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

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 свойств цепных дробей и квадратных сравнений.

Порядок выполнения работы:

1. Разобрать алгоритм разложения чисел в цепную дробь и привести его 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 Теоретические сведения по рассмотренным темам

Цепные дроби и подходящие дроби

С помощью алгоритма Евклида любое рациональное число можно представить в виде специального выражения, которое называется цепной дробью.

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 НОД по алгоритму Евклида имеет вид:

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

Эти равенства можно переписать в виде:

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

Тогда рациональное число r можно представить следующим образом:

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

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

Опр. Выражение такого вида называется *цепной дробью* с неполными частными $q_1, q_2, \dots, q_k$ и обозначается как $(q_1; q_2, \dots, q_k)$.

Опр. Для цепной дроби $\frac{a_0}{a_1} = (q_1; q_2, \dots, q_k)$ выражения $\delta_1 = q_1, \delta_2 = q_1 + \frac{1}{q_2}, \dots, \delta_k = q_1 + \frac{1}{q_2 + \frac{1}{\dots + \frac{1}{q_{k-1} + \frac{1}{q_k}}}}$ называются *подходящими дробями* конечной цепной дроби $(q_1; q_2, \dots, q_k)$ и обозначаются символами $\delta_i = (q_1; q_2, \dots, q_i)$.

Числитель P_i и знаменатель Q_i подходящих дробей δ_i вычисляются по формулам: $P_i = q_i P_{i-1} + P_{i-2}, Q_i = q_i Q_{i-1} + Q_{i-2}$, где $P_{-1} = 0, P_0 = 1, Q_{-1} = 1, Q_0 = 0$.

Приложения цепных дробей $\frac{a}{m}$

1. Решение линейных диофантовых уравнений а$x + by = c$;

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

3. Решение линейных сравнений $ax \equiv b \ (mod\ m)$.

Решение линейных диофантовых уравнений а$\boldsymbol{x} + \boldsymbol{by} = \boldsymbol{c}$

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

Рассмотрим простейшие диофантовы уравнения вида $ax - by = c$, где $a, b, c \in \boldsymbol{Z}$, $a \neq 0$, $b > 0$. Требуется найти все целые значения x, y, удовлетворяющие этому уравнению.

Данное уравнение разрешимо в том и только в том случае, когда $(a, b)|c$.

Сначала решим уравнение вида $ax - by = 1$, $(a, b) = 1$. Пусть $\frac{P_k}{Q_k}$ – последняя подходящая дробь для числа $\frac{a}{b}$. Тогда $a = P_k$, $b = Q_k$. Значит выполняется равенство $a(-1)^{k-1}Q_{k-1} - b(-1)^{k-1}P_{k-1} = 1$.

Следовательно, $x = (-1)^{k-1}Q_{k-1}$, $y = (-1)^{k-1}P_{k-1}$ являются решениями диофантового уравнения $ax - by = 1$. Множество всех решений этого уравне6ния описывается формулами $x = (-1)^{k-1}Q_{k-1} + bt, y = (-1)^{k-1}P_{k-1} + at, t \in \boldsymbol{Z}$. Пусть теперь для описанного выше уравнения выполняется $d = (a, b)$ и $d|c$. Тогда уравнение $a_1x - b_1y = c_1$, где $a_1 = \frac{a}{d}, b_1 = \frac{b}{d}, c_1 = \frac{c}{d}$ равносильно исходному уравнению. С учетом предыдущего случая (когда $c_1 = 1$) выписываем все решения уравнения $a_1x - b_1y = c_1$: $x = (-1)^{k-1}Q_{k-1}c_1 + b_1t, y = (-1)^{k-1}P_{k-1}c_1 + a_1t, t \in \boldsymbol{Z}$.

Вычисление обратных элементов в кольце вычетов $\boldsymbol{Z_m}$

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

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

Решение линейных сравнений $\boldsymbol{ax \equiv b\ (mod\ m)}$

Аналогично предыдущему пункту, рассматривается вспомогательное уравнение (относительно неизвестных x и y): $ax - my \equiv b\ (mod\ m)$, 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.

Вычисление символов Лежандра и Якоби

Пусть $p > 2$ – простое число.

<u>Опр.</u> Число $a \in \boldsymbol{Z}_p$ называется *квадратичным вычетом по модулю* p, если

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

<u>Опр.</u> В противном случае число a называется *квадратичным невычетом по модулю* p.

QR_p – множество всех квадратичных вычетов по модулю p,

QNR_p – множество всех квадратичных невычетов по модулю p.

<u>Распознавание квадратичных вычетов</u>:

<u>Опр.</u> Для нечетного простого числа p *символом Лежандра* числа $a \in \boldsymbol{Z}$ называется выражение:

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

Свойства символа Лежандра:

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

2. $\left(\frac{ac^2}{p}\right) = (\frac{a}{p})$ 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. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)(\frac{b}{p})$;

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.

Опр. Пусть натуральное число $n = p_1^{\alpha_1} .. p_k^{\alpha_k}$.

Символом Якоби числа $a \in \boldsymbol{Z}$ называется выражение

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

Символ Якоби для простого числа n совпадает с символом Лежандра и удовлетворяет почти всем свойствам символа Лежандра. Символ Якоби позволяет упростить вычисление символа Лежандра $(\frac{a}{p})$ (без разложения числа a на множители).

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 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 Z_p с арифметикой только этого поля. Вероятность успеха $P_a = \frac{1}{2}$.

3 Практическая реализация

3.1 Описание и оценка сложности алгоритмов

Алгоритм разложения числа в цепную дробь

Вход: Целые числа $a > 0$ и $b > 0$.

