回文数と 196 問題

SHIMURA Masato
JCD02773@nifty.com
URL:http://homepage3.nifty.com/asagaya_avenue

2012 年 9 月 23 日

目次

1 はじめに 1

2 回文数の解析 3

1 はじめに

- みつはしはつみ
- かつしかしつか
- ますふちふすま (店)

手許にある名前から回文を作ってみた。いろは 48 文字を一度ずつ使う回文の名作もあるようだ。

数にも回文数 (palindrome numbers) がある。次のように順序を逆にした数を加えて回文数を作る。何回か繰り返すと回文数が現れる。

西山豊 [1] に回文数の詳細な解説がある。氏は日本 IBM の出身で計算科学面での詳細な検討もなされている。

```
   kaibun 285
0  285 0
1  867 0
2 1635 0
3 6996 1
```

```
285   +   582   =   867
867   +   768   =  1635
1635  +  5361   =  6996
```

1.1 経過と解説

一個の数の回文数を作るスクリプトを作成する

1. 数をリバースする。見た目よりも難しい。基底変換を用いた。

   antibase(#:) を用いて 10 進法で分解する

```
 10 10 10 #:  285
2 8 5
```

```
|. 10 10 10 #:  285
5 8 2
```

```
 reverse=: 3 : 0
Num=:# ": y
10&#. |.(Num # 10) #: y
)
```

   base(#.) で 10 進法に戻す

```
10&#. |. 10 10 10 #:  285
582
```

2. *y* + *reverse y*

```
kaibun_sub0=: 3 : ' y + reverse  y'
```

3. 回文の判定。 解と解をリバースしたものを比べる。回文になっていればループを抜け出し、なってなければ繰り返す。

```
kaibun=: 3 : 0
NB.  kaibun 89x
Plus=. kaibun0 y
Ans=.< 0, y
 for_ctr. i. 25 do.
  Tmp=. (>: ctr), Plus, Judge=. Plus = reverse Plus
  Ans=. Ans,<Tmp
```

```
    if. 1 =  Judge  do. goto_ans.
       else.
          Plus=. kaibun_sub0 Plus
      end.
 end.
 label_ans.
;"1 ,.Ans
)
```

2　回文数の解析

一つずつ解を求めるには大変なのである程度まとめて、何回で回文になるかも調べる。25 回を越えると回文になりにくいことが知られている。

回文数の計算の反復回数は取り合えず 25 回としている。25 回で回文が見つからないものは個別に検討する

2.1　1-100

1. 多倍長で計算するため拡張精度を用いる。(100x など 1 箇所は x を付けておく)
2. 1-100 までの解析 84 のケース (=57+27) は 1,2 回で回文になる。

```
 100 analysis0_kaibun 1 100x
+-----+-------+
| 1 57|89 24 1|
| 2 27|98 24 1|
| 3  5|       |
| 4  4|       |
| 6  4|       |
|24  2|       |  NB. 2 case
+-----+-------+
```

3. 4 ケースは 6 回で回文になる。4 個目の数が 1 になっていれば回文数である。

```
 2}.  100 analysis_kaibun 1 100x

<x, times ,1=yes>
+------+------+------+-------+
|59 3 1|69 4 1|79 6 1|89 24 1|
|68 3 1|78 4 1|88 6 1|98 24 1|
|77 3 1|87 4 1|97 6 1|       |
|86 3 1|96 4 1|99 6 1|       |
|95 3 1|      |      |       |
+------+------+------+-------+
```

4. 89 と 98 は 24 回で回文になり 1-100 は全て回文になる

2.2 196 問題

1. 101-200 の回文数

```
 100 analysis0_kaibun 101 200x
+------+---------+
|  1 44|196 100 0|
|  2 22|         |
|  3 14|         |
|  4  7|         |
|  5  3|         |
|  6  1|         |
|  7  3|         |
|  8  1|         |
| 11  1|         |
| 15  1|         |
| 23  1|         |
|100  1|         | NB. 100 回の結果は右で 0 になる
```

```
+------+---------+
```

