

Лабораторная работа №7

Дискретное логарифмирование в конечном поле. Метод Полларда

Исламова С.М.

Информация

Докладчик

- Исламова Сания Маратовна
- студент уч. группы НПИмд-01-24
- Российский университет дружбы народов
- 1132249576@pfur.ru
- <https://github.com/SaniyaIslamova26>



Вводная часть

Актуальность

- Реализация ρ -метода Полларда для дискретного логарифмирования
- Работа с большими числами в языке Julia
- Понимание вероятностных методов в криптографии
- Изучение задач, лежащих в основе современных криптографических систем
- Анализ сложности решения задачи дискретного логарифмирования

Объект и предмет исследования

- ρ -метод Полларда для дискретного логарифмирования
- Задача дискретного логарифмирования в конечных полях
- Конечные поля и их алгебраические свойства
- Метод “черепахи и зайца” для поиска коллизий
- Язык программирования Julia для реализации криптографических алгоритмов

Цели и задачи

- Реализовать ρ -метод Полларда для решения задачи дискретного логарифмирования
- Исследовать эффективность метода на различных входных данных
- Проанализировать поведение алгоритма в зависимости от параметров поля
- Изучить математические основы метода и его криптографическое значение
- Проверить корректность реализации на тестовых примерах

Теоретическая часть

Задача дискретного логарифмирования (Discrete Logarithm Problem, DLP)

Для заданных простого числа (p), основания (a) (образующего элемента мультиплексивной группы (\mathbb{Z}_p^*)) и числа (b) требуется найти целое число (x) такое, что:

$$[a^x \equiv b \pmod{p}]$$

где ($1 < a < p$), ($1 < b < p$).

Математические основы

Конечные поля

Множество классов вычетов по модулю простого числа (p) образует конечное поле ($\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z}$), которое обладает следующими свойствами:

- Сложение и умножение определены по модулю (p)
- Существует мультипликативная группа (\mathbb{Z}_p^*) порядка ($p-1$)
- Каждый ненулевой элемент имеет обратный по умножению

Отношение сравнимости

Для целых чисел (a, b) и модуля ($m > 1$): $[a \equiv b \pmod m]$

Отношение сравнимости является отношением эквивалентности и разбивает множество целых чисел на (m) классов вычетов.

р-Метод Полларда для дискретного логарифмирования

Основная идея

Метод основан на поиске коллизии в псевдослучайной последовательности, порождаемой итеративным применением специального отображения. При обнаружении коллизии (равенства двух элементов последовательности) возникает уравнение, из которого можно найти искомый логарифм.

Алгоритмические компоненты

- 1. Ветвящееся отображение:** $[f(c) =]$
- 2. Представление логарифмов:** Каждый элемент последовательности представляется в виде ($a^{\wedge}\{x\} b^{\wedge}\{y\} \pmod p$), где логарифм вычисляется как линейная функция ($+ x$).
- 3. Метод “черепахи и зайца” (Флойда):**
 - Инициализация двух последовательностей с одинаковым начальным значением
 - Одна последовательность (“черепаха”) делает по одному шагу на каждой итерации
 - Другая последовательность (“заяц”) делает по два шага на каждой итерации
 - Когда эти последовательности встречаются, обнаружена коллизия

Математическое обоснование

Пусть: - ($c_i = a^{i_1} b^{i_2} \dots p$) - ($d_i = a^{i_1} b^{i_2} \dots p$)

При коллизии ($c_k = d_m$) имеем: [$a^{k_1} b^{k_2} \dots a^{m_1} b^{m_2} \dots p$]

Подставляя ($b^{a^x} p$), получаем: [$a^{k_1 + m_1} b^{k_2 + m_2} \dots p$]

Отсюда: [$k_1 + m_1 \equiv k_2 + m_2 \pmod{r}$] где (r) - порядок элемента (a).

Решая это линейное сравнение: [$(k_1 - m_1)x \equiv (m_1 - k_2) \pmod{r}$]

Вычислительная сложность

- Среднее время работы: ($O()$), где (r) - порядок элемента (a)
- Память: ($O(1)$) (алгоритм требует хранения только нескольких переменных)
- Вероятностный характер: гарантирует нахождение решения с высокой вероятностью

Криптографическое значение

- Задача дискретного логарифмирования является основой многих криптографических протоколов (Диффи-Хеллмана, Эль-Гамаля, цифровых подписей)
- Стойкость этих систем зависит от вычислительной сложности решения DLP
- ρ -метод Полларда является одним из наиболее эффективных алгоритмов общего назначения для решения DLP
- Понимание метода важно для оценки стойкости криптографических систем и разработки криptoаналитических инструментов

Особенности реализации на языке Julia

1. **Типы данных:** Использование `BigInt` для работы с большими числами
2. **Модульная арифметика:** Функции `mod()` и `powermod()` для эффективных вычислений
3. **Алгоритмы теории чисел:** Функция `gcdx()` для реализации расширенного алгоритма Евклида
4. **Генерация случайных чисел:** Функция `rand()` для инициализации параметров алгоритма

