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
Арбузова Матвея Александровича

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

Саратов 2023

## 1 Постановка задачи

Целью данной лабораторной работы является 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 Теоретические сведения по рассмотренным темам, алгоритмы и их сложности

### Дискретный логарифм

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

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

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

$$a^x = b.$$

Обозначается $x = \log_a b$.

Задача нахождения дискретного логарифма имеет большую сложность вычислений.

### Методы вычисления дискретных логарифмов

### 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 $O(\sqrt{B} \log B)$.

**$\rho$-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 $\rho$-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 $\{y_i\}$, $0 \leq i \leq k$. 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 $\{y_i\}$, $0 \leq i \leq k$ с вероятностью не менее $1 - e^{\lambda} = 1 - \varepsilon$ найдутся совпадающие члены $y_i = y_j, 0 \ \leq i < j \ \leq k$.

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 $G$.

**Индекс-метод дискретного логарифмирования в конечном простом поле**

Даны $g$ – образующий элемент группы $GF(p)^*$ и $h \in GF(p)^*$.

Требуется найти $x = \log_g h$.

Будем считать, что $GF(p) = Z_p$.

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 $\delta$ — некоторая небольшая константа.

В результате будет построена система линейных уравнений над кольцом $Z_{p-1}$ 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 $Z_{p-1}$ 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

$$L_p[c] = L_p\left[\frac{1}{2}, c\right] = \exp\left(\big(c + o(1)\big)(\log p \log\log p)^{\frac{1}{2}}\right) = L^{c+o(1)}$$ 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 $Z_{p-1}$ не является полем и имеются ненулевые необратимые элементы в $Z_{p-1}$, то не всякий шаг алгоритма Гаусса может быть реализован. Однако на практике это не является существенным ограничением. Действительно, на главную диагональ можно стремиться ставить обратимые элементы $Z_{p-1}$. Другой подход заключается в решении системы по $mod\ r_j^{t_j}$, где $p-1=\prod_j r_j^{t_j}$ — каноническое разложение числа $p-1$. Для этого достаточно уметь решать систему по простым модулям $r_j$, то есть над полем. Решения системы тогда легко найти, применив китайскую теорему об остатках.

## 3 Практическая реализация

### 3.1 Псевдокоды рассмотренных алгоритмов

**Псевдокод алгоритма Гельфонда-Шенкса вычисления дискретного логарифма в произвольной циклической группе**

Ввод $m$ – порядок конечной циклической группы, a – образующий элемент группы, b – элемент группы
Если $(m < 1)$
  вывести «Число $m$ должно быть больше 0»
Если ($a$ не удовлетворяет условию $1 < a < m$)
  вывести «$a$ должно удовлетворять условию: $1 < a < m$»
Если ($a$ не является образующим элементом)
  вывести «$a$ не является образующим элементом»
Если ($b$ не удовлетворяет условию $0 \le b < m$)
  вывести «$b$ должно удовлетворять условию: $0 \le b < m$»
Если результат функции $GS(m,a,b)$ не пустой, то вывести его, иначе вывести «Не удалось решить логарифм»

Функция $GS(m,a,b)$:
  $r = [\sqrt{m}] + 1$
  Создаётся список $degA$ для хранения упорядоченных пар вида $(k, a^k), 1 \le k \le r-1$, при этом они автоматически сортируются по второму элементу
  $promA = 1$
  В цикле по $i$ от 1 до $r-1$:
    $promA = (promA * a)\ (mod\ m)$
    Добавить в $degA$ пару $(i, promA)$
  $a_1 = a^{-r}\ (mod\ m)$
  $promA = 1$
  В цикле по $i$ от 0 до $r-1$:
    $ab = (promA * b)\ (mod\ m)$

```
        В цикле по всем парам из degA:
            Если (a^k = ab)
                Вернуть k + r * i в качестве результата
        promA = (promA * a_1) (mod m)
    Вернуть пустой результат
```

## Псевдокод алгоритма $\rho$-метода Полларда вычисления дискретного логарифма в произвольной циклической группе

```
    Ввод m – порядок конечной циклической группы, a – образующий элемент группы, b – элемент группы, ε
    Если (m < 1)
        вывести «Число m должно быть больше 0»
    Если (a не удовлетворяет условию 1 < a < m)
        вывести «a должно удовлетворять условию: 1 < a < m»
    Если (a не является образующим элементом)
        вывести «a не является образующим элементом»
    Если (b не удовлетворяет условию 0 ≤ b < m)
        вывести «b должно удовлетворять условию: 0 ≤ b < m»
    Если (ε не удовлетворяет условию 0 < ε < 1)
        вывести «ε должно удовлетворять условию: 0 < ε < 1»
    Если результат функции roPollard(m, a, b, ε) не пустой, то вывести его, иначе вывести «Не удалось решить логарифм»

    Функция Funf(x, a, b, m):
        i = 0
        Выполняется i = i + 1 пока i < 3 и x > U_i
        Если (i = 0)
            Вернуть b * x (mod m)
        Если (i = 1)
            Вернуть x * x (mod m)
        Вернуть a * x (mod m)

    Функция FunAlf(y, alf, m):
        i = 0
        Выполняется i = i + 1 пока i < 3 и x > U_i
        Если (i = 0)
            Вернуть alf (mod m)
        Если (i = 1)
            Вернуть alf * 2 (mod m)
        Вернуть (alf + 1) (mod m)

    Функция FunBeta(y, beta, m):
        i = 0
        Выполняется i = i + 1 пока i < 3 и x > U_i
        Если (i = 0)
            Вернуть (beta + 1) (mod m)
        Если (i = 1)
            Вернуть beta * 2 (mod m)
        Вернуть beta (mod m)

    Функция roPollard(m, a, b, ε):
        part = m/3
        Если (m (mod 3) = 0)
```

