Государственное образовательное учреждение высшего
профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №2

Отчет на тему:
«Умножение Матриц»

Квасников Александр Владимирович

ИУ7-56

Москва 2018

Оглавление

1. Аналитическая часть

1.1 Постановка задачи

В задачи лабораторной работы входит:

1. Изучение и реализация алгоритмов умножения матриц: стандартный и алгоритм Винограда

2. Улучшение реализации алгоритма Винограда

3. Теоретическая оценка алгоритмов умножения матриц

5. Сравнение времени работы алгоритмов умножения матриц

1.2 Описание алгоритмов

Стандартный (базовый) алгоритм умножения матриц

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

Алгоритм Винограда

Алгоритм Винограда - более эффективный алгоритм умножения матриц, использующий предварительную обработку для уменьшения количества операций.

Так как умножение матриц представляет собой скалярное произведение строк и столбцов:

$$U=(u1,u2,u3,u4),\ W=(w1,w2,w3,w4)$$

$$U \cdot W = u1w1 + u2w2 + u3w3 + u4w4$$

можно, представив правую часть последнего уравнения в виде

$$U \cdot W = (u1 + w2)(u2 + w1) + (u3 + w4)(u4 + w3) - u1u2 - u3u4 - w1w2 - w3w4$$

, предварительно рассчитать часть $u1u2 - u3u4 - w1w2 - w3w4$ для каждой строки первой матрицы и для каждого столбца второй.

2. Конструкторская часть

В конструкторской части описан ход разработки алгоритмов, а также сравнение рекурсивной и итеративной реализаций

2.1 Разработка алгоритмов

Для упрощения разработки алгоритмов были нарисованы их схемы:

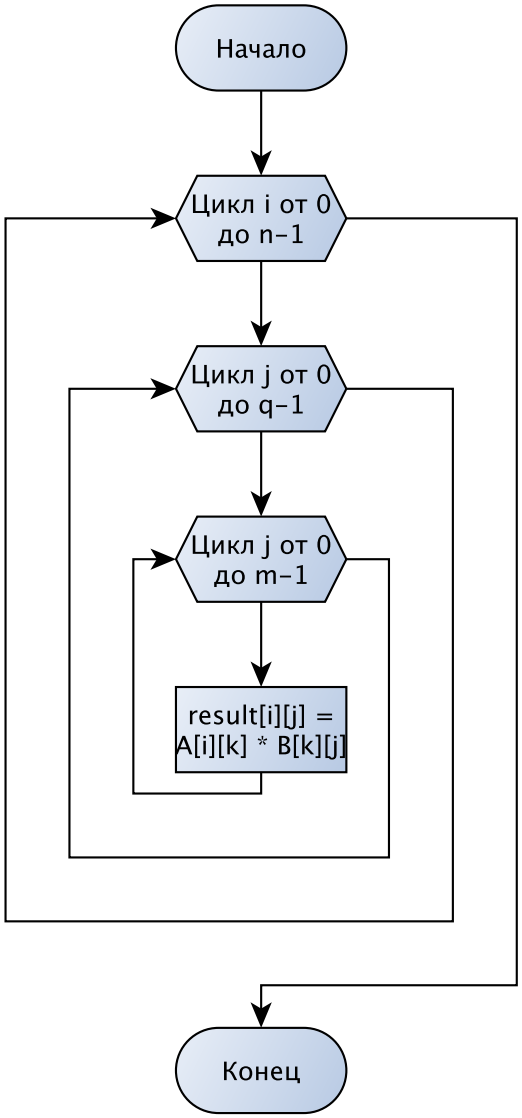


Рис. 1.1 Схема базового алгоритма умножения матриц

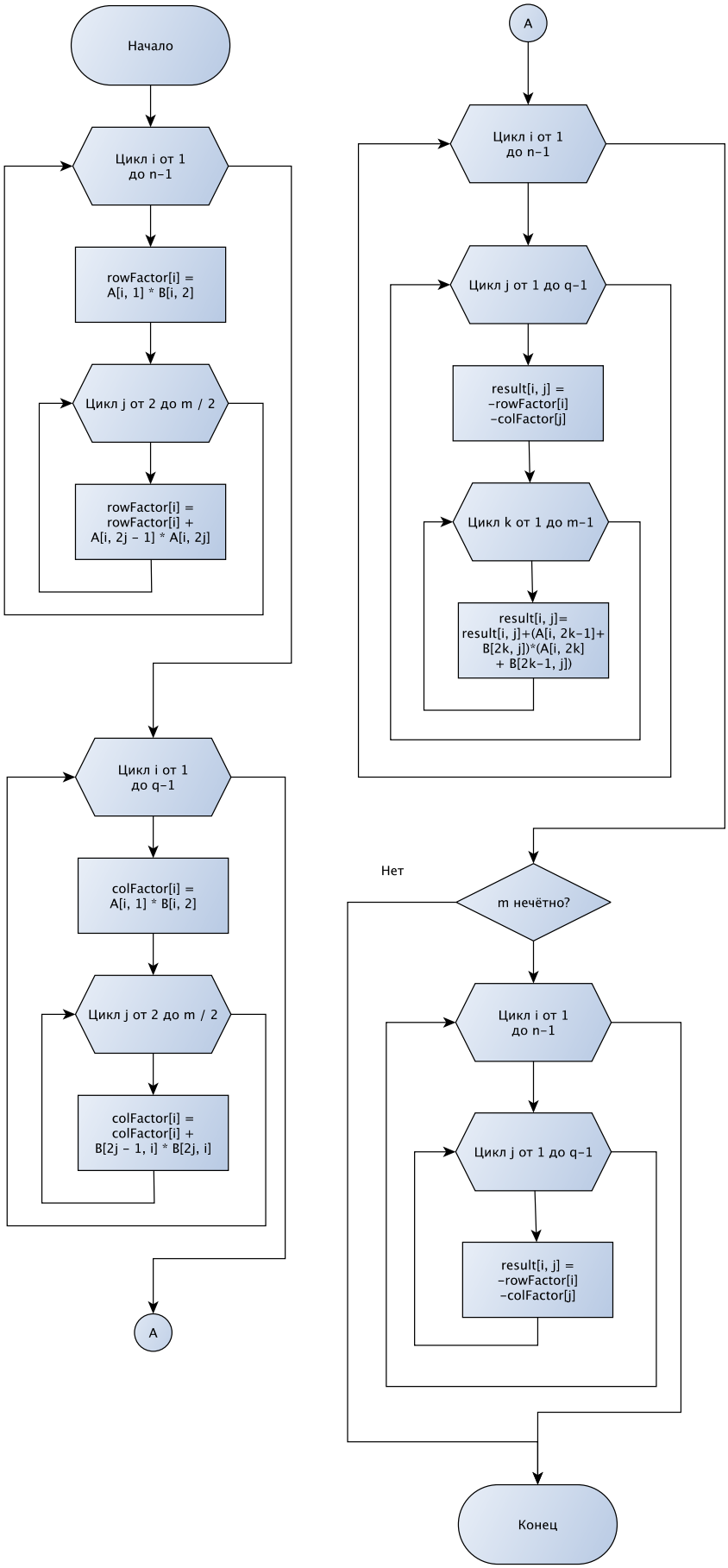


Рис. 1.2 Схема алгоритма Винограда

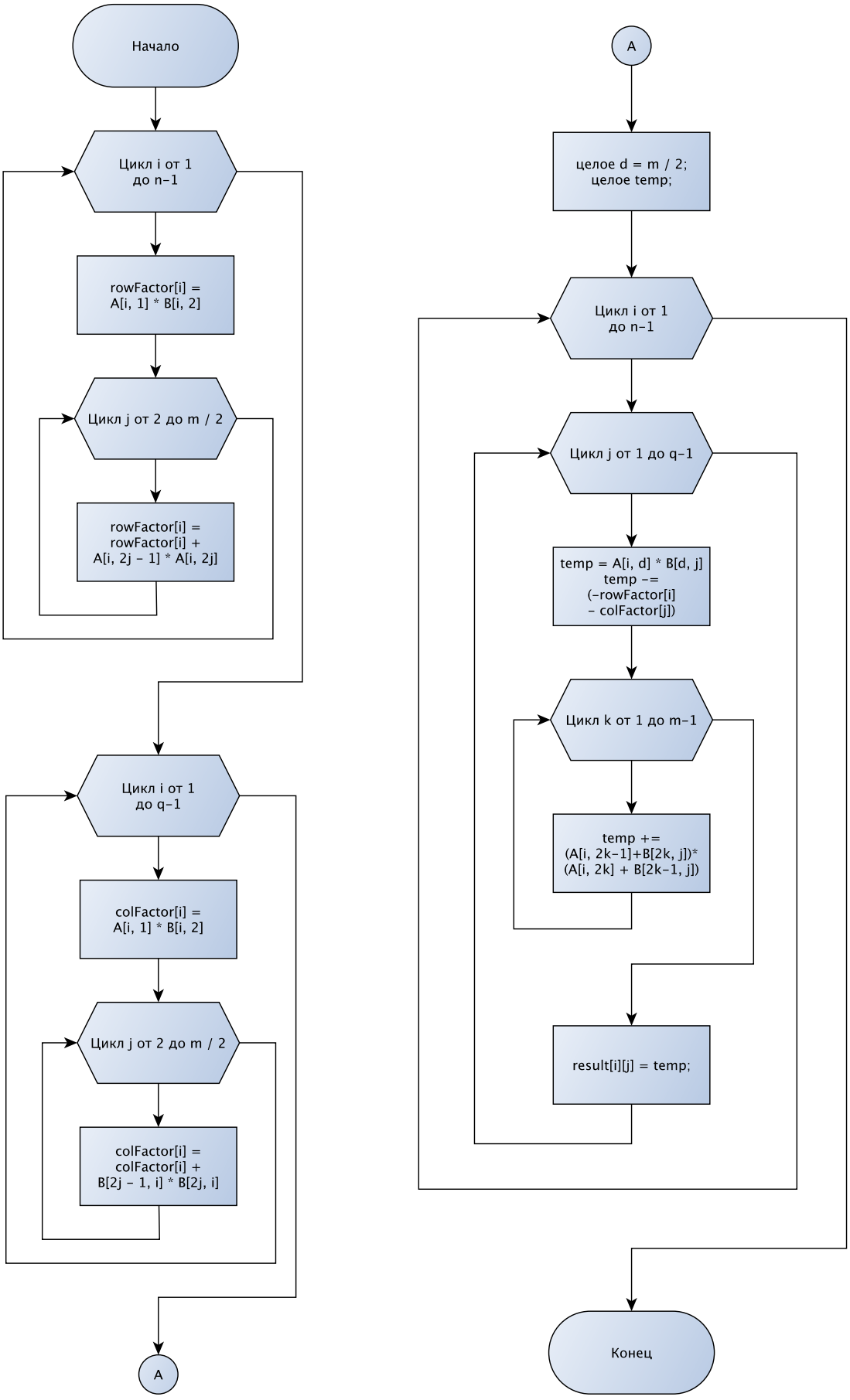


Рис. 1.3 Схема улучшенного алгоритма Винограда

2.1.1 Разработка базового алгоритма умножения матриц

Базовый алгоритм легко реализуется простым тройным циклом. В представленном ниже фрагменте кода матрица *result* была предварительно заполнена нулями.

Листинг 1. - Базовый алгоритм умножения матриц, псевдокод

```
Умнож(матр. A, целое n, целое m, матр. B, целое q, матр. result)
{
    Цикл i от 0 до n-1
    Цикл j от 0 до q-1
    Цикл k от 0 до m-1
    result[i][j] = result[i][j] + A[i][k] * B[k][j];
}
```

2.1.2 Разработка алгоритма Винограда

При работе с матрицами нечётной общей размерности требуется дополнительный двойной цикл после вычисления матрицы *result*. Ожидается, что этот цикл будет ощутимо замедлять работу алгоритма

