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

Алексеева Александра Александровича

Преподаватель

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

подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

1 Цель работы и порядок её выполнения

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

Порядок выполненных 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 Теория

2.1 Разложение 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:

$$
\begin{aligned}
&a_0 = a_1 q_1 + a_2,\ 0 \le a_2 < a_1,\\
&a_1 = a_2 q_2 + a_3,\ 0 \le a_3 < a_2,\\
&\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\\
&a_{k-2} = a_{k-1} q_{k-1} + a_k,\ 0 \le a_k < a_{k-1},\\
&a_{k-1} = a_k q_k,\quad \text{где } a_k = \text{НОД}(a_0, a_1) = 1.
\end{aligned}
$$

Эти равенства можно переписать в виде:

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

Тогда рациональное число r можно представить следующим образом:

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

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

<u>Определение</u>. Выражение вида

$$q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + q_{k-1} + \cfrac{1}{q_k}}}$$

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

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

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

Каждая подходящая дроби δ_i ($i = \overline{1, k}$) 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:

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

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

<u>Псевдокод разложения чисел в цепную дробь</u>

Вход. Целые числа a_0, a_1 : НОД(a_0, a_1) = 1.

Выход: Разложение чисел в цепную дробь (q_1; q_2, …, q_k), числитель подходящих дробей P, знаменатель подходящих дробей Q.

<u>Шаг 1</u>. Инициализировать массив *fractions*.

<u>Шаг 2</u>. Вычислить $q = a_0 / a_1$, $r = a_0 \pmod{a_1}$.

<u>Шаг 3</u>. Проверка r:

- Если $r = 0$, то переход на шаг 4.
- Если $r = 1$, то положить в массив *fractions* элемент a_1 и перейти на шаг 4.
- Иначе положить $a_0 = a_1$, $a_1 = r$ и перейти к шагу 2.

<u>Шаг 4</u>. Инициализировать массив P = [0, 1] и Q = [1, 0].

<u>Шаг 5</u>. Цикл по i от 1 до *fractions*.size().

Шаг 5.1. Положить в массив P значение *fractions*[i] • P[i + 1] + P[i].

Шаг 5.2. Положить в массив Q значение *fractions*[i] • Q[i + 1] + Q[i].

<u>Шаг 6</u>. Результат: разложения числа в цепную дробь *fractions*, числитель подходящих дробей P, знаменатель подходящих дробей Q.

2.2 Приложения цепных дробей

1. Решение линейных диофантовых уравнений $ax + by = c$.

2. Вычисление обратных элементов в кольце вычетов $\mathbf{Z}_m$.

3. Решение линейных сравнений $ax \equiv b \pmod{m}$.

2.2.1 Решение линейных диофантовых уравнений

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

Например, диофантовым уравнением является уравнение вида

$$ax - by = c$$

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 $P_kQ_{k-1} - P_{k-1}Q_k = (-1)^k$, $\frac{a}{b} = \delta_k = \frac{P_k}{Q_k}$ следует, что

$$a(-1)^k Q_{k-1} - b(-1)^k P_{k-1} = 1\ ,$$

т.е. значения $x = (-1)^k Q_{k-1}$, $y = (-1)^k P_{k-1}$ 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:

$$x = (-1)^k cQ_{k-1} + bt,\ \ y = (-1)^k cP_{k-1} + at,$$

где t – произвольное целое число.

Псевдокод решения линейных диофантовых уравнений

Вход: Целые числа a, b, c : НОД(a, b) = 1.

Выход: Целые числа x, y : $ax - by = c$.

Шаг 1. Найти разложение *fractions* чисел $\frac{a}{b}$ в цепную дробь.

Шаг 2. Найти числительно P и знаменатель Q подходящих дробей.

Шаг 3. Вычислить $x = (-1)^k \bullet Q[k-1] \bullet c$ и $y = (-1)^k \bullet P[k-1] \bullet c$.

Шаг 4. Результат: целые числа x и y : $ax - by = c$.

2.2.2 Вычисление обратных элементов в кольце вычетов $\mathbf{Z}_m$

Для того, чтобы обратный элемент для a в кольце вычетов $\mathbf{Z}_m$ существовал, необходимо чтобы НОД(a, m) = 1.

