Министерство науки и высшего образования 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)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчёт

по лабораторной работе № 2

Название: Алгоритмы умножения матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б (Группа) ______ (Подпись, дата) Д.В. Сусликов (И.О. Фамилия)

Преподователь ______ (Подпись, дата) Л.Л. Волкова (И.О. Фамилия)

Москва, 2020 г.

Оглавление

Введение

Цель данной лабораторной работы - изучение алгоритмов умножения матрицы, получение навыков улучшения алгоритмов и подсчёта их трудоёмкости. В ходе работы будут рассмотрены 3 алгоритма:

1) стандартный алгоритм умножения матриц;
2) алгоритм Винограда;
3) улучшенный алгоритм Винограда.

В лабораторной работе требуется:

1) изучить алгоритмы умножения матриц;
2) оптимизировать алгоритм Винограда;
3) дать теоритическую оценку стандартного алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда;
4) реализовать три алгоритма умножения матриц;
5) сравнить алгоритмы умножения матриц.

1 | Аналитическая часть

В данном разделе представлены математические описания алгоритмов умножения матриц.

1.1 Стандартный алгоритм умножения матриц

Матрица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

Пусть даны две прямоугольные матрицы A и B размеров $[m * n]$ и $[n * k]$ соответственно. В результате произведение матриц A и B получим матрицу C размера $[m * k]$.

$$c_{i,j} = \sum_{r=1}^{m} a_{ir} b_{rj} \qquad (i = 1, 2, ...l; j = 1, 2, ...n) \tag{1.1}$$

Операция умножения двух матриц выполнима только в том случае, если число столбцов в первой матрице совпадает с числом строк во второй, то эти две матрицы можно перемножить.

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. [1]

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:

$$V * W = v_1 w_1 + v_2 w_2 + v_3 w_3 + v_4 w_4 \tag{1.2}$$

Это равенство можно переписать в виде:

$$V * W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4 \tag{1.3}$$

Менее очевидно, что 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. На практике это означает, что над предварительно обработанными элементами придется выполнять лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения.

Вывод

Таким образом, получилось узнать, что из себя представляет умножение матриц. Были описаны стандартный алгоритм умножения и Винограда, объяснено преимущество второго над первым.

2 | Конструкторская часть

В данном разделе представлены схемы алгоритмов и дана оценка их трудоемкости.

2.1 Схемы алгоритмов

Ниже на Рисунке 1 представлена схема стандартного алгоритма умножения матриц.

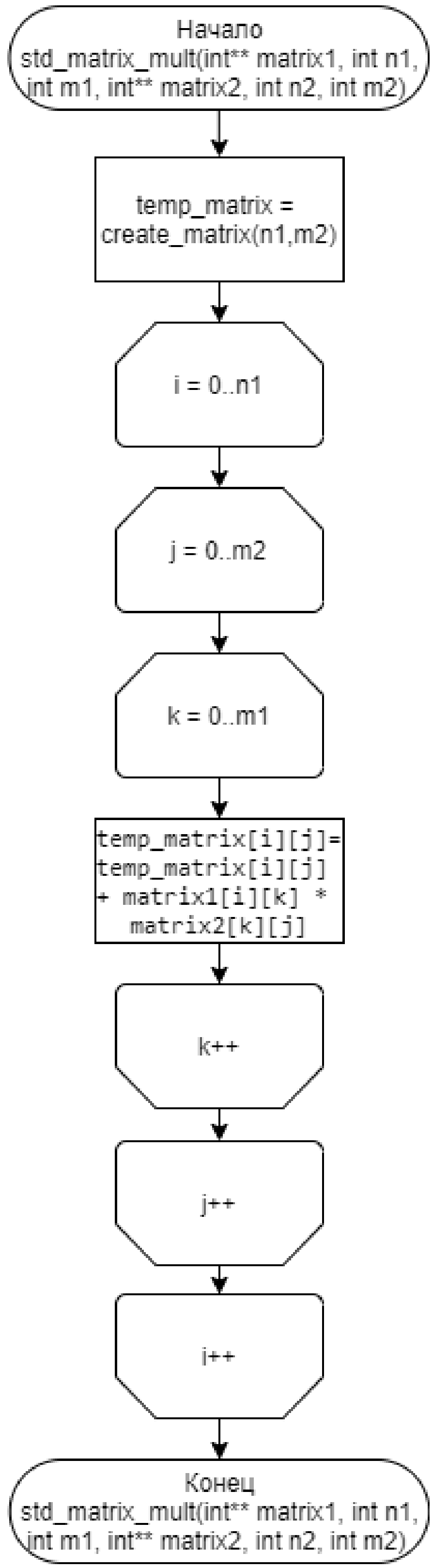

Рисунок 1 - Схема стандартного алгоритма умножения матриц

Далее на Рисунке 2 можно увидеть схему алгоритма Винограда.

