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

Цепные дроби и квадраты сравнения

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГАФИИ»

студентки 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

Преподаватель

профессор, д.ф.-м.н. ___________ В. А. Молчанов

подпись, дата

Саратов 2023

Введение

В данной лабораторной работе поставлена задача рассмотрения алгоритмов разложения чисел в цепную дробь, приложений цепных дробей, вычисления символов Лежандра и Якоби, алгоритмы извлечения квадратного корня в кольце вычетов, написание алгоритмов для изученных тем.

Цель работы и порядок её выполнения

Цель работы – изучение основных свойств цепных дробей и квадратных сравнений.

Задачи работы:

- Разобрать алгоритм разложения чисел в цепную дробь и привести их программную
- Рассмотреть алгоритмы приложений цепных дробей и привести их программную реализацию.
- Разобрать алгоритмы вычисления символов Лежандра и Якоби и привести их программную реализацию.
- Рассмотреть алгоритмы извлечения квадратного корня в кольце вычетов.

1 Теоретическая часть

1.1 Разложения чисел в цепную дробь

С помощью алгоритма Евклида любое рациональное число можно представить в виде специального выражения, которое называется цепной дробью и которое играет важную роль в алгебре, теории чисел, криптографии и во многих других областях математики. Рассмотрим рациональное число r, представленное в виде несократимой дроби $r = \frac{a_0}{a_1}$. Так как НОД$(a_0, a_1) = 1$, то результат вычисления этого наибольшего общего делителя по алгоритму Евклида имеет вид:

$$a_0 = a_1 q_1 + a_2, 0 \leq a_2 \leq a_1,$$

$$a_1 = a_2 q_2 + a_3, 0 \leq a_3 \leq a_2,$$

$$\dots$$

$$a_{k-2} = a_{k-1} q_{k-1} + a_k, 0 \leq a_k \leq a_{k-1},$$

$$a_{k-1} = a_k q_k,$$

где $a_k =$ НОД$(a_0, a_1) = 1$.

Рациональное число r можно представить следующим образом:

$$r = \frac{a_0}{a_1} = q_1 + \frac{a_2}{a_1} = q_1 + \frac{1}{\frac{a_2}{a_1}} = q_1 + \frac{1}{q_2 + \frac{a_3}{a_2}} = \dots = q_1 + \cfrac{1}{q_2 + \cfrac{1}{\cfrac{1}{q_{k+1} + \cfrac{1}{q_k}}}}$$

где q_1 – целое число и $q_2, \dots, q_k$ – целые положительные числа.

<u>Определение.</u> Выражение вида

$$r = \frac{a_0}{a_1} = q_1 + \frac{a_2}{a_1} = q_1 + \frac{1}{\frac{a_2}{a_1}} = q_1 + \frac{1}{q_2 + \frac{a_3}{a_2}} = \dots = q_1 + \cfrac{1}{q_2 + \cfrac{1}{\cfrac{1}{q_{k+1} + \cfrac{1}{q_k}}}}$$

принято называть цепной (или непрерывной) дробью с неполными частными $q_1, q_2, \dots, q_k$ и обозначать символом $(q_1; q_2, \dots, q_k)$.

<u>Алгоритм разложения чисел в цепную дробь:</u>

Вход. Целые сила a, b.

Выход. Коэффициенты разложения в цепную дробь.

Шаг 1. $a_1 = a, a_2 = b, q = [\,]$.

Шаг 2. Пока НОД$(a_1, a_2) \neq 0$.

Шаг 2.1. Добавить целую часть от деления a_1 на a_2 в список q.

Шаг 2.2. Обновить значение $a_1 = a_1 \, mod \, a_2$.

Шаг 2.3. Поменять местами значения a_1 и a_2.

Шаг 3. К списку q добавить целую часть от деления a_1 на a_2.

Шаг 4. Вернуть список q.

Сложность алгоритма: $O(max(a, b))$.

1.2 Приложение цепных дробей

- Решение линейных диофантовых уравнений $ax + by = c$;
- Вычисление обратных элементов в кольце вычетов Z_m;
- Решение линейных сравнений $ax \equiv b \ (mod\ m)$.

Приведенные выше свойства числителей и знаменателей подходящих дробей цепной дроби $\frac{a}{m} = (q_1; q_2, \dots, q_k)$ дают эффективный способ решения диофантовых уравнений.

Определение. Диофантовыми уравнениями называются алгебраические уравнения с целочисленными коэффициентами, решение которых отыскивается в целых числах.

