МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«Северный (Арктический) федеральный университет имени М.В. Ломоносова»

(наименование высшей школы/ филиала/ института/ колледжа)

ОТЧЕТ

о лабораторном практикуме

По дисциплине: Математические методы защиты информации

На тему «Дискретное логарифмирование»

Выполнила обучающаяся:
Антоненко Николай Олегович
(ФИО)
Направление подготовки / специальность:
10.03.01
Информационная безопасность
(код и наименование)
Курс: 2
Группа: 151313

Руководитель:
Тарасов Александр Петрович
(ФИО руководителя)

Отметка о зачете ____________________ ____________________
(отметка прописью) (дата)

Руководитель ____________________ Тарасов А.П
(подпись руководителя) (инициалы, фамилия)

Архангельск 2024

Лабораторная работа №4.

1. Порождающий элемент — это элемент, который порождает группу, в этом случае уравнение дискретного логарифмирования всегда имеет решение.

Дискретный логарифм — это решение уравнения для заданных элементов, которое состоит в нахождении целого неотрицательного числа, удовлетворяющего этому уравнении.

2. Написать программы для реализации алгоритма метода Госпера для дискретного логарифмирования, как изображено на рисунке 1.

```python
import math

# Основная функция для нахождения дискретного логарифма
def gosper_log(base, value, modulus):
    # Инициализация таблицы для факториальных значений
    factorials = [1]#список
    for i in range(1, modulus):
        factorials.append((factorials[-1] * i) % modulus)#добавление в список

    # Функция для преобразования числа в факториальный базис
    def factorial_baziz(n):
        kof = []# список коэфициентов
        i = 1
        while n > 0:
            kof.append(n % i)#исползуем деление и остаток для опр коэфициентов
            n //= i
            i += 1
        return kof
    # Преобразуем базу и значение в факториальный базис
    base_factors = factorial_baziz(base)
    value_factors = factorial_baziz(value)
    log = 0#вычисление дискретного логарифма
    for i in range(len(value_factors)):
        log += value_factors[i] * pow(base, i, modulus)
        log %= modulus

    return log

# Пример использования
base = 2  # Основание логарифма
value = 28  # Значение, для которого ищем логарифм
modulus = 32  # Модуль, по которому выполняем операции

result = gosper_log(base, value, modulus)
print(f'Дискретный логарифм для {base} в {value} (mod {modulus}) = {result}')
```

IDLE Shell 3.12.2

File Edit Shell Debug Options Window Help

```
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/krivo/OneDrive/Рабочий стол/1.py
Дискретный логарифм для 2 в 28 (mod 32) = 24
```

Рисунок 1 – принцип работы алгоритма метода Госпера

Написать программы для реализации алгоритма Полига-Хеллмана, как изображено на рисунке 2.

```python
import math
import random
def gcd(a, b):
    while b:  # Пока b не равно нулю
        a, b = b, a % b  # Присваиваем a значение b, a b - остаток от деления a на b
    return a  # Возвращаем наибольший общий делитель

# Функция, используемая в алгоритме Полларда
def f(x, n):
    return (x * x + 1) % n  # Возвращаем (x^2 + 1) по модулю n

# Функция для нахождения делителя числа n с использованием ро-метода Полларда
def pollards_rho(n):
    if n % 2 == 0:  # Если n четное, возвращаем 2
        return 2

    x = random.randint(1, n - 1)  # Случайным образом выбираем начальное значение x
    y = x  # Инициализируем y таким же значением, что и x
    d = 1  # Инициализируем d как 1 (начальное значение для делителя)

    while d == 1:  # Пока d равно 1, продолжаем итерации
        x = f(x, n)  # Обновляем x с использованием функции f
        y = f(f(y, n), n)  # Обновляем y дважды с использованием функции f (ускорение)
        d = gcd(abs(x - y), n)  # Вычисляем НОД (x - y) и n

    if d == n:  # Если d равно n, возвращаем None (не удалось найти делитель)
        return None  # Не удалось найти делитель
    return d  # Возвращаем найденный делитель

# Пример использования функции для факторизации числа
if __name__ == "__main__":
    n = 8051  # Пример числа для факторизации
    factor = pollards_rho(n)  # Нахождение делителя числа n с использованием ро-метода Полларда
    if factor:
        print(f"Найден делитель {factor} для числа {n}")  # Выводим найденный делитель
    else:
        print("Не удалось найти делитель")  # Сообщаем, что не удалось найти делитель
```

Рисунок 2 - принцип работы алгоритма Полига-Хеллмана

Написать программы для реализации алгоритма исчисления порядка, как изображено на рисунке 3.

```python
def mod_exp(base, exp, mod):#для вычисления (base^exp) % mod с использованием метода быстрого возведения в степень.
    result = 1
    base = base % mod# Приводим base к значению по модулю
    while exp > 0:
        if exp % 2 == 1:  # Если exp нечетное, умножаем result на base
            result = (result * base) % mod# Умножаем result на base и берем по модулю mod
        exp = exp >> 1  # Делим exp на 2
        base = (base * base) % mod  # Удваиваем base и берем по модулю
    return result

def order(g, n):#  Функция для нахождения порядка элемента g в группе по модулю n.
    if gcd(g, n) != 1:
        raise ValueError(f"g ({g}) и n ({n}) не являются взаимно простыми, порядок не определен.")
    k = 1
    while mod_exp(g, k, n) != 1:  # Продолжаем увеличивать k, пока (g^k % n) не станет равно 1
        k += 1
    return k

def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Пример использования
if __name__ == "__main__":
    g = 2  # Основание
    n = 17  # Модуль
    result = order(g, n)
    print(f'Порядок элемента {g} в группе по модулю {n} = {result}')
```

Рисунок 3 - принцип работы алгоритма исчисления порядка

3. 1. Существуют ли первообразные корни по модулю n, и если существуют, то сколько их, как изображено на рисунке 4.

a) n = 15; b) n = 71; c) n = 53; d) n = 202; e) n = 16; f) n = 25.

