

Лабораторная работа №5

Математические основы защиты информации и информационной безопасности

Минов К.В, НПМмд-02-2319

октября 2023

Российский университет дружбы народов

Москва, Россия

- 1) Реализовать на языке программирования вероятностные алгоритмы проверки чисел на простоту

Пусть a - целое число. Числа ± 1 , $\pm a$ называются тривиальными делителями числа a .

Целое число $p \in \mathbb{Z}/\{0\}$ называется простым, если оно не является делителем единицы и не имеет других делителей, кроме тривиальных. В противном случае число $p \in \mathbb{Z}/\{-1, 0, 1\}$ называется составным.

Детерминированный алгоритм всегда действует по одной и той же схеме и гарантированно решает поставленную задачу (или не дает никакого ответа).

Вероятностный алгоритм использует генератор случайных чисел и дает не гарантированно точный ответ. Вероятностные алгоритмы в общем случае не менее эффективны, чем детерминированные (если используемый генератор случайных чисел всегда дает набор одних и тех же чисел, зависящих от входных данных, то вероятностный алгоритм становится детерминированным).

Для проверки на простоту числа n вероятностным алгоритмом выбирают случайное число a ($1 < a < n$) и проверяют условия алгоритма. Если число n не проходит тест по основанию a , то алгоритм выдает результат “Число n составное”, и число n действительно является составным.

Если же n проходит тест по основанию a , ничего нельзя сказать о том, действительно ли число n является простым. Последовательно проведя ряд проверок таким тестом для разных a и получив для каждого из них ответ “Число n , вероятно, простое”, можно утверждать, что число n является простым с вероятностью, близкой к 1.

- Реализуем тест Ферма

```
import java.util.Random;

public class FermatTest {

    public static String fermat(int n) {
        Random random = new Random();
        int a = random.nextInt((n - 2) + 1) + 2; // Генерация случайного числа в диапазоне [2, n-2]
        int r = modPow(a, n - 1, n); // Вычисление  $a^{(n-1)} \bmod n$ 

        if (r == 1) {
            return "Число " + n + " вероятно простое";
        } else {
            return "Число " + n + " составное";
        }
    }

    // Вспомогательная функция для вычисления  $a^b \bmod n$ 
    private static int modPow(int a, int b, int n) {
        int result = 1;
        a = a % n;
        while (b > 0) {
            if (b % 2 == 1) {
                result = (result * a) % n;
            }
            b /= 2;
            a = (a * a) % n;
        }
        return result;
    }

    public static void main(String[] args) {
        int numberToTest = 35; // Замените на желаемое число для теста
        System.out.println(fermat(numberToTest));
    }
}
```

Figure 1: Рис.1: Тест Ферма

- Реализуем вычисление символа Якоби

```
public class JacobiSymbol {  
    // Метод для вычисления символа Якоби (a/n)  
    private static int jacobiSymbol(int a, int n) {  
        // Базовый случай  
        if (a == 0 || n == 1) return 0;  
        throw new IllegalArgumentException("Входные данные должны быть положительными целыми числами.");  
    }  
  
    if (a == 1) {  
        return (n == 1) ? 1 : 0;  
    } else if (a == -1) {  
        return 1;  
    }  
  
    // Проверка условия Лежандра  
    if (a < 0) {  
        int sign = (n % 4 == 3) ? -1 : 1;  
        return sign * jacobiSymbol(-a, n);  
    } else if (a % 2 == 0) {  
        int sign = ((n % 8 == 1) || (n % 8 == 7)) ? 1 : -1;  
        return sign * jacobiSymbol(a / 2, n);  
    } else {  
        // Алгоритм взаимной обратности  
        int sign = 1;  
        if (n % 4 == 3 && a % 4 == 3) sign = -1;  
        return sign * jacobiSymbol(n % a, a);  
    }  
}
```

Figure 2: Рис.2: Вычисление символа Якоби

- Реализуем тест Соловья-Штрассена

```
// Вспомогательная функция для вычисления a^b mod n
private static BigInteger power(BigInteger a, BigInteger b, BigInteger n) {

// Функция для проверки простоты числа с использованием теста Миллера-Рабина
private static boolean millerRabinTest(BigInteger n, int iterations) {
    if (n.compareTo(BigInteger.valueOf(2)) < 0) {
        return false; // Число должно быть больше или равно 2
    }

    if (n.equals(BigInteger.valueOf(2))) {
        return true; // 2 - простое число
    }

    if (n.and(BigInteger.ONE).equals(BigInteger.ZERO)) {
        return false; // Четное число (кроме 2) не является простым
    }

    // Представление n - 1 в виде 2^s * d
    BigInteger d = n.subtract(BigInteger.ONE);
    int s = 0;
    while (d.and(BigInteger.ONE).equals(BigInteger.ZERO)) {
        s++;
        d = d.shiftRight(1);
    }

    Random rand = new Random();

    for (int i = 0; i < iterations; i++) {
        // Выбор случайного числа a в интервале [2, n-2]
        BigInteger a = BigInteger.valueOf(2) + rand.nextInt((int)(n.intValue() - 3));

        // Вычисление a^d mod n
        BigInteger z = power(a, d, n);
```

Figure 3: Рис.3: Тест Соловья-Штрассена

- Реализуем тест Миллера-Рабина

```
// Функция для проверки простоты числа с использованием теста Соломона-Уилкинсона
private static boolean solovayStrassenTest(BigInteger n, int iterations) {
    if (n.compareTo(BigInteger.valueOf(2)) < 0) {
        return false; // Число меньше двух, это не простое
    }

    if (n.equals(BigInteger.valueOf(2))) {
        return true; // 2 - простое число
    }

    Random rand = new Random();

    for (int i = 0; i < iterations; i++) {
        // Выбираем случайное число a в диапазоне [2, n-2]
        BigInteger a = BigInteger.valueOf(2) + rand.nextInt((n.compareTo(BigInteger.ZERO) - 1));

        // Вычисляем символ Якоби a^{(n-1)/2} mod n
        int jacobi = JacobiSymbol(a, n);
        BigInteger exponent = n.subtract(BigInteger.ONE).divide(BigInteger.valueOf(2));
        BigInteger result = modularExponentiation(a, exponent, n);

        // Проверяем условие теста Соломона-Уилкинсона
        if (jacobi == 1 || (result.equals(BigInteger.ONE) && !result.equals(n.subtract(BigInteger.ONE)))) {
            return false; // n - составное число
        }
    }

    return true; // Вероятно, n простое
}
```

Figure 4: Рис.4: Тест Миллера-Рабина

- В ходе выполнения данной лабораторной работы были реализованы вероятностные алгоритмы проверки чисел на простоту

