計算機プログラミング I (5)

配列を使ったアルゴリズム

2002 年 11 月 7 日　増原 英彦

配列を使うことで一覧表のようなものを使ったアルゴリズムをプログラムすることができるようになる。ここでは一覧表のようなものを使うアルゴリズムを 2 つ紹介する。

1　素数

問題: 正の整数 x が素数かを調べる

1.1　単純なアルゴリズム

単純には、2 から $(x-1)$ の間に x の約数があるかどうかを調べればよい。

アルゴリズム 1(素数): i を $2, 3, 4, \ldots, x$ と変化させ、x の約数である最小の数を見つける。その数が x であれば x は素数、x 未満であれば x は素数でない。 ∎

これをプログラムにすると以下のようになる:

```
1  // 整数 x が素数であるかどうかを調べる
2  int i = 2;
3  while ( i は x の約数か? ) {
4          i++;
5  }
6  // この時点で i は x の最小の約数
7  if (x==i) {
8      x が素数だったときの処理
9  }
```

練習 5-1: (素数)

- このアルゴリズムを用いて、2 から 10 万までの数について (1) 最大の素数と (2) 素数の個数を求めるプログラムを作れ。(参考: 資料 web ページ Prime.java)
- 上のプログラムを (a)2 から 2 万まで、(b)2 から 5 万まで、(c)2 から 10 万までについて調べるように変更し、それぞれの実行時間を測れ。さらに、100 万までについて調べるときの時間を予想せよ。

  プログラムの実行時間を測るには、実行の際にターミナルウィンドウ (kterm) で「`time java Prime` Return 」のように入力する。

1.2　エラトステネスのふるい

素数 p が 1 つ見つかったとき、素数の倍数 $2p, 3p, \ldots$ は素数でない。そこで、最初は全ての数を素数の候補としておき、小さい順に候補を見てゆき、素数を見つける度に素数の倍数を候補から消してゆけば、生き残った候補が素数になる。

このアルゴリズムは紀元前の学者エラトステネスによって考られたもので「エラトステネスのふるい (the sieve of Eratosthenes)」と呼ばれる。

アルゴリズム 2(エラトステネスのふるい):

- 2 から x までのそれぞれの数が「素数でない」ことを示す表を作る。最初は全ての数が「素数かも知れない」としておく
- i を 2 から $x-1$ まで順に変化させ、
  - 表の i 番目が「素数でない」であれば、素数でないので次の i の繰返しを行う
  - 表の i 番目が「素数かも知れない」であれば、i は素数である。表の $2i, 3i, \ldots$ 番目を「素数でない」に変える
- 最後に表の x 番目が「素数かも知れない」のままであれば x は素数である

最初の数回について「素数でないこと示す表」が変化していくかを見ると次のようになる:

- 最初は全てが「素数かも知れない」である
  $2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, \ldots$
- 2 が素数と分かるので、2 の倍数 $(4, 6, 8, \ldots)$ を「素数でない」とする。(素数でないものに下線を引いて示す)
- $2, 3, \underline{4}, 5, \underline{6}, 7, \underline{8}, 9, \underline{10}, 11, \underline{12}, 13, \underline{14}, 15, \underline{16}, 17, \underline{18}, 19, \underline{20}, 21, \underline{22}, 23, \underline{24}, 25, \underline{26}, 27, \underline{28}, 29, \underline{30}, 31, \underline{32}, 33, \underline{34}, \ldots$
- 3 が素数と分かるので、3 の倍数 $(6, 9, 12, \ldots)$ を「素数でない」とする。
- $2, 3, \underline{4}, 5, \underline{6}, 7, \underline{8}, \underline{9}, \underline{10}, 11, \underline{12}, 13, \underline{14}, \underline{15}, \underline{16}, 17, \underline{18}, 19, \underline{20}, \underline{21}, \underline{22}, 23, \underline{24}, 25, \underline{26}, \underline{27}, \underline{28}, 29, \underline{30}, 31, \underline{32}, \underline{33}, \underline{34}, \ldots$
- 5 が素数と分かるので、5 の倍数 $(10, 15, 20, \ldots)$ を「素数でない」とする。
- $2, 3, \underline{4}, 5, \underline{6}, 7, \underline{8}, \underline{9}, \underline{10}, 11, \underline{12}, 13, \underline{14}, \underline{15}, \underline{16}, 17, \underline{18}, 19, \underline{20}, \underline{21}, \underline{22}, 23, \underline{24}, \underline{25}, \underline{26}, \underline{27}, \underline{28}, 29, \underline{30}, 31, \underline{32}, \underline{33}, \underline{34}, \ldots$
- 7 が素数と分かるので、7 の倍数 $(14, 21, 28, \ldots)$ を「素数でない」とする。
- $2, 3, \underline{4}, 5, \underline{6}, 7, \underline{8}, \underline{9}, \underline{10}, 11, \underline{12}, 13, \underline{14}, \underline{15}, \underline{16}, 17, \underline{18}, 19, \underline{20}, \underline{21}, \underline{22}, 23, \underline{24}, \underline{25}, \underline{26}, \underline{27}, \underline{28}, 29, \underline{30}, 31, \underline{32}, \underline{33}, \underline{34}, \ldots$

