Geradores de números pseudo-aleatórios:
Inversive congruential, Lagged Fibonacci e Linear congruential

Lucas João Martins

1 Códigos das implementações

icg.py

```
# -*- coding: utf-8 -*-
"""Implementacao do 'Inversive congruential generator'"""


class icg:
    """Classe que constroi o gerador de numeros
       pseudo-aleatorios

    Atributos:
        seed: semente do gerador
        q: numero que e feito o modulo da formula
        a: numero multiplicador da formula
        c: numero que faz a adicao da formula, ou, pode ser
           utilizado como um
           resultado
    """

    def __init__(self, seed, q, a, c):
        """Construtor com todos os atributos obrigatorios

        Args:
            ja especificado na classe
        """
        self.seed = seed
        self.q = q
        self.a = a
        self.c = c

    def generate(self, n):
        """Gerador de numeros pseudo-aleatorios

        Gera uma lista com n-1 numeros pseudo-aleatorios
           conforme o algoritmo
        icg ou seja, utiliza da seguinte relacao:
```

```
32              x_0 = seed
33              x_{i+1} = (a * x^{-1}_i + c) mod q  if x_i != 0
34              x_{i+1} = c                         if x_i == 0
35                  Onde x^{-1}_i e a funcao modular inversa
36
37          Args:
38              n: valor representa quantidade de numeros que
                   serao gerados - 1
39
40          Returns:
41              lista com os n + 1 numeros gerados
42          """
43          result = [self.seed]
44          for i in range(n):
45              number = 0
46              if result[i] == 0:
47                  number = self.c
48              else:
49                  x = (self.a *
                        self.modular_inverse(result[i]) + self.c)
50                  number = x % self.q
51
52              result.append(number)
53
54          return result
55
56      def modular_inverse(self, x):
57          """Funcao que calcula a modular inversa de um numero
58
59          Forma 'naive' de se calcular a modular inversa de um
                numero x mod q,
60          onde q e informado na construcao do gerador.
                Considerada 'naive' por nao
61          usar do algoritmo de euclides estendido. Funciona da
                seguinte maneira:
62              Calcula x * i mod q para todo i entre 0 e q - 1,
                    o i que gerar o
63              resultado 1 sera a modular inversa
64
65          Args:
66              x: numero do qual se quer saber a modular
                   inversa com q
67
68          Returns:
69              a modular inversa de um numero x mod q
70
71          Raises:
72              Exception: quando nao existe modular inversa
73          """
```

```
74          for i in range(self.q):
75              result = (x * i) % self.q
76              if result == 1:
77                  return i
78
79          raise Exception("Nao foi possivel calcular a
               'inversa modular'")
80
81  if __name__ == '__main__':
82      psng = icg(1, 5, 2, 3)
83      print(psng.generate(30))
```

