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 квадратные сравнения

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Сенокосова Владислава Владимировича

Преподаватель, профессор _____ В.А. Молчанов
подпись, дата

Саратов 2024

Содержание

1 Цель работы и порядок выполнения

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

Порядок выполнения работы

1. Разобрать алгоритм разложения чисел в цепную дробь и привести их программную реализацию.

2. Рассмотреть алгоритмы приложений цепных дробей и привести их программную реализацию.

3. Разобрать алгоритмы вычисления символов Лежандра и Якоби и привести их программную реализацию.

4. Рассмотреть алгоритмы извлечения квадратного корня в кольце вычетов.

2 Алгоритм разложения несократимой дроби в цепную дробь

С помощью алгоритма Евклида любое рациональное число можно представить в виде специальной конструкции, которая называется цепной дробью и которая играет важную роль в алгебре, теории чисел 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:

$$a_0 = a_1 q_1 + a_2, 0 \le a_2 < a_1,$$
$$a_1 = a_2 q_2 + a_3, 0 \le a_3 < a_2,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots..$$
$$a_{k-2} = a_{k-1} q_{k-1} + a_k, 0 \le a_k < a_{k-1},$$
$$a_{k-1} = a_k q_k, a_k = \text{НОД}(a_0, a_1) = 1$$

Эти равенства можно переписать в виде:

$$\frac{a_0}{a_1} = q_1 + \frac{a_2}{a_1}; \frac{a_1}{a_2} = q_2 + \frac{a_3}{a_2}, \ldots, \frac{a_{k-2}}{a_{k-1}} = q_{k-1} + \frac{a_k}{a_{k-1}} = q_{k-1} + \frac{1}{q_k}$$

Тогда рациональное число r можно представить следующим образом:

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

где q_1– целое число и q_2, …, q_k – целые положительные числа.

Определение. Выражение такого вида

$$q_1 + \frac{1}{q_2 + \frac{1}{\ldots + \frac{1}{q_{k-1} + \frac{1}{q_k}}}}$$

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

Таким образом, любая несократимая рациональная дробь $\frac{a_0}{a_1}$ может быть представлена в виде цепной дроби $\frac{a_0}{a_1} = (q_1; q_2, ..., q_k)$ с неполными частными $q_1; q_2, ..., q_k$, полученными в результате вычисления по алгоритму Евклида наибольшего общего делителя взаимно простых чисел a_0, a_1.

Определение. Для цепной дроби $\frac{a_0}{a_1} = (q_1; q_2, ..., q_k)$ выражения

$$\delta_1 = q_1, \delta_2 = q_1 + \frac{1}{q_2}, \delta_3 = q_1 + \frac{1}{q_2 + \frac{1}{q_3}}, \ldots, \delta_k = q_1 + \frac{1}{q_2 + \frac{1}{\ldots + \frac{1}{q_{k-1} + \frac{1}{q_k}}}}$$

называются подходящими дробями конечной цепной дроби $(q_1; q_2, ..., q_k)$ и обозначаются символами $\delta_i = (q_1; q_2, ..., q_i)$ где $1 \le i \le k$.

Аналогично определяются подходящие дроби $\delta_i = (q_1; q_2, ..., q_i)$ для бесконечной цепной дроби $(q_1; q_2, ..., q_k, ...)$.

Каждая подходящая дробь $\delta_i = (q_1; q_2, ..., q_i)$ является несократимой рациональной дробью $\delta_i = \frac{P_i}{Q_i}$ с числителем P_i и знаменателем Q_i, которые вычисляются по следующим рекуррентным формулам:

$$P_i = q_i P_{i-1} + P_{i-2}, Q_i = q_i Q_{i-1} + Q_{i-2}$$

с начальными условиями $P_{-1} = 0, P_0 = 1, Q_{-1} = 1, Q_0 = 0$

Вычисление числителей и знаменателей подходящих дробей с учетом

$$P_i = q_i P_{i-1} + P_{i-2}, Q_i = q_i Q_{i-1} + Q_{i-2}$$

оформляется в виде следующей таблицы:

i	-1	0	1	2	3	…	k-1	k
q_i			q_1	q_2	q_3		q_{k-1}	q_k
P_i	0	1	P_1	P_2	P_3		P_{k-1}	P_k
Q_i	1	0	Q_1	Q_2	Q_3		Q_{k-1}	Q_k

Сложность построения цепных дробей вытекает из сложности работы алгоритма Евклида. В ходе работы был применён классический алгоритм, а значит для него справедлива теорема Ламе и количество итераций (делений) не превосходит $1 + log_R N$. Где R – это золотое сечение равное $\frac{1+\sqrt{5}}{2}$ (корень квадратного уравнения $x^2 - x - 1 = 0$), а N – такое, что в алгоритме Евклида для $a, b: 0 < b < a \leq N$. Так как операция деления занимает n^2, то общая сложность будет $O\left(n^2 * (1 + log_R N)\right)$, то есть можно сказать, что сложность алгоритма составляет $O(n^2)$.

Описание алгоритма построения по заданной несократимой дроби $\frac{a}{b}$ цепной:

Вход: два целых числа a и b, такие что $a > 0$ и $b > 0$.

Выход: список коэффициентов разложения в цепную дробь.

1. Если $a < b$, поменять местами a и b.
2. Инициализировать пустой список для хранения коэффициентов цепной дроби.
3. Пока $b \neq 0$:
   - 3.1 Найти частное от деления a на b и добавить его в список.
   - 3.2 Присвоить a значение b, а b — остаток от деления предыдущего a на b.
4. Когда $b = 0$, вернуть список коэффициентов цепной дроби.

Псевдокод алгоритма:

Начало алгоритма

Функция $chain\_div(a, b)$:

    Если $a < b$:

        Поменять местами a и b

    Инициализировать пустой список res

    Пока $b != 0$:

        $div\_ = a // b$

Добавить $div\_$ в res

Присвоить a значение b

Присвоить b значение остатка $a\ mod\ b$

Вернуть res

Конец алгоритма

Сложность алгоритма: $O(n^2)$

Описание алгоритма построения по заданной несократимой дроби $\frac{a}{b}$ подходящих дробей:

Вход: два целых числа a и b, такие что $a > 0$ и $b > 0$.

Выход: два списка: P и Q, содержащие числители и знаменатели последовательных подходящих дробей.

1. Инициализировать два списка $P = [0,1]$ и $Q = [1,0]$, которые будут хранить числители и знаменатели подходящих дробей соответственно.
2. Пока $b \neq 0$:
   - 2.1 Вычислить целую часть q от деления a на b.
   - 2.2 Добавить в список P следующее значение: $P[-1] \times q + P[-2]$ (рекуррентная формула для числителя).
   - 2.3 Добавить в список Q следующее значение: $Q[-1] \times q + Q[-2]$ (рекуррентная формула для знаменателя).
   - 2.4 Присвоить a значение b, а b — остаток от деления предыдущего на b.
3. Когда $b = 0$, вернуть списки P и Q, содержащие все числители и знаменатели подходящих дробей.

