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

Мызникова Сергея Анатольевича

|  |  |  |
| --- | --- | --- |
| Преподаватель, профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc179796713)

[2 Теоретические сведения 4](#_Toc179796714)

[2.1 Алгоритм разложения чисел в цепную дробь 4](#_Toc179796715)

[2.2 Решение Диофантовых уравнений 7](#_Toc179796716)

[2.3 Решение линейных сравнений 7](#_Toc179796717)

[2.4 Поиск обратного элемента в кольце вычетов Zm 8](#_Toc179796718)

[2.5 Символ Лежандра 8](#_Toc179796719)

[2.6 Символ Якоби 9](#_Toc179796720)

[2.7 Алгоритм извлечения квадратного корня в кольце вычетов 9](#_Toc179796721)

[3 Результаты работы 10](#_Toc179796722)

[3.1 Алгоритмы: описание и временная сложность 10](#_Toc179796723)

[3.1.1 Алгоритм разложения чисел в цепную дробь 10](#_Toc179796724)

[3.1.2 Решение диофантовых уравнений 10](#_Toc179796725)

[3.1.3 Решение линейных сравнений 11](#_Toc179796726)

[3.1.4 Поиск обратного элемента 11](#_Toc179796727)

[3.1.5 Символ Лежандра 12](#_Toc179796728)

[3.1.6 Символ Якоби 12](#_Toc179796729)

[3.1.7 Алгоритм извлечения квадратного корня в кольце вычетов 13](#_Toc179796730)

[3.2 Тестирование алгоритмов 14](#_Toc179796731)

[ЗАКЛЮЧЕНИЕ 18](#_Toc179796732)

[ПРИЛОЖЕНИЕ А 19](#_Toc179796733)

# **1 Постановка задачи**

Цель работы — изучение основных методов проверки простоты чисел и их программная реализация.

Порядок выполнения работы:

1. Рассмотреть тест Ферма проверки чисел на простоту и привести его программную реализацию.
2. Рассмотреть тест Соловея-Штрассе на проверки чисел на простоту и привести его программную реализацию.
3. Рассмотреть тест Миллера-Рабина и привести его программную реализацию.

# **2 Теоретические сведения**

## **2.1 Тест Ферма**

Плотность распределения простых чисел:

1 − 10 - 40%, 1 − 100 - 25%, 1 − 1000 - 17%, 1 – 105 - <10%.

## **2.2 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 .

Предположим, что – последняя подходящая дробь в представлении непрерывной дробью рационального числа , где НОД(a, b) = 1. Тогда , . Перепишем уравнение для соседних подходящих дробей: . Получаем одно решение диофантового уравнения : , .

Остальные имеют вид:

В общем случае диофантово уравнение разрешимо, если число c делится на НОД(a, b):

## **2.3 Решение линейных сравнений**

Аналогично диофантовым уравнениям решаются уравнения первой степени вида: . Для решения нужно взять обе части диофантового уравнения в виде: по модулю m. Это сравнение будет разрешимо тогда, когда b делится на НОД(a, m). Решение будет иметь вид:

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

Обратным к числу *a* по модулю *m* называется такое число *b*, что: ab 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*.

Для решения задачи по поиску обратного элемента в кольце необходимо решить следующее линейное сравнение: ab 1 (mod m). Для этого можно воспользоваться диофантовыми уравнениями и получить выражение следующего вида: ax + my = 1.

## **2.5 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*.

Для нечетного простого числа символом Лежандра числа *a* ∈ Z называется выражение:

* + - 1. =
      2. для любого , НОД(c, p) = 1
      3. для НОД(a, p) = 1
      4. Квадратичный закон взаимности Гаусса

Для любых нечетных простых чисел

## **2.6 Символ Якоби**

**Определение.** Пусть натуральное число

Символом Якоби числа называется выражение:

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 (без разложения числа на множители).

## **2.7 Алгоритм извлечения квадратного корня в кольце вычетов**

Для решения сравнения , где – квадратичный вычет, – нечетное простое число можно использовать вероятностный алгоритм извлечения квадратного корня в поле Zp с арифметикой только этого поля. Вероятность успеха .

# **3 Результаты работы**

## **3.1 Алгоритмы: описание и временная сложность**

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

Вход: два целых числа a, b.

