МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Дискретное логарифмирование в конечных группах**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Сенокосова Владислава Владимировича

|  |  |  |
| --- | --- | --- |
| Преподаватель, профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2024

**Содержание**

[1 Цель работы и порядок выполнения 3](#_Toc180163030)

[2 Метод Гельфонда-Шенкса вычисления дискретного логарифма 4](#_Toc180163031)

[3 ρ-метод Полларда вычисления дискретного логарифма 6](#_Toc180163032)

[4 Метод вычисления дискретного логарифма с помощью сведения к собственным подгруппам 9](#_Toc180163033)

[5 Тестирование алгоритмов дискретного логарифмирования 11](#_Toc180163034)

[ЗАКЛЮЧЕНИЕ 14](#_Toc180163035)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 15](#_Toc180163036)

[ПРИЛОЖЕНИЕ А 16](#_Toc180163037)

# **1 Цель работы и порядок выполнения**

**Цель работы** — изучение основных методов дискретного логарифмирования в конечных группах и их программная реализация. Порядок выполнения работы

1. Рассмотреть метод Гельфонда-Шенкса вычисления дискретного логарифма и привести его программную реализацию.

2. Рассмотреть ρ-метод Полларда вычисления дискретного логарифма и привести его программную реализацию.

3. Рассмотреть метод вычисления дискретного логарифма с помощью сведения к собственным подгруппам.

# **2 Метод Гельфонда-Шенкса вычисления дискретного логарифма**

**Определение:** Дискретным логарифмом (показателем) элемента группы по основанию называется число , являющееся решением уравнения .

Будем обозначать дискретный логарифм через

Пусть – конечная циклическая группа порядка , – образующий элемент и . Дискретным логарифмом элемента группы по основанию называется число , являющееся решением уравнения .

Алгоритм Гельфонда-Шенкса.

**Вход:** конечная циклическая группа , верхняя оценка для порядка группы , элемент .

**Выход:** число .

1. Вычислить и вычислить элементы , упорядочить массив пар по второй координате.
2. Вычислить , для каждого проверить, является ли элемент второй координатой какой-либо пары из упорядоченного на первом шаге массива пар. Если , то запомнить число .
3. Среди всех чисел, найденных на втором этапе, выбрать наименьшее, это значение и будет искомым значением .

Сложность метода Гельфонда-Шенкса равна .

Псевдокод алгоритма Гельфонда-Шенкса

Входные данные:

— генератор конечной циклической группы.

— элемент, для которого необходимо найти логарифм.

— порядок группы (значение модульного числа).

Выходные данные:

Значение дискретного логарифма , такое что . Начало алгоритма:

1. Вычислить значение .

2. Создать таблицу соответствий для "малых шагов" для от 0 до.

3. Вычислить фактор .

4. Для каждого значения от 0 до :

4.1. Вычислить .

4.2. Если содержится в таблице малых шагов, вычислить результат .

5. Вернуть минимальное значение из найденных.

Конец алгоритма

Сложность реализованного метода равна .

# **3 ρ-метод Полларда вычисления дискретного логарифма**

Дана конечная циклическая группа , известен ее порядок и .

Метод Полларда применим к любой циклической группе , чьи элементы представлены таким образом, что их можно разбить на три примерно равные, попарно непересекающиеся части . При этом должен существовать эффективный способ проверки, к какому из этих подмножеств принадлежит данный элемент группы. Это – вероятностный алгоритм дискретного логарифмирования в группе, имеющий среднюю временную сложность порядка .

Если то можно взять

Определим функцию на таким образом, что

Будет построена рекуррентная последовательность . Из определения функции нетрудно заметить, что при любом , для некоторых . Также нетрудно заметить, что последовательности задаются следующими рекуррентными соотношениями:

При вычислении очередного члена последовательности , числа вычисляются по известным очень легко. При этом для любого выполняется равенство .

Алгоритм.