Например, диофантовым уравнением является уравнение вида $ax - by = 1$ с целыми неотрицательными коэффициентами a, b. Если коэффициенты a, b удовлетворяют условию НОД$(a, b) = 1$ и $\frac{P_{k-1}}{Q_{k-1}}$ – предпоследняя подходящая дробь представления числа $\frac{a}{b}$ в виде цепной дроби, то из равенств $P_k Q_{k-1} - P_{k-1} Q_k = (-1)^k$, $\frac{a}{b} = \lambda_k = \frac{P_k}{Q_k}$ следует, что $a(-1)^k Q_{k-1} - b(-1)^k P_{k-1} = 1$, т.е. значения $x = (-1)^k Q_{k-1}, y = (-1)^k P_{k-1} = 1$ являются целочисленными решениями уравнения $ax - by =$

1. Легко видеть, что все целые решения исходного диофантова уравнения $ax - by = 1$ находятся по формулам:

$$x = (-1)^k Q_{k-1} + bt, y = (-1)^k P_{k-1} + at,$$

где t – произвольное целое число.

Нетрудно убедиться, что все решения диофантова уравнения $ax - by = c$ с взаимно простыми коэффициентами a, b находятся по формулам:

$$x = (-1)^k c Q_{k-1} + bt,$$
$$y = (-1)^k c P_{k-1} + at,$$

где t – произвольное целое число.

Алгоритм решения диофантова уравнения:

Вход. Целые числа a, b, c.

Выход. Решение уравнения в виде пары (x, y), удовлетворяющие уравнению $ax + by = c$.

Шаг 1. При помощи расширенного алгоритма Евклида находим $gcd =$ НОД(a, b).

Шаг 2. Проверяем делится ли c на gcd, если нет, то решения не существует, иначе переходим на шаг 3.

Шаг 3. Делим a, b, c на gcd.

Шаг 4. Создаем два массива p, q для линейных коэффициентов в цепной дроби и заполняем их через цепную дробь числа a/b.

Шаг 5. Представляем значения из p и q в качестве коэффициентов для x и y, т.к. x и y могут быть выражены через эти коэффициенты.

Шаг 6. Возвращаем получившиеся x и y как решение диофантова уравнения.

Сложность алгоритма: $O(max(a, b))$.

Алгоритм вычисления обратного элемента в кольце:

Вход. Целые числа a, m.

Выход. Обратный элемент x или ничего, если он не существует.

Шаг 1. При помощи расширенного алгоритма Евклида находим $gcd =$ НОД(a, b), а также коэффициенты x, y, такие что $ax + my = gcd$.

Шаг 2. Проверяем $gcd == 1$, если нет, то обратный элемент не существует, иначе переходим на шаг 3.

Шаг 3. Вычисляем обратный элемент по модулю $x = (x \, mod \, m + m) \, mod \, m$.

Шаг 4. Возвращаем x в качестве результата.

Сложность алгоритма: $O(log(max(a, m)))$.

Алгоритм решения линейного сравнения:

Вход. Целые числа a, b, c.

Выход. Решение уравнения в виде пары (x, y), удовлетворяющие уравнению $ax + by = c$.

Шаг 1. Находим $gcd =$ НОД(a, b).

Шаг 2. Проверяем делится ли c на gcd, если нет, то решения не существует, иначе переходим на шаг 3.

Шаг 3. Делим a, b, c на gcd.

Шаг 4. Создаем два массива p, q для линейных коэффициентов в цепной дроби и заполняем их через цепную дробь числа a/b.

Шаг 5. Представляем значения из p и q в качестве коэффициентов для x и y, т.к. x и y могут быть выражены через эти коэффициенты.

Шаг 6. Возвращаем получившиеся x и y как решение диофантова уравнения.

Сложность алгоритма: $O(max(a, b))$.

1.3 Вычисление символов Лежандра и Якоби

Определение. Для нечетного простого числа p символом Лежандра числа $a \in Z$ называется выражение

$$\left(\frac{a}{p}\right) = \begin{cases} 1, \text{если } a - \text{квадратичный вычет по модулю } p; \\ -1, \text{если } a - \text{квадратичный невычет по модулю } p; \\ 0, \text{если } a \equiv 0 \ (mod\ p). \end{cases}$$

Свойства символа Лежандра:

- $a \equiv b\ (mod\ p) \rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$.
- $\left(\frac{ac^2}{p}\right) = \left(\frac{a}{p}\right)$ для любого $c \in Z$, НОД$(c, p) = 1$.
- Критерий Эйлера: $\left(\frac{a}{p}\right) = a^{\frac{p-2}{2}}(mod\ p)$ для НОД$(a, p) = 1$.