$part = part + 1$

$prom = part$

В цикле по $i$ от 0 до 2:

$U_i = prom$

$prom = prom + part$

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

В бесконечном цикле, пока не будет условия выхода, выполнять:

$i = 0$

$s = Random(0, m - 1)$

В список $y$ положить $a^s (mod\, m)$ и результат функции $Funf(a^s (mod\, m), a, b, m)$

В список $alfs$ положить $s$ и результат функции $FunAlf(y_i, s, m)$

В список $bets$ положить 0 и результат функции $FunBeta(y_i, 0, m)$

$index =$ размер списка $y$

В бесконечном цикле, пока не будет условия выхода, выполнять:

Если $(i \neq 0)$

Удалить первый элемент из списков $y, alfs, bets$

$index = index - 1$

В цикле по $j$ от 0 до 2:

В список $alfs$ положить результат функции $FunAlf(y_{index}, alfs_{index}, m)$

В список $bets$ положить результат функции $FunBeta(y_{index}, bets_{index}, m)$

В список $y$ положить результат функции $Funf(y_{index}, a, b, m)$

$index = index + 1$

Если $(y_0 \neq y_{index})$

Если $(i < k)$

Перейти к следующей итерации второго бесконечного цикла

Вернуть пустой ответ в качестве результата

Выйти из второго бесконечного цикла

$aUr = (bets_0 - bets_{index})\ (mod\, m)$

$d = aUr^{-1} (mod\, m)$

Если (выполняется $\sqrt{m} < d < m$

Перейти к следующей итерации первого бесконечного цикла

$bUr = (alfs_{index} - alfs_0)\ (mod\, m)$

$promm = m$

Если $(d > 1)$

Если $(bUr\ (mod\ a) \neq 0)$

Перейти к следующей итерации первого бесконечного цикла

$aUr = \frac{aUr}{d}$

$bUr = \frac{bUr}{d}$

$promm = \frac{promm}{d}$

$x =$ решению сравнения от $aUr, bUr$ и $promm$

Если ($d = 1$ или $a^x (mod\, m) = b$)

Вернуть в качестве результата $x$

$xNext = (x + promm)\ (mod\, m)$

Пока $x \neq xNext$ выполнять:

Если $(a^{xNext} (mod\, m) = b)$

Вернуть в качестве результата $xNext$

$xNext = (xNext + promm)(mod\, m)$

Выйти из первого бесконечного цикла
Вернуть пустой ответ в качестве результата

## Псевдокод алгоритма индекс-метода логарифмирования в конечном простом поле

Ввод $p$ – порядок конечной циклической группы, $g$ – образующий элемент группы, $h$ – элемент группы
Если $(p < 1)$
вывести «Число $p$ должно быть больше 0»
Если $(p \text{ не является простым})$
вывести «Число $p$ должно быть простым»
Если ($g$ не удовлетворяет условию $1 < g < p$)
вывести «$g$ должно удовлетворять условию: $1 < g < p$»
Если ($g$ не является образующим элементом)
вывести «$g$ не является образующим элементом»
Если ($h$ не удовлетворяет условию $0 \leq h < p$)
вывести «$g$ должно удовлетворять условию: $0 \leq g < m$»
Если результат функции $IndexMethod(g, h, p)$ не пустой, то вывести его, иначе вывести «Не удалось решить логарифм»

Функция $CanonDecomp(b, factorBase)$:
Создаётся пустой вектор $res$
$c = b$
В цикле по $i$ от 0 до размера $factorBase$
$a = 0$
Пока $(c \,(mod\, factorBase_i) \neq 0)$
$a = a + 1$
$c = \frac{c}{factorBase_i}$
Добавить $a$ в $res$
Если $(c \neq 1)$
Вернуть пустой результат
Вернуть $res$ в качестве результата

Функция $IndexMethod(g, h, p)$:
$B = \exp\left((\log p \log\log p)^{\frac{1}{2}}\right)^{\frac{1}{2}}$
В список $factorBase$ кладутся все простые числа $\leq B$
$i = 0$
Пока $(i < \text{размера}\, factorBase)$ выполнять:
$m = \text{случайное число от } 0 \text{ до } p - 2$
$b = g^m (mod\, p)$
Если $(b = 0)$
Перейти к следующей итерации цикла пока без изменения $i$
С помощью функции $CanonDecomp(b, factorBase)$ находится вектор $(l_1, l_2, .., l_{\pi(B)})$
Если (вектор $(l_1, l_2, .., l_{\pi(B)})$ пустой)
Перейти к следующей итерации цикла пока без изменения $i$
Сохранить вектор $(l_1, l_2, .., l_{\pi(B)})$ в список $equation$
Решить систему методом Гаусса, результат записать в $x$
Если (решение пустое)

