Gerador Congruente Linear de números pseudo aleatórios(Documentação)

Autor:

Álbero Ítalo Souza dos Santos
playerghost@edu.unifor.br

Introdução

Esse documento tem o objetivo de instruir o leitor na geração de números pseudo aleatórios. Para tal, usamos a linguagem de programação Java, consequentemente é necessário ter um conhecimento básico da mesma ou de lógica de programação, para dar continuidade.

Números aleatórios e pseudo aleatórios, estão constantemente presentes no nosso cotidiano. Eles vão desde os sorteios de loteria, que são exemplos de pseudo aleatórios, até a nível molecular, como por exemplos números gerados através de ruído atmosférico(estes, por sua vez, não serão abordados nesse documento.

Preparativos

Para o ambiente de testes ou implementação, é necessário que sua máquina esteja preparada para os requistos básicos de desenvolvimento Java. Os principais são: Possuir instalado um JDK(de preferência atualizado)[1] e uma IDE[2], visto que é desnecessário o trabalho de se configurar e usar um terminal para tal.

Conceitos e implementação

Primeiramente, utilizaremos a Biblioteca HashSet[3] do Java. Um HashSet funciona de forma similar a um vetor, porém o tamanho não é fixo. Não necessariamente armazena os elementos de forma ordenada, pois sua posição se dá a partir de seu Hash code, que é gerado durante a inserção e é único para cada elemento. Além disso, nessa estrutura não é permitido a inserção de elementos repetidos.

Abaixo segue a implementação dessa estrutura em Java(Assim como será todo o código)

```
Set<Integer> numerosGerados = new HashSet<>();
```

Em seguida, utilizaremos a Biblioteca Math[4], em conjunto com a classe Random[5], para gerar um número pseudo aleatório de forma nativa. Será necessário um while loop, onde em seu parâmetro de checagem será usado uma função interna do HashSet, que terá como retorno o seu tamanho, que será checado antes de cada iteração do while.

E enquanto o tamanho do HashSet for diferentes de 6, que diz respeito a quantidade de números a serem gerados, o conteúdo do while será executado. Conteúdo esse que será a chamada da função Math.Random()[6], a qual faz uso da classe Random e retorna um número pseudo aleatório entre 0 e 1, que em seguida será multiplicado por 60 para converter para o alcance de 0 a 60 e somado 1, para excluir o número, que não queremos que seja gerado.

Esse resultado sofrerá um projeção para inteiro, visto que ele pode ser fracionário e só queremos números inteiros. Após isso, ele será armazenado em uma variável local do tipo inteiro que será referenciada na linha seguinte, onde esse número será inserido no HashSet. Aqui cabe a observação de que o número gerado só será inserido no HashSet se ele já não estiver lá, pois como foi dito, essa estrutura de dados não permite a inserção de números repetidos.

Segue o bloco de código

```
while (numerosGerados.size() != 6) {
      int number = (int) (Math.random() * 60 + 1);

      numerosGerados.add(number);
}
```

Após isso, será possível printar no terminal os números gerados, com o seguinte bloco de código:

```
System.out.println("Números gerados através da função Math.Random:");

System.out.println(Arrays.toString(numerosGerados.toArray()));
```

Se formos a fundo na documentação do próprio Java, encontraremos a informação de que o algoritmo por trás da função Math.Random, é a implementação de um gerador congruente linear, que usa como semente o tempo do sistema, obtido através da função System.nanoTime()[7] que será explicado em seguida.

Um gerador congruencial linear (LCG) é um algoritmo que produz uma sequência de números pseudo-randomizados calculados com uma equação linear descontínua por partes. O método representa um dos mais antigos e conhecidos algoritmos geradores de números pseudo-aleatórios.[8] A teoria por trás deles não é muito complexa de entender e são facilmente implementados e rápidos, especialmente em hardware de computador que pode fornecer aritmética modular por truncamento de bit de armazenamento.

A equação matemática que define esse gerador é: $X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0$.

- X_0 é o valor inicial ou “semente”; $X_0 \geq 0$
- a é o multiplicador; $a \geq 0$.
- c é o incremento; $c \geq 0$.
- m é o módulo; $m > X_0$, $m > a$, $m > c$.

X_n deve estar entre [0, m-1], n ≥ 0.

E isso se trata de um gerador congruente linear. Através dessa equação podemos fazer um exemplo que X_0 = a = c = 7, m = 10 e gerar a respectiva sequência de números: 7, 6, 9, 0, 7, 6, 9, 0, …

Detalhe, que nesse exemplo, por usarmos números pequenos como parâmetros, principalmente a semente, a chance de haver repetições é muito alta.

