4.3 双子素数

素数の話題は尽きることがない。話題の一つに**双子素数**というものがある。2 を除けば素数は奇数であるが、ときどき 17, 19 や 41, 43 のように連続して素数が現れることがある。これらを双子素数と呼ぶのだ。

素数が無数にあることは証明されている。それはユークリッド[1)]による証明が有名である。証明の大筋は次のようなものだ。

素数が有限個しかないと**仮定**し、素数を 2, 3, 5, ..., P とする。このとき $2\cdot3\cdot5\cdots P+1$ なる数を作れば、これは P より大きい新たな素数となるから、素数が P までしかないことに矛盾する。この矛盾は素数が有限個であると仮定したことに起因している。すなわち素数は有限個ではない。

このように、ある仮定から始めて矛盾を引き出し、結果、仮定が間違っていると結論する証明方法は**背理法**と呼ばれる。

ところで、素数がどれぐらいの頻度で現れるかとか、双子素数も無数にあるのかなど、素数には未解決の問題が多い。素数の頻度というのも変な感じもするが、ガウスは素数が現れる頻度について、次のような見解を持っていた。

$$1 \text{ から } N \text{ までには、素数はおよそ } \frac{N}{\log_e N} \text{ 個含まれる。} \tag{1}$$

たとえば $N=100$ とすると、(1) より

$$\frac{100}{\log_e 100} \approx 21.7$$

だが、実際は 25 個の素数がある。ぴったりではないが、まあ近いところをついている。式は N が大きいほど、より正しい頻度を教えてくれる。もし N を与えたとき、本当にぴたり正確な素数の数を求める式が発見されれば、画期的なことであるが、いまのところそのような式は発見されていない。またガウスの式とは別の、素数の頻度を計算する式はあるが、(1) の美しさにはかなわないのだ。何とも不思議なことである。

双子があれば三つ子も考えたくなるのが人情で、3, 5, 7 のような三つ子素数がどれくらいあるかと考えてしまう。だが結論を言おう。これ以外の三つ子素数はない。連続する三つの奇数は三つ子素数にはならないのだ。

---

1) ユークリッド (330?B.C.–275?B.C.) : ギリシアの数学者。

EX. なぜ連続する三つの奇数が三つ子素数にならないか考えてみよ。

三つ子素数がないと分かったところで、双子素数に集中しよう。と言っても場当たり的に双子素数を探すのでは効率が悪い。徹底した調査で、どのような条件のとき双子素数が現れるのかが分かれば、プログラムも組みやすいものだ。そこで、ここでは次の単純な事実をもとに、双子素数を見つけていこう。

2, 3 を除くすべての素数は $6m \pm 1$ の形をしている。

そう、素数は決して $6m \pm 2$ や $6m \pm 3$ のような形をしていないのだ。誤解しないでもらいたいことがある。私は「素数は $6m \pm 1$ の形をしている」と主張しているのであって、決して「$6m \pm 1$ の形をしている数は素数である」と言っているのではない。逆は必ずしも正しくないとは、まさにこのことを言う。

EX. なぜすべての素数が $6m \pm 1$ の形をしているか説明せよ。

そういうことなら話は簡単だ。$6m - 1$ が素数であるかどうか調べ、素数であるときに限り $6m + 1$ が素数であるかどうか調べればよい。運良く二つとも素数なら、それが双子素数ということだ。

programming list [TwinPnum.java]

```
1: public class TwinPnum {
2:
3:     final static int SUP = 100;
4:
5:     public static void main(String[] args) {
6:         for(int m = 1; 6 * m < SUP; m++) {
7:             boolean sgn = pchk(6 * m - 1) && pchk(6 * m + 1);
8:
9:             if(sgn == true)
10:                System.out.println("(" + (6 * m - 1) + ", " +
                                                        (6 * m + 1) + ")");
11:        }
12:    }
13:
14:    public static boolean pchk(int n) {
15:        int i = 1;
16:        boolean flag = true;
17:
18:        while((i += 2) <= Math.sqrt(n)) {
```

```
19:             if(n % i > 0) {
20:                 ;
21:             } else {
22:                 flag = false;
23:                 break;
24:             }
25:         }
26:         return flag;
27:     }
28: }
```

プログラムが [PrimeNum.java] に較べ、とても長くなったように感じるかもしれない。だが、そんなことはない。pchk() メソッドは [PrimeNum.java] の 8:–21:行目の焼き直しに過ぎない。大きな違いは pchk() が値を返すメソッドだから、flag の値を返すことだ。しかし [PrimeNum.java] とちょっとだけ値の扱いが違っている。これはあとで述べよう。

結局、双子素数を見つけるプログラムは、main(){} 内のわずか 6 行分である。では main() メソッドを見ておこう。

6:行目の m は $6m \pm 1$ の m に対応した整数変数である。また、7:行目の sgn は $6m \pm 1$ が共に素数になったかどうかを判定させるための変数だが、この型 boolean は少しばかり特殊だ。ふつう変数には任意の数値や文字列が代入されるのだが、boolean 型の変数は "真 (true)" と "偽 (false)" の 2 値だけを取る。これまでのプログラムでは、正の値か 0 かで素数を判定したと思うが、このような二者択一の判定には boolean の方が適している。

プログラムは 6:行目の for 構文で、上限に設定した SUP までの整数を調べることにしている。いまは 100 までの双子素数を調べたいのだ。そのための条件を 6 * m < SUP; にしていることに注意を払ってほしい。m < SUP; としてしまうと、600 前後までの双子素数を出力してしまう。だからといって何も困ることはないけどね。

さて、このプログラムの実質的な内容は 7:行目に凝縮されている。まず pchk(6 * m - 1) だが、6 * m - 1 の値—最初は 5 ということだ—を pchk() メソッドに渡して、素数かどうかの判定を仰いでいる。そして最初の 5 は素数なので、pchk() は "真" を返してくる。

また、pchk(6 * m + 1) も同様だ。これは最初 7 を判定するので、やはり "真" を返してくるのである。すると問題は && が何をするのかということだ。A && B は**論理演算子**の一つで、A と B が共に真のときに限り、A && B は真になる。それ以外は偽と判定される。さらに A && B の特徴は、

A が偽になればB の真偽に関わらず、A && B は偽であるから、そのときは B をいちいち計算しないのだ。このことは計算量を押さえる役に立つ。

いずれにせよ 7:行目では、$6m \pm 1$ が共に素数になったときだけ、sgn が真になるということが大事だ。

それによって、9:–10:行目で双子素数だけが画面に表示されるのである。また、sgn が偽の場合は何もする必要がないので、10:行目の次に空文を置いてもよいが省略してある。