JAPLA 研究会資料 2009/2/28

q:(素因数分解)から約数を求める
－divisors をトレースする－

西川　利男

１．はじめに

親和数または友愛数(amicable number)を求めるＪのプログラムが山下紀幸氏からFAX メールにより送られてきた。数年前、Ｊでフィボナッチ数、オイラー数など整数論の問題を一緒に検討したときに、読み合った教科書、Beiler, ″Recreations in the Number Theory″[1] を今なお考究されているようで、敬服するばかりである。

Ｊでは素因数分解を求める強力なツール q:により一発で値が得られるが、これからすべての約数を求めるのは簡単そうに見えて案外やっかいである。　J Phrases を見ていたら、約数を与える定義 divisors があった。たしかにあざやかにすべての約数が求められる。しかしその定義は tacit を用いたまさにコンパクトなＪらしいプログラムだが、その意味を理解するには、筆者にとってはかなりの「頭の体操」になった。その次第を報告する。

２．素因数から約数へ

簡単な例により考えてみよう。例えば220 を取り上げる。

```
   q: 220
2 2 5 11
```

約数のすべては上の因数から、0 個、1 個、2 個、3 個、4 個のそれぞれからあらゆる組合わせの値の積を取ることで得られる。

0 個　1
1 個　2, 5, 11
2 個　(2・2), (2・5), (2・11), (5・11)
すなわち　4,　10,　22,　55
3 個　(2・2・5), (2・2・11), (2・5・11)
すなわち　20,　44,　110
4 個　(2・2・5・11)
すなわち　220

これから、自身に等しいものを除いた約数の和を取ると 284 になる。一方、284 について同様に約数を求め、その和を取ると、220 になる。

このような１組の数が親和数または友愛数(amicable number)であり、Beiler の本によると 220 と 284 は聖書の昔から知られていたとのことである。

[1] Albert H. Beiler, ″Recreations in the Number Theory - The Queens of Mathematics Entertains″, Chapter 4, ″Just Between Friends″, p.26-30, 2nd ed. Dover Pub. Inc, New York (1966).

３．約数の定義 divisors と odometer とその働き

J Phrases の divisors と odometer の定義を次に示す。

```
    divisors=: /:~@(~. */ .^"1  odometer@:>:@(#/.~))@q:
```

これはサブプログラムとして odometer を呼んでいる。

```
    odometer=: #: i.@(*/)
```

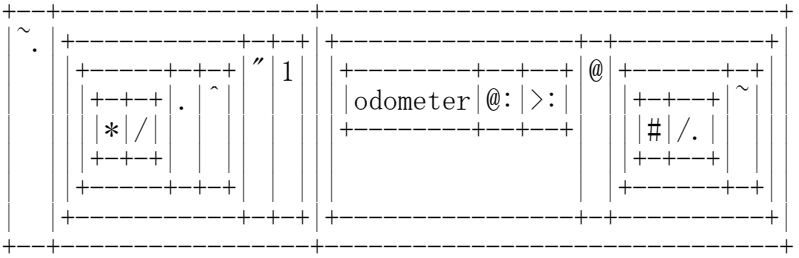
先の例 220 に対して実行すると、次のようにすべての約数が返される。

```
   divisors 220
 1  2  4  5  10  11  20  22  44  55  110  220
```

一歩ずつ、ゆっくりと説明してゆくことにする。メインプログラム divisors を大きくとらえると、q:により素因数を求めて、メインの処理を行い、最後に/:~により結果を昇順にする、という構成になる。したがって心臓部は以下のコーディングである。

```
    ~. */ .^"1  odometer@:>:@(#/.~)
```

上のＪコードを引数なしで打ち込むとその構造が以下のように示される。



これから分かることは、これは３つの要素からなる fork 構造の関数である。

これを詳細に検討するため、次のように３つの関数 f, g, h に分けて扱う。

```
    f =: ~.
    g =: */ .^"1
    h =: odometer@:>:@(#/.~)
```

ここまでを１段階ずつやってみると次のようになる。

```
       q: 220
    2 2 5 11
       (f g h) 2 2 5 11
    1 11 5 55 2 22 10 110 4 44 20 220
        /:~ 1 11 5 55 2 22 10 110 4 44 20 220
    1 2 4 5 10 11 20 22 44 55 110 220
```

さらに内部の fork の部分をやってみる。まず f については次のようになる。

```
        f 2 2 5 11
    2 5 11
```

f は ~. で重複したものを排除する。

つづいて h 部分だが、これはさらに次の内部処理に分かれる。まず

```
    #/.~
```

を行なった結果に対して、

```
    >:
```

を行い、さらにサブ関数

```
    odometer
```

を実行する。

最初の#/.~は次のように２項関数として働く。

```
      (2 2 5 11) (#/.) (2 2 5 11)
    2 1 1
```

このコーディングはどう理解するか。接続詞(/.)は左引数(2 2 5 11)をキーとして、右引数(2 2 5 11)に動詞(#)を作用させる。

接続詞(/.)キーは非常に分かり難いが、動詞(<)を用いた次の例で納得行くだろう。

```
      (1 2 1 3) (</.) (2 2 5 11)
+---+-+--+
|2 5|2|11|
+---+-+--+
      (1 2 3 3) (</.) (2 2 5 11)
+-+-+----+
|2|2|5 11|
+-+-+----+
```

