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 конечном поле

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
Алексеева Александра Александровича

Преподаватель
профессор, д.ф.-м.н. _____ В. А. Молчанов
подпись, дата

Саратов 2023

1 Постановка задачи

Цель работы – изучение основных методов дискретного логарифмирования в конечном поле 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 в конечных полях.

2 Теория

Пусть $G = \langle a \rangle$ — конечная циклическая группа порядка m, т.е.

$$G = \{a^0 = 1, a^1 = a, a^2, .., a^{m-1}\}.$$

Определение. *Дискретным логарифмом* элемента $b \in G$ называется число $x \in \{0,1,\ldots,m-1\}$, для которого

$$a^x = b.$$

Обозначается $x = \log_a b$.

Задача нахождения дискретного логарифма имеет большую сложность вычислений.

2.1 Метод Гельфонда-Шенкса

Алгоритм Гельфонда-Шенкса вычисления дискретного логарифма в произвольной циклической группе, элементы которой линейно упорядочены

Вход. Конечная линейно упорядоченная группа $G = \langle a \rangle$, верхняя оценка порядка группы $|G| \leq B$ и $b \in G$.

Выход. $x = \log_a b$.

Шаг 1. Вычислить $r = \left[\sqrt{B}\right] + 1$. Вычислить элементы $a, a^2, .., a^{r-1}$ и упорядочить по второй координате множество пар $(k, a^k), 1 \leq k \leq r-1$;

Шаг 2. Вычислить $a_1 = a^{-r}$. Для каждого $0 \leq i \leq r-1$ вычислить a_1^i и проверить, является ли элемент $a_1^i b$ второй координатой какой-нибудь пары из упорядоченного множества, построенного на шаге 1. Если $a_1^i b = a^k$, то $a^{-ri} b = a^k, b = a^k a^{ri} = a^{k+ri}$ запомнить $k + ri$;

Шаг 3. Найти число x, равное наименьшему значению среди чисел $k + ri$, вычисленных на предыдущем шаге. В результате получаем $x = \log_a b$.

Замечание. При $B = |G|$ шаг 3 можно пропустить.

Сложность алгоритма: на шагах 1,2 $O(r \log r)$ операций в группе G – в результате $O(\sqrt{B} \log B)$.

Псевдокод алгоритма Гельфонда-Шенкса

Ввод m – порядок конечной циклической группы, a – образующий элемент группы, b – элемент группы
Если (m не простое)
    Вывести «Число m должно быть простым»
Если (a не является образующим элементом)
    вывести «a не является образующим элементом»
$r = [\sqrt{m}] + 1$
Создаётся список as
В цикле по i от 1 до $r - 1$:
    Добавить пару (i, a^i) в список as
Отсортировать список as по второму элементу
$a_1 = a^{-r} \pmod m$
Создаём список k_ri, в котором будут хранится значения $k + ri$
В цикле по i от 1 до $r - 1$:
    $a_1 b = a_1^i b$
    В цикле по всему списку as:
        Если ($a^k = a_1 b$)
            Добавить значение $k + ri$ в список k_ri
Вернуть наименьшее значение в списке k_ri

2.2 ρ-метод Полларда

Дана конечная циклическая группа $G = \langle a \rangle$ порядка m и элемент $b \in G$. Причем группа разбита на три примерно равные части U_1, U_2, U_3 с простым алгоритмом проверки вхождения элементов в эти части.

Определяется преобразование $f: G \to G$ для элементов $x \in G$ по формуле:

$$f(x) = \begin{cases} bx, \text{если } x \in U_1, \\ x^2, \text{если } x \in U_2, \\ ax, \text{если } x \in U_3. \end{cases}$$

Для случайно выбранного значения $s \in \boldsymbol{Z}_m$ рассматривается рекуррентная последовательность:

$$y_i = f(y_{i-1}), i \geq 1, y_0 = a^s.$$

Тогда $y_i = a^{\alpha_i} b^{\beta_i}$ для рекуррентно заданных последовательностей:

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

$$\beta_0 = 0, \beta_{i+1} = \begin{cases} \beta_i + 1 \ (mod\ m), \text{если } y_i \in U_1, \\ 2\beta_i \ (mod\ m), \text{если } y_i \in U_2, \\ \beta_i \ (mod\ m), \text{если } y_i \in U_3. \end{cases}$$

Так как при этом

$$y_i = a^{\alpha_i} b^{\beta_i} = a^{\alpha_i} (a^x)^{\beta_i} = a^{\alpha_i + \beta_i x},$$

то выполняется

$$\log_a y_i = \beta_i x + \alpha_i \ (mod\ m).$$

Алгоритм ρ-метода Полларда вычисления дискретного логарифма в произвольной циклической группе

Вход. Конечная группа $G = \langle a \rangle$ порядка m, элемент $b \in G$, определенная выше функция $f: G \to G$ и число $\varepsilon > 0$.