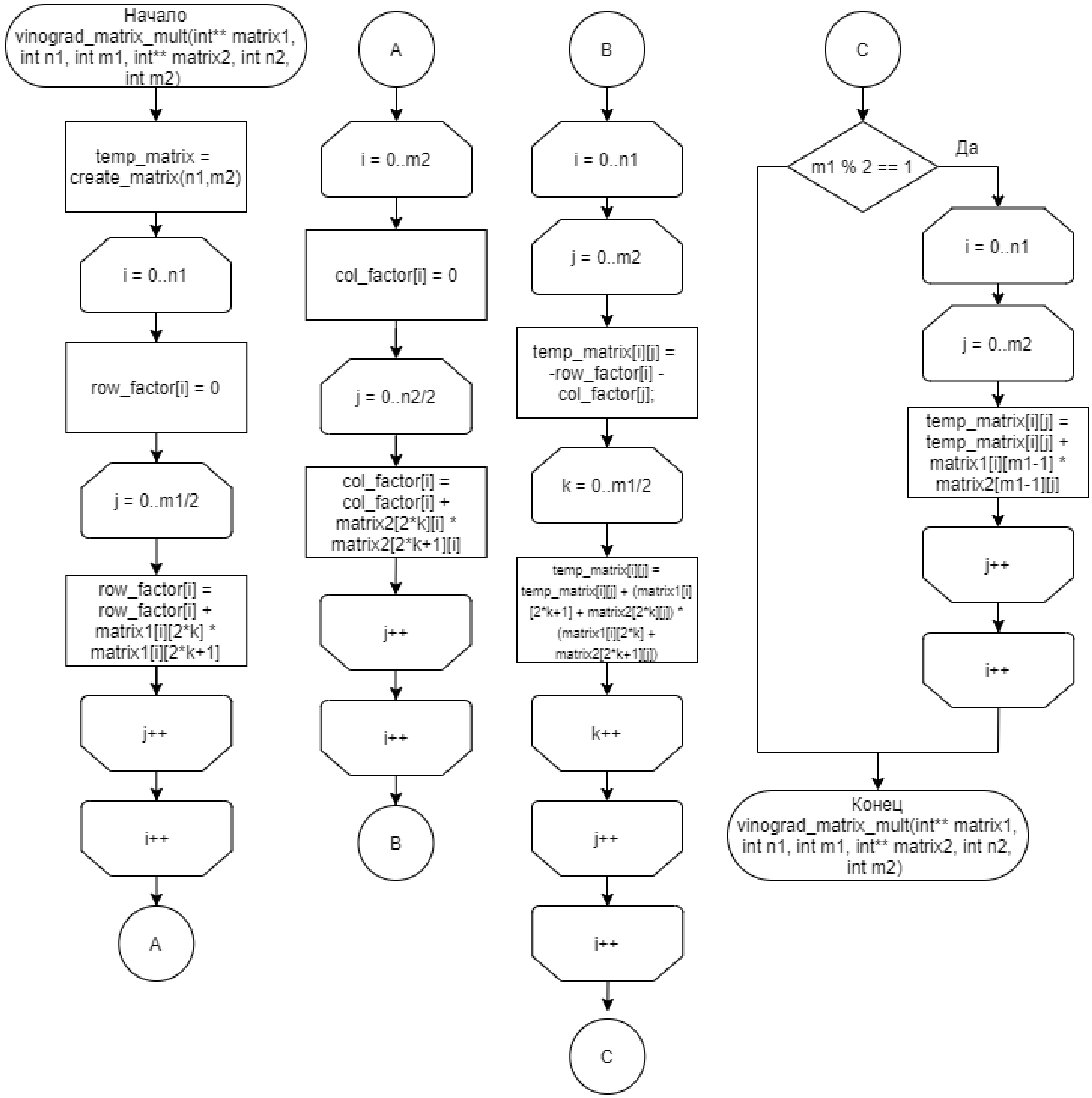

Рисунок 2 - Схема алгоритма Винограда

Ниже на Рисунке 3 показана схема модифицированного алгоритма Винограда.

