Лабораторная Работа No 5

Математические Основы Защиты Информации и Информационной Безопасности

Хосе Фернандо Леон Атупанья | НФИмд-01-24

Содержание

1. Цель работы

Вычислить максимальный общий делитель, используя алгоритмы, представленные в лабораторном рабочем материале 5.

2. Выполнение лабораторной работы

2.1 Алгоритм, реализующий тест Ферма

В этой отчете реализуем тест Ферма на простоту, вероятностный алгоритм, используемый для проверки того, является ли число простым. Тест основан на Малой теореме Ферма, которая предполагает, что если число n составное.

Эта строка импортирует модуль Random, который предоставляет функции для генерации случайных чисел. Эта функция, fermat_primality_test, принимает целое число n в качестве входных данных. Цель этой функции - определить, является ли входное целое число n простым, используя вероятностный тест Ферма.Функция также проверяет, является ли число n четным, используя значение even(n). Четные числа (кроме 2) не являются простыми.

```
#Алгоритм, реализующий тест Ферма
using Random

function fermat_primality_test(n::Int)
    if n < 5 || iseven(n)
        return "Input must be a odd integer greater than or equ
    end
```

Эта строка генерирует случайное целое число a в диапазоне [2, n-2], модуль мощности(a, n - 1, n) эффективно вычисляет значение a^(n-1) (mod n). Если r равно 1, тест Ферма предполагает, что n может быть простым, поэтому функция возвращает "Число n, вероятно, простое". Иначе "Число n составное".

```
    a = rand(2:(n - 2))
    r = powermod(a, n - 1, n)
    if r == 1
        return "The number n is a probably prime"
    else
        return "The number n is composite"
    end
end
```

Он отображает сообщение с приглашением, а затем считывает вводимые пользователем данные с помощью функции readline().

```
function input(prompt:: AbstractString)
    print(prompt)
    return chomp(readline())
end
```

Затем введенная строка преобразуется в целое число с помощью синтаксического анализа (Int, n) и сохраняется в num1. Мы вызываем нашу функцию с аргументом num1, чтобы получить результат.

```
n = input("n = ")
num1 = parse(Int, n)

result = fermat_primality_test(num1)
println(result)
```

OUTPUT:

```
julia> 17
17
The number n is a probably prime
```

2.2 Алгоритм вычисления символа Якоби

Вычислите символ Якоби (n/a), важную функцию в теории чисел, часто используемую в алгоритмах, связанных с проверкой на простоту и квадратичными вычетами. Символ Якоби обобщает символ Лежандра и может быть вычислен для любого целого числа a и любого положительного нечетного числа n.

Функция jacobi_symbol принимает два целочисленных входных сигнала, a и n. Если a равно 1, функция возвращает g как символ Якоби. (1/n) всегда равно 1 для любых нечетных n.

```
function jacobi_symbol(a::Int, n::Int)
    g = 1

    if a == 0
        return 0
    end

    if a == 1
        return g
    end
```

Этот цикл удаляет все множители, равные 2, из a, многократно деля его на 2, пока оно не станет нечетным. Значение s зависит от того, сколько раз a было разделено на 2: Если k (количество делений на 2) четное, s остается равным 1. Если k нечетное, s зависит от значения n (mod 8).

```
    k = 0
    while iseven(a)
        a = div(a, 2)
        k += 1
    end

    a1 = a

    if iseven(k)
        s = 1
    else
        if n % 8 == 1 || n % 8 == 7
            s = 1
        else
            s = -1
        end
    end
```

Если a1 равно 1, то результатом вычисления символа Якоби будет просто произведение g * s, и функция вернет это значение. Функция применяет закон квадратичной взаимности, который изменяет s в зависимости от значений a1 и n по модулю 4. После настройки s и обновления a и n функция вызывает саму себя рекурсивно с a = n % a1 и n = a1.

```
    if a1 == 1
        return g * s
    end

    if n % 4 == 3 && a1 % 4 == 3
        s = -s
    end

    a = n % a1
    n = a1
    g *= s

    return jacobi_symbol(a, n) * g
end
```

jacobi_symbol(num1, num2) вызывает символьную функцию Якоби с предоставленными входными данными.

```
a = input("a = ")
num1 = parse(Int, a)

n = input("n = ")
num2 = parse(Int, n)

result = jacobi_symbol(num1, num2)
println("Jacobi symbol (a/b): ", result)
```

OUTPUT:

```
julia> 17
17
n = 5
Jacobi symbol (a/b): -1
```

2.3 Алгоритм, реализующий тест Соловэя-Штрасеена

Этот код реализует тест на простоту Соловея-Штрассена, вероятностный алгоритм, используемый для проверки вероятности того, что число является простым. Тест основан на свойствах символов Якоби и модульной арифметике.

