Федеральное государственное бюджетное образовательное учреждениевысшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский институт)»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

О т ч ё т

п о л а б о р а т о р н о й р а б о т е 2

Дисциплина: Анализ Алгоритмов

Тема лабораторной работы работы: Трудеомкость алгоритмов умножения матриц

Студентки гр. ИУ7-51б _____ **Сушина А.Д.**
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Волкова Л.Л.**
(Подпись, дата) (И.О. Фамилия)

Москва, 2019г

Оглавление

Введение

Умножение матриц — это один из базовых алгоритмов, который широко применяется в различных численных методах, и в частности в алгоритмах машинного обучения.

Произведением матрицы $A m \times n$ на матрицу $B n \times k$ называется матрица $C m \times k$ такая, что элемент матрицы C, стоящий в i-ой строке и j-ом столбце, т.е. элемент $c i j$, равен сумме произведений элементов i-ой строки матрицы A на соответствующие элементы j-ого столбца матрицы B.
Алгоритм Копперсмита-Винограда— алгоритм умножения квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. Алгоритм Копперсмита—Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита—Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров.

Цель лабораторной работы изучить трудоемкости алгоритмов умножения матриц и способы оптимизации этих алгоритмов.

Задачи:

- Изучение алгоритмов стандартного умножения матриц и алгоритма Винограда.
- Получение практических навыков при реализации стандартного алгоритма умножения матриц и алгоритма Винограда.
- Оптимизация алгоритма Винограда тремя способами.
- Подсчет трудоемкости каждой из реализаций.
- экспериментальное подтверждение различий во временно́й эффективности работы оптимизированных и неоптимизированного алгоритмов Винограда
- описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1. Аналитическая часть

В этом разделе представлено описание алгоритмов.

1.1. Описание алгоритмов

Стандартный алгоритм перемножения матриц

Стандартный алгоритм умножения матриц предполагает следование определению произведения матриц.

Пусть даны две прямоугольные матрицы A и B размерности m*n и n*q соответственно:

$$A=\begin{bmatrix} a11 & a12 & ... & a1n \\ a21 & a22 & ... & a2n \\ ... & ... & ... & ... \\ am1 & am2 & ... & amn \end{bmatrix} \quad B=\begin{bmatrix} b11 & b12 & ... & b1q \\ b21 & b22 & ... & b2q \\ ... & ... & ... & ... \\ bn1 & bn2 & ... & bnq \end{bmatrix}$$

Тогда матрица C размерностью m*q:

$$C=\begin{bmatrix} c11 & c12 & ... & c1q \\ c21 & c22 & ... & c2q \\ ... & ... & ... & ... \\ cm1 & cm2 & ... & cmq \end{bmatrix}$$, в которой: $c_{ij}=\sum_{r=1}^{n} a_{ir} * b_{rj}$,

называется их *произведением*.

Алгоритм Винограда

Алгоритм Винограда умножения матриц основан на снижении доли умножений в алгоритме. Предполагается, что некоторые произведения можно вычислить заранее, а затем переиспользовать при вычислении произведений матриц.

Рассмотрим два вектора $V=(v1, v2, v3, v4)$ и $W=(w1, w2, w3, w4)$.

Их скалярное произведение равно:

$$V * W = v1\,w1 + v2\,w2 + v3\,w3 + v4\,w4$$

Это равенство можно переписать в виде:

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

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем первое: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого

столбца второй, что позволяет выполнять для каждого элемента лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Алгоритм Винограда состоит из следующих шагов:

1. Вычисление горизонтальных произведений MulH

$$MulH_i = \sum_{k=0}^{N/2} A_{i,2k} * A_{i,2k+1}$$

2. Вычисление вертикальных произведений MulV

$$MulM_j = \sum_{k=0}^{N/2} B_{2k,j} * B_{2k+1,j}$$

3. Вычисление матрицы результата

$$c_{ij} = -MulH_i - MulV_j + \sum_{k=0}^{N/2} \left(A_{i,2k} + B_{2k+1,j}\right) * \left(A_{i,2k+1} + B_{2k,j}\right)$$

4. Корректирование матрицы в случае нечетного N

$$C_{i,j} = C_{i,j} + A_{i,N-1} * B_{N-1,j}$$

2. Конструкторская часть

2.1. Разработка алгоритмов

В этом разделе представлены блок схемы алгоритмов.

