6.3 友愛数・婚約数

完全数には親戚のようなものが存在する。220 の純粋な約数の和は

$$1+2+4+5+10+11+20+22+44+55+110=284$$

である。もちろん完全数ではない。ところが、284 についても同じことをすると

$$1+2+4+71+142=220$$

である。なんと、互いに完全数の性質を補完し合っているみたいではないか。このようなペアを友愛数と呼んでいる。ここでは友愛数のペアを求めてみよう。

[hs script]

```
measuresum :: Int -> Int
measuresum n = sum [a | a <- [1..n-1], n `mod` a == 0]

friendnums :: Int -> [(Int, Int)]
friendnums 1 = []
friendnums n
    | measuresum msum == n = fs ++ [(n, msum) | n < msum]
    | otherwise            = fs
    where msum = measuresum n
          fs   = friendnums (n-1)
```

friendnums 関数は、引数 n までにある友愛数のペアを求めるスクリプトになっている。その際、純粋な約数の和を求めておく必要があるので、完全数を求めるときに使った mesuresum 関数も持ち出してある。スクリプトは、ある数の純粋な約数の和の、そのまた純粋な約数の和がもとの数であるという友愛数の定義そのものになってる。友愛数のペアをリストに追加する際、n < msum を条件にしたのは、完全数や同じ組の友愛数を表示しないためである。つまり、完全数の純粋な約数の和はもとの数になるから—それが完全数の定義だから—広い意味で完全数も友愛数である。しかし、わざわざそれを表示することはないだろう。また、n の友愛数 m を見つけたら、同時に m の友愛数は n なので、同じペアが重ねて表示されてしまう。これらを除くために n < msum を条件に加えている。実際、10000 未満の友愛数を求めると、だいぶ時間を要して 5 組が見つかる。

(ghci env.)

```
*Main> friendnums 10000
[(220,284),(1184,1210),(2620,2924),(5020,5564),(6232,6368)]
```

(220, 284) は古くから知られたもので、(2620, 2924) をはじめとしてオイラーは 60 組以上の友愛数を見つけたようである。ところが、(1184, 1210) のペアを見逃していたというのだから不思

議だ。そして 100 年以上後に、16 歳の少年、パガニーニ[1]がオイラーの見逃していた (1184, 1210) を見つけたと言われている。なんてこった。

友愛数に似たペアに婚約数がある。完全数も友愛数も約数を合計する際、1 は加えるが自分自身は加えない。そもそも 1 と自分自身はどんな数でも持っている自明の約数だから、そのうち 1 は加えて自分自身を加えないのはバランスを欠いているのではないだろうか。それなら、約数の合計に 1 を加えなければよいだろう、と考えるのは自然な流れだ。そこで今度は、1 と自分自身を含まない約数を庭いじりだけで通じる用語として “真の約数” と呼び、真の約数の和を求めることにしてみよう。

これを調べるスクリプトは、`measuresum` 関数において `a <- [1..n-1]` を `a <- [2..n-1]` にするだけである。これを `measuresum'` 関数として `engagenums` 関数に仕立てよう。

[hs script]

```
measuresum' :: Int -> Int
measuresum' n = sum [a | a <- [2..n-1], n `mod` a == 0]

engagenums :: Int -> [(Int, Int)]
engagenums 1 = []
engagenums n
    | measuresum' msum == n = fs ++ [(n, msum) | n < msum]
    | otherwise             = fs
    where msum = measuresum' n
          fs   = engagenums (n-1)
```

10000 未満の婚約数を調べると、やはり時間を要して友愛数とはおもむきの違うペアが現れる。

(ghci env.)

```
*Main> engagenums 10000
[(48,75),(140,195),(1050,1925),(1575,1648),(2024,2295),(5775,6128),(8892,1
6587),(9504,20735)]
```

友愛数のペアと婚約数のペアを見比べて気づくことはないだろうか。そう、友愛数はすべて偶数のペアで、婚約数はすべて奇数と偶数のペアになっていることだ。これは非常に不思議なことで、これまでに見つかった友愛数と婚約数の組み合わせは、すべてそうなのだ。友愛数とか婚約数の名称がどうやってついたかは知らない。ただ、古代ギリシアの数字感に、1 は神、2（偶数）は女、3（奇数）は男、5（$2+3$）は結婚、... というものがある。それに照らすと、偶数どうしのペアが友愛数、奇数と偶数のペアが婚約数という名称は的を射ている。

ところで友愛数や婚約数の考えを発展させると、a の約数の和 $\to b$、b の約数の和 $\to c$、c の約数の和 $\to a$ のように、3 数で巡回する組や 4 数で巡回する組など、いくらでも拡張することがで

