



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 по дисциплине «Анализ Алгоритмов»

Тема Алгоритмы умножения матриц

Студент Нисуев Н.Ф.

Группа ИУ7-52Б

Преподаватель Волкова Л. Л., Строганов Д.В.

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	2
1 Аналитическая часть	3
1.1 Описание алгоритмов	3
1.1.1 Классический алгоритм умножения матриц	3
1.1.2 Алгоритм Винограда умножения матриц	4
2 Конструкторская часть	5
2.1 Представление алгоритмов	5
2.2 Модель вычислений	11
2.3 Трудоемкость алгоритмов	11
2.3.1 Классический алгоритм перемножения матриц	12
2.3.2 Алгоритм Винограда	12
2.3.3 Оптимизированный алгоритм Винограда	13
3 Технологическая часть	15
3.1 Требования к программному обеспечению	15
3.2 Средства реализации	15
3.3 Реализация алгоритмов	15
4 Исследовательская часть	20
4.1 Технические характеристики	20
4.2 Описание используемых типов данных	20
4.3 Время выполнения алгоритмов	21
4.4 Вывод	23
ЗАКЛЮЧЕНИЕ	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	25

ВВЕДЕНИЕ

Матрицы представляют собой таблицы чисел, взаимосвязанных между собой.

Цель лабораторной работы — исследование алгоритмов умножения матриц следующими методами:

- классическим алгоритмом;
- алгоритмом Винограда;
- оптимизированного алгоритма Винограда.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- реализовать указанные алгоритмы;
- выполнить оценку трудоемкости разрабатываемых алгоритмов;
- сравнение требуемого времени выполнения алгоритмов;
- описать и обосновать полученные результаты.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы умножения матриц.

1.1 Описание алгоритмов

Классический алгоритм умножения матриц реализует математическое определение этого процесса и обладает асимптотической сложностью $O(n^3)$.

Алгоритм Винограда, благодаря своей асимптотической сложности $O(n^{2.3755})$, считается одним из наиболее эффективных методов для умножения матриц с точки зрения времени.

Пусть даны матрицы A с размерами $N \times M$ и B с размерами $M \times K$. В результате умножения матрицы A на матрицу B получается матрица C с размером $N \times K$.

1.1.1 Классический алгоритм умножения матриц

Пусть даны матрицы A размерностью $N \times M$ и матрица B размерностью $M \times k$:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \dots & \dots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mk} \end{pmatrix}. \quad (1.1)$$

Тогда умножением матрицы A на матрицу B называется, где матрица C :

$$C = A \times B = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1k} \\ c_{21} & c_{22} & \dots & c_{2k} \\ \dots & \dots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nk} \end{pmatrix}, \quad (1.2)$$

где

$$c_{ij} = \sum_{k=1}^m a_{ik} \cdot b_{kj} \quad (i = \overline{1, n}, \quad j = \overline{1, k}). \quad (1.3)$$

1.1.2 Алгоритм Винограда умножения матриц

Пусть даны матрицы A и B , имеющие размерность 4×4 .

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}. \quad (1.4)$$

Для получения очередного элемента c_{ij} матрицы C в классическом алгоритме умножения матрицы выполняется по формуле:

$$c_{ij} = \begin{pmatrix} a_{n1} & a_{n2} & a_{n3} & a_{n4} \end{pmatrix} \times \begin{pmatrix} b_{j1} \\ b_{j2} \\ b_{j3} \\ b_{j4} \end{pmatrix}, \quad (1.5)$$

где a_{ni} , $i = \overline{1, 4}$ - элементы n -ой строки матрицы A ; b_{jk} , $k = \overline{1, 4}$ - элементы j -ого столбца матрицы B .

В алгоритме Винограда для ускорения вычислений уменьшается количество затратных операций умножения, заменяя их сложениями. Для этого проводится предварительная обработка, при которой сохраняются некоторые значения, что позволяет в дальнейшем заменить часть умножений операциями сложения:

$$c_{ij} = (a_{n1} + b_{j2})(a_{n2} + b_{j1}) + (a_{n3} + b_{j4})(a_{n4} + b_{j3}) - a_{n1}a_{n2} - a_{n3}a_{n4} - b_{j1}b_{j2} - b_{j3}b_{j4}, \quad (1.6)$$

где элементы $a_{n1}a_{n2}$, $a_{n3}a_{n4}$, $b_{j1}b_{j2}$, $b_{j3}b_{j4}$ - значения, которые получаются в предварительной обработке.

ВЫВОД

В этом разделе рассмотрены классический алгоритм и алгоритм Винограда для умножения матриц. Ключевые различия между ними заключаются в предварительной обработке данных и количестве выполняемых операций умножения.

2 Конструкторская часть

В данном разделе будут приведены схемы алгоритмов умножения матриц, а также оценены трудоемкости каждого из алгоритмов.

2.1 Представление алгоритмов

На вход алгоритмов подаются две матрицы: M_1 размера $M \times N$, M_2 размера $N \times K$, где M, N, K — неотрицательные целые числа; на выходе — матрица M_3 размера $M \times K$.

На рисунках 2.1 — 2.3 приведены схемы трех алгоритмов умножения матриц: классического, Винограда, оптимизированного Винограда.