Выход: Цепная дробь $\frac{a}{b} = (q_0; q_1, \dots q_k)$, числители P_i и знаменатели Q_i подходящих дробей.

Шаг 1. Инициализация значений: $P_{-1} = 0, P_0 = 1$, $Q_{-1} = 1, Q_0 = 0$, $i = 1$;

Шаг 2. Вычислить q_i как целую часть от деления $\frac{a}{b}$;

Шаг 3. Вычислить $P_i = q_i P_{i-1} + P_{i-2}$;

Шаг 4. Вычислить $Q_i = q_i Q_{i-1} + Q_{i-2}$;

Шаг 5. Пересчитать a и b следующим образом: $a = b, b = a \ (mod\ b)$:

Шаг 6. Если $b = 0$, то выдать $q = (q_0; q_1, \dots q_k), P = (P_{-1}, P_0, \dots P_k), Q = (Q_{-1}, Q_0, \dots Q_k)$. Иначе положить $i = i + 1$ и перейти к шагу 2.

Временная сложность алгоритма $O(n^2)$, где n – битовая длина наибольшего из чисел a и b.

Алгоритм решения диофантовых уравнений

Вход: Целые числа $a > 0$, $b > 0$, $c > 0$.

Выход: Целые числа x, y, которые являются решением уравнения.

Шаг 1. Вычислить $P = (P_{-1}, P_0, \dots P_k)$ и $Q = (Q_{-1}, Q_0, \dots Q_k)$ с помощью алгоритма разложения числа $\left(\frac{a}{b}\right)$ в цепную дробь;

Шаг 2. Вычислить $d = \text{НОД}(a, b)$;

Шаг 3. Если c не кратно d, выдать «Нет решений», иначе вычислить $x = (-1)^{k-1} Q_{k-1} \frac{c}{d}, y = (-1)^{k-1} P_{k-1} \frac{c}{d}$;

Шаг 4. Выдать x и y.

Временная сложность алгоритма $O(n^2)$, где n – битовая длина наибольшего из чисел a и b.

Алгоритм вычисления обратного элемента в кольце $\boldsymbol{Z_m}$:

Вход: Целые числа $a > 0, m > 0$ такие, что $\text{НОД}(a, m) = 1$.

Выход: Обратный элемент $a^{-1} \in \boldsymbol{Z}_m$.

Шаг 1. Вычислить $Q = (Q_{-1}, Q_0, \dots Q_k)$ с помощью алгоритма разложения числа $\left(\frac{a}{m}\right)$ в цепную дробь;

Шаг 2. Вычислить $x = (-1)^{k-1} Q_{k-1} (\text{mod}\, m)$;

Шаг 3. Выдать $a^{-1} = x$.

Временная сложность алгоритма $O(n^2)$, где n – битовая длина наибольшего из чисел a и b.

Алгоритм решения линейных сравнений $\boldsymbol{ax \equiv b\ (mod\ m)}$:

Вход: Целые числа $a > 0, b > 0, m > 0$.

Выход: x – решение линейного сравнения.

Шаг 1. Вычислить $Q = (Q_{-1}, Q_0, \dots Q_k)$ с помощью алгоритма разложения числа $\left(\frac{a}{m}\right)$ в цепную дробь;

Шаг 2. Вычислить $d = \text{НОД}(a, m)$;

Шаг 3. Если b не делится на d, то решений нет, иначе перейти на шаг 4;

Шаг 4. Вычислить $x = (-1)^{k-1} Q_{k-1} \frac{b}{d} (\text{mod}\, m)$;

Шаг 5. Выдать x.

Временная сложность алгоритма $O(n^2)$, где n – битовая длина наибольшего из чисел a и b.

Алгоритм вычисления символа Якоби, который для простого $\boldsymbol{p}$ совпадает с символом Лежандра:

Вход: Целые числа $a \in \boldsymbol{Z}, p > 0$ – простое.

Выход: Значение символа Якоби $(\frac{a}{p})$.

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

Шаг 2. Если $b < 0$, то по свойству 4) выделяем множитель $(\frac{-1}{p})$;

Шаг 3. Если b – четное, то представляем $b = 2^t a_1$ и при нечетном t вычисляем $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$;

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

Шаг 5. При необходимости вернуться к шагу 1.

Временная сложность алгоритма $O(\log^2 n)$.

Вероятностный алгоритм Чипполы извлечения квадратного корня в $\boldsymbol{Z_p}$ с полиномиальной арифметикой.

Вход: Нечетное простое число p, $a \in \boldsymbol{Z}$, $(a, p) = 1$, $\left(\frac{a}{p}\right) = 1$.

Выход: x_0 – решение уравнения $x^2 = a$ в Z_p.

Шаг 1. Случайным образом выбрать такое $b, 0 \leq b \leq p - 1$, что $(\frac{b^2-4a}{p}) = -1$;

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

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

Временная сложность алгоритма $O(\log^3 p)$.

Вероятностный алгоритм извлечения квадратного корня в $\boldsymbol{Z_p}$ с арифметикой только этого поля.

Вход: Нечетное простое число p, $p - 1 = 2^m q$, НОД$(q, 2) = 1$, $a \in \boldsymbol{Z}$, $\left(\frac{a}{p}\right) = 1$.

Выход: x_0 – решение уравнения $x^2 \equiv a (mod\, p)$.

