МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Н. Э. БАУМАНА

УДК _____
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

Преподаватель

_____

«_____» _____ 2020 г.

ДИСЦИПЛИНА АНАЛИЗ АЛГОРИТМОВ

ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Произведение матриц
(промежуточный)

Студент _____ Ф.М. Набиев

Преподаватели Л.Л. Волкова, Ю.В. Строганов

Москва, 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной лабораторной работы является изучение методов оценки и улучшения алгоритмов на примере умножения матриц. Будут рассмотрены алгоритм умножения матриц, основанный на определении этой операции, алгоритм Винограда и модифицированный алгоритм Винограда. Для достижения поставленной цели необходимо решить следующие задачи:

— изучить стандартный алгоритм умножения умножения матриц и алгоритм Винограда;

— улучшить алгоритм Винограда;

— дать теоретическую оценку изученным алгоритмам;

— реализовать рассмотренные алгоритмы;

— сравнить реализованные алгоритмы.

1 Аналитический раздел

В данном разделе будет определена теоретическая база, необходимая для реализации поставленных задач.

В соответствии с книгой [1] дадим определение произведения двух матриц. Пусть даны прямоугольные матрицы A и B. Размеры этих матриц $n \times r$ и $r \times m$ соответственно. Тогда результатом умножения матрицы A на матрицу B называется такая матрица C размера $n \times m$, что:

$$c_{ij} = \sum_{k=1}^{r}(a_{ik} \cdot b_{kj}) \tag{1.1}$$

где $i = \overline{1, n}$, $j = \overline{1, m}$.

1.1 Описание алгоритмов

Рассмотрим алгоритмы вычисления произведения матриц.

1.1.1 Классический алгоритм умножения

Классический алгоритм умножения матриц дословно повторяет определение данной операции:

$$\begin{bmatrix} a_{11} & ... & a_{1r} \\ ... & ... & ... \\ a_{n1} & ... & a_{nr} \end{bmatrix} \times \begin{bmatrix} b_{11} & ... & b_{1m} \\ ... & ... & ... \\ b_{r1} & ... & b_{rm} \end{bmatrix} = \begin{bmatrix} c_{11} & ... & c_{1m} \\ ... & ... & ... \\ c_{n1} & ... & c_{nm} \end{bmatrix} \tag{1.2}$$

где c_{ij} вычисляется по формуле (1.1)[2].

1.1.2 Алгоритм Винограда

В качестве примера рассмотрим два вектора длинной 4: $U = (u_1, u_2, u_3, u_4)$ и $V = (v_1, v_2, v_3, v_4)$. Их скалярное произведение равно:

$$U \times V = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3 + u_3 \cdot v_4 \tag{1.3}$$

Левую часть равенства (1.3) можно записать в виде:

$$(u_1 + v_2) \cdot (u_2 + v_1) + (u_3 + v_4) \cdot (u_4 + v_3) - u_1 \cdot u_2 - u_3 \cdot u_4 - v_1 \cdot v_2 - v_3 \cdot v_4 \tag{1.4}$$

Выражение (1.4) допускает предварительную обработку в случае умножения двух матриц. Для каждой строки можно вычислить выражение (1.5):

$$u_1 \cdot u_2 + u_3 \cdot u_4 \tag{1.5}$$

А для каждого столбца - выражение (1.6):

$$v_1 \cdot v_2 + v_3 \cdot v_4 \tag{1.6}$$

Таким образом, выходит, что над заранее обработанными данными необходимо выполнить лишь 2 умножения, 5 сложений и 2 вычитания.

1.2 Вывод

Были рассмотрены классический алгоритм алгоритм умножения матриц и алгоритм Винограда, определено, что суть второго заключается в уменьшении количества операций умножения за счёт усложнения последовательности вычислений.

2 Конструкторский раздел

В данном разделе будет проведена конкретизация поставленных задач, составлены и проанализированы алгоритмы.

2.1 Модель

IDEF0 модель задачи вычисления произведения двух матриц представлена на рисунке 2.1.

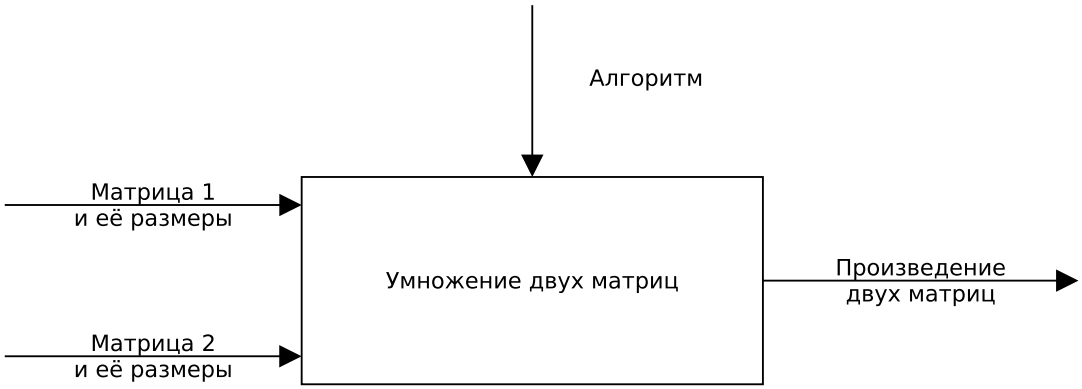

Рисунок 2.1 — IDEF0 модель

2.2 Разработка алгоритмов

Для непосредственной реализации вышеописанных алгоритмов важно иметь их некоторые упрощённые визуальные представления, так как чтение таких представлений упрощает написание кода. Подходящим для этого вариантом визуализации являются схемы алгоритмов.

2.2.1 Классический алгоритм умножения матриц

Схема классического алгоритма умножения матриц приведена на рисунке 2.2.

