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»

Отчёт о лабораторной работе №2 по дисциплине "Анализ алгоритмов" на тему "Умножение матриц"

Студент Коняев Е.А

Группа ИУ7-53Б

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

Дата сдачи отчета_____

Оценка (баллы) _____

Москва — 2022 г.

Оглавление

Введение

Матрица – это математический объект, который представляет собой таблицу чисел из определенного количества строк и столбцов.

Операции обработки матриц, а в частности их умножение, является одной из самых фундаментальных операций в современных вычислениях. Оно выполняется миллиарды раз в день по всему миру в вычислениях для линейной алгебры, статистики, биоинформатики, физики и разных других областей науки, техники и обработки сигналов и образов.

Умножение двух матриц возможно только в том случае, если число столбцов первой матрицы совпадает с числом строк во второй. Результирующая матрица будет иметь столько же строк, сколько в первой матрице, и столько же столбцов, сколько во второй. При этом данная операция не является в общем случае коммутативной.

Глава 1

Аналитическая часть

1.1 Цель и задачи

Целью данной работой является изучение и реализация алгоритмов умножения матриц, вычисление трудоемкости этих алгоритмов. В данной лабораторной работе рассматривается стандартный алгоритм умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда. Для достижения поставленной цели требуется решить следующие задачи:

1) изучить и рассмотреть алгоритм классического умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда;

2) построить блок-схемы данных алгоритмов;

3) реализовать каждый из алгоритмов;

4) оценить трудоемкость алгоритмов;

5) экспериментально оценить временные характеристики алгоритмов;

6) сделать вывод на основании проделанной работы.

1.2 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A_{lm} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \dots & a_{lm} \end{pmatrix}, \quad B_{mn} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}, \tag{1.1}$$

тогда матрица C, полученная в результате умножения этих матриц, будет иметь вид

$$C_{ln} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \dots & c_{ln} \end{pmatrix}, \tag{1.2}$$

где

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj} \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{1.3}$$

Стандартный алгоритм реализует данную формулу.

1.3 Алгоритм Копперсмита – Винограда

В стандартном умножении двух матриц каждый элеменит результирующей матрицы представляет собой скалярное произведение двух векторов исходных матриц (строки из первой матрицы и столбца из второй). Такое умножение допускает предварительную обработку, которая позволяет часть вычислений выполнить заранее.

Рассмотрим два вектора $V = (v_1, v_2, v_3, v_4)$ и $W = (w_1, w_2, w_3, w_4)$. Их скалярное произведение: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что эквивалентно (1.4)

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1v_2 - v_3v_4 - w_1w_2 - w_3w_4. \tag{1.4}$$

Хоть выражение в правой части (1.4) имеет большее количество операций, чем стандартный алгоритм, но оно допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки и столбца первой и второй матрицы соответсвенно. Таким образом в рассматриваемом выше примере в процессе самого умножения требуется выполнить лишь 2 операции умножения, и 5 операций сложения (вместо 4 операций умножения и 3 операций сложения в стандартном алгоритме). Так как операция сложения выполняется в вычислительных машинах быстрее, чем умножению, то данный алгоритм по теоретической оценке должен работать быстрее, чем стандартный алгоритм умножения матриц.

Вывод

В данном разделе были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда.

Глава 2

Конструкторская часть

2.1 Модель вычислений

Для вычисления трудоемкости будем использовать следующую модель вычислений.

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

$$+, -, \%, ==, !=, <, >, <=, >=, [], ++, --, << , >> . \tag{2.1}$$

2) операции из списка (2.2) имеют трудоемкость 2.

$$*, /. \tag{2.2}$$

3) трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.3).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

4) трудоемкость вызова функции равна 0.

5) трудоемкость цикла рассчитывается, как (2.4).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

2.2 Трудоёмкость алгоритмов

2.2.1 Стандартный алгоритм умножения матриц

Пусть N – количество строк в первой матрице, M – количество колонок во второй матрице, Q – количество колонок в первой матрице. Трудоёмкость стандартного алгоритма умножения матриц состоит из:

1) внешнего цикла по $i \in [0..N)$, трудоёмкость которого: $f = 2 + N \cdot (2 + f_{body})$;

2) цикла по $j \in [0..M)$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;

3) скалярного умножения двух векторов - цикл по $k \in [0..Q)$, трудоёмкость которого: $f = 2 + 10Q$.