Шаг 1. Случайным образом выбрать такое b, что $(\frac{b}{p}) = -1$;

Шаг 2. Вычислить последовательность $a_1, \dots, a_n$ элементов поля $\boldsymbol{Z_p}$ и последовательность $k_1, \dots, k_n$ по правилу:

- $a_1 = a, a_{i+1} = a_i b^{2^{m-k_i}} \pmod p, i \geq 1$;
- k_i – наименьшее $k \geq 0$, при котором $a_i^{2^k q} \equiv 1 \pmod p$.

Выполнение шага 2 заканчивается в тот момент, когда выполняется равенство $k_n = 0$;

Шаг 3. Вычислить последовательность $r_n, \dots, r_1$ элементов поля $\boldsymbol{Z_p}$ по правилу: $r_n = a^{\frac{q+1}{2}} (\operatorname{mod} p)$, $r_i = r_{i+1} \left(b^{2^{m-k_i-1}} \right)^{-1} (\operatorname{mod} p)$, $i \geq 1$;

Шаг 4. Положить $x_0 = r_1$ – искомое решение.

Временная сложность алгоритма $O(\log^4 p)$.

3.2 Псевдокоды рассмотренных алгоритмов

Псевдокод алгоритма разложения числа в цепную дробь

Ввод a, b
Если $(a \leq 0)$ или $(b \leq 0)$
    вывести «Числа должны быть больше 0»
Вывести результат функции $ContinuedFraction(a, b)$

Функция $ContinuedFraction(a, b)$:
    $P = \{0, 1\}$
    $Q = \{1, 0\}$
    $q = \{\}$
    $i = 2$
    Пока $(b \neq 0)$
        в q добавить $(\frac{a}{b})$
        в P добавить $(q_i P_{i-1} + P_{i-2})$
        в Q добавить $(q_i Q_{i-1} + Q_{i-2})$
        $c = a$
        $a = b$
        $b = c\ (mod\ b)$
        $i = i + 1$
    Вернуть $q = (q_2; q_3, \dots q_k), P = (P_0, P_1, \dots P_k), Q = (Q_0, Q_1, \dots Q_k)$

Псевдокод алгоритма решения диофантовых уравнений

Ввод a, b, c
Если $(a \leq 0)$ или $(b \leq 0)$ или $(c \leq 0)$
    вывести «Числа должны быть больше 0»
Вывести результат функции $DiophantineEquations(a, b, c)$

Функция $DiophantineEquations(a, b, c)$:
    $res = \{\}$
    $d = \text{НОД}(a, b)$
    Если $(c\ (mod\ d) \neq 0)$
        вернуть «Нет решений»
    Вычисление $P = (P_0, P_1, \dots P_k)$, $Q = (Q_0, Q_1, \dots Q_k)$ с помощью функции $ContinuedFraction(a, b)$
    В res добавить $(-1)^{k-1} Q_{k-1} \frac{c}{d}$
    В res добавить $(-1)^{k-1} P_{k-1} \frac{c}{d}$
    Вернуть $x = res_0, y = res_1$

Псевдокод алгоритма вычисления обратного элемента в кольце Z_m

Ввод a, m
Если $(a \leq 0)$ или $(m \leq 0)$
    вывести «Числа должны быть больше 0»
Если $(\text{НОД}(a, m) \neq 1)$
    вывести «Числа должны быть взаимно простыми»
Вывести результат функции $InverseElement(a, m)$

Функция $InverseElement(a, m)$:
    Вычисление $Q = (Q_0, Q_1, \dots Q_k)$ с помощью функции $ContinuedFraction(a, m)$
    $x = (-1)^{k-1} Q_{k-1} (\text{mod}\ m)$

```
        Вернуть x
```

Псевдокод алгоритма решения линейных сравнений $ax \equiv b \ (mod\ m)$:

```
Ввод a, b, m
Если (a ≤ 0) или (b ≤ 0) или (m ≤ 0)
        вывести «Числа должны быть больше 0»
Вывести результат функции Comparison(a, b, m)

Функция Comparison(a, b, m):
        Вычисление Q = (Q_0, Q_1, ... Q_k) с помощью функции ContinuedFraction(a, m)
        d = НОД(a, m)
        Если (b (mod d) ≠ 0)
                Вернуть «Решений нет»
        x = (−1)^(k−1) Q_(k−1) b/d (mod m)
        Вернуть x
```

Псевдокод алгоритма вычисления символа Якоби

```
Ввод a, p
Если (a ≤ 0) или (p ≤ 0)
        вывести «Числа должны быть больше 0»
Вывести результат функции Jac(a, p)

Функция Jac(a, p):
        Если (НОД(a, p) ≠ 1)
                Вернуть 0
        Иначе
                r = 1
                Пока (a ≠ 0)
                        t = 1
                        Пока (a делится на 2)
                                t = t + 1
                                a = a/2
                        Если (t (mod 2) ≠ 0)
                                Если (b (mod 8) = 3) или (b (mod 8) = 5)
                                        r = r * (−1)
                        Если (a (mod 4) = 3) и (b (mod 4) = 3)
                                r = r * (−1)
                        c = a
                        Если (c ≠ 0)
                                a = b (mod c)
                        b = c
                Вернуть r
```

Псевдокод вероятностного алгоритма Чипполы

```
Ввод a, p
Если (a ≤ 0) или (p ≤ 0)
        вывести «Числа должны быть больше 0»
Если (Jac(a, p) ≠ 1)
        вывести «Символ Якоби должен равняться 1»
```

```
Вывести результат функции SquareRootChip(a,p)

Функция SquareRootChip(a,p):
	J = 5
	Пока (J ≠ −1)
		случайно выбирается число 0 ≤ b ≤ p − 1
		J = Jac(b ∗ b − 4 ∗ a, p)
	c = {0, 0, … , 1}, размерность: (p+1)/2 + 1
	d = {a, −b (mod p), 1}
	res = NODPol(c, d, p)
	Вернуть res

Функция NODPol(c,d,p):
	Полином p0 = c
	Полином p1 = d
	Пока (p1 не обратится в 0)
		q1 = p0/p1 (mod p) - деление полиномов над полем, алгоритм PDF
		p0prom = p1
		p1prom = (p0 − (p1 ∗ q1(mod p))(mod p) - остаток от деления
		p0 = p0prom
		p1 = p1prom
	Вернуть p0
```

