Министерство образования и науки 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)

ОТЧЕТ
По лабораторной работе №2
По курсу «Анализ алгоритмов»
Тема: «Алгоритмы умножения матриц»

Студент: Зейналов З. Г.
Группа: ИУ7-51
Преподаватель: Волкова Л.Л.

Москва, 2019г.

Оглавление

Введение

Целью данной лабораторной работы является исследование существующих алгоритмов умножения матриц и трудоемкости их вычисления.

Выберем следующую модель вычислений:

1. Трудоемкость базовых операций
   Операции $+, -, *, \ /, \%, =, >, <, \geq, \leq, ==, \neq, [\ ], +=, -=$ - имеют стоимость 1.
2. Трудоемкость условного перехода
   Условный переход имеет стоимость 0, при этом оцениваем расчет условия:

$$
\begin{array}{l}
if\ (n\ \%\ 2 == 1) \\
\{ \\
\quad \text{Тело1} \\
\} \\
else \\
\{ \\
\quad \text{Тело2} \\
\} \\
f_{if} = f_{\text{условия}} + \begin{bmatrix} f_{\text{тело1}}, \text{при нечетном } N \\ f_{\text{тело2}}, \text{при четном} \end{bmatrix}
\end{array}
$$

3. Трудоемкость цикла
   $f_{\text{цикла}} = f_{\text{инициализации}} + f_{\text{сравнения}} + N\left(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}\right)$,
   где N – число повторений цикла

Задачи работы:

Задачами данной лабораторной работы являются:

1. Реализовать следующие алгоритмы умножения матриц
    - Стандартный алгоритм
    - Алгоритм Винограда
    - Оптимизированный алгоритм винограда
2. Проанализировать трудоемкости данных алгоритмов
3. Провести экспериментальный анализ по замерам времени.

1. Аналитическая часть

В данном разделе будут представлены описания алгоритмов, формулы и оценки сложностей алгоритмов

1.1 Описание алгоритмов

Классический алгоритм умножения матриц:

Пусть даны 2 матрицы A и B размерностями m * l и l * n соотвественно:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1l} \\ a_{21} & a_{22} & \cdots & a_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{ml} \end{bmatrix},$$

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

Тогда матрица C размерностью m x n:

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

в которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \ (i = \overline{1..m}, j = \overline{1..n})$$

является их произведением.

Операция умножения двух матриц выполнима только в том случае, если число столбцов первой матрицы равно числу строк второй матрицы. В частности умножение всегда выполнимо, если размерности матриц совпадают и равны m и n соответственно, причем m=n !. Таким образом, из существования матричного произведения AB не следует существование BA.

1.2 Алгоритм умножения Копперсмита-Винограда

Алгоритм Копперсмита—Винограда — алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом (англ.). В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2,3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита—Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

Если рассмотреть 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 2 вектора: $V = (v1, v2, v3, v4)$ и $W = (w1, w2, w3, w4)$

Их скалярное произведение равно:

$$(V * W) = v1 * w1 + v2 * w2 + v3 * w3 + v4 * w4 \ (1)$$

Что эквивалентно

$$(V * M) = (v1 + w2) * (v2 + w1) + (v3 * w4) * (v4 + w3) -$$

$$-v1 * v2 - v3 * v4 - w1 * w2 - w3 * w4 \ (2)$$

Не очевидным остается тот факт, что выражение в правой части формулы 2 допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого стобца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие 5 сложений, а также два сложения.

2. Конструкторская часть

В данном разделе будут размещены схемы алгоритмов и сравнительный анализ рекурсивной и не рекурсивной реализаций.

