Государственное образовательное учреждение высшего профессионального образования
«Московский Государственный Технический Университет им Н. Э. Баумана.»

Отчет
По лабораторной работе №2
По курсу «Анализ Алгоритмов»

Тема: «Исследование сложности алгоритмов умножения матриц»

Студент: Губайдуллин Р.Т.
Группа: ИУ7-51

Преподаватели: Волкова Л.Л.
Строганов Ю.В.

Постановка задачи:

В ходе лабораторной работы выполнить:
1. Изучить алгоритмы умножения матриц: базовый и алгоритм Винограда;
2. Модернизировать алгоритм Винограда;
3. Сделать теоретическую оценку алгоритмов умножения;
4. Реализовать три алгоритма:
   - Базовый;
   - Алгоритм Винограда;
   - Модернизированный алгоритм Винограда
5. Сравнить время работы алгоритмов умножения матриц.

Описание алгоритмов.

Базовый алгоритм умножения

Для вычисления произведения двух матриц A и B каждая строка матрицы A умножается на каждый столбец матрицы B. Затем сумма данных произведений записывается в соответствующую ячейку результирующей матрицы.

Пример умножения двух матриц:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} * \begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix} = \begin{bmatrix} a*A + b*C + c*E & a*B + b*D + c*F \\ d*A + e*C + f*E & d*B + e*D + f*F \end{bmatrix}$$

Листинг: Базовый алгоритм умножения матрицы.

```
void Matrix::ClassicMultiplication(Matrix &m1, Matrix &m2, Matrix &m3) {
    for (int i = 0; i < m1.col; ++i) {
        for (int j = 0; j < m2.line; ++j) {
            m3.matr[i][j] = 0;
            for (int k = 0; k < m1.line; ++k)
                m3.matr[i][j] += m1.matr[i][k] * m2.matr[k][j];
        }
    }
    return;
}
```

Входные данные: m1 - первая матрица, m2 - вторая матрица.
Выходные данные: m3 - результирующая матрица.

Алгоритм Винограда.

Алгоритм Винограда считается более эффективным благодаря сокращению количества операций умножения. Результат умножения двух матриц представляет собой скалярное произведение соответствующих строки и столбца. Можно заметить, что такое умножение позволяет выполнить заранее часть работы:

$U = (u_1, u_2, u_3, u_4)$

$V = (v_1, v_2, v_3, v_4)$

$U * V = u_1 * v_1 + u_2 * v_2 + u_3 * v_3 + u_4 * v_4$

Это равенство можно переписать в виде:

$$U * V = (u_1 + v_2) * (v_1 + u_2) + (u_3 * v_4) * (u_3 + v_4) - u_1 * u_2 - u_3 * u_4 - v_1 * v_2 - v_3 * v_4$$

При этом значения $u_1 * u_2,\ u_3 * u_4,\ v_1 * v_2,\ v_3 * v_4$ можно рассчитать заранее.

Однако под массивы данных коэффициентом требуется дополнительная память. В случае нечетного количества столбцов первой матрицы требуются дополнительные вычисления.

Листинг: Алгоритм Винограда.

```
void Matrix::VinogradMultiplication(Matrix &m1, Matrix &m2, Matrix &result) {
    int d = m1.line / 2;

    // Вычисление rowFactors для m1
    for (int i = 0; i < m1.col; ++i) {
        m1.rowFactor[i] = m1.matr[i][0] * m1.matr[i][1];
        for (int j = 1; j < d; ++j)
            m1.rowFactor[i] += m1.matr[i][2 * j] * m1.matr[i][2 * j + 1];
    }

    // Вычисление columnFactor для m2
    for (int i = 0; i < m2.line; ++i) {
        m2.columnFactor[i] = m2.matr[0][i] * m2.matr[1][i];
        for (int j = 1; j < d; ++j)
            m2.columnFactor[i] += m2.matr[2 * j][i] * m2.matr[2 * j + 1][i];
    }

    // Вычисление матрицы result
    for (int i = 0; i < m1.col; ++i) {
        for (int j = 0; j < m2.line; ++j) {
            result.matr[i][j] = -m1.rowFactor[i] - m2.columnFactor[j];

            for (int k = 0; k < d; ++k)
                result.matr[i][j] += (m1.matr[i][2 * k] + m2.matr[2 * k + 1][j]) *(m1.matr[i][2 * k + 1] + m2.matr[2 * k]
[j]);
        }
    }

    if (m1.line % 2 == 1)
        for (int i = 0; i < m1.col; ++i)
            for (int j = 0; j < m2.line; ++j)
                result.matr[i][j] += m1.matr[i][m1.line - 1] * m2.matr[m1.line - 1][j];

    return;
}
```

Входные данные: m1 - первая матрица, m2 - вторая матрица.
Выходные данные: result - результирующая матрица.

