Отчёт по лабораторной работе №5 Математические основы защиты информации и информационной безопасности

Вероятностные алгоритмы проверки чисел на простоту

Выполнил: Махорин Иван Сергеевич,
НФИмд-02-21, 1032259380

Содержание

Список иллюстраций

1 Цель работы

Изучить вероятностные алгоритмы проверки чисел на простоту и научиться их реализовывать.

2 Выполнение лабораторной работы

2.1 Реализация вычисления символа Якоби

Символ Якоби — теоретико-числовая функция двух аргументов, введённая К. Якоби в 1837 году. Является квадратичным характером в кольце вычетов. Символ Якоби обобщает символ Лежандра на все нечётные числа, большие единицы. Символ Кронекера — Якоби, в свою очередь, обобщает символ Якоби на все целые числа, но в практических задачах символ Якоби играет гораздо более важную роль, чем символ Кронекера — Якоби.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.1) и (рис. 2.2):

```
# Функция для вычисления символа Якоби
function jacobi_symbol(a, n)
    # Проверка на тривиальные случаи
    if n % 2 == 0 || n < 3
        throw(DomainError("n must be odd and >= 3"))
    end
    if a < 0 || a >= n
        a = mod(a, n)
    end

    # Инициализация переменных
    g = 1
    a_current = a
    n_current = n

    while true
        # Шаг 2: если a_current == 0, вернуть 0
        if a_current == 0
            return 0
        end
        # Шаг 3: если a_current == 1, вернуть g
        if a_current == 1
            return g
        end

        # Шаг 4: выделение степени двойки из a_current
        k = 0
        a_temp = a_current
        while a_temp % 2 == 0
            k += 1
            a_temp ÷= 2
        end
```

Рис. 2.1: Реализация вычисления символа Якоби

```
        a1 = a_temp

        # Шаг 5: вычисление s на основе k и n_current
        if k % 2 == 0
            s = 1
        else
            n_mod8 = n_current % 8
            if n_mod8 == 1 || n_mod8 == 7
                s = 1
            else
                s = -1
            end
        end

        # Шаг 6: если a1 == 1, вернуть g * s
        if a1 == 1
            return g * s
        end

        # Шаг 7: проверка условий для изменения s
        if n_current % 4 == 3 && a1 % 4 == 3
            s = -s
        end

        # Шаг 8: обновление переменных для следующей итерации
        a_current = n_current % a1
        n_current = a1
        g = g * s
    end
end
```

Рис. 2.2: Реализация вычисления символа Якоби

2.2 Реализация алгоритма, реализующего тест Ферма

Тест простоты Ферма в теории чисел — это тест простоты натурального числа n, основанный на малой теореме Ферма.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.3):

```julia
# Тест Ферма
function fermat_test(n, trials=1)
    # Проверка входных данных
    n < 5 && throw(DomainError("n must be >= 5"))
    n % 2 == 0 && return "Число $n составное"

    for _ in 1:trials
        # Шаг 1: выбор случайного основания a
        a = rand(2:(n-2))
        # Шаг 2: вычисление a^(n-1) mod n
        r = powermod(a, n-1, n)
        # Шаг 3: проверка условия
        if r != 1
            return "Число $n составное"
        end
    end
    return "Число $n, вероятно, простое"
end
```

Рис. 2.3: Реализация алгоритма, реализующего тест Ферма

2.3 Реализация алгоритма, реализующего тест Соловэя-Штрассена

Тест Соловея—Штрассена — вероятностный тест простоты, открытый в 1970-х годах Робертом Мартином Соловеем совместно с Фолькером Штрассеном. Тест всегда корректно определяет, что простое число является простым, но для составных чисел с некоторой вероятностью он может дать неверный ответ. Основное

преимущество теста заключается в том, что он, в отличие от теста Ферма, распознает числа Кармайкла как составные.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.4):

```julia
# Тест Соловэя-Штрассена
function solovay_strassen_test(n, trials=1)
    # Проверка входных данных
    n < 5 && throw(DomainError("n must be >= 5"))
    n % 2 == 0 && return "Число $n составное"

    for _ in 1:trials
        # Шаг 1: выбор случайного основания a
        a = rand(2:(n-2))
        # Проверка НОД(a, n)
        if gcd(a, n) > 1
            return "Число $n составное"
        end
        # Шаг 2: вычисление a^((n-1)/2) mod n
        r = powermod(a, (n-1)÷2, n)
        # Шаг 3: проверка условий
        if r != 1 && r != n-1
            return "Число $n составное"
        end
        # Шаг 4: вычисление символа Якоби
        s = jacobi_symbol(a, n)
        # Шаг 5: проверка соответствия
        if (r == n-1) && (s == -1)
            continue
        elseif r == s
            continue
        else
            return "Число $n составное"
        end
    end
    return "Число $n, вероятно, простое"
end
```

Рис. 2.4: Реализация алгоритма, реализующего тест Соловэя-Штрассена

2.4 Реализация алгоритма, реализующего тест Миллера-Рабина

Тест Миллера—Рабина — вероятностный полиномиальный тест простоты. Тест Миллера—Рабина, наряду с тестом Ферма и тестом Соловея—Штрассена, позволяет эффективно определить, является ли данное число составным. Однако, с его помощью нельзя строго доказать простоту числа. Тем не менее тест Миллера—Рабина часто используется в криптографии для получения больших случайных простых чисел.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.5) и (рис. 2.6):

```
# Тест Миллера-Рабина
function miller_rabin_test(n, trials=1)
    # Проверка входных данных
    n < 5 && throw(DomainError("n must be >= 5"))
    n % 2 == 0 && return "Число $n составное"

    # Шаг 1: представление n-1 в виде 2^s * r
    s = 0
    r = n - 1
    while r % 2 == 0
        s += 1
        r ÷= 2
    end

    for _ in 1:trials
        # Шаг 2: выбор случайного основания a
        a = rand(2:(n-2))
        # Шаг 3: вычисление y = a^r mod n
        y = powermod(a, r, n)
        # Шаг 4: проверка условий
        if y != 1 && y != n-1
            j = 1
            while j <= s-1 && y != n-1
                # Шаг 4.2.1: возведение в квадрат по модулю
                y = powermod(y, 2, n)
                # Шаг 4.2.2: проверка на нетривиальный корень
                if y == 1
                    return "Число $n составное"
                end
                j += 1
            end
```

Рис. 2.5: Реализация алгоритма, реализующего тест Миллера-Рабина

```
            # Шаг 4.3: окончательная проверка
            if y != n-1
                return "Число $n составное"
            end
        end
    end
    return "Число $n, вероятно, простое"
end
```

Рис. 2.6: Реализация алгоритма, реализующего тест Миллера-Рабина

Проверим работу алгоритмов (рис. 2.7):

```
miller_rabin_test (generic function with 2 methods)
```

```
# Проверка чисел на простоту
n = 101
a = 7

println("Тест Ферма: ", fermat_test(n, 5))
println("Тест Соловэя-Штрассена: ", solovay_strassen_test(n, 5))
println("Тест Миллера-Рабина: ", miller_rabin_test(n, 5))
# println("\nСимвол Якоби для a=$a и n=$n: ", jacobi_symbol(a, n))
```

```
Тест Ферма: Число 101, вероятно, простое
Тест Соловэя-Штрассена: Число 101, вероятно, простое
Тест Миллера-Рабина: Число 101, вероятно, простое
```

Рис. 2.7: Проверка

3 Список литературы. Библиография

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