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

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 h группы G по основанию g называется число $x \in \{0, 1, \ldots, m - 1\}$, являющееся решением уравнения $g^x = h$.

Будем обозначать дискретный логарифм через $\log_g h$.

Пусть $(G; \cdot)$ – конечная циклическая группа порядка m, g – образующий элемент G и $h \epsilon G$. Дискретным логарифмом элемента h группы G по основанию g называется число $x \epsilon \{0, 1, \ldots, m - 1\}$, являющееся решением уравнения $g^x = h$.

Алгоритм Гельфонда-Шенкса.

Вход: конечная циклическая группа $G = \langle g \rangle$, верхняя оценка для порядка группы $|G| \leq B$, элемент $h \epsilon G$.

Выход: число $x = log_g h$.

1. Вычислить $r = \left[\sqrt{B}\right] + 1$ и вычислить элементы $g^a, 0 \leq a \leq r - 1$, упорядочить массив пар (a, g^a) по второй координате.

2. Вычислить $g_1 = g^{-r}$, для каждого $b, 0 \leq b \leq r - 1$ проверить, является ли элемент $g_1^b h$ второй координатой какой-либо пары из упорядоченного на первом шаге массива пар. Если $g_1^b h = g^a$, то запомнить число $x = a + rb$.

3. Среди всех чисел, найденных на втором этапе, выбрать наименьшее, это значение и будет искомым значением $x = log_g h$.

Сложность метода Гельфонда-Шенкса равна $O(rlogr)$.

Псевдокод алгоритма Гельфонда-Шенкса

Входные данные:

g — генератор конечной циклической группы.

h — элемент, для которого необходимо найти логарифм.

m — порядок группы (значение модульного числа).

Выходные данные:

Значение дискретного логарифма x, такое что $g^x \equiv h \ (mod \ m)$.
Начало алгоритма:

1. Вычислить значение $r = \lceil \sqrt{m} \rceil$.
2. Создать таблицу соответствий для "малых шагов" $\{g^a \ mod \ m : a\}$ для a от 0 до $r - 1$.
3. Вычислить фактор $factor = g^{-r} \ mod \ m$.
4. Для каждого значения b от 0 до $r - 1$:
   4.1. Вычислить $value = (h * factor^b) \ mod \ m$.
   4.2. Если $value$ содержится в таблице малых шагов, вычислить результат $x = b * r + table[value]$.
5. Вернуть минимальное значение x из найденных.

Конец алгоритма

Сложность реализованного метода равна $O(rlogr)$.

3 ρ-метод Полларда вычисления дискретного логарифма

Дана конечная циклическая группа $G = \langle g \rangle$, известен ее порядок $|G| = m$ и $h \in G$.

Метод Полларда применим к любой циклической группе G, чьи элементы представлены таким образом, что их можно разбить на три примерно равные, попарно непересекающиеся части $G = U_1 \cup U_2 \cup U_3$. 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 $O(\sqrt{m})$.

Если $G = Z_p^*$, то можно взять

$$U_1 = \{a \in G \mid 0 < a < \frac{p}{3}\}$$

$$U_2 = \{a \in G \mid \frac{p}{3} \le a < \frac{2p}{3}\}$$

$$U_3 = \{a \in G \mid \frac{2p}{3} \le a < p\}$$

Определим функцию f на G таким образом, что

$$f(a) = \begin{cases} ha, a \in U_1 \\ a^2, a \in U_2. \\ ga, a \in U_3 \end{cases}$$

Будет построена рекуррентная последовательность $y_i = f(y_{i-1}), i \ge 1, y_0 = g^s$. Из определения функции f нетрудно заметить, что при любом $i \ge 0$ $y_i = h^{\beta_i} g^{\alpha_i}$, для некоторых $\alpha_i, \beta_i \in \mathbb{Z}_m$. Также нетрудно заметить, что последовательности $\{\alpha_i\}$, $\{\beta_i\}$ задаются следующими рекуррентными соотношениями:

$$\alpha_0 = s, \alpha_{i+1} = \begin{cases} \alpha_i (mod\ m), & y_i \in U_1; \\ 2\alpha_i (mod\ m), & y_i \in U_2; \\ \alpha_i + 1 (mod\ m), & y_i \in U_3; \end{cases}$$

$$\beta_0 = s, \beta_{i+1} = \begin{cases} \beta_i + 1 (mod\, m), & y_i \in U_1; \\ 2\beta_i (mod\, m), & y_i \in U_2; \\ \beta_i (mod\, m), & y_i \in U_3. \end{cases}$$

