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»

Отчет по лабораторной работе № 2 по курсу «Анализ алгоритмов»

Тема Алгоритмы умножения матриц

Студент Жаворонкова А. А.

Группа ИУ7-56Б

Оценка (баллы)

Преподаватель Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение **4**

1 Аналитическая часть **6**

1.1 Классический алгоритм умножения матриц 6

1.2 Алгоритм Винограда 7

1.3 Оптимизированный алгоритм Винограда 8

1.4 Вывод 8

2 Конструкторская часть **9**

2.1 Разработка алгоритмов 9

2.2 Модель вычислений 16

2.3 Трудоемкость алгоритмов 17

2.3.1 Классический алгоритм 17

2.3.2 Алгоритм Винограда 18

2.3.3 Оптимизированный алгоритм Винограда 20

2.4 Классы эквивалентности тестирования 22

2.5 Использование памяти 23

2.6 Вывод 24

3 Технологическая часть **25**

3.1 Средства реализации 25

3.2 Описание используемых типов данных 25

3.3 Сведения о модулях программы 25

3.4 Реализация алгоритмов 26

3.5 Функциональные тесты 28

3.6 Вывод 30

4 Исследовательская часть **31**

4.1 Технические характеристики 31

4.2 Демонстрация работы программы 31

4.3 Время выполнения алгоритмов 35

4.4 Вывод 38

Заключение **40**

Список используемых источников **41**

Введение

В математике и программировании часто приходится прибегать к использованию матриц. Существует огромное количество областей их применения в этих сферах. Например, матрицы активно используются при выводе различных формул в физике:

— градиент;

— дивергенция;

— ротор.

Также часто применяются и операции над матрицами — сложение, возведение в степень, умножение. При различных задачах размеры матрицы могут достигать больших значений. Поэтому оптимизация операций работы над матрицами является важной задачей в программировании. Об оптимизации операции умножения пойдет речь в данной лабораторной работе.

Целью данной работы является описание, реализация и исследование алгоритмов умножения матриц — классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. Для достижения поставленной цели необходимо выполнить следующие задачи:

— описать классический алгоритм умножения матриц, алгоритмы Винограда и его оптимизации;

— реализовать классический алгоритм умножения матриц, алгоритмы Винограда и его оптимизации;

— провести тестирование по методу черного ящика для реализаций классического алгоритма умножения матриц, алгоритмов Винограда и его оптимизации;

— провести сравнительный анализ по времени и по памяти реализаций классического алгоритма умножения матриц, алгоритмов Винограда и его оптимизации;

— описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы умножения матриц: классический, Винограда и его оптимизации.

1.1 Классический алгоритм умножения матриц

Произведением матриц A и B называется матрица C такая, что число строк и столбцов матрицы C равно количеству строк матрицы A и столбцов матрицы B соответственно.

Необходимым условием умножения двух матриц является равенство количества столбцов первой матрицы количеству строк второй матрицы.

Пусть даны матрицы A $[a \times b]$ и B $[b \times d]$:

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

$$B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{pmatrix}, \tag{1.2}$$

тогда произведением матриц A и B будет считаться матрица C [a x d]:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}, \tag{1.3}$$

где

$$C_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}, \;\; i = 1, 2, ..., a; \;\; j = 1, 2, ..., d. \tag{1.4}$$

Классический алгоритм [3] умножения двух матриц работает по формуле (1.4).

1.2 Алгоритм Винограда

Пусть умножаются матрицы A, содержащая M строк и Q столбцов, и B, содержащая Q строк и N столбцов. Результатом умножения будет матрица C с M стоками и N столбцами.

Каждый элемент C_{ij} матрицы C вычисляется как произведение строки i первой матрицы на столбец j второй.

$$C_{ij} = \begin{bmatrix} a_{i1} & a_{i2} & a_{i3} & a_{i4} \end{bmatrix} \cdot \begin{bmatrix} b_{1j} \\ b_{2j} \\ b_{3j} \\ b_{4j} \end{bmatrix} = a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + a_{i4}b_{4j} \quad (1.5)$$

Результат умножения представим в виде:

$$\begin{aligned} & a_{i1}b_{1j} + a_{i2}b_{2j} + a_{i3}b_{3j} + a_{i4}b_{4j} = \\ & (a_{i1} + b_{2j})(a_{i2} + b_{1j}) + (a_{i3} + b_{4j})(a_{i4} + b_{3j}) - \\ & - a_{i1}a_{i2} - a_{i3}a_{i4} - b_{1j}b_{2j} - b_{3j}b_{4j} \end{aligned} \quad (1.6)$$

При этом слагаемые

$$-a_{i1}a_{i2} - a_{i3}a_{i4} - b_{1j}b_{2j} - b_{3j}b_{4j}$$

могут быть вычислены заранее и использованы повторно.

Таким образом, алгоритм Винограда [4] снижает долю операций умножения в произведении матриц.

1.3 Оптимизированный алгоритм Винограда

Оптимизация будет включать в себя:

— замену операций вида $x = x + k$ на $x+ = k$;

— замену умножения на 2 на побитовый сдвиг влево;

— предвычисление некоторых слагаемых для алгоритма.

1.4 Вывод

В данном разделе было дано математическое описание классического алгоритма умножения матриц, алгоритма Винограда и описана суть его оптимизации.

К разрабатываемой программе предъявляются следующие требования.

1. Программа должна предоставлять функциональность расчета произведения матриц классическим алгоритмом, алгоритмом Винограда и его оптимизации.

2. Реализуемое ПО будет работать в двух режимах — пользовательском, в котором можно выбрать алгоритм и вывести для него рассчитанное значение, а также экспериментальном режиме, в котором можно произвести сравнение реализаций алгоритмов по времени работы на различных входных данных.

3. В первом режиме в качестве входных данных в программу будет подаваться две матрицы, также реализовано меню для вызова алгоритмов и замеров времени. Программа должна корректно обрабатывать случай ввода матриц с размерами, для которых умножение невозможно.

4. Во втором режиме будет происходить измерение процессорного времени работы программы, будут построены зависимости времени работы от совпадающих размеров квадратных матриц. Входные матрицы будут сгенерированы автоматически для заданного совпадающего размера матрицы.

2 Конструкторская часть

В этом разделе будут представлены схемы алгоритмов классического умножения матриц, Винограда и его оптимизации.

2.1 Разработка алгоритмов

На рисунках 2.2 — 2.7 представлены схемы алгоритмов классического умножения матриц, Винограда и его оптимизации.

