Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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»

Лабораторная работа № 1
по дисциплине «Анализ алгоритмов»

Тема Умножение матриц

Студент Ильченко Е. А.

Группа ИУ7-54Б

Преподаватели Волкова Л.Л.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является исследование алгоритмов умножения матриц. В рамках исследования рассматриваются классический алгоритм умножения, алгоритм Винограда и его оптимизированная версия.

Для достижения поставленной цели необходимо решить следующие задачи:

— дать описание классического алгоритма умножения матриц и алгоритма Винограда;

— разработать оптимизированный вариант алгоритма Винограда;

— провести оценку трудоёмкости рассматриваемых алгоритмов;

— реализовать программное обеспечение с поддержкой двух режимов работы: выполнения единичного умножения матриц и массированный замер времени работы реализаций алгоритмов;

— выполнить замеры времени выполнения реализованных алгоритмов;

— провести сравнительный анализ алгоритмов на основе полученных результатов измерений.

1 Аналитическая часть

1.1 Понятие матрицы

Матрица – таблица чисел a_{ik}, состоящая из m строк и n столбцов, вида

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \tag{1.1}$$

a_{ij} – *элемент матрицы* в i строке и j столбце.

Операции над матрицами:

1) **сложение матриц:** Матрицы одинакового размера складываются поэлементно [1];

2) **умножение матрицы на число:** Каждый элемент матрицы умножается на скаляр [1];

3) **умножение матриц:** Произведение матриц A размером $m \times n$ и B размером $n \times k$ определяется как матрица C размером $m \times k$. В общем случае умножение $AB \neq BA$ (не коммутативно) [1].

Виды матриц:

1) **квадратная матрица:** матрица, у которой количество строк равно количеству столбцов ($m = n$) [1]

2) **единичная матрица:** квадратная матрица E, у которой элементы на главной диагонали равны 1, а остальные равны 0 [1]:

$$e_{ij} = \begin{cases} 1, & \text{если } i = j \\ 0, & \text{если } i \neq j \end{cases} \tag{1.2}$$

3) **диагональная матрица:** квадратная матрица, у которой все элементы вне главной диагонали равны нулю [1]:

$$d_{ij} = \begin{cases} d_{ii}, & \text{если } i = j \\ 0, & \text{если } i \neq j \end{cases} \tag{1.3}$$

4) **нулевая матрица:** матрица, все элементы которой равны нулю [1]:

$$0_{ij} = 0 \text{ для всех } i, j \tag{1.4}$$

1.2 Классический алгоритм умножения матриц

Даны две матрицы:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1t} \\ b_{21} & b_{22} & \dots & b_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mt} \end{pmatrix} \tag{1.5}$$

Результатом их произведения является матрица:

$$C = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1t} \\ c_{21} & c_{22} & \dots & c_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nt} \end{pmatrix} \tag{1.6}$$

При умножении матрицы $A_{n\times m}$ на матрицу $B_{m\times t}$ получается матрица $C_{n\times t}$, каждый элемент которой вычисляется по формуле [2]:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj} \tag{1.7}$$

Временная сложность классического алгоритма умножения матриц составляет $O(n^3)$ [2].

1.3 Алгоритм Винограда умножения матриц

Алгоритм Винограда основан на оптимизации вычисления скалярного произведения векторов [3]. Основная идея заключается в предварительной обработке данных, которая позволяет сократить количество операций умножения.

Даны два вектора $U = (u_1, u_2, u_3, u_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение можно записать как:

$$U \cdot W = u_1w_1 + u_2w_2 + u_3w_3 + u_4w_4 \tag{1.8}$$

Это выражение можно преобразовать к виду [3]:

$$U \cdot W = (u_1 + w_2)(u_2 + w_1) + (u_3 + w_4)(u_4 + w_3) - u_1u_2 - u_3u_4 - w_1w_2 - w_3w_4 \tag{1.9}$$

Хотя количество операций увеличилось, второе и третье слагаемые можно вычислить заранее для каждой строки и столбца исходных матриц. Это позволяет выполнять только два умножения и несколько сложений для каждого элемента результирующей матрицы.

Для матриц $A_{n\times m}$ и $B_{m\times k}$ алгоритм работает следующим образом:

1) вычисляются вспомогательные векторы:

$$MulH_i = \sum_{j=1}^{m/2} a_{i,2j-1} \cdot a_{i,2j} \tag{1.10}$$

$$MulV_j = \sum_{i=1}^{m/2} b_{2i-1,j} \cdot b_{2i,j} \tag{1.11}$$

2) элементы результирующей матрицы вычисляются по формуле:

$$c_{ij} = -MulH_i - MulV_j + \sum_{k=1}^{m/2} (a_{i,2k-1} + b_{2k,j}) \cdot (a_{i,2k} + b_{2k-1,j}) \tag{1.12}$$

3) если m нечётное, добавляется дополнительный член:

$$c_{ij} += a_{i,m} \cdot b_{m,j} \tag{1.13}$$

Алгоритм Винограда сокращает количество операций умножения примерно в 2 раза по сравнению с классическим алгоритмом [3].

1.4 Оптимизированный алгоритм Винограда умножения матриц

Оптимизированная версия алгоритма Винограда по условию лабораторной работы включает:

1) предварительное вычисление значений ($N - 1$, $N/2$ и др.);

2) операцию умножения на 2 программно реализовывать как побитовый сдвиг влево на 1;

Оптимизации позволяют дополнительно ускорить выполнение алгоритма за счёт уменьшения количества арифметических операций и оптимизации работы с памятью.

Вывод

В аналитической части рассмотрены три алгоритма умножения матриц:

— классический алгоритм – базовый подход с временной сложностью $O(n^3)$;

— алгоритм Винограда – оптимизация с предварительным вычислением вспомогательных векторов, сокращающая количество умножений;

— оптимизированный алгоритм Винограда – улучшенная версия с битовыми операциями и кэшированием.

2 Конструкторская часть

2.1 Требования к реализации

К программе предъявлены ряд функциональных требований: входные данные – размеры матрицы, целые числа и матрица, двумерный массив целых чисел, выходные данные – матрица.

К программе предъявлены ряд требований:

— наличие интерфейса для выбора действий;

— наличие стандартной реализации матрицы с динамическим выделением памяти;

— наличие функциональности замера процессорного времени алгоритмов умножения матриц;

— реализация трёх алгоритмов умножения матриц: классического, Винограда и оптимизированного Винограда.