lfg.py

```
1   # -*- coding: utf-8 -*-
2   """Implementacao do 'Lagged Fibonacci generator'"""
3   import random
4
5
6   class lfg:
7       """Classe que constroi o gerador de numeros
           pseudo-aleatorios
8
9       Atributos:
10          lags: tupla de dois valores que irao representar os
               'lags' da formula
11          exponent: expoente que ira na potencia de 2 que e
               feita de modulo da
12          formula
13      """
14
15      def __init__(self, lags, exponent):
16          """Construtor com todos os atributos obrigatorios
17
18          Args:
19              seed: lista vazia que ira conter a semente do
                   gerador
20              alem dos ja especificados na classe
21          """
22          self.j = lags[0]
23          self.k = lags[1]
24          self.m = 2**exponent
25          self.seed = []
26
27      def make_seed(self):
28          """Monta a lista de numeros que sera utilizada como
               semente no gerador
29
30          Para o funcionamento correto do algoritmo e
               necessario que k valores
```

```
31          aleatorios sejam utilizados como semente. Nessa
               implementacao isso e
32          feito da seguinte forma:
33              Utiliza-se o modulo random do python para gerar
                   um numero entre 0 e
34              1, para assim multiplicar esse valor por 10000
35              (um valor arbitrario) e por fim pegar a parte
                   inteira desse
36              resultado e adicionar na lista de sementes
37          """
38          i = 0
39          while i < self.k:
40              self.seed.append((int(random.random() * 10000)))
41              i += 1
42
43      def valid_amount(self, n):
44          """Verifica a validade dos argumentos utilizados na
               geracao dos numeros
45
46          Argumentos serao validos quando:
47              j < k
48              n - j < k
49              n - k < k
50
51          Args:
52              n: quantidade de numeros que serao gerados
53
54          Returns:
55              True se os argumentos sao validos, senao False
56          """
57          return True if n - self.j < self.k and self.j <
               self.k else False
58
59      def alfg_generate(self, n):
60          """Gerador de numeros pseudo-aleatorios com a
               operacao de adicao
61
62          Gera uma lista com n numeros pseudo-aleatorios
               conforme o algoritmo
63          alfg (additive lagged fibonacci generator), ou seja,
               ha outras versoes
64          que ao inves da adicao para montar o p do algoritmo
               utilizam da:
65              subtracao
66              multiplicacao
67              xor
68          O algoritmo utiliza da seguinte relacao:
69              Para cada numero que sera gerado a lista de
                   sementes e iterada:
```

```
70                  Quando e o primeiro elemento da lista de
                        sementes:
71                      Calcula-se o pseudo-aleatorio p:
72                          p  = (seed_{n - j} + seed_{n - k})
                                mod m
73                           Onde seed e a lista de sementes
74                  Quando e o ultimo elemento da lista de
                        sementes:
75                      A posicao recebe o pseudo-aleatorio
                            calculado
76                  Nos outros casos:
77                      O elemento a direita da lista vem para a
                            posicao atual
78
79          Args:
80              n: quantidade de numeros que serao gerados
81
82          Returns:
83              lista com os n numeros gerados
84
85          Raises:
86              Exception: quando os argumentos sao considerados
                    invalidos
87          """
88          if not self.valid_amount(n):
89              raise Exception("Valores invalidos!")
90
91          result = []
92          self.make_seed()
93          i = 0
94          while i < n:
95              for j in range(self.k):
96                  if j == 0:
97                      x = self.seed[n - self.j] + self.seed[n
                            - self.k]
98                      number = x % self.m
99                  elif 0 < j < self.k - 1:
100                     self.seed[j] = self.seed[j + 1]
101                 else:
102                     self.seed[j] = number
103                     result.append(number)
104             i += 1
105
106         return result
107
108 if __name__ == '__main__':
109     psng = lfg((70, 100), 4)
110     print(psng.alfg_generate(150))
```

lcg.py

```
# -*- coding: utf-8 -*-
"""Implementacao do 'Linear congruential generator'"""


class lcg:
    """Classe que constroi o gerador de numeros
       pseudo-aleatorios

    Atributos:
        multiplier: numero multiplicador da formula
        increment: numero que faz a adicao da formula
        seed: semente do gerador
        modulus: numero que e feito o modulo da formula
    """

    def __init__(self, multiplier, increment, seed, modulus):
        """Construtor com todos os atributos obrigatorios

        So constroi o gerador se os atributos sao
           considerados validos para a
        geracao dos numeros

        Args:
            ja especificado na classe

        Raises:
            Exception: quando os argumentos sao considerados
               invalidos
        """
        if modulus > 0 and 0 < multiplier < modulus and \
           0 <= increment < modulus and 0 <= seed < modulus:
            self.a = multiplier
            self.c = increment
            self.r0 = seed
            self.m = modulus
        else:
            raise Exception("Argumentos invalidos!")

    def generate(self, n):
        """Gerador de numeros pseudo-aleatorios

        Gera uma lista com n numeros pseudo-aleatorios
           conforme o algoritmo lcg,
        ou seja, utiliza da seguinte relacao:
            X_{n+1} = (a * X_n + c) mod m
                Quando n = 0, entao X_n = semente do gerador

        Args:
```

```
45              n: quantidade de numeros que serao gerados
46
47          Returns:
48              lista com os n numeros gerados
49          """
50          result = []
51          for i in range(n):
52              if i == 0:
53                  previous = self.r0
54              else:
55                  previous = result[i-1]
56
57              result.append((self.a * previous + self.c) %
                    self.m)
58
59          return result
60
61  if __name__ == '__main__':
62      psng = lcg(1664525, 1013904223, 10, 2**32)
63      print(psng.generate(1000))
```

2 Explicação dos algoritmos

3 Tabelas

4 Comparação entre os algoritmos

5 Complexidade dos algoritmos

6 Os números são aleatórios?

7 Referências