Удалить из списка $equation$ последний вектор и перейти к следующей итерации цикла пока без изменения $i$

$i = i + 1$

Если ($x$ не найден)

Вернуть пустой ответ в качестве результата

В бесконечном цикле пока не будет условия выхода, выполнять:

$m$ = случайное число от 0 до $p - 2$

$b = g^m * h(mod\ p)$

Если ($b = 0$)

Перейти к следующей итерации бесконечного цикла

С помощью функции $CanonDecomp(b, factorBase)$ находится вектор $(r_1, r_2, .., r_{\pi(B)})$

Если (вектор $(r_1, r_2, .., r_{\pi(B)})$ пустой)

Перейти к следующей итерации бесконечного цикла

$res = 0$

В цикле по $i$ от 0 до $\pi(B)$

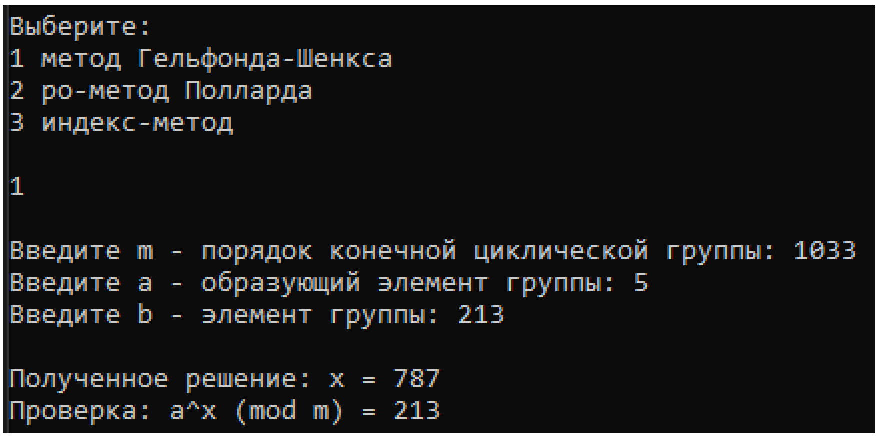
$res = (res + r_i * x_i)(mod\ p - 1)$

$res = (res - m)(mod\ p - 1)$

Вернуть $res$ в качестве результата

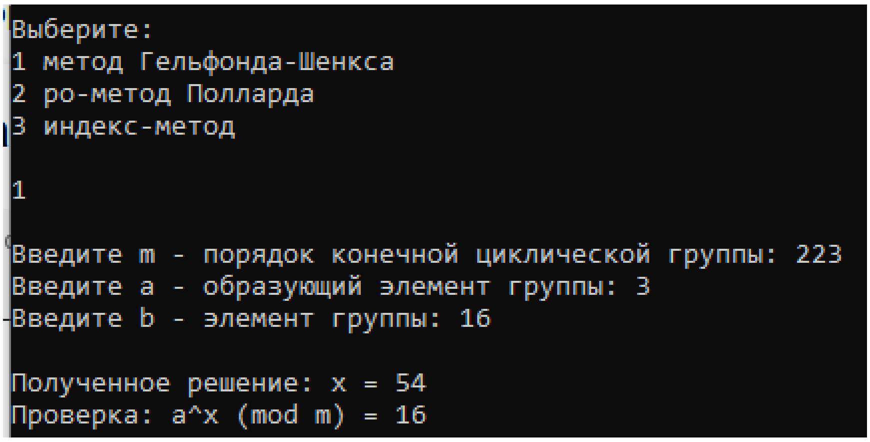
### 3.2 Результаты тестирования программы

Тестирование алгоритма Гельфонда-Шенкса вычисления дискретного логарифма в произвольной циклической группе (рисунки 1-2).



```
Выберите:
1 метод Гельфонда-Шенкса
2 ро-метод Полларда
3 индекс-метод

1

Введите m - порядок конечной циклической группы: 1033
Введите a - образующий элемент группы: 5
Введите b - элемент группы: 213

Полученное решение: x = 787
Проверка: a^x (mod m) = 213
```

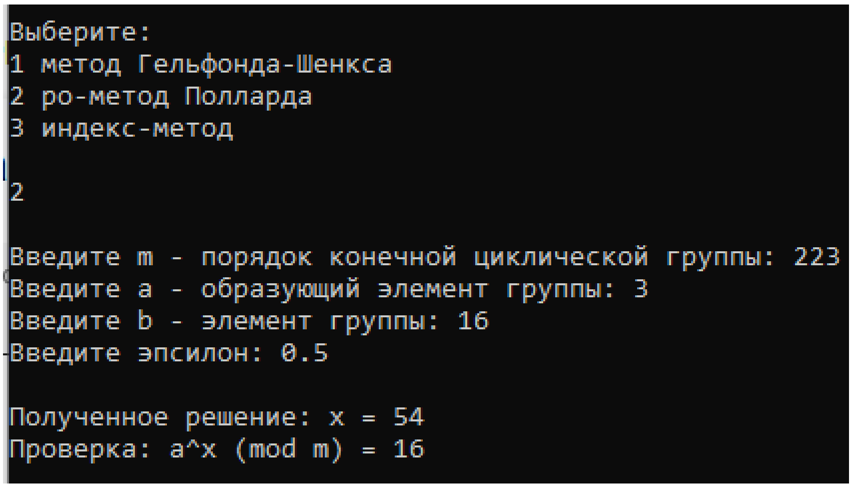
Рисунок 1 – Результат первого теста алгоритма Гельфонда-Шенкса



```
Выберите:
1 метод Гельфонда-Шенкса
2 ро-метод Полларда
3 индекс-метод

1

Введите m - порядок конечной циклической группы: 223
Введите a - образующий элемент группы: 3
Введите b - элемент группы: 16

Полученное решение: x = 54
Проверка: a^x (mod m) = 16
```