[1]イタリアの音楽家、ニコロ・パガニーニとは別人。

きる。このとき約数の和としてとるのは、純粋な約数の和でもよいし、真の約数の和でもよい。有名なところでは、純粋な約数の和を考えたとき、$12496 \to 14288 \to 15472 \to 14536 \to 14264$ という 5 数の巡回がある。このような数は社交数と呼ばれる。社交数には 5 数以外の社交の輪がいくつも見つかっているが、不思議なことに 3 数の社交の輪は見つかっていないようだ。三角関係を成り立たせるのは難しいのだろうか。

ただし、社交数の組を見つけるスクリプトを書くのはちょっと苦労する。というのは、ある数の純粋な約数の和は `measuresum` 関数で簡単に分かっても、それを何回調べてもとに戻るかが分からないので、ちょっとつかみ所がないからだ。たとえば、数 a の巡回を 20 回調べて最初の純粋な約数の和に一致しないからといって、a が社交の輪にないとは言い切れないのである。実際に、28 数で巡回するものがあるというから一筋縄ではいかない。でも、ある数が社交の輪にあるかどうかは、簡単なスクリプトで何となく調べられる。

[hs script]
```
findsocials :: [Int] -> [Int]
findsocials (x:xs)
    | measuresum x `elem` xs = (x:xs)
    | otherwise              = findsocials (measuresum x : x : xs)
```

この `findsocials` 関数はだいぶ荒っぽいことをする関数である。まず、リストの先頭の純粋な約数の和を調べてみる。それがリストに含まれていれば、リストの先頭の数は、最低でも 2 個の社交の輪にいることになる。もっとも 2 個の社交の輪は友愛数を意味するけど。たとえば

(ghci env.)
```
*Main> findsocials [284]
[220,284]
*Main> findsocials [12496]
[14264,14536,15472,14288,12496]
```

のように引数を与えれば、場当たり的に社交数または友愛数を特定できる。

もし、リストの先頭の純粋な約数の和がリストに含まれていなければ、どうなるだろう。その場合は、そいつをリストの先頭に加えて同じことをすればよい。スクリプトが荒っぽいことをしているというのは、このことを意味している。最初の引数が社交の輪になければ、スクリプトは止まらないことになる。本当にそうだろうか。

(ghci env.)
```
*Main> findsocials [365]
[0,0,1,79,365]
*Main> findsocials [700]
[0,0,1,11,21,51,141,195,285,1939,7469,4900,4844,2100,2044,1092,1036,700]
```

2 例ほど試してみた。いずれも先頭に [0,0, が表示されて終了している。これは次の理由による。

ある数の純粋な約数の和を繰り返し計算していると、たまたま和が素数になることがある。365 から始めたら 79 が、700 から始めたら 11 がそれに当たる。素数において純粋な約数は何だろう。純粋な約数は 1 は認めて、自分自身は除いていたので、それは 1 だ。じゃあ、1 の純粋な約数は何だ？　定義に従えば、1 は認めて 1 は除くことになる。矛盾してるぞ。実は、measuresum 関数は引数 1 に対して 0 を返してくる。すると 0 の純粋な約数は何かとなるが、これにも measuresum 関数は 0 を返すのである。それでリストには 0 が 2 個含まれて、スクリプトは終了したのである。

単発で友愛数を探すのは大変だから、map を使って系統的に調べてみよう。そのために

(ghci env.)
```
*Main> let findsocials' n = findsocials [n]
```

と再定義した上で map findsocials' [6..28] と入力してみよう。わざわざ findsocials' 関数を再定義したのは、引数を [[6]..[28]] のように与えることができないからである。

(ghci env.)
```
*Main> map findsocials' [6..28]
[[6,6],[0,0,1,7],[0,0,1,7,8],[0,0,1,3,4,9],[0,0,1,7,8,10],[0,0,1,11],[0,0,
1,3,4,9,15,16,12],[0,0,1,13],[0,0,1,7,8,10,14],[0,0,1,3,4,9,15],[0,0,1,3,4
,9,15,16],[0,0,1,17],[0,0,1,11,21,18],[0,0,1,19],[0,0,1,7,8,10,14,22,20],[
0,0,1,11,21],[0,0,1,7,8,10,14,22],[0,0,1,23],[0,0,1,17,55,36,24],[6,6,25],
[0,0,1,3,4,9,15,16,26],[0,0,1,13,27],[28,28]]
```

あちゃー。なんて見づらい結果なんだろう。目を凝らして見てもらいたい。最初の 6 と最後の 28 は完全数だから、自分自身で社交の輪を作っていて、他の数は社交の輪にないのでリストが [0,0, で終わっているのが分かるだろう。25 が一瞬、社交の輪の中にあるように思えるが、途中で純粋な約数の和が 6 になっただけである。

さあ、これで好きなだけ社交数の探索ができるぞ、と思ったら大間違い。たとえば 138 は純粋な約数の和を次々計算していくと値が大きくなるばかりで、輪もできず和が素数にもならずで終わる気配を見せない。マシなスクリプトにするには、もう少し調整が必要である。