:
8         matrix_result = Matrix(n, p)
9     for i in range(n):
10        for j in range(p):
11            for k in range(m):
12                matrix_result[i][j] = matrix_result[i][j] + \
13                    matrix_a[i][k] * matrix_b[k][j]
14    return matrix_result

```

Листинг 3.2 – Программный код умножения матриц по Винограду

```

1 def multiply_matrixes_vinograd(matrix_a, matrix_b) -> list[list[int]]:
2     n, m = matrix_a.get_size()
3     q, p = matrix_b.get_size()
4     if m != q:
5         print('Dismatch matrix sizes.')
6         return
7     else:
8         matrix_result = Matrix(n, p)
9         d = int(m / 2)
10        mul_u = [0] * n
11        for i in range(n):
12            for j in range(d):
13                mul_u[i] = mul_u[i] + \
14                    matrix_a[i][2*j] * matrix_a[i][2*j + 1]
15
16        mul_w = [0] * p
17        for i in range(p):
18            for j in range(d):
19                mul_w[i] = mul_w[i] + \
20                    matrix_b[2*j][i] * matrix_b[2*j + 1][i]
21
22        for i in range(n):
23            for j in range(p):
24                matrix_result[i][j] = -mul_u[i] - mul_w[j]
25                for k in range(d):
26                    matrix_result[i][j] = matrix_result[i][j] + \
27                        (matrix_a[i][2*k] + matrix_b[2*k+1][j]) * \
28                        (matrix_a[i][2*k+1] + matrix_b[2*k][j])
29
30        if m % 2 == 1:
31            for i in range(n):
32                for j in range(p):
33                    matrix_result[i][j] = matrix_result[i][j] + \
34                        matrix_a[i][m-1] * matrix_b[m-1][j]
35

```

```
36         return matrix_result
```

Листинг 3.3 – Программный код нахождения расстояния Левенштейна рекурсивно с использованием матрицы

```
1 def multiply_matrixes_vinograd_optimized(matrix_a, matrix_b) -> list[
  list[int]]:
2     n, m = matrix_a.get_size()
3     q, p = matrix_b.get_size()
4     if m != q:
5         print('Несовпадение размеров матриц ')
6         return
7     else:
8         matrix_result = Matrix(n, p)
9         is_odd = m % 2
10        dec_m = m - 1
11        mul_u = [0] * n
12        if is_odd:
13            m -= 1
14        for i in range(n):
15            buf = 0
16            for j in range(0, m, 2):
17                buf += matrix_a[i][j] * matrix_a[i][j + 1]
18            mul_u[i] = buf
19
20        mul_w = [0] * p
21        for i in range(p):
22            buf = 0
23            for j in range(0, m, 2):
24                buf += matrix_b[j][i] * matrix_b[j + 1][i]
25            mul_w[i] = buf
26
27        for i in range(n):
28            for j in range(p):
29                buf = -(mul_u[i] + mul_w[j])
30                for k in range(0, m, 2):
31                    buf += (matrix_a[i][k] + matrix_b[k+1][j]) * \
32                        (matrix_a[i][k+1] + matrix_b[k][j])
33
34                if is_odd == 1:
35                    buf += matrix_a[i][dec_m] * matrix_b[dec_m][j]
36                matrix_result[i][j] = buf
37
38        return matrix_result
```

Листинг 3.4 – Программный код нахождения расстояния Дамерау-Левенштейна рекурсивно

```

1 def recursive_damerau_levenshtein(str_1: str, str_2: str) -> int:
2     if str_1 == '' or str_2 == '':
3         return abs(len(str_1) - len(str_2))
4     match = 0 if str_1[-1] == str_2[-1] else 1
5     insert = recursive_damerau_levenshtein(str_1, str_2[:-1]) + 1
6     delete = recursive_damerau_levenshtein(str_1[:-1], str_2) + 1
7     replace = recursive_damerau_levenshtein(str_1[:-1], str_2[:-1]) + \
8         match
9
10    if len(str_1) > 1 and len(str_2) > 1 and str_1[-1] == str_2[-2] \
11        and str_2[-1] == str_1[-2]:
12        distance = min(insert, delete, replace,
13            recursive_damerau_levenshtein(str_1[:-2], str_2[:-2]) +
14            1)
15    else:
16        distance = min(insert, delete, replace)
17    return distance

```

3.4 Вспомогательные функции

На листингах представлены программные модули, которые используются в данных функциях:

Листинг 3.5 – Программный код создания для кэша в виде строки

```

1 def create_row(len_row: int, flag_row: int) -> list[int]:
2     row = list()
3     if flag_row == 1:
4         for i in range(len_row):
5             row.append(i)
6     else:
7         for i in range(len_row):
8             row.append(0)
9     return row

```

Листинг 3.6 – Программный код обмена двух строк

```

1 def swap_rows(row_1: list[int], row_2: list[int]) \
2     -> Tuple[list[int], list[int]]:
3     temp_row = list()
4     temp_row = deepcopy(row_1)
5     row_1 = deepcopy(row_2)
6     row_2 = deepcopy(temp_row)
7     return row_1, row_2

```

3.5 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов:

Таблица 3.1 – Таблица тестов

№	Описание теста	Слово 1	Слово 2	Алгоритм	
				Левенштейн	Дамерау-Левенштейн
1	Пустые строки	”	”	0	0
2	Нет повторяющихся символов	deersору	раздел	8	8
3	Инверсия строк	insert	tresni	6	6
4	Два соседних символа	heart	heatr	2	1
5	Одинаковые строки	таблица	таблица	0	0
6	Одна строка меньше другой	город	горо	1	1

Вывод

В данном разделе был выбран язык программирования, среда разработки. Реализованы функции, описанные в аналитическом разделе, и проведено их тестирование методом черного ящика по таблице 3.1.