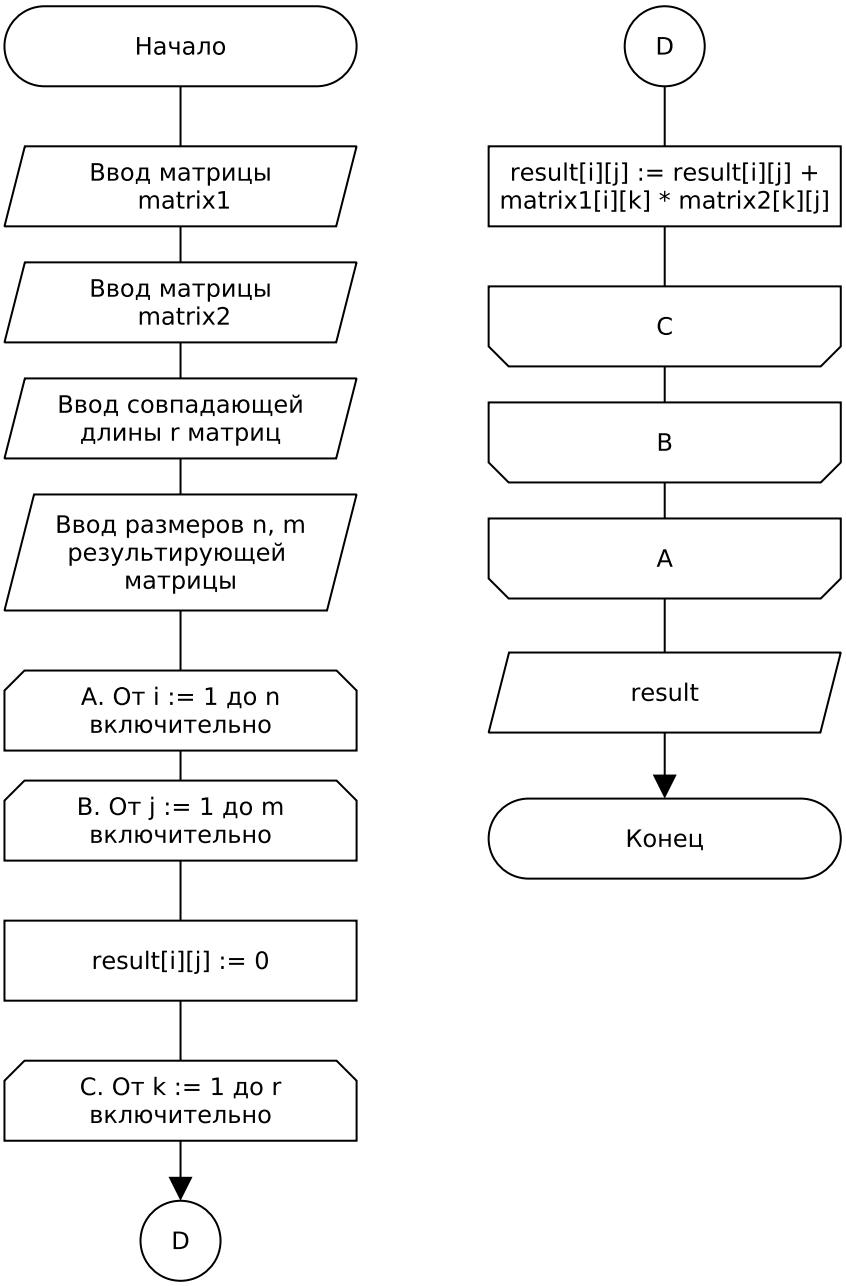

Рисунок 2.2 — Классический алгоритм

2.2.2 Алгоритм Винограда

Схема алгоритма Винограда приведена на рисунках 2.3 и 2.4.