Псевдокод вероятностного алгоритма извлечения квадратного корня в Z_p с арифметикой только этого поля

```
Ввод a,p
Если (a ≤ 0) или (p ≤ 0)
	вывести «Числа должны быть больше 0»
Если (Jac(a,p) ≠ 1)
	вывести «Символ Якоби должен равняться 1»
Вывести результат функции SquareRoot(a,p)

Функция SquareRoot(a,p):
	q = p – 1
	m = 0
	Пока (q (mod 2) = 0)
		q = q/2
		m = m + 1
	случайно выбирается число 0 ≤ b ≤ p − 1
	Пока (Jac(b,p) ≠ −1)
		случайно выбирается число 0 ≤ b ≤ p − 1
	ai = {a}
	В цикле по i от 0 до бесконечности
		deg = 1
		k = 0
		Пока (((ai_i)^(deg∗q) (mod p)) ≠ 1)
			k = k + 1
			deg = deg ∗ 2
		В ki добавить k
		Если (ki_i = 0)
			Выйти из цикла
```

В ai добавить $a_i b^{2^{m-k_i}} \pmod p$

$n =$ размерности ki

$ri_n = a_n^{\frac{q+1}{2}} \pmod p$

В цикле по i от $n-1$ до 0

$bi = b^{2^{m-k_{i-1}}} (mod\ p)$

$biobr = InverseElement(bi, p)$

$ri_i = (ri_{i+1} * biobr)\ (mod\ p)$

Вернуть ri_0

3.3 Результаты тестирования программы

Тестирование алгоритма разложения числа в цепную дробь (рисунок 1).

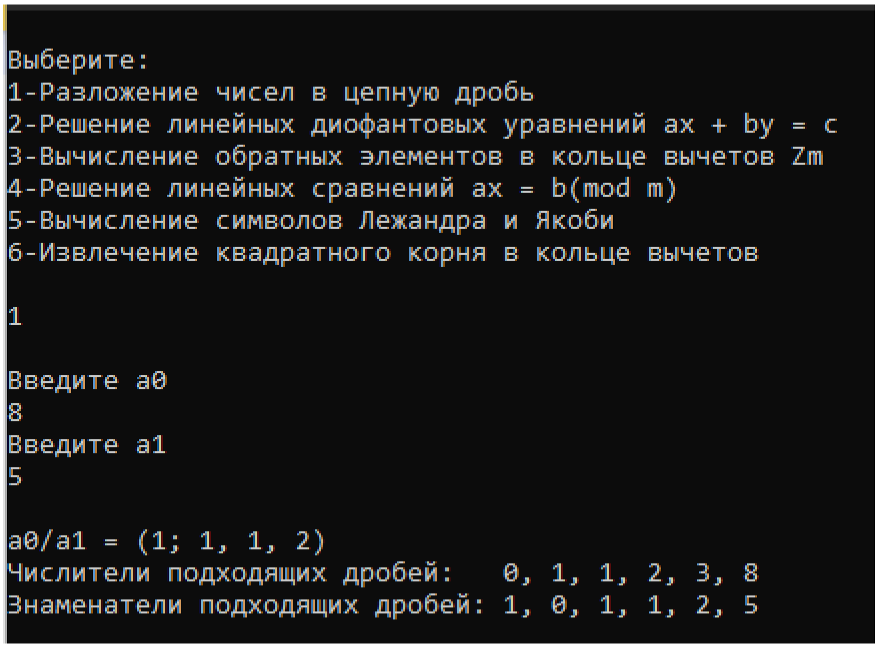
```
Выберите:
1-Разложение чисел в цепную дробь
2-Решение линейных диофантовых уравнений ax + by = c
3-Вычисление обратных элементов в кольце вычетов Zm
4-Решение линейных сравнений ax = b(mod m)
5-Вычисление символов Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

1

Введите a0
8
Введите a1
5

a0/a1 = (1; 1, 1, 2)
Числители подходящих дробей:   0, 1, 1, 2, 3, 8
Знаменатели подходящих дробей: 1, 0, 1, 1, 2, 5
```

Рисунок 1 – Результат теста алгоритма разложения числа в цепную дробь

Тестирование алгоритма решения диофантовых уравнений (рисунки 2-3).