**Вход:** конечная циклическая группа , порядка , элемент , введенная выше функция число .

**Выход:** число с вероятностью не менее .

1. Вычислить ;
2. Положить , выбрать случайное , вычислить . Запомнить две тройки и перейти к шагу 4;
3. Положить , найти Запомнить две тройки и перейти к шагу 4;
4. Если , то проверить выполнение условия . Если это условие выполнено, то перейти к шагу 3. В противном случае остановить алгоритм и сообщить, что вычислить не удалось. Если , то перейти к шагу 5;
5. Вычислить . Если , то перейти на шаг 2 и выбрать новое значение . В противном случае решить сравнение . Если то единственное решение сравнения равно искомому . Если же , то сравнение имеет различных решений по модулю . Для каждого из этих решений проверить выполнимость равенства и найти истинное решение .

Сложность -метода Полларда равна операций в группе.

Псевдокод ρ-метода Полларда

Входные данные:

— генератор конечной циклической группы.

— элемент, для которого требуется найти дискретный логарифм.

— порядок группы (значение модульного числа).

— вероятность успеха (по умолчанию 0.9) (При вычислении сложности будет обозначаться как )

Выходные данные:

Значение дискретного логарифма x, такое что , либо -1, если решение не найдено.

Начало алгоритма

1. Инициализация переменных:

1.1 : оценка верхней границы числа итераций, необходимых для нахождения цикла.

1.2 Переменная s случайным образом инициализируется в диапазоне .

1.3 : начальные значения, зависящие от , и .

2. Обновление значений переменных и в цикле до тех пор, пока не будет найдено совпадение (обнаружен цикл).

3. Если найдено совпадение:

3.1 Вычисляется делитель и решается сравнение для нахождения логарифма .

3.2 Если , то найдено решение, возвращается .

4. Если не найдено решение за определенное количество итераций, возвращается .

Конец алгоритма

Сложность -метода Полларда равна

# **4 Метод вычисления дискретного логарифма с помощью сведения к собственным подгруппам**

Покажем, что решение задачи дискретного логарифмирования в циклической группе , у которой известен порядок - составное число, сводится к решению задач дискретного логарифмирования в подгруппах группы

Пусть даны — конечная циклическая группа порядка — образующий элемент и . Пусть также — составное число. Тогда либо = где либо , где простое число, .

Рассмотрим сначала первый случай*.*

Пусть Тогда содержит две циклических подгруппы Нетрудно видеть, что Причем единственные подгруппы группы порядков соответственно.

Рассмотрим элементы Так как

то и, следовательно, Вычислим в группе При этом выполняются равенства Так как то Получаем систему сравнений

Которая равносильна системе сравнений

Так как то из этой системы неизвестный может быть найден по китайской теореме об остатках.

Рассмотрим теперь второй случай, то есть , где – целое число, Представим неизвестный показатель в ичной системе счисления где Алгоритм вычисления состоит в последовательном вычислении

Сначала найдем . Для этого вычислим и Элемент имеет порядок и, следовательно, порождает подгруппу порядка При этом из равенства вытекает равенство Действительно, из

Вычислим в группе В результате из равенства получим равенство , где . Так как элемент имеет порядок то он порождает подгруппу порядка и Таким образом, выполнив одно логарифмирование в мы свели задачу к вычислению дискретного логарифма в группе меньшего порядка Продолжая процесс таким образом вычислим всепроделав логарифмирований в

Сложность алгоритма:

# **5 Тестирование алгоритмов дискретного логарифмирования**

Программная реализация представлена на языке Python. Протестируем каждый из реализованных алгоритмов на следующих входных параметрах : (2, 23, 37), (3, 13, 17), (7, 167, 587), (78, 765, 1579), (2, 10, 19), (2, 22, 29), (2, 23, 90). В качестве будет значение 0.5. Дальнейшие вычисления представлены на рисунках.