Трудоёмкость стандартного алгоритма равна трудоёмкости внешнего цикла, можно вычислить ее, подставив циклы тела (2.5)

$$f_{base} = 2 + N \cdot (4 + M \cdot (4 + 10Q)) = 2 + 4N + 4NM + 10NMQ \approx 10NMQ \tag{2.5}$$

2.2.2 Алгоритм Копперсмита — Винограда

Трудоёмкость алгоритма Копперсмита — Винограда состоит из:

1) создания векторов размерами N и M.

$$f_{create} = N + M; \tag{2.6}$$

2) заполнения вектора N.

$$f_{rows} = 3 + \frac{Q}{2} \cdot (5 + 12N); \tag{2.7}$$

3) заполнения вектора cols.

$$f_{cols} = 3 + \frac{Q}{2} \cdot (5 + 12M); \tag{2.8}$$

4) цикла заполнения матрицы для чётных размеров.

$$f_{cycle} = 2 + N \cdot (4 + M \cdot (11 + \frac{25}{2} \cdot Q)); \tag{2.9}$$

5) цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный.

$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + N \cdot (4 + 14M), & \text{иначе.} \end{cases} \tag{2.10}$$

Итого, для худшего случая (нечётный размер матриц):

$$f_{wino\_w} = N + M + 12 + 8N + 5Q + 6NQ + 6MQ + 25NM + \frac{25}{2}NMQ \approx 12.5 \cdot NMQ \tag{2.11}$$

Для лучшего случая (чётный размер матриц):

$$f_{wino\_b} = N + M + 10 + 4N + 5Q + 6NQ + 6MQ + 11NM + \frac{25}{2}NMQ \approx 12.5 \cdot NMQ \tag{2.12}$$

2.2.3 Оптимизированный алгоритм Копперсмита — Винограда

Оптимизации алгоритма:

1) операции вида $x = x + k$ заменены на $x+ = k$;

2) умножение и деление на 2 заменено побитовым сдвигом;

3) происходит предвычисление некоторых слагаемых алгоритма.

Трудоёмкость улучшенного алгоритма Копперсмита — Винограда состоит из:

1) создания векторов N и M.
$$f_{init} = N + M; \tag{2.13}$$

2) заполнения вектора N.
$$f_{rows} = 2 + \frac{Q}{2} \cdot (4 + 8N); \tag{2.14}$$

3) заполнения вектора M.
$$f_{cols} = 2 + \frac{Q}{2} \cdot (4 + 8M); \tag{2.15}$$

4) цикла заполнения матрицы для чётных размеров.
$$f_{cycle} = 2 + N \cdot (4 + M \cdot (8 + 9Q)) \tag{2.16}$$

5) цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный.
$$f_{last} = \begin{cases} 2, & \text{чётная,} \\ 4 + N \cdot (4 + 12M), & \text{иначе.} \end{cases} \tag{2.17}$$

Итого, для худшего случая (нечётный общий размер матриц) имеем:
$$f = N + M + 10 + 4Q + 4QM + 4QN + 8N + 20NM + 9ABC \approx 9NMQ \tag{2.18}$$

Для лучшего случая (чётный общий размер матриц) имеем:
$$f = N + M + 8 + 4Q + 4QM + 4QN + 4N + 8NM + 9NMQ \approx 9NMQ \tag{2.19}$$

2.3 Описания алгоритмов

На рисунках ниже показаны схемы алгоритмов простого умножения матриц, алгоритмом Винограда и оптимизированного Винограда.