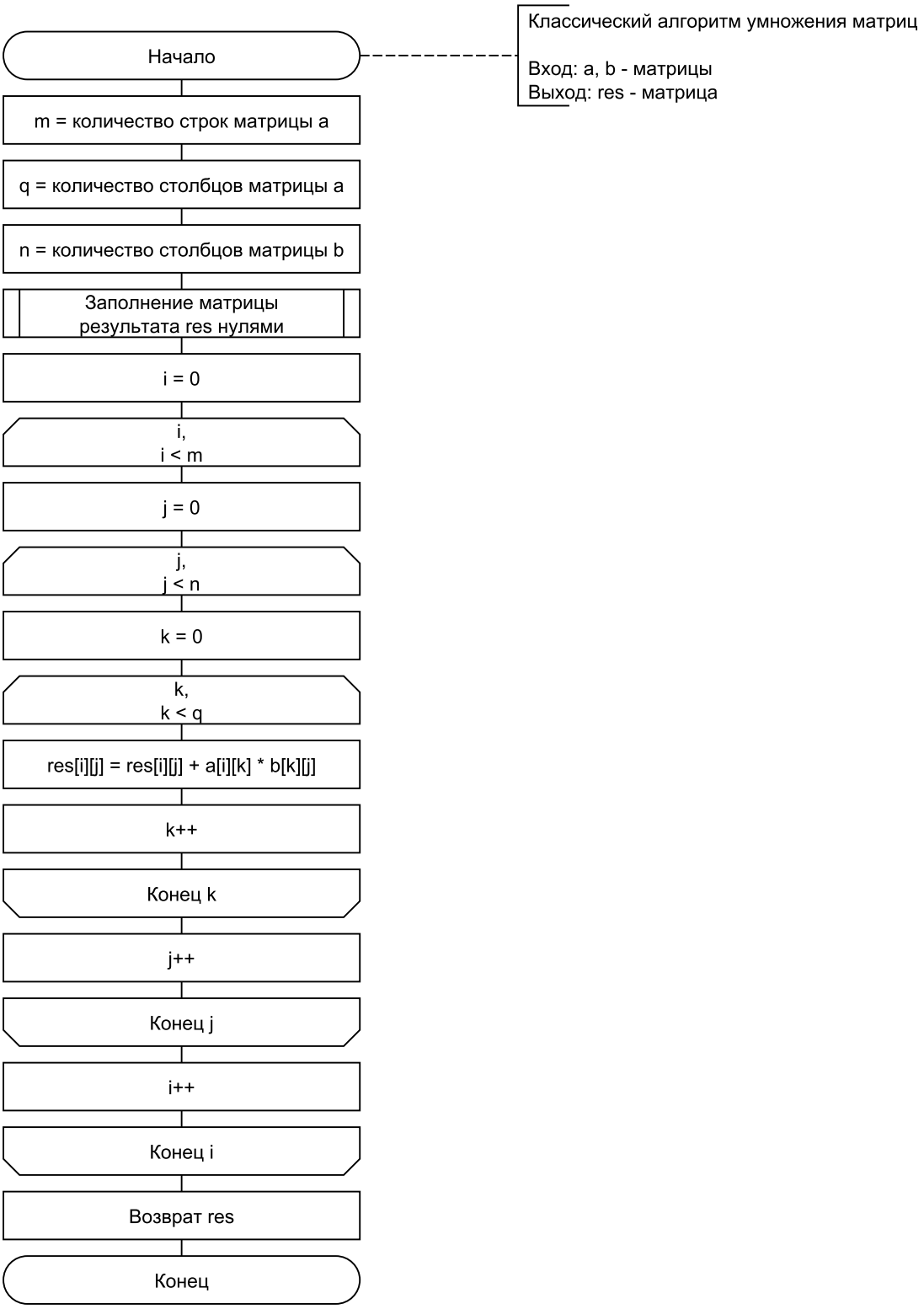

Рисунок 2.1 – Схема алгоритма классического умножения матриц

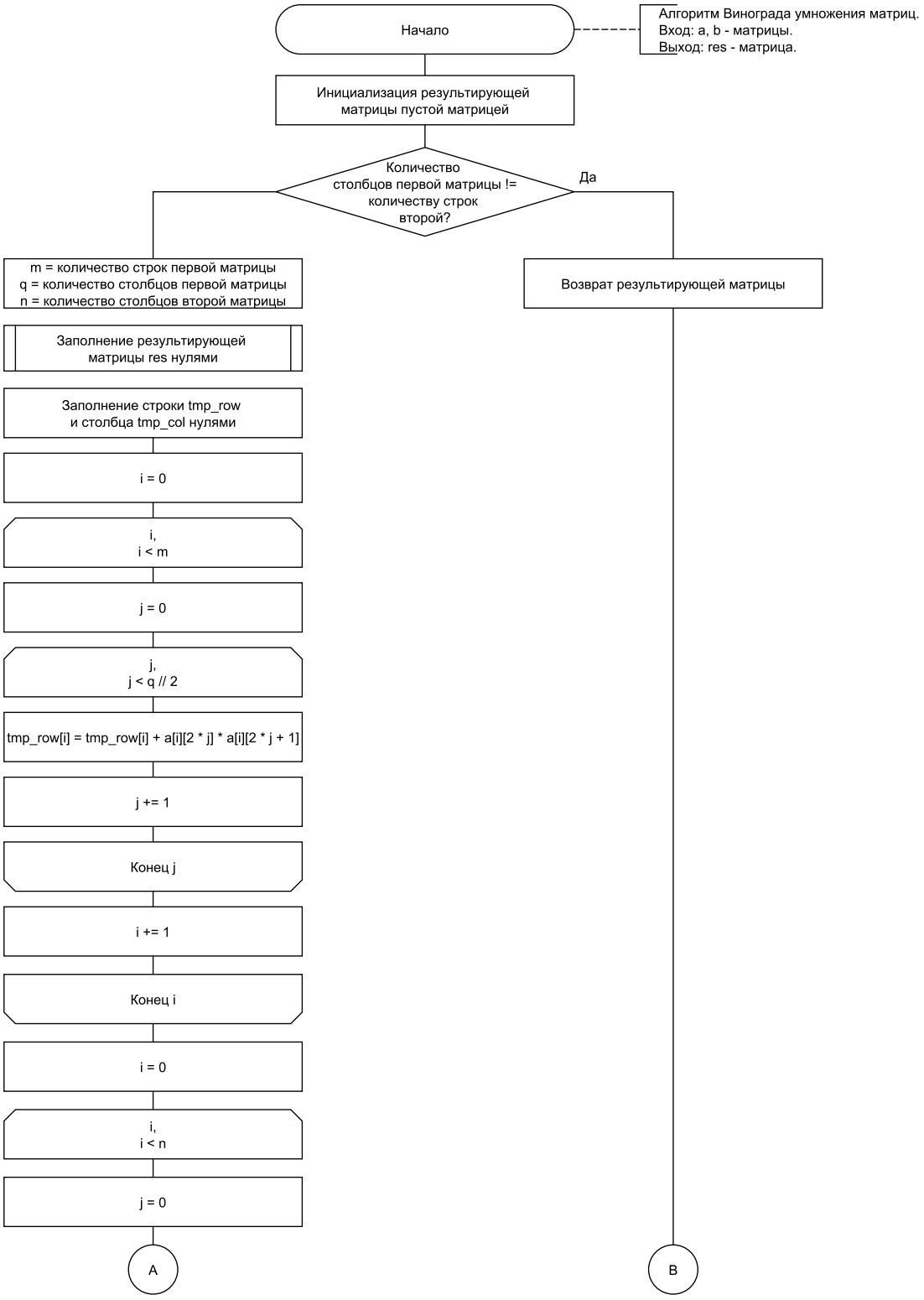

Рисунок 2.2 – Схема алгоритма Винограда (часть 1)