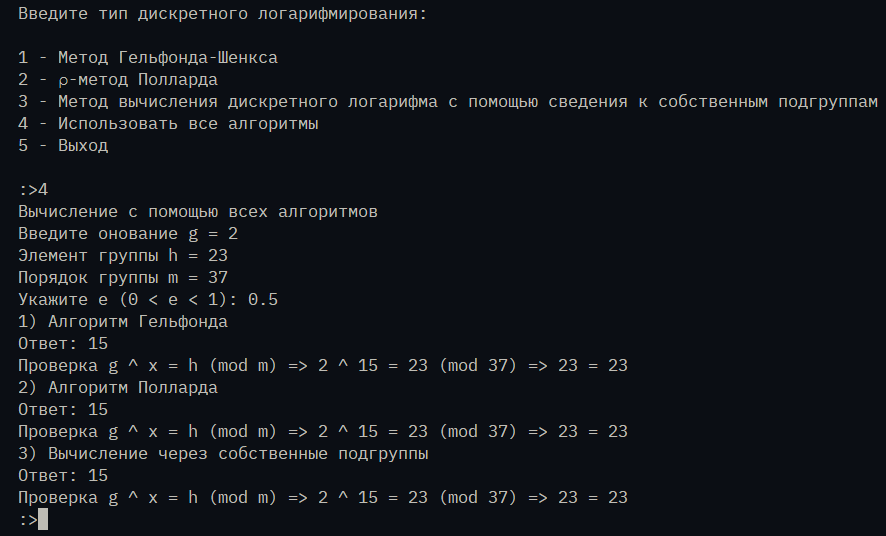


Рисунок 1 – Вычисление дискретного логарифма для параметров (2, 23, 37)

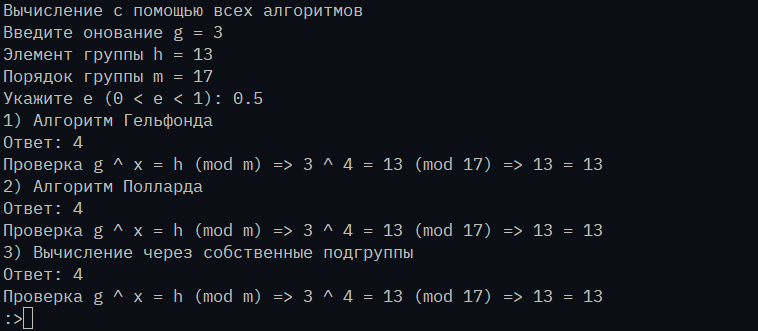


Рисунок 2 - Вычисление дискретного логарифма для параметров (3, 13, 17)

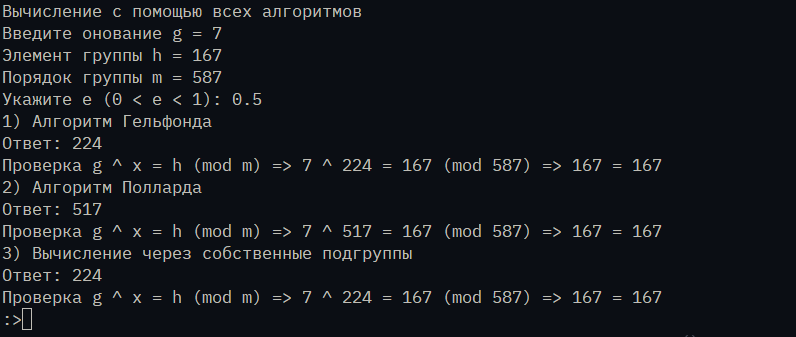
**

Рисунок 3 - Вычисление дискретного логарифма для параметров (7, 167, 587)

