Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный технический университет
имени Н. Э. Баумана (национальный исследовательский университет»)

Курс: «Анализ алгоритмов»

Лабораторная работа №2

Тема работы:
«Умножение матриц»

Студент: Волков Е. А.
Преподаватели: Волкова Л. Л.
Строганов Ю. В.
Группа: ИУ7-55Б

2019

Содержание

Введение

Цель лабораторной работы: изучение метода динамического программирования на материале алгоритмов стандартного умножения матриц и умножения матриц по Винограду.

Задачи работы:

1) изучение стандартного алгоритма умножения матриц и алгоритма умножения матриц по Винограду;
2) оптимизация алгоритма Винограда;
3) применение метода динамического программирования для реализации указанных алгоритмов;
4) практические навыки реализации указанных алгоритмов: стандартного алгоритма умножения и умножения по Винограду в двух реализациях: оптимизированной и неоптимизированной;
5) сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени);
6) экспериментальное подтверждение различий во временно́й эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени на варьирующихся размерностях матриц;
7) описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитический раздел

В данном разделе анализируются алгоритмы вычисления произведения матриц по стандартному алгоритму и алгоритму Винограда.

1.1 Описание алгоритмов

Матрица A размера $m \times n$ — это прямоугольная таблица чисел, расположенных в m строках и n столбцах:

$$A = \begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{mn} & a_{m2} & ... & a_{mn} \end{pmatrix}$$

где $a_{ij}(i = 1, \ldots, m; j = 1, \ldots, n)$ — это элементы матрицы A. Первый индекс i — это номер строки, второй индекс j — это номер столбца, на пересечении которых расположен элемент a_{ij} [2].

Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений.

1.1.1 Стандартный алгоритм

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 [1]:

$$\begin{bmatrix} a_{11} & a_{12} & ... & a_{1q} \\ a_{21} & a_{22} & ... & a_{2q} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mq} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & ... & b_{1n} \\ b_{21} & b_{22} & ... & b_{2n} \\ ... & ... & ... & ... \\ b_{q1} & b_{q2} & ... & b_{qn} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11} * b_{11} + ... + a_{1q} * b_{q1} & ... & ... & a_{11} * b_{1n} + ... + a_{1q} * b_{qn} \\ a_{21} * b_{11} + ... + a_{2q} * b_{q1} & ... & ... & a_{21} * b_{1n} + ... + a_{2q} * b_{qn} \\ ... & ... & ... & ... \\ a_{m1} * b_{11} + ... + a_{mq} * b_{q1} & ... & ... & a_{m1} * b_{1n} + ... + a_{mq} * b_{qn} \end{bmatrix}.$$

1.1.2 Алгоритм Винограда

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 $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение равно:

$$V \cdot W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4.$$

Это равенство можно представить так:

$$V \cdot W = (v_1 + w_2) * (w_1 + v_2) + (v_3 + w_4) * (w_3 + v_4) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4.$$

Такой подход позволяет вычислять $-v_1 v_2 - v_3 v_4$ и $-w_1 w_2 - w_3 w_4$ заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй. Также в алгоритме Винограда содержится меньше затратных по времени операций умножения, по сравнению со стандартным[1].

Вывод

В данном разделе были рассмотрены алгоритмы стандартного умножения матриц и алгоритм Винограда, которые решает ту же задачу за меньшее количество операций, путем предварительных вычислений частей произведения и путем использования меньшего количества операции умножения.

2 Конструкторский раздел

В данном разделе описываются шаги по оптимизации алгоритма Винограда, содержатся схемы алгоритмов и сравнительный анализ алгоритмов (рассматриваются стандартный алгоритм умножения матриц, неоптимизированный алгоритм Винограда и оптимизированный алгоритм Винограда).

2.1 Оптимизация алгоритма Винограда

Заполнение массива под горизонтальные (вертикальные) произведения:

1) замена = на +=;
2) замена во внутреннем цикле шага цикла с 1 на 2 ⇒ происходит замена j*2 на j.

Тройной цикл:

1) замена = на +=;
2) замена в цикле по q шага цикла с 1 на 2 ⇒ происходит замена k*2 на k;
3) вычисление $q2 = q - 1$ заранее;
4) использование буфера для накопления результата по циклу q и занесение результата в $res\_matr[i][j]$ после цикла;
5) вычисление горизонтального и вертикального произведения заранее отрицательным.

Условный переход:

1) замена = на +=;
2) замена во внутреннем цикле шага цикла с 1 на 2 ⇒ происходит замена j*2 на j;
3) вычисление q2 = q - 1 заранее.

2.2 Разработка алгоритмов

На рис. 1, 2 и 3 приведены схемы стандартного алгоритма умножения матриц и алгоритма Винограда в оптимизированной и неоптимизированной реализациях.

