7. 順列の生成法†

山崎　秀記††

1. まえがき

本稿では n 個の要素からなる**順列**すべてを効率よく生成する手続きを紹介する．これ自身興味ある問題のようで，1812年にはすでに，すべての順列を辞書式順序で生成する手続きが発表されている[1]．問題そのものが単純で，解くことはそう易しくないので先人の興味を引いたのだろう．またその意味でプログラミングの良い練習問題でもある．

さて実際の場面であらゆる場合をしらみつぶしに調べてみたいときに，n 次の順列をすべて生成する手続きがほしくなることがある．しかし結論を言ってしまえば，それは n が大きいときはさけた方がよい．n 次の順列は全部で $n!$ 個あり，どう工夫しても $n!$ の手間がかかる．

10!＝　　362万8800
11!＝　　3991万6800
12!＝　4億7900万1600
13!＝62億2702万0800

上の表からわかるように，すべての順列が生成できるのはたかだか $n=12$ くらいまでである．しかも実際の応用問題では，順列生成にかかる手間より，生成された順列の処理にかかる手間の方がはるかに大きいのが普通である．だとすればどの生成方法を用いるかは余り差がない．

結局，ある問題についてしらみつぶしに調べたいとき，すべての順列をいかに効率よく生成するかという問題に精力をそそぐよりも，どうしたらすべての順列を生成しなくてもすむか，しらみつぶしの探査をどう避けるか，に精力をそそぐべきである．

それでも，どうしてもすべての順列を生成する必要があるときは本稿が参考になるだろう．またそれ以上にしらみつぶしの探査を避けるのにも役立つ．

2	9	4
7	5	3
6	1	8

図-1

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

図-2

例えば3次の魔法陣（例，図-1）をすべて求めるとき，1から9までの数からなる順列 $P_1P_2\cdots P_9$ をすべて生成し，それぞれについて，図-2 が魔法陣になるかどうか調べる，としよう．このとき，辞書式順序（順列を9桁の数とみて小さい順で順列を生成する方法をとれば，順列の生成が大幅に省略できる．まず，123456789 という順列から調べるわけだが，123… となる順列は $P_1+P_2+P_3=6\neq 15$ なので P_4 から P_9 をどう入れ替えても魔法陣にならない．したがって，順列 123987654 まで 6! 個の順列の生成が省略できる．つまりこの問題については $P_{i+1}\cdots P_n$ があらゆる順序で入れ換わった後初めて P_i が換わるような順列生成法が適しているといえる．

こんどは次のような問題を考えてみよう．A本社とそのa営業所，B本社とそのb営業所，他にC社，D社の6カ所を今日中に廻りたい．営業所へ着く前に必ずその本社に寄るとして最も速く訪れる順序を求める．この場合，Aとa，Bとbの相対的順序が，この順序になっていない順列は調べる必要がない．したがってこの問題に適している順列生成法は，各 i について $1,2,\cdots,i$ の相対的位置が不変のまま，$i+1,\cdots,n$ があらゆる位置に現れ，その後で初めて i の位置が変わる，といった手続きである．

最も速い順列生成の方法というのは実用上さして重要ではないにしても，いろいろな順列生成の手法を知ることは意味があるし，それ自身興味を引く問題である．本稿では2章で順列中の2要素の交換によって次次と新しい順列を生成していく方法を，3章では巡回置換による方法を，4章では辞書式順序の順成生成法を解説する．

† Algorithms for Generating Permutations by Hideki YAMASAKI (Tokyo Institute of Technology Department of Information Science).

†† 東京工業大学理学部情報科学科

ここで以下の章の記述を簡単にするためにいくつかの記法を定めよう．n次の順列は大きさnの配列Pで表わす．要素は1からnまでとし最初は常に $P[1]=1,\cdots,P[n]=n$ である．（要素が1からnまでの数であるという仮定は手続き T4 以外では本質的でない．）配列 P の一部を参照したいことがよくある．そこで $P[i],\cdots,P[j]$ $(1\leq i<j\leq n)$ を $P[i..j]$ で表わす．したがって $P[1..n]$ は配列 P 全体を表わす．

2章では $P[i]$ と $P[j]$ の**交換**が基本操作となる．これを $P[i\leftrightarrow j]$ と書く．例えば $P[1..5]=43125$ に対し，$P[2\leftrightarrow 4]$ を実行した結果は $P[1..5]=42135$ である．**巡回置換**は $P[\overrightarrow{i..j}]$ で表わす．これは $P[i]$ に $P[i+1]$ を，$P[i+1]$ に $P[i+2]$ を，…，$P[j-1]$ に $P[j]$ を，そして，$P[j]$ には $P[i]$ を代入することを意味する．例えば，$P[1..5]=43125$ に，$P[\overrightarrow{2..4}]$ を実行した結果は $P[1..5]=41235$ である．$P[i..j]$ を**逆順**に並べかえる操作は $P[\overleftarrow{i..j}]$ で表わす．$P[1..5]=43125$ のとき $P[\overleftarrow{2..5}]$ を実行すると，$P[1..5]=45213$ となる．