Псевдокод алгоритма представлен ниже

Листинг 2. - Алгоритм Винограда, псеводкод

```
Виноград(матр. A, целое n, целое m, матр. B, целое q, матр. result)
{
    массив row_factor[n];

    Цикл i от 1 до n-1
    rowFactor[i] = A[i, 1] * A[i, 2]
    Цикл j от 2 до m / 2
    rowFactor[i] = rowFactor[i] + A[i, 2j - 1] * A[i, 2j]

    Цикл i от 1 до q-1
    colFactor[i] = B[1, i] * B[2, i]
    Цикл j от 2 до  m / 2
    colFactor[i] = colFactor[i] + B[2j - 1, i] * B[2j, i]


    Цикл i от 1 до n-1
    Цикл j от 1 до q-1
    result[i, j] = -rowFactor[i] - colFactor[j]
    Цикл k от 1 до m/2
    result[i, j]=result[i, j]+(A[i, 2k-1]+B[2k, j])*(A[i, 2k] +
B[2k-1, j])


    Если m % 2 != 0
    Цикл i от 1 до n-1
    Цикл j от 0 до q-1
    result[i, j] = result[i, j] + A[i, b] * B[b, j]
}
```

2.1.3 Разработка улучшенного алгоритма Винограда

Алгоритм винограда был улучшен следующими приёмами, перечисленными ниже.