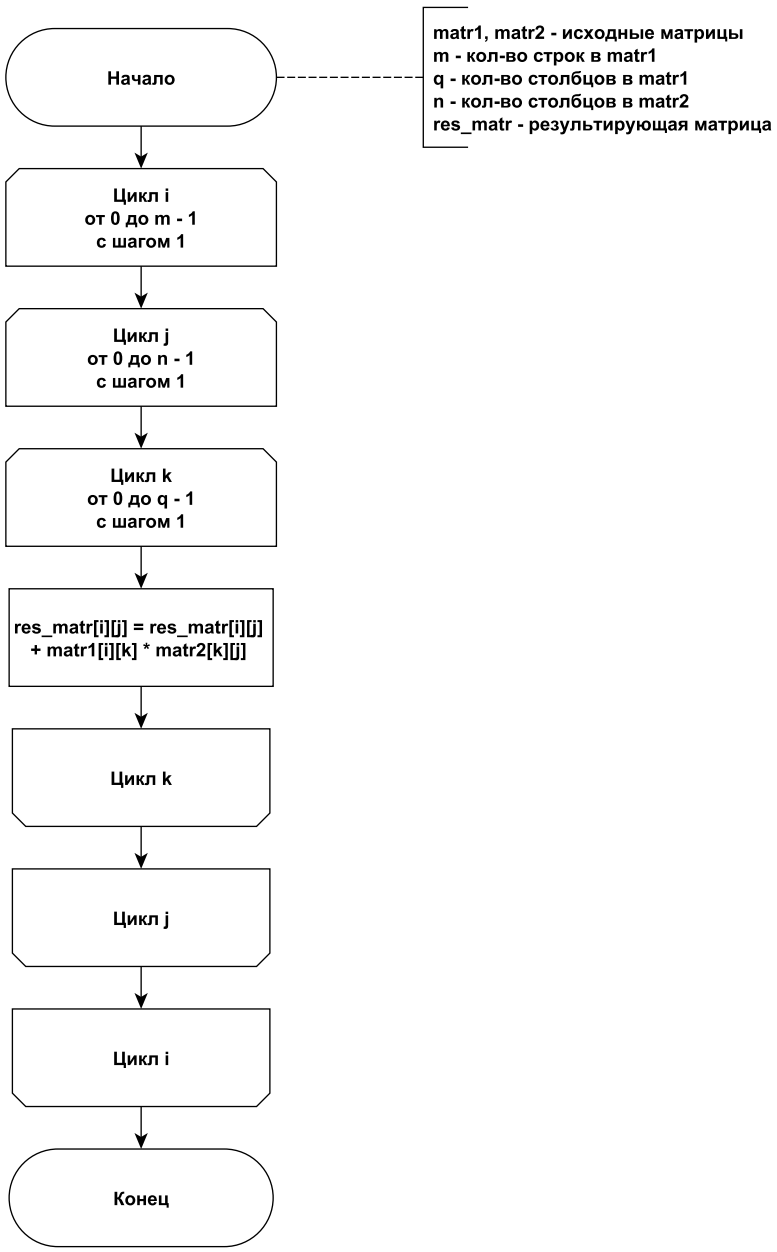

Рис. 1: Схема стандартного алгоритма умножения матриц

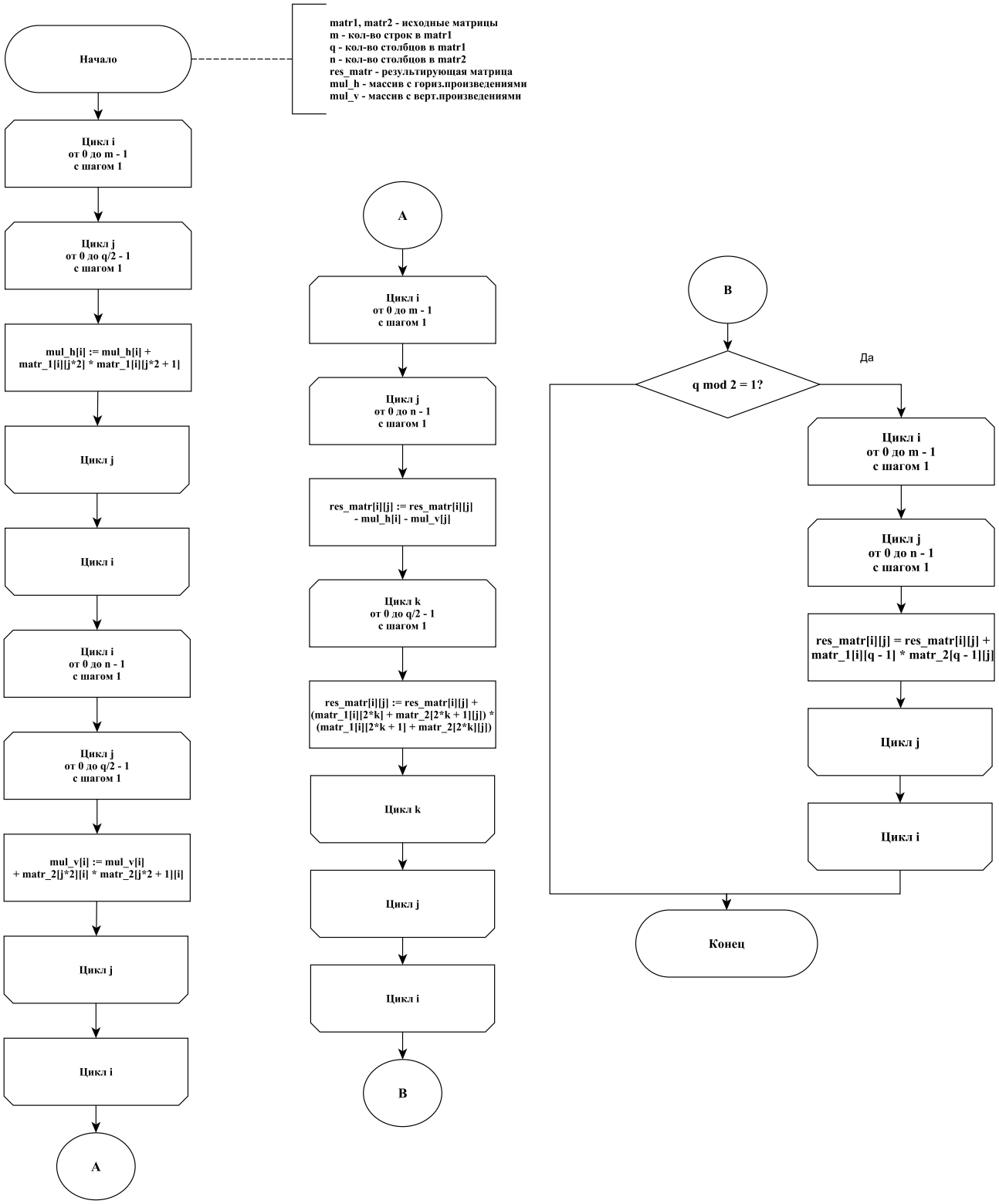

Рис. 2: Схема неоптимизированного алгоритма умножения матриц по Винограду

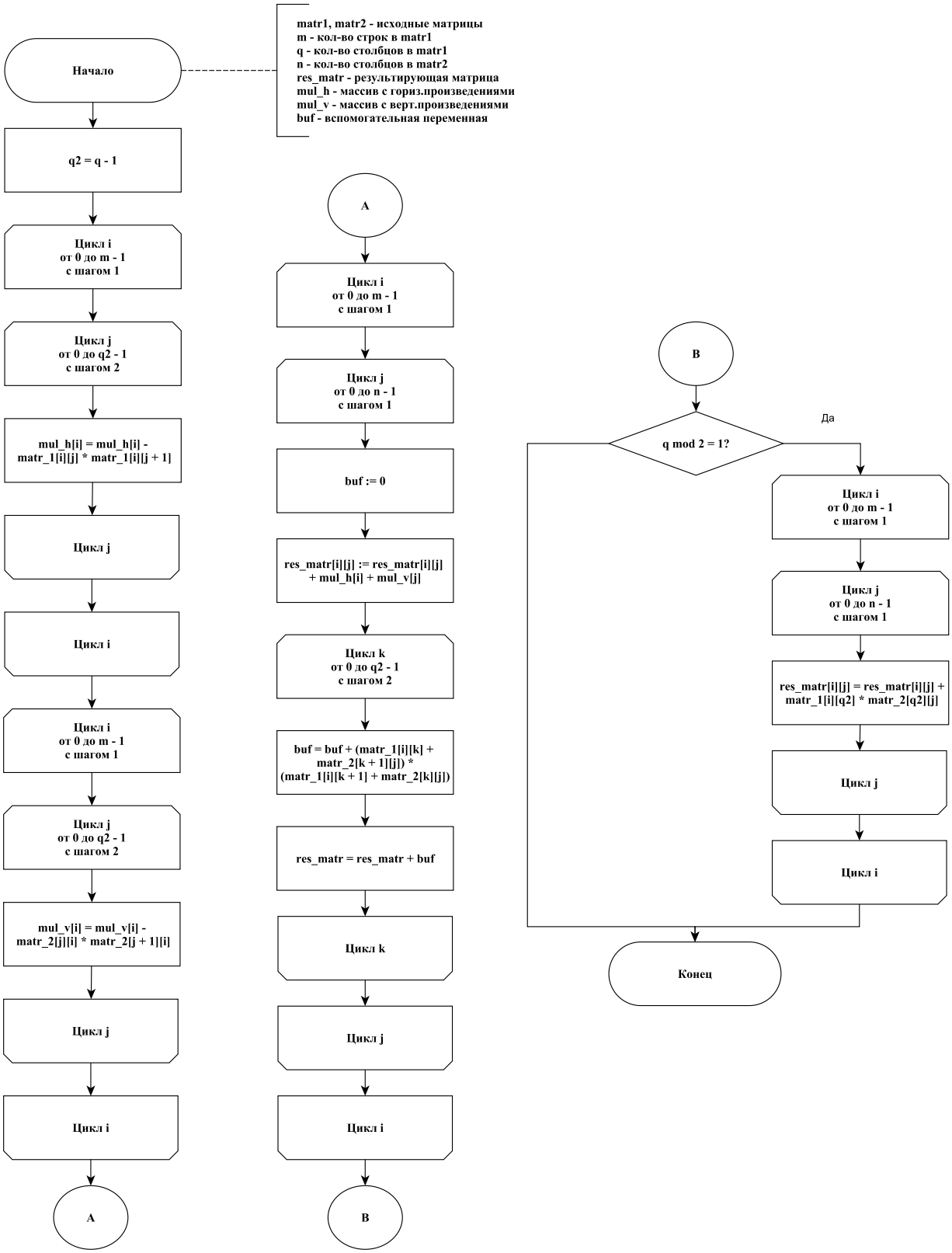

Рис. 3: Схема оптимизированного алгоритма умножения матриц по Винограду

2.3 Сравнительный анализ алгоритмов

Трудоемкость алгоритмов измеряется в количестве необходимых операций. Введем модель вычислений:

1) Базовые операции стоимостью 1: +, -, /, *, =, ==, <=, >= !=, +=, -=, *=, /=, [ ].

2) Стоимость цикла:

$$f_{\text{цикла}} = f_{\text{иниц}} + f_{\text{сравн}} + N(f_{\text{тела}} + f_{\text{инкр}} + f_{\text{сравн}}) = 2 + N(f_{\text{тела}} + 2)$$

3) Стоимость условного перехода примем за 0, стоимость вычисления условия остается

Найдем вычислительную сложность алгоритмов для матриц $A[M \times Q]$ и $B[Q \times N]$.