Программа работает в двух режимах: одиночное выполнение умножения введённых пользователем матриц, а также массированный замер времени выполнения реализаций алгоритмов умножения матриц.

При первом режиме на вход поступают две целочисленные матрицы, на выход – результирующая матрица. Она пустая, если перемножение матриц невозможно из-за их размеров.

При втором режиме работы на вход поступают начальный размер исходных матриц, конечный размер, шаг и число итераций, на выход – результат замера времени в виде таблицы.

2.2 Разработка алгоритмов

Раздел содержит блок-схемы, описывающие следующие алгоритмы: классический алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

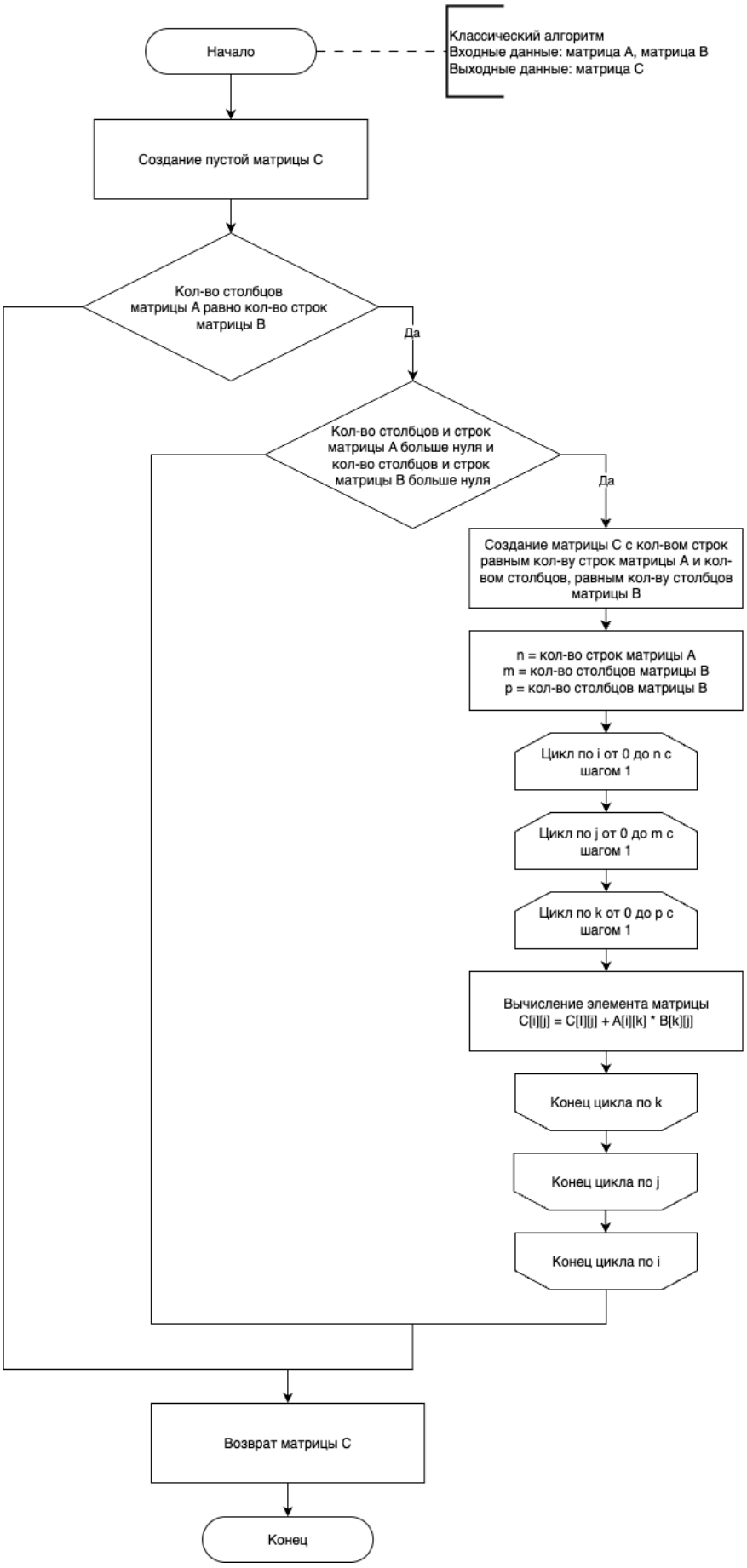

Рисунок 2.1 — Схема классического алгоритма

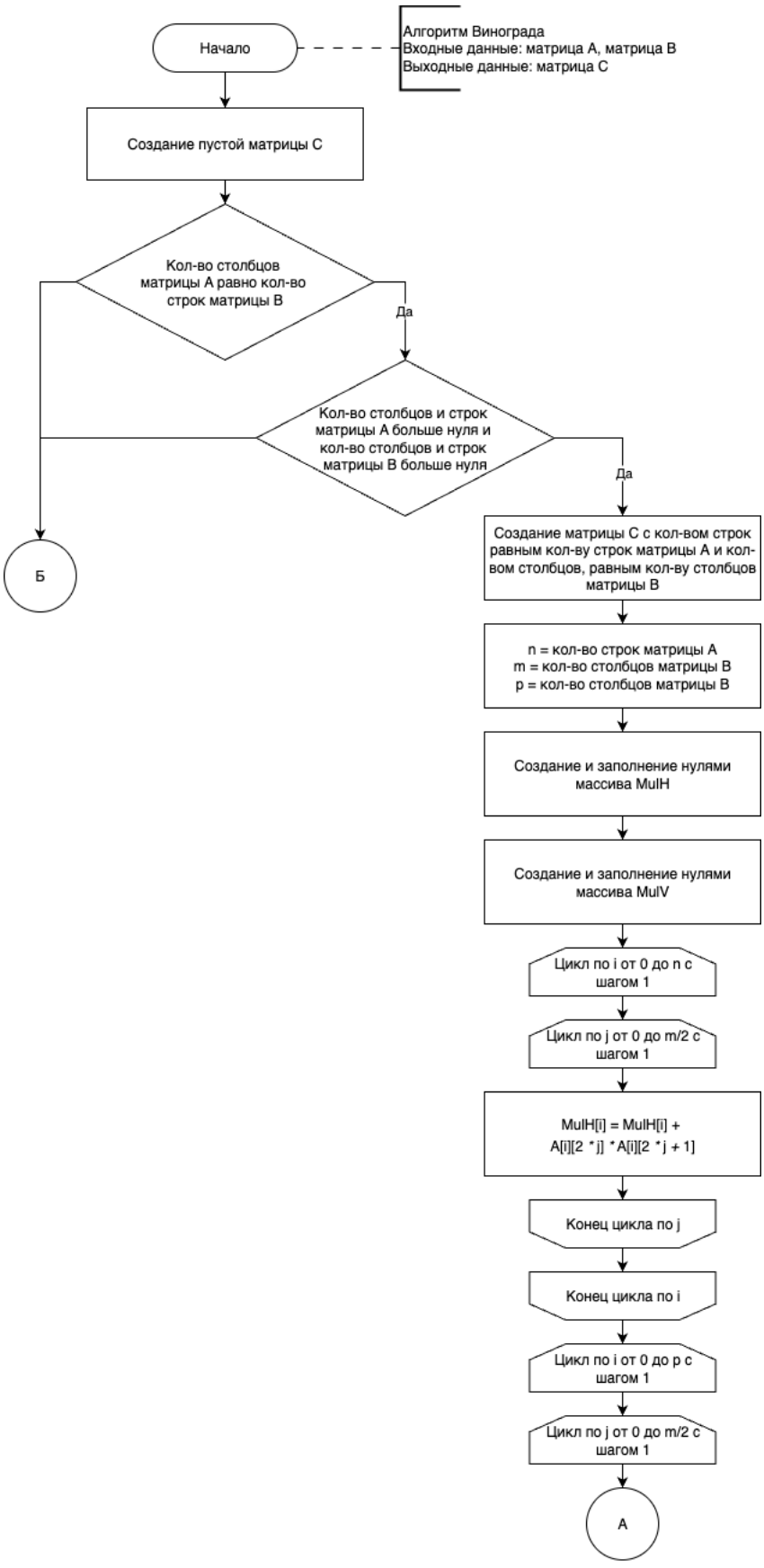

Рисунок 2.2 — Схема алгоритма Винограда (часть 1)

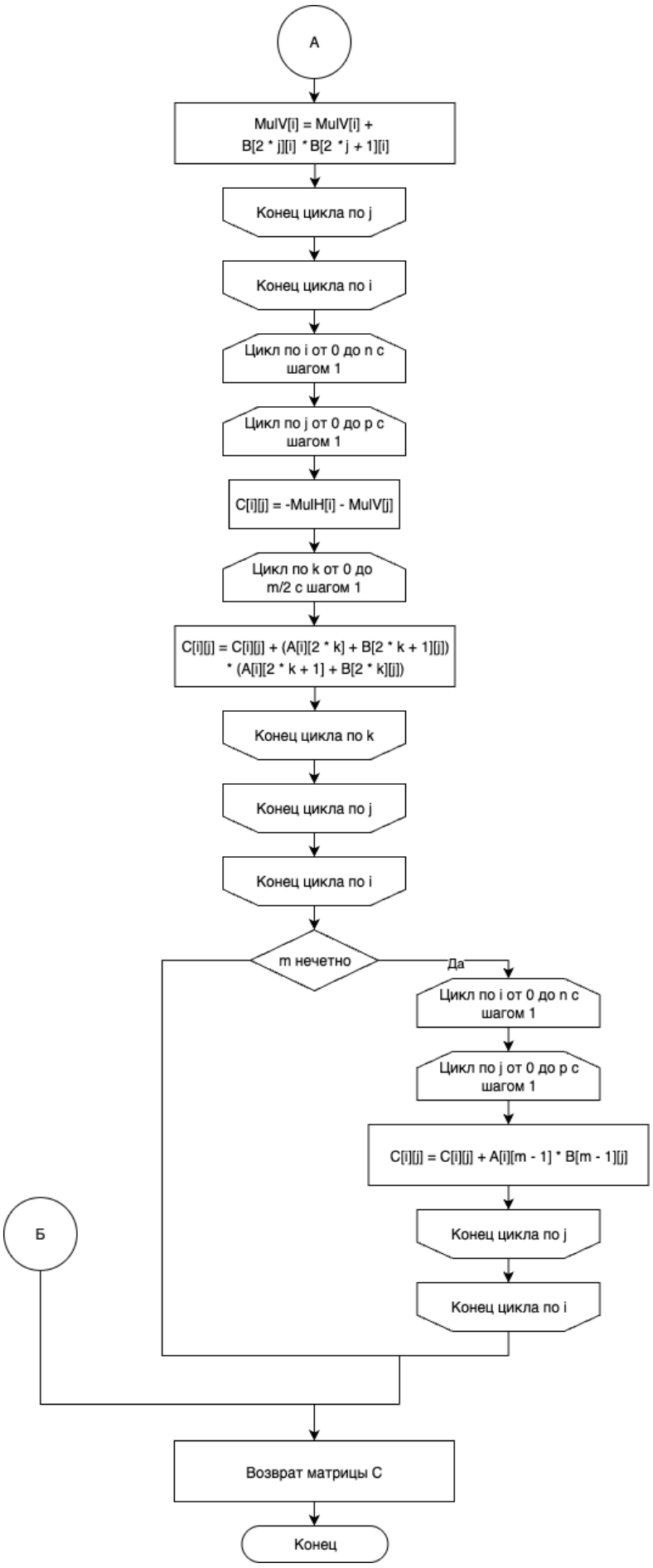

Рисунок 2.3 — Схема алгоритма Винограда (часть 2)