При вычислении очередного члена последовательности $y_i = f(y_{i-1})$, числа α_i, β_i вычисляются по известным $\alpha_{i-1}, \beta_{i-1}$ очень легко. При этом для любого $i \geq 0$ выполняется равенство $log_g y_i \equiv \beta_i x + \alpha_i \ (mod\, m)$.

Алгоритм.

Вход: конечная циклическая группа $G = \langle g \rangle$, порядка m, элемент $h \in G$, введенная выше функция $f: G \to G$, число $\varepsilon > 0$.

Выход: число $x = log_g h$ с вероятностью не менее $1 - \varepsilon$.

1. Вычислить $T = \left[\sqrt{2m \ln\left(\frac{1}{\varepsilon}\right)}\right] + 1$;

2. Положить $i = 1$, выбрать случайное $s \in \mathbb{Z}_m$, вычислить $y_0 = g^s, y_1 = f(y_0)$. Запомнить две тройки $(y_0, \alpha_0, \beta_0), (y_1, \alpha_1, \beta_1)$ и перейти к шагу 4;

3. Положить $i = i + 1$, найти $y_i = f(y_{i-1}), y_{2i} = f\big(f(y_{2i-2})\big)$. Запомнить две тройки $(y_i, \alpha_i, \beta_i), (y_{2i}, \alpha_{2i}, \beta_{2i})$ и перейти к шагу 4;

4. Если $y_i \neq y_{2i}$, то проверить выполнение условия $i < T$. Если это условие выполнено, то перейти к шагу 3. В противном случае остановить алгоритм и сообщить, что $x = log_g h$ вычислить не удалось. Если $y_i = y_{2i}$, то перейти к шагу 5;

5. Вычислить $(\beta_i - \beta_{2i}, m) = d$. Если $\sqrt{m} < d \leq m$, то перейти на шаг 2 и выбрать новое значение s. В противном случае решить сравнение $\alpha_{2i} - \alpha_i \equiv (\beta_i - \beta_{2i}) x (mod\, m)$. Если $d = 1$, то единственное решение сравнения равно искомому $log_g h$. Если же $1 < d \leq \sqrt{m}$, то сравнение имеет d различных решений по модулю m. Для каждого из этих решений проверить выполнимость равенства $g^x = h$ и найти истинное решение $x = log_g h$.

Сложность (p)-метода Полларда равна $O(\sqrt{m}\sqrt{\ln\frac{1}{\varepsilon}})$ операций в группе.

Псевдокод ρ-метода Полларда

Входные данные:

g — генератор конечной циклической группы.

h — элемент, для которого требуется найти дискретный логарифм.

m — порядок группы (значение модульного числа).

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

Выходные данные:

Значение дискретного логарифма x, такое что $g^x \equiv h \ (mod\, m)$, либо -1, если решение не найдено.

Начало алгоритма

1. Инициализация переменных:

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

   1.2 Переменная s случайным образом инициализируется в диапазоне $[1, m - 1]$.

   1.3 y, a, b: начальные значения, зависящие от s, h, и g.

2. Обновление значений переменных y, a и b в цикле до тех пор, пока не будет найдено совпадение $y == y1$ (обнаружен цикл).

3. Если найдено совпадение:

   3.1 Вычисляется делитель $gcd(b - b1, m)$ и решается сравнение для нахождения логарифма x.

   3.2 Если $pow(g, x, m) == h$, то найдено решение, возвращается x.

4. Если не найдено решение за определенное количество итераций, возвращается -1.

Конец алгоритма

Сложность (p)-метода Полларда равна $O(\sqrt{m}\sqrt{\ln\frac{1}{\varepsilon}})$

4 Метод вычисления дискретного логарифма с помощью сведения к собственным подгруппам

Покажем, что решение задачи дискретного логарифмирования в циклической группе G, у которой известен порядок $|G| = m$ - составное число, сводится к решению задач дискретного логарифмирования в подгруппах группы G.