Рис 1. Стандартный алгоритм умножения матриц

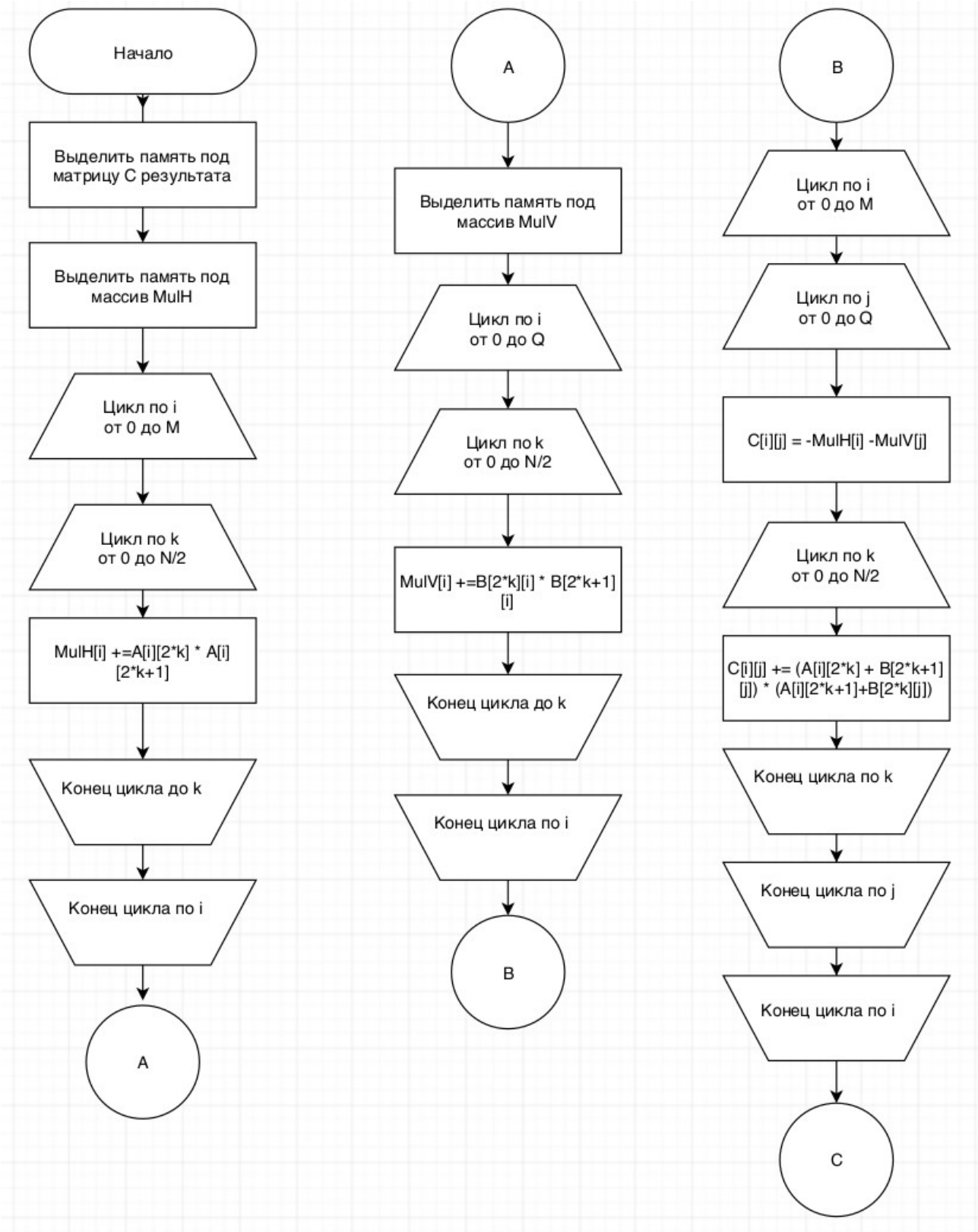
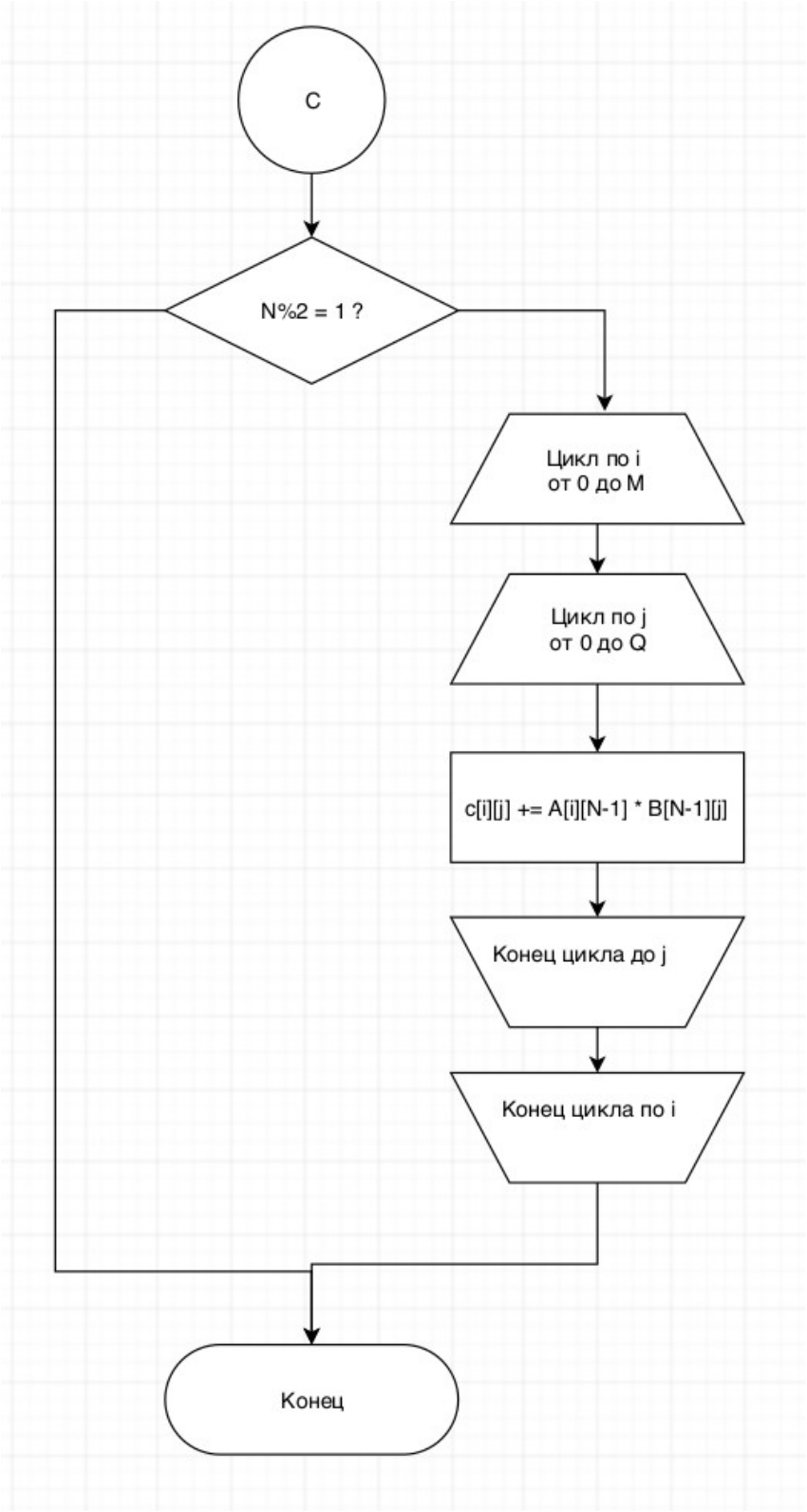