Выход. $x = \log_a b$ с вероятностью не менее $1 - \varepsilon$.

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

Шаг 2. Положить $i = 1$, выбрать случайное $s \in Z_m$ и вычислить $y_0 = a^s,\ y_1 = f(y_0)$. Запомнить две тройки $(y_0, \alpha_0, \beta_0), (y_1, \alpha_1, \beta_1)$ и перейти к шагу 3;

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

Шаг 4. Если $y_i \neq y_{2i}$, то проверить условие $i < k$. Если это условие выполнено, то перейти к шагу 3. В противном случае закончить вычисления и сообщить, что значение $x = \log_a b$ вычислить не удалось.

Если же $y_i = y_{2i}$, то

$$\log_a y_i = \beta_i x + \alpha_i = \log_a y_{2i} = \beta_{2i} x + \alpha_{2i} \ (mod\ m),$$

$$\alpha_{2i} - \alpha_i \equiv (\beta_i - \beta_{2i}) x \ (mod\ m).$$

И для решения сравнения перейти к шагу 5;

Шаг 5. Вычислить $\text{НОД}(\beta_i - \beta_{2i}, m) = d$. Если $\sqrt{m} < d < m$, то перейти на шаг 2 и выбрать новое значение $s \in Z_m$.

В противном случае решить сравнение

$$\alpha_{2i} - \alpha_i \equiv (\beta_i - \beta_{2i}) x \ (mod\ m).$$

Если $d = 1$, то единственное решение последнего сравнения равно значению $\log_a b$. Если $1 < d \leq \sqrt{m}$, то последнее сравнение имеет d различных решений по модулю m. Для каждого из этих решений проверить выполнимость равенства $a^x = b$ и найти истинное значение $x = \log_a b$.

Обоснование. Применим теорему о «парадоксе дней рождений» к последовательности $\{y_i\}$, $0 \le i \le k$. Тогда для

$$\lambda = \ln\frac{1}{\varepsilon}, S = G, |S| = |G| = m, k = \left[\sqrt{2\sqrt{m}\ln\frac{1}{\varepsilon}}\right] + 1$$

среди членов последовательности $\{y_i\}$, $0 \le i \le k$ с вероятностью не менее $1 - e^{\lambda} = 1 - \varepsilon$ найдутся совпадающие члены $y_i = y_j, 0 \le i < j \le k$.

Значит, в ходе работы алгоритма с вероятностью не менее $1 - \varepsilon$ будет построена пара $y_i = y_{2i}$ и в силу равенства $\log_a y_i = \beta_i x + \alpha_i \ (mod\ m)$ выполняется $\alpha_{2i} - \alpha_i \equiv (\beta_i - \beta_{2i})x \ (mod\ m)$. *Это сравнение разрешимо и имеет ровно* НОД$(\beta_i - \beta_{2i}, m) = d$ решений (число которых ограничено значением $\sqrt{m}$).

Сложность вычислений: $O(\sqrt{m}\sqrt{\ln\frac{1}{\varepsilon}})$ операций в группе G.

Псевдокод ρ-метода Полларда

def newThrees(*m*, *a*, *b*, список *elPrev*):

*u*1 = *m* / 3, *u*2 = *u*1 * 2, *u*3 = *m*

Если (*elPrev*[0] < *u*1)

Вернуть список [*b* * *elPrev*[0] (mod *m*), *elPrev*[1], *elPrev*[2] + 1 (mod *m*-1)

Если (*elPrev*[0] < *u*2)

Вернуть [*elPrev*[0]^2 (mod *m*), 2 * *elPrev*[1] (mod *m*–1), 2 * *elPrev*[2] (mod *m*-1)

Если (*elPrev*[0] < *u*3)

Вернуть [*a* * *elPrev*[0] (mod *m*), *elPrev*[1] + 1 (mod *m*-1), *elPrev*[2]]

Ввод m – порядок конечной циклической группы, a – образующий элемент группы, b – элемент группы, ε

Если (m не простое)

Вывести «Число m должно быть простым»

Если (a не является образующим элементом)

вывести «a не является образующим элементом»

k = $\left[\sqrt{2\sqrt{m}ln\frac{1}{\varepsilon}}\right] + 1$

step2:

s = Random(0, *m* – 1)

Создать список *threes* размером $2k + 2$ = [[a^s (mod m), s, 0], newThrees(m, a, b, threes[0], [], …, []]

В цикле по *i* от 2 до *k* включительно:

threes[*i*] = newThrees(*m*, *a*, *b*, *threes*[*i* – 1])

helpThrees = newThrees(*m*, *a*, *b*, *threes*[2*i* – 2])

threes[2*i*] = newThrees(*m*, *a*, *b*, *helpThrees*)

В цикле по *i* от 2 до *k* включительно:

d = НОД(*threes*[*i*][2] – *threes*[2*i*][2], *m* – 1)