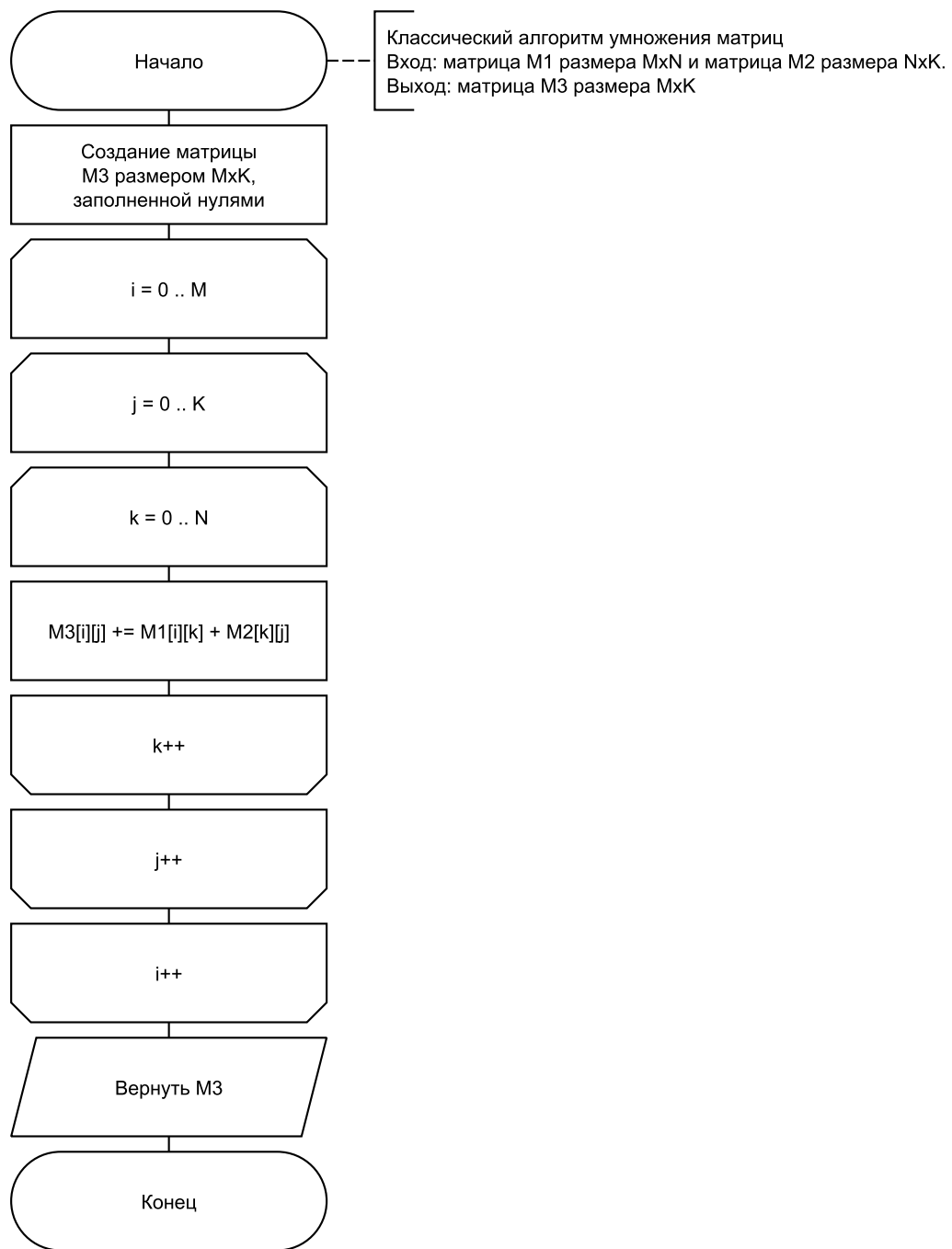
