In [3]:

```
using Printf

# Простая проверка простоты
function isprime_simple(n)
    n < 2 && return false
    for i in 2:min(n-1, 1000)
        n % i == 0 && return false
    end
    return true
end

# Алгоритм Полларда
function pollard_rho_dlog_simple(p, a, b)
    r = p - 1  # порядок

    # Функция f
    function f(c, alpha, beta)
        if c < p ÷ 2
            return (mod(c * a, p), mod(alpha + 1, r), beta)
        else
            return (mod(c * b, p), alpha, mod(beta + 1, r))
        end
    end

    # Инициализация
    u = rand(0:r-1)
    v = rand(0:r-1)

    c = mod(powermod(a, u, p) * powermod(b, v, p), p)
    d = c
    alpha_c, beta_c = u, v
    alpha_d, beta_d = u, v

    # Поиск коллизии
    for i in 1:10000
        c, alpha_c, beta_c = f(c, alpha_c, beta_c)
        d, alpha_d, beta_d = f(d, alpha_d, beta_d)
        d, alpha_d, beta_d = f(d, alpha_d, beta_d)

        if c == d
            A = mod(beta_c - beta_d, r)
            B = mod(alpha_d - alpha_c, r)

            # Решаем A*x ≡ B (mod r)
            g = gcd(A, r)
            if B % g != 0
                return nothing
            end

            A_div = div(A, g)
            B_div = div(B, g)
            r_div = div(r, g)
```

```
            inv = invmod(A_div, r_div)
            x0 = mod(B_div * inv, r_div)

            for k in 0:g-1
                x = mod(x0 + k * r_div, r)
                if powermod(a, x, p) == b
                    return x
                end
            end
        end
    end
    return nothing
end

# Запуск программы
println("="^50)
println("Лабораторная работа 7: Дискретное логарифмирование")
println("="^50)

# Пример из задания
println("\nПример: 10^x ≡ 64 (mod 107)")
x = pollard_rho_dlog_simple(107, 10, 64)
if x !== nothing
    println("Найдено x = $x")
    println("Проверка: 10^$x mod 107 = $(powermod(10, x, 107))")
else
    println("Решение не найдено")
end

# Ввод своих значений
println("\nВведите свои значения p a b:")
print("p = ")
p = parse(Int, readline())
print("a = ")
a = parse(Int, readline())
print("b = ")
b = parse(Int, readline())

x_result = pollard_rho_dlog_simple(p, a, b)
if x_result !== nothing
    println("Решение: x = $x_result")
    println("Проверка: $a^$x_result mod $p = $(powermod(a, x_result, p))")
else
    println("Решение не найдено")
end
```

```
================================================
Лабораторная работа 7: Дискретное логарифмирование
================================================

Пример: 10^x ≡ 64 (mod 107)
Найдено x = 20
Проверка: 10^20 mod 107 = 64

Введите свои значения p a b:
p = stdin> 107
a = stdin> 10
b = stdin> 64
Решение: x = 20
Проверка: 10^20 mod 107 = 64
```

In [ ]:

Лабораторная работа №7: Дискретное логарифмирование в конечном поле

Реализация p-метода Полларда

Импортируем необходимые модули

using Printf using Random

""" pollard_rho_dlog(p, a, b; max_iter=10000)

Алгоритм Полларда для дискретного логарифмирования в конечном поле GF(p).

Аргументы:

- `p` : простое число (модуль)
- `a` : основание (генератор)
- `b` : значение, для которого ищем логарифм
- `max_iter` : максимальное число итераций

Возвращает:

- `x` : решение сравнения a^x ≡ b (mod p)

- `nothing`: если решение не найдено """ function pollard_rho_dlog(p, a, b; max_iter=10000)

Проверяем простоту p (упрощённая проверка)

!isprime(p) && error("p должно быть простым числом")

В конечном поле GF(p) порядок мультипликативной группы равен p-1

Для a, являющегося генератором, порядок также равен p-1

r = p - 1

""" Ветвящееся отображение f(c) Согласно алгоритму из лабораторной работы:

  - При c < p/2: умножаем на основание a
  - При c ≥ p/2: умножаем на b

Также отслеживаем логарифмы как линейные функции: log(c) = alpha + beta*x* """ *function f(c, alpha, beta) if c < p ÷ 2 # c → a*c mod p # При умножении на a логарифм увеличивается на 1: log(a*c) = log(c) + 1 return (mod(c * a, p), mod(alpha + 1, r), beta) else # c → b*c mod p
При умножении на b логарифм увеличивается на x: log(b*c) = log(c) + x # Поскольку b = a^x, то log(b) = x return (mod(c * b, p), alpha, mod(beta + 1, r)) end end