рис 2. Алгоритм Винограда(часть 1)

рис 3. Алгоритм Винограда(часть 2)



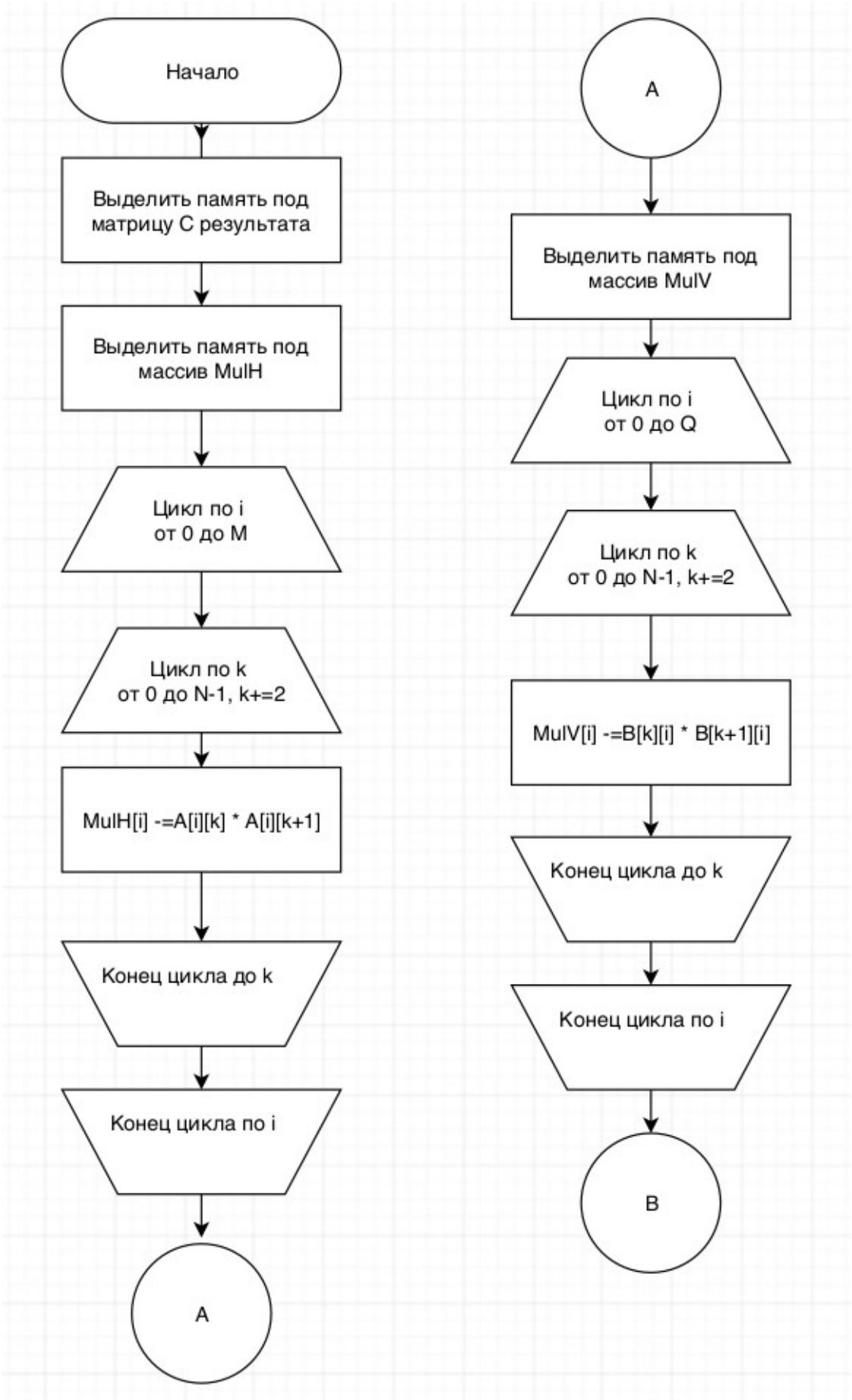


рис 4. Алгоритм Винограда(оптимизированный) часть 1

