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»

Лабораторная работа №2
по дисциплине "Анализ Алгоритмов"

Тема Умножение матриц

Студент Рядинский К. В.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение

Алгоритм Копперсмита - Винограда — алгоритм умножение квадратных матриц, предложенный в 1987 году Д. Копперсмитом и Ш. Виноградом. В исходной версии асимптотическая сложность алгоритма составляла $O(n^{2.3755})$, где n — размер стороны матрицы. Алгоритм Копперсмита – Винограда, с учетом серии улучшений и доработок в последующие годы, обладает лучшей асимптотикой среди известных алгоритмов умножения матриц.

На практике алгоритм Копперсмита — Винограда не используется, так как он имеет очень большую константу пропорциональности и начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров. Поэтому пользуются алгоритмом Штрассена по причинам простоты реализации и меньшей константе в оценке трудоемкости.

Алгоритм Штрассена предназначен для быстрого умножения матриц. Он был разработан Фолькером Штрассеном в 1969 году и является обобщением метода умножения Карацубы на матрицы.

В отличие от традиционного алгоритма множения матриц, алгоритм Штрассена умножает матрицы за время $\Theta(n^{\log_2 7}) = O(n^{2.81})$

Несмотря на то, что алгоритм Штрассена является асимптотически не самым быстрым из существующих алгоритмов быстрого умножения матриц, он проще программируется и эффективнее при умножении матриц относительно малого размера.

Задачи лабораторной работы:

— реализовать классический алгоритм умножения матриц;
— реализовать алгоритм Копперсмита — Винограда;
— реализовать улучшенный Алгоритм Копперсмита – Винограда;
— рассчитать их трудоемкость;
— сравнить их временные характеристики экспериментально;
— на основании проделанной работы сделать выводы.

1 Аналитическая часть

В данном разделе будут рассмотрены алгоритмы умножения матриц

1.1 Стандартный алгоритм

Пусть даны две прямоугольные матрицы

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l1} & a_{l2} & \cdots & a_{lm} \end{pmatrix}, \tag{1}$$

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}. \tag{2}$$

Тогда матрица C размерностью $l \times n$

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l1} & c_{l2} & \cdots & c_{ln} \end{pmatrix}, \tag{3}$$

в которой:

$$c_{ij} = \sum_{r=1}^{m} a_{ir} b_{rj}, \quad (i = \overline{1,l}; j = \overline{1,n}) \tag{4}$$

будет называться произведением матриц A и B. Стандартный алгоритм реализует данную формулу.

1.2 Алгоритм Копперсмита – Винограда

В результате умножения двух матриц, каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить, что такое умножение допускает предварительную обработку, позволяющую часть работы 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 равно: $V \cdot W = v_1 w_1 + \cdots + v_4 w_4$, что эквивалентно

$$V \cdot W = (v_1 + w_1)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \quad (5)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырех умножений - шесть, а вместо трех сложений - десять, выражение в правой части последнего равенства допускает предварительную обработку: его части можно вычислить заранее и запомнить для каждой строки первой матрицы и для каждого столбца второй, то для каждого элемента будет необходимо выполнить лишь первые два умножения и последующие пять сложений, а также дополнительно два сложения. Из-за того, что операция сложения быстрее операции умножения, алгоритм должен работать быстрее стандартного

1.3 Вывод

Были рассмотрены алгоритмы классического умножения матриц и алгоритм Винограда, основное отличие которых — наличие предварительной обработки, а также количество операций умножения.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 1-3 приведены схемы алгоритмов простого умножения матриц, умножения матриц по Копперсмиту–Винограду и улучшенного умножения матриц по Копперсмиту–Винограду соответственно.

Для алгоритма Винограда худшим случаем являются матрицы с нечётным общим размером, а лучшим - с чётным, так как отпадает необходимость в последнем цикле.