Шаг 1: Инициализация

Выбираем случайные начальные значения для u и v

u = rand(0:r-1) v = rand(0:r-1)

Вычисляем начальное значение c = a^u * b^v mod p

c = mod(powermod(a, u, p) * powermod(b, v, p), p) d = c # Начинаем с того же значения

Инициализируем коэффициенты для логарифмов:

log_a(c) = alpha_c + beta_c * x

log_a(d) = alpha_d + beta_d * x

alpha_c, beta_c = u, v alpha_d, beta_d = u, v

Шаг 2: Поиск коллизии методом "черепахи и зайца"

c делает один шаг на каждой итерации, d - два шага

for i in 1:max_iter # Обновляем c (один шаг - "черепаха") c, alpha_c, beta_c = f(c, alpha_c, beta_c)

```
  # Обновляем d (два шага - "заяц")
```

```
  d, alpha_d, beta_d = f(d, alpha_d, beta_d)
  d, alpha_d, beta_d = f(d, alpha_d, beta_d)

  # Проверяем коллизию (c == d mod p)
  if c == d
      # Шаг 3: Решение сравнения

      # Из равенства c = d получаем:
      # alpha_c + beta_c * x ≡ alpha_d + beta_d * x (mod r)
      # Преобразуем к виду: A*x ≡ B (mod r)
      A = mod(beta_c - beta_d, r)
      B = mod(alpha_d - alpha_c, r)

      # Находим НОД(A, r) для проверки разрешимости
      g = gcd(A, r)

      # Условие разрешимости: B должно делиться на g
      if B % g != 0
          return nothing  # Нет решений
      end

      # Приводим сравнение к виду: (A/g)*x ≡ (B/g) (mod r/g)
      A_div = div(A, g)
      B_div = div(B, g)
      r_div = div(r, g)

      # Находим обратный элемент к A_div по модулю r_div
      inv = invmod(A_div, r_div)

      # Находим частное решение
      x0 = mod(B_div * inv, r_div)

      # Проверяем все возможные решения: x = x0 + k*(r/g), k
= 0..g-1
      for k in 0:g-1
          x = mod(x0 + k * r_div, r)
          # Проверяем, является ли x решением
          if powermod(a, x, p) == b
              return x
          end
      end

      return nothing  # Ни одно из возможных решений не
подошло
  end
end

return nothing # Превышено максимальное число итераций end
```

""" solve_dlog_example()

Решает пример из лабораторной работы: 10^x ≡ 64 (mod 107) Демонстрирует работу алгоритма на конкретном примере """ function solve_dlog_example() p = 107 a = 10 b = 64

```
    println("Решаем задачу: $a^x ≡ $b (mod $p)")
    println("Из лабораторной работы ожидаем x = 20")

    x = pollard_rho_dlog(p, a, b)

    if x !== nothing
        println("Найдено решение: x = $x")
        println("Проверка: $a^$x mod $p = $(powermod(a, x, p))")

        if powermod(a, x, p) == b
            println("✓ Проверка пройдена успешно!")
        else
            println("✗ Проверка не пройдена!")
        end
    else
        println("Решение не найдено")
    end

end
```

""" get_random_problem(bit_size=8)

Генерирует случайную задачу дискретного логарифмирования. Используется для тестирования алгоритма.

Аргументы:

- `bit_size` : размер простого числа в битах

Возвращает:

- `p` : простое число
- `a` : генератор поля
- `b` : a^x mod p
- `true_x` : истинное значение x (для проверки) """ function get_random_problem(bit_size=8)

Находим простое число заданного размера

p = nextprime(rand(2^(bit_size-1):2^bit_size))

Находим первообразный корень по модулю p (генератор мультипликативной группы)

Генератор - это элемент порядка p-1

a = 2 # Начинаем проверку с 2 while true # Проверяем, что a взаимно просто с p и имеет максимальный порядок if gcd(a, p) == 1 && order_mod(a, p) == p-1 break end a += 1 end

Выбираем случайный показатель x

x = rand(1:p-2)

Вычисляем b = a^x mod p

b = powermod(a, x, p)

return p, a, b, x end

""" order_mod(a, p)

Вычисляет порядок элемента a в мультипликативной группе по модулю p. Порядок - наименьшее натуральное r такое, что a^r ≡ 1 (mod p).

Аргументы:

- `a`: элемент группы

- `p` : модуль

Возвращает:

- `r` : порядок элемента a """ function order_mod(a, p) for r in 1:p-1 if powermod(a, r, p) == 1 return r end end return p-1 end

""" isprime(n)

Упрощённая проверка числа на простоту. В реальных приложениях следует использовать более эффективные алгоритмы.

