Geração de prováveis números primos através do método Miller-Rabin

Aluno: Lucas Pereira da Silva (10100754)

O teste de primalidade Miller-Rabin consiste em um teste probabilístico para verificação de primalidade de um determinado número. Caso o teste retorne falso, então com certeza o número não é primo. Porém, caso o teste retorne verdadeiro, então existe uma grande probabilidade de que o número seja realmente primo e exsite também uma pequena possibilidade de que o número não seja primo.

No segundo caso, onde o teste retorna verdadeiro, a probabilidade de erro é dada por 4^{-k}, onde k é a quantidade de vezes que se aplicará o teste com uma base a diferente. Assim, percebemos que aplicando o teste apenas um vez e tendo um resultado positivo, então a probabilidade de erro é de 25%. No nosso caso, o teste é aplicado 10 vezes e, por isso, a probabilidade de erro em caso de resposta positiva é de apenas 0,00001%.

- Dado um suposto primo p. Primeiramente escreve-se: $(p-1) = 2^s.d$
- Onde s é o maior expoente possível tal que 2^s divide $(p-1)$ e d é a divisão de $(p-1)$ por 2^s.
- Consedere um inteiro a tal que $1 < a < p$. Caso o teste retorne negativo, a é chamado de testemunha contra a primalidade de p.
- Se $a^d \not\equiv 1 \bmod p$ e $a^{2r.d} \not\equiv -1 \bmod p$ para todo $r \in \{0, 1, \ldots, s-1\}$, então p não é primo. Caso contrário, existe uma grande probabilidade de p ser primo.

No programa que foi implementado, primeiramente é gerado um número aléatório n com x digítos decimais, onde x é definido pelo usuário. Após isso, o teste descrito acima é aplicado 10 vezes utilizando 10 a's diferentes. Caso a resposta seja verdadeira para os 10 testes, então o número n gerado é retornado como provável primo. Caso contrário o processo é repetido até que um provável primo n seja encontrado.

Exemplos:

- Teste de primalidade do 9 tendo $s = 3$ e $d = 1$.
  - Utilizando $a = 5$:
    - $a^d \not\equiv 1 \bmod p \rightarrow 5^1 \not\equiv 1 \bmod 9 \rightarrow$ verdadeiro
    - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 5^{0.1} \not\equiv -1 \bmod 9 \rightarrow$ verdadeiro

        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 5^{2.1} \not\equiv -1 \bmod 9 \rightarrow$ verdadeiro
        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 5^{4.1} \not\equiv -1 \bmod 9 \rightarrow$ verdadeiro
        - Como para todos os testes e reposta foi verdadeira, então o número não é primo. Portanto 5 é testemunha de contra primalidade de 9.
- Teste de primalidade do 5 tendo $s = 2$ e $d = 1$.
    - Utilizando $a = 3$:
        - $a^{d} \not\equiv 1 \bmod p \rightarrow 3^{1} \not\equiv 1 \bmod 5 \rightarrow$ verdadeiro
        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 3^{0.1} \not\equiv -1 \bmod 5 \rightarrow$ verdadeiro
        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 3^{2.1} \not\equiv -1 \bmod 5 \rightarrow$ falso
        - Como o último teste retornou falso, então 5 é um provável número primo. Será realizado o teste com outro a para aumentar a certeza da resposta.
    - Utilizando $a = 2$:
        - $a^{d} \not\equiv 1 \bmod p \rightarrow 2^{1} \not\equiv 1 \bmod 5 \rightarrow$ verdadeiro
        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 2^{0.1} \not\equiv -1 \bmod 5 \rightarrow$ verdadeiro
        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 2^{2.1} \not\equiv -1 \bmod 5 \rightarrow$ falso
        - Como o último teste retornou falso, então 5 é um provável número primo. Será realizado o teste com outro a para aumentar a certeza da resposta.
    - Utilizando $a = 4$:
        - $a^{d} \not\equiv 1 \bmod p \rightarrow 4^{1} \not\equiv 1 \bmod 5 \rightarrow$ verdadeiro
        - $a^{2r.d} \not\equiv -1 \bmod p \rightarrow 4^{0.1} \not\equiv -1 \bmod 5 \rightarrow$ falso
        - Como no segundo teste já obtivemos falso, então 5 é um provável número primo.
- Foram realizados três testes com, $a = 3$, $a = 2$ e $a = 4$. Todos os testes apontaram 5 como um provável número primo. Isso siginifica que com uma certeza de **98,43%**, pode-se afirmar que 5 é primo.

