

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

Lucas João Martins

1 Códigos das implementações

mr.py

```
1 # -*- coding: utf-8 -*-
2
3 import random
4 import sys
5
6
7 class mr:
8
9     def __init__(self, k, bottom, up):
10         self.k = k
11         self.bottom = bottom
12         self.up = up
13
14     def make_number(self):
15         i = 2
16         while i % 2 == 0:
17             i = random.randint(self.bottom, self.up)
18         return i
19
20     def decomposite(self, n):
21         s, d = (0, 0)
22         while True:
23             x, y = divmod(n, 2)
24             if y == 0:
25                 s += 1
26                 n = x
27                 continue
28             else:
29                 d = n
30                 break
31
32         return (s, d)
33
34     def primarity_test(self, n, s_d):
```

```

35     s, d = s_d
36     i = 0
37     while i < self.k:
38         i += 1
39         a = random.randint(2, n-1)
40         x = pow(a, d, n+1)
41         if x == 1 or x == n:
42             continue
43
44         r = 0
45         while r <= s-1:
46             r += 1
47             x = pow(x, 2, n+1)
48             if x == 1:
49                 return False
50             if x == n:
51                 break
52
53         if x == n:
54             continue
55
56         return False
57
58     return True
59
60     def generate(self):
61         while True:
62             n = self.make_number()
63             s_d = self.decomposite(n-1)
64             if not self.primarility_test(n-1, s_d):
65                 continue
66             return n
67
68 if __name__ == '__main__':
69     png = mr(10, int(sys.argv[1]), int(sys.argv[2]))
70     print(png.generate())

```

fermat.py

```

1  # -*- coding: utf-8 -*-
2
3  import random
4  import sys
5  import math
6
7
8  class fermat:
9
10     def __init__(self, k, bottom, up):
11         self.k = k

```

```

12         self.bottom = bottom
13         self.up = up
14
15     def make_number(self):
16         i = 2
17         while i % 2 == 0:
18             i = random.randint(self.bottom, self.up)
19         return i
20
21     def primarility_test(self, n):
22         i = 0
23         while i < self.k:
24             i += 1
25             a = random.randint(1, n)
26             if math.gcd(a, n) != 1 or pow(a, n-1, n) != 1:
27                 return False
28
29         return True
30
31     def generate(self):
32         while True:
33             n = self.make_number()
34             if not self.primarility_test(n):
35                 continue
36             return n
37
38 if __name__ == '__main__':
39     png = fermat(10, int(sys.argv[1]), int(sys.argv[2]))
40     print(png.generate())

```

lucas.py

```

1  # -*- coding: utf-8 -*-
2
3  import random
4  import sys
5
6
7  class ss:
8
9      def __init__(self, k, bottom, up):
10         self.k = k
11         self.bottom = bottom
12         self.up = up
13
14     def make_number(self):
15         i = 2
16         while i % 2 == 0:
17             i = random.randint(self.bottom, self.up)
18         return i

```

```

19
20     def primarility_test(self, n):
21         i = 0
22         while i < self.k:
23             i += 1
24             a = random.randint(2, n-1)
25             if math.gcd(a, n) != 1 or pow(a, n-1, n) != 1:
26                 return False
27
28         return True
29
30     def generate(self):
31         while True:
32             n = self.make_number()
33             if not self.primarility_test(n):
34                 continue
35             return n
36
37 if __name__ == '__main__':
38     png = ss(10, int(sys.argv[1]), int(sys.argv[2]))
39     print(png.generate())

```

2 Explicação dos algoritmos

3 Comparação entre os algoritmos

4 Complexidade dos algoritmos

5 Referências