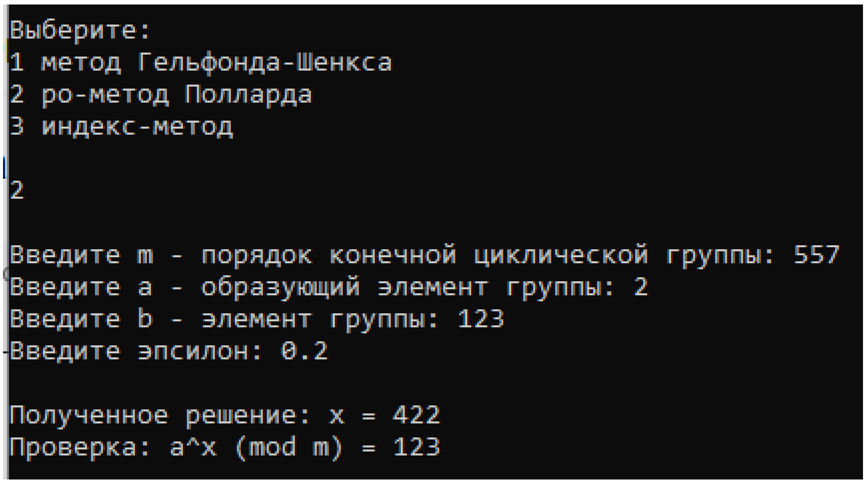
Рисунок 2 – Результат второго теста алгоритма Гельфонда-Шенкса

Тестирование алгоритма $\rho$-метода Полларда вычисления дискретного логарифма в произвольной циклической группе (рисунки 3-4).



```
Выберите:
1 метод Гельфонда-Шенкса
2 ро-метод Полларда
3 индекс-метод

2

Введите m - порядок конечной циклической группы: 223
Введите a - образующий элемент группы: 3
Введите b - элемент группы: 16
Введите эпсилон: 0.5

Полученное решение: x = 54
Проверка: a^x (mod m) = 16
```

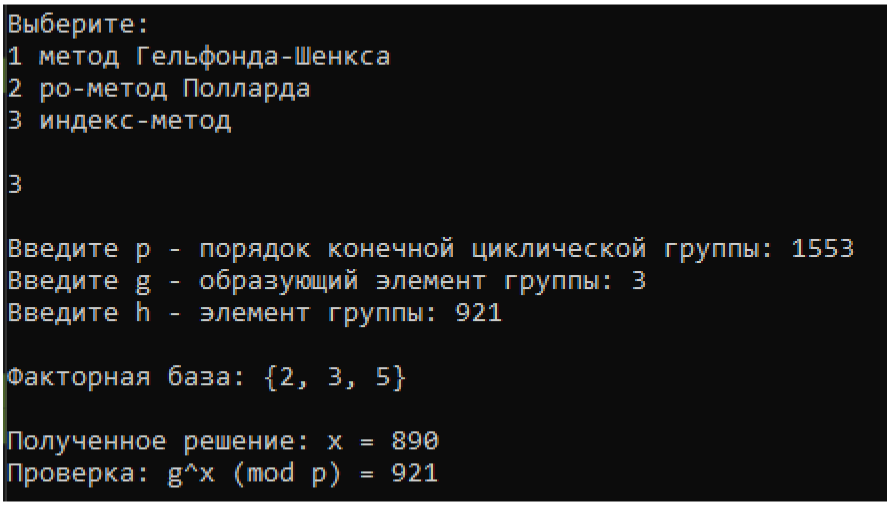
Рисунок 3 – Результат первого теста $\rho$-метода Полларда



```
Выберите:
1 метод Гельфонда-Шенкса
2 ро-метод Полларда
3 индекс-метод

2

Введите m - порядок конечной циклической группы: 557
Введите a - образующий элемент группы: 2
Введите b - элемент группы: 123
Введите эпсилон: 0.2

Полученное решение: x = 422
Проверка: a^x (mod m) = 123
```

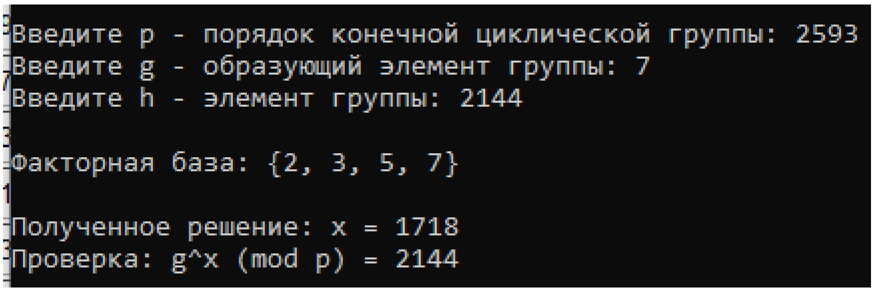
Рисунок 4 – Результат второго теста $\rho$-метода Полларда