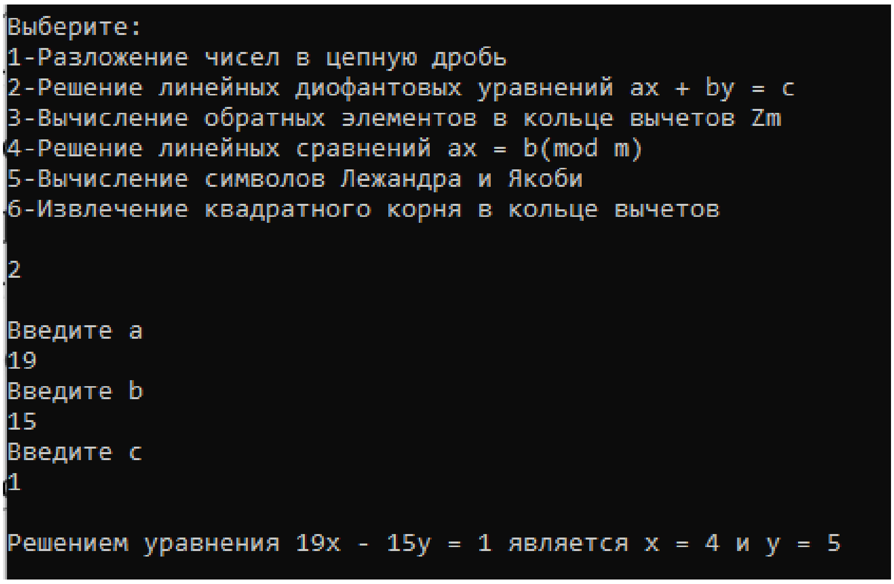
```
Выберите:
1-Разложение чисел в цепную дробь
2-Решение линейных диофантовых уравнений ax + by = c
3-Вычисление обратных элементов в кольце вычетов Zm
4-Решение линейных сравнений ax = b(mod m)
5-Вычисление символов Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

2

Введите a
19
Введите b
15
Введите c
1

Решением уравнения 19x - 15y = 1 является x = 4 и y = 5
```

Рисунок 2 – Результат теста алгоритма решения диофантовых уравнений при $\boldsymbol{c = 1}$

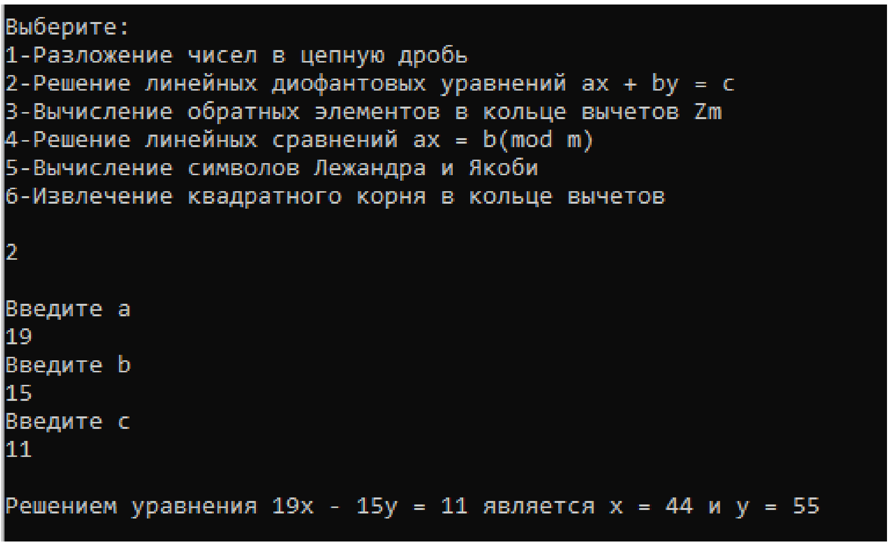

Рисунок 3 – Результат теста алгоритма решения диофантовых уравнений при $\boldsymbol{c = 11}$

Тестирование алгоритма вычисления обратного элемента в кольце Z_m(рисунки 4-5).

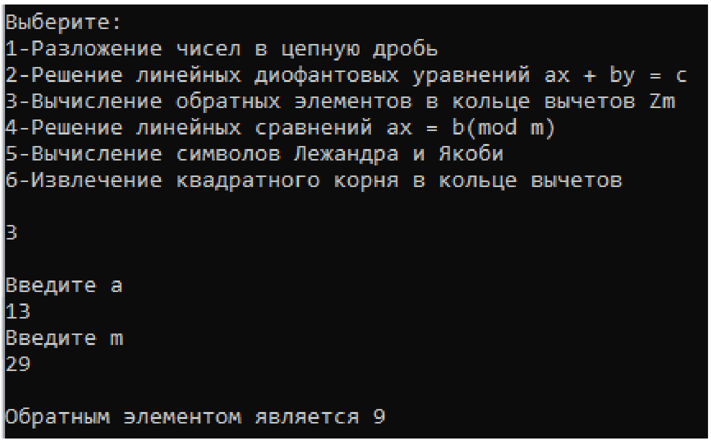

Рисунок 4 – Результат теста алгоритма вычисления обратного элемента в кольце $\boldsymbol{Z_m}$

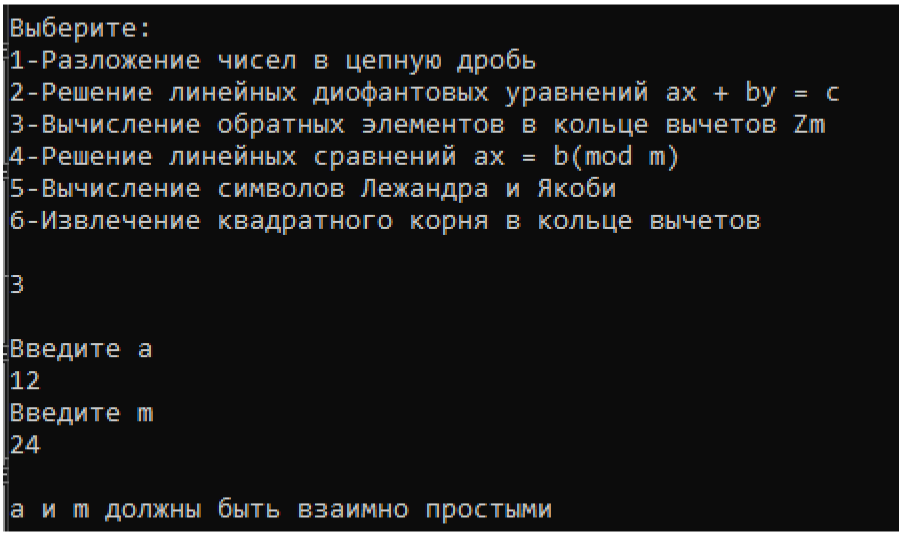

Рисунок 5 – Результат теста алгоритма вычисления обратного элемента в кольце $\boldsymbol{Z_m}$ при вводе таких a и m, что НОД$(a, m) \neq 1$

