Анализ Алгоритмов

Лабораторная работа №2

«Алгоритмы умножения матриц»

Юмаев Артур Русланович

ИУ7-55

Преподаватель: Волкова Л.Л.

Оглавление

[Введение 3](#_Toc23953127)

[Постановка задачи 4](#_Toc23953128)

[1. Аналитическая часть 5](#_Toc23953129)

[Классический подход 5](#_Toc23953130)

[Алгоритм Винограда 6](#_Toc23953131)

[Оптимизированный алгоритм Винограда 6](#_Toc23953132)

[2. Конструкторская часть 7](#_Toc23953133)

[Оценка трудоемкости алгоритмов 17](#_Toc23953134)

[Классический алгоритм умножения матриц 17](#_Toc23953135)

[Алгоритм Винограда 17](#_Toc23953136)

[Оптимизированный алгоритм Винограда 18](#_Toc23953137)

[3. Технологическая часть 19](#_Toc23953138)

[4. Исследовательская часть 22](#_Toc23953139)

[Заключение 24](#_Toc23953140)

# Введение

Алгоритмы перемножение матриц имеют довольно обширное применение. В данное время они используются в области искусственного интеллекта. Перемножение матриц – одна из основных операций, на которых строятся алгоритмы искусственных нейронных сетей, которые могут анализировать изображения, находить скрытые взаимосвязи и классифицировать их, заранее обучившись на некой тренировочной выборки. Перемножения матриц также используются в прикладной физике, математике, математической статистике и многих других прикладных науках.

# Постановка задачи

Цель: изучить алгоритмы умножения матриц.

Для достижения поставленной цели требуется решить следующие задачи.

* Реализовать классический алгоритм умножения матриц.
* Изучить и реализовать алгоритм Винограда.
* Разработать и реализовать оптимизированный вариант алгоритма Винограда.
* Выбрать модель оценки трудоёмкости и по ней дать оценку трудоёмкости классическому алгоритму умножения матриц, алгоритму Винограда (для лучшего и худшего случая) и оптимизированному алгоритму Винограда (для лучшего и худшего случая).
* Сделать замеры времени для алгоритмов.
* Результаты экспериментов сравнить с теоретическими оценками трудоёмкости.
* Сделать выводы.

# Аналитическая часть

В данный момент существуют несколько алгоритмов перемножения матриц. Ниже в таблице 1 приведен их список с коэффициентом ω, который показывает сложность алгоритмов ).

Таблица 1

Сравнение эффективности по времени алгоритмов умножения матриц

|  |  |
| --- | --- |
| Алгоритм | ω |
| Классический (1950) | 3.0 |
| Алгоритм Пана (1978) | 2.78041 |
| Алгоритм Бини (1979) | 2.7799 |
| Алгоритм Шёнхаге (1981) | 2.695 |
| Алгоритм Копперсмита — Винограда (1990) | 2.3727 |

Алгоритм, который будет реализован в данной работе является одним из самых эффективных на данный момент.

В данном разделе будет приведено математическое описание алгоритмов перемножения матриц. Буду рассмотрены 3 подхода: классический, алгоритм Винограда и оптимизированный алгоритм Винограда.

## Классический подход

Предположим, что необходимо получить матрицу Для нахождения значений элементов матрицы используют следующее выражение:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Классический алгоритм напрямую реализует эту формулу.

## Алгоритм Винограда

Можно заметить, что элементы из суммы выражения 1 можно переписать как:

т. е. как сумму произведения сумм и двух произведений. Учитывая, что упомянутые два произведения можно рассчитать заранее для обработки двух элементов матрицы теперь нужно не сложение и два умножения, а умножение и два сложения, что проще с точки зрения вычислений. Таким образом, алгоритм Винограда состоит в следующем.

1. Совершить расчет заранее двух произведений для каждого ряда и столбца матрицы-результата (одно произведение считается для ряда, другое для столбца). Для хранения результатов используется промежуточный буфер;
2. По вышеприведённой формуле осуществить расчёт каждого элемента матрицы;
3. В случае, если в произведении b – нечётное число, пройтись во второй раз по матрице, дополняя элементы недостающим элементом (который не был описан вышеописанной суммой).

Можно заметить, что пункт 3 необходимо выполнять только в некоторых случаях, но если это происходит, то получается существенное увеличение времени работы алгоритма.

## Оптимизированный алгоритм Винограда

1. Внутри тройного цикла накапливать результат в буфер, а вне цикла сбрасывать его в ячейку матрицы.
2. Заранее вычислить d = [n / 2], где [x] – целая часть x,  
   где n = b в матрице B(b, c)
3. Заменить MulH[i] = MulH[i] + … на MulH[i] += … (аналогично для MulV),  
   где MulH и MulB – временные массивы для предварительного рассчета сумм произведений

# 2. Конструкторская часть

В данном разделе будут приведены схемы алгоритмов умножения матрицы. Классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. На рисунках 1-2 изображен классический алгоритм умножения матриц. Сложность данного алгоритма составляет O(N^3).