Улучшенный алгоритм Винограда

Улучшения:

- Добавление буфера buf для уменьшения количества обращений к результирующей матрице;
- Замена $buf = -m1.rowFactor[i] - m2.columnFactor[j];$ на $buf -= m1.rowFactor[i] + m2.columnFactor[j];$ для уменьшения количества операций с трех $(=, -, -)$ до двух $(-=, +)$.
- Избавление от циклов, в которых добавлялись члены в случае нечетной размерности матрицы. Замена их (в основном цикле) на тернарное выражение, которым инициализируются буфер при $n\%2 = 1:\ buf = (flag?m1.matr[i][m1.line - 1] * m2.matr[m2.line - 1][j] : 0);$

Листинг: Модифицированный алгоритм Винограда.

```
void Matrix::VinogradImprovedMultiplication(Matrix &m1, Matrix &m2, Matrix &result) {
    int d = m1.line / 2;
    bool flag = m1.line % 2 == 1;
    double buf;

    // Вычисление rowFactors для m1
    for (int i = 0; i < m1.col; ++i) {
        m1.rowFactor[i] = m1.matr[i][0] * m1.matr[i][1];
        for (int j = 1; j < d; ++j)
            m1.rowFactor[i] += m1.matr[i][2 * j] * m1.matr[i][2 * j + 1];
    }

    // Вычисление columnFactor для m2
    for (int i = 0; i < m2.line; ++i) {
        m2.columnFactor[i] = m2.matr[0][i] * m2.matr[1][i];
        for (int j = 1; j < d; ++j)
            m2.columnFactor[i] += m2.matr[2 * j][i] * m2.matr[2 * j + 1][i];
    }

    // Вычисление матрицы result
    for (int i = 0; i < m1.col; ++i) {
        for (int j = 0; j < m2.line; ++j) {
            buf = (flag ? m1.matr[i][m1.line - 1] * m2.matr[m2.line - 1][j] : 0);
            buf -= m1.rowFactor[i] + m2.columnFactor[j];
            for (int k = 0; k < d; ++k)
                buf += (m1.matr[i][2 * k] + m2.matr[2 * k + 1][j]) * (m1.matr[i][2 * k + 1] + m2.matr[2 * k][j]);
            result.matr[i][j] = buf;
        }
    }
    return;
}
```

Входные данные: m1 - первая матрица, m2 - вторая матрица.
Выходные данные: result - результирующая матрица.

Теоретическая оценка.

Базовый алгоритм умножения матриц

$$f_a = 2 + n * (2 + 2 + t * (2 + 2 + 10m)) = 10mnt + 4nt + 4n + 2 \sim O(n^3)$$

Алгоритм Винограда

f_a = 2 * [2 + m * (2 + 5 + 2 + 2 + (N/2 - 1) * (3 + 6 + 6))] + [2 + m * (2 + 2 + n * (2 + 4 + 3 + 2 + N/2 * (2 + 10 + 10)))] + [2 + 2 + m * (2 + 2 + n * (2 + 8 + 5))]

Последнее слагаемое входит в трудоемкость худшего случая (когда общая размерность - нечетная).
В лучшем случае последнее слагаемое = 2 (операции проверки на четность).

Трудоемкость алгоритма:
Худший случай: $11Nmn + 41mn + 6m + 10$
Лучший случай: $11Nmn + 26mn - 2m + 8$

Модифицированный алгоритм Винограда

f_a = 2 * [2 + m * (2 + 5 + 2 + 2 + (N/2 - 1) * (2 + 5 + 5))] + [2 + m * (2 + 2 + n * (2 + [4 + 3 + 2]/[2] + 2 + 2 + 2 + N/2 * (2 + 8 + 19) + 3))]