Практическая реализация

Реализация на языке программирования Julia разложение чисел на множители: ρ -алгоритма Полларда для разложения чисел на множители

```
# Заголовок программы
println("ρ-метод Полларда для дискретного логарифмирования")
# Бесконечный цикл для многократного использования программы
while true
    # Запрос ввода данных от пользователя
    println("\nВведите p a b r через пробел (или 'выход' для завершения):")
    # Чтение введенной строки с клавиатуры
    input = readline()
    # Проверка команды выхода из программы
    input == "выход" && break
    # Блок обработки ошибок ввода
    try
        # Разделение строки на части и преобразование в BigInt
        p,a,b,r = parse.(BigInt, split(input))
        # Определение функции ρ-метода Полларда
        function ρ(p,a,b,r)
            # Инициализация: случайные u, v из [0, r-1]
            u,v = rand(0:r-1,2)
            # Вычисление начальной точки c = a^u * b^v mod p
            c = powermod(a,u,p)*powermod(b,v,p)%p
            # Черепаха и заяц: начальные точки одинаковы
            d = c
            # Инициализация логарифмов: Log(c) = u + v*x, Log(d) = u + v*x
            α1,β1,α2,β2 = u,v,u,v
            # Цикл поиска коллизии (метод Флойда)
            while (c = (c<p÷2 ? a*c : b*c)%p) != d
                # Обновление d (два шага)
                (d = (d<p÷2 ? a*d : b*d)%p; d = (d<p÷2 ? a*d : b*d)%p)
                # Обновление логарифмов для c (один шаг)
                α1,β1 = (α1+(c<p÷2))%r, (β1+(c≥p÷2))%r
                # Обновление логарифмов для d (два шага)
                α2,β2 = (α2+2(d<p÷2))%r, (β2+2(d≥p÷2))%r
            end
        end
    end
end
```

```

end
# После нахождения коллизии: решение уравнения
# Находим коэффициенты  $(\beta_1 - \beta_2) * x \equiv (\alpha_2 - \alpha_1) \pmod{r}$ 
g,x,_ = gcdx((β1-β2)%r, r)
# Проверка разрешимости и возврат решения
(Δα=(α2-α1)%r)%g ≠ 0 ? nothing : (x*Δα÷g)%r
end
# Вызов функции и получение результата
x = p(p,a,b,r)
# Вывод результата
println(x==nothing ? "Нет решений" : "x = $x")
# Обработка ошибок при некорректном вводе
catch
    println("Ошибка: введите 4 числа или 'выход'")
end
end
# Сообщение о завершении программы
println("Программа завершена")

```

```

Lab07.jl

C:\> Users > detal > Downloads > ▲ Lab07.jl > ...
1 #Лабораторная работа #7
2 #Название: Дискретное логарифмирование в конечном поле
3 #Автор: Смирнова Елена
4 #Группа: ИМиР-01-24
5
6 # Логика программы
7 println("Метод Полларда для дискретного логарифмирования")
8 # бесконечный цикл для многократного использования программы
9 while true
10     # Вводим числа изначально от пользователя
11     println("Введите a в б в через пробел или 'выход' для завершения:")
12     # Чтение введенной строки с клавиатуры
13     input = readline()
14     # Проверка введенного ввода на программы
15     input == "выход" && break
16     # блок обработки ошибок ввода
17     try
18         # Разделение строки на части и преобразование в BigInt
19         p,a,b,r = parse.(BigInt, split(input))
20         # Вызов функции вычисления логарифма по методу Полларда
21         function [B,a,b,r]
22             # Инициализация случайных u, v из из [0, n-1]
23             U,V = rand(0:n-1)
24             # Несколько случайных точек с альфа ^ k mod p
25             c = powmod(u,p)*powmod(b,v,p)%r
26             # Чертежка в звезду: начальные точки одинаковы
27             B1,B2 = c
28             # Инициализация логарифма: log(c) = u + v*alpha, log(d) = u + v*alpha
29             B1,B1,D,B2 = u,v,u,v
30             # Вычисление d (все модуль)
31             while c != (c*p+2)%r || d < (d*p+2)%r || B1*D %r != B2*D %r
32                 # Обновление d (дело шаг)
33                 d = (d*p+2)%r
34                 # Вычисление логарифма для d (дело шаг)
35                 B1,D = (B1*(c*p+2)%r, (B1*(c*p+2)%r)^p)
36                 # Обновление логарифма для d (дело шаг)
37                 B2,D = (B2*(c*p+2)%r, (B2*(c*p+2)%r)^p)
38             end
39             # После нахождения коллизии: решение уравнения
40             x = (B1-D)%r
41             g,x,_ = gcdx((B1-D)%r, (B2-D)%r * (x-D))%r
42             # Проверка разрешимости и возврат решения
43             (Δα=(B1-D)%r)%g ≠ 0 ? nothing : (x*Δα÷g)%r
44         end
45         # Вывод функции и получение результата
46         x = B(p,a,b,r)
47         # Вывод результата
48         println(x==nothing ? "Нет решений" : "x = $x")
49         # Обработка ошибок при некорректном вводе
50     catch
51         println("Ошибка: введите 4 числа или 'выход'")
52     end
53 end
54 # Сообщение о завершении программы
55 println("Программа завершена")
56
57

```

The screenshot shows a terminal window with the following text:

```
ПРОБЛЕМЫ ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ

ρ-метод Полларда для дискретного логарифмирования

Введите p a b r через пробел (или 'выход' для завершения):
julia> 1 2 3 4
1 2 3 4
x = 0

Введите p a b r через пробел (или 'выход' для завершения):
45 67 102 15
x = -2

Введите p a b r через пробел (или 'выход' для завершения):
ВЫХОД
Ошибка: введите 4 числа или 'выход'
Введите p a b r через пробел (или 'выход' для завершения):
выход
Программа завершена
```

Результаты

- Выполнены все необходимые действия для реализации задач лабораторной работы №7: успешно реализовано на языке программирования Julia дискретное логарифмирование в конечном поле: ρ-алгоритма Полларда для задач дискретного логарифмирования. ## Вывод

Реализовано на языке программирования Julia дискретное логарифмирование в конечном поле: ρ-алгоритма Полларда для задач дискретного логарифмирования.