Рассмотрим вспомогательное уравнение (относительно неизвестных x и y): $ax - my \equiv c \pmod{m}$. Необходимо взять остаток по модулю m от обеих частей уравнения, тогда получится $ax \equiv c \pmod{m}$. Элемент x равняется a^{-1} в том случае, если $ax \pmod{m} = 1$, следовательно, $c = 1$. Таким образом, для нахождения обратного элемента необходимо линейное диофантово уравнение $ax - by = c$ при $b = \mathrm{m}$ и $c = 1$.

Псевдокод вычисления обратных элементов в кольце вычетов $\mathbf{Z}_m$

Вход: Целые числа a, m : НОД(a, m) = 1.

Выход: Целое число a^{-1} : $a \bullet a^{-1} = 1 \pmod{m}$.

Шаг 1. Найти разложение *fractions* чисел $\frac{a}{m}$ в цепную дробь.

Шаг 2. Найти знаменатель Q подходящих дробей.

Шаг 3. Вычислить $x = (-1)^k \bullet Q[k-1] \pmod{m}$.

Шаг 4. Результат: x : $a \bullet x \pmod{m} = 1$.

2.2.3 Решение линейных сравнений $ax \equiv b \pmod{m}$

Аналогично предыдущему пункту, рассматривается вспомогательное уравнение (относительно неизвестных x и y): $ax - my = b \pmod{m}$, которое является линейным диофантовым уравнением, и имеет решение при условии, что b делится на НОД(a, m), которое можно найти с помощью разложения цепной дроби. Таким образом, для получения решения сравнения необходимо решить диофантово уравнение с помощью разложения цепной дроби.

Псевдокод решения линейных сравнений $ax \equiv b \pmod{m}$

Вход. Целые числа a, b, m : НОД(a, m) = 1.

Выход: Целое число x : $ax \equiv b \pmod{m}$.

Шаг 1. Найти разложение *fractions* чисел $\frac{a}{m}$ в цепную дробь.

Шаг 2. Найти знаменатель Q подходящих дробей.

Шаг 3. Вычислить $x = (-1)^k \bullet Q[k-1] \bullet b \pmod{m}$.

Шаг 4. Результат: x : $ax \equiv b \pmod{m}$.

2.3 Вычисления символов Лежандра и Якоби

Пусть $p > 2$ – простое число.

Определение. Число $a \in \boldsymbol{Z}_p$ называется *квадратичным вычетом по модулю p*, если

$$(\exists x \in \boldsymbol{Z})\ \ x^2 \equiv a \pmod{p}.$$

В противном случае число a называется *квадратичным невычетом по модулю p*.

Определение. Для нечётного просто числа p *символом Лежандра* числа $a \in \boldsymbol{Z}$ называется выражение

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

Свойства символа Лежандра:

1. $a \equiv b\ (mod\ p)\ \Rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$;

2. $\left(\frac{ac^2}{p}\right) = \left(\frac{a}{p}\right)$ для любого $c \in \boldsymbol{Z}$;

3. *Критерий Эйлера* $\left(\frac{a}{p}\right) = a^{\frac{p-1}{2}}\ (mod\ p)$ для НОД$(a, p) = 1$;

4. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$;

5. $\left(\frac{1}{p}\right) = 1, \left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = \begin{cases} 1, \text{если } p \equiv 1\ (mod\ 4) \\ -1, \ \text{если } p \equiv 3\ (mod\ 4) \end{cases}$;

6. $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = \begin{cases} 1, \text{если } p \equiv \pm 1\ (mod\ 8) \\ -1, \text{если } p \equiv \pm 3\ (mod\ 8) \end{cases}$;

7. *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.

Определение. Пусть натуральное число $n = p_1^{\alpha_1}..p_k^{\alpha_k}$. *Символом Якоби* числа $a \in \mathbf{Z}$ называется выражение

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \cdot \ldots \cdot \left(\frac{a}{p_k}\right)^{\alpha_k}.$$

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).

Псевдокод вычисления символа Якоби $\left(\frac{a}{p}\right)$, который для простого p совпадает с символом Лежандра.

Вход: Целые числа a, p.

Выход: Символы Якоби jac.

Шаг 1. Инициализируем $jac = 1$.

