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 мультипликативной группы (_p^*)) и числа (b) требуется найти целое число (x) такое, что:

[a^x b (p)]

где (1 < a < p), (1 < b < p).

Математические основы

Конечные поля

Множество классов вычетов по модулю простого числа (p) образует конечное поле (_p = /p), которое обладает следующими свойствами: - Сложение и умножение определены по модулю (p) - Существует мультипликативная группа (_p^*) порядка (p-1) - Каждый ненулевой элемент имеет обратный по умножению

Отношение сравнимости

Для целых чисел (a, b) и модуля (m > 1): [a b (m) m (a - b)]

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

ρ-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^{} b^{} (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} b^{_i} (p)) - (d_i = a^{_i} b^{_i} (p))

При коллизии (c_k = d_m) имеем: [a^{_k} b^{_k} a^{_m} b^{_m} (p)]

Подставляя (b a^x (p)), получаем: [a^{_k + _k x} a^{_m + _m x} (p)]

Отсюда: [_k + _k x _m + _m x (r)] где (r) - порядок элемента (a).

Решая это линейное сравнение: [(_k - _m)x (_m - _k) (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 криптоаналитических инструментов

Особенности реализации на языке Julia

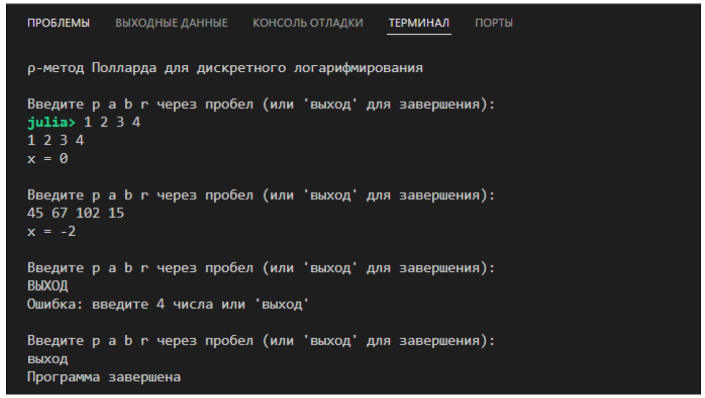
1. **Типы данных**: Использование `BigInt` для работы с большими числами
2. **Модульная арифметика**: Функции `mod()` и `powermod()` для эффективных вычислений
3. **Алгоритмы теории чисел**: Функция `gcdx()` для реализации расширенного алгоритма Евклида
4. **Генерация случайных чисел**: Функция `rand()` 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) !=
                  # 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
            # После нахождения коллизии: решение уравнения
            # Находим коэффициенты (β1-β2)*x ≡ (α2-α1) (mod 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 = ρ(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")
```



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.