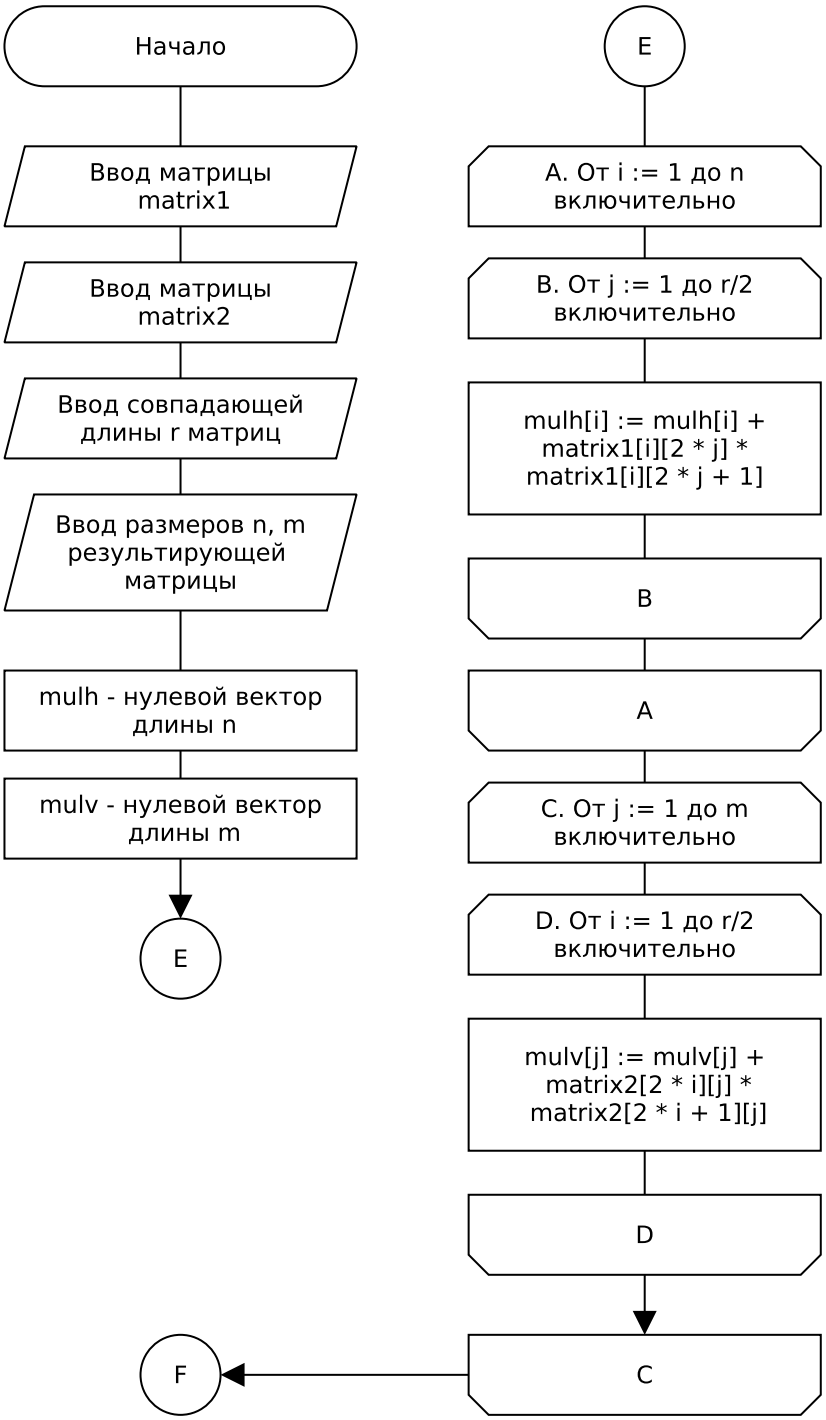

Рисунок 2.3 — Алгоритм Винограда, часть 1

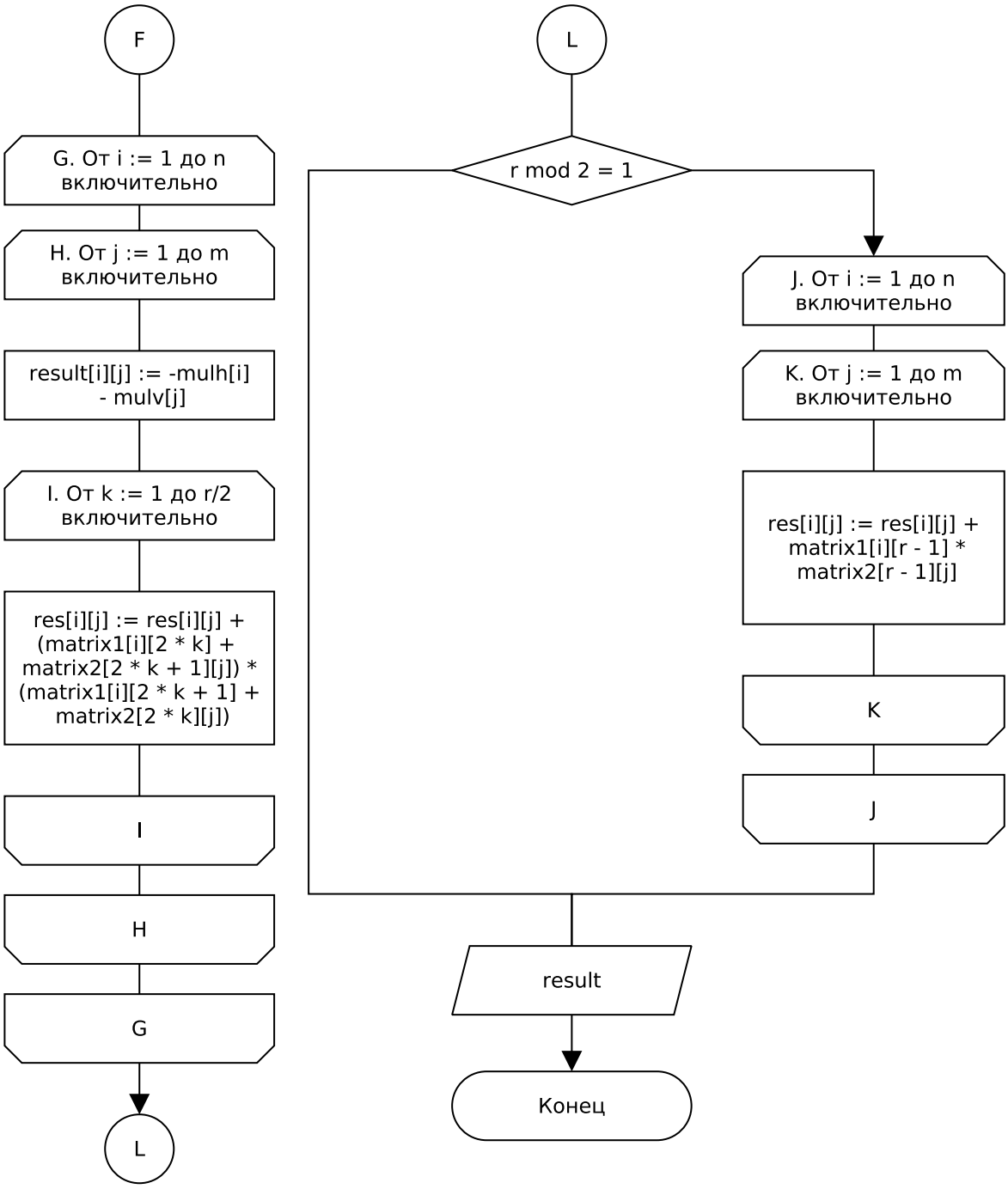

Рисунок 2.4 — Алгоритм Винограда, часть 2

2.3 Трудоёмкость алгоритмов

Для того, чтобы произвести оценку трудоёмкости рассматриваемых алгоритмов, определим стоимость базовых операций:

— операции стоимостью 0:

— условный переход;

— операции стоимостью 1:

— операции сравнения;

— арифметическая унарная операция -;

— арифметические бинарные операции: +, -;

— операции инкремента и декремента;

— битовые операции;

— операции работы с памятью: разыменование, взятие адреса;

— операции присваивания: простая и сложные, совмещенные с операциями из этого списка;

— операции стоимостью 2:

— арифметические бинарные операции: *, /, %;

— операции работы с памятью: индексация;

— составные операции присваивания, совмещенные с другими операциями этого списка.

Тогда стоимость выполнения си-подобного параметрического цикла for определяется стоимостью полей инициализации, условия и модификации его заголовка, а так же стоимостью тела этого цикла и количеством его срабатываний.

2.3.1 Классический алгоритм

Оценим трудоёмкость классического алгоритма умножения двух матриц. Обозначения идентичны обозначениям на рисунке 2.2.

Данный алгоритм начинается с параметрического цикла от $i := 1$ до n. Значит, этот цикл сработает n раз, а цена его заголовка составляет $1 + 1 + 1 = 1 + 2$, где цена инициализации составляет 1, а остальная цена

заголовка равна 2. Подобных циклов всего 3 и каждый следующий вложен в предыдущий.

Внутри второго присутствует обращение к элементу матрицы и присваивание этому элементу значение 0. Стоимость этого выражения составляет $2 + 2 + 1 = 5$.

Внутри третьего цикла присутствует выражение, состоящее из 4 обращений к элементу матрицы, одного умножения, одного сложения и одного присваивания. Следовательно, его стоимость равна $4 \cdot (2 + 2) + 2 + 1 + 1 = 20$.

Имеем итоговую стоимость всего алгоритма:

$$\begin{gathered} 1 + n \cdot (3 + m \cdot (3 + 5 + r \cdot (2 + 20))) = \\ 22nmr + 8nm + 3n + 1 \end{gathered} \tag{2.1}$$

2.3.2 Алгоритм Винограда

Теперь проведём оценку трудоёмкости алгоритма Винограда. Обозначения идентичны обозначениям на рисунка 2.3 и 2.4.