A implementação dessa técnica, em Java pode ser feita da seguinte maneira:

```
System.out.println("Números gerados através da implementação
matemática\nde um gerador de números pseudo aleatórios congruente
linear:");
//X0 = 7, Valor inicial; Deve ser X0 >= 0.
//a = 7, Multiplicador; Deve ser a >= 0.
//c = 7, Incremento; Deve ser c >= 0.
//m = 6, Módulo; Deve ser m > X0, m > a, m > c.
//x = (aXn + c) módulo m, n Deve ser n >= 0 n >= n.

int x = 7;

for (int i = 1; i <= 6; i++) {
    x = ((x * 7) + 7) % 60;

    System.out.print(x + " ");
}
```

Visando melhorar a eficiência, é possível combinar dois geradores, para então ter um gerador congruente linear combinado.

Um gerador congruencial linear combinado (CLCG)[9] é um algoritmo gerador de número pseudo-aleatório baseado na combinação de dois ou mais geradores congruentes lineares (LCG). Um LCG tradicional tem um período que é inadequado para simulação uma complexa. Ao combinar dois ou mais LCGs, números aleatórios com um período mais longo e melhores propriedades estatísticas podem ser criados.

O algoritmo é definido como: $X_i \equiv \left(\sum_{j=1}^{k} (-1)^{j-1} Y_{i,j}\right) \pmod{(m_1 - 1)}$

- m_1 é o módulo do LCG.
- $Y_{i,\ j}$ é a entrada de número i, do LCG de número j.
- X_i é o valor gerado de número i.

L'Ecuyer propõe um gerador linear combinado que utiliza dois LCGs em geradores de números aleatórios combinados eficientes e portáteis para processadores de 32 bits. Para ser mais preciso, os geradores congruenciais usados são na verdade multiplicativos, uma vez que c1 = c2 = 0.

Os parâmetros usados para os MCGs são:

- a1 = 40014
- m1 = 2147483563
- a2 = 40692
- m2 = 2147483399

Os dois MCGs, $Y_{0,1}$, $Y_{0,2}$, são semeados. É recomendado que os valores das sementes estejam nesta faixa [1, m1 − 1] e [1, m2 − 1] [1, m1 − 1] e [1, m2 − 1], respectivamente.

Podemos encontrar os valores dos dois MCGs usando o seguinte algoritmo:

- $Y_{i+1,1} = a1 \times Y_{i,1} (\text{mod } m1)$
- $Y_{i+1,2} = a1 \times Y_{i,2} (\text{mod } m2)$

Após descobrir os valores de $Y_{i+1,1}$ e $Y_{i+1,2}$, podemos encontrar X_{i+1}:

- $X_{i+1} = (Y_{i+1,1} - Y_{i+1,2}) \text{ mod } m1 - 1$

E então conseguiremos gerar o número aleatório da seguinte forma:

- $R_{i+1} = X_{i+1} / m1$
- for $X_{i+1} > 0$ $(X_{i+1} \; m1) + 1$
- for $X_{i+1} < 0$ $(m1 - 1) \; m1$
- for $X_{i+1} = 0$

A técnica acima pode ser implementada em java da seguinte forma:

```
System.out.println("Números gerados através da implementação matemática\nde um
gerador de números pseudo aleatórios congruente linear combinado:");
//Combina dois geradores congruentes lineares, consequentemente aumentando o
//perido, o que faz com leve mais tempo para haver repetições

int a1 = 40014;
int m1 = 2147483563;
int a2 = 40692;
int m2 = 2147483399;
int y1 = (int) Math.floor(Math.random() * m1) + 1;
int y2 = (int) Math.floor(Math.random() * m2) + 1;

float r = 0;

for (int i = 1; i <= 6; i++) {

     y1 = (a1 * y1) % m1;
     y2 = (a2 * y2) % m2;

     float aux = (y1 - y2) % m1;

     if (aux > 0) {
        r = (aux / m1);
```

```
    } else if (aux < 0) {
       r = (aux / m1) + 1;
    } else {
       r = (m1 - 1) / m1;
    }

    System.out.print((int) (r * 60) + 1 + " ");
}
```

Análise

Se organizarmos nossos geradores em métodos, poderemos utilizá-los para gerar histogramas e assim, observar o padrão com o quais os números são gerados, se existe alguma tendência ou algo do tipo.

