

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
17                primo
18             bottom: valor qual o numero primo gerado deve
19                    ser maior
20             up: valor qual o numero primo gerado deve ser
21                 menor
22
23         """
24         self.k = k
25         self.bottom = bottom
26         self.up = up
27
28     def decomposite(self, n):
29         """Escreve um numero par no formato 2^s * d
30
31         Determina s e d provenientes da transformacao do
32         numero par recebido
33         para o formato 2^s * d. O argumento recebido e
34         assecuradamente par,
```

```

29     pois trata-se do valor do qual se deseja determinar
30     a primalidade menos
31     um, e, esse valor e par devido a forma que ele foi
32     gerado
33
34     Args:
35         n: numero par que sera decomposto
36
37     Returns:
38         tupla com os valores de s e d no formato (s, d)
39     """
40     s, d = (0, 0)
41     while True:
42         x, y = divmod(n, 2)
43         if y == 0:
44             s += 1
45             n = x
46             continue
47         else:
48             d = n
49             break
50
51     return (s, d)
52
53 def primality_test(self, n, s_d):
54     """Determina se um numero e primo de maneira
55     probabilistica
56
57     Atraves do teste de miller-rabin define se o numero
58     mais um passado
59     como argumento e um provavel primo ou nao. 0
60     argumento recebido e
61     asseguradoamente par, pois trata-se do valor do qual
62     se deseja
63     determinar a primalidade menos um, e, esse valor e
64     par devido a forma
65     que ele foi gerado
66
67     Args:
68         n: numero par no qual o teste sera aplicado
69         s_d: tupla que representa a decomposicao de n em
70              $2^s * d$ , onde d e
71             impar
72
73     Returns:
74         True se n+1 e um provavel primo, senao False
75     """
76     s, d = s_d
77     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
95             [bottom, up]
96
97         Returns:
98             numero possivelmente primo gerado
99         """
100         while True:
101             n = u.utils.make_number(self.bottom, self.up)
102             s_d = self.decomposite(n-1)
103             if not self.primalty_test(n-1, s_d):
104                 continue
105             return n
106
107 if __name__ == '__main__':
108     bottom = int('1' + '0' * int(sys.argv[1]), 2)
109     up = int('1' * int(sys.argv[2]), 2)
110     png = mr(10, bottom, up)
111     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
18                primo
19             bottom: valor qual o numero primo gerado deve
20                ser maior
21             up: valor qual o numero primo gerado deve ser
22                menor
23
24         """
25         self.k = k
26         self.bottom = bottom
27         self.up = up
28
29     def primality_test(self, n):
30         """Determina se um numero e primo de maneira
31            probabilistica
32
33         Atraves do teste de fermat define se o numero
34            passado como argumento
35            e um provavel primo ou nao. O argumento recebido e
36            asseguradoamente
37            impar devido a forma que ele foi gerado
38
39         Args:
40             n: numero impar no qual o teste sera aplicado
41
42         Returns:
43             True se n e um provavel primo, senao False
44
45         """
46         i = 0
47         while i < self.k:
48             i += 1
49             a = random.randint(1, n)
50             if math.gcd(a, n) != 1 or pow(a, n-1, n) != 1:
51                 return False
52
53         return True
54
55     def generate(self):

```

```

48         """Gera um numero possivelmente primo entre a faixa
49             [bottom, up]
50
51         Returns:
52             numero possivelmente primo gerado
53         """
54         while True:
55             n = u.utils.make_number(self.bottom, self.up)
56             if not self.primalty_test(n):
57                 continue
58             return n
59
60 if __name__ == '__main__':
61     bottom = int('1' + '0' * int(sys.argv[1]), 2)
62     up = int('1' * int(sys.argv[2]), 2)
63     png = fermt(10, bottom, up)
64     print(png.generate())

```

lucas.py

```

1  # -*- coding: utf-8 -*-
2  """Gerador de numeros primos com base no 'Lucas primality
3      test'"""
4
5  import random
6  import sys
7  import primefac as pf
8  import utils as u
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
18                 primo
19             bottom: valor qual o numero primo gerado deve
20                 ser maior
21             up: valor qual o numero primo gerado deve ser
22                 menor
23         """
24         self.k = k
25         self.bottom = bottom
26         self.up = up
27
28     def prime_factors(self, n):
29         """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
        asseguradoamente
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):
73         """Gera um numero possivelmente primo entre a faixa
74             [bottom, up]
75
76         Returns:
77             numero possivelmente primo gerado
78         """
79         while True:
80             n = u.utils.make_number(self.bottom, self.up)
81             if not self.primalidade_test(n):
82                 continue
83             return n
84
85 if __name__ == '__main__':
86     bottom = int('1' + '0' * int(sys.argv[1]), 2)
87     up = int('1' * int(sys.argv[2]), 2)
88     png = lucas(10, bottom, up)
89     print(png.generate())

```

utils.py

```

1  # -*- coding: utf-8 -*-
2  """Funcoes uteis para a geracao de numeros primos"""
3
4  import random
5
6
7  class utils:
8      """Classe que possui as funcoes uteis"""
9
10     @staticmethod
11     def make_number(bottom, up):
12         """Retorna um inteiro aleatorio impar entre a faixa
13             [bottom, up]
14
15         Args:
16             bottom: menor valor da faixa de valores
17             up: maior valor da faixa de valores
18
19         Returns:
20             Inteiro impar
21         """
22         i = 2
23         while i % 2 == 0:
24             i = random.randint(bottom, up)
25         return i

```