Как видно на рисунке 2.3, в начале схемы данного алгоритма присутствуют два однотипных двойных цикла. Для упрощения вычислений, рассмотрим первый. В предыдущем пункте уже было объяснено, что стоимость заголовка такого цикла составляет 3, а выполнится он n раз (инициализация выполняется всего один раз, поэтому 1 + 2n). Стоимость заголовка вложенного цикла выше на 2 - там присутствует операция целочисленного деления, а выполнится он $\frac{r}{2}$ раз. Внутри второго цикла находится только одно выражение, состоящее из из двух обращений к элементу матрицы, двух обращений к элементу вектора, трёх произведений, двух сложений и одного присваивания. Цена этого выражения равна $2 \cdot (2 + 2) + 2 \cdot 2 + 3 \cdot 2 + 2 \cdot 1 + 1 = 21$. Стоимость всего цикла составляет:

$$1 + n \cdot (3 + \frac{r}{2} \cdot (4 + 21)) = 12.5 \cdot nr + 3n + 1 \tag{2.2}$$

А стоимость следующего:

$$1 + m \cdot (3 + \frac{r}{2} \cdot (4 + 21)) = 12.5 \cdot mr + 3m + 1 \tag{2.3}$$

Далее следует тройной цикл, с точки зрения структуры аналогичный описанному в предыдущем пункте, но с важными отличиями:

— стоимость самого вложенного цикла выше на 2, так как в его условии присутствует дополнительная операция деления;

— самый вложенный цикл выполнится не за r раз, а за $\frac{r}{2}$ раз;

— стоимость выражения, с которого начинается тело второго цикла, равна 11;

— стоимость выражения, из которого состоит тело третьего цикла, равна 37.

Стоимость всего цикла равна:

$$1 + n \cdot (3 + m \cdot (3 + 11 + \frac{r}{2} \cdot (4 + 37))) = 20.5nmr + 14nm + 3n + 1 \quad (2.4)$$

Схема данного алгоритма завершается ветвлением, результатом которого может быть всего два исхода: выполнение двойного цикла или пропуск этого цикла. Цена проверки условия этого ветвления составляет $2 + 1 = 3$. Стоимость заголовков и количество срабатываний следующих циклов ранее было вычислено: 3, n раз и 2, m раз соответственно. Выражение, находящееся во вложенном цикле, состоит из 4 обращений к элементу матрицы, одного умножения, двух вычитаний, одного сложения и одного присваивания. Значит, его стоимость равна $4 \cdot (2 + 2) + 2 + 2 \cdot 1 + 1 + 1 = 22$. Тогда стоимость всего цикла равна:

$$1 + n \cdot (3 + m \cdot (2 + 22)) = 24nm + 3n + 1 \quad (2.5)$$

Таким образом, можно подсчитать итоговую стоимость всего алгоритма:

$$20.5nmr + 14nm + 12.5nr + 12.5mr + 6n + 3m + 6 + Z \quad (2.6)$$

где

$$Z = \begin{cases} 24nm + 3n + 1, & \text{если r нечётно} \\ 0, & \text{иначе} \end{cases} \quad (2.7)$$

2.3.3 Модификация алгоритма Винограда

Очевидно, что алгоритм Винограда при перемножении больших матриц будет работать быстрее классического алгоритма умножения. Но даже этот результат можно улучшить, произведя ряд модификаций:

— замена комбинации операции присваивания и операции сложения на соответствующую составную операцию присваивания. Например, выражение вида $a = a + ...$ будет заменено на $a+ = ...$;

— замена операции целочисленного деления на 2 операцией битового логического сдвига вправо на 1. Аналогично с операцией целочисленного умножения на 2 и операцией битового логического сдвига влево на 1;

— замена в проверке на чётность операции получения остатка от деления на 2 на операцию побитового умножения на 1;

— перенос из условия цикла лишних вычислений во внешнюю область видимости;

— буферизация значений, записываемых в память посредством использования операции разыменования, при условии, что это происходит в цикле.

С учётом произведенных изменений, имеем схему улучшенного алгоритма Винограда на рисунках 2.5 и 2.6.