Тест Соловея-Штрассена основан на выборе случайной базы для проведения вероятностного тестирования на первичность. Эта функция вычисляет символ Якоби (n/a) для заданных целых чисел a и n. Символ Якоби имеет решающее значение для сравнения свойств a относительно n в тесте.

```
using Random

function jacobi_symbol(a::Int, n ::Int)
    g = 1

    while a != 0
        #Step 4: Factor out powers of 2 in a to find a1(odd par
        k = 0
        while iseven(a)
            a = div(a, 2)
            k += 1
        end
        a1 = a

        #Determine s based on k and n mod 8
        s = 1
        if isodd(k)
            if n % 8 == 3 || n % 8 == 5
                s = -1
            end
        end
```

Эта часть делит a на 2 до тех пор, пока оно не станет нечетным, считая деления в k. Если k нечетно, а n по модулю 8 равно 3 или 5, значение s равно -1; в противном случае значение s остается равным 1. Это условие применяет закон квадратичной взаимности, который регулирует s на основе значений n и a1 по модулю 4.

```
        if n % 4 == 3 && a1 % 4 == 3
            s = -s
        end

        #Update g, a and n
        g *= s
        a, n = n % a1, a1
    end

    return g == 1 ? 1 : (n == 1 ? g : 0)
end
```

Он проверяет, является ли n допустимым входным значением (нечетное целое число, большее или равное 5). Если r не равно ни 1, ни n−1, функция делает вывод, что n является составным. Если r (mod n) не равно s, то n является составным. Если они равны, то n "вероятно, простое число".

```
function solovay_strassen_test(n::Int)
    if n < 5 || iseven(n)
        return "Input must be an odd integer greater than or eq
    end

    a = rand(2: n - 2)
    r = powermod(a, div(n - 1, 2), n)

    if r != 1 && r != n - 1
        return "The number n is composite"
    end

    s = jacobi_symbol(a, n)

    if r % n != s
        return "The number n is compisite"
    else
        return "The number n is probably prime"
    end
end
```

Функция solovay_strassen_test вызывается с номером 1, и результат выводится на печать.

```
n = input("n = ")
num1 = parse(Int, n)

r = solovay_strassen_test(num1)
println(r)
```

OUTPUT:

```
julia> 15
15
The number n is composite
```

2.4 Алгоритм, реализующий тест Миллера-Рабина

Этот код реализует тест на простоту Миллера-Рабина, вероятностный алгоритм, используемый для определения того, является ли данное целое число n простым.

Функция miller_rabin_test принимает целое число n в качестве входных данных для проверки на примитивность. Функция переписывает n−1 в виде 2^s*r, где r - нечетное число, а s - неотрицательное целое число.

```julia
using Random

function miller_rabin_test(n::Int)
    r = n - 1
    s = 0
    while iseven(r)
        r = div(r, 2)
        s += 1
    end

    a = rand(2: n - 2)
    y = powermod(a, r, n)
```

В этом цикле: y возводится в квадрат по модулю n с точностью до s−1 раз. Если в какой-либо точке y становится равным 1, функция делает вывод, что n является составным, поскольку это значение указывает на сбой в выполнении условия Миллера-Рабина. Если y достигает n−1 до завершения цикла, тест рассматривает этот экземпляр как потенциально простой и завершает цикл.

Если все проверки пройдены, функция возвращает "Число n, вероятно, простое". Поскольку тест является вероятностным, он не гарантирует простоту, а только то, что n, вероятно, простое.

```julia
    if y != 1 && y != n - 1
        j = 1
        while j <= s - 1 && y != n - 1
            y = powermod(y, 2, n)
            if y == 1
                return "The number n is composite"
            end
            j += 1
        end

        if y != n - 1
            return "The number n is composite"
        end
    end
    return "The number n is probably prime"
end
```

Выводится результат теста, либо "вероятно, простой", либо "сложный".

```
n = input("n = ")
num1 = parse(Int, n)

r = miller_rabin_test(num1)
println(r)
```

OUTPUT:

```
julia> 17
17
The number n is probably prime
```

3. Выводы

Каждый алгоритм обеспечивает различный баланс скорости и точности, при этом алгоритм Миллера-Рабина, как правило, является наиболее надежным для практического использования, особенно когда надежность имеет решающее значение. Тест Ферма, хотя и быстрый, более уязвим к ошибкам при работе с определенными составными числами, и метод Соловея-Штрассена находится между ними, предлагая разумный компромисс. Комбинирование этих методов или использование метода Миллера-Рабина с несколькими итерациями может обеспечить высокий уровень достоверности результата.

Для практических целей тест Миллера-Рабина часто является предпочтительным из-за его эффективности и высокой точности, особенно для приложений в криптографии, где важна простота.