Código Fonte

```
package br.ufsc.inf.ine5429.primo;


import java.math.BigInteger;
import java.lang.Math;
```

```
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.Random;

public class Primo {
	private static final BigInteger DOIS = BigInteger.ONE.add(BigInteger.ONE);
	private static final BigInteger TRES = DOIS.add(BigInteger.ONE);
	private static final Integer QUANTIDADE_DE_TESTES = 10;
	private Integer quantidadeDeDigitos;
	private Random aleatorio;

	public Primo(Integer quantidadeDeDigitos) {
		this.quantidadeDeDigitos = quantidadeDeDigitos;
		aleatorio = new Random();
	}

	/*Gera um número aléatório com x digítos decimais e testa esse número através do Miller-Rabin. Quando o número gerado aleatóriamente for um provável primo, então este é retornado.*/
	public BigInteger encontrarProvavelPrimo() {
		BigInteger provavelPrimo = null;
		BigInteger provavelPrimoMenosUm;
		BigInteger multiplicador;
		Integer maiorExpoente;
		do {
			do {
				provavelPrimo = sortearProvavelPrimo();
			} while (provavelPrimo.compareTo(DOIS) <= 0);
			provavelPrimoMenosUm = provavelPrimo.subtract(BigInteger.ONE);
			maiorExpoente = encontrarMaiorExpoente(provavelPrimoMenosUm);
			multiplicador = encontrarMultiplicador(maiorExpoente, provavelPrimoMenosUm);
		} while (!testarPrimo(provavelPrimo, provavelPrimoMenosUm, maiorExpoente, multiplicador));
		return provavelPrimo;
	}

	/*Dado um suposto número primo realiza o teste de Miller-Rabin 10 vezes para testar a primalidade do suposto número primo.*/
	private Boolean testarPrimo(BigInteger provavelPrimo, BigInteger provavelPrimoMenosUm, Integer maiorExpoente, BigInteger multiplicador) {
```

```
            for (Integer contador = 0; contador < QUANTIDADE_DE_TESTES;
contador++) {
                  if (!testarPrimalidade(provavelPrimo, provavelPrimoMenosUm,
maiorExpoente, multiplicador)) {
                        return false;
                  }
            }
            return true;
      }

      /*Realiza o teste de Miller-Rabin para um suposto número primo utilizando
um a aleatório.*/
      private Boolean testarPrimalidade(BigInteger provavelPrimo, BigInteger
provavelPrimoMenosUm, Integer maiorExpoente, BigInteger multiplicador) {
            BigInteger testemunha =
sortearTestemunhaDePrimalidade(provavelPrimo);
            Boolean congruenteAUm = testemunha.modPow(multiplicador,
provavelPrimo).equals(BigInteger.ONE);
            if (congruenteAUm) {
                  return true;
            }
            for (Integer contador = 0; contador < maiorExpoente; contador++) {
                  Boolean congruenteAMenosUm =
testemunha.modPow(DOIS.pow(contador).multiply(multiplicador),
provavelPrimo).equals(provavelPrimoMenosUm);
                  if (congruenteAMenosUm) {
                        return true;
                  }
            }
            return false;
      }

      /*Gera o sorteio aleatório de um provável número primo com x digítos
decimais. A quantidade de digítos decimais é especificada pelo usuário.*/
      private BigInteger sortearProvavelPrimo() {
            Integer primeiroDigito = 0;
            String provavelPrimoTextual;
            BigInteger provavelPrimo;
            do {
                  primeiroDigito = aleatorio.nextInt(10);
            } while (primeiroDigito == 0);
            provavelPrimoTextual = primeiroDigito.toString();
```

```
            for (Integer contador = 1; contador < quantidadeDeDigitos; contador+
+) {
                  provavelPrimoTextual += aleatorio.nextInt(10);
