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 3](#_Toc178632526)

[2 Алгоритм разложения несократимой дроби в цепную дробь 4](#_Toc178632527)

[3 Алгоритмы приложений цепных дробей 9](#_Toc178632528)

[3.1 Решение Диофантовых уравнений 9](#_Toc178632529)

[3.2 Решение линейных сравнений 11](#_Toc178632530)

[3.3 Поиск обратного элемента в кольце вычетов Zm 12](#_Toc178632531)

[4 Алгоритмы вычисления символов Лежандра и Якоби 14](#_Toc178632532)

[4.1 Символ Лежандра 14](#_Toc178632533)

[4.2 Символ Якоби 16](#_Toc178632534)

[5 Алгоритмы извлечения квадратного корня в кольце вычетов 18](#_Toc178632535)

[6 Тестирование реализованных алгоритмов 21](#_Toc178632536)

[6.1 Алгоритм разложения чисел в цепную дробь 21](#_Toc178632537)

[6.3 Алгоритмы приложений цепных дробей 21](#_Toc178632538)

[6.4 Алгоритмы вычисления символов Лежандра и Якоби 23](#_Toc178632539)

[6.5 Алгоритмы извлечения квадратного корня в кольце вычетов 23](#_Toc178632540)

[ЗАКЛЮЧЕНИЕ 25](#_Toc178632541)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 26](#_Toc178632542)

[ПРИЛОЖЕНИЕ А 27](#_Toc178632543)

# **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 , представленное в виде несократимой дроби = . Так как (, ) = 1, то результат вычисления этого наибольшего общего делителя по алгоритму Евклида имеет вид:

………………………………..

,

Эти равенства можно переписать в виде:

Тогда рациональное число можно представить следующим образом:

где – целое число и , …, – целые положительные числа.

**Определение.** Выражение такого вида

принято называть цепной (или непрерывной) дробью с неполными частными , , …, и обозначать символом (;, ..., ).

Таким образом, любая несократимая рациональная дробь может быть представлена в виде цепной дроби = (;, ..., ) с неполными частными ;, ..., , полученными в результате вычисления по алгоритму Евклида наибольшего общего делителя взаимно простых чисел .

**Определение.** Для цепной дроби = (;, ..., ) выражения

называются подходящими дробями конечной цепной дроби (;, ..., ) и обозначаются символами = (;, ..., ) где .

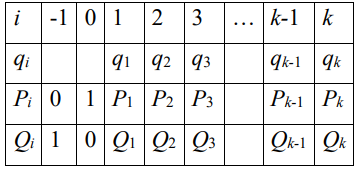
Аналогично определяются подходящие дроби = (;, ..., ) для бесконечной цепной дроби (;, ..., ).

Каждая подходящая дробь = (;, ..., ) является несократимой рациональной дробью с числителем и знаменателем , которые вычисляются по следующим рекуррентным формулам:

с начальными условиями

Вычисление числителей и знаменателей подходящих дробей с учетом

оформляется в виде следующей таблицы:



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 . Где – это золотое сечение равное (корень квадратного уравнения ), а – такое, что в алгоритме Евклида для . Так как операция деления занимает , то общая сложность будет , то есть можно сказать, что сложность алгоритма составляет .

**Описание алгоритма построения по заданной несократимой дроби цепной:**

**Вход:** два целых числа и , такие что и .  
**Выход:** список коэффициентов разложения в цепную дробь.

1. Если , поменять местами и .
2. Инициализировать пустой список для хранения коэффициентов цепной дроби.
3. Пока :
   1. Найти частное от деления на и добавить его в список.
   2. Присвоить значение , а — остаток от деления предыдущего на .
4. Когда , вернуть список коэффициентов цепной дроби.

**Псевдокод алгоритма:**

Начало алгоритма

Функция

Если :

Поменять местами и

Инициализировать пустой список

Пока :

Добавить в

Присвоить значение

Присвоить значение остатка

Вернуть

Конец алгоритма

**Сложность алгоритма:**

**Описание алгоритма построения по заданной несократимой дроби подходящих дробей:**

**Вход:** два целых числа и , такие что и .  
**Выход:** два списка: и , содержащие числители и знаменатели последовательных подходящих дробей.