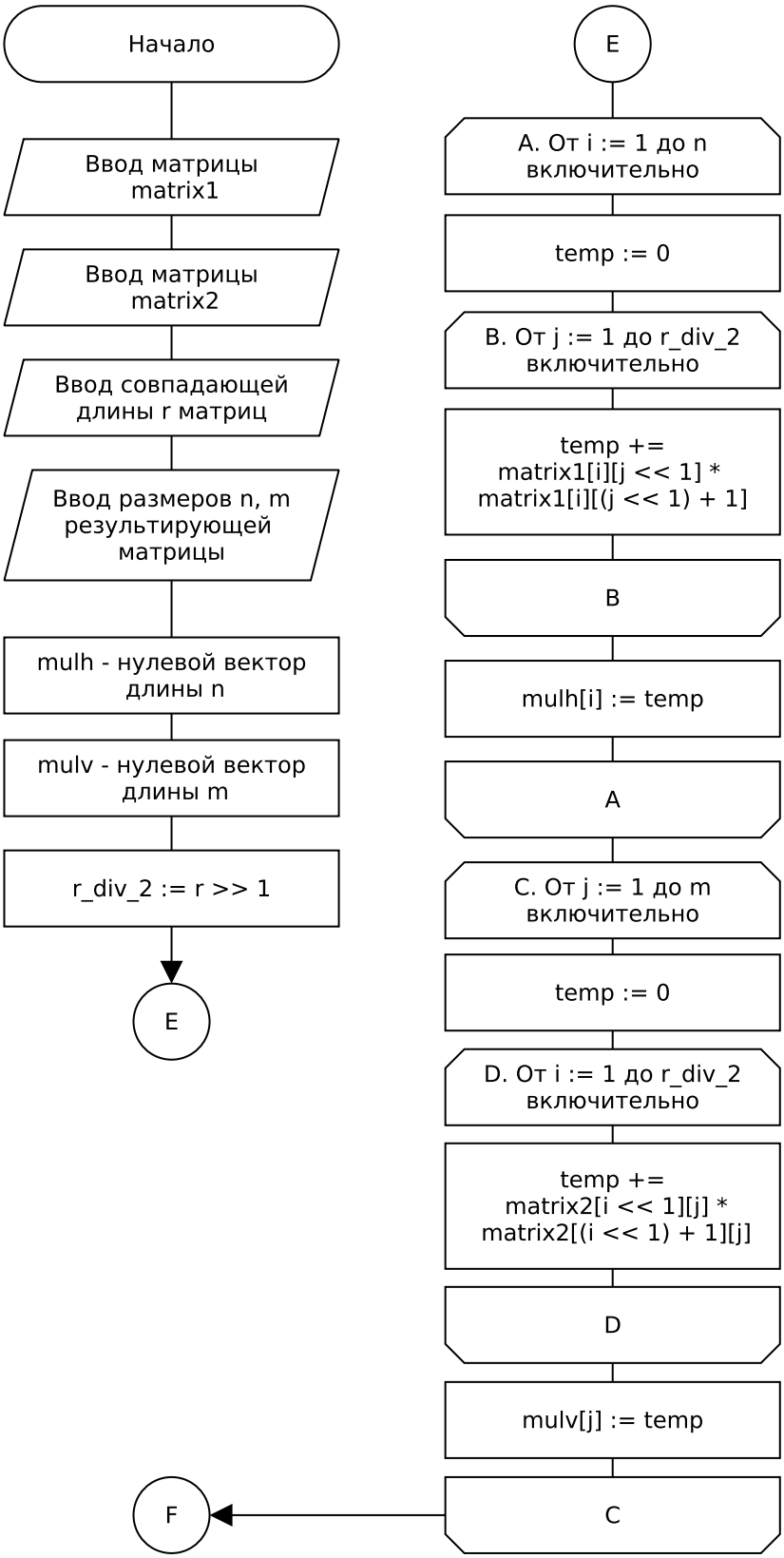

Рисунок 2.5 — Модифицированный алгоритм Винограда, часть 1

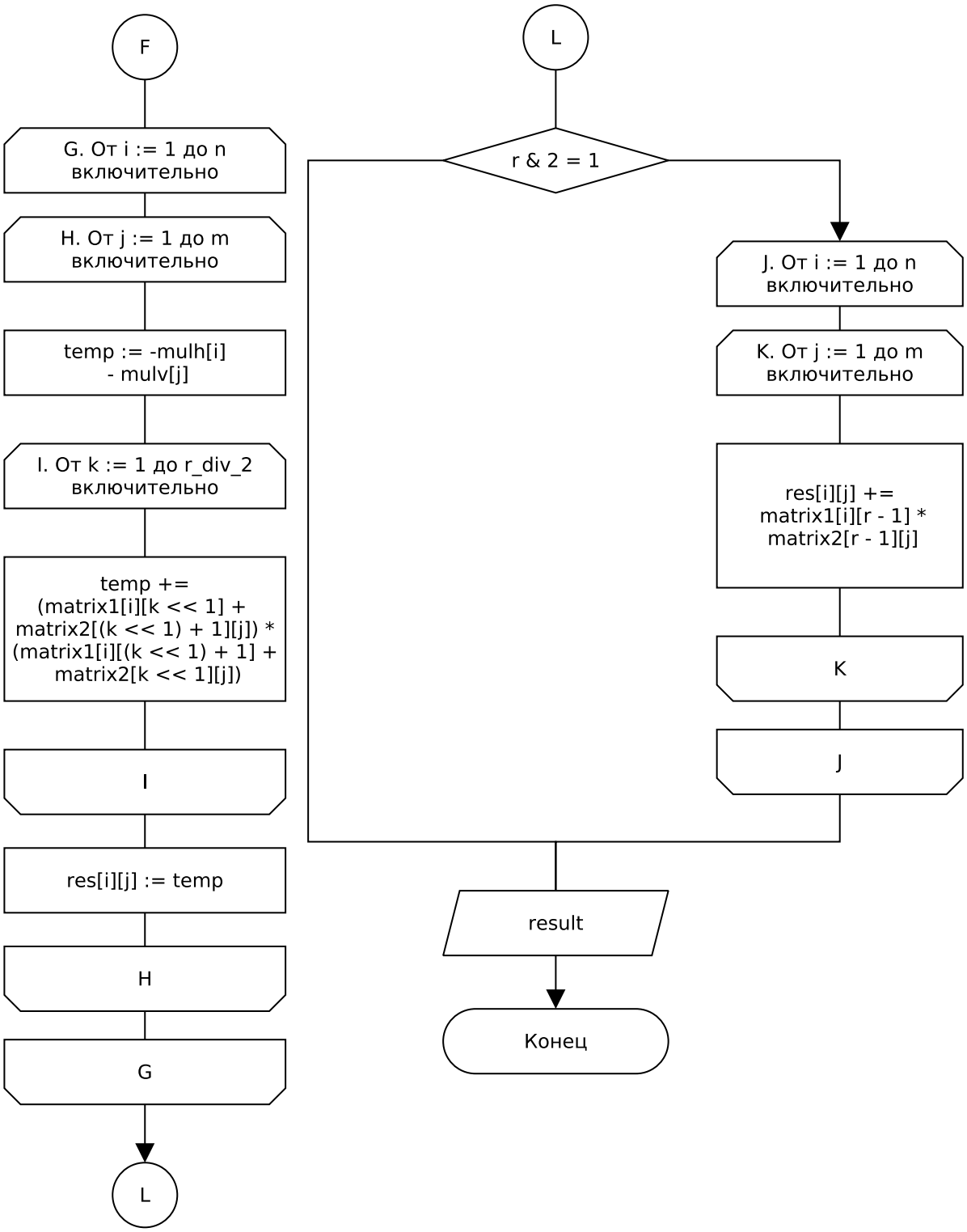

Рисунок 2.6 — Модифицированный алгоритм Винограда, часть 2

Имея схему модифицированного алгоритма Винограда, можем произвести оценку его сложности. Вначале схемы присутствует вычисление $r/2$ и присваивание полученного значения в переменную. Так как деление на 2

осуществляется при помощи двоичного сдвига, то итоговая стоимость данного выражения составляет 2.

