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 по лабораторной работе №2 по курсу "Анализ алгоритмов"

Тема Умножение матриц. Алгоритм Винограда

Студент Ивахненко Д.А.

Группа ИУ7-56Б

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Целью работы является исследование алгоритмов перемножения матриц.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- реализовать классический алгоритм перемножения матриц;
- реализовать алгоритм Винограда перемножения матриц.
- реализовать улушенный алгоритм Винограда перемножения матриц.
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ временных характеристик алгоритмов экспериментально;

1 Аналитическая часть

1.1 Классический алгоритм перемножения матриц

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

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

где

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

называется их произведением[1].

1.2 Алгоритм Винограда для перемножения матриц

Если рассмотреть результат умножения двух матриц, то видно, что каждый элемент в нем представляет собой скалярное произведение соответствующих строки и столбца исходных матриц. Можно заметить также, 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: $V \cdot W = v_1w_1 + v_2w_2 + v_3w_3 + v_4w_4$, что экви-

валентно (1.4):

$$V \cdot W = (v_1 + w_2)(v_2 + w_1) + (v_3 + w_4)(v_4 + w_3) - v_1 v_2 - v_3 v_4 - w_1 w_2 - w_3 w_4. \quad (1.4)$$

Несмотря на то, что второе выражение требует вычисления большего количества операций, чем стандартный алгоритм: вместо четырёх умножений - шесть, а вместо трёх 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, что позволит для каждого элемента выполнять лишь два умножения и пять сложений, складывая затем только лишь с 2 предварительно посчитанными суммами соседних элементов текущих строк и столбцов. Из-за того, что операция сложения быстрее операции умножения в ЭВМ, на практике алгоритм должен работать быстрее стандартного.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунке 2.1 приведена схема стандартного алгоритма перемножения матриц. На рисунках 2.2 – 2.4 и 2.5 – 2.7 приведены схемы алгоритма Винограда и оптимизированного алгоритма Винограда для перемножения матриц соответственно.

Для алгоритма Винограда худшим случаем являются матрицы нечетной размерности, а лучшим – четной, поскольку в таком случае отпадает необходимость в последнем цикле.

Данный алгоритм может быть улучшен путем использования следующих оптимизаций:

- использование конструкции a += x вместо a = a + x;
- накопление результата в буфер, а затем помещение буфера в ячейку;
- замена в циклах k < N/2, k += 1 на k < N, k += 2;
- предварительное вычисление констант таких как is_odd = N % 2;
- замена операции взятия остатка по модулю два (N % 2) на битовое умножение на 1 (N & 1);
- объединение основного внешнего цикла с последним.

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

1. Трудоемкость базовых операций:

   Пусть операции из (2.1) имеют единичную трудоемкость.

$$+, -, =, ==, !=, <, >, <=, >=, [], +=, -=, <<, >> \qquad (2.1)$$

Пусть операции из (2.2) имеют трудоемкость 2.

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

2. Трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.3).

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

3. Трудоемкость цикла рассчитывается по C-подобной модели, то есть "for (i=0; i<N; i+=1) как (2.4).

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

4. Трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

2.3.1 Стандартный алгоритм перемножения матриц

Трудоёмкость стандартного алгоритма перемножения матриц складывается из трудоемкостей:

- Внешнего цикла по $i \in [1..M]$, трудоёмкость которого: $f = 2 + M \cdot (2 + f_{body})$;
- Цикла по $j \in [1..Q]$, трудоёмкость которого: $f = 2 + Q \cdot (2 + f_{body})$;
- Скалярного умножения двух векторов - цикл по $k \in [1..N]$, трудоёмкость которого: $f = 2 + 11N$;

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

$$f_{base} = 2+M\cdot(4+Q\cdot(4+11N)) = 2+4M+4MQ+11MQN \approx 11MQN \tag{2.5}$$

2.3.2 Алгоритм Винограда

Трудоёмкость алгоритма Винограда для перемножения матриц складывается из трудоемкостей:

1. Создания векторов $MulH$ и $MulV$ (2.6):

$$f_{create} = f_{MulH} + f_{MulV}; \tag{2.6}$$

2. Заполнения вектора $MulH$ (2.7):

$$f_{MulH} = 2 + M(2 + 4 + \frac{N}{2}(4 + 6 + 2 + 1 + 2 \cdot 3)) = \frac{19}{2}MN + 6M + 2; \tag{2.7}$$

3. Заполнения вектора $MulV$ (аналогично 2.7):

$$f_{MulV} = \frac{19}{2}QN + 6Q + 2; \tag{2.8}$$

4. Цикла заполнения матрицы для чётных размеров (2.9):

$$f_{cycle} = 2 + M(4 + Q(13 + \frac{N}{2} \cdot 32)) = 16MNQ + 13MQ + 4M + 2; \tag{2.9}$$

5. Цикла, для дополнения умножения суммой последних нечётных строки и столбца, если общий размер нечётный (2.10):

$$f_{last} = 3 + \begin{cases} 0, & \text{если N - четно} \\ 13MQ + 4M + 2, & \text{иначе.} \end{cases} \tag{2.10}$$

Суммарная трудоёмкость:

$$f_{win-common} = f_{create} + f_{cycle} + f_{last} = 16MNQ + 13MQ + \frac{19}{2}(MN + QN) + \\ + 10M + 6Q + 9 + \begin{cases} 0 & \text{если N - четно} \\ 13MQ + 4M + 2 & \text{иначе.;} \end{cases} \tag{2.11}$$

Итого, для худшего случая (нечётный размер матриц):

$$f_{win-worst\_case} = 16MNQ + 26MQ + \frac{19}{2}(MN + QN) + 14M + 6Q + 11; \quad (2.12)$$

Для лучшего случая (чётный размер матриц):

$$f_{win-best\_case} = 16MNQ + 13MQ + \frac{19}{2}(MN + QN) + 10M + 6Q + 9; \quad (2.13)$$

2.3.3 Оптимизированный Алгоритм Винограда

Трудоёмкость оптимизированного алгоритма Винограда для перемножения матриц складывается из трудоемкостей:

1. Создания векторов $MulH$ и $MulV$ (2.14):

$$f_{create} = f_{MulH} + f_{MulV}; \quad (2.14)$$

2. Заполнения вектора $MulH$ (2.15):

$$f_{MulH} = 2+M(2+1+2+\frac{N}{2}(2+4+2+1+1)+2) = 5MN+7M+2; \quad (2.15)$$

3. Заполнения вектора $MulV$ (аналогично 2.15):

$$f_{MulV} = 5QN + 7Q + 2; \quad (2.16)$$

4. Цикла заполнения матрицы для чётных размеров (2.17):

$$\begin{aligned} f_{cycle} &= 2 + M(4 + Q(8 + \frac{17N}{2}) + 4 + \begin{cases} 0, & \text{если N - четно} \\ 9, & \text{иначе.} \end{cases}) = \\ &= \frac{17}{2}MNQ + MQ \cdot \begin{cases} 12, & \text{если N - четно} \\ 21, & \text{иначе.} \end{cases} + 4M + 2 \end{aligned} \quad (2.17)$$

5. Определения четности N (2.18):

$$f_{odd-check} = 2; \tag{2.18}$$

Суммарная трудоёмкость:

$$f_{win\_imp-common} = f_{create} + f_{cycle} + f_{odd-check} = \frac{17}{2}MNQ + 5(MN + QN) + \\ + 9M + 7Q + 8 + \begin{cases} 12MQ & \text{если N - четно} \\ 21MQ & \text{иначе.} \end{cases} \tag{2.19}$$

Итого, для худшего случая (нечётный размер матриц):

$$f_{win\_imp-worst\_case} = \frac{17}{2}MNQ + 5(MN + QN) + 21MQ + 9 + 7Q + 8; \tag{2.20}$$

Для лучшего случая (чётный размер матриц):

$$f_{win\_imp-best\_case} = \frac{17}{2}MNQ + 5(MN + QN) + 12MQ + 9 + 7Q + 8; \tag{2.21}$$

2.4 Вывод

В данном разделе были рассмотрены схемы алгоритмов умножения матриц, введена модель вычисления трудоёмкости, рассчитаны трудоёмкости алгоритмов. Из формул 2.5, 2.13, 2.21 видно, что все три алгоритма имеют зависимость от размерности матриц, однако оптимизированный алгоритм Винограда имеет наилучший коэффициент перед слагаемым MQN, равный 8, из чего следует, что он должен быть наиболее эффективным.

