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 Алгоритмы умножения матриц

Студент Жаринов М. А.

Группа ИУ7-52Б

Преподаватели Волкова Л. Л., Строганов Д. В.

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной лабораторной работы — исследовать алгоритмы умножения матриц: алгоритм Винограда, стандартный алгоритм и оптимизированный алгоритм Винограда. Для достижения данной цели необходимо выполнить задачи:

1) описать стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда;
2) ввести модель вычислений и выполнить в ней оценку трудоёмкости алгоритмов;
3) реализовать алгоритмы умножения матриц и провести их тестирование;
4) выполнить замеры процессорного времени выполнения реализаций алгоритмов умножения матриц при варьируемом линейном размере квадратных перемножаемых матриц;
5) провести сравнительный анализ алгоритмов по данным результатов замеров.

1 Аналитическая часть

1.1 Основные сведения о матрицах

В линейной алгебре матрицей называется совокупность элементов (чаще всего чисел), организованных в виде прямоугольной таблицы. Матрица, содержащая M строк и N столбцов, называется матрицей размера $M \times N$.

Элемент, расположенный на пересечении i-й строки и j-го столбца, обозначается как a_{ij}. Общий вид матрицы A представляется по формуле (1.1).

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & \dots & a_{MN} \end{pmatrix} \tag{1.1}$$

Одной из операций, производимой с матрицами, является умножение матриц. Умножение матрицы A размера $M \times P$ на матрицу B размера $E \times N$ возможно только в том случае, если количество столбцов первой матрицы совпадает с количеством строк второй, то есть $P = E$. Результирующая матрица C будет иметь размер $M \times N$.

1.2 Стандартный алгоритм умножения матриц

Стандартный, или классический, метод умножения матриц является прямой реализацией их математического определения [1]. Пусть даны матрица A размера $M \times P$ и матрица B размера $P \times N$. Их произведением будет матрица C размера $M \times N$.

Каждый элемент c_{ij} результирующей матрицы вычисляется как скалярное произведение i-й строки матрицы A и j-го столбца матрицы B. Формально это выражается следующей суммой:

$$c_{ij} = \sum_{k=1}^{P} a_{ik} \cdot b_{kj} \tag{1.2}$$

Этот подход прост в реализации, однако характеризуется высокой вычислительной сложностью, которая для квадратных матриц размера $N \times N$ составляет $O(N^3)$.

1.3 Алгоритм Винограда

Алгоритм, предложенный Шмуэлем Виноградом, представляет собой метод, основанный на реорганизации вычислений с целью сокращения числа дорогостоящих операций умножения за счёт увеличения числа менее затратных операций сложения [2].

Основная идея заключается в преобразовании скалярного произведения. Для этого вводятся предварительно вычисляемые величины:

— факторы строк (*row_factor*): для каждой i-й строки матрицы A;

— факторы столбцов (*col_factor*): для каждого j-го столбца матрицы B.

Эти величины рассчитываются как суммы произведений пар соседних элементов:

$$\text{row\_factor}_i = \sum_{k=1}^{\lfloor P/2 \rfloor} a_{i,2k-1} \cdot a_{i,2k} \tag{1.3}$$

$$\text{col\_factor}_j = \sum_{k=1}^{\lfloor P/2 \rfloor} b_{2k-1,j} \cdot b_{2k,j} \tag{1.4}$$

Элемент c_{ij} результирующей матрицы вычисляется по формуле (1.5), которая содержит вдвое меньше умножений в основном цикле:

$$c_{ij} = \sum_{k=1}^{\lfloor P/2 \rfloor} (a_{i,2k-1} + b_{2k,j}) \cdot (a_{i,2k} + b_{2k-1,j}) - \text{row\_factor}_i - \text{col\_factor}_j \tag{1.5}$$

В случае, если размерность P является нечетной, к результату, полученному по формуле (1.5), необходимо прибавить произведение последних элементов строки и столбца: $a_{i,P} \cdot b_{P,j}$. Предварительные вычисления факторов строк и столбцов выносятся за пределы основного тройного цикла, что и обеспечивает прирост производительности.

1.4 Оптимизированный алгоритм Винограда

При практической реализации базового алгоритма Винограда возможен ряд оптимизаций, направленных на снижение накладных расходов и более эффективное использование процессорного времени:

— кэширование инвариантов цикла. Значения, которые не изменяются в ходе итераций, например, половина размера внутренней размерности ($\lfloor P/2 \rfloor$), вычисляются один раз до начала цикла. Это исключает повторное выполнение одной и той же операции;

— замена умножения на побитовые сдвиги. Операции умножения на 2, используемые для индексации (например, *2 · k* и *2 · k + 1*), могут быть заменены на более быструю операцию побитового сдвига влево (*k << 1*).

Совокупность этих улучшений позволяет дополнительно ускорить выполнение алгоритма, особенно на больших размерах матриц.

Вывод

В аналитической части были рассмотрены три подхода к умножению матриц. Стандартный алгоритм является простым в реализации, но вычислительно затратным. Алгоритм Винограда предлагает более эффективный метод за счёт сокращения числа умножений, а его оптимизированная версия представляет собой практическую доработку, направленную на снижение накладных расходов при реализации.