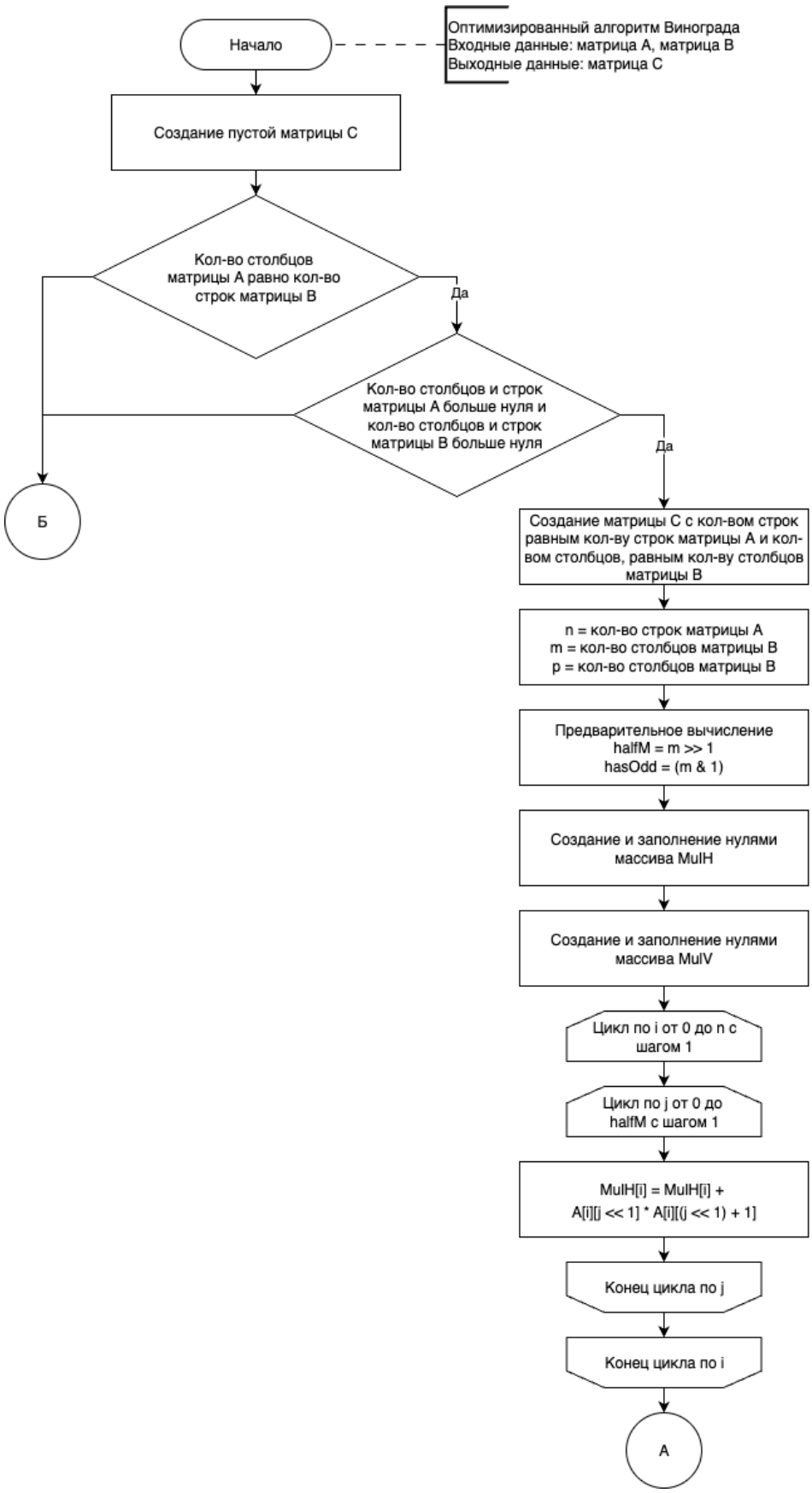

Рисунок 2.4 — Схема оптимизированного алгоритма Винограда (часть 1)

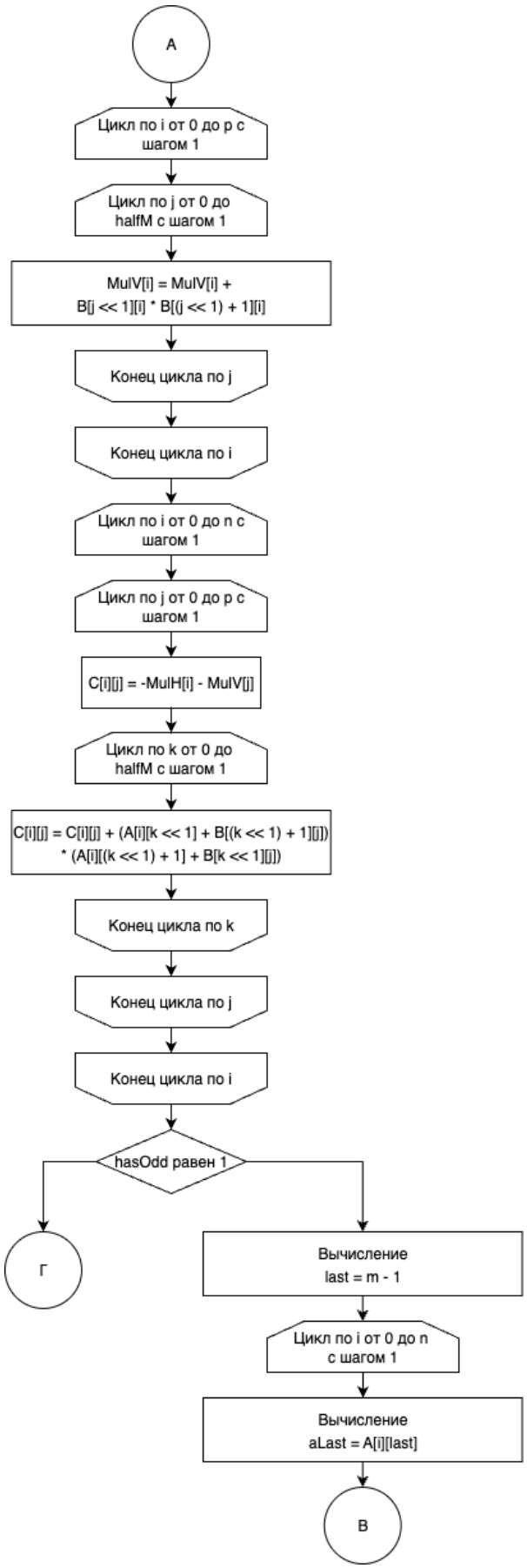

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда (часть 2)