Сложность алгоритма: $O(n^2)$

Псевдокод алгоритма

Начало алгоритма

Функция $suitable\_fractions(a, b)$:

Инициализировать $P\ =\ [0, 1], Q\ =\ [1, 0]$

Пока $b\ !=\ 0$:

$q = a // b$

Добавить $q * P[-1] + P[-2]$ в P

Добавить $q * Q[-1] + Q[-2]$ в Q

Присвоить a значение b

Присвоить b значение остатка $a \ mod \ b$

Вернуть P и Q

Конец алгоритма

Сложность алгоритма: $O(n^2)$

3 Алгоритмы приложений цепных дробей

3.1 Решение 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.

Важные свойства подходящих дробей:

1. Числители и знаменатели двух последовательных подходящих дробей удовлетворяют равенству $P_i Q_{i-1} - P_{i-1} Q_i = (-1)^i$ для всех $i = 1, \dots, k$

2. P_i, O_i взаимно просты

3. $\frac{P_i}{Q_i} - \frac{P_{i-1}}{Q_{i-1}} = \frac{(-1)^i}{Q_i Q_{i-1}}$

4. $\delta_{2n} > \delta_{2n-1}$

5. $P_i Q_{i-2} - P_{i-2} Q_i = q_i (-1)^{i-1}$

6. $\frac{P_i}{Q_i} - \frac{P_{i-1}}{Q_{i-1}} = \frac{q_i (-1)^{i-1}}{Q_i Q_{i-2}}$

7. $\delta_{2n} = \frac{P_{2n}}{Q_{2n}}$ – убывающая последовательность

$\delta_{2n-1} = \frac{P_{2n-1}}{Q_{2n-1}}$ – возрастающая последовательность

$$\delta_1 < \delta_3 < \cdots < \delta_{2n-1} \dots \leq \frac{a}{b} \leq \cdots \delta_{2n} \dots . < \delta_4 < \delta_2$$

8. $\frac{P_n}{Q_n} = q_1 + \sum_{i=1}^{n} \frac{(-1)^i}{Q_i Q_{i-1}}$ сходится по признаку Лейбница

9. $Q_n = q_n Q_{n-1} + Q_{n-2} \geq Q_{n-1} + Q_{n-2} \geq 2 Q_{n-2} \geq 2^{\frac{n-2}{2}}$

10. $|a - \delta_i| \leq |\delta_i - \delta_{i-1}| = \frac{1}{Q_i Q_{i-1}}$

11. Если все $q_i > 0$, то $\lim_{n \to \infty} \delta_n = a$, в частности, любое число $a \in R_+$ представляется цепной дробью.

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 $P_kQ_{k-1} - P_{k-1}Q_k = (-1)^k$ следует, что $a(-1)^kQ_{k-1} - b(-1)^kP_{k-1} = 1$ то есть значения являются целочисленными решениями уравнения $ax - by = 1$. Легко видеть, что все целые решения исходного диофантова уравнения $ax - by = 1$ находятся по формулам:

$$x = (-1)^kQ_{k-1} + bt, \qquad y = (-1)^kP_{k-1} + at$$

где t – произвольное целое число.

Нетрудно убедиться, что все решения диофантова уравнения

$$ax - by = c$$

с взаимно простыми коэффициентами a, b находятся по формулам:

$$x = (-1)^kcQ_{k-1} + bt, \qquad y = (-1)^kcP_{k-1} + at$$

где t – произвольное целое число.

Описание алгоритма решения Диофантового уравнения:

Вход: три целых числа a, b, c, где уравнение имеет вид $ax + by =$ c

Выход: одно решение (x, y) системы линейного Диофантова уравнения.

1. Найти наибольший общий делитель $gcd(a, b)$.
2. Если c делится на $gcd(a, b)$, то:
   - 2.1 Упрощаем уравнение: делим a, b, c на $gcd(a, b)$.
   - 2.2 Вычисляем подходящие дроби для a и b с помощью алгоритма цепных дробей.
   - 2.3 Определяем k, длину списка числителей P.
   - 2.4 Вычисляем решения x и y с помощью формул:
     - 2.4.1 $x = (-1)^k \times c \times Q[-2]$, где Q — список знаменателей.
     - 2.4.2 $y = (-1)^k \times c \times P[-2]$, где P — список числителей.
3. Возвращаем одно решение (x, y).

Если условие делимости c на $gcd(a, b)$ не выполняется, то решение не существует.

Сложность алгоритма: Так как основывается на вычислении подходящих дробей, то сложность можно оценить как $O(n^2)$

Псевдокод алгоритма:

Начало алгоритма

Функция $diofant(a, b, c)$:

Найти $gcd\_ = gcd(a, b)$

Если $c \bmod gcd\_ == 0$:

$a = a // gcd\_$

$b = b // gcd\_$

$c = c // gcd\_$

Вычислить P и Q с помощью $suitable\_fractions(a, b)$

k = длина списка P

$x = (-1)\text{^}k * c * Q[-2]$

$y = (-1)\text{^}k * c * P[-2]$

Вернуть x, y

Конец алгоритма

Сложность алгоритма: $O(n^2)$

3.2 Решение линейных сравнений

Пусть НОД$(a, m) = d$. Сравнение $ax \equiv b(mod\ m)$ невозможно, если b не делится на d. При b, кратном d, сравнение имеет d решений.

Поиск решений. Так как ранее были рассмотрены решения диофантовых уравнений, то для нахождения решения достаточно просто решить диофантовое уравнение $ax - my = b$, а затем взять найденное значение x, после чего взять его по модулю m. Сложность нахождения корня линейного сравнения основывается на сложности алгоритма Евклида, то есть O$(log^2 N)$. Где $N - max(a, m)$, (в случае бинарных операций) и $O(n^2)$ при обычных арифметических операциях.

Алгоритм решения линейных сравнений:

Вход: три целых числа a, b, n, где уравнение имеет вид $ax \equiv b \ (mod\ n)$.

Выход: одно решение x уравнения.

1. Решить Диофантово уравнение $ax - bn = 0$ (или эквивалентно $ax + ny = b$) с помощью функции решения Диофантова уравнения $diofant(a, n, b)$.
2. Получить значение x, которое будет решением линейного сравнения $ax \equiv b \ (mod\ n)$.
3. Вернуть x как результат.

