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 дисциплине "Анализ алгоритмов"

Тема Алгоритмы умножения матриц

Студент Шматко К. М.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватель: Волкова Л. Л.

Москва — 2023 г.

Содержание

Введение

В данной лабораторной работе будут рассмотрены алгоритмы умножения матриц. В программировании и математике матрицы играют значительную роль и используются в разнообразных областях. Они являются важными инструментами при решении физических задач, таких как вычисление градиента, дивергенции и ротора.

Важным аспектом работы с матрицами являются различные операции, такие как сложение, возведение в степень и умножение. В реальных задачах матрицы могут иметь большие размеры, поэтому оптимизация операций над ними является важной задачей в программировании. В данной лабораторной работе мы рассмотрим оптимизацию операции умножения матриц.

Целью данной лабораторной работы является описание, реализация и исследование алгоритмов умножения матриц.

Для достижения данной цели необходимо выполнить следующие задачи.

1) Описать два алгоритма умножения матриц.
2) Создать программное обеспечение, реализующее следующие алгоритмы:
   - классический алгоритм умножения матриц;
   - алгоритм Винограда;
   - оптимизированный алгоритм Винограда.
3) Оценить трудоемкость алгоритмов.
4) Провести анализ затрат работы программы по времени, выяснить влияющие на них характеристики.
5) Провести сравнительный анализ между алгоритмами.

1 Аналитическая часть

В этом разделе будут рассмотрены классический алгоритм умножения матриц и алгоритм Винограда, а также его оптимизированная версия.

1.1 Матрица

Матрицей называется таблица чисел a_{ik} вида

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \tag{1.1}$$

состоящая из m строк и n столбцов [1]. Числа a_{ik} называются её *элементами*.

Пусть A – матрица, тогда $A_{i,j}$ – элемент этой матрицы, который находится на *i-ой* строке и *j-ом* столбце.

Можно выделить следующие операции над матрицами:

1) сложение матриц одинакового размера;

2) вычитание матриц одинакового размера;

3) умножение матриц в случае, если количество столбцов первой матрицы равно количеству строк второй матрицы. В итоговой матрице количество строк будет равно количеству строк первой матрицы, а количество столбцов - количеству столбцов второй матрицы.

Замечание: операция умножения матриц не коммутативна, если A и B – квадратные матрицы, а C – результат их перемножения, то произведение AB и BA дадут разный результат C.

1.2 Классический алгоритм

Пусть даны две матрицы

$$A_{nq} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1q} \\ a_{21} & a_{22} & \dots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nq} \end{pmatrix}, \quad B_{qm} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{q1} & b_{q2} & \dots & b_{qm} \end{pmatrix}, \tag{1.2}$$

тогда матрица C

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

где

$$c_{ij} = \sum_{r=1}^{q} a_{ir} b_{rj} \quad (i = \overline{1,n}; j = \overline{1,m}) \tag{1.4}$$

будет называться произведением матриц A и B [1].

Классический алгоритм реализует данную формулу 1.4.

1.3 Алгоритм Винограда

Алгоритм Винограда — это алгоритм умножения матриц. Данный алгоритм имеет лучшую асимптотику среди известных алгоритмов умножения матриц.

Рассмотрим пример: пусть есть два вектора $U = (u_1, u_2, u_3, u_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение можно записать как:

$$U \cdot W = u_1 w_1 + u_2 w_2 + u_3 w_3 + u_4 w_4$$

Это равносильно следующему выражению:

$$U \cdot W = (u_1+w_2)(u_2+w_1)+(u_3+w_4)(u_4+w_3)-u_1u_2-u_3u_4-w_1w_2-w_3w_4 \quad (1.5)$$

Пусть матрицы A, B и C имеют соответствующие размеры. Согласно идее Винограда, скалярное произведение, вычисленное по формуле 1.6, может быть оптимизировано следующим образом:

$$C_{ij} = \sum_{k=1}^{q/2}(a_{i,2k-1}+b_{2k,j})(a_{i,2k}+b_{2k-1,j})-\sum_{k=1}^{q/2} a_{i,2k-1}a_{i,2k}-\sum_{k=1}^{q/2} b_{2k-1,j}b_{2k,j} \quad (1.6)$$

Важно отметить, что выражения во втором и третьем слагаемых в формуле 1.6 могут быть предварительно вычислены для каждой строки матрицы A и каждого столбца матрицы B. Это позволяет сэкономить время, так как эти значения можно запомнить и использовать при вычислении элементов матрицы C. Также важно, что при нечётном значении размера матрицы нужно дополнительно добавить произведения крайних элементов соответствующих строк и столбцов.

При реализации алгоритма Винограда можно внести следующие оптимизации.

1) Значение $\frac{N}{2}$, которое используется как ограничение цикла для вычисления предварительных данных, можно кэшировать.

2) Операцию умножения на 2 лучше реализовать как побитовый сдвиг влево на 1.

3) Операции сложения и вычитания с присваиванием следует использовать через операторы $+=$ и $-=$ (если они доступны в выбранном языке программирования).

Вывод

В данном разделе были рассмотрены алгоритмы умножения матриц – классического и Винограда, который имеет большую эффективность за

счёт предварительных вычислений. Также были рассмотрены оптимизации, которые можно учесть при программной реализации алгоритма Винограда.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов перемножения матриц – стандартного, Винограда и оптимизации алгоритма Винограда.

2.1 Разработка алгоритмов

На вход алгоритмов подаются матрицы A и B, причем количество столбцов одной матрицы должно совпадать с количеством строк второй матрицы.

На рисунке 2.1 представлена схема алгоритма для стандартного умножения матриц. На рисунках 2.2 и 2.3 схема алгоритма Винограда умножения матриц, а на 2.4 и 2.5 - схема оптимизированного алгоритма Винограда.