Далее следуют два уже рассмотренных однотипных цикла с новой операцией присваивания и буферизацией вычисляемого значения вектора. С учётом произведённых изменений имеем общую стоимость первого цикла:

$$1 + n \cdot (3 + 1 + 3 + \frac{r}{2} \cdot (2 + 15)) = 8.5nr + 7n + 1 \tag{2.8}$$

и второго:

$$1 + m \cdot (3 + 1 + 3 + \frac{r}{2} \cdot (2 + 15)) = 8.5mr + 7m + 1 \tag{2.9}$$

Теперь рассмотрим стоимость модифицированного тройного цикла:

$$1 + n \cdot (3 + m \cdot (4 + \frac{r}{2} \cdot (2 + 25) + 5) = 13.5nmr + 9nm + 3n + 1 \tag{2.10}$$

Остается вычислить стоимость завершающего алгоритм ветвления. Цена условия равна 2. а стоимость блока, выполняемого в случае истинности этого условия, составляет:

$$1 + n \cdot (3 + m \cdot (2 + 17)) = 19nm + 5n + 1 \tag{2.11}$$

Таким образом имеем суммарную цену:

$$13.5nmr + 9nm + 8.5nr + 8.5mr + 10n + 7m + 7 + Z \tag{2.12}$$

где

$$Z = \begin{cases} 19nm + 5n + 1, & \text{если r нечётно} \\ 0, & \text{иначе} \end{cases} \tag{2.13}$$

2.4 Вывод

По итогу проведённой оценки трудоёмкости разработанных алгоритмов можно сделать вывод, что при $n \to \infty$ алгоритм Винограда будет работать быстрее, чем стандартный алгоритм умножения матриц. В свою очередь модифицированный алгоритм Винограда теоретически должен работать быстрее алгоритма Винограда.

3 Технологический раздел

В данном разделе будут составлены требования к программному обеспечению, выбраны средства реализации и определены тестовые данные.

3.1 Требования к программному обеспечению

Требования к вводу:

— $n_1, m_1 \in \mathbb{N}$ - размеры первой матрицы;

— $n_2, m_2 \in \mathbb{N}$ - размеры второй матрицы;

— $n1 \cdot m1$ действительных чисел, разделенных символами-разделителями (пробел, перенос строки, табуляция и т.п.) - элементы первой матрицы, перечисленные построчно;

— $n2 \cdot m2$ действительных чисел, разделенных символами-разделителями - элементы второй матрицы, перечисленные построчно.

Требования к выводу:

— результат умножения матриц.

Требования к программе:

— выбор алгоритма происходит через аргументы командной строки путём передачи его номера:

1) стандартный алгоритм;

2) алгоритм Винограда;

3) модифицированный алгоритм Винограда.

3.2 Средства реализации

Для реализации программы вычисления редакционного расстояния мной был выбран язык программирования C++. В рамках текущей задачи данный язык программирования имеет ряд существенных преимуществ:

— Статическая типизация;

— Близость к низкоуровневому C при наличии многих возможностей высокоуровненных языков;

— Встроенная библиотека std::chrono, позволяющая измерять процессорное время [3].

3.3 Листинги кода

Листинг 3.1 — Классический алгоритм

```
1  void ClassicMul(double **res, double **matrix1, double **matrix2,
2                  size_t n, size_t m, size_t r)
3  {
4      for (size_t i = 0; i < n; i++)
5      {
6          for (size_t j = 0; j < m; j++)
7          {
8              res[i][j] = 0;
9
10             for (size_t k = 0; k < r; k++)
11             {
12                 res[i][j] += matrix1[i][k] * matrix2[k][j];
13             }
14         }
15     }
16 }
```

Листинг 3.2 — Алгоритм Винограда

```
1  void WinogradMul(double **res, double **matrix1, double **matrix2,
2                   size_t n, size_t m, size_t r)
3  {
4      double *mulh = new double[n]();
5      double *mulv = new double[m]();
6
7      for (size_t i = 0; i < n; i++)
8      {
9          for (size_t j = 0; j < r / 2; j++)
10         {
11             mulh[i] = mulh[i] + matrix1[i][2 * j] * matrix1[i][2 * j + 1];
12         }
13     }
14
15     for (size_t j = 0; j < m; j++)
```

```
16      {
17          for (size_t i = 0; i < r / 2; i++)
18          {
19              mulv[j] = mulv[j] + matrix2[2 * i][j] * matrix2[2 * i + 1][j];
20          }
21      }
22
23      for (size_t i = 0; i < n; i++)
24      {
25          for (size_t j = 0; j < m; j++)
26          {
27              res[i][j] = -(mulh[i] + mulv[j]);
28
29              for (size_t k = 0; k < r / 2; k++)
30              {
31                  res[i][j] = res[i][j] +
32                              (matrix1[i][2 * k] + matrix2[2 * k + 1][j]) *
33                              (matrix1[i][2 * k + 1] + matrix2[2 * k][j]);
34              }
35          }
36      }
37
38      if (r % 2 == 1)
39      {
40          for (size_t i = 0; i < n; i++)
41              for (size_t j = 0; j < m; j++)
42                  res[i][j] = res[i][j] +
43                              matrix1[i][r - 1] * matrix2[r - 1][j];
44      }
45
46      delete[] mulh;
47      delete[] mulv;
48  }
```

Листинг 3.3 — Модифицированный аллгоритм Винограда

```
1  void OWinogradMul(double **res, double **matrix1, double **matrix2,
2                    size_t n, size_t m, size_t r)
3  {
4      double temp;
5
6      double *mulh = new double[n]();
7      double *mulv = new double[m]();
8
9      double r_div_2 = r >> 1;
```

```
10
11      for (size_t i = 0; i < n; i++)
12      {
13          temp = 0;
14
15          for (size_t j = 0; j < r_div_2; j++)
16          {
17              temp += matrix1[i][j << 1] * matrix1[i][(j << 1) + 1];
18          }
19
20          mulh[i] = temp;
21      }
22
23      for (size_t j = 0; j < m; j++)
24      {
25          temp = 0;
26
27          for (size_t i = 0; i < r_div_2; i++)
28          {
29              temp += matrix2[i << 1][j] * matrix2[(i << 1) + 1][j];
30          }
31
32          mulv[j] = temp;
33      }
34
35      for (size_t i = 0; i < n; i++)
36      {
37          for (size_t j = 0; j < m; j++)
38          {
39              temp = -(mulh[i] + mulv[j]);
40
41              for (size_t k = 0; k < r_div_2; k++)
42              {
43                  temp += (matrix1[i][k << 1] + matrix2[(k << 1) + 1][j]) *
44                          (matrix1[i][(k << 1) + 1] + matrix2[k << 1][j]);
45              }
46
47              res[i][j] = temp;
48          }
49      }
50
51      if (r & 1)
52      {
53          for (size_t i = 0; i < n; i++)
54              for (size_t j = 0; j < m; j++)
55                  res[i][j] += matrix1[i][r - 1] * matrix2[r - 1][j];
```

```
56        }
57
58        delete[] mulh;
59        delete[] mulv;
60    }
```

3.4 Примеры работы

На рисунках 3.1, 3.2, 3.4 приведены примеры работы реализованной программы в случаях пустого ввода, ввода невозможных размеров матриц и корректного ввода соответственно.