Тестирование алгоритма решения линейных сравнений $ax \equiv b \ (mod\ m)$(рисунок 6).

```
Выберите:
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 Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

4

Введите a
12
Введите b
2
Введите m
29

Решением линейного сравнения 12x = 2(mod 29) является x = 5
```

Рисунок 6 – Результат теста алгоритма решения линейных сравнений

Тестирование алгоритма вычисления символа Лежандра и Якоби (рисунок 7-9).

```
Выберите:
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 Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

5

Введите a
88
Введите p
347

Символ Якоби от (88/347) равняется -1
```

Рисунок 7 – Результат первого теста алгоритма вычисления символа Лежандра и Якоби

```
Выберите:
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 Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

5

Введите a
219
Введите p
383

Символ Якоби от (219/383) равняется 1
```

Рисунок 8 – Результат второго теста алгоритма вычисления символа Лежандра и Якоби

```
Выберите:
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 Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

5

Введите a
19
Введите p
57

Символ Якоби от (19/57) равняется 0
```

Рисунок 9 – Результат третьего теста алгоритма вычисления символа Лежандра и Якоби

Тестирование алгоритмов извлечения квадратного корня в Z_p (рисунок 10).

```
Выберите:
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 Лежандра и Якоби
6-Извлечение квадратного корня в кольце вычетов

6

Введите a
219
Введите p
383

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

Решение, полученное вторым алгоритмом: x = 169
Проверка: 169 * 169(mod 383) = 219
```

Рисунок 10 – Результат теста алгоритмов извлечения квадратного корня в $\boldsymbol{Z_p}$

4 Выводы по работе

В ходе выполнения лабораторной работы был рассмотрен алгоритм разложения чисел в цепную дробь и вычисления подходящих дробей. Кроме того, были изучены алгоритмы приложений цепных дробей – а именно, решение диофантовых уравнений, вычисление обратного элемента в кольце Z_m и решение линейных сравнений $ax \equiv b \ (mod\ m)$. Также были разобраны алгоритмы вычисления символов Лежандра и Якоби, и алгоритмы извлечения квадратного корня в кольце вычетов.

В практической части лабораторной работы была написана программа на языке C++, содержащая в себе реализацию всех описанных в теоретической части алгоритмов, работоспособность каждого из которых продемонстрирована на рисунках 1-10.

