

# Лабораторная работа №6

Математические основы защиты информации и информационной безопасности

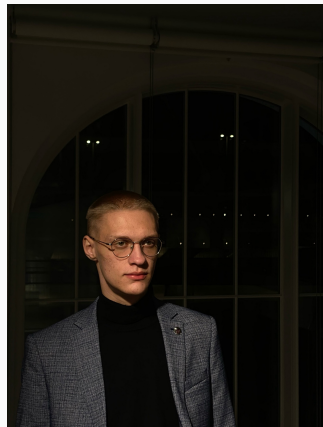
---

Махорин И. С.

2025

Российский университет дружбы народов имени Патриса Лумумбы, Москва, Россия

- Махорин Иван Сергеевич
- Студент группы НФИмд-02-25
- Студ. билет 1032259380
- Российский университет дружбы народов имени Патриса Лумумбы



- Изучить алгоритм разложения чисел на множители и научиться его реализовывать.

## Выполнение лабораторной работы

---

# Реализация р-Метода Полларда

```
# Алгоритм, реализующий р-метод Полларда

# Импорт необходимых модулей Julia
using Random # Для генерации случайных чисел
using Printf  # Для форматированного вывода

# Безопасное умножение по модулю (для больших чисел)
function modmul(a::Integer, b::Integer, m::Integer)
    # Используем BigInt чтобы избежать переполнения при умножении больших чисел
    return (BigInt(a) * BigInt(b)) % BigInt(m)
end

# Быстрое возведение в степень по модулю
function modpow(a::Integer, e::Integer, m::Integer)
    # Используем встроенную оптимизированную функцию powermod
    return powermod(BigInt(a), BigInt(e), BigInt(m))
end

# Простой тест Миллера-Рабина на простоту
function miller_rabin(n::Integer, k::Int=5)
    # Проверка крайних случаев
    if n < 2
        return false
    end
    # проверка малых простых чисел для ускорения
    small_primes = [2,3,5,7,11,13,17,19,23,29]
    for p in small_primes
        if n == p
            return true
        end
    end
```

Рис. 1: Реализация р-Метода Полларда

# Реализация р-Метода Полларда

```
# Если n делится на маленькое простое - составное
if n % p == 0
    return false
end

end

# представим  $n-1 = 2^r * d$ , где d нечётное
d = n - 1
r = 0

# Выделяем степени двойки из n-1
while d % 2 == 0
    d = d / 2 # Целочисленное деление
    r = r + 1
end

# повторяем проверку k раз для увеличения точности
for _ in 1:k
    a = rand(2:n-2) # случайное основание от 2 до n-2
    x = modpow(a, d, n) #  $a^d \bmod n$ 
    # Если  $x = \pm 1 \bmod n$ , то n вероятно простое
    if x == 1 || x == n-1
        continue
    end
    passed = false
    # Проверяем последовательные квадраты
    for _ in 1:r-1
        x = modmul(x, x, n) #  $x = x^2 \bmod n$ 
        if x == n-1
            passed = true
            break
        end
    end
end

# Если не нашли свидетельство простоты - число составное
if !passed
    return false
end

end
```

Рис. 2: Реализация р-Метода Полларда

# Реализация р-Метода Полларда

```
    return true # Все тесты пройдены - вероятно простое
end

# Обёртка для проверки простоты
function isprime_small(n::Integer)
    return miller_rabin(n) # Просто вызываем тест Миллера-Рабина
end

# р-метод Полларда
function pollard_rho(n::Integer; maxiters::Int=10^6)
    # Проверка на чётность
    if n % 2 == 0
        return 2 # Если чётное - сразу возвращаем 2
    end
    Random.seed!() # Инициализация генератора случайных чисел
    # Несколько попыток с разными начальными значениями
    for attempt in 1:10
        c = rand(1:n-1) # Случайная константа в алгоритме
        x = rand(0:n-1) # Начальное значение x
        y = x           # Начальное значение y
        d = 1           # НОД
        iter = 0
        # Основной цикл алгоритма
        while d == 1 && iter < maxiters
            # Функция  $f(x) = x^2 + c \bmod n$ 
            x = (modmul(x,x,n) + c) % n # x делаем один шаг
            y = (modmul(y,y,n) + c) % n # y делаем два шага
            y = (modmul(y,y,n) + c) % n
            # Вычисляем НОД разности и n
            d = gcd(abs(x - y), n)
            iter += 1
        end
    end
```