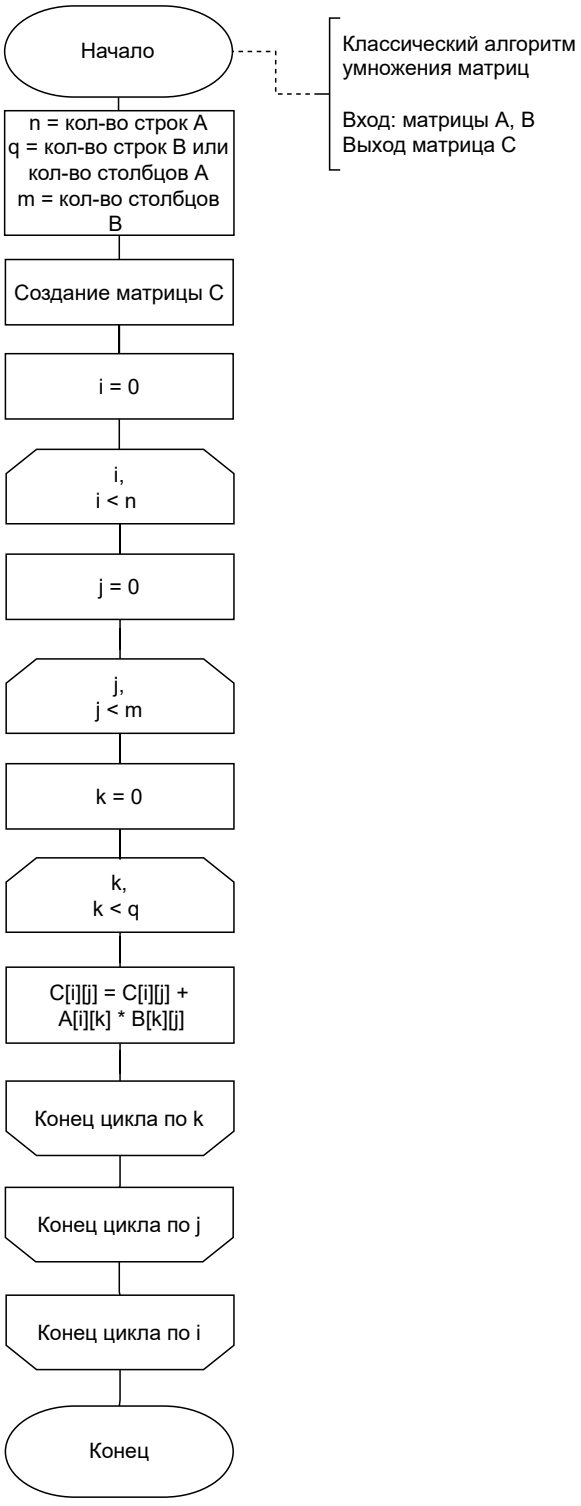

Рисунок 2.1 – Схема стандартного алгоритма умножения матриц

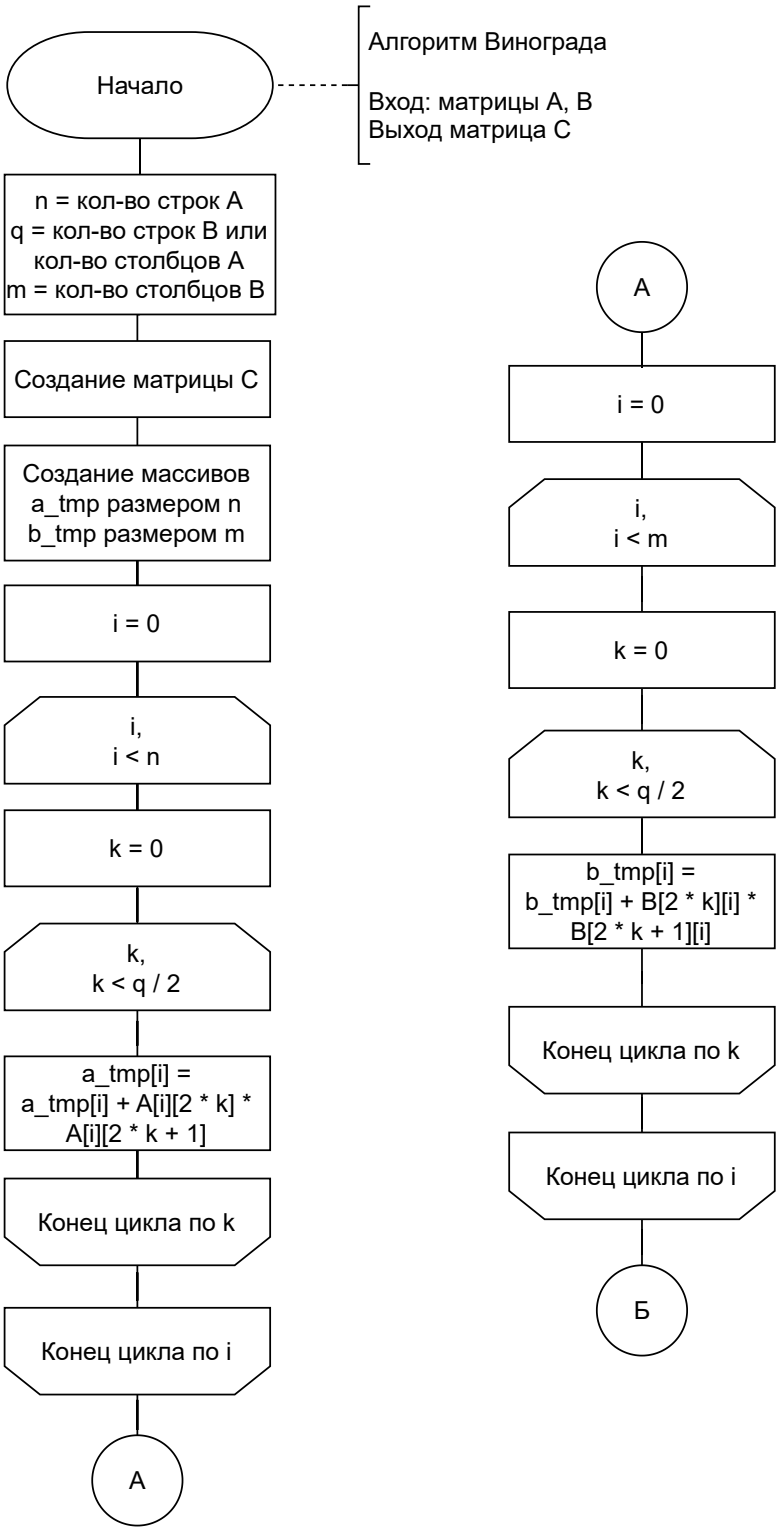

Рисунок 2.2 – Схема умножения матриц по алгоритму Винограда (Часть 1)

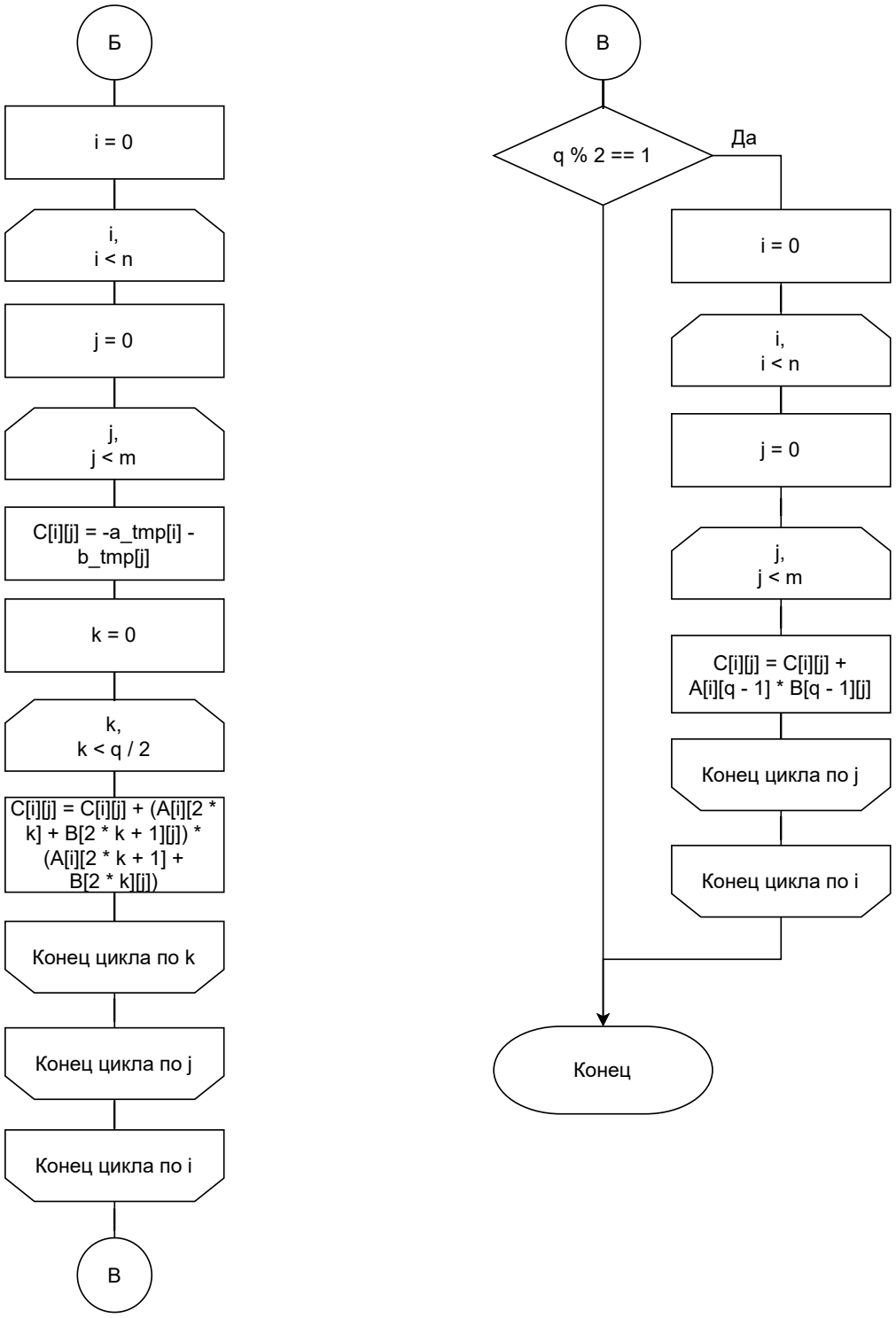

Рисунок 2.3 – Схема умножения матриц по алгоритму Винограда (Часть 2)