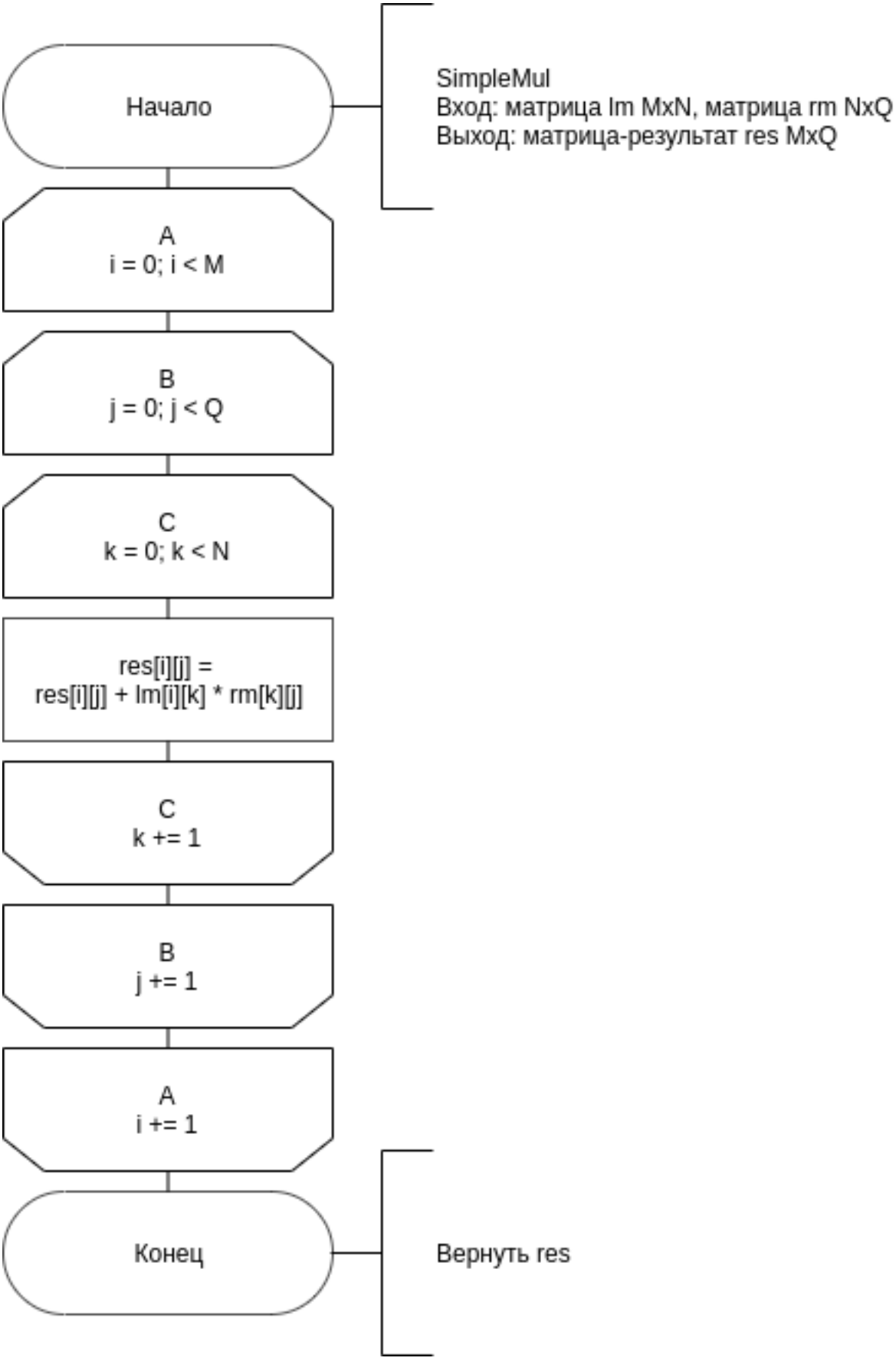

Рисунок 2.1 – Схема стандартного алгоритма перемножения матриц

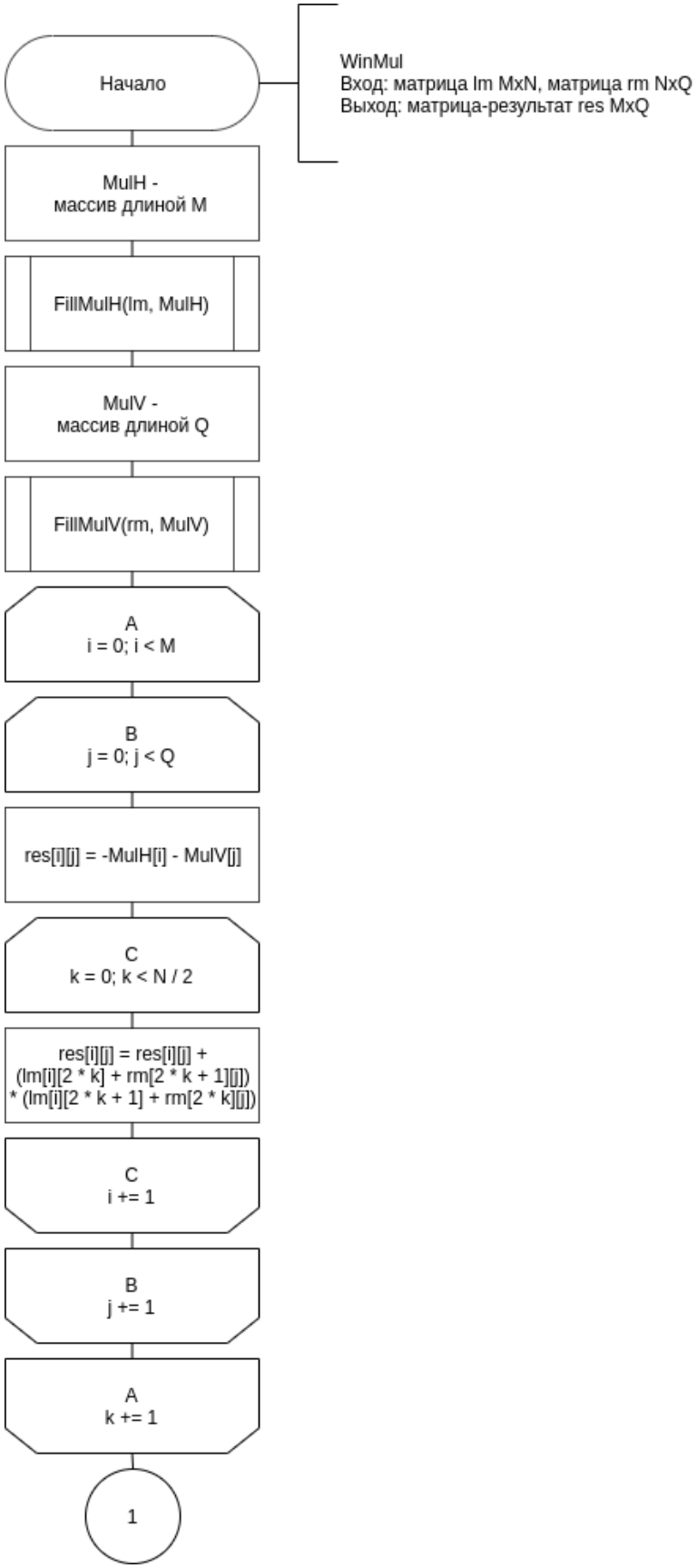

Рисунок 2.2 – Схема алгоритма Винограда для перемножения матриц

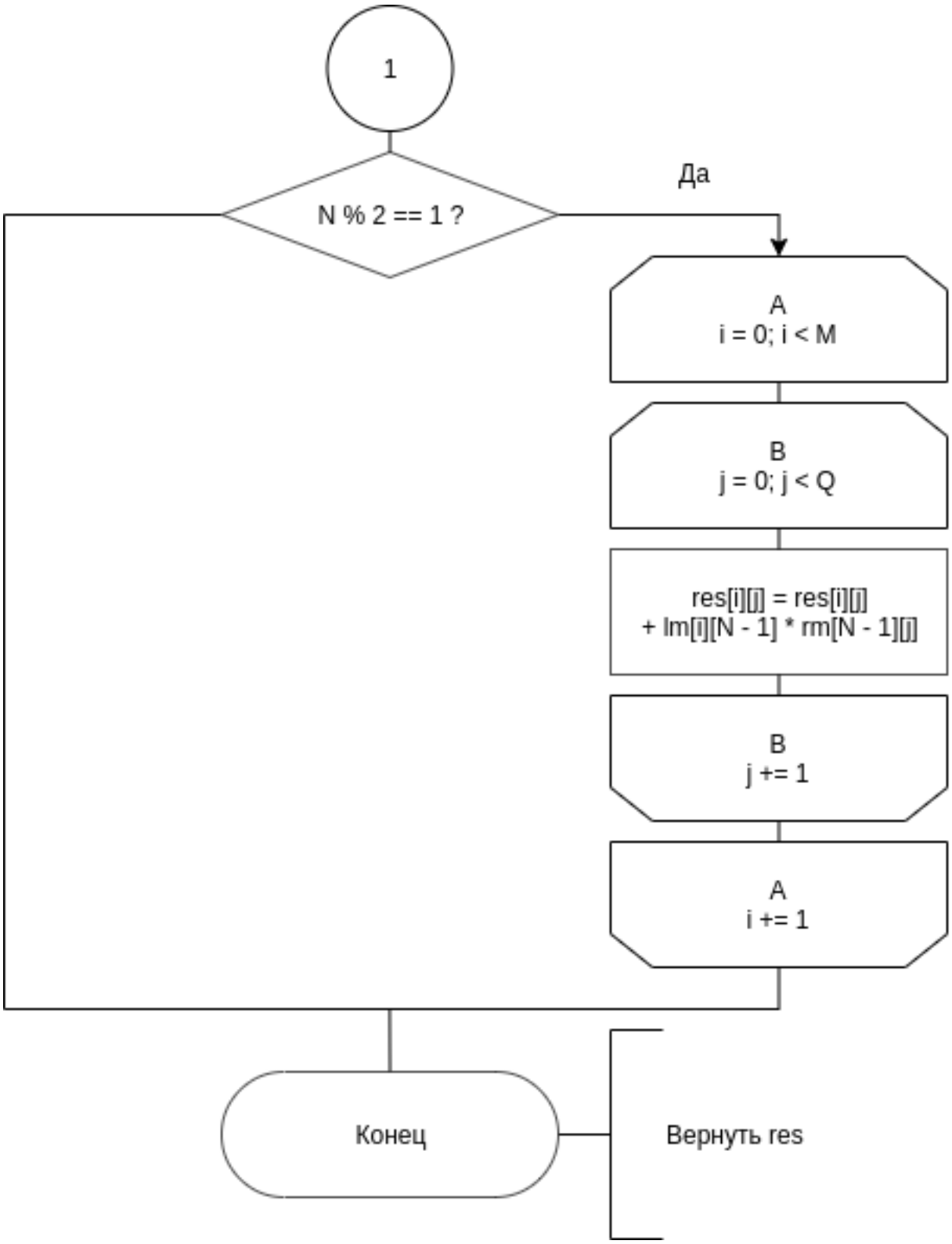

