*Государственное образовательное учреждение*
*высшего профессионального образования*
***«Московский государственный технический***
***университет имени Н. Э. Баумана»***
***(МГТУ им. Н.Э. Баумана)***

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**ЛАБОРАТОРНАЯ РАБОТА №2**

**на тему:**

# Алгоритмы перемножения матриц

Студент ИУ7-54: Морозов И. А
Преподаватель: Погорелов. Д. А.

Москва 2018

# Введение

Целью данной лабораторной работы является изучение и сравнение обычного и оптимизированного метода перемножения матриц - алгоритма Винограда. А так же получение практических навыков типовой оптимизации и реализации алгоритма Винограда.

# 1. Аналитическая часть

## 1.1 Описание алгоритмов

### 1.1.1 Алгоритм перемножения матриц

Описание алгоритма:

Пусть даны две прямоугольные матрицы A и B размерности l x m и m x n соответственно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}.$$

Тогда матрица C размерностью l x n:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{bmatrix},$$

в которой:

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

называется их произведением

### 1.1.2 Алгоритм Винограда

<u>Описание алгоритма:</u>

Усовершенствованный алгоритм умножения матриц таким образом, что 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 :

$$V = (v1, v2, v3, v4) \text{ и } W = (w1, w2, w3, w4).$$

Их скалярное произведение равно:

$$V \cdot W = v1w1 + v2w2 + v3w3 + v4w4$$

Это равенство можно переписать в виде :

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + w1)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$$

## 2. Конструкторская часть

Для исследований будут реализованы три алгоритма: алгоритм перемножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

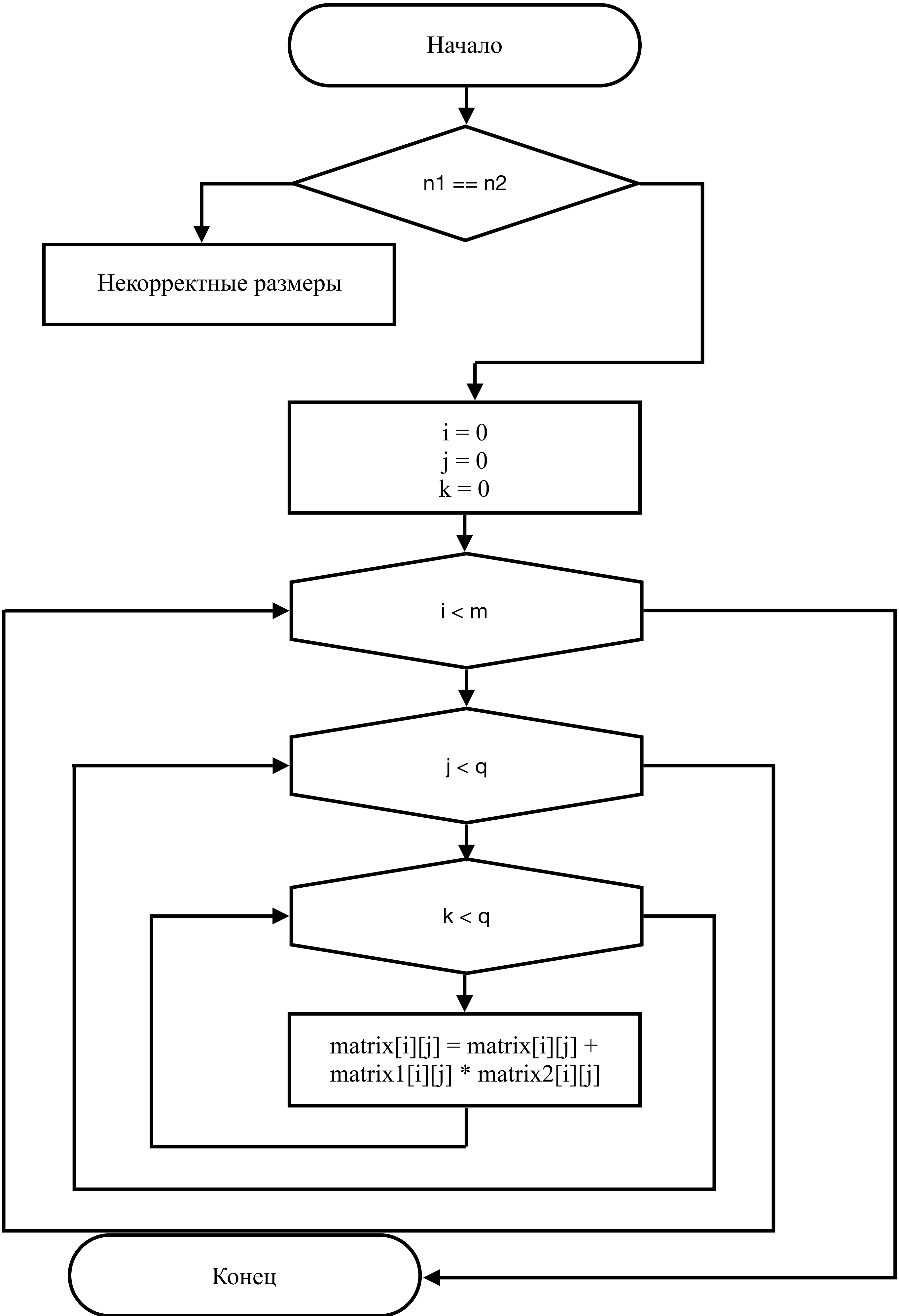
## 2.1 Разработка алгоритмов

### 2.1.1 Схема алгоритма перемножения матриц

<u>Входные данные:</u>

matrix1 - первая матрица
m - количество строк первой матрицы
n1 - количество столбцов первой матрицы
matrix2 - вторая матрица
n2 - количество строк второй матрицы