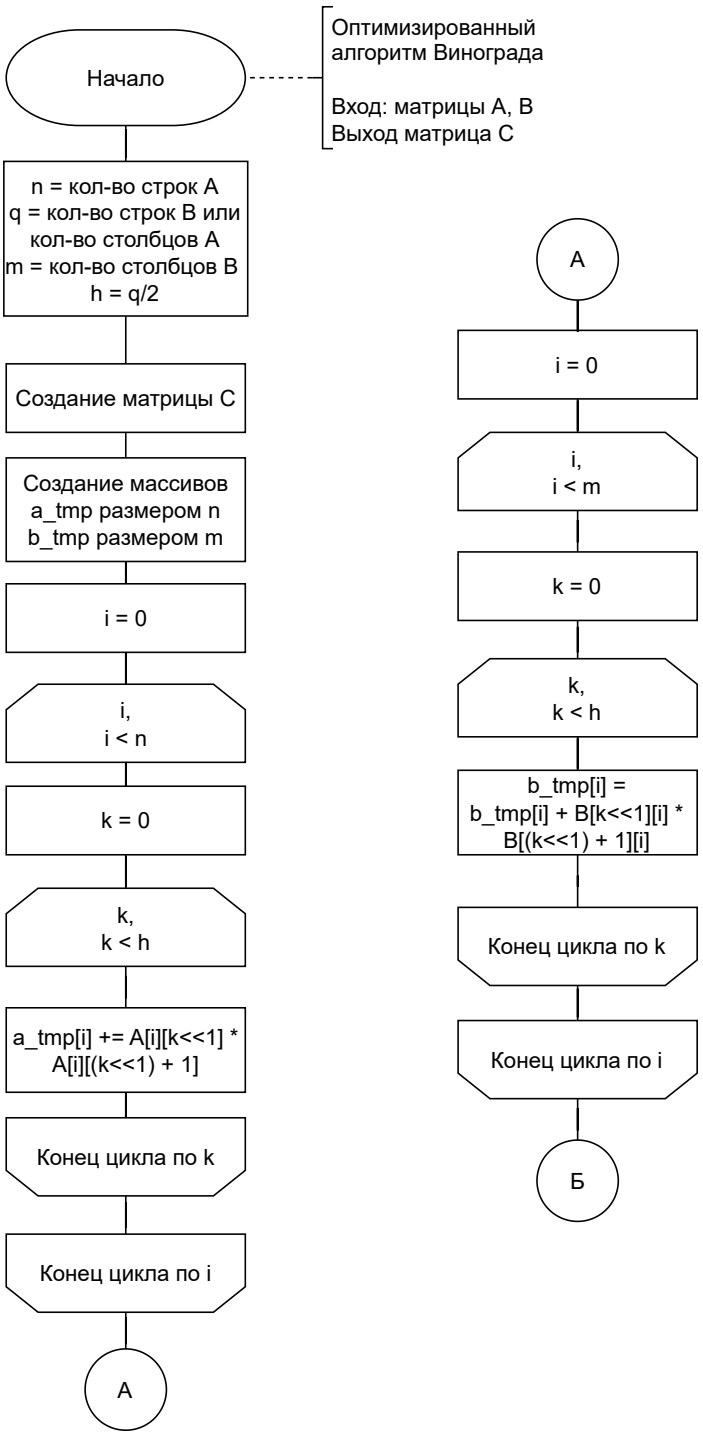

Рисунок 2.4 – Схема умножения матриц по оптимизированному алгоритму Винограда (Часть 1)

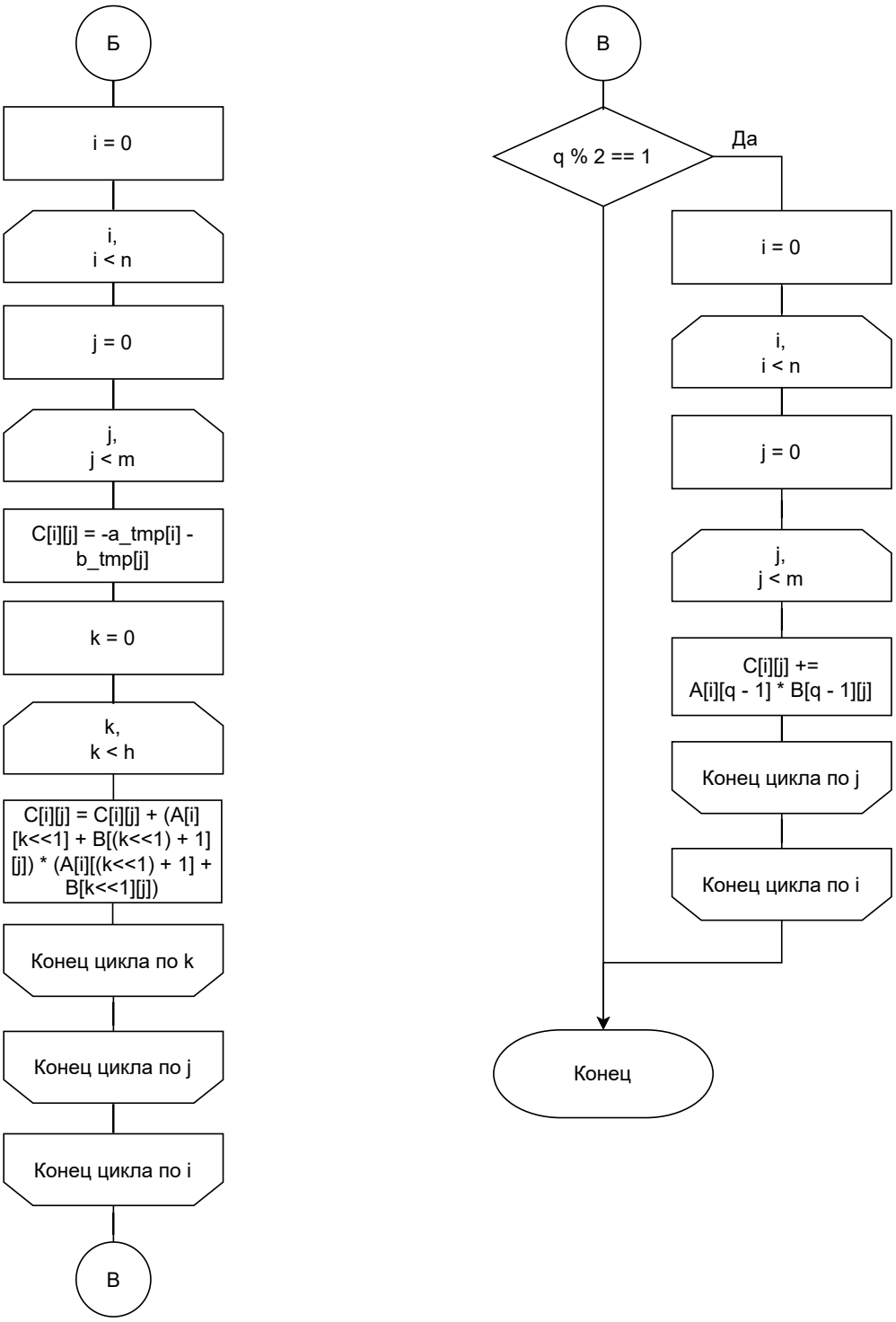

Рисунок 2.5 – Схема умножения матриц по оптимизированному алгоритму Винограда (Часть 2)

2.2 Модель вычислений для проведения оценки трудоемкости

Введем модель вычислений, которая потребуется для определения трудоемкости каждого отдельного взятого алгоритма сортировки.

1) Трудоемкость базовых операций имеет:

   — равную 1:

$$+, -, =, + =, - =, ==, ! =, <, >, <=, >=, [], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

   — равную 2:

$$*, /, \%, * =, / =, \% = \tag{2.2}$$

2) Трудоемкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{лучший случай} \\ max(f_1, f_2), & \text{худший случай} \end{cases} \tag{2.3}$$

3) Трудоемкость цикла:

$$f_{for} = f_{\text{инициализация}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

4) Трудоемкость передачи параметра в функции и возврат из функции равны 0.

2.3 Трудоемкость алгоритмов

Была рассчитана трудоемкость алгоритмов умножения матриц.

