NumberTheory with SageMath

Following exercises are from *Fundamentals of Number Theory* written by *Willam J. Leveque*.

Chapter 1

p. 5

`prime_pi(x)`: the number of prime numbers that are less than or equal to x. (same as $\pi(x)$ in textbook.)

```
sage: prime_pi(10)
4
sage: prime_pi(10^3)
168
sage: prime_pi(10^10)
455052511
```

Also, you can see

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\log(x)} = 1$$

with following code.

```
sage: for i in range(4, 13):
....:     print "%.3f" % (prime_pi(10^i) * log(10^i) / 10^i)
....:
1.132
1.104
1.084
1.071
1.061
1.054
1.048
1.043
1.039
```

p. 7

`divisors(n)`: the divisors of n.
`number_of_divisors(n)`: the number of divisors of n ($= \tau(n)$).

```
sage: divisors(12)
[1, 2, 3, 4, 6, 12]
sage: number_of_divisors(12)
6
```

For example, you may construct a table of values of $\tau(n)$ for $1 \leq n \leq 5$ as follows:

```
sage: for i in range(1, 6):
....:     print "t(%d) : %d" % (i, number_of_divisors(i))
....:
t(1) : 1
t(2) : 2
t(3) : 2
t(4) : 3
t(5) : 2
```

p. 19

Division theorem: If a is positive and b is any integer, there is exactly one pair of integers q and r such that the conditions

$$b = aq + r \qquad (0 \leq r < a)$$

holds.
In SageMath, you can get quotient and remainder of division by '`a // b`' and '`a % b`'.

```
sage: -7 // 3
-3
sage: -7 % 3
2
sage: -7 == 3 * (-7  // 3) + (-7 % 3)
True
```

p. 21

`factor(n)`: prime decomposition for integer n.

```
sage: factor(12)
2^2 * 3
sage: for p, e in factor(12):
....:     print p, e
....:
2 2
3 1
```

p. 24

`sigma(n, k)`: The sum of k^{th} powers of the positive divisors of $|n|$. For example,

$$sigma(12, 3) = 1^3 + 2^3 + 3^3 + 4^3 + 6^3 + 12^3 = 2044.$$

```
sage: sigma(12, 2)
210
sage: sigma(12, 3)
2044
```

Note that we can rewrite `sigma(n, k)` as

$$\prod_{i=1}^{m}(1^k + p_i^k + \cdots p_i^{ke_i})$$

when $n = p_1^{e_1} \cdots p_m^{e_m}$.

```
sage: def my_sigma(n, k):
....:     prod = 1
....:     for p, e in factor(n):
....:         psum = 0
....:         for i in range(0, e + 1):
....:             psum += p^(k * i)
....:         prod *= psum
....:     return prod
....:
sage: my_sigma(12, 3) == sigma(12, 3)
True
```

Chapter 2

p. 32

`gcd(n, m)`: Finds the greatest common divisor of n and m.

```
sage: gcd(15, 21)
3
```

p. 33

`xgcd(n, m)`: Finds *bezout's coefficient* and the GCD of n and m.

```
sage: g, a, b = xgcd(123, 54)
sage: gcd(123, 54) == g
True
sage: a * 123 + b * 54 == g
True
```

Note that we can find *bezout's coefficient* with *extended euclidean algorithm*.

```
sage: def my_xgcd(n, m):
....:     if n == 0:
....:         return (m, 0, 1) if m >= 0 else (-m, 0, -1)
....:     else:
....:         g, x, y = my_xgcd(m % n, n)
....:         return (g, y - (m // n) * x, x)
....:
sage: g, a, b = my_xgcd(123, 54)
sage: gcd(123, 54) == g
True
sage: a * 123 + b * 54 == g
True
```

Also you can see how euclidean algorithm works with following code.

```
sage: def my_euclidean(n, m):
....:     if abs(m) < abs(n):
....:         m, n = n, m
....:     r = m % n
....:     while r != 0:
....:         print "%d = %d * %d + %d" % (m, n, m // n, r)
....:         m, n = n, r
....:         r = m % n
....:     print "%d = %d * %d" % (m, n, m // n)
....:
sage: my_euclidean(741, 715)
741 = 715 * 1 + 26
715 = 26 * 27 + 13
26 = 13 * 2
```