Рисунок 2.3 – Продолжение схемы алгоритма Винограда для перемножения матриц

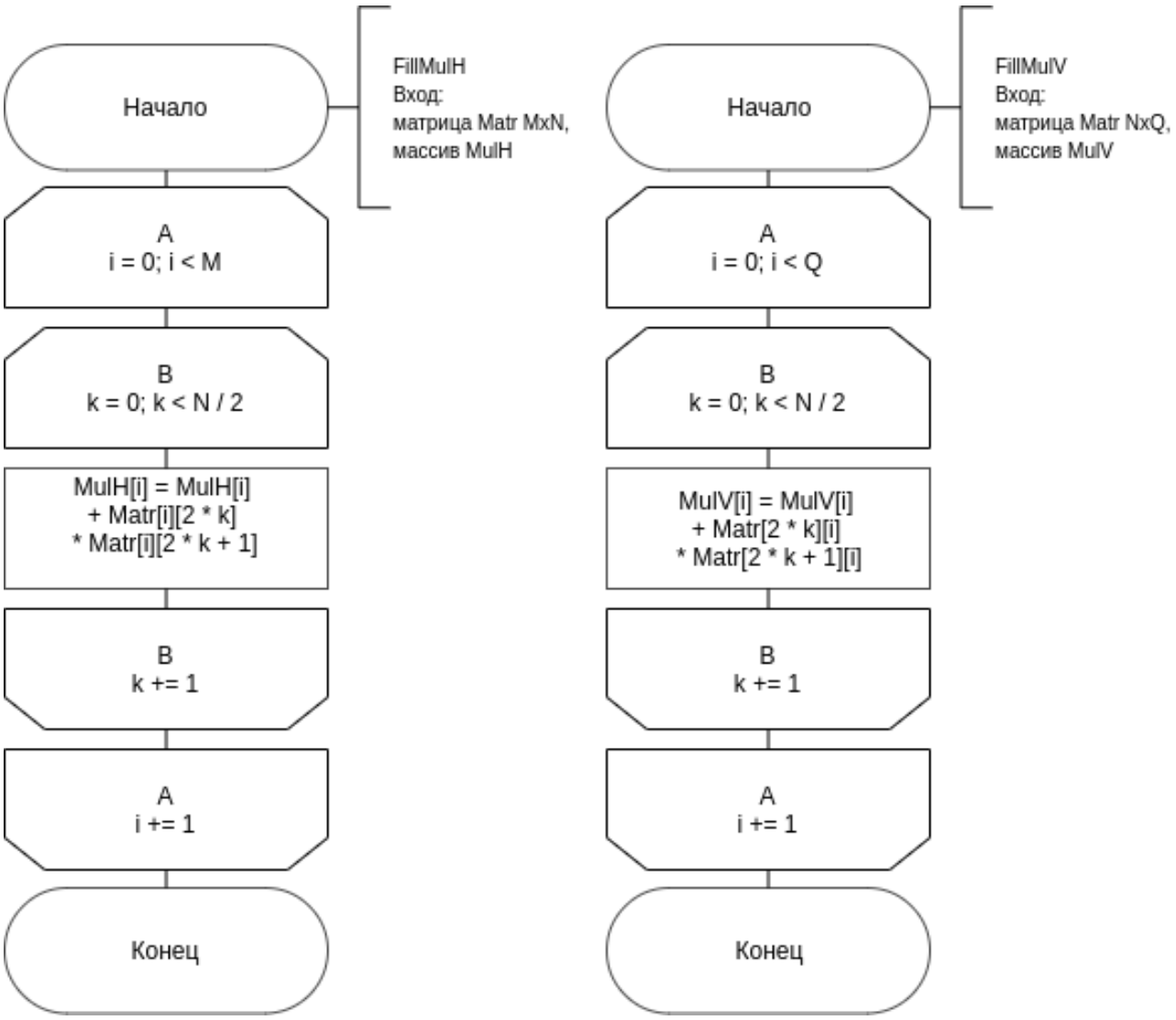

Рисунок 2.4 – Схема алгоритмов предварительных вычислений в алгоритме Винограда

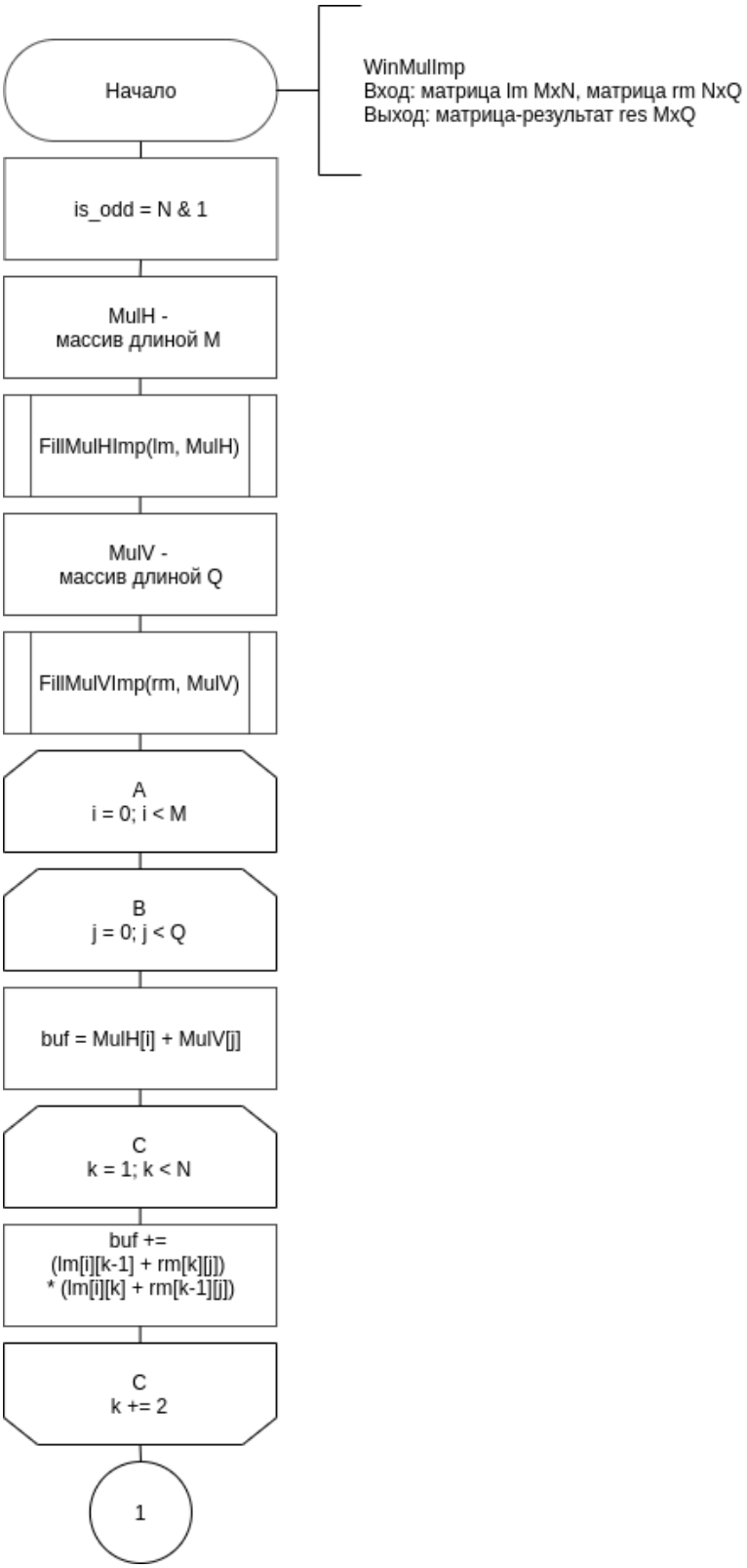