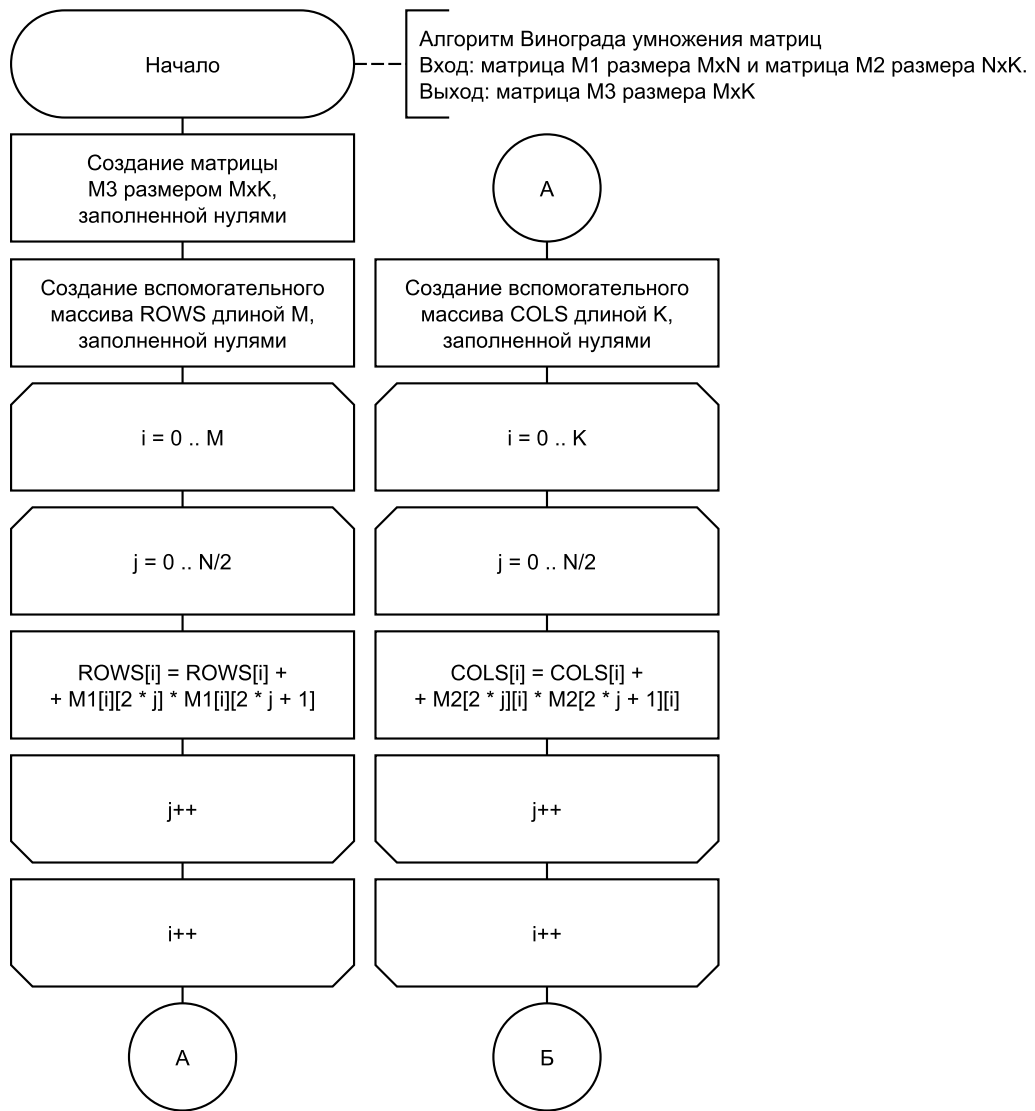