Note that $gcd(741, 715) = 13$. The implementation of *extended euclidean algorithm* with *least remainder*(**p. 35**) is left for readers.

p. 44

`lcm(n, m)`: Finds the least common multiplier for n, m.

```
sage: lcm(15, 21)
105
```

You can check

$$gcd(n, m) \times lcm(n, m) = n \times m.$$

```
sage: gcd(15, 21) * lcm(15, 21) == 15 * 21
True
```

Chapter 3

p. 50

`IntegerModRing(n)`: Declare residue class with modulo n.

```
sage: R = IntegerModRing(10)
sage: R(11)
1
sage: R(13)
3
sage: R(3) + R(8)
1
sage: R(3) * R(8)
4
sage: R(3)^-1
7
```

You can implement your own modular inverse function

`invmod(a, n)`: Find b such that $a \times b \equiv 1 \pmod{n}$

with extended euclidean algorithm.

p. 53

`eulder_phi(n)`: Computes the *Euler's totient function* $\phi(n)$.

```
sage: euler_phi(17)
16
sage: euler_phi(36)
12
sage: 3^euler_phi(10) % 10 == 1
True
```

Note that $a^{\phi(n)} \equiv 1 \pmod{n}$ when $(a, n) = 1$.

p. 54

You can check **Theorem 3.8** as following.

```
sage: def my_euler_phi(n):
....:     res = n
....:     for p, e in factor(n):
....:         res *= (1 - 1/p)
....:     return res
....:
sage: my_euler_phi(24) == euler_phi(24)
True
```

p. 60

`crt(a, b, m, n)`: Find a solution x to the congruences $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$.
`crt(rs, ms)`: Find a simultaneous solution x to the congruences $x \equiv r_i \pmod{m_i}$.

```
sage: crt([2, 3], [3, 4])
11
sage: crt(2, 3, 3, 4)
11
sage: crt([3, 5, 7], [4, 21, 25])
1307
```

Note that 1307 is a solution of following system of linear modular equations:

$$\begin{aligned} x &\equiv 3 \pmod{4} \\ x &\equiv 5 \pmod{21} \\ x &\equiv 7 \pmod{25} \end{aligned}$$

p. 63

`IntegerModRing(n)[]`: Declare polynomial ring of residue class with modulo n.

```
sage: R.<x> = IntegerModRing(5)[]
sage: (x - 1) * (x - 2) == (x + 4) * (x + 3)
True
```

Also, you can use `roots()` to find roots of the polynomial.

```
sage: R.<x> = IntegerModRing(5)[]
sage: ((x - 5) * (x - 1)^2 * (x - 2)^3).roots()
[(0, 1), (1, 2), (2, 3)]
```

Chapter 4

p. 79

`primitive_root(q)`: Calculates a primitive roots of q where q is 2, 4, p^n or $2p^n$ for a positive integer n and a prime p.

```
sage: primitive_root(17)
3
```

3 is primitive root in $Z/17Z$ and we have

$$3^0 = 1,\quad 3^1 = 3,\quad 3^2 = 9,\quad 3^3 = 10,\ 3^4 = 13,\ 3^5 = 5,$$
$$3^6 = 15,\ 3^7 = 11,\quad 3^8 = 16,\ 3^9 = 14,\ 3^{10} = 8,\ 3^{11} = 7,$$
$$3^{12} = 4,\ 3^{13} = 12,\ 3^{14} = 2,\ 3^{15} = 1,\ 3^{16} = 1$$

Also, you can find discrete logarithm of x to the base g in Z/qZ as following.

```
sage: p = 17
sage: R = IntegerModRing(p)
sage: g = primitive_root(p)
sage: x = R(g^10)
sage: x.log(g)
10
```

p. 90

`nth_root(n)`: Find n-th roots of object if it exists.

```
sage: R = IntegerModRing(17)
sage: g = R(primitive_root(17))
sage: g.nth_root(3)^3 == g
True
sage: g.nth_root(5)^5 == g
True
```

Chapter 5

p. 99

`kronecker(a, n)`: Calculates the *Legendre symbol* or it's generelization, *kronecker symbol*.

```
sage: kronecker(2, 17)
1
sage: kronecker(14, 17)
-1
sage: quadratic_residues(23)
[0, 1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18]
sage: [x for x in range(23) if kronecker(x, 23) == 1]
[1, 2, 3, 4, 6, 8, 9, 12, 13, 16, 18]
```

