JAPLA 研究会資料 2021/2/15

# Ｊでピーター・フランクル数列を計算する

西川　利男

先の JAPLA 研究会でＪによるコラッツの数列の計算を報告した。
コロナ禍の自粛とて、雑誌数学セミナーを拾い読みしていたら、テレビタレントとしても有名なピーター・フランクルが同誌のコラム「エレガントな解答を求む」に似たような数列を発表しているのを見つけた。[1]
[1]「数学セミナー」2020，９月号，p.88，　日本評論社
数学としての解答及び検討は同誌にあるが、私は先のコラッツの数列の続きとしてＪで計算してみた。

## １．ピーター・フランクル数列

数列の生成のルールは同誌より次のとおりである。

---

出題 1　◉出題者

**ピーター・フランクル**

四桁の数字 $A$ に対し，次の二つの操作を好きな順番で繰り返して，1 に辿り着きたい．

（1）　各桁の数字の順番を入れ換える(ただし 0 を先頭に置いてはいけない)
（2）　偶数を 2 で割る

例えば $A = 1000$ の場合は，最初は(2)をやるしかない．

$1000 \to 500 \to 250 \to 520 \to 260 \to 130 \to 310$
$\to 155$

とするとダメになる．

しかし

$1000 \to 500 \to 250 \to 125 \to 512 \to 256 \to 128$
$\to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1$

なら目標達成だ．

さて，1 に辿り着くことができる四桁の数字 $A$ をすべて求めてください！

---

## ２．Ｊによるピーター・フランクル数列の計算のプログラム

フランクル数列の１に至る過程の探索を行うには、その小道具となる単位操作プログラム定義が必要である。それぞれ次ぎのようになる。

・与えられた数Ｎの桁数を得る
　＝　10を底とする対数をとり、その小数点以下を切り捨てる
　　<. 10 ^. N
・与えられた10進数を１桁ずつバラバラの数値にする
　＝　桁数だけの10を左引数として#. により得られる
　　( (<. 10 ^. N) # 10) #. N
・与えられた数Ｎが偶数か奇数を判定する
　＝　２で割った余りが０なら偶数、１なら奇数
　　2 | N

以上の準備をした上で、フランクル数列の生成の規則を書き上げる。
与えられた数を引数として、プログラムｆを以下のように定義した。

上の小道具を使って、１桁の場合、２桁の場合、３桁の場合、４桁の場合とそれぞれ分けて行うことが必要になる。これにはＪの select. case. do. 構文を用いて行う。

```
f =: 3 : 0
N =. y.
if. 0 = N do. 0 return. end.
Keta =. <. 10 ^. N
NN =. (Keta#10) #: N
select. Keta
  case. 4 do.
    'D C B A' =. NN
    if. 1 = */2|NN do. 0 return. end.
    if. (0 = 2|A)  do. -: N return. end.
    if. (0 = 2|B) do. (4 # 10) #. A, C, D, B return. end.
    if. (0 = 2|C) do. (4 # 10) #. A, B, D, C return. end.
    if. (0 = 2|D) do. (4 # 10) #. A, B, C, D return. end.
  case. 3 do.
    'C B A' =. NN                          NB. 100位をC, 10位をB, 1位をA
    if. 1 = */2|NN do. 0 return. end.      NB. すべて奇数なら 0を返す
    if. 0 = 2|A do. -: N return. end.      NB. Aが偶数なら2で割る
    if. (0 = 2|B) do. (3 # 10) #. A, C, B return. end.
                  NB. Bが偶数なら1位をBとし 100位をA、10位をCとする
    if. (0 = 2|C) do. (3 # 10) #. A, B, C return. end.
                  NB. Cが偶数なら1位をCとし 100位をA、10位をBとする
  case. 2. do.
    'B A' =. (10)#:NN
     if. 1 = */2|NN do. 0 return. end.
     if. 0 = 2|A do. -: N return. end.
     if. 1 = 2|A  do. (2 # 10) #. A, B return. end.
  case. 1  do.
     if. (0 = 2 | N) do. -: N return. end.
     if. (1 = 2 | N) do. 0    return. end.
end.
```