5 Код программы

```
#include <iostream>
#include <vector>

using namespace std;


int NegativeMod(int a, int m) {
    while (a < 0)
        a = a + m;
    return a % m;
}


int StandartEuclid(int a, int b) {
    if (b == 0)
        return a;
    else
        return StandartEuclid(b, a % b);
}


vector <vector <int>> ContinuedFraction(int a, int b) {
    vector <int> P, Q, q(2);
    vector <vector <int>> res;
    P = { 0, 1 };
    Q = { 1, 0 };
    int c, i = 2;
    while (b != 0) {
        q.push_back(a / b);
        P.push_back(q[i] * P[i - 1] + P[i - 2]);
        Q.push_back(q[i] * Q[i - 1] + Q[i - 2]);
        c = a;
        a = b;
        b = c % b;
        i++;
    }
    q.erase(q.begin(), q.begin() + 2);
    res.push_back(q);
    res.push_back(P);
    res.push_back(Q);
    return res;
}


void PrintPQ(vector <int> vec) {
    int size = vec.size();
    for (int i = 0; i < size - 1; i++)
        cout << vec[i] << ", ";
    cout << vec[size - 1] << "\n";
}


void CFInit() {
    int a0, a1;
```

```
    cout << "\nВведите a0\n";
    cin >> a0;
    if (a0 < 1){
        cerr << "\na0 должно быть больше 0\n";
        return;
    }
    cout << "Введите a1\n";
    cin >> a1;
    if (a1 < 1) {
        cerr << "\na1 должно быть больше 0\n";
        return;
    }
    vector <vector <int>> CF = ContinuedFraction(a0, a1);
    vector <int> q, P, Q;
    q = CF[0];
    P = CF[1];
    Q = CF[2];
    if (q.size() == 1)
        cout << "\na0/a1 = " << q[0] << "\n";
    else {
        cout << "\na0/a1 = (" << q[0] << "; ";
        for (int i = 1; i < q.size() - 1; i++)
            cout << q[i] << ", ";
        cout << q[q.size() - 1] << ")\n";
    }
    cout << "Числители подходящих дробей:   ";
    PrintPQ(P);
    cout << "Знаменатели подходящих дробей: ";
    PrintPQ(Q);
    return;
}


vector <int> DiophantineEquations(int a, int b, int c) {
    vector <vector <int>> CF = ContinuedFraction(a, b);
    vector <int> res;
    int nod = StandartEuclid(a, b);
    if (c % nod != 0)
        return res;
    int k = CF[1].size() - 2;
    int P, Q;
    P = CF[1][k];
    Q = CF[2][k];
    res.push_back(Q * (c / nod));
    res.push_back(P * (c / nod));
    if (k % 2 != 0) {
        res[0] = -1 * res[0];
        res[1] = -1 * res[1];
    }
    return res;
}


void DEInit() {
    int a, b, c;
    cout << "\nВведите a\n";
```

```
    cin >> a;
    if (a < 1) {
        cerr << "\na должно быть больше 0\n";
        return;
    }
    cout << "Введите b\n";
    cin >> b;
    if (b < 1) {
        cerr << "\nb должно быть больше 0\n";
        return;
    }
    cout << "Введите c\n";
    cin >> c;
    if (c < 1) {
        cerr << "\nc должно быть больше 0\n";
        return;
    }
    vector <int> DE = DiophantineEquations(a, b, c);
    if (DE.size() == 0) {
        cout << "Нет решений\n";
        return;
    }
    cout << "\nРешением уравнения " << a << "x - " << b << "y = " << c << "
является x = " << DE[0] << " и y = " << DE[1] << "\n";
    return;
}


int InverseElement(int a, int m) {
    vector <vector <int>> CF = ContinuedFraction(a, m);
    int x, k = CF[2].size() - 2;
    if (k % 2 != 0) {
        x = -1 * CF[2][k];
        x = NegativeMod(x, m);
    }
    else
        x = CF[2][k] % m;
    return x;
}


void IEInit() {
    int a, m;
    cout << "\nВведите a\n";
    cin >> a;
    if (a < 1) {
        cerr << "\na должно быть больше 0\n";
        return;
    }
    cout << "Введите m\n";
    cin >> m;
    if (m < 1) {
        cerr << "\nm должно быть больше 0\n";
        return;
    }
    if (StandartEuclid(a, m) != 1) {
```

```
        cerr << "\na и m должны быть взаимно простыми\n";
        return;
    }
    cout << "\nОбратным элементом является " << InverseElement(a, m) << "\n";
    return;
}


int Comparison(int a, int b, int m) {
    vector <vector <int>> CF = ContinuedFraction(a, m);
    int nod = StandartEuclid(a, m);
    int x, k = CF[2].size() - 2;
    if (k % 2 != 0) {
        x = -1 * CF[2][k] * (b / nod);
        x = NegativeMod(x, m);
    }
    else
        x = (CF[2][k] * (b / nod)) % m;
    return x;
}


void CInit() {
    int a, b, m;
    cout << "\nВведите a\n";
    cin >> a;
    if (a < 1) {
        cerr << "\na должно быть больше 0\n";
        return;
    }
    cout << "Введите b\n";
    cin >> b;
    if (b < 1) {
        cerr << "\nb должно быть больше 0\n";
        return;
    }
    cout << "Введите m\n";
    cin >> m;
    if (m < 1) {
        cerr << "\nm должно быть больше 0\n";
        return;
    }
    int nod = StandartEuclid(a, m);
    if (b % nod != 0) {
        cout << "\nНет решений\n";
        return;
    }
    cout << "\nРешением линейного сравнения " << a << "x = " << b << "(mod " <<
m << ") является x = " << Comparison(a, b, m) << "\n";
}


int Exponentiation(int x, int n, int m) {
    int N = n, Y = 1, Z = x % m;
    while (N != 0) {
        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;
    }
    return Y % m;
}


int Jac(int a, int b) {
    if (StandartEuclid(a, b) != 1)
        return 0;
    else {
        int r = 1;
        while (a != 0) {
            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);
            int c = a;
            if (c != 0)
                a = Exponentiation(b, 1, c);
            b = c;
        }
        return r;
    }
}


void JInit() {
    int a, p;
    cout << "\nВведите a\n";
    cin >> a;
    if (a < 1) {
        cerr << "\na должно быть больше 0\n";
        return;
    }
    cout << "Введите p\n";
    cin >> p;
    if (p < 1) {
        cerr << "\np должно быть больше 0\n";
        return;
    }
    a = a % p;
```

```
    cout << "\nСимвол Якоби от (" << a << "/" << p << ") равняется " << Jac(a, p) << "\n";
}


vector <int> RevPol(vector <int> a) {
    vector <int> a1;
    for (int i = a.size() - 1; i >= 0; i--)
        a1.push_back(a[i]);
    return a1;
}


vector <int> RaznPol(vector <int> a1, vector <int> b1, int field) {