Geradores em métodos

```
public static Object[] nativeRandomGenerator() {
//HashSet funciona de forma similar a um vetor, porém o tamanho não é fixo.
//Não necessariamente armazena os elementos de forma ordenada, pois sua posição
//dá-se
//a partir de seu Hash code. Além disso, nessa estrutura não é permitido a
//inserção de elementos repetidos.
    Set<Integer> numerosGerados = new HashSet<>();

    while (numerosGerados.size() != 6) {
          int number = (int) (Math.random() * 60 + 1);

          numerosGerados.add(number);
    }

    return numerosGerados.toArray();
}

public static void nativeRandomGeneratorPrint() {
    System.out.println(" ");
    System.out.println("Números gerados através da função Math.Random:");
    System.out.println(Arrays.toString(nativeRandomGenerator()));
}

public static int[] linearCongruentGenerator() {
//X0 = 7, Valor inicial; Deve ser X0 >= 0.
//a = 7, Multiplicador; Deve ser a >= 0.
//c = 7, Incremento; Deve ser c >= 0.
//m = 6, Módulo; Deve ser m > X0, m > a, m > c.
//x = (aXn + c) módulo m, n Deve ser n >= 0 n >= n.
    int numerosGeados[] = new int[6];

    int x = 7;
```

```
    for (int i = 1; i <= 6; i++) {
        x = ((x * 7) + 7) % 60;

        numerosGeados[i - 1] = x;
    }

    return numerosGeados;
}

public static void linearCongruentGeneratorPrint() {
    System.out.println(" ");
    System.out.println("Números gerados através da implementação matemática\nde
um gerador de números pseudo aleatórios congruente linear:");

    System.out.println(Arrays.toString(linearCongruentGenerator()));
}

public static int[] linearCombinedCongruentGenerator() {
//Combina dois geradores congruentes lineares, consequentemente aumentando o
período,
//o que faz com leve mais tempo para haver repetições
    int a1 = 40014;
    int m1 = 2147483563;
    int a2 = 40692;
    int m2 = 2147483399;

    int y1 = (int) Math.floor(Math.random() * m1) + 1;
    int y2 = (int) Math.floor(Math.random() * m2) + 1;

    float r = 0;

    int numerosGerados[] = new int[6];

    for (int i = 1; i <= 6; i++) {

        y1 = (a1 * y1) % m1;
        y2 = (a2 * y2) % m2;

        float aux = (y1 - y2) % m1;

        if (aux > 0) {
           r = (aux / m1);
        } else if (aux < 0) {
          r = (aux / m1) + 1;
        } else {
          r = (m1 - 1) / m1;
        }

        numerosGerados[i - 1] = (int) (r * 60) + 1;
```

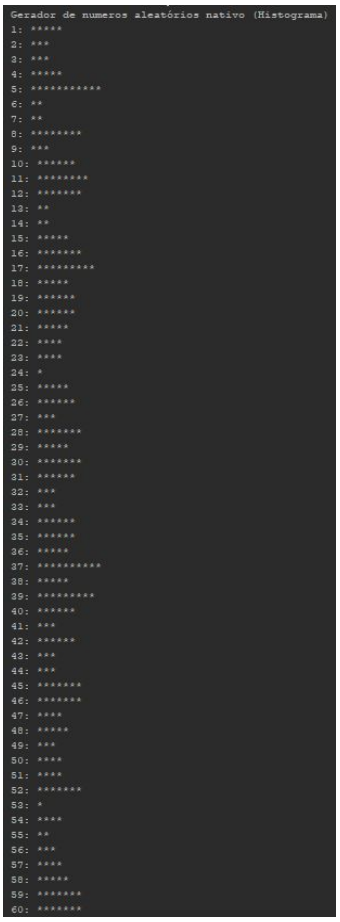
```
    return numerosGerados;
}

public static void linearCombinedCongruentGeneratorPrint() {
    System.out.println(" ");
    System.out.println("Números gerados através da implementação matemática\nde
um gerador de números pseudo aleatórios congruente linear combinado:");

    System.out.println(Arrays.toString(linearCombinedCongruentGenerator()));
}
```