Сложность алгоритма: $O(n^2)$

Псевдокод реализованного алгоритма:

Начало алгоритма

Функция $system\_comparisons(a, b, n)$:

Решить уравнение $ax - ny = b$ с помощью $diofant(a, n, b)$

Получить x из решения $diofant(a, n, b)$

Вернуть x

Конец алгоритма

Сложность алгоритма: $O(n^2)$

3.3 Поиск обратного элемента в кольце вычетов Z_m

Пусть задан некоторый натуральный модуль m, и рассмотрим кольцо, образуемое этим модулем (т.е. состоящее из чисел от 0 до m-1). Тогда для некоторых элементов этого кольца можно найти обратный элемент. Обратным к числу a по модулю m называется такое число b, что: $a \cdot b \equiv 1 \pmod m$, и его нередко обозначают как a^{-1}.

Понятно, что для нуля обратного элемента не существует никогда; для остальных же элементов обратный может как существовать, так и нет. Утверждается, что обратный существует только для тех элементов, которые взаимно просты с модулем m. Рассмотрим вспомогательное уравнение (относительно неизвестных x и y): $ax + my = 1$

Это линейное диофантово уравнение второго порядка. Из условия НОД(a, m) = 1 следует, что это уравнение имеет решение, которое можно найти с помощью расширенного алгоритма Евклида (отсюда же, кстати говоря, следует, что при НОД(a, m) $\neq 1$, решения, а потому и обратного элемента, не существует).

С другой стороны, если мы возьмём от обеих частей уравнения остаток по модулю m, то получим: $ax \equiv 1 \pmod{m}$. Таким образом, найденный x и будет являться обратным к a. Сложность нахождения обратного элемента основывается на сложности алгоритма Евклида, то есть $O(log^2 N)$. Где $N - max(a, m)$

Алгоритм поиска обратного элемента в кольце вычетов:

Вход: два целых числа a и m, где a — число, для которого нужно найти обратный элемент по модулю m.

Выход: число x, такое что $ax \equiv 1 \ (mod\ m)$

1. Решить Диофантово уравнение $ax + my = 1$ с помощью функции $diofant(a, m, 1)$. Это уравнение находит такие x и y, что $ax + my = 1$.
2. Вернуть x, так как это и есть обратный элемент числа a по модулю m, то есть $ax \equiv 1 \ (mod\ m)$.

Сложность алгоритма: $O(n^2)$

Псевдокод реализованного алгоритма:

Начало алгоритма

Функция $inv\_el(a, m)$:

Решить уравнение $ax + my = 1$ с помощью $diofant(a, m, 1)$

Получить x из решения $diofant(a, m, 1)$

Вернуть x

Конец алгоритма

Сложность алгоритма: $O(n^2)$

4 Алгоритмы вычисления символов Лежандра и Якоби

4.1 Символ Лежандра

Число $a \in \mathrm{Z}_p$ называется квадратичным вычетом по модулю p, если ($\exists x \in \mathrm{Z}$) $x^2 \equiv a\ (mod\ p)$.В противном случае число a называется квадратичным невычетом по модулю p. Символом Лежандра числа $a \in \mathrm{Z}$ называется выражение:

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

Вычисление символа Лежандра осуществляется на основе свойств основным из которых является критерий Эйлера, согласно которому любой квадратичный вычет удовлетворяет сравнению $a^{\frac{p-1}{2}} = 1\ (mod\ p)$, а квадратичный невычет удовлетворяет $a^{\frac{p-1}{2}} = -1\ (mod\ p)$. Таким образом имеем следующие свойства, на которых будет основываться алгоритм вычисления символа Лежандра:

1. $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}}\ (mod\ p)$. В частности, $\left(\frac{1}{p}\right)$ для любого простого $p \neq 2$, $\left(\frac{-1}{p}\right) = 1$ при $p = 1\ (mod\ 4)$ и $\left(\frac{-1}{p}\right) = -1$ при $p = 3\ (mod\ 4)$
2. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$
3. $\left(\frac{b}{p}\right)\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right)$
4. $\left(\frac{a+kp}{p}\right) = \left(\frac{a}{p}\right)$ для любого $k \in Z$
5. $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1, \text{если } p = \pm 1\ (mod\ 8) \\ -1, \text{если } p = \pm 3\ (mod\ 8) \end{cases}$
6. Квадратичный закон взаимности Гаусса

$$\left(\frac{p}{q}\right) = \left(\frac{q}{p}\right)(-1)^{\frac{(p-1)}{2}\frac{q-1}{2}}$$

Для любых нечетных простых чисел p, q.

Алгоритм вычисления символа Лежандра:

Вход: два числа a и p, где p — нечетное простое число.

Выход: символ Лежандра $\left(\frac{a}{p}\right)$, который показывает, является ли a квадратичным вычетом по модулю p.

1. Если $a < 0$, то по свойству 2 выделяем множитель $\left(\frac{-1}{p}\right)$
2. Заменяем a на остаток от деления на p
3. Представляем $a = p_1^{\alpha} * \ldots * p_k^{\alpha}$ и вычисляем по свойству 2

$$\left(\frac{a}{p}\right) = \left(\frac{p_1}{p}\right)^{\alpha_1} * \ldots * \left(\frac{p_k}{p}\right)^{\alpha_k}$$

   При этом опускаем множители сс четными степенями α_i и вместо множителей $\left(\frac{p_i}{p}\right)^{\alpha_i}$ с нечетными степенями α_i оставляем $\left(\frac{p_i}{p}\right)$,

4. Если $p_i = 2$, то вычисляем по свойству 5 $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$
5. К остальным символам $\left(\frac{p_i}{p}\right)$ применяется квадратичный закон взаимности Гаусса
6. При необходимости возвращаемся к пункту 2

Сложность алгоритма: $O(\log^2 p)$

Псевдокод реализованного алгоритма:

```
Начало алгоритма
Функция legendre(a, p):
    Если p не является простым:
    Вернуть None
    Если a < 0:
        Если p % 4 == 1:
            Вернуть legendre(-a, p)
        Иначе:
            Вернуть -legendre(-a, p)
    a = a % p
    Если a == 0:
```

Вернуть 0

Если $a == 1$:

Вернуть 1

Если $a == 2$:

Если $p \% 8 == 1$ или $p \% 8 == 7$:

Вернуть 1

Иначе:

Вернуть -1

Если a чётное:

Вернуть $legendre(2, p) * legendre(a // 2, p)$

Если a нечётное: Если $a \% 4 == 3$ и $p \% 4 == 3$:

Вернуть $-legendre(p, a)$

Иначе: Вернуть $legendre(p, a)$

Конец алгоритма

Сложность алгоритма: $O(\log^2 p)$

4.2 Символ Якоби

Символ Якоби для простого числа n совпадает с символом Лежандра и удовлетворяет почти всем свойствам символа Лежандра (хотя в общем случае не связан с квадратичными вычетами). Главным образом, символ Якоби используется для быстрого вычисления символа Лежандра. Символ Лежандра, в свою очередь, необходим для проверки разрешимости квадратичного сравнения по модулю простого числа. Но считать его по определению — достаточно долгая по времени процедура.

