Отчёт по лабораторной работе 7

Супонина Анастасия Павловна

Содержание

Список иллюстраций

Список таблиц

Элементы списка иллюстраций не найдены.

Цель работы

Ознакомиться с дискретным логарифмированием и научиться выполнять его программно при помощи p-метод Полларда.

Задание

Реализовать p-метод Полларда для дискретного логарифмирования на языке программирования Julia.

Теоретическое введение

Дискретное логарифмирование в конечном поле

Задача дискретного логарифмирования, как и задача разложения на множители, применяется во многих алгоритмах криптографии с открытым ключом. Предложенная в 1976 году У. Диффи и М. Хеллманом для установления сеансового ключа, эта задача послужила основой для создания протоколов шифрования и цифровой подписи, доказательств с нулевым разглашением и других криптографических протоколов.

Пусть над некоторым множеством Ω произвольной природы определены операции сложения «+» и умножения «·». Множество Ω называется кольцом, если выполняются следующие условия:

1. Сложение коммутативно: $ a + b = b + a $ для любых $ a, b ∈ Ω $;

2. Сложение ассоциативно: $ (a + b) + c = a + (b + c)$ для любых $a, b, c \in \Omega$;

3. Существует нулевой элемент $0 \in \Omega$ такой, что $a + 0 = a$ для любого $a \in \Omega$;

4. Для каждого элемента $a ∈ Ω $ существует противоположный элемент $-a \in \Omega$, такой, что $(-a) + a = 0$;

5. Умножение дистрибутивно относительно сложения:

   $ a · (b + c) = a · b + a · c, (a + b) · c = a · c + b · c, $

   для любых $a, b, c \in \Omega$.

Если в кольце $ Ω $ умножение *коммутативно*: $ a · b = b · a $ для любых $a, b \in \Omega$, то кольцо называется коммутативным.

Если в кольце Ω умножение *ассоциативно*: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ для любых $a, b, c \in \Omega$, то кольцо называется ассоциативным.

Если в кольце Ω существует *единичный элемент* e такой, что $a · e = e · a = a $ для любого $a \in \Omega$, то кольцо называется кольцом с единицей.

Если в ассоциативном, коммутативном кольце Ω с единицей для каждого ненулевого элемента a существует обратный элемент $a^(-1) ∈ Ω $такой, что $a^{(-1)} \cdot a = e$, то кольцо называется **полем**.

Пусть $m \in N, m > 1$. Целые числа a и b называются сравнимыми по модулю m (обозначается $a \equiv b(mod m)$), если разность a – b делится на m. Некоторые свойства отношения сравнимости:

1. Рефлексивность:$ a ≡ a (mod m)$.

2. Симметричность:$ если a ≡ b (mod m), то b ≡ a (mod m)$.

3. Транзитивность:$ если a ≡ b (mod m) и b ≡ c (mod m), то a ≡ c (mod m)$.

Отношение, обладающее свойствами рефлексивности, симметричности и транзитивности, называется отношением эквивалентности. Отношение сравнимости является отношением эквивалентности на множестве Z целых чисел.

Отношение эквивалентности разбивает множество, на котором оно определено, на классы эквивалентности. Любые два класса эквивалентности либо не пересекаются, либо совпадают.

Классы эквивалентности, определяемые отношением сравнимости, называются классами вычетов по модулю m. Класс вычетов, содержащий число a, обозначается $a\big(mod(m)\big)$ или $\bar{a}$ и представляет собой множество чисел вида $a + km$, где $k \in Z$; число a называется представителем этого класса вычетов.

Множество классов вычетов по модулю m обозначается Z/mZ, состоит ровно из m элементов и относительно операций сложения и умножения является кольцом классов вычетов по модулю m.

Пример. Если $m = 2, \text{то} Z/2Z = 0(mod2), 1(mod2)$, $где $0(mod2) = 2Z$ - множество всех чётных чисел, $1(mod2) = 2Z + 1$ - множество всех нечётных чисел.

Обозначим $F_p = Z/2Z, p$ – простое целое число и назовём конечным полем с p элементами. Задача дискретного логарифмирования в конечном поле F_p формулируется так: для данных целых чисел $a и b, a > 1, b > p$, найти логарифм – такое целое число x, что $a^x \equiv b(modp)$ (если такое число существует). По аналогии с вещественными числами используется обозначение $x = log_a b$.

Безопасность соответствующих криптосистем основана на том, что, зная числа a, x, p, вычислить a^x (mod p) легко, а решить задачу дискретного логарифмирования трудно. Рассмотрим p–Метод Полларда, который можно применить и для задач дискретного логарифмирования. При этом случайное отображение f должно обладать не только сжимающими свойствами, но и вычислимостью логарифма (логарифм числа f(c) можно выразить через неизвестный логарифм x и log_a f(c)). Для дискретного логарифмирования в качестве случайного отображения f чаще всего используется ветвящееся отображение, например:

$$f(c) = \begin{cases} a^c, \text{при } c < p/2, \\ b^c, \text{при } c \geq p/2 , \end{cases}$$

При c < p/2 имеем $ log_a f(c) = log_a a^c + 1$, при $c ≥ p/2 – log_a f(c) = log_a b^c + x $.