2 Конструкторская часть

2.1 Требования к программе

К разрабатываемому программному обеспечению предъявляется следующий набор функциональных требований:

1) программное обеспечение должно предоставлять пользователю текстовый интерфейс для выбора режима работы;

2) необходимо реализовать поддержку двух основных режимов:

   — умножение двух матриц с ручным вводом данных;

   — массовое тестирование для замера производительности на матрицах различных размеров.

3) в ручном режиме программа должна запрашивать размеры и элементы двух исходных матриц, выполнять валидацию вводимых данных (размеры должны быть положительными числами) и проверять совместимость матриц для операции умножения;

4) в режиме массового тестирования должны выполняться замеры процессорного времени для каждого из реализованных алгоритмов. Результаты замеров следует выводить в наглядном табличном формате.

2.2 Разработка алгоритмов

Визуальное представление логики работы алгоритмов представлено в виде блок-схем. На рисунке 2.1 показана схема стандартного алгоритма. Схема алгоритма Винограда показана на рисунках 2.2 и 2.3. Схема оптимизированного алгоритма Винограда представлена на рисунках 2.4 и 2.5.

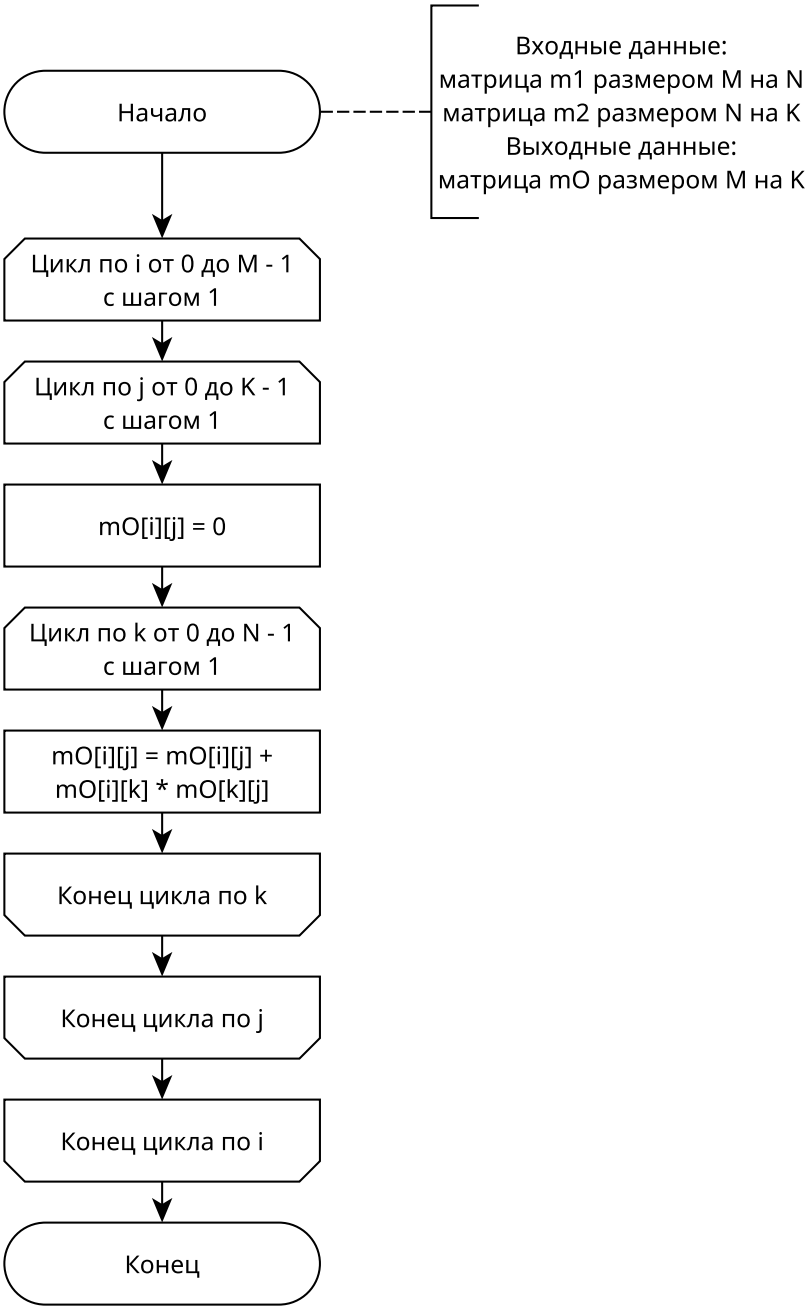

Рисунок 2.1 — Схема стандартного алгоритма умножения матриц

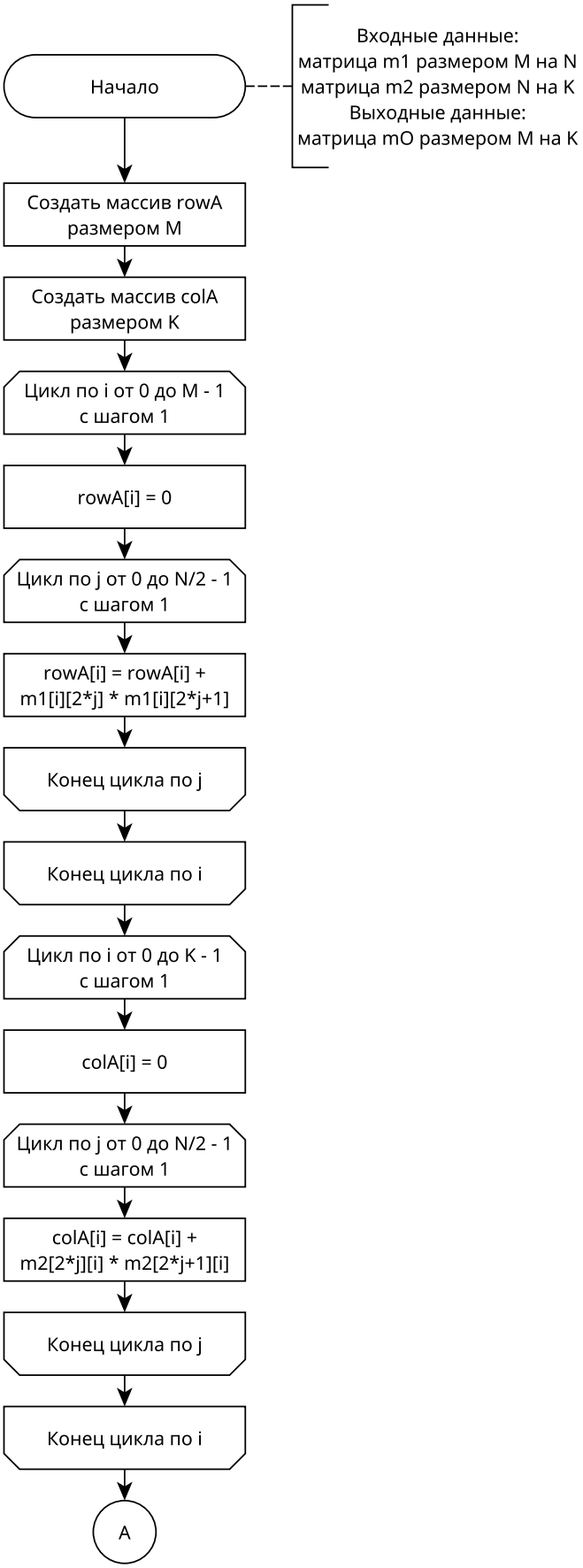

Рисунок 2.2 — Схема алгоритма Винограда (часть 1)

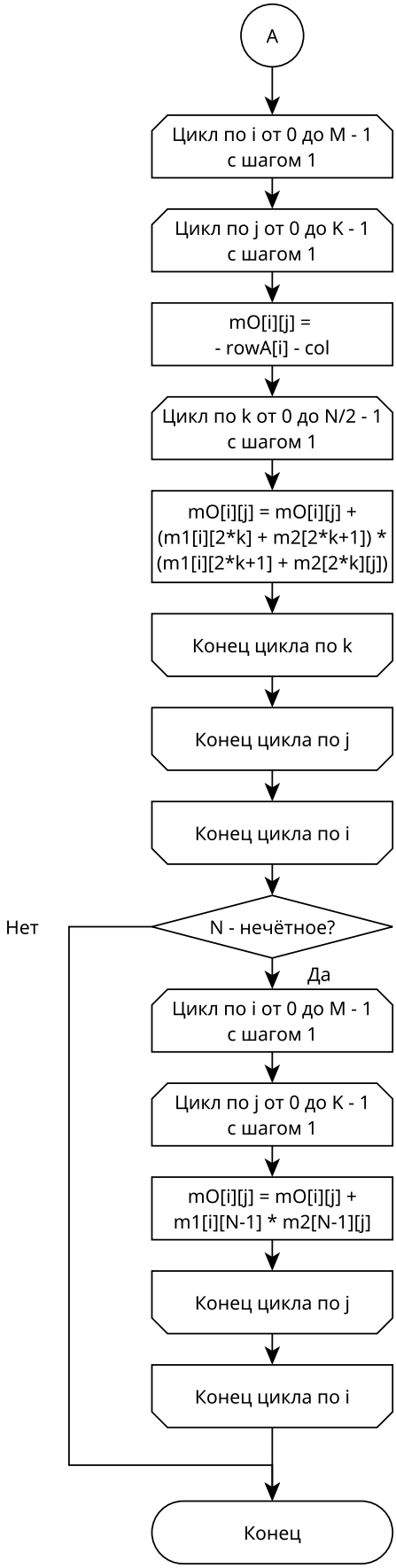

Рисунок 2.3 — Схема алгоритма Винограда (часть 2)