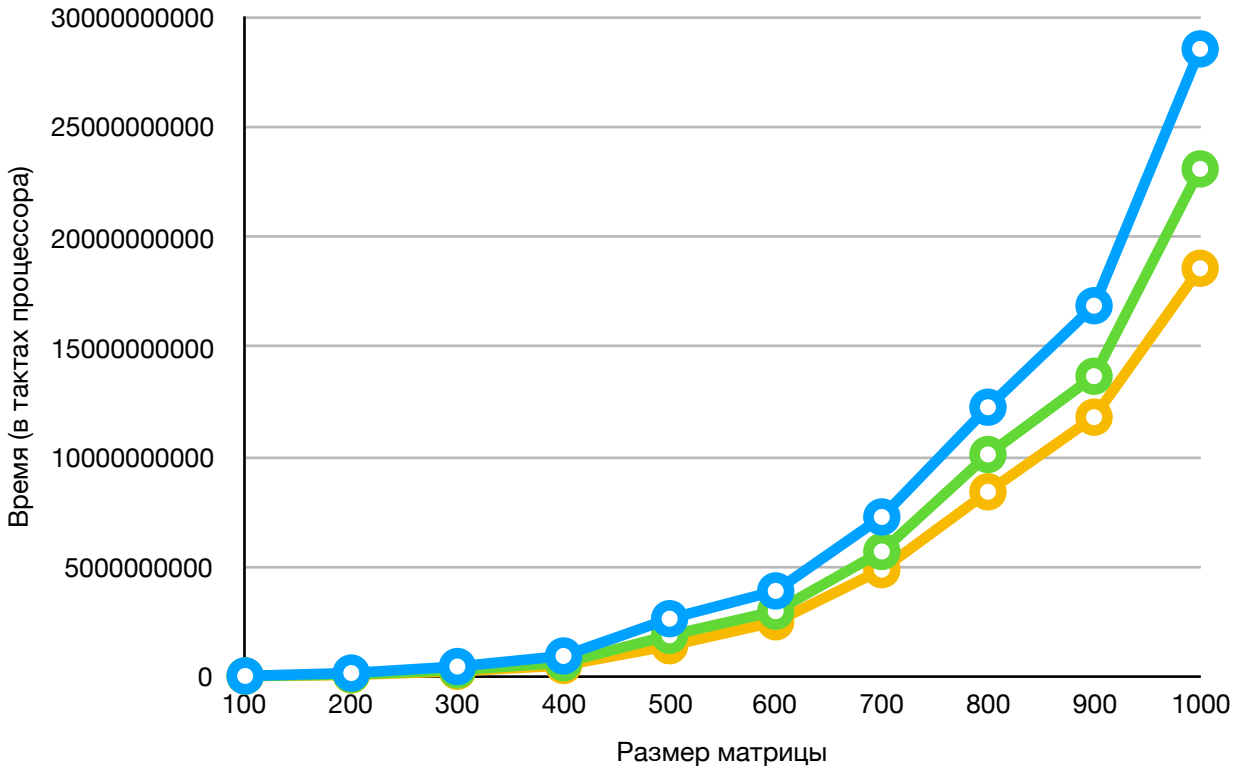
Трудоемкость алгоритма:
Худший случай: $10Nmn + 32mn + 2m + 6$
Лучший случай: $10Nmn + 25mn + 2m + 6$

Эксперимент

В качестве эксперимента произведены десять замеров времени каждого из трех алгоритмов умножения матриц с размерами $100 * 100, \ldots 1000 * 1000$ с шагом 100; $101 * 101 \ldots 1001 * 1001$ с шагом 100. Время измеряется в тактах процессора.