2.1 Разработка алгоритмов

На рисунках 1 – 11 приведены схемы алгоритмов, демонстрирующие работу

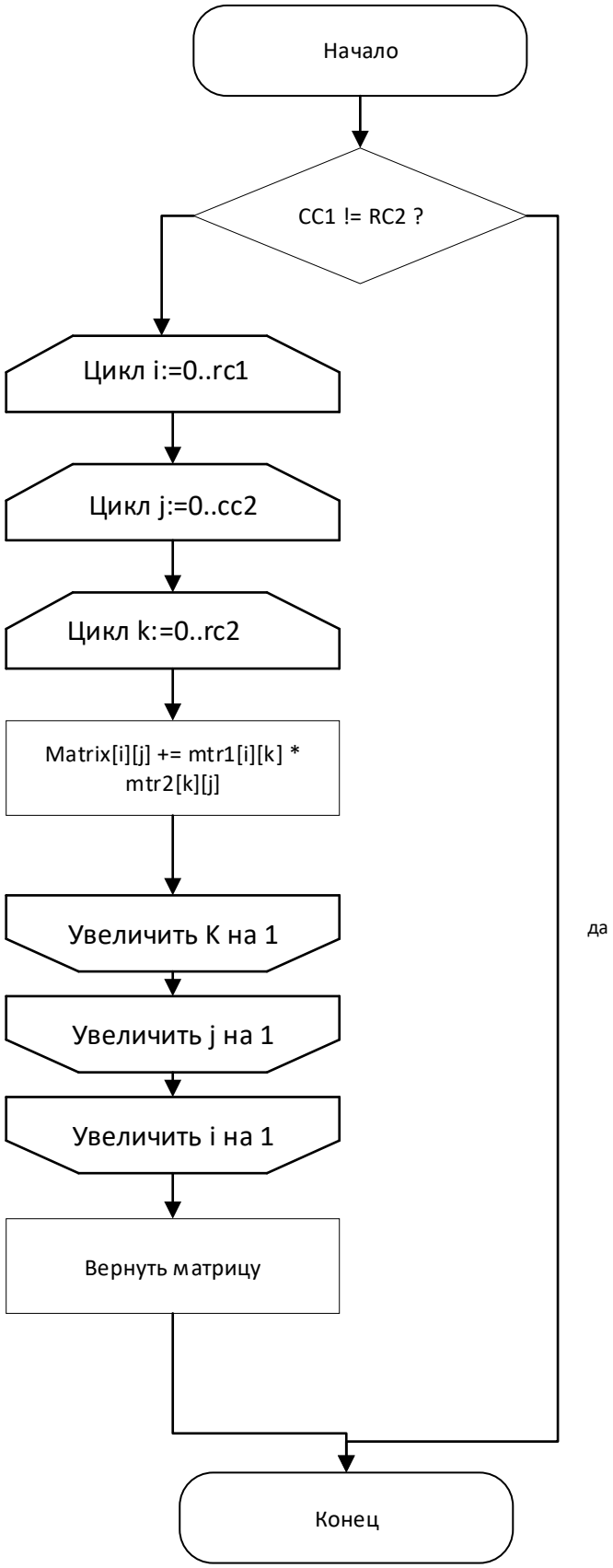

Рисунок 1. Классический алгоритм умножения матриц

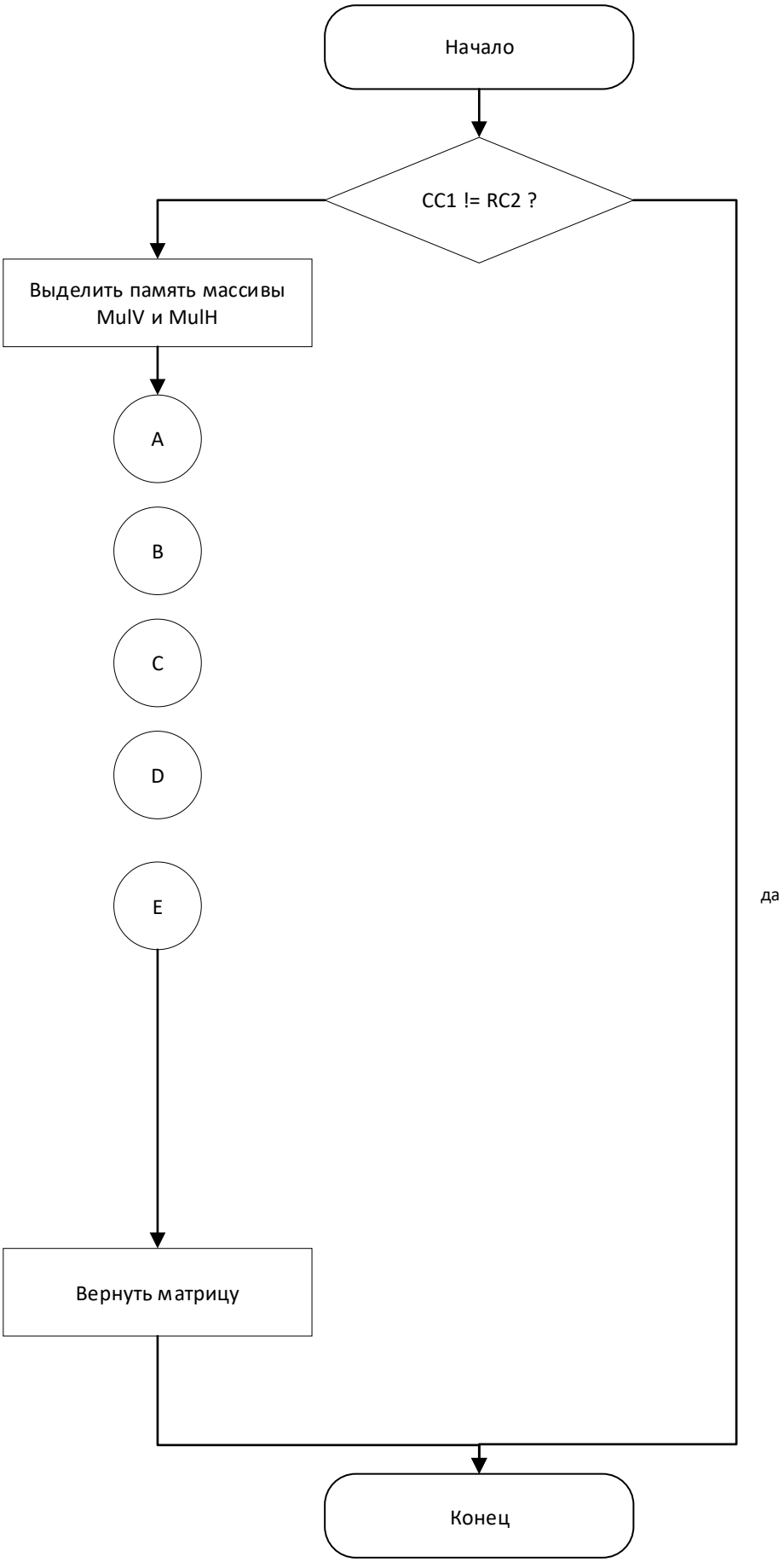

Рисунок 2. Алгоритм умножения Винограда

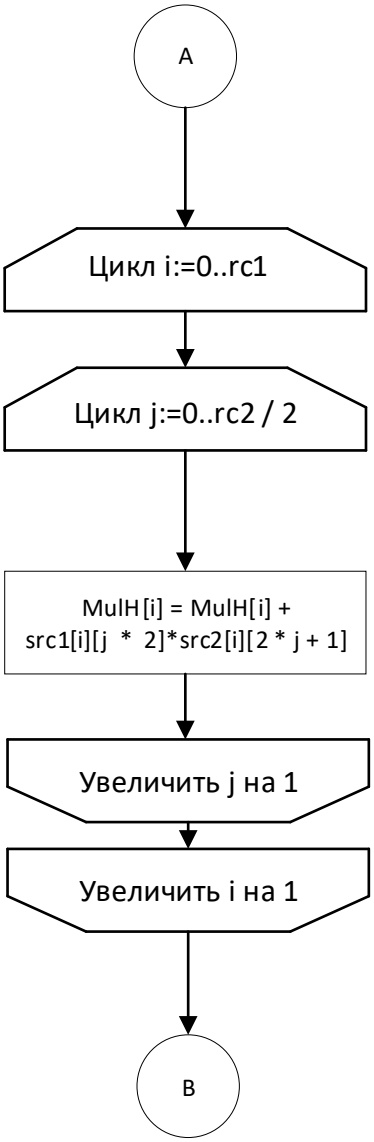

Рисунок 3. Алгоритм умножения Винограда (продолжение)