Алгоритм вычисления символа Лежандра:

Вход. Целое число a и нечетное простое число p.

Выход. Значение символа Лежандра.

Шаг 1. Если $a < 0 \rightarrow$ по свойству 4) выделяем множитель $\left(\frac{-1}{p}\right)$.

Шаг 2. Заменяем a на остаток от деления на p.

Шаг 3. Представляем $a = p_1^{a_1} * \ldots * p_k^{a_k}$ и вычисляем по свойству 4) $\left(\frac{a}{p}\right) = \left(\frac{p_1}{p}\right)^{a_1} * \ldots * \left(\frac{p_k}{p}\right)^{a_k}$, при этом опускаем множители с четными степенями a_i и вместо множителей $\left(\frac{p_i}{p}\right)^{a_i}$ с нечетными степенями a_i оставляем $\left(\frac{p_i}{p}\right)$.

Шаг 4. Если $p_i = 2$, то вычисляем по свойству 6) $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$.

Шаг 5. К остальным символам $\frac{p_i}{p}$ применяется квадратичный закон взаимности Гаусса.

Шаг 6. При необходимости возвращаемся к шагу 2.

Сложность алгоритма: $O(log^2 p)$.

Определение. Пусть дано натуральное число $n = p_1^{a_1} * \ldots * p_k^{a_k}$.Символом Якоби числа $a \in Z$ называется выражение

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{a_1} * \ldots * \left(\frac{a}{p_k}\right)^{a_k}.$$

Символ Якоби для простого числа n совпадает с символом Лежандра и удовлетворяет почти всем свойствам символа Лежандра (хотя в общем случае символ Якоби не связан с квадратичными вычетами).

Символ Якоби позволяет упростить вычисление символа Лежандра $\left(\frac{a}{p}\right)$ (без разложения числа a на множители).

Алгоритм вычисления символа Якоби для простого числа p совпадает с алгоритмом вычисления символа Лежандра.

<u>Простой алгоритм вычисления символа Якоби:</u>

<u>Вход.</u> Целое число a и нечетное простое число p.

<u>Выход.</u> Значение символа Якоби.

Шаг 1. Заменяем a на p, что $a \equiv b \ (mod\ p)$ и $|b| < \frac{p}{2}$.

Шаг 2. Если $b < 0 \rightarrow$ по свойству 4) выделяем множитель $\left(\frac{-1}{p}\right)$.

Шаг 3. Если $d\ mod\ 2 = 0 \rightarrow b = 2^t * a_1$ и, если $t\ mod\ 2\ \neq 1 \rightarrow \left(\frac{2}{p}\right) = (-1)\left(\frac{p^2-1}{8}\right)$.

Шаг 4. К символу $\left(\frac{a_1}{p}\right)$ применяется квадратичный закон взаимодействия Гаусса.

Шаг 5. При необходимости возвращаемся к шагу 1.

<u>Сложность алгоритма:</u> $O(\log^2 p)$.

1.4 Извлечение квадратного корня в кольце вычетов

1. Если $p \equiv 3\ (mod\ 4)$, то $p = 4m + 3$ и $x = \pm a^{m+1}$, $a \in QR_p \rightarrow (a)^{\frac{p-1}{2}} \equiv 1 \equiv a^{2m+1}\ (mod\ p)$, $x = a^{m+1}$ удовлетворяет $x^2 \equiv a^{2m+1}a \equiv a\ (mod\ p)$.
2. Если $p \equiv 5\ (mod\ 8)$, то $p = 8m + 5$ и $x = \pm a^{m+1}$ или $x = \pm a^{m+1} * 2^{2m+1}$, $a \in QR_p \rightarrow (a)^{\frac{p-1}{2}} \equiv 1 \equiv a^{4m+2} =$

$(a^{2m+1})^2 (mod\, p) \to a^{2m+1} \equiv 1\ (mod\, p)$ или $2^{2m+1} \equiv -1\ (mod\, p)$.

В первом случае: $x = a^{m+1}$, $x^2 \equiv a^{2m+1} a \equiv a (mod\, p)$.

Во втором случае рассматриваем $2 \in QNR_p$, так как $\left(\frac{2}{p}\right) = -1$ в силу $p \equiv -3\ (mod\ 8)$.