Если ($\sqrt{m} < d < m$)

Перейти на *step2*

Создать список *res*

В цикле по *x* от 1 до *m* или пока размер *res* ≠ *d*:

Если (threes[*i*][2] – threes[2*i*][2] * *x* (mod *m*-1) = threes[*i*][1] – threes[2*i*][1] (mod *m*-1))

Добавить в список *res* значение *x*

В цикле по *j* от 0 до размера *res*:

Если ($a^{res[j]}$ (mod *m*) = *b*)

Вернуть *res*[*j*]

Вывести «Дискретный логарифм вычислить не удалось»

2.3 Метод вычисления дискретного логарифма в конечных полях

Даны g – образующий элемент группы $GF(p)^*$ и $h \in GF(p)^*$. Требуется найти $x = \log_g h$.

Будем считать, что $GF(p) = \boldsymbol{Z}_p$.

Пусть B – некоторое натуральное число, параметр метода. Определим факторную базу $S_B = \{2,3,5,..,q\}$ – множество первых простых чисел, не превосходящих $B, |S_B| = \pi(B)$. Значение параметра B выбирается таким образом, чтобы минимизировать сложность алгоритма.

Алгоритм индекс-метода логарифмирования в конечном простом поле

Вход. Простое нечетное число p, $Z_p^* = \langle g \rangle, h \in Z_p^*$.

Выход. Значение $x = \log_g h$.

Шаг 1. Выбрать значение параметра B. Построить факторную базу S_B;

Шаг 2. Выбрать случайное m, $0 \le m \le p-2$, найти вычет $b \in Z_p^*$, $b \equiv g^m \ (mod\ p)$;

Шаг 3. Проверить число b на B-гладкость. Если b является B-гладким, то вычислить его каноническое разложение $b = \prod_{i=1}^{\pi(B)} q_i^{l_i}$ и запомнить строку $(l_1, l_2, .., l_{\pi(B)})$.

Из соотношений

$$\begin{cases} b = \prod_{i=1}^{\pi(B)} q_i^{l_i} = g^{\sum_{i=1}^{\pi(B)} l_i \log_g q_i} \ (mod\ p) \\ b \equiv g^m \ (mod\ p) \end{cases}$$

вытекает сравнение

$$m \equiv \sum_{i=1}^{\pi(B)} l_i x_i \ (mod\ p-1),$$

где $x_i = \log_g q_i$.

Повторять шаги 2 и 3 до тех пор, пока число найденных строк не превысит $N = \pi(B) + \delta$, где δ — некоторая небольшая константа.

В результате будет построена система линейных уравнений над кольцом $\boldsymbol{Z}_{p-1}$ относительно неизвестных $x_i = \log_g q_i \,, q_i \in S_B$:

$$m_j \equiv \sum_{i=1}^{\pi(B)} l_{ji} x_i \ (mod\ p-1), 1 \le j \le N.$$

Полученная система заведомо совместна;

Шаг 4. Решить полученную на предыдущем шаге систему линейных уравнений над кольцом $\boldsymbol{Z}_{p-1}$ методом Гаусса. Если система имеет более одного решения, то вернуться на шаг 2 и получить несколько новых линейных соотношений. Затем вернуться к шагу 4;

Шаг 5. (Вычисление индивидуального логарифма). Выбрать случайное $m, 0 \le m \le p-2$, найти вычет $b \equiv hg^m \ (mod\ p), b \in Z_p^*$. Проверить число b на B-гладкость. Если b является B-гладким, то

$$\begin{cases} b = \prod_{i=1}^{\pi(B)} q_i^{r_i} = g^{\sum_{i=1}^{\pi(B)} r_i \log_g q_i} \ (mod\ p) \\ b \equiv hg^m = g^x g^m = g^{x+m} \ (mod\ p) \end{cases}$$

и, следовательно,

$$x \equiv -m + \sum_{i=1}^{\pi(B)} r_i x_i \ (mod\ p-1).$$

При $p \to \infty$ оптимальное значение $B = L_p[1/2]$ и сложность всего алгоритма оценивается величиной $L_p[2]$, где

$$L_p[c] = L_p\left[\frac{1}{2}, c\right] = \exp\left(\left(c + o(1)\right)(\log p \log\log p)^{\frac{1}{2}}\right) = L^{c+o(1)}$$ для

$L = \exp\left((\log p \log\log p)^{\frac{1}{2}}\right)$.

Замечание. На шаге 4 система линейных уравнений решается методом Гаусса. Так как $\boldsymbol{Z}_{p-1}$ не является полем и имеются ненулевые необратимые элементы в $\boldsymbol{Z}_{p-1}$, то осуществляется другой подход к приведению элементов на главной диагонали к единице. Для того, чтобы элемент *x* на главной диагонали был равен 1, необходимо пройтись по столбцу и найти в нём такой элемент *y*, что НОД(*x*, *y*) = 1. После этого, с помощью расширенного алгоритма Евклида нужно получить такие коэффициенты *a* и *b*, что $x \cdot a + y \cdot b = 1$. Затем строки СЛУ должны быть умножены на коэффициенты a, b и сложены, чтобы элемент на главной диагонали был равен 1.