Рисунок 2.1 – Схема классического алгоритма умножения матриц



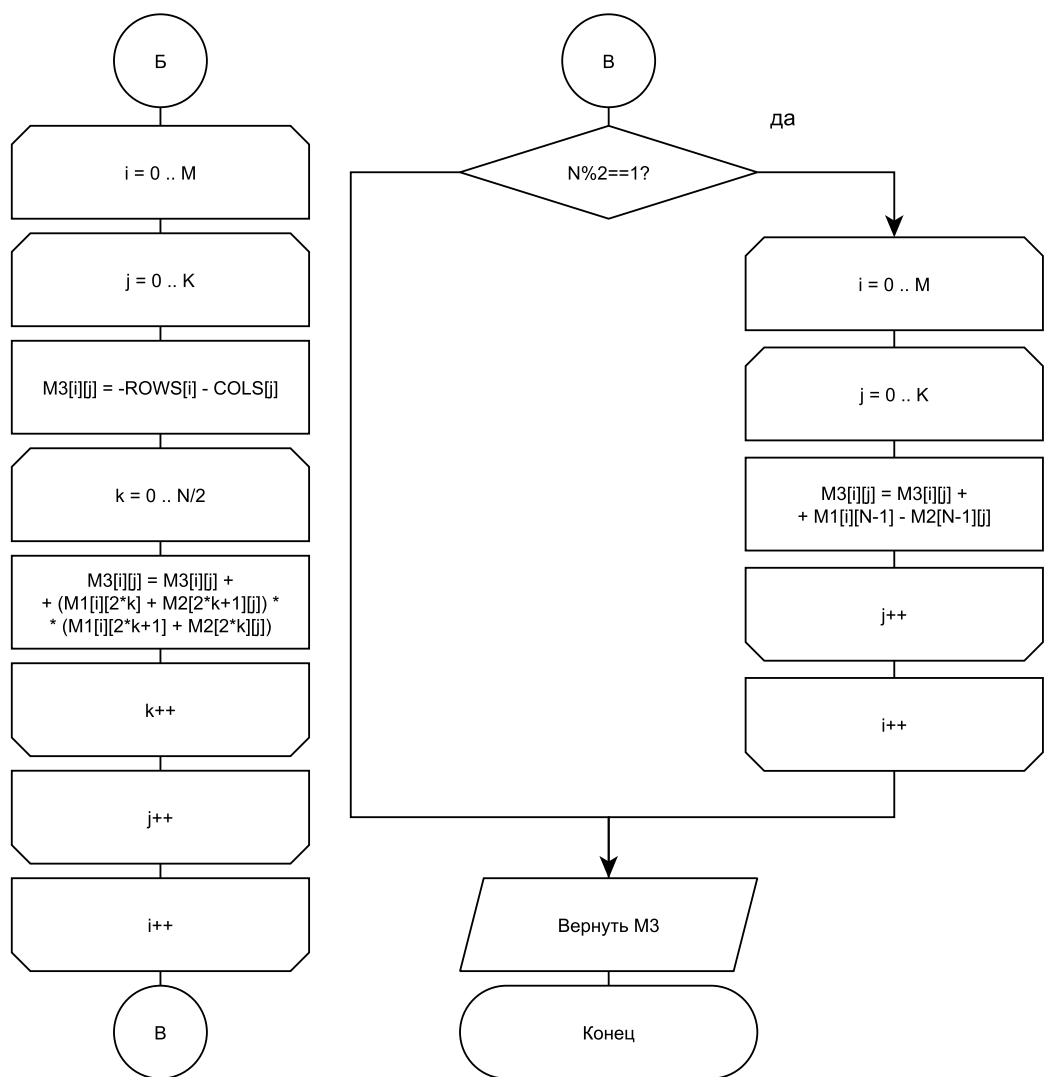
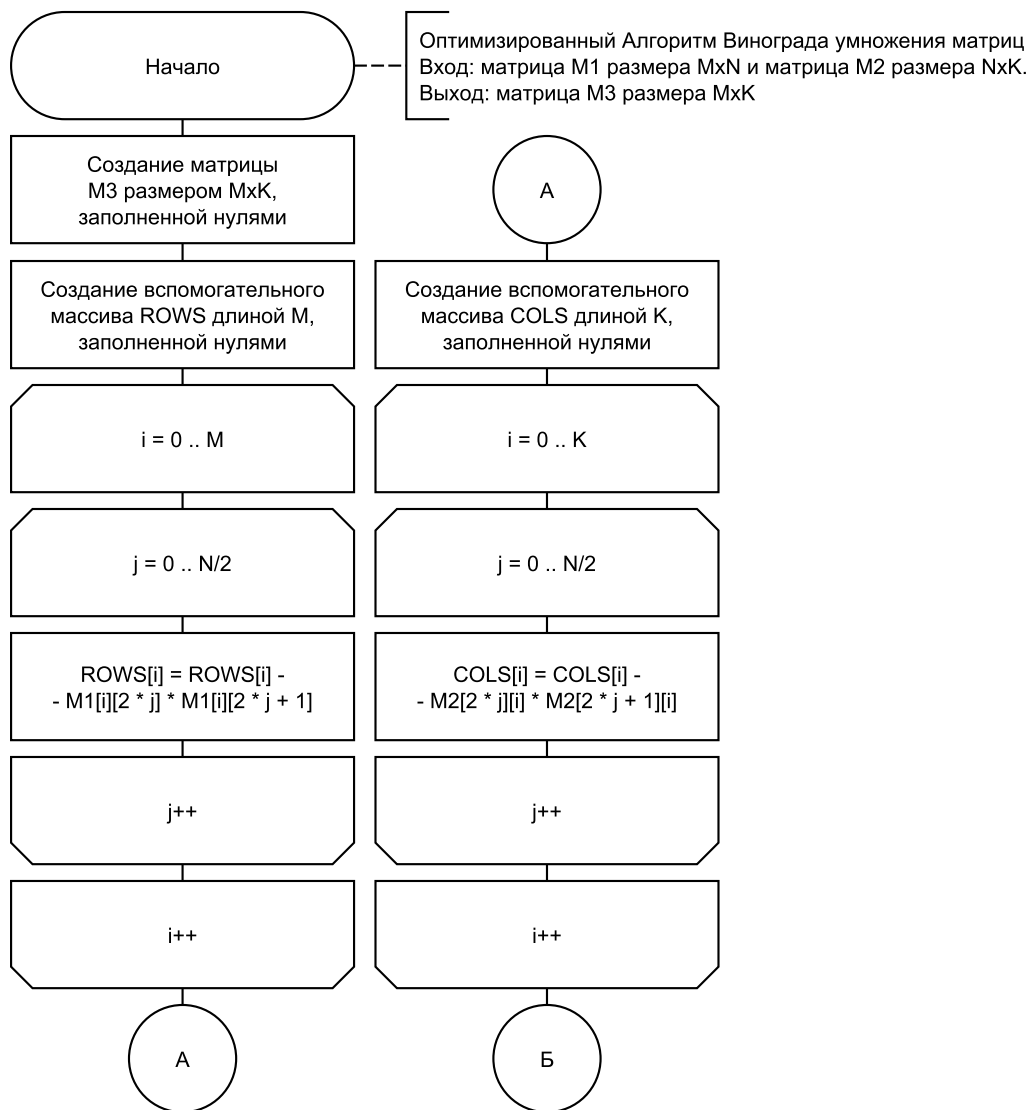