Шаг 2. Пока $a \neq 1$:

Шаг 2.1. Вычислить $a = a$ (mod p). Если $|a| > \frac{p}{2}$, то $a = |a - p|$ и $jac = jac \cdot -1^{\frac{p-1}{2}}$.

Шаг 2.2. Если a (mod 2) = 0, то представить $a = 2^t \cdot a_1$.

Шаг 2.3. Если t (mod 2) = 1, то $jac = jac \cdot -1^{\frac{p^2-1}{8}}$.

Шаг 2.4. Swap(a, p).

Шаг 2.5. Вычислить $jac = jac \cdot -1^{\frac{p-1}{2} \cdot \frac{a-1}{2}}$.

Шаг 3. Если p – простое число, то результат: символ Якоби и символ Лежандра = jac. В противном случае результат: символ Якоби = jac.

2.4 Извлечение квадратного корня в кольце вычетов

Решение сравнения $x^2 \equiv a(mod\ p)$:

1) если $p \equiv 3(mod\ 4)$, то $p = 4m + 3$ и $x = \pm a^{m+1}, a \in QR_p \Longrightarrow (a)^{\frac{p-1}{2}} \equiv 1 \equiv a^{2m+1}(mod\ p)$, $x = a^{m+1}$ 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 $x = \pm a^{m+1}2^{2m+1}$, $a \in QR_p \Longrightarrow (a)^{\frac{p-1}{2}} \equiv 1 \equiv a^{4m+1} = (a^{2m+1})^2(mod\ p) \Longrightarrow a^{2m+1} \equiv 1(mod\ p)$ или $a^{2m+1} \equiv -1(mod\ p)$;

3) в общем случае применяются специальные полиномиальные вероятностные алгоритмы:

3.1) Вероятностный алгоритм Чипполы извлечения квадратного корня в поле $\boldsymbol{Z}_p$ с полиномиальной арифметикой. Вероятность успеха $P_a \geq \frac{1}{2} - \frac{1}{2p}$;

3.2) Вероятностный алгоритм извлечения квадратного корня в поле $\boldsymbol{Z}_p$ с арифметикой только этого поля. Вероятность успеха $P_a = \frac{1}{2}$.

Псевдокод вероятностного алгоритма Чипполы извлечения квадратного корня в поле $\boldsymbol{Z}_p$ с полиномиальной арифметикой

Вход: Целые числа a, p : p – простое число и $(\frac{a}{p}) = 1$.

Выход: Целое число x : x^2 (mod p) = a.

Шаг 1. Сгенерировать b : $(\frac{b^2-4a}{p}) = -1$.

Шаг 2. Положить полином $p_1 = y^{\frac{p+1}{2}}$ и полином $p_2 = y^2 - by + a$.

Шаг 3. Выполнить деление полиномов p_1 на p_2.

Шаг 4. Положить $x = p_1[0]$.

Шаг 5. Результат: $x : x^2 \pmod p = a$.

Псевдокод вероятностного алгоритма извлечения квадратного корня в поле $\boldsymbol{Z}_p$ с арифметикой только этого поля

Вход: Целые числа a, p : p – простое число и $(\frac{a}{p}) = 1$.

Выход: Целое число $x : x^2 \pmod p = a$.

Шаг 1. Представить $p - 1 = 2^m \cdot q$.

Шаг 2. Сгенерировать $b : (\frac{b}{p}) = -1$.

Шаг 3. Инициализировать массив $kArr$.

Шаг 4. Цикл по i от 1 до ∞:

Шаг 4.1. Положить $k = 0$.

Шаг 4.2. Пока $a^{2^k q} \pmod p \neq 1$, увеличивать k на 1.

Шаг 4.3. Добавить k в массив $kArr$.

Шаг 4.4. Если $k = 0$, перейти к шагу 5.

Шаг 4.5. Вычислить $a = ab^{2^{m-kArr[i]-1}} \pmod p$

Шаг 5. Положить $r = a^{\frac{q+1}{2}} \pmod p$.

Шаг 6. Цикл по i от $kArr$.size() – 2 до 0:

Шаг 6.1. Вычислить $r = (b^{2^{m-kArr[i]-1}})^{-1} \pmod p$.

Шаг 7. Результат: $x = r : x^2 \pmod p = a$.