Псевдокод индекс-метода логарифмирования в конечном простом поле

```
Функция factorizationNum(base, n):
    Создаётся список res = [[base[0], 0], [base[1], 0], …, [base[-1], 0]]
    В цикле по i от 0 до размера base:
        Пока (n (mod base[i]) ≠ 0):
            base[i][1] = base[i][1] + 1
            n = n / base[i]
    Вернуть список [res, n]

Ввод p – порядок конечной циклической группы, g – образующий элемент группы, h – элемент группы
Если (p не простое)
    Вывести «Число p должно быть простым»
```

Если (g не является образующим элементом)
    вывести «g не является образующим элементом»

B = если $p < 8193$ то 13 иначе $\log2(p)$

step2:

Создать список *canonDecompose*

Пока (размер *canonDecompose* ≠ размеру *base* + 10):

$m = \text{Random}(1, p - 2)$

$b = g^m \pmod p$

factorsNum = *factorizationNum*(*base*, b)

Если (*factorsNum*[1] ≠ 1)

Перейти к следующей итерации

Создать список *degs*

Цикл по i от 0 до размера *factorsNum*:

Добавить в *degs* значение *factorsNum*[0][i][1]

Добавить в список *degs* значение m

Добавить в список *canonDecompose* список *degs*

Если (СЛУ *canonDecompose* не имеет решения)

Перейти на *step2*

Цикл по i от 0 до размера *canonDecompose*[0] – 1:

Если (*canonDecompose*[i][i] = 0)

Цикл по j от i + 1 до размера *canonDecompose*:

Если (*canonDecompose*[j][i] ≠ 0)

swap(*canonDecompose*[i], *canonDecompose*[j])

break

Решить СЛУ *canonDecompose* и записать в *solSLU*

Бесконечный цикл:

$m = \text{Random}(0, p - 2)$

$b = g^m \cdot h \pmod p$

factorsNum = *factorizationNum*(*base*, b)

Если (*factorsNum*[1] ≠ 1)

Перейти к следующей итерации

res = 0

Цикл по i от 0 до размера *base*:

res = *res* + *factorsNum*[0][i][1] * *solSLU*[i] $\pmod{p - 1}$

res = *res* – $m \pmod{p - 1}$

Вернуть *res* в качестве результата

3 Результаты работы

3.1 Тестирование программы

```
        Дискретное логарифмирование в конечном поле
1 - Метод Гельфонда-Шенкса
2 - ро-метод Полларда
3 - индекс-метод
1

Порядок конечной циклической группы p: 997
Образующий элемент группы a: 29
Элемент группы b: 423

x = 19
Проверка: a^x (mod m) = 423
```

```
1 - Метод Гельфонда-Шенкса
2 - ро-метод Полларда
3 - индекс-метод
2

Порядок конечной циклической группы p: 37
Образующий элемент группы a: 2
Элемент группы b: 23
Значение epsilon: 0.5

x = 15
Проверка: a^x (mod m) = 23
```

```
1 - Метод Гельфонда-Шенкса
2 - ро-метод Полларда
3 - индекс-метод
3

Порядок конечной циклической группы p: 999983
Образующий элемент группы a: 41
Элемент группы b: 53425

СЛУ:
(0, 5, 1, 2, 1, 0, 0, 0, 6999)
(0, 5, 1, 0, 0, 1, 0, 0, 12257)
(0, 2, 2, 0, 0, 2, 1, 0, 17410)
(0, 1, 1, 0, 1, 1, 0, 0, 13861)
(4, 0, 2, 1, 0, 0, 1, 0, 6248)
(3, 2, 1, 2, 0, 1, 0, 0, 4649)
(1, 6, 1, 0, 0, 0, 0, 1, 3119)
(2, 4, 0, 1, 1, 1, 0, 0, 30542)
(2, 2, 0, 0, 0, 0, 1, 0, 5438)
(0, 3, 0, 1, 1, 0, 1, 1, 5228)
(3, 3, 1, 2, 0, 0, 0, 0, 14467)
(1, 2, 0, 0, 2, 0, 2, 0, 5012)
(4, 0, 0, 0, 1, 2, 0, 0, 558)
(0, 3, 1, 1, 0, 2, 0, 0, 28653)
(2, 3, 0, 1, 0, 0, 1, 1, 9672)
(2, 3, 1, 1, 0, 0, 0, 1, 32357)
(2, 0, 1, 0, 0, 0, 2, 0, 20409)
(5, 1, 0, 0, 0, 1, 2, 0, 4082)
```

```
Разрешённая СЛУ:
(1, 999980,      1, 999981,      0, 999981,      1,      0,   1599)
(0,      1, 999979,      0,      0, 999979, 999980,      0, 977419)
(0,      0,      1,      1,      1,      1,      2,      1,  10381)
(0,      0,      0,      1, 333312, 666686, 999965, 999972, 372052)
(0,      0,      0,      0,      1, 941128, 539235, 555554, 534948)
(0,      0,      0,      0,      0,      1, 676934, 900065, 855378)
(0,      0,      0,      0,      0,      0,      1, 403590, 934334)
(0,      0,      0,      0,      0,      0,      0,      1, 405656)

Решение СЛУ: (575370, 786164, 305001, 812378, 146314, 776346, 282316, 405656)
x = 328586
Проверка: a^x (mod m) = 53425
```