Выход: таблица разложения подходящих дробей

Шаг 1: положить в массив P числа 0 и 1

Шаг 2: положить в массив Q числа 1 и 0

Шаг 3: пока b не равно 0 выполнить следующее

Шаг 3.1: положить c = целая часть от a / b

Шаг 3.2: добавить c в массив

Шаг 3.3: по формуле вычислить новый элемент и добавить в массив P

Шаг 3.4: по формуле вычислить новый элемент и добавить в массив Q

Шаг 3.5: присвоить a целую часть от a / b, а b присвоить остаток и вернуться к шагу 1

Шаг 2: вывести массив с разложением, P, Q

**Временная сложность алгоритма**. O(L(a)\*L(b))

### **3.1.2 Решение диофантовых уравнений**

Вход: три целых числа a, b, c.

Выход: коэффициенты x, y

Шаг 1: найти НОД(a, b)

Шаг 2: если c делится на НОД, то продолжить

Шаг 3: вычислить разложение числа в цепную дробь a / b

Шаг 4: вычислить x в соответствии с формулой

Шаг 5: вычислить y в соответствии с формулой

Шаг 6: вывести x, y

**Временная сложность алгоритма.** O(2\*L(a)\*L(b) + 2\*L(c)\*L() \*L()

### **3.1.3 Решение линейных сравнений**

Вход: три целых числа a, b, m.

Выход: коэффициент x

Шаг 1: найти НОД(a, b)

Шаг 2: если c делится на НОД, то продолжить

Шаг 3: вычислить разложение числа в цепную дробь a / b

Шаг 4: вычислить x в соответствии с формулой

Шаг 5: вернуть x

**Временная сложность алгоритма**. O(2\*L(a)\*L(b) + L(b)\*L() \*)

### **3.1.4 Поиск обратного элемента**

Вход: два целых числа a, m

Выход: обратный элемент

Шаг 1: найти НОД(a, 1)

Шаг 2: если c делится на НОД, то продолжить

Шаг 3: вычислить разложение числа в цепную дробь a / 1

Шаг 4: вычислить x в соответствии с формулой

Шаг 5: вернуть x

**Временная сложность алгоритма**. O(2\*L(a) + L() \*)

### **3.1.5 Символ Лежандра**

Вход:

Выход: 1 или -1 или 0

Шаг 1: если a < 0, то по свойству 4 выделить множитель

Шаг 2: заменяем a на остаток от деления на p

Шаг 3: представляем a = и при этом опускаем множители с четными степенями, а у множителей с нечетными убираем степени и вычисляем

Шаг 4: если pi = 2, то вычисляем по свойству 6

Шаг 5: к остальным символам применяем квадратичный закон взаимности Гаусса.

**Временная сложность алгоритма**. O(log2p +

### **3.1.6 Символ Якоби**

Вход:

Выход: 1 или -1 или 0

Шаг 1: заменить на такое , что и

Шаг 2: если , то по свойству 2 рассмотренном при описании символа Лежандра выделяем множитель

Шаг 3: если b – четное, то представляем и при нечетном вычисляем

Шаг 4: к символу применяется квадратичный закон взаимности Гаусса

**Временная сложность алгоритма.** O(log2p)

### **3.1.7 Алгоритм извлечения квадратного корня в кольце вычетов**

Вход: , нечетное простое p

Выход: квадратный корень

Шаг 1: вычисляем разложение p – 1 = 2mq

Шаг 2: случайным образом выбрать b такое, что

Шаг 3: вычислить последовательность элементов поля Zp и последовательность по правилу:

1.

2. – наименьшее k 0, при котором

Выполнение шага 2 заканчивается в тот момент, когда выполняется равенство .

Шаг 3: вычислить последовательность по правилу:

, , i > 0

Шаг 4: положить - искомое решение

**Временная сложность алгоритма.** O(log4p)

## **3.2 Тестирование алгоритмов**

Разложение чисел в цепную дробь

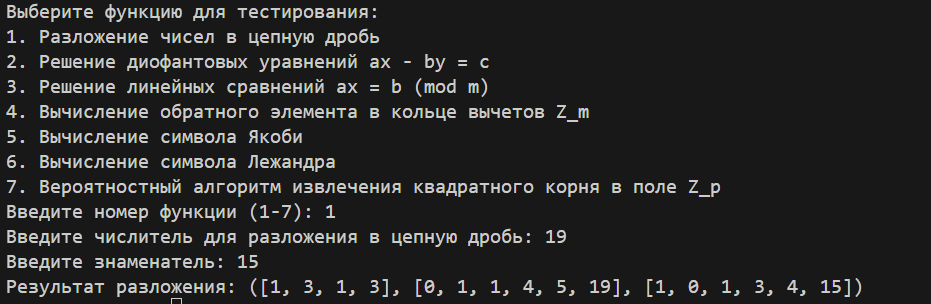


Рисунок 1. Тестирование разложения числа в цепную дробь

Решение диофантовых уравнений

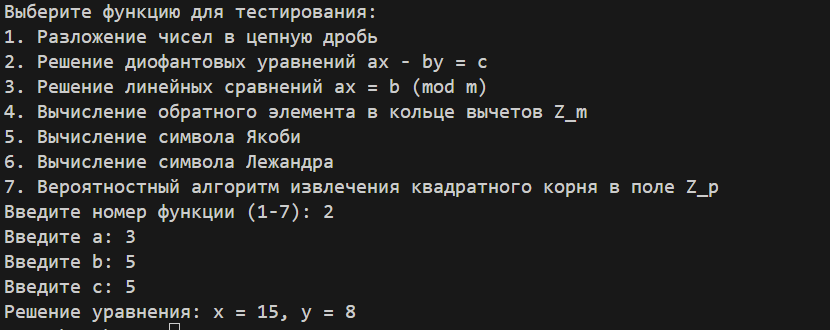


Рисунок 2. Тестирование алгоритма решения диофантовых уравнений

Решение линейных сравнений

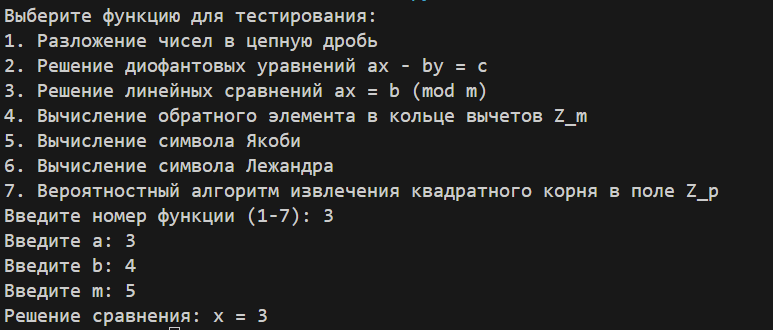


Рисунок 3. Тестирование алгоритма решения линейных сравнений

Вычисление обратного элемента в кольце вычетов

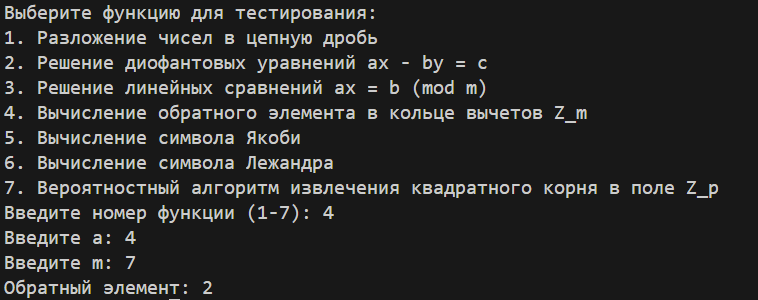


Рисунок 4. Тестирование алгоритма вычисления обратного элемента в кольце вычетов

Вычисление символа Якоби

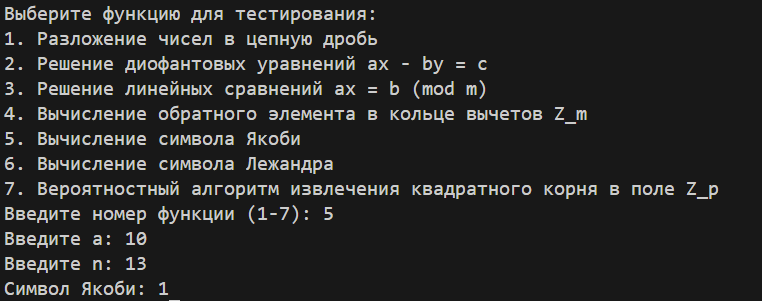


Рисунок 5. Тестирование алгоритма вычисления символа Якоби

Вычисления символа Лежандра

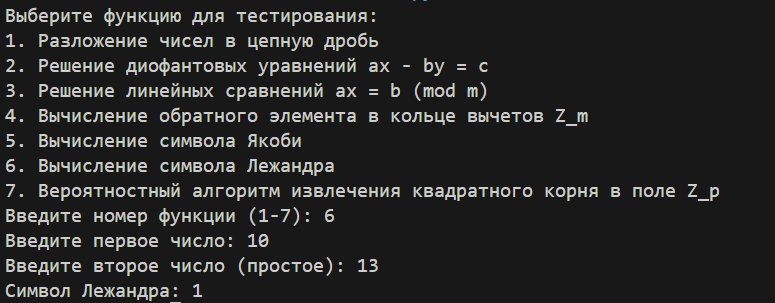


Рисунок 6. Тестирование алгоритма вычисления символа Лежандра

Вероятностный алгоритм извлечения квадратного корня в поле

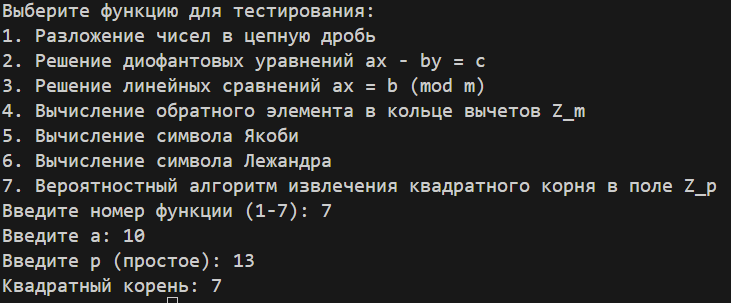


Рисунок 7. Тестирование алгоритма извлечения квадратного корня в поле

# **ЗАКЛЮЧЕНИЕ**

В ходе работы били рассмотрены алгоритм разложения чисел в цепную дробь, приложения к алгоритму разложения чисел в цепную дробь, а именно, решение диофантовых уравнений, решение линейных сравнений и поиск обратного элемента в кольце вычетов. Также были рассмотрены алгоритмы вычисления символов Лежандра и Якоби и вероятностный алгоритм извлечения квадратного корня в поле.

Для проверки работоспособности этих алгоритмов была реализована программа на языке Python. Программы были протестированы и была подтверждена правильность их работы. Также была произведена оценка временной сложности алгоритмов.

# **ПРИЛОЖЕНИЕ А**

**Программа с реализованными алгоритмами**

import random

import sympy

# Разложение чисел в цепную дробь

def continued\_fraction(numerator, denominator):

    result = []

    P = [0, 1]

    Q = [1, 0]

    i = 2

    while denominator != 0:

        integer\_part = numerator // denominator

        result.append(integer\_part)