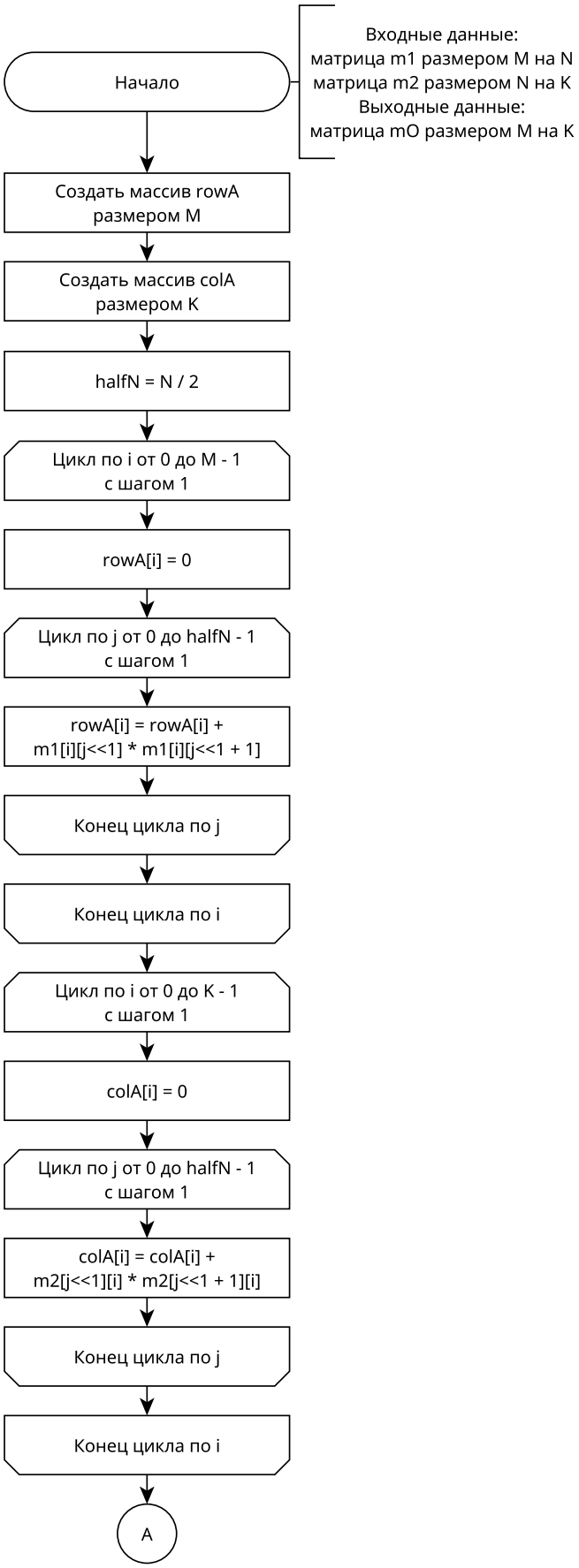

Рисунок 2.4 — Схема оптимизированного алгоритма Винограда (часть 1)

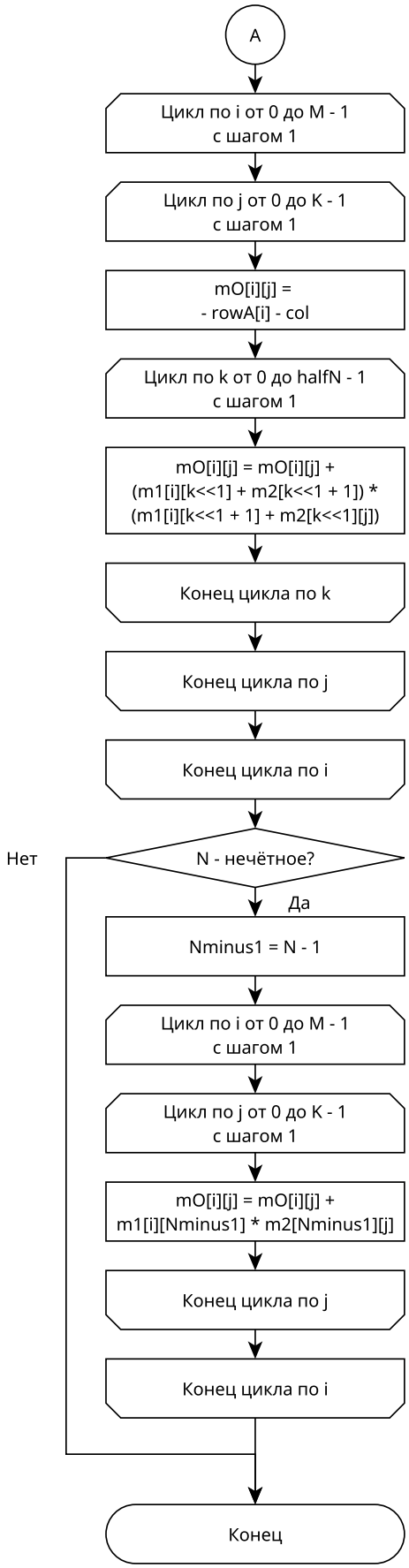

Рисунок 2.5 — Схема оптимизированного алгоритма Винограда (часть 2)

2.3 Модель вычислений

Для проведения теоретического анализа и оценки трудоёмкости алгоритмов вводится следующая модель вычислений:

1) стоимость базовых операций. Операции присваивания, сравнения, инкремента/декремента, простого сложения/вычитания и индексации массива ($=, ==, !=, <, >, <=, >=, ++, --, +, -, []$) имеют условную стоимость, равную 1;

2) стоимость арифметических операций. Операции умножения, деления и взятия остатка ($\cdot, /, \%$) считаются более затратными и имеют условную стоимость, равную 2;

3) трудоёмкость условного оператора. Для оператора *if (условие)* {блок X} *else* {блок Y} общая трудоёмкость (f_{if}) складывается из стоимости вычисления условия ($f_{\text{условия}}$) и стоимости выполнения соответствующей ветви:

$$f_{\text{if}} = f_{\text{условия}} + \begin{cases} \min(f_X, f_Y), & \text{в лучшем случае} \\ \max(f_X, f_Y), & \text{в худшем случае} \end{cases} \tag{2.1}$$