С помощью алгоритма быстрого возведения в степень это делается за $O(log^3 p)$ битовых операций (если не использовать быстрое умножение и деление). А вычисление символа Якоби требует только $O(log^2 p)$ битовых операций.

Алгоритм вычисления символа Якоби:

Вход: два числа a и p, где p — нечётное целое число. **Выход:** символ Якоби $\left(\frac{a}{p}\right)$, который обобщает символ Лежандра для случая, когда p не является простым числом.

1. Заменить a на такое b, что $a = b\ (mod\ p)$ и $|b| < \frac{p}{2}$
2. Если $b < 0$, то по свойству 2 рассмотренном при описании символа Лежандра выделяем множитель $\left(\frac{-1}{p}\right)$
3. Если b – четное, то представляем $b = 2^t a_1$ и при нечетном t вычисляем $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$
4. К символу $\left(\frac{a_1}{p}\right)$ применяется квадратичный закон взаимности Гаусса
5. При необходимости возвращаемся к пункту 1

Сложность реализованного алгоритма: $O(log^2 n)$

Псевдокод реализованного алгоритма:

Начало алгоритма

Функция $jacobi(a, n)$:

$a = a \,\%\, n$

$t = 1$

Пока $a\ != 0$:

Пока $a \,\%\, 2 == 0$:

$a = a \,//\, 2$

$r = n \,\%\, 8$

Если $r == 3$ или $r == 5$:

$t = -t$

Поменять местами a и n

Если $a \,\%\, 4 == 3$ и $n \,\%\, 4 == 3$:

$t = -t$

$a = a \,\%\, n$

Если $n == 1$:

Вернуть t

Иначе:

Вернуть 0

Конец алгоритма

Сложность реализованного алгоритма: $O(log^2 n)$

5 Алгоритмы извлечения квадратного корня в кольце вычетов

Для решения сравнения $x^2 = n \, mod \, p$, где n – квадратичный вычет, p – нечетное простое число можно использовать алгоритм Тонелли — Шенкса который записывается следующим образом.

Алгоритм вычисления квадратного корня в кольце вычетов:

Вход: целые числа a и простое p, где нужно найти x, такое что $x^2 \equiv a \, (mod \, p)$.

Выход: два значения r и $p - r$, являющиеся решениями уравнения $x^2 \equiv a \, (mod \, p)$, если решение существует.

1. Проверить, является ли a квадратичным вычетом по модулю p с помощью символа Лежандра. Если символ Лежандра $\left(\frac{a}{n}\right) \neq 1$, решения не существует, вернуть $None, None$.
2. Если$p = 3 \, (mod \, 4)$, решение вычисляется напрямую как $r = a^{\frac{(p+1)}{4}} \, mod \, p$.
3. Найти такие q и s, что $p - 1 = q * 2^s$, где q нечетное
4. Найти z, такой что $\left(\frac{z}{p}\right) = -1$ (то есть z не является квадратичным вычетом по модулю p.
5. Инициализировать $m = s, c = z^q \, (mod \, p), t = a^q (mod \, p), r = a^{\frac{q+1}{2}} (mod \, p)$
6. Пока $t \neq 1$:

   6.1 Найти минимальное i, такое что $t^{2^i} = 1 \, (mod \, p)$

   6.2 Вычислить $b = c^{2^{m-i-1}} (mod \, p)$

   6.3 Обновите значения: $m = i, c = b^2 (mod \, p), t = t * c \, (mod \, p), r = r * b \, (mod \, p)$
7. Вернуть два решения r и $p - r$.

Сложность алгоритма: $O(\log^4 p)$

Псевдокод реализованного алгоритма:

Начало алгоритма

Функция $mod\_sqrt(a, p)$:

Если $legendre(a, p)\ !=\ 1$:

Вернуть None, None

Если p % $4\ ==\ 3$:

$r\ =\ a\hat{}\{(p\ +\ 1)\ //\ 4\}$ % p

Вернуть $r, p\ -\ r$

$q\ =\ p\ -\ 1$

$s\ =\ 0$

Пока q % $2\ ==\ 0$:

$q\ =\ q\ //\ 2$

$s\ +=\ 1$

$z\ =\ 2$

Пока $legendre(z, p)\ !=\ -1$:

$z\ +=\ 1$

$m\ =\ s$

$c\ =\ z\hat{}q$ % p

$t\ =\ a\hat{}q$ % p

$r\ =\ a\hat{}\{(q\ +\ 1)\ //\ 2\}$ % p

Пока $t\ !=\ 1$:

$t\_pow\ =\ t$

Для i от 1 до m:

$t\_pow\ =\ t\_pow\hat{}2$ % p

Если $t\_pow\ ==\ 1$:

Прервать цикл

$b\ =\ c\hat{}\{2\hat{}\{m\ -\ i\ -\ 1\}\}$ % p

$m\ =\ i$

$c\ =\ b\hat{}2$ % p

$t = t * c \% p$

$r = r * b \% p$

Вернуть $r, p - r$

Конец алгоритма

Сложность алгоритма: $O(\log^4 p)$

6 Тестирование реализованных алгоритмов

6.1 Алгоритм разложения чисел в цепную дробь

Вычислим цепные дроби для дробей $\frac{15}{7}$ и $\frac{5}{8}$. В результате получим следующие разложения $\frac{15}{7} = [2, 7]$, $\frac{5}{8} = [1,1,1,2]$. Подробная информация представлена на рисунке 1.

```
Укажите номер операции:
1 - Разложение дроби вида a / b в цепную
2 - Вычисление подходящих дробей
3 - Решение Диофантового уравнения
4 - Решение линейных сравнений
5 - Вычисление обратного элемента в кольце вычетов
6 - Вычисление символа Лежандра и Якоби
7 - Извлечение квадратного корня в кольце вычетов
8 - Выход
:>1
Выбрано: Разложение дроби вида a / b в цепную
a = 15
b = 7
Цепная дробь: [2, 7]
:>1
Выбрано: Разложение дроби вида a / b в цепную
a = 5
b = 8
Цепная дробь: [1, 1, 1, 2]
:>
```

Рисунок 1 – Вычисление цепных дробей

Вычислим теперь для данных примеров подходящие дроби. Данные отображены на рисунке 2.

```
:>2
Выбрано: Вычисление подходящих дробей для дроби вида a / b
a = 5
b = 8
Числители P:  [0, 1, 0, 1, 1, 2, 5]
Знаменатели Q:  [1, 0, 1, 1, 2, 3, 8]
:>2
Выбрано: Вычисление подходящих дробей для дроби вида a / b
a = 15
b = 7
Числители P:  [0, 1, 2, 15]
Знаменатели Q:  [1, 0, 1, 7]
:>
```

Рисунок 2 – Вычисление подходящих дробей

6.1 Алгоритмы приложений цепных дробей

Решим диофантово уравнение следующего вида: $34x - 23y = 7$. Решение приведено на рисунке 3. Также на рисунке представлено решение для произвольного параметра $t = 34$.

