# Лабораторная работа №7: отчет.

**Дискретное логарифмирование в конечном поле**

Евдокимов Максим Михайлович. Группа - НФИмд-01-24.

# Содержание

# Список иллюстраций

# Цели и задачи работы

## Цель лабораторной работы

Вычисление дискретных логарифмов в конечном поле.

## Задание

Реализовать алгоритм, реализующий p-Метод Полларда для задач дискретного логарифмирования программно.

# Теоретическое введение

## алгоритм ρ-Метод Полларда для задач дискретного логарифмирования

ρ-Метод Полларда (Pollard's rho method for discrete logarithms) — это алгоритм, используемый для решения задачи дискретного логарифмирования в конечных полях. Задача дискретного логарифмирования заключается в нахождении целого числа $x$ такого, что:

$$g^x \equiv h \pmod{p}$$

где $g$, $h$, и $p$ — известные целые числа, а $p$ — простое число.

### Основные шаги алгоритма

1. **Разделение последовательности**:
   - Последовательность $\{g^0, g^1, g^2, ..., g^{p-1}\}$ разбивается на три подмножества $S_0$, $S_1$, и $S_2$ на основе некоторого правила. Например, можно использовать остаток от деления на 3.
2. **Функция перехода**:
   - Определяется функция перехода $f(x)$, которая перемещает элементы между подмножествами. Обычно используется следующая функция:

$$f(x) = \begin{cases} x \cdot g \pmod p & \text{если } x \in S_0, \\ x \cdot h \pmod p & \text{если } x \in S_1, \\ x^2 \pmod p & \text{если } x \in S_2. \end{cases}$$

3. **Поиск коллизии**:

   - Используем метод “черепахи и зайца” для поиска коллизии.

4. **Решение уравнения**:

   - Пусть найдены $i = 5$ и $j = 10$. Тогда:

     $$5^5 \equiv 3 \pmod 7$$

     Проверяем:

     $$5^5 = 3125 \equiv 3 \pmod 7$$

     Таким образом, $x = 5$ является решением.

## Применение алгоритма

Алгоритм р-Метода Полларда используется для решения задач дискретного логарифмирования в криптографии, где эта задача играет важную роль. Например, в системах шифрования на основе эллиптических кривых и в протоколах обмена ключами Диффи-Хеллмана.

## Преимущества и недостатки

**Преимущества**: - **Эффективность**: Алгоритм работает быстрее, чем полный перебор, особенно для больших $p$. - **Простота реализации**: Алгоритм относительно прост в реализации.

**Недостатки**: - **Не гарантирует успех**: Алгоритм может не найти решение, если не найдена коллизия. - **Зависимость от параметров**: Эффективность алгоритма зависит от выбора параметров $S_0$, $S_1$, и $S_2$.

## Заключение

Алгоритм р-Метода Полларда — это мощный инструмент для решения задач дискретного логарифмирования в конечных полях. Он использует метод "черепахи и зайца" для поиска коллизий и позволяет эффективно находить решения в криптографических задачах.

# Ход работы

## Расширенный алгоритм Евклида для нахождения обратного элемента

```julia
# Расширенный алгоритм Евклида для нахождения обратного элемента
function gcd_extended(a, b)
    if a == 0
        return b, 0, 1
    end
    gcd, x1, y1 = gcd_extended(b % a, a)
    x = y1 - (b ÷ a) * x1
    y = x1
    return gcd, x, y
end
```

```
gcd_extended (generic function with 1 method)
```

Рис. 1: Расширенный алгоритм Евклида

## Функция для вычисления следующего элемента в последовательности

```julia
# Функция для вычисления следующего элемента в последовательности
function discrete_log_pollard_rho(g, h, p)
    function next_element(x, a, b)
        if x < p // 3
            return (g * x) % p, (a + 1) % (p - 1), b
        elseif x < 2 * p // 3
            return (x * x) % p, (2 * a) % (p - 1), (2 * b) % (p - 1)
        else
            return (h * x) % p, a, (b + 1) % (p - 1)
        end
    end

    # Инициализация
    x1, a1, b1 = 1, 0, 0
    x2, a2, b2 = next_element(x1, a1, b1)
    # Нахождение цикла
    while x1 != x2
        x1, a1, b1 = next_element(x1, a1, b1)
        x2, a2, b2 = next_element(next_element(x2, a2, b2)...)
    end