        P.append(result[i - 2] \* P[i - 1] + P[i - 2])

        Q.append(result[i - 2] \* Q[i - 1] + Q[i - 2])

        remainder = numerator % denominator

        numerator, denominator = denominator, remainder

        i += 1

    return result, P, Q

def normal\_Euclid(a, b):

    while a !=0 and b != 0:

        if a > b:

            a = a % b

        else:

            b = b % a

    return a + b

# Решение диофантовых уравнений ax - by = c

def linear\_diofant(a, b, c):

    nod = normal\_Euclid(a, b)

    if c % nod != 0:

        return (0, 0)

    else:

        res, P, Q = continued\_fraction(a, b)

        x = pow(-1, len(res) - 2) \* c / nod \* Q[-2] + b

        y = pow(-1, len(res) - 2) \* c / nod \* P[-2] + a

        return (int(x), int(y))

# Решение линейных сравнений ax = b (mod m)

def compare(a, b, m):

    nod = normal\_Euclid(a, m)

    if b % nod != 0:

        return 0

    else:

        res, P, Q = continued\_fraction(a, m)

        x = pow(-1, len(res) - 2) \* b / nod \* Q[-2]

        return int(x % m)

# Вычисление обратного элемента в кольце вычетов Zm

def reverse(a, m):

    return compare(a, 1, m)

# Вычисление символа Якоби

def jacobi\_symbol(a, n):

    if n <= 0 or n % 2 == 0:

        return 0

    result = 1

    while a != 0:

        while a % 2 == 0:

            a //= 2

            if n % 8 in [3, 5]:

                result = -result

        a, n = n, a

        if a % 4 == 3 and n % 4 == 3:

            result = -result

        a %= n

    return result if n == 1 else 0

def factorization\_number(num):

    n = num

    i = 2

    result = []

    while i \* i <= n:

        while n % i == 0:

            result.append(i)

            n //= i

        i += 1

    if n > 1:

        result.append(n)

    return result

# Вычисление символа Лежандра

def legendre\_symbol(first, second):

    a = first

    factors = factorization\_number(second)

    if len(factors) == 1:

        p = second

    else:

        return None

    numerators = []

    result = 1

    if a < 0:

        numerators.append(-1)

        a = abs(a)