3 Результаты работы

3.1 Оценки сложности рассмотренных алгоритмов

Разложение чисел в цепную дробь – $O(n^2)$, где n – битовая длина наибольшего числа из a и b;

Решение линейных диофантовых уравнений – $O(n^2)$, где n – битовая длина наибольшего числа из a и b;

Вычисление обратных элементов в кольце вычетов $\mathbf{Z}_m$ – $O(n^2)$, где n – битовая длина наибольшего числа из a и b;

Решение линейных сравнений $ax \equiv b \pmod{m}$ – $O(n^2)$, где n – битовая длина наибольшего числа из a и b;

Вычисления символов Лежандра и Якоби – $O(\log^2 n)$;

Извлечение квадратного корня в кольце вычетов с полиномиальной арифметикой – $O(\log^3 p)$;

Извлечение квадратного корня в кольце вычетов с арифметикой только этого поля – $O(\log^4 p)$.

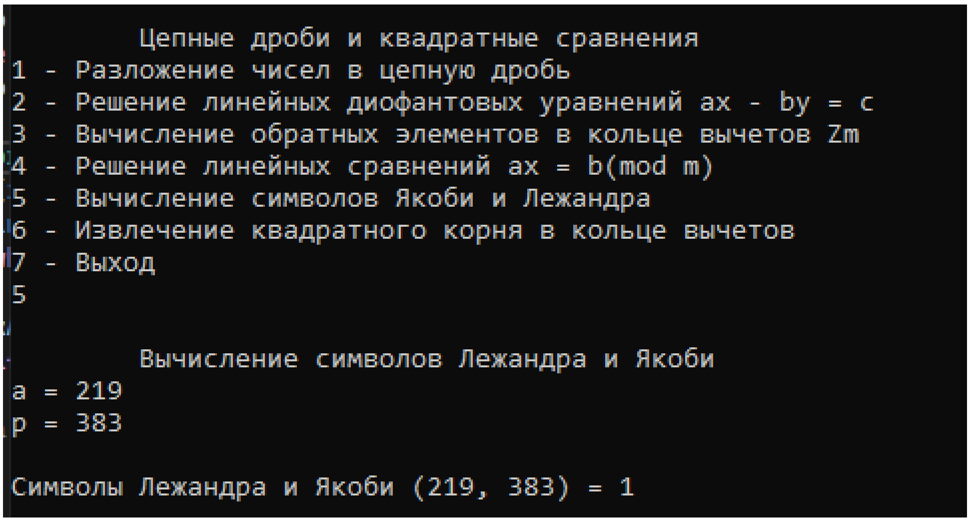
3.2 Результаты тестирования программ

```
        Цепные дроби и квадратные сравнения
1 - Разложение чисел в цепную дробь
2 - Решение линейных диофантовых уравнений ax - by = c
3 - Вычисление обратных элементов в кольце вычетов Zm
4 - Решение линейных сравнений ax = b(mod m)
5 - Вычисление символов Якоби и Лежандра
6 - Извлечение квадратного корня в кольце вычетов
7 - Выход
1

        Разложение чисел в цепную дробь
a0 = 18
a1 = 5

a0 / a1 = (3; 1, 1, 2)
Числитель подходящих дробей: 0, 1, 3, 4, 7, 18
Знаменатель подходящих дробей: 1, 0, 1, 1, 2, 5
```