2.3.1 Классический алгоритм

Для стандартного алгоритма умножения матриц трудоемкость будет слагаться из:

— внешнего цикла по $i \in [1 \dots N]$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;

— цикла по $j \in [1 \dots Q]$, трудоёмкость которого: $f = 2+2+Q \cdot (2+f_{body})$;

— цикла по $k \in [1 \dots M]$, трудоёмкость которого: $f = 2 + 2 + 14M$;

Поскольку трудоемкость стандартного алгоритма равна трудоемкости внешнего цикла, то:

$$\begin{aligned} f_{standart} &= 2 + N \cdot (2 + 2 + Q \cdot (2 + 2 + M \cdot (2 + 8 + 1 + 1 + 2))) = \\ &= 2 + 4N + 4NQ + 14NMQ \approx 14NMQ = O(N^3) \end{aligned} \tag{2.5}$$

2.3.2 Алгоритм Винограда

Чтобы вычислить трудоемкость алгоритма Винограда, нужно учесть следующее:

— создания и инициализации массивов a_{tmp} и b_{tmp}, трудоёмкость которых указана в формуле (2.6);

$$f_{init} = N + M \tag{2.6}$$

— заполнения массива a_{tmp}, трудоёмкость которого указана в формуле (2.7);

$$\begin{aligned} f_{atmp} &= 2 + N \cdot (2 + 4 + \frac{Q}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ &= 2 + 6N + \frac{19NQ}{2} = 2 + 46 + 9,5NQ \end{aligned} \tag{2.7}$$

— заполнения массива b_{tmp}, трудоёмкость которого указана в формуле (2.8);

$$f_{btmp} = 2 + M \cdot (2 + 4 + \frac{Q}{2} \cdot (4 + 6 + 1 + 2 + 3 \cdot 2)) = \\ = 2 + 6M + \frac{19QM}{2} = 2 + 6M + 9,5QM \tag{2.8}$$

— цикла заполнения для чётных размеров, трудоёмкость которого указана в формуле (2.9);

$$f_{cycle} = 2 + N \cdot (4 + Q \cdot (2 + 7 + 4 + \frac{Q}{2} \cdot (4 + 28))) = \\ = 2 + 4N + 13NM + \frac{32NQM}{2} = 2 + 4N + 13NM + 16NQM \tag{2.9}$$

— цикла, который дополнительно нужен для подсчёта значений при нечётном размере матрицы, трудоемкость которого указана в формуле (2.10);

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + M \cdot (2 + 14)), & \text{иначе} \end{cases} \tag{2.10}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем:

$$f_{worst} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMQ = O(N^3) \tag{2.11}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f_{best} = f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 16NMQ = O(N^3) \tag{2.12}$$

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

— кэширования значения $\frac{Q}{2}$ в циклах, которое равно 3;

— создания и инициализации массивов a_{tmp} и b_{tmp} (2.6);

— заполнения массива a_{tmp}, трудоёмкость которого (2.7);

— заполнения массива b_{tmp}, трудоёмкость которого (2.8);

— цикла заполнения для чётных размеров, трудоёмкость которого указана в формуле (2.13);

$$\begin{aligned} f_{cycle} &= 2 + N \cdot (4 + M \cdot (4 + 7 + \frac{Q}{2} \cdot (2 + 10 + 5 + 2 + 4))) = \\ &= 2 + 4N + 11NM + \frac{23NQM}{2} = \\ &= 2 + 4N + 11NM + 11,5 \cdot NQM \end{aligned} \tag{2.13}$$

— условие, которое нужно для дополнительных вычислений при нечётном размере матрицы, трудоемкость которого указана в формуле (2.14);

$$f_{check} = 3 + \begin{cases} 0, & \text{чётная} \\ 2 + N \cdot (4 + M \cdot (2 + 10)), & \text{иначе} \end{cases} \tag{2.14}$$

Тогда для худшего случая (нечётный общий размер матриц) имеем:

$$f_{bad} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMQ = O(N^3) \tag{2.15}$$

Для лучшего случая (чётный общий размер матриц) имеем:

$$f_{best} = 3 + f_{init} + f_{atmp} + f_{btmp} + f_{cycle} + f_{check} \approx 11NMQ = O(N^3) \tag{2.16}$$

Вывод

В данном разделе на основе теоретических данных были построены схемы требуемых алгоритмов и проведена теоретическая оценка трудоемкости алгоритмов, показавшая, что трудоёмкость выполнения алгоритма Винограда в случае оптимизации в 1.2 раза меньше, чем у простого алгоритма Винограда.

3 Технологическая часть

В данном разделе будут описаны требования к программному обеспечению, средства реализации, выбранные типы данных, листинг кода и функциональные тесты.

3.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

— принимает на вход две матрицы, где количество столбцов одной матрицы совпадает с количеством строк второй матрицы;

— выдает матрицу, являющуюся результатом перемножения двух матриц;

— имеет интерфейс для выбора действий;

— имеет функциональность замера процессорного времени работы реализаций алгоритмов.

3.2 Средства реализации

Для реализации данной лабораторной работы был выбран язык JavaScript [2]. Данный выбор обусловлен наличием у языка встроенной функции *process.cpuUsage*() для измерения процессорного времени. Это позволяет удовлетворить требованиям для выполнения лабораторной работы.

Время выполнения программы было замерено с использованием функции *process.cpuUsage()* из программной платформы Node.js [3].

3.3 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры данных:

— количество строк — целое число типа *number*;

— количество столбцов — целое число типа *number*;

— матрица — двумерный массив значений типа *number*.

3.4 Сведения о модулях программы

Данная программа разбита на следующие модули.

— `main.js` — файл, содержащий точку входа в программу, из которой происходит запуск работы алгоритмов и вывод графиков.

— `algorithms.js` — файл содержит функции алгоритмов умножения матриц.

— `times.cjs` — файл содержит функции, замеряющие процессорное время алгоритмов умножения матриц.

— `index.html` — файл веб-страницы, который используется для отображения результата работы алгоритмов.

— `style.css` — файл, который используется для определения стилей интерфейса веб-страницы.

— `measures.json` — файл, в который `times.cjs` записывает результаты замеров работы алгоритмов, и из которого впоследствии считывает данные файл `main.js` для построения графиков.

3.5 Реализация алгоритмов

В листинге 3.1 приведена реализация алгоритма создания матрицы. В листингах 3.2 – 3.4 приведены реализации алгоритмов умножения матриц — классический, алгоритм Винограда и оптимизированный алгоритм Винограда.

Листинг 3.1 – Функция создания матрицы для алгоритмов умножения матриц

```
function createMatrix(matrix, n, m, fill) {
    for (let i = 0; i < n; i++) {
        matrix[i] = [];
        for (let j = 0; j < m; j++) {
            matrix[i][j] = fill;
        }
    }
}
```

Листинг 3.2 – Функция классического алгоритма умножения матриц

```
export function multiplyClassic(matrixA, matrixB) {
    let n = matrixA.length;
    let m = matrixB[0].length;
    let q = matrixB.length;
    let res = [];
    createMatrix(res, n, m, 0);
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
            for (let k = 0; k < q; k++) {
                res[i][j] = res[i][j] + matrixA[i][k] *
                    matrixB[k][j];
            }
        }
    }

    return res;
}
```

Листинг 3.3 – Функция умножения матриц по алгоритму Винограда

```
export function multiplyVin(matrixA, matrixB) {
    let n = matrixA.length;
    let m = matrixB[0].length;
    let q = matrixB.length;
    let res = [];
    createMatrix(res, n, m, 0);
    let mulH = new Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        for (let k = 0; k < Math.floor(q / 2); k++) {
            mulH[i] = mulH[i] + matrixA[i][2 * k] *
               matrixA[i][2 * k + 1];
        }
    }
    let mulV = new Array(m).fill(0);
    for (let i = 0; i < m; i++) {
        for (let k = 0; k < Math.floor(q / 2); k++) {
            mulV[i] = mulV[i] + matrixB[2 * k][i] * matrixB[2 *
               k + 1][i];
        }
    }
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
            res[i][j] = -mulH[i] - mulV[j];
            for (let k = 0; k < Math.floor(q / 2); k++) {
                res[i][j] = res[i][j] + (matrixA[i][2 * k] +
                   matrixB[2 * k + 1][j]) * (matrixA[i][2 * k +
                   1] + matrixB[2 * k][j]);
            }
        }
    }
    if (q % 2 === 1) {
        for (let i = 0; i < n; i++) {
            for (let j = 0; j < m; j++) {
                res[i][j] = res[i][j] + matrixA[i][q-1] *
                   matrixB[q-1][j];
            }
        }
    }
    return res;
}
```

