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»

Отчет
по лабораторной работе № 2

Дисциплина: Анализ алгоритмов

Тема: Алгоритмы умножения матриц

Студент: Платонова Ольга

Группа: ИУ7-55Б

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

Москва.
2020 г.

Оглавление

Введение

В данной лабораторной работе требуется изучить и реализовать алгоритмы классического умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда; осуществить расчет трудоемкости алгоритмов; выполнить сравнительный анализ по времени и трудоемкости для каждой из реализаций.

1. Аналитическая часть

В данном разделе рассмотрим понятия и формулы, связанные с умножением матриц классическим способом и с помощью алгоритма Винограда.

1.1. Классическое умножение матриц

Произведением матрицы A[m×n] на матрицу B[n×k] называется матрица C[m×k] такая, что элемент матрицы C, стоящий в i-ой строке и j-ом столбце, т. е. элемент c_{ij}, равен сумме произведений элементов i-ой строки матрицы A на соответствующие элементы j-ого столбца матрицы B.

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \tag{1.1}$$

Замечание

1. Операция умножения двух матриц выполнима тогда и только тогда, когда число столбцов в первом сомножителе равно числу строк во втором; в этом случае говорят, что матрицы согласованы.
2. Из факта существования произведения *AB* не следует существование произведения *BA* [1].

1.2. Алгоритм Винограда

Из алгоритма классического умножения матриц следует, что каждый элемент результирующей матрицы представляет собой скалярное произведение строки на соответствующий столбец. Работа алгоритма Винограда основана на вычислении двух векторов, скалярное произведение которых позволяет сформировать результирующую матрицу. Рассмотрим эти вектора:

$$V = (v1, v2, v3, v4)$$
$$W = (w1, w2, w3, w4) \tag{1.2}$$

Их скалярное произведение равно:

$$V \bullet W = v1w1 + v2w2 + v3w3 + v4w4 \tag{1.3}$$

Это равенство можно переписать в виде:

$$V \bullet W = (v1 + w2)(v2 + w1) + (v3 + w4)(v4 + w3) - v1v2 - v3v4 - w1w2 - w3w4 \tag{1.4}$$

Отметим, что полученное равенство может быть предварительно вычислено. Для этого требуются вектора, которые являются результатом умножения элементов каждой строки первой матрицы, и столбца второй [2].

1.3. Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и умножения методом Винограда. Основным преимуществом алгоритма Винограда является предварительная обработка произведения правой части формулы *(1.4)*, позволяющая уменьшать количество операций умножения.

2. Конструкторская часть

В данном разделе рассмотрим схемы для каждого из алгоритмов.

2.1. Разработка алгоритмов

На рисунке 1 изображена схема алгоритма *классического умножения* матриц.

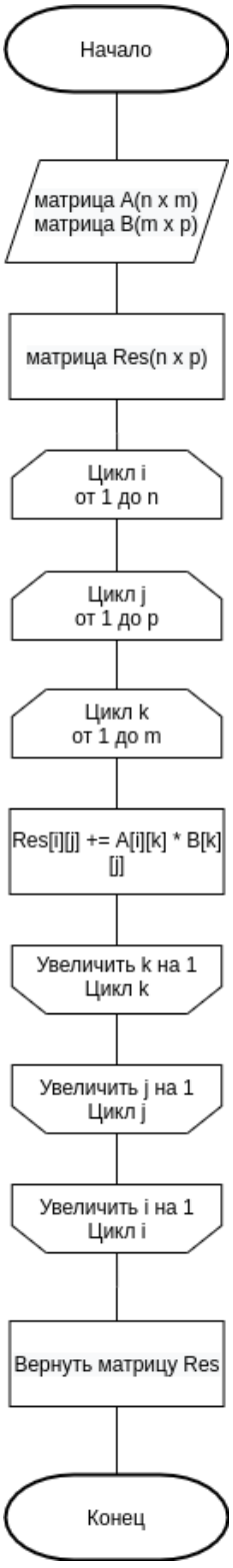


Рисунок 1. Схема алгоритма классического умножения матриц.

На рисунке 2 изображена схема *алгоритма Винограда* умножения матриц.