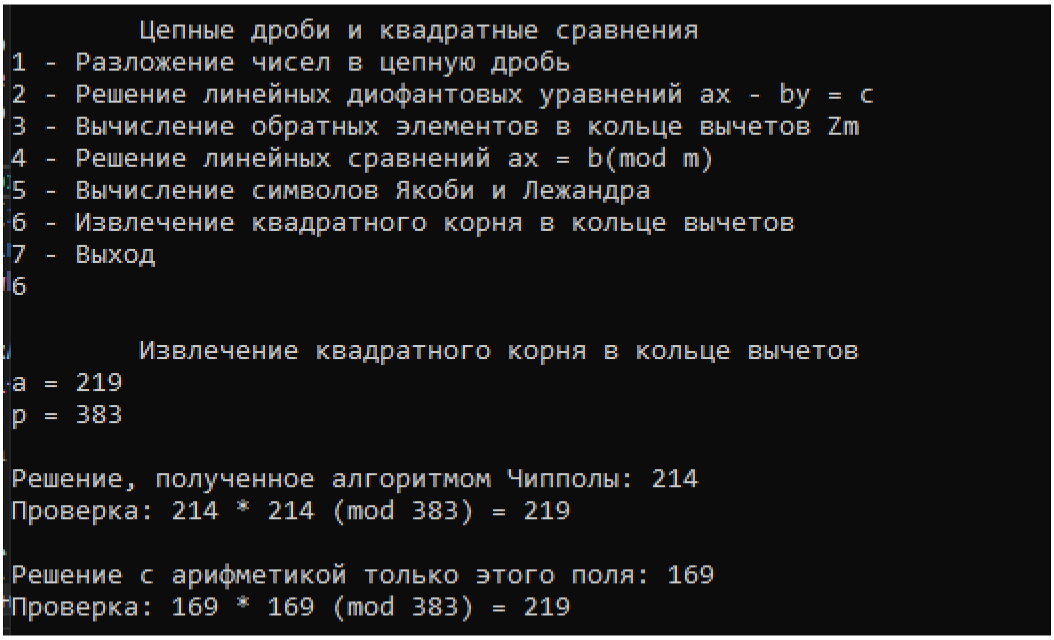
```
        Цепные дроби и квадратные сравнения
1 - Разложение чисел в цепную дробь
2 - Решение линейных диофантовых уравнений ax - by = c
3 - Вычисление обратных элементов в кольце вычетов Zm
4 - Решение линейных сравнений ax = b(mod m)
5 - Вычисление символов Якоби и Лежандра
6 - Извлечение квадратного корня в кольце вычетов
7 - Выход
2

        Решение линейных диофантовых уравнений ax - by = c
a = 5
b = 6
c = 3

5x - 6y = 3
x = -3
y = -3
```

```
        Цепные дроби и квадратные сравнения
1 - Разложение чисел в цепную дробь
2 - Решение линейных диофантовых уравнений ax - by = c
3 - Вычисление обратных элементов в кольце вычетов Zm
4 - Решение линейных сравнений ax = b(mod m)
5 - Вычисление символов Якоби и Лежандра
6 - Извлечение квадратного корня в кольце вычетов
7 - Выход
3

        Вычисление обратных элементов в кольце вычетов Zm
a = 5
m = 6

Обратный элемент для a: 5
```

```
        Цепные дроби и квадратные сравнения
1 - Разложение чисел в цепную дробь
2 - Решение линейных диофантовых уравнений ax - by = c
3 - Вычисление обратных элементов в кольце вычетов Zm
4 - Решение линейных сравнений ax = b(mod m)
5 - Вычисление символов Якоби и Лежандра
6 - Извлечение квадратного корня в кольце вычетов
7 - Выход
4

        Решение линейных сравнений ax = b(mod m)
a = 5
b = 2
m = 3

5x = 2(mod 3)
x = 1
```



```
        Цепные дроби и квадратные сравнения
1 - Разложение чисел в цепную дробь
2 - Решение линейных диофантовых уравнений ax - by = c
3 - Вычисление обратных элементов в кольце вычетов Zm
4 - Решение линейных сравнений ax = b(mod m)
5 - Вычисление символов Якоби и Лежандра
6 - Извлечение квадратного корня в кольце вычетов
7 - Выход
5

        Вычисление символов Лежандра и Якоби
a = 219
p = 383

Символы Лежандра и Якоби (219, 383) = 1
```



```
        Цепные дроби и квадратные сравнения
1 - Разложение чисел в цепную дробь
2 - Решение линейных диофантовых уравнений ax - by = c
3 - Вычисление обратных элементов в кольце вычетов Zm
4 - Решение линейных сравнений ax = b(mod m)
5 - Вычисление символов Якоби и Лежандра
6 - Извлечение квадратного корня в кольце вычетов
7 - Выход
6

        Извлечение квадратного корня в кольце вычетов
a = 219
p = 383

Решение, полученное алгоритмом Чиполы: 214
Проверка: 214 * 214 (mod 383) = 219

Решение с арифметикой только этого поля: 169
Проверка: 169 * 169 (mod 383) = 219
```