1. Инициализировать два списка и , которые будут хранить числители и знаменатели подходящих дробей соответственно.
2. Пока :
   1. Вычислить целую часть от деления на .
   2. Добавить в список следующее значение: (рекуррентная формула для числителя).
   3. Добавить в список следующее значение: (рекуррентная формула для знаменателя).
   4. Присвоить a значение , а — остаток от деления предыдущего на .
3. Когда , вернуть списки и , содержащие все числители и знаменатели подходящих дробей.

**Сложность алгоритма:**

**Псевдокод алгоритма**

Начало алгоритма

Функция

Инициализировать

Пока

Добавить в

Добавить в

Присвоить значение

Присвоить значение остатка

Вернуть и

Конец алгоритма

**Сложность алгоритма:**

# **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

с целыми неотрицательными коэффициентами .

**Важные свойства подходящих дробей:**

* + 1. Числители и знаменатели двух последовательных подходящих дробей удовлетворяют равенству для всех
    2. взаимно просты
    3. -
    4. - =
    5. – убывающая последовательность

– возрастающая последовательность

* + 1. сходится по признаку Лейбница
    2. все , в частности, любое число представляется цепной дробью.

Если коэффициенты *a*, *b* удовлетворяют условию (*a*, *b*)=1 и – предпоследняя подходящая дробь представления числа в виде цепной дроби, то из равенств следует, что то есть значения являются целочисленными решениями уравнения . Легко видеть, что все целые решения исходного диофантова уравнениянаходятся по формулам:

где – произвольное целое число.

Нетрудно убедиться, что все решения диофантова уравнения

с взаимно простыми коэффициентами находятся по формулам:

где – произвольное целое число.

**Описание алгоритма решения Диофантового уравнения:**

**Вход:** три целых числа , где уравнение имеет вид

**Выход:** одно решение системы линейного Диофантова уравнения.

1. Найти наибольший общий делитель .
2. Если делится на , то:
   1. Упрощаем уравнение: делим , на
   2. Вычисляем подходящие дроби для и с помощью алгоритма цепных дробей.
   3. Определяем , длину списка числителей .
   4. Вычисляем решения и с помощью формул:
      1. , где — список знаменателей.
      2. где — список числителей.
3. Возвращаем одно решение .

Если условие делимости на не выполняется, то решение не существует.

**Сложность алгоритма:** Так как основывается на вычислении подходящих дробей, то сложность можно оценить как

**Псевдокод алгоритма:**

Начало алгоритма

Функция

Найти

Если

Вычислить и с помощью

= длина списка

Вернуть

Конец алгоритма

**Сложность алгоритма:**

## **3.2 Решение линейных сравнений**

Пусть . Сравнение невозможно, если не делится на . При , кратном , сравнение имеет решений.

Поиск решений. Так как ранее были рассмотрены решения диофантовых уравнений, то для нахождения решения достаточно просто решить диофантовое уравнение , а затем взять найденное значение , после чего взять его по модулю . Сложность нахождения корня линейного сравнения основывается на сложности алгоритма Евклида, то есть (). Где при обычных арифметических операциях.

**Алгоритм решения линейных сравнений:**

**Вход:** три целых числа , где уравнение имеет вид .  
**Выход:** одно решение уравнения.

1. Решить Диофантово уравнение (или эквивалентно ) с помощью функции решения Диофантова уравнения
2. Получить значение , которое будет решением линейного сравнения .
3. Вернуть как результат.

**Сложность алгоритма:**

**Псевдокод реализованного алгоритма:**

Начало алгоритма

Функция

Решить уравнение с помощью

Получить из решения

Вернуть

Конец алгоритма

**Сложность алгоритма:**

## **3.3 Поиск обратного элемента в кольце вычетов Zm**

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* ∙ *b* 1 (mod *m*), и его нередко обозначают как .