2.3.1 Стандартный алгоритм

По формуле:

$$f = 2 + M(2 + 2 + Q(2 + 2 + N(2 + \underbrace{8}_{[\,]} + \underbrace{1}_{=} + \underbrace{1}_{+} + \underbrace{1}_{*}))),$$

получаем:

$$f = 13MNQ + 4MQ + 4M + 2.$$

2.3.2 Неоптимизированный алгоритм Винограда

Заполнение массива под горизонтальные произведения:

$$f = 2 + M(2 + 3 + \frac{Q}{2}(3 + \underbrace{6}_{[\,]} + \underbrace{1}_{=} + \underbrace{2}_{+} + \underbrace{3}_{*})) = \frac{15}{2}MQ + 5M + 2.$$

Заполнение массива под вертикальные произведения (аналогично):

$$f = \frac{15}{2}NQ + 5N + 2.$$

Непосредственно вычисление произведения матриц:

$$f = 2 + M(2 + 2 + N(2 + \underbrace{8}_{[\,]} + \underbrace{1}_{=} + \underbrace{1}_{-} + 3 + \frac{Q}{2}(3 + \underbrace{12}_{[]} + \underbrace{1}_{=} + \underbrace{5}_{+} + \underbrace{5}_{*}))) =$$

$$= 13MNQ + 15MN + 4M + 2.$$

Условный переход:

$$f = 2 + \begin{bmatrix} 0, \text{если } Q - \text{четное} \\ 2 + M(2 + 2 + N(2 + \underbrace{8}_{[\,]} + \underbrace{1}_{=} + \underbrace{3}_{+} + \underbrace{1}_{*})), \text{иначе} = \end{bmatrix}$$

$$= 2 + \begin{cases} 0, \text{если } Q - \text{четное} \\ 15MN + 4M + 2, \text{иначе.} \end{cases}$$

Итого:

$$f = 13MNQ + \frac{15}{2}MQ + \frac{15}{2}NQ + 15MN + 9M + 5N + 8 + \begin{cases} 0, \text{если } Q - \text{четное} \\ 15MN + 4M + 2, \text{иначе.} \end{cases}$$

2.3.3 Оптимизированный алгоритм Винограда

Заполнение массива под горизонтальные произведения:

$$f = 2 + M(2 + 2 + \frac{Q}{2}(2 + \underbrace{5}_{[\,]} + \underbrace{1}_{-=} + \underbrace{1}_{+} + \underbrace{1}_{*})) = 5MQ + 4M + 2.$$

Заполнение массива под вертикальные произведения (аналогично горизонтальным):

$$f = 5NQ + 4N + 2.$$

Непосредственно вычисление произведения матриц:

$$f = 1 + 2 + M(2 + 2 + N(2 + \underbrace{4}_{[\,]} + \underbrace{1}_{=} + \underbrace{1}_{+} + \underbrace{1}_{buf} + \underbrace{3}_{matr[i][j]=buf} +$$

$$= 2 + \frac{Q}{2}(2 + \underbrace{8}_{[\,]} + \underbrace{1}_{+=} + \underbrace{4}_{+} + \underbrace{1}_{*}))) = 8MNQ + 14MN + 4M + 3.$$

Условный переход:

$$f = 2 + \begin{cases} 0, \text{если } Q - \text{четное} \\ 2 + M(2 + 2 + N(2 + \underbrace{6}_{[\,]} + \underbrace{1}_{+=} + \underbrace{1}_{*})), \text{иначе} = \end{cases}$$

$$= 2 + \begin{cases} 0, \text{если } Q - \text{четное} \\ 10MN + 4M + 2, \text{иначе.} \end{cases}$$

Итого:

$$f = 8MNQ + 5MQ + 5NQ + 14MN + 8M + 4N + 9 + \begin{cases} 0, \text{если } Q - \text{четное} \\ 10MN + 4M + 2, \text{иначе.} \end{cases}$$

2.3.4 Анализ результатов

Оценивать алгоритмы необходимо по самому быстро растущему слагаемому, т. е. по тому в котором одновременно содержатся три множителя M, N и Q.

Таким образом самым эффективным оказывается модифицированный алгоритм Винограда: его стоимость $\approx 8MNQ$ операций. Стоимость стандартного алгоритма и неоптимизированного алгоритма Винограда составляет $\approx 13MNQ$ операций. Анализ остальных слагаемых стоимостей этих алгоритмов показывает, что стандартный алгоритм эффективнее ($4MQ$ операций против $\frac{15}{2}NQ + \frac{15}{2}MQ + 15MN$ операций для матриц четной размерности и $4MQ$ операций против $\frac{15}{2}NQ + \frac{15}{2}MQ + 30MN$ операций для матриц нечетной размерности).

Вывод

В данном разделе были представлены схемы стандартного алгоритма умножения матриц и алгоритм умножения матриц по Винограду в двух модификациях: оптимизированный и неоптимизированный.

Был произведен аналитический анализ трудоемкости алгоритмов, в ходе которого выяснилось, что самым эффективным является оптимизированный алгоритм Винограда (стоимость $\approx 8MNQ$ операций), менее эффективен стандартный алгоритм (стоимость $\approx 13MNQ + 4MQ$ операций). Самым медленным оказался неоптимизированный алгоритм Винограда (стоимость $\approx 13MNQ\frac{15}{2}NQ + \frac{15}{2}MQ + 15MN$ операций).

3 Технологический раздел

В данном разделе будут описаны требования к программному обеспечению и средства реализации, приведены листинг программы и тестовые данные.

3.1 Требования к программному обеспечению

Входные данные:

1) три целых положительных числа - размерности матриц: M, N и Q.

2) две матрицы размера M x Q и Q x N, заполненные целыми числами

Выходные данные: матрица размера M x N, полученная в результате умножения исходных, с помощью трех алгоритмов умножения матриц: стандартного, неоптимизированного по Винограду, оптимизированного по Винограду.

На рис. 4 приведена функциональная схема вычисления произведения матриц.