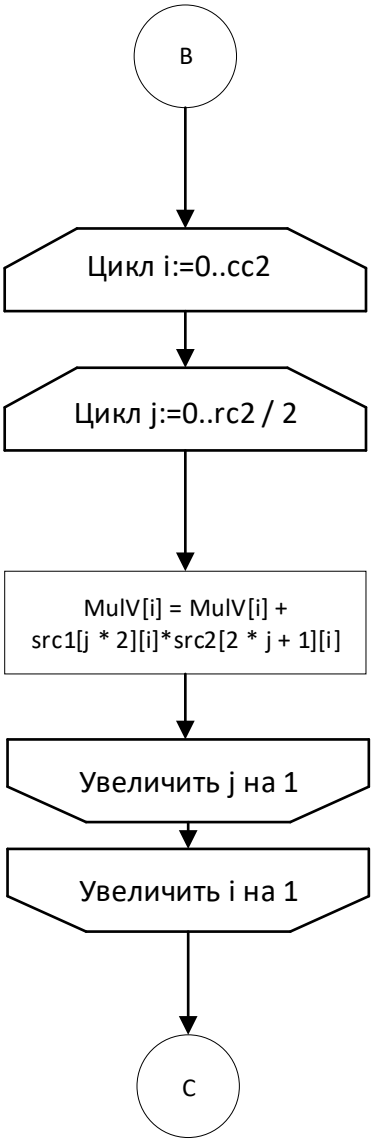

Рисунок 4. Алгоритм умножения Винограда (продолжение)

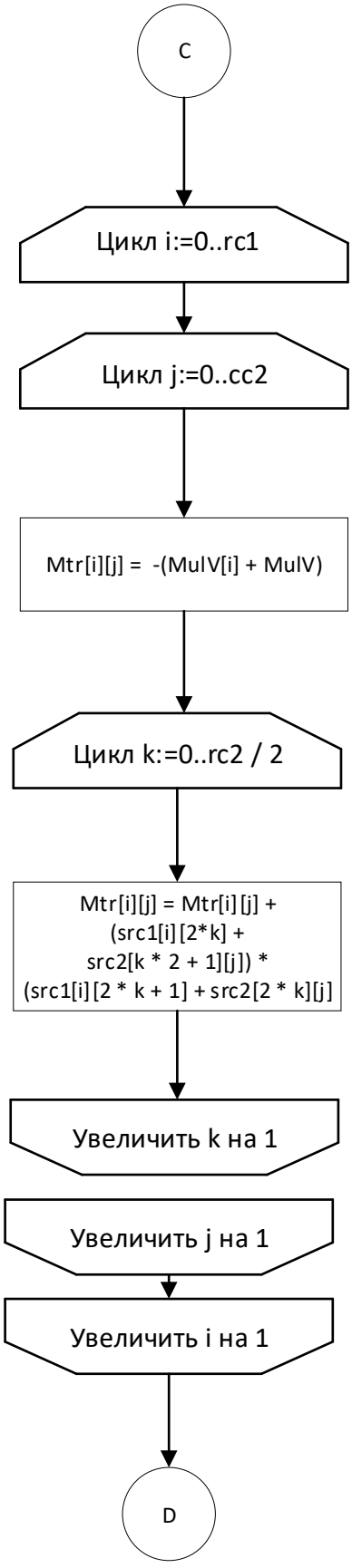

Рисунок 5. Алгоритм умножения Винограда (продолжение)