以下の章で与える手続きでは，順列生成手続きを“主プログラム”と考えて，順列を生成するたびにその順列を処理する手続き process を呼ぶようにしてある．

なお，この解説は Sedgewick[6] を主に参考にして書いた．これは順列生成の秀れた概説であり，詳しい参考文献がついている．より詳しく知りたい読者は，これを手掛りにすることをお勧めする．

2. 交換による方法

ある順列から新しい順列をえる手近な方法は，順列の2個の要素を交換することだろう．この章では交換によって，n次の順列すべてを次々に生成する手続きを紹介する．

2.1 再帰的方法

$P[1..n]$ の順列をすべて生成するには，まず $P[1..n-1]$ の順列をすべて生成し，次に $P[n]$ と $P[1..n-1]$ のどれかひとつとを交換し，再び $P[1..n-1]$ のすべての順列を生成する，ということをくり返せばよい．$P[n]$ に $1,\cdots,n$ がすべて1度ずつ現れるようにすれば，$p[1..n]$ のすべての順列を生成できる．具体的に，$P[n]$ をどれと交換すればよいかという問題はさておいて（後で見るように何通りも解がある），この考えに沿った手続きを書いてみよう．考え方が再帰的なので，再帰呼び出しを用いても手続きを書けるが，ここでは他の手続きとの比較上，再帰呼び出しを用いないで書く．配列 $c[i]$ で $P[i]$ と $P[1..i-1]$ のどれかとの交換の回数（正しくは回数+1）を表わす．$c[i]<i$ なる最小の i (≥ 2) を見つけて，$P[i]$ での交換を行い，このとき $c[1..i-1]$ をそれぞれ1に置き直せば目的の手続きがえられる．

```
procedure T1;
  process;
  for i:=2 to n do c[i]:=1 od; i:=2;
  repeat if c[i]<i
           then P[i↔index]; process;
                c[i]:=c[i]+1; i:=2
           else c[i]:=1; i:=i+1 fi
  until i>N.
```

さて $P[i]$ の $c[i]$ 回目の交換の相手の指標 index を求める問題が残っている．表として覚えておくこともできるが，Well のアルゴリズム[8] では

index:=**if** (i は偶数) **then** $i-c[i]$ **else** $i-1$,

Heap のアルゴリズム[2] では

index:=**if** (i は偶数) **then** $c[i]$ **else** 1,

である．Heap のアルゴリズムの計算例を示そう．($n=4$) $\dot{1}\dot{2}\dot{3}4$. $\dot{2}\dot{1}\dot{3}4$, $\dot{3}\dot{1}\dot{2}4$, $1\dot{3}\dot{2}\dot{4}$, $2\dot{3}\dot{1}\dot{4}$, $3\dot{2}\dot{1}\dot{4}$, $4\dot{2}\dot{1}\dot{3}$, $2\dot{4}\dot{1}\dot{3}$, $1\dot{4}\dot{2}\dot{3}$, $4\dot{1}\dot{2}\dot{3}$, $2\dot{1}43$, $1\dot{2}43$, $1\dot{3}\dot{4}2$, $3\dot{1}\dot{4}2$, $4\dot{1}\dot{3}2$, $1\dot{4}\dot{3}2$, $3\dot{4}\dot{1}2$, $4\dot{3}\dot{1}2$, $4\dot{3}21$. $3\dot{4}21$, $2\dot{4}31$, $4\dot{2}\dot{3}1$, $3\dot{2}41$, 2341.（・印は次の順列を生成するための交換の場所を表わす．）

これらの手続きの特徴は，$P[1..i-1]$の順列をすべて生成してからはじめて $P[i]$ を動かすことである．

2.2 繰り返し的方法

$1,\cdots,n-1$ の各順列に対して，その要素の間（両端を含めてnカ所ある）にnを順に入れていくと $1,\cdots,n$ の順列がすべてえられる．この考え方に従って，Johnson[3] と Trotter[7] は独立に 1962年に，隣接要素の交換 $n!-1$ 回で n 次の順列をすべて生成する手続きを示した．

配列 $c[i]$ で要素 i が何回動いたかを覚えておけば，$c[i]<i$ なる最大の i が次に移動する要素である．したがって手続きの枠組みは次のようになる．

```
process;
for i:=2 to n do c[i]:=1 od; i:=n
repeat if c[i]<i
         then P[index↔index+1]; process;
              c[i]:=c[i]+1; i:=n;
         else c[i]:=1; i:=i-1 fi
```

until $i<2$.