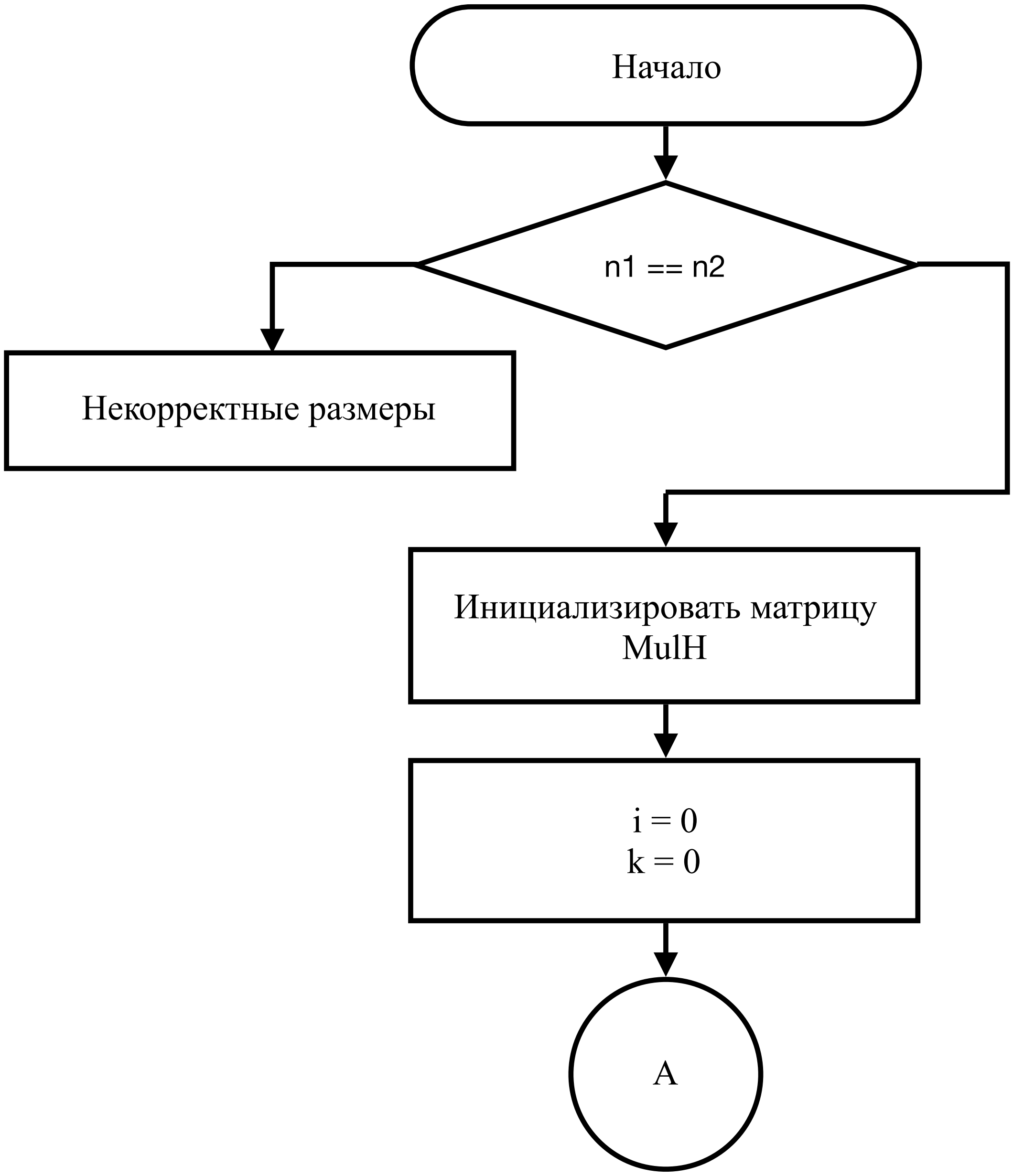
q - количество столбцов второй матрицы

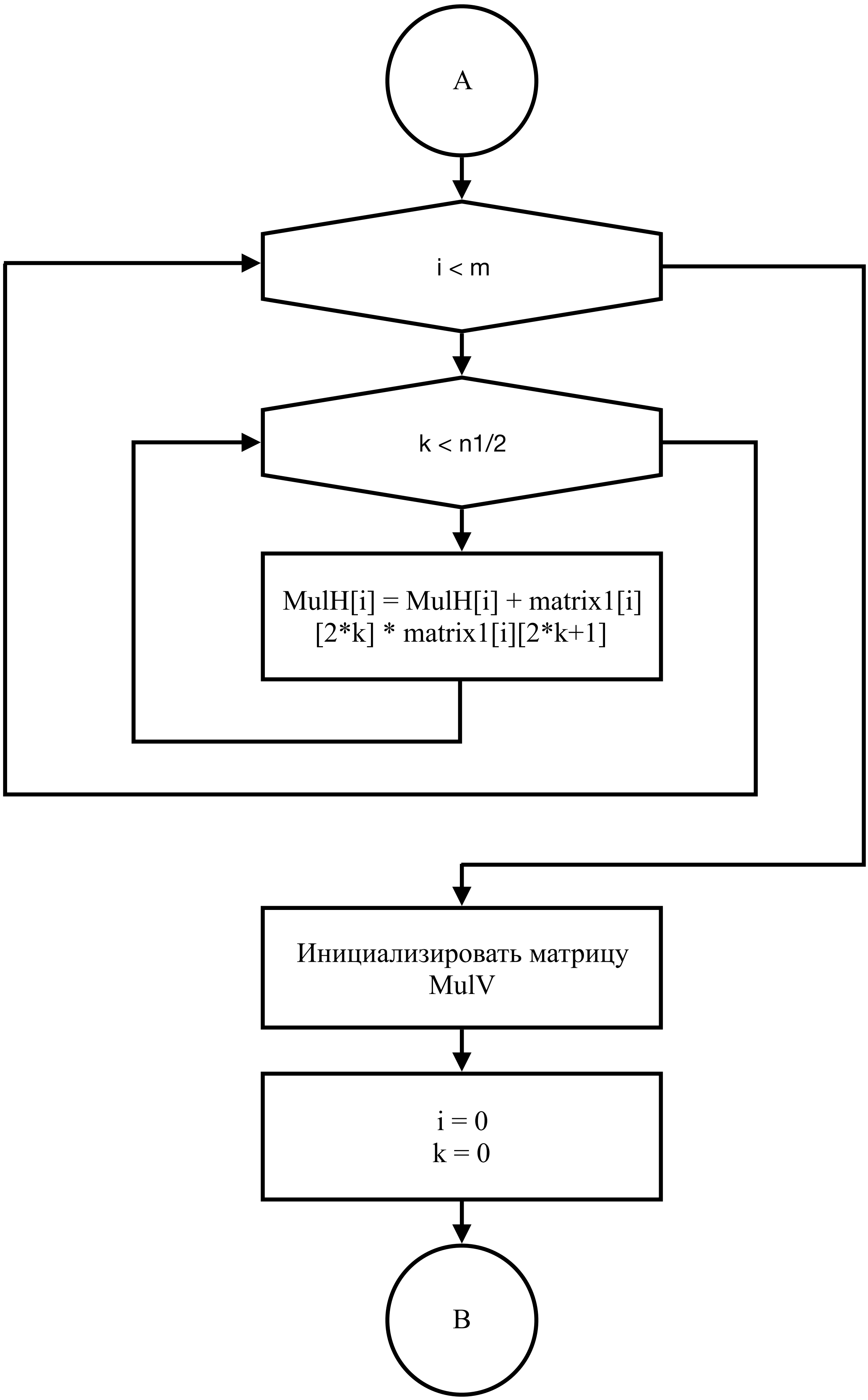


## 2.1.2 Схема алгоритма