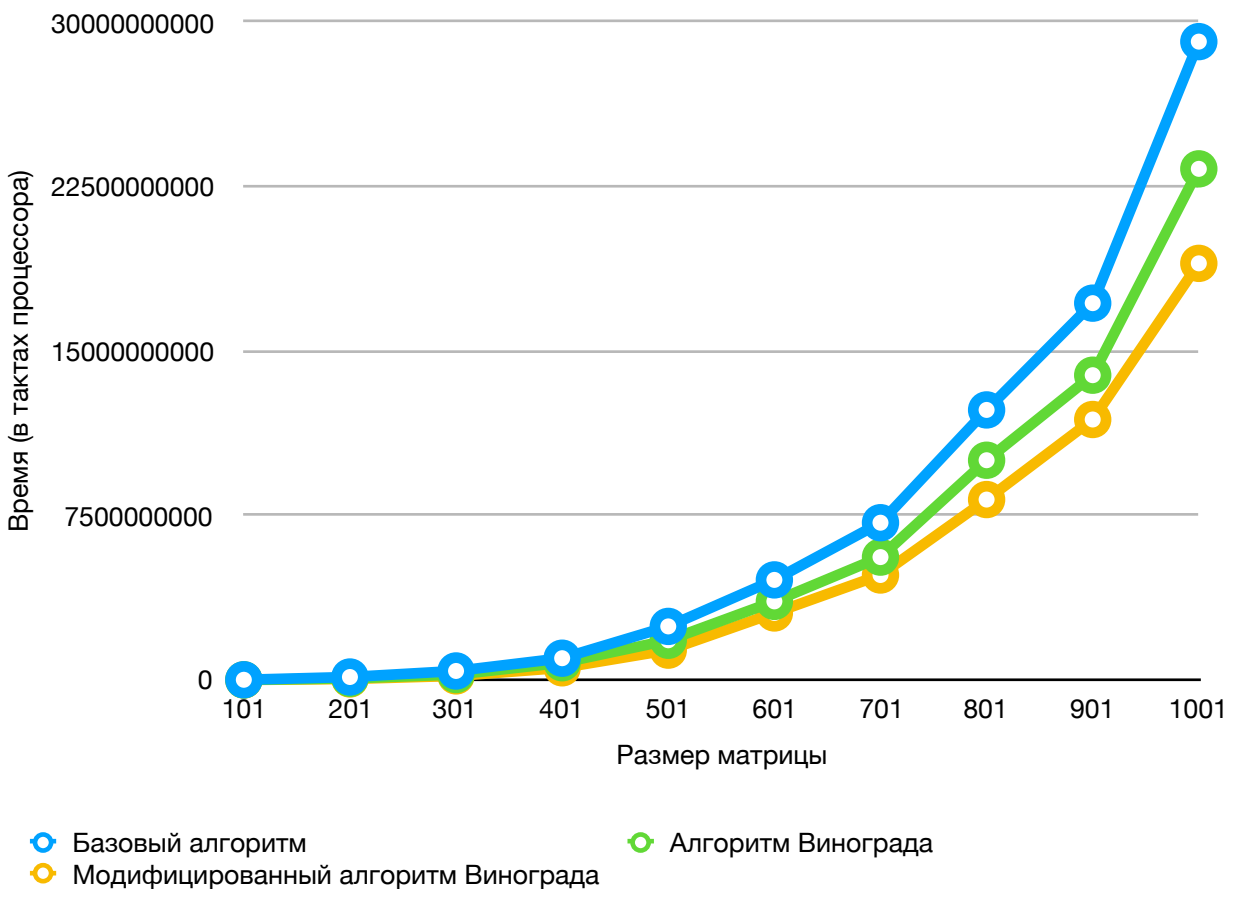
Матрица четной размерности

Размерность матрицы	Базовый алгоритм	Алгоритм Винограда	Модифицированный алгоритм Винограда
100x100	16100159	11396871	8743605
200x200	147219581	82987322	61495156
300x300	444158507	288414791	228034987
400x400	926736499	617592003	493645635
500x500	2624256914	1866420696	1393580714
600x600	3889751416	2965106118	2482419636
700x700	7258480045	5685497795	4858299281
800x800	12257558989	10104316309	8404661759
900x900	16882578924	13670193185	11804895752
1000x1000	28579981384	23114317626	18587474115



Матрица нечётной размерности

Размерность матрицы	Базовый алгоритм	Алгоритм Винограда	Модифицированный алгоритм Винограда
101x101	18316671	12203854	9084025
201x201	144296110	81825724	55308717
301x301	419919531	257840651	194325867
401x401	1000852424	794011347	563365977
501x501	2449194448	1818276994	1368470482
601x601	4568250688	3589578048	3062059710
701x701	7173195762	5606188319	4791526109
801x801	12311223214	10019509698	8227897651
901x901	17175267401	13893266693	11874964830
1001x1001	29088905057	23293540642	18987920049



Выводы

В результате проведенных испытаний алгоритмов умножения матриц установлено, что

1. Алгоритм Винограда выигрывает в быстродействии с определенного порога размерности (~300-400) и быстродействие растет по сравнению с размером матрицы;
2. В классическом алгоритме разница времени выполнения умножения между матрицами с размерностью, отличающимися на 1, незначительна, однако в алгоритме Винограда и улучшенном алгоритме Винограда разница заметна из-за дополнительных вычислений при нечетном количестве элементов.

Заключение

В ходе лабораторной работе я изучил алгоритмы умножения матриц: базовый, Винограда и улучшенный Винограда. Рассчитал теоретическую оценку данным алгоритмам. Программно реализовал и протестировал алгоритмы на быстродействие.