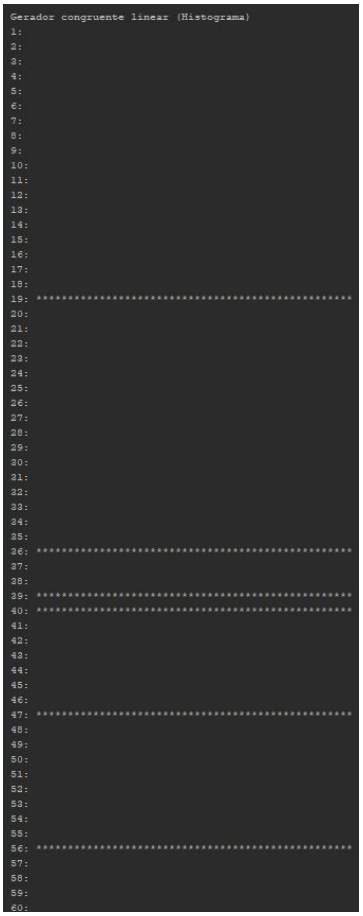
Feito isso, podemos criar os histogramas no nosso método main, que ficará da seguinte maneira:

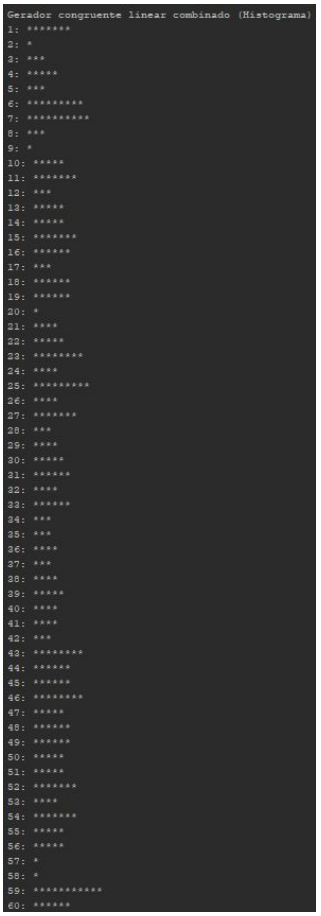
```
public static void main(String[] args) {
    int histograma1[] = new int[60];
    int histograma2[] = new int[60];
    int histograma3[] = new int[60];

    nativeRandomGeneratorPrint();
    linearCongruentGeneratorPrint();
    linearCombinedCongruentGeneratorPrint();

    for (int i = 0; i < 50; i++) {
        Object numerosgeradosAux1[] = nativeRandomGenerator();
        int numerosgeradosAux3[] = linearCongruentGenerator();
        int numerosgeradosAux4[] = linearCombinedCongruentGenerator();

        for (int j = 0; j < numerosgeradosAux1.length; j++) {
            histograma1[(int) numerosgeradosAux1[j] - 1]++;
            histograma2[numerosgeradosAux3[j] - 1]++;
            histograma3[numerosgeradosAux4[j] - 1]++;
         }
     }

    System.out.println("Gerador de numeros aleatórios nativo (Histograma)");

    for (int i = 0; i < histograma1.length; i++) {
        System.out.print(i + 1 + ": ");

        for (int j = 0; j < histograma1[i]; j++) {
            System.out.print("*");
         }

        System.out.println("");
    }

    System.out.println("\nGerador congruente linear (Histograma)");

    for (int i = 0; i < histograma2.length; i++) {
        System.out.print(i + 1 + ": ");
```

```
        for (int j = 0; j < histograma2[i]; j++) {
            System.out.print("*");
        }

        System.out.println("");
    }

    System.out.println("\nGerador congruente linear combinado (Histograma)");


       for (int i = 0; i < histograma3.length; i++) {
            System.out.print(i + 1 + ": ");

            for (int j = 0; j < histograma3[i]; j++) {
                System.out.print("*");
            }

            System.out.println("");
        }
}
```

E assim irão ficar os histogramas:







Referências

[1] Disponível em: Java SE Downloads .Acesso em 26/10/2020.
[2] Disponível em: Principais IDEs para desenvolvimento Java .Acesso em 26/10/2020.
[3] Disponível em: HashSet (Java Platform SE 7) .Acesso em 26/10/2020.
[4] Disponível em: Math (Java Platform SE 8) .Acesso em 26/10/2020.
[5] Disponível em: Random (Java Platform SE 8) .Acesso em 26/10/2020.
[6] Disponível em: java.util.Random() .Acesso em 26/10/2020.
[7] Disponível em: System.currentTimeMillis() .Acesso em 26/10/2020.
[8] Disponível em: Linear congruential generator .Acesso em 26/10/2020
[9] Disponível em: Combined Linear Congruential Generator for Pseudo-random Number Generation .Acesso em 26/10/2020.