You can check multiplicative property of *Legendre symbol* by

```
sage: def my_kronecker(a, n):
....:     prod = 1
....:     for p, e in factor(a):
....:         if e % 2 == 0:
....:             continue
....:         else:
....:             prod *= kronecker(p, n)
....:     return prod
....:
sage: my_kronecker(48, 31) == kronecker(48, 31)
True
sage: my_kronecker(48, 37) == kronecker(48, 37)
True
```

To check the mulitiplicative property of *Jacobi symbol*,

```
sage: kronecker(2, 3) * kronecker(2, 5) == kronecker(2, 3 * 5)
True
```

Chapter 6

p. 127

`moebius(n)`: Calculate *Moebius function* $\mu(n)$. $\mu(n) = 0$ if n is not square free, and otherwise equals $(-1)^r$, where n has r distinct prime factors. For simplicity, $\mu(0) = 0$ and $\mu(1) = 1$.

```
sage: moebius(12)
0
sage: moebius(-5)
-1
sage: moebius(-35)
1
sage: moebius(-7)
-1
sage: moebius(4)
0
```

p. 131

`floor(x)`: Find the largest integer not suceeding x. (Same as $[x]$)
`ceil(x)`: Find the smallest integer not preceding x.

```
sage: ceil(5.5)
6
sage: ceil(5)
5
sage: floor(5.5)
5
sage: floor(5)
5
```

p. 154

`zeta(s)`: The *Riemann zeta function*.

```
sage: zeta(1)
Infinity
sage: zeta(2)
1/6*pi^2
sage: zeta(2.)
1.64493406684823
sage: zeta(RealField(200)(2))
1.64493406684822643647241516664602518921894990120679843773568
sage: zeta(0)
-1/2
```

Chapter 7

p. 187

`three_squares(n)`: Write the integer n as a sum of three integer squares if possible.

```
sage: three_squares(389)
(1, 8, 18)
sage: three_squares(946)
(9, 9, 28)
sage: three_squares(2986)
(3, 24, 49)
```

`four_squares(n)`: Write the integer n as a sum of four integer squares.

```
sage: four_squares(3)
(0, 1, 1, 1)
sage: four_squares(13)
(0, 0, 2, 3)
sage: four_squares(130)
(0, 0, 3, 11)
sage: four_squares(1101011011004)
(90, 102, 1220, 1049290)
```

Chapter 8

p. 198 (Pell's equation)

Let $\frac{p_i}{q_i}$ denote the sequence of convergents to the regular continued fraction for $\sqrt{n}$. This sequence is unique. Then the pair (x, y) solving Pell's equation and minimizing x satisfies $x = p_i$ and $y = q_i$ for some i. This pair is called the fundamental solution. Thus, the fundamental solution may be found by performing the continued fraction expansion and testing each successive convergent until a solution to Pell's equation is found.

`continued_fraction(x)`: Find continued faction of given real number. You can put `nterms` to limit the number of terms.
`numerator(n)`: Find the numerator of the n-th partial convergent of continued fraction.
`denominator(n)`: Find the numerator of the n-th partial convergent of continued fraction.

```
sage: continued_fraction(13/27)
[0; 2, 13]
sage: 0 + 1/(2 + 1/13)
13/27
sage: continued_fraction(sqrt(2))
[1; 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...]
sage: continued_fraction(sqrt(21))
[4; 1, 1, 2, 1, 1, 8, 1, 1, 2, 1, 1, 8, 1, 1, 2, 1, 1, 8, 1, ...]
sage: continued_fraction(pi)
[3; 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, ...]
sage: continued_fraction(pi, nterms = 5)
[3; 7, 15, 1, 292]
```

Also, you can solve *Pell's equation* by following:

```
sage: def solve_pell(N, numTry = 100):
....:     cf = continued_fraction(sqrt(N))
....:     for i in range(numTry):
....:         denom = cf.denominator(i)
....:         numer = cf.numerator(i)
....:         if numer^2 - N * denom^2 == 1:
....:             return numer, denom
....:     return None, None
....:
sage: solve_pell(21)
(55, 12)
sage: 55^2 - 21 * 12^2
1
```