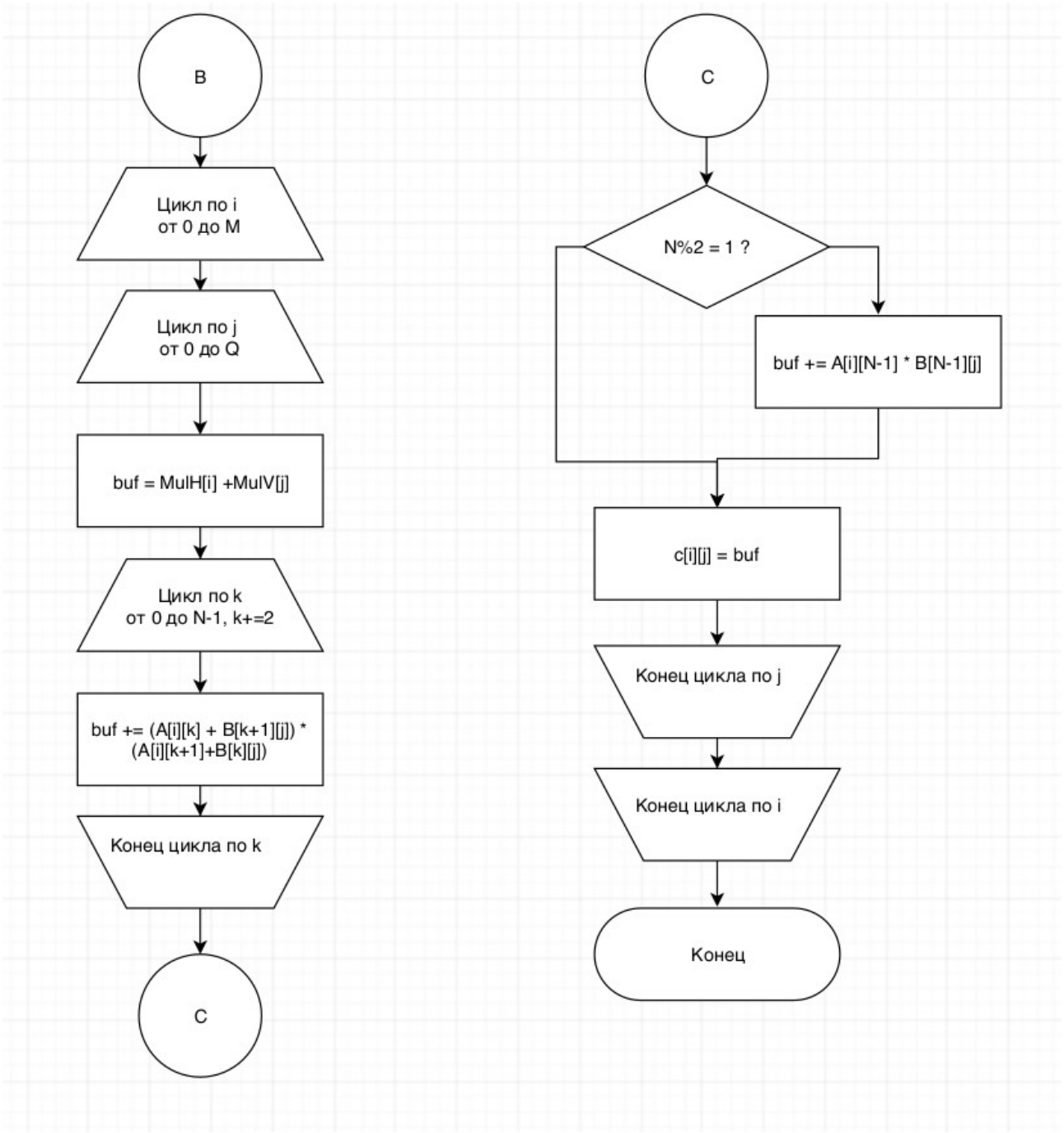


рис 5. Алгоритм Винограда (оптимизированный) часть 2

2.2. Подсчет трудоемкости алгоритмов

Модель вычислений:

1. Трудоемкость базовых операций.