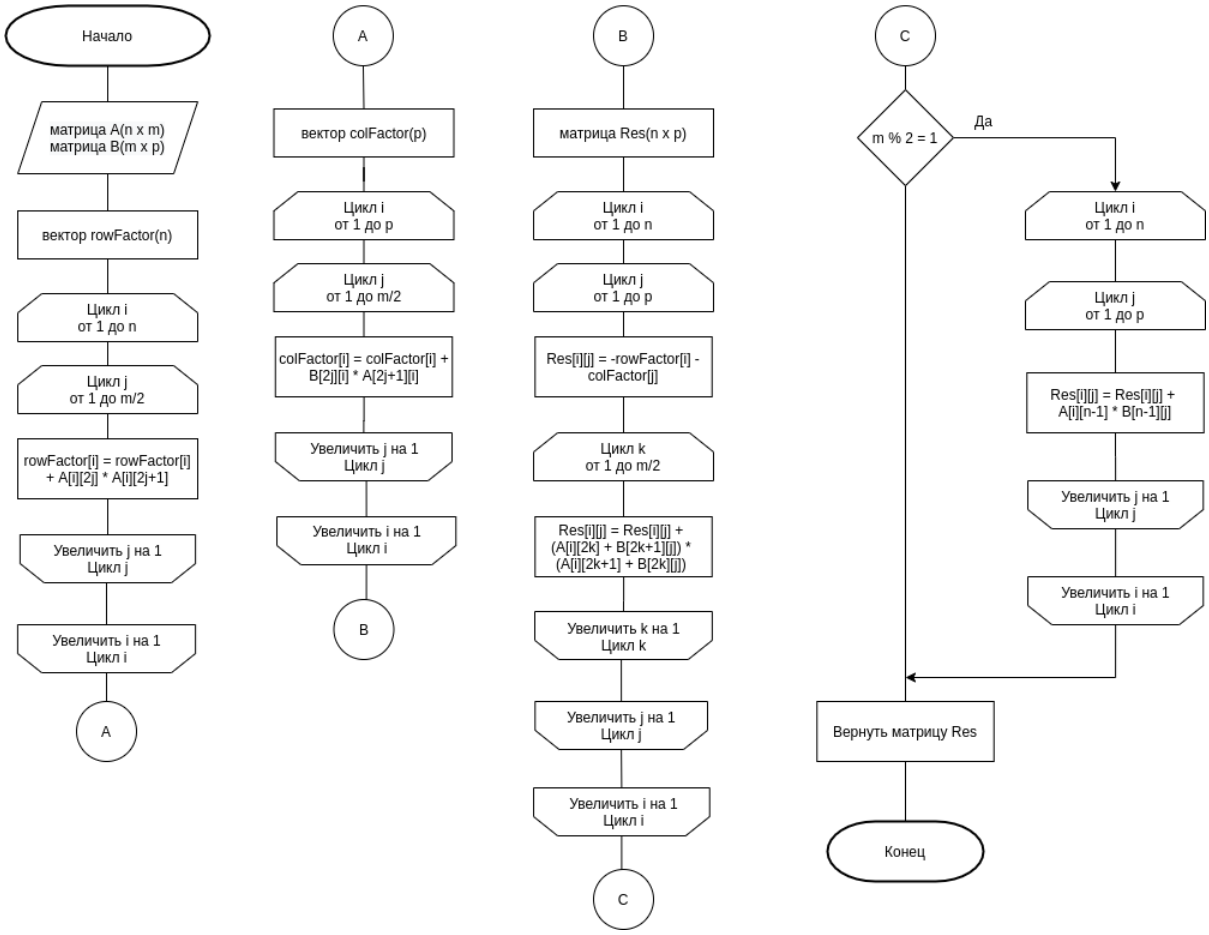


Рисунок 2. Схема алгоритма Винограда.

На рисунке 3 изображена схема *оптимизированного алгоритма Винограда* умножения матриц.