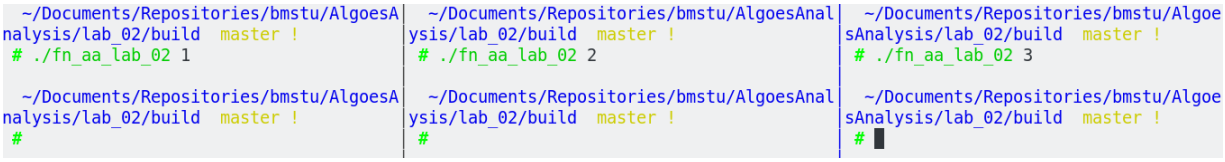

Рисунок 3.1 — Пустой ввод

Рисунок 3.2 — Невозможный размер

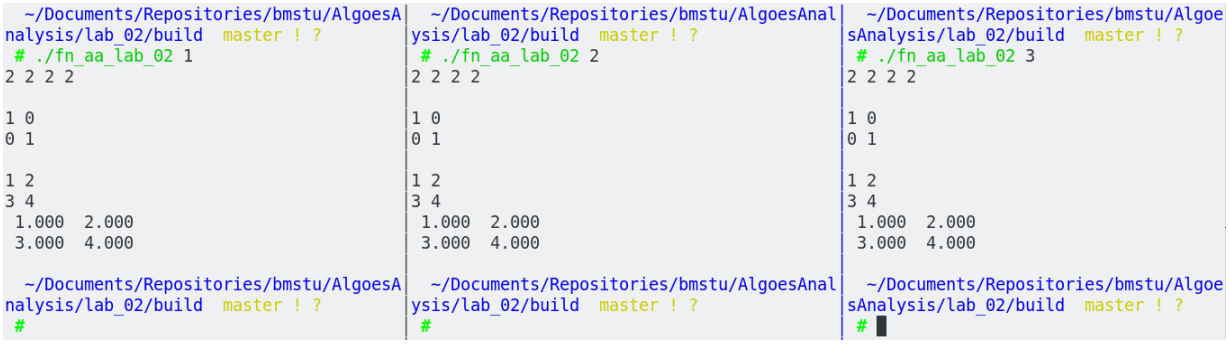

Рисунок 3.3 — Корректный ввод

3.5 Описание тестирования

Для проверки корректной работы реализованной программы были подготовлены тестовые данные, представленные в таблице 3.1.

Таблица 3.1 — Тестовые данные и ожидаемые результаты

Первая матрица	Вторая матрица	Ожидаемый результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 2 3 4 5 6	1 2 3	14 32

Тестирование было проведено успешно. Его результаты приведены в таблицах 3.2, 3.3, 3.4.

Таблица 3.2 — Результаты тестирования стандартного алгоритма

Первая матрица	Вторая матрица	Полученный результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 2 3 4 5 6	1 2 3	14 32

Таблица 3.3 — Результаты тестирования алгоритма Винограда

Первая матрица	Вторая матрица	Полученный результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 2 3 4 5 6	1 2 3	14 32

Таблица 3.4 — Результаты тестирования модифицированного алгоритма Винограда

Первая матрица	Вторая матрица	Полученный результат
1 2 3 4	1 2 3 4	7 10 15 22
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	30 36 42 66 81 96 102 126 150
1 2 3 4 5 6	1 2 3	14 32

3.6 Вывод

Для реализации программы были выбраны средства разработки: язык программирование C++, библиотека std::chrono, а так же подготовлены тестовые данные и успешно проведено тестирование.

4 Исследовательский раздел

В данном разделе будут производиться эксперементы над корректно реализованной программой.

4.1 Эксперименты по замеру времени

На рисунках 4.1 и 4.2 изображены графики зависимости времени выполнения программы от размера входных матриц. Первый график соответствует квадратным матрицам с чётным размером, а второй - нечётным.