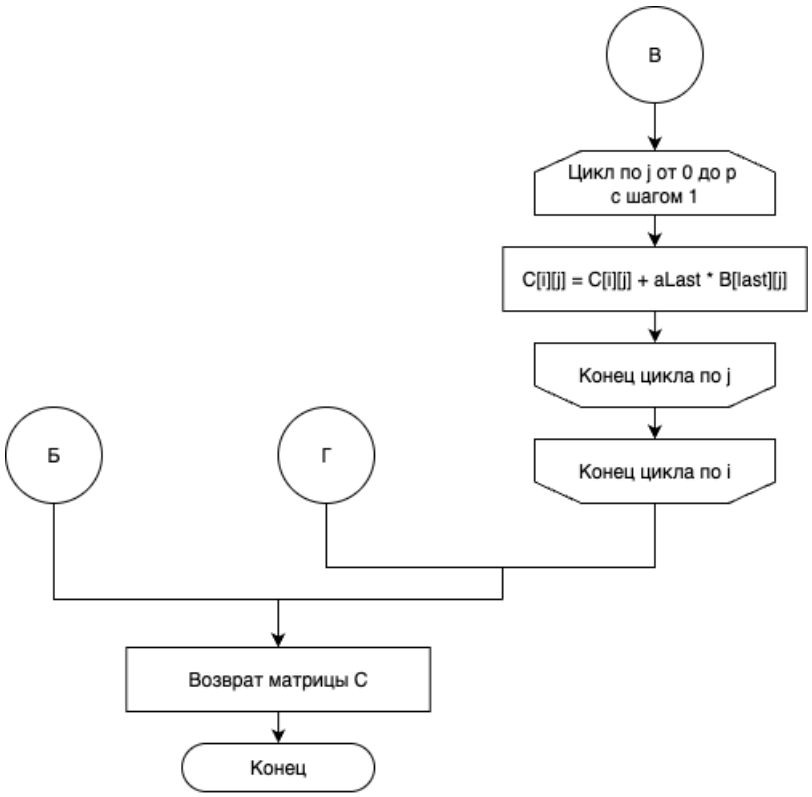

Рисунок 2.6 — Схема оптимизированного алгоритма Винограда (часть 3)

2.3 Модель вычислений

Для проведения оценки трудоёмкости была введена модель вычислений.

1) трудоёмкость базовых операций:
   — равна 1 для: =, !=, ==, <, >, <=, >=, |, ||, &, &&, |=, &=, +, ++, -, –, +=, -=, [], $<<$, $>>$
   — равна 2 для: *, /, *=, /=, %, %=
2) трудоёмкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_1, f_2), & \text{л. с.} \\ max(f_1, f_2), & \text{х. с.} \end{cases}$$

3) трудоёмкость цикла:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + M_{\text{итераций}} \cdot (f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}})$$

2.4 Трудоёмкость алгоритмов

2.4.1 Классический алгоритм

Для классического алгоритма трудоёмкость будет складываться из:

— цикла по $i \in [0, N)$:

$$f_i = 1 + 1 + N \cdot (f_{\text{тела}} + 1 + 1) = 2 + N \cdot (f_{\text{тела}} + 2)$$

— цикла по $j \in [0, M)$:

$$f_j = 1 + 1 + M \cdot (f_{\text{тела}} + 1 + 1) = 2 + M \cdot (f_{\text{тела}} + 2)$$

— цикла по $k \in [0, P)$:

$$f_k = 1 + 1 + P \cdot (f_{\text{тела}} + 1 + 1) = 2 + P \cdot (f_{\text{тела}} + 2)$$

— операции умножения и сложения элементов:

$$f_{mult} = 8 + 2 + 2 = 12$$

Таким образом:

$$f_{standard} = 2+N\cdot(2+M\cdot(2+P\cdot(12+2)+2)+2) = 14NMP+4NM+4N+2 \approx 14NMP = O(N^3)$$

где N — количество строк первой матрицы, M — количество столбцов второй матрицы, P — количество столбцов первой матрицы.

2.4.2 Алгоритм Винограда

Для алгоритма Винограда трудоёмкость будет складываться из:

— расчёта MulH:

$$f_{MulH} = 2 + N \cdot (4 + M/2 \cdot ((6 + 3 + 6) + 4) + 2) = 9.5NM + 6N + 2$$

— расчёта MulV:

$$f_{MulV} = 2 + P \cdot (4 + M/2 \cdot ((6 + 3 + 6) + 4) + 2) = 9.5PM + 6P + 2$$

— расчёта элементов результирующей матрицы C:

$$f_c = 2 + N \cdot (2 + P \cdot (4 + M/2 \cdot (12 + 6 + 10 + 4) + 2 + 7) + 2) = 16NPM + 13NP + 4N + 2$$

— проверки размера на нечётность:

$$f_{odd} = 3 + \begin{cases} 0, & \text{л. с.} \\ 2 + N \cdot (2 + P \cdot (8 + 4 + 2 + 2) + 2), & \text{х. с.} \end{cases} = 3 + \begin{cases} 0, & \text{л. с.} \\ 16NP + 4N + 2, & \text{х. с.} \end{cases}$$

Таким образом:

$$f_{winograd} = 9.5NM + 6N + 2 + 9.5PM + 6P + 2 + 16NPM + 13NP + 4N + 2 + f_{odd}$$

$$f_{winograd} = 16NPM + 13NP + 9.5NM + 9.5PM + 10N + 6P + 9 + \begin{cases} 0, & \text{л. с.} \\ 16NP + 4N + 2, & \text{х. с.} \end{cases}$$

где N — количество строк первой матрицы, M — количество столбцов первой матрицы, P — количество столбцов второй матрицы.

Для лучшего случая (чётный размер матриц) имеем:

$$f_{best} = 16NPM + 13NP + 9.5NM + 9.5PM + 10N + 6P + 9 \approx 16NPM = O(N^3)$$

Для худшего случая (нечётный размер матриц) имеем:

$$f_{worst} = 16NPM + 13NP + 9.5NM + 9.5PM + 10N + 6P + 9 + 16NP + 4N + 2 \approx 16NPM = O(N^3)$$

