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

Lucas João Martins

1 Códigos das implementações

mr.py

```python
# -*- coding: utf-8 -*-

import random
import sys
import utils as u


class mr:

    def __init__(self, k, bottom, up):
        self.k = k
        self.bottom = bottom
        self.up = up

    # documentar sobre seguranca que nao eh impar
    def decomposite(self, n):
        s, d = (0, 0)
        while True:
            x, y = divmod(n, 2)
            if y == 0:
                s += 1
                n = x
                continue
            else:
                d = n
                break

        return (s, d)

    def primality_test(self, n, s_d):
        s, d = s_d
        i = 0
        while i < self.k:
            i += 1
```

```
            a = random.randint(2, n-1)
            x = pow(a, d, n+1)
            if x == 1 or x == n:
                continue

            r = 0
            while r <= s-1:
                r += 1
                x = pow(x, 2, n+1)
                if x == 1:
                    return False
                if x == n:
                    break

            if x == n:
                continue

            return False

        return True

    def generate(self):
        while True:
            n = u.utils.make_number(self.bottom, self.up)
            s_d = self.decomposite(n-1)
            if not self.primality_test(n-1, s_d):
                continue
            return n

if __name__ == '__main__':
    png = mr(10, int(sys.argv[1]), int(sys.argv[2]))
    print(png.generate())
```

fermat.py

```
# -*- coding: utf-8 -*-

import random
import sys
import math
import utils as u


class fermat:

    def __init__(self, k, bottom, up):
        self.k = k
        self.bottom = bottom
        self.up = up

```

```
16      def primality_test(self, n):
17          i = 0
18          while i < self.k:
19              i += 1
20              a = random.randint(1, n)
21              if math.gcd(a, n) != 1 or pow(a, n-1, n) != 1:
22                  return False
23
24          return True
25
26      def generate(self):
27          while True:
28              n = u.utils.make_number(self.bottom, self.up)
29              if not self.primality_test(n):
30                  continue
31              return n
32
33  if __name__ == '__main__':
34      png = fermat(10, int(sys.argv[1]), int(sys.argv[2]))
35      print(png.generate())
```

lucas.py

```
1   # -*- coding: utf-8 -*-
2
3   import random
4   import sys
5   import primefac as pf
6   import utils as u
7
8
9   class lucas:
10
11      def __init__(self, k, bottom, up):
12          self.k = k
13          self.bottom = bottom
14          self.up = up
15
16      def prime_factors(self, n):
17          return list(pf.primefac(n))
18
19      def primality_test(self, n):
20          i = 0
21          prime_factors = self.prime_factors(n-1)
22          while i < self.k:
23              i += 1
24              a = random.randint(2, n-1)
25              if pow(a, n-1, n) != 1:
26                  return False
27
```

```
28              for q in prime_factors:
29                  if pow(a, (n-1)//q, n) != 1:
30                      if q == prime_factors[-1]:
31                          return True
32                      else:
33                          continue
34                  else:
35                      break
36
37          return False
38
39      def generate(self):
40          while True:
41              n = u.utils.make_number(self.bottom, self.up)
42              if not self.primality_test(n):
43                  continue
44              return n
45
46  if __name__ == '__main__':
47      png = lucas(10, int(sys.argv[1]), int(sys.argv[2]))
48      print(png.generate())
```

2 Explicação dos algoritmos

3 Comparação entre os algoritmos

4 Complexidade dos algoritmos

5 Referências