Тестирование алгоритма индекс-метода логарифмирования в конечном простом поле (рисунки 5-6).



```
Выберите:
1 метод Гельфонда-Шенкса
2 ро-метод Полларда
3 индекс-метод

3

Введите p - порядок конечной циклической группы: 1553
Введите g - образующий элемент группы: 3
Введите h - элемент группы: 921

Факторная база: {2, 3, 5}

Полученное решение: x = 890
Проверка: g^x (mod p) = 921
```

Рисунок 5 – Результат первого теста индекс-метода



```
Введите p - порядок конечной циклической группы: 2593
Введите g - образующий элемент группы: 7
Введите h - элемент группы: 2144

Факторная база: {2, 3, 5, 7}

Полученное решение: x = 1718
Проверка: g^x (mod p) = 2144
```

Рисунок 6 – Результат первого теста индекс-метода

## 4 Выводы по работе

В ходе выполнения лабораторной работы были рассмотрены алгоритмы вычисления дискретных логарифмов в конечных полях. А именно – метод Гельфонда-Шенкса, $\rho$-метод Полларда и индекс-метод.

В практической части лабораторной работы была написана программа на языке C++, содержащая в себе реализацию всех описанных в теоретической части алгоритмов, работоспособность которых продемонстрирована на рисунках 1-6.