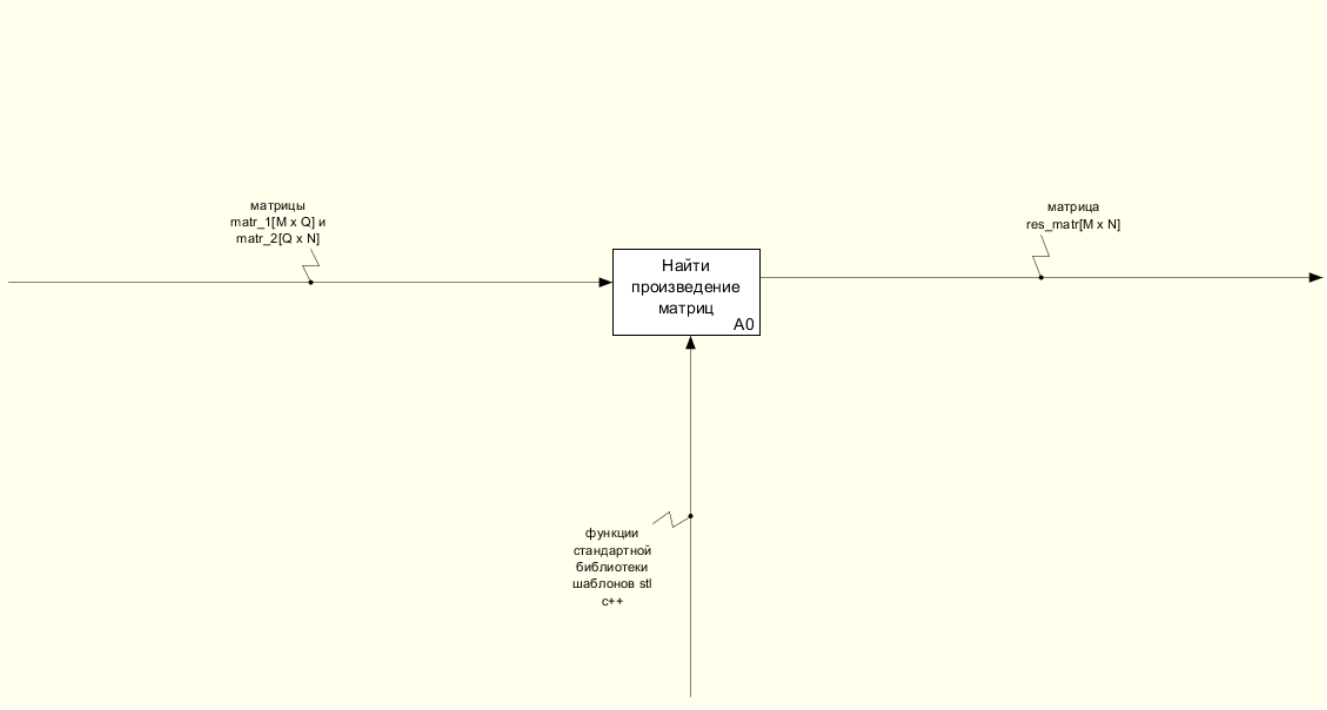

Рис. 4: Функциональная схема вычисления произведения матриц

3.2 Средства реализации

Программа написана на языке C++, т. к. этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов, обладает высокой эффективностью и значительным набором стандартных

классов и процедур. В качестве среды разработки использовался фреймворк QT 5.13.1.

Для обработки матриц был использован стандартный контейнерный класс std::vector.

Для замера времени выполнения программы использовалась библиотека ctime.

3.3 Листинг программы

В листингах 1, 2 и 3 представлены функции стандартного умножения матриц и умножения матриц по Винограду в оптимизированной и неоптимизированной реализациях

Листинг 1: Стандартный алгоритм умножения матриц

```
void mult_matrix_std(Matrix matr_1, Matrix matr_2, Matrix &res_matr) {
    size_t m = matr_1.size();
    size_t q = matr_1[0].size();
    size_t n = matr_2[0].size();

    for(size_t i = 0; i < m; i++) {
        for(size_t j = 0; j < n; j++) {
            for(size_t k = 0; k < q; k++) {
                res_matr[i][j] += matr_1[i][k] * matr_2[k][j];
            }
        }
    }
}
```

Листинг 2: Неоптимизированный алгоритм Винограда

```
void mult_matrix_vinograd(Matrix matr_1, Matrix matr_2, Matrix &res_matr) {
    size_t m = matr_1.size();
    size_t q = matr_1[0].size();
    size_t n = matr_2[0].size();

    Vector mul_h(m, 0);
    Vector mul_v(n, 0);

    for(size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < q/2; j++) {
            mul_h[i] = mul_h[i] + matr_1[i][j*2] * matr_1[i][j*2 + 1];
        }
    }


    for(size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < q/2; j++) {
            mul_v[i] = mul_v[i] + matr_2[j*2][i] * matr_2[j*2 + 1][i];
        }
```

```
    }

    for(size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            res_matr[i][j] = - mul_h[i] - mul_v[j];
            for(size_t k = 0; k < q/2; k++) {
                res_matr[i][j] = res_matr[i][j] + (matr_1[i][2*k] + matr_2
                    [2*k + 1][j]) *
                                    (matr_1[i][2*k + 1] + matr_2[2*k][j]);
            }

        }
    }

    if (q % 2 == 1) {
        for(size_t i = 0; i < m; i++) {
            for (size_t j = 0; j < n; j++) {
                res_matr[i][j] = res_matr[i][j] + matr_1[i][q - 1] * matr_2[
                    q - 1][j];
            }
        }
    }
}
```