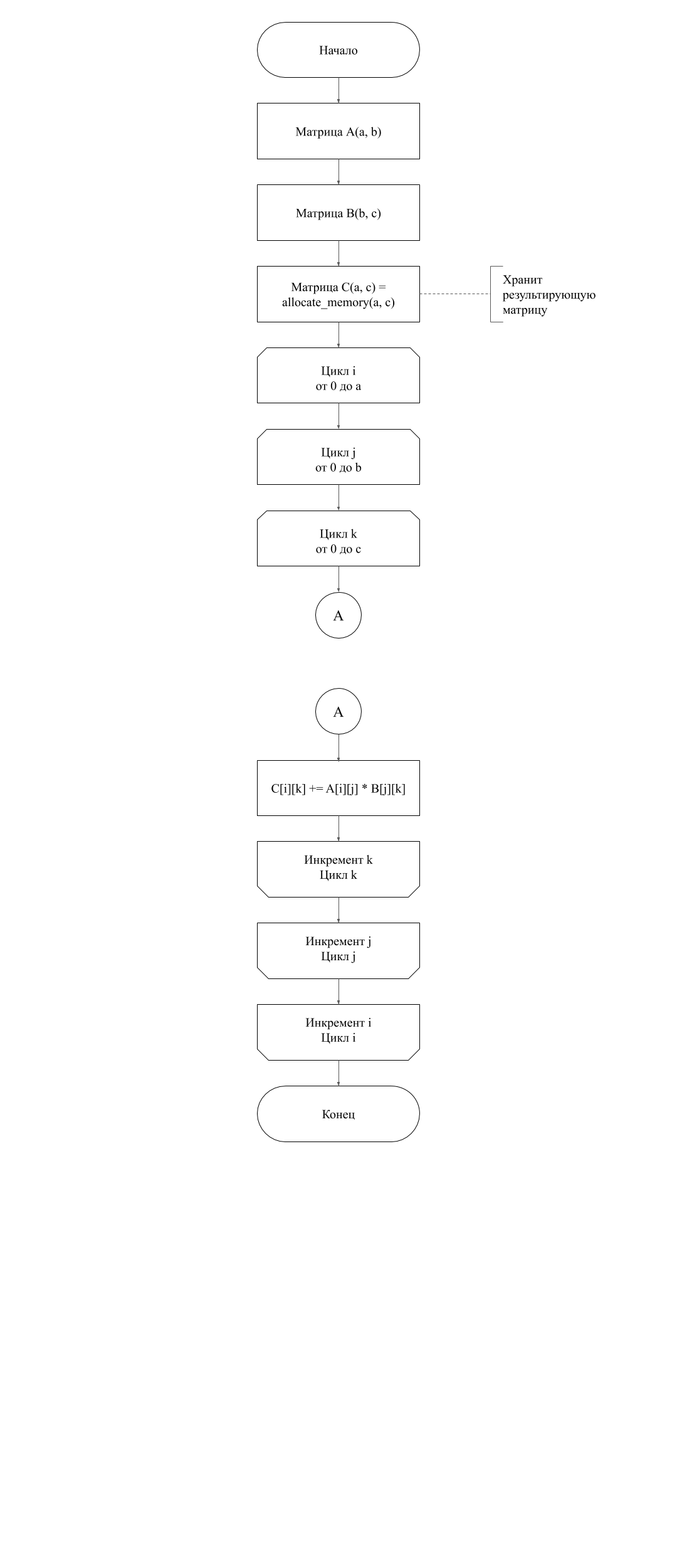


Рисунок 1 - Классический алгоритм умножения матриц. Часть 1

На рисунке 2 изображена вторая часть классичесого алгоритма для умножения матриц.

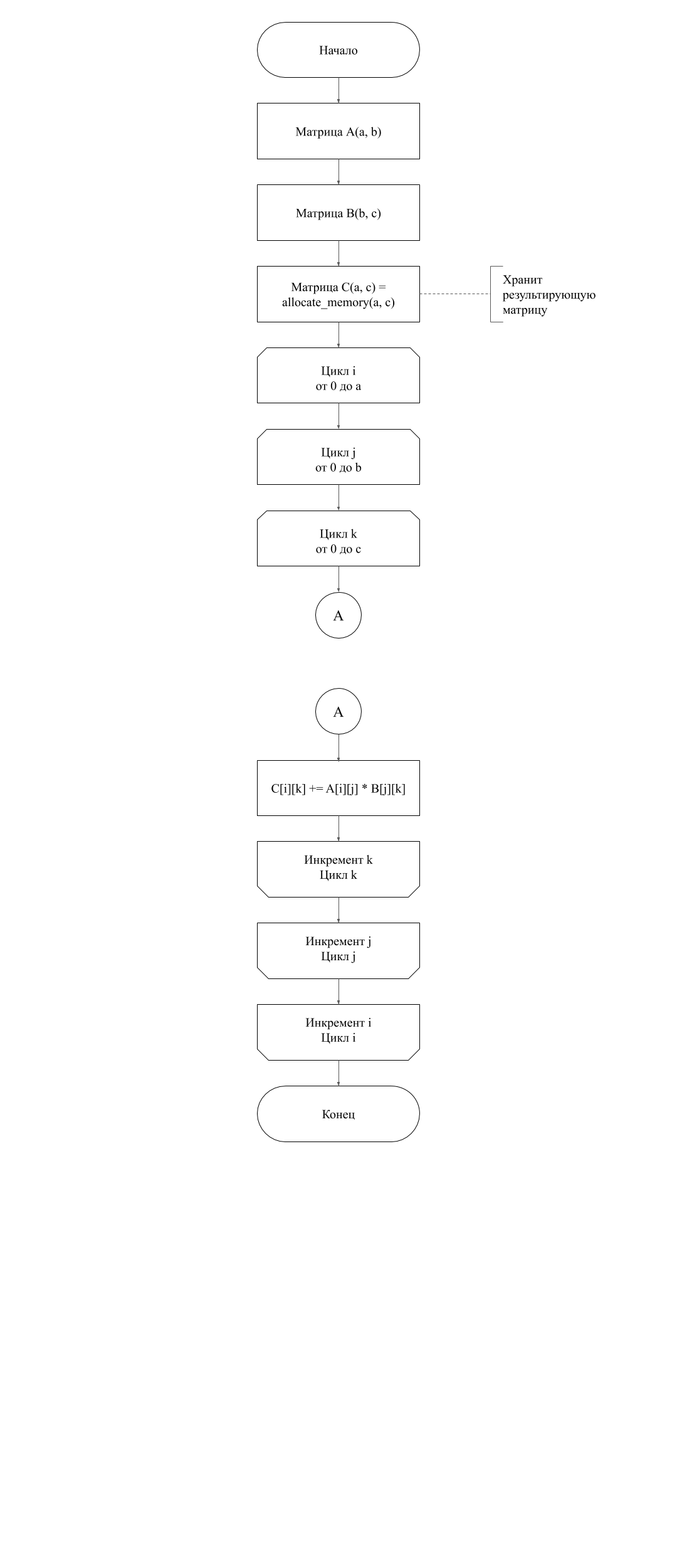


Рисунок 2 - Классический алгоритм умножения матриц. Часть 2

На рисунке 3 изображена схема алгоритма Винограда умножения матриц. На нем визуально описано заполнение двух временных масивов для выполнения повторяющихся операций заранее.