## 5 Код программы

```
#include <iostream>
#include <vector>
#include <random>
#include <boost/multiprecision/cpp_int.hpp>
#include <boost/math/constants/constants.hpp>
#include <boost/multiprecision/cpp_dec_float.hpp>
#include <boost/random/uniform_int.hpp>
#include <boost/random/variate_generator.hpp>
#include <set>


using namespace std;
using namespace boost::multiprecision;
namespace mp = boost::multiprecision;


cpp_int ModNegative(cpp_int a, cpp_int p) {
    if (a < 0)
        a = a + p * (((-1 * a) / p) + 1);
    return a % p;
}


vector <cpp_int> ExtendedEuclid(cpp_int a, cpp_int b) {
    vector <cpp_int> res(3);
    if (a == 0) {
        res = { b, 0, 1 };
        return res;
    }
    vector <cpp_int> c = ExtendedEuclid(b % a, a);
    res = { c[0], c[2] - (b / a) * c[1], c[1] };
    return res;
}


cpp_int Exponentiation(cpp_int x, cpp_int n, cpp_int m) {
    cpp_int N = n, Y = 1, Z = x % m;
    while (N != 0) {
        cpp_int lastN = N % 2;
        N = N / 2;
        if (lastN == 0) {
            Z = (Z * Z) % m;
            continue;
        }
        Y = (Y * Z) % m;
        if (N == 0)
            break;
        Z = (Z * Z) % m;
    }
    Y = Y % m;
    return Y;
}
```

```
      cpp_int Jac(cpp_int a, cpp_int b) {
          if (ExtendedEuclid(a, b)[0] != 1)
              return 0;
          else {
              cpp_int r = 1;
              while (a != 0) {
                  cpp_int t = 0;
                  while (a % 2 == 0) {
                      t = t + 1;
                      a = a / 2;
                  }
                  if (t % 2 != 0)
                      if (Exponentiation(b, 1, 8) == 3 || Exponentiation(b, 1,
8) == 5)
                          r = r * (-1);
                  if (Exponentiation(a, 1, 4) == 3 && Exponentiation(b, 1, 4)
== 3)
                      r = r * (-1);
                  cpp_int c = a;
                  if (c != 0)
                      a = Exponentiation(b, 1, c);
                  b = c;
              }
              return r;
          }
      }


      bool MilRab(cpp_int p, cpp_int k) {
          if (p == 1 || p == 2 || p == 3)
              return true;
          if (p % 2 == 0)
              return false;
          cpp_int t = p - 1;
          cpp_int s = 0;
          while (t % 2 == 0) {
              t = t / 2;
              s++;
          }
          for (cpp_int i = 0; i < k; i++) {
              cpp_int a = rand() % (p - 3) + 2;
              if (ExtendedEuclid(p, a)[0] > 1)
                  return false;
              cpp_int x = Exponentiation(a, t, p);
              if (x == 1 || x == p - 1)
                  continue;
              for (cpp_int g = 1; g < s; g++) {
                  x = x * x % p;
                  if (x == 1)
                      return false;
                  if (x == p - 1)
                      break;
              }
              if (x != p - 1)
                  return false;
          }
```

```
        return true;
    }


    cpp_int Random(cpp_int minim, cpp_int maxim) {
        random_device gen;
        boost::random::uniform_int_distribution<cpp_int> ui(minim, maxim);
        return ui(gen);
    }


    cpp_int GS(cpp_int m, cpp_int a, cpp_int b ) {
        cpp_int r = sqrt(m) + 1;
        multiset <pair <cpp_int, cpp_int>> degA;
        cpp_int promA = 1;
        for (cpp_int i = 1; i < r; i++) {
            promA = promA * a % m;
            degA.insert(make_pair(promA, i));
        }
        cpp_int  a1  =  ModNegative(ExtendedEuclid(Exponentiation(a,  r,  m), 
m)[1], m);
        promA = 1;
        for (cpp_int i = 0; i < r; i++) {
            cpp_int ab = promA * b % m;
            for (pair <cpp_int, cpp_int> j : degA) {
                if (j.first == ab)
                    return j.second + r * i;
            }
            promA = promA * a1 % m;
        }
        return -20;
    }


    cpp_int ObrazEl(cpp_int a, cpp_int p) {
        vector<cpp_int> fact;
        cpp_int phi = p - 1, n = phi;
        for (cpp_int i = 2; i * i <= n; ++i)
            if (n % i == 0) {
                fact.push_back(i);
                while (n % i == 0)
                    n /= i;
            }
        if (n > 1)
            fact.push_back(n);
        for (cpp_int res = 2; res <= a; ++res) {
            bool ok = true;
            for (size_t i = 0; i < fact.size() && ok; ++i)
                ok &= Exponentiation(res, phi / fact[i], p) != 1;
            if (ok)  return res;
        }
        return -1;
    }


    vector <cpp_int> TakeParams() {
```

```
        cpp_int m, a, b;
        cout << "\nВведите m - порядок конечной циклической группы: ";
        cin >> m;
        if (m < 1) {
            cerr << "\nm должно быть больше 0\n";
            return {};
        }
        if (!MilRab(m, 5)) {
            cerr << "\nm должно быть простым\n";
            return {};
        }
        srand(time(0));
        cout << "Введите a - образующий элемент группы: ";
        cin >> a;
        if (a < 2 || a > m - 1) {
            cerr << "\na должно удовлетворять условию: 1 < a < m\n";
            return {};
        }
        if (ObrazEl(a, m) != a) {
            cerr << "\na не является образующим элементом\n";
            return {};
        }
        cout << "Введите b - элемент группы: ";
        cin >> b;
        if (b < 0 || b > m - 1) {
            cerr << "\nb должно удовлетворять условию: 0 <= b < m\n";
            return {};
        }
        return { m, a, b };
    }


    void GSInit() {
        vector <cpp_int> params = TakeParams();
        if (params.size() == 0)
            return;
        cpp_int resultat;
        resultat = GS(params[0], params[1], params[2]);
        if (resultat == -20)
            cout << "\nНе удалось решить логарифм\n";
        else {
            cout << "\nПолученное решение: x = " << resultat << "\n";
             cout << "Проверка: a^x (mod m) = " << Exponentiation(params[1],
resultat, params[0]) << "\n";
        }
        return;
    }

    vector <cpp_int> Us(3);


    cpp_int Funf(cpp_int x, cpp_int a, cpp_int b, cpp_int m) {
        int i;
        for (i = 0; i < 3 && x > Us[i]; i++);
        if (i == 0)
            return b * x % m;
```

```