            }
            provavelPrimo = new BigInteger(provavelPrimoTextual);
            if (provavelPrimo.mod(DOIS).equals(BigInteger.ZERO)) {
                  provavelPrimo = provavelPrimo.add(BigInteger.ONE);
            }
            return provavelPrimo;
      }

      /*Gera um a aleatório para ser utilizado no teste de Miller-Rabin.*/
      private BigInteger sortearTestemunhaDePrimalidade(BigInteger
provavelPrimo) {
            BigInteger testemunhaDePrimalidade = null;
            Integer quantidadeDeBits =
calcularQuantidadeDeBits(quantidadeDeDigitos);
            do {
                  testemunhaDePrimalidade = new BigInteger(quantidadeDeBits,
aleatorio);
            } while (testemunhaDePrimalidade.compareTo(provavelPrimo) >= 0 ||
testemunhaDePrimalidade.compareTo(BigInteger.ONE) <= 0);
            return testemunhaDePrimalidade;
      }

      /*Encontra o maior expoente com base 2 que divide o número fornecido. Ou
seja, no teste de Miller-Rabin, encontra o s.*/
      private Integer encontrarMaiorExpoente(BigInteger primoMenosUm) {
            Integer maiorExpoente = 1;
            while
(primoMenosUm.mod(DOIS.pow(maiorExpoente)).equals(BigInteger.ZERO)) {
                  maiorExpoente += 1;
            }
            return (maiorExpoente - 1);
      }

      /*Encontra o d utilizado no teste de Miller-Rabin.*/
      private BigInteger encontrarMultiplicador(Integer maiorExpoente,
BigInteger primoMenosUm) {
            BigInteger multiplicador =
primoMenosUm.divide(DOIS.pow(maiorExpoente));
            return multiplicador;
```

```
	}

	/*Calcula a quantidade de bits necessários para representar um número com x digítos decimais.*/
	private Integer calcularQuantidadeDeBits(Integer quantidadeDeDigitosDecimais) {
		Double base = 2.0;
		Double maiorNumero = Math.pow(10, quantidadeDeDigitosDecimais);
		return (int) (Math.log10(maiorNumero) / Math.log10(base));
	}

	/*Calcula a probabilidade de erro em caso de resposta positiva do teste de Miller-Rabin.*/
	public static Double forecerProbabilidadeDeErro() {
		return Math.pow(4, -QUANTIDADE_DE_TESTES);
	}

	/*Calcula a probabilidade de acerto em caso de resposta positiva do teste de Miller-Rabin.*/
	public static Double fornecerProbabilidadeDeAcerto() {
		return (1 - forecerProbabilidadeDeErro());
	}

	public static void main(String[] argumentos) {
		Scanner leitor = new Scanner(System.in);
		System.out.printf("Gerador de prováveis números primos. A probabilidade de erro é de: %f.\n", forecerProbabilidadeDeErro());
		System.out.print("Digite a quantidade máxima de digítos decimais do provável número primo que deseja gerar: ");
		try {
			Integer quantidadeDeDigitos = leitor.nextInt();
			leitor.nextLine();
			Primo primo = new Primo(quantidadeDeDigitos);
			Boolean encerrar = false;
			do {
				System.out.printf("Provável primo gerado: %d.\n", primo.encontrarProvavelPrimo());
				try {
					System.out.print("Deseja gerar outro primo (S/n)? ");
					encerrar = leitor.nextLine().equals("n");
```

```
                        } catch (InputMismatchException excecao) {
                              System.out.println("Programa encerrado.");
                              System.exit(0);
                        }
                  } while (!encerrar);
                  System.out.println("Programa encerrado.");
            } catch (InputMismatchException excecao) {
                        System.out.println("Programa encerrado.");
                        System.exit(0);
            }
      }
}
```