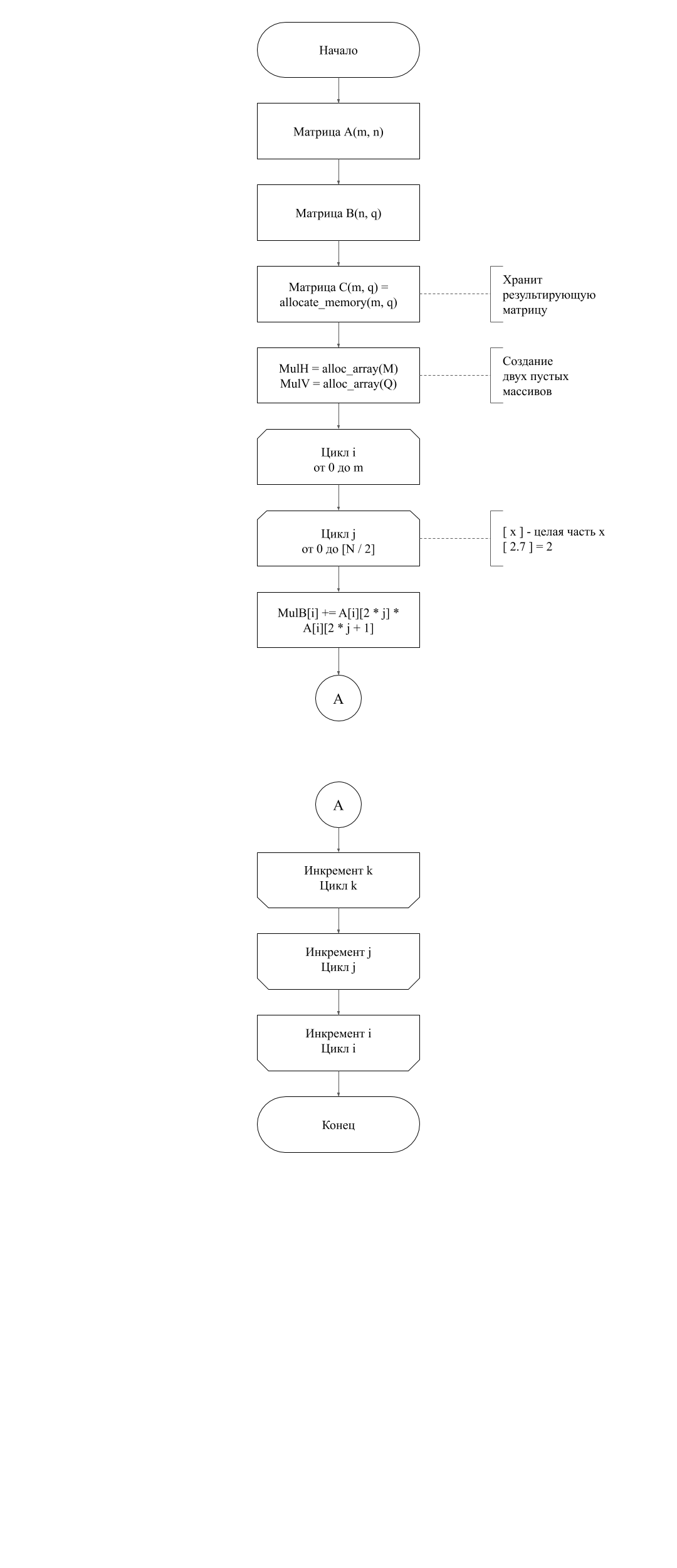


Рисунок 3 - Алгоритм Винограда. Часть 1

На рисунке 4 показано заполнение второго массива в Алгоритме Винограда.