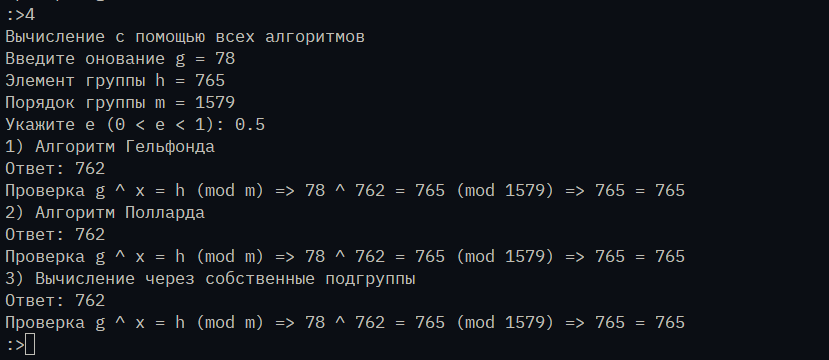
**

Рисунок 4 - Вычисление дискретного логарифма для параметров (78, 765, 1579)

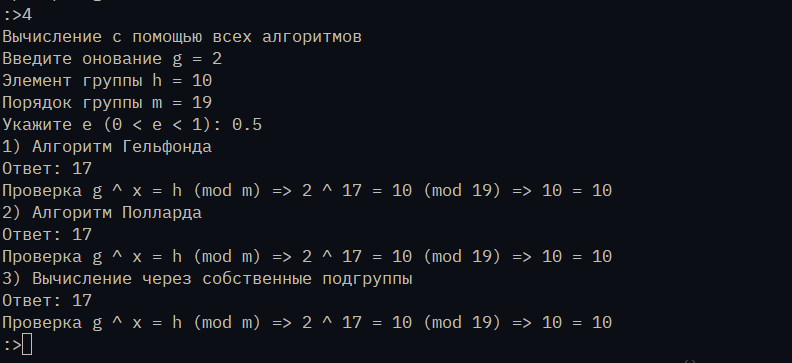
**

Рисунок 5 - Вычисление дискретного логарифма для параметров (2, 10, 19)

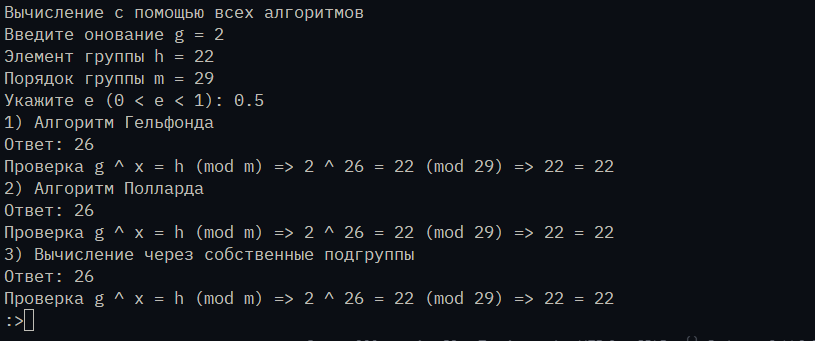
**

Рисунок 6 - Вычисление дискретного логарифма для параметров (2, 22, 29)

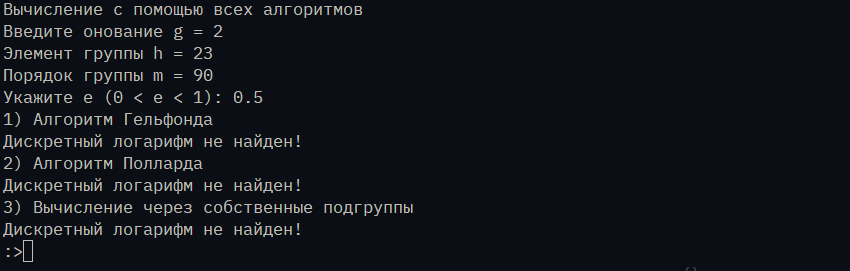
**

Рисунок 7 - Вычисление дискретного логарифма для параметров (2, 23, 90)

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы были изучены и программно реализованы три основных метода вычисления дискретного логарифма: метод Гельфонда-Шенкса, ρ-метод Полларда и метод сведения задачи к собственным подгруппам.