Листинг 3: Оптимизированный алгоритм Винограда

```
void mult_matrix_vinograd_optimiz(Matrix matr_1, Matrix matr_2, Matrix &
    res_matr) {
    size_t m = matr_1.size();
    size_t q = matr_1[0].size();
    size_t q2 = q - 1;
    size_t n = matr_2[0].size();

    Vector mul_h(m, 0);
    Vector mul_v(n, 0);

    for(size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < q2; j+=2) {
            mul_h[i] -= matr_1[i][j] * matr_1[i][j + 1];
        }
    }


    for(size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < q2; j+=2) {
            mul_v[i] -= matr_2[j][i] * matr_2[j + 1][i];
        }
    }

    for(size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < n; j++) {
            res_matr[i][j] = mul_h[i] + mul_v[j];
            int buf = 0;
            for(size_t k = 0; k < q2; k+=2) {
                buf += (matr_1[i][k] + matr_2[k + 1][j]) *
                                    (matr_1[i][k + 1] + matr_2[k][j]);
```

```
30                }
31                res_matr[i][j] += buf;
32            }
33        }
34
35
36        if (q % 2 == 1) {
37            for(size_t i = 0; i < m; i++) {
38                for (size_t j = 0; j < n; j++) {
39                    res_matr[i][j] += matr_1[i][q2] * matr_2[q2][j];
40                }
41            }
42        }
43    }
```

3.4 Тестовые данные

Программа должна корректно умножать матрицы при следующих входных данных:

1) матрицы 1×1:

$$[2] \times [3] = [6];$$

2) умножение на нулевую матрицу:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix};$$

3) умножение на единичную матрицу:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix};$$

4) умножение(матрицы 2×2) на матрицу с положительными числами:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix};$$

5) умножение (матрицы 2×2) на матрицу с отрицательными числами:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} -1 & -2 \\ -3 & -4 \end{bmatrix} = \begin{bmatrix} -7 & -10 \\ -15 & -22 \end{bmatrix};$$

6) умножение (матрицы 3×3) на матрицу с положительными числами:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix};$$

7) умножение (матрицы 2×2) на матрицу с отрицательными числами:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{bmatrix} = \begin{bmatrix} -30 & -36 & -42 \\ -66 & -81 & -96 \\ -102 & -126 & -150 \end{bmatrix};$$

8) умножение на прямоугольную матрицу с нечётным количеством столбцов:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix};$$

9) умножение на прямоугольную матрицу с чётным количеством столбцов:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 50 & 60 \\ 114 & 140 \end{bmatrix};$$

10) умножение матриц с большими числами:

$$\begin{bmatrix} 1000 & 2000 & 3000 \\ 4000 & 5000 & 6000 \\ 7000 & 8000 & 9000 \end{bmatrix} \times \begin{bmatrix} 1000 & 2000 & 3000 \\ 4000 & 5000 & 6000 \\ 7000 & 8000 & 9000 \end{bmatrix} = \begin{bmatrix} 30000000 & 36000000 & 42000000 \\ 66000000 & 81000000 & 96000000 \\ 102000000 & 126000000 & 150000000 \end{bmatrix};$$

Результаты работы всех трех алгоритмов должны совпадать.

Вывод

В данном разделе были рассмотрены требования к программному обеспечению, в качестве средств реализации выбраны язык C++ и фреймворк QT версии 5.13.1, приведён листинг программы и тестовые данные.

4 Исследовательский раздел

4.1 Примеры работы

На рис. 5, 6, 7 и 8 приведены примеры работы программы для различных входных данных.

```
Enter number of matrix_1 rows: -5
Incorrect input. Try again: sdfsdf
Incorrect input. Try again: 0
Incorrect input. Try again: 2
Enter number of matrix_1 columns: 2
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
standard:
1 2
3 4
Enter number of matrix 2 columns: sdfger
Incorrect input. Try again: 2
matr[0][0] = er
Incorrect input. Try again: 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
Matrix 1:1 2
3 4
Matrix 2:
1 2
3 4
Result of multiplication of standard:
7 10
15 22
Result of multiplication by Vinograd:
7 10
15 22
Result of multiplication by Vinograd is optimized:
7 10
15 22
```

Рис. 5: Пример работы программы для некорректного ввода

```
Enter number of matrix_1 rows: 1
Enter number of matrix_1 columns: 1
matr[0][0] = 2
standard:
2
Enter number of matrix 2 columns: 1
matr[0][0] = 3
Matrix 1:2
Matrix 2:
3
Result of multiplication of standard:
6
Result of multiplication by Vinograd:
6
Result of multiplication by Vinograd is optimized:
6
```

Рис. 6: Пример работы программы для матриц 1×1

```
Enter number of matrix_1 rows: 2
Enter number of matrix_1 columns: 2
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
standard:
1 2
3 4
Enter number of matrix 2 columns: 2
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
Matrix 1:1 2
3 4
Matrix 2:
1 2
3 4
Result of multiplication of standard:
7 10
15 22
Result of multiplication by Vinograd:
7 10
15 22
Result of multiplication by Vinograd is optimized:
7 10
15 22
```

Рис. 7: Пример работы программы для матриц четной размерности (2×2)

```
Enter number of matrix_1 rows: 2
Enter number of matrix_1 columns: 3
matr[0][0] = 1
matr[0][1] = 2
matr[0][2] = 3
matr[1][0] = 4
matr[1][1] = 5
matr[1][2] = 6
standard:
1 2 3
4 5 6
Enter number of matrix 2 columns: 2
matr[0][0] = 1
matr[0][1] = 2
matr[1][0] = 3
matr[1][1] = 4
matr[2][0] = 5
matr[2][1] = 6
Matrix 1:1 2 3
4 5 6
Matrix 2:
1 2
3 4
5 6
Result of multiplication of standard:
22 28
49 64
Result of multiplication by Vinograd:
22 28
49 64
Result of multiplication by Vinograd is optimized:
22 28
49 64
```

Рис. 8: Пример работы программы для матриц нечетной размерности (2×3 и 3×2)

4.2 Результаты тестирования

Все тесты из раздела 3.4 успешно пройдены.

