Geração de números primos:
Miller-Rabin, Fermat e Lucas

Lucas João Martins

1 Códigos das implementações

mr.py

```
1  # -*- coding: utf-8 -*-
2  """Gerador de numeros primos com base no 'Miller-Rabin
       primality test'"""
3
4  import random
5  import sys
6  import utils as u
7
8
9  class mr:
10     """Constroi o gerador de numeros primos"""
11
12     def __init__(self, k, bottom, up):
13         """Construtor com todos os atributos obrigatorios
14
15         Args:
16             k: acuracia da determinacao se o numero gerado e
                   primo
17             bottom: valor qual o numero primo gerado deve
                   ser maior
18             up: valor qual o numero primo gerado deve ser
                   menor
19         """
20         self.k = k
21         self.bottom = bottom
22         self.up = up
23
24     def decomposite(self, n):
25         """Escreve um numero par no formato 2^s * d
26
27         Determina s e d provenientes da transformacao do
               numero par recebido
28         para o formato 2^s * d. O argumento recebido e
               asseguradamente par,
```

```
29          pois trata-se do valor do qual se deseja determinar
               a primalidade menos
30          um, e, esse valor e par devido a forma que ele foi
               gerado
31
32          Args:
33              n: numero par que sera decomposto
34
35          Returns:
36              tupla com os valores de s e d no formato (s, d)
37          """
38          s, d = (0, 0)
39          while True:
40              x, y = divmod(n, 2)
41              if y == 0:
42                  s += 1
43                  n = x
44                  continue
45              else:
46                  d = n
47                  break
48
49          return (s, d)
50
51      def primality_test(self, n, s_d):
52          """Determina se um numero e primo de maneira
               probabilistica
53
54          Atraves do teste de miller-rabin define se o numero
               mais um passado
55          como argumento e um provavel primo ou nao. O
               argumento recebido e
56          asseguradamente par, pois trata-se do valor do qual
               se deseja
57          determinar a primalidade menos um, e, esse valor e
               par devido a forma
58          que ele foi gerado
59
60          Args:
61              n: numero par no qual o teste sera aplicado
62              s_d: tupla que representa a decomposicao de n em
                   2^s * d, onde d e
63              impar
64
65          Returns:
66              True se n+1 e um provavel primo, senao False
67          """
68          s, d = s_d
69          i = 0
```

```
70          while i < self.k:
71              i += 1
72              a = random.randint(2, n-1)
73              x = pow(a, d, n+1)
74              if x == 1 or x == n:
75                  continue
76
77              r = 0
78              while r <= s-1:
79                  r += 1
80                  x = pow(x, 2, n+1)
81                  if x == 1:
82                      return False
83                  if x == n:
84                      break
85
86              if x == n:
87                  continue
88
89              return False
90
91          return True
92
93      def generate(self):
94          """Gera um numero possivelmente primo entre a faixa
               [bottom, up]
95
96          Returns:
97              numero possivelmente primo gerado
98          """
99          while True:
100             n = u.utils.make_number(self.bottom, self.up)
101             s_d = self.decomposite(n-1)
102             if not self.primality_test(n-1, s_d):
103                 continue
104             return n
105
106 if __name__ == '__main__':
107     png = mr(10, int(sys.argv[1]), int(sys.argv[2]))
108     print(png.generate())
```

fermat.py

```
1  # -*- coding: utf-8 -*-
2  """Gerador de numeros primos com base no 'Fermat primality
       test'"""
3
4  import random
5  import sys
6  import math
```

```
7  import utils as u
8
9
10 class fermat:
11     """Constroi o gerador de numeros primos"""
12
13     def __init__(self, k, bottom, up):
14         """Construtor com todos os atributos obrigatorios
15
16         Args:
17             k: acuracia da determinacao se o numero gerado e
                   primo
18             bottom: valor qual o numero primo gerado deve
                   ser maior
19             up: valor qual o numero primo gerado deve ser
                   menor
20         """
21         self.k = k
22         self.bottom = bottom
23         self.up = up
24
25     def primality_test(self, n):
26         """Determina se um numero e primo de maneira
               probabilistica
27
28         Atraves do teste de fermat define se o numero
               passado como argumento
29         e um provavel primo ou nao. O argumento recebido e
               asseguradamente
30         impar devido a forma que ele foi gerado
31
32         Args:
33             n: numero impar no qual o teste sera aplicado
34
35         Returns:
36             True se n e um provavel primo, senao False
37         """
38         i = 0
39         while i < self.k:
40             i += 1
41             a = random.randint(1, n)
42             if math.gcd(a, n) != 1 or pow(a, n-1, n) != 1:
43                 return False
44
45         return True
46
47     def generate(self):
48         """Gera um numero possivelmente primo entre a faixa
               [bottom, up]
```

```
49
50          Returns:
51              numero possivelmente primo gerado
52          """
53          while True:
54              n = u.utils.make_number(self.bottom, self.up)
55              if not self.primality_test(n):
56                  continue
57              return n
58
59  if __name__ == '__main__':
60      png = fermat(10, int(sys.argv[1]), int(sys.argv[2]))
61      print(png.generate())
```

