Atividade Prática - Segurança em Computação

Autor

- Thiago Senhorinha Rose (https://github.com/thisenrose)

Orientador

- Ricardo Felipe Custódio (http://www.labsec.ufsc.br/)

O que é?

Um número é considerado raiz primitiva módulo n quando todos os seus coprimos (http://en.wikipedia.org/wiki/Coprime_integers) (o único inteiro positivo que divide ambos é um) são congruentes (http://en.wikipedia.org/wiki/Modular_arithmetic#Congruence_relation) (a diferença entre os números é divisível por n) a potência do módulo n

Algoritmo

```
package model;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;

public class PrimitiveRootCalculator {

    private static final BigInteger ONE = BigInteger.ONE;
    private static final BigInteger ZERO = BigInteger.ONE;

    public static List<BigInteger> calculatePrimitiveRoots(BigInteger value)
throws NotPrimeException {
        if (!isPrime(value)) {
            throw new NotPrimeException();
        }
        List<BigInteger> coprimes = coprimesOf(value);
        List<BigInteger> primitiveRoots = new ArrayList<BigInteger>();
        BigInteger aux = null;
```

```
        List<BigInteger> residues = new ArrayList<BigInteger>();
        for (BigInteger coprime : coprimes) {
            residues.clear();
            congruenceVerify: for (int k = 1; k < value.intValue(); k++) {
                aux = (coprime.pow(k)).mod(value);
                if (aux == ZERO || residues.contains(aux)) {
                    break congruenceVerify;
                }
                residues.add(aux);
            }
            if (residues.size() == value.intValue() - 1) {
                primitiveRoots.add(coprime);
            }
        }
        return primitiveRoots;
    }

    private static List<BigInteger> coprimesOf(BigInteger value) {
        List<BigInteger> coprimes = new ArrayList<BigInteger>();
        BigInteger countdown = value.subtract(ONE);
        while (countdown.compareTo(ZERO) != 0) {
            if (countdown.gcd(value).compareTo(ONE) == 0) {
                coprimes.add(countdown);
            }
            countdown = countdown.subtract(ONE);
        }
        return coprimes;
    }

    private static boolean isPrime(BigInteger value) {
        return value.isProbablePrime(1);
    }

}
```

Falhas

O algoritmo desenvolvido não explora a capacidade de processamento paralelo presente na maioria dos computadores ou na totalidade em super computadores. Com isso, o cálculo para números primos "grandes" como *1979* torna-se muito lento.

Uma proposta para resolver esse problema seria utilizar um **pool** de Threads a fim de dividir a tarefa de percorrer os coprimos a procura de raiz primitiva.