Винограда

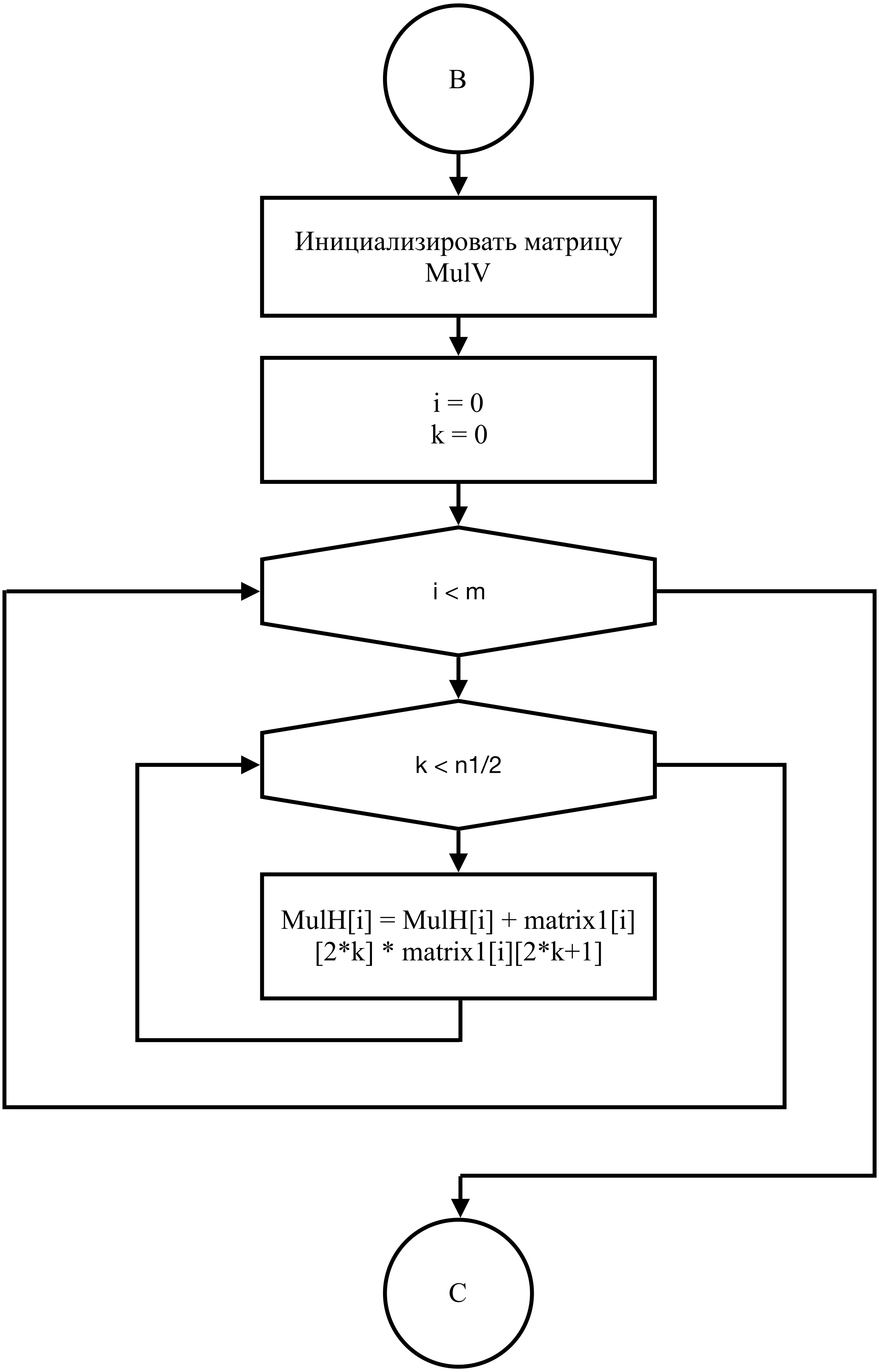
Входные данные:

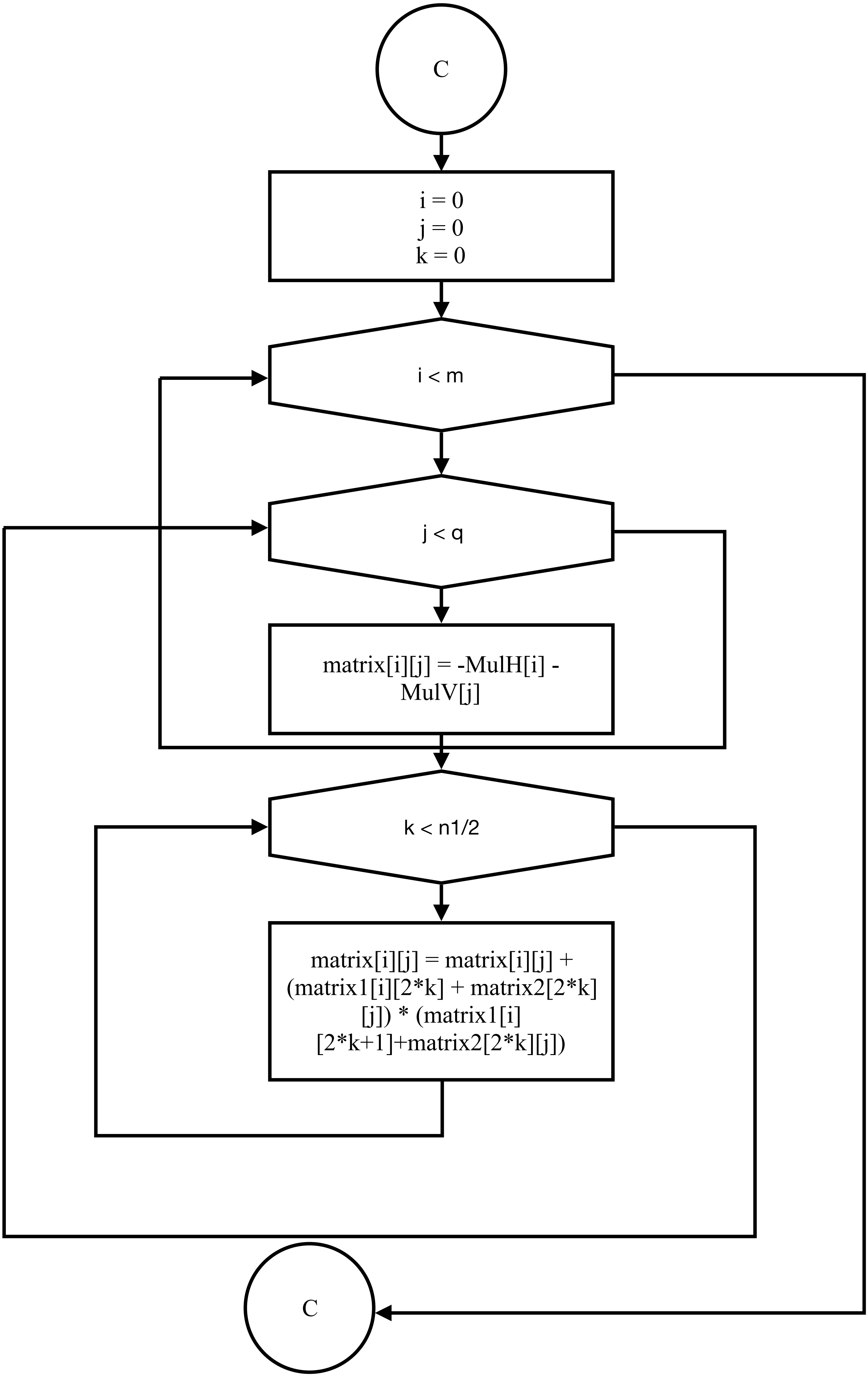
matrix1 - первая матрица
m - количество строк первой матрицы
n1 - количество столбцов первой матрицы
matrix2 - вторая матрица
n2 - количество строк второй матрицы
q - количество столбцов второй матрицы