3.3 Код программы

```
#include "iostream"
#include "vector"
#include "algorithm"
#include "cmath"

using namespace std;
int myPow(int x, int y) {
	int res = 1;
	for (int i = 0; i < y; i++)
		res *= x;
	return res;
}


//Обычный алгоритм Евклида
int usualEuclid(int a, int b) {
	if (a < b)
		swap(a, b);
	if (a < 0 || b < 0) {
```

```
            cout << "Алгоритм Евклида не может найти НОД(" << a << ", " << b
<< ")!";
            throw string{ "Выполнение невозможно: a < 0 или b < 0" };
        }
        else if (b == 0)
            return a;

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


//Расширенный алгоритм Евклида
pair <int, int> advancedEuclid(int a, int b) {
        if (a < 0 || b < 0) {
            cout << "Алгоритм Евклида не может найти НОД(" << a << ", " << b
<< ")!";
            throw string{ "Выполнение невозможно: a < 0 или b < 0" };

        }

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

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

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

        return make_pair(xPrev, yPrev);
}


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


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

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


int powClosed(int x, int y, int mod) {//Возводим число в степени по модулю
        if (y == 0)
                return 0;

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

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


bool miller_rabin(int n, int k = 10) {
        if (n == 0 || n == 1)
                return false;

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

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

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

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

```
        return true;
}


/////////////////////////////////////////////////////////////////РАЗЛОЖЕНИЕ ЧИСЕЛ
В ЦЕПНУЮ ДРОБЬ///////////////////////////////////////////////////////////////
//Разложение в цепную дробь
void continuousFraction(int a0, int a1, vector <int>& fractions) {
        int q = a0 / a1, r = a0 % a1;
        fractions.push_back(q);
        if (r == 0)
                return;
        else if (r == 1)
                fractions.push_back(a1);
        else
                continuousFraction(a1, r, fractions);
}


void mainFraction() {
        cout << "\n\tРазложение чисел в цепную дробь";
        int a0, a1;
        cout << "\na0 = ";
        cin >> a0;
        cout << "a1 = ";
        cin >> a1;

        if (usualEuclid(a0, a1) != 1) {
                cout << "\nНОД(" << a0 << ", " << a1 << ") != 1";
                return;
        }

        vector <int> fractions;
        continuousFraction(a0, a1, fractions);
        vector <int> P{ 0, 1 };
        vector <int> Q{ 1, 0 };
        for (int i = 0; i < fractions.size(); i++) {
                P.push_back(fractions[i] * P[i + 1] + P[i]);
                Q.push_back(fractions[i] * Q[i + 1] + Q[i]);
        }

        cout << "\na0 / a1 = (" << fractions[0] << "; ";
        for (int i = 1; i < fractions.size(); i++) {
                if (i == fractions.size() - 1)
                        cout << fractions[i] << ")";
                else
                        cout << fractions[i] << ", ";
        }
        cout << "\nЧислитель подходящих дробей: " << P[0];
        for (int i = 1; i < P.size(); i++)
                cout << ", " << P[i];
        cout << "\nЗнаменатель подходящих дробей: " << Q[0];
        for (int i = 1; i < Q.size(); i++)
                cout << ", " << Q[i];
}


/////////////////////////////////////////////////////////////////////ПРИЛОЖЕНИЯ
ЦЕПНЫХ ДРОБЕЙ//////////////////////////////////////////////////////////////////
//Решение линейных диофантовых уравнений
void solDiophant() {
        cout << "\n\tРешение линейных диофантовых уравнений ax - by = c";
```

```
	int a, b, c;
	cout << "\na = ";
	cin >> a;
	cout << "b = ";
	cin >> b;
	cout << "c = ";
	cin >> c;

	if (usualEuclid(a, b) != 1) {
		cout << "\nЧисла " << a << " и " << b << " не являются 
взаимнопростыми!";
		return;
	}

	vector <int> fractions;
	continuousFraction(a, b, fractions);

	vector <int> P{ 0, 1 };
	vector <int> Q{ 1, 0 };
	for (int i = 0; i < fractions.size() - 1; i++) {
		P.push_back(fractions[i] * P[i + 1] + P[i]);
		Q.push_back(fractions[i] * Q[i + 1] + Q[i]);
	}

	cout << "\n" << a << "x - " << b << "y = " << c;
	cout << "\nx = " << myPow(-1, fractions.size()) * Q.back() * c;
	cout << "\ny = " << myPow(-1, fractions.size()) * P.back() * c;
}