    a %= p

    arr = factorization\_number(a)

    count\_dict = {}

    for num in arr:

        if num in count\_dict:

            count\_dict[num] += 1

        else:

            count\_dict[num] = 1

    for key, count in count\_dict.items():

        if count % 2 == 1:

            numerators.append(key)

    for elem in numerators:

        if elem == -1:

            result \*= int(pow(-1, (p - 1) // 2))

            continue

        if elem == 2:

            result \*= int(pow(-1, (p \* p - 1) // 8))

        else:

            result \*= legendre\_symbol(p, elem) \* int(pow(-1, ((p - 1) // 2) \* ((elem - 1) // 2)))

    return result

# Вероятностный алгоритм извлечения квадратного корня в поле Zp с арифметикой только этого поля.

def sq(a, p):

    if not sympy.isprime(p):

        return None

    if legendre\_symbol(a, p) != 1:

        return None

    q = p - 1

    m = 0

    while q % 2 == 0:

        q //= 2

        m += 1

    while True:

        b = random.randint(0, p - 1)

        if legendre\_symbol(b, p) == -1:

            break

    buff = [a]

    k\_buff = []

    while int(pow(buff[-1], q, p)) != 1:

        for k in range(p):

            if int(pow(buff[-1], int(pow(2, k, p)) \* q, p)) == 1:

                k\_buff.append(k)

                buff.append(buff[-1] \* int(pow(b, int(pow(2, m - k\_buff[-1], p)), p)) % p)

                break

    k\_buff.append(0)

    buff.append(buff[-1] \* int(pow(b, int(pow(2, m - k\_buff[-1], p)), p)) % p)

    r = int(pow(buff[-1], (q + 1) // 2, p))

    for i in range(len(k\_buff) - 1, -1, -1):

        l = reverse(int(pow(b, int(pow(2, m - k\_buff[i] - 1, p)), p)), p)

        r = (r \* l) % p

    return r

def test\_function\_choice(choice):

    if choice == 1:

        numerator = int(input("Введите числитель для разложения в цепную дробь: "))

        denominator = int(input("Введите знаменатель: "))

        result = continued\_fraction(numerator, denominator)

        print(f"Результат разложения: {result}")

    elif choice == 2:

        a = int(input("Введите a: "))

        b = int(input("Введите b: "))

        c = int(input("Введите c: "))

        result = linear\_diofant(a, b, c)

        if result == (0, 0):

            print("Числа не подходят, уравнение не разрешимо.")

        else:

            print(f"Решение уравнения: x = {result[0]}, y = {result[1]}")

    elif choice == 3:

        a = int(input("Введите a: "))

        b = int(input("Введите b: "))

        m = int(input("Введите m: "))

        result = compare(a, b, m)

        if result == 0:

            print("Уравнение не разрешимо.")

        else:

            print(f"Решение сравнения: x = {result}")

    elif choice == 4:

        a = int(input("Введите a: "))

        m = int(input("Введите m: "))

        result = reverse(a, m)

        if result is None:

            print("Уравнение не разрешимо.")

        else:

            print(f"Обратный элемент: {result}")

    elif choice == 5:

        a = int(input("Введите a: "))

        n = int(input("Введите n: "))

        result = jacobi\_symbol(a, n)

        print(f"Символ Якоби: {result}")

    elif choice == 6:

        first = int(input("Введите первое число: "))

        second = int(input("Введите второе число (простое): "))

        result = legendre\_symbol(first, second)

        if result is None:

            print(f"Второе число должно быть простым")

        else:

            print(f"Символ Лежандра: {result}")

    elif choice == 7:

        a = int(input("Введите a: "))

        p = int(input("Введите p (простое): "))

        result = sq(a, p)

        if result is None:

            print("Числа не подходят.")

        else:

            print(f"Квадратный корень: {result}")

def main():

    print("Выберите функцию для тестирования:")

    print("1. Разложение чисел в цепную дробь")

    print("2. Решение диофантовых уравнений ax - by = c")

    print("3. Решение линейных сравнений ax = b (mod m)")

    print("4. Вычисление обратного элемента в кольце вычетов Z\_m")

    print("5. Вычисление символа Якоби")

    print("6. Вычисление символа Лежандра")

    print("7. Вероятностный алгоритм извлечения квадратного корня в поле Z\_p")

    choice = int(input("Введите номер функции (1-7): "))

    if 1 <= choice <= 7:

        test\_function\_choice(choice)

    else:

        print("Неверный выбор. Пожалуйста, введите число от 1 до 7.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()