Trabalho individual sobre raízes primitivas

INE5429 - Segurança em Computação

Aluno: Fernando Paladini

Seja *n* um número primo, então *g* é uma **raiz primitiva módulo *n*** se para todo inteiro *a* coprimo a *n*, existe um inteiro k∈Z* tal que $g^k \equiv a \pmod{n}$, onde Z* é o conjunto dos inteiros excluindo-se o zero e um inteiro *b* é coprimo a um inteiro *c* caso mdc(b,c) = 1. Note que as potências de *g*, no caso de *n* ser um número primo, não podem ser maiores do que *n - 1*, pois um "ciclo" é formado, algo que pode ser notado pelo Pequeno Teorema de Fermat. Um número primo *n*, por definição, não possui nenhum divisor além de 1 e dele próprio, assim sendo, todos os números no intervalo de 1 até n – 1 são coprimos a ele. Dessa forma, podemos afirmar que ϕ(n) = n – 1. Alguns exemplos de raízes primitivas módulo n podem ser encontrados abaixo.

3 é uma raíz primitiva módulo 7:

$$
\begin{aligned}
3^1 &= 3 = 3^0 \times 3 \equiv 1 \times 3 = 3 \equiv 3 \pmod 7 \\
3^2 &= 9 = 3^1 \times 3 \equiv 3 \times 3 = 9 \equiv 2 \pmod 7 \\
3^3 &= 27 = 3^2 \times 3 \equiv 2 \times 3 = 6 \equiv 6 \pmod 7 \\
3^4 &= 81 = 3^3 \times 3 \equiv 6 \times 3 = 18 \equiv 4 \pmod 7 \\
3^5 &= 243 = 3^4 \times 3 \equiv 4 \times 3 = 12 \equiv 5 \pmod 7 \\
3^6 &= 729 = 3^5 \times 3 \equiv 5 \times 3 = 15 \equiv 1 \pmod 7
\end{aligned}
$$

2 é uma raiz primitiva módulo 5:

$$
\begin{aligned}
2^0 = 1,\ 1 \pmod 5 = 1, \text{so } 2^0 \equiv 1 \\
2^1 = 2,\ 2 \pmod 5 = 2, \text{so } 2^1 \equiv 2 \\
2^3 = 8,\ 8 \pmod 5 = 3, \text{so } 2^3 \equiv 3 \\
2^2 = 4,\ 4 \pmod 5 = 4, \text{so } 2^2 \equiv 4
\end{aligned}
$$

2 é uma raíz primitiva módulo 13:

$$
\begin{aligned}
2^0 &= 1 = 2^0 \equiv 1 = 1 \equiv 1 \pmod{13} \\
2^1 &= 2 = 2^0 \cdot 2 \equiv 1 \cdot 2 = 2 \equiv 2 \pmod{13} \\
2^2 &= 4 = 2^1 \cdot 2 \equiv 2 \cdot 2 = 4 \equiv 4 \pmod{13} \\
2^3 &= 8 = 2^2 \cdot 2 \equiv 4 \cdot 2 = 8 \equiv 8 \pmod{13} \\
2^4 &= 16 = 2^3 \cdot 2 \equiv 8 \cdot 2 = 16 \equiv 3 \pmod{13} \\
2^5 &= 32 = 2^4 \cdot 2 \equiv 3 \cdot 2 = 6 \equiv 6 \pmod{13} \\
2^6 &= 64 = 2^5 \cdot 2 \equiv 6 \cdot 2 = 12 \equiv 12 \pmod{13} \\
2^7 &= 128 = 2^6 \cdot 2 \equiv 12 \cdot 2 = 24 \equiv 11 \pmod{13} \\
2^8 &= 256 = 2^7 \cdot 2 \equiv 11 \cdot 2 = 22 \equiv 9 \pmod{13} \\
2^9 &= 512 = 2^8 \cdot 2 \equiv 9 \cdot 2 = 18 \equiv 5 \pmod{13} \\
2^{10} &= 1024 = 2^9 \cdot 2 \equiv 5 \cdot 2 = 10 \equiv 10 \pmod{13} \\
2^{11} &= 2048 = 2^{10} \cdot 2 \equiv 10 \cdot 2 = 20 \equiv 7 \pmod{13} \\
2^{12} &= 4096 = 2^{11} \cdot 2 \equiv 7 \cdot 2 = 14 \equiv 1 \pmod{13}
\end{aligned}
$$