4) трудоёмкость цикла. Для цикла *for* с M итерациями общая трудоёмкость (f_{for}) рассчитывается как сумма стоимости инициализации (f_{init}), всех проверок условия ($M + 1$ раз) и выполнения тела цикла и инкремента (M раз):

$$f_{\text{for}} = f_{\text{init}} + (M + 1) \cdot f_{\text{условия}} + M \cdot (f_{\text{тело}} + f_{\text{инкремент}}) \tag{2.2}$$

2.4 Расчёт трудоёмкости алгоритмов

На основе принятой модели вычислений проведём оценку теоретической трудоёмкости для каждого из алгоритмов. Пусть матрицы имеют размеры $M \times N$ и $N \times K$.

2.4.1 Стандартный алгоритм

Трудоёмкость стандартного алгоритма определяется тремя вложенными циклами. Применив формулу (2.2), получается итоговую оценку:

$$f_{\text{std}} = 2 + M \cdot (2 + K \cdot (5 + N \cdot (2 + 12))) = 14MNK + 5MK + 2M + 2 \approx 14MNK \tag{2.3}$$

2.4.2 Алгоритм Винограда

Общая трудоёмкость складывается из четырёх этапов: вычисление факторов строк (f_1), факторов столбцов (f_2), основной цикл (f_3) и коррекция для нечётного случая (f_{if}).

1) заполнение *row_factor*: $f_1 = 2 + M \cdot (8 + \frac{N}{2} \cdot 19) = \frac{19}{2}MN + 8M + 2$;

2) заполнение *col_factor*: $f_2 = 2 + K \cdot (8 + \frac{N}{2} \cdot 19) = \frac{19}{2}NK + 8K + 2$;

3) основной цикл: $f_3 = 2 + M \cdot (4 + K \cdot (13 + \frac{N}{2} \cdot 32)) \approx 16MNK$;

4) коррекция для нечётного N: f_{if} в худшем случае составляет $16MK + 4M + 5$.

Итоговая трудоёмкость в худшем случае (нечётное N) составляет:

$$f_{\text{total\_vino}} = 16MNK + 29MK + \frac{19}{2}(MN + NK) + 16M + 8K + 11 \approx 16MNK \quad (2.4)$$

2.4.3 Оптимизированный алгоритм Винограда

1) предрасчет *halfN*: $f_0 = 3$;

2) заполнение *row_factor*: $f_1 = 2 + M \cdot (6 + \frac{N}{2} \cdot 17) = \frac{17}{2}MN + 6M + 2$;

3) заполнение *col_factor*: $f_2 = 2 + K \cdot (6 + \frac{N}{2} \cdot 17) = \frac{17}{2}NK + 6K + 2$;

4) основной цикл: $f_3 = 2 + M \cdot (4 + K \cdot (11 + \frac{N}{2} \cdot 26)) \approx 13MNK$;

5) коррекция для нечётного N: f_{if} в худшем случае составляет $14MK + 4M + 6$.

Итоговая трудоёмкость в худшем случае (нечётное N) составляет:

$$f_{\text{total\_vino}} = 13MNK + 27MK + \frac{17}{2}(MN + NK) + 14M + 6K + 15 \approx 13MNK \quad (2.5)$$

Вывод

В конструкторской части были сформулированы требования к программной реализации и представлены блок-схемы, описывающие логику работы трёх алгоритмов умножения матриц. Была введена модель вычислений, на основе которой проведён теоретический анализ трудоёмкости.

Результаты анализа показывают, что алгоритм Винограда имеет большую трудоёмкость по сравнению со стандартным алгоритмом, а его оптимизированная версия имеют преимущество перед стандартным методом.

3 Технологическая часть

3.1 Средства реализации

Для реализации алгоритмов был выбран язык C. Для измерения процессорного времени процесса используется функция clock [3] из заголовочного файла time.h. Для построения графиков используется утилита gnuplot.

3.2 Реализации алгоритмов

В листинге 3.1 представлен исходный код реализации стандартного алгоритма умножения матриц.

```
int multiply_std(matrix_t matrix_in_1, matrix_t matrix_in_2, matrix_t
   matrix_out) {
  if (matrix_in_1.cols != matrix_in_2.rows)
    return ERR_DIMENSIONS_MULTIPLY;
  for (size_t i = 0; i < matrix_out.rows; i++)
    for (size_t j = 0; j < matrix_out.cols; j++) {
      matrix_out.elements[i][j] = 0;
      for (size_t k = 0; k < matrix_in_1.cols; k++)
        matrix_out.elements[i][j] = matrix_out.elements[i][j] +
           matrix_in_1.elements[i][k] * matrix_in_2.elements[k][j];
    }
  return OK;
}
```