このように過去に見つかった素数の倍数を消してゆくことで、最初のアルゴリズムにあった無駄が省かれていることが分かる。

(以下空白)

これをプログラムにする。整数が「素数かも知れない」か「素数でない」かは真偽値 (boolean 型の値) によって表わすことができる。

i が素数かどうかを調べるためには、2 以上 i 未満の数が「素数でない」かどうかを覚えておく必要があるので、boolean 型の配列を作り、その i 番目が false なら「整数 i が素数でない」ことを表わすことにする。

```
1  public class Eratosthenes {
2      public static void main(String[] args) {
3          int max = 100000;                    // 調べる最大値
4
5          boolean[] mayBePrime = boolean 型の大きさ max の配列を作る;
6          for (int x = 2; x < max; x++){
7              mayBePrime の x 番目を true にする
8          }
9
10         for (int x = 2; x < max; x++){
11             if ( mayBePrime の x 番目は true か? ) {
12                 x が素数だったときの処理
13                 for (int i = 2; i*x < max; i++) {
14                     mayBePrime の i*x 番目を false とする
15                 }
16             }
17         }
18     }
19 }
```

このプログラムでは配列mayBePrime が整数が「素数かも知れない」ことを覚えておくために使われる。

最初、全ての数は素数かも知れないの状態なので、最初の for 文によってmayBePrime の全ての値を true にする。

次の for 文では x を 2 から順に変化させてゆく。$(x-1)$ まで処理を終えたときに、mayBePrime の x 番目が true であれば、x は素数である。そのときには、内側の for 文で、mayBePrime の (x の整数倍番目) を false にしている。

練習 **5-2:** (エラトステネスのふるい)　このアルゴリズムを用いて、2 から 10 万までの数について最大の素数と素数の個数を求めるプログラムを作れ。(参考: 資料 web ページ Eratosthenes.java)

練習 **5-3:** (比較)　2 つの素数を求めるアルゴリズムについて、調べる範囲を (a)2 から 2 万まで、(b)2 から 5 万まで、(c)2 から 10 万までにした場合の実行時間を比較せよ。

練習 **5-4:**　エラトステネスのふるいのアルゴリズムでは、x が素数であると分かったときに、$2x, 3x, 4x, \ldots$ を「素数でない」と変えていた。

しかし、$2x, 3x, 4x, \ldots, (x-1)x$ は全て x より小さい素数の倍数でもあるので、これらの数を「素数でない」と改めて変更する必要はない。この無駄を省くようにプログラムを変更せよ。

2　組み合わせ数

m 個の中から n 個を選ぶときの選び方がいくつあるかを組み合わせ数といい、${}_mC_n$ と書く。${}_mC_n$ は次のような性質を満たす:

- n が 0 のときは 1 通り
- n が m のときも 1 通り
- それ以外のときは、${}_{m-1}C_{n-1} + {}_{m-1}C_n$ 通り

m, n が与えられたときに ${}_mC_n$ を計算するアルゴリズムは次のようになる:

- ${}_0C_0, {}_1C_0, {}_1C_1, {}_2C_0, {}_2C_1, {}_2C_2, \ldots, {}_mC_n$ を覚えておく表を作る
- i を $0, 1, 2, \ldots, m$ まで変化させる
  - ${}_iC_0, {}_iC_i$ をそれぞれ 1 とする

- ${}_iC_1, {}_iC_2, \ldots, {}_iC_{i-1}$ を上の式を使って計算する

これをプログラムにするときには、${}_iC_j$ の値を覚えるための配列を作ればよく、概略は以下のようになる:

```
1  public class Combination {
2      public static void main(String[] args) {
3          int m = 30, n = 15;
4          配列 c を作る
5
6          for (int i = 0; i <= m; i++) {
7              c[i][0] = 1;
8              c[i][i] = 1;
9              c[i][1] から c[i][i-1] までを、それぞれ計算しセットする
10         }
11         System.out.println(c[m][n]);
12     }
13 }
```

練習 5-5: (組み合わせ数)　上のプログラムを完成させ、${}_{30}C_{15}$ の値を求めよ。(参考: 資料 web ページ Combination.java)