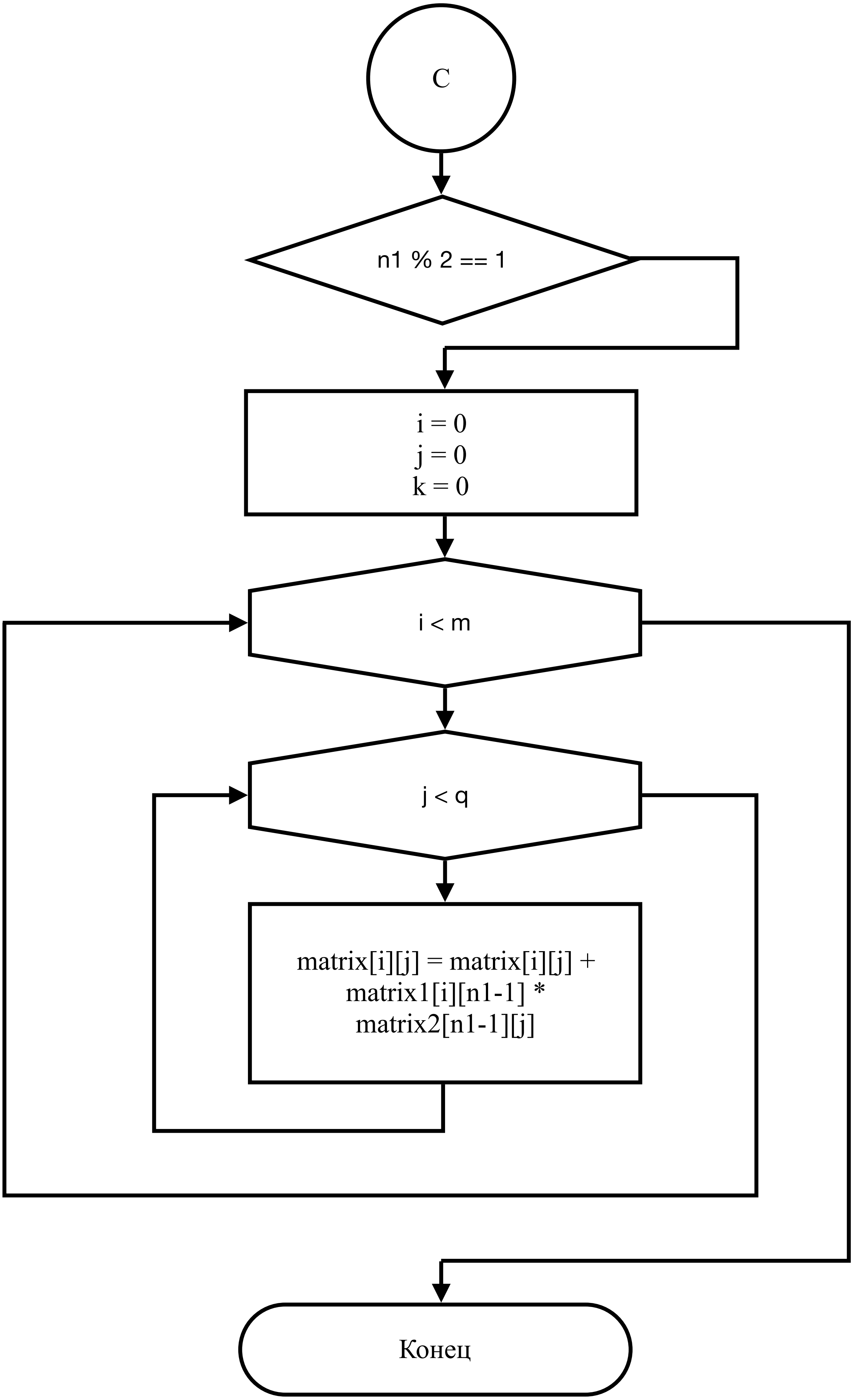




A
i < m
k < n1/2
MulH[i] = MulH[i] + matrix1[i][2*k] * matrix1[i][2*k+1]
Инициализировать матрицу MulV
i = 0
k = 0
B