Каждый из методов продемонстрировал свои преимущества и ограничения в зависимости от характеристик конечной группы, таких как её порядок и структура. Метод Гельфонда-Шенкса оказался эффективным при работе с группами, где известен порядок, но имеет ограничения по времени при увеличении порядка группы. ρ-метод Полларда показал свою эффективность за счёт использования случайных блужданий и оптимальных временных затрат при достаточной вероятностной модели. Метод сведения к собственным подгруппам оказался полезен при разбиении задачи на более простые подзадачи, что снижает сложность решения для более крупных групп.

Программы для всех трёх методов были успешно реализованы и протестированы на различных наборах данных. В ходе тестирования программы корректно вычисляли дискретные логарифмы в группах с заданными параметрами, что подтверждает их правильность и надёжность.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Глухов М. М. и др. Введение в теоретико-числовые методы криптографии: учеб. пособие - Москва : Лань, 2011.

2. Маховенко Е.Б. Теоретико-числовые методы в криптографии. М.: Гелиос APB, 2006.

3. Черемушкин, А. В. Лекции по арифметическим алгоритмам в криптографии. - Москва : МЦНМО, 2002.

4. Панкратова И.А. Теоретико-числовые методы в криптографии. Томск, 2009.

5. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. М.:МЦНМО, 2003.

6. Венбо Мао. Современная криптография: теория и практика. М.:Вильямс, 2005.

# **ПРИЛОЖЕНИЕ А**

**Реализованные программы для лабораторной работы**

import math

import random

from sympy import isprime

*# Функция gcd для вычисления наибольшего общего делителя*

def gcd(a, b):

    while b != 0:

        a, b = b, a % b

    return a

*# Расширенный алгоритм Евклида для нахождения решения сравнений*

def gcd\_xt(a, b):

    s0, t0 = 1, 0

    s1, t1 = 0, 1

    while b != 0:

        q = a // b

        a, b = b, a % b

        s0, s1 = s1, s0 - q \* s1

        t0, t1 = t1, t0 - q \* t1

    return a, s0, t0

*# Алгоритм Гельфонда - Шанкса*

def log\_gelfond\_shanks(g, h, m):

    r = int(math.sqrt(m)) + 1

    table = {pow(g, a, m): a for a in range(r)}

    factor = pow(g, -r, m)

    res = []

    for b in range(r):

        value = (h \* pow(factor, b, m)) % m

        if value in table:

            res.append(b \* r + table[value])

    return min(res)

*# Функция f для шага Полларда*

def f(a, h, g, p):

    if 0 < a < p / 3:

        return (h \* a) % p

    elif p / 3 <= a < 2 \* p / 3:

        return pow(a, 2, p)

    else:

        return (g \* a) % p

*# Функция для обновления a*

def set\_element\_a(y, m, a):

    if 0 < y < m / 3:

        return a % m

    elif m / 3 <= y < 2 \* m / 3:

        return (2 \* a) % m

    else:

        return (a + 1) % m

*# Функция для обновления b*

def set\_element\_b(y, m, b):

    if 0 < y < m / 3:

        return (b + 1) % m

    elif m / 3 <= y < 2 \* m / 3:

        return (2 \* b) % m

    else:

        return b % m

*# Функция обновления элементов для y, a и b*

def set\_next\_elements(y, a, b, h, g, m):

    new\_y = f(y, h, g, m)

    new\_a = set\_element\_a(y, m, a)

    new\_b = set\_element\_b(y, m, b)

    return new\_y, new\_a, new\_b

*# Функция решения сравнения*

def solve\_comparison(a, b, p):

    g = gcd(a, p)

    if g > 1:

        if b % g != 0:

            return []

        a\_prime = a // g

        b\_prime = b // g

        p\_prime = p // g

        \_, x\_prime, \_ = gcd\_xt(a\_prime, p\_prime)

        x\_prime = (x\_prime \* b\_prime) % p\_prime

        solutions = [(x\_prime + k \* p\_prime) % p for k in range(g)]

        return solutions

    else:

        \_, a\_inv, \_ = gcd\_xt(a, p)

        x = (a\_inv \* b) % p

        return [x]

*# Основная функция для расчета дискретного логарифма*

def pollard\_rho\_discrete\_log(g, h, m, eps=0.9):

    T = int(math.sqrt(2 \* m \* math.log(1 / eps))) + 1

    while True:

        i = 1

        s = random.randint(1, m - 1)

        y, a, b = set\_next\_elements(pow(g, s, m), s, 0, h, g, m)

        y1, a1, b1 = set\_next\_elements(y, a, b, h, g, m)

        count = 10\_000

        while True:

            if y == y1:

                d = gcd((b - b1) % m, m)

                lst = solve\_comparison((a1 - a) % m, (b - b1) % m, m)

                if not lst:

                    break

                for x in lst:

                    if pow(g, x, m) == h % m:

                        return x

*# if i >= T:*

*#     return -1*

            if count == 0:

                return -1

            count -= 1

            i += 1

            y, a, b = set\_next\_elements(y, a, b, h, g, m)

            y1, a1, b1 = set\_next\_elements(y1, a1, b1, h, g, m)

            y1, a1, b1 = set\_next\_elements(y1, a1, b1, h, g, m)

*# Является ли заданное число m степенью простого числа*

def is\_prime\_power(m):

    for p in range(2, int(math.sqrt(m)) + 1):

        if isprime(p):

            n = 1

            while p \*\* n <= m:

                if p \*\* n == m:

                    return True, p, n

                n += 1

    return False, None, None

*# Разложение числа на его простые множители*

def find\_coprime\_factors(m):

    for m1 in range(2, m // 2 + 1):

        if m % m1 == 0:

            m2 = m // m1

            if math.gcd(m1, m2) == 1:

                return True, m1, m2

    return False, None, None

*# Проверка на то как раскладывается число m*

def check\_m\_factorization(m):

    prime\_power, p, n = is\_prime\_power(m)

    if prime\_power:

        return True, p, n

    coprime\_factors, m1, m2 = find\_coprime\_factors(m)

    if coprime\_factors:

        return False, m1, m2

    return "No valid factorization found"

*# Расширенный алгоритм Евклида*

def gcd\_xt(a, b):

    s0, t0 = 1, 0

    s1, t1 = 0, 1

    while b != 0:

        q = a // b

        a, b = b, a % b

        s0, s1 = s1, s0 - q \* s1

        t0, t1 = t1, t0 - q \* t1

    return s0, t0, a

*# Нахождение произведения элеметов в массиве*

def prod(list):

    res = 1

    for val in list: res \*= val

    return res

*# Китайсткая теорема об остатках (система сравнений)*

def china\_theorem(params, moduls):

    M = prod(moduls)

    u = 0

    for i, m in enumerate(moduls):

        c = M // m

        d, \_, \_ = gcd\_xt(c, m)

        u += c \* d \* params[i]

    return u % M

*# Нахождение диксретного алгоритма перебором*

def discrete\_log(base, value, mod):

    for x in range(mod):

        if pow(base, x, mod) == value:

            return x

    return None

*# Вычисление дискретного логарифма через собственные подгруппы*

def log\_subgroups(g, h, m):

    m = m - 1

    flag, m1, m2 = check\_m\_factorization(m)

    if flag:

        x = discrete\_log(g, h, m + 1)

        return x

    else:

*# print(g, m2, h, g \*\* m2 % (m + 1), h \*\* m2 % (m + 1), m + 1)*

*# print(g, m1, h, g \*\* m1 % (m + 1), h \*\* m1 % (m + 1), m + 1)*

        p = m + 1

        x\_1 = discrete\_log(g \*\* m2 % p, h \*\* m2 % p, p)

        x\_2 = discrete\_log(g \*\* m1 % p, h \*\* m1 % p, p)

        x = china\_theorem([x\_2, x\_1], [m2, m1])

        return x

def input\_data():

    g = input("Введите онование g = ").strip()

    h = input("Элемент группы h = ").strip()

    m = input("Порядок группы m = ").strip()

    while not (g.isdigit() and h.isdigit() and m.isdigit()):

        g = input("g = ").strip()

        h = input("h = ").strip()

        m = input("m = ").strip()

    return int(g), int(h), int(m)

if \_\_name\_\_ == "\_\_main\_\_":

    type\_ = """Введите тип дискретного логарифмирования: \n

1 - Метод Гельфонда-Шенкса

2 - ρ-метод Полларда

3 - Метод вычисления дискретного логарифма с помощью сведения к собственным подгруппам

4 - Использовать все алгоритмы

5 - Выход\n"""

    print(type\_)

    param = None

    while param not in ["1", "2", "3", "4", "5"]:

        param = input(":>")

        match param:

            case "1":

                print("Метод Гельфонда-Шенкса")

                try:

                    g, h, m = input\_data()

                    x = log\_gelfond\_shanks(g, h, m)

                    print("Ответ:", x)

                    print(f"Проверка g ^ x = h (mod m) =>

{g} ^ {x} = {h} (mod {m}) =>

{pow(g, x, m)} = {h % m}")

                except Exception as err:

                    print(err)

                    print("Дискретный логарифм не найден!")

                param = None

            case "2":

                print("ρ-метод Полларда")

                try:

                    g, h, m = input\_data()

                    eps = float(input("Укажите e (0 < e < 1): "))

                    x = pollard\_rho\_discrete\_log(g, h, m, eps)

                    print("Ответ:", x)

                    print(f"Проверка g ^ x = h (mod m) =>

{g} ^ {x} = {h} (mod {m}) =>

{pow(g, x, m)} = {h % m}")

                except Exception as err:

                    print(err)

                    print("Дискретный логарифм не найден!")

                param = None

            case "3":

                print("Метод вычисления дискретного логарифма

с помощью сведения к собственным подгруппам")

                try:

                    g, h, m = input\_data()

                    x = log\_subgroups(g, h, m)

                    print("Ответ:", x)

                    print(f"Проверка g ^ x = h (mod m) =>

{g} ^ {x} = {h} (mod {m}) =>

{pow(g, x, m)} = {h % m}")

                except Exception:

                    print("Дискретный логарифм не найден!")

                param = None

            case "4":

                print("Вычисление с помощью всех алгоритмов")

                g, h, m = input\_data()

                eps = float(input("Укажите e (0 < e < 1): "))

                print("1) Алгоритм Гельфонда")

                try:

                    x = log\_gelfond\_shanks(g, h, m)

                    print("Ответ:", x)

                    print(f"Проверка g ^ x = h (mod m) =>

{g} ^ {x} = {h} (mod {m}) =>

{pow(g, x, m)} = {h % m}")

                except Exception:

                    print("Дискретный логарифм не найден!")

                print("2) Алгоритм Полларда")

                try:

                    x = pollard\_rho\_discrete\_log(g, h, m, eps)

                    if x == -1:

                        print("Дискретный логарифм не найден!")

                    else:

                        print("Ответ:", x)

                        print(f"Проверка g ^ x = h (mod m) =>

{g} ^ {x} = {h} (mod {m}) =>

{pow(g, x, m)} = {h % m}")

                except Exception:

                    print("Дискретный логарифм не найден!")

                print("3) Вычисление через собственные подгруппы")

                try:

                    x = log\_subgroups(g, h, m)

                    print("Ответ:", x)

                    print(f"Проверка g ^ x = h (mod m) =>

{g} ^ {x} = {h} (mod {m}) =>

{pow(g, x, m)} = {h % m}")

                except Exception:

                    print("Дискретный логарифм не найден!")

                param = None

            case "5":

                param = "5"