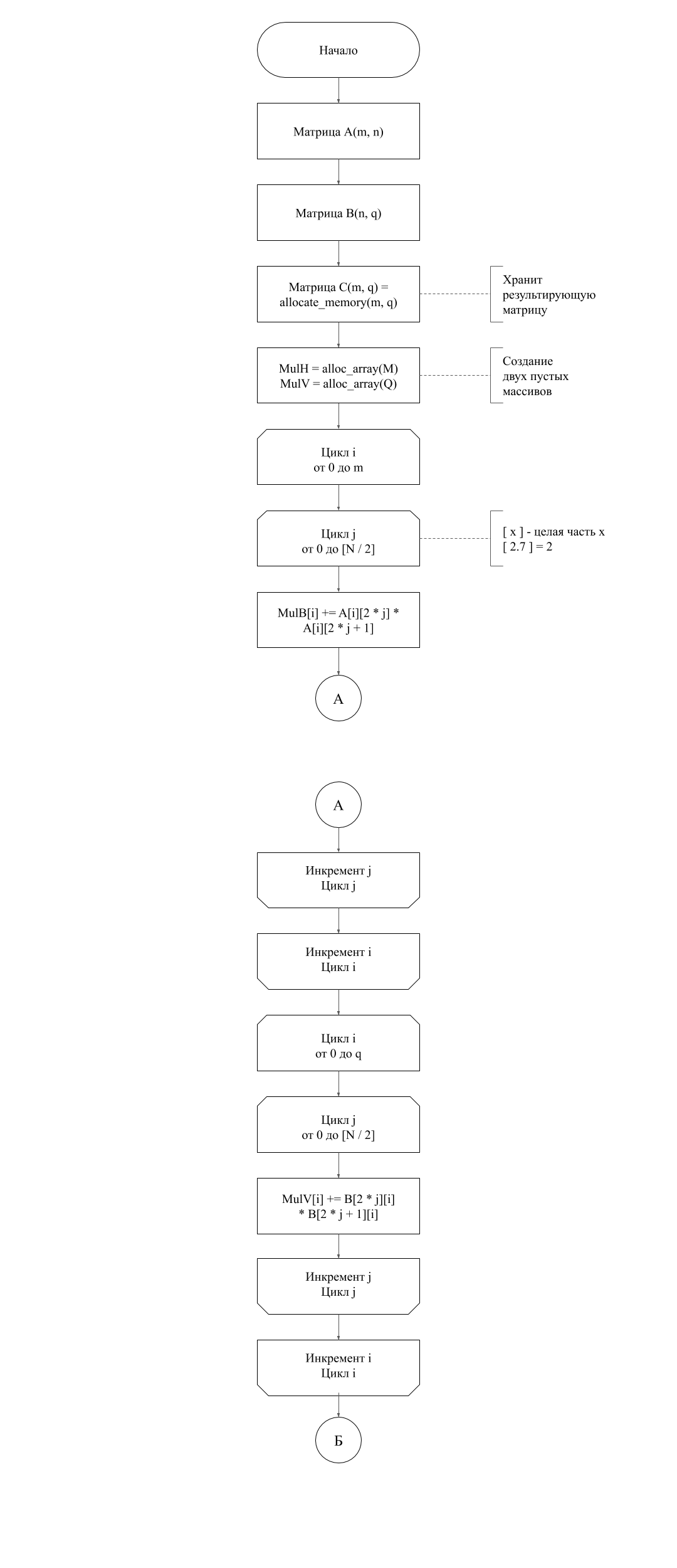


Рисунок 4 - Алгоритм Винограда. Часть 2

На рисунке 5 показана часть В алгоритма Винограда, в ней мы заносим значения из временных массивов в матрицу С и считаем произведения оставшихся элементов.

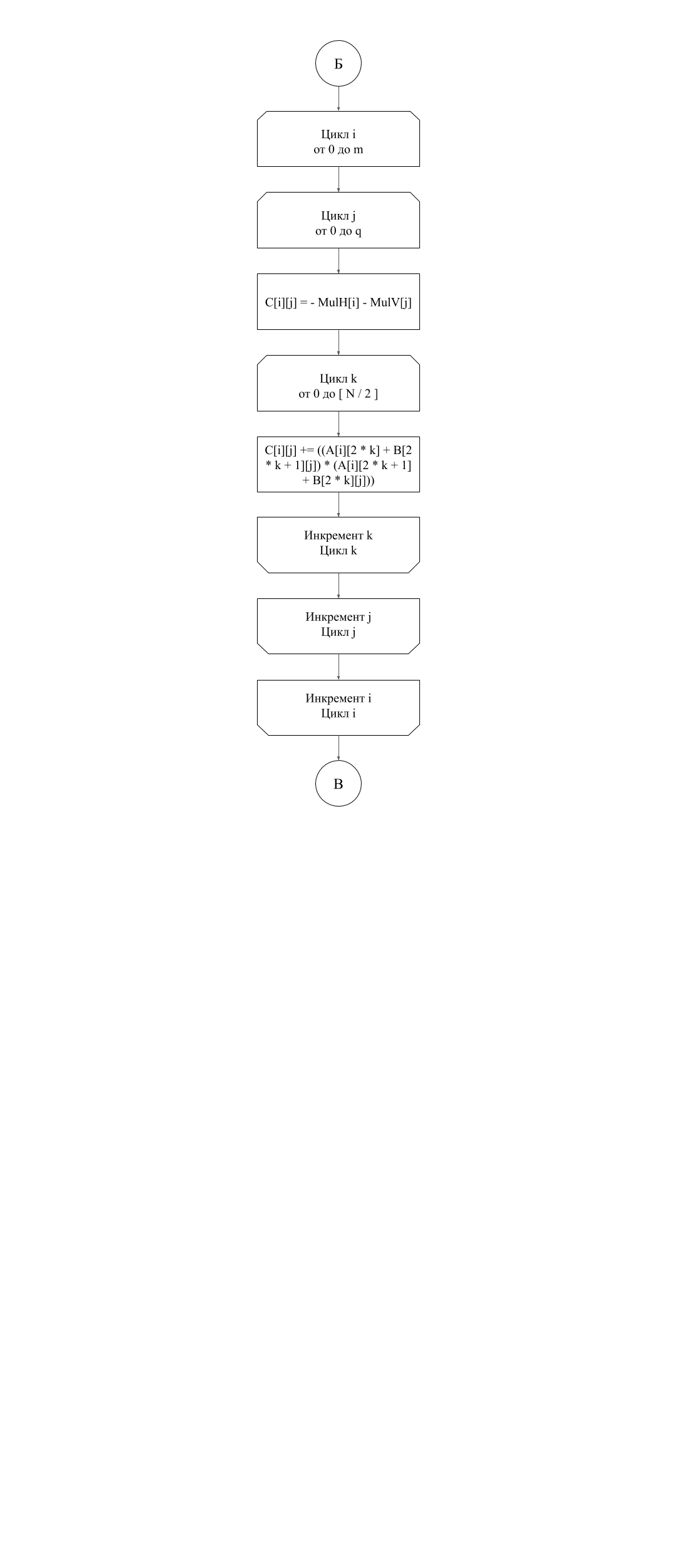


Рисунок 5 - Алгоритм Винограда. Часть Б

На рисунке 6 показана вторая часть В алгоритма Винограда, в ней заполняем оставшуюся часть матрицы, если ее длина нечетная или возвращаем результирующую матрицу.