Аргументы:

- `n` : проверяемое число

Возвращает:

- `true` : если число простое
- `false` : если число составное """ function isprime(n) n < 2 && return false n == 2 && return true n % 2 == 0 && return false

Проверяем делители до квадратного корня

limit = trunc(Int, sqrt(n)) for i in 3:2:min(limit, 10000) n % i == 0 && return false end return true end

""" nextprime(n)

Находит следующее простое число после n.

Аргументы:

- `n` : начальное число

Возвращает:

- `p` : первое простое число ≥ n """ function nextprime(n) n = n < 2 ? 2 : n n % 2 == 0 && (n += 1) while !isprime(n) n += 2 end return n end

""" main()

Основная функция программы. Организует взаимодействие с пользователем

и демонстрацию алгоритма. """ function main() println("="^60) println("ЛАБОРАТОРНАЯ РАБОТА №7") println("Дискретное логарифмирование в конечном поле") println("ρ-метод Полларда") println("="^60)

```
    # Демонстрация работы на примере из лабораторной работы
    println("\n" * "="^40)
    println("1. ПРИМЕР ИЗ ЛАБОРАТОРНОЙ РАБОТЫ")
    println("="^40)
    solve_dlog_example()

    # Генерация и решение случайной задачи
    println("\n" * "="^40)
    println("2. СЛУЧАЙНАЯ ЗАДАЧА ДЛЯ ТЕСТИРОВАНИЯ")
    println("="^40)
    p, a, b, true_x = get_random_problem(8)
    println("Сгенерирована задача:")
    println("  p = $p (простое число)")
    println("  a = $a (генератор поля)")
    println("  b = $b")
    println("  Истинное значение x = $true_x")

    println("\nЗапуск алгоритма Полларда...")
    x = pollard_rho_dlog(p, a, b)

    if x !== nothing
        println("\nРезультат:")
        println("  Найденное решение: x = $x")
        println("  Проверка: $a^$x mod $p = $(powermod(a, x, p))")

        if x == true_x
            println("  ✓ Найденное решение совпадает с истинным!")
        else
            # Проверяем, является ли найденное x тоже решением
            if powermod(a, x, p) == b
                println("  ⚠ Найдено другое решение (задача может
    иметь несколько решений)")
            else
                println("  ✗ Ошибка: найденное x не является
    решением")
            end
        end
    else
        println("\nРешение не найдено")
    end

    # Решение задачи от преподавателя
    println("\n" * "="^40)
    println("3. РЕШЕНИЕ ЗАДАЧИ ОТ ПРЕПОДАВАТЕЛЯ")
```

```
println("="^40)
println("Введите параметры задачи в формате: p a b")
println("Пример: 107 10 64")
println("Для выхода введите 'exit'")
println("-"^40)

while true
    print("\n> ")
    input = readline()

    if lowercase(input) == "exit"
        println("Выход из программы...")
        break
    end

    try
        values = split(input)
        if length(values) != 3
            println("Ошибка: нужно ввести ровно 3 числа")
            continue
        end

        p_input = parse(Int, values[1])
        a_input = parse(Int, values[2])
        b_input = parse(Int, values[3])

        println("\nЗадача: $a_input^x ≡ $b_input (mod
$p_input)")
        println("Запуск алгоритма...")

        # Замеряем время выполнения
        start_time = time()
        x_result = pollard_rho_dlog(p_input, a_input, b_input)
        end_time = time()

        if x_result !== nothing
            println("\n✓ Решение найдено!")
            println("  x = $x_result")
            println("  Проверка: $a_input^$x_result mod
$p_input = $(powermod(a_input, x_result, p_input))")
            println("  Время выполнения: $(round(end_time -
start_time, digits=4)) секунд")
        else
            println("\n✗ Решение не найдено")
            println("Возможные причины:")
            println("  1. Задача не имеет решения")
            println("  2. Превышено максимальное число
итераций")
```

```
                println("  3. p не является простым числом")
                println("  4. a не является генератором поля")
            end

            println("\n" * "-"^40)
            println("Введите следующую задачу или 'exit' для
    выхода")

        catch e
            println("Ошибка ввода: $e")
            println("Пожалуйста, введите три целых числа через
    пробел")
        end
    end

end
```

==========

ЗАПУСК ПРОГРАММЫ

==========

if abspath(PROGRAM_FILE) == @**FILE** println("Начало выполнения лабораторной работы №7") println("Версия Julia: $(VERSION)") println("Дата и время запуска: $(now())") println()

```
    main()

    println("\n" * "="^60)
    println("Лабораторная работа завершена успешно!")
    println("="^60)

end
```

In [ ]: