サイエンティフィック・システム研究会　　2011 年 10 月 20 日

セルフデベロップメントとプログラミング

和田英一

IIJ イノベーションインスティテュート

講義を聞いたり本を読んだりしても, その内容がすぐに理解できるわけではない. 思い返し, 考え直し, 理解したいという努力の継続の結果, 少しずつ身につく. ところで, その理解の努力を強力に援助してくれるのが, 計算機である. 計算機のプログラムを書き, 実験を繰り返すと, それまで中途半端だった理解が急に深まることが少なくない.

手で繰り返す実験はたかが知れているが, 計算機を使えば膨大な処理がたちどこに出来, 多くのケースについて知見が得られる. そしてそれぞれのテーマが興味深いことを知る.

そういう面白い例のいくつかを紹介したい.

Self Development, Programming, Numerical Experiment, Puzzle, Computer Simulation, Graphic Languages

はじめに

富士通研究所初代社長の尾見半左右氏は 70 歳を過ぎてから学位論文「電子計算機の巨大化の限界に関する研究」を提出, 1973 年 9 月に東京工業大学から工学博士の学位を得られた. 論文を指導した東工大の川上正光教授から, その快挙に対し, 佐藤一斎の言志四録の「少にして学ばば即ち壮にして為すこと有り, 壮にして学ばば即ち老いて衰へず, 老いて学ばば即ち死して朽ちず」の言葉を贈られた.

いつまでも学ぶ態度を持ち続けるのが, セルフデベロップメント, 自己啓発であろう. 自己啓発は, 個人ごとに多くのやり方があって然るべきである. 私の場合, 仕事のプログラムは書くこともあるが, 自分のために書くことが殆んどである. 自分のため, すなわち自己啓発のためであって, 僭越ながら, 今日はそういう計算機活用の簡単で面白い例を紹介したい.

私がプログラミングに取り掛かるのは, 本を読んでいて, これは一体どういうことかという疑問を解こうとする時が多い. 特に翻訳をしている時は, 内容が分からないと訳せないわけだが, そういう場合に計算機が活躍する.

以下, あれこれやってみたのは, 元富士通社長 小林大祐氏の「ともかくやってみろ」精神のプログラミング版である.

Eratosthenes の篩

Eratosthenes の篩の話は, 子供の頃に読んだような気がするが, 大人になってからでも, やってみられるのは, せいぜい 100 くらいまでである. しかし, 素数定理を知ったとして, それを確かめるには, 1 万, 10 万とか 100 万くらいまで素数の個数を知る必要がある. 手元に計算機があれば, 篩は簡単に実行出来る.

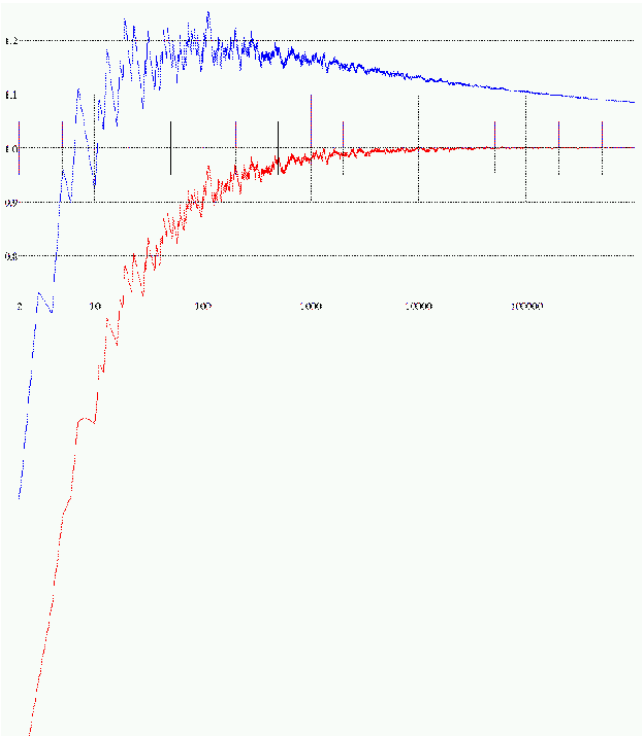
100 万までの Eratosthenes の篩をやってみよう.

```
(define length 500000)                          ; 篩は奇数だけなので半分でよい.
(define sieve (make-bit-string length #t))      ; 篩 sieve を作る 最初全部ビットをたてる
(bit-string-clear! sieve 0)                     ;1 の場所をクリアする
(define k 0) (define b 10) (define c 1)

(define (loop)
 (set! k (bit-substring-find-next-set-bit sieve (+ k 1) length))
 (if k (let ((p (+ k k 1)))                     ;k の場所が 1 なら 2k+1 が素数
    (if (> p b) (begin (display (list b c)) (set! b (* b 10))))
    (set! c (+ c 1))                            ; カウントを増やす
    (do ((q (/ (- (* p p) 1) 2) (+ q p))) ((>= q length))
       (bit-string-clear! sieve q)) (loop))     ;(p*p-1)/2 から p 置きにビットをクリア
  c))

(loop)

(10 4)(100 25)(1000 168)(10000 1229)(100000 9592)
;Value: 78498
```



100 万までには, 78498 個の素数のあることが分かる.
Gauss の素数定理では (/ 1000000 (log 1000000)) なので計算してみると
⇒ 72382.41365054197
一方 Legendre の定理では
(/ 1000000 (- (log 1000000) 1.08366))
⇒ 78543.1776352779
この 2 つの素数定理を図にしたのが左だ.

Wilson の定理 ($(p-1)! \equiv -1 (\bmod p)$ iff p is prime or 1, 従って $\frac{(x-1)!+1}{x}$ は, x が素数か 1 の時に限り整数) を読んだら, プログラムを書く.

```
(define (wilson x)
 (/ (+ (factorial (- x 1)) 1) x))

(map wilson (a2b 2 20))
 ;(a2b a b) は a から b-1 までのリスト
(1 1 7/4 5 121/6 103 5041/8 40321/9 362881/10 329891
 39916801/12 36846277 6227020801/14 87178291201/15
 1307674368001/16 1230752346353 355687428096001/18
 336967037143579)
```

Carmichael 数

Fermat のテストを騙す数 (合成数なのに, 素数の Fermat テストを通過する) があるという. さっそくやってみる. 下の表の最左端は 4 から 28 までの合成数 n で, その右に $1 \le a < n$ について, n を法として a^n を計算したものである. その値が丁度 a になるものは赤字で示す.

4	1	0	1																	
6	1	4	3	4	1															
8	1	0	1	0	1	0	1													
9	1	8	0	1	8	0	1	8												
10	1	4	9	6	5	6	9	4	1											
12	1	4	9	4	1	0	1	4	9	4	1									
14	1	4	9	2	11	8	7	8	11	2	9	4	1							
15	1	8	12	4	5	6	13	2	9	10	11	3	7	14						
16	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1					
18	1	10	9	10	1	0	1	10	9	10	1	0	1	10	9	10	1			
20	1	16	1	16	5	16	1	16	1	0	1	16	1	16	5	16	1	16	1	
21	1	8	6	1	20	6	7	8	15	13	8	6	13	14	15	1	20	15	13	20

```
22  1  4  9 16  3 14  5 20 15 12 11 12 15 20  5 14  3 16  9  4  1
24  1 16  9 16  1  0  1 16  9 16  1  0  1 16  9 16  1  0  1 16  9 16  1
25  1  7 18 24  0  1  7 18 24  0  1  7 18 24  0  1  7 18 24  0  1  7 18 24
26  1  4  9 16 25 10 23 12  3 22 17 14 13 14 17 22  3 12 23 10 25 16  9  4  1
27  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26  0  1 26
28  1 16 25  4  9  8 21  8  9  4 25 16  1  0  1 16 25  4  9  8 21  8  9  4 25 16  1
```

さて, Carmichael 数, 例えば $n = 561$ では, その段全体が赤字になるものである.

```
(map (lambda (a) (modulo (expt a 561) 561)) (a2b 1 561))
=> (1 2 3 4 5 6 7 8 9 10 ... 560)
```

面白い! Carmichael 数には, 1105, 1729, 2465, 2821, 6601,... などがあるらしい.

関数を使った分類ルーチン

Sussman さんたちの Struncture and Interpretation of Computer Programs を訳していた時であった. 普通なら左のように書くプログラムが右のように書いてあった.

```
(define (extract sequence)
  (if (null? sequence) (cons '() '())
    (let ((result (extract (cdr sequence)))
          (next-item (car sequence)))
      (let ((a (car result)) (b (cdr result)))
        (if (even? next-item)
           (cons (cons next-item a) b)
           (cons a (cons next-item b)))))))

(let ((result (extract '(3 1 4 1 5 9 2))))
  (display (car result)) (display (cdr result)))
```

```
(define (extract sequence receive)
 (if (null? sequence)
   (receive '() '())
   (extract (cdr sequence)
    (lambda (a b)
      (let ((next-item (car sequence)))
        (if (even? next-item)
          (receive (cons next-item a) b)
          (receive a (cons next-item b))))))))
(extract '(3 1 4 1 5 9 2)
 (lambda (a b) (display a) (display b)))
(4 2) (3 1 1 5 9)
```

これなどは, 実行してみなければ, どう動くのか分からない. extract が渡す関数の中で引数を出力するようにして走らせてみる.

```
(define (extract sequence receive)
 (if (null? sequence)
   (receive '() '())
   (extract (cdr sequence)
    (lambda (a b)
      (display a) (display b) (newline) ←この行挿入
      (let ((next-item (car sequence)))
        (if (even? next-item)
          (receive (cons next-item a) b)
          (receive a (cons next-item b))))))))
```

```
出力
()()
(2)()
(2)(9)
(2)(5 9)
(2)(1 5 9)
(4 2)(1 5 9)
(4 2)(1 1 5 9)
(4 2)(3 1 1 5 9)
```

出力を見ると, どう動いたかが判明する. つまり, 関数 extract は, 順次に短くなるリストとある関数を引数として, 繰り返し呼び出される. 最後に空リストで呼び出されると, 渡された関数の引数 a と b に空リストを渡して呼び出す.

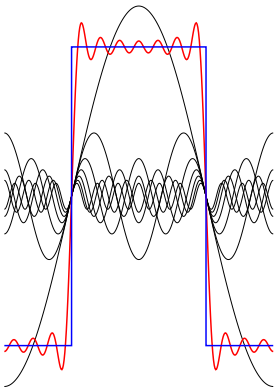
すると, 呼び出された関数 (1 段外で渡した引数の関数) が, 2 つの空リストを引数として呼ばれたので, その本体が実行され, 自分が渡されたリストの先頭が偶数か奇数かにより, 次に渡された関数を呼び出すための引数を用意し, 呼び出す.

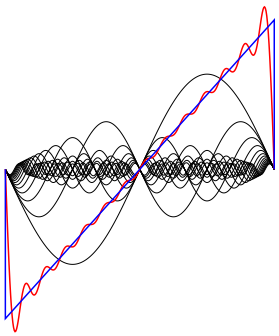
最後は, 最初に extract を呼び出した時に渡した関数が呼び出され, その引数が本体内で使えることになる. 巧妙な関数呼出し方である.

Fourier 変換

大学 3 年の時, 坪井忠二先生の演習で, 周期関数の Fourier 変換を計算する課題があった. 学部学生は手回し計算機も使わせて貰えず, 三角関数との掛算は計算尺, 合計には算盤を使った.

しかし, パラメトロン計算機が自由に使えるようになると, Fourier 変換は何でもなかった. 本に書いてあるように分解出来, 合成出来た.





f(0)	cos(6π/N)
f(1)	cos(2π/N)
f(2)	cos(4π/N)
f(3)	cos(6π/N)
f(m)	cos(2mπ/N)
f(N-1)	cos(2(N-1)π/N)

上の図で左はステップ関数, 中は鋸波関数. どちらも青線が元の関数, 黒が各周期の三角関数, 赤が合成した結果である.

Fourier 変換は大体次のようなプログラムを書くであろう.

```
(define nn 64)

(define (a n)
 (let ((an 0))
  (do ((m 0 (+ m 1))) ((= m nn))
    (set! an (+ (* (f m) (cos (/ (* 2 pi (+ (* m n) 0.5)) nn))) an)))
   (/ an nn)))
```

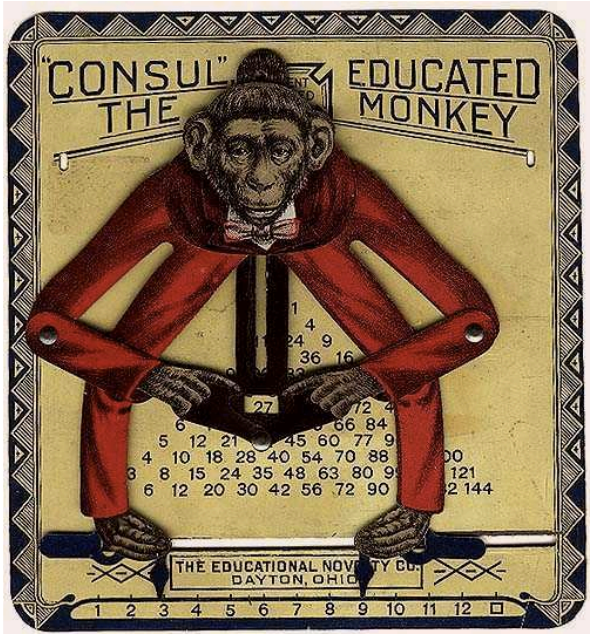
しかし, 演習の方法は違って, 上の図の右にあるような周期関数の表と, 三角関数の表をまず用意する. そして対応に注意しながら計算を続けるのであった.

最近は計算機が高速なので, 三角関数を何度計算しても痛痒を感じないが, ちゃんとしたプログラムを書くなら, 考えなければならない.

```
(define (a n)
 (/ (apply + (map (lambda (m) (* (vector-ref cs (modulo (* n m) 64))
  (vector-ref fs m))) (a2b 0 64))) 64))
```

高橋秀俊先生のお奨めは, 三角関数の表も加法定理で作るのだが, これは 1 回きりなので, そう凝ることもあるまい.

学者猿コンサル

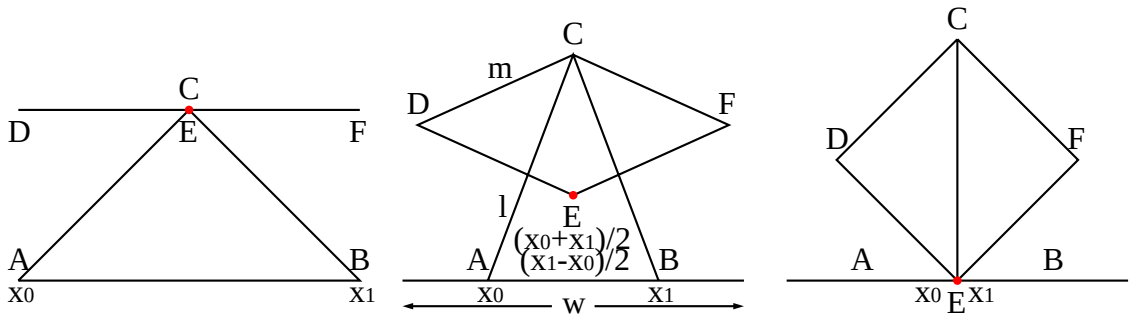


Consul, the Educated Monkey という計算関連の教育玩具がある. ウェブページで調べると, 1916 年頃にアメリカで売り出されたものらしい.

使い方はこうだ. 猿の両足を, 図のように下の目盛の 3 と 9 に合わせると, 猿の手が示す枠の中に積の 27 が現れるのである. 目盛の右端の 12 の右には四角が見えるが, 片足をそこに合わせると, 他方の足の二乗が現れる.

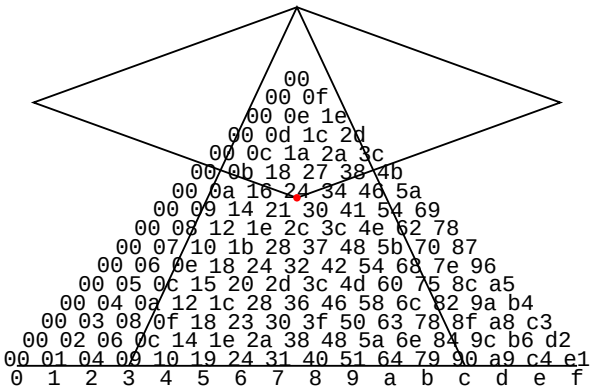
これは乗算表であるが, この表は差し換えることが出来, 加算にも使える.

次の図でその原理を示す. まず中の図で, 猿の足の位置が A, B(座標は x_0, x_1) とする. 足の動く範囲を w とする. 猿の頭が C で, AC の長さを l, C から肘 D までの長さを m とし, 頭と肘と解の位置関係は菱形とする. また∠ ACD は固定で, 今は 45° にし. たまた $l = w/\sqrt{2}$, $m = l/\sqrt{2}$ とする. そうすると, 解の点 E の座標は $(x_0 + x_1)/2, (x_1 - x_0)/2$ となる.



左は足を一杯に開いた時, 右は足を揃えた時の様子を示す.

右の図は, 上のパターンで十六進法の乗算表を作ってみたところである. 3 に c(つまり 12) を掛けると 24(十進で 36) になることを示す. x_1 が x_0 より左へ来ると, 解の位置は基線より下になる.

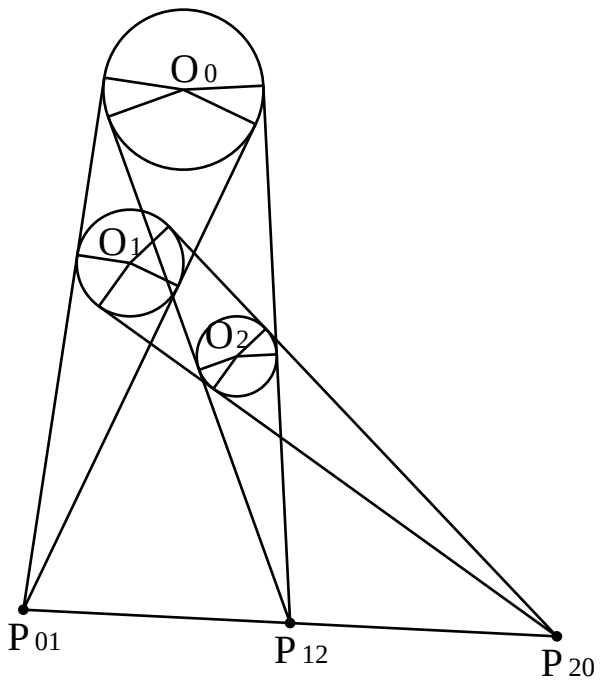


三つの円

夏の始め頃, 「とっておきのパズル」という本を送ってくれた人がいる. 面白く, しかも難しい問題が山のように書いてある本だ. うっかりはまると大変なので, なるべく近寄らないようにしてはいるが, 誘惑も絶ちがたく, ときどき, ちらと眺めては考え始めたりする.

そこに三つの円という幾何学の問題があった. 問題は簡単だが, どこから取り掛かったらよいかわからない. こういうパズルだ.

平面上の 2 つの円の共通接線の交点を, それらの円の焦点という. 3 つの円があると, 焦点は 3 個出来るわけだが, その 3 点は一直線上にあることを証明せよというのである.



手元に計算機があるから, とりあえず強引にやってみることにした. 図では確かにそうなる.

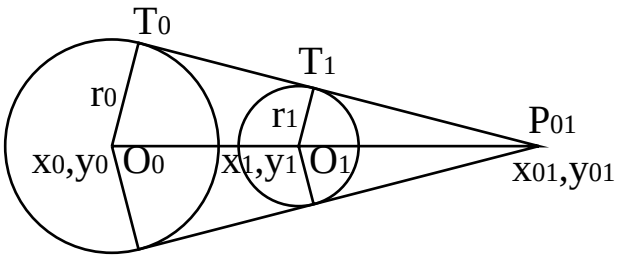
下の図の円 O_0 は中心の座標が x_0, y_0, 半径が r_0, 円 O_1 は座標が x_1, y_1, 半径が r_1 であるとする. また共通接線の 1 本と, 円との接点を T_0, T_1, 共通接線の交点を P_{01} とする. するとその座標 x_{01}, y_{01} が得られる. 同様にして

$x_{01} = (r_0x_1 - r_1x_0)/(r_0 - r_1)$, $y_{01} = (r_0y_1 - r_1y_0)/(r_0 - r_1)$

$x_{12} = (r_1x_2 - r_2x_1)/(r_1 - r_2)$, $y_{12} = (r_1y_2 - r_2y_1)/(r_1 - r_2)$

$x_{20} = (r_2x_0 - r_0x_2)/(r_2 - r_0)$, $y_{20} = (r_2y_0 - r_0y_2)/(r_2 - r_0)$

これから, この三角形の面積を計算すればよい. 3 点が一直線上にあれば, 面積は 0 になる. 私は risa/asir で計算した.



プログラムを調べるためのシミュレータやトレーサ

昔の計算機で使っていた面白そうなプログラムに出会うことがある. アセンブリ言語や機械語で書いてあると, どのように動くのか, 理解しにくい. 昔でなくても現今でも TAOCP の MMIX 言語を見ると, 途方に暮れることがある.

その時, 役に立つのが, 元の計算機のシミュレータである. 昔の計算機はアーキテクチャが簡単だから, シミュレータを書くのは何でもない.

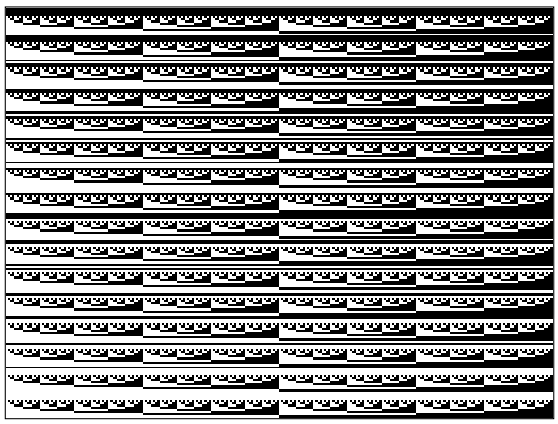
2 つの例を紹介したい. 慶応義塾大学名誉教授の大駒誠一氏は, 古い計算機のソフトウェア的復刻と称し, いくつものシミュレータを書いて, 昔のプログラムをシミュレートしている.

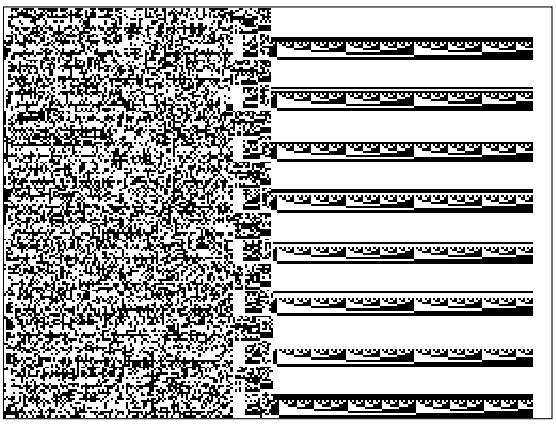
その大駒氏が, 私が 1958 年に書いた, パラメトロン計算機 PC-1 のイニシアルオーダーを解読しようとして, PC-1 のシミュレータを書き, イニシアルオーダーの機能が完全に理解できたといわれた. そこまですれば, そうであろうと思う.

私の場合は, DEC(Digital Equipment Corporation) のミニコン, PDP-8 のシミュレータである. ミニコンだけあって, 書くのは簡単であった.

私はそのシミュレータの上で, オランダの van der Poel 先生の作られた Lisp を走らせるのが目的であった. 通常のシミュレータだと, プログラムカウンタ, レジスタの内容が順次に出力されるが, このシミュレータでは, リスト処理のゴミ集め (Garbage Collection) を視覚化する.

ゴミ集めの視覚化は, 遥か昔, 慶応義塾大学の中西さんが, Apple コンピュータでやって見せたことがあり, そんなことがやってみたかったのである.





この図のそれぞれは, 1 語 12 ビットで 4096 語のメモリーを, 縦 192 ビット, 横 256 ビットで示す. 192 ビットは 12 ビットの語が下から 0 番地, 1 番地, ..., 15 番地と 16 語並び, そういう 16 語が, 256 段あって, 都合 4096 語になっている.

左は 0 番地には 0, 1 番地には 1, ..., 4095 番地には 4095 を格納したところである. 右は左半分が Lisp のプログラム, 右半分がヒープ領域で, ヒープ領域の各セルは, 2n 番地と 2n+1 番地の 2 語からなり, 奇数番地の car 部は 0, 偶数番地の cdr 部には, 次の自由セルの番地が入っている. Lisp のプログラムを入力するとちゃんと動く.

実際に動く様子は, この話の当日にお目にかける.

参考文献

佐藤一斎 川上正光注, 言志晩録 60, 言志四録 (三) 講談社学術文庫 276, 1980.
高橋秀俊, フーリエ変換よもやま話, 数理と現象, 岩波書店, 1975.
ピーター・ウィンクラー 坂井公他訳, とっておきの数学パズル, 日本評論社, 2011.