このような動詞ｆを定義すれば、これを用いて１ステップずつ数列の探索をおこなうことができる。

値1000に対して、動詞ｆを作用させて、すこしずつやってみよう。

```
f 1000 => 500
f 500 => 250
f 250 => 125
f 125 => 512
f 512 => 256
f 256 => 128
f 128 => 64
f 64 => 32
f 32 => 16
f 16 => 8
```

```
   f 8 => 4
   f 4 => 2
   f 2 => 1
```
とめでたく、1にたどりつくことができた。
　Jでは、便利な副詞power ^: を用いて、つぎのようにして、繰り返し実行をおこなうことができる。
```
   f (^: 1) 1000
500
   f (^: 2) 1000
250
```
つまり、動詞fを1回、２回と作用させる。
　さらに、これをまとめて行うには、つぎのようにする。
```
   f (^: (1, 2)) 1000
500 250
```
　動詞i.を用いて、次のように行える。
```
   f (^: (i.16)) 1000
1000 500 250 125 512 256 128 64 32 16 8 4 2 1 0 0
```
ここで、最後に奇数1となったので、その後は、0がつづくことになる。

　別の値1002でやってみる。
```
   f (^: (i.16)) 1002
1002 501 150 75 0 0 0 0 0 0 0 0 0 0 0 0
```
　なお、途中501に対して、510という選択もできるが、このときも見つからない。
```
   f (^: (i.16)) 510
510 255 552 276 138 69 96 48 24 12 6 3 0 0 0 0
```

## ３．実行に当たってーいくつかのプログラムの手直し

とりあえず作ったピーター・フランクル数列をプログラムｆを実行して、判明したことがある。
まずは３桁の値の場合を取り上げる。３桁の値をそれぞれ次のように分ける。

百位　　十位　　一位
C　　B　　A

・すべて奇数のときは、不成功　　→　０を返す
・一位Ａが偶数のとき、２で割る　　→　次へ進む

上のような単純な場合以外に、次のような場合を考慮しなくてはならない。
偶数の数字を一位とするような数字の入れ替えが必要である。

・偶数の桁が１つある場合
　　百位Ｃが偶数のとき　→　Ａ Ｂ Ｃ
　　→　Ｂ Ａ Ｃ
　　十位Ｂが偶数のとき　→　Ａ Ｃ Ｂ
　　→　Ｃ Ａ Ｂ
・偶数の桁が２つある場合
　　百位Ｃが偶数のとき　→　Ａ Ｂ Ｃ
　　→　Ｂ Ａ Ｃ
　　十位Ｂが偶数のとき　→　Ａ Ｃ Ｂ
　　→　Ｃ Ａ Ｂ

いずれにしても、２つの選択をしなければならない。そのため、プログラムの途中でマニュアルでの２つの分岐を行うようにつぎのように手直してfff とした。

```
  case. 3 do. NB. 3 keta ======
    'C B A' =. NN
    if. (0 = 2|A) do. -: N return. end.
    if. 1 = */2|NN do. 0 return. end.
    wr 'Enter 1 or 2'
    yn =. rd 1
    if. '1' = yn
      do.
        (3 # 10) #. C, A, B return.
      else.
        (3 # 10) #. A, C, B return.
    end.
```

上の計算を繰り返し実行するには、Ｊの副詞パワー　^:を用いたプログラムで行うことができる。

つまり、繰り返し実行の際、１が得られたら成功、
１が得られず０で続くときはたら不成功、とするような最終のプログラムを
frankとして、次のように定義した。

```
frank =: 3 : 0
40 frank y.
:
NB. F =: f (^: (i.x.)) y.
F =: fff (^: (i.x.)) y.
FF =: (F i. 0) {. F
if. (1 = {: FF)
  do. wr 'success'
  else. wr 'fail'
end.
FF
)
```

実行のようすは次のようになる。

```
   frank 1000
Enter 1 or 2
1
Enter 1 or 2
1
fail
1000 500 250 125 152 76 38 19
   frank 1000
Enter 1 or 2
2
Enter 1 or 2
2
success
1000 500 250 125 512 256 128 64 32 16 8 4 2 1
   frank 1000
Enter 1 or 2
1
Enter 1 or 2
2
success
1000 500 250 125 512 256 128 64 32 16 8 4 2 1
   frank 1000
Enter 1 or 2
2
Enter 1 or 2
1
fail
1000 500 250 125 152 76 38 19
```

つまり、成功として1が得られるときもあるが、選択の仕方によっては、1が得られず不成功になるときもある。

別の値でやってみよう。

```
   frank 1002
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
2
Enter 1 or 2
2
Enter 1 or 2
2
fail
1002 501 150 75
   frank 1002
Enter 1 or 2
2
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
```

```
2
Enter 1 or 2
2
fail
1002 501 510 255 255 525 552 276 138 69 96 48 24 12 6 3


   frank 1004
Enter 1 or 2
2
Enter 1 or 2
2
Enter 1 or 2
2
Enter 1 or 2
2
success
1004 502 251 125 512 256 128 64 32 16 8 4 2 1


   frank 1006
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
2
Enter 1 or 2
1
fail
1006 503 530 265 526 263 236 118 59
   frank 1006
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
1
Enter 1 or 2
1
success
1006 503 530 265 256 128 64 32 16 8 4 2 1
```

## ４．おわりに

まことに、すっきりしない妙なＪプログラムとなった。本当は４桁の値についても、行わなくてはならないが、トライアンドエラーを重ねており、なかなか大変なことで、報告するには至らなかった。

プログラムリスト

```
f =: 3 : 0
N =. y.
if. 0 = N do. 0 return. end.
Keta =. >. 10 ^. N + 0.1
NN =. (Keta#10) #: N
select. Keta
  case. 4 do. NB. 4 keta ======
    'D C B A' =. NN
    if. 1 = */2|NN do. 0 return. end.
    if. (0 = 2|A)  do. -: N return. end.
    if. (0 = 2|B) do. (4 # 10) #. A, C, D, B return. end.
    if. (0 = 2|C) do. (4 # 10) #. A, B, D, C return. end.
    if. (0 = 2|D) do. (4 # 10) #. A, B, C, D return. end.
  case. 3 do. NB. 3 keta ======
    'C B A' =. NN
     if. 1 = */2|NN do. 0 return. end.
     if. 0 = 2|A do. -: N return. end.
     if. (0 = 2|B) do. (3 # 10) #. A, C, B return. end.
     if. (0 = 2|C) do. (3 # 10) #. A, B, C return. end.
  case. 2 do. NB. 2 keta =======
    'B A' =. (10)#:NN
    if. 1 = */2|NN do. 0 return. end.
    if. 0 = 2|A do. -: N return. end.
    if. 1 = 2|A  do. (2 # 10) #. A, B return. end.
  case. 1 do. NB. 1 keta =======
    if. (0 = 2 | N) do. -: N return. end.
    if. (1 = 2 | N) do. 0    return. end.
end.
)

wr =: 1!:2&2
rd =: 1!:1


fff =: 3 : 0
N =. y.
if. 0 = N do. 0 return. end.
Keta =. >. 10 ^. N + 0.1
NN =. (Keta#10) #: N
select. Keta
  case. 4 do. NB. 4 keta ======
    'D C B A' =. NN
    if. 1 = */2|NN do. 0 return. end.
    if. (0 = 2|A) do. -: N return. end.
    if. (0 = 2|B) do. N =: (4 # 10) #. A, C, D, B return. end.
    if. (0 = 2|C) do. N =: (4 # 10) #. A, B, D, C return. end.
    if. (0 = 2|D) do. N =: (4 # 10) #. A, B, C, D return. end.
  case. 3 do. NB. 3 keta ======
    'C B A' =. NN
    if. (0 = 2|A) do. -: N return. end.
    if. 1 = */2|NN do. 0 return. end.
    wr 'Enter 1 or 2'
    yn =. rd 1
    if. '1' = yn
      do.
        (3 # 10) #. C, A, B return.
      else.
        (3 # 10) #. A, C, B return.
```

```
    end.
  case. 2 do. NB. 2 keta =======
    'B A' =. (10)#:NN
    if. 1 = */2|NN do. 0 return. end.
    if. 0 = 2|A  do. -: N return. end.
    if. 1 = 2|A  do. (2 # 10) #. A, B return. end.
  case. 1 do. NB. 1 keta =======
    if. (0 = 2 | N) do. -: N return. end.
    if. (1 = 2 | N) do. 0    return. end.
end.
)

frank =: 3 : 0
40 frank y.
:
NB. F =: f (^: (i.x.)) y.
F =: fff (^: (i.x.)) y.
FF =: (F i. 0) {. F
if. (1 = {: FF)
  do. wr 'success'
  else. wr 'fail'
end.
FF
)

NB.   frank 1024
NB. success
NB. 1024 512 256 128 64 32 16 8 4 2 1
NB.   frank 1000
NB. success
NB. 1000 500 250 125 512 256 128 64 32 16 8 4 2 1
NB.   frank 976
NB. success
NB. 976 488 244 122 61 16 8 4 2 1
NB.   frank 116
NB. success
NB. 116 58 29 92 46 23 32 16 8 4 2 1
NB.   frank 1952
NB. success
NB. 1952 976 488 244 122 61 16 8 4 2 1
NB.   frank 2048
NB. success
NB. 2048 1024 512 256 128 64 32 16 8 4 2 1


NB.   frank 1952
NB. success
NB. 1952 976 488 244 122 61 16 8 4 2 1
NB.   frank 2048
NB. success
NB. 2048 1024 512 256 128 64 32 16 8 4 2 1
```