Рисунок 2.5 – Схема оптимизированного алгоритма Винограда для перемножения матриц

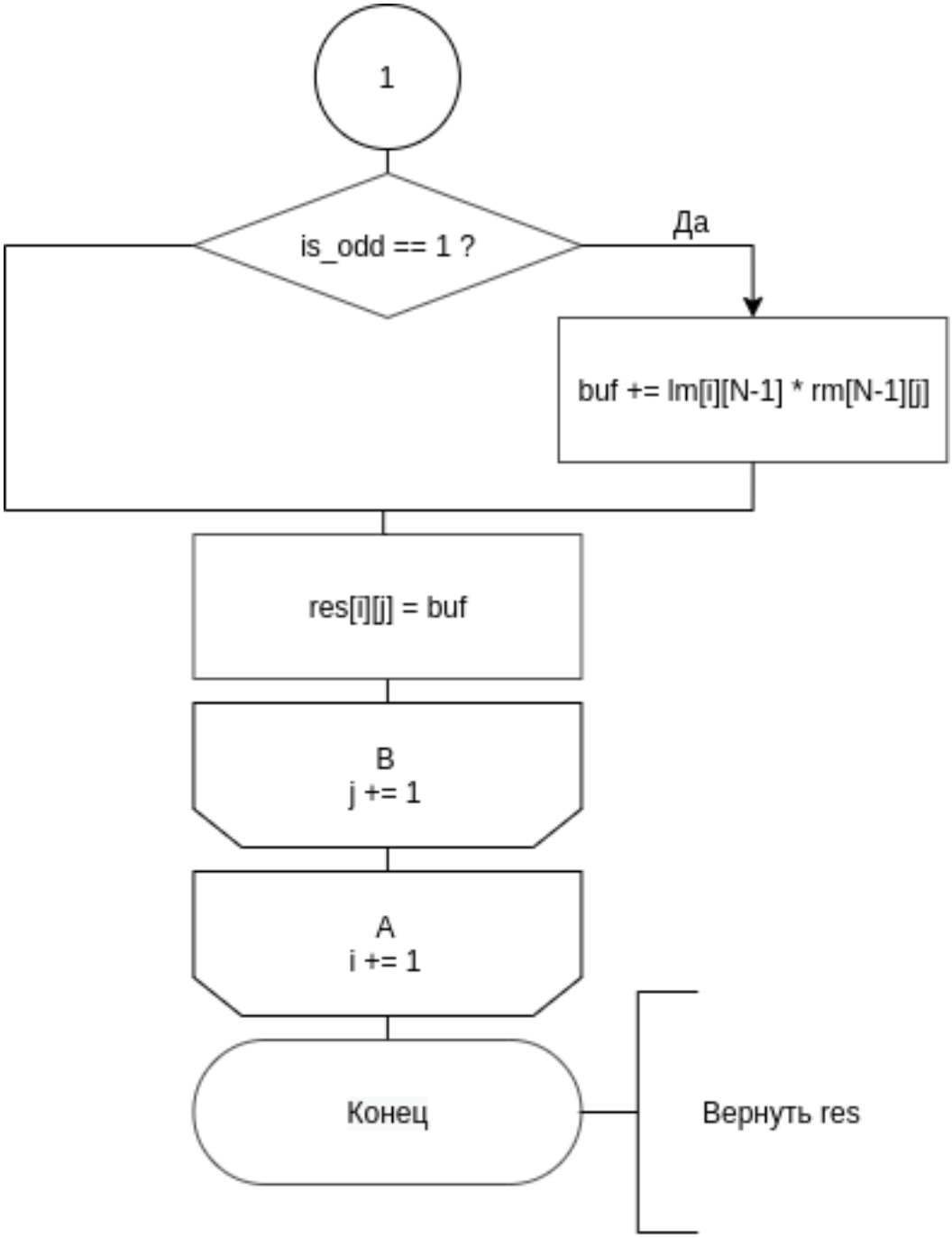

Рисунок 2.6 – Продолжение схемы оптимизированного алгоритма Винограда для перемножения матриц