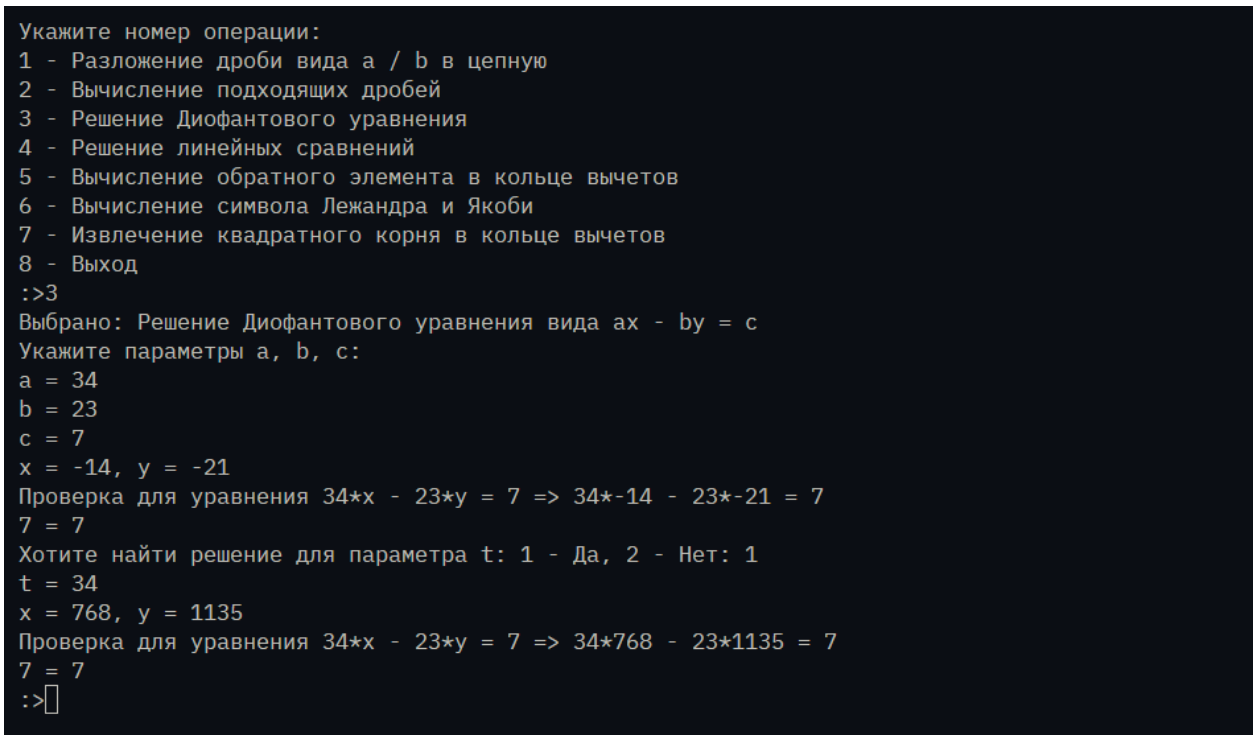

Рисунок 3 – Решение диофантового уравнения

Решим линейное сравнение следующего вида: $4x = 5\ (mod\ 11)$, для такого сравнения $x = 15$. Проверка и решение представлено на рисунке 4.

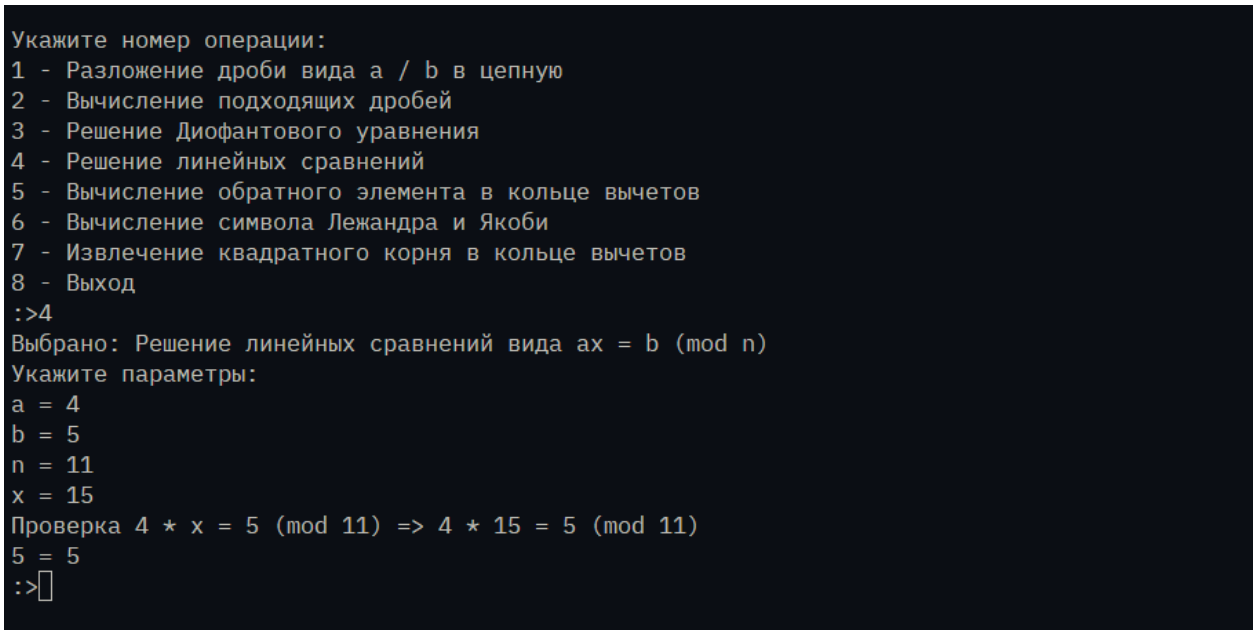

Рисунок 4 – Решение линейных сравнений

Вычислим обратный элемент для числа 5 по модулю 7, ответом будет 3. Рисунок 5 демонстрирует решение данной задачи.