Следующие операции стоят +1: +, -, *, %, =, <, >, <=, >=, ==, !=, [], +=, -=

2. Трудоемкость условного перехода.

Стоимость условного перехода 0. При этом сам расчет условия оцениваем.

$$f_{if} = f_{усл} + \begin{bmatrix} f_{б1}, \text{если условие выполнилось} \\ f_{б2}, \text{если условие не выполнилось} \end{bmatrix}$$

3. Трудоемкость цикла for.

$$f_{for} = f_{иниц} + f_{сравн} + N * (f_{тела} + f_{инкр} + f_{сравн})$$

Стандартный алгоритм

Вычисление матрицы:

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

Оптимизированный вариант:

Заменим c[i][j] = c[i][j] + на c[i][j] +=

Вместо 6 операций получаем 3.

Тогда трудоемкость: $f_{стд} = 10MNQ + 4MQ + 4M + 2 \approx 10MNQ$

Алгоритм Винограда

Вычисление MulH: $f_I = 2 + M * (2 + 3 + \frac{N}{2}(3 + 1 + 6 + 2 + 3)) = \frac{15}{2}MN + 5M + 2$

Вычисление MulV: $f_{II} = \frac{15}{2}QN + 5Q + 2$

Вычисление матрицы:

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

Для нечетного N

$$f_{IV} = 2 + \begin{bmatrix} 0, N \text{ четное} \\ 12MQ + 4M + 2, N \text{ нечетное} \end{bmatrix}$$

Итого:

Лучший случай:

$$f_{Вин\ лучшее} = \frac{15}{2}MN + 5M + 2 + \frac{15}{2}QN + 5Q + 2 + 13MNQ + 12MQ + 4M + 2 + 2 + \begin{bmatrix} 0, N \text{ четное} \\ 12MNQ + 4M + 2, N \text{ нечетное} \end{bmatrix}$$

$$f = 13MNQ + 12MQ + \frac{15}{2}MN + \frac{15}{2}QN + 9M + 5Q + 8$$

Худший случай:

$$f = 13MNQ + 24MQ + \frac{15}{2}MN + \frac{15}{2}QN + 13M + 5Q + 10$$

Алгоритм Винограда (оптимизированный)

Вычисление MulH: $f_I = 2 + M * (2 + 3 + \frac{N}{2}(2 + 1 + 5 + 1 + 1)) = \frac{10}{2} MN + 5M + 2$

Вычисление MulV: $f_{II} = \frac{10}{2} QN + 5Q + 2$

Вычисление матрицы: $f_{III} = 2 + M * (2 + 2 + Q(2 + 4 + 3 + \frac{N}{2} * (6 + 8) + 2 + \begin{bmatrix} 0, N\ четное \\ 4 + 4, N\ нечетное \end{bmatrix} + 3))$

$$f = 7MNQ + 14MQ + MQ \begin{bmatrix} 0, N\ четное \\ 4 + 4, N\ нечетное \end{bmatrix} + 4M + 2$$

Итого:

Лучший случай: $f_{Вин\ лучшее} = \frac{10}{2} MN + 5M + 2 + \frac{10}{2} QN + 5Q + 2 + 7MNQ + 14MQ + 4M + 2$

$$f = 7MNQ + 14MQ + \frac{10}{2} MN + \frac{10}{2} QN + 9M + 5Q + 6$$

Худший случай:

$$f = 7MNQ + 22MQ + \frac{10}{2} MN + \frac{10}{2} QN + 9M + 5Q + 6$$

3.Технологическая часть

3.1.Требования к программному обеспечению

На вход программа получает две матрицы с размерами MxN и NxQ. На выходе получается матрица размером MxQ.
Так же должна быть реализована функция для тестирования и функции замера времени.

3.2.Средства реализации

Для реализации программы был выбран язык C++ в связи с возможностью прибегать к использованию ООП, а так же с моими личным опытом работы с этим ЯП. Среда разработки — Qtcreator. Для работы с матрицами были реализован свой класс Matrix.

3.3.Листинг кода

Далее представлен листинг программы.