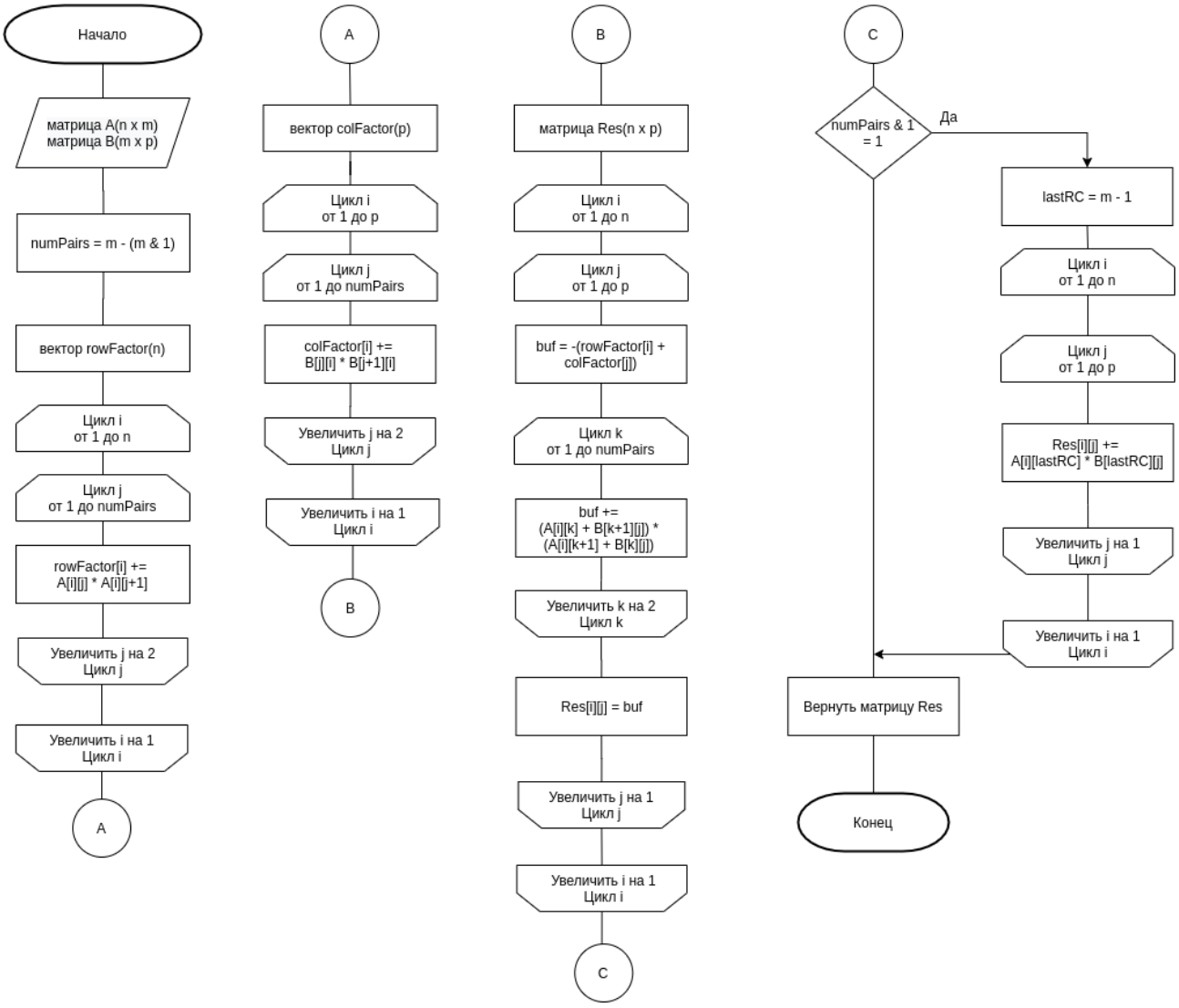


Рисунок 3. Схема оптимизированного алгоритма Винограда.

2.2. Анализ трудоемкости алгоритмов

Введем модель вычисления трудоемкости для анализа описанных выше алгоритмов [3].

1) Стоимость базовых операций единична
   =, +, -, *, /, %, <, >, <=, =>, !=, [], +=, -=, *=, /=, ++, --

2) Стоимость цикла *for (int i = 0; i < M; i++) {}*
   $f_{for} = f_{\text{иниц}} + f_{\text{сравн}} + M(f_{\text{тела}} + f_{\text{иниц}} + f_{\text{сравн}})$
   $f_{for} = 2 + M(f_{\text{тела}} + 2)$

3) Стоимость ветвлений
Стоимость перехода к одной из ветвей – 0.

$$f_{if} = f_{\text{выч.усл}} + \begin{cases} min(f_a, f_b) & - \textit{лучший случай} \\ max(f_a, f_b) & - \textit{худший случай} \end{cases}$$

где f_a, f_b – ветви условного перехода.