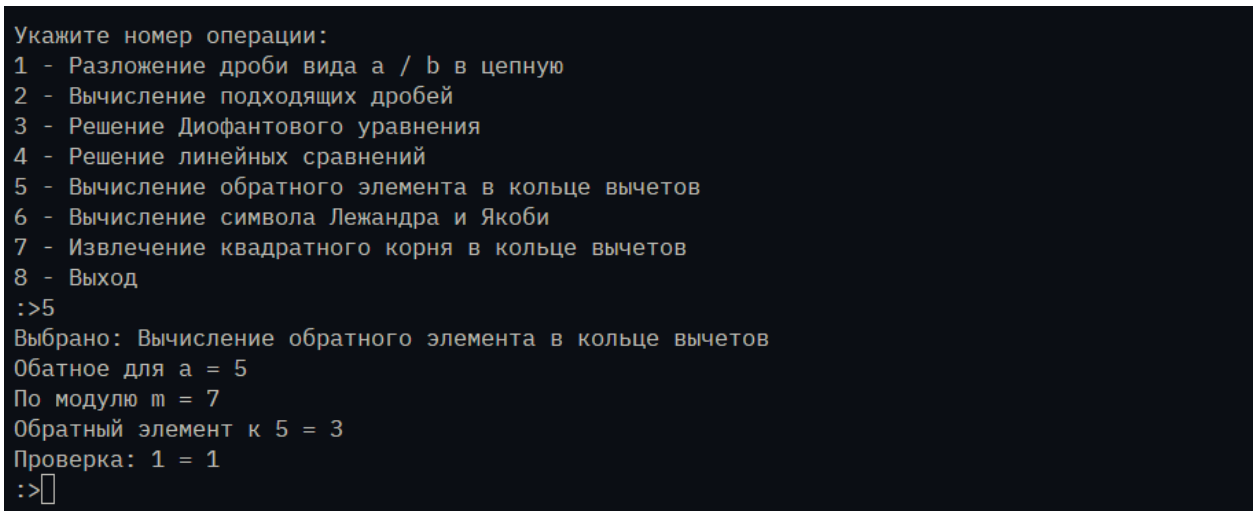

Рисунок 5 – Вычисление обратного элемента в кольце вычетов

6.2 Алгоритмы вычисления символов Лежандра и Якоби

Вычислим символы Лежандра и Якоби для: $\left(\frac{5}{21}\right)$, $\left(\frac{5}{7}\right)$. Заметим, что символ Лежандра требует по условию простое p, а символ Якоби не накладывает таких условий. Решение представлено на рисунке 6.

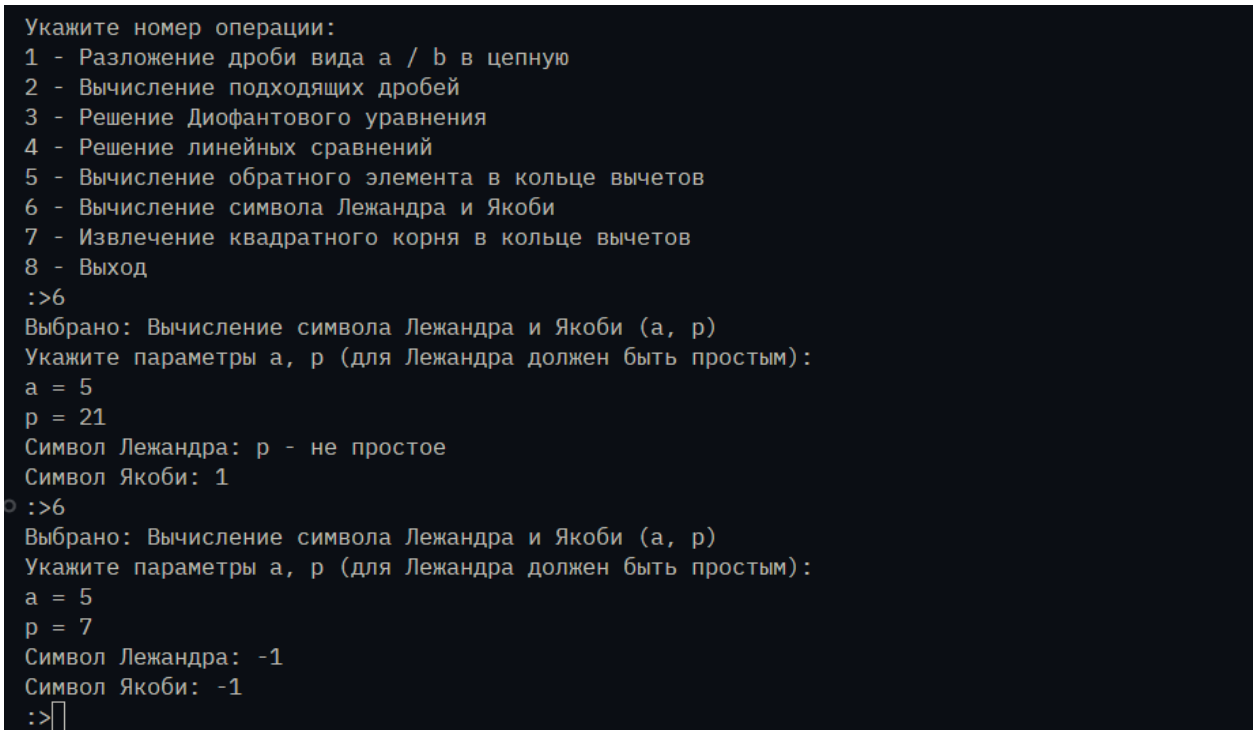

Рисунок 6 – Вычисление символов Лежандра и Якоби

6.3 Алгоритмы извлечения квадратного корня в кольце вычетов

Вычислим: $x^2 = 10\ mod\ 13, x^2 = 186\ mod\ 401, x^2 = 5\ mod\ 9$, решения с результатами тестирования представлены на рисунке 7.