Стандартный алгоритм

```
Matrix standart(Matrix m1, Matrix m2) {
    int N = m1.cols();
    int M = m1.rows();
    int Q = m2.cols();
    Matrix c(M, Q);

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < Q; j++) {
            for (int k = 0; k < N; k++) {
                c[i][j] = c[i][j]+m1[i][k]*m2[k][j];
            }
        }
    }
    return c;
}
```

Стандартный алгоритм(оптимизация)

```
Matrix standartO(Matrix m1, Matrix m2) {
    int N = m1.cols();
    int M = m1.rows();
    int Q = m2.cols();
    Matrix c(M, Q);

    for (int i = 0; i < M; i++) {
        for (int j = 0; j < Q; j++) {
            for (int k = 0; k < N; k++) {
                c[i][j] += m1[i][k]*m2[k][j];
            }
        }
    }
    return c;
}
```

Алгоритм Винограда

```
Matrix Vinograd(Matrix A, Matrix B) {
   int N = A.cols();
   int M = A.rows();
   int Q = B.cols();
   Matrix c(M, Q);

   std::vector<int> MulH(M, 0);
   for (int i = 0; i < M; i++)
   {
      for (int k = 0; k < N/2; k++)
      {
         MulH[i] = MulH[i] + A[i][2*k] * A[i][2*k+1];
      }
   }

   std::vector<int> MulV(Q, 0);
   for (int i = 0; i < Q; i++)
   {
      for (int k = 0; k < N/2; k++)
      {
         MulV[i] = MulV[i] + B[2*k][i]*B[2*k+1][i];
      }

   }


   for (int i = 0; i < M; i++) {
      for (int j = 0; j < Q; j++) {
         c[i][j] = -MulH[i] - MulV[j];
         for (int k = 0; k < N/2; k++) {
            c[i][j] = c[i][j] + (A[i][2*k] + B[2*k+1][j])*(A[i][2*k+1] + B[2*k][j]);
         }
      }
   }
   if (N%2 == 1) {
      for (int i = 0; i < M; i++) {
         for (int j = 0; j < Q; j++) {
            c[i][j] = c[i][j] + A[i][N-1]*B[N-1][j];
         }
      }
   }

   return c;
}
```

```
Matrix Vinograd2(Matrix A, Matrix B) {
   int N = A.cols();
   int M = A.rows();
   int Q = B.cols();
   Matrix c(M, Q);

   std::vector<int> MulH(M,0);
   for (int i = 0; i < M; i++)
   {
      for (int k = 0; k < N-1; k += 2)
      {
         MulH[i] -= A[i][k] * A[i][k+1];
      }

   }

   std::vector<int> MulV(Q,0);
   for (int i = 0; i < Q; i++)
   {
      for (int k = 0; k < N-1; k += 2)
      {
         MulV[i] -= B[k][i]*B[k+1][i];
      }
   }

   for (int i = 0; i < M; i++) {
      for (int j = 0; j < Q; j++) {
         int buf = MulH[i] + MulV[j];
         for (int k = 0; k < N-1; k += 2) {
            buf += (A[i][k] + B[k+1][j])*(A[i][k+1] + B[k][j]);
         }
         if (N%2 == 1) {
             buf += A[i][N-1]*B[N-1][j];
         }
         c[i][j] = buf;
      }
   }
   return c;
}
```

Алгоритм Винограда(оптимизированный)

3.4.Описание тестирования

Тестирование будет реализовано в виде отдельной функции в программе, которую можно запустить по желанию пользователя. Для каждого алгоритма реализована своя функция с тестами, однако тесты для всех алгоритмов одинаковые.

Тестирование по следующим данным:

1. Поверка работы с единичными матрицами: [3] и [4]

2. Проверка работы с матрицами из 2 элементов: $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$ и $\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$

3. Проверка работы с матрицами с нечетным количеством элементов.

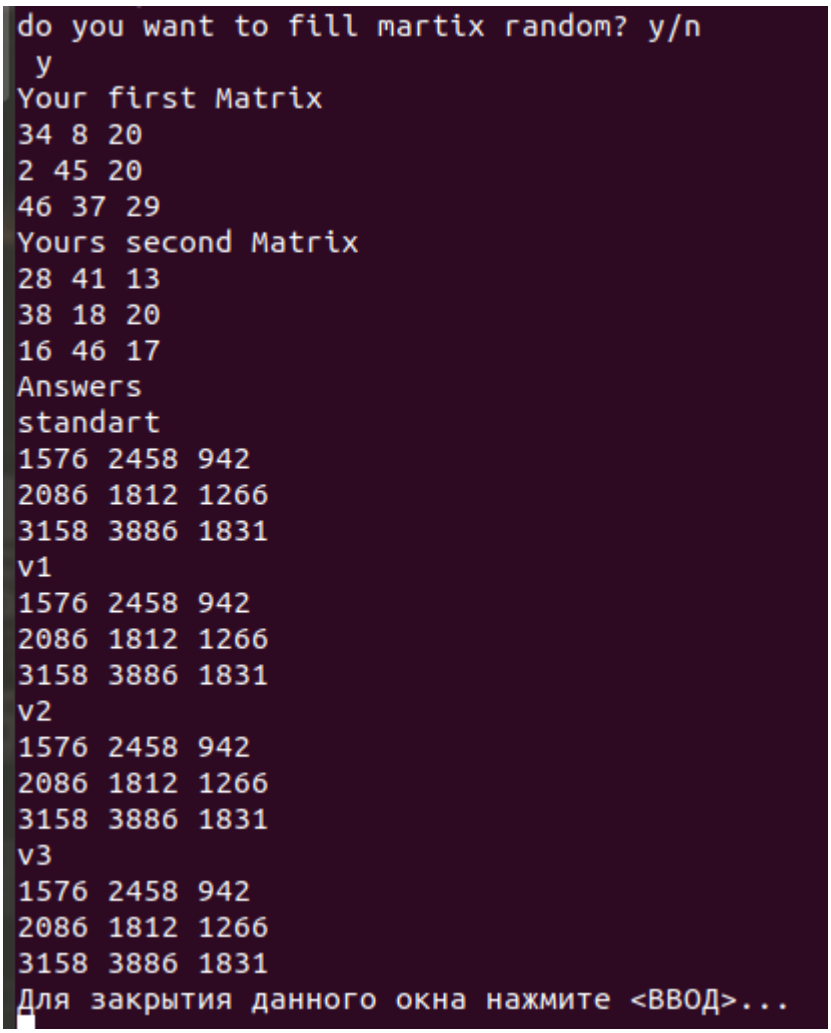
4. Проверка работы с матрицами с четным количеством элементов.