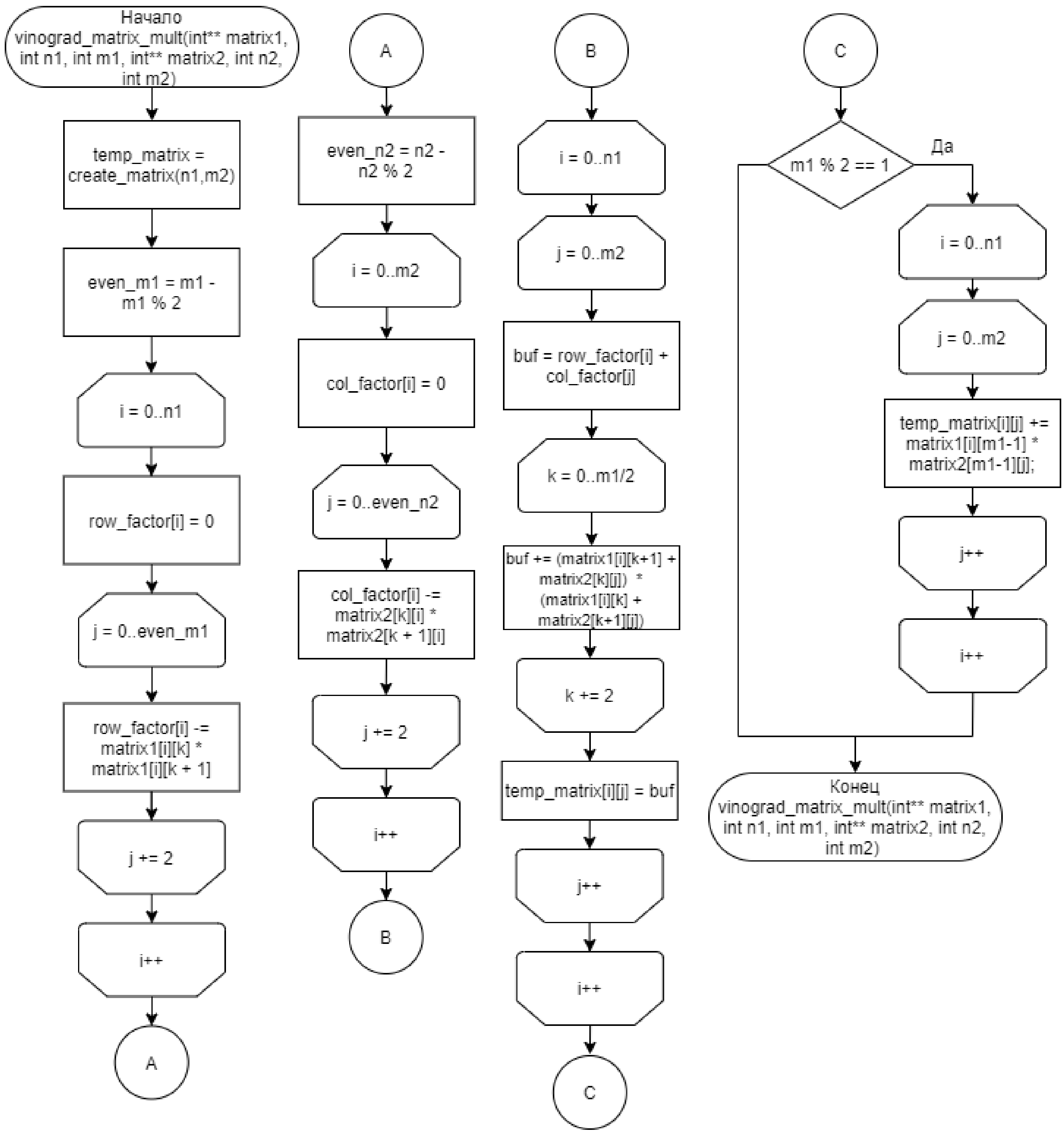

Рисунок 3 - Схема модифицированного алгоритма Винограда

2.2 Отличие модифицированного алгоритма Винограда от обычного

Улучшения:

1) нет нужды внутри цикла каждый раз пересчитывать $m1/2$ и $n2/2$, так как заранее посчитано $even\_m1 = m1 - m\%2$ и $even\_n2 = n2 - n2\%2$;

2) так как есть $even\_m1$ и $even\_n2$, то индекс будет изменятся на 2 - $j+ = 2$, следовательно нет нужды умножать на 2, как в обычном алгоритме, где индекс матрицы считался $matrix1[i][2 * j]$;

3) в обычном алгоритме значение $-row\_factor[i] - col\_factor[j]$ присваивается элементу матрицы умножения, и в следующем цикле каждый раз происходит обращение к нему. В модифицированном алгоритме Винограда это значение присваивается буферу, и лишь потом после выполнения цикла записывается в матрицу-результат;

4) подставлены $+=$ и $-=$, где возможно.