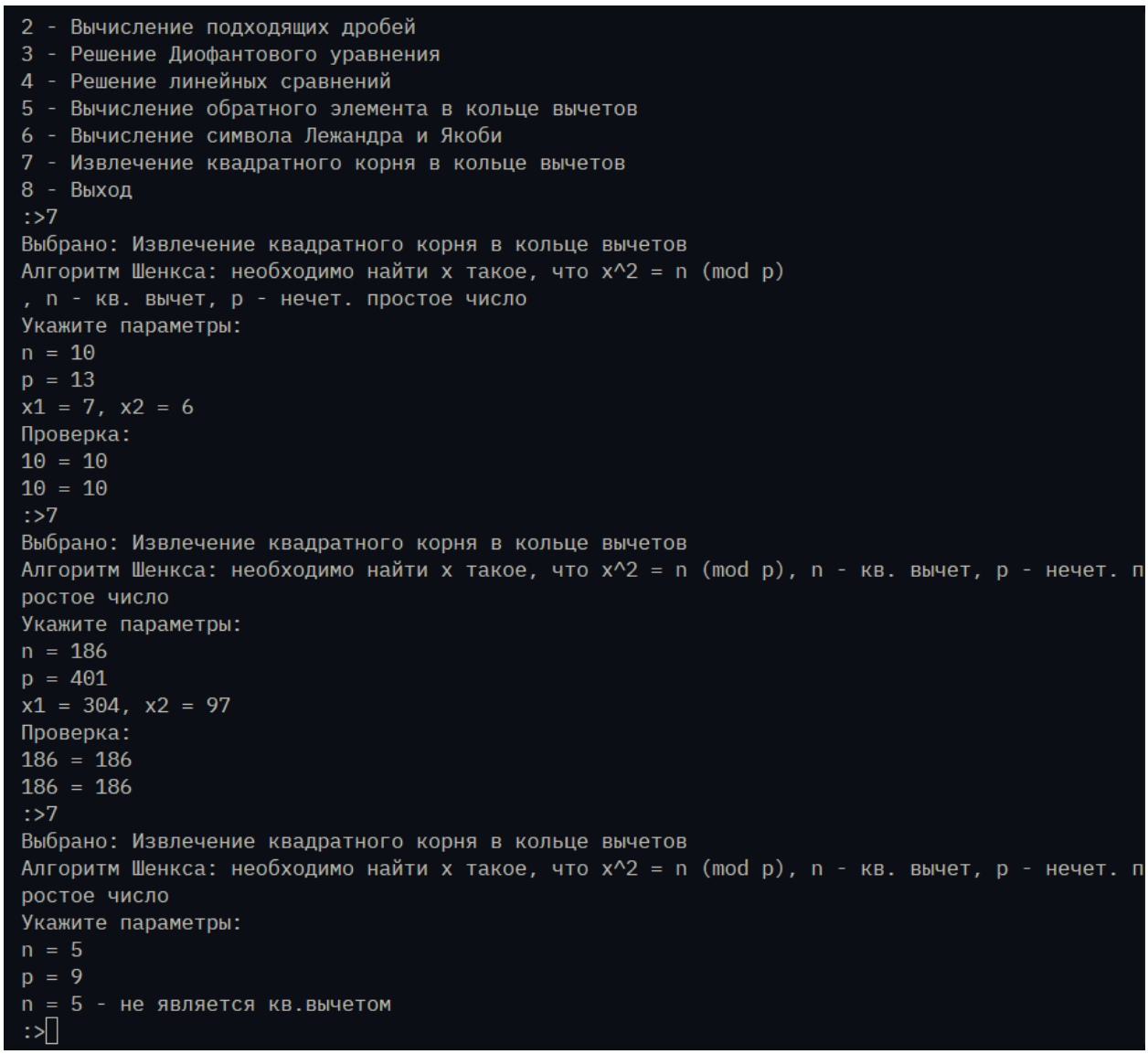

Рисунок 7 - Извлечение квадратного корня в кольце вычетов

ЗАКЛЮЧЕНИЕ

В данной работе были изучены основные свойства цепных дробей и квадратных сравнений, а также их применение в решении различных задач теории чисел.

В первой части работы был разобран алгоритм разложения чисел в цепные дроби, который является важным инструментом для представления чисел в виде последовательности приближений. Программная реализация алгоритма показала, что цепные дроби могут быть эффективно использованы для представления иррациональных чисел.

Во второй части работы были рассмотрены приложения цепных дробей, такие как:

1. Поиск обратного элемента в кольце вычетов,
2. Решение линейных сравнений,
3. Решение диофантовых уравнений.

Эти задачи имеют большое значение в теории чисел и криптографии. Программная реализация показала, что цепные дроби могут быть полезны для нахождения обратных элементов, решения систем линейных сравнений по модулю и нахождения целочисленных решений диофантовых уравнений.

Далее в работе были исследованы алгоритмы вычисления символов Лежандра и Якоби, которые позволяют определить, является ли число квадратичным вычетом по модулю простого или составного числа.

Наконец, была рассмотрена задача извлечения квадратного корня в кольце вычетов, в том числе алгоритм Тонелли-Шенкса, который решает квадратные сравнения по модулю простого числа.

Таким образом, данная работа позволила исследовать как теоретические основы, так и практические аспекты применения цепных дробей и квадратных сравнений. Полученные знания и программные реализации могут быть использованы для решения задач теории чисел, криптографии и других областей, требующих работы с числами в кольце вычетов.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