Данный алгоритм можно оптимизировать:

1. Убрать деления в цикле.
2. Замена выражения $a = a + \cdots$ на $a+ = \cdots$.
3. Увеличить шаг в цикле до 2.

2.2 Трудоемкость алгоритмов

Для последующего вычисления трудоемкости необходимо ввести модель вычислений.

1. $+, -, /, \%, =, \neq, <, >, \leq, \geq, [], *, ++$ — трудоемкость 1.
2. Трудоемкость оператора выбора *if* условие *then A else B* рассчитывается, как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A & \text{если условие выполняется,} \\ f_B & \text{иначе.} \end{cases} \tag{6}$$

3. Трудоемкость цикла расчитывается, как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}}). \tag{7}$$

4. Трудоемкость вызова функции равна 0.

2.2.1 Классический алгоритм

Трудоемкость классического алгоритма:

$10MNQ + 4MQ + 4M + 2$

2.2.2 Алгоритм Винограда

Трудоемкость алгоритма Винограда:

Первый цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

Второй цикл: $\frac{15}{2} \cdot MN + 5 \cdot M + 2$

Третий цикл: $13 \cdot MNQ + 12 \cdot MQ + 4 \cdot M + 2$

Условный переход:

$$\begin{cases} 2 & \text{,если размер матрицы нечетный} \\ 15 \cdot QM + 4 \cdot M + 2 & \text{,иначе} \end{cases}$$

Итого:

$$\frac{15}{2} \cdot MN + 5M + 2 + \frac{15}{2} \cdot MN + 5M + 2 + 13MNQ +$$
$$+ 12MQ + 4M + 2 + \begin{cases} 2 & \text{,если размер матрицы нечетный} \\ 15 \cdot MQ + 4M + 2 & \text{,иначе.} \end{cases}$$

2.3 Оптимизированный алгоритм Винограда

Рассмотрим трудоемкость оптимизированного алгоритма Винограда:

Первый цикл: $\frac{11}{2} \cdot MN + 4M + 2$

Второй цикл: $\frac{11}{2} \cdot MN + 4M + 2$

Третий цикл: $\frac{15}{2} \cdot MNQ + 9MQ + 4M + 2$

Условный переход: $\begin{cases} 1 & \text{, если размер матрицы нечетный} \\ 10MQ + 4M + 2 & \text{, иначе.} \end{cases}$

Итого: $\frac{11}{2} \cdot MN + 4M + 2 + \frac{11}{2}MN + 4M + 2 + \frac{15}{2}MNQ + 9MQ + 4M + 2 +$
$\begin{cases} 1 & \text{, если размер матрицы нечетный} \\ 10MQ + 4M + 2 & \text{, иначе.} \end{cases}$

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы требуемых алгоритмов и проведена теоретическая оценка трудоемкости.