4.3 Постановка эксперимента

Необходимо выполнить следующие замеры времени:

1. Сравнить время умножения матриц стандартным алгоритмом и алгоритмом Винограда в оптимизированной и неоптимизированной модификациях на матрицах размером от 100×100 до 1000×1000 с шагом 100.

2. Провести аналогичное сравнение на матрицах размером от 101×101 до 1001×1001 с шагом 100.

4.4 Сравнительный анализ на материале экспериментальных данных

На рис. 9 приводятся графики сравнения времени вычисления алгоритмов при матрицах четной размерности.

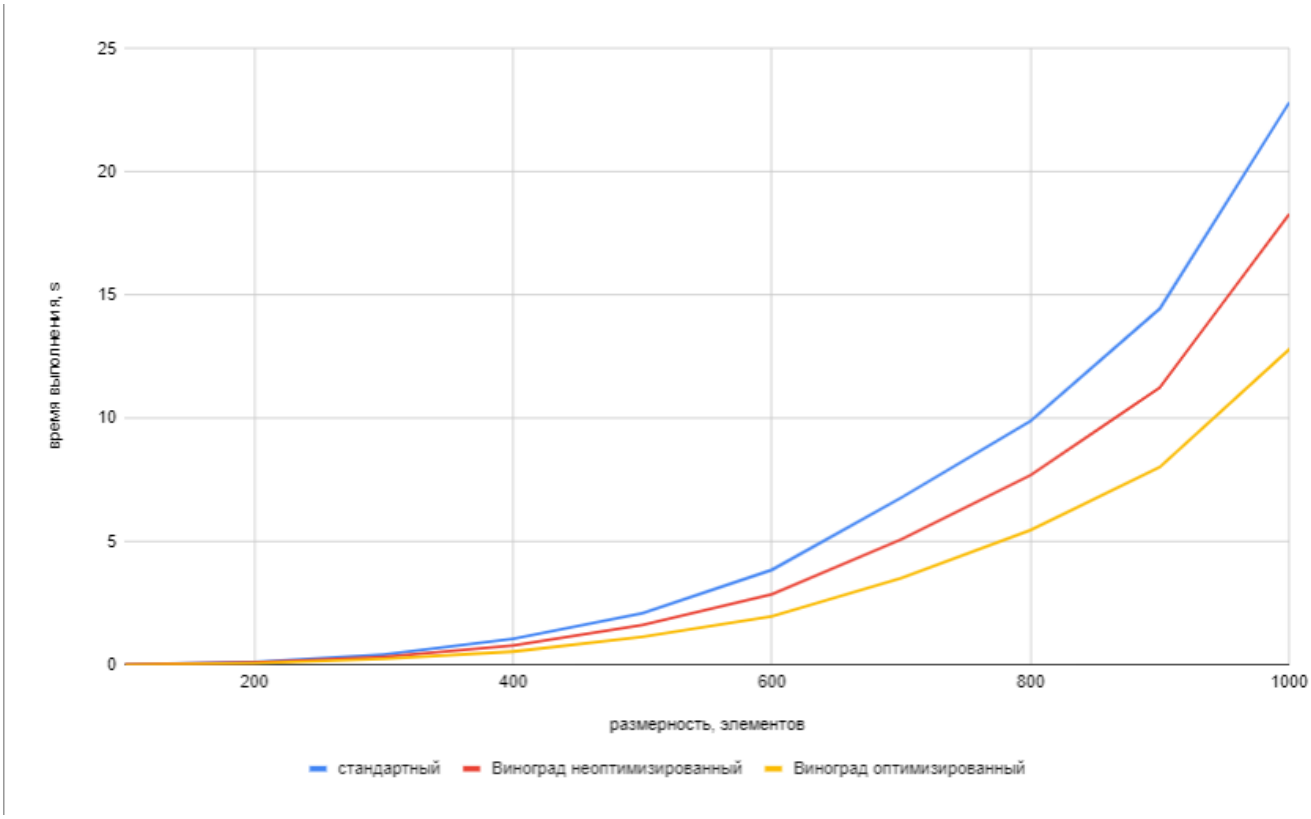

Рис. 9: Сравнение времени выполнения алгоритмов на матрицах четной размерности

Как видно, алгоритм оптимизированный алгоритм Винограда выигрывает по быстродействию (на 30% быстрее неоптимизированного алгоритма Винограда и на 45% быстрее стандартного алгоритма на матрицах размером 1000×1000). Хуже себя показал неоптимизированный алгоритм Винограда (на 15% быстрее стандартного алгоритма на матрицах размером 1000×1000). Медленнее всех отработал стандартный алгоритм.

На рис. 10 приводятся графики сравнения времени вычисления алгоритмов при матрицах нечетной размерности.

