Разложение чисел на множители

Миличевич Александра

15 февраля, 2024, Москва,

Россия

Российский Университет Дружбы Народов

Цели и задачи

Цель лабораторной работы

Изучение задачи дискретного логарифмирования.

Выполнение лабораторной работы

Задача дискретного логарифмирования

Решение задачи дискретного логарифмирования состоит в нахождении некоторого целого неотрицательного числа x, удовлетворяющего уравнению. Если оно разрешимо, у него должно быть хотя бы одно натуральное решение, не превышающее порядок группы.

p-алгоритм Полларда

- Вход. Простое число p, число a порядка r по модулю p, целое число b б $1 < b < p$, отображение f, обладающее сжимающими свойствами и сохраняющее вычислимость логарифма.
- Выход. показатель x, для которого $a^x = b \pmod p$ если такой показатель существует.

1. Выбрать произвольные целые числа u, v и положить $c = a^u b^v \pmod p, d = c$
2. Выполнять $c=f(c) \pmod p$, $d=f(f(d)) \pmod p$, вычисляя при этом логарифмы для c и d как линейные функции от x по модулю r, до получения равенства $c = d \pmod p$
3. Приняв логарифмы для c и d, вычислить логарифм x решением сравнения по модулю r. Результат x или РЕШЕНИЯ НЕТ.

Функция `pollard_step(x, a, b, params

Эта функция реализует один шаг алгоритма Полларда для дискретного логарифмирования.

Вход:

`x` (int): Текущее значение `x`.

`a` (int): Текущее значение `a`.

`b` (int): Текущее значение `b`.

`params` (tuple): Параметры (G, H, P, Q).

Функция `pollard_step(x, a, b, params

Выход:

Кортеж обновленных значений `(x, a, b)`.

Как работает:

1. **Разделение на подмножества:** Использует `x % 3` для определения подмножества.

2. **Обновление значений в зависимости от подмножества:**

   * Если `x % 3 == 0`: `x` умножается на `G` по модулю `P`, `a` увеличивается на 1 по модулю `Q`.

   * Если `x % 3 == 1`: `x` умножается на `H` по модулю `P`, `b` увеличивается на 1 по модулю `Q`.

   * Если `x % 3 == 2`: `x` возводится в квадрат по модулю `P`, `a` и `b` умножаются на 2 по модулю `Q`

Пример работы алгоритма

```python
def pollard_step(x, a, b, params):
    """
    Шаг алгоритма Полларда для дискретного логарифмирования.

    Args:
        x (int): Текущее значение x.
        a (int): Текущее значение a.
        b (int): Текущее значение b.
        params (tuple): Параметры (G, H, P, Q).

    Returns:
        tuple: Обновленные значения x, a, b.
    """
    G, H, P, Q = params
    subset = x % 3 # Выбираем подмножество

    if subset == 0:
        x = (x * G) % P
        a = (a + 1) % Q

    if subset == 1:
        x = (x * H) % P
        b = (b + 1) % Q

    if subset == 2:
        x = (x * x) % P
        a = (a * 2) % Q
        b = (b * 2) % Q

    return x, a, b
```

Рис. 1: pollard_step

Функция `pollard_rho_discrete_log(generator, value, prime)

Вычисление дискретного логарифма:
Вычисляется числитель a - A и знаменатель B - b.
Вычисляется обратный элемент знаменателя по модулю Q с помощью функции modular_inverse.
Вычисляется дискретный логарифм: (inverse_denominator * numerator) % Q.

Код

```python
def pollard_rho_discrete_log(generator, value, prime):
    """
    Реализация алгоритма Полларда для дискретного логарифмирования.

    Args:
        generator (int): Генератор группы.
        value (int): Значение, для которого ищется дискретный логарифм.
        prime (int): Простое число (порядок группы).

    Returns:
        int: Дискретный логарифм (если найден) или сообщение об ошибке.
    """
    Q = (prime - 1) // 2 # Порядок подгруппы
    x = (generator * value) % prime # Начальное значение x
    a = 1 # Начальное значение a
    b = 1 # Начальное значение b

    X = x # Текущее значение X
    A = a # Текущее значение A
    B = b # Текущее значение B

    # Основной цикл поиска коллизии
    for i in range(1, prime):
        # Заяц
        x, a, b = pollard_step(x, a, b, (generator, value, prime, Q))

        # Черепаха
        X, A, B = pollard_step(X, A, B, (generator, value, prime, Q))
        X, A, B = pollard_step(X, A, B, (generator, value, prime, Q))

        # Если найдена коллизия, выходим из цикла
        if x == X:
            break

    numerator = a - A # Вычисляем числитель
    denominator = B - b # Вычисляем знаменатель

    try:
      # Вычисляем обратный элемент к знаменателю по модулю Q
      inverse_denominator = modular_inverse(denominator, Q)
    except:
        return "Не удалось найти обратный элемент"

    # Вычисляем дискретный логарифм
    result = (inverse_denominator * numerator) % Q

    if verify(generator, value, prime, result):
        return result
    else:
        return result + Q
```

Рис. 2: pollar_rho_descrete_log

Функция `modular_inverse(a, n)

Этот код вычисляет обратное значение числа a по модулю n с использованием расширенного алгоритма Евклида. Функция extended_euclidean(a, n) возвращает кортеж, из которого берется второй элемент, представляющий собой обратное значение. Если обратное значение существует, оно возвращается как результат работы функции

Код

```python
def modular_inverse(a, n):
    """
    Вычисляет обратное к 'a' по модулю 'n'.

    Args:
        a (int): Число, для которого ищется обратное.
        n (int): Модуль.

    Returns:
        int: Обратное к 'a' по модулю 'n'.
    """
    return extended_euclidean(a, n)[1]
```

Рис. 3: modular inverse

Функция `verify(generator, value, prime, x)

Этот код проверяет правильность вычисленного дискретного логарифма x для значения value по модулю prime с использованием генератора generator.
Функция pow(generator, x, prime) вычисляет значение генератора, возведенного в степень x по модулю prime, и сравнивает его с value. Если значения совпадают, функция возвращает True, что означает, что логарифм вычислен верно, иначе возвращается False.

Код

```python
def verify(generator, value, prime, x):
    """
    Проверяет правильность вычисленного дискретного логарифма.

    Args:
        generator (int): Генератор группы.
        value (int): Значение, для которого ищется дискретный логарифм.
        prime (int): Простое число (порядок группы).
        x (int): Вычисленный дискретный логарифм.

    Returns:
        bool: True, если логарифм верный, False в противном случае.
    """
    return pow(generator, x, prime) == value
```

Рис. 4: modular inverse

Выводы

Результаты выполнения лабораторной работы

Изучили задачу дискретного логарифмирования.