Листинг 3.1 — Реализация стандартного алгоритма умножения матриц

В листинге 3.2 представлен исходный код реализации алгоритма Винограда.

```
int multiply_vinograd(matrix_t matrix_in_1, matrix_t matrix_in_2,
   matrix_t matrix_out) {
  size_t M = matrix_out.rows;
  size_t N = matrix_in_1.cols;
  size_t K = matrix_out.cols;
  if (matrix_in_1.cols != matrix_in_2.rows)
    return ERR_DIMENSIONS_MULTIPLY;
  double *row_array = malloc(sizeof(double) * M);
  if (row_array == NULL)
    return ERR_NO_MEMORY;
  double *col_array = malloc(sizeof(double) * K);
  if (col_array == NULL)
    return ERR_NO_MEMORY;
  for (size_t i = 0; i < M; i++) {
```

```
    row_array[i] = 0;
    for (size_t j = 0; j < N / 2; j++)
      row_array[i] = row_array[i] + matrix_in_1.elements[i][2 * j] *
         matrix_in_1.elements[i][2 * j + 1];
  }
  for (size_t i = 0; i < K; i++) {
    col_array[i] = 0;
    for (size_t j = 0; j < N / 2; j++)
      col_array[i] = col_array[i] + matrix_in_2.elements[2 * j][i] *
         matrix_in_2.elements[2 * j + 1][i];
  }
  for (size_t i = 0; i < M; i++) {
    for (size_t j = 0; j < K; j++) {
      matrix_out.elements[i][j] = -row_array[i] - col_array[j];
      for (size_t k = 0; k < N / 2; k++) {
        matrix_out.elements[i][j] = matrix_out.elements[i][j] + (
           matrix_in_1.elements[i][2 * k] + matrix_in_2.elements[2 * k
           + 1][j]) * (matrix_in_1.elements[i][2 * k + 1] + matrix_in_2
           .elements[2 * k][j]);
      }
    }
  }
  if (N % 2 == 1) {
    for (size_t i = 0; i < M; i++)
      for (size_t j = 0; j < K; j++)
        matrix_out.elements[i][j] = matrix_out.elements[i][j] +
           matrix_in_1.elements[i][N - 1] * matrix_in_2.elements[N -
           1][j];
  }
  free(row_array);
  free(col_array);
  return OK;
}
```

Листинг 3.2 — Реализация алгоритма Винограда

Исходный код реализации оптимизированного алгоритма Винограда представлен в Приложении А в листинге 4.1.

3.3 Функциональные тесты

В таблице 3.1 представлены данные и результаты тестирования реализаций алгоритмов умножения матриц.

Таблица 3.1 — Функциональные тесты для реализации алгоритмов умножения матриц

Матрица 1	Матрица 2	Ожидаемый результат	Фактический результат
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} 7 \end{pmatrix}$	Сообщение об ошибке	Некорректные измерения
$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$	$\begin{pmatrix} \ \end{pmatrix}$	Сообщение об ошибке	Некорректные измерения
$\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$	$\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$

Все тесты пройдены успешно для всех трёх функций, реализующих рассматриваемые алгоритмы.

Вывод

В технологической части определены необходимые средства реализации, и с их помощью реализованы алгоритмы умножения матриц. Успешно проведено функциональное тестирование реализаций алгоритмов.

4 Исследовательская часть

4.1 Технические характеристики

Замеры процессорного времени проводились на ноутбуке с характеристиками:

— операционная система — Debian Unstable GNU/Linux с ядром версии 6.16.8-1;

— процессор — AMD Ryzen 5 7535HS с 6 физическими ядрами и 12 логическими ядрами с максимальной тактовой частотой 4.60 ГГц [4];

— оперативная память — 16 Гбайт типа LPDDR5 в 4 каналах с тактовой частотой 6400 МГц с таймингами CL 19 tRCD 15 tRP 17 tRAS 34 tRC 51.

4.2 Замеры процессорного времени

Замеры процессорного времени проводились для двух серий данных:

— квадратные матрицы размерами от 100 на 100 до 400 на 400 с шагом изменения 10;

— квадратные матрицы размерами от 101 на 101 до 401 на 401 с шагом изменения 10.

Для каждого размера производилось 100 измерений, результатом замера является среднее арифметическое из этих измерений. Матрицы заполнялись случайными числами. Время измеряется в миллисекундах.

Во время выполнения замеров ноутбук был подключён к сети, никаких прочих программ, кроме системных, запущено не было.

Результаты измерений для первой серии данных представлены в таблице 4.1.

Результаты измерений для второй серии данных представлены в таблице 4.2.

Таблица 4.1 — Результаты измерений для чётных размеров

Размер	Стандартный алгоритм	Алгоритм Винограда	Опт. алгоритм Винограда
100	2.93	1.85	1.85
110	3.93	2.49	2.49
120	4.97	3.12	3.14
130	6.30	3.97	3.98
140	7.95	5.02	5.04
150	9.72	6.19	6.19
160	12.17	8.02	8.02
170	14.21	9.17	9.15
180	17.05	10.87	10.93
190	19.87	12.58	12.58
200	23.28	15.03	15.02
210	26.86	17.01	17.01
220	30.98	19.81	19.78
230	35.53	22.61	22.66
240	40.15	25.81	25.77
250	45.79	28.64	28.64
260	51.43	32.37	32.38
270	57.90	36.20	36.16
280	64.11	40.43	40.42
290	71.77	45.03	45.04
300	78.25	51.60	51.73
310	87.81	54.79	54.78
320	96.52	66.32	66.43
330	105.15	65.06	65.11
340	115.30	72.79	72.91
350	126.99	78.67	78.60
360	136.21	86.17	86.35
370	149.56	93.61	93.66
380	160.85	103.75	104.04
390	174.90	109.09	109.37
400	187.34	119.06	119.25