Выполним вычисление трудоемкости для каждого из алгоритмов.

2.2.1. Трудоемкость алгоритма классического умножения

Поэтапное вычисление общей трудоемкости алгоритма представлено в таблице 1.

Таблица 1. Заполнение результирующей матрицы

Операция	**Стоимость**
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}})$
Цикл (j от 0 до P)	$2 + P(2 + f_{\text{тела2}})$
Цикл (k от 0 до M)	$2 + M(2 + f_{\text{тела3}})$
Умножение элементов	8

Итоговая трудоемкость: $f_{\text{клс}} = 10MNP + 4NP + 4N + 2$

2.2.2. Трудоемкость алгоритма Винограда

Поэтапное вычисление общей трудоемкости алгоритма представлено в таблицах 2.1, 2.2, 2.3.

Таблица 2.1. Заполнение вектора

Операция	**Стоимость**
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}})$
Цикл (j от 0 до M/2)	$3 + \frac{M}{2}(3 + f_{\text{тела2}})$
Умножение элементов	12

Вектор rowFactor(N): $\frac{15}{2}$MN + 5N + 2

Вектор colFactor(P): $\frac{15}{2}$MP + 5P + 2

Таблица 2.2. Заполнение результирующей матрицы

Операция	**Стоимость**
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}})$
Цикл (j от 0 до P)	$2 + P(2 + f_{\text{тела2}})$
Цикл (k от 0 до M/2)	$7 + 3 + \frac{M}{2}(3 + f_{\text{тела3}})$
Умножение элементов	23

Матрица Res[NxP]: 13MNP + 12NP + 4N + 2

Таблица 2.3. Ветвление

Операция	**Стоимость**
Сравнение	2
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}})$
Цикл (j от 0 до P)	$2 + P(2 + f_{\text{тела2}})$
Умножение элементов	13

Ветвление: 15PN + 4N + 2

Итоговая трудоемкость:

$$f = 13MNP + \frac{15}{2}MN + \frac{15}{2}MP + 12NP + 5N + 5P + 4N + 6 + \begin{cases} 2 & \text{в случае четности} \\ 15PN + 4N + 2 & \text{иначе} \end{cases}$$

2.2.3. Трудоемкость оптимизированного алгоритма Винограда

Поэтапное вычисление общей трудоемкости алгоритма представлено в таблицах 3.1, 3.2, 3.3.

Таблица 3.1. Заполнение вектора

Операция	**Стоимость**
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}})$
Цикл (j от 0 до M/2)	$2 + \frac{M}{2}(2 + f_{\text{тела2}})$
Умножение элементов	8

Вектор rowFactor(N): $5MN + 4N + 2$

Вектор colFactor(P): $5MP + 4P + 2$

Таблица 3.2. Заполнение результирующей матрицы

Операция	**Стоимость**
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}})$
Цикл (j от 0 до P)	$2 + P(2 + f_{\text{тела2}})$
Цикл (k от 0 до M/2)	$5 + 2 + \frac{M}{2}(2 + f_{\text{тела3}}) + 3$
Умножение элементов	14

Матрица Res[NxP]: $8MNP + 12NP + 4N + 2$

3.3. Ветвление

Операция	**Стоимость**
Сравнение	1
Цикл (i от 0 до N)	$2 + N(2 + f_{\text{тела1}}) + 2$
Цикл (j от 0 до P)	$2 + P(2 + f_{\text{тела2}})$
Умножение элементов	8

Ветвление: $10PN + 4N + 4$

Итоговая трудоемкость:

$$f = 8MNP + 12NP + 4N + 2 + 5MN + 4N + 2 + 5MP + 4P + 2 + \begin{cases} 1 & \text{в случае четности} \\ 10PN + 4N + 4 & \text{иначе} \end{cases}$$

2.3. Вывод

В данном разделе были представлены схемы алгоритмов и рассчитана трудоемкость для каждого. Оптимизированный алгоритм Винограда имеет наименьшую трудоемкость. Так, порядок 8 был достигнут благодаря использованию буфера и вычислению нескольких переменных заранее, а не в цикле. Также была произведена оптимизация следующего вида:

- A = A + B A += B
- for (i = 0; i < M/2; i++) A[2*i][j] for (i = 0; i < nP; i+=2) A[i][j].