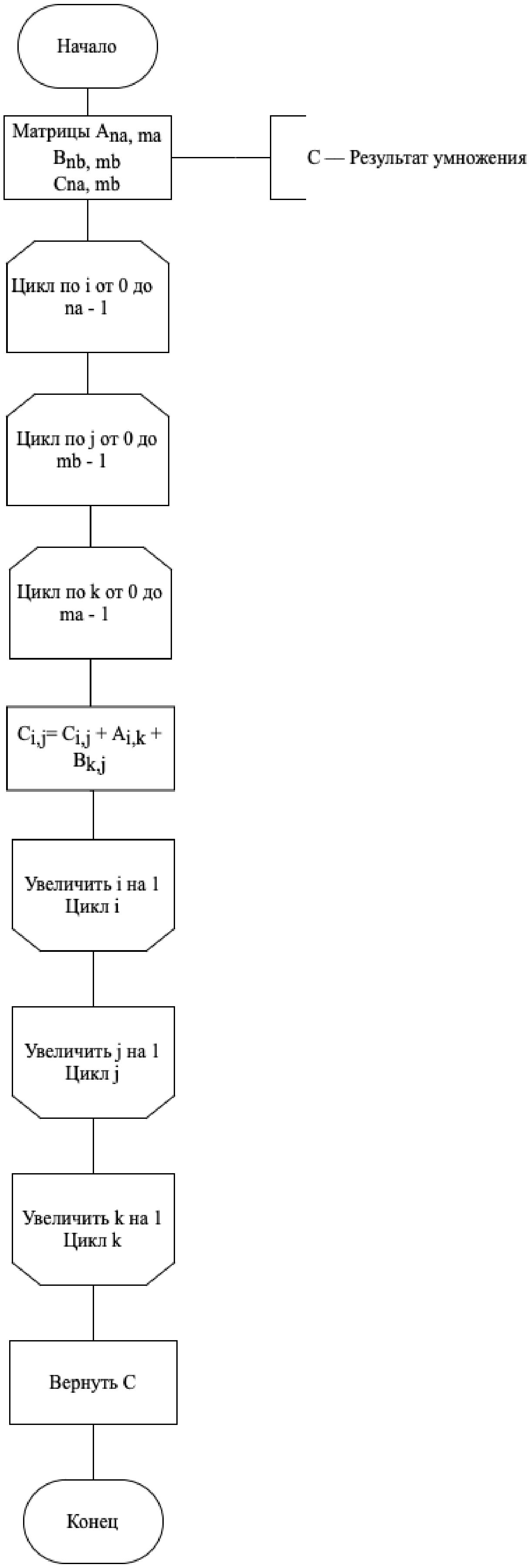

Рис. 1 — Схема алгоритма простого умножения матриц

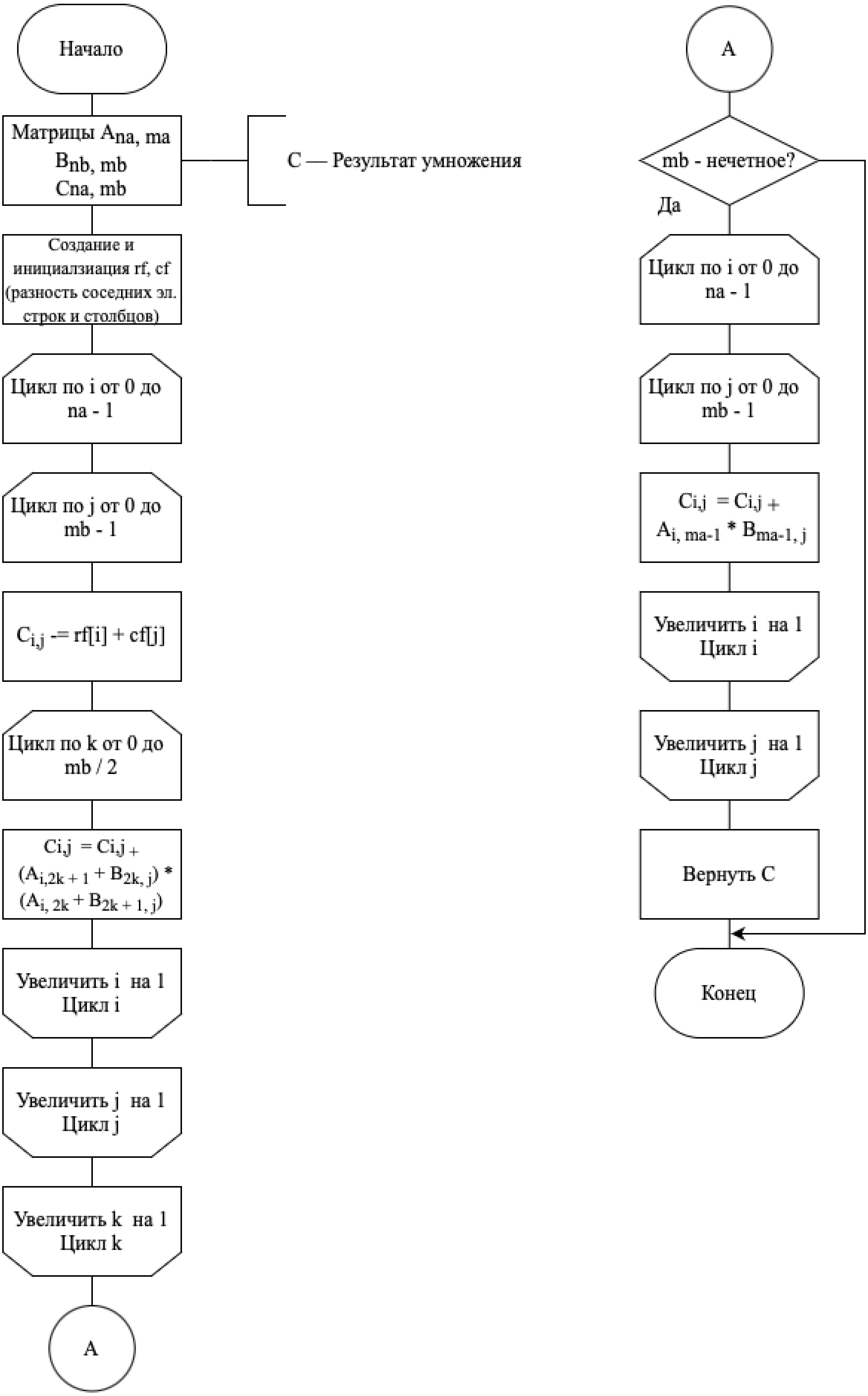

Рис. 2 — Схема алгоритма Винограда умножения матриц

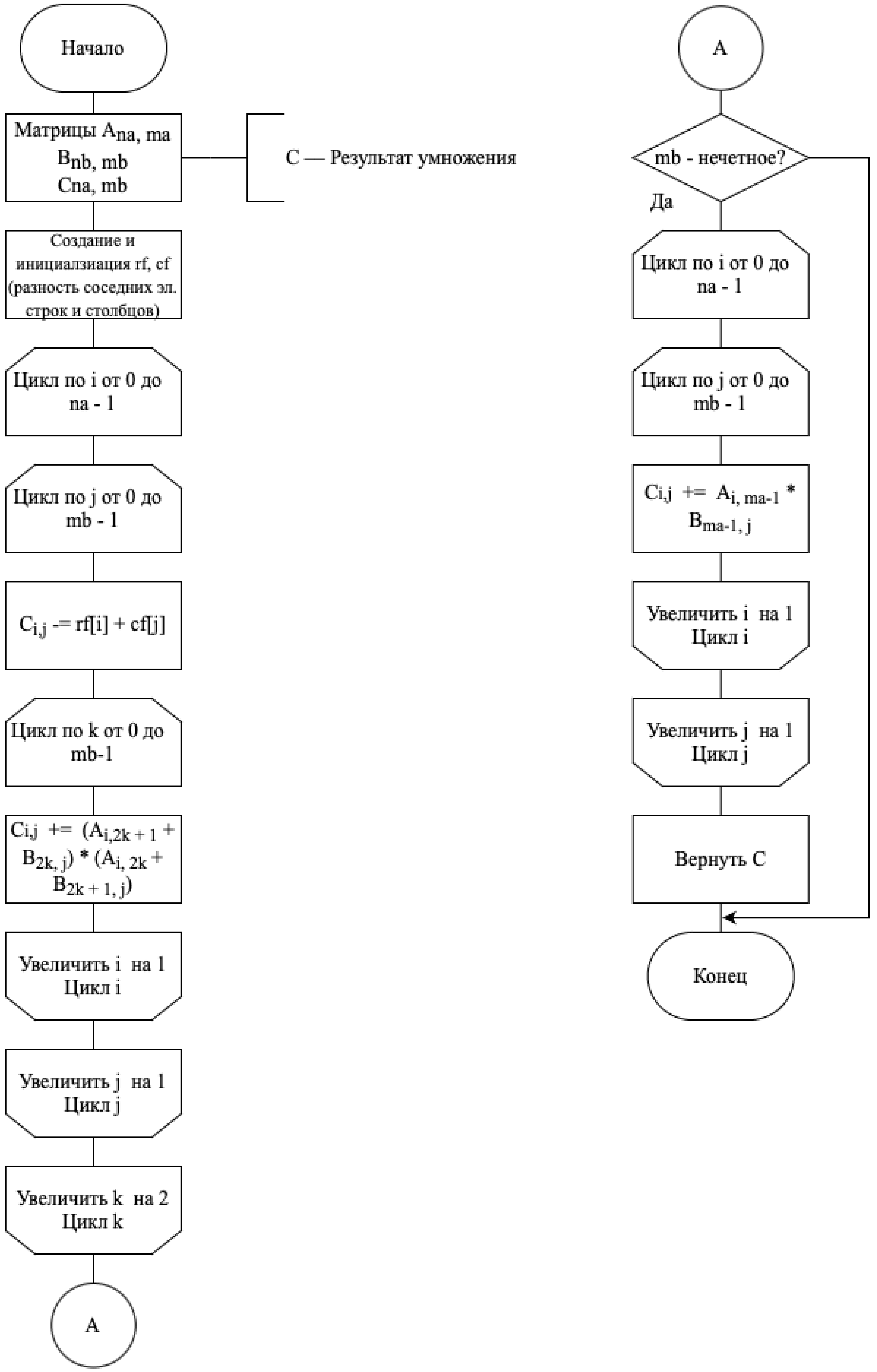

Рис. 3 — Схема оптимизированного алгоритма Винограда умножения матриц

3 Технологическая часть

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

3.1 Требования к программному обеспечению

К программе предъявляется ряд требований:
- на вход подаются размеры матриц (натуральные числа) и самы матрицы, которые нужно перемножить;
- на выходе — результаты умножения матриц алгоритмами простого умножения матриц, умножения матриц по Копперсмиту–Винограду и улучшенного умножения матриц по Копперсмиту–Винограду.

3.2 Средства реализации

Для реализации программ был выбран язык программирования Rust [1], так как этот язык предоставляет как низкоуровневые интерфейсы, так и высокоуровневые, этот язык безопасен при работе с памятью. Также данный язык был выбран потому, что в нем присутствует инструментарий для замера процессорного времени и тестирования.

3.3 Листинги кода

Листинг 1: Алгоритм простого умножения матриц

```
pub fn default_mult(m1 : & Matrix<T>, m2 : & Matrix<T>) ->
    Result<Matrix<T>, &'static str> {
    if m1.col != m2.rows {
        return Err("The number of columns must be equal to
    number of rows");
    }
    let mut out = Matrix::new_zero(m1.rows, m2.col);
    for i in 0..m1.rows {
        for j in 0..m2.col {
            for k in 0..m1.col {
```

```
                out[[i, j]] += m1[[i, k]] * m2[[k, j]];
            }
        }
    }
    Ok(out)
}
```