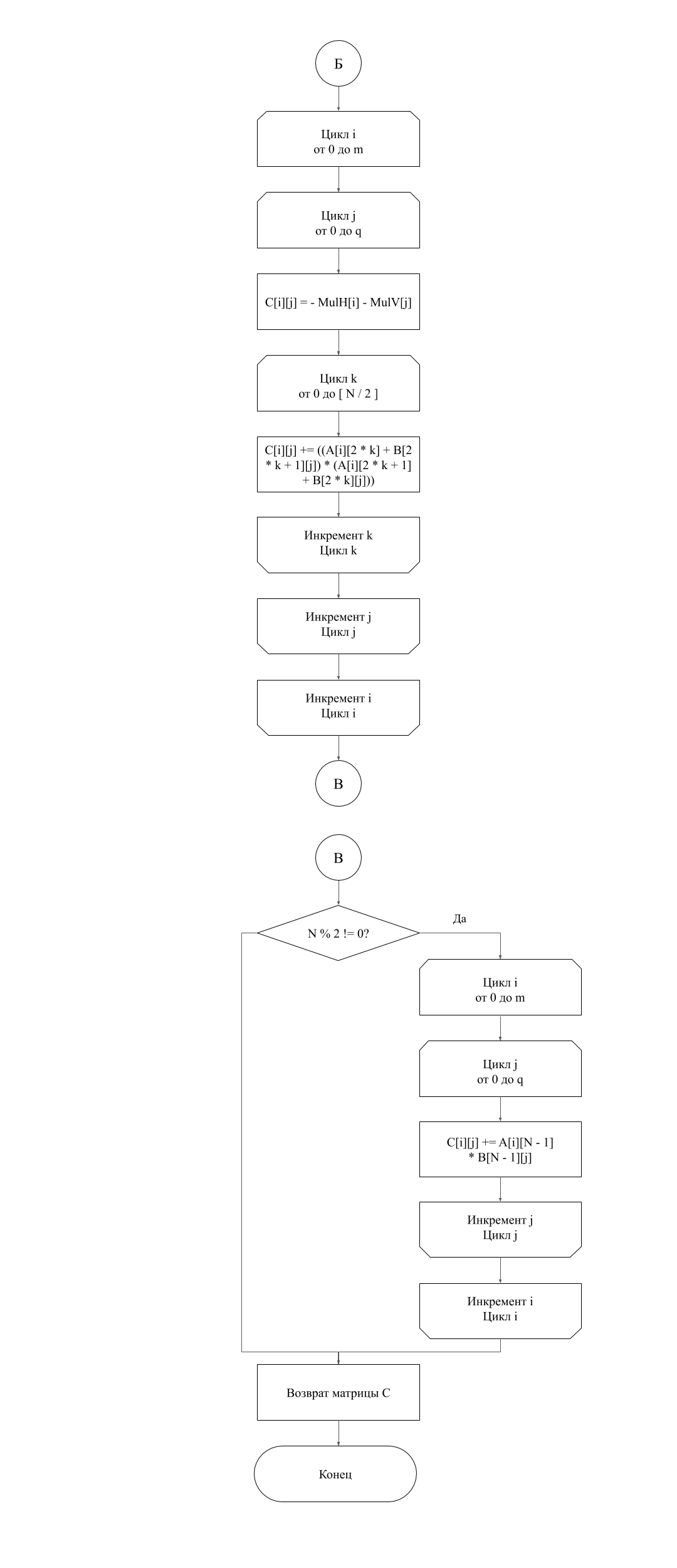


Рисунок 6 - Алгоритм Винограда. Часть В

На рисунке 7 изображено начало оптимизированного алгоритма винограда, создание временных массивов и подсчет сумм из матрицы А в массив MulH.

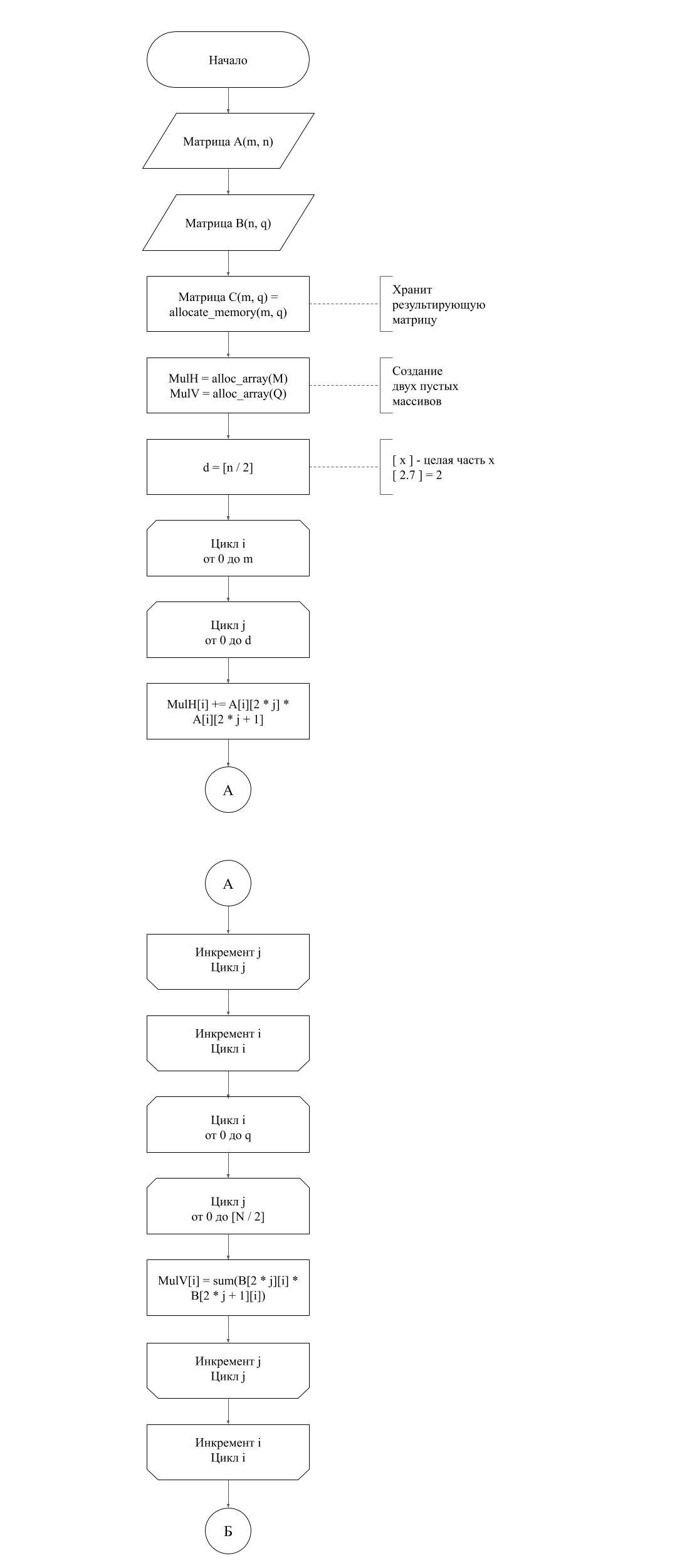


Рисунок 7 - Оптимизированный алгоритм Винограда. Часть А

На рисунке 8 изображен подсчет сумм из матрицы B в массив MulV.