3.2 Код программы

```
#include <iostream>
#include <vector>
#include <string>
#include <set>
#include <map>
#include <cmath>
#include <iomanip>
#include "Pattern.cpp"
#include "boost/multiprecision/cpp_int.hpp"
#include <boost/multiprecision/cpp_dec_float.hpp>

using namespace std;
using namespace boost::multiprecision;

class Pattern
{
private:
	static vector <long long> deg2(long long el, long long n)
{//Раскладываем число на степени двойки
		vector <long long> res;
		while (n != 0) {
			if (n / el == 1) {
				res.push_back(el);
				n -= el;
				el = 1;
			}
			else
				el *= 2;
		}
		return res;
	}


	static long long multMod(long long n, long long mod, vector <pair <long
long, long long>> lst) {//Умножаем число по модулю
		if (lst.size() == 1) {
			long long res = 1;
			for (unsigned short i = 0; i < lst[0].second; i++)
				res = res * lst[0].first % mod;
			return res;
		}
		else if (lst[0].second == 1) {
			long long el = lst[0].first;
			lst.erase(lst.begin());
```

```
                        return (el * multMod(n, mod, lst)) % mod;
                }
                else {
                        for (unsigned short i = 0; i < lst.size(); i++)
                                if (lst[i].second > 1) {
                                        lst[i].first = (lst[i].first *
lst[i].first) % mod;
                                        lst[i].second /= 2;
                                }
                        return multMod(n, mod, lst);
                }
        }


        static long long partition(vector <pair <short, long long>>& a, long
long start, long long end) {
                long long pivot = a[end].second;
                long long pIndex = start;

                for (unsigned short i = start; i < end; i++) {
                        if (a[i].second <= pivot) {
                                swap(a[i], a[pIndex]);
                                pIndex++;
                        }
                }

                swap(a[pIndex], a[end]);
                return pIndex;
        }

public:
        static long long powClosed(long long x, long long y, long long mod)
{//Возводим число в степени по модулю
                if (y == 0)
                        return 1;

                vector <long long> lst = deg2(1, y);
                vector <pair <long long, long long>> xDegs;
                for (unsigned short i = 0; i < lst.size(); i++)
                        xDegs.push_back(make_pair(x, lst[i]));

                long long res = multMod(x, mod, xDegs);
                return res;
        }


        static bool miller_rabin(long long n, long long k = 10) {
                if (n == 0 || n == 1)
                        return false;
                else if (n == 2 || n == 3)
                        return true;

                long long d = n - 1;
                long long s = 0;
                while (d % 2 == 0) {
                        s++;
                        d = d / 2;
                }

                long long nDec = n - 1;
                for (unsigned short i = 0; i < k; i++) {
                        long long a = rand() % nDec;
                        if (a == 0 || a == 1)
                                a = a + 2;
```

```
                        long long x = powClosed(a, d, n);
                        if (x == 1 || x == nDec)
                                continue;

                        bool flag = false;
                        for (unsigned short j = 0; j < s; j++) {
                                x = (x * x) % n;
                                if (x == nDec) {
                                        flag = true;
                                        break;
                                }
                        }
                        if (!flag)
                                return false;
                }

                return true;
        }


        static void quicksort(vector <pair <short, long long>>& a, long long
start, long long end) {
                if (start >= end) {
                        return;
                }

                long long pivot = partition(a, start, end);
                quicksort(a, start, pivot - 1);
                quicksort(a, pivot + 1, end);
        }


        static long long usualEuclid(long long a, long long b) {
                if (a < b)
                        swap(a, b);
                if (a < 0 || b < 0)
                        throw string{ "Выполнение невозможно: a < 0 или b < 0" };
                else if (b == 0)
                        return a;

                long long r = a % b;
                return usualEuclid(b, r);
        }


        static pair <long long, long long> advancedEuclid(long long a, long long
b) {
                if (a < 0 || b < 0)
                        throw string{ "Выполнение невозможно: a < 0 или b < 0" };

                long long q, aPrev = a, aCur = b, aNext = -1;
                long long xPrev = 1, xCur = 0, xNext;
                long long yPrev = 0, yCur = 1, yNext;
                while (aNext != 0) {
                        q = aPrev / aCur;
                        aNext = aPrev % aCur;
