Relatório INE5429
Segurança em Computação
Números Primos

Bruno Marques do Nascimento*

14 de Abril de 2018

Algoritmos:

1 Teste de primalidade de Miller-Rabin

O teste de primalidade de Miller-Rabin é um algoritmo que testa se um dado número n é primo probabilisticamente. Este teste utiliza-se de um conjunto de igualdades que são válidas para números primos, e que são utilizadas para testar a primalidade de n. Ele se baseia no lema de Euclides e na contrapositiva do lema de Fermat. Sendo assim, ao realizar um teste sobre o número n se for encontrado um a, tal que $a^d \not\equiv 1 \pmod{n}$ e $a^{2^r d} \not\equiv -1 \pmod{n}$ para todo $0 \leq r \leq s-1$, então n não é primo.

Tabela 1 – Miller-Rabin (6 execuções)

Tamanho do Número (bits)	**Tempo médio para gerar (microssegundos)**	**Desvio Padrão (microssegundos)**
40	2437,00	1576,05
56	1233,34	3544,45
80	9227,64	6484,48
128	14064,55	34214,85
168	42083,98	44683,58
224	227973,35	172281,57
256	61404,35	81073,24
512	1023952,25	1716422,89
1024	12457123,04	20190124,49
2048	171019203,55	643825280,23
4096	1044962651,26	1266221570,34

*brunomn95@gmail.com - Universidade Federal de Santa Catarina

miller-rabin.py

```
import random
import sys
import time as t

MICRO = 1000000

def miller_rabin(n, k=40):
    # Casos <= 4
    if (n <= 1):     return False
    if (n <= 3):     return True
    if (n == 4):     return False
    if (n % 2 == 0): return False

    # Encontre inteiros s, d, com s > 0, d ímpar, de modo que (n - 1 = 2^s * d);
    d = n - 1
    s = 0
    while d % 2 == 0:
        d //= 2;
        s += 1

    # Laço responsável por realizar k testes
    for _ in range(k):
        # Selecione um inteiro aleatório a, tal que 1 < a < n - 1;
        a = random.randrange(2, n-1)

        # if a^d mod n = 1 then return("inconclusive");
        x = pow(a, d, n)
        if x == 1 or x == n-1:
            return True

        # Laço que irá executar 's-1' vezes.
        for _ in range(s-1):
            x = pow(x,2,n)
            if x == 1:
                return False
            if x == n-1:
                return True

    return False

nbits = int(sys.argv[1]) # Número de bits, definido pelo usuário

start = t.time() # Tempo de início da execução.
while True:
    # Número aleatório com N bits
    n = random.getrandbits(nbits)
    if miller_rabin(n) == True:
        break
end = t.time()  # Tempo de término da execução.

total_time = (end - start) * MICRO

print("Random value = " + str( n ))
print("%.2f" % total_time + " microssegundos")
```

2 Teste de primalidade de Fermat

Assim como o **Teste de primalidade de Miller-Rabin**, o de Fermat também determina probabilisticamente quando um dado número n é primo ou não. O teorema de Fermat diz que se n é primo e a não é divisível por n, então $a^{n-1} \equiv 1 \pmod{n}$, dessa maneira para testarmos se n é primo, devemos selecionar valores aleatórios para a que não são divisíveis por n, e observar se a equivalência se mantém verdadeira. Caso ela não se mantenha verdadeira para um valor de a, então n não é primo. Assim, caso essa equivalência se mantenha verdadeira para um ou mais valores de a, n é provavelmente primo.

Tabela 2 – Fermat (6 execuções)

Tamanho do Número (bits)	Tempo médio para gerar (microssegundos)	Desvio Padrão (microssegundos)
40	415,56	110,24
56	566,00	177,19
80	1048,92	309,24
128	3080,24	1041,94
168	7325,17	2426,65
224	16478,18	28404,10
256	12502,55	20410,04
512	126065,61	50266,13
1024	1056584,59	1100742,88
2048	19111389,87	28232101,87
4096	196762201,78	118842421,62

fermat.py

```
import random
import sys
import time as t

MICRO = 1000000

def fermat(n, k=40):
# Casos <= 4
    if (n <= 1):     return False
    if (n <= 3):     return True
    if (n == 4):     return False
    if (n % 2 == 0): return False

    for _ in range(k):
        a = random.randrange(2, n-1)

        if pow(a, n-1, n) != 1:
            return False

    return True

nbits = int(sys.argv[1]) # Número de bits, definido pelo usuário

start = t.time() # Tempo de início da execução.
while True:
    # Número aleatório com N bits
    n = random.getrandbits(nbits)
    if fermat(n) == True:
        break
end = t.time()  # Tempo de término da execução.
```

```
total_time = (end - start) * MICRO

print("Random value = " + str( n ))
print("%.2f" % total_time + " microssegundos")
```

3 Comparação e complexidade

Ambos algoritmos implementados são muito parecidos, são testes de primalidade probabilisticas, que irão fornecer números primos que podem posteriormente ser testado de maneira mais incisiva. O **Teste de primalidade de Miller-Rabin** possui complexidade de $O(k \cdot log^3 n)$ onde k é a quantidade de diferentes valores para a testados. (WIKIPEDIA, b). E o **Teste de primalidade de Fermat** possui complexidade $\tilde{O}(k \cdot log^2 n)$, (WIKIPEDIA, a).