4.Экспериментальная часть

4.1.Примеры работы

Пример работы с четным N

```
Input M: 4
Input N: 4
Input Q: 4
do you want to fill martix random? y/n
 y
Your first Matrix
11 10 27 3
45 39 21 20
42 39 14 27
34 21 48 13
Yours second Matrix
47 7 15 26
29 12 17 23
19 7 35 13
8 29 31 19
Answers
standart
1344 473 1373 924
3805 1510 2693 2720
3587 1643 2620 2684
3223 1203 2950 2238
v1
1344 473 1373 924
3805 1510 2693 2720
3587 1643 2620 2684
3223 1203 2950 2238
v2
1344 473 1373 924
3805 1510 2693 2720
3587 1643 2620 2684
3223 1203 2950 2238
v3
1344 473 1373 924
3805 1510 2693 2720
3587 1643 2620 2684
3223 1203 2950 2238
Для закрытия данного окна нажмите <ВВОД>...
```



Пример работы с нечетным N

Пример работы с разными M и Q.

```
Input M: 4
Input N: 2
Input Q: 5
do you want to fill martix random? y/n
 y
Your first Matrix
23 11
42 1
17 21
2 9
Yours second Matrix
49 34 21 31 49
44 30 5 37 8
Answers
standart
1611 1112 538 1120 1215
2102 1458 887 1339 2066
1757 1208 462 1304 1001
494 338 87 395 170
v1
1611 1112 538 1120 1215
2102 1458 887 1339 2066
1757 1208 462 1304 1001
494 338 87 395 170
v2
1611 1112 538 1120 1215
2102 1458 887 1339 2066
1757 1208 462 1304 1001
494 338 87 395 170
v3
1611 1112 538 1120 1215
2102 1458 887 1339 2066
1757 1208 462 1304 1001
494 338 87 395 170
Для закрытия данного окна нажмите <ВВОД>...
```

4.2.Результаты тестирования

Все тесты прошли успешно.

В таблице представлены тесты и ответы. На рисунке представлени результат тестирования.

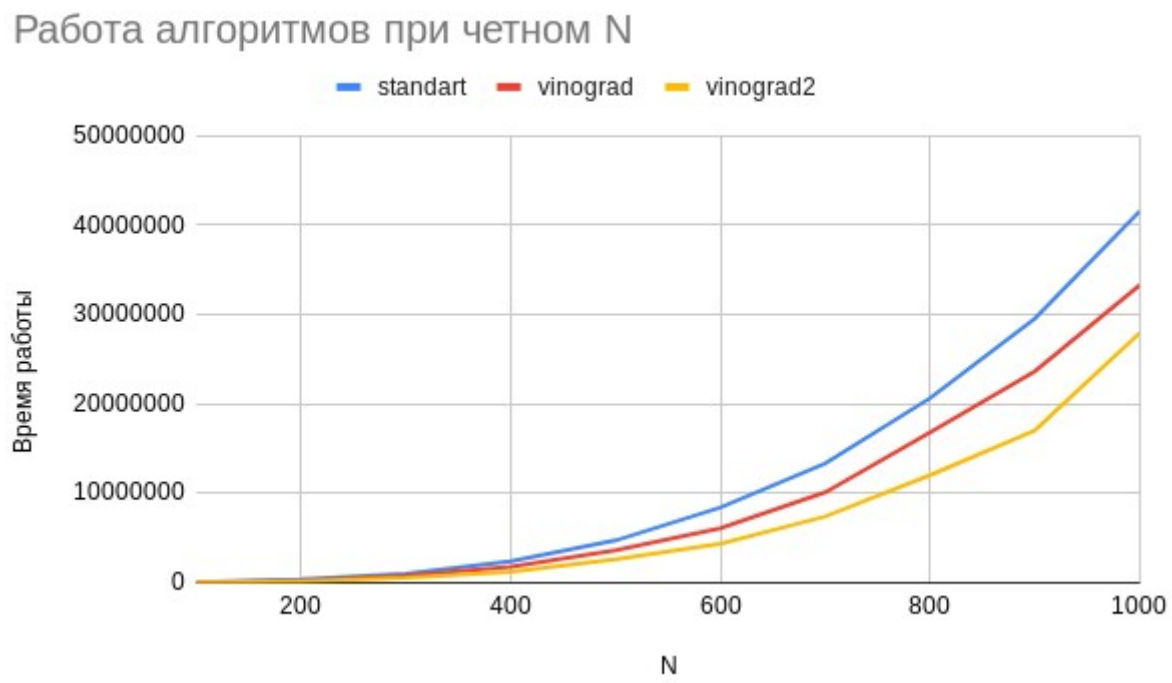
№ теста	Матрица 1	Матрица 2	Ответ
1	[3]	[4]	[12]
2	$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$	$\begin{bmatrix} 3 & 6 \\ 3 & 6 \end{bmatrix}$

