Contents

Chapter 1

Functions

1.1 prime – 素数判定, 素数生成

1.1.1 trialDivision – 試し割り算

trialDivision(`n`**:** ***integer***, `bound`**:** ***integer/float*=0) →** ***True/False***

奇数に対する試し割り算.

`bound` は素数の探索範囲. もし `bound` が与えられ,`n` の平方根よりも小さいという条件のもと 1 を返せば, それは `bound` 以下に素因数がないことを意味する.

1.1.2 spsp – 強擬素数テスト

spsp(`n`**:** ***integer***, `base`**:** ***integer***, `s`**:** ***integer*=None,** `t`**:** ***integer*=None)**
→ ***True/False***

`base` を基にした強擬素数テスト.

`s` と `t` は $n-1 = 2^{\mathtt{s}}\mathtt{t}$ かつ `t` は奇数, となるような数.

1.1.3 smallSpsp – 小さい数に対する強擬素数テスト

smallSpsp(`n`**:** ***integer*) →** ***True/False***

10^{12} より小さい整数 `n` に対する強擬素数テスト.

4 回の強擬素数テストによって 10^{12} より小さい整数が素数かどうか決定するには十分なものである.

1.1.4 miller – Miller の素数判定

miller(n: *integer*) → *True/False*

Miller の素数判定.

このテストは GRH のもと有効です.config を見てください.

1.1.5 millerRabin – Miller-Rabin の素数判定

millerRabin(n: *integer*, times: *integer*=20) → *True/False*

Miller の素数判定.

miller との違いは,Miller-Rabin メソッドは早いが確率的なアルゴリズムであり, 一方で, miller は GRH のもと決定性アルゴリズムとなる.
times (初期設定は 20) は繰り返しの数. エラーの確率は多くても 4^{-times} となる.

1.1.6 lpsp – Lucas テスト

lpsp(n: *integer*, a: *integer*, b: *integer*) → *True/False*

Lucas 擬素数テスト.

もし n がパラメータ a と b の, すなわち,$x^2 - \mathtt{a}x + \mathtt{b}$ についての, Lucas 擬素数なら True を返す.

1.1.7 fpsp – Frobenius テスト

fpsp(n: *integer*, a: *integer*, b: *integer*) → *True/False*

Frobenius 擬素数テスト.

もし n がパラメータ a と b の, すなわち,$x^2 - \mathtt{a}x + \mathtt{b}$ についての, Frobenius 擬素数なら True を返す.

1.1.8 by_primitive_root – Lehmer's test

by_primitive_root(n: *integer*, divisors: *sequence*) → *True/False*

Lehmer の素数判定法 [2].

n が素数のときかつそのときに限り True を返す.
このメソッドは原始根の存在に基づいて n が素数であることを示す. このために $n-1$ の素因数を知っていることが必要である.

divisors は $n-1$ の素因数のシーケンス (list, tuple, etc).

1.1.9 full_euler – Brillhart & Selfridge's test

full_euler(n: *integer*, divisors: *sequence*) → *True/False*

Brillhart & Selfridge の素数判定法 [1].

このメソッドは $\varphi(n) = n-1$ の成立により n が素数であることを示す, ただし φ はオイラーのファイ関数 (**euler** を参照). このために $n-1$ の素因数を知っていることが必要である.

divisors は $n-1$ の素因数のシーケンス (list, tuple, etc).

1.1.10 apr – Jacobi 和テスト

apr(n: *integer*) → *True/False*

APR (Adleman-Pomerance-Rumery) 素数判定または Jacobi 和テストと呼ばれる判定法.

n は 32 より小さい素因数がないと仮定する. また n がいくつかの底に対する spsp (強擬素数テスト) を通過したと仮定する.

1.1.11 primeq – 自動的な素数判定

primeq(n: *integer*) → *True/False*

素数判定に対する便利な関数.

n のサイズに依存して **trialDivision**, **smallSpsp** または **apr** を使う.

1.1.12 prime – n 番目の素数

prime(n: *integer*) → *integer*

n 番目の素数を返す.

1.1.13 nextPrime – 次の素数を生成

nextPrime(n: *integer*) → *integer*

与えられた整数 n より大きい数の中で, 最も小さい素数を返す.

1.1.14 randPrime – ランダムに素数を生成

randPrime(n: *integer*) → *integer*

10 進 n 桁の素数をランダムに返す.

1.1.15 generator – 素数生成

generator((None)) → *generator*

2 から ∞ までの素数を生成する (ジェネレータとして).

1.1.16 generator_eratosthenes – Eratosthenes の篩を使っている素数生成

generator_eratosthenes(n: *integer*) → *generator*

Eratosthenes の篩を使って n までの素数を順に生成する.

1.1.17 primonial – 素数の積

primonial(p: *integer*) → *integer*

以下の積を返す

$$\prod_{q \in \mathbb{P}_{\leq \mathtt{p}}} q = 2 \cdot 3 \cdot 5 \cdots \mathtt{p}\ .$$

1.1.18　properDivisors – 真の約数

properDivisors(`n`: *integer*) → *list*

`n` の真の約数を返す (1 と `n` を除いた `n` の全ての約数).

小さな素数の積に対してのみ役に立つ．より一般的な場合には **proper_divisors** を使用.
出力は全ての真の約数のリスト.

1.1.19　primitive_root – 平方根

primitive_root(`p`: *integer*) → *integer*

`p` の平方根を返す.

`p` は奇素数でなければならない.

1.1.20　Lucas_chain – Lucas 数列

**Lucas_chain(`n`: *integer*, `f`: *function*, `g`: *function*, `x_0`: *integer*, `x_1`: *integer*)
→ (*integer*, *integer*)**

以下のように定義される数列 $\{x_i\}$ に対する値 (x_n, x_{n+1}) を返す:

$$x_{2i} = \mathtt{f}(x_i)$$
$$x_{2i+1} = \mathtt{g}(x_i, x_{i+1}) ,$$

初項は `x_0`, `x_1`.

`f` は 1 変数の整数関数. `g` は 2 変数の整数関数.

Examples

```
>>> prime.primeq(131)
True
>>> prime.primeq(133)
False
>>> g = prime.generator()
>>> g.next()
2
>>> g.next()
```

```
3
>>> prime.prime(10)
29
>>> prime.nextPrime(100)
101
>>> prime.primitive_root(23)
5
```

Bibliography

[1] J. Brillhart and J. L. Selfridge. Some factorizations of $2^n \pm 1$ and related results. *Math. Comp.*, Vol. 21, pp. 87–96, 1967.

[2] D. H. Lehmer. Tests for primality by the converse of Fermat's theorem. *Bull. Amer. Math. Soc.*, Vol. 33, pp. 327–340, 1927.