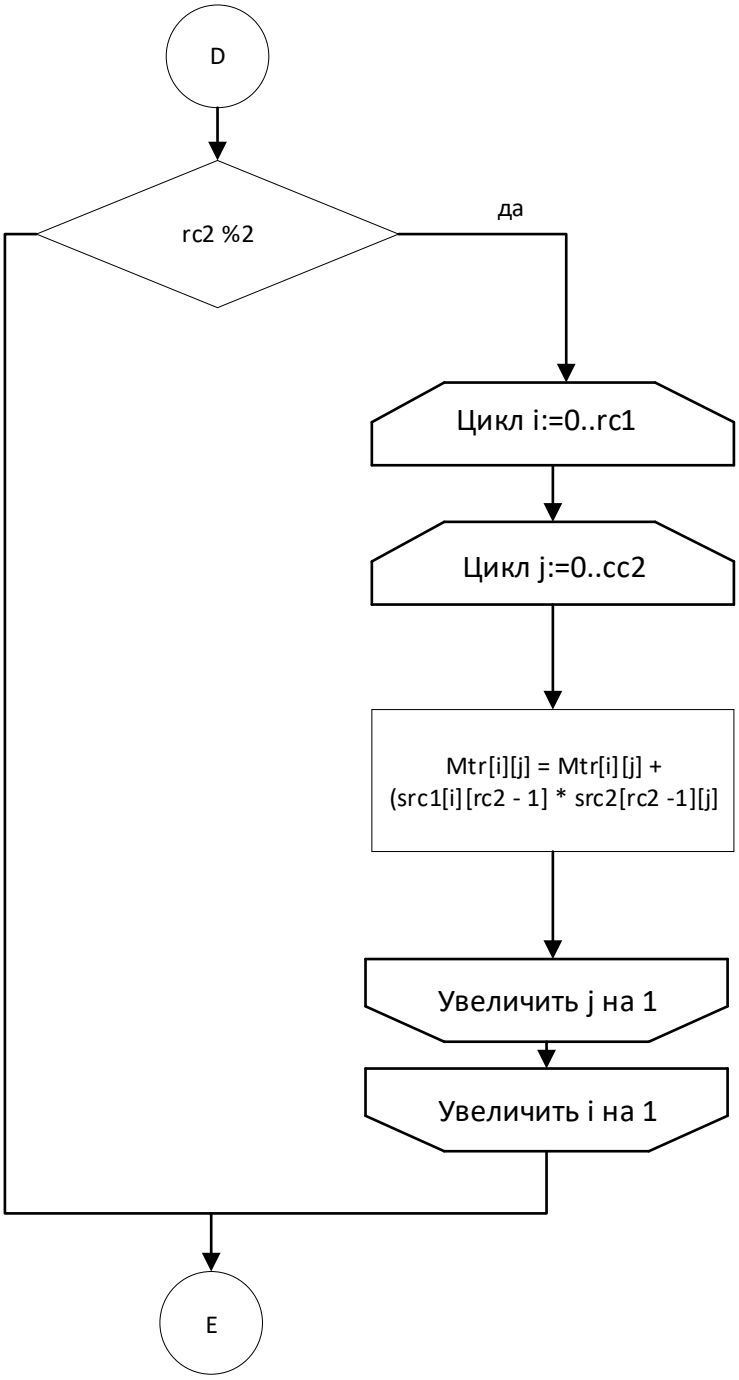

Рисунок 6. Алгоритм умножения Винограда (продолжение)