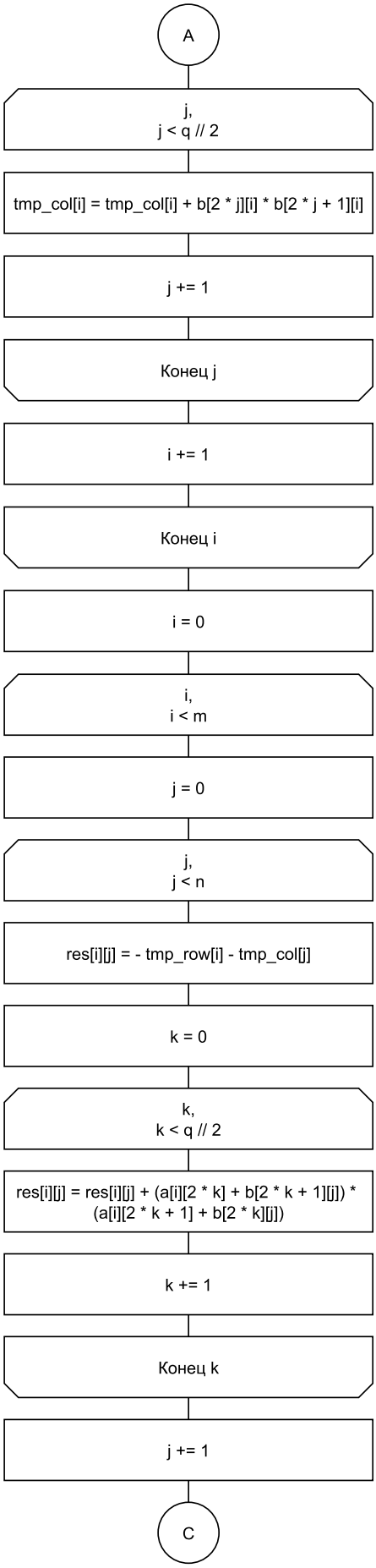

Рисунок 2.3 – Схема алгоритма Винограда (часть 2)

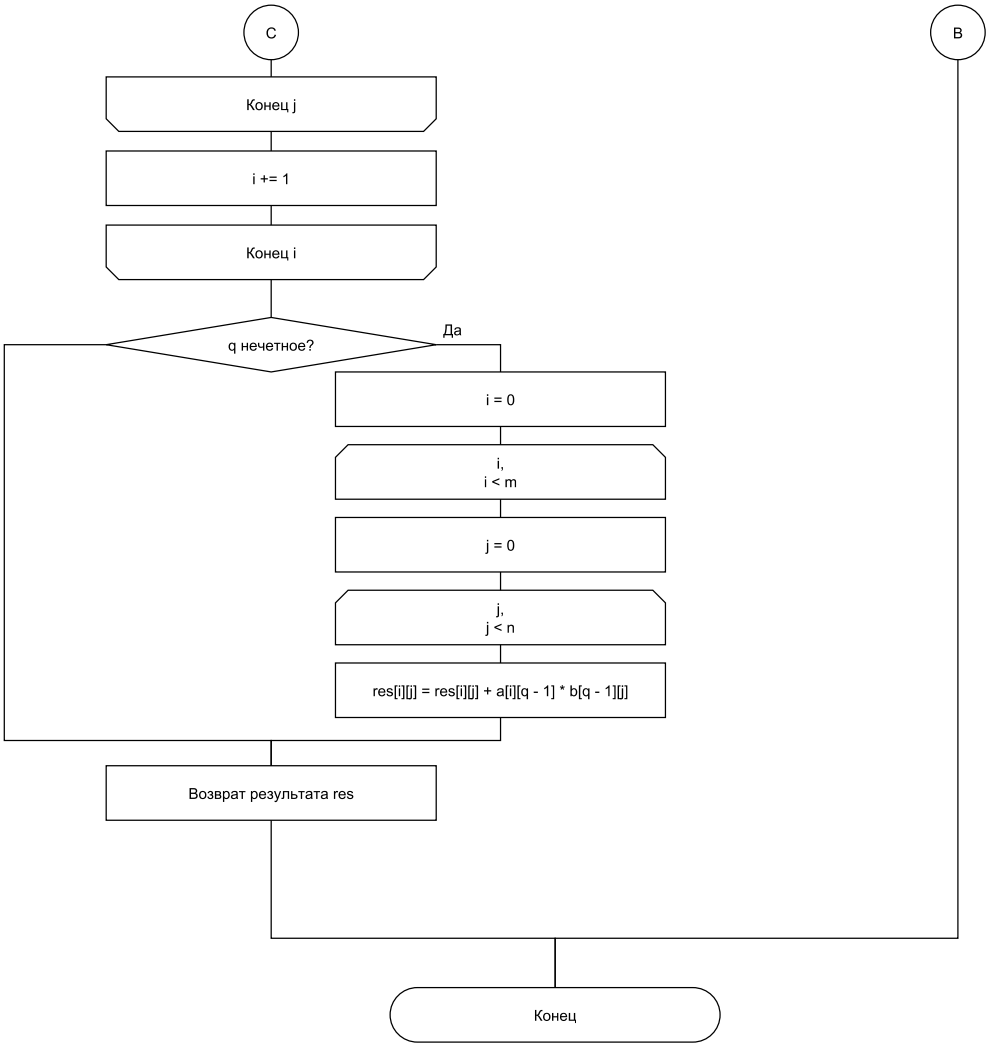

Рисунок 2.4 – Схема алгоритма Винограда (часть 3)