Рисунок 2.2 – Схема алгоритма Винограда умножения матриц



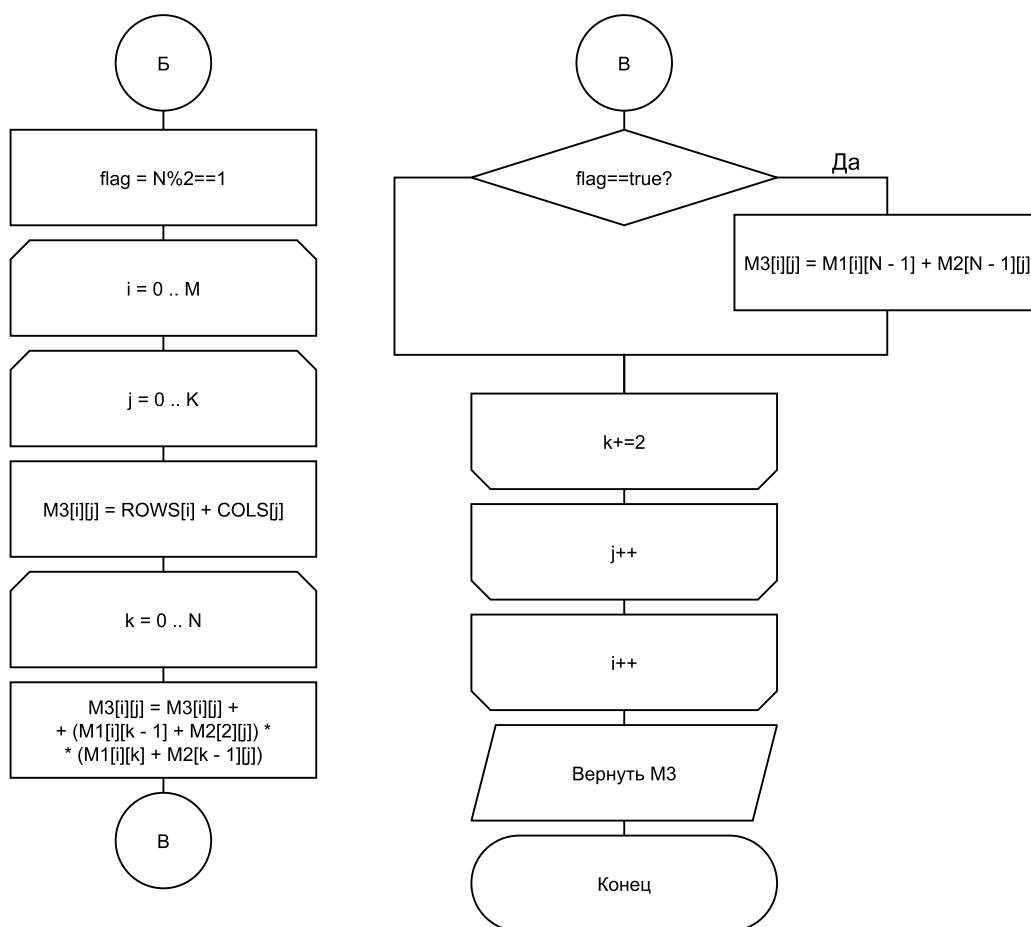


Рисунок 2.3 – Схема оптимизированного алгоритма Винограда

Оптимизация алгоритма Винограда заключается:

- в инкременте наиболее вложенного счетчика цикла на 2;
- в объединении III и IV части алгоритма;
- в введении декремента при вычислении вспомогательных массивов.

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, *, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \quad (2.1)$$

2. трудоемкость условного оператора `if условие then A else B` рассчитывается как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2.2)$$

3. трудоемкость цикла рассчитывается как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \quad (2.3)$$

4. трудоемкость вызова функции/возврата результата равна 0.

2.3 Трудоемкость алгоритмов

В следующих частях будут рассчитаны трудоемкости представленных ранее классического алгоритма, алгоритма Винограда, оптимизированного алгоритма Винограда. Трудоемкость инициализации результирующей матрицы учитываться

не будет, поскольку данное действие есть во всех алгоритмах и не является самым трудоемким.

Введем обозначения:

- M — кол-во строк первой матрицы;
- N — кол-во столбцов первой матрицы и кол-во строк второй матрицы;
- K — кол-во столбцов второй матрицы.

2.3.1 Классический алгоритм перемножения матриц

Трудоемкость стандартного алгоритма умножения матриц состоит из:

- внешнего цикла по $i \in [1..M]$, трудоемкость которого: $f = 2 + M \cdot (2 + f_{body})$;
- цикла по $j \in [1..K]$, трудоемкость которого: $f = 2 + K \cdot (2 + f_{body})$;
- цикла по $q \in [1..N]$, трудоемкость которого: $f = 2 + 10 \cdot N$.

Трудоемкость классического алгоритма равна трудоемкости внешнего цикла. Ее можно вычислить, подставив циклы тела (2.4):

$$f_{classic} = 2 + M \cdot (4 + K \cdot (4 + 10N)) = 2 + 4M + 4MK + 10MKN \approx 10MKN \quad (2.4)$$

2.3.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда состоит из:

- создания и инициализации массивов $ROWS$ и $COLS$, трудоемкость которого (2.11):

$$f_{init} = M + K; \quad (2.5)$$

- заполнения массива $ROWS$, трудоемкость которого (2.6):

$$f_{ROWS} = 2 + M \cdot (4 + \frac{N}{2} \cdot 15); \quad (2.6)$$

- заполнения массива $COLS$, трудоемкость которого (2.7):

$$f_{COLS} = 2 + K \cdot (4 + \frac{N}{2} \cdot 15); \quad (2.7)$$

- цикла заполнения для четных размеров, трудоемкость которого (2.8):

$$f_{cycle} = 2 + M \cdot (2 + K \cdot (2 + 7 + 4 + \frac{N}{2} \cdot 25)); \quad (2.8)$$

- цикла, для дополнения результирующего массива суммой последних нечетных строки и столбца, если общий размер нечетный, трудоемкость которого (2.9):

$$f_{last} = \begin{cases} 2, & \text{размер четный,} \\ 2 + M \cdot (2 + 9K), & \text{иначе.} \end{cases} \quad (2.9)$$

Итого, результирующая трудоемкость алгоритма Винограда равна (2.10)

$$f_{final} = f_{init} + f_{ROWS} + f_{COLS} + f_{cycle} + f_{last} \approx 12MNK \quad (2.10)$$

Алгоритм Винограда (неоптимизированный) имеет большую трудоемкость, чем классический алгоритм.