Листинг 2: Алгоритм умножения матриц Винограда

```
pub fn vinograd_mult(m1 : &Matrix<T>, m2 : &Matrix<T>) ->
    Result<Matrix<T>, &'static str> {
    if m1.rows != m1.col || m2.col != m2.rows || m1.col != m2
    .col {
        return Err("Matrices must be square and have equal
    size");
    }

    let mut out : Matrix<T> = Matrix::new_zero(m1.rows, m2.
    col);

    let mut row_factor : Vec<T> = vec![Default::default(); m1
    .rows];
    let mut col_factor : Vec<T> = vec![Default::default(); m2
    .col];

    for i in 0..m1.rows {
        for j in 0..m1.col / 2 {
            row_factor[i] += m1[[i, j * 2]] * m1[[i, j * 2 +
    1]]
        }
    }

    for i in 0..m2.col {
        for j in 0..m2.rows / 2 {
            col_factor[i] += m2[[j * 2, i]] * m2[[j * 2 + 1,
    i]]
        }
    }

    for i in 0..m1.rows {
        for j in 0..m2.col {
```

```
                out[[i, j]] -= row_factor[i] + col_factor[j];

                for k in 0..m1.col / 2 {
                    out[[i, j]] += (m1[[i, 2 * k + 1]] + m2[[2 * k, j]]) * (m1[[i, 2 * k]] + m2[[2 * k + 1, j]]);
                }
            }
        }

        if m1.col % 2 > 0{
            for i in 0..m1.rows {
                for j in 0..m2.col {
                    out[[i, j]] += m1[[i, m1.col - 1]] * m2[[m1.col - 1, j]];
                }
            }
        }

        Ok(out)
}
```

Листинг 3: Оптимизированный алгоритм умножения матриц Винограда

```
pub fn vinograd_opt_mult(m1 : & Matrix<T>, m2 : & Matrix<T>) -> Result<Matrix<T>, &'static str> {
    if m1.rows != m1.col || m2.col != m2.rows || m1.col != m2.col {
        return Err("Matrices must be square and have equal size");
    }

    let mut out : Matrix<T> = Matrix::new_zero(m1.rows, m2.col);

    let mut row_factor : Vec<T> = vec![Default::default(); m1.rows];
    let mut col_factor : Vec<T> = vec![Default::default(); m2.col];

    let m1_col_mod = m1.col % 2;
    let m2_row_mod = m2.rows % 2;
```

```

    for i in 0..m1.rows {
        for j in (0..m1.col - m1_col_mod).step_by(2) {
            row_factor[i] += m1[[i, j]] * m1[[i, j + 1]]
        }
    }

    for i in 0..m2.col {
        for j in (0..m2.rows - m2_row_mod).step_by(2) {
            col_factor[i] += m2[[j, i]] * m2[[j + 1, i]]
        }
    }

    for i in 0..m1.rows {
        for j in 0..m2.col {
            let mut buf : T = -(row_factor[i] + col_factor[j]);

            for k in (0..m1.col - m1_col_mod).step_by(2) {
                buf += (m1[[i, k + 1]] + m2[[k, j]]) * (m1[[i, k]] + m2[[k + 1, j]]);
            }

            out[[i, j]] = buf;
        }
    }

    if m1.col % 2 > 0 {
        let m1c = m1.col - 1;
        for i in 0..m1.rows {
            for j in 0..m2.col {
                out[[i, j]] += m1[[i, m1c]] * m2[[m1c, j]];
            }
        }
    }

    Ok(out)
}
```

3.4 Тестирование функций

В таблице 1 приведены модульные тесты для функций умножения матриц выше перечисленными методами. Все тесты были пройдены успешно.

Таблица 1 — Тестирование функций умножения матриц

Матрица 1	Матрица 2	Ожидаемый результат
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$	$\begin{pmatrix} 9 & 12 \\ 24 & 33 \end{pmatrix}$
$\begin{pmatrix} 8 \end{pmatrix}$	$\begin{pmatrix} 4 \end{pmatrix}$	$\begin{pmatrix} 32 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 \end{pmatrix}$	$\begin{pmatrix} 3 & 4 \end{pmatrix}$	Умножение невозможно

3.5 Вывод

Были разработаны и протестированы реализации алгоритмов: простой алгоритм умножения матриц, алгоритм умножения матриц по Копперсмиту – Винограду и улучшенный алгоритм умножения матриц по Копперсмиту – Винограду.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики электронно-вычислительнй машины, на которой выполнялось тестирование:

— операционная система: macOS BigSur версия 11.4;

— оперативная память: 8 гигабайт LPDDR4 [3];

— процессор: Apple M1.