1. Замена всех арифметических операций (+, - , и т.д.) после оператора присваивания на +=, -= и т.д. где это возможно
2. Занесение в циклах вычисления множителей вычисления первых двух элементов во внутренний цикл j

3. Введение переменной temp для сокращения числа обращений к матрице result через операторы индексирования
4. Замена цикла, корректирующего результат при нечётной общей размерности матриц, на аналогичные проверки в цикле умножения
5. Вычисление переменных m / 2 и чётности общей размерности заранее

Листинг 3. - Улучшенный алгоритм Винограда, псевдокод

```
Виноград(матр. A, целое n, целое m, матр. B, целое q, матр. result)
{
    массив row_factor[n];

    Цикл i от 1 до n-1
    rowFactor[i] = 0
    Цикл j от 1 до m / 2
    rowFactor[i] += A[i, 2j - 1] * A[i, 2j]

    Цикл i от 1 до q-1
    colFactor[i] = 0
    Цикл j от 1 до  m / 2
    colFactor[i] += B[2j - 1, i] * B[2j, i]

    целое d = m / 2;
    целое odd = m % 2 ? 1 : 0
    целое temp;
    Цикл i от 1 до n-1
    Цикл j от 1 до q-1
    temp = odd ? A[i][m-1] * B[m-1][j] : 0
    temp -= row_factor[i] + col_factor[j]
    Цикл k от 0 до d
    temp +=(A[i, 2k-1]+B[2k, j])*(A[i, 2k] + B[2k-1, j])
    result[i][j] = temp
}
```

2.2 Теоретическая оценка алгоритмов

Рассчёт сложности для стандартного алгоритма умножения матриц:

```
Цикл i_от 0 до n-1
    Цикл j от 0 до q-1
        Цикл k от 0 до m-1
            result[i][j] =
result[i][j] + A[i][k] * B[k][j];
```

$2 + n(2 + 2 + q(2 + 2 + m(2 + 8 + 1 + 1 + 1))) = 2 + 4n + 4nq + 13mnq$

Рассчёт сложности алгоритма Винограда:

```
Цикл i_от 1 до n
    rowFactor[i] = A[i, 1] * A[i, 2]
    Цикл j от 2 до m / 2
        rowFactor[i] = rowFactor[i] + A[i,
2j - 1] * A[i, 2j]
```

$2 + n(2 + 5 + 1 + 1 + 2 + (m/2 - 2)(2 + 6 + 1 + 3 + 1 + 1))$

```
Цикл i от 1 до q
    colFactor[i] = B[1, i] * B[2, i]
    Цикл j от 2 до m / 2
        colFactor[i] += B[2j - 1, i] *
B[2j, i]
```

$2 + n(2 + 5 + 1 + 1 + 2 + (m/2 - 2)(2 + 6 + 1 + 3 + 1 + 1))$

```
Цикл i от 1 до n-1
    Цикл j от 1 до q-1
        result[i, j] = -rowFactor[i] -
colFactor[j]
        Цикл k от 1 до m/2
            result[i, j]=result[i, j]+
(A[i, 2k-1]+B[2k, j])*(A[i, 2k] + B[2k-1,
j])
```

$2 + n(2 + 2 + q(2 + 4 + 1 + 2 + 2 + (m/2)(2 + 12 + 1 + 3 + 2 + 5))) = 2 + 4n + 11qn + 12.5mqn$

```
Если m % 2 != 0
    Цикл i от 1 до n-1
        Цикл j от 0 до q-1
            result[i, j] = result[i, j] +
A[i, b] * B[b, j]
```

$1 + 1 + 2 + n(2 + 2 + q(2 + 8 + 1 + 1 + 1)) = 3 + 4n + 13qn$

Сложность алгоритма Винограда, если общая размерность чётная:

$6 - 13n + 7mn - 17q + 7mq + 11nq + 12.5mnq$

Сложность алгоритма Винограда, если общая размерность нечётная:

$9 - 9n + 7mn - 17q + 7mq + 24nq + 12.5mnq$

<u>Расчёт сложности оптимизированного алгоритма Винограда:</u>

```
Цикл i от 1 до n
    rowFactor[i] = 0
    Цикл j от 1 до m / 2
        rowFactor[i] += A[i, 2j - 1] *
A[i, 2j]
```

$2 + n(2 + 1 + 1 + 2 + (m/2)(2 + 5 + 1 + 3 + 1)) = 2 + 6n + 6mn$

```
Цикл i от 1 до q
    colFactor[i] = 0
    Цикл j от 1 до  m / 2
        colFactor[i] += B[2j - 1, i] *
B[2j, i]
```