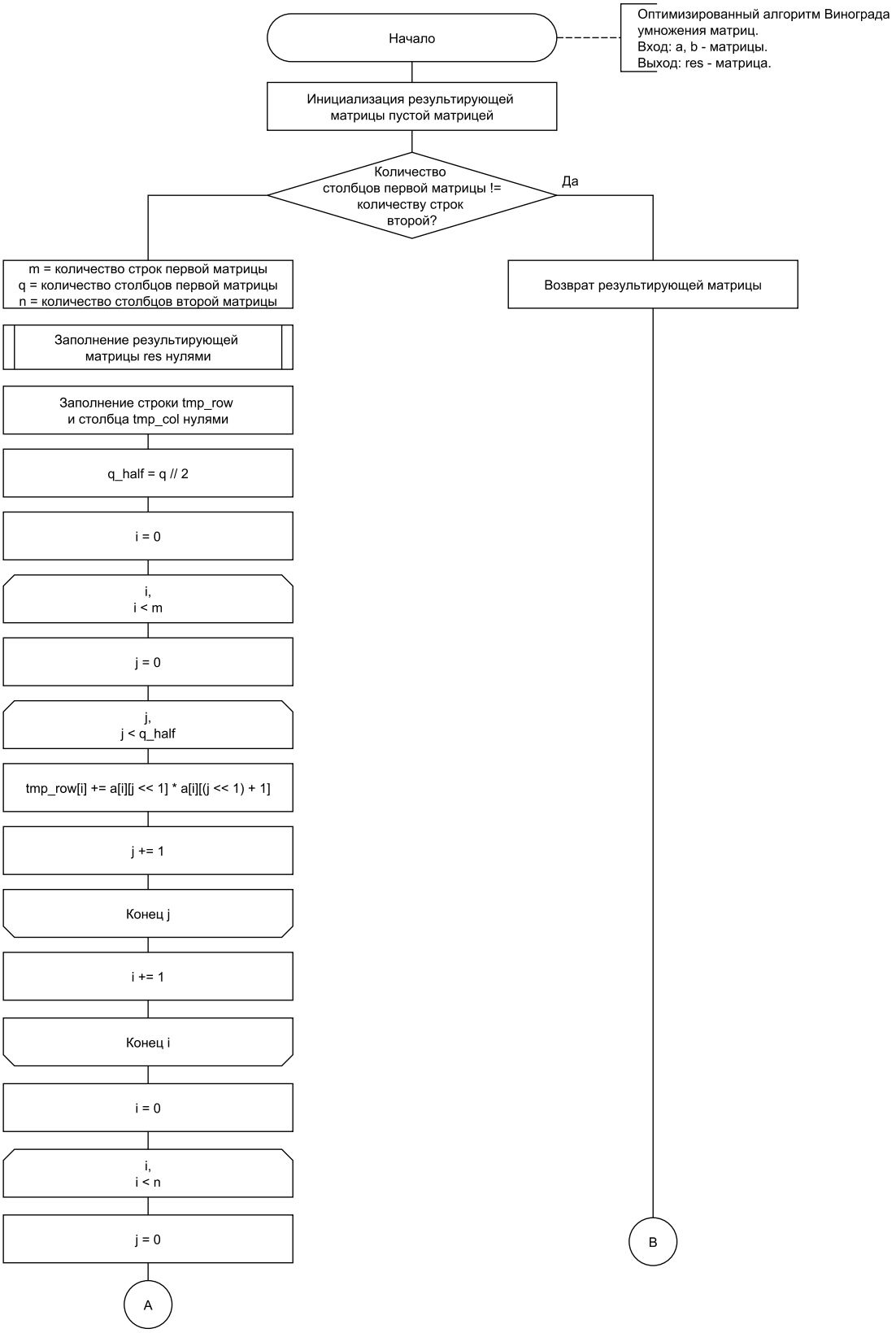

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда (часть 1)

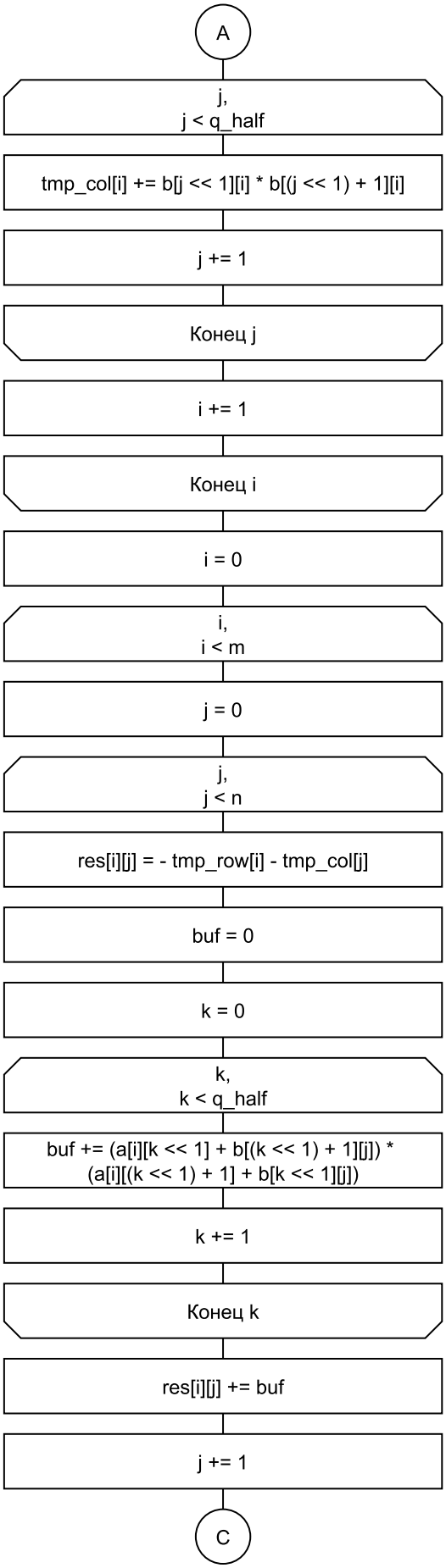

Рисунок 2.6 – Схема оптимизированного алгоритма Винограда (часть 2)