//Вычисление обратных элементов в кольце вычетов Zm
int revEl(int a, int m) {
	if (usualEuclid(a, m) != 1)
		return -1;

	vector <int> fractions;
	continuousFraction(a, m, fractions);
	vector <int> Q{ 1, 0 };
	for (int i = 0; i < fractions.size() - 1; i++)
		Q.push_back(fractions[i] * Q[i + 1] + Q[i]);

	int res = myPow(-1, fractions.size()) * Q.back() % m;
	while (res < 0)
		res += m;

	return res;
}


void mainRevEl() {
	cout << "\n\tВычисление обратных элементов в кольце вычетов Zm";
	int a, m;
	cout << "\na = ";
	cin >> a;
	cout << "m = ";
	cin >> m;

	int revA = revEl(a, m);
	if (revA == -1)
		cout << "\nЧисла " << a << " и " << m << " не являются 
взаимнопростыми!";
	else
		cout << "\nОбратный элемент для a: " << revA;
```

```
}


//Решение линейных сравнений ax = b(mod m)
void solCompr() {
	cout << "\n\tРешение линейных сравнений ax = b(mod m)";
	int a, b, m;
	cout << "\na = ";
	cin >> a;
	cout << "b = ";
	cin >> b;
	cout << "m = ";
	cin >> m;

	if (usualEuclid(a, m) != 1) {
		cout << "\nЧисла " << a << " и " << m << " не являются
взаимнопростыми!";
		return;
	}

	vector <int> fractions;
	continuousFraction(a, m, fractions);
	vector <int> Q{ 1, 0 };
	for (int i = 0; i < fractions.size() - 1; i++)
		Q.push_back(fractions[i] * Q[i + 1] + Q[i]);

	int x = myPow(-1, fractions.size()) * Q.back() * b % m;
	while (x < 0)
		x += m;

	cout << endl << a << "x = " << b << "(mod " << m << ")" << "\nx = " <<
x;
}


/////////////////////////////////////////////////////////////////ВЫЧИСЛЕНИЕ
СИМВОЛА ЯКОБИ И
ЛЕЖАНДРА/////////////////////////////////////////////////////////////////
//Вычисление символа Якоби и Лежандра
int symbolJacobi(int a, int p) {
	int res = 1;
	while (a != 1) {
		a = a % p;
		if (abs(a) >= p / 2) {
			a = abs(a - p);
			res *= myPow(-1, (p - 1) / 2);
		}

		if (a % 2 == 0) {
			int t = 0;
			while (a % 2 == 0) {
				t++;
				a /= 2;
				if (a == 0)
					return 0;
			}
			if (t % 2 == 1)
				res *= myPow(-1, (p * p - 1) / 8);
		}

		swap(a, p);
		res *= myPow(-1, ((p - 1) / 2) * ((a - 1) / 2));
	}
```

```
        return res;
}


void mainSymbolJacobi() {
        cout << "\n\tВычисление символов Лежандра и Якоби";
        int a, p;
        cout << "\na = ";
        cin >> a;
        cout << "p = ";
        cin >> p;

        int jac = symbolJacobi(a, p);
        if (!miller_rabin(p))
                cout << "\nСимвол Лежандра: не может быть вычислен, т.к. p - не
простое \nСимвол Якоби: " << jac;
        else
                cout << "\nСимволы Лежандра и Якоби (" << a << ", " << p << ") =
" << jac;
}


////////////////////////////////////////////////////////ИЗВЛЕЧЕНИЕ КВАДРАТНОГО
КОРНЯ В КОЛЬЦЕ
ВЫЧЕТОВ/////////////////////////////////////////////////////////////////
//С помощью полиномиальной арифметики
int pdf(int a, int p) {
        int b = rand() % p;
        while (symbolJacobi(b * b - 4 * a, p) != -1)
                b = (b + 1) % p;

        int degP1 = (p + 1) / 2, degP2 = 2;
        int* p1 = new int[degP1 + 1];
        p1[degP1] = 1;
        for (int i = 0; i < degP1; i++)
                p1[i] = 0;
        int* p2 = new int[degP2 + 1];
        p2[0] = a, p2[1] = -b, p2[2] = 1;

        vector <int> q(degP1 - degP2 + 1);
        for (int k = degP1 - degP2; k >= 0; k--) {
                q[k] = (p1[degP2 + k] * revEl(p2[degP2], p)) % p;
                for (int j = degP2 + k - 1; j >= k; j--)
                        p1[j] -= q[k] * p2[j - k];
        }

        int res = p1[0];
        while (res < 0)
                res += p;
        return res % p;
}