Таблица 4.2 — Результаты измерений для нечётных размеров

Размер	Стандартный алгоритм	Алгоритм Винограда	Опт. алгоритм Винограда
101	2.97	1.89	1.90
111	3.94	2.50	2.51
121	5.10	3.23	3.24
131	6.45	4.09	4.10
141	8.09	5.11	5.13
151	9.94	6.33	6.33
161	12.14	7.66	7.68
171	14.45	10.32	10.36
181	17.15	11.07	11.10
191	20.08	12.91	12.93
201	23.50	15.55	15.35
211	27.18	17.80	17.84
221	31.41	20.03	19.93
231	36.24	22.58	22.65
241	41.04	25.97	25.99
251	45.93	30.17	30.17
261	52.09	33.26	33.28
271	58.70	37.02	37.04
281	65.74	40.94	40.96
291	73.18	45.09	45.13
301	80.78	51.27	51.50
311	90.11	56.99	57.00
321	100.24	63.97	64.09
331	109.53	70.64	70.55
341	117.81	79.33	79.35
351	129.05	79.43	79.50
361	140.94	87.74	87.62
371	153.18	96.03	96.39
381	165.31	101.90	102.02
391	175.56	115.94	116.15
401	190.88	123.81	123.82

Графики зависимости времени выполнения от размера матрицы для чётных размеров приведены на рисунке 4.1.

Графики зависимости времени выполнения от размера матрицы для нечётных размеров приведены на рисунке 4.2.

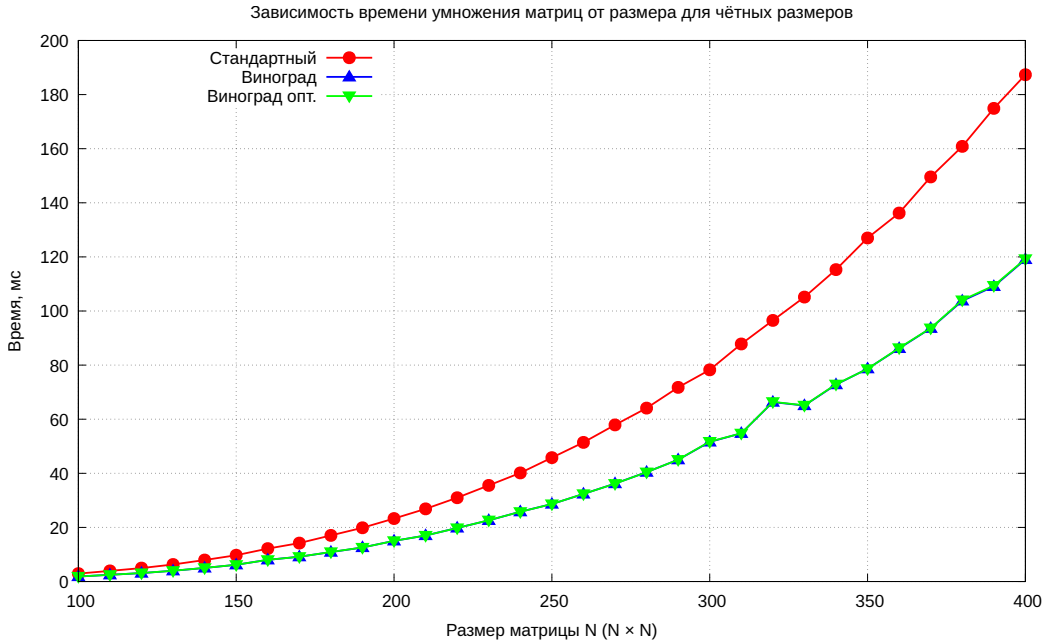

Рисунок 4.1 — Зависимость времени выполнения от размера матрицы для чётных размеров