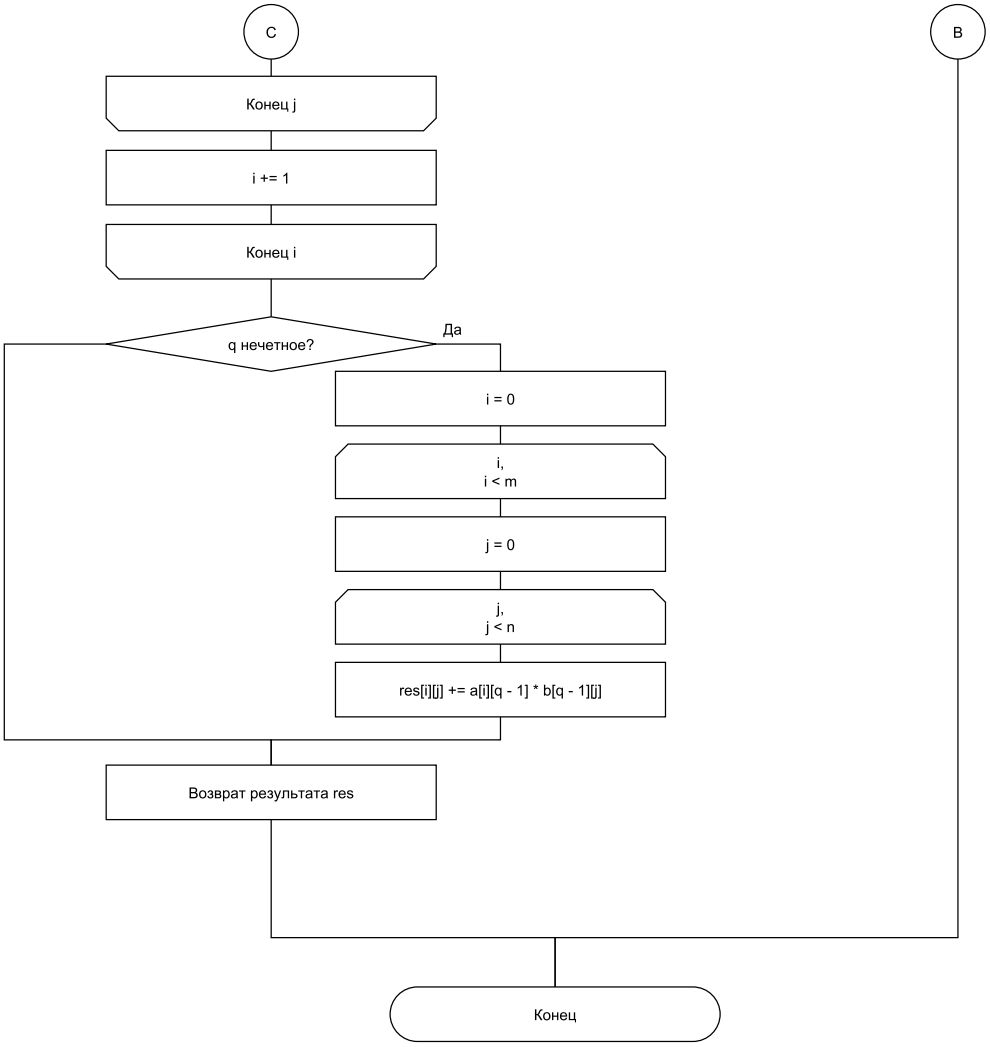

Рисунок 2.7 – Схема оптимизированного алгоритма Винограда (часть 3)

2.2 Модель вычислений

1. Трудоемкость базовых операций.

   Следующие операторы имеют трудоемкость 1:

   $$+, -, + =, - =, =, ==, ! =, >=, <=, >, <,$$

   $$>>, <<, [], \&, |, \&\&, ||, ++, --$$

   Следующие операторы имеют трудоемкость 2:

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

2. Условный оператор.

   Для конструкций вида:

```
1 if (условие)
2 {
3     Блок 1;
4 }
5 else
6 {
7     Блок 2;
8 }
```

   Пусть трудоемкость блока 1 — f_1, блока 2 — f_2. Пусть также трудоемкость условного перехода — 0.

   Тогда трудоемкость условного оператора:

$$f_{if} = f_{\text{вычисления условия}} + \begin{bmatrix} \min(f_1, f_2), \text{лучший случай} \\ \max(f_1, f_2), \text{худший случай} \end{bmatrix} \tag{2.1}$$

3. Трудоемкость циклов.

   Трудоемкость циклов вычисляется по следующей формуле:

$$\begin{aligned} f_{\text{цикла}} = & f_{\text{инициализации}} + f_{\text{сравнения}} + \\ & + M_{\text{шагов}} \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \end{aligned} \tag{2.2}$$

2.3 Трудоемкость алгоритмов

Рассчитаем трудоемкость алгоритмов умножения матриц.

2.3.1 Классический алгоритм

Трудоемкость классического алгоритма умножения матриц:

$$
\begin{aligned}
f_{\text{классический}} = {} & f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{\text{инициализации}} + f_{\text{сравнения}} + Q \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{[]} + f_{[]} + f_{=} + f_{[]} + f_{[]} + f_{+} + f_{[]} + f_{[]} + f_{*} + f_{[]} + f_{[]}))) = \\
& = 1 + 1 + M \cdot (1 + 1 + 1 + 1 + N \cdot (1 + 1 + 1 + 1 + Q \cdot \\
& \cdot (1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 2 + 1 + 1))) = \\
& = 14MNQ + 4MN + 4M + 2
\end{aligned}
\tag{2.3}
$$

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда будет складываться из следующих составляющих:

— вычисление слагаемых $-a_{i1}a_{i2} - a_{i3}a_{i4}$.

$$
\begin{aligned}
f_I = {} & f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{\text{инициализации}} + f_{\text{сравнения}} + Q/2 \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{[]} + f_{=} + f_{[]} + f_{+} + f_{[]} + f_{*} + f_{[]} + f_{*} + f_{[]} + f_{*} + \\
& + f_{+} + f_{[]})) = \\
& = 1 + 1 + M \cdot (1 + 1 + 1 + 3 + Q/2 \cdot (1 + 3 + 1 + 1 + 1 + 1 + \\
& + 1 + 2 + 1 + 2 + 1 + 2 + 1 + 1)) = \\
& = \frac{19MQ}{2} + 6M + 2
\end{aligned}
\tag{2.4}
$$

— вычисление слагаемых $-b_{1j}b_{2j} - b_{3j}b_{4j}$.

Аналогично получим:

$$
f_{II} = \frac{19NQ}{2} + 6N + 2 \tag{2.5}
$$

— заполнение результирующей матрицы.

$$
\begin{aligned}
f_{III} = {} & f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{[]} + f_{[]} + f_{=} + f_{-} + f_{[]} + f_{-} + f_{[]} + f_{\text{инициализации}} + \\
& + f_{\text{сравнения}} + Q/2 \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + f_{[]} + f_{[]} + f_{=} + \\
& + f_{[]} + f_{[]} + f_{+} + f_{[]} + f_{*} + f_{[]} + f_{+} + f_{*} + f_{+} + f_{[]} + f_{[]} + \\
& + f_{*} + f_{[]} + f_{*} + f_{+} + f_{[]} + f_{+} + f_{*} + f_{[]} + f_{[]}))) = \\
& = 1 + 1 + M \cdot (1 + 1 + 1 + 1 + N \cdot (1 + 1 + 1 + 1 + \\
& + 1 + 1 + 1 + 1 + 1 + 1 + 3 + Q/2 \cdot (1 + 3 + 1 + 1 + 1 + \\
& + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 2 + 1 + 2 + 1 + \\
& + 1 + 1 + 2 + 1 + 1))) = \\
& = 16MNQ + 13MN + 4M + 2
\end{aligned}
\tag{2.6}
$$

— учет нечетного Q.

$$
f_{IV} = f_{\text{условия}} + \begin{bmatrix} \min(f_1, f_2), \text{лучший случай} \\ \max(f_1, f_2), \text{худший случай} \end{bmatrix}
\tag{2.7}
$$

$$
\begin{aligned}
f_{\text{условия}} &= f_{\%} + f_{==} = \\
&= 2 + 1 = 3
\end{aligned}
\tag{2.8}
$$

$$
\begin{aligned}
f_1 = {} & f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{[]} + f_{[]} + f_{=} + f_{[]} + f_{[]} + f_{+} + f_{[]} + f_{-} + f_{[]} + f_{*} + f_{-} + \\
& + f_{[]} + f_{[]} = \\
& = 1 + 1 + M \cdot (1 + 1 + 1 + 1 + N \cdot (1 + 1 + 1 + 1 + 1 + 1 + 1 + \\
& + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 1)) = \\
& = 16MN + 4M + 2
\end{aligned}
\tag{2.9}
$$

$$
f_2 = 0
\tag{2.10}
$$

Тогда формула 2.7 примет вид:

$$f_{IV} = 3 + \begin{bmatrix} 0, \text{лучший случай: Q четное} \\ 16MN + 4M + 2, \text{худший случай: Q нечетное} \end{bmatrix} \tag{2.11}$$