                        aPrev = aCur; aCur = aNext;

                        xNext = xPrev - (xCur * q);
                        xPrev = xCur; xCur = xNext;

                        yNext = yPrev - (yCur * q);
                        yPrev = yCur; yCur = yNext;
```

```
                }

                return make_pair(xPrev, yPrev);
        }
};



class discrLogarithm
{
private:
        //Проверка первообразного элемента
        static void checkPrimitEl(long long g, long long m)
        {
                if (g >= m)
                        throw string{ "Первообразный элемент превышает порядок!"
};

                long long x = g;
                for (long long i = 2; i < m; i++)
                {
                        x = (x * g) % m;
                        if (i != m - 1 && x == 1)
                                throw string{ "Элемент " + to_string(g) + " не
является образующим элементом!" };
                }
        }


        static pair <short, long long> findEl(vector <pair <short, long long>>
as, long long el)
        {
                for (unsigned short i = 0; i < as.size(); i++)
                        if (as[i].second == el)
                                return as[i];
                return make_pair(0, 0);
        }


        //Вычисляется новая тройка во 2-ом алгоритме
        static vector <long long> newThrees(long long m, long long a, long long
b, vector <long long> elPrev)
        {
                long long u1 = m / 3.0;
                long long u2 = u1 * 2;
                long long u3 = m;

                if (elPrev[0] < u1)
                        return vector <long long> {b * elPrev[0] % m, elPrev[1],
(elPrev[2] + 1) % (m - 1)};
                else if (elPrev[0] < u2)
                        return vector <long long> {elPrev[0] * elPrev[0] % m, 2 *
elPrev[1] % (m - 1), 2 * elPrev[2] % (m - 1) };
                else
                        return vector <long long> {a * elPrev[0] % m, (elPrev[1]
+ 1) % (m - 1), elPrev[2]};
        }


        //Создаётся факторная база
        static vector <short> createBase(short B)
        {
                vector <short> base;
                for (unsigned short i = 1; i <= B; i++)
```

```
                        if (Pattern::miller_rabin(i))
                                base.push_back(i);
                return base;
        }


        //Каноническое представление числа относительно факторной базы
        static pair <map <short, long long>, short> factorizationNum(vector
<short> base, long long n)
        {
                pair <map <short, long long>, short > res;
                for (unsigned short i = 0; i < base.size(); i++)
                        res.first.insert(make_pair(base[i], 0));

                for (unsigned short i = 0; i < base.size(); i++)
                        while (n % base[i] == 0)
                        {
                                res.first[base[i]] += 1;
                                n /= base[i];
                        }

                res.second = n;
                return res;
        }


        //Проверка, имеет ли СЛУ решение
        static bool haveSolutionSLU(vector <vector <long long>> matrix)
        {
                for (unsigned short i = 0, rawSize = matrix[0].size() - 1; i <
rawSize; i++)
                {
                        bool flag = false;
                        for (unsigned short j = 0, colSize = matrix.size(); j <
colSize; j++)
                                if (matrix[j][i] > 0) {
                                        flag = true;
                                        break;
                                }
                        if (!flag)
                                return false;
                }
                return true;
        }


        //Перестановка строк, чтобы на главной диагонали не было нулей
        static bool swapColms(vector <vector <long long>>& matrix, short
indexColm)
        {
                for (unsigned short i = indexColm + 1; i < matrix.size(); i++)
                        if (matrix[i][indexColm] != 0)
                        {
                                swap(matrix[indexColm], matrix[i]);
                                return true;
                        }
                return false;
        }


        //Умножение строки на число
        static vector <long long> multRawToNum(vector <long long> raw, long long
num, long long field) {
                vector <long long> res;
```

```
                for (unsigned short i = 0, rawSize = raw.size(); i < rawSize;
i++)
                {
                        int x = (raw[i] * num) % field;
                        while (x < 0)
                                x += field;
                        res.push_back(x);
                }
                return res;
        }


        //Сложение строк
        static vector <long long> addRaws(vector <long long> a, vector <long
long> b, long long field) {
                vector <long long> res;
                for (unsigned short i = 0; i < a.size(); i++) {
                        long long x = (a[i] + b[i]) % field;
                        while (x < 0)
                                x += field;