2.4.3 Оптимизированный алгоритм Винограда

Для оптимизированного алгоритма Винограда трудоёмкость будет складываться из:
— предварительного вычисления констант:

$$f_{const} = 2 + 2$$

— расчёта MulH (с использованием битовых операций):

$$f_{MulH} = 2 + N \cdot (2 + M/2 \cdot (5 + 4 + 2 + 2) + 2) = 6.5MN + 4N + 2$$

— расчёта MulV (с использованием битовых операций):

$$f_{MulV} = 2 + P \cdot (2 + M/2 \cdot (5 + 4 + 2 + 2) + 2) = 6.5MP + 4P + 2$$

— расчёта элементов результирующей матрицы C (с использованием битовых операций):

$$f_c = 2+N\cdot(2+P\cdot(2+M/2\cdot(10+9+2+2)+4+3+2)+2) = 11.5NPM+11PN+4N+2$$

— проверки размера на нечётность и обработки нечётного случая (с оптимизацией доступа к памяти):

$$f_{odd} = 1+\begin{cases} 0, & \text{л. с.} \\ 2 + N \cdot (3 + 2 + P \cdot (4 + 1 + 2 + 2) + 2), & \text{х. с.} \end{cases} = 1+\begin{cases} 0, & \text{л. с.} \\ 9NP + 7N + 2, & \text{х. с.} \end{cases}$$

Таким образом:

$$f_{optimized\_winograd} = 2+2+6.5MN+4N+2+6.5MP+4P+2+11.5NMP+11PN+4N+2+f_{odd}$$

$$f_{optimized\_winograd} = 11.5NMP+6.5MN+6.5MP+11PN+4P+8N+11+\begin{cases} 0, & \text{л. с.} \\ 9NP + 7N + 2, & \text{х. с.} \end{cases}$$

где N — количество строк первой матрицы, M — количество столбцов первой матрицы, P — количество столбцов второй матрицы.

Для лучшего случая (чётный размер матриц) имеем:

$$f_{best} = 11.5NMP + 6.5MN + 6.5MP + 20PN + 4P + 15N + 13 \approx 11.5NPM = O(N^3)$$

Для худшего случая (нечётный размер матриц) имеем:

$$f_{worst} = 11.5NMP + 6.5MN + 6.5MP + 11PN + 4P + 8N + 11 \approx 16NPM = O(N^3)$$

Вывод

В данном разделе представлены блок-схемы и выполнена оценка трудоёмкости трёх алгоритмов умножения матриц: стандартного, Винограда и оптимизированного Винограда. Результаты расчётов, основанные на введённой модели вычислений, расположили алгоритмы по возрастанию вычислительной сложности следующим образом: оптимизированный Виноград, стандартный алгоритм, алгоритм Винограда.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов умножения матриц был выбран язык программирования C, поскольку он соответствует требованиям лабораторной работы. Измерение времени выполнения алгоритмов осуществлялось с использованием функции `clock_gettime()` из библиотеки `time.h`, которая обеспечивает высокую точность измерений. Разработка программного обеспечения проводилась в интегрированной среде разработки CLion.

3.2 Реализация алгоритмов

Были реализованы классический алгоритм умножения матриц (листинг 3.1), алгоритм Винограда (листинг 3.2) и оптимизированный алгоритм Винограда (листинг 3.3).

Листинг 3.1 — Классический алгоритм

```
Matrix standard_multiplication(const Matrix *A, const Matrix *B) {
  Matrix C = {0};

  if (A->cols != B->rows) {
    printf("Error: matrix dimensions mismatch (%d != %d)\n", A->cols, B
       ->rows);
    return C;
  }

  if (A->rows <= 0 || A->cols <= 0 || B->rows <= 0 || B->cols <= 0) {
    printf("Error: matrix dimensions must be positive\n");
    return C;
  }

  create_matrix(&C, A->rows, B->cols);

  int n = A->rows;
  int m = B->cols;
  int p = A->cols;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      for (int k = 0; k < p; ++k) {
        C.data[i][j] += A->data[i][k] * B->data[k][j];
      }
    }
  }
```

```
  return C;
}
```

Листинг 3.2 — Алгоритм Винограда

```
Matrix vinograd_multiplication(const Matrix *A, const Matrix *B) {
  Matrix C = {0};

  if (A->cols != B->rows) {
    printf("Error: matrix dimensions mismatch (%d != %d)\n", A->cols, B
       ->rows);
    return C;
  }

  if (A->rows <= 0 || A->cols <= 0 || B->rows <= 0 || B->cols <= 0) {
    printf("Error: matrix dimensions must be positive\n");
    return C;
  }

  create_matrix(&C, A->rows, B->cols);

  int n = A->rows;
  int m = A->cols;
  int p = B->cols;

  int *MulH = (int*)calloc((size_t)n, sizeof(int));
  int *MulV = (int*)calloc((size_t)p, sizeof(int));

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

  for (int i = 0; i < p; ++i) {
    for (int j = 0; j < m / 2; ++j) {
      MulV[i] += B->data[2 * j][i] * B->data[2 * j + 1][i];
    }
  }

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < p; ++j) {
```

```
      C.data[i][j] = -MulH[i] - MulV[j];
      for (int k = 0; k < m / 2; ++k) {
        C.data[i][j] += (A->data[i][2 * k] + B->data[2 * k + 1][j]) * (
            A->data[i][2 * k + 1] + B->data[2 * k][j]);
      }
    }
  }

  if (m % 2 == 1) {
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < p; ++j) {
        C.data[i][j] += A->data[i][m - 1] * B->data[m - 1][j];
      }
    }
  }

  free(MulH);
  free(MulV);

  return C;
}
```

Листинг 3.3 — Оптимизированный алгоритм Винограда

```
Matrix optimized_vinograd_multiplication(const Matrix *A, const Matrix
   *B) {
  Matrix C = {0};

  if (A->cols != B->rows) {
    printf("Error: matrix dimensions mismatch (%d != %d)\n", A->cols, B
       ->rows);
    return C;
  }

  if (A->rows <= 0 || A->cols <= 0 || B->rows <= 0 || B->cols <= 0) {
    printf("Error: matrix dimensions must be positive\n");
    return C;
  }

  create_matrix(&C, A->rows, B->cols);

  const int n = A->rows;
  const int m = A->cols;
```

```
const int p = B->cols;

const int halfM = m >> 1;
const int hasOdd = (m & 1);

int *MulH = (int*)calloc((size_t)n, sizeof(int));
int *MulV = (int*)calloc((size_t)p, sizeof(int));

for (int i = 0; i < n; ++i) {
  for (int j = 0; j < halfM; ++j) {
    MulH[i] += A->data[i][j << 1] * A->data[i][(j << 1) + 1];
  }
}

for (int i = 0; i < p; ++i) {
  for (int j = 0; j < halfM; ++j) {
    MulV[i] += B->data[j << 1][i] * B->data[(j << 1) + 1][i];
  }
}

for (int i = 0; i < n; ++i) {
  for (int j = 0; j < p; ++j) {
    C.data[i][j] = -MulH[i] - MulV[j];
    for (int k = 0; k < halfM; ++k) {
      C.data[i][j] += (A->data[i][k << 1] + B->data[(k << 1) + 1][j])
          * (A->data[i][(k << 1) + 1] + B->data[k << 1][j]);
    }
  }
}

if (hasOdd) {
  const int last = m - 1;
  for (int i = 0; i < n; ++i) {
    const int aLast = A->data[i][last];
    for (int j = 0; j < p; ++j) {
      C.data[i][j] += aLast * B->data[last][j];
    }
  }
}

free(MulH);
```

```
  free(MulV);

  return C;
}
```