Таким образом, трудоемкость алгоритма Винограда:

$$\begin{aligned} f_{\text{Виноград}} &= f_I + f_{II} + f_{III} + f_{IV} = \\ &= \frac{19MQ}{2} + 6M + 2 + \frac{19NQ}{2} + 6N + 2 + 16MNQ + \\ &+ 13MN + 4M + 2 + 3 + \\ &+ \begin{bmatrix} 0, \text{лучший случай: Q четное} \\ 16MN + 4M + 2, \text{худший случай: Q нечетное} \end{bmatrix} = \\ &= \frac{19MQ}{2} + \frac{19NQ}{2} + 16MNQ + 13MN + 10M + 6N + 9 \end{aligned} \tag{2.12}$$

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда будет складываться из следующих составляющих:

— предвычисление $q//2$.

$$f_{\text{пред}} = 2 \tag{2.13}$$

— вычисление слагаемых $-a_{i1}a_{i2} - a_{i3}a_{i4}$.

$$\begin{aligned} f_I &= f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\ &+ f_{\text{инициализации}} + f_{\text{сравнения}} + Q/2 \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\ &+ f_{[]} + f_{+=} + f_{[]} + f_{<<} + f_{[]} + f_{*} + f_{[]} + f_{<<} + \\ &+ f_{+} + f_{[]})) = \\ &= 1 + 1 + M \cdot (1 + 1 + 1 + 1 + Q/2 \cdot (1 + 1 + 1 + 1 + 1 + 1 + \\ &+ 1 + 2 + 1 + 1 + 1 + 1)) = \\ &= \frac{13MQ}{2} + 4M + 2 \end{aligned} \tag{2.14}$$

— вычисление слагаемых $-b_{1j}b_{2j} - b_{3j}b_{4j}$.

Аналогично получим:

$$f_{II} = \frac{13NQ}{2} + 4N + 2 \tag{2.15}$$

— заполнение результирующей матрицы.

$$\begin{aligned} f_{III} = {} & f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\ & + f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\ & + f_{[]} + f_{[]} + f_{=} + f_{-} + f_{[]} + f_{-} + f_{[]} + f_{=} + f_{\text{инициализации}} + \\ & + f_{\text{сравнения}} + Q/2 \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + f_{+=} + f_{[]} + \\ & + f_{<<} + f_{[]} + f_{+} + f_{<<} + f_{+} + f_{[]} + f_{[]} + f_{*} + f_{[]} + f_{<<} + \\ & + f_{+} + f_{[]} + f_{+} + f_{<<} + f_{[]} + f_{[]}))) = \\ & = 1 + 1 + M \cdot (1 + 1 + 1 + 1 + N \cdot (1 + 1 + 1 + 1 + 1 + 1 + \\ & + 1 + 1 + 1 + 1 + 1 + 1 + Q/2 \cdot (1 + 1 + 1 + 1 + 1 + 1 + 1 + \\ & + 1 + 1 + 1 + 1 + 2 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1))) = \\ & = \frac{21MNQ}{2} + 12MN + 4M + 2 \end{aligned} \tag{2.16}$$

— учет нечетного Q.

$$f_{IV} = f_{\text{условия}} + \begin{bmatrix} \min(f_1, f_2), \text{лучший случай} \\ \max(f_1, f_2), \text{худший случай} \end{bmatrix} \tag{2.17}$$

$$\begin{aligned} f_{\text{условия}} = f_{\%} + f_{==} = \\ = 2 + 1 = 3 \end{aligned} \tag{2.18}$$

$$
\begin{aligned}
f_1 = {} & f_{\text{инициализации}} + f_{\text{сравнения}} + M \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{\text{инициализации}} + f_{\text{сравнения}} + N \cdot (f_{\text{инкремента}} + f_{\text{сравнения}} + \\
& + f_{[]} + f_{[]} + f_{+=} + f_{[]} + f_{-} + f_{[]} + f_{*} + f_{-} + \\
& + f_{[]} + f_{[]} = \\
& = 1 + 1 + M \cdot (1 + 1 + 1 + 1 + N \cdot (1 + 1 + 1 + 1 + 1 + \\
& + 1 + 1 + 1 + 2 + 1 + 1 + 1)) = \\
& = 13MN + 4M + 2
\end{aligned}
\tag{2.19}
$$

$$
f_2 = 0 \tag{2.20}
$$

Тогда формула 2.17 примет вид:

$$
f_{IV} = 3 + \begin{bmatrix} 0, \text{лучший случай: Q четное} \\ 13MN + 4M + 2, \text{худший случай: Q нечетное} \end{bmatrix} \tag{2.21}
$$

Таким образом, трудоемкость оптимизированного алгоритма Винограда:

$$
\begin{aligned}
f_{\text{Виноград}} = {} & f_{\text{пред}} + f_I + f_{II} + f_{III} + f_{IV} = \\
& = 2 + \frac{13MQ}{2} + 4M + 2 + \frac{13NQ}{2} + 4N + 2 + \frac{21MNQ}{2} + \\
& + 12MN + 4M + 2 + 3 + \\
& + \begin{bmatrix} 0, \text{лучший случай: Q четное} \\ 13MN + 4M + 2, \text{худший случай: Q нечетное} \end{bmatrix} = \\
& = \frac{21MNQ}{2} + \frac{13MQ}{2} + \frac{13NQ}{2} + 12MN + 8M + 4N + 11 + \\
& + \begin{bmatrix} 0, \text{лучший случай: Q четное} \\ 13MN + 4M + 2, \text{худший случай: Q нечетное} \end{bmatrix}
\end{aligned}
\tag{2.22}
$$

2.4 Классы эквивалентности тестирования

Для тестирования выделены следующие классы эквивалентности:

1. одна из матриц пустая;
2. количество строк первой матрицы не равно количеству столбцов второй;
3. умножение матриц с размерами $[1 \times 1]$;
4. умножение квадратных матриц;
5. умножение прямоугольных матриц.

2.5 Использование памяти

1. Классический алгоритм:
   - две входные матрицы: $M \cdot Q \cdot sizeof(int) + Q \cdot N \cdot sizeof(int)$;
   - результирующая матрица: $M \cdot N \cdot sizeof(int)$;
   - дополнительные переменные: $3 \cdot sizeof(int)$.

   Таким образом, необходимая память:

$$sizeof(int) \cdot (M(Q+N) + QN + 3) \tag{2.23}$$

2. Алгоритм Винограда:
   - две входные матрицы: $M \cdot Q \cdot sizeof(int) + Q \cdot N \cdot sizeof(int)$;
   - результирующая матрица: $M \cdot N \cdot sizeof(int)$;
   - массив-строка для слагаемых: $M \cdot sizeof(int)$;
   - массив-столбец для слагаемых: $N \cdot sizeof(int)$;
   - дополнительные переменные: $3 \cdot sizeof(int)$.

   Таким образом, необходимая память:

$$sizeof(int) \cdot (M(Q+N+1) + N(Q+1) + 3) \tag{2.24}$$

3. Оптимизированный алгоритм Винограда:

— две входные матрицы: $M \cdot Q \cdot sizeof(int) + Q \cdot N \cdot sizeof(int)$;

— результирующая матрица: $M \cdot N \cdot sizeof(int)$;

— массив-строка для слагаемых: $M \cdot sizeof(int)$;

— массив-столбец для слагаемых: $N \cdot sizeof(int)$;

— дополнительные переменные: $5 \cdot sizeof(int)$.

Таким образом, необходимая память:

$$sizeof(int) \cdot (M(Q + N + 1) + N(Q + 1) + 5) \tag{2.25}$$

Вывод

Сравнивая формулы 2.24 и 2.25, можно сделать вывод, что оптимизированный алгоритм Винограда требует больше памяти, чем классический: затраты на $2 \cdot sizeof(int)$ байт больше. Но наименьшие затраты по памяти у классического алгоритма умножения матриц.

2.6 Вывод

В данном разделе были представлены схемы алгоритмов, рассматриваемых в лабораторной работе.

3 Технологическая часть

В данном разделе будут рассмотрены средства реализации, а также представлены листинги алгоритмов Винограда и его оптимизации.

3.1 Средства реализации

В данной работе для реализации был выбран язык программирования *Python* [2]. В текущей лабораторной работе требуется замерить процессорное время для выполняемой программы, а также построить графики. Все эти инструменты присутствуют в выбранном языке программирования.

Время работы было замерено с помощью функции *process_time* из библиотеки *time* [1].

3.2 Описание используемых типов данных

При реализации будут использованы следующие типы и структуры данных:

— тип *int* для количества строк и столбцов матрицы;

— *двумерный массив ячеек типа int* для входных матриц, а также матрицы результата.

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

— *main.py* — файл, содержащий весь служебный код;

— *algorythms.py* — файл, содержащий реализации алгоритмов.

3.4 Реализация алгоритмов

В листингах 3.1 — 3.3 представлены реализации классического алгоритма, алгоритмов Винограда и его оптимизации.

Листинг 3.1 – Классический алгоритм

```
def classic(a, b):
    m, q, n = len(a), len(a[0]), len(b[0])
    res = [[0] * n for _ in range(m)]

    for i in range(m):
        for j in range(n):
            for k in range(q):
                res[i][j] = res[i][j] + a[i][k] * b[k][j]
    return res
```

Листинг 3.2 – Алгоритм Винограда

```
def vinograd(a, b):
    res = []
    if len(a[0]) != len(b):
        print("Умножение матриц с такими размерами невозможно")
        return res

    m, q, n = len(a), len(a[0]), len(b[0])
    for _ in range(m):
        res.append([0] * n)
    tmp_row, tmp_col = [0] * m, [0] * n

    for i in range(m):
        for j in range(0, q // 2):
            tmp_row[i] = tmp_row[i] + a[i][2 * j] * a[i][2 * j + 1]

    for i in range(n):
        for j in range(0, q // 2):
            tmp_col[i] = tmp_col[i] + b[2 * j][i] * b[2 * j + 1][i]

    for i in range(m):
        for j in range(n):
            res[i][j] = - tmp_row[i] - tmp_col[j]
```

```
23              for k in range(0, q // 2):
24                  res[i][j] = res[i][j] + (a[i][2 * k] + b[2 * k +
                        1][j]) * (a[i][2 * k + 1] + b[2 * k][j])
25
26      if q % 2 == 1:
27          for i in range(m):
28              for j in range(n):
29                  res[i][j] = res[i][j] + a[i][q - 1] * b[q - 1][j]
30
31      return res
```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```
1  def vinograd_optimised(a, b):
2      res = []
3      if len(a[0]) != len(b):
4          print("Умножение матриц с такими размерами невозможно")
5          return res
6
7      m, q, n = len(a), len(a[0]), len(b[0])
8      for _ in range(m):
9          res.append([0] * n)
10     tmp_row, tmp_col = [0] * m, [0] * n
11     q_half = q // 2
12
13     for i in range(m):
14         for j in range(0, q_half):
15             tmp_row[i] += a[i][j << 1] * a[i][(j << 1) + 1]
16
17     for i in range(n):
18         for j in range(0, q_half):
19             tmp_col[i] += b[j << 1][i] * b[(j << 1) + 1][i]
20
21     for i in range(m):
22         for j in range(n):
23             res[i][j] = - tmp_row[i] - tmp_col[j]
24             buf = 0
25             for k in range(0, q_half):
26                 buf += (a[i][k << 1] + b[(k << 1) + 1][j]) *
                        (a[i][(k << 1) + 1] + b[k << 1][j])
27             res[i][j] += buf
28
```

```
29      if q % 2 == 1:
30          for i in range(m):
31              for j in range(n):
32                  res[i][j] += a[i][q - 1] * b[q - 1][j]
33
34      return res
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы Винограда и его оптимизации. Тесты *для всех алгоритмов* пройдены успешно.

Таблица 3.1 – Функциональные тесты

			Ожидаемый результат		
№	Матрица 1	Матрица 2	Классический	Виноград	Виноград (опт.)
1	$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке	Сообщение об ошибке
2	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	Сообщение об ошибке	Сообщение об ошибке	Сообщение об ошибке
3	$\begin{pmatrix} 2 \end{pmatrix}$	$\begin{pmatrix} 7 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$
4	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$
5	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
6	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$	$\begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$	$\begin{pmatrix} 22 & 28 \\ 49 & 64 \end{pmatrix}$
7	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$

3.6 Вывод

Были представлены реализации классического алгоритма умножения матриц, алгоритмов Винограда и его оптимизации, которые были описаны в предыдущем разделе. Также в данном разделе была приведена информация о выбранных средствах для разработки алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, а также проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее:

— операционная система: Windows 11, x64;

— оперативная память: 8 Гб;

— процессор: AMD Ryzen 5 5500U с видеокартой Radeon Graphics 2.10 ГГц.

Во время замеров времени ноутбук был нагружен только встроенными приложениями окружения.

4.2 Демонстрация работы программы

На рисунках 4.1 — 4.3 представлен результат работы программы.

```
Меню:
1) Классический алгоритм
2) Алгоритм Винограда
3) Алгоритм Винограда (оптимизированный)
4) Построить графики

0) Выйти

Выберите пункт меню: 1
Введите количество строк: 2
Введите количество столбцов: 3
Введите матрицу построчно:
1 2 3
4 5 6
Введите количество строк: 3
Введите количество столбцов: 2
Введите матрицу построчно:
1 2
3 4
5 6
+-------+-------+
|  22   |  28   |
|  49   |  64   |
+-------+-------+
```

Рисунок 4.1 – Пример работы программы (часть 1)

```
Меню:
1) Классический алгоритм
2) Алгоритм Винограда
3) Алгоритм Винограда (оптимизированный)
4) Построить графики

0) Выйти

Выберите пункт меню: 2
Введите количество строк: 2
Введите количество столбцов: 2
Введите матрицу построчно:
1 2
3 4
Введите количество строк: 2
Введите количество столбцов: 2
Введите матрицу построчно:
1 2
3 4
+-------+-------+
|   7   |  10   |
|  15   |  22   |
+-------+-------+
```

Рисунок 4.2 – Пример работы программы (часть 2)

```
Меню:
1) Классический алгоритм
2) Алгоритм Винограда
3) Алгоритм Винограда (оптимизированный)
4) Построить графики

0) Выйти

Выберите пункт меню: 3
Введите количество строк: 1
Введите количество столбцов: 2
Введите матрицу построчно:
1 2
Введите количество строк: 2
Введите количество столбцов: 1
Введите матрицу построчно:
3
4
+-------+
|  11   |
+-------+
```

Рисунок 4.3 – Пример работы программы (часть 3)

4.3 Время выполнения алгоритмов

Как было сказано выше, используется функция замера процессорного времени process_time из библиотеки time на Python. Функция возвращает пользовательское процессорное время типа float.

Использовать функцию приходится дважды, затем из конечного времени нужно вычесть начальное, чтобы получить результат.

Замеры проводились для размеров матриц от 10 до 100 с шагом 10 по 100 раз на различных входных данных, а также для нечетных размеров — от 11 до 101 с шагом 10 по 100 раз.

Результаты замеров приведены в таблицах 4.1, 4.2 (время в мс).

Таблица 4.1 – Результаты замеров времени для четных размеров матриц

Размер матрицы	Классический алгоритм	Алгоритм Винограда	Опт. алгоритм Винограда
10	0.266	0.219	0.125
20	1.719	0.625	0.938
30	5.469	5.0	3.125
40	12.656	10.938	7.031
50	22.031	20.0	17.031
60	27.344	34.688	26.562
70	47.969	52.5	40.625
80	69.219	74.844	65.312
90	107.031	112.812	89.375
100	138.125	153.281	124.688

Таблица 4.2 – Результаты замеров времени для нечетных размеров матриц

Размер матрицы	Классический алгоритм	Алгоритм Винограда	Опт. алгоритм Винограда
11	0.312	0.203	0.188
21	1.875	1.406	1.094
31	3.594	4.531	3.594
41	9.375	10.781	10.0
51	19.375	21.406	17.812
61	32.5	36.094	28.594
71	49.688	55.938	44.219
81	72.969	84.531	68.594
91	108.125	116.406	94.219
101	139.375	164.062	132.656

На рисунках 4.4, 4.5 приведена визуализация результатов замеров.

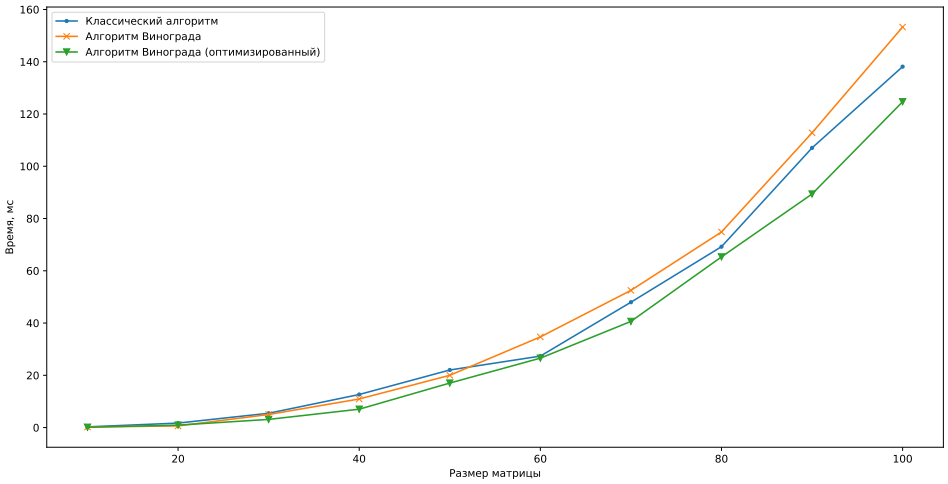

Рисунок 4.4 – Визуализация результатов замеров для четных размеров матриц

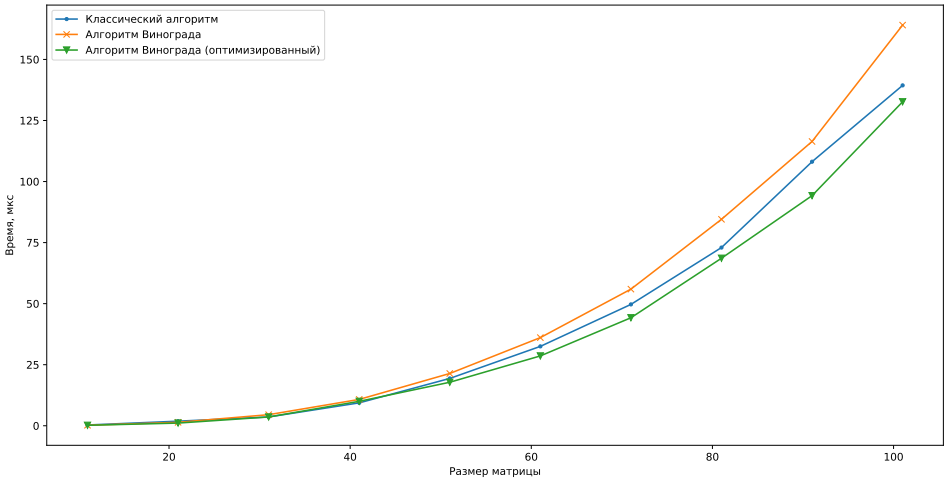

Рисунок 4.5 – Визуализация результатов замеров для нечетных размеров матриц

4.4 Вывод

В результате эксперимента было получено, что при четных размерах матриц реализация классического алгоритма умножения матриц работает быст-

рее алгоритма Винограда при размерах матриц больше $[50 \times 50]$. При нечетных размерах — при размерах больше $[21 \times 21]$.

Заключение

В результате исследования было определено, что при четных размерах матриц реализация классического алгоритма умножения матриц работает быстрее алгоритма Винограда при размерах матриц больше $[50 \times 50]$. При нечетных размерах — при размерах больше $[21 \times 21]$. Также было получено, что оптимизированный алгоритм Винограда требует больше памяти, чем классический: затраты на $2 \cdot sizeof(int)$ байт больше. Но наименьшие затраты по памяти у классического алгоритма умножения матриц.

Цель, которая была поставлена в начале лабораторной работы была достигнута: описаны, реализованы и исследованы алгоритмы умножения матриц: классический, Винограда и его оптимизации. В ходе выполнения были решены все задачи:

— описаны алгоритмы Винограда и его оптимизации;

— реализованы алгоритмы Винограда и его оптимизации;

— проведено тестирование по методу черного ящика для реализаций алгоритмов Винограда и его оптимизации;

— проведен сравнительный анализ по времени и по памяти реализаций алгоритмов Винограда и его оптимизации;

— описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчетно-пояснительная записка к работе.

Список используемых источников

1. time — Time access and conversions [Электронный ресурс]. — URL: `https://docs.python.org/3/library/time.html#functions` (дата обр. 10.10.2023).
2. Welcome to Python [Электронный ресурс]. — URL: `https://www.python.org` (дата обр. 10.10.2023).
3. *Н.С. Бахвалов Н.П. Жидков Г. К.* Численные методы. — М.: Наука, 1987.
4. *Т. Кормен Ч. Лейзерсон Р. Р.* Алгоритмы: построение и анализ. — МЦНТО, 1999.