Листинг 3.4 – Функция умножения матриц по оптимизированному алгоритму Винограда

```
export function multiplyVinOpt(matrixA, matrixB) {
    let n = matrixA.length;
    let m = matrixB[0].length;
    let q = matrixB.length;
    let res = [];
    let middle = Math.floor(q / 2);
    createMatrix(res, n, m, 0);
    let mulH = new Array(n).fill(0);
    for (let i = 0; i < n; i++) {
        for (let k = 0; k < middle; k++) {
            mulH[i] += matrixA[i][k << 1] * matrixA[i][(k << 1)
               + 1];
        }
    }
    let mulV = new Array(m).fill(0);
    for (let i = 0; i < m; i++) {
        for (let k = 0; k < middle; k++) {
            mulV[i] += matrixB[k << 1][i] * matrixB[(k << 1) +
               1][i];
        }
    }
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
            res[i][j] = -mulH[i] - mulV[j];
            for (let k = 0; k < middle; k++) {
                res[i][j] += (matrixA[i][k << 1] + matrixB[(k
                   << 1) + 1][j]) * (matrixA[i][(k << 1) + 1] +
                    matrixB[k << 1][j]);
            }
        }
    }
    if (q % 2 === 1) {
        for (let i = 0; i < n; i++) {
            for (let j = 0; j < m; j++) {
                res[i][j] += matrixA[i][q-1] * matrixB[q-1][j];
            }
        }
    } return res;
}
```

3.6 Функциональные тесты

В таблицах 3.1 и 3.2 приведены функциональные тесты для алгоритмов умножения матриц. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты для классического алгоритма умножения матриц

Матрица 1	**Матрица 1**	**Результат для классического алгоритма**
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$(\ \ )$	Сообщение об ошибке
$\begin{pmatrix} 1 & 5 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 5 \\ 3.4 & 0 \\ -41 & 9 \end{pmatrix}$	$\begin{pmatrix} -3 & 2 & 0.4 \\ -19 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -98 & 17 & 20.4 \\ -10.2 & 6.8 & 1.36 \\ -48 & -55 & 19.6 \end{pmatrix}$
$\begin{pmatrix} 21 \end{pmatrix}$	$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 210 \end{pmatrix}$

Таблица 3.2 – Функциональные тесты для алгоритма Винограда

Матрица 1	**Матрица 1**	**Результат для алгоритма Винограда**
$\begin{pmatrix} 1 & 5 & 7 \\ 2 & 6 & 8 \\ 3 & 7 & 9 \end{pmatrix}$	$(\ \ )$	Сообщение об ошибке
$\begin{pmatrix} 1 & 5 & 7 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 5 \\ 3.4 & 0 \\ -41 & 9 \end{pmatrix}$	$\begin{pmatrix} -3 & 2 & 0.4 \\ -19 & 3 & 4 \end{pmatrix}$	$\begin{pmatrix} -98 & 17 & 20.4 \\ -10.2 & 6.8 & 1.36 \\ -48 & -55 & 19.6 \end{pmatrix}$
$\begin{pmatrix} 21 \end{pmatrix}$	$\begin{pmatrix} 10 \end{pmatrix}$	$\begin{pmatrix} 210 \end{pmatrix}$

Вывод

Были реализованы алгоритмы умножения матриц – стандартного, Винограда и оптимизированного алгоритма Винограда. Проведено тестирование реализаций алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, проведение исследования и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени.

— Процессор: 11th Gen Intel(R) Core(TM) i5-1135G7 2.42 ГГц.

— Оперативная память: 16 ГБайт.

— Операционная система: Windows 11 Домашняя 64-разрядная система версии 22H2 [4].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.2 Демонстрация работы программы

Программа получает на вход 2 матрицы и выдает 3 матрицы соответствующих алгоритмов умножения – стандартного, Винограда и оптимизированного алгоритма Винограда.

На рисунках 4.1 – 4.5 представлена демонстрация работы программы.

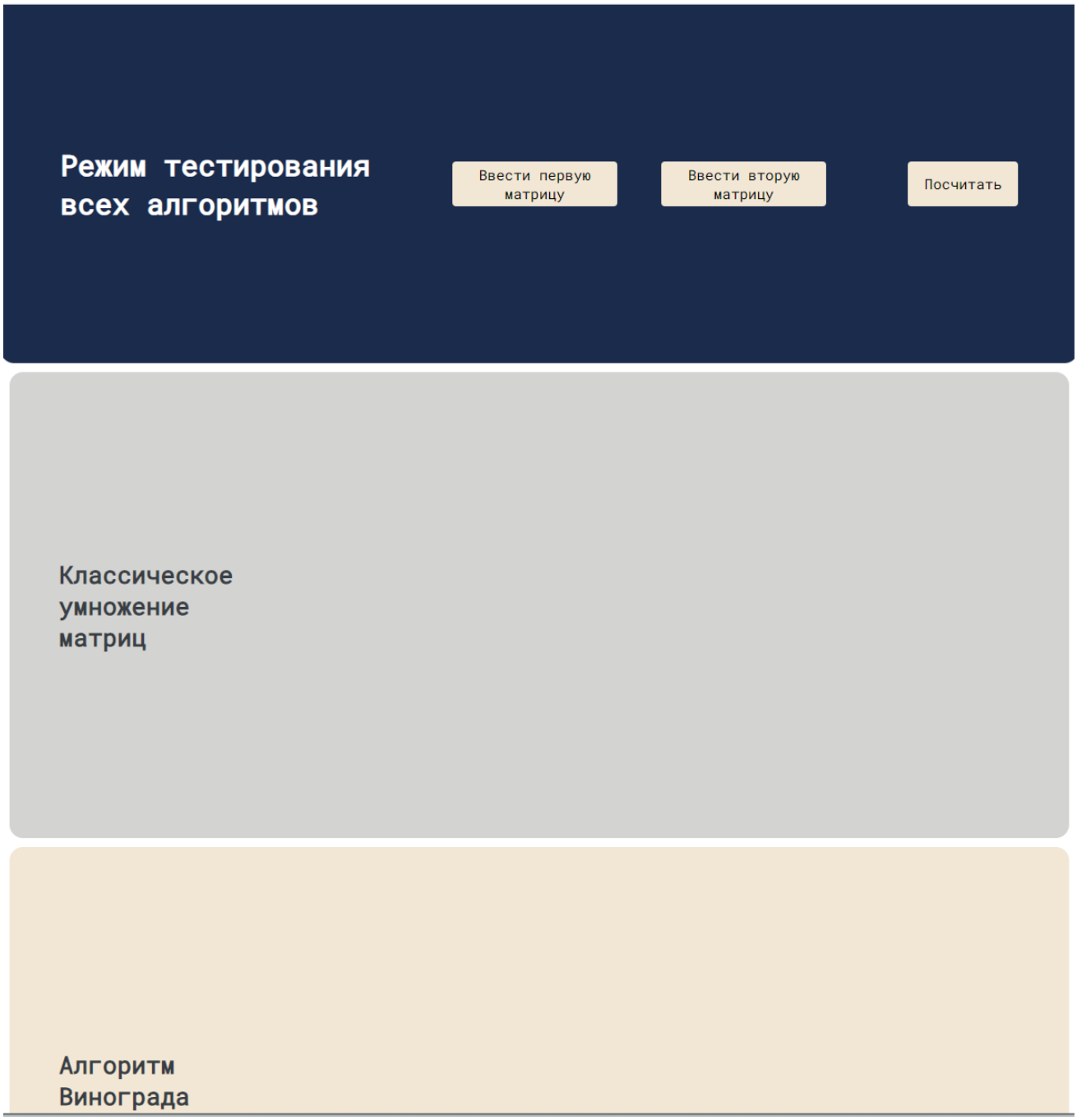

Рисунок 4.1 – Демонстрация работы программы при умножении матриц(часть 1)