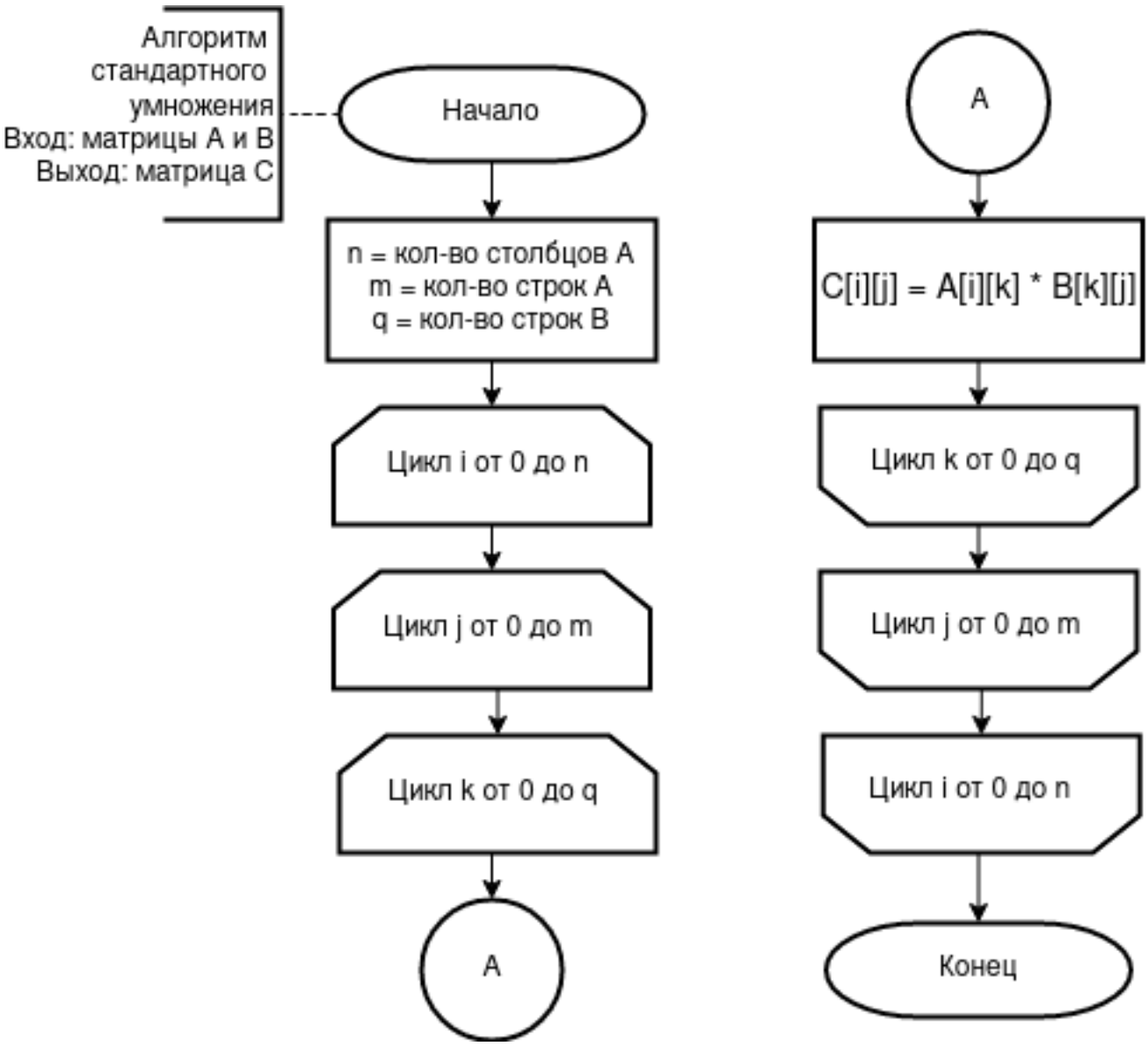

Рисунок 2.1 – Схема классического алгоритма умножения матриц

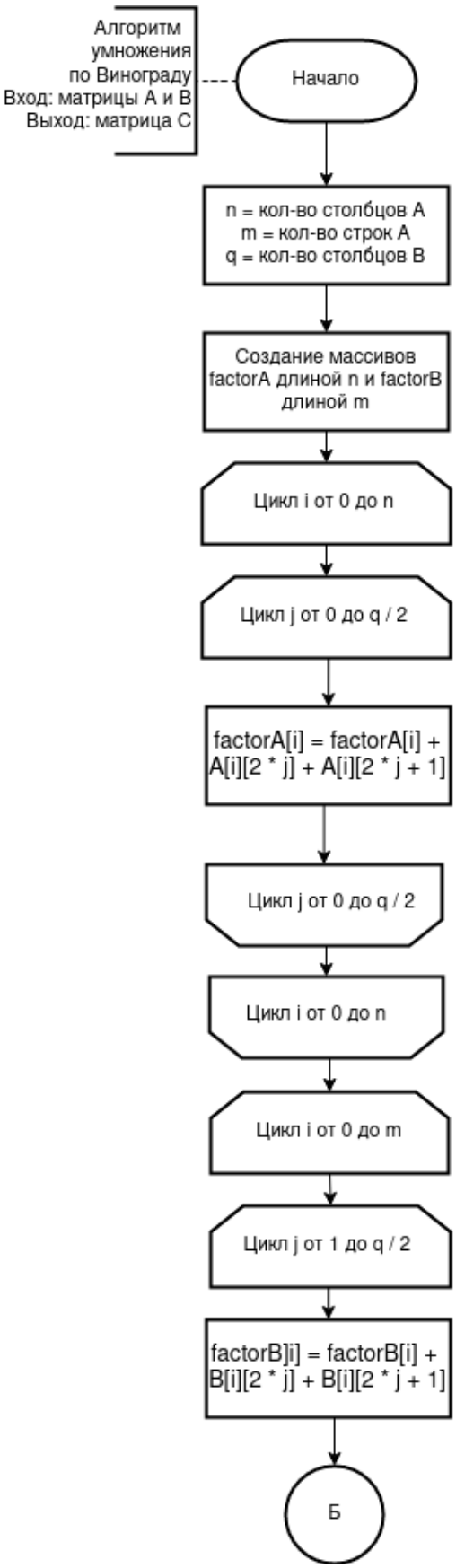

Рисунок 2.2 – Схема умножения матриц алгоритмом Винограда, ч.1

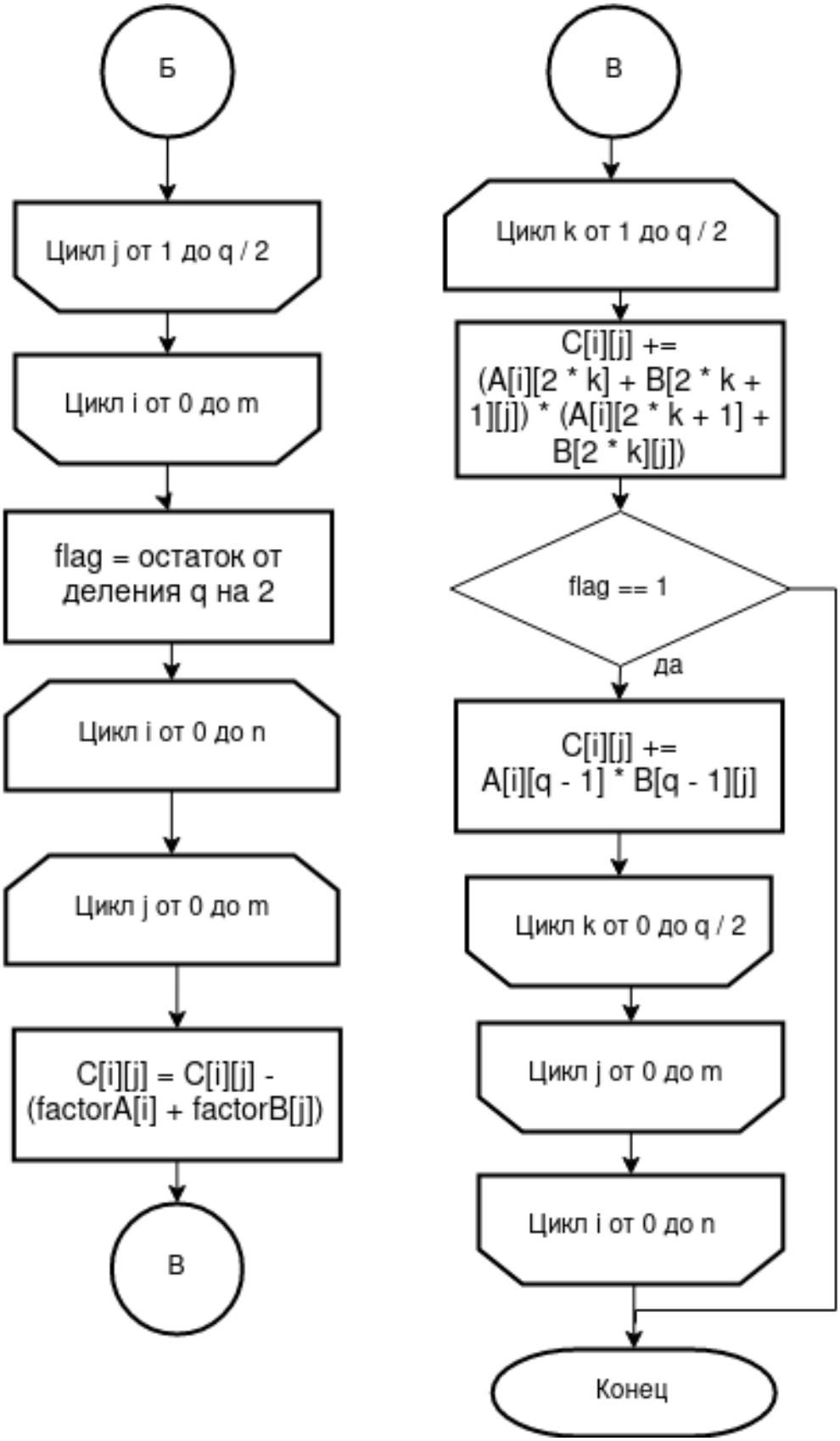

Рисунок 2.3 – Схема умножения матриц алгоритмом Винограда, ч.2

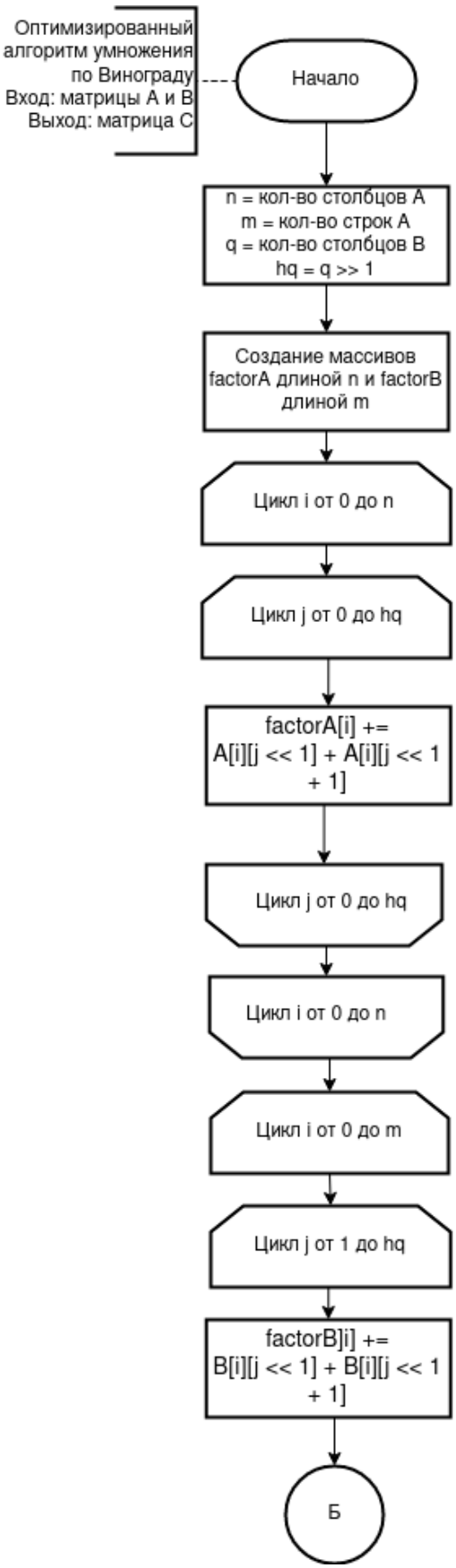

Рисунок 2.4 – Схема умножения матриц оптимизированным алгоритмом Винограда, ч.1

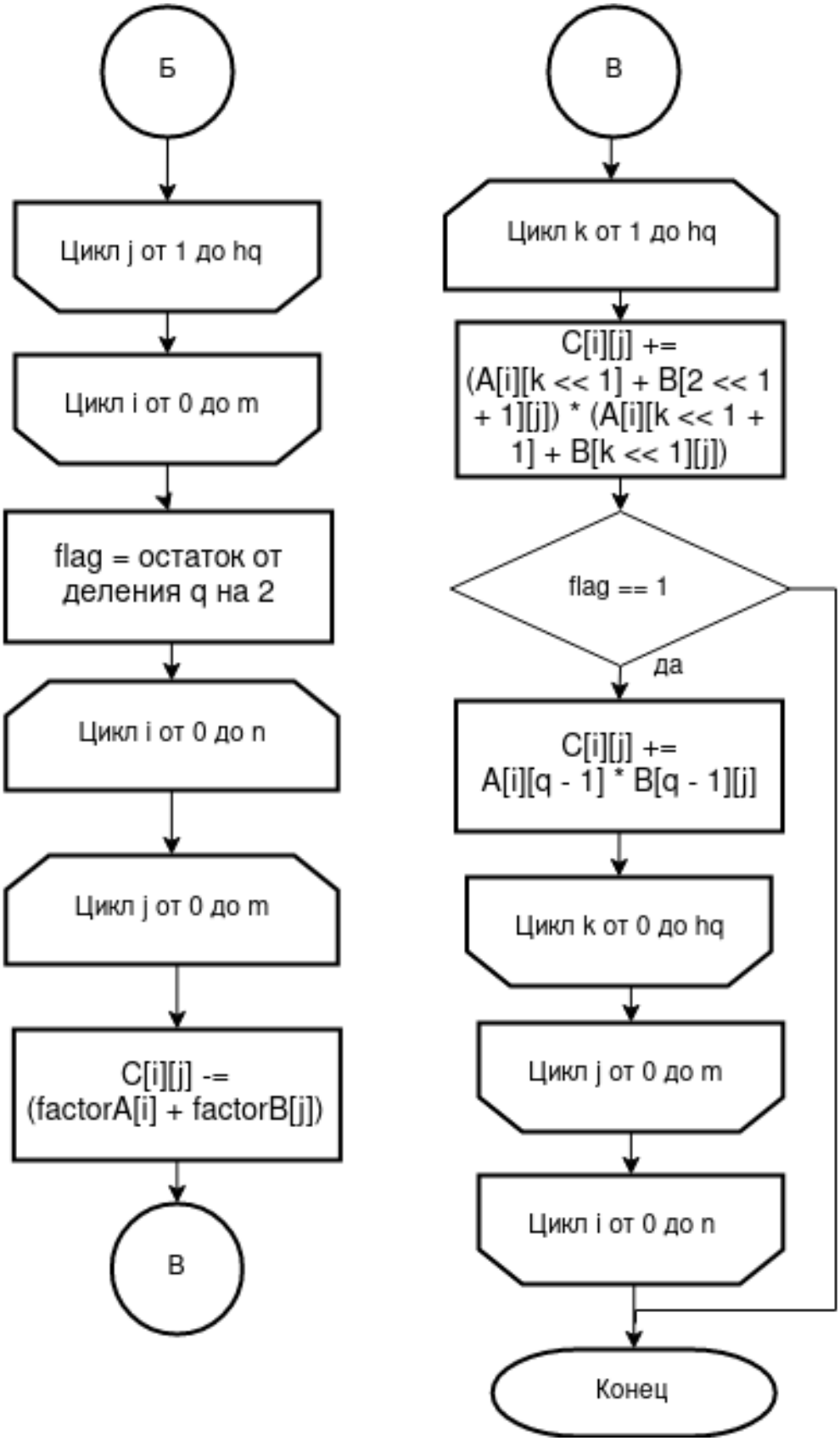

Рисунок 2.5 – Схема умножения матриц оптимизированным алгоритмом Винограда, ч.2

Вывод

В данном разделе были представлены описания умножения матриц классическим алгоритмом, алгоритмом Винограда и оптимизированным алгоритмом Винограда, а так же проведена оценка их трудоемкости.

Глава 3

Технологическая часть

В данном разделе приведены требования к ПО, обоснования выбора языка программирования, среды разработки, приведен способ замера времени выполнения, а также приведены листинги кода.

3.1 Требования к ПО

В программе должна присутствовать возможность:

1) ввода исходных матриц, участвующих в умножении;
2) поиска матрицы, полученной в результате умножения, с помощью одного из трех рассматриваемых алгоритмов;