Тогда $2^{\frac{p-1}{2}} = 2^{4m+2} = (2^{2m+1})^2 \equiv -1 (mod\, p)$, $a^{2m+1}(2^{2m+1})^2 \equiv 1 (mod\, p)$ и для $x = a^{m+1} * 2^{2m+1}$ получаем $x^2 \equiv (a^{m+1} 2^{2m+1})^2 \equiv a (mod\, p)$.

В общем случае применяются специальные полиномиальные вероятностные алгоритмы.

<u>Алгоритм извлечения квадратного корня в кольце вычетов:</u>

<u>Вход.</u> Нечетное простое число $p, a \in Z, \left(\frac{a}{p}\right) = 1$.

<u>Выход.</u> x_0 – решение сравнения $x^2 \equiv a (mod\, p)$.

Шаг 1. Случайным образом выбирается $b, 0 \le b \le p - 1$, такое, что $\left(\frac{b^2 - 4a}{p}\right)$.

Шаг 2. Положить $f(y) = y^2 - by + a$.

Шаг 3. Вычислить x_0 – остаток от деления $y^{\frac{p+1}{2}}$ на $f(y)$. Тогда x_0 – искомое решение.

Шаг 4. Вернуть x_0.

<u>Сложность алгоритма</u>: $O(log^3\, p)$.

2 Псевдокоды программ

2.1 Псевдокод алгоритма разложения чисел в цепную дробь

<u>Алгоритм разложения чисел в цепную дробь.</u>

```
Процедура Разложить_на_цепную_дробь(a,b):
      a1 = a
      a2 = b
      q = []
      Пока a1 mod a2 != 0:
            частное = a1 div a2
            Добавить к q частное
```

```
            a1 = a1 mod a2
            Поменять значениями a1 и a2
      Конец пока
      целая_часть = a1 div a2
      Добавить к q целую_часть
      Вернуть q
Конец процедуры
```

2.2 Псевдокод приложения цепных дробей

Решение диофантова уравнения

```
Процедура Решить_диофантово_Уравнение(a,b,c):
      НОД = Евклид(a, b)  # Находим НОД(a, b) с помощью алгоритма
Евклида
      Если c mod НОД != 0
            Вернуть «Решений не существует»
      Конец Если
      a = a div НОД
      b = b div НОД
      c = c div НОД
      p = []; q = []
      Пока b != 0:
            частное = a div b
            остаток = a mod b
            Добавить к p частное
      Добавить к q остаток
      Конец Пока
      a = b
      b = остаток
      x = 0
      y = 1
      n = Длина(p)
      Для всех i от n-1 до 0 с шагом -1:
            новый_x = y
            новый_y = x – p[i] * y
      Конец Для
      Вернуть (x * c, y * c)
Конец процедуры
```

Вычисление обратного элемента в кольце.

```
Процедура Найти_Обратный_Элемент(a, m):
      НОД, x, y = Расширенный_Евклид(a, m)       # Вычисляем
НОД(a, m) и коэффициенты x, y
      Если НОД != 1:
            Вернуть «Обратный элемент не существует»
      Конец Если
      x = (x mod m + m) mod m
      Вернуть x
Конец процедуры
```

2.3 Псевдокоды вычислений символов Лежандра и Якоби

Псевдокод вычисления символа Лежандра.

```
Процедура Вычислить_Символ_Лежандра(a, p):
	Если a < 0:
		a = a mod p
	Конец Если
	Если a == 0:
		Вернуть 0
	Конец Если
	Символ_Лежандра = 1
	Пока a != 1:
		Если a mod 2 == 0:
			a = a div 2
		Конец Если
		Если p mod 8 == 3 ИЛИ p mod 8 == 5:
			Символ_Лежандра = - Символ_Лежандра
		Иначе:
			Поменять значениями a и p
		Конец Если
		Если a mod 4 == 3 В p mod 4 == 3:
			Символ_Лежандра = - Символ_Лежандра
		Конец Если
	Конец Пока
	Вернуть Символ_Лежандра
Конец Процедуры
```

