カプレカー数をＪで

SHIMURA Masato
jcd02773@nifty.ne.jp

2011 年 4 月 22 日

目次

はじめに

アメリカンサイエンス誌に永きに亘って「数学ゲーム」を執筆し読者を楽しませた Martin Gardner が２０１０年に逝去した。これを契機として日本語版を出版している日経サイエンスから別冊として一松　信訳「マーチン・ガードナーの数学ゲーム I(新装版)」が刊行された。
数々の肩のこらない数学問題が掲載されているがこの最初の「9 つの挑戦的な問題」にカプレカー数が取り上げられている。

D.R.Kaprekar はインドの数学者でレクリエーション数学や数論の分野で 20 冊程の著書がある。カプレカーはいろいろな興味深い数を提示している。

1　カプレカーの数

1.1　カプレカー定数

全ての数が同一でない任意の 4 桁の数 (たとえば 1926 から始める)。その数字を大きいほうから順に並べ (9621)、次に数字の順序を逆にして (1269)、始めの数から後の数を引く (9621-1269=8352)。引き算で得られた数字についてこの操作を反復していくと (8 回以下で) **カプレカー定数** (6174) に達する。この数は何回繰り返しても自分自身に戻る

数字の分離　ソートするには数字を一個づつ分離しなければならない。

```
divide=: >@ (". L:0)@( {@>)@ ": NB. 4683 --> 4 6 8 3
```

文字化して (":) 一個づつ Box に入れてから ({@>) 数値に戻し (".)、ボックスを開く (>)

@は動詞を連結して複合動詞を作る接続詞で右から左に順次作用する。

L:0 はボックスの中の動詞に個別に作用し、ループを用いないで並列演算を行う便利な作用素。

ソート　アップソートは（/:~)、ダウンソートは（\:~）である。

;で動詞をボックスに入れて連結し、並列で演算する

```
   (\:~;/:~)@divide 1926
```

9 6 2 1	1 2 6 9

数値の結合　　10 進化 (10&#.) を用いて 10 進数に戻す

```
 > 10&#. L:0 (\:~;/:~)@divide 1926
9621 1269
```

関数定義での反復　J の関数定義での便利な簡易反復法。i.8 とすると 8 回のループの経過を表示する

```
calc_kap=:  -/@ sort0          NB. calc


   calc_kap ^:(i.8) 1926
1926 8352 6174 6174 6174 6174 6174 6174
```

同時計算　引数の値をボックスで囲み　L:0 を用いる。確かに 6174 になる

```
> calc_kap(^:10)L:0 (4683; 5472; 3973)
6174 6174 6174
```

1.1.1　Script

```
divide=: >@ (". L:0)@( {@>)@ ":              NB. 4683 --> 4 6 8 3
sort0=:[: > [: 10&#. L:0  (\:~ ; /:~)@divide NB. down&up sort
NB.  10&#. is  connect 4683<-- 4 6 8 3
calc_kap=:  -/@: sort0                       NB. calc
```

- 3 桁は 495

```
  > calc_kap (L:0) (^:8) 428 ; 534 ; 873 ; 172
495 495 495 495
```

- 3 桁の数の間でも収束までの演算回数は異なる。534 は遅い

```
 calc_kap (L:0) ^:(i.8) 428 ; 534 ; 873 ; 172
```

- 5 桁-8 桁は巡回する

```
   calc_kap  ^:(i.8)  53278
53278 63954 61974 82962 75933 63954 61974 82962
```

- 次に現れるのは９桁である

```
   calc_kap (L:0) ^:(i.8)  53278;639162;7364915;62531864;631748925 ;8019267354x
|53278|639162|7364915|62531864|631748925|8019267354|
|63954|842652|8419752|74308653|864197532|9753086421|
|61974|640854|8629632|84308652|864197532|9753086421|
|82962|820872|7629633|86308632|864197532|9753086421|
|75933|864432|7429653|86326632|864197532|9753086421|
|63954|629964|7419753|64326654|864197532|9753086421|
|61974|749943|8429652|43208766|864197532|9753086421|
|82962|652644|7619733|85317642|864197532|9753086421|
```

- この方法は桁数が大きくなり表示に浮動小数点が出てくるとうまくいかない。常に拡張表示 (`1 x:`) を用いて反復するように改良する

```
calc_kap=: 1 x: -/@: sort0      NB. calc
```

1.2 カプレカーの自己数

カプレカーが 1949 年に発見した

> 自己数とは生成元を持たない数である。素数である自己数は自己素数と呼ばれる。

桁足し　先の divide で分離した数を（桁に関係なく）全て足す。47 → 11 (→ 2)

生成数　47 の桁足し 11 を 47 に加えると 58　(生成数)。元の 47 を生成元という。自己数とは生成数を持たない数を言う。

カプレカの自己数の判別法　全ての自己数を生成する再帰的でない公式は見つかっていないが、任意の数についてそれが生成数か自己数かを判別する簡単な方法をカプレカーは発見している。

生成数の数列　divide を用いて作成できる。(`[]`) は右引数でこの場合は逐次顕れた数を引用する

```
 []+ +/@: divide) ^:(i.10) 47
47 58 71 79 95 109 119 130 134 142
```

そこで問題　1974 年の次の自己数は何年か

1.2.1 カプレカの判定法

1975　　自己数の判定法

1. 桁足しを行う
   1975 → 22

```
   add_column 1975
22
```

2. 一桁の数になるまで桁足しを繰り返す
   1975 → 22 → 4
```
   find_C_sub 1975
4
```
3. 最後の桁足し数（4）が偶数なら $\frac{1}{2}$ し、奇数なら 9 を加えて $\frac{1}{2}$ する。この数を **C** とする
```
>   find_C (L:0)1975; 1996
2 8
```
4. N = 1975 から C = 2 を引いた数 (1973) を対象に (1) の桁足しを行う（22)。1973 + 22 ≠ 1975
5. 1973 から 9 を引いた 1964 を対象に桁足し (1) を行う。1964 + 20 ≠ 1975
   以降 9 づつ引いて桁数まで繰り返す。ここでは 4 回でフィットしなければ自己数である
6. 1955 + 20 = 1975 ここでフィットしたので 1975 は自己数ではなく 1955 で生成された数である
7. 1974 はフィットしないので自己数である
```
   check_self (L:0) 1974; 1975
   1974(C=6)            1975(C=2)
|   6   15   24   33|   2   11   20   29| NB. C C+9 C+9*2 C+9*3
|  24   24   15   15|  20   20   20   20| NB. add_column
|   0    0    0    0|   0    0    1    0| NB. 1 is fit
```

1900 年代の自己数　check_self_many は自己数を抜き出す
```
   check_self_many 1900+i.100
1906 1917 1919 1930 1941 1952 1963 1974 1985 1996
```
[*1]

自己素数　いずれも自己素数ではない。1 p: n は素数の判定、q:は因数分解を行う。
```
   1 p: 2 3 4
1 1 0


 1 p: check_self_many 1900+i.100
0 0 0 0 0 0 0 0 0 0
a,. q: a=. check_self_many 1900+i.100
1906  2 953   0  0 0  0
1917  3   3   3 71 0  0
1919 19 101   0  0 0  0
1930  2   5 193  0 0  0
1941  3 647   0  0 0  0
1952  2   2   2  2 2 61
1963 13 151   0  0 0  0
1974  2   3   7 47 0  0
```

[*1] 1919 は自己数と思うが

```
1985  5 397   0  0 0  0
1996  2   2 499  0 0  0
」
```

10,100,1000,10000,10000,100000 **は自己数ではない**　これらは 1 のフラグの立った上段の数からの生成数である

```
 check_self(L:0) 10;100;1000;10000;100000
 10    100     1000                    10000                    100000
|5|95 86 77|995 986 977 968|9995 9986 9977 9968 9959|99995 99986 99977 99968 99959 99950|
|5| 5 14 23|  5  14  23  32|   5   14   23   32   41|    5    14    23    32    41    50|
|5|14 14 14| 23  23  23  23|  32   32   32   32   32|   41    41    41    41    41    32|
|1| 0  1  0|  0   0   1   0|   0    0    0    1    0|    0     0     0     0     1     0|
```

100 **万は自己数である故貴重である**　.

```
  check_self 1000000
999995 999986 999977 999968 999959 999950 999941
     5      14      23      32      41      50      59
    50      50      50      50      50      41      41
     0       0       0       0       0       0       0
```

2　References

マーチン · ガードナー 一松　信訳「マーチン · ガードナーの数学ゲーム I(新装版)」日経サイエンス社　2010

付録 A　Script

```
NB. Kaprekar number
NB. written by SHIMURA Masato //15 Apr 2011
NB. Usage  calc_kap ^:(i.10) 4683
NB. 4683 5175 5994 5355 1998 8082 8532 6174 6174 6174
NB. Usage: calc_kap(^:10)L:0 (4683; 5472; 3973)
NB. --------------------
NB. | 6174| 6174| 6174|
NB. --------------------
divide=: >@ (". L:0)@( {@>)@ ": NB. 4683 --> 4 6 8 3
sort0=:[: > [: 10&#. L:0  (\:~ ; /:~)@divide NB. down&up sort
NB.  10&#. is  connect 4683<-- 4 6 8 3
calc_kap=: 1 x: -/@: sort0      NB. calc

NB. --------------------------------
NB. Kaprekar's self numbers
NB. *add_column  1975
NB. *find_C_sub1 1975
NB. *find_C 1985
NB. check_self (L:0) 1974 1975
NB. check_self_many 2000+i.100
add_column =: +/@:divide

find_C_sub=: 3 : 0
NB. add keta /do until  1 dight
NB. usage: u 1975
NB. ans is 22 4
tmp=. y
while. (# divide tmp)> 1 do.
tmp=. add_column tmp
end.
)

find_C=: 3 : 0
NB. usage: u 1975
NB. halve but if odd before add 9
C0=. find_C_sub y
select. 1= 2| C0           NB. even or odd
```

```
 case. 0 do.  C=. -: C0    NB. even
 case. 1 do.  C=. -: 9+C0  NB. odd
end.
)

check_self=: 3 : 0
tmp=.  # divide y
C=: find_C y
TGT=: +/\C,(<:tmp)#9
IND=: 1= *y-TGT
TGT=: IND#TGT
VAL=: > add_column L:0 {@>  y-TGT
(y-TGT),TGT,VAL,:y=VAL+y-TGT
)

NB. calc at once and pick self numbers
NB. check_self_many 1900+i.100
NB. 1906 1917 1919 1930 1941 1952 1963 1974 1985 1996
check_self_many=: 3 : 0
y#~ 0= > +/@: {: L:0 check_self L:0 {@> y
)
```