ここで要素 i は P[index] か P[index+1] に入っている．index の求め方を調べるために，どんな列を生成したいのか眺めてみよう．

$\dot{1}\dot{2}\dot{3}\dot{4}$, $\dot{1}\dot{2}\dot{4}\dot{3}$, $\dot{1}\dot{4}\dot{2}\dot{3}$, $\dot{4}\dot{1}\dot{2}\dot{3}$ (4が左へ)

$\dot{4}\dot{1}\dot{3}\dot{2}$, $\dot{1}\dot{4}\dot{3}\dot{2}$, $\dot{1}\dot{3}\dot{4}\dot{2}$, $\dot{1}\dot{3}\dot{2}\dot{4}$,

(次に3が1つ左へ動き，4は今度は右へ移動する)

$\dot{3}\dot{1}\dot{2}\dot{4}$, $\dot{3}\dot{1}\dot{4}\dot{2}$, $\dot{3}\dot{4}\dot{1}\dot{2}$, $\dot{4}\dot{3}\dot{1}\dot{2}$

(3も 1,2,3 の中では左端に来たので，2が左へ.)

$\dot{4}\dot{3}\dot{2}\dot{1}$, $\dot{3}\dot{4}\dot{2}\dot{1}$, $\dot{3}\dot{2}\dot{4}\dot{1}$, $\dot{3}\dot{2}\dot{1}\dot{4}$

(3が今度は右へ)

$\dot{2}\dot{3}\dot{1}\dot{4}$, $\dot{2}\dot{3}\dot{4}\dot{1}$, $\dot{2}\dot{4}\dot{3}\dot{1}$, $\dot{4}\dot{2}\dot{3}\dot{1}$

$\dot{4}\dot{2}\dot{1}\dot{3}$, $\dot{2}\dot{4}\dot{1}\dot{3}$, $\dot{2}\dot{1}\dot{4}\dot{3}$, 2134

(4,3,2 すべて端に来ている．1を動かす番になったので終り.)

隣接要素の交換だけですますために，各要素が左へ動いたり右へ動いたりすることに注意しよう．これを論理型の配列 $d[i]$ で，左なら真，右なら偽，と覚えておく．さて問題は，要素 i は $c[i]$ 回目の移動（交換）のときどの位置にいるか，である．1から i までの中では，左へ動いているときは $i+1-c[i]$ 番目にいて，右へ動いているときは $c[i]$ 番目に居る．全体の中では，$i+1$ から n までの中でそのとき左端に来ている要素（即ち $d[j]$ が真から偽に変った j）の数だけ右へずらさなければならない．このずれを x で現し，次の手続きがえられる．

procedure T2; (Johnson[3] Trotter[7])

```
process;
for i:=2 to n do c[i]:=1; d[i]:=true od; i:=n;
repeat if c[i]<i
          then if d[i] then index:=i-c[i]+x
                       else index:=c[i]+x fi;
               P[index↔index+1]; process;
               c[i]:=c[i]+1; i:=n; x:=0
          else if d[i] then x:=x+1 fi;
               d[i]:=not d[i];
               c[i]:=1; i:=i-1 fi
until i<2.
```

この方法の特徴は，1から $i-1$ までの相対的順序が不変のまま，i から n までがあらゆる位置を動くことである．

3. 巡回置換による方法

この章では基本演算として $P[1..i]$ の巡回置換 $\overleftarrow{P[1..i]}$ を用いる．巡回置換を用いる方法にも再帰的方法と繰り返し的方法とがあるがここでは繰り返し的方法を述べよう．

procedure T3;

```
process;
for i:=2 to n do c[i]:=1 od; i:=n;
repeat P←[1..i];
       if c[i]<i
         then c[i]:=c[i]+1; i:=n; process
         else c[i]:=1; i:=i-1 fi
until i<2
```

2章の手続きとは異なり，巡回置換を施す位置を求める手間は要らない．しかし巡回置換自身の手間がかかることと（$\overleftarrow{P[1..i]}$ は i に比例する手間がかかる），$\overleftarrow{P[1..i]}$ を $(i-1)$ 回施したあと，もう一度 $\overleftarrow{P[1..i]}$ を施して順列を元の状態に戻している（手続き中の $\overleftarrow{P[1..i]}$ の位置に注意）ことから，このアルゴリズムは決して効率のよいものではない．にもかかわらずここで紹介した理由は次のような単純な手続きがえられるからである．