Псевдокод вычисления символа Якоби.

```
Процедура Вычислить_Символ_Якоби(a, p):
	Пока a != 0:
		a = a mod p
		Если a == 0:
			Вернуть 0
		Конец Если
		Если a < 0:
			a = a * (-1)
		Конец Если
		Если a == 1:
			Вернуть 1
		Конец Если
		Если a mod 2 == 0:
			b = a
			t = 0
		Конец Если
		Пока b mod 2 == 0:
			b = b div 2
			t = t + 1
		Конец Пока
		Если t mod 2 != 0 И (p mod 8 == 3 ИЛИ p mod == 5):
			Вернуть -1
		Конец Если
		a = b
		Если a == 2:
			Если p mod 8 == 3 ИЛИ p mod 8 == 5:
				Вернуть -1
```

```
            Конец Если
            Вернуть 1
        Конец Если
        Поменять значениями a и p
        Если a mod 4 == 3 И p mod 4 == 3:
            Вернуть (-1) * Вычислить_Символ_Якоби(-1, p)
        Конец Если
    Конец Пока
    Вернуть 1
Конец процедуры
```

2.4 Псевдокод извлечения квадратного корня в кольце вычетов

Извлечение квадратного корня в кольце вычетов

```
Процедура Извлечь_Квадратный_Корень(p, a):
Если a < 0:
    a = a + p
Конец Если
Пока True:
    b = Случайное_Число(0, p - 1)
    x = (b *b – 4 * a) mod p
    Если Вычислить_Символ_Якоби(x, p) == 1:
        y = 0
    Конец Если
Конец Пока
Пока True:
    z = (y * y – b * y + a) mod p
    Если z == 0:
        Вернуть y, y = (y+1) mod p
    Конец Если
Конец Пока
```

3 Тестирование программы

На рисунках 1-6 можно увидеть работу, реализуемой программы, по рассмотренным алгоритмам.

```
[1] --> разложение в цепную дробь

[2] --> решение диофантова уравнения

[3] --> вычисление обратного элемента в кольце

[4] --> решение линейного стравнения

[5] --> вычисление символов Лежандра и Якоби

[6] --> извлечение квадратных корней в кольце вычетов

[0] --> выход из программы


Введите выбор:
1

Введите дробь (числитель знаменатель):
13 15
Разложение в цепную дробь: [0, 1, 6, 2]
```

Рисунок 1 – Разложение в цепную дробь

```
[1] --> разложение в цепную дробь

[2] --> решение диофантова уравнения

[3] --> вычисление обратного элемента в кольце

[4] --> решение линейного стравнения

[5] --> вычисление символов Лежандра и Якоби

[6] --> извлечение квадратных корней в кольце вычетов

[0] --> выход из программы


Введите выбор:
2

Введите параметры Диофантового уравнения (a b c):
5 4 6
Решение Диофантового уравнения: [10, -11]
```