    vector <int> a = a1;
    vector <int> b = b1;
    int m = max(a.size(), b.size());
    vector <int> c1;
    c1.resize(m, 0);
    if (a.size() > b.size()) {
        while (a.size() != b.size())
            b.push_back(0);
    }
    if (a.size() < b.size()) {
        while (a.size() != b.size())
            a.push_back(0);
    }
    for (int i = 0; i < m; i++) {
        c1[i] = a[i] - b[i];
        if (c1[i] < 0)
            c1[i] = c1[i] + field;
    }
    int k = 0;
    int i = c1.size() - 1;
    while (i != -1 && c1[i] == 0) {
        k++;
        i--;
    }
    c1.resize(c1.size() - k);
    return c1;
}


vector <int> YmnojPol(vector <int> a, vector <int> b, int field) {
    a = RevPol(a);
    b = RevPol(b);
    vector <int> c1;
    int raz = (a.size() - 1) + (b.size() - 1) + 1;
    c1.resize(raz, 0);
    for (int i = 0; i < a.size(); i++)
        for (int j = 0; j < b.size(); j++)
            c1[i + j] = (c1[i + j] + (a[i] * b[j])) % field;
    c1 = RevPol(c1);
    return c1;
}
```

```
vector <int> PDF(vector <int> c, vector <int> d, int p) {
    vector <int> q;
    int m = c.size() - 1;
    int n = d.size() - 1;
    q.resize((m + 1) - (n + 1) + 1);
    int check = 0;
    for (int k = (m + 1) - (n + 1); k >= 0; k--) {
        if (c[n + k] % d[n] != 0) {
            for (int i = 0; i < p; i++)
                if ((d[n] * i) % p == 1) {
                    q[k] = (c[n + k] * i) % p;
                    break;
                }
        }
        else
            q[k] = c[n + k] / d[n];
        if (c[n + k] == 0 && n > n + k - 1 && k > 1) {
            vector <int> a = { 0 };
            return a;
        }
        for (int j = (n + 1) + k - 1; j >= k; j--)
            c[j] = c[j] - ((q[k] * d[j - k]) % p);
        for (int i = 0; i < c.size(); i++)
            if (c[i] < 0)
                c[i] = c[i] + p;
    }
    return q;
}


vector<int> NODPol(vector<int> c, vector<int> d, int p) {
    vector <int> res1;
    vector <int> p0 = c, p1 = d, p0prom, p1prom;
    while (!(p1.size() == 1 && p1[0] == 0)) {
        if (p1.size() == 0)
            break;
        vector <int> q1 = PDF(p0, p1, p);
        p0prom.clear(); p0prom = p1;
        p1prom.clear(); p1prom = RaznPol(p0, YmnojPol(p1, q1, p), p);
        p0.clear();     p0 = p0prom;
        p1.clear();     p1 = p1prom;
    }
    return p0;
}


int SquareRootChip(int a, int p) {
    int b, J = 5;
    srand(time(0));
    do {
        b = rand() % p;
        J = Jac(NegativeMod(b * b - 4 * a, p), p);
    } while (J != -1);
    vector <int> c((p + 1) / 2 + 1, 0), d(3);
    b = NegativeMod(-1 * b, p);
```

```
    c[c.size() - 1] = 1;
    d = { a, b, 1 };
    vector<int> res = NODPol(c, d, p);
    return res[0];
}


unsigned long long TwoDeg(unsigned long long deg) {
    unsigned long long res = 1;
    for (int j = 1; j <= deg; j++)
        res = res * 2;
    return res;
}


unsigned long long SquareRoot(unsigned long long a, unsigned long long p) {
    unsigned long long q = p - 1, m = 0, b;
    vector <unsigned long long> ai, ki;
    while (q % 2 == 0) {
        q = q / 2;
        m = m + 1;
    }
    b = rand() % p;
    while (Jac(b, p) != -1)
        b = rand() % p;
    ai.push_back(a);
    for (int i = 0;; i++) {
        unsigned long long deg = 1;
        unsigned long long k = 0;
        while (Exponentiation(ai[i], deg * q, p) != 1) {
            k++;
            deg = deg * 2;
        }
        ki.push_back(k);
        if (ki[i] == 0)
            break;
        ai.push_back((ai[i] * Exponentiation(b, TwoDeg(m - ki[i]), p)) % p);
    }
    int size = ki.size();
    vector <unsigned long long> ri(size);
    ri[size - 1] = Exponentiation(ai[size - 1], (q + 1) / 2, p);
    for (int i = size - 2; i > -1; i--) {
        unsigned long long bi = Exponentiation(b, TwoDeg(m - ki[i] - 1), p);
        long long biobr = NegativeMod(InverseElement(bi,p), p);
        ri[i] = (ri[i + 1] * biobr) % p;
    }
    return ri[0];
}


void SRInit() {
    int a, p;
    cout << "\nВведите a\n";
    cin >> a;
    if (a < 1) {
        cerr << "\na должно быть больше 0\n";
```

```
        return;
    }
    cout << "Введите p\n";
    cin >> p;
    if (p < 1) {
        cerr << "\np должно быть больше 0\n";
        return;
    }
    if (Jac(a, p) != 1) {
        cerr << "\nСимвол Якоби от (a/p) должен равняться 1\n";
        return;
    }
    int SR1 = SquareRootChip(a, p);
    cout << "\nРешение, полученное алгоритмом Чиполлы: x = " << SR1 << "\n";
    cout << "Проверка: " << SR1 << " * " << SR1 << "(mod " << p << ") = " <<
NegativeMod(SR1 * SR1, p) << "\n";
    int SR2 = SquareRoot(a, p);
    cout << "\nРешение, полученное вторым алгоритмом: x = " << SR2 << "\n";
    cout << "Проверка: " << SR2 << " * " << SR2 << "(mod " << p << ") = " <<
NegativeMod(SR2 * SR2, p) << "\n";
}


int main() {
    setlocale(LC_ALL, "Russian");
    int k = 100;
    while (k != 0) {
        cout << "\nВыберите:\n";
        cout << "1-Разложение чисел в цепную дробь\n";
        cout << "2-Решение линейных диофантовых уравнений ax + by = c\n";
        cout << "3-Вычисление обратных элементов в кольце вычетов Zm\n";
        cout << "4-Решение линейных сравнений ax = b(mod m)\n";
        cout << "5-Вычисление символов Лежандра и Якоби\n";
        cout << "6-Извлечение квадратного корня в кольце вычетов\n\n";
        cin >> k;
        if (k == 1) {
            CFInit();
            continue;
        }
        if (k == 2) {
            DEInit();
            continue;
        }
        if (k == 3) {
            IEInit();
            continue;
        }
        if (k == 4) {
            CInit();
            continue;
        }
        if (k == 5) {
            JInit();
            continue;
        }
        if (k == 6) {
```

```
            SRInit();
            continue;
        }
        cerr << "Введённый вариант ответа отсутствует\n";
        k = 100;
    }
    return 0;
}
```