```

Рис. 2: Функция для вычисления часть 1

```
    # Решаем уравнение
    r = (b2 - b1) % (p - 1)
    if r == 0
        return "Решение не найдено"
    end

    d, x, y = gcd_extended(r, p - 1)
    if d != 1
        return "Решение не найдено"
    end

    l = ((a1 - a2) * x % (p - 1) + (p - 1)) % (p - 1)
    return l
end
```

```
discrete_log_pollard_rho (generic function with 1 method)
```

Рис. 3: Функция для вычисления часть 2

## Тестовые значение

```julia
# Тестовые значение
using Primes
n = 10
p = map(x -> primes(50)[rand(1:15)], 1:n) # модуль в конечном поле F_p, где F*_p состоит из всех целых чисел от 1 до p-1
a = map(x -> rand(2:5), p) # генератор, примитивный элемент при возведении в степень по модулю p может порождать все элементы группы F*_p
b = map(x -> rand(1:x-1), p) # это элемент группы F*_p для которого мы хотим найти дискретный логарифм
x = fill(0, n)
println("p: ", p, "\na: ", a, "\nb: ", b)
```

```
p: [7, 47, 41, 17, 23, 37, 47, 31, 7, 7]
a: [5, 2, 3, 2, 3, 5, 2, 5, 4, 2]
b: [3, 36, 29, 6, 8, 36, 29, 28, 3, 5]
```

Рис. 4: Стартовые значения и переменные

# Результаты и тесты

```
# Вызов и вывод результатов
for i in 1:n
    result = discrete_log_pollard_rho(a[i], b[i], p[i])
    x[i] = typeof(result) == String ? -1 : result
    println("При p = ", p[i], ", a = ", a[i], ", b = ", b[i], "; дискретный логарифм x = ", result)
end
print(x)
```

```
При p = 7, a = 5, b = 3; дискретный логарифм x = 5
При p = 47, a = 2, b = 36; дискретный логарифм x = 17
При p = 41, a = 3, b = 29; дискретный логарифм x = Решение не найдено
При p = 17, a = 2, b = 6; дискретный логарифм x = Решение не найдено
При p = 23, a = 3, b = 8; дискретный логарифм x = 21
При p = 37, a = 5, b = 36; дискретный логарифм x = Решение не найдено
При p = 47, a = 2, b = 29; дискретный логарифм x = Решение не найдено
При p = 31, a = 5, b = 28; дискретный логарифм x = Решение не найдено
При p = 7, a = 4, b = 3; дискретный логарифм x = Решение не найдено
При p = 7, a = 2, b = 5; дискретный логарифм x = Решение не найдено
[5, 17, -1, -1, 21, -1, -1, -1, -1, -1]
```

Рис. 5: Вызов и вывод результатов

```julia
# проверка по a^x = b (mod p) по Алгоритму Гельфонда - Шенкса
function analysisX(g, h, p)
    res = nothing
    for x in 1:50
        aX = g^x
        divA = div(aX, p)
        differ = aX - (divA * p)
        if differ == b
            res = x
            break
        end
    end
    return res
end

for i in 1:n
    if x[i] == -1
        an = analysisX(p[i], a[i], b[i])
        println("Для случая ", i, " значение x = ", an == nothing ? " нет на интервале [1,50]" : an)
    else
        aX = a[i]^x[i]
        divA = div(aX, p[i])
        differ = aX - (divA * p[i])
        println("Для случая ", i, " значение x = ", x[i], differ == b[i] ? " (точно правильно)" : " (неправильно)")
    end
end
```

Рис. 6: Проверка по a^x = b (mod p)


Для случая 1 значение x = 5 (точно правильно)
Для случая 2 значение x = 17 (точно правильно)
Для случая 3 значение x =  нет на интервале [1,50]
Для случая 4 значение x =  нет на интервале [1,50]
Для случая 5 значение x = 21 (точно правильно)
Для случая 6 значение x =  нет на интервале [1,50]
Для случая 7 значение x =  нет на интервале [1,50]
Для случая 8 значение x =  нет на интервале [1,50]
Для случая 9 значение x =  нет на интервале [1,50]
Для случая 10 значение x =  нет на интервале [1,50]

Рис. 7: Результаты проверки

# Выводы по проделанной работе

## Вывод

В ходе выполнение лабораторной работы был изучен и реализован способ определения дискретного логарифма для дискретного логарифмирования в конечном поле с использованием алгоритм, реализующий р-Метод Полларда для задач дискретного логарифмирования.

# Список литературы

1. Параллельный метод Полларда решения задачи дискретного логарифмирования с использованием детерминированной функции разбиения на множества
2. Поиск дискретного логарифма 2015 Сергей Николенко
3. Доступно о криптографии на эллиптических кривых