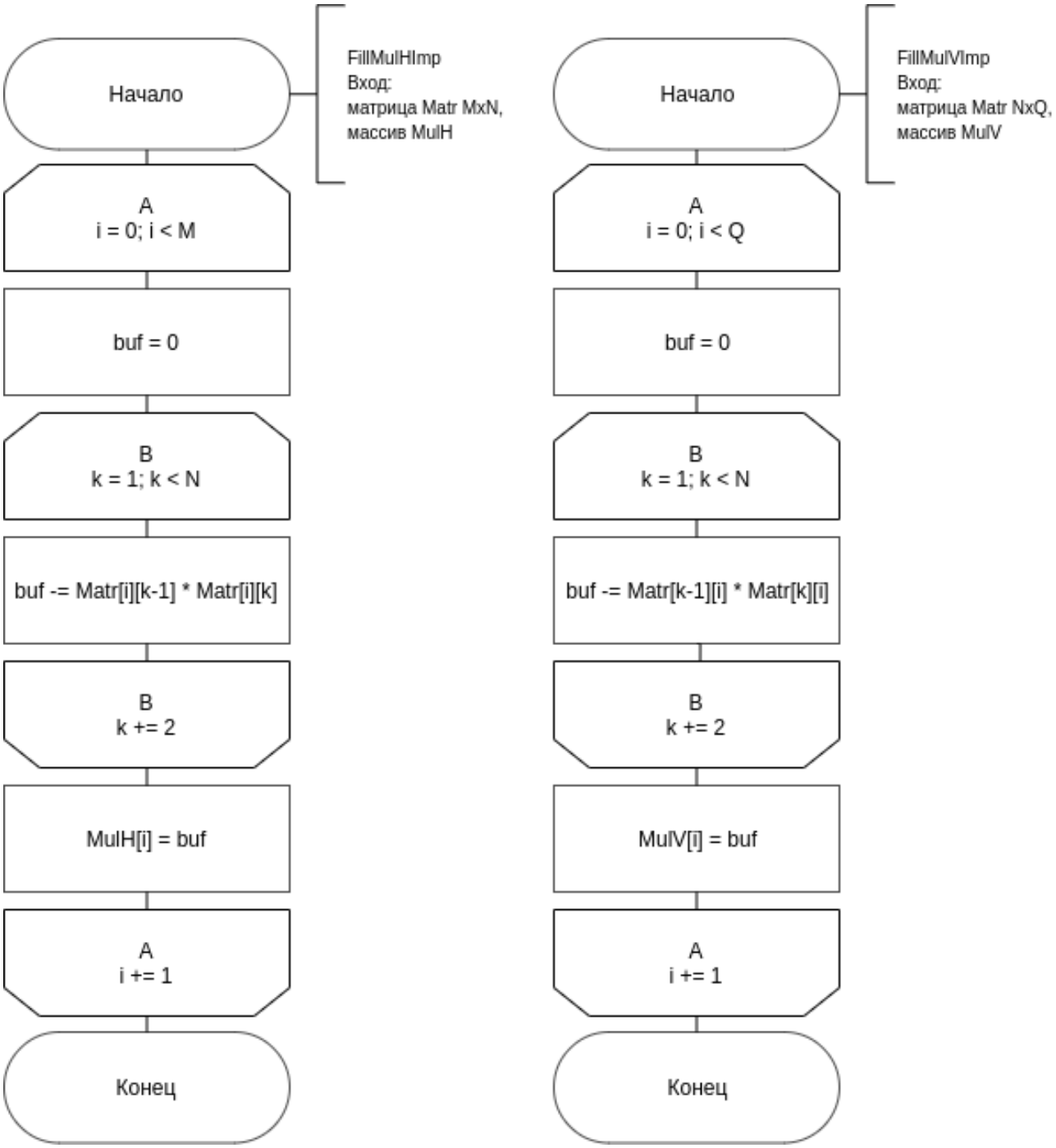

Рисунок 2.7 – Схема алгоритмов предварительных вычислений в оптимизированном алгоритме Винограда

3 Технологическая часть

3.1 Требование к ПО

К программе предъявляется ряд требований:

- предоставить возможность ввода пользователем целочисленных матриц, которые необходимо перемножить;
- предоставить возможность автозаполнения матриц заданных размерностей случайными числами;
- печать результатов перемножения матриц, используя классического алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда.

3.2 Средства реализации

Для реализации программы нахождения расстояние Левенштейна был выбран язык программирования Python3[2]. Данный выбор обусловлен простотой и скоростью написания программ, а также наличием встроенных библиотек для построения графиков функций и тестирования. В качестве среды разработки был выбран `PyCharm`[3], как наиболее популярная IDE для Python3.

3.3 Реализация алгоритмов

В листингах 3.1-3.3 представлены реализации классического алгоритма перемножения матриц, алгоритма Винограда, а также оптимизированного алгоритма Винограда.

Листинг 3.1 – Функция классического перемножения матриц

```
def simple_mul(lm: MatrixInt, rm: MatrixInt) -> MatrixInt:
    m, n, q = lm.m, lm.n, rm.n
    res = MatrixInt(m, q)
```

```
    i = 0
    while i < m:
        j = 0
        while j < q:
            k = 0
            while k < n:
                res[i][j] += lm[i][k] * rm[k][j]
                k += 1
            j += 1
        i += 1

return res
```

Листинг 3.2 – Функция для перемножения матриц алгоритмом Винограда

```
def win_mul(lm: MatrixInt, rm: MatrixInt) -> MatrixInt:
    m, n, q = lm.m, lm.n, rm.n

    mul_h = VectorInt(m)
    fill_mul_h(mul_h, lm)

    mul_v = VectorInt(q)
    fill_mul_v(mul_v, rm)

    res = MatrixInt(m, q)
    i = 0
    while i < m:
        j = 0
        while j < q:
            k = 0
            res[i][j] = - mul_h[i] - mul_v[j]
            while k < n // 2:
                res[i][j] = res[i][j] \
                            + (lm[i][2 * k] + rm[2 * k + 1][j]) \
                            * (lm[i][2 * k + 1] + rm[2 * k][j])
                k += 1
            j += 1
        i += 1

    if n % 2 == 1:
        i = 0
        while i < m:
            j = 0
            while j < q:
                res[i][j] = res[i][j] + lm[i][n - 1] * rm[n - 1][j]
                j += 1
            i += 1
    return res
```

```
def fill_mul_h(mul_h: VectorInt, matrix: MatrixInt) -> None:
    m, n = matrix.m, matrix.n

    i = 0
    while i < m:
        k = 0
        while k < n // 2:
            mul_h[i] = mul_h[i] + matrix[i][2 * k] * matrix[i][2 * k + 1]
            k += 1
        i += 1


def fill_mul_v(mul_v: VectorInt, matrix: MatrixInt) -> None:
    n, q = matrix.m, matrix.n

    i = 0
    while i < q:
        k = 0
        while k < n // 2:
            mul_v[i] = mul_v[i] + matrix[2 * k][i] * matrix[2 * k + 1][i]
            k += 1
        i += 1
```

Листинг 3.3 – Функция для перемножения матриц оптимизированным алгоритмом Винограда

```
def win_mul_imp(lm: MatrixInt, rm: MatrixInt) -> MatrixInt:
    m, n, q = lm.m, lm.n, rm.n

    mul_h = VectorInt(m)
    fill_mul_h_imp(mul_h, lm)

    mul_v = VectorInt(q)
    fill_mul_v_imp(mul_v, rm)

    is_odd = n & 1
    res = MatrixInt(m, q)
    i = 0
    while i < m:
        j = 0
        while j < q:
            buf, k = mul_h[i] + mul_v[j], 1
            while k < n:
                buf += (lm[i][k - 1] + rm[k][j]) * (lm[i][k] + rm[k - 1][j])
                k += 2
            if is_odd == 1:
                buf += lm[i][n - 1] * rm[n - 1][j]
```

```
            res[i][j] = buf
            j += 1
        i += 1

return res

def fill_mul_h_imp(mul_h: VectorInt, matrix: MatrixInt) -> None:
    m, n = matrix.m, matrix.n

    i = 0
    while i < m:
        buf, k = 0, 1
        while k < n:
            buf -= matrix[i][k - 1] * matrix[i][k]
            k += 2
        mul_h[i] = buf
        i += 1


def fill_mul_v_imp(mul_v: VectorInt, matrix: MatrixInt) -> None:
    n, q = matrix.m, matrix.n

    i = 0
    while i < q:
        buf, k = 0, 1
        while k < n:
            buf -= matrix[k - 1][i] * matrix[k][i]
            k += 2
        mul_v[i] = buf
        i += 1
```

3.4 Тестовые данные

В таблице 3.1 приведены тесты для функций, реализующих стандартный алгоритм умножения матриц, алгоритм Винограда и оптимизированный алгоритм Винограда.

Первая матрица	Вторая матрица	Ожидаемый результат
$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$
$\begin{pmatrix} 1 & 0 & 2 \\ 1 & 0 & 2 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 3 & 0 \\ 3 & 0 \end{pmatrix}$
(2)	(3)	(6)
$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 3 & 2 & 1 \\ 3 & 2 & 1 \\ 3 & 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 18 & 12 & 6 \\ 18 & 12 & 6 \\ 18 & 12 & 6 \end{pmatrix}$

Таблица 3.1 – Тестирование функций

3.5 Вывод

В данном разделе были рассмотрены исходные коды реализованных алгоритмов, а также тестовые случаи для них.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Ubuntu[4] Linux x86_64.
- Память: 16 GiB.
- Процессор: AMD Ryzen™ 7 4700U[5].

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

4.2 Время выполнения алгоритмов

Алгоритмы тестировались при помощи встроенного встроенного модуля `timeit`[6]. Специальная функция делает 50 замеров и в качестве результата возвращает среднее значение.

Результаты замеров приведены в таблицах 4.1–4.2. На рисунках 4.1 и 4.2 приведены графики, иллюстрирующие зависимость времени работы алгоритмов умножения от размеров матриц.

4.3 Вывод

Оптимизированный алгоритм Винограда для перемножения матриц работает в ≈ 1.5 раза **быстрее** классического алгоритма перемножения матриц при четных размерностях и ≈ 1.4 раза – нечетных. В то же время, неоптимизированный алгоритм Винограда работает в ≈ 1.03 раза **медленнее** классического алгоритма перемножения матриц при четных размерностях и ≈ 1.07 раза – нечетных.

Таблица 4.1 – Таблица замеров времени выполнения алгоритмов в секундах для четных размерностей

Размерность	Простой	Виноград	Виноград (оптимиз.)
50	0.044	0.047	0.031
100	0.354	0.367	0.242
150	1.254	1.242	0.841
200	2.834	2.940	1.927
250	5.613	5.740	3.880

Таблица 4.2 – Таблица замеров времени выполнения алгоритмов в секундах для нечетных размерностей

Размерность	Простой	Виноград	Виноград (оптимиз.)
51	0.051	0.053	0.038
101	0.378	0.407	0.286
151	1.255	1.374	0.889
201	2.991	3.216	2.077
251	5.889	6.194	3.972