        if (i == 1)
            return x * x % m;
        return a * x % m;
    }


    cpp_int FunAlf(cpp_int y, cpp_int alf, cpp_int m) {
        int i;
        for (i = 0; i < 3 && y > Us[i]; i++);
        if (i == 0)
            return alf % m;
        if (i == 1)
            return alf * 2 % m;
        return (alf + 1) % m;
    }


    cpp_int FunBeta(cpp_int y, cpp_int beta, cpp_int m) {
        int i;
        for (i = 0; i < 3 && y > Us[i]; i++);
        if (i == 0)
            return (beta + 1) % m;
        if (i == 1)
            return beta * 2 % m;
        return beta % m;
    }


    pair <cpp_int, vector <cpp_int>> ChainFraction(cpp_int a, cpp_int b) {
        vector <cpp_int> Q = { 1, 0 };
        cpp_int prom, q;
        for (; b != 0;) {
            q = a / b;
            Q.push_back(q * (Q[Q.size() - 1] + Q[Q.size() - 2]));
            prom = a;
            a = b;
            b = prom % b;
        }
        return {a, Q };
    }


    cpp_int resolve(cpp_int a, cpp_int b, cpp_int m) {
        pair <cpp_int, vector <cpp_int>> gcd = ChainFraction(a, m);
        if (b % gcd.first != 0)
            return m - 1;
        cpp_int x = (gcd.second.size() - 2) % 2 == 0 ? 1 : -1;
        x = x *(gcd.second[gcd.second.size() - 2])* b / gcd.first;
        x = ModNegative(x, m);
        return x;
    }


    vector <cpp_int> roPollard(cpp_int m, cpp_int a, cpp_int b,
cpp_dec_float_50 epsil) {
        cpp_int part = m / 3;
```

```
        if (m % 3 != 0)
            part++;
        cpp_int prom = part;
        for (int i = 0; i < 3; i++) {
            Us[i] = prom;
            prom = prom + part;
        }
        cpp_int m1 = m - 1;
        cpp_int sqrtM(mp::sqrt(cpp_dec_float_50(m1)));
        cpp_dec_float_50 kprom = cpp_dec_float_50(sqrtM) * mp::sqrt(2 * log(1
/ epsil));
        cpp_int k(kprom + 1);
        for (;;) {
            int i = 0;
            cpp_int s = Random(0, m - 1);
            vector<cpp_int>    y    =    {    Exponentiation(a,    s,    m),
Funf(Exponentiation(a, s, m), a, b, m) };
            vector<cpp_int> alfs = { s, FunAlf(y[i], s, m1) };
            vector<cpp_int> bets = { 0, FunBeta(y[i], 0, m1) };
            int lastIdx = y.size() - 1;
            for (;;i++) {
                if (i != 0) {
                    y.erase(y.begin());
                    alfs.erase(alfs.begin());
                    bets.erase(bets.begin());
                    lastIdx--;
                    for (int j = 0; j < 2; j++) {
                        alfs.push_back(FunAlf(y[lastIdx],      alfs[lastIdx],
m1));
                        bets.push_back(FunBeta(y[lastIdx],     bets[lastIdx],
m1));
                        y.push_back(Funf(y[lastIdx], a, b, m));
                        lastIdx++;
                    }
                }
                if (y[0] != y[lastIdx]) {
                    if (i < k)
                        continue;
                    return {};
                }
                break;
            }
            cpp_int aUr = ModNegative(bets[0] - bets[lastIdx], m1);
            cpp_int d = ExtendedEuclid(aUr, m1)[0];
            if (sqrtM < d && d < m)
                continue;
            cpp_int  bUr  =  ModNegative(alfs[lastIdx]  -  alfs[0],  m1),  modCh  =
m1;
            if (d > 1) {
                if (bUr % a != 0)
                    continue;
                aUr = aUr / d;
                bUr = bUr / d;
                modCh = modCh / d;
            }
            cpp_int x = resolve(aUr, bUr, modCh);
```

```
            if (d == 1 || Exponentiation(a, x, m) == b)
                return {x};
            for (cpp_int xNext = (x + modCh) % m; x != xNext; ) {
                if (Exponentiation(a, xNext, m) == b)
                    return {xNext };
                xNext = (xNext + modCh) % m;
            }
            break;
        }
        return {};
    }


    void roPolInit() {
        cpp_dec_float_50 epsil;
        vector <cpp_int> params = TakeParams();
        if (params.size() == 0)
            return;
        cout << "Введите эпсилон: ";
        cin >> epsil;
        if (epsil <= 0 || epsil >= 1) {
            cerr << "\nэпсилон должно удовлетворять условию: 0 < эпсилон <
1\n";
            return;
        }
        vector   <cpp_int>   resultat   =   roPollard(params[0],   params[1],
params[2], epsil);
        if (resultat.size() == 0)
            cout << "\nНе удалось решить логарифм\n";
        else {
            cout << "\nПолученное решение: x = " << resultat[0] << "\n";
            cout << "Проверка: a^x (mod m) = " << Exponentiation(params[1],
resultat[0], params[0]) << "\n";
        }
        return;
    }


    vector <cpp_int> allPrime = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
41, 43, 47, 53, 59, 61, 67,
            71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173,
            179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281,
            283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367,
373, 379, 383, 389, 397, 401, 409,
            419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487,
491, 499, 503, 509, 521, 523, 541,
            547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617,
619, 631, 641, 643, 647, 653, 659,
            661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
757, 761, 769, 773, 787, 797, 809,
            811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883,
887, 907, 911, 919, 929, 937, 941,
            947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021,
1031, 1033, 1039, 1049, 1051, 1061,
```

```
            1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129,
1151, 1153, 1163, 1171, 1181, 1187,
            1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277,
1279, 1283, 1289, 1291, 1297, 1301,
            1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409,
1423, 1427, 1429, 1433, 1439, 1447,
            1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511,
1523, 1531, 1543, 1549, 1553, 1559,
            1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621,
1627, 1637, 1657, 1663, 1667, 1669,
            1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759,
1777, 1783, 1787, 1789, 1801, 1811,