$2 + q(2 + 1 + 1 + 2 + (m/2)(2 + 5 + 1 + 3 + 1)) = 2 + 6q + 6mq$

```
целое d = m / 2;
    целое odd = m % 2 ? 1 : 0
    Цикл i от 1 до n
        Цикл j от 1 до q
            temp = odd ? A[i][m-1] *
B[m-1][j] : 0
            temp -= row_factor[i] +
col_factor[j]
            Цикл k от 0 до d
                temp +=(A[i, 2k-1]+B[2k,
j])*(A[i, 2k] + B[2k-1, j])
                result[i][j] = temp
```

$1 + 1 + 1 + 1 + 2 + n(2 + 2 + q(2 + 1 + 1 + 2 + 1 + 1 + 2 + (m/2)(2 + 8 + 1 + 2 + 2 + 5)2 + 1)) = 6 + 4n + 13qn + 12.5mqn$

```
целое d = m / 2;
целое odd = m % 2 ? 1 : 0
Цикл i от 1 до n
    Цикл j от 1 до q
        temp = odd ? A[i][m-1] * B[m-1]
[j] : 0
        temp -= row_factor[i] +
col_factor[j]
        Цикл k от 0 до d
            temp +=(A[i, 2k-1]+B[2k,
j])*(A[i, 2k] + B[2k-1, j])
            result[i][j] = temp
```

$1 + 1 + 1 + 1 + 2 + n(2 + 2 + q(2 + 4 + 1 + 2 + 1 + 1 + 2 + (m/2)(2 + 8 + 1 + 2 + 2 + 5)2 + 1)) = 6 + 4n + 20qn + 12.5mqn$

Сложность оптимизированного алгоритма Винограда если общая размерность чётная:

$10 + 10n + 6mn + 6q + 6mq + 13nq + 12.5mnq$

Сложность оптимизированного алгоритма Винограда если общая размерность нечётная:

$10 + 10n + 6mn + 6q + 6mq + 20nq + 12.5mnq$

Сравнение трудоёмкостей трёх алгоритмов представлено ниже.

Таблица 1. Сравнение трудоёмкостей алгоритмов умножения матриц

	Стандартный	**Виноград**	**Улучшенный Виноград**
Стандартный	одинаково	на $0.5 m n q - 7 m n - 7 m q - 7 n q + 17 n + 17 q - 4$ медленнее	на $0.5 m n q - 6 m n - 6 m q - 9 n q - 6 n - 6 q - 8$ медленнее
Виноград	на $0.5 m n q - 7 m n - 7 m q - 7 n q + 17 n + 17 q - 4$ быстрее	одинаково	на $m n + m q - 2 n q - 23 n - 23 q - 4$ медленнее
Улучшенный Виноград	на $0.5 m n q - 6 m n - 6 m q - 9 n q - 6 n - 6 q - 8$ быстрее	на $m n + m q - 2 n q - 23 n - 23 q - 4$ быстрее	одинаково

3. Технологическая часть

В технологической части описаны аппаратные характеристики машины, на которой запускался реализованный алгоритм и представлена реализация на языке программирования

3.1 Требования к программному обеспечению

ОС: mac OSX

Среда: Xcode

3.2 Средства реализации

Программа реализована на языке C стандарта C99

Язык Си был выбран за возможность ручной работы с памятью, скорости и возможности быстро и точно измерять процессорное время, затраченное программой.

3.3 Листинг кода

Листинг 4. - Код программы

```
void winograd(int** A, int n, int m, int** B, int q, int** result)
{
    int d = m / 2;

    int row_factor[n];

    for (int i = 0; i < n; ++i)
    {
        row_factor[i] = A[i][0] * A[i][1];
        for (int j = 1; j < m/2; ++j)
            row_factor[i] = row_factor[i] + A[i][2*j] * A[i][2*j+1];
    }

    int col_factor[q];

    for (int i = 0; i < q; ++i)
    {
        col_factor[i] = B[0][i] * B[1][i];
        for (int j = 1; j < m/2; ++j)
            col_factor[i] = col_factor[i] + B[2*j][i] * B[2*j+1][i];
    };


    for (int i = 0; i < n; ++i)
        for (int j = 0; j < q; ++j)
        {
            result[i][j] = -(row_factor[i] + col_factor[j]);
            for (int k = 0; k < m/2; ++k)
            {
                result[i][j] = result[i][j] + \
                (A[i][2*k] + B[2*k+1][j]) * (A[i][2*k+1] + B[2*k][j]);
            }
        }


    if (m%2 != 0)
    {
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < q; ++j)
            {
                result[i][j] += A[i][m-1] * B[m-1][j];
            }
    }
}
```

```
void winograd_plus(int** A, int n, int m, int** B, int q, int** result)
{
    int d = m / 2;
    int odd = m % 2 ? 1 : 0;

    int row_factor[n];

    for (int i = 0; i < n; ++i)
    {
        row_factor[i] = 0;
        for (int j = 0; j < d; ++j)
            row_factor[i] += A[i][2*j] * A[i][2*j+1];
    }

    int col_factor[q];

    for (int i = 0; i < q; ++i)
    {
        col_factor[i] = 0;
        for (int j = 0; j < d; ++j)
            col_factor[i] += B[2*j][i] * B[2*j+1][i];
    }

    int temp;

    for (int i = 0; i < n; ++i)
        for (int j = 0; j < q; ++j)
        {
            temp = odd ? A[i][m-1] * B[m-1][j] : 0;
            temp -= row_factor[i] + col_factor[j];

            for (int k = 0; k < d; ++k)
                temp +=  \
                (A[i][2*k] + B[2*k+1][j]) * (A[i][2*k+1] + B[2*k][j]);

            result[i][j] = temp;
        }
}
```