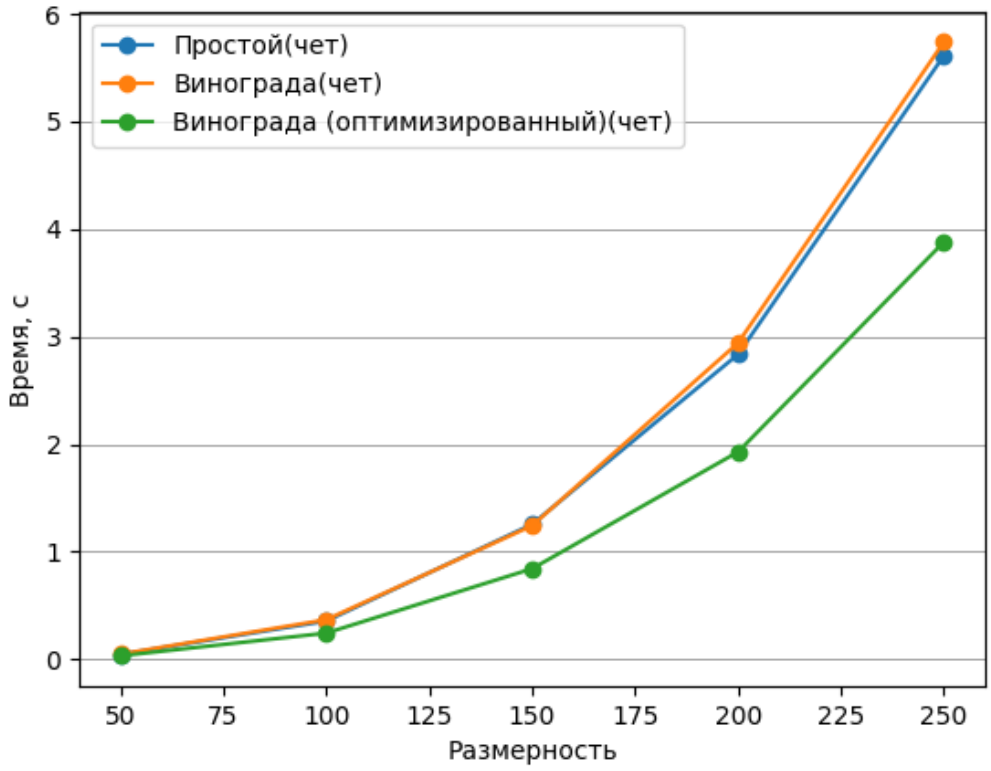

Рисунок 4.1 – Зависимость времени работы реализаций алгоритмов от чётной размерности квадратной матрицы

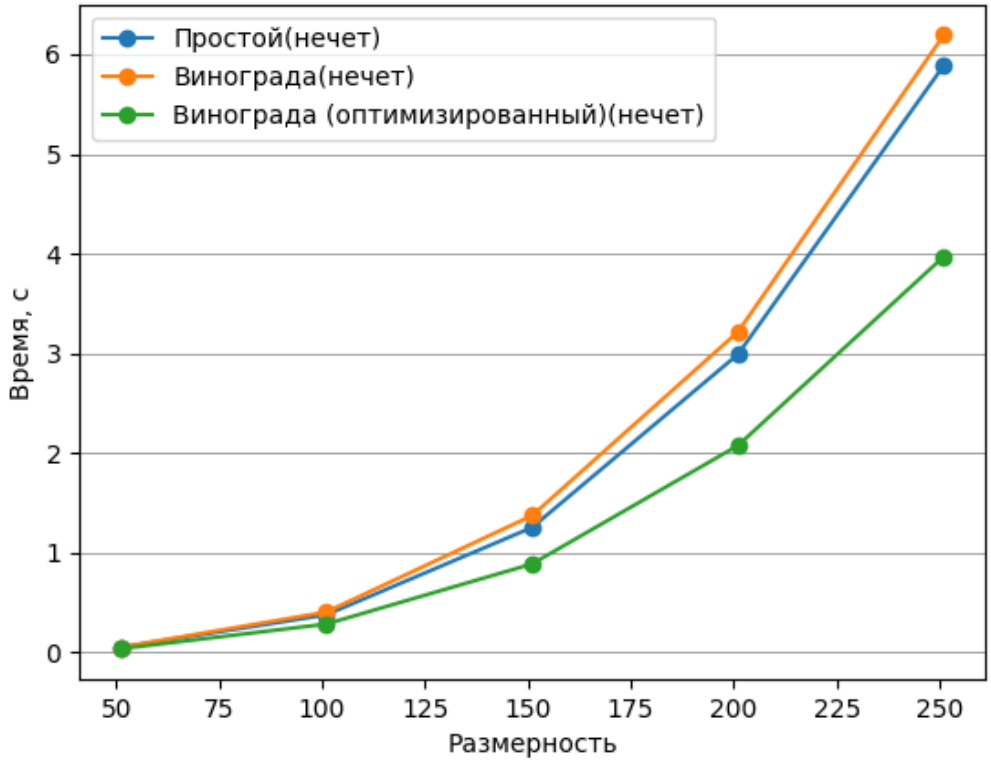

Рисунок 4.2 – Зависимость времени работы реализаций алгоритмов от нечётной размерности квадратной матрицы

Заключение

В ходе выполнения лабораторной работы была проделана следующая работа:

- реализованы классический алгоритм перемножения матриц и алгоритм Винограда;
- реализован оптимизированный алгоритм Винограда;
- рассчитана трудоемкость для каждого алгоритма;
- проведен анализ временных характеристик экспериментально;
- сделаны выводы по поводу эффективности всех реализованных алгоритмов.

Литература

[1] Погорелов Дмитрий Александрович. Оптимизация классического алгоритма Винограда для перемножения матриц // Журнал №1 (Vol. 49). URL: https://www.gyrnal.ru/statyi/ru/1153/.

[2] Python 3.9.7 documentation [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/` (дата обращения: 28.09.2021).

[3] PyCharm: IDE для профессиональной разработки на Python [Электронный ресурс]. Режим доступа: `https://www.jetbrains.com/ru-ru/pycharm/` (дата обращения: 28.09.2021).

[4] Ubuntu 20.04.3 LTS [Электронный ресурс]. Режим доступа: `https://ubuntu.com/download/desktop` (дата обращения: 28.09.2020).

[5] Проессор AMD Ryzen™ 7 4700U [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-7-4700u` (дата обращения: 28.09.2021).

[6] timeit — Measure execution time of small code snippets [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/timeit.html` (дата обращения: 28.09.2021).