3.3 Функциональные тесты

В данном разделе представлены функциональные тесты для разработанных алгоритмов умножения матриц: классического алгоритма (таблица 3.1), алгоритма Винограда (таблица 3.2) и оптимизированного алгоритма Винограда (таблица 3.3). Тесты включают проверку корректности вычислений для различных размеров матриц и обработку ошибочных входных данных.

Таблица 3.1 — Функциональные тесты для классического алгоритма

Матрица A	**Матрица B**	**Ожидаемый результат**	**Полученный результат**
$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}7\end{pmatrix}$	$\begin{pmatrix}21\end{pmatrix}$	$\begin{pmatrix}21\end{pmatrix}$
$\begin{pmatrix}2 & 4 & 6\end{pmatrix}$	$\begin{pmatrix}3 & 5 & 7\end{pmatrix}$	Error message	Error message
$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	Error message	Error message
$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$	$\begin{pmatrix}2 & 2\\2 & 2\end{pmatrix}$	$\begin{pmatrix}10 & 10\\18 & 18\end{pmatrix}$	$\begin{pmatrix}10 & 10\\18 & 18\end{pmatrix}$
$\begin{pmatrix}2 & 3 & 4\\5 & 6 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2\\3 & 4\\5 & 6\end{pmatrix}$	$\begin{pmatrix}31 & 40\\58 & 76\end{pmatrix}$	$\begin{pmatrix}31 & 40\\58 & 76\end{pmatrix}$
$\begin{pmatrix}4\\6\\8\end{pmatrix}$	$\begin{pmatrix}3 & 5 & 2\end{pmatrix}$	$\begin{pmatrix}12 & 20 & 8\\18 & 30 & 12\\24 & 40 & 16\end{pmatrix}$	$\begin{pmatrix}12 & 20 & 8\\18 & 30 & 12\\24 & 40 & 16\end{pmatrix}$

Таблица 3.2 — Функциональные тесты для алгоритма Винограда

Матрица A	**Матрица B**	**Ожидаемый результат**	**Полученный результат**
$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}7\end{pmatrix}$	$\begin{pmatrix}21\end{pmatrix}$	$\begin{pmatrix}21\end{pmatrix}$
$\begin{pmatrix}2 & 4 & 6\end{pmatrix}$	$\begin{pmatrix}3 & 5 & 7\end{pmatrix}$	Error message	Error message
$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	Error message	Error message
$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$	$\begin{pmatrix}2 & 2\\2 & 2\end{pmatrix}$	$\begin{pmatrix}10 & 10\\18 & 18\end{pmatrix}$	$\begin{pmatrix}10 & 10\\18 & 18\end{pmatrix}$
$\begin{pmatrix}2 & 3 & 4\\5 & 6 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2\\3 & 4\\5 & 6\end{pmatrix}$	$\begin{pmatrix}31 & 40\\58 & 76\end{pmatrix}$	$\begin{pmatrix}31 & 40\\58 & 76\end{pmatrix}$
$\begin{pmatrix}4\\6\\8\end{pmatrix}$	$\begin{pmatrix}3 & 5 & 2\end{pmatrix}$	$\begin{pmatrix}12 & 20 & 8\\18 & 30 & 12\\24 & 40 & 16\end{pmatrix}$	$\begin{pmatrix}12 & 20 & 8\\18 & 30 & 12\\24 & 40 & 16\end{pmatrix}$

Таблица 3.3 — Функциональные тесты для оптимизированного алгоритма Винограда

Матрица A	**Матрица B**	**Ожидаемый результат**	**Полученный результат**
$\begin{pmatrix}3\end{pmatrix}$	$\begin{pmatrix}7\end{pmatrix}$	$\begin{pmatrix}21\end{pmatrix}$	$\begin{pmatrix}21\end{pmatrix}$
$\begin{pmatrix}2 & 4 & 6\end{pmatrix}$	$\begin{pmatrix}3 & 5 & 7\end{pmatrix}$	Error message	Error message
$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$	$\begin{pmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{pmatrix}$	Error message	Error message
$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix}$	$\begin{pmatrix}2 & 2\\2 & 2\end{pmatrix}$	$\begin{pmatrix}10 & 10\\18 & 18\end{pmatrix}$	$\begin{pmatrix}10 & 10\\18 & 18\end{pmatrix}$
$\begin{pmatrix}2 & 3 & 4\\5 & 6 & 7\end{pmatrix}$	$\begin{pmatrix}1 & 2\\3 & 4\\5 & 6\end{pmatrix}$	$\begin{pmatrix}31 & 40\\58 & 76\end{pmatrix}$	$\begin{pmatrix}31 & 40\\58 & 76\end{pmatrix}$
$\begin{pmatrix}4\\6\\8\end{pmatrix}$	$\begin{pmatrix}3 & 5 & 2\end{pmatrix}$	$\begin{pmatrix}12 & 20 & 8\\18 & 30 & 12\\24 & 40 & 16\end{pmatrix}$	$\begin{pmatrix}12 & 20 & 8\\18 & 30 & 12\\24 & 40 & 16\end{pmatrix}$

Все функциональные тесты были успешно пройдены.

Вывод

В данном разделе были реализованы три алгоритма умножения матриц на языке программирования C: классический алгоритм, алгоритм Винограда и оптимизированный алгоритм Винограда. Все алгоритмы включают проверку корректности входных данных и обработку ошибок.

Проведённые функциональные тесты подтвердили корректность работы всех реализованных алгоритмов.

4 Исследовательская часть

4.1 Характеристики ЭВМ

Замеры проводились на устройстве со следующими характеристиками:

— процессор Apple M4 Pro;

— оперативная память 24 Гб;

— операционная система macOS Sequoia 15.6.1.