4. Экспериментальная часть

4.1 Постановка эксперимента

Алгоритмы будут протестированы по скорости работы и будет показана требуемая память.

Тестирование по памяти будет проводиться с помощью встроенной утилиты XCode.

Диаграмма 1. Тест времени выполнения, чётная общая размерность матриц

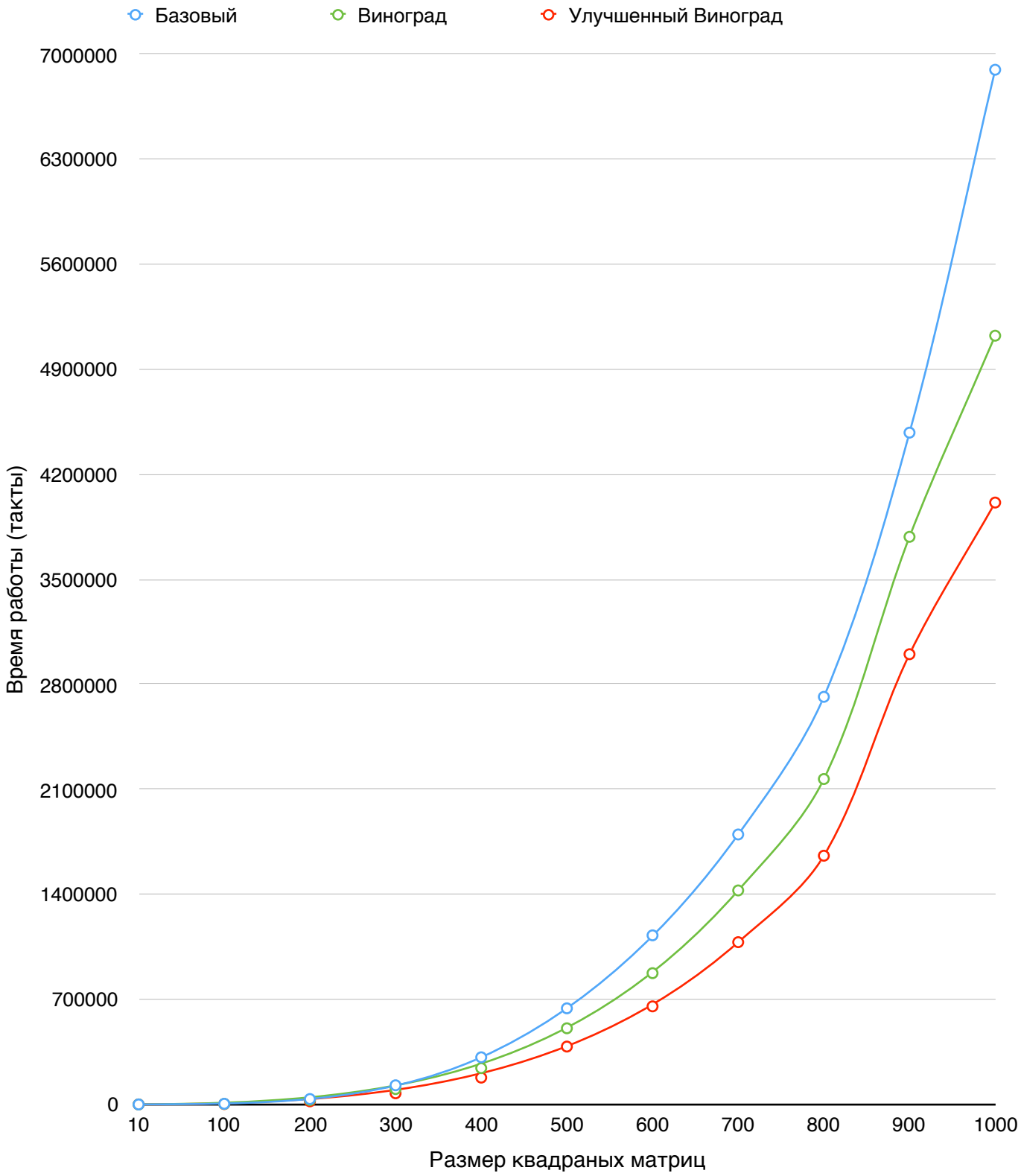