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 (*a*, *m*) = 1 следует, что это уравнение имеет решение, которое можно найти с помощью расширенного алгоритма Евклида (отсюда же, кстати говоря, следует, что при (*a*, *m*) ≠ 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* (mod *m*). Таким образом, найденный *x* и будет являться обратным к *a*. Сложность нахождения обратного элемента основывается на сложности алгоритма Евклида, то есть (). Где

**Алгоритм поиска обратного элемента в кольце вычетов:**

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

**Выход:** число , такое что

1. Решить Диофантово уравнение с помощью функции . Это уравнение находит такие и , что .
2. Вернуть , так как это и есть обратный элемент числа по модулю , то есть

**Сложность алгоритма:**

**Псевдокод реализованного алгоритма:**

Начало алгоритма

Функция

Решить уравнение с помощью

Получить из решения

Вернуть

Конец алгоритма

**Сложность алгоритма:**

# **4 Алгоритмы вычисления символов Лежандра и Якоби**

## **4.1 Символ Лежандра**

Число *a* ∈ Z*p* называется квадратичным вычетом по модулю *p*, если (∃*x* ∈ Z) .В противном случае число *a* называется квадратичным невычетом по модулю *p*. Символом Лежандра числа *a* ∈ Z называется выражение:

Вычисление символа Лежандра осуществляется на основе свойств основным из которых является критерий Эйлера, согласно которому любой квадратичный вычет удовлетворяет сравнению , а квадратичный невычет удовлетворяет . Таким образом имеем следующие свойства, на которых будет основываться алгоритм вычисления символа Лежандра:

* + - 1. . В частности, для любого простого , при и при
      2. Квадратичный закон взаимности Гаусса

Для любых нечетных простых чисел

**Алгоритм вычисления символа Лежандра:**

**Вход:** два числа и , где — нечетное простое число.  
**Выход:** символ Лежандра , который показывает, является ли квадратичным вычетом по модулю.

1. Если 2 выделяем множитель
2. Заменяем на остаток от деления на
3. Представляем и вычисляем по свойству 2

При этом опускаем множители сс четными степенями и вместо множителей c нечетными степенями оставляем ,

1. Если , то вычисляем по свойству 5
2. К остальным символам применяется квадратичный закон взаимности Гаусса
3. При необходимости возвращаемся к пункту 2

**Сложность алгоритма:**

**Псевдокод реализованного алгоритма:**

Начало алгоритма

Функция

Если не является простым:

Вернуть

Если

Если

Вернуть

Иначе:

Вернуть

Если

Вернуть 0

Если

Вернуть 1

Если

Если или :

Вернуть 1

Иначе:

Вернуть -1

Если a чётное:

Вернуть

Если a нечётное: Если и :

Вернуть

Иначе: Вернуть

Конец алгоритма

**Сложность алгоритма:**

## **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 3 битовых операций (если не использовать быстрое умножение и деление). А вычисление символа Якоби требует только 2 битовых операций.

**Алгоритм вычисления символа Якоби:**

**Вход:** два числа и где — нечётное целое число.  
 **Выход:** символ Якоби , который обобщает символ Лежандра для случая, когда не является простым числом.

1. Заменить на такое , что и
2. Если , то по свойству 2 рассмотренном при описании символа Лежандра выделяем множитель
3. Если b – четное, то представляем и при нечетном вычисляем
4. К символу применяется квадратичный закон взаимности Гаусса
5. При необходимости возвращаемся к пункту 1

**Сложность реализованного алгоритма:** 2

**Псевдокод реализованного алгоритма:**

Начало алгоритма

Функция

Пока

Пока

Если или :

Поменять местами a и n

Если и :

Если :

Вернуть

Иначе:

Вернуть

Конец алгоритма

**Сложность реализованного алгоритма:** 2

# **5 Алгоритмы извлечения квадратного корня в кольце вычетов**

Для решения сравнения , где – квадратичный вычет, – нечетное простое число можно использовать алгоритм Тонелли — Шенкса который записывается следующим образом.

**Алгоритм вычисления квадратного корня в кольце вычетов:**

**Вход:** целые числа и простое , где нужно найти , такое что .  
**Выход:** два значения и , являющиеся решениями уравнения , если решение существует.

1. Проверить, является ли квадратичным вычетом по модулю с помощью символа Лежандра. Если символ Лежандра , решения не существует, вернуть
2. Если, решение вычисляется напрямую как .
3. Найти такие и, что = , где нечетное
4. Найти , такой что (то есть не является квадратичным вычетом по модулю
5. Инициализировать
6. Пока
   1. Найти минимальное i, такое что
   2. Вычислить
   3. Обновите значения:
7. Вернуть два решения и .