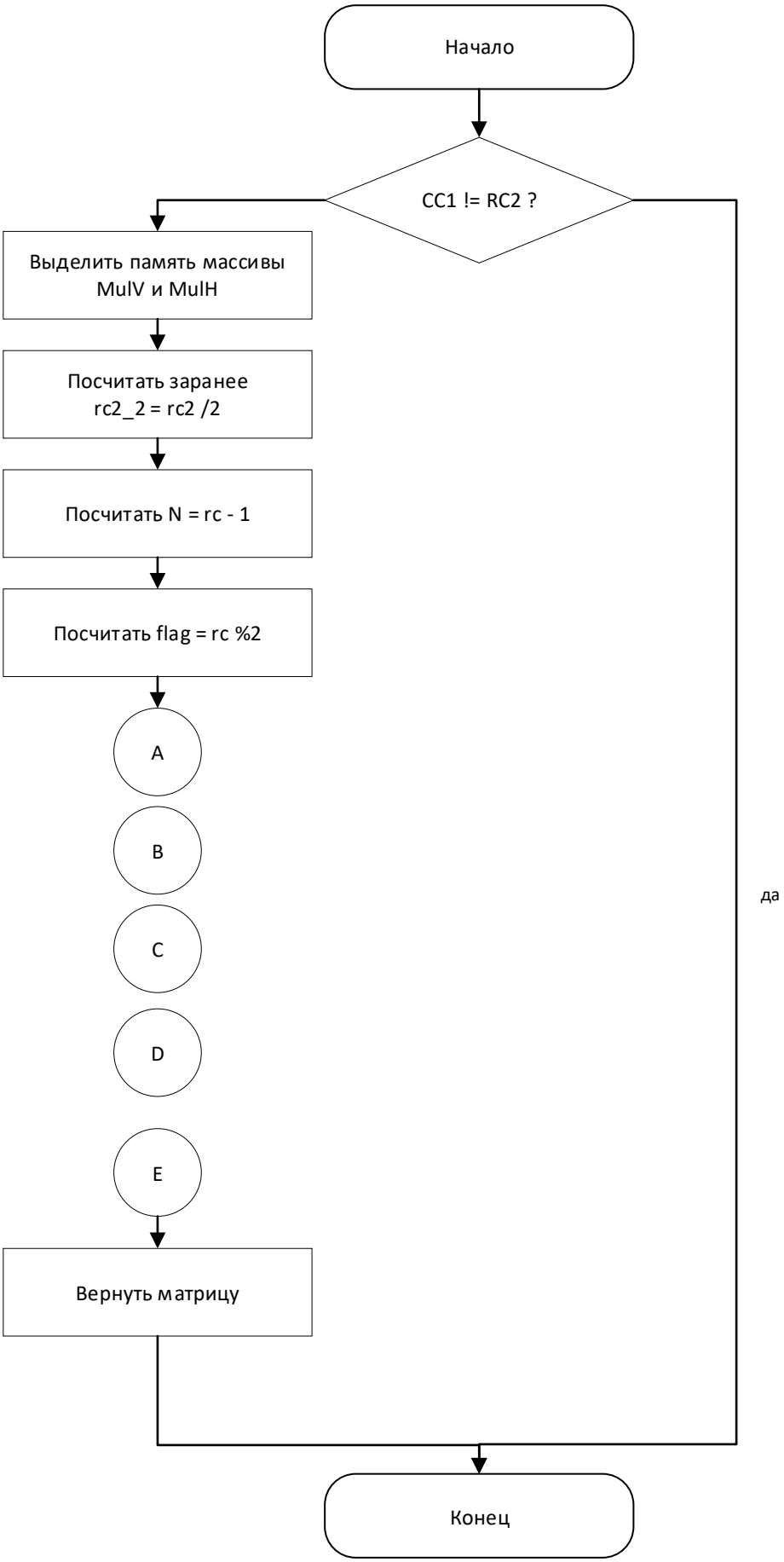

Рисунок 7. Оптимизированный алгоритм умножения Винограда

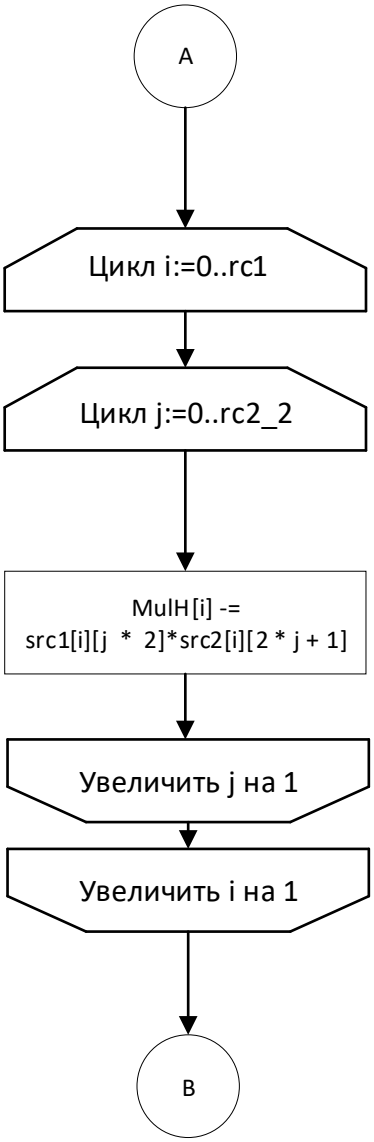

Рисунок 8. Оптимизированный алгоритм умножения Винограда (продолжение)

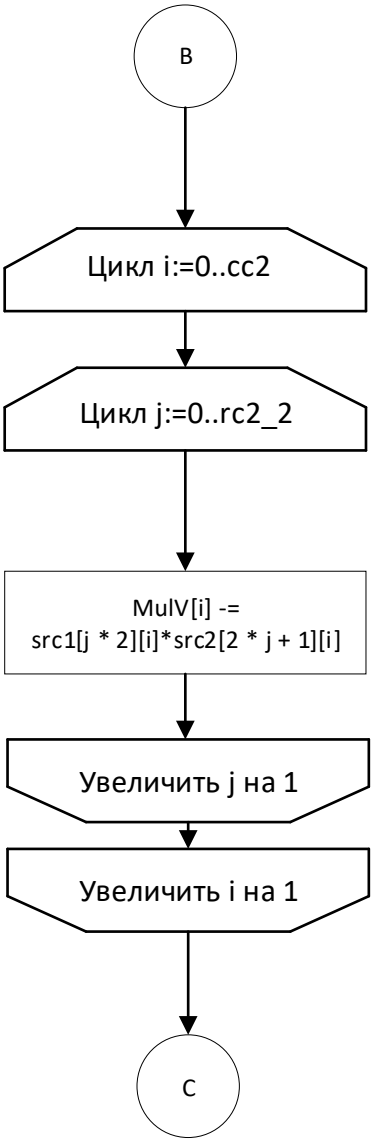

Рисунок 9. Оптимизированный алгоритм умножения Винограда (продолжение)

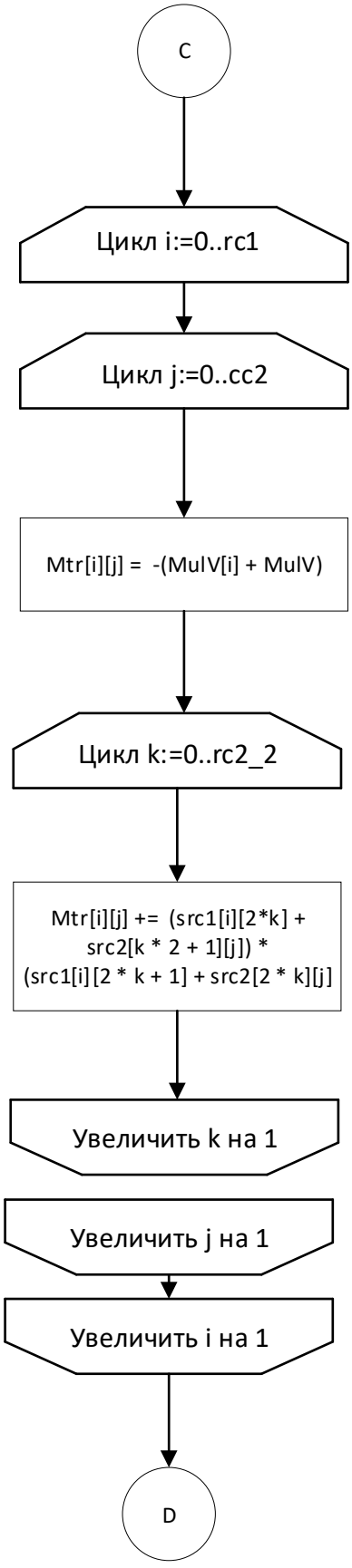

Рисунок 10. Оптимизированный алгоритм умножения Винограда (продолжение)