2.3 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1) Операции, чья стоимость 1: $=, +, *, \simeq, <, >, \geq, \leq, ==, != , [], +=, -=, *=, /=, ++, --$;

2) стоимость цикла:

   $f_{for} = f_{init} + f_{comp} + M(f_{body} + f_{increment} + f_{comp})$

   Пример: $for(i = 0, i < M; i++)/ * body * /$

   Результат: $2 + M(2 + f_{body})$;

3) стоимость условного оператора

   Пусть goto (переход к одной из ветвей) стоит 0, тогда

$$f_f = \begin{cases} min(f_A, f_B), & \text{лучший случай} \\ max(f_A, f_B), & \text{худший случай} \end{cases}$$

Оценим трудоемкость алгоритмов.

2.3.1 Трудоемкость общей первичной проверки

Трудоемкость проверки $if((m1! = n2)||n1 == 0||n2 == 0)return$; - 5.

2.3.2 Трудоемкость стандартного алгоритма умножения

Расчёт:

$2 + n_1(2 + 2 + m_2(2 + 3 + 2 + m_1(2 + 11))) = 13n_1m_1m_2 + 7n_1m_2 + 4n_1 + 2$

2.3.3 Трудоемкость алгоритма Винограда

Расчёт:

Первый цикл: $2+n_1(2+2+3+\frac{m_1}{2}(3+12)) = 2+n_1(7+\frac{15m_1}{2}) = \frac{15}{2}n_1m_1+7n_1+2$

Второй цикл: аналогично, $\frac{15}{2}m_2n_2 + 7m_2 + 2$

Третий цикл: $2+n_1(2+2+m_2(2+7+3+\frac{m_1}{2}(3+23))) = 13n_1m_1m_2+12n_1m_2+4n_1 + 2$

Условие: $\begin{bmatrix} 2 & \text{, невыполнение} \\ 15n_1m_2 + 4n_1 + 4 & \text{, выполнение} \end{bmatrix}$

Результат: $13n_1m_1m_2 + \frac{15}{2}n_1m_1 + \frac{15}{2}m_2n_2 + 12n_1m_2 + 7n_1 + 7m_2 + 4n_1 + 6 + \begin{bmatrix} 2 & \text{, невыполнение} \\ 15n_1m_2 + 4n_1 + 4 & \text{, выполнение} \end{bmatrix}$

2.3.4 Трудоемкость модифицированного алгоритма Винограда

Расчёт:

Первый цикл: $3 + 2 + n_1(2 + 2 + 2 + \frac{m_1}{2}(2 + 8)) = 5n_1m_1 + 6n_1 + 5$

(3 - на переменную вместо $m_1/2$)

Второй цикл: аналогично, $5m_2n_2 + 6m_2 + 5$

Третий цикл: $2 + n_1(2 + 2 + m_2(2 + 4 + 2 + \frac{m_1}{2}(2 + 14) + 3)) = 8n_1m_1m_2 + 11n_1m_2 + 4n_1 + 2$

Условие: $\begin{bmatrix} 2 & \text{, невыполнение} \\ 12n_1m_2 + 4n_1 + 4 & \text{, выполнение} \end{bmatrix}$

Результат: $8n_1m_1m_2 + 5n_1m_1 + 5m_2n_2 + 11n_1m_2 + 6n_1 + 6m_2 + 12 + \begin{bmatrix} 2 & \text{, невыполнение} \\ 12n_1m_2 + 4n_1 + 4 & \text{, выполнение} \end{bmatrix}$

Вывод

В итоге, код алгоритма умножения матриц Винограда больше, чем у стандартного, но трудоемкость меньше. Так же есть воможность модифицировать алгоритм Винограда, чтоб трудоемкость была еще меньше.

3 | Технологическая часть

В данном разделе даны общие требования к программе, средства реализации и реализация алгоритмов.

3.1 Общие требования

Требования к вводу:

1) вводятся размеры матриц;

2) вводятся (или автоматически генерируются) матрицы.