```

1  """Testes unitarios dos codigos desenvolvidos na geracao de
   numeros primos"""
2
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 !=
14                             0)
15
16     # mr
17     def test_right_decomposite(self):
18         png = mr(10, 100, 10000)
19         self.assertEqual(png.decomposite(12), (2, 3))
20
21     def test_wrong_decomposite(self):
22         png = mr(10, 100, 10000)
23         self.assertNotEqual(png.decomposite(186), (2, 5))
24
25     def test_true_primality_mr(self):
26         png = mr(10, 100, 10000)
27         self.assertTrue(png.primality_test(12, (2, 3)))
28
29     def test_false_primality_mr(self):
30         png = mr(10, 100, 10000)
31         self.assertFalse(png.primality_test(26, (1, 13)))
32
33     # fermat
34     def test_true_primality_fermat(self):
35         png = fermat(10, 100, 10000)
36         self.assertTrue(png.primality_test(8837))
37
38     def test_false_primality_fermat(self):
39         png = fermat(10, 100, 10000)
40         self.assertFalse(png.primality_test(297))
41
42     # lucas
43     def test_right_prime_factors(self):
44         png = lucas(10, 100, 10000)
45         self.assertEqual(png.prime_factors(75), [3, 5, 5])
46
47     def test_wrong_prime_factors(self):

```



```

47         png = lucas(10, 100, 10000)
48         self.assertNotEqual(png.prime_factors(17), [1, 2,
49                               17])
50
51     def test_true_primality_lucas(self):
52         png = lucas(10, 100, 10000)
53         self.assertTrue(png.primal_test(1249))
54
55     def test_false_primality_lucas(self):
56         png = lucas(10, 100, 10000)
57         self.assertFalse(png.primal_test(8826))
58
59 if __name__ == '__main__':
60     unittest.main()

```

2 Sobre a geração

Com a proposta no enunciado do trabalho: “Uma forma de se gerar números primos é primeiro gerar um número aleatório ímpar (grande) e depois testá-lo para saber se é primo. Caso não seja, gera-se outro número aleatório até que seja primo.”. Portanto, essa foi a metodologia utilizada para a geração de números primos, onde a parte de verificação de primalidade foi realizada de três diferentes formas: Miller-Rabin, Fermat e Lucas.

Para a geração de um número aleatório ímpar (grande) havia duas opções:

- trabalhar com o código desenvolvido no trabalho anterior, ou
- utilizar o módulo `random` do *Python*.

Optou-se pela segunda opção, pois acredito que ela possua um melhor desempenho, já que o módulo já é desenvolvido há vários anos por diversos programadores diferentes. Além disso, a escolha de Miller-Rabin, Fermat e Lucas foi feita com base na popularidade desses métodos, pois eles estão entre os que possuem mais material disponível na internet.

O teste de primalidade de Miller-Rabin foi inicialmente desenvolvido como determinístico por Gary L. Miller, mas depois foi modificado para ser um algoritmo probabilístico por Michael O. Rabin. Ele se baseia na propriedade do pseudoprime forte.

Já a verificação de primalidade de Fermat é baseada no pequeno teorema de Fermat. Esse teorema diz que se n é primo e a não é divisível por p , então:

$$a^{n-1} \equiv 1 \pmod{n} \quad (1)$$

Por fim, o teste de Lucas, trabalho do matemático Francês de mesmo nome no século XIX, também realiza uma verificação probabilística, mas por outro lado necessita que os fatores primos de $n - 1$ já sejam conhecidos, onde n é o número que será verificado.

Por se tratarem de verificações probabilísticas e não determinísticas, todos os testes implementados podem indicar que um número composto é um número primo, porém a chance disso acontecer pode ser minimizada com o aumento de iterações nos algoritmos. Com essa característica é importante frisar que os números gerados são possivelmente primos.

3 Comparação entre os algoritmos

Com a verificação do Miller-Rabin foi possível gerar um número primo em todos os tamanhos solicitados. Já com a verificação de Fermat foi possível gerar um número primo com quase todos os tamanhos solicitados, só ficou impraticável para 4096 bits, onde ocorreu de se passar 15 minutos e o programa não parar a execução. Por fim, com a verificação de Lucas só foi possível gerar um número primo com até 128 bits, porque com valores acima disso (e.g. 168) o tempo ficou muito longo, mais que 15 minutos, e, também nada do programa parar.

Imagino que o motivo da demora no Fermat esteja associado com a utilização do `math.gcd(a, n)` na linha 42. Já no Lucas, acredito que a lentidão deve estar associada com a utilização de um módulo externo em `pf.primefac(n)` na linha 38. Por fim, na tabela 1 há o detalhamento dos testes realizados.

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](#)

7 Tabelas

Tabela 1: 1 número primo grande

Gerador	Tamanho do número em bits	Tempo gasto
Lucas	40	0m0,040s
Fermat	40	0m0,037s
Miller-Rabin	40	0m0,050s
Lucas	56	0m0,097s
Fermat	56	0m0,043s
Miller-Rabin	56	0m0,063s
Lucas	80	0m2,927s
Fermat	80	0m0,047s
Miller-Rabin	80	0m0,063s
Lucas	128	12m18,283s
Fermat	128	0m0,050s
Miller-Rabin	128	0m0,050s
Lucas	168	IMPRATICÁVEL
Fermat	168	0m0,050s
Miller-Rabin	168	0m0,077s
Fermat	224	0m0,053s
Miller-Rabin	224	0m0,050s
Fermat	256	0m0,073s
Miller-Rabin	256	0m0,087s
Fermat	512	0m0,493s
Miller-Rabin	512	0m0,390s
Fermat	1024	0m2,693s
Miller-Rabin	1024	0m1,533s
Fermat	2048	0m26,563s
Miller-Rabin	2048	0m5,683s
Fermat	4096	IMPRATICÁVEL
Miller-Rabin	4096	0m18,910s