配列 $c[i]$ の役目は，$\overleftarrow{P[1..i]}$ を何回行ったかを覚えておくことである．$\overleftarrow{P[1..i]}$ を実行するのは $c[i]<i$ なる最大の数のときである．このときまでに手続きは $\overleftarrow{P[1..n]}$ を n 回実行して $P[n]=n$, $\overleftarrow{P[1..n-1]}$ を $n-1$ 回実行して $P[n-1]=n-1,\cdots,\overleftarrow{P[1..i+1]}$ を $i+1$ 回実行して，$P[i+1]=i+1$, である．しかし $\overleftarrow{P[1..i]}$ はまだ i 回実行されていないので $P[i]<i$ である．即ちこの手続きは $P[i]<i$ なる最大の i に対して $\overleftarrow{P[1..i]}$ を実行する．配列 $c[i]$ は必要ない．

procedure T4; (Langdon[4])

```
process;
i:=n;
repeat P←[1..i];
       if P[1..i]<i then i:=n; process
                    else i:=i-1 fi
until i<2.
```

この手続きは見かけ上最も単純なものだろう．

4. 辞書式順序の生成

この章ではすべての順列を辞書式順序で生成する手続きを示す．再帰的に考えてみよう．$P[1..n]$ のすべての順列を辞書式順序で生成するには，まず $P[2..n]$ を辞書式順序で生成する．その結果 $P[2..n]$ は

大きい方から逆に並んでいるから，$P[1]$ と $P[n]$ を交換し，$P[2..n]$ を逆に並べかえる．再び $P[2..n]$ の順列全部を辞書式順序で生成するとこんどは，$P[1]=2, P[n-1]=3, P[n]=1$ となるから $P[1]$ と $P[n-1]$ を交換し $P[2..n]$ を逆に並べかえる．こうしていけば $P[1]$ に $1, 2, \cdots, n$ がこの順で現れ，$P[1..n]$ の順列を辞書式順序ですべて生成できる．基本方針はこれでよいのだが，次の手続きでは，配列の添字の計算と簡単にするために，順列を配列 $P[1..n]$ ではなく，$P[1..n]=Q[n..1]$ なる配列 $Q[1..n]$ を用いて表わす．

procedure T5; (Ord-Smith[5])

```
process;
for i:=2 to n do c[i]:=1 od; i:=2
repeat if c[i]<i
          then Q[i↔c[i]]; Q[←1..i−1→];
               process;
               c[i]:=c[i]+1; i:=2
          else c[i]:=1; i:=i+1 fi
until i>n.
```

$Q[n..1]=(1, 2, \cdots, n)$ で始めれば，$Q[n..1]$ に1, 2, $\cdots$, n のすべての順列が辞書式順序で現れる．もしもとの配列 $P[1..n]$ で計算したければ，

$Q[i\leftrightarrow c[i]]$ を $P[n+1-i\leftrightarrow n+1-c[i]]$ に，

$Q[\overleftarrow{1..i-1}]$ を $P[\overleftarrow{n+2-i..n}]$ に，

置きかえればよい．

配列 $P[1..n]$ に戻って議論を進めよう．上の手続きで命令 $P[n+1-i\leftrightarrow n+1-c[i]]$ が行われるときの $(n+1-i)$ の値は，$P[j]<P[j+1]>P[j+2]>\cdots>P[n]$ なる j の値に等しく，$P[j]$ の交換の相手は $P[j..n]$ 中の $P[j]$ の次に小さい要素である．このことからある順列 $P[1..n]$ が与えられたとき，それを辞書式順序で次にくる順列に変換するには次のようにすればよいことがわかる．

procedure T6;

```
j:=n−1;
while j<1 or P[j]>P[j+1] do j:=j−1 od;
if j>1 then
  k:=n;
  while P[j]>P[k] do k:=k−1 od;
  P[j↔k]; P[←j+1..n→]
  fi
```

この手続きを最初から順列全体が逆順になるまで繰り返せばすべての順列が生成できる．それが Fischer-Krause[1] の方法である．

参考文献

1) Fischer, L. L. and Krause, K. C.: Lehrbuch der Combinationslehre und der Arithmetik, Dresden (1812).
2) Heap, B. R.: Permutations by interchanges, Comput. J., Vol. 18, No. 1, pp. 293-294 (1963).
3) Johnson, S. M.: Generation of Permutations by adjacent transposition, Math. Comput., Vol. 17, pp. 282-285 (1963).
4) Longdon, G. G., Jr.: An algorithm for generating permutations, Comm. ACM, Vol. 10, No. 5, pp. 298-299 (1967).
5) Ord-Smith, R. J.: Generation of Permutations in Lexicographic Order, Comm. ACM, Vol. 11, No. 2, p. 117 (1968).
6) Sedgewick, R.: Permutation Generation Methods, Comput. Surv., Vol. 9, No. 2, pp. 137-155 (1977).
有澤 誠訳：順列生成の手法，bit, Vol. 10, No. 6, pp. 69-97 (1978).
7) Trotter, H. F.: Perm, Comm. ACM, Vol. 5, No. 8, pp. 434-435 (1962).
8) Wells, M. B.: Generations of Permutations by transposition, Math. Comput., Vol. 15, pp. 192-195 (1961).

(昭和57年12月1日受付)