**Сложность алгоритма:**

**Псевдокод реализованного алгоритма:**

Начало алгоритма

Функция :

Если :

Вернуть None, None

Если :

Вернуть

Пока

Пока

Пока :

Для от 1 до:

Если :

Прервать цикл

Вернуть

Конец алгоритма

**Сложность алгоритма:**

# **6 Тестирование реализованных алгоритмов**

## **6.1 Алгоритм разложения чисел в цепную дробь**

Вычислим цепные дроби для дробей и . В результате получим следующие разложения = , . Подробная информация представлена на рисунке 1.

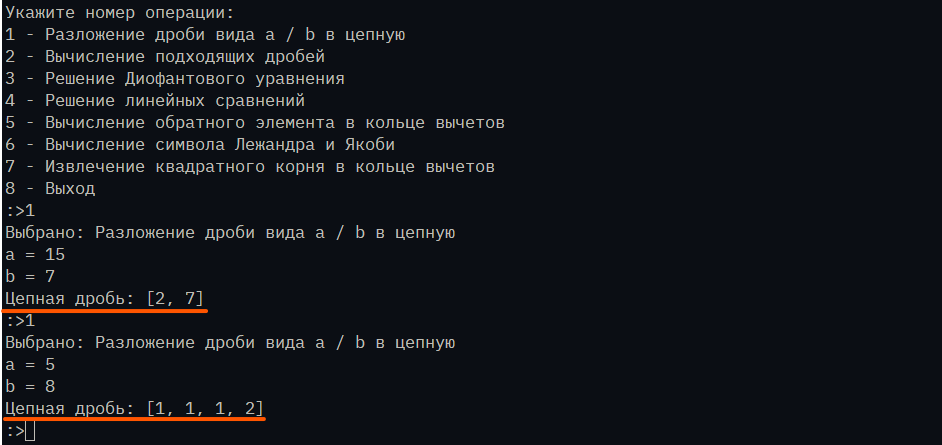


Рисунок 1 – Вычисление цепных дробей

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

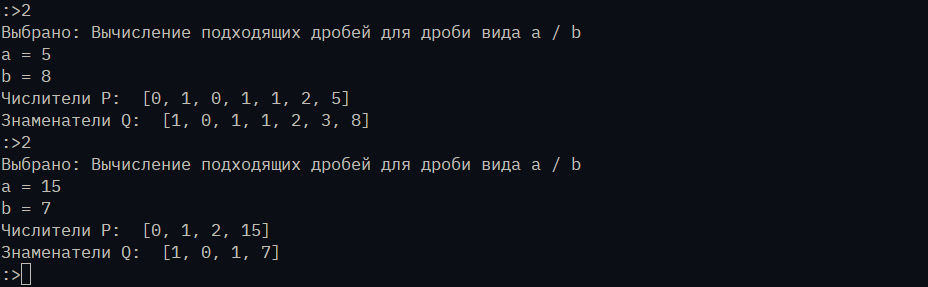


Рисунок 2 – Вычисление подходящих дробей

## **Алгоритмы приложений цепных дробей**

Решим диофантово уравнение следующего вида: . Решение приведено на рисунке 3. Также на рисунке представлено решение для произвольного параметра .