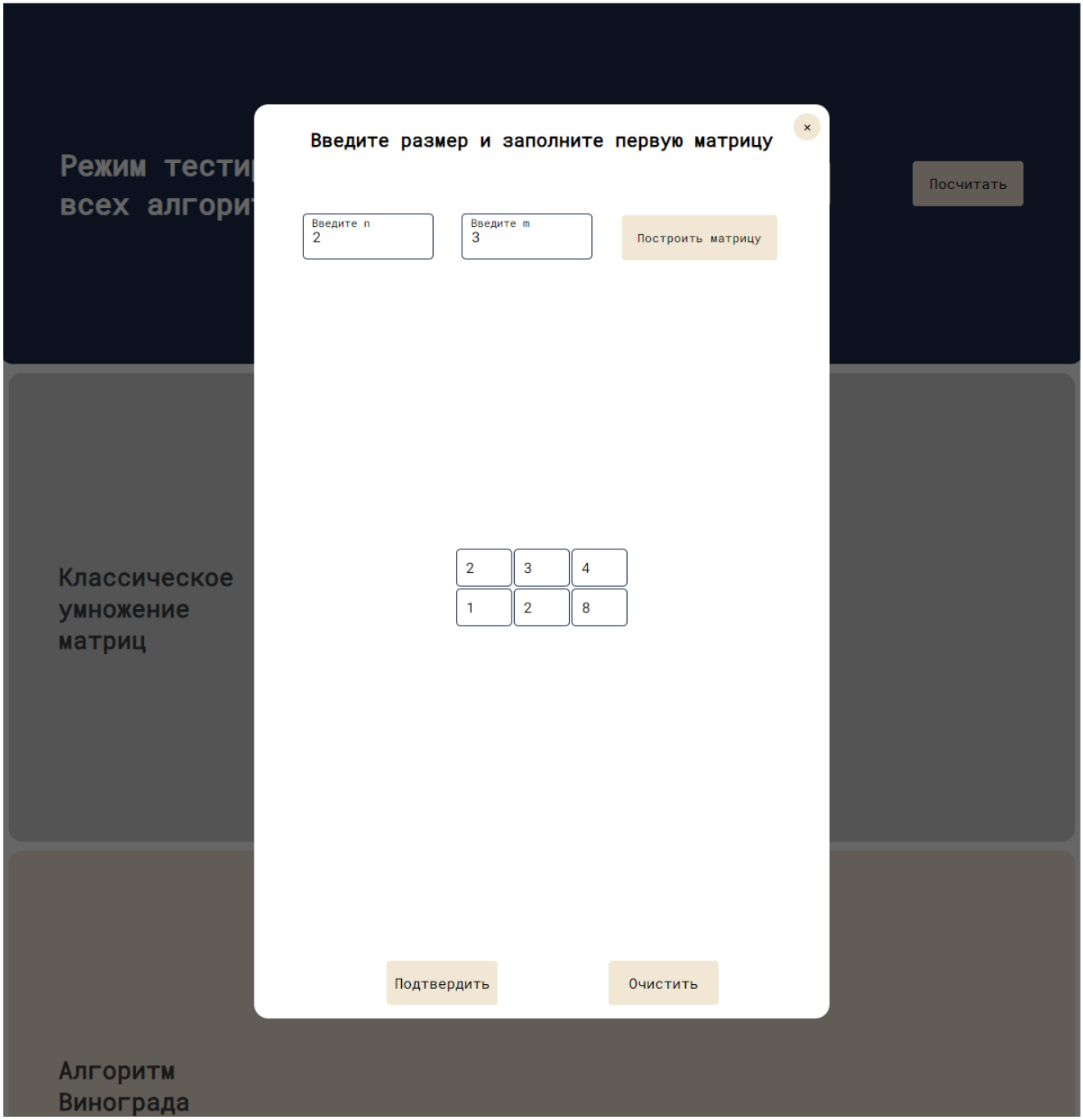

Рисунок 4.2 – Демонстрация работы программы при умножении матриц (часть 2)

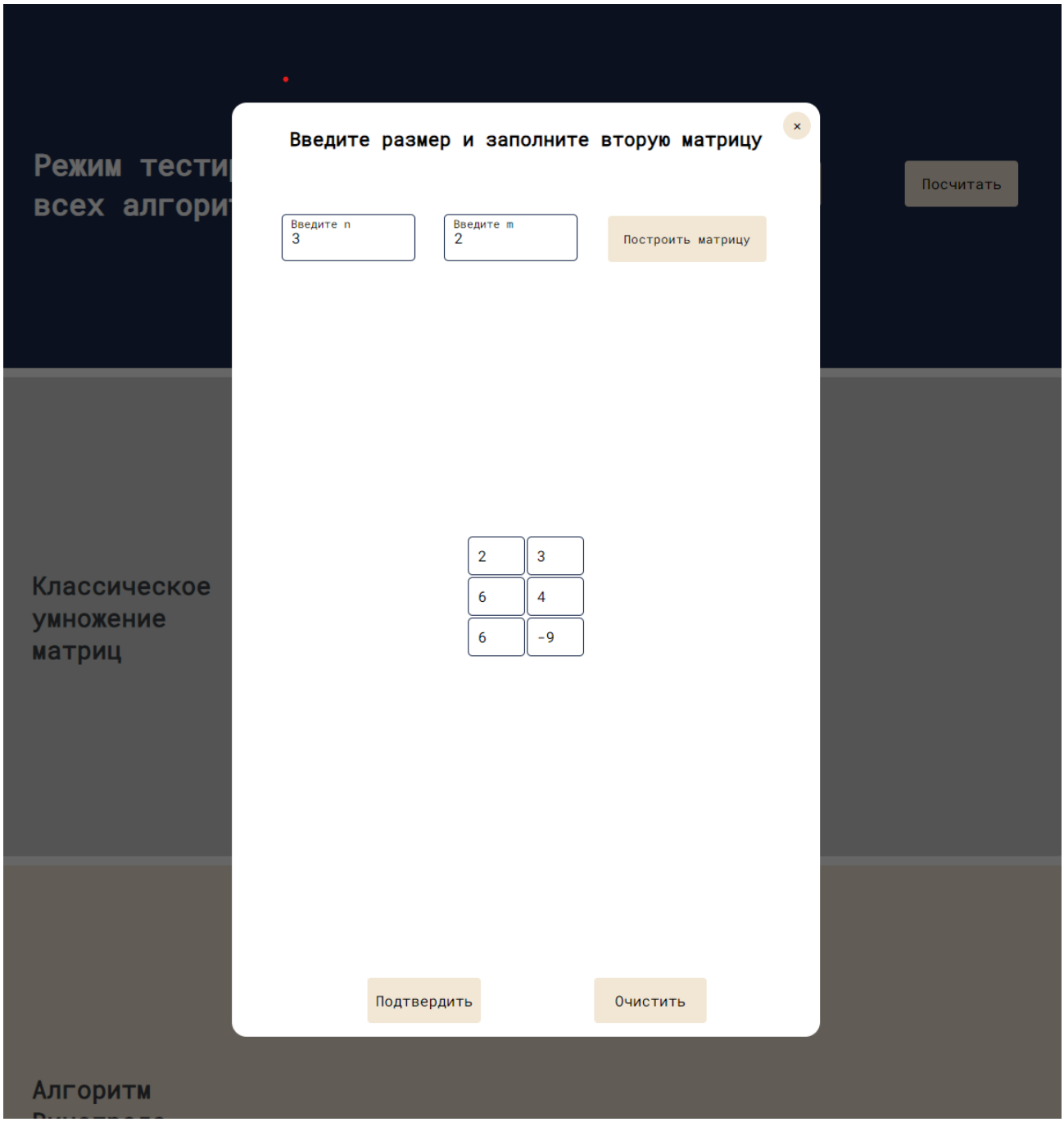

Рисунок 4.3 – Демонстрация работы программы при умножении матриц (часть 3)