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Был проведен замер времени работы каждого из алгоритмов с помощью библиотеки Criterion [2]. Эта библиотека замеряет процессорное время выполнения функции и усредняет его (проводится не менее 100 замеров). В таблицах 2, 3 содержатся результаты исследований при четном и нечетном размерах матриц.

На рисунках 4, 5 демонстрируется зависимость времени выполнения конкретных реалзиаций алгоритмов умножения матриц от размера стороны квадратной матрицы.

Таблица 2 — Время выполнения реализаций алгоритмов (в секундах) при четном размере матрицы

Размер	К	В	ОВ
100	0.0738	0.0591	0.0540
200	0.589	0.482	0.422
400	4,749	3,8028	3,3414
800	38,577	30,073	26,665

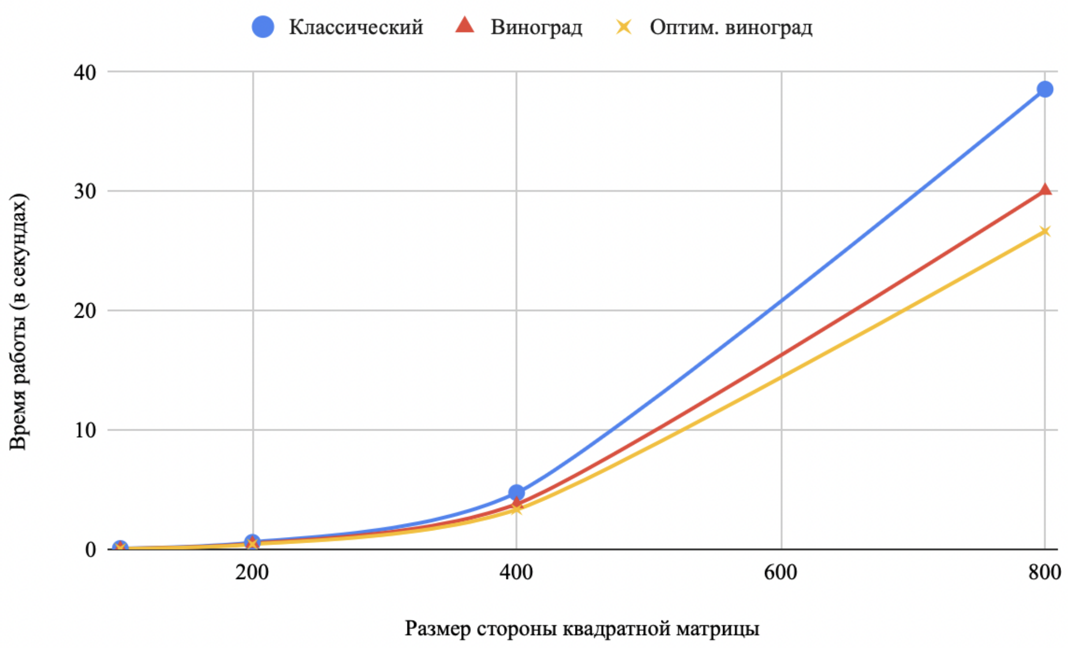

Рис. 4 — Зависимость времени выполнения алгоритмов от четного размера стороны квадратной матрицы

Таблица 3 — Время выполнения реализаций алгоритмов (в секундах) при нечетном размере матрицы

Размер	К	В	ОВ
101	0.0741E	0.0616	0.0551
201	0.590	0.475	0.429
401	4,7312	3,7858	3,3499
801	38,591	30,11	26,655