2. 高階の回文数は左引数でループ回数を指定して求める。

```
  23 analysis0_kaibun 101 200x
+-----+--------+
| 1 44|187 23 1|   NB. 187 は 23 回で 1
| 2 22|196 23 0|   NB. 196 は      0
| 3 14|        |
| 4  7|        |
| 5  3|        |
| 6  1|        |
| 7  3|        |
| 8  1|        |
|11  1|        |
|15  1|        |
|23  2|        |
+-----+--------+
```

3. 196 は 25 回で回文になっていない。196 が回文が見つからない最小の数で、かってアシモフが問題にしていた。西山の調査によると 1930 年代にアシモフ以前に既に提起されていたようだ。

2.3 1-2000

1. 2000 までで回文になる回数 (左) と 25 回までに回文にならなかった数のリスト (右) メモリ節約のため回文数は省いてある

```
    25  analysis0_kaibun  1 2000x
+------+---------+
| 1 785| 196 25 0|
| 2 538| 295 25 0|
| 3 294| 394 25 0|
```

```
| 4 143| 493 25 0|
| 5  57| 592 25 0|
| 6  39| 689 25 0|
| 7  30| 691 25 0|
| 8  12| 788 25 0|
| 9   7| 790 25 0|
|10   2| 879 25 0|
|11   8| 887 25 0|
|13   5| 978 25 0|
|14   3| 986 25 0|
|15   9|1495 25 0|
|16   9|1497 25 0|
|17   8|1585 25 0|
|18   3|1587 25 0|
|19   1|1675 25 0|
|21   8|1677 25 0|
|22   2|1765 25 0|
|23   8|1767 25 0|
|24   2|1855 25 0|
|25  26|1857 25 0|
|       |1945 25 0|
|       |1947 25 0|
|       |1997 25 0|
+------+---------+
```

2. J の拡張精度 (x) を用いないと内部でどこまで計算しているかが良く見えない。1000 以下の収束しない数を 1000 回まで回してみた (表示は 100 回で) が回文は得られなかった。しかし 2 のパターンに収斂している

```
;("1) ,. , L:0  kaibun L:0 {@> 196 295 394 493 689  788 790

196 100 4475777153449051561729069927156150844362774644 0
295 100 4475777153449051561729069927156150844362774644 0
```

```
394 100 44757771534490515617290699271561508443627774644 0
493 100 44757771534490515617290699271561508443627774644 0
689 100 89405544168971032134590308543213017887145550388 0
788 100 89405544168971032134590308543213017887145550388 0
790 100 44757771534490515617290699271561508443627774644 0
```

3. ここからは計算力学の世界である。一つの数をどこまでも計算する。力が尽きたらハングするが 1!:2&2 を用いて落ちる直前まで画面に計算過程を結果を書き込むようにした

```
kaibun_manytimes=: 4 : 0
NB. Ussage: 1100 kaibun_manytimes 196x
Plus=. kaibun_sub0 y
 for_ctr. i. x do.
1!:2&2  Tmp=. (>: ctr), Plus, Judge=. Plus = reverse Plus
   if. 1 =  Judge  do.   goto_ans.
     else. Plus=. kaibun_sub0 Plus
   end.
  end.
 label_ans.
y, 0 2 {Tmp
)
```

4. 表示の一部。 x を 1000,10000 などと指定する。1000 回で回文とならない場合はほとんどが回文でない。
   西山によれば 100 回を越える数は 10^{10}、200 回を越える数は 10^{15} を超える。確認されている範囲では 236 回で 10^{16} のオーダーである。

5. 更にメモリの節約。 回文の数値を抜いて回数と 0/1 のみの表示にする。
   `1!:2&2  Tmp=. (>: ctr), Plus, Judge=. Plus = reverse Plus`
   → `1!:2&2  0 2{ Tmp=. (>: ctr), Plus, Judge=. Plus = reverse Plus`
6. 196x を 5000 回試してみた。非力なノートでも計算できたがメモリーの限界を超えると J 言語のみならず、システム全体がメモリーを奪われてハングすることもあ

るのでご用心

```
  5000  kaibun_manytimes 196x

5000 0
196 5000 45326625029496741376995950392294478427312476234131647301602740430549409692006638558733443305893652417563759393217460976594162095279347606803487356124987390036147598860359191989086476536575747659386997500099740095749424666806421785246525440955285 21...
```

References

西山豊「数学の楽しみ」現代数学社 2007