                        res.push_back(x);
                }
                return res;
        }

public:
        static long long gelfondThanks(long long m, long long a, long long b)
        {
                if (!Pattern::miller_rabin(m))
                        throw string{ "Порядок группы " + to_string(m) + " не
является простым числом!" };
                checkPrimitEl(a, m);

                long long r = round(std::sqrt(m)) + 1;
                vector <pair <short, long long>> as;
                for (unsigned short i = 1; i < r; i++)
                        as.push_back(make_pair(i, Pattern::powClosed(a, i, m)));
                Pattern::quicksort(as, 0, as.size() - 1);

                long long a1 = (Pattern::powClosed(Pattern::advancedEuclid(a,
m).first, r, m) + m) % m;
                set <long long> k_ri;
                for (unsigned short i = 0; i < r; i++)
                {
                        long long a1i = Pattern::powClosed(a1, i, m);
                        pair <short, long long> k_ak = findEl(as, a1i * b % m);
                        if (k_ak.first != 0)
                                k_ri.insert((k_ak.first + r * i) % m);
                }

                for (auto i = k_ri.begin(); i != k_ri.end(); i++)
                        if (Pattern::powClosed(a, *i, m) == b)
                                return *i;
                for (unsigned short i = 1;; i++)
                        if (Pattern::powClosed(a, i, m) == b)
                                return i;
        }


        static long long roMethodPollarda(long long m, long long a, long long b,
double eps)
        {
                if (!Pattern::miller_rabin(m))
```

```
                        throw string{ "Порядок группы " + to_string(m) + " не
является простым числом!" };
                checkPrimitEl(a, m);

                long long k = round(std::sqrt(2 * std::sqrt(m - 1) * log(1 /
eps))) + 1;
                unsigned short count = 0;
        step2:
                count++;
                long long s = rand() % (m - 1);

                vector <vector <long long>> threes(2 * k + 2);
                threes[0] = { Pattern::powClosed(a, s, m), s, 0 };
                threes[1] = newThrees(m, a, b, threes[0]);
                for (unsigned short i = 2; i <= k; i++)
                {
                        threes[i] = newThrees(m, a, b, threes[i - 1]);
                        vector <long long> helpThrees = newThrees(m, a, b,
threes[2 * i - 2]);
                        threes[2 * i] = newThrees(m, a, b, helpThrees);
                }

                for (unsigned short i = 2; i <= k; i++)
                        if (threes[i][0] == threes[2 * i][0])
                        {
                                long long mDec = m - 1;
                                long long d = Pattern::usualEuclid((threes[i][2]
- threes[2 * i][2] + mDec) % mDec, mDec);
                                if (std::sqrt(m) < d && d <= m)
                                        goto step2;

                                long long alphaRes = (threes[2 * i][1] -
threes[i][1] + mDec) % mDec;
                                long long betaRes = (threes[i][2] - threes[2 *
i][2] + mDec) % mDec;
                                vector <long long> res;
                                for (long long x = 1; x < m && res.size() != d;
x++)
                                        if (betaRes * x % mDec == alphaRes)
                                                res.push_back(x);

                                for (unsigned short j = 0; j < res.size(); j++)
                                        if (Pattern::powClosed(a, res[j], m) == b)
                                                return res[j];
                        }
                if (count < m / eps)
                        goto step2;

                throw string{ "Дискретный логарифм вычислить не удалось!" };
                return 0;
        }


        static long long indexMethod(long long p, long long a, long long h)
        {
                if (!Pattern::miller_rabin(p))
                        throw string{ "Порядок группы " + to_string(p) + " не
является простым числом!" };
                checkPrimitEl(a, p);

                short B = p < 8193 ? 13 : round(log2(p));
                vector <short> base = createBase(B);

        step2:
```

```
		vector <vector <long long>> canonDecompose;
		while (canonDecompose.size() != base.size() + 10)
		{
			int m = rand() % (p - 2) + 1;
			long long b = Pattern::powClosed(a, m, p);

			pair <map <short, long long>, long long> factorsNum =
factorizationNum(base, b);
			if (factorsNum.second != 1)
				continue;

			vector <long long> degs;
			for (auto i = factorsNum.first.begin(); i !=
factorsNum.first.end(); i++)
				degs.push_back(i->second);
			degs.push_back(m);
			canonDecompose.push_back(degs);
		}
		vector <vector <long long>> copyCanonDecompose = canonDecompose;

		if (!haveSolutionSLU(canonDecompose))
			goto step2;
		for (unsigned short i = 0; i < canonDecompose[0].size() - 1; i++)
			if (canonDecompose[i][i] == 0)
				if (!swapColms(canonDecompose, i))
					goto step2;