            1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901,
1907, 1913, 1931, 1933, 1949, 1951,
            1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029,
2039, 2053, 2063, 2069, 2081, 2083
     };


     vector <cpp_int> Base(cpp_int B) {
         vector <cpp_int> res;
         cout << "\nФакторная база: {";
         for (int i = 0; i < allPrime.size() - 1; i++) {
             if (allPrime[i] <= B) {
                 res.push_back(allPrime[i]);
                 cout << allPrime[i];
             }
             if (allPrime[i + 1] <= B)
                 cout << ", ";
             else {
                 cout << "}\n";
                 break;
             }
         }
         return res;
     }


     vector <cpp_int> CanonDecomp(cpp_int b, vector<cpp_int> qi) {
         vector<cpp_int> li;
         cpp_int c = b;
         for (int i = 0; i < qi.size(); i++) {
             cpp_int a = 0;
             for (; c % (qi[i]) == 0;) {
                 a++;
                 c = c/qi[i];
             }
             li.push_back(a);
         }
         if (c != 1)
             return {};
         return li;
     }
```

```
vector<vector<cpp_int>> Mult(vector<vector<cpp_int>> a,
vector<vector<cpp_int>> b, cpp_int p) {
    vector<vector<cpp_int>> result;
    for (int i = 0; i < a.size(); i++) {
        vector<cpp_int> resi;
        for (int j = 0; j < b[0].size(); j++) {
            cpp_int sum = 0;
            for (int k = 0; k < b.size(); k++)
                sum = (sum + a[i][k] * b[k][j]) % p;
            resi.push_back(sum);
        }
        result.push_back(resi);
    }
    return result;
}


vector <cpp_int> modifJordan(vector<vector<cpp_int>> eq, cpp_int p) {
    int n = eq.size();
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            vector<cpp_int> rs = { eq[j][i], ModNegative(-1 * eq[i][i],
p) };
            vector<cpp_int> arr = ExtendedEuclid(eq[i][i], eq[j][i]);
            vector<cpp_int> xy = { ModNegative(arr[1], p),
ModNegative(arr[2], p) };
            vector<vector<cpp_int>> a = { xy, rs };
            vector<vector<cpp_int>> b = { eq[i], eq[j] };
            vector<vector<cpp_int>> res = Mult(a, b, p);
            eq[i] = res[0];
            eq[j] = res[1];
        }
        cpp_int aiiInv = 0;
        if (ExtendedEuclid(eq[i][i], p)[0] != 1)
            return {};
        aiiInv = ModNegative(ExtendedEuclid(eq[i][i], p)[1], p);
        for (int k = 0; k < eq[i].size(); k++)
            eq[i][k] = eq[i][k] * aiiInv % p;
        for (int j = 0; j < i; j++) {
            cpp_int aji = eq[j][i];
            for (int k = 0; k < eq[j].size(); k++)
                eq[j][k] = ModNegative(eq[j][k] - (eq[i][k] * aji), p);
        }
    }
    int lastIdx = eq[0].size() - 1;
    vector<cpp_int> x;
    for (int i = 0; i < eq.size(); i++)
        x.push_back(eq[i][lastIdx]);
    return x;
}



vector <cpp_int> IndexMethod(cpp_int g, cpp_int h, cpp_int p) {
    cpp_dec_float_50 sqrtP = log(cpp_dec_float_50(p));
    cpp_int p1 = p - 1;
```

```
    cpp_dec_float_50 b = sqrt(exp(sqrt(sqrtP * log(sqrtP))));
    cpp_int B(b);
    vector<cpp_int> factorBase = Base(B);
    vector<vector<cpp_int>> equation;
    vector <cpp_int> x;
    for (int i = 0; i < factorBase.size();) {
        cpp_int m = Random(0, p - 2);
        cpp_int b = Exponentiation(g, m, p);
        if (b == 0)
            continue;
        vector<cpp_int> li = CanonDecomp(b, factorBase);
        if (li.size() == 0)
            continue;
        li.push_back(m);
        equation.push_back(li);
        x = modifJordan(equation, p1);
        if (x.size() == 0) {
            equation.erase(equation.end() - 1);
            continue;
        }
        i++;
    }
    if (x.size() == 0)
        return {};
    for (;;) {
        cpp_int m = Random(0, p - 2);
        cpp_int b = Exponentiation(g, m, p) * h % p;
        if (b == 0)
            continue;
        vector<cpp_int> ri = CanonDecomp(b, factorBase);
        if (ri.size() == 0)
            continue;
        cpp_int res = 0;
        for (int i = 0; i < ri.size(); i++)
            res = res + (ri[i] * (x[i])) % p1;
        res = ModNegative(res - m, p1);
        return { res };
    }
}


void IMlInit() {
    cpp_int p, g, h;
    cout << "\nВведите p - порядок конечной циклической группы: ";
    cin >> p;
    if (p < 1) {
        cerr << "\np должно быть больше 0\n";
        return;
    }
    if (!MilRab(p, 5)) {
        cerr << "\np должно быть простым\n";
        return;
    }
    srand(time(0));
    cout << "Введите g - образующий элемент группы: ";
    cin >> g;
```

```
        if (g < 2 || g > p - 1) {
            cerr << "\ng должно удовлетворять условию: 1 < g < p\n";
            return;
        }
        if (ObrazEl(g, p) != g) {
            cerr << "\ng не является образующим элементом\n";
            return;
        }
        cout << "Введите h - элемент группы: ";
        cin >> h;
        if (h < 0 || h > p - 1) {
            cerr << "\nh должно удовлетворять условию: 0 <= h < m\n";
            return;
        }
        vector <cpp_int> resultat = IndexMethod(g, h, p);
        if (resultat.size() == 0)
            cout << "\nНе удалось решить логарифм\n";
        else {
            cout << "\nПолученное решение: x = " << resultat[0] << "\n";
            cout  <<  "Проверка:  g^x  (mod  p)  =  "  <<  Exponentiation(g,
resultat[0], p) << "\n";
        }
        return;
    }


    int main() {
        setlocale(LC_ALL, "Russian");
        cpp_int c = 100;
        cout << "\nДискретное логарифмирование в конечном поле.";
        while (c != 0) {
            cout << "\nВыберите:\n";
            cout << "1 метод Гельфонда-Шенкса \n";
            cout << "2 ро-метод Полларда\n";
            cout << "3 индекс-метод\n\n";
            cin >> c;
            if (c == 1) {
                GSInit();
                continue;
            }
            if (c == 2) {
                roPolInit();
                continue;
            }
            if (c == 3) {
                IMlInit();
                continue;
            }
            cerr << "Введённый вариант ответа отсутствует\n";
            c = 100;
        }
        return 0;
    }
```