Требования к программе:

1) при вводе неправильных размеров матриц программа не должна завершаться аварийно;

2) должно выполняться корректное умножение матриц.

3.2 Средства реализации

В лабораторной работе был использован язык C++[1], так как он известен, и на нём было написано множество предыдущих работ.

Среда разработки - Qt[2].

Для замеров процессорного времени была использована функция $clock()$[3].

3.3 Реализация алгоритмов

Листинг 1 - Алгоритм стандартного умножения матриц

```
void std_matrix_mult(int** matrix1, int n1, int m1,\
int** matrix2, int n2, int m2)
{
  if ((m1 != n2) || n1 == 0 || n2 == 0)
  {
    std::cout << "Incorrect matrixes" << std::endl;
    return;
  }

  int** temp_matrix = create_matrix(n1, m2);

  for (int i = 0; i < n1; i++)
  {
    for (int j = 0; j < m2; j++)
    {
      temp_matrix[i][j] = 0;
      for (int k = 0; k < m1; k++)
      temp_matrix[i][j] = temp_matrix[i][j] + matrix1[i][k] *
         matrix2[k][j];
    }
  }

  print_matrix(temp_matrix, n1, m2);
  delete_matrix(temp_matrix, m1);
}
```

Листинг 2 - Алгоритм Винограда

```
void vinograd_matrix_mult(int** matrix1, int n1, int m1,\
int** matrix2, int n2, int m2)
{
  if ((m1 != n2) || n1 == 0 || n2 == 0)
  {
    std::cout << "Incorrect matrixes" << std::endl;
    return;
  }

  int** temp_matrix = create_matrix(n1, m2);

  int row_factor[n1];
  for (int i = 0; i < n1; i++)
  {
    row_factor[i] = 0;
    for (int k = 0; k < m1 / 2; k++)
    row_factor[i] = row_factor[i] + matrix1[i][2 * k] * matrix1[i
      ][2 * k + 1];
  }

  int col_factor[m2];
  for (int i = 0; i < m2; i++)
  {
    col_factor[i] = 0;
    for (int k = 0; k < n2 / 2; k++)
    col_factor[i] = col_factor[i] + matrix2[2 * k][i] * matrix2[2 *
      k + 1][i];
  }

  for (int i = 0; i < n1; i++)
  {
    for (int j = 0; j < m2; j++)
    {
      temp_matrix[i][j] = -row_factor[i] - col_factor[j];
      for (int k = 0; k < m1 / 2; k++)
      temp_matrix[i][j] = temp_matrix[i][j] + (matrix1[i][2 * k +
        1] + matrix2[2 * k][j])
```

```
            * (matrix1[i][2 * k] + matrix2[2 * k + 1][j]);
        }
    }

    if (m1 % 2 == 1)
    {
        for (int i = 0; i < n1; i++)
        {
            for (int j = 0; j < m2; j++)
            temp_matrix[i][j] = temp_matrix[i][j] + matrix1[i][m1 - 1] *
                matrix2[m1 - 1][j];
        }
    }

    print_matrix(temp_matrix, n1, m2);
    delete_matrix(temp_matrix, m1);
}
```

Листинг 3 - Модифицированный алгоритм Винограда

```
void vinograd_modified_matrix_mult(int** matrix1, int n1, int m1,\
int** matrix2, int n2, int m2)
{
    if ((m1 != n2) || n1 == 0 || n2 == 0)
    {
        std::cout << "Incorrect matrixes" << std::endl;
        return;
    }

    int** temp_matrix = create_matrix(n1, m2);

    int row_factor[n1];
    int even_m1 = m1 - m1 % 2;
    for (int i = 0; i < n1; i++)
    {
        row_factor[i] = 0;
        for (int k = 0; k < even_m1; k += 2)
        row_factor[i] -= matrix1[i][k] * matrix1[i][k + 1];
    }

