# あるパズル

玉井 哲雄
(筑波大学)

## 1　ことの始まり

この間，テレビでパスル特集という番組を長時間やっていて，そのごく一部を見たのだが，そこで次のような面白い問題が出された．

次の□に１から９までの数を重複なく入れて，等式を成り立つようにせよというのである．

$$\frac{\square}{\square\square}+\frac{\square}{\square\square}+\frac{\square}{\square\square}=1$$

その場で 30 分ぐらい考えてみたが，答は見つからなかった．暇とやる気のある方は，この先を読むのをやめて考えてみていただきたい．

少し試してみると，次のようなことが分かる．

1. 多くの組み合わせは，１を下回る．たとえば，もっとも大きそうな 9/14+8/25+7/36 でやっと 1.16 程度である（これが最大値かどうかは明かでない）．だから，1,2,3,4 のような小さな数字はなるべく分母にくるようにしなければならないし，7,8,9 のような大きな数字はなるべく分子に持ってくる必要がある．

2. ３つの分数を通分した結果は，比較的簡単な分母をもつはずである．そのためには，各分数が約分によって簡単化され，さらにそれらの分母どうしに公約数があるようなケースが想定される．そう考えるとやっかいなのは 5 と 7 である．とくに 5 は，分子になるとすると，5 を約数に持つ分母が考えられないし，また分母の下位の桁におくとすると，5 が分母の素因数として残って，それは他の２つの項の分母には含まれない約数となる，というように使い方が限られる．

この考察から，5 を分母の上位桁にもってきた 9/54 や 7/56 を含む場合を考えたが，うまくいかない．かなり近かったのが 9/12+3/48+7/56 だが，これは 15/16 となる．この辺であきらめた．このパズルは，視聴者からのファックスによる回答を受け付けるというものだったが，あとで家人に聞いたところによると，20 分で正しい答を送ってきた人がいたそうだ．ただ，解答そのものは家の者も聞いていない．

## 2　プログラムを作ってみることにする

その後もこの問題が気にはなっていたので，しばらくしてから，プログラムを書いて答を出そうという気になった．ここでやっと，多少はSEAMAILらしい話になるわけだ．

プログラムといっても，典型的な玩具プログラムである．要求仕様だのシステム設計だのという話ではない．次のような道具立てがあれば，後は簡単にできることはすぐ分かる．

1. 1 ～ 9 の順列を次々と生成する仕組み
2. 有理数の計算

既存のものをなるべく活用しようというのが，現在のソフトウェア工学の精神であろう．有理数を扱える手近な言語に Common Lisp があるので，それを使うことに決めた．Mathematica のような数式処理系を使ってもいいのだが，正直なところあまり使ったことがないので今回は敬遠した．

それでも順列の生成に便利な機能が組み込みであるなら，Mathematica を使ってみる気になるところだったが，ちょっと調べた限りではぴったりのものがない．たとえば (a b c) というリストが与えられて，その全順列をリストとして返す関数といったものはあるのだが，それではとても役に立たない（遅延評価でもあれば別だが）．きっとどこかには，適当なライブラリがあるに違いないが．

そこで順列生成は本から探すことにした．手近なところで見ると，野下浩平さんの「基本的算法（岩波情報科学講座）」に２つの有名なアルゴリズムが載っている．ただ，アルゴリズムの書き方は再帰的で，全部の順列を出力するものであり，当面の目的にはかなり変換する必要がある．もうひとつ思いだしたのが，Dijkstra の *Discipline of Programming* にあるアルゴリズムで，これを Pascal プログラムにしたのが，土居範久さんの Pascal の教科書に載っている．これは，野下さんの本にある方法より効率は悪いが，１つづつ新しい順列を生成する点が，以下で述べるようなプログラムの構成の中で使うには便利である．生成する順序は，辞書式順序の昇順である（つまり 9 桁の整数と見た時，段々大きくなる）．

プログラムの基本構造を，次のように考えた．

---

順列を初期化する．
以下を繰り返す．
　有理数式に変換・評価．
　=1　なら結果を書き出して，繰り返しをぬける．
　次の順列を生成．

この構造は素直だと思うが，順列生成のモジュールは，一回の呼び出しで新しい順列を一つだけ生成するというものを想定している．もし，次々と順列を生成するようなモジュールを想定するなら（つまり野下さんの本にあるアルゴリズムをそのまま素直に実現したプログラムを想定するなら），コルーチンにするか，逆にそちらを上位へ持ってきて，新しい順列が得られる度に有理数式を評価し検査することになろう．

次は，基本となるデータ構造である．といっても，問題となるのは１から９までの数からなる順列をどう表現するか，という点だけである．このデータ構造は，順列を生成する部分で変更され，有理数式を計算する部分で参照される．単純に大きさ９の１次元配列にすればよいだろう．

## 3　プログラム

ここまで方針が決まれば，プログラムにするのは単純作業である．実際，１時間ぐらいで作成し結果もでた．そんな簡単なプログラムでも，自分の書いたものを人前にみせるのは恥ずかしいという気持ちがある．しかし，せっかくの機会だから，恥をさらして諸兄姉のご批判を仰ぐとしよう．

まず，1–9 の順列をしまう配列を digits という名前とし，その大きさ N とともに大域変数とする．配列の初期値を与えるのは，後の便宜のため関数にしておく．

```
(defvar digits (make-array 9))
(defvar N 9)
(defun init-digits ()
  (setq digits #(1 2 3 4 5 6 7 8 9)))
```

プログラムの全体の構造は，すでに示した通りで，それを関数にしたのが次である．

```
(defun find-solution ()
  (init-digits)
  (dotimes (i (factorial 9))
           (when (= (add-three-rationals) 1)
                 (print-result)
                 (return))
           (next-permutation)))
```

ここで繰り返しの回数を 9! までとしているのは，ほとんど無限ループといっているようなものだが，プログラムに虫があって本当に無限ループに入っ

てしまったり，万一問題の不備で答がなかったりした場合の，最後の歯止めのつもりである．階乗の計算は，教科書の再帰関数のところには必ず出てくるが，この場合，そのような教科書的なプログラムでいいだろう．たとえば，

```
(defun factorial (n)
  (if (zerop n) 1 (* n (factorial (1- n)))))
```

これをあえて逐次型に書き直そうとすると，案外間違えたりする．むしろ，最近のコンパイラでは自動的に最適化してくれるから，任せた方がよい．

問題の式を計算するところは，Common Lisp では有理数計算を勝手にやってくれるので，楽である．□/□□ の計算は，次のようにすればよい．

```
(defun a-by-bc (a b c)
  (/ a (+ (* 10 b) c)))
```

これを使って 3 つの有理数の和を求めるところは，

```
(defun add-three-rationals ()
  (let ((sum 0))
    (do ((i 0 (+ 3 i))) ((> i 6) sum)
        (incf sum (a-by-bc (aref digits (+ i 2))
                           (aref digits (+ i 0))
                           (aref digits (+ i 1)))))))
```

となる．ここで $d_0/d_1d_2$ でなく，わざわざ $d_2/d_0d_1$ を計算するようにしているのは，初期値として (1 2 3 4 5 6 7 8 9) を与えているが，すでに述べた理由により，1 は分母の上位の桁に来る可能性が高いので，探索でそのケースを優先するようにしたものである．順列が辞書式昇順に出てくるので，1 の位置は一番最後に動く．これ以外には，探索を高速化するための工夫は何もしていない．

結果の出力は，単純である．

```
(defun print-result ()
  (format t "~d/~d~d + ~d/~d~d + ~d/~d~d = 1"
          (aref digits 2) (aref digits 0) (aref digits 1)
          (aref digits 5) (aref digits 3) (aref digits 4)
          (aref digits 8) (aref digits 6) (aref digits 7)))
```

さて，順列生成は参照した Pascal プログラムを単純に焼き直した．そして，テストしたらエラーが出た．今回のプログラムで出たほとんど唯一のエラーである．Common Lisp では C と同じように，配列の添字は 0 から始まることになっている．Pascal では，添字の範囲は任意にとれるが，元のプログラムでは 1 から $n$ までになっていた．それをうっかりそのまま引き移して，エ

ラーを出したものである．それを手直しするため，新たに N1 という大域変数を定義しておく．

```
(defvar N1 (1- N))
```

これを用いて，順列生成のプログラムは次のようになる．

```
(defun next-permutation ()
  (let ((i (1- N1)) (j N1))
    (loop (when (< (aref digits i) (aref digits (1+ i)))
                (return))
          (decf i))
    (loop (when (> (aref digits j) (aref digits i))
                (return))
          (decf j))
    (swap i j)
    (incf i) (setq j N1)
    (loop (when (>= i j) (return))
          (swap i j)
          (incf i) (decf j))))

(defun swap (i j)
  (let ((x (aref digits j)))
    (setf (aref digits j) (aref digits i))
    (setf (aref digits i) x)))
```

これですべてである．

## 4 結果

実行してみた．待つことしばし．次のような答が出た．

```
>(find-solution)
9/12 + 5/34 + 7/68 = 1
NIL
```

これには，うなってしまった．確かに 5 と 7 がポイントだったが，1 つの分数の中だけで比較的簡単な形に約分することしか考えなかったのが，手落ちである．それにしても，分母に 17 を素因数として持つものを持ってくる可能性など，考えもしなかった．

この他に解はないのだろうか．よくできた問題であると信じると，多分この他には解がないのだろう（項の入れ換えという自明の別解は考えないとし

て）．もちろん，このプログラムを解が見つかったところで止めないで，最後までループを回せばいいわけだが，そこまでやる気がしなかった．

なお，使用計算機は今や旧式の SUN3/60．処理系は KCL．コンパイルして，解が見つかるまでの時間を測ったところ，次のようであった．

```
>(time (find-solution))
9/12 + 5/34 + 7/68 = 1
real time : 119.283 secs
run time  : 102.767 secs
NIL
```

## 参考文献


[1] 土居範久：PASCAL 入門，培風館，1985.

[2] 野下浩平，高岡忠雄，町田元：基本的算法（岩波講座 情報科学 10），岩波書店，1983.

[3] 湯浅太一，萩谷昌己：Common Lisp 入門，岩波書店，1986.


[**付記**] 原稿を送った後で，このプログラムの欠陥に気づいた．欠陥といってもプログラムの仕様をきちんと示していないので，このプログラムの動作にあった仕様を想定すれば，問題はないともいえる．仕様が，

> 「式 $\frac{\square}{\square\square}+\frac{\square}{\square\square}+\frac{\square}{\square\square}=1$ の □ に，1 ～ 9 の自然数を重複なく入れて，等式を成り立たせるような組合せがあれば，そのひとつを出力する.」

というものだとすると，解がない場合については何も言っていないから，このプログラムで構わないだろう．

しかし，本文中で少し余計なことを書いた．いわく,「ここで繰り返しの回数を 9! までとしているのは, … プログラムに虫があって本当に無限ループに入ってしまったり，万一問題の不備で答がなかったりした場合の，最後の歯止めのつもりである.」 あるいは,「もちろん，このプログラムを解が見つかったところで止めないで，最後までループを回せばいいわけだが, … 」これがまずい．

もし繰り返しを途中で脱けず，最後まで，つまり 9! まで回ったとしよう．最後の順列は (9 8 7 6 5 4 3 2 1) で，これに対し `add-three-rationals` を計算し，結果が 1 になるかどうか判定する．そこまではよい．その後，この状態で `next-permutation` を実行する．ところが，土居さんの教科書にあっ

たプログラムを機械的に移植したこの手続きでは，このような辞書式順序で最大値になるような順列の次を求めることは想定していない．無理に実行すると，内部変数の i が $-1$ となって，配列の添字の範囲から外れてしまう．

かといって，繰り返しを $9! - 1$ まで実行するのでは，$(9\ 8 \ldots 1)$ のケースを確かめないことになる．だから，繰り返しの終了の判定を頭でやらずに，next-permutation を呼ぶ前で行なうようにするか，next-permutation の仕様を変えて，辞書式順序の最大値が来ても，エラーにならないようにするしかないだろう．

たとえば，辞書式順序の最大値が来たら初期値（辞書式の最小値）に戻るようにするに変更すると，

```
(defun next-permutation ()
  (let ((i (1- N1)) (j N1))
    (loop (when (or (minusp i)
                    (< (aref digits i) (aref digits (1+ i))))
                (return))
          (decf i))
    (when (minusp i)
          (init-digits) (return-from next-permutation))
    (loop (when (> (aref digits j) (aref digits i))
                (return))
          (decf j))
    (swap i j)
    (incf i) (setq j N1)
    (loop (when (>= i j) (return))
          (swap i j)
          (incf i) (decf j))))
```

この他にも，きっとおかしいところがあるだろう．プログラムを実際に読んで下さる方がどのくらいいるか分からないが，なんでもご指摘いただけるとありがたい．