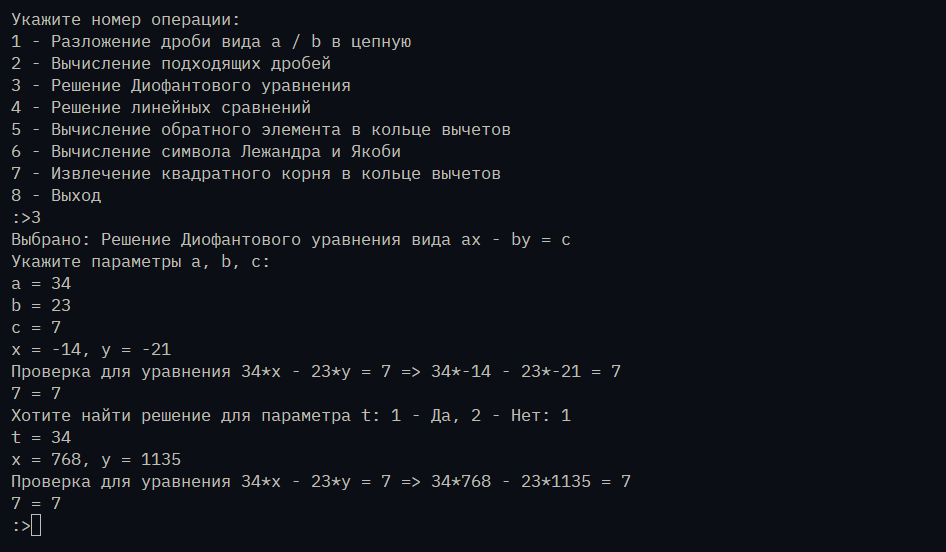


Рисунок 3 – Решение диофантового уравнения

Решим линейное сравнение следующего вида: , для такого сравнения . Проверка и решение представлено на рисунке 4.

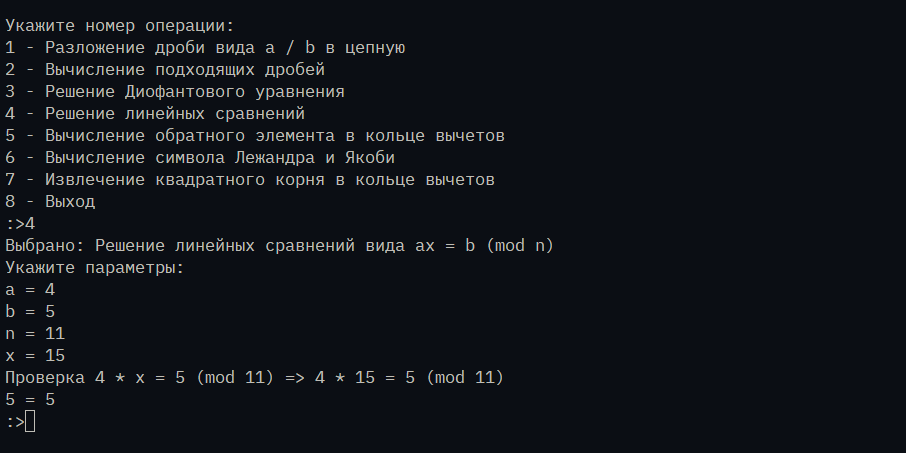


Рисунок 4 – Решение линейных сравнений

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

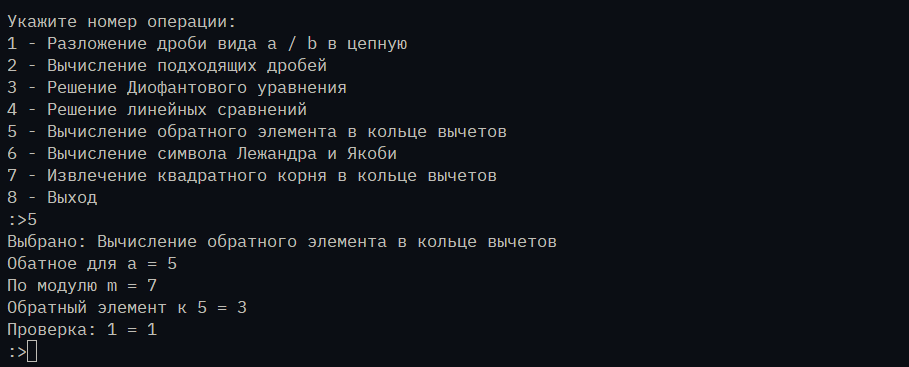


Рисунок 5 – Вычисление обратного элемента в кольце вычетов

## **Алгоритмы вычисления символов Лежандра и Якоби**

Вычислим символы Лежандра и Якоби для: , . Заметим, что символ Лежандра требует по условию простое , а символ Якоби не накладывает таких условий. Решение представлено на рисунке 6.