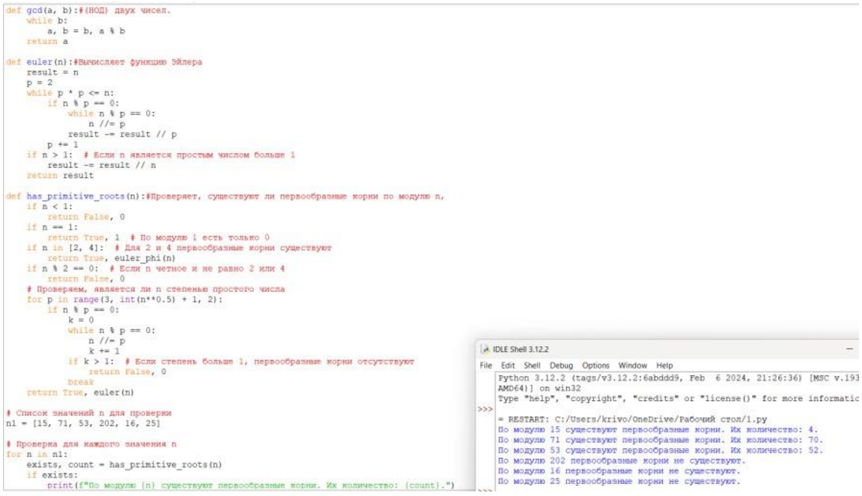

Рисунок 5 – проверка на первообразные корни по модулю

2. Найти первообразные корни по следующим модулям, как изображено на рисунке 6.

a) 3; c) 27; e) 26; g) 43; i) 169; k) 89;

b) 9; d) 13; f) 18; h) 86; j) 4; l) 41.

3. Вычислить следующие дискретные логарифмы, пользуясь алгоритмом «шаг младенца – шаг великана», как изображено на рисунке 7.

a) $\log_3 14 \bmod \varphi(31)$; b) $\log_5 42 \bmod \varphi(47)$; c) $\log_3 30 \bmod \varphi(89)$.

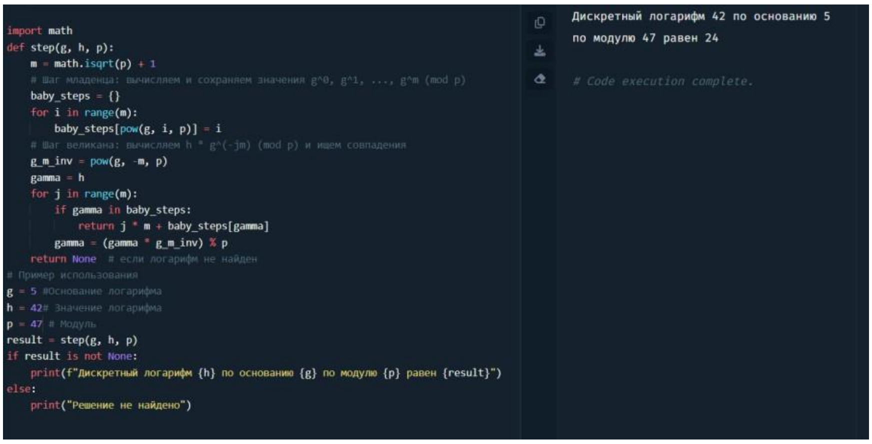

Рисунок 7 – вычисление дискретных логарифмов

Ответы: a) 18, b) 24, c) 87.

4. При помощи алгоритма исчисления порядка вычислить следующие дискретные логарифмы, как изображено на рисунке 8.

a) $\log_3 57 \bmod \varphi(89)$; b) $\log_3 61 \bmod \varphi(79)$; c) $\log_3 279 \bmod \varphi(587)$.

```python
def gcd(a, b):# поиск НОД
    if b == 0:
        return a
    return gcd(b, a % b)#рекурсия для б и остаток от деления а на б
def eul(n):# вычисления кол-ва чисел меньших n
    res = 1
    for i in range(2, n):
        if gcd(i, n) == 1:
            res += 1
    return res
def pow(x, y, p):# возведение в степень по модцлю
    res = 1
    x = x % p# приведение числа к значению по модулю
    while y > 0:
        if y % 2 == 1:
            res = (res * x) % p
        y = y // 2
        x = (x * x) % p
    return res
def discr_log(base, num, mod):#нахождение дискретного логарифма
    phi_value = eul(mod)
    for x in range(1, phi_value + 1):
        if pow(base, x, mod) == num:
            return x
    return None
#Пример
if __name__ == "__main__":
    base = 3  # Основание логарифма
    num = 57  # Число, для которого ищем логарифм
    mod = 89  # Модуль
    result = discr_log(base, num, mod)  # Вызываем функцию для нахождения дискретного логари
    if result is not None:
        print(f"Дискретный логарифм для {base}^{result} ≡ {num} (mod {mod})")  # Выводим ре
    else:
        print("Не удалось найти дискретный логарифм")  # Сообщаем, что логарифм не найден
```

```
Дискретный логарифм для 3^36 ≡ 57 (mod 89)

# Code execution complete.
```

Рисунок 8 – вычисление дискретных логарифмов при помощи исчисления порядка

Ответы: a) 36, b) 45, c) 15.