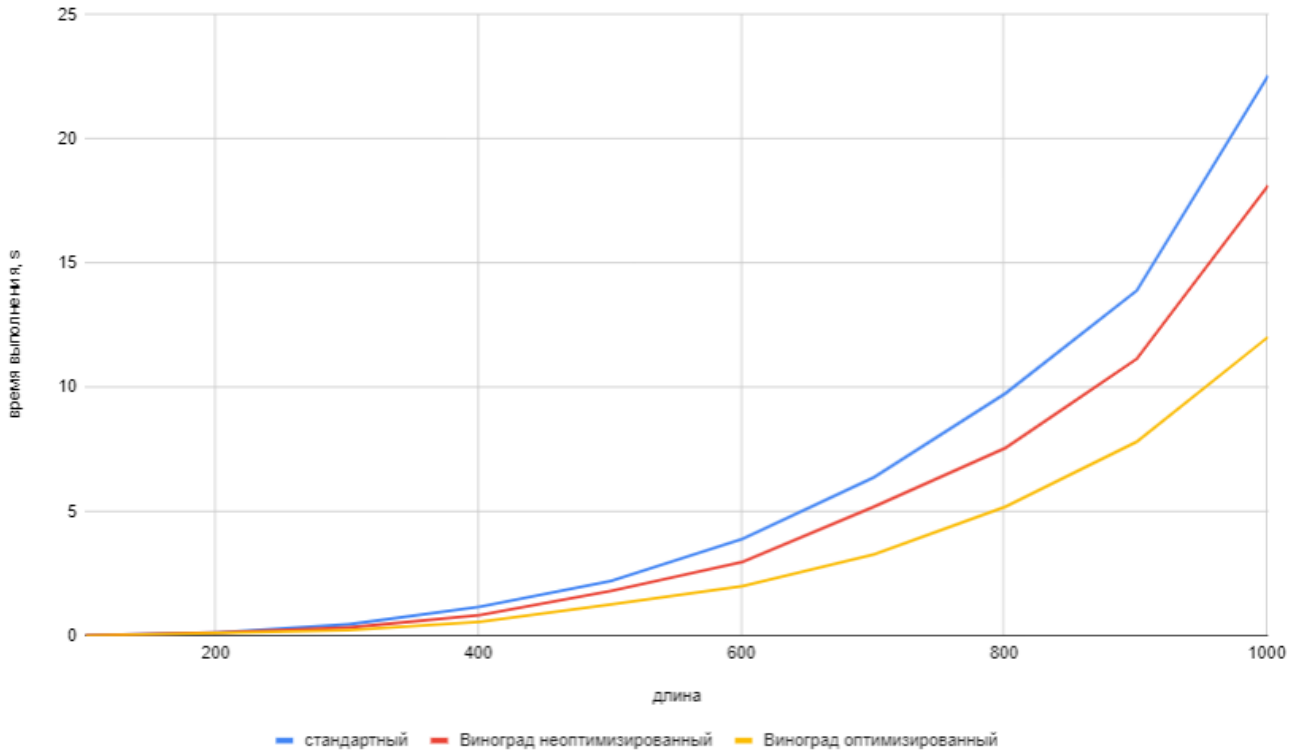

Рис. 10: Сравнение времени выполнения алгоритмов на матрицах нечетной размерности

Как видно из этих зависимостей, результаты данного замера идентичны результатам предыдущего,эффективность алгоритмов практически не меняется при нечетных размерностях исходных матриц.

Результаты замеров лишь частично совпали с результатами анализа трудоемкости алгоритмов. Оптимизированный алгоритм Винограда, как и ожидалось, оказался самым быстрым, однако стандартный алгоритм уступил в быстродействии неоптимизированному алгоритм Винограда. Такое несоответствие связано, скорее всего с тем, что в разделе 2.3 мы приняли за 1 трудоёмкость операций, трудоемкость которых на деле отличается (например у операций сложения и умножения).

Вывод

В результате тестирования программой были успешно пройдены все заявленные тесты. Эксперименты замера времени показали, что самым эффективным по времени выполнения является оптимизированный алгоритм Винограда, а самым убыточным стандартный алгоритм. Результаты замеров времени не совсем совпали с результатами анализа трудоемкости алгоритмов - стандартный алгоритм, вопреки ожиданиям оказался медленнее неоптимизированного алгоритма Винограда.

Заключение

В ходе работы было выполнено следующее:

1) изучены стандартный алгоритм умножения матриц и алгоритм умножения матриц по Винограду;
2) оптимизирован алгоритм Винограда
3) применен метод динамического программирования для реализации указанных алгоритмов;
4) получены практические навыки реализации указанных алгоритмов: стандартного алгоритма умножения и умножения по Винограду в двух реализациях: оптимизированной и неоптимизированной;
5) проведен сравнительный анализ алгоритмов по затрачиваемым ресурсам (времени);
6) экспериментально подтверждено различие во временно́й эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени на варьирующихся размерностях матриц;
7) описаны и обоснованы полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

Аналитический анализ трудоемкости алгоритмов показал, что самым эффективным является оптимизированный алгоритм Винограда (сложность $\approx 8MNQ$ операций), сложности стандартного алгоритма и неоптимизированного алгоритма Винограда оказались практически одинаковыми (сложность $\approx 13MNQ$ операций).

Экспериментальный анализ эффективности алгоритмов по времени показал, что самым эффективным является оптимизированный алгоритм Винограда(на 30% быстрее неоптимизированного алгоритма Винограда и на 45% быстрее стандартного алгоритма на матрицах размером 1000×1000). Хуже себя показал неоптимизированный алгоритм Винограда (на 15% быстрее стандартного алгоритма на матрицах размером 1000×1000). Медленнее всех отработал стандартный алгоритм.

Результаты аналитического и эксперементального анализа совпали лишь частично: оптимизированный алгоритм Винограда оказался самым быстрым, а стандартный алгоритм, вопреки ожиданиям, оказался медленнее неоптимизированного Винограда. Такое несоответствие связано, скорее всего с тем, что в разделе 2.3 мы приняли за 1 трудоёмкость операций, трудоемкость которых на деле отличается (например у операций сложения и умножения).

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.-М.:Техносфера, 2009.

[2] Матрицы и определители [Электронный ресурс]. – Режим доступа: http://matematika.electrichelp.ru/matricy-i-opredeliteli/, свободный – (07.10.2019)