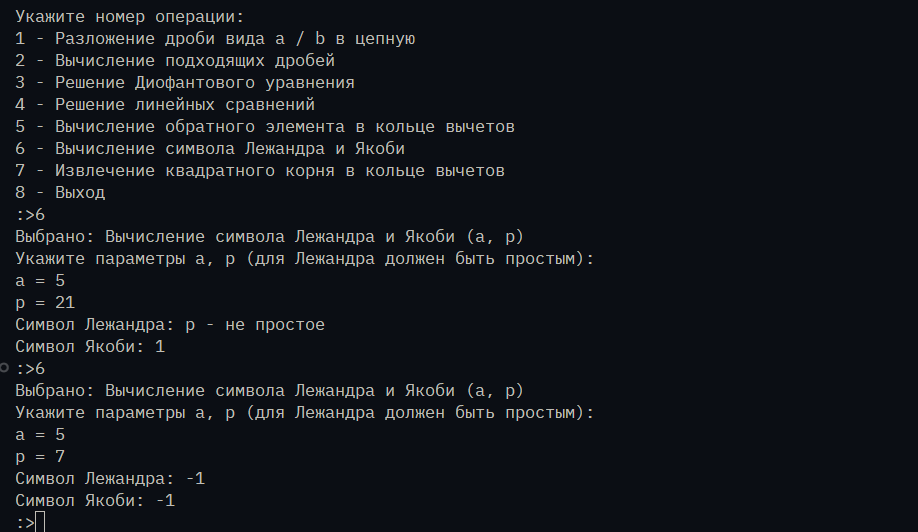


Рисунок 6 – Вычисление символов Лежандра и Якоби

## **Алгоритмы извлечения квадратного корня в кольце вычетов**

Вычислим: , , решения с результатами тестирования представлены на рисунке 7.

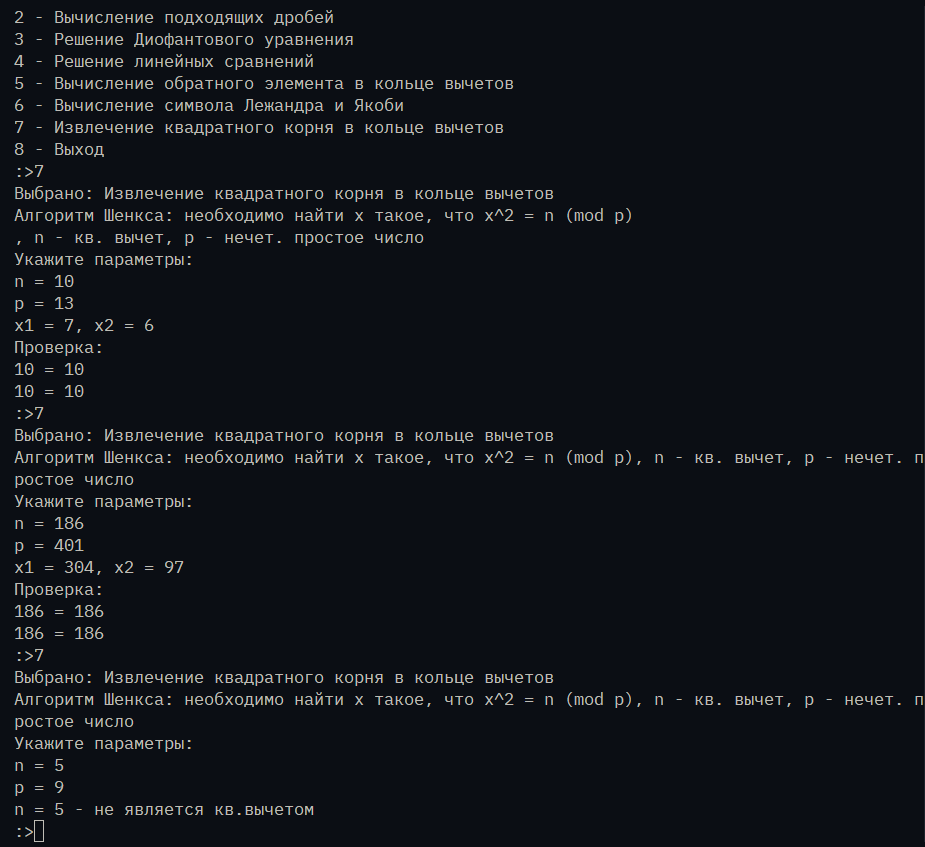


Рисунок 7 - Извлечение квадратного корня в кольце вычетов

# **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 APB, 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**

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

*# 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 а, 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"