Диаграмма 2. Тест времени выполнения, нечётная общая размерность матриц

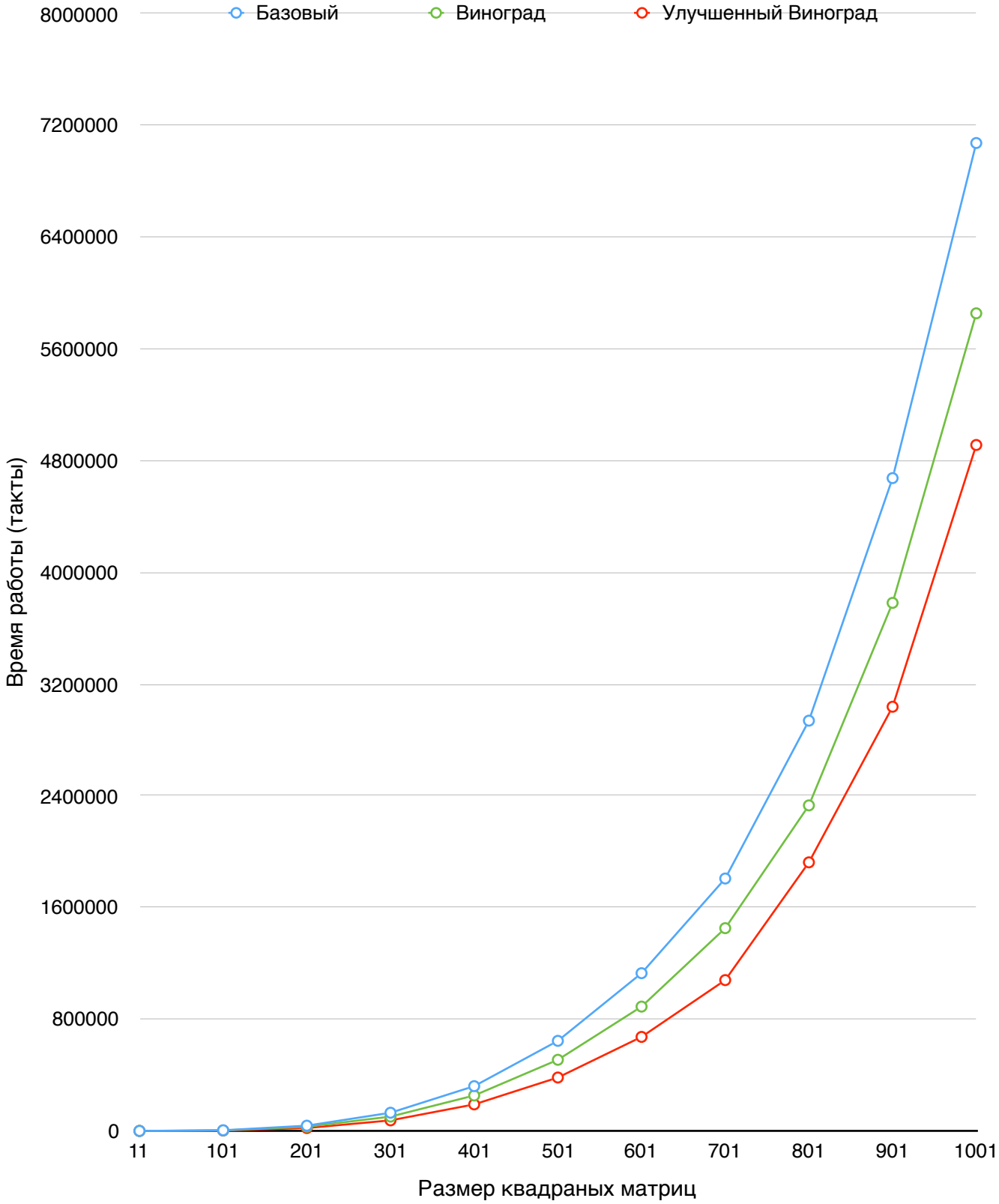
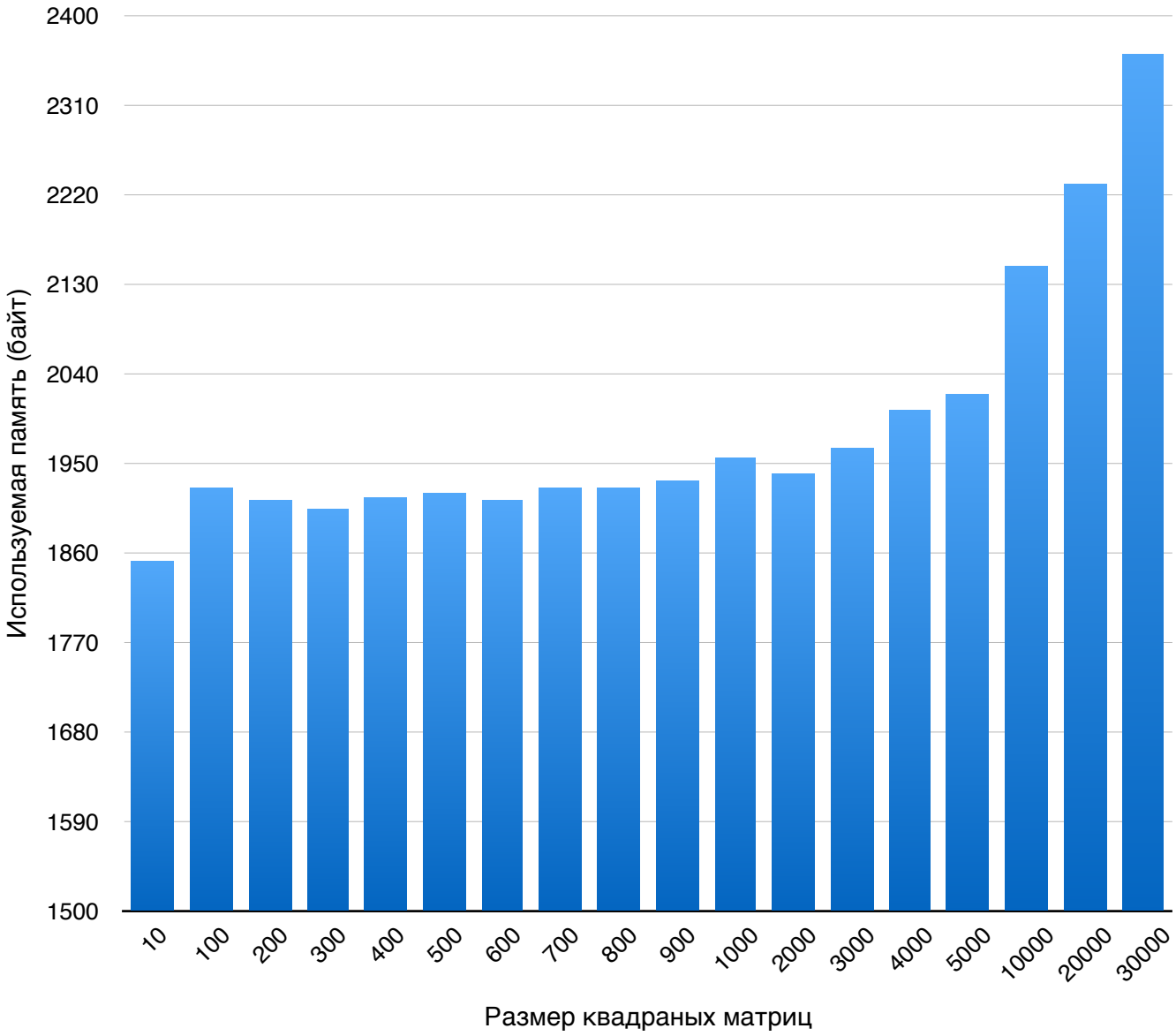


Диаграмма 3. Тест используемой памяти



4.2 Сравнительный анализ на результатах эксперимента

В эксперименте сразу прослеживается главный недостаток алгоритма Винограда - уменьшение сложности (по сравнению со стандартным алгоритмом) приводит к сильному увеличению необходимой для работы памяти.

Заключение

В данной работе были реализованы и протестированы алгоритмы умножения матриц. Также была улучшена реализация алгоритма Винограда.

Как видно из результатов экспериментов, алгоритм винограда с улучшениями предпочтительнее для матриц любой размерности, если нет ограничений по памяти.