3. Технологическая часть

В данном разделе рассмотрим язык программирования, среду разработки, требуемые инструменты для реализации. Также представим непосредственно реализацию.

3.1. Требования к программному обеспечению

I. Программа должна предусматривать ввод двух массивов заданной длины. Заполнение массивов должно быть доступно как с клавиатуры, так и автоматически.
II. Выбор применяемого алгоритма осуществляется пользователем из списка алгоритмов, предложенных в меню.
III. На выходе программа выводит результирующую матрицу.
IV. Также необходимо предусмотреть выполнение замеров процессорного времени для каждого из алгоритмов.

3.2. Средства реализации

В данной работе используется язык программирования C++, из-за удобства представления матрицы в виде объекта класса (сокращает количество аргументов для передачи в функцию). Среда разработки – Qt.

Для замеров процессорного времени использовалась функция clock() [4].

3.3. Реализация алгоритмов

Листинг 1

Классический алгоритм умножения матриц.

```
Matrix standartMul(Matrix matrA, Matrix matrB)
{
   Matrix resMatr(matrA.numRows, matrB.numColumns);

   for (int i = 0; i < matrA.numRows; i++) {
      for (int j = 0; j < matrB.numColumns; j++) {
         for (int k = 0; k < matrA.numColumns; k++) {
            resMatr.ptr[i][j] += matrA.ptr[i][k] * matrB.ptr[k][j];
         }
      }
   }
   return resMatr;
}
```

Листинг 2

Алгоритм Винограда умножения матриц.

```
Matrix vinogradMul(Matrix matrA, Matrix matrB)
{
   int rowA = matrA.numRows;
   int sizeAB = matrA.numColumns;
   int colB = matrB.numColumns;

   int numPairs = sizeAB / 2;

   //Произведение элементов в каждой строке matrA
   int *rowFactor = createVector(rowA);
   for (int i = 0; i < rowA; i++) {
      for (int j = 0; j < numPairs; j++) {
         rowFactor[i] = rowFactor[i] + matrA.ptr[i][2 * j] * matrA.ptr[i][2 * j + 1];
      }
   }

   //Произведение элементов в каждом столбце matrB
   int *colFactor = createVector(colB);
   for (int i = 0; i < colB; i++) {
      for (int j = 0; j < numPairs; j++) {
         colFactor[i] = colFactor[i] + matrB.ptr[2 * j][i] * matrB.ptr[2 * j + 1][i];
      }
   }

   //Вычисление произведения
   Matrix matrRes(rowA, colB);
   for (int i = 0; i < rowA; i++) {
      for (int j = 0; j < colB; j++) {
         matrRes.ptr[i][j] = - rowFactor[i] - colFactor[j];
         for (int k = 0; k < numPairs; k++) {
            matrRes.ptr[i][j] = matrRes.ptr[i][j] + (matrA.ptr[i][2 * k] + matrB.ptr[2 * k + 1][j]) *
                                    (matrA.ptr[i][2 * k + 1] + matrB.ptr[2 * k][j]);
         }
      }
   }

   //Случай нечетности
   if (sizeAB % 2 != 0) {
      for (int i = 0; i < rowA; i++) {
         for (int j = 0; j < colB; j++) {
            matrRes.ptr[i][j] = matrRes.ptr[i][j] + matrA.ptr[i][sizeAB - 1] * matrB.ptr[sizeAB - 1][j];
         }
      }
   }
   free(colFactor);
   free(rowFactor);
   return matrRes;
}
```