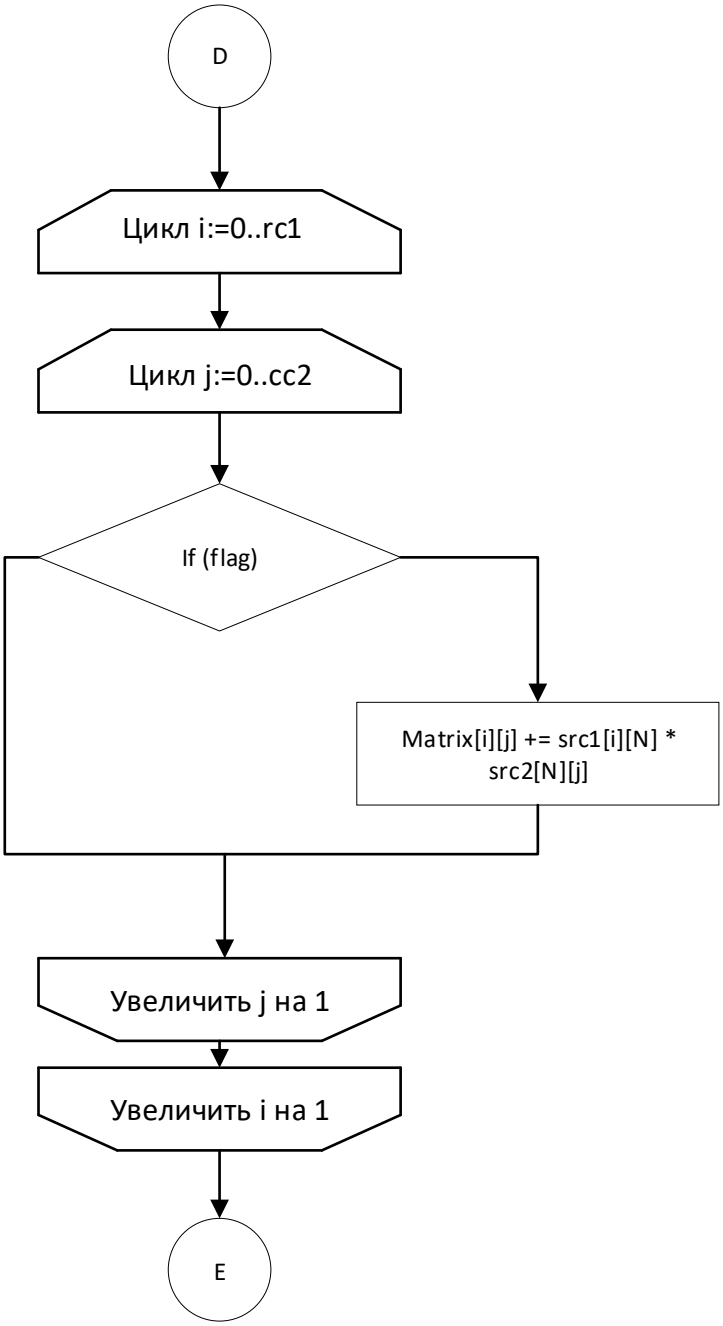

Рисунок 11. Оптимизированный алгоритм умножения Винограда (продолжение)

2.2 Расчет трудоемкости

Пусть заданы 2 матрица размерностями m*n u n * q. Используя выше представленную модель вычислений, произведем расчет трудоемкости алгоритмов умножения матриц.

- Стандартный алгоритм

$$f_{\text{стд}} = 2 + M\left(2 + 2 + Q\left(2 + 2 + N(2 + 8 + 1 + 1 + 1)\right)\right) = 13MNQ + 4MQ + 4M + 2 \approx 13MNQ$$

- Алгоритм Винограда

$$f_I = 2 + M\left(2 + 3 + \frac{N}{2}(3 + 1 + 6 + 2 + 3)\right) = \frac{15}{2}MN + 5M + 2$$

$$f_{II} = 2 + Q\left(2 + 3 + \frac{N}{2}(3 + 1 + 6 + 2 + 3)\right) = \frac{15}{2}QN + 5Q + 2$$

$$f_{III} = 2 + M\left(2 + 2 + Q(2 + 7 + 3 + \frac{N}{2}(3 + 6 + 17))\right) = 13MNQ + 12MQ + 4M + 2$$

$$f_{IIII} = 2 + \begin{cases} 0, & N \text{ четное} \\ 12MQ + 4M + 2, & \text{иначе} \end{cases}$$

$$f_{\text{вин}} = f_I + f_{II} + f_{III} + f_{IIII} = \frac{15}{2}MN + \frac{15}{2}QN + 9M + 8 + 5Q + 13MNQ + 12MQ + \begin{cases} 0, & N \text{ четное} \\ 12MQ + 4M + 2, & \text{иначе} \end{cases} \approx 13MNQ$$

- Оптимизированный Алгоритм Винограда

$$f_{\text{вин}^*} = \frac{10}{2}MN + 4M + 2 + \frac{10}{2}MN + 4M + 2 + \frac{18}{2}MNQ + 12MQ + 4M + 2 \approx 9MNQ$$

Оптимизированный алгоритм Винограда схематично представляет собой обычный алгоритм Винограда, за исключением следующих оптимизаций:

1) Вычисление заранее N – 1 и flag = N % 2
2) Вычисление $\frac{n1}{n2}$ происходит заранее
3) Замена C[i][j] = C[i][j]+… на C[i][j] +=
4) последний цикл для нечетных элементов включен в основной цикл, используя дополнительные операции в случае нечетности

3 Технологическая часть

В данном разделе будут приведены Требования к программному обеспечению, средства реализации, листинг кода и примеры тестирования.

3.1 Требования к программному обеспечению

На вход подаются 2 целочисленные матрицы, на выходе должен возвращаться результат их умножения либо сообщение о невозможности их умножения.

3.2 Средства реализации

В качестве языка программирования был выбран C++ в связи с его широким функционалом и быстротой работы, а так же благодаря привычному для меня синтаксису и семантики языка. Среда разработки - Qt. Время работы процессора замеряется с помощью функции std::chrono().