Рисунок 2 – Решение Диофантова уравнения

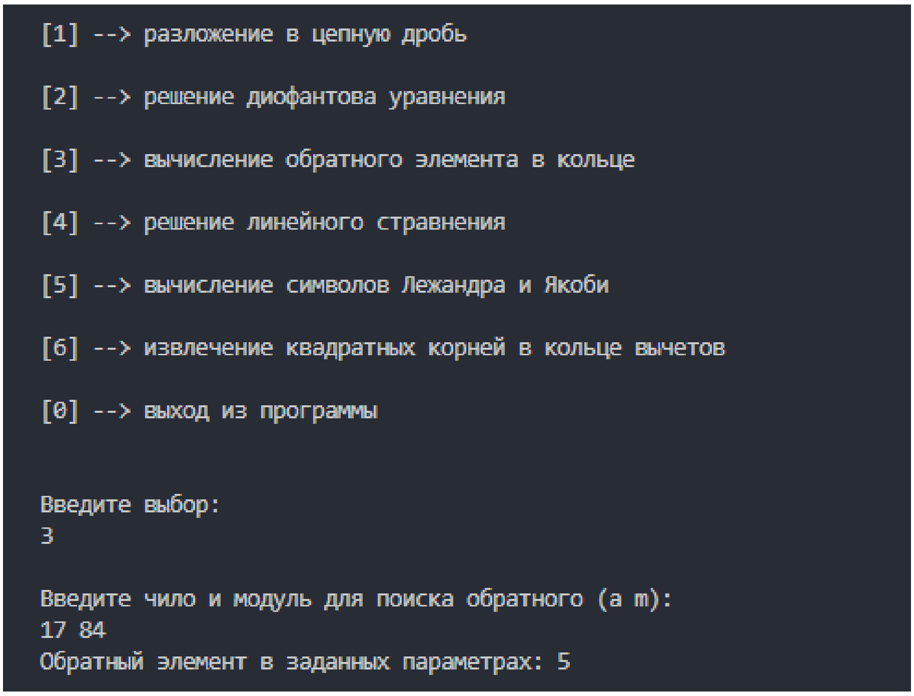


Рисунок 3 – Вычисление обратного элемента в кольце