Листинг 3

Модифицированный алгоритм Винограда умножения матриц.

```
Matrix vinogradOptMul(Matrix matrA, Matrix matrB)
{
   int rowA = matrA.numRows;
   int sizeAB = matrA.numColumns;
   int sizeABmod2 = 1 & sizeAB;
   int colB = matrB.numColumns;

   int numPairsOpt = sizeAB - sizeABmod2;

   //Произведение элементов в каждой строке matrA
   int *rowFactor = createVector(rowA);
   for (int i = 0; i < rowA; i++) {
      for (int j = 0; j < numPairsOpt; j += 2) {
         rowFactor[i] += matrA.ptr[i][j] * matrA.ptr[i][j + 1];
      }
   }

   //Произведение элементов в каждом столбце matrB
   int *colFactor = createVector(colB);
   for (int i = 0; i < colB; i++) {
      for (int j = 0; j < numPairsOpt; j += 2) {
         colFactor[i] += matrB.ptr[j][i] * matrB.ptr[j + 1][i];
      }
   }

   //Вычисление произведения
   Matrix matrRes(rowA, colB);
   for (int i = 0; i < rowA; i++) {
      for (int j = 0; j < colB; j++) {
         int buf = - (rowFactor[i] + colFactor[j]);
         for (int k = 0; k < numPairsOpt; k += 2) {
            buf += (matrA.ptr[i][k] + matrB.ptr[k + 1][j]) *
                  (matrA.ptr[i][k + 1] + matrB.ptr[k][j]);
         }
         matrRes.ptr[i][j] = buf;
      }
   }

   //Случай нечетности sizeAB
   if (sizeABmod2) {
      int lastRC = sizeAB - 1;
      for (int i = 0; i < rowA; i++) {
         for (int j = 0; j < colB; j++) {
            matrRes.ptr[i][j] += matrA.ptr[i][lastRC] * matrB.ptr[lastRC][j];
         }
      }
   }
   free(colFactor);
   free(rowFactor);
   return matrRes;
}
```

3.4. Описание тестирования

Тестирование осуществляется по принципу «черного ящика».

Для проверки корректности программы необходимо предусмотреть наборы различных тестов, включающих в себя случаи матриц, размерности 1, единичных матриц, эквивалентных матриц.

4. Экспериментальная часть

В данном разделе рассмотрим примеры работы программы, произведем тестирование, выполним эксперименты по замеру времени, а также выполним сравнительный анализ полученных данных.

4.1. Примеры работы программы

Взаимодействие с программой (меню, выбор способа создания матрицы, выбор алгоритма, вывод результата) представлено на рисунке 4.

```
Choose the option:
Multiply matrix.......1
Mesure time...........2
1

Matrix creation
First matrix
        Number of rows:2
        Number of columns:3
Second matrix
        Number of rows:3
        Number of columns:5

First matrix
Keyboard input.......1
Generate.............2
1
Input elements.
2 6 4
3 7 0


Second matrix
Keyboard input.......1
Generate.............2
2

1 7 0 7 5
7 1 3 6 1
5 4 5 7 5

Choose the option:
        Standart multiplication.............1
        Vinograd............................2
        Vinograd optimized..................3
        Exit................................4
1

Result matrix
64 36 38 78 36
52 28 21 63 22
Choose the option:
        Standart multiplication.............1
        Vinograd............................2
        Vinograd optimized..................3
        Exit................................4
2

Result matrix
64 36 38 78 36
52 28 21 63 22
```

Рисунок 4. Пример взаимодействия с программой.

4.2. Результаты тестирования

Результаты тестирования приведены в таблице 4.

