エラトステネスのふるい

エラトステネスのふるい

エラトステネスのふるいとは?

エラトステネスのふるいには2つの法則を必要とします。

```
1) 素数の倍数は素数ではない
2) 素数でない数(合成数)は必ずその数の平方根以下に分解できる
-----
1)
2,3,5,7 は素数であるから、
4,6,10,14 は素数ではない

2)
98 は合成数である。98の平方根は9ちょっとである。
98 = 2 * 49
どのようにしても、10以上の数2つに分解することはできない。なぜなら、
10 * 10 = 100
だからである。
```

この2つの法則をアルゴリズムに変える作業が、今回の課題です。

アルゴリズム

```
n より小さい素数をすべて調べるとする

1) 最初の素数 2 は「素数である」と確定する
2) 2 の倍数はすべて「素数でない」と確定する
3) 2 の次の数 3 は「素数である」と確定する
4) 3 の倍数はすべて「素数でない」と確定する
5) 3 の次の数 4 はすでに「素数でない」と確定されている
6) 4 の次の数 5 は「素数である」と確定する
7) 5 の倍数はすべて「素数でない」と確定する
8) この処理を n の平方根まで繰り返す
9) n の平方根以下で「素数である」と確定された数と
   n の平方根以上で「素数ではない」と確定されていない数が、
   n 以下のすべての素数であると確定する
```

実際に n = 20 でやってみます。

```
(素数でない)と確定されたら消すことにする
20 の平方根は 4 ちょっと (16(4*4) < 20 < 25(5*5))

(  ) 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
( 2 ) 2 3   5   7   9    11    13    15    17    19
( 3 ) 2 3   5   7        11    13          17    19

4 はすでに消えている

よって上で残っているのが素数である
```

ソースコード

```
using System;

namespace Prime2
{
        class MainClass
        {
                public static void Main(string[] args)
                {
```

```
                        const int n = 1000;
                        bool[] prime = new bool[n];
                        prime[0] = false;
                        prime[1] = false;
                        int count = 0;
                        for(int i=2;i<n;i++)
                        {
                                prime[i] = true;
                        }
                        for(int i=2; (i * i) <n;i++)
                        {
                                if(prime[i] == true)
                                {
                                        for(int y = (i << 1);y<n;y += i)
                                        {
                                                if(prime[y] == true)
                                                {
                                                        prime[y] = false;
                                                        count++;
                                                }
                                        }
                                }
                        }
                        for(int i=2;i<n;i++)
                        {
                                if(prime[i] == true)
                                        Console.WriteLine(i);
                        }
                        Console.WriteLine("{0} times repeat",count);;
                }
        }
}
```

このプログラムは実質的な処理の部分だけだと830回で処理しました。試し割りよりも100分の1近く少ないです。

解説

このプログラムは3つのステップにわかれています。

1. 配列の初期化(すべて素数である(true)と仮定する)
2. 素数でないものを false にしていく(n の平方根まで)
3. 表示する

これは実際にはひとつにまとめることも可能です。配列には初期値として false が入るようなので、それを無理矢理使えば初期設定は必要ありません。また、処理と表示を組み合わせることも可能です。

まず、最初のループを見てください。

```
for(int i=2; (i * i) <n;i++)
```

これが n の平方根まで繰り返す方法ですが、C#のMathクラスのSqrt()メソッドを使って、平方根をあらかじめ求めておけば、より簡潔に記述できます。

```
int m = (int)Math.Sqrt(n);
for(int i=2; i<m;i++)
```

forの中身が重要です。エラトステネスのふるいの最初の法則はここで使われます。

```
if(prime[i] == true)
{
        for(int y = (i << 1);y<n;y += i)
        {
                if(prime[y] == true)
                {
                        prime[y] = false;
                        count++;
                }
        }
}
```

最初の1行は「i が素数であるとき」です。最初は 2 です。

```
for(int y = (i << 1);y<n;y += i)
```

この形式は初めてみるでしょう。これまでに使ったことがありません。「<<」というのは「シフト演算子」で、左シフトを意味します。これは2進法の理解が必要です。たとえば、2を左に1シフトした場合をシュミレーションします。

```
2 は4bitの2進法で表すと、
0010
である

これを左に1シフトするので、右に0を一つ付け、一番左の0を一つ消す。
0100

これは
0100 = 2^3*0 + 2^2*1 + 2^1*0 + 2^0*0
     = 8*0 + 4*1 + 2*0 + 1*0
     = 4
```

左シフトは「2倍」に、右シフトは「1/2倍」になります。2進法の特性です。このようなシフト演算子は普通の演算子に比べて極めて高速に動作します。ですから、2のべき乗を扱うときには有効に活用したい機能の一つです。

```
y += i
```

これは分かりますね。yにはiの2倍・3倍・4倍が入っていきます。「+=」ではなく「*=」を使うと2乗・3乗・4乗が入っていきます。

次のifはなくても大丈夫なはずです。たとえば 15 を消す場合、3 のときにすでに false になっているので、5 のときは false の上書きをせずに無視する、という処理です。

これがエラトステネスのふるいをプログラムで実装したソースコードです。

素数判定への応用

エラトステネスのふるいは「n までの素数をすべて列挙する」ためのものですが、拡張することによって、「n が素数かどうかを判定する」ためにも使うことができます。

ただし、上記の例のように配列の初期化をやるのであれば意味がないので、無理矢理 false と true を逆に読みかえて書いてみました。

```
using System;

namespace Prime4
{
        class MainClass
        {
                public static void Main(string[] args)
                {
                        int n = 9967;
                        int i = Check(n);
                        Console.WriteLine(n);
                        Console.WriteLine(i);
                }
```

```
                public static int Check(int n)
                {
                        int sqrt = (int)Math.Sqrt(n);
                        int sqrt2 = (int)Math.Sqrt(sqrt);

                        bool[] prime = new bool[sqrt];
                        for(int i=2;i<sqrt;i++)
                        {
                                if(!prime[i])
                                {
                                        if(n % i == 0) return i;
                                        else if(i < sqrt2)
                                        {
                                                for(int j=(i << 1);j<sqrt;j+=i)
                                                        if(!prime[j]) prime[j] = true;
                                        }
                                }
                        }
                        return 1;
                }
        }
}
```

これには (n の平方根) ビットの外部領域が必要です。ですが、計算量は $O(n^{1/2})$(n の平方根) ですので、O(n)の「試し割り」よりもはるかに高速に処理できます。

エラトステネスのふるいと試し割りを組み合わせたものですが、実際に割るのは「素数」だけですので、極めて効率的にチェックできます。n が素数の場合は最小の約数である「1」を、n が素数でない場合は1を除く最小の約数を表示します。