Выполнение лабораторной работы

Алгоритм, реализующий p–Метод Полларда для задач дискретного логарифмирования.

Вход. Простое число p, число a порядка r по модулю p, целое число b, 1 < b < p; отображение f, обладающее сжимающими свойствами и сохраняющее вычислимость логарифма.

```
1   p = 107
2   a = 10
3   r = 53
4   b = 64
```

Входные данные

1. Выбрать произвольные целые числа u, v и положить c ← a^u b^v (mod p), d ← c.

Для c и для d значения u, v присваиваю разные для счета.

```
17  # Инициализация значений для c и d
18  u, v = 2, 2  # Для c
19  U, V = 2, 2  # Для d
20
21  c = a^u * b^v % p  # Начальное значение c
22  d = c  # Начальное значение d
```

Инициализация значений для c и d

2. Выполнять c ← f(c) (mod p), d ← f(f(d)) (mod p), вычисляя при этом логарифмы для c и d как линейные функции от x по модулю r, до получения равенства c ≡ d (mod p).

Для начала отдельно создаю функцию, которая будет считать отображение, а также увеличивать значения u, v на 1 на каждом шаге.

```
6   # Функция, определяющая преобразование
7   function otobr(t, z, w)
8       if t < r
9           z += 1
10          return mod(a * t, p), z, w
11      else
12          w += 1
13          return mod(b * t, p), z, w
14      end
15  end
```

Функция отображения

Первый раз применяю отображение отдельно, так как c = d, а в дальнейшнем это будет являться условие выхода из цикла. потом записываю основную функцию, которая применяет отображения пока не получит равные значения c и d. Таким образом ищу коллизии.

```
c, u, v = otobr(c, u, v)
d, U, V = otobr(d, U, V)
d, U, V = otobr(d, U, V)

println("Обновленное значение c: ", c)
println("Обновленное значение d: ", d)

# Функция для обнаружения коллизий
function second(c, d, u, v, U, V)
    while c != d
        c, u, v = otobr(c, u, v)  # Обновление для медленного указателя
        d, U, V = otobr(d, U, V)  # Первое обновление для быстрого указателя
        d, U, V = otobr(d, U, V)  # Второе обновление для быстрого указателя
        println("Текущее значение c: $c, d: $d")
    end
    return c, d, u, v, U, V
end

# Находим коллизию
c, d, u, v, U, V = second(c, d, u, v, U, V)
```

Обнаружение коллизий

3. Приравняв логарифмы для c и d, вычислить логарифм x решением сравнения по модулю r. Результат: x или “Решений нет”.

Создаю функцию для вычисления обратного элемента, чтобы потом вычислить значение x.

```
# Функция для вычисления обратного элемента по модулю
function invmod(a, m)
    g, x, _ = gcdx(a, m)
    if g != 1
        throw(ArgumentError("Обратного элемента не существует"))
    else
        return mod(x, m)
    end
end
```

обратное значение

И создаю функцию для вычисления логарифмов и нахождения x

```
# Функция для вычисления x из логарифмов
function compute_x(u, v, U, V, r)
    # Вычисляем разницу логарифмов
    delta_v = mod(v - V, r)  # Теперь Δv используется с x
    delta_u = mod(U - u, r)

    if delta_v == 0
        return "Решений нет"  # Если delta_v = 0, решения не существует
    end

    delta_v_inv = invmod(delta_v, r)

    # Вычисляем x
    x = mod(delta_u * delta_v_inv, r)  # Здесь x = Δu * (Δv)^-1 mod r
    return x
end
```

вычисление логарифмов

Выход. Показатель x, для которого a^x ≡ b (mod p), если такой показатель существует. Также для сравнения значений с табличными значениями данными для проверки в лабораторной работы, вывожу значения c и d на каждой иттерации, в результате чего, получаю следующий вывод в консоли.

```
Начальное значение c: 4
Начальное значение d: 4
Обновленное значение c: 40
Обновленное значение d: 79
Текущее значение c: 79, d: 56
Текущее значение c: 27, d: 75
Текущее значение c: 56, d: 3
Текущее значение c: 53, d: 86
Текущее значение c: 75, d: 42
Текущее значение c: 92, d: 23
Текущее значение c: 3, d: 53
Текущее значение c: 30, d: 92
Текущее значение c: 86, d: 30
Текущее значение c: 47, d: 47
Итоговое значение c: 47
Итоговое значение d: 47
Итоговое значение u: 7
Итоговое значение v: 8
Итоговое значение U: 13
Итоговое значение V: 13
Логарифм x: 20
```

Результат

Выводы

В процессе выполнения работы, я реализовала алгоритм p-Полларда для задач дискретного логарифмирования на языке программирования Julia.

Список литературы

::: Пособие по лабораторной работе 5 {file:///C:/Users/bermu/Downloads/lab05.pdf}