2.3.3 Оптимизированный алгоритм Винограда

Трудоемкость оптимизированного алгоритма Винограда состоит из:

- создания и инициализации массивов $A1$ и $B1$ а также доп. переменной, хранящей $N/2 == 1$, трудоемкость которого (2.11):

$$f_{init} = M + K + 4; \quad (2.11)$$

- заполнения массива $ROWS$, трудоемкость которого (2.12):

$$f_{A1} = 2 + M \cdot (4 + \frac{N}{2} \cdot 15); \quad (2.12)$$

— заполнения массива $COLS$, трудоемкость которого (2.13):

$$f_{B1} = 2 + K \cdot (4 + \frac{N}{2} \cdot 15); \quad (2.13)$$

— цикла заполнения для четных размеров, трудоемкость которого (2.14):

$$f_{cycle} = 2 + M \cdot (2 + K \cdot (2 + 7 + 2 + \frac{N}{2} \cdot 17) + f_{last}); \quad (2.14)$$

где f_{last} — IV часть алгоритма, трудоемкость которой равна (2.15):

$$f_{last} = \begin{cases} 1, & \text{размер четный,} \\ 1 + 11, & \text{иначе.} \end{cases} \quad (2.15)$$

Итого, результирующая трудоемкость оптимизированного алгоритма Винограда равна (2.16)

$$f_{final} = f_{init} + f_{A1} + f_{B1} + f_{cycle} + f_{last} \approx 8.5MNK \quad (2.16)$$

Оптимизированный алгоритм Винограда имеет меньшую трудоемкость, по сравнению с классическим алгоритмом.

ВЫВОД

В данном разделе были представлены схемы алгоритмов умножения матриц, а также приведены трудоемкости каждого из трех алгоритмов.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации, листинги кода.

3.1 Требования к программному обеспечению

Входные данные: две матрицы, где количество столбцов первой матрицы равна количеству строк второй матрицы;

Выходные данные: матрица, где количество строк равно количеству строк первой матрицы, а количество столбцов равно количеству столбцов второй матрицы.

3.2 Средства реализации

В данной работе для реализации был выбран язык программирования C [1]. Выбор обусловлен наличием функции вычисления процессорного с помощью функции *clock* [2].

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 представлены реализации алгоритмов нахождения расстояния Левенштейна и Дамерау–Левенштейна.

Листинг 3.1 – Классический алгоритм

```
1 matrix_t *std_matrix_mult(const matrix_t * const mtr1, const
  matrix_t * const mtr2) {
2     if (mtr1 == NULL || mtr2 == NULL) return NULL;
3     if (mtr1->cols != mtr2->rows) return NULL;
4
5     matrix_t *res = init_matrix_by_size(mtr1->rows, mtr2->cols);
6     if (!res) return NULL;
7
8     for (size_type i = 0; i < mtr1->rows; ++i) {
9         for (size_type j = 0; j < mtr2->cols; ++j) {
10             value_type sum = 0;
11             for (size_type k = 0; k < mtr1->cols; ++k) {
12                 sum += mtr1->data[i][k] * mtr2->data[k][j];
13             }
14             res->data[i][j] = sum;
15         }
16     }
17     return res;
18 }
```

Листинг 3.2 – Алгоритм Винограда

```
1 matrix_t *winograd_matrix_mult(const matrix_t * const mtr1, const
  matrix_t * const mtr2) {
2     if (mtr1 == NULL || mtr2 == NULL) return NULL;
3     if (mtr1->cols != mtr2->rows) return NULL;
4
5     size_type rows1 = mtr1->rows;
6     size_type cols1 = mtr1->cols;
7     size_type cols2 = mtr2->cols;
8
9     matrix_t *res = init_matrix_by_size(rows1, cols2);
10    if (!res) return NULL;
11
12    value_type *row_factor = (value_type *)malloc(rows1 *
      sizeof(value_type));
13    if (!row_factor) goto end;
14    for (size_type i = 0; i < rows1; ++i) {
15        row_factor[i] = 0;
16        for (size_type j = 0; j < cols1 / 2; ++j) {
```

```

17         row_factor[i] += mtr1->data[i][2 * j] * mtr1->data[i][2
18             * j + 1];
19     }
20 }
21 value_type *col_factor = (value_type *)malloc(cols2 *
22     sizeof(value_type));
23 if (!col_factor) goto free_row_factor;
24 for (size_type j = 0; j < cols2; ++j) {
25     col_factor[j] = 0;
26     for (size_type i = 0; i < cols1 / 2; ++i) {
27         col_factor[j] += mtr2->data[2 * i][j] * mtr2->data[2 * i
28             + 1][j];
29     }
30 }
31 for (size_type i = 0; i < rows1; ++i) {
32     for (size_type j = 0; j < cols2; ++j) {
33         res->data[i][j] = -row_factor[i] - col_factor[j];
34         for (size_type k = 0; k < cols1 / 2; ++k) {
35             res->data[i][j] += (mtr1->data[i][2 * k] +
36                 mtr2->data[2 * k + 1][j]) *
37                 (mtr1->data[i][2 * k + 1] +
38                     mtr2->data[2 * k][j]);
39         }
40     }
41 }
42 if (cols1 % 2 == 1) {
43     for (size_type i = 0; i < rows1; ++i) {
44         for (size_type j = 0; j < cols2; ++j) {
45             res->data[i][j] += mtr1->data[i][cols1 - 1] *
46                 mtr2->data[cols1 - 1][j];
47         }
48     }
49 }
50
51 free(col_factor);
52 free(row_factor);
53
54 return res;