3.3 Листинг кода

Ниже будут представлены реализации алгоритмов

Реализация классического алгоритма – листинг 1

Реализация алгоритма Винограда – листинг 2

Реализация Оптимизированного алгоритма Винограда – листинг 3

Листинг 1. Реализация классического умножения матриц.

```
int multiplicateMatrix(int ***result, int &rc, int &cc, int** src1, int
**src2, int rc1, int cc1, int rc2, int cc2)
{
    if (cc1 != rc2)
    {
        return ERR_MTR_SIZE;
    }
    if (src1 == NULL || src2 == NULL)
    {
        return ERR_EMPTY_MATRIX;
    }
    rc = rc1;
    cc = cc2;
    int **tmpMtr = allocateMatrix(rc, cc);
    for (int i = 0; i < rc1; i++)
    {
        for (int j = 0; j < cc2; j++)
        {
            for (int k = 0; k < rc1; k++)
            {
                tmpMtr[i][j] = tmpMtr[i][j] + src1[i][k] * src2[k][j];
            }
        }
    }
    *result = tmpMtr;
}
```

Листинг 2. Реализация алгоритма Винограда

```
int multiplicateMatrixVinograd(int ***result, int &rc, int &cc, int**
src1, int **src2, int rc1, int cc1, int rc2, int cc2)
{
    if (cc1 != rc2)
    {
        return ERR_MTR_SIZE;
    }
    if (src1 == NULL || src2 == NULL)
    {
        return ERR_EMPTY_MATRIX;
    }
    rc = rc1;
    cc = cc2;
    int **tmpMtr = allocateMatrix(rc, cc);
    int *MulH = (int*)calloc(rc1, sizeof(int));
    int *MulV = (int*)calloc(cc2, sizeof(int));
    for (int i = 0; i < rc1; i++)
    {
        for (int j = 0; j < rc2 / 2; j++)
        {
            MulH[i] = MulH[i] + src1[i][j * 2] * src1[i][2 * j + 1];
            //cout << MulH[i] << endl;
        }
    }
    for (int i = 0; i < cc2; i++)
    {
        for (int j = 0; j < rc2 / 2; j++)
        {
            MulV[i] = MulV[i] + src2[j * 2][i] * src2[2 * j + 1][i];
            //cout << src2[j * 2][i] * src2[2 * j + 1][i] << endl;
        }
    }
    for (int i = 0; i < rc1; i++)
    {
        for (int j = 0; j < cc2; j++)
        {
            tmpMtr[i][j] = - (MulH[i] + MulV[j]);

            for (int k = 0; k < rc2 / 2; k++)
            {
                tmpMtr[i][j] = tmpMtr[i][j] + (src1[i][2 * k] + src2[2
* k + 1][j]) * (src1[i][2 * k + 1] + src2[2 * k][j]);
            }
        }
    }
    if (rc2 % 2)
    {
        for (int i = 0; i < rc1; i++)
        {
            for (int j = 0; j < cc2; j++)
            {
                tmpMtr[i][j] = tmpMtr[i][j] + src1[i][rc2 - 1] *
src2[rc2 - 1][j];
            }
        }
    }
    *result = tmpMtr;
}
```

Листинг 3. Реализация оптимизированного Алгоритма Винограда

```
int optimizedMultiplication(int ***result, int &rc, int &cc, int**
src1, int **src2, int rc1, int cc1, int rc2, int cc2)
{
    if (cc1 != rc2)
    {
        return ERR_MTR_SIZE;
    }
    if (src1 == NULL || src2 == NULL)
    {
        return ERR_EMPTY_MATRIX;
    }
    int rc2_2 = rc2 >> 1;                                    // opti
    rc = rc1;
    cc = cc2;
    int **tmpMtr = allocateMatrix(rc, cc);
    int *MulH = (int*)calloc(rc1, sizeof(int));
    int *MulV = (int*)calloc(cc2, sizeof(int));
    for (int i = 0; i < rc1; i++)
    {
        for (int j = 0; j < rc2_2; j++)                      // opti
        {
            MulH[i] -= src1[i][j * 2] * src1[i][2 * j + 1];  // opti
        }
    }
    for (int i = 0; i < cc2; i++)
    {
        for (int j = 0; j < rc2_2; j++)                      // opti
        {
            MulV[i] -= src2[j * 2][i] * src2[2 * j + 1][i];  // opti
        }
    }
    int N = rc2 - 1;                                         // opti
    bool flag = rc2 % 2;                                     // opti
    for (int i = 0; i < rc1; i++)
    {
        for (int j = 0; j < cc2; j++)
        {
            tmpMtr[i][j] = MulH[i] + MulV[j];
            for (int k = 0; k < rc2_2; k++)                  // opti
            {
                tmpMtr[i][j] += (src1[i][k * 2] + src2[2 * k + 1][j])
* (src1[i][2 * k + 1] + src2[2 * k][j]);                     // opti
            }
        }
    }
    for (int i = 0; i < rc1; i++)
    {
        for (int j = 0; j < cc2; j++)
        {
            if (flag)                                        // opti
            {
                tmpMtr[i][j] += src1[i][N] * src2[N][j];     // opti
            }
        }
    }
    *result = tmpMtr;
}
```

4 Экспериментальная часть

В данном разделе будут приведены примеры работы программы, постановка эксперимента и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1 Примеры работы

Пример 1:

Матрица A:
1 2 3
4 5 6
Матрица B:
0
0
0

Результат:
0
0
0

Пример 2:

Матрица A:
1 2
4 5
Матрица B:
 0 3
-6 1

Результат:
-12 5
-30 17

4.2 Результаты тестирования

Таблица 1. Результаты тестирования

№	A	B	Ожидаемый результат	Полученный результат
1	Нулевая матрица	Нулевая матрица	Нулевая матрица	Нулевая матрица
2	Нулевая	случайная	нулевая	нулевая
3	Случайная	Нулевая	Нулевая	Нулевая
4	Единичная	Квадратная	B	B
5	Квадратная	Единичная	A	A
6	Размера M x N	Размера Q x N	Ошибка размерностей	Ошибка размерностей
7	Размера M x N	Размера N x M	Размера M x M	Размера M x M

В таблице 1 в столбцах результатов указаны результаты тестирования. 1 столбец – номер тестового случая, 2 и 3 столбцы – исходные матрицы, 4 – ожидаемый результат, 5 – полученный результат. Если в исходных матрицах не указаны размерности – значит, что они одинакового порядка.