lucas.py

```
1   # -*- coding: utf-8 -*-
2   """Gerador de numeros primos com base no 'Lucas primality
        test'"""
3
4   import random
5   import sys
6   import primefac as pf
7   import utils as u
8
9
10  class lucas:
11      """Constroi o gerador de numeros primos"""
12
13      def __init__(self, k, bottom, up):
14          """Construtor com todos os atributos obrigatorios
15
16          Args:
17              k: acuracia da determinacao se o numero gerado e
                    primo
18              bottom: valor qual o numero primo gerado deve
                    ser maior
19              up: valor qual o numero primo gerado deve ser
                    menor
20          """
21          self.k = k
22          self.bottom = bottom
23          self.up = up
24
25      def prime_factors(self, n):
26          """Determina os fatores primos de um numero
27
28          Com o auxilio do modulo primefac gera-se os fatores
                primos do numero
```

```
29          passado como argumento, onde os fatores primos sao
               os numeros primos
30          que dividem o argumento de maneira exata
31
32          Args:
33              n: numero do qual sera determinado os fatores
                   primos
34
35          Returns:
36              Uma lista com os fatores primos
37          """
38          return list(pf.primefac(n))
39
40      def primality_test(self, n):
41          """Determina se um numero e primo de maneira
               probabilistica
42
43          Atraves do teste de lucas define se o numero passado
               como argumento
44          e um provavel primo ou nao. O argumento recebido e
               asseguradamente
45          impar devido a forma que ele foi gerado
46
47          Args:
48              n: numero impar no qual o teste sera aplicado
49
50          Returns:
51              True se n e um provavel primo, senao False
52          """
53          i = 0
54          prime_factors = self.prime_factors(n-1)
55          while i < self.k:
56              i += 1
57              a = random.randint(2, n-1)
58              if pow(a, n-1, n) != 1:
59                  return False
60
61              for q in prime_factors:
62                  if pow(a, (n-1)//q, n) != 1:
63                      if q == prime_factors[-1]:
64                          return True
65                      else:
66                          continue
67                  else:
68                      break
69
70          return False
71
72      def generate(self):
```

```
        """Gera um numero possivelmente primo entre a faixa
           [bottom, up]

        Returns:
            numero possivelmente primo gerado
        """
        while True:
            n = u.utils.make_number(self.bottom, self.up)
            if not self.primality_test(n):
                continue
            return n

if __name__ == '__main__':

    png = lucas(10, int(sys.argv[1]), int(sys.argv[2]))
    print(png.generate())
```

utils.py

```
# -*- coding: utf-8 -*-
"""Funcoes uteis para a geracao de numeros primos"""

import random


class utils:
    """Classe que possui as funcoes uteis"""

    @staticmethod
    def make_number(bottom, up):
        """Retorna um inteiro aleatorio impar entre a faixa
           [bottom, up]

        Args:
            bottom: menor valor da faixa de valores
            up: maior valor da faixa de valores

        Returns:
            Inteiro impar
        """
        i = 2
        while i % 2 == 0:
            i = random.randint(bottom, up)
        return i
```

tests.py

```
"""Testes unitarios dos codigos desenvolvidos na geracao de
   numeros primos"""

```

```
3  import unittest
4  from lucas import lucas
5  from mr import mr
6  from fermat import fermat
7  from utils import utils
8
9
10 class Tests(unittest.TestCase):
11     # utils
12     def test_odd(self):
13         self.assertTrue(utils.make_number(100, 1000) % 2 !=
               0)
14
15     # mr
16     def test_right_decomposite(self):
17         png = mr(10, 100, 10000)
18         self.assertEqual(png.decomposite(12), (2, 3))
19
20     def test_wrong_decomposite(self):
21         png = mr(10, 100, 10000)
22         self.assertNotEqual(png.decomposite(186), (2, 5))
23
24     def test_true_primality_mr(self):
25         png = mr(10, 100, 10000)
26         self.assertTrue(png.primality_test(12, (2, 3)))
27
28     def test_false_primality_mr(self):
29         png = mr(10, 100, 10000)
30         self.assertFalse(png.primality_test(26, (1, 13)))
31
32     # fermat
33     def test_true_primality_fermat(self):
34         png = fermat(10, 100, 10000)
35         self.assertTrue(png.primality_test(8837))
36
37     def test_false_primality_fermat(self):
38         png = fermat(10, 100, 10000)
39         self.assertFalse(png.primality_test(297))
40
41     # lucas
42     def test_right_prime_factors(self):
43         png = lucas(10, 100, 10000)
44         self.assertEqual(png.prime_factors(75), [3, 5, 5])
45
46     def test_wrong_prime_factors(self):
47         png = lucas(10, 100, 10000)
48         self.assertNotEqual(png.prime_factors(17), [1, 2,
               17])
49
```

```
50      def test_true_primality_lucas(self):
51          png = lucas(10, 100, 10000)
52          self.assertTrue(png.primality_test(1249))
53
54      def test_false_primality_lucas(self):
55          png = lucas(10, 100, 10000)
56          self.assertFalse(png.primality_test(8826))
57
58  if __name__ == '__main__':
59      unittest.main()
```

2 Explicação dos algoritmos

3 Comparação entre os algoritmos

4 Complexidade dos algoritmos que verificam primalidade

- Miller-Rabin: no pior caso é $O(k * s)$.
  - Devido aos laços nas linhas 70 e 78 do código.
- Fermat: no pior caso é $O(k * \log n)$.
  - Devido ao laço na linha 39 e da chamada de método na linha 42.
- Lucas: no pior caso é $O(k * \log n)$.
  - Devido aos laços nas linhas 55 e 61 do código. O número de fatores primos da linha 61 pode ser aproximado para $\log n$.

5 Código-fonte

GitHub

6 Referências

Miller-Rabin:

- Riemann's Hypothesis and Tests for Primality, Gary L. Miller
- Wikipedia
- Algorithm Implementation no Wikibooks
- Rosetta Code

Fermat:

- Khan Academy
- Wikipedia

Lucas:

- Módulo primefac
- Wikipedia