Implementação em Python (testado apenas na versão 3.5.2) do algoritmo solicitado:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

'''
Os métodos implementados permitem encontrar as raízes primitivas de um
inteiro n.
Utiliza o método de cálculo de MDC disponível na biblioteca padrão do
Python (fractions.gcd).

Como usar:
    python raizes.py <n_desejado>

O programa fará os cálculos e então imprimirá na tela as raízes primitivas
do <n_desejado> inserido. Exemplo:

    python raizes.py

'''

from random import randrange
from fractions import gcd
import sys

def num_coprimos(n):
    '''
    Função que calcula o totiente de Euler de n (ou o número de coprimos
de 'n').
    '''
    num_coprimos = 0
    for i in range(n):
        if (gcd(i, n) == 1):
            num_coprimos += 1
    return num_coprimos

def fatores_primos(n):
    '''
    Função que calcula todos os fatores primos de 'n' (até que a sua raiz
quadrada seja atingida).
    Retorna uma lista ordenada dos fatores primos de 'n'.
    '''
    fatores = set()
    i = 2

    while (i**2 <= n):
        if (n % i == 0):
            n = n // i
            fatores.add(i)
        else:
            i += 1
    fatores.add(n)
    return sorted(fatores)

def raizes_primitivas(n):
    '''
    Função que encontra todas as raízes primitivas de um inteiro n.
    '''
    p = num_coprimos(n)
```

```
    fatores = fatores_primos(p)
    limite = num_coprimos(p)
    raizes_p = set()

    while (len(raizes_p) != limite):
        a = randrange(1, n)
        if (all(pow(a, p // f, n) != 1 for f in fatores)    ):
            raizes_p.add(a)

    return sorted(raizes_p)

if (__name__ == "__main__"):

    # Verificando se existe algum argumento para o programa
    if (len(sys.argv) <= 1):
        print("Por favor, insira um valor numerico n.")
        sys.exit(0)

    n = int(sys.argv[1])
    raizes_primitivas = raizes_primitivas(n)