B
Инициализировать матрицу MulV
i = 0
k = 0
i < m
k < n1/2
MulH[i] = MulH[i] + matrix1[i][2*k] * matrix1[i][2*k+1]
C


C
i = 0
j = 0
k = 0
i < m
j < q
matrix[i][j] = -MulH[i] - MulV[j]
k < n1/2
matrix[i][j] = matrix[i][j] + (matrix1[i][2*k] + matrix2[2*k][j]) * (matrix1[i][2*k+1]+matrix2[2*k][j])
C


С
n1 % 2 == 1
i = 0
j = 0
k = 0
i < m
j < q
matrix[i][j] = matrix[i][j] + matrix1[i][n1-1] * matrix2[n1-1][j]
Конец

### 2.1.3 Оптимизированный алгоритм Винограда

Оптимизированный алгоритм Винограда схематично представляет собой обычный алгоритм Винограда,
за исключением типовых оптимизаций, таких как:

1. Вычисление и сохранение $\frac{n1}{2}$ заранее.
2. Использование битового сдвига, вместо деление на 2
3. Замена $a = a + \ldots$ на $a += \ldots$

## 2.2 Сравнительный анализ алгоритмов

Для вычисления произведения двух матриц каждая строка первой почленно умножается на каждый столбец второй. Затем подсчитывается сумма таких произведений и записывается в соответствующую клетку результата.
Таким образом мы получаем сложность

$$13MNQ + 4MQ + 4M + 2 \sim 13MNQ$$ , где

M количество строк первой матрицы, N количество столбцов и количество строк второй матрицы, и Q - количество строк второй матрицы.

При описании алгоритма Винограда мы отметили, что при умножении двух вектором, мы получаем следующее равенство

$$V \cdot W = v1w1 + v2w2 + v3w3 + v4w4$$

После преобразований:

$$V \cdot W = (v1 + w2)(v2 + w1) + (v3 + w1)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4$$

Можно проверить эквивалентность двух последних выражений. Кажется, что второе выражение 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.

Таким образом сложность алгоритма Винограда:

$$\frac{15}{2}MN + 9M + \frac{15}{2}QN + 5Q + 6 + \frac{26}{2}MNQ + 12MQ \sim 13\mathrm{MNQ}$$

Теоретически мы получили ту же сложность, что и при стандартном умножении, но если оптимизировать алгоритм Винограда путем замены некоторых операций и предварительно сохраняя значения, чтобы не вычислять их повторно в цикле, типовые оптимизации описаны в разделе 2.1.3.

Таким образом мы получаем следующую сложность:

$$\frac{10}{2}MN + 4M + 2 + \frac{10}{2}MN + 4M + 2 + \frac{18}{2}MNQ + 12MQ + 4M + 2 \sim 9MNQ$$

Что значительнее превосходит алгоритм умножения матриц

# 3. Технологическая часть

В данном разделе будет представлено описание используемого языка программирование, а также будет показан листинг кода функций, работающих согласно указанным выше алгоритмам.

## 3. 1 Требования к программному обеспечению

Данная программа была реализована на языке C++ компилятор для которого поддерживается многими операционными системами

Компилятор: g++

## 3. 2 Средства реализации

Программа была реализована на операционной системе MacOS в среде разработки Xcode

## 3. 3 Листинг кода

## 3. 3. 1 Листинг алгоритма умножения матриц

matrix1 - первая матрица
matrix2 - вторая матрица
matrix - результирующая матрица

```
int multiplyMatrix(const std::vector<std::vector<int> > matrix1, const int M, const
int N1, const std::vector<std::vector<int> > matrix2, const int N2, const int Q,
std::vector<std::vector<int> > *matrix) // сложность 13MNQ + 4MQ + 4M + 2
{
  if( N1 != N2 ) return INCORRECT_SIZES;

    for( int i = 0; i < M; ++i )
    {
        for( int j = 0; j < Q; ++j )
        {
            for( int k = 0; k < N1; ++k )
            {
                (*matrix)[i][j] = (*matrix)[i][j] + matrix1[i][k] * matrix2[k][j];
            }
        }
    }

    return 0;
}
```