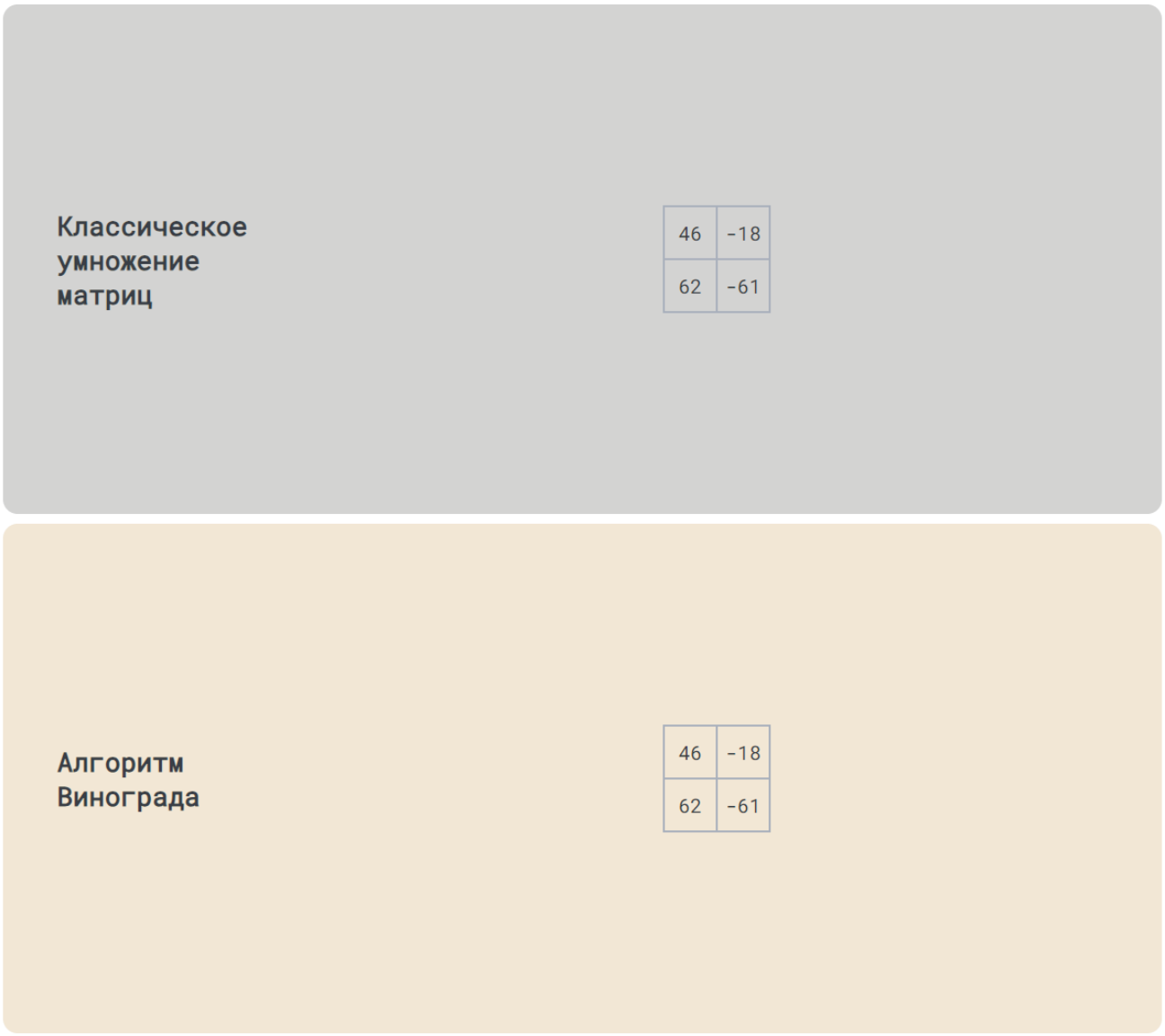

Рисунок 4.4 – Демонстрация работы программы при умножении матриц (часть 4)

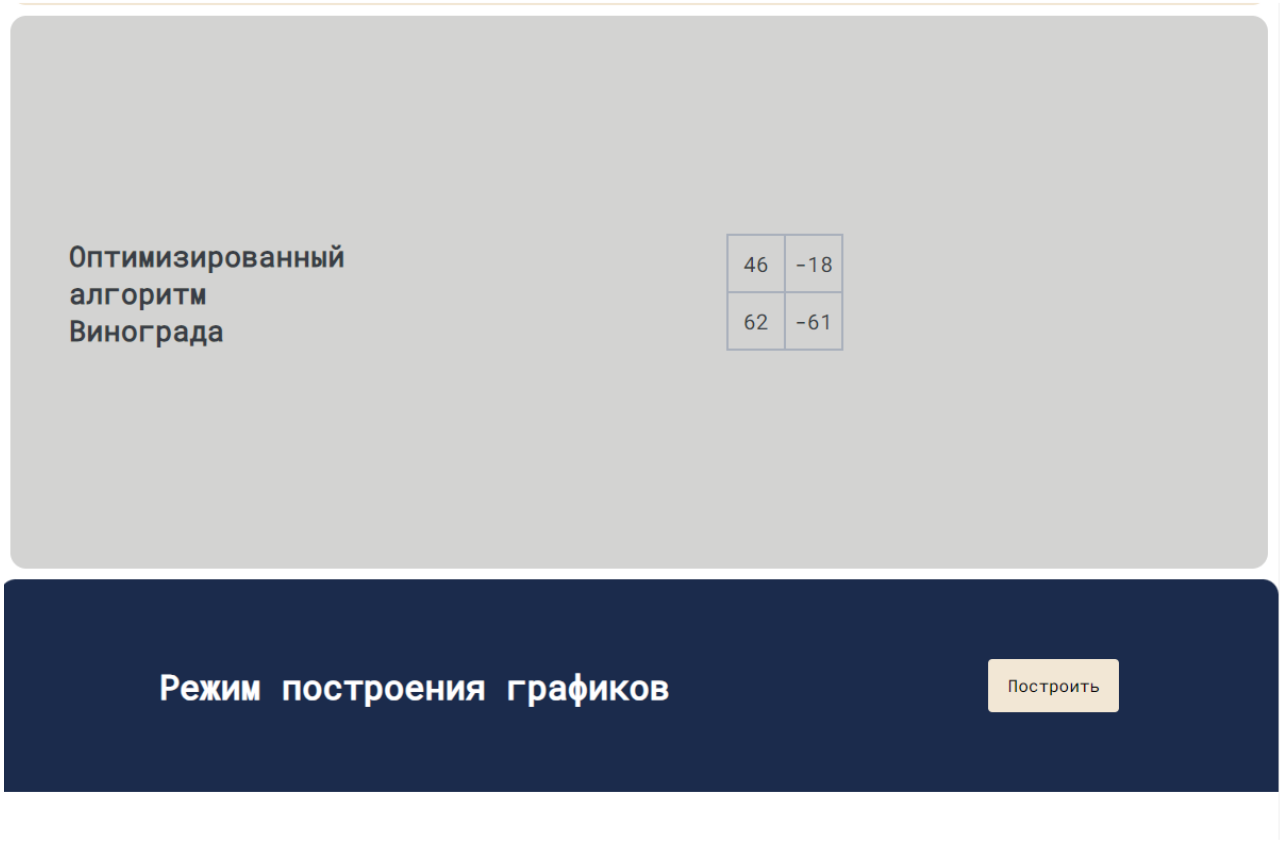

Рисунок 4.5 – Демонстрация работы программы при умножении матриц (часть 5)

4.3 Временные характеристики

Результаты эксперимента замеров по времени приведены в таблицах 4.1, 4.2.

В таблице 4.1 приведены результаты замеров по времени алгоритмов умножения матриц при четных размерах квадратных матриц.

Таблица 4.1 – Замер по времени для матриц, размер которых от 10 до 200

Размер	**Классический, нс**	**Виноград, нс**	**Оптимизированный Виноград, нс**
10	0	0	0
20	0	0	0
30	156.25	0	0
40	156.25	156.25	156.25
50	468.75	468.75	312.5
60	781.25	468.75	625
70	1406.25	937.5	937.5
80	2187.5	1406.25	1406.25
90	2343.75	1562.5	1875
100	3281.25	2500	2187.5
150	11406.25	8125	8281.25
200	26093.75	19687.5	19218.75

В таблице 4.2 приведены результаты замеров по времени алгоритмов умножения матриц при нечетных размерах квадратных матриц.