    print("As raízes primitivas de {} são:".format(n))
    print(raizes_primitivas)
```

Ora, sabemos que para encontrar as raízes primitivas de um número primo *g*, podemos gerar todas as potências de *g* na forma $g^m \equiv 1 \pmod{n}$, para *m* com valores dentro do intervalo [1, n – 1]. Para estes números serem raízes primitivas, temos que $(a^m)^{(n-1)/d} \equiv (a^{n-1})^{m/d} \equiv 1 \pmod{n}$, logo precisamos de um *d* cujo valor seja igual a 1. Para encontrar as raízes primitivas no algoritmo Python proposto acima, utilizam-se números aleatórios de forma a realizar alguns chutes dos valores possíveis para a raiz primitiva. Dado que $a^{(p-1)/q} \not\equiv 1 \pmod{p}$ para todos os fatores primos de p – 1, podemos verificar se *a* é uma raiz primitiva módulo *n* (construindo então uma lista de todas as suas raízes primitivas).Utilizando o algoritmo, podemos obter alguns valores para as perguntas realizadas no enunciado do trabalho.

O número primo 1013 tem tem 440 raízes primitivas, que são:

C:\Users\tulio\Desktop\raizes_primitivas>python raizes.py 1013

As raízes primitivas de 1013 são:

[3, 5, 7, 12, 17, 18, 20, 26, 27, 28, 29, 30, 31, 33, 37, 38, 39, 41, 42, 47, 48, 50, 55, 57, 59, 61, 63, 67, 68, 69, 70, 72, 75, 77, 80, 86, 91, 98, 101, 103, 104, 105, 106, 107, 108, 109, 112, 115, 116, 120, 124, 125, 129, 131, 132, 133, 137, 139, 142, 146, 147, 148, 151, 152, 153, 156, 158, 161, 162, 164, 166, 168, 170, 175, 178, 179, 186, 188, 191, 192, 194, 198, 200, 213, 215, 219, 220, 221, 222, 226, 227, 228, 234, 236, 237, 238, 239, 243, 244, 245, 246, 249, 252, 254, 255, 260, 261, 263, 265, 267, 268, 269, 270, 272, 276, 279, 280, 282, 286, 290, 291, 297, 298, 300, 301, 307, 308, 310, 313, 314, 317, 320, 323, 326, 330, 333, 334, 338, 339, 341, 342, 343, 344, 346, 349, 351, 354, 355, 357, 359, 362, 363, 364, 365, 366, 369, 370, 371, 378, 381, 383, 386, 389, 390, 391, 392, 394, 398, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 412, 414, 415, 418, 420, 421, 422, 423, 428, 429, 432, 434, 435, 436, 439, 443, 445, 447, 448, 450, 451, 457, 458, 460, 462, 463, 464, 465, 466, 467, 470, 471, 480, 481, 482, 489, 494, 496, 497, 499, 500, 502, 511, 513, 514, 516, 517, 519, 524, 531, 532, 533, 542, 543, 546, 547, 548, 549, 550, 551, 553, 555, 556, 562, 563, 565, 566, 568, 570, 574, 577, 578, 579, 581, 584, 585, 590, 591, 592, 593, 595, 598, 599, 601, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 615, 619, 621, 622, 623, 624, 627, 630, 632, 635, 642, 643, 644, 647, 648, 649, 650, 651, 654, 656, 658, 659, 662, 664, 667, 669, 670, 671, 672, 674, 675, 679, 680, 683, 687, 690, 693, 696, 699, 700, 703, 705, 706, 712, 713, 715, 716, 722, 723, 727, 731, 733, 734, 737, 741, 743, 744, 745, 746, 748, 750, 752, 753, 758, 759, 761, 764, 767, 768, 769, 770, 774, 775, 776, 777, 779, 785, 786, 787, 791, 792, 793, 794, 798, 800, 813, 815, 819, 821, 822, 825, 827, 834, 835, 838, 843, 845, 847, 849, 851, 852, 855, 857, 860, 861, 862, 865, 866, 867, 871, 874, 876, 880, 881, 882, 884, 888, 889, 893, 897, 898, 901, 904, 905, 906, 907, 908, 909, 910, 912, 915, 922, 927, 933, 936, 938, 941, 943, 944, 945, 946, 950, 952, 954, 956, 958, 963, 965, 966, 971, 972, 974, 975, 976, 980, 982, 983, 984, 985, 986, 987, 993, 995, 996, 1001, 1006, 1008, 1010]

As raízes primitivas de 23 são:

```
C:\Users\tulio\Desktop\raizes_primitivas>python raizes.py 23
As raízes primitivas de 23 são:
[5, 7, 10, 11, 14, 15, 17, 19, 20, 21]
```

Dentre as aplicações possíveis para as raízes primitivas, as principais estão relacionadas a criptografia. Um exemplo particular deste ramo é o uso de raízes primitivas módulo no esquema de troca de chaves conhecido como Diffie-Hellman. Nesse método, que é extremamente popular dentro da segurança em computação, os autores utilizaram o número primo e uma de suas raízes primitivas para calcular a chave privada final. Além deste uso dentro da criptografia, não encontrei outros usos que não seja na matemática ou dentro da computação, embora tenha lido em alguns fóruns que existem áreas que se utilizam de tal artefato matemático.