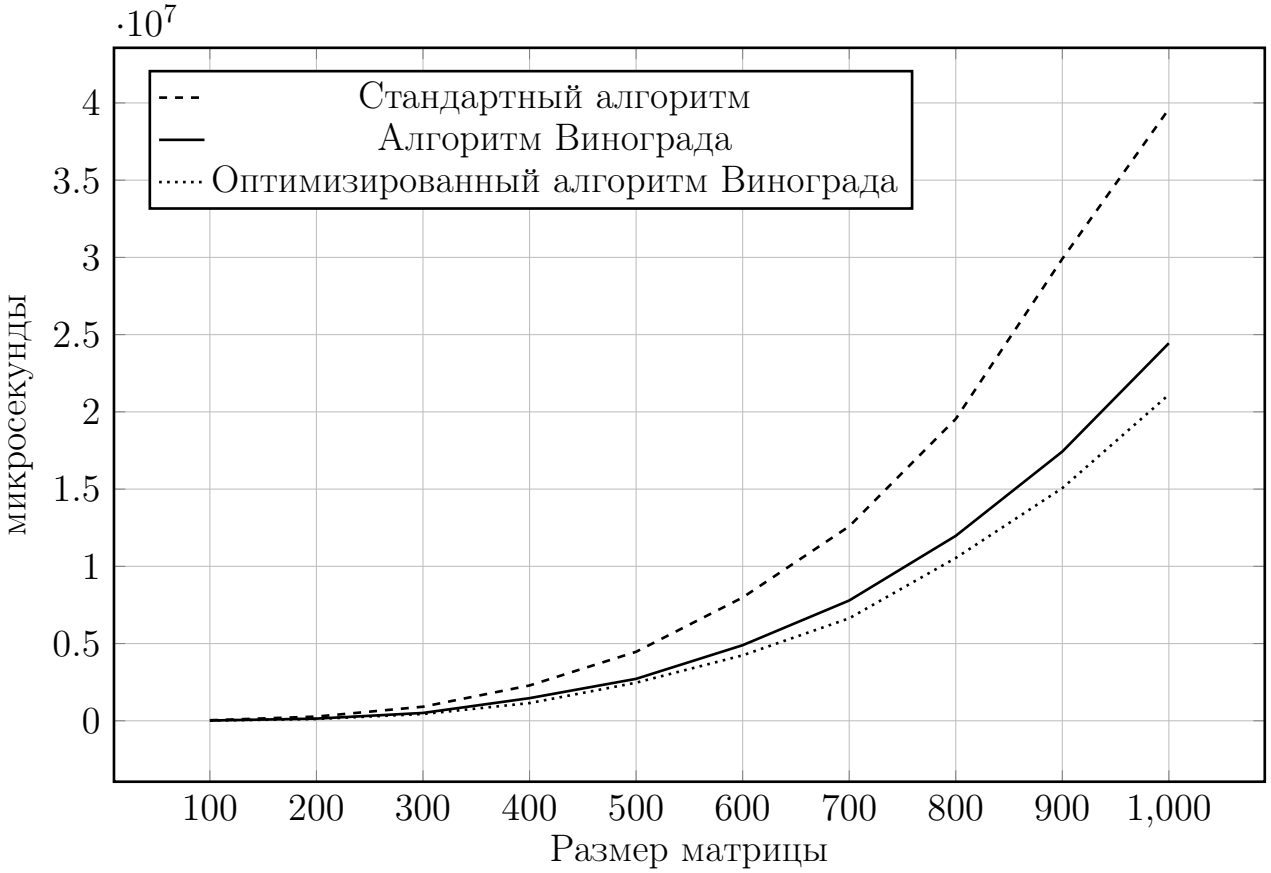

Рисунок 4.1 — Чётные размеры

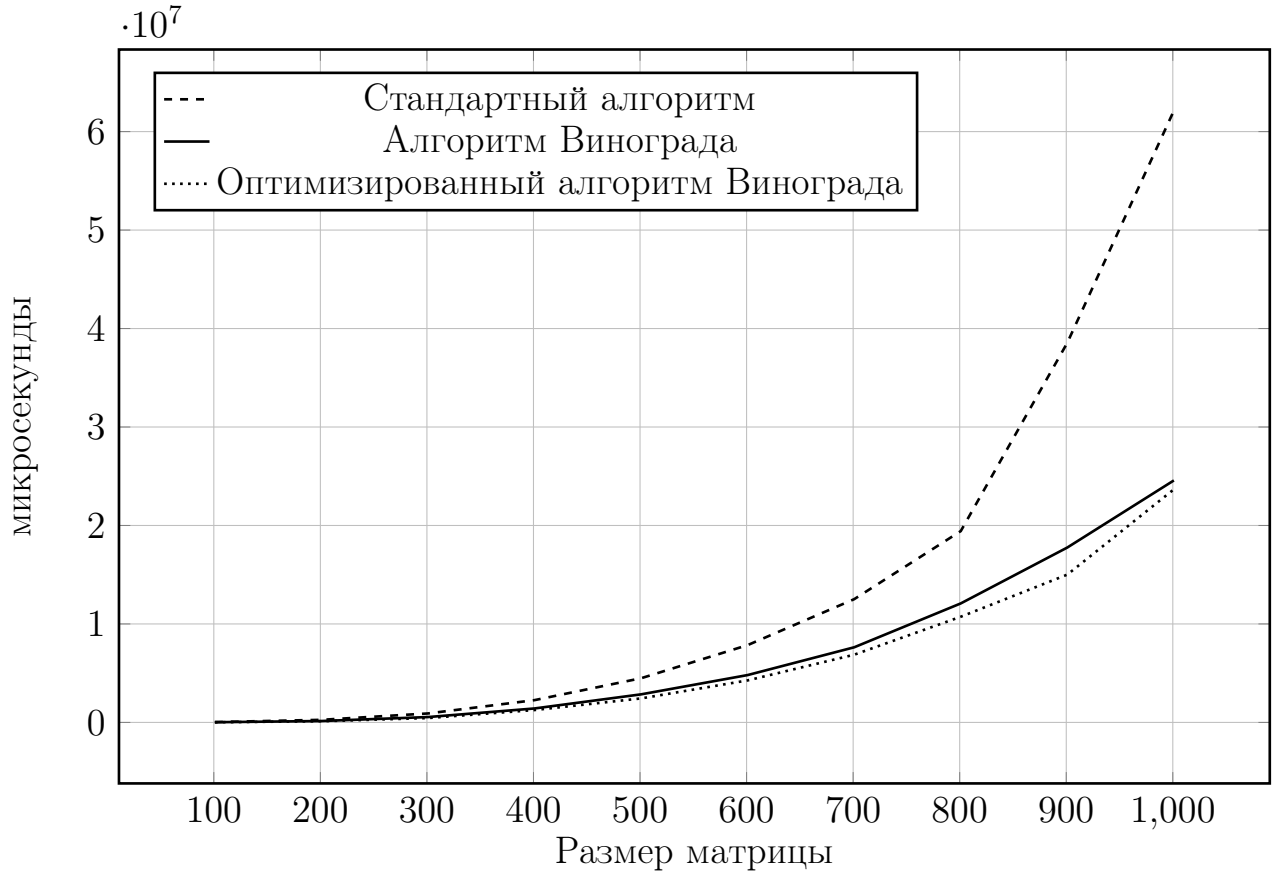

Рисунок 4.2 — Нечётные размеры

4.2 Вывод

В результате эксперементов по замеру времени подтвердились тезисы, сделанные в конструкторском разделе: модифицированный алгоритм Винограда работает быстрее алгоритма Винограда, а алгоритм Винограда - быстрее стандартного алгоритма умножения матриц.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было проведено сравнение стандартного алгоритма умножения матриц с алгоритмом Винограда и его оптимизированной версией.

Было проведено сравнение времени работы алгоритмов в случаях чётных и нечётных размеров квадратной матрицы и сделаны следующие выводы:

а) алгоритм Винограда работает быстрее стандартного алгоритма умножения матриц;

б) модифицированный алгоритм Винограда работает быстрее, чем алгоритм Винограда.

Все цели, поставленные на эту работу, были выполнены:

а) реализованы алгоритмы умножения матриц;

б) проведен сравнительный анализ стандартного алгоритма и улучшенных алгоритмов по затрачиваемым ресурсам;

в) проведено экспериментальное подтверждение различий во временной эффективности сравниваемых алгоритмов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Беллман Р. Введение в теорию матриц [Текст] - Москва: Мир, 1969

2. Корн Г. Алгебра матриц и матричное исчисление [Текст]

3. Володин Ф. Основные концепции библиотеки chrono (C++) [Электронный ресурс]. URL: https://habr.com/ru/post/324984/