## 3. 3. 2 Листинг алгоритма Винограда

matrix1 - первая матрица
matrix2 - вторая матрица
matrix - результирующая матрица

```
int vinogradMultiplyMartix(const std::vector<std::vector<int> > matrix1, const int
M, const int N1, const std::vector<std::vector<int> > matrix2, const int N2, const int
Q, std::vector<std::vector<int> > *matrix)
{
        if( N1 != N2 ) return INCORRECT_SIZES;

        std::vector<int> MulH(M);
```

```
	for(int i = 0; i < M; ++i) { MulH[i] = 0; }

	for( int i = 0; i < M; ++i )
	{
		for( int k = 0; k < N1/2; ++k )
		{
			MulH[i] = MulH[i] + matrix1[i][2*k] * matrix1[i][2*k+1];
		}
	}

	std::vector<int> MulV(Q);
	for(int i = 0; i < Q; ++i) MulV[i] = 0;

	for( int i = 0; i < Q; ++i )
	{
		for( int k = 0; k < N1/2; ++k )
		{
			MulV[i] = MulV[i] + matrix2[2*k][i] * matrix2[2*k+1][i];
		}
	}

	for( int i = 0; i < M; ++i )
	{
		for( int j = 0; j < Q; ++j )
		{
			(*matrix)[i][j] = -MulH[i] - MulV[j];
			for( int k = 0; k < N1/2; ++k )
			{
				(*matrix)[i][j] = (*matrix)[i][j] + ( matrix1[i][2*k] +
matrix2[2*k+1][j] ) *
				( matrix1[i][2*k+1] + matrix2[2*k][j] );
			}
		}
	}

	if( N1 % 2 == 1 )
	{
		for( int i = 0; i < M; ++i )
		{
			for( int j = 0; j < Q; ++j )
```

```
                {
                        (*matrix)[i][j] = (*matrix)[i][j] + matrix1[i][N1-1] *
matrix2[N1-1][j];
                }
            }
        }

        return 0;
}
```

## 3. 3. 2 Листинг алгоритма оптимизированного алгоритма Винограда

matrix1 - первая матрица
matrix2 - вторая матрица
matrix - результирующая матрица

```
int optimizeVinogradMultiplyMartix(const std::vector<std::vector<int> > matrix1,
const int M, const int N1, const std::vector<std::vector<int> > matrix2, const int N2,
const int Q, std::vector<std::vector<int> > *matrix)
{
        if( N1 != N2 ) return INCORRECT_SIZES;

        int half_n = N1/2;

        std::vector<int> MulH(M);
        for(int i = 0; i < M; ++i) MulH[i] = 0;

        for( int i = 0; i < M; ++i )
        {
                for( int k = 0; k < half_n; ++k )
                {
                        MulH[i] += matrix1[i][(k<<1)] * matrix1[i][(k<<1)+1];
                }
        }

        std::vector<int> MulV(Q);
        for(int i = 0; i < Q; ++i) MulV[i] = 0;
```

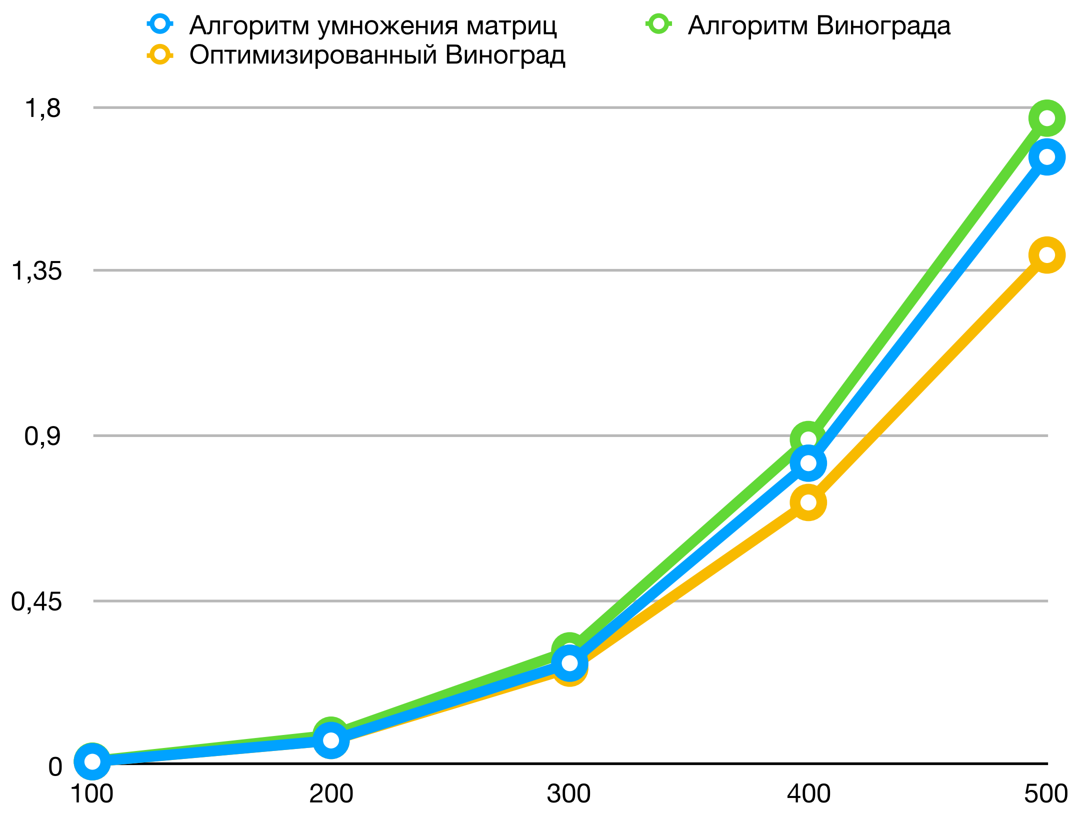
```
	for( int i = 0; i < Q; ++i )
	{
		for( int k = 0; k < half_n; ++k )
		{
			MulV[i] += matrix2[k<<1][i] * matrix2[(k<<1)+1][i];
		}
	}

	for( int i = 0; i < M; ++i )
	{
		for( int j = 0; j < Q; ++j )
		{
			(*matrix)[i][j] = -MulH[i] - MulV[j];
			for( int k = 0; k < half_n; ++k )
			{
				(*matrix)[i][j] += ( matrix1[i][k<<1] + matrix2[(k<<2)+1]
[j] ) *
				( matrix1[i][(k<<2)+1] + matrix2[k<<2][j] );
			}
		}
	}

	if( N1 % 2 == 1 )
	{
		for( int i = 0; i < M; ++i )
		{
			for( int j = 0; j < Q; ++j )
			{
				(*matrix)[i][j] += matrix1[i][N1-1] * matrix2[N1-1][j];
			}
		}
	}

	return 0;
}
```

