Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Informática e Estatística
INE5429 - Segurança em Computação
Professor Ricardo Felipe Custódio
Aluno: Felipe Nedel Mendes de Aguiar Matrícula: 11100877

Miller-Rabin

O **teste de Miller-Rabin** é um teste probabilistico da primitividade de um número *n*. O teste é baseado no fato de que se um número *n* não passar pelo teste, *n* com certeza é um número composto (ou seja, não-primo). Se o número passar no teste, ele é primo, com uma probabilidade P > 0,75. A margem de erro pode ser diminuida aleatoriamente, aplicando-se o teste várias vezes ao mesmo número *n*.

O algoritmo é importante na criptografia assimétrica onde a necessidade de uma grande quantidade de números primos grandes é vital para a segurança dos algoritmos. Esse números são demasiado grandes para os testes convencionais de primalidade.

Instruções

Para executar o programa, utilize um prompt de comando qualquer. Execute o comando:

java –jar MillerRabin.jar (digitos)

onde *digitos* é a quantidade de dígitos desejada para o número primo gerado. Por exemplo:

java -jar .\MillerRabin.jar 100

Numeros primos muito grandes podem demorar a ser gerados.

O numero 43178651632576540624361486181067636381016173101305116878135614780848645532302053221105436380117234 47 eh um provavel primo (100 digitos).

Código Fonte

```
package MillerRabin;

import java.math.BigInteger;

public class Main {

    public static void main(String[] args) {

        // Contorle da quantidade de argumentos;
        if (args.length > 1) {
            System.out.println("Por favor utilize apenas um
argumento.");
            System.exit(0);
        }
        if (args.length < 1) {
            System.out.println("Por favor insira um argumento.");
            System.exit(0);
        }

        // Tamanho do primo a ser gerado;
        int size = 0;
        try {
            size = Integer.parseInt(args[0]);
        } catch (Exception e) {
            System.out.println("Argumento invalido: " + args[0]);
            System.exit(0);
        }

        if ( size > 80) {
            System.out.println("Numeros primos muito grandes podem
demorar a ser gerados.");
        }

        String number = "";
        BigInteger probablyPrime = BigInteger.ZERO;
        boolean isProbablyPrime = false;

        // Enquanto o número não for provavelmente primo;
        while (!isProbablyPrime) {
            number = "";
            int random = (int) (Math.random() * 9) + 1;
            number += random;
            for (int counter = 1; counter < size; counter++) {
                random = (int) (Math.random() * 9);
                number += random;
            }

            probablyPrime = new BigInteger(number);
            isProbablyPrime = MathUtils.isPrime(probablyPrime);
        }
```

```
        System.out.println("\nO numero " + probablyPrime + " eh um
provavel primo (" + size + " digitos).\n");

    }
}


package MillerRabin;

import java.math.BigInteger;

/**
 * @author Felipe Nedel
 *
 */
public class MathUtils {

    private static final BigInteger ZERO = BigInteger.ZERO;
    private static final BigInteger UM = BigInteger.ONE;
    private static final BigInteger DOIS = BigInteger.valueOf(2);
    private static final BigInteger TRES = BigInteger.valueOf(3);
    private static final BigInteger CINCO = BigInteger.valueOf(5);
    private static final BigInteger DEZ = BigInteger.TEN;

    /**
     * Produto modular
     *
     * @param a
     * @param b
     * @param n
     */
    private static BigInteger modProd(BigInteger a, BigInteger b,
BigInteger n) {
        if (b == ZERO) {
            return ZERO;
        }
        if (b == UM) {
            return a.mod(n);
        }

        BigInteger sub = b.subtract(b.mod(DEZ)).divide(DEZ);
        BigInteger number = modProd(a, sub, n);
        BigInteger mult = number.multiply(DEZ);
        BigInteger module = b.mod(DEZ);
        return mult.add(module.multiply(a)).mod(n);
    }

    /**
     * Exponenciação modular: a^b mod n
     *
     * @param a
     * @param b
     * @param n
     */
    private static BigInteger modPow(BigInteger a, BigInteger b,
BigInteger n) {
        if (b == ZERO) {
            return UM;
        }
        if (b == UM) {
```

```
            return a.mod(n);
        }
        if (b.mod(DOIS) == ZERO) {
            BigInteger modpow = modPow(a, b.divide(DOIS), n);
            return modProd(modpow, modpow, n);
        }
        return modProd(a, modPow(a, b.subtract(UM), n), n);
    }

    /**
     * Verifica se o número dado é primo
     *
     * @param n o número a ser testado
     * @return true se o número for primo. Senão, false.
     */
    public static boolean isPrime(BigInteger n) {
        // 2, 3 e 5 são números primos;
        if (n.equals(DOIS) || n.equals(TRES) || n.equals(CINCO)) {
            return true;
        }
        // Se o número for divisível por 2, 3, ou 5, então o número
não é primo;
        if (n.mod(DOIS).equals(ZERO) || n.mod(TRES).equals(ZERO) ||
n.mod(CINCO).equals(ZERO)) {
            return false;
        }
        // Senão, se for menor que 25, é primo;
        if (n.compareTo(BigInteger.valueOf(25)) < 0) {
            return true;
        }

        // Execução do algoritmo;
        BigInteger d = n.subtract(UM);
        BigInteger s = ZERO;

        // Fatoração de potências de 2 para n-1;
        while (d.mod(DOIS).compareTo(ZERO) == 0) {
            s = s.add(UM);
            d = d.divide(DOIS);
        }

        // Array de primos base;
        int[] primeValues = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
37, 41 };
        boolean continueFlag = false;
        BigInteger x;

        for (int index = 0; index < primeValues.length; index++) {
            BigInteger a = BigInteger.valueOf(primeValues[index]);
            x = modPow(a, d, n);

            // Se x = 1 ou x = n-1, chama o próximo valor do array
base de primos;
            if (x.equals(UM) || x.equals(n.subtract(UM))) {
                continue;
            }
            for (int counter = 1; counter <= s.intValue(); counter++)
{
                x = modProd(x, x, n);
                if (x.equals(UM)) {
                    // O número não é primo;
```

```
                    return false;
                }
                if (x.equals(n.subtract(UM))) {
                    continueFlag = true;
                    break;
                }
            }
            if (continueFlag) {
                continue;
            }
            // O número não é primo;
            return false;
        }

        // O número é primo;
        return true;
    }
}
```