```

```
    int col_factor[m2];
    int even_n2 = n2 - n2 % 2;
    for (int i = 0; i < m2; i++)
    {
      col_factor[i] = 0;
      for (int k = 0; k < even_n2; k += 2)
      col_factor[i] -= matrix2[k][i] * matrix2[k + 1][i];
    }

    for (int i = 0; i < n1; i++)
    {
      for (int j = 0; j < m2; j++)
      {
        int buf = row_factor[i] + col_factor[j];
        for (int k = 0; k < even_m1; k += 2)
        buf += (matrix1[i][k + 1] + matrix2[k][j])
        * (matrix1[i][k] + matrix2[k + 1][j]);

        temp_matrix[i][j] = buf;
      }
    }

    if (m1 % 2 == 1)
    {
      for (int i = 0; i < n1; i++)
      {
        for (int j = 0; j < m2; j++)
        temp_matrix[i][j] += matrix1[i][m1 - 1] * matrix2[m1 - 1][j];
      }
    }

    print_matrix(temp_matrix, n1, m2);
    delete_matrix(temp_matrix, m1);
  }
```

Вывод

По итогу, написанная программа соотвествует всем описанным выше требованиям, алгоритмы были реализованы на C++, так как данный язык известен, выполнено на нём много прошлых работ.

4 | Экспериментальная часть

В данном разделе представлены результаты работы программы и приведен анализ времени работы кажого из алгоритмов.

4.1 Примеры работы программы

На Рисунке 4 представлены меню и ввод матриц.

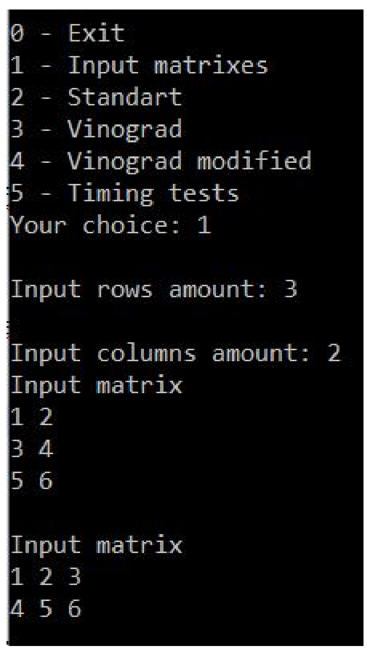

Рисунок 4 - Меню выбора и ввод матриц

На Рисунке 5 изображен результат работы стандартного алгоритма умножения матриц.

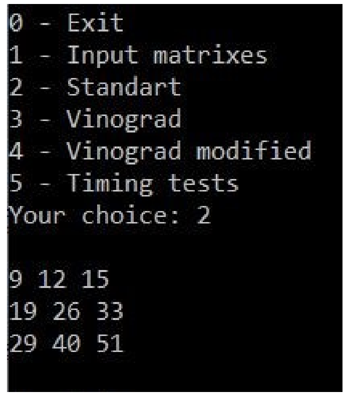

Рисунок 5 - Стандартный алгоритм умножения матриц

На Рисунке 6 показан результат работы алгоритма Винограда.

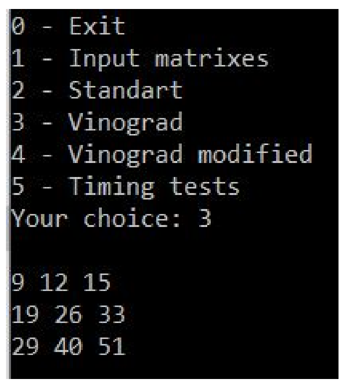

Рисунок 6 - Алгоритм Винограда

На Рисунке 7 показан результат работы модифицированного алгоритма Винограда.

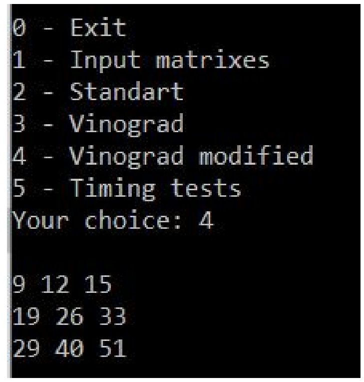

Рисунок 7 - Модифицированный алгоритм Винограда

4.2 Анализ времени работы алгоритмов

Эксперементы проводятся на квадратных матрицах. Элементы матриц задаются случайным образом.

В первом случае размеры матриц: 100x100, 200x200, 300x300, 400x400, 500x500.

Литература

1) Бьерн Страуструп. Язык программирования C++. -URL:

   https://codernet.ru/books/c_plus/bern_straustrup_yazyk_programmirovaniya_ c_specia

   (дата обращения: 01.10.2020). Текст: электронный.

2) Qt. -URL:

   https://www.qt.io/ (дата обращения: 01.10.2020). Текст: электронный.

3) Функция $clock$. -URL:

   https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019
   (дата обращения: 01.10.2020). Текст: электронный.