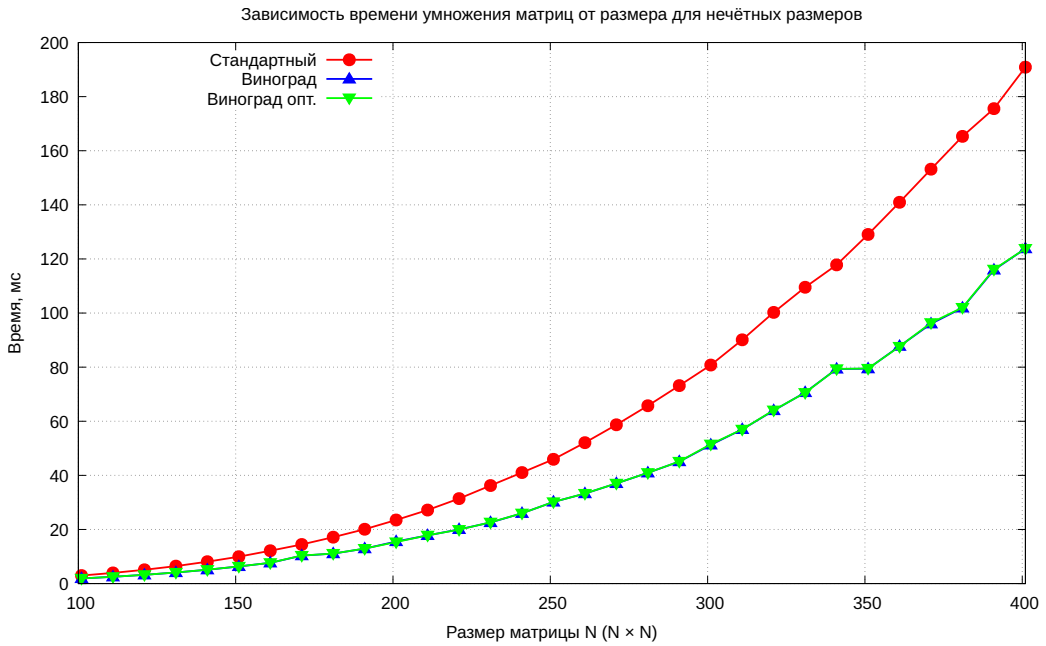

Рисунок 4.2 — Зависимость времени выполнения от размера матрицы для нечётных размеров

4.3 Вывод

В исследовательской части были произведены измерения зависимости процессорного времени работы реализации алгоритма от размера квадратной матрицы. Получены следующие результаты:

— вычисление произведения матриц нечётного размера алгоритмом Винограда происходит в среднем на 1-2% быстрее;

— оптимизация алгоритма Винограда не приводит к ускорению работы реализации алгоритма;

— реализация алгоритма Винограда работает быстрее стандартного в среднем в 1.35 раза.

ЗАКЛЮЧЕНИЕ

Цель данной лабораторной работы была достигнута: исследованы алгоритмы умножения матриц: алгоритм Винограда, стандартный алгоритм и оптимизированный алгоритм Винограда. Все задачи решены:

1) описаны стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда;
2) введена модель вычислений и выполнена в ней оценка трудоёмкости алгоритмов;
3) реализованы алгоритмы умножения матриц и проведено их тестирование;
4) выполнены замеры процессорного времени выполнения реализаций алгоритмов умножения матриц при варьируемом линейном размере квадратных перемножаемых матриц;
5) проведён сравнительный анализ алгоритмов по данным результатов замеров — реализация алгоритма Винограда работает быстрее стандартного в среднем в 1.35 раза.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Линейная алгебра : учеб. пособие / Н. В. Гредасова, М. А. Корешникова, Н. И. Желонкина [и др.] ; Мин-во науки и высш. образования РФ. — Екатеринбург : Изд-во Урал. ун-та, 2019. — 88 с.

2. Основы теории алгоритмов: учеб. пособие / А.А. Волосевич —– Минск: БГУИР, 2007 - 51 с.

3. clock(3) — Linux manual page [Электронный ресурс]. Режим доступа: `https://man7.org/linux/man-pages/man3/clock.3.html` (Дата обращения: 02.10.2025).

4. AMD Ryzen 5 7535HS [Электронный ресурс]. Режим доступа: `https://www.amd.com/en/products/processors/laptop/ryzen/7000-series/amd-ryzen-5-7535hs.html` (Дата обращения: 04.10.2025).

Приложение А

```
int multiply_vinograd_opt(matrix_t matrix_in_1, matrix_t matrix_in_2,
   matrix_t matrix_out) {
  size_t M = matrix_out.rows;
  size_t N = matrix_in_1.cols;
  size_t K = matrix_out.cols;
  if (matrix_in_1.cols != matrix_in_2.rows)
    return ERR_DIMENSIONS_MULTIPLY;
  double *row_array = malloc(sizeof(double) * M);
  if (row_array == NULL)
    return ERR_NO_MEMORY;
  double *col_array = malloc(sizeof(double) * K);
  if (col_array == NULL)
    return ERR_NO_MEMORY;
  size_t half_N = N / 2;
  for (size_t i = 0; i < M; i++) {
    row_array[i] = 0;
    for (size_t j = 0; j < half_N; j++)
      row_array[i] = row_array[i] + matrix_in_1.elements[i][(j << 1)] *
          matrix_in_1.elements[i][(j << 1) + 1];
  }
  for (size_t i = 0; i < K; i++) {
    col_array[i] = 0;
    for (size_t j = 0; j < half_N; j++)
      col_array[i] = col_array[i] + matrix_in_2.elements[(j << 1)][i] *
          matrix_in_2.elements[(j << 1) + 1][i];
  }
  for (size_t i = 0; i < M; i++) {
    for (size_t j = 0; j < K; j++) {
      matrix_out.elements[i][j] = -row_array[i] - col_array[j];
      for (size_t k = 0; k < half_N; k++) {
        matrix_out.elements[i][j] = matrix_out.elements[i][j] + (
            matrix_in_1.elements[i][(k << 1)] + matrix_in_2.elements[(k
            << 1) + 1][j]) * (matrix_in_1.elements[i][(k << 1) + 1] +
            matrix_in_2.elements[(k << 1)][j]);
      }
    }
  }
  if (N % 2 == 1) {
    size_t N_minus_1 = N - 1;
    for (size_t i = 0; i < M; i++)
```

```
      for (size_t j = 0; j < K; j++)
        matrix_out.elements[i][j] = matrix_out.elements[i][j] +
           matrix_in_1.elements[i][N_minus_1] * matrix_in_2.elements[
           N_minus_1][j];
  }
  free(row_array);
  free(col_array);
  return OK;
}
```

Листинг 4.1 — Реализация оптимизированного алгоритма Винограда