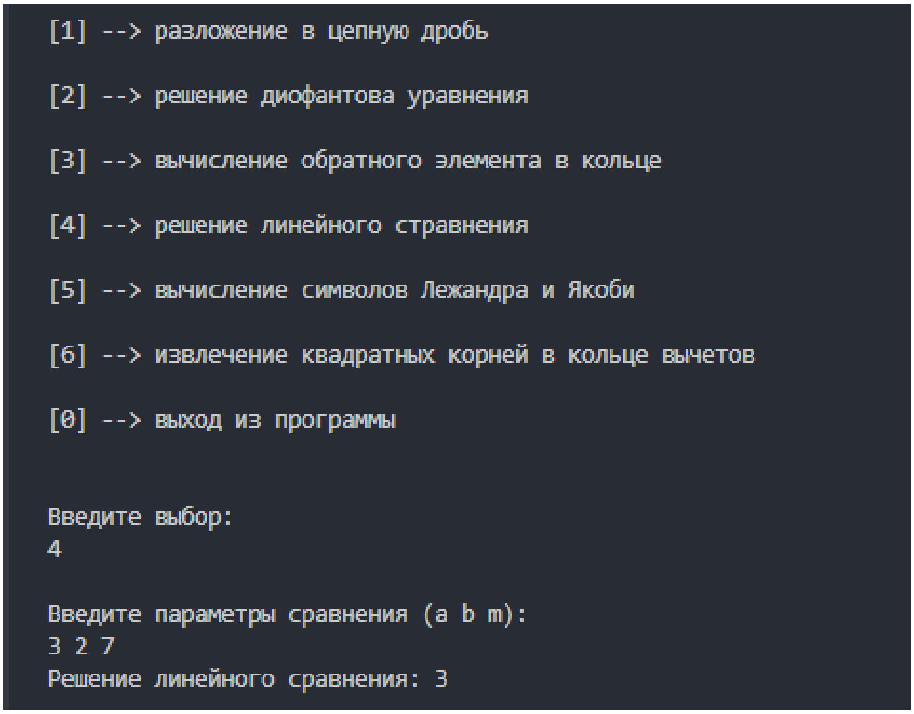
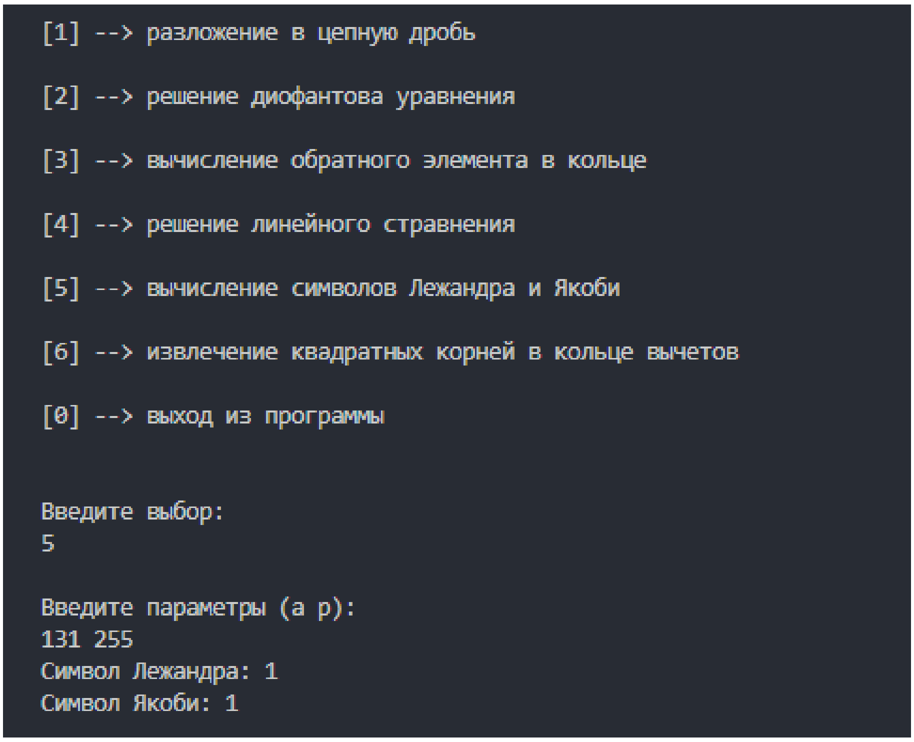
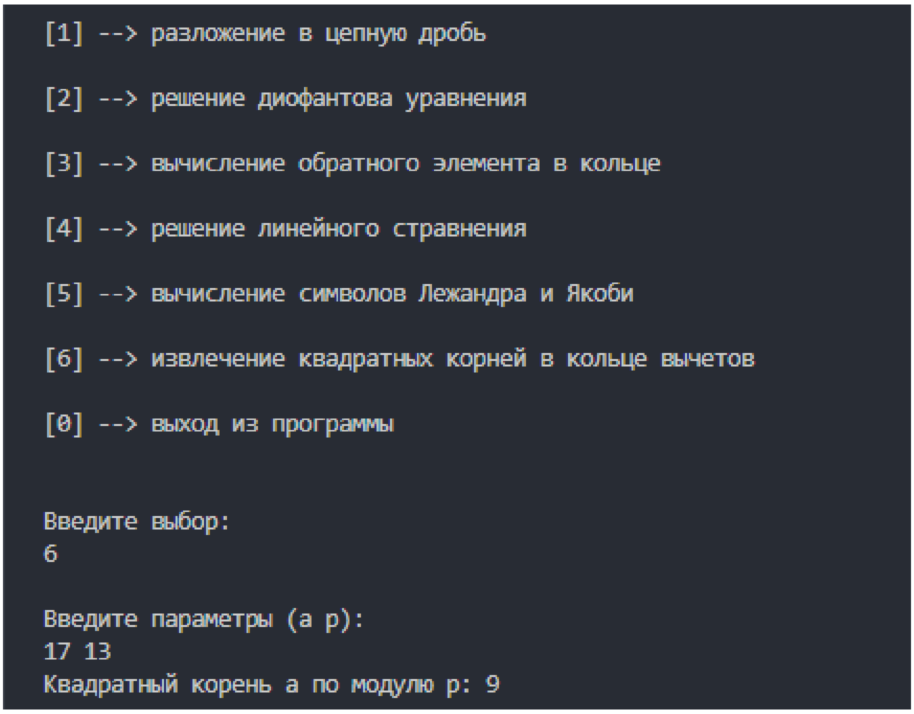