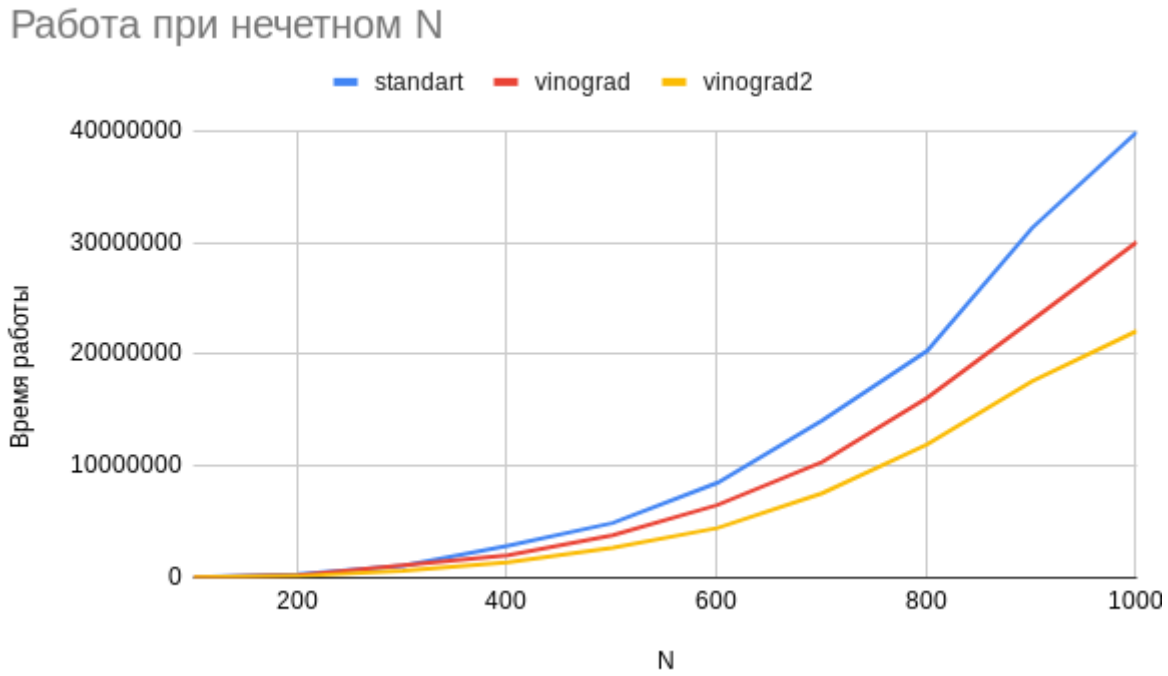
3	$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$	$\begin{bmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{bmatrix}$
4	$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$	$\begin{bmatrix} 3 & 6 & 9 \\ 6 & 12 & 18 \end{bmatrix}$
5	$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 10 & 20 & 30 & 40 \\ 10 & 20 & 30 & 40 \\ 10 & 20 & 30 & 40 \\ 10 & 20 & 30 & 40 \end{bmatrix}$

```
TEST VINOGRAD
test 1: ok
test 2: ok
test 3: ok
test 4: ok
test 5: ok
TEST VINOGRAD OPTIMIZE
test 1: ok
test 2: ok
test 3: ok
test 4: ok
test 5: ok
TEST standart
test 1: ok
test 2: ok
test 3: ok
test 4: ok
test 5: ok
```

4.3.Постановка эксперимента по замеру времени

Были проведены временные эксперименты для матриц от 100x100 до 1000x1000 с шагом 100 и для матриц от 101x101 до 1001x1001 с шагом 100. Для каждого замера взят средний результат из 50 замеров.





По полученным графикам видно, что алгоритм Винограда работает лучше стандартного на больших числах. Даже в худшем случае, при нечетном N он значительно выигрывает по скорости.

Заключение

В ходе лабораторной работы были изучены алгоритмы умножения матриц: стандартный, Винограда и оптимизированный Винограда. Был оптимизирован алгоритм Винограда и подсчитана трудоемкость для каждого из алгоритмов. Все алгоритмы были реализованы на языке C++. Были проведены эксперименты по замеру времени