3) замера процессорного времени выполнения реализаций алгоритмов.

3.2 Выбор языка программирования и среды разработки

Для реализации трех алгоритмов сортировок был выбор язык C, так как данный язык является быстродейственным.

Средой разработки был выбран CLion. Данный выбор обусловлен тем, что данная среда предоставляет возможность разработки приложений под C/C++ и имеет инструменты для отладки кода.

3.3 Выбор библиотеки и способа для замера времени

Для замера времени выполнения сортировок использовалась стандартная функция библиотеки <time.h> языка C — clock(), которая замеряет процессорное время. Если измерить время перед началом выполнения алгоритма, и после его окончания, то можно получить время выполнения функции. Реализация данной функции приведена в [1].

Поскольку все процессорное время не отдается какой-либо одной задаче (в связи с явлением вытеснения процессов из ядра, квантование процессорного времени), то требуется усреднить результаты вычислений: замерить совокупное время выполнения реализации алгоритма N раз и вычислить среднее время выполнения.

3.4 Реализации алгоритмов

В листингах 3.1,3.2,3.3 приведены реализации алгоритмов умножения матриц классическим методом, Винограда и оптимизорованного Винограда соответсвенно.

Листинг 3.1 – Листинг алгоритма классического умножения матриц

```
matrix_t *mul_matrix_def(matrix_t *first_mat, matrix_t *second_mat)
{
    matrix_t *res = create_matrix(first_mat->rows, second_mat->cols);

    int eq_dim = first_mat->cols;

    for (int i = 0; i < res->rows; ++i) {
  for (int j = 0; j < res->cols; ++j) {
    (res->elements)[i][j] = 0;
    for (int k = 0; k < eq_dim; ++k) {
  (res->elements)[i][j] += (first_mat->elements)[i][k] * (second_mat->
     elements)[k][j];
    }
  }
    }

    return res;
}
```

Листинг 3.2 – Листинг алгоритма Винограда умножения матриц

```
matrix_t *mul_matrix_vinograd(matrix_t *first_mat, matrix_t *second_mat)
{
    matrix_t *res = create_matrix(first_mat->rows, second_mat->cols);
    int *factorA = calloc(first_mat->rows, sizeof(int));
    int *factorB = calloc(second_mat->cols, sizeof(int));

    int eq_dim = first_mat->cols;

    for (int i = 0; i < first_mat->rows; ++i)
  for (int j = 0; j < eq_dim / 2; ++j)