Таблица 4. Результат тестирования

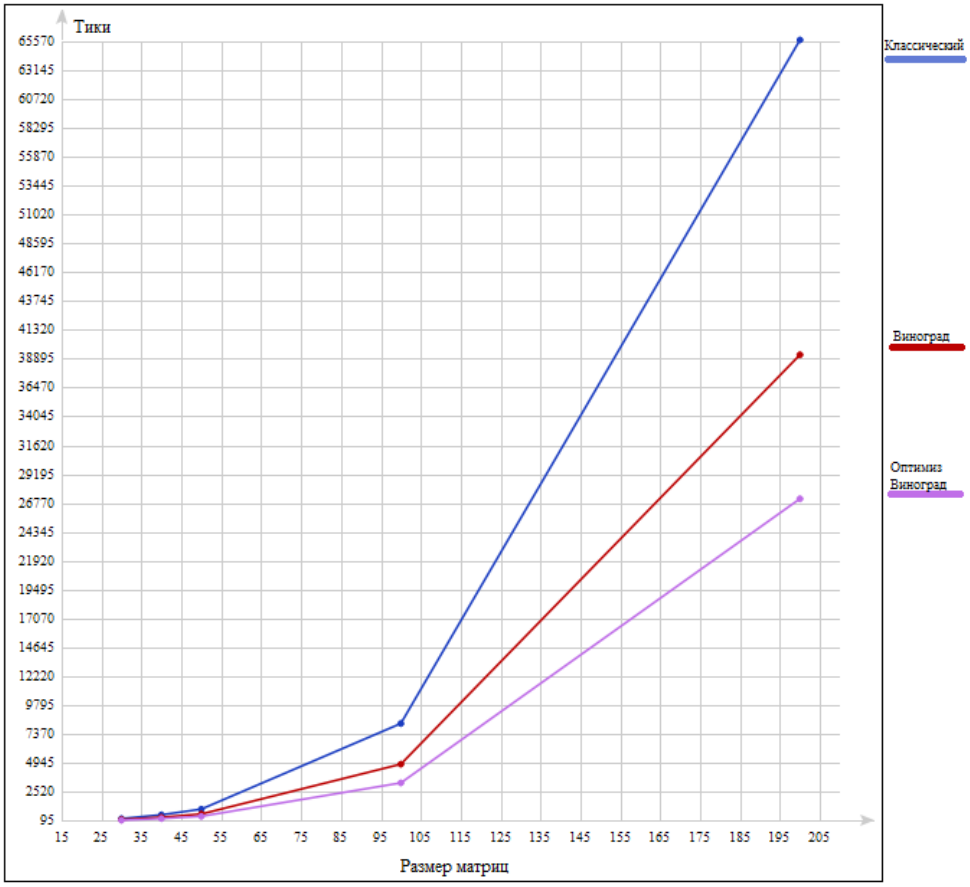
Входные параметры		Ожидаемый результат	Полученный результат
Матрица A	Матрица B		
4	8	32	32
1 7 0 7 5 7	1 3 6 1 5 4 5 7 5 4 6 0 7 1 8 8 6 6	36 52 41 29 47 4 81 63 123 83 107 70	36 52 41 29 47 4 81 63 123 83 107 70
1 0 0 0 1 0 0 0 1	1 7 0 7 5 7 1 3 6 1 5 4	1 7 0 7 5 7 1 3 6 1 5 4	1 7 0 7 5 7 1 3 6 1 5 4
1 2 3 4 8 2 8 4 6	1 2 3 4 8 2 8 4 6	33 30 25 52 80 40 72 72 68	33 30 25 52 80 40 72 72 68

Поскольку результаты работы алгоритмов совпадают, в столбце «Полученный результат» приведено значение, одинаковое для всех алгоритмов.

4.3. Сравнительный анализ времени

Выполним эксперименты по замеру времени, вычисляя произведение квадратных матриц. Эксперименты проводятся на матрицах размерами 20, 21, 50, 51, 100, 101, 200, 201. Поскольку работа алгоритма Винограда зависит от четности размеров, разобьем тестирование на четный и нечетный случаи. Результаты замеров приведены на рисунке 4.1.

Рисунок 4.1. График зависимости времени работы алгоритмов от размеров матриц.



Отметим, что самым долгим алгоритмом для матриц больших размеров является классический алгоритм умножения. Самым быстрым – оптимизированный алгоритм Винограда. Что касается матриц небольших размеров (до 50), то скорость алгоритмов не сильно отличается.

Заключение

В ходе лабораторной работы мы разработали и реализовали алгоритмы умножения матриц: классический, Винограда, оптимизированный Винограда, а также провели анализ трудоемкости и скорости работы каждого из метода.

В результате анализа было установлено, что самую низкую трудоемкость имеет оптимизированный алгоритм Винограда, который является самым быстрым для работы с матрицами больших размеров. Самым трудоемким является неоптимизированный алгоритм Винограда, который работает быстрее классического метода.

Так, на квадратных матрицах с размерами 200, классический алгоритм работает дольше в 1.6 раз Винограда и в 2.4 раза дольше оптимизированного Винограда. Также наблюдается разница в случае четного и нечетного размера: в случае нечетности алгоритмы незначительно замедляются (около 7%).

Список литературы

1. Линейная алгебра.-URL: http://www.academiaxxi.ru/WWW_Books/HM/La/toc.htm (дата обращения: 05.09.2020). Текст: электронный.
2. S. A. Winograd. A new algorithm for inner product // IEEE Trans. Comp. — 1968. — C-18.
3. Л. Д. Ефимова (2010). Быстрые гибридные алгоритмы умножения матриц.
4. Техническая документация.-URL: https://docs.microsoft.com/ru-ru/cpp/c-runtime-library/reference/clock?view=vs-2019 (дата обращения: 20.09.2020). Текст: электронный.