Программа успешно прошла все тестовые случаи, все полученные результаты совпали с ожидаемыми.

4.3 Постановка эксперимента по замеру времени

Для произведения замеров времени выполнения реализаций алгоритмов будет использована следующая формула $t = \frac{Tn}{N}$, где t – время выполнения, N – количество замеров. Неоднократное измерение времени необходимо для построения более гладкого графика.

Количество замеров будет взято равным 100.

Тестирование будет проведено на одинаковых входных данных. 1) Матрицы размерностями от 100x100 до 800x800 с шагом 100 для матриц четных размеров и от 101x101 до 801x801 с шагом 100 для матриц с нечетных размеров.

4.4 Сравнительный анализ на материале экспериментальных данных

Ниже приведены графики зависимости временных затрат (в тиках процессора) от размеров входных данных.

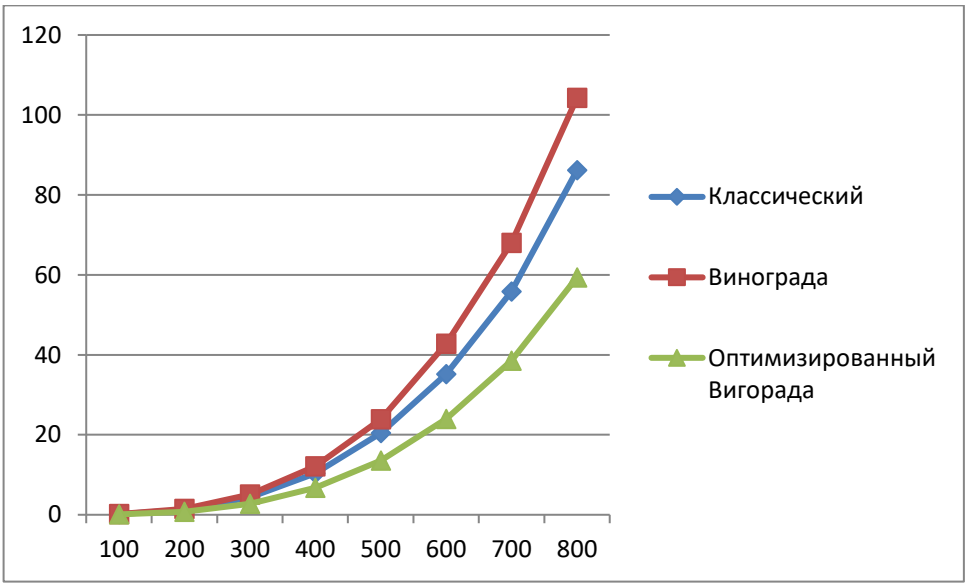

Рисунок 12. Алгоритмы умножения матриц (четных размеров)

На Рисунке 12 видно, что оптимизированный алгоритм Винограда превосходит стандартный алгоритм на ≈ 30 % и неоптимизированный алгоритм почти на ≈ 45%

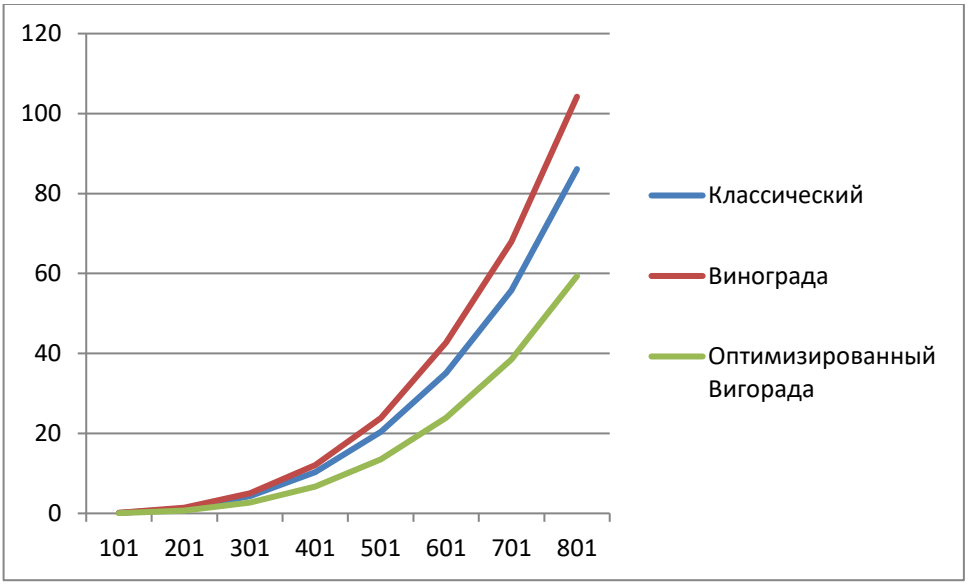

Рисунок 13. Алгоритмы умножения матриц (нечетных размеров)

На рисунке 13 видно, что Оптимизированный алгоритм не потерял свое превосходство, стандартный алгоритм не изменил своего времени работы, а алгоритм Винограда стал работать медленнее .

Заключение

В ходе работы были изучены и реализованы алгоритмы стандартного умножения матриц, алгоритма Винограда и оптимизированного алгоритма Винограда. Был проведен сравнительный анализ перечисленных алгоритмов по трудоемкости и экспериментально выявлена временная разница работы алгоритмов. Классический алгоритм в неоптимизированном виде является более эффективным чем алгоритм винограда, однако после ряда оптимизаций, алгоритм Винограда становится значительно быстрее классического.