    factorA[i] = factorA[i] + (first_mat->elements)[i][2 * j] * (
       first_mat->elements)[i][2 * j + 1];

    for (int i = 0; i < second_mat->cols; ++i)
  for (int j = 0; j < eq_dim / 2; ++j)
    factorB[i] = factorB[i] + (second_mat->elements)[2 * j][i] * (
       second_mat->elements)[2 * j + 1][i];

    for (int i = 0; i < res->rows; ++i) {
  for (int j = 0; j < res->cols; ++j) {
    (res->elements)[i][j] = -factorA[i] - factorB[j];
    for (int k = 0; k < eq_dim / 2; ++k) {
  (res->elements)[i][j] = (res->elements)[i][j] + ((first_mat->
     elements)[i][2 * k] + (second_mat->elements)[2 * k + 1][j]) * ((
```

```
            first_mat->elements)[i][2 * k + 1] + (second_mat->elements)[2 * k
            ][j]);
22          }
23      }
24        }
25
26        if (eq_dim % 2 != 0) {
27      for (int i = 0; i < res->rows; ++i)
28          for (int j = 0; j < res->cols; ++j)
29        (res->elements)[i][j] = (res->elements)[i][j] + (first_mat->elements
              )[i][first_mat->cols - 1] * (second_mat->elements)[second_mat->
              rows - 1][j];
30        }
31
32        free(factorA);
33        free(factorB);
34
35        return res;
36    }
```