```

```

52
53     free_row_factor:
54     free(row_factor);
55     end:
56     free_matrix(res);
57     return res;
58 }

```

Листинг 3.3 – Оптимизированный алгоритм Винограда

```

1 matrix_t *optimized_winograd_matrix_mult(const matrix_t * const
    mtr1, const matrix_t * const mtr2) {
2     if (mtr1 == NULL || mtr2 == NULL) return NULL;
3     if (mtr1->cols != mtr2->rows) return NULL;
4
5     size_type rows1 = mtr1->rows;
6     size_type cols1 = mtr1->cols;
7     size_type cols2 = mtr2->cols;
8
9     matrix_t *res = init_matrix_by_size(rows1, cols2);
10    if (!res) return NULL;
11
12    value_type *row_factor = (value_type *)malloc(rows1 *
        sizeof(value_type));
13    if (!row_factor) goto end;
14    for (size_type i = 0; i < rows1; ++i) {
15        row_factor[i] = 0;
16        for (size_type j = 0; j < cols1 / 2; ++j) {
17            row_factor[i] -= mtr1->data[i][2 * j] * mtr1->data[i][2
                * j + 1];
18        }
19    }
20
21    value_type *col_factor = (value_type *)malloc(cols2 *
        sizeof(value_type));
22    if (!col_factor) goto free_row_factor;
23    for (size_type j = 0; j < cols2; ++j) {
24        col_factor[j] = 0;
25        for (size_type i = 0; i < cols1 / 2; ++i) {
26            col_factor[j] -= mtr2->data[2 * i][j] * mtr2->data[2 * i
                + 1][j];
27        }

```

```

28     }
29
30     bool flag = cols1 % 2 == 1;
31     for (size_type i = 0; i < rows1; ++i) {
32         for (size_type j = 0; j < cols2; ++j) {
33             res->data[i][j] = row_factor[i] + col_factor[j];
34             for (size_type k = 1; k < cols1; k += 2) {
35                 res->data[i][j] += (mtr1->data[i][k - 1] +
36                                     mtr2->data[k][j]) *
37                                     (mtr1->data[i][k] +
38                                     mtr2->data[k - 1][j]);
39             }
40             if (flag) {
41                 res->data[i][j] += mtr1->data[i][cols1 - 1] *
42                                     mtr2->data[cols1 - 1][j];
43             }
44         }
45     }
46
47     return res;
48
49     free_row_factor:
50     free(row_factor);
51     end:
52     free_matrix(res);
53     return res;
54 }

```

ВЫВОД

В этом разделе рассмотрены требования к программному обеспечению, описаны используемые инструменты реализации, а также приведены фрагменты кода для умножения матриц с использованием классического алгоритма, алгоритма Винограда и его оптимизированной версии.

4 Исследовательская часть

4.1 Технические характеристики

Характеристики используемого оборудования:

- Операционная система — Windows 11 Home [3]
- Память — 16 Гб.
- Процессор — 12th Gen Intel(R) Core(TM) i7-12700H @ 2.30 ГГц [4]
- Микроконтроллер — STM32F303 [5]

4.2 Описание используемых типов данных

Используемые типы данных:

размер матрицы — целое число типа *size_t*;

матрица — *double * **.

4.3 Время выполнения алгоритмов

Результаты замеров времени работы трех алгоритмов умножения матриц приведены в таблице ???. Время работы замерялось на микроконтроллере *STM32F303* с тактовой частотой до 72 МГц. Замеры времени проводились на матрицах одинаковой длины и усреднялись для каждого набора одинаковых экспериментов. Каждое значение получено путем взятия среднего из 100 измерений. Зависимости времени умножения от размера матрицы для трех алгоритмов представлены на рисунке 4.1.

Таблица 4.1 – Время работы алгоритмов (в мс)

Размер матрицы	Классический	Виноград	Виноград (оптимизированный)
0	0.00	0.23	0.00
2	0.24	0.52	0.18
4	1.03	0.74	1.48
6	2.07	3.14	2.51
8	4.56	5.67	4.72
10	7.82	9.13	8.24
12	14.07	14.62	13.57
14	22.49	23.21	21.08
16	30.13	34.95	28.37
18	50.48	48.33	39.58
20	60.72	65.83	54.97
22	85.34	82.19	74.23
24	110.27	107.89	98.64
26	140.12	135.48	125.32
28	174.68	170.57	154.98
30	215.32	205.86	195.47
32	260.59	245.32	230.14
34	320.13	290.68	275.79
36	375.72	350.21	330.48
38	440.65	405.34	380.73
40	510.48	475.27	450.32
42	590.23	540.19	510.87
44	680.79	620.35	600.24
46	790.41	710.74	690.68
48	910.52	810.59	780.13
50	1040.19	930.48	870.77