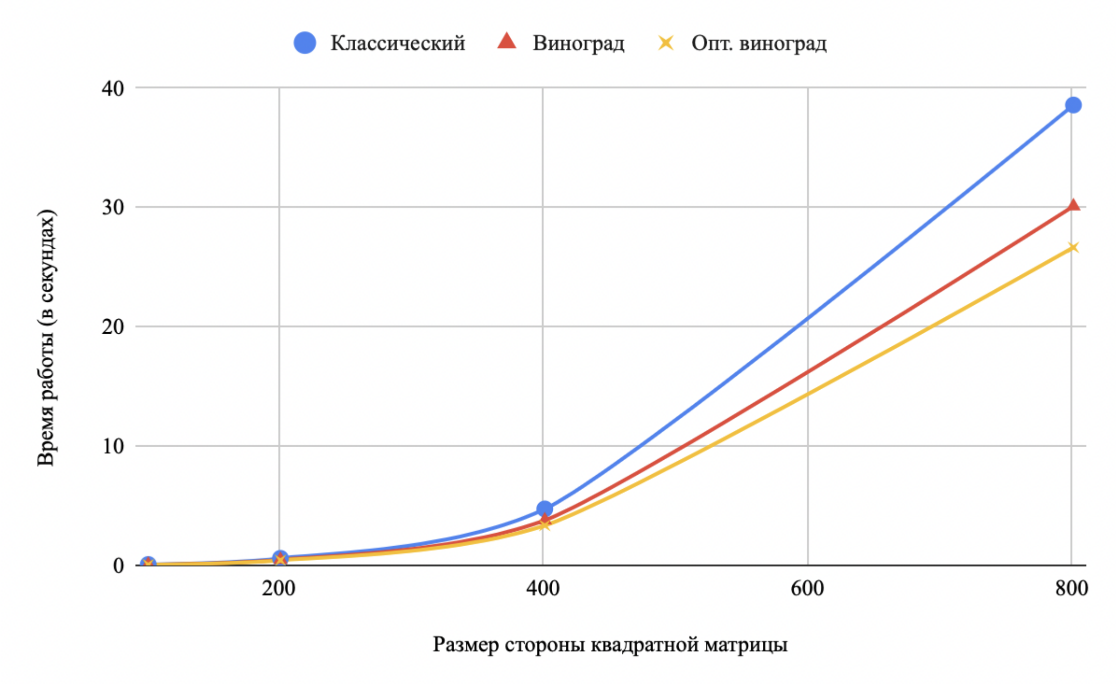

Рис. 5 — Зависимость времени выполнения алгоритмов от нечетного размера стороны квадратной матрицы

4.3 Вывод

Время работы реализации алгоритма Копперсмита–Винограда быстрее классического алгоритма умножения матриц примерно на 25-30% быстрее. В то же время оптимизированный алгоритм Копперсмита–Винограда быстрее оригинального на 10-15%. Таким образом на матрицах значительного размера (больше 200) следует использовать алгоритм Копперсмита–Винограда, так как он значительно быстрее (таблица 2).

Заключение

В ходе выполнения работы была достигнута цель выполнены все поставленные задачи:

- реализовать классический алгоритм умножения матриц;
- реализовать алгоритм Копперсмита — Винограда;
- реализовать улучшенный Алгоритм Копперсмита — Винограда;
- рассчитать их трудоемкость;
- сравнить их временные характеристики экспериментально;
- на основании проделанной работы сделать выводы.

Экспериментально были установлены различия в производительности различных алгоритмов умножения матриц. Оптимизированный алгоритм Копперсмита–Винограда имеет меньшую сложность, нежели классический алгоритм умножения матриц. Так при размерах матриц 800×800 классический алгоритм отстает от алгоритма Копперсмита-Винограда на 45%.

Литература

1. Блэнди Дж., Орендорф Дж. Программирование на языке Rust = Programming Rust. — ДМК Пресс, 2018. — 550 с. — ISBN 978-5-97060-236-2.
2. Criterion.rs - Statistics-driven benchmarking library for Rust [Электронный ресурс] https://github.com/bheisler/criterion.rs (дата обращения: 12 октября 2021 г.)
3. LPDDR4 [Электронный ресурс] https://ru.wikipedia.org/wiki/LPDDR#LPDDR4 (дата обращения: 12 октября 2021 г.)