

## 6...の夢見鳥

### 6.1 完全数

6は興味深い数の一つだ。それはどういうことかという、6の約数は1, 2, 3, 6だが、

$$1 + 2 + 3 = 6$$

となっている。この性質を満たす数は6の近辺にはない。たとえば8は、約数として1, 2, 4, 8を持つが、明らかに $1 + 2 + 4 \neq 8$ である。約数をもれなく取り出すことができれば、6の次にこの性質を満たす数が28であることを知るの容易（たやす）い。実際、28の約数は1, 2, 4, 7, 14, 28であるから

$$1 + 2 + 4 + 7 + 14 = 28$$

になっている。この性質を持つ数を完全数と呼ぶ。

性質を誤解のないように言えば、完全数とは

自分自身を除く約数の総和が自分自身に等しい数

であると言える。なんと、日本語にするほうが式で説明するより難しくなっていないかい？ 数学とはそんなものだ。

でも、なんで約数の和が自分自身に等しいと完全なんだ？ 完全の名称はむしろ、その性質を持つ数が6と28の他にすぐ見つからなかったからかもしれない。6は神が世界を作った日数だとか、28は月の周期日数<sup>1)</sup>だとか、そんなところが完全っぽい雰囲気だったんだろう。どうでもいいけど。

それなら、28の次の完全数はいくつだろうか。ちょっと計算すればすぐに見つかりそうだ。素数のように、明らかに完全数になり得ないものは省ける。約数の数が少ないものや(素数)<sup>n</sup>のような数もたいてい省けそうである。このように候補となる数を絞っていけば、意外に早く見つかるかもしれない。悪いね。そんな簡単には見つからないのだ。よほど根気よくなければ、次の完全数を見つけるのは難しい。根気が勝負になるときこそコンピュータの出番というわけだ。

---

1) 公転周期は約27.3日、満ち欠けの周期は約29.5日。28日とはちょっとずれてる。

いきなり完全数を求めるスクリプトを書くのも大変だろうから、まずは約数を見つけるスクリプトから見ていこう。それは『

```
\begin{luacode*}
function measures(n)
  tex.print('1' .. ' ')
  for i = 2, n//2 do
    if n % i == 0 then
      tex.print(i .. ' ')
    end
  end
  tex.print(n)
end
\end{luacode*}
```

』という関数で、処理すると『`\directlua{measures(45)}`』と書いて『1 3 5 9 15 45』が出力される。

いままでの知識さえあれば十分理解できるスクリプトだ。コードは与えられた数に対して、約数をすべて表示することになっている。ところで、 $n$  の約数に 1 と  $n$  があるのは当然なので、これらは最初と最後に表示することにした。これが最初と最後の `tex.print` 文である。

`for` 構文は  $n$  を 2 から順々に割っていくものだ。約数を求める行為は素数を求める行為と違うので、2 で割れても 4 で割れるか試す必要があるし、1 ずつ大きな数で割る必要もある。ただし、 $\frac{n}{2}$  までの数で試せばよい。それより大きい数が約数になることはないのだから。整数でしか割らないのだから、指定する範囲は  $n//2$  まででよい。

`n % i` が割り切れれば  $i$  は約数であるから出力すればよい。そうでなければ次を試すまでだ。最後に残った数を表示すれば終了だ。

スクリプトは約数を列挙するものだが、素数を見つける役にも立つ。なぜなら、素数は 1 と自分自身しか約数を持たない数だから、たとえば『`\directlua{measures(17)}`』と書いて『1 17』が出力される。

ところで、ここに提示した約数を求めるスクリプトは、単純きわまりないアルゴリズムであるだけに無駄が多い。当然のことながら、たとえばある数  $n$  が  $i$  で割れたなら、その商である  $\frac{n}{i}$  も約数である。つまり約数は基本的に二つ同時に見つけることができる。基本的と表現したのは、49 が 7 で割れたからと言って、 $\frac{49}{7}$  が別の約数とは言えないからだ。しかし、除数  $i$  と商  $\frac{n}{i}$  を一緒に表示

してしまえば、割り算は  $\sqrt{n}$  まで試せばよいことになる。これは以前、素数を求めた際に使った手法だし、計算量を減らす効果も十分だ。余裕があればこの考え方でスクリプトを書き換えてほしい。

だが、完全数を求めるためには、これではまったく不十分なのだ。あとで分かるように、完全数は素数とは較べものにならないくらい稀にしか現れない。つまり、計算量が減ること自体は喜ばしいけれど、結局ほとんどの数で調査が不発に終わる。効果的に調べるには、計算量を減らすのではなく、調査しなくてもよい数を飛ばすことである。

さて、約数をすべて列挙するスクリプトが書けたので、それらの和をとれば完全数を探ることができる。そのための関数が『

```
\begin{luacode*}
function pnfind(n)
  for m = 6, n do
    sum = 1
    for i = 2, m//2 do
      if m % i == 0 then
        sum = sum + i
      end
    end
    if m == sum then
      tex.print(m .. ' ')
    end
  end
end
\end{luacode*}
```

』で、『`\directlua{pnfind(10000)}`』と書いて『6 28 496 8128』が出力される。28の次の完全数は496か。たしかに、すぐには見つかりそうもない数だ。

スクリプトは関数 `measures` に、和を求めるために変数 `sum` を追加したものである。`sum` は自分自身以外の約数の和を加算する変数だ。最初の完全数は6であることが分かっているので、`n = 6` から始めている。

`sum` に1を代入したのは、最初の約数が自明の1だからである。あとは、約数が見つかるたびに `sum` に加えるだけだ。そして、与えられた数と約数の合計が等しくなったら、それが完全数である。

やってみて容易に分かるように、完全数は非常に少ない。5番目の完全数を探すためには相当の

時間を覚悟しなくてはならない。それもそのはずで、完全数は  $2^{n-1}(2^n - 1)$  の形をしている。いま探した四つの数でちょっと確認してほしい。これでは指数関数的に大きな数になってしまうから、完全数がどんなにたくさんあったとしても、気軽に発見できないのだ。

そこで忠告をしておこう。このスクリプトで 5 番目の完全数を探す暴挙に出ないように。5 番目の完全数は 8 桁の数だが Lua の守備範囲だと思えないでほしい。このアルゴリズムは、8128 を見つけるのでさえ少々時間を要する。組版の際、ちょっと間が空いただろう。単純に考えても、1 桁増えるごとに計算時間は 10 倍になるから、8 桁の解を見つけるには 10000 倍の時間がかかる。8128 を 0.1 秒で見つけられても、5 番目の完全数を見つけるには 1000 秒（16 分 40 秒）もかかるのだ。だけど、その間にお茶を入れる時間ができるので、必ずしも悪いことばかりではない。

結局のところ、コンピュータの計算速度を過信してはいけないということだ。そのために、どうしても人の手で、効率的なアルゴリズムが必要になるのである。手っ取り早く 5 番目までの完全数を知りたいなら、 $2^{n-1}(2^n - 1)$  の形の数だけ調査するスクリプトに変更すればよい。驚くほど早く結果を目にすることができる。でも、今度はお茶を入れる暇がなくなるけど。