//С арифметикой только этого поля
int sqrtFromZp(int a, int p) {
        int m = 0, q = p - 1;
        while (q % 2 != 1) {
                m++;
                q /= 2;
        }

        int b = rand() % p;
        while (symbolJacobi(b, p) != -1)
```

```
                b = (b + 1) % p;

        vector <int> kArr;
        for (int i = 1;; i++) {
                int k = 0;
                while (powClosed(a, myPow(2, k) * q, p) != 1)
                        k++;
                kArr.push_back(k);
                if (k == 0)
                        break;
                a = (a * myPow(b, myPow(2, m - kArr.back()))) % p;
        }

        int r = powClosed(a, (q + 1) / 2, p);
        for (int i = kArr.size() - 2; i >= 0; i--)
                r = (r * advancedEuclid(myPow(b, myPow(2, m - kArr[i] - 1)),
p).first) % p;

        return r;
}


void mainSqrtFromZp() {
        cout << "\n\tИзвлечение квадратного корня в кольце вычетов";
        int a, p;
        cout << "\na = ";
        cin >> a;
        cout << "p = ";
        cin >> p;

        if (!miller_rabin(p)) {
                cout << "\nЧисло p должно быть простым!";
                return;
        }
        if (symbolJacobi(a, p) != 1) {
                cout << "Символ Лежандра (" << a << ", " << p << ") != 1";
                return;
        }

        int resChip = pdf(a, p);
        while (resChip * resChip % p != a)
                resChip = pdf(a, p);
        cout << "\nРешение, полученное алгоритмом Чипполы: " << resChip;
        cout << "\nПроверка: " << resChip << " * " << resChip << " (mod " << p
<< ") = " << resChip * resChip % p;

        int res = sqrtFromZp(a, p);
        while (res * res % p != a)
                res = sqrtFromZp(a, p);
        cout << "\n\nРешение с арифметикой только этого поля: " << res;
        cout << "\nПроверка: " << res << " * " << res << " (mod " << p << ") = "
<< res * res % p;
}


int main() {
        srand(time(0));
        setlocale(LC_ALL, "ru");
        for (;;) {
                cout << "\tЦепные дроби и квадратные сравнения";
                cout << "\n1 - Разложение чисел в цепную дробь \n2 - Решение
линейных диофантовых уравнений ax - by = c";
```

```
			cout << "\n3 - Вычисление обратных элементов в кольце вычетов Zm 
\n4 - Решение линейных сравнений ax = b(mod m)";
			cout << "\n5 - Вычисление символов Якоби и Лежандра \n6 - 
Извлечение квадратного корня в кольце вычетов \n7 - Выход \n";
			int x;
			cin >> x;
			switch (x) {
			case 1:
				mainFraction();
				cout << "\n\n";
				break;
			case 2:
				solDiophant();
				cout << "\n\n";
				break;
			case 3:
				mainRevEl();
				cout << "\n\n";
				break;
			case 4:
				solCompr();
				cout << "\n\n";
				break;
			case 5:
				mainSymbolJacobi();
				cout << "\n\n";
				break;
			case 6:
				mainSqrtFromZp();
				cout << "\n\n";
				break;
			case 7:
				return 0;
			default:
				cout << "Incorrect. Try again \n\n";
			}
		}
}
```

ЗАКЛЮЧЕНИЕ

В ходе данной работы был разобран алгоритм разложения чисел в цепную дробь и его приложения: разложение чисел в цепную дробь используется при решении диофантовых уравнений, поиске обратных элементов в кольце вычетов $\mathbf{Z}_m$ и при решении линейных сравнений. Был разобраны символы Якоби и Лежандра и написан алгоритм для их нахождения. Также были реализованы алгоритмы извлечения квадратного корня в кольце вычетов с помощью полиномиальной арифметики и арифметики только данного поля.