Таблица 4.2 – Замер по времени для матриц, размер которых от 11 до 201

Размер	**Классический, нс**	**Виноград, нс**	**Оптимизированный Виноград, нс**
11	0	0	0
21	0	0	0
31	156.25	0	0
41	156.25	156.25	156.25
51	468.75	156.25	356.25
61	937.5	468.75	781.25
71	1718.75	781.25	1250
81	2187.5	1406.25	1250
91	2343.75	1718.75	1718.75
101	3437.5	2500	2656.25
151	10937.5	9218.75	8593.75
201	34218.75	22187.5	20156.25

По таблице 4.1 был построен график 4.6. Исходя из этих данных можно понять, что лучшего всего работает оптимизированный алгоритм Винограда.

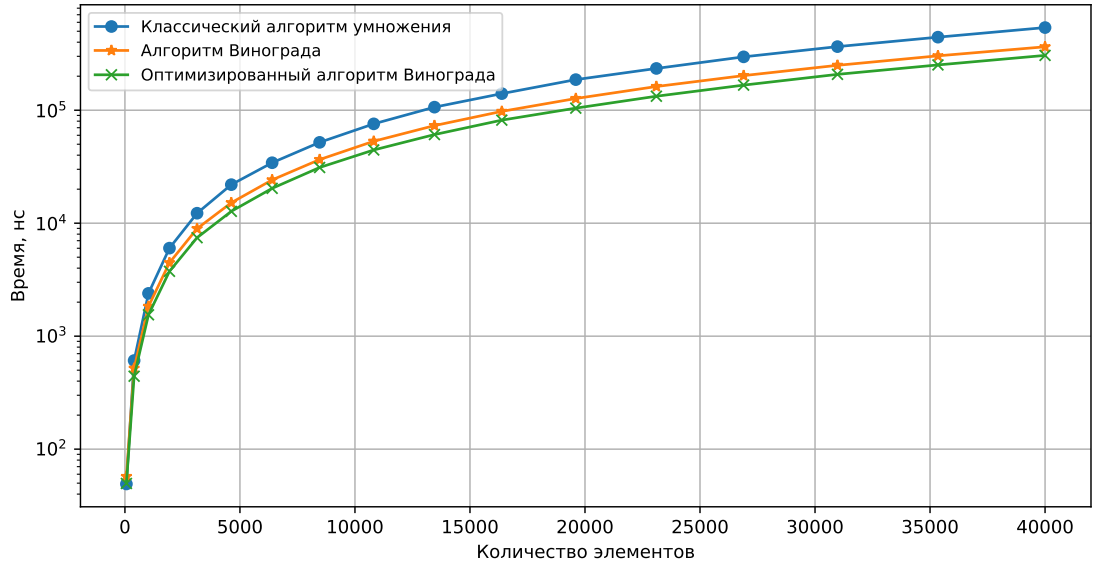

Рисунок 4.6 – Сравнение по времени алгоритмов умножения матриц на чётных размерах матриц

По таблице 4.2 был построен графики 4.7. Исходя из этих данных можно понять, что лучшего всего работает оптимизированный алгоритм Винограда.

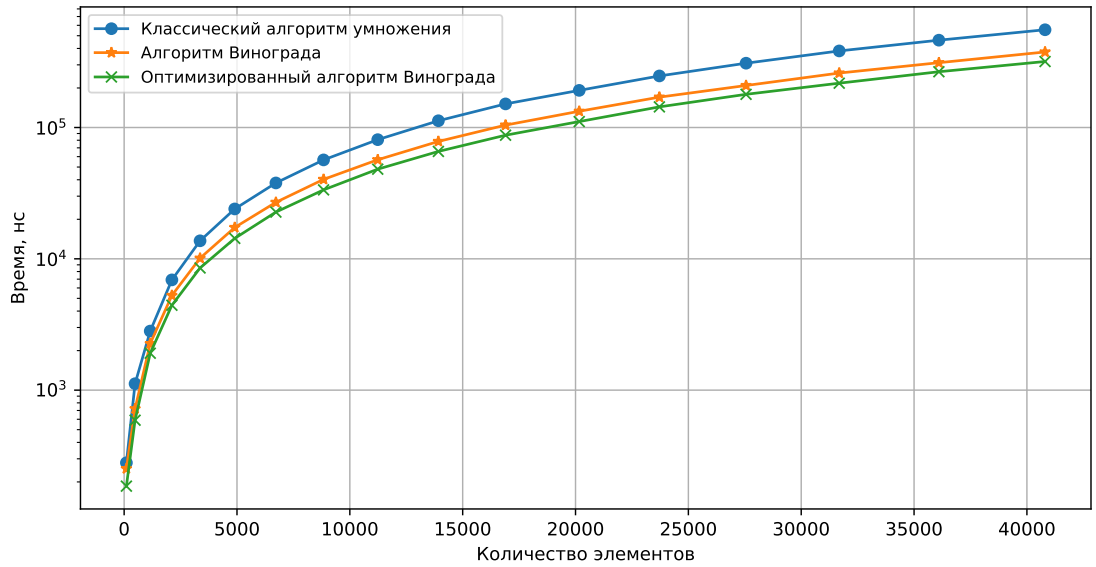

Рисунок 4.7 – Сравнение по времени алгоритмов умножения матриц на нечётных размерах матриц

4.4 Вывод

В результате эксперимента было получено, что при больших размерах матриц (свыше 10), оптимизированный алгоритм Винограда быстрее стандартного алгоритма и алгоритма Винограда. Также было выяснено, что на чётных размерах реализация алгоритма Винограда быстрее, чем на нечётных размерах матриц, что обусловлено необходимостью проводить дополнительные вычисления для крайних строк и столбцов. Следовательно, стоит использовать алгоритм Винограда для матриц, которые имеют чётные размеры.

Заключение

В результате исследования было определено, что стандартный алгоритм умножения матриц проигрывает по времени алгоритму Винограда из-за того, что в алгоритме Винограда часть вычислений происходит заранее, а также сокращается часть сложных операций – операций умножения, поэтому предпочтение следует отдавать алгоритму Винограда. Но лучшие показатели по времени выдает оптимизированный алгоритм Винограда – он быстрее алгоритма Винограда на размерах матриц свыше 10 из-за замены операций равно и плюс на операцию плюс-равно, а также за счёт замены операции умножения операцией сдвига, что дает проводить часть вычислений быстрее. Поэтому при выборе самого быстрого алгоритма предпочтение стоит отдавать оптимизированному алгоритму Винограда. Также стоит упомянуть, что алгоритм Винограда работает на чётных размерах матриц быстрее, чем на нечётных, что связано с тем, что нужно произвести часть дополнительных вычислений для крайних строк и столбцов матриц, поэтому алгоритм Винограда лучше работает чётных размерах матриц.

Цель, которая была поставлена в начале лабораторной работы была достигнута, а также в ходе выполнения лабораторной работы были решены следующие задачи.

1) Описаны два алгоритмы умножения матриц.
2) Создано программное обеспечение, реализующее следующие алгоритмы:
   - классический алгоритм умножения матриц;
   - алгоритм Винограда;
   - оптимизированный алгоритм Винограда.
3) Оценены трудоемкости алгоритмов.
4) Проведен анализ затрат работы программы по времени, выяснены влияющие на них характеристики.
5) Проведен сравнительный анализ между алгоритмами.

Список использованных источников

1 Баварин И. И. Высшая математика: учебник по естественно-научным направлениям и специальностям. — М.: Гуманит. изд. центр ВЛАДОС, 2003.

2 Документация по JavaScript [Электронный ресурс]. — Режим доступа: `https://262.ecma-international.org/13.0/` (дата обращения: 17.09.2023).

3 Node.js function process.cpuUsage([previousValue]) [Электронный ресурс]. — Режим доступа: `https://nodejs.org/api/process.html#processcpuusagepreviousvalue` (дата обращения: 17.09.2023).

4 Windows 11 [Электронный ресурс]. — Режим доступа: `https://www.microsoft.com/ru-ru/software-download/windows11` (дата обращения: 18.09.2023).