Рис. 3: Реализация р-Метода Полларда

# Реализация р-Метода Полларда

```
# Если нашли нетривиальный делитель
if d != 1 && d != n
    return d
end
end
return nothing # Делитель не найден
end

# Метод Ферма (для чисел с близкими множителями)
function fermat_factor(n::Integer; maxtries::Int=10^6)
    # Проверка на четность
    if n % 2 == 0
        return (2, n ÷ 2) # Если четное - сразу возвращаем делители
    end
    # Начинаем с корня из n
    a = ceil{Int, sqrt}(BigFloat(n))
    b2 = a*a - n # Вычисляем b^2 = a^2 - n
    tries = 0
    while tries < maxtries
        if b2 >= 0
            b = isqrt(b2) # Целочисленный квадратный корень
            # Проверим, является ли b2 полным квадратом
            if b*b == b2
                p = a - b # Первый множитель
                q = a + b # Второй множитель
                # Проверим корректность разложения
                if p > 1 && q > 1 && p*q == n
                    return (p, q) # Возвращаем найденные множители
                end
            end
        end
        a += 1 # Увеличиваем a
        b2 = a*a - n # Пересчитываем b2
    end
end
```

Рис. 4: Реализация р-Метода Полларда



# Реализация р-Метода Полларда

```
        tries += 1
    end
    return nothing # Множители не найдены
end

# Рекурсивное разложение с использованием Полларда и Ферма
function factorize_num(n::Integer; out::Vector{Int64}=Int64[])
    # Базовый случай рекурсии
    if n == 1
        return out
    end
    # Если число простое - добавляем в результат
    if isprime_small(n)
        push!(out, Int(n)) # Преобразуем к Int и добавляем
        return out
    end
    # Пробуем метод Полларда
    d = pollard_rho(n)
    if d == nothing
        # Если Поллард не сработал, пробуем метод Ферма
        f = fermat_factor(n)
        if f != nothing
            (p, q) = f # Распаковываем множители
            # Рекурсивно разлагаем оба множителя
            factorize_num(p; out=out)
            factorize_num(q; out=out)
            return out
        else
            # Пробуем Полларда с увеличенным лимитом итераций
            d = pollard_rho(n; maxiters=5_000_000)
            if d == nothing
                # Если все методы не сработали - ошибка
                error("Не удалось найти делитель для n = $n")
            end
        end
    end
end
```

Рис. 5: Реализация р-Метода Полларда

## Реализация р-Метода Полларда

```
        end
    end
end
d = Int(abs(d)) # Приводим к Int и берем модуль
# Рекурсивно разлагаем найденный делитель и частное
factorize_num(d; out=out)
factorize_num(Int(n ÷ d); out=out)
return out
end

# Подсчёт степеней множителей (вместо StatsBase)
function count_factors(factors::Vector{Int})
    counts = Dict{Int,Int}{} # Создаем словарь для подсчета
    for f in factors
        # Увеличиваем счетчик для данного множителя
        counts[f] = get(counts, f, 0) + 1
    end
    return counts
end
```

Рис. 6: Реализация р-Метода Полларда

```
# Пример использования
n = 1359331 # Пример из методички
@printf("Разложение числа n = %d\n", n)

# Получаем все множители (с повторениями)
factors = factorize_num(n)
sort!(factors) # Сортируем для удобства чтения
@printf("Найденные множители: %s\n", repr(factors))

# Подсчитываем степени множителей
counts = count_factors(factors)

# Красиво выводим каноническое разложение
@printf("Каноническое разложение: ")
first = true
for (p,e) in sort(collect(counts)) # Сортируем по простым числам
    if !first
        print(" * ") # Разделитель между множителями
    end
    if e == 1
        print(p)      # Если степень 1 - выводим только число
    else
        print("$p^$e") # Иначе выводим с показателем степени
    end
    first = false
end
println()

Разложение числа n = 1359331
Найденные множители: [1151, 1181]
Каноническое разложение: 1151 * 1181
```

Рис. 7: Проверка

## Вывод

---

- В ходе выполнения лабораторной работы был изучен алгоритм разложения чисел на множители, а также написан его алгоритм на языке Julia.

## Список литературы. Библиография

---

[1] Julia: <https://docs.julialang.org/en/v1/>