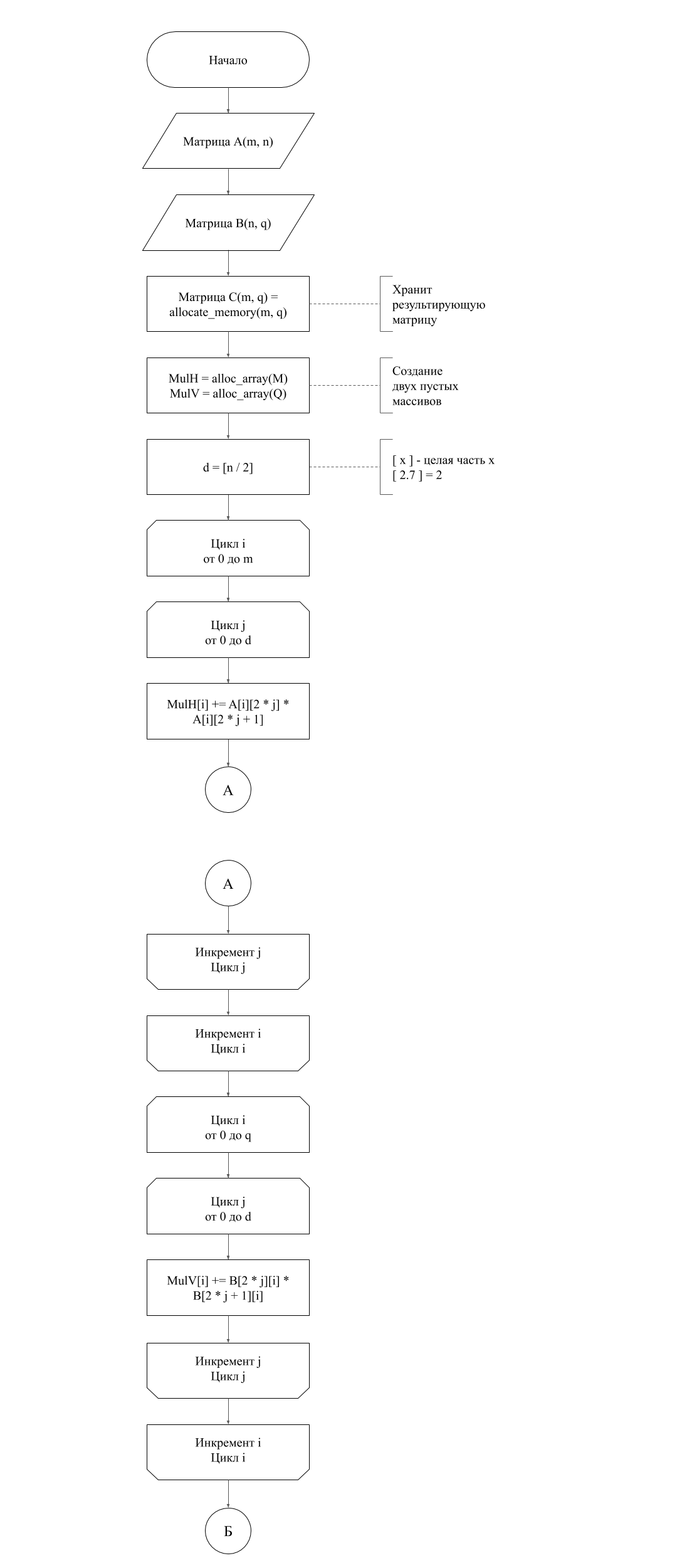


Рисунок 8 - Оптимизированный алгоритм Винограда. Часть Б

На рисунке 9 изображено заполнение результирующей матрицы С из временных массивов MulV и MulH.

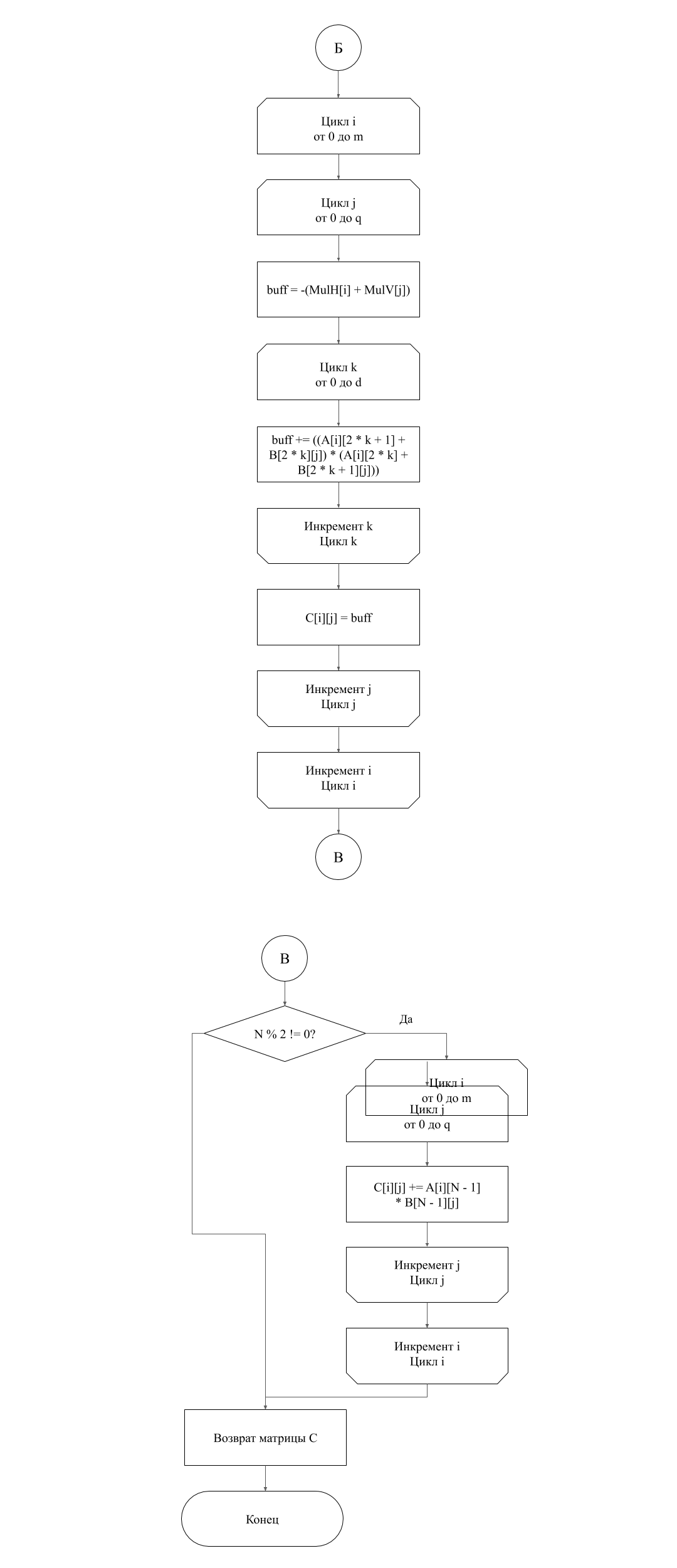


Рисунок 9 - Оптимизированный алгоритм Винограда. Часть В

На рисунке 10 показана вторая часть В оптимизированном алгоритма Винограда, в ней заполняем оставшуюся часть матрицы, если ее длина нечетная или возвращаем результирующую матрицу.

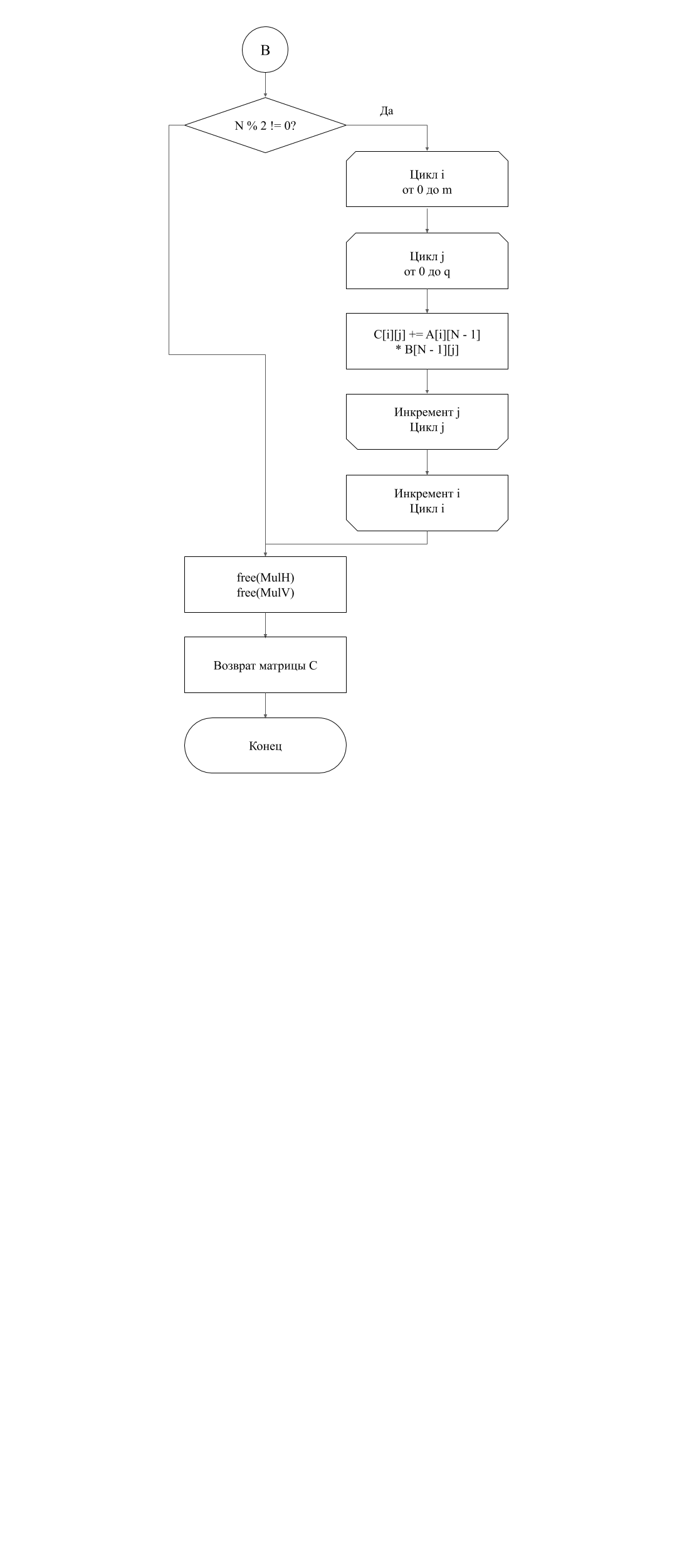


Рисунок 10 - Оптимизированный алгоритм Винограда. Часть С

## Оценка трудоемкости алгоритмов

Используется C-подобная модель оценки трудоёмкости.

Трудоёмкость операций:

1. Трудоемкость операций +, −, =, + =, − =, <, > ==, ++ равна 1.
2. Трудоемкость операций \*, /, % равна 2.
3. Трудоемкость операции [. . .] равна 3.

### Классический алгоритм умножения матриц

Вывод:

Итоговая формула:

### Алгоритм Винограда

Итоговая формула для лучшего случая:

Для худшего случая:

### Оптимизированный алгоритм Винограда

Итоговая формула для лучшего случая:

Для худшего случая:

# 3. Технологическая часть

В данном разделе будут приведены листинги для каждого из алгоритмов на языке Python, реализации алгоритмов приведены в листингах 1 – 3.

Листинг 1. Классический алгоритм умножения матриц

|  |
| --- |
| def classic\_matrix\_mult(A, B):  A, B = np.array(A), np.array(B)    if len(B) != len(A[0]):  print("Error! Different dimension!")  return None  n = len(A)  m = len(A[0])  t = len(B[0])  C = np.zeros((A.shape[0], B.shape[1]))  for i in range(n):  for j in range(m):  for k in range(t):  C[i][k] += A[i][j] \* B[j][k]  return C |

Листинг 2. Алгоритм Винограда

|  |
| --- |
| def classic\_winograd(A, B):  M = len(A)  N = len(B)  Q = len(B[0])    C = np.zeros((M, Q))  if N != len(A[0]):  print("Different dimension of the matrics")  return None  MulB = np.zeros((M))  MulV = np.zeros((Q))  for i in range(M):  for j in range(N // 2):  MulB[i] += A[i][2 \* j] \* A[i][2 \* j + 1]  for i in range(Q):  for j in range(N // 2):  MulV[i] += B[2 \* j][i] \* B[2 \* j + 1][i]  for i in range(M):  for j in range(Q):  C[i][j] = - MulB[i] - MulV[j]  for k in range(N // 2):  C[i][j] += ((A[i][2 \* k] + B[2 \* k + 1][j]) \*  (A[i][2 \* k + 1] + B[2 \* k][j]))  if N % 2:  for i in range(M):  for j in range(Q):  C[i][j] += A[i][N - 1] \* B[N - 1][j]  return np.array(C) |

Листинг 3. Оптимизированный алгоритм Винограда

|  |
| --- |
| def optimized\_winograd(A, B):  M = len(A)  N = len(B)  Q = len(B[0])    C = np.zeros((M, Q)) # Результирующая матрица  if N != len(A[0]):  print("В матрицах А(m, n), B(q, r) n != q")  return None  # Оптимизация №1 - избавиться от деления в цикле  d = N // 2  MulH = np.zeros((M))  MulV = np.zeros((Q))  # Оптимизация #2.1 – замена MulH[i] = MulH[i] + … на MulH[i] +=  for i in range(M):  for j in range(d):  MulH[i] += A[i][2 \* j] \* A[i][2 \* j + 1]  # Оптимизация #2.2 – замена MulV[i] = MulV[i] + … на MulV[i] +=  for i in range(Q):  for j in range(d):  MulV[i] += B[2 \* j][i] \* B[2 \* j + 1][i]  # Оптимизация №3 накопление результата в буфер  for i in range(M):  for j in range(Q):  buff = -(MulH[i] + MulV[j])  for k in range(d):  buff += ((A[i][2 \* k + 1] + B[2 \* k][j]) \* \  (A[i][2 \* k] + B[2 \* k + 1][j]))  # Сброс буфера в ячейку  C[i][j] = buff    if N % 2 != 0:  for i in range(M):  for j in range(Q):  C[i][j] += A[i][N - 1] \* B[N - 1][j]  # Очистка временных массивов  del MulH  del MulV  return C |

# 4. Исследовательская часть

В данном разделе будет приведено сравнение классического алгоритма умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Тестирование проводилась на сначала квадратных матрицах четного размера от *50* до *400* в шагом *50*, далее также проводились измерения на матрицах нечетного размера:

Замер времени проводился с помощью библиотеки time в Python 3.7 и метода process\_time().

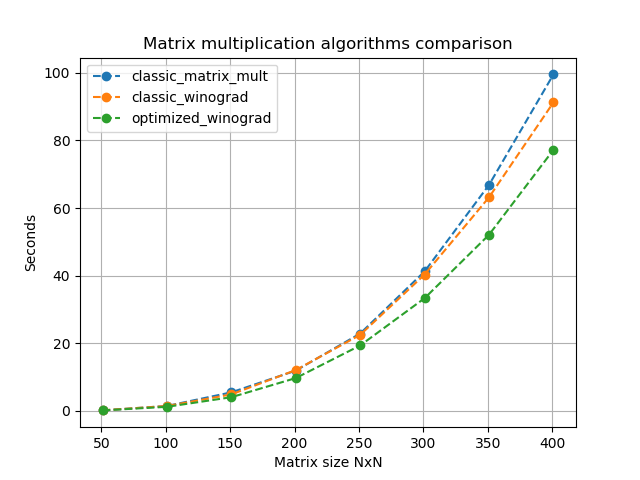


Рисунок 11 - Сравнение алгоритмов по скорости работы на матрицах четных размеров

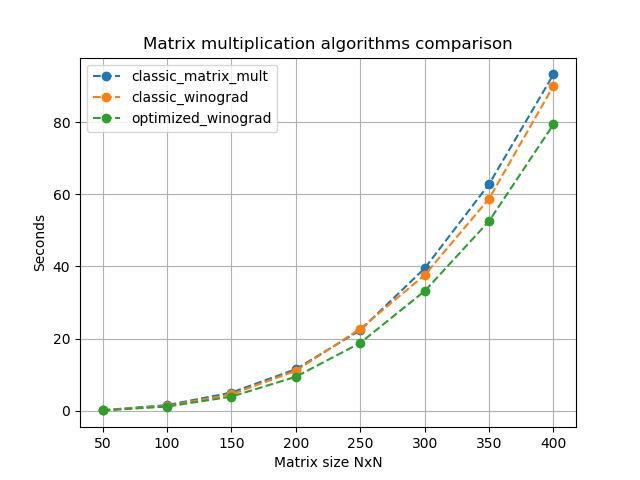


Рисунок 12 - Сравнение алгоритмов по скорости работы на матрицах нечетных размеров

По опыту видно, что матрицы с четным размером в среднем считаются быстрее, чем матрицы с нечетным размером.

# Заключение

Необходимо учитывать влияние размера входных данных на скорость выполнения алгоритмов. Трудоёмкость позволяет примерно оценить изменение времени, необходимого на выполнение алгоритма, в зависимости от входных параметров (размеры входных данных), но не позволяет оценить относительную скорость выполнения алгоритмов относительно друг друга.

В результате выполнения данной работы был реализован классический алгоритм умножения матриц. Был изучен и реализован алгоритм Винограда. Был разработан и реализован оптимизированный вариант алгоритма Винограда. Была выбрана модель оценки трудоёмкости и по ней были даны оценки трудоёмкости классическому алгоритму умножения матриц, алгоритму Винограда и оптимизированному алгоритму Винограда. Проведены замеры времени для алгоритмов. Было проведено сравнение результатов экспериментов с теоретическими оценками трудоёмкости. По итогам проведения работы были сделаны выводы.