4 Dificuldades encontradas

Devido a complexidade dos algoritmos, o tempo total para gerar um número e realizar o teste de primalidade sobre ele torna-se demorado quando o número de *bits* começa a aumentar, podendo tomar mais de 30 minutos, por exemplo, ao tentar gerar um número primo de 4096 *bits*. Outro fato relevante, é que este tempo é dependente da "sorte" de gerar um numero aleatório que seja primo para que o teste convirja rapidamente, acarretando em um elevado desvio padrão.

5 Execuções

Tabela 3 – Execuções Miller-Rabin (microssegundos)

BITS\EXECUÇÃO	1	2	3	4	5	6	MEDIA	STD DEV
40	2339,84	2743,96	1082,66	4654,65	2534,15	26,46	2437,00	1576,05
56	9079,69	643,25	23,84	166,42	4580,97	1823,43	1233,34	3544,45
80	19163,61	6067,28	16526,94	2106,43	7554,53	10900,74	9227,64	6484,48
128	74996,23	544,31	17953,16	10175,94	2077,1	70564,03	14064,55	34214,85
168	50626,75	33541,2	128947,97	18716,81	15667,68	89344,02	42083,98	44683,58
224	366577,39	473574,16	343026,88	105931,76	57917,36	112919,81	227973,35	172281,57
256	25068,76	187540,53	212288,38	54706,34	68102,36	37176,85	61404,35	81073,24
512	296193,36	371868,37	1380643,61	4848669,29	1838359,83	667260,89	1023952,25	1716422,89
1024	31207163,33	1754965,07	4173250,68	10158452,99	14755793,09	54736176,01	12457123,04	20190124,49
2048	1748714846,37	132029213,43	107674439,67	171298158,41	170740248,68	340060755,01	171019203,55	643825280,23
4096	374507446,53	1224821036,1	865104266,41	378419603,59	2000634170,77	3690580049,04	1044962651,26	1266221570,34

Tabela 4 – Execuções Fermat (microssegundos)

BITS\EXECUÇÃO	1	2	3	4	5	6	MEDIA	STD DEV
40	443,7	375,03	387,43	383,14	664,71	478,03	415,57	110,24
56	552,18	579,83	927,45	463,96	511,17	765,32	566,01	177,19
80	1654,39	1067,16	869,51	1472,95	990,63	1030,68	1048,92	309,24
128	3141,4	3112,79	2456,9	2285,24	5201,34	3047,7	3080,25	1041,94
168	7531,4	5510,81	11207,1	4567,86	9233,47	7118,94	7325,17	2426,65
224	19263,03	83230,97	11532,31	7150,41	13693,33	28469,09	16478,18	28404,10
256	12604	8593,08	29173,37	61172,49	12401,1	10271,55	12502,55	20410,04
512	109261,27	142869,95	182988,41	207906,96	96101,76	82098,01	126065,61	50266,13
1024	3349493,98	2081247,09	909789,8	356384,52	1203379,39	740375,28	1056584,60	1100742,88
2048	26155449,63	1641473,53	58278342,25	12067330,12	5318920,37	68050973,89	19111389,88	28232101,87
4096	208888615,61	302012347,7	35766744,14	10214559,56	184635787,96	259303761,01	196762201,79	118842421,62

Referências

GEEKSFORGEEKS. *Primality Test | Set 3 (Miller–Rabin)*. [S.l.]. Disponível em: <https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/>. Acesso em: 29 abril 2018. Nenhuma citação no texto.

STALLINGS, W. *Cryptography and Network Security: Principles and Practice*. 6th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2013. ISBN 0133354695, 9780133354690. Nenhuma citação no texto.

WIKIPEDIA. *Fermat primality test*. [S.l.]. Disponível em: <https://en.wikipedia.org/wiki/Fermat_primality_test>. Acesso em: 29 abril 2018. Citado na página 4.

WIKIPEDIA. *Miller–Rabin primality test*. [S.l.]. Disponível em: <https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>. Acesso em: 29 abril 2018. Citado na página 4.