Листинг 3.3 – Листинг оптимизированного алгоритма Винограда умножения матриц

```
1     matrix_t *mul_matrix_vinograd_optimized(matrix_t *first_mat, matrix_t *
          second_mat)
2     {
3         matrix_t *res = create_matrix(first_mat->rows, second_mat->cols);
4         int *factorA = calloc(first_mat->rows, sizeof(int));
5         int *factorB = calloc(second_mat->cols, sizeof(int));
6
7         int eq_dim = first_mat->cols;
8         int half_eq_dim = eq_dim >> 1;
9
10        for (int i = 0; i < first_mat->rows; ++i)
11      for (int j = 0; j < half_eq_dim; ++j)
12          factorA[i] += (first_mat->elements)[i][j << 1] * (first_mat->
                elements)[i][(j << 1) + 1];
13
14        for (int i = 0; i < second_mat->cols; ++i)
15      for (int j = 0; j < half_eq_dim; ++j)
16          factorB[i] += (second_mat->elements)[j << 1][i] * (second_mat->
                elements)[(j << 1) + 1][i];
17
18        for (int i = 0; i < res->rows; ++i) {
19      for (int j = 0; j < res->cols; ++j) {
20          (res->elements)[i][j] = -factorA[i] - factorB[j];
21          for (int k = 0; k < half_eq_dim; ++k) {
22        (res->elements)[i][j] += ((first_mat->elements)[i][k << 1] + (
              second_mat->elements)[(k << 1) + 1][j]) * ((first_mat->elements)[
              i][(k << 1) + 1] + (second_mat->elements)[(k << 1)][j]);
23          }
```

```
24          }
25            }
26
27            if (eq_dim % 2 != 0) {
28          for (int i = 0; i < res->rows; ++i)
29              for (int j = 0; j < res->cols; ++j)
30            (res->elements)[i][j] += (first_mat->elements)[i][first_mat->cols -
                1] * (second_mat->elements)[second_mat->rows - 1][j];
31            }
32
33            free(factorA);
34            free(factorB);
35
36            return res;
37        }
```

3.5 Тестирование алгоритмов

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы классического умножения матриц, умножения алгоритмом Винограда и оптимизированного Винограда. Все тесты пройдены успешно.

Таблица 3.1 – Тестирование трех рассматриваемых алгоритмов умножения матриц для разных входный данных

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	$\begin{pmatrix} 22 & 28 \\ 22 & 28 \end{pmatrix}$
(2)	(3)	(6)
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 4 & 8 & 12 \\ 4 & 8 & 12 \\ 4 & 8 & 12 \end{pmatrix}$

Вывод

В данном разделе были разработаны исходные коды алгоритмов классического умножения матриц, умножения алгоритмом Винограда и оптимизированного Винограда.

Глава 4

Экспериментальная часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

1) операционная система Windows-10, 64-bit;
2) оперативная память 16 ГБ;
3) процессор Intel(R) Core(TM) i7-8565U CPU @ 1.80 ГГц, 4 ядра, 8 логических процессоров.

4.2 Замеры времени

Введем обозначения: Def – классический алгоритм умножения матриц, Vin – умножение матриц алгоритмом Винограда, OpVin – умножение матриц оптимизированным алгоритмом Винограда. Все рассматриваемые в этом разделе матрицы – квадратные.

В таблице 4.1 приведены результаты замеров времени алгоритмов для входных матриц, имеющих различные четные размеры.

Таблица 4.1 – Таблица замера времени выполнения алгоритмов на входных матрицах, имеющих четные размеры

Количество строк и столбцов	Def	Vin	OpVin
50	835	623	536
100	5678	4446	4155
200	45255	41572	34487
300	155263	129265	126086

Зависимость времени работы алгоритмов умножения матриц от четных размеров умножаемых матриц представлена на рис. 4.1.