## 4. Экспериментальная часть

В данном разделе будут представлено сравнение временных характеристик алгоритмов.

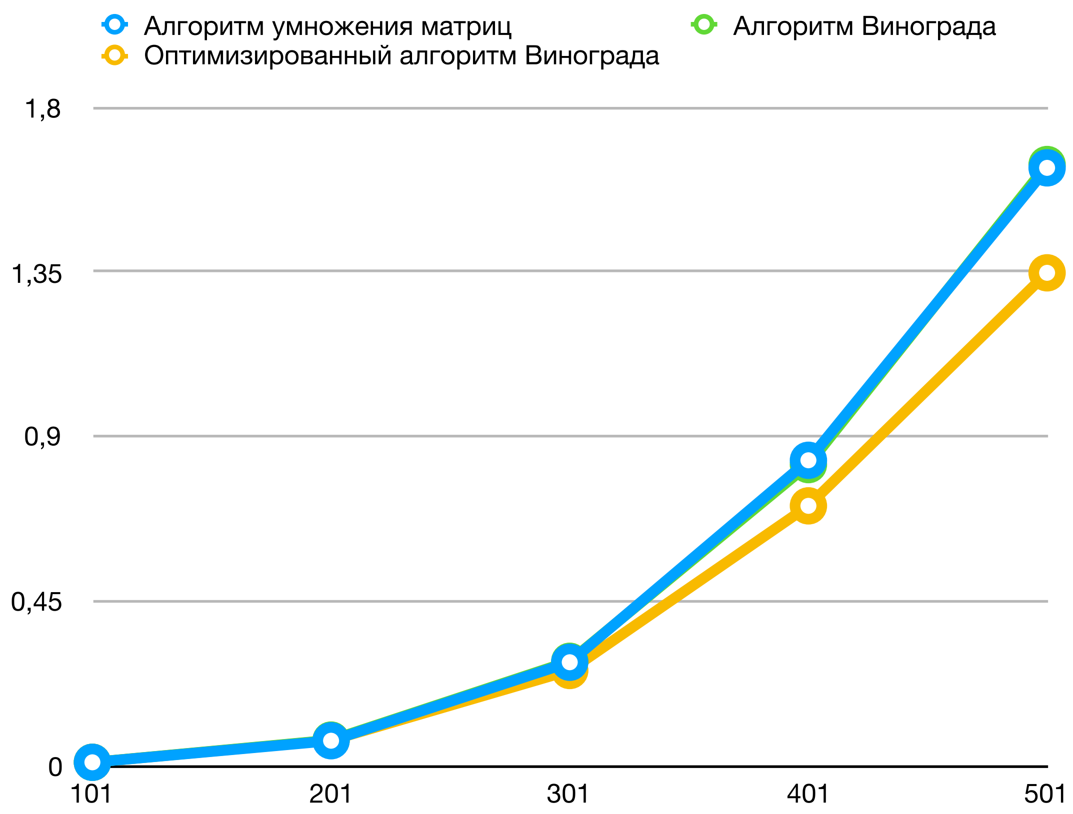
## 4.1 Постановка эксперимента

Первый эксперимент производится для лучшего случая на матрицах с размерами от 100 x 100 до 500 x 500 с шагом 100, так как уже на размере 500 приходилось ожидать около минуты. Каждый эксперимент тем не менее был повторен 100 раз для усреднения значений




Эксперимент подтвердил ожидания: не оптимизированная версия Винограда работает действительно почти так же как и алгоритм умножения матриц

Второй эксперимент производится для худшего случая, когда поданы матрицы с нечетными размерами от 101 x 101 до 501 x 501 с шагом 100




Как видно из результатов поставленных экспериментов, оптимизированный алгоритм Винограда во всех ситуациях превосходит алгоритм умножения матриц.

## Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы умножения матриц, а именно: алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда
Был проведен анализ данных алгоритмов. Оптимизированный алгоритм Винограда показал себя как наиболее эффективный для большинства случаев, что сходится с ожидаемым результатом.