Рисунок 4 – Решение линейного сравнения



```
[1] --> разложение в цепную дробь

[2] --> решение диофантова уравнения

[3] --> вычисление обратного элемента в кольце

[4] --> решение линейного стравнения

[5] --> вычисление символов Лежандра и Якоби

[6] --> извлечение квадратных корней в кольце вычетов

[0] --> выход из программы


Введите выбор:
5

Введите параметры (a p):
131 255
Символ Лежандра: 1
Символ Якоби: 1
```

Рисунок 5 - Вычисление символов Лежандра и Якоби



```
[1] --> разложение в цепную дробь

[2] --> решение диофантова уравнения

[3] --> вычисление обратного элемента в кольце

[4] --> решение линейного стравнения

[5] --> вычисление символов Лежандра и Якоби

[6] --> извлечение квадратных корней в кольце вычетов

[0] --> выход из программы


Введите выбор:
6

Введите параметры (a p):
17 13
Квадратный корень a по модулю p: 9
```

Рисунок 6 - Извлечение квадратного корня в кольце вычетов

ПРИЛОЖЕНИЕ А

Код программы lab2.rb

```
class Methods

  def initialize
  end

  def make_chain(a,b)
    q = []

    while a % b != 0
      q << (a / b).to_i
      a %= b
      a, b = b, a
    end

    q << (a / b).to_i

    return q
  end

  def diophantus(a, b, c)
    gcd, x, y = exea(a, b)
    p, q = [0, 1], [1, 0]
    q_chain = make_chain(a, b)

    q_chain.each do |q_i|
      p << q_i * p[-1] + p[-2]
      q << q_i * q[-1] + q[-2]
    end

    p = p[2..-1]
    q = q[2..-1]

    if c % gcd == 0
      k = p.length
      a /= gcd
      b /= gcd
      c /= gcd
      x = (-1) ** k * q[-2] * c + b
      y = -1 * ((-1) ** k * p[-2] * c + a)
      return x, y
    end
  end

  def inv_mod(a, m)
    g, x, _ = exea(a, m)

    if g != 1
      return nil
    else
      return (x % m + m) % m
    end
  end
```

```
def jacobi(a, p)
  return 0 if exea(a, p)[0] != 1

  r = 0
  t = 1
  a = a % p

  while a != 0
    while a % 2 == 0
      a /= 2
    end

    t = -t if p % 8 == 3 || p % 8 == 5

    r = p
    p = a
    a = r

    t = -t if a % 4 == 3 && p % 4 == 3

    a = a % p

    return t if p == 1
  end
end

def sqrt(a, p)
  q = p - 1
  m = 0

  while q % 2 == 0
    q /= 2
    m += 1
  end

  b = rand(1..p)

  while lezhandr(b, p) != -1
    b = rand(1..p)
  end

  _as = [a]
  ks = [min_k(a, p, q)]
  k = ks[0]

  while k != 0
    a = (a * b ** (2 ** (m - k))) % p
    k = min_k(a, p, q)
    _as << a
    ks << k
  end

  rs = []
  r = (_as[-1] ** ((q + 1) / 2)) % p
  rs << r
```

```
    for i in 0.._as.length - 2
      bc = b ** (2 ** (m - ks[-i - 2] - 1))
      r = (rs[i] * inv_mod(bc, p)) % p
      rs << r
    end

    return rs[0]
  end

  def lezhandr(a, p)
    return 1 if a == 1

    b = a % p
    b -= p if b > p / 2

    t = b > 0 ? 2 : 1

    b = -b
    k = 0

    while b % 2 == 0
      b /= 2
      k += 1
    end

    c = b
    t_1 = 1

    t_1 = (-1) ** ((p - 1) / 2) if t % 2 == 1
    k_1 = 1
    k_1 = (-1) ** ((p * p - 1) / 8) if k % 2 == 1

    return t_1 * k_1 if c == 1
    return t_1 * k_1 * (-1) ** (((c - 1) / 2) * ((p - 1) / 2)) *
lezhandr(p, c)
  end

  private

  def exea(a, b)
    if a == 0
      return b, 0, 1
    end

    gcd, x1, y1 = exea(b % a, a)
    x = y1 - (b / a) * x1
    y = x1

    return gcd, x, y
  end

  def linear_comparison(a, b, m)
    x, _ = diophantus(a, m, b)
    x %= m
    return x
  end
```

```
  def min_k(a, p, q)
    k = 0

    while (a ** (2 ** k * q)) % p != 1
      k += 1
    end

    return k
  end
end

require './methods.rb'

@methods = Methods.new

def s1
  puts "\nВведите дробь (числитель знаменатель):"
  input = gets.strip.split.map(&:to_i)
  puts "Разложение в цепную дробь:
#{@methods.make_chain(input[0], input[1])}"
end

def s2
  puts "\nВведите параметры Диофантового уравнения (a b c):"
  input = gets.strip.split.map(&:to_i)
  puts "Решение Диофантового уравнения:
#{@methods.diophantus(input[0], input[1], input[2])}"
end

def s3
  puts "\nВведите чило и модуль для поиска обратного (a m):"
  input = gets.strip.split.map(&:to_i)
  puts "Обратный элемент в заданных параметрах:
#{@methods.inv_mod(input[0], input[1])}"
end

def s4
  puts "\nВведите параметры сравнения (a b m):"
  input = gets.strip.split.map(&:to_i)
  puts "Решение линейного сравнения:
#{@methods.diophantus(input[0], input[2], input[1])[0] % input[2]}"
end

def s5
  puts "\nВведите параметры (a p):"
  input = gets.strip.split.map(&:to_i)
  puts "Символ Лежандра: #{@methods.lezhandr(input[0],
input[1])}"
  puts "Символ Якоби: #{@methods.jacobi(input[0], input[1])}"
end

def s6
  puts "\nВведите параметры (a p):"
  input = gets.strip.split.map(&:to_i)
  puts "Квадратный корень a по модулю p:
#{@methods.sqrt(input[0], input[1])}"
end
```

```
def go
  input = -1

  while input != 0
    puts "\n[1] --> разложение в цепную дробь"
    puts "\n[2] --> решение диофантова уравнения"
    puts "\n[3] --> вычисление обратного элемента в кольце"
    puts "\n[4] --> решение линейного стравнения"
    puts "\n[5] --> вычисление символов Лежандра и Якоби"
    puts "\n[6] --> извлечение квадратных корней в кольце
вычетов"
    puts "\n[0] --> выход из программы"

    puts "\n\nВведите выбор:"
    input = gets.strip.to_i

    case input
    when 1
      s1
    when 2
      s2
    when 3
      s3
    when 4
      s4
    when 5
      s5
    when 6
      s6
    else
      puts "\n\nКонец работы программы"
      return
    end
  end
end

go
```