Сравнение алгоритмов умножения матриц

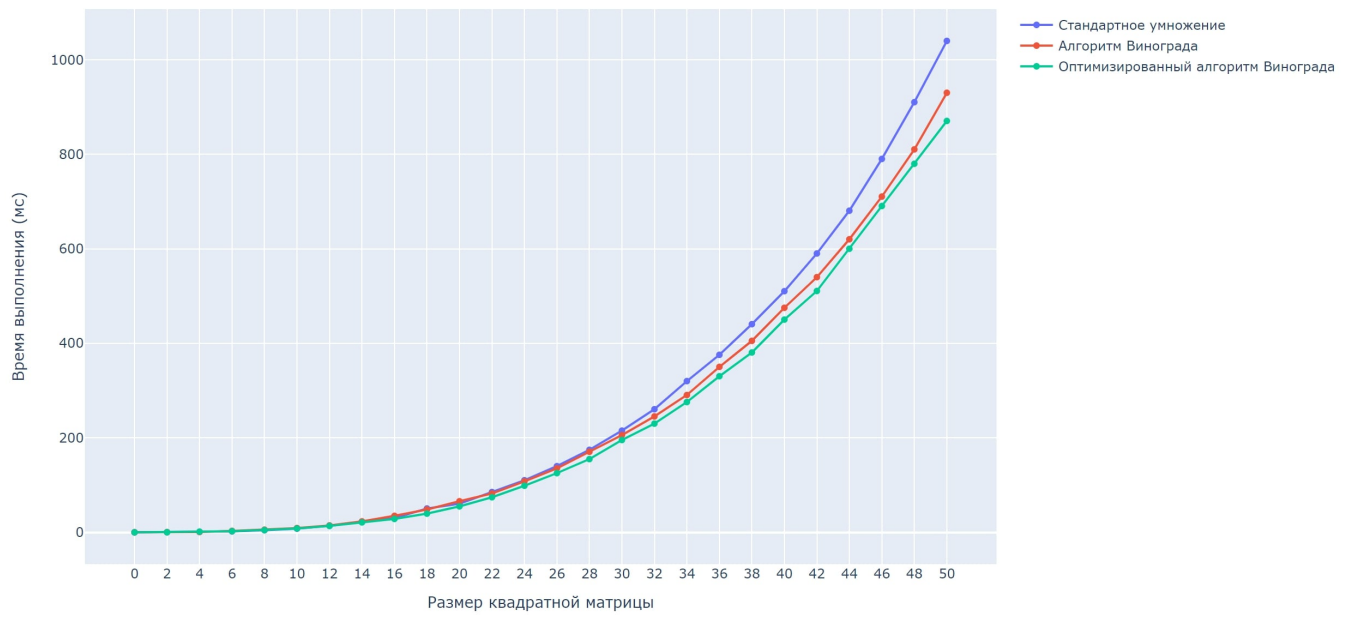


Рисунок 4.1 – Сравнение алгоритмов по времени

4.4 Вывод

В ходе исследования установлено, что для больших размеров матриц (более 10) алгоритм Винограда оказывается более чем в 1.2 раза быстрее классического алгоритма, а оптимизированная версия алгоритма Винограда превосходит стандартный алгоритм почти в 1.3 раза. Это позволяет заключить, что для таких данных наиболее целесообразно применять оптимизированный алгоритм Винограда.

Также было выявлено, что алгоритм Винограда работает эффективнее на матрицах с четными размерами, поскольку для нечетных матриц требуются дополнительные вычисления для обработки крайних строк и столбцов. Таким образом, алгоритм Винограда рекомендуется использовать при работе с матрицами четных размеров.

ЗАКЛЮЧЕНИЕ

Исследование показало, что классический алгоритм умножения матриц уступает по времени алгоритму Винограда примерно в 1.2 раза. Это связано с тем, что в алгоритме Винограда часть вычислений производится заранее, а количество сложных операций, таких как умножение, сокращается, что делает его более предпочтительным. Однако наилучшие результаты по скорости демонстрирует оптимизированный алгоритм Винограда, который на матрицах размером более 10 работает примерно в 1.3 раза быстрее, чем классический алгоритм. Это достигается за счет использования операций плюс-равно вместо равно и плюс, замены умножений сдвигами, а также предварительного вычисления некоторых элементов. Таким образом, для достижения максимальной производительности предпочтительнее использовать оптимизированный алгоритм Винограда.

В ходе выполнения данной лабораторной работы были решены следующие задачи: В ходе выполнения данной лабораторной работы были решены следующие задачи:

- реализованы указанные алгоритмы;
- выполнена оценка трудоемкости разрабатываемых алгоритмов;
- проведено сравнение требуемого времени выполнения алгоритмов;
- описаны и обоснованы полученные результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по языку C [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/c-language/> (дата обращения: 01.09.2024).
2. clock(3) - Linux manual page [Электронный ресурс]. — Режим доступа: <https://man7.org/linux/man-pages/man3/clock.3.html> (дата обращения: 01.09.2024).
3. Windows technical documentation for developers and IT pros [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/en-us/windows/> (дата обращения: 06.10.2024).
4. Intel® Core™ i7-12700H Processor [Электронный ресурс]. — Режим доступа: <https://ark.intel.com/content/www/us/en/ark/products/132228/intel-core-i7-12700h-processor-24m-cache-up-to-4-70-ghz.html> (дата обращения: 07.10.2024).
5. Микроконтроллер STM32F303 Discovery [Электронный ресурс]. — Режим доступа: <https://www.st.com/en/microcontrollers-microprocessors/stm32f303.html> (дата обращения: 04.10.2024).