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

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

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

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

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

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

ПРИЛОЖЕНИЕ А

Реализованные программы для лабораторной работы

```python
import sympy

# Обычный алгоритм Евклида (НОД двух чисел)
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a


# Цепная дробь
def chain_div(a, b):
    if a < b:
        a, b = b, a
    res = []
    while b:
        div_ = a // b
        res.append(div_)
        a, b = b, a % b
    return res


# Вычисление подходящих дробей
def suitable_fractions(a, b):
    P = [0, 1]
    Q = [1, 0]
    while b:
        q = a // b
        P.append(q * P[-1] + P[-2])
        Q.append(q * Q[-1] + Q[-2])
        a, b = b, a % b
    return P, Q


# Решение Диофантового уравнения
def diofant(a, b, c):
    gcd_ = gcd(a, b)
    if c % gcd_ == 0:
        a, b, c = a // gcd_, b // gcd_, c // gcd_
        P, Q = suitable_fractions(a, b)
        k = len(P)
        x = pow(-1, k) * c * Q[-2]
        y = pow(-1, k) * c * P[-2]
    else:
        return None, None
    return x, y
```

```python
# Общее решение Диофантового уравнения
def diofant_global(a, b, x, y, t):
    return x + b * t, y + a * t


# Решение линейных сравнений
def system_comparisons(a, b, n):
    x, _ = diofant(a, n, b)
    return x


# Получение обратного элемента
def inv_el(a, m):
    x, _ = diofant(a, m, 1)
    return x


# Символ Лежандра
def legendre(a, p):
    if not sympy.isprime(p):
        return None
    if a < 0:
        if p % 4 == 1:
            return legendre(-a, p)
        else:
            return -legendre(-a, p)
    a = a % p
    if a == 0:
        return 0
    if a == 1:
        return 1
    if a == 2:
        if p % 8 == 1 or p % 8 == 7:
            return 1
        else:
            return -1
    if a % 2 == 0:
        return legendre(2, p) * legendre(a // 2, p)
    if a % 2 == 1:
        if a % 4 == 3 and p % 4 == 3:
            return -legendre(p, a)
        else:
            return legendre(p, a)

# Символ Якоби
def jacobi(a, n):
    a = a % n
    t = 1
    while a != 0:
        while a % 2 == 0:
```

```
            a //= 2
            r = n % 8
            if r == 3 or r == 5:
                t = -t
        a, n = n, a
        if a % 4 == 3 and n % 4 == 3:
            t = -t
        a = a % n
    return t if n == 1 else 0


# Извлечение квадратного корня в кольце вычетов
def mod_sqrt(a, p):
    if legendre(a, p) != 1:
        return None, None
    if p % 4 == 3:
        r = pow(a, (p + 1) // 4, p)
        return r, p - r
    q = p - 1
    s = 0
    while q % 2 == 0:
        q //= 2
        s += 1
    z = 2
    while legendre(z, p) != -1:
        z += 1
    m = s
    c = pow(z, q, p)
    t = pow(a, q, p)
    r = pow(a, (q + 1) // 2, p)
    while t != 1:
        t_pow = t
        i = 0
        for i in range(1, m):
            t_pow = pow(t_pow, 2, p)
            if t_pow == 1:
                break
        b = pow(c, pow(2, (m - i - 1)), p)
        m = i
        c = pow(b, 2, p)
        t = (t * c) % p
        r = (r * b) % p
    return r, p - r


if __name__ == "__main__":
    type_ = print("""
Укажите номер операции:
1 - Разложение дроби вида a / b в цепную
2 - Вычисление подходящих дробей
```

```
3 - Решение Диофантового уравнения
4 - Решение линейных сравнений
5 - Вычисление обратного элемента в кольце вычетов
6 - Вычисление символа Лежандра и Якоби
7 - Извлечение квадратного корня в кольце вычетов
8 - Выход""")
    param = None
    while param not in ["1", "2", "3", "4", "5", "6", "7", "8"]:
        param = input(":>").strip()
        match param:
            case "1":
                print("Выбрано: Разложение дроби вида a / b в цепную")
                while not ((a := input("a = ").strip()).isdigit()):
                    a = input("a = ")
                while not ((b := input("b = ").strip()).isdigit()):
                    b = input("b = ")
                res = chain_div(int(a), int(b))
                print(f"Цепная дробь: {res}")
                param = None
            case "2":
                print("Выбрано:
                      Вычисление подходящих дробей для дроби вида a / b")
                while not ((a := input("a = ").strip()).isdigit()):
                    a = input("a = ")
                while not ((b := input("b = ").strip()).isdigit()):
                    b = input("b = ")
                P, Q = suitable_fractions(int(a), int(b))
                print("Числители P: ", P)
                print("Знаменатели Q: ", Q)
                param = None
            case "3":
                print("Выбрано:
                       Решение Диофантового уравнения вида ax - by = c")
                print("Укажите параметры a, b, c:")
                while not ((a := input("a = ").strip()).isdigit()):
                    a = input("a = ")
                while not ((b := input("b = ").strip()).isdigit()):
                    b = input("b = ")
                while not ((c := input("c = ").strip()).isdigit()):
                    c = input("c = ")
                x, y = diofant(int(a), int(b), int(c))
                if x is None:
                    print(f"Решений нет")
                else:
                    print(f"x = {x}, y = {y}")
                    print(f"Проверка для уравнения {a}*x - {b}*y = {c}
                           => {a}*{x} - {b}*{y} = {c}")
                    print(f"{int(a) * x - int(b) * y} = {c}")
                    flag = input("Хотите найти решение для параметра t:
                                 1 - Да, 2 - Нет: ")
```

```
                if flag == "1":
                    t = input("t = ")
                    x1, y1 = diofant_global(int(a), int(b), x,
                                            y, int(t))
                    print(f"x = {x1}, y = {y1}")
                    print(f"Проверка для уравнения {a}*x - {b}*y =
                           {c} => {a}*{x1} - {b}*{y1} = {c}")
                    print(f"{int(a) * x1 - int(b) * y1} = {c}")
            param = None
        case "4":
            print("Выбрано: Решение линейных сравнений вида
                  ax = b (mod n)")
            print("Укажите параметры:")
            while not ((a := input("a = ").strip()).isdigit()):
                a = input("a = ")
            while not ((b := input("b = ").strip()).isdigit()):
                b = input("b = ")
            while not ((n := input("n = ").strip()).isdigit()):
                n = input("n = ")
            x = system_comparisons(int(a), int(b), int(n))
            if x is None:
                print("Нет решений")
            else:
                print("x =", x)
                print(f"Проверка {a} * x = {b} (mod {n})
                       => {a} * {x} = {b} (mod {n})")
                print(f"{int(a) * x % int(n)} = {int(b) % int(n)}")
            param = None
        case "5":
            print("Выбрано:
                  Вычисление обратного элемента в кольце вычетов")
            a = input("Обатное для a = ")
            m = input("По модулю m = ")
            while gcd(int(a), int(m)) != 1:
                print(f"gcd({a}, {m}) != 1")
                a = input("Обатное для a = ")
                m = input("По модулю m = ")
            x = inv_el(int(a), int(m))
            print(f"Обратный элемент к {a} =", x % int(m))
            print(f"Проверка: {int(a) * x % int(m)} = 1")
            param = None
        case "6":
            print("Выбрано:
                   Вычисление символа Лежандра и Якоби (a, p)")
            print("Укажите параметры a, p
                  (для Лежандра должен быть простым): ")
            a = input("a = ")
            while not ((p := input("p = ").strip()).isdigit()):
                p = input("p = ")
            if a.split()[0] == "-":
```

```
                res1 = legendre(-int(a), int(p))
                res2 = jacobi(-int(a), int(p))
            else:
                res1 = legendre(int(a), int(p))
                res2 = jacobi(int(a), int(p))
            if res1 is None:
                res1 = "p - не простое"
            print(f"Символ Лежандра: {res1}")
            print(f"Символ Якоби: {res2}")
            param = None
        case "7":
            print("Выбрано:
                   Извлечение квадратного корня в кольце вычетов")
            print("Алгоритм Шенкса: необходимо найти x такое,
                  что x^2 = n (mod p), n - кв. вычет,
                  p - нечет. простое число")
            print("Укажите параметры:")
            n = input("n = ")
            p = input("p = ")
            res1, res2 = mod_sqrt(int(n), int(p))
            if res1 is None:
                print(f"n = {n} - не является кв.вычетом")
            else:
                print(f"x1 = {res1}, x2 = {res2}")
                print(f"Проверка:\n{res1**2 % int(p)} =
                      {int(n) % int(p)}\n{res2**2 % int(p)} =
                      {int(n) % int(p)}")
            param = None
        case "8":
            param = "8"
```