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 modulo n quando todos os seus coprimos (http://en.wikipedia.org/wiki/Coprime_integers) (o único inteiro positivo que divide ambos é um) são congruantes (http://en.wikipedia.org/wiki/Modular_arithmetic#Congruence_relation) (a diferença entre os números é divisível por n) a potencia do modulo n

Algoritmo

```
package model;

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

public class PrimitiveRootCalculator {

    private final static Random random = new Random();
    private static final BigInteger ONE = BigInteger.ONE;
    private static final BigInteger ZERO = BigInteger.ONE;

    public synchronized static BigInteger calculate(BigInteger value) throws
NotPrimeException,
            PrimitiveRootNotFoundException {
        if (!isPrime(value)) {
            throw new NotPrimeException();
        }
```

```
        List<BigInteger> coprimesAlreadyTested = new ArrayList<BigInteger>();
        BigInteger aux = null;
        List<BigInteger> residues = new ArrayList<BigInteger>();
        BigInteger coprime = getOneCoprimeOfValueThatIsNotInList(value, coprimesAl
readyTested);
        while (coprime != null) {
            for (int k = 1; k < value.intValue(); k++) {
                aux = (coprime.pow(k)).mod(value);
                if (aux == ZERO || residues.contains(aux)) {
                    break;
                }
                residues.add(aux);
            }
            if (residues.size() == value.intValue() - 1) {
                return coprime;
            }
            coprimesAlreadyTested.add(coprime);
            residues.clear();
            coprime = getOneCoprimeOfValueThatIsNotInList(value, coprimesAlreadyTe
sted);
        }
        throw new PrimitiveRootNotFoundException();
    }

    private static BigInteger getOneCoprimeOfValueThatIsNotInList(BigInteger
 value, List<BigInteger> list) {
        BigInteger probableCoprime = ZERO;
        while (probableCoprime != null) {
            probableCoprime = new BigInteger(String.valueOf(random.nextInt(value.
intValue())));
            if (list.contains(probableCoprime)) {
                list.add(probableCoprime);
                if (list.size() >= value.intValue()) {
                    break;
                }
                continue;
            }
            if (probableCoprime.gcd(value).compareTo(ONE) == 0) {
                return probableCoprime;
            }
        }
        return null;
    }

    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 divir a tarefa de percorrer as potências de um coprimo a procura de algum indicador que este não é raiz primitiva .