Пусть даны G — конечная циклическая группа порядка m, g — образующий элемент G и $h \in G$. Пусть также m — составное число. Тогда либо $m = m_1 m_2$ где $1 < m_1, m_2 < m, (m_1, m_2) = 1$, либо $m = p^n$, где p — простое число, $n \geq 2$.

Рассмотрим сначала первый случай.

Пусть $g_i = g^{m_i}, i = 1, 2$. Тогда G содержит две циклических подгруппы $G_i = \langle g_i \rangle, i = 1, 2$. Нетрудно видеть, что $|G_1| = m_2, |G_2| = m_1$. Причем G_1, G_2 – единственные подгруппы группы G порядков m_2, m_1 соответственно.

Рассмотрим элементы $h_i = h^{m_i}, i = 1, 2$. Так как

$$h_1^{m_2} = h^{m_1 m_2} = h^m = 1, \qquad h_2^{m_1} = h^{m_1 m_2} = h^m = 1,$$

то $ord(h_1) | m_2, ord(h_2) | m_1$, и, следовательно, $h_i \in G_i, i = 1, 2$. Вычислим $x_i = \log_{g_i} h_i$ в группе $G_i, i = 1, 2$. При этом выполняются равенства $g_i^{x_i} = g^{x_i m_i} = h^{m_i}, i = 1,2$. Так как $g^x = h$, то $g^{x m_i} = h^{m_i}, i = 1,2$. Получаем систему сравнений

$$\begin{cases} x m_1 = x_1 m_1 (mod\ m) \\ x m_2 = x_2 m_2 (mod\ m) \end{cases}$$

Которая равносильна системе сравнений

$$\begin{cases} x = x_2\ (mod\ m_1) \\ x = x_1 (mod\ m_2) \end{cases}$$

Так как $(m_1, m_2) = 1$, то из этой системы неизвестный $x = \log_g h$ может быть найден по китайской теореме об остатках.

Рассмотрим теперь второй случай, то есть $m = p^n$, где p – целое число, $n \geq 2$. Представим неизвестный показатель $x = \log_g h \in \{0, \dots, p^n - 1\}$ в $p -$

ичной системе счисления $x = \sum_{i=0}^{n-1} x_i p^i = x_0 + x'p$, где $0 \leq x_i \leq p - 1, 0 \leq x' \leq p^{n-1} - 1$. Алгоритм вычисления $x = \log_g h$ состоит в последовательном вычислении $x_0, \ldots., x_{n-1}$.

Сначала найдем x_0. Для этого вычислим $g_0 = g^{p^{n-1}}$ и $h_0 = h^{p^{n-1}}$. Элемент g_0 имеет порядок p и, следовательно, порождает подгруппу G_0 порядка p в G. При этом из равенства $g^x = h$ вытекает равенство $g_0^{x_0} = h_0$. Действительно, из $g^{(x_0+x'p)p^{n-1}} = g^{(x_0)p^{n-1}} = h^{p^{n-1}}, g_0^{x_0} = h_0$.

Вычислим $x_0 = \log_{g_0} h_0$ в группе G_0. В результате из равенства $g^{x_0+x'p} = h$ получим равенство $g_1^{x'} = h_1$, где $g_1 = g^p, h_1 = hg^{-x_0}$ и $0 \leq x' \leq p^{n-1} - 1$. Так как элемент g_1 имеет порядок p^{n-1}, то он порождает подгруппу G_1 порядка p^{n-1} и $x' = \log_{g_1} h_1$. Таким образом, выполнив одно логарифмирование в G_0, мы свели задачу к вычислению дискретного логарифма в группе G_1 меньшего порядка p^{n-1}. Продолжая процесс таким образом вычислим все $x_0, \ldots, x_{n-1}$, проделав n логарифмирований в G_0.