Замеры времени проводились, когда ноутбук был загружен только системными приложениями.

4.2 Результаты замера времени

В данном разделе представлены результаты измерения времени выполнения алгоритмов умножения матриц. Замеры проводились для квадратных матриц размером от 100×100 до 800×800 с шагом 100, а также для матриц размером от 101×101 до 801×801 с шагом 100. Каждый замер выполнялся 100 раз для получения более точных результатов.

Таблица 4.1 — Результаты замера времени для матриц размером 100-800

N	Классический (мкс)	Виноград (мкс)	Опт. Виноград (мкс)
100	1713	1362	1357
200	12684	10152	10159
300	46280	37632	37420
400	109879	90012	89321
500	222837	179346	168155
600	385627	314130	292214
700	618165	503980	481834
800	938280	766001	732577

На основе таблицы 4.1 был построен график зависимости времени от размера матриц (рисунок 4.1).

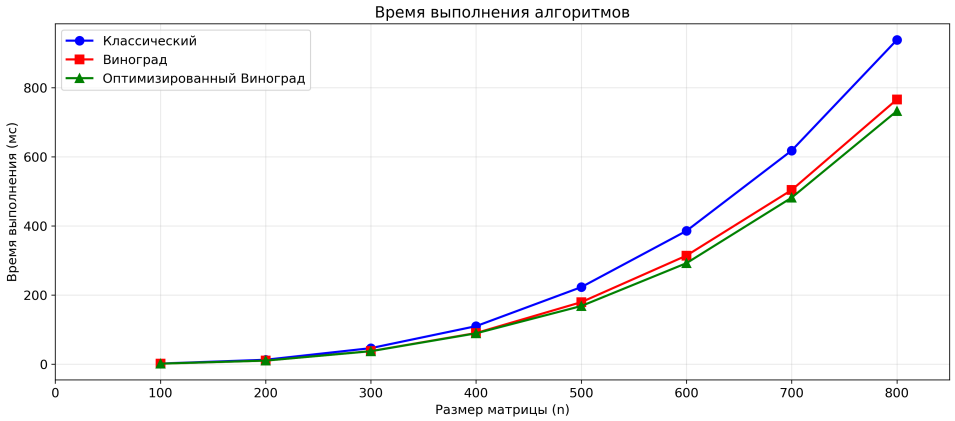

Рисунок 4.1 — Сравнение времени на чётных размерах

Таблица 4.2 — Результаты замера времени для матриц размером 101-801

N	Классический (мкс)	Виноград (мкс)	Опт. Виноград (мкс)
101	1774	1437	1423
201	13426	10755	10730
301	48758	39697	39373
401	114676	94518	93645
501	229696	186372	175773
601	396448	322997	310996
701	630847	513647	501286
801	952198	775787	742479

На основе таблицы 4.2 был построен график зависимости времени от размера матриц (рисунок 4.2).

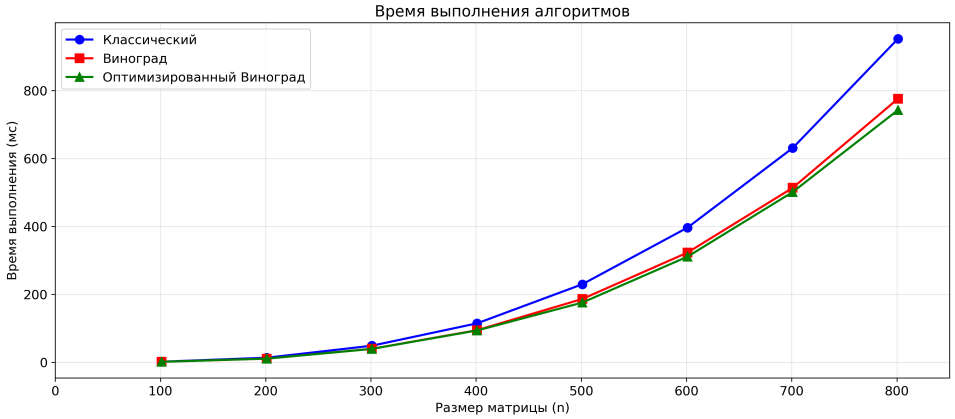

Рисунок 4.2 — Сравнение времени на нечётных размерах

Вывод

Согласно результатам исследования, алгоритм Винограда работает быстрее стандартного метода умножения матриц. Эта разница в скорости становится особенно заметной с увеличением размеров матриц. Наилучшая производительность достигается оптимизированной версией алгоритма Винограда, которая улучшена за счёт двух ключевых изменений: предварительный расчёт переменных и замена умножения на два на двоичный сдвиг целого числа.

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы была успешно достигнута основная цель: реализованы различные алгоритмы умножения матриц, проведён их сравнительный анализ и исследование.

Все поставленные задачи выполнены в полном объёме:

— проведено описание классического алгоритма умножения матриц и алгоритма Винограда;

— разработана оптимизированная версия алгоритма Винограда;

— выполнена оценка трудоёмкости представленных алгоритмов, которая показала, что оптимизированная версия алгоритма Винограда демонстрирует наименьшую трудоёмкость, затем идут классический алгоритм и алгоритма Винограда;

— создано программное обеспечение, поддерживающее два режима работы: одиночное выполнение операции умножения и массовый замер времени выполнения для всех реализованных алгоритмов;

— проведены замеры процессорного времени работы всех алгоритмов;

— осуществлён сравнительный анализ алгоритмов на основе полученных экспериментальных данных. По его результатам классический алгоритм уступает в скорости оптимизированным методам; алгоритм Винограда демонстрирует значительное ускорение по сравнению с классическим подходом за счёт уменьшения количества операций умножения; оптимизированная версия алгоритма Винограда обеспечивает наилучшие показатели производительности благодаря применению дополнительных оптимизаций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вербицкий О. В. Линейная алгебра. – М.: Юрайт, 2018. – 422 с.
2. Кормен Т. Х. и др. Алгоритмы: построение и анализ. – М.: Вильямс, 2009. – 1296 с.
3. Winograd S. A new algorithm for inner product // IEEE Transactions on Computers. – 1968. – Vol. C-17. – P. 693-694.
4. Кнут Д. И. Искусство программирования. Том 1. Основные алгоритмы. – М.: Вильямс, 1997. – 720 с.