		long long pDec = p - 1;
		for (unsigned short i = 0; i < canonDecompose[0].size(); i++)
		{
			if (canonDecompose[i][i] > 1) {
				for (unsigned short j = i + 1; j <
canonDecompose.size(); j++)
					if (canonDecompose[j][i] != 0 &&
Pattern::usualEuclid(canonDecompose[i][i], canonDecompose[j][i]) == 1)
					{
						pair <long long, long long> coefs
= Pattern::advancedEuclid(canonDecompose[i][i], canonDecompose[j][i]);
						canonDecompose[i] =
addRaws(multRawToNum(canonDecompose[i], coefs.first, pDec),

			multRawToNum(canonDecompose[j], coefs.second, pDec),

			pDec);

						break;
					}
				if (canonDecompose[i][i] > 1)
					goto step2;
			}
			for (unsigned short k = i + 1; k < canonDecompose.size();
k++)
			{
				canonDecompose[k] = addRaws(canonDecompose[k],
multRawToNum(canonDecompose[i], -canonDecompose[k][i], pDec), pDec);
				if (k < canonDecompose[0].size() - 1 &&
canonDecompose[k][k] == 0)
				{
					if (!swapColms(canonDecompose, k))
						goto step2;
					k--;
				}
			}
		}
```

```
		while (canonDecompose.size() != base.size())
			canonDecompose.pop_back();

		cout << "\nСЛУ: ";
		for (unsigned short i = 0, size = copyCanonDecompose.size(); i <
size; i++)
		{
			string res = "\n(";
			for (unsigned short j = 0, iSize =
copyCanonDecompose[i].size(); j < iSize; j++)
				res += j == copyCanonDecompose[i].size() - 1 ?
to_string(copyCanonDecompose[i][j]) + ")" : to_string(copyCanonDecompose[i][j])
+ ", ";
			cout << res;
		}

		cout << "\n\nРазрешённая СЛУ: ";
		for (unsigned short i = 0, sizeP = to_string(p).length(), size =
canonDecompose.size(); i < size; i++)
		{
			cout << "\n(" << canonDecompose[i][0] << ", ";
			for (unsigned short j = 1, iSize =
canonDecompose[i].size(); j < iSize; j++)
			{
				if (j == canonDecompose[i].size() - 1)
					cout << setw(sizeP) <<
canonDecompose[i][j] << ")";
				else
					cout << setw(sizeP) <<
canonDecompose[i][j] << ", ";
			}
		}

		vector <long long> solSLU(canonDecompose.size(), 0);
		for (short i = canonDecompose.size() - 1; i >= 0; i--)
			for (short j = i + 1; j < canonDecompose[i].size(); j++)
			{
				solSLU[i] = j == canonDecompose[i].size() - 1 ?
(solSLU[i] + canonDecompose[i][j]) % pDec : solSLU[i] - 1 * canonDecompose[i][j]
* solSLU[j];
				while (solSLU[i] < 0)
					solSLU[i] += pDec;
			}

		cout << "\n\nРешение СЛУ: (";
		for (unsigned short i = 0, sizeSolSLU = solSLU.size(); i <
sizeSolSLU; i++)
		{
			if (i == sizeSolSLU - 1)
				cout << solSLU[i] << ")";
			else
				cout << solSLU[i] << ", ";
		}


		while (true)
		{
			int m = rand() % (p - 2) + 1;
			long long b = Pattern::powClosed(a, m, p) * h % p;

			pair <map <short, long long>, long long> factorsNum =
factorizationNum(base, b);
			if (factorsNum.second != 1)
```

```
                                continue;

                        long long res = 0;
                        for (unsigned short i = 0; i < base.size(); i++)
                                res = (res + factorsNum.first[base[i]] *
solSLU[i]) % (p - 1);
                        res = (res - m) % (p - 1);
                        return res;
                }
        }
};



int main()
{
        srand(time(NULL));
        setlocale(LC_ALL, "ru");
        cout << "\tДискретное логарифмирование в конечном поле";

        unsigned short choice;
        for (;;)
        {
                cout << "\n1 - Метод Гельфонда-Шенкса \n2 - ро-метод Полларда \n3
- индекс-метод \n";
                cin >> choice;

                long long p, a, b;
                cout << "\nПорядок конечной циклической группы p: ";
                cin >> p;
                cout << "Образующий элемент группы a: ";
                cin >> a;
                cout << "Элемент группы b: ";
                cin >> b;

                try
                {
                        if (choice == 1)
                        {
                                long long res = discrLogarithm::gelfondThanks(p,
a, b);
                                cout << "\nx = " << res << "\nПроверка: a^x (mod
m) = " << Pattern::powClosed(a, res, p);
                        }
                        else if (choice == 2)
                        {
                                cout << "Значение epsilon: ";
                                double eps;
                                cin >> eps;

                                long long res =
discrLogarithm::roMethodPollarda(p, a, b, eps);
                                cout << "\n\nx = " << res << "\nПроверка: a^x
(mod m) = " << Pattern::powClosed(a, res, p);
                        }
                        else if (choice == 3)
                        {
                                long long res = discrLogarithm::indexMethod(p, a,
b);
                                while (res < 0)
                                        res += p - 1;
                                cout << "\nx = " << res << "\nПроверка: a^x (mod
m) = " << Pattern::powClosed(a, res, p);
                        }
```

```
                }
                catch (string& error)
                {
                        cout << endl << error;
                }

                cout << endl;
        }
}
```