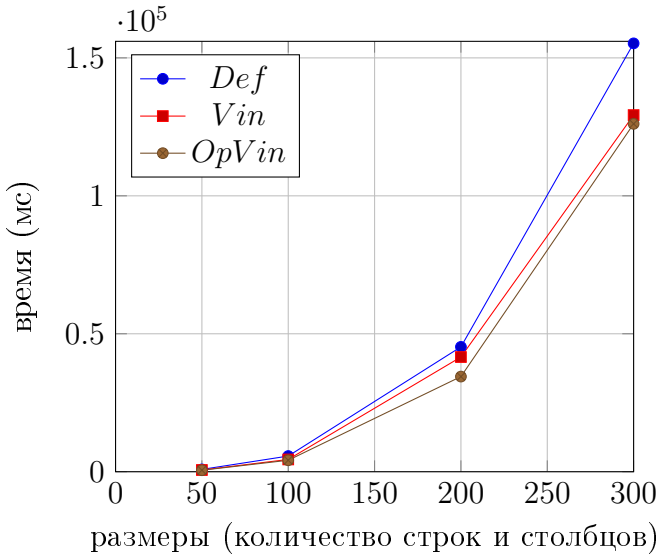

Рисунок 4.1 – Зависимость времени от размера входных матриц

В таблице 4.2 приведены результаты замеров времени алгоритмов для входных матриц, имеющих различные нечетные размеры.

Таблица 4.2 – Таблица замера времени выполнения алгоритмов на входных матрицах, имеющих нечетные размеры

Количество строк и столбцов	Def	Vin	OpVin
51	695	614	574
101	5928	4484	4324
201	44531	36619	34975
301	158599	135065	133109

Зависимость времени работы алгоритмов умножения матриц от нечетных размеров умножаемых матриц представлена на рис. 4.2.

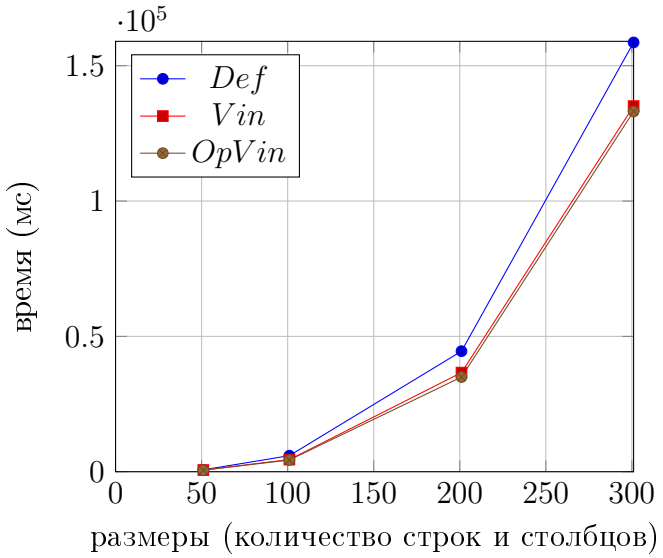

Рисунок 4.2 – Зависимость времени от размера входных матриц

Вывод

Результаты замеров показали, что алгоритм Винограда и оптимизированный алгоритм Винограда работают быстрее классического алгоритма умножения матриц за счет того, что в них меньше операций умножения. При этом оптимизированный алгоритм Винограда работает немного быстрее обычного алгоритма Винограда за счет введеных оптимизаций.

Заключение

Цель лабораторной работы достигнута – были изучены и реализованы алгоритмы умножения матриц классическим методом, Виноградом и оптимизированным Виноградом. Все задачи решены:

1) были изучены и рассмотрены алгоритмы классического умножения матриц, алгоритм Винограда и модифицированный алгоритм Винограда;
2) были построены блок-схемы данных алгоритмов;
3) был реализован каждый из алгоритмов;
4) была оценена их трудоемкость;
5) были экспериментально получены временные характеристики алгоритмов;
6) были сделаны выводы на основании проделанной работы

На основании проведенных экспериментов и оценки трудоемкости было определено, что оптимизированный алгоритм Винограда умножения матриц имеет меньшую сложность, нежели классический алгоритм. При этом алгоритм Винограда обгоняет простой алгоритм в среднем в 1.3 раза, а оптимизированный алгоритм Винограда – в 1.6 раз. При этом для обоих алгоритмов Винограда требуется выделение дополнительной памяти.

Список использованных источников

[1] The Open Group Base Time [Электронный ресурс]. - URL: https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/time.h.html (дата обращения: 23.10.2022).

[2] Алгоритм Копперсмита — Винограда [Электронный ресурс]. - URL: http://ru.math.wikia.com/wiki/Algorythm_Coppersmita_Vinograda (дата обращения: 23.10.2022).

[3] Реализация алгоритма умножения матриц по Винограду на языке Haskell. анисимов Н.С, Строганов Ю.В. [Электронный ресурс]. - URL: https://cyberleninka.ru/article/n/realizatsiya-algoritma-umnozheniya-matrits-po-vinogradu-na-yazyke-haskell/viewer) (дата обращения: 23.10.2022).