Сложность алгоритма: $O(\sqrt[4]{m})$

5 Тестирование алгоритмов дискретного логарифмирования

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

```
Введите тип дискретного логарифмирования:

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

:>4
Вычисление с помощью всех алгоритмов
Введите онование g = 2
Элемент группы h = 23
Порядок группы m = 37
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Ответ: 15
Проверка g ^ x = h (mod m) => 2 ^ 15 = 23 (mod 37) => 23 = 23
2) Алгоритм Полларда
Ответ: 15
Проверка g ^ x = h (mod m) => 2 ^ 15 = 23 (mod 37) => 23 = 23
3) Вычисление через собственные подгруппы
Ответ: 15
Проверка g ^ x = h (mod m) => 2 ^ 15 = 23 (mod 37) => 23 = 23
:>
```

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

```
Вычисление с помощью всех алгоритмов
Введите онование g = 3
Элемент группы h = 13
Порядок группы m = 17
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Ответ: 4
Проверка g ^ x = h (mod m) => 3 ^ 4 = 13 (mod 17) => 13 = 13
2) Алгоритм Полларда
Ответ: 4
Проверка g ^ x = h (mod m) => 3 ^ 4 = 13 (mod 17) => 13 = 13
3) Вычисление через собственные подгруппы
Ответ: 4
Проверка g ^ x = h (mod m) => 3 ^ 4 = 13 (mod 17) => 13 = 13
:>
```

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

```
Вычисление с помощью всех алгоритмов
Введите онование g = 7
Элемент группы h = 167
Порядок группы m = 587
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Ответ: 224
Проверка g ^ x = h (mod m) => 7 ^ 224 = 167 (mod 587) => 167 = 167
2) Алгоритм Полларда
Ответ: 517
Проверка g ^ x = h (mod m) => 7 ^ 517 = 167 (mod 587) => 167 = 167
3) Вычисление через собственные подгруппы
Ответ: 224
Проверка g ^ x = h (mod m) => 7 ^ 224 = 167 (mod 587) => 167 = 167
:>
```

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

```
:>4
Вычисление с помощью всех алгоритмов
Введите онование g = 78
Элемент группы h = 765
Порядок группы m = 1579
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Ответ: 762
Проверка g ^ x = h (mod m) => 78 ^ 762 = 765 (mod 1579) => 765 = 765
2) Алгоритм Полларда
Ответ: 762
Проверка g ^ x = h (mod m) => 78 ^ 762 = 765 (mod 1579) => 765 = 765
3) Вычисление через собственные подгруппы
Ответ: 762
Проверка g ^ x = h (mod m) => 78 ^ 762 = 765 (mod 1579) => 765 = 765
:>
```

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

```
:>4
Вычисление с помощью всех алгоритмов
Введите онование g = 2
Элемент группы h = 10
Порядок группы m = 19
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Ответ: 17
Проверка g ^ x = h (mod m) => 2 ^ 17 = 10 (mod 19) => 10 = 10
2) Алгоритм Полларда
Ответ: 17
Проверка g ^ x = h (mod m) => 2 ^ 17 = 10 (mod 19) => 10 = 10
3) Вычисление через собственные подгруппы
Ответ: 17
Проверка g ^ x = h (mod m) => 2 ^ 17 = 10 (mod 19) => 10 = 10
:>
```

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

```
Вычисление с помощью всех алгоритмов
Введите онование g = 2
Элемент группы h = 22
Порядок группы m = 29
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Ответ: 26
Проверка g ^ x = h (mod m) => 2 ^ 26 = 22 (mod 29) => 22 = 22
2) Алгоритм Полларда
Ответ: 26
Проверка g ^ x = h (mod m) => 2 ^ 26 = 22 (mod 29) => 22 = 22
3) Вычисление через собственные подгруппы
Ответ: 26
Проверка g ^ x = h (mod m) => 2 ^ 26 = 22 (mod 29) => 22 = 22
:>
```

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

```
Вычисление с помощью всех алгоритмов
Введите онование g = 2
Элемент группы h = 23
Порядок группы m = 90
Укажите e (0 < e < 1): 0.5
1) Алгоритм Гельфонда
Дискретный логарифм не найден!
2) Алгоритм Полларда
Дискретный логарифм не найден!
3) Вычисление через собственные подгруппы
Дискретный логарифм не найден!
:>
```

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 АРВ, 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

```python
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)
```

```python
        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:
```

```python
        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 дикскретного 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"
```