上の例は１というキーを与えた要素２と５をボックスで囲み、キー２と３の要素は単独で囲む。下の例では、キー３を与えた要素５と 11 をボックスで囲む。

元に戻って、先のコーディングは２が２個、５と 11 とは１個であることが得られる。これによりべきの数を算出している。その次の関数(>:)により３ ２ ２となる。

ここでサブ関数

```
   odometer
+--+-----------+
|#:|+--+-+-----+|
|  ||i.|@|+-+-+||
|  ||  | ||*|/|||
|  ||  | |+-+-+||
|  |+--+-+-----+|
+--+-----------+
```

を検討してみる。これは hook 構造である。すなわち

```
   y #: (i.@(*/) y)
```

のように実行される。

```
     (i.@(*/)) 3 2 2
  0 1 2 3 4 5 6 7 8 9 10 11
```

つまり、3*2*2=12　までの連続整数が生成される。したがって odometer の実行は

```
  3 2 2 #: 0 1 2 3 4 5 6 7 8 9 10 11
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
2 0 0
2 0 1
2 1 0
2 1 1
```

これは何を示しているのだろうか。０から 11 までの数の３進２進２進での表示である。

これにより、あらゆるべき乗の組合わせが得られる。ちなみにodometerを英語の辞書で引くと「距離計」という訳語が出ていた。

最終段階のfork構造のg部分である。これはランク１(″1)により実行する。

```
  2 5 11 ^ 0 0 0
1 1 1
  2 5 11 ^ 0 0 1
1 1 11
  2 5 11 ^ 0 1 0
1 5 1
  2 5 11 ^ 0 1 1
1 5 11
  2 5 11 ^ 1 0 0
2 1 1
  2 5 11 ^ 1 0 1
2 1 11
  2 5 11 ^ 1 1 0
2 5 1
  2 5 11 ^ 1 1 1
2 5 11
  2 5 11 ^ 2 0 0
4 1 1
  2 5 11 ^ 2 0 1
4 1 11
  2 5 11 ^ 2 1 0
4 5 1
  2 5 11 ^ 2 1 1
4 5 11
```

このようにしてすべての因数の組合わせが得られた。したがって、これらの各組ごとの積をとれば、すべての約数が得られることになる。

```
   */″(1) (2 5 11) ^″(1)  3 2 2 #: 0 1 2 3 4 5 6 7 8 9 10 11
1 11 5 55 2 22 10 110 4 44 20 220
```

４．親和数または友愛数(amicable number)

上の約数の関数divisorsを用いれば、親和数または友愛数(amicable number)を求める関数amicableは以下のように簡単に作られる。

```
   amicable=: +/@:}:@divisors
```

実行は次のようになる。

```
   amicable 220
284
   amicable 284
220
```

Beilerの本には29のリンクを持つ次のような長大な親和数があげられているが、これも一瞬で得られる。

```
   amicable^:(i.29) 14316
```

14316 19116 31704 47616 83328 177792 295488 629072 589786 294896 358336
418904 366556 274924 275444 243760 376736 381028 285778 152990 122410 97946
48976 45946 22976 22744 19916 17716 14316

５．西川のもっと簡単な改良プログラム

一通りやった後で、もっと簡単なプログラムでも出来るのではないか、と試みたのが次の改良版プログラムである。

まず、q: により素因数を得た上で、それぞれに1を付加して、次のボックスを作る。

```
   q: 220
2 2 5 11
   R =. <"(1) 1 ,. 2 2 5 11
   R
+---+---+---+----+
|1 2|1 2|1 5|1 11|
+---+---+---+----+
```

このボックスデータRに対して、Catalogue { を利用して、あらゆる組み合わせを生成する。

```
   >,{ R
1 1 1  1
1 1 1 11
1 1 5  1
1 1 5 11
1 2 1  1
1 2 1 11
1 2 5  1
1 2 5 11
2 1 1  1
2 1 1 11
2 1 5  1
2 1 5 11
2 2 1  1
2 2 1 11
2 2 5  1
2 2 5 11
```

この結果を行ごとに掛けて、重複したものを取り除けばよい。あとは昇順に並べかえる。

```
   /:~ ~. */"(1) >,{ R
1 2 4 5 10 11 20 22 44 55 110 220
```

プログラムとしたものは次の通りである。

```
NB. Nishikawa's More Simple Program 2009/2/21
divisor =: 3 : 0
Q =. q: y.
R =. <"(1) 1,. Q
/:~ ~. */"(1) > , { R
)
```

このように、Jではいろいろな機能がそなわっているので、自分で工夫してプログラムすれば良い。プログラムは一通りに限ったものではない。