論　考

順列生成アルゴリズムによる
魔方陣の総当たり探索

Round robin search of magic square
by permutation generation algorithm

Author　: 志多　友史　(Yuji Shida)
Date　: 2017/4/12
Keywords　: 順列生成(permutation generation), 辞書順(lexical order), 総当たり(round robin),
非再帰関数(non recursive function), 魔方陣(magic square),
C 言語(C programming language)

Abstract:==========

本稿では順列生成アルゴリズムを作成し、その応用例として魔方陣の総当たり探索を行った。順列生成アルゴリズムの作成については、数々のウェブページで紹介されているが、そのほとんどが再帰関数によって作られている。本稿では再帰関数を用いずに順列の生成を行う方法を考える。

In this report, I made permutation generation algorithm and searched the magic square by round robin method as the applied example. The making of the permutation generation algorithm is introduced in many web pages, but the most are made by recursive function. Therefore, I tried to make the permutation generation algorithm without recursive function.

==========

１．序論(Introduction)

順列の生成アルゴリズムには再帰関数を用いた方法が頻繁に用いられているが、本稿ではあえて非再帰関数でこれを実現する。その後、この順列生成アルゴリズムの応用例として魔方陣の総当たり探索を行う。

２．理論(Theory)

本稿のプログラムを構成する「順列の生成」と「魔方陣の確認」に関する基礎理論について記す。

２．１．順列の生成

まず、具体例を用いて順列生成の規則性について整理する。例えば 1 から 5 までの正の整数を用いた順列の総数は ${}_5P_5=5!=5*4*\cdots*1=120$ より 120 個である。これを樹形図で表すと次のようになる。

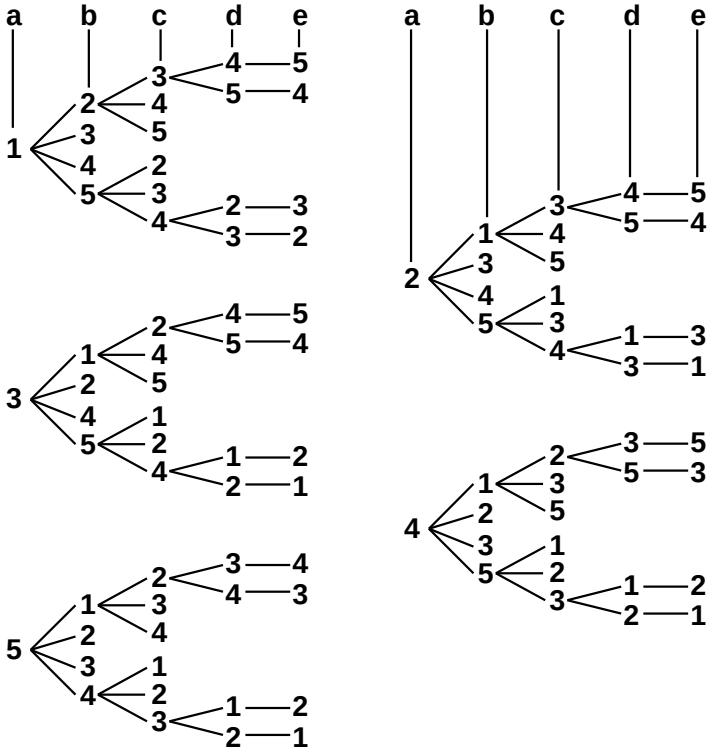

図２．１　順列の樹形図

ここで、最初と最後の順列に着目すると、最初の順列は(a,b,c,d,e)が昇順、最後の順列は降順となっている。これより順列の生成における終了判定は「現状の順列が降順」である事といえる。この判定基準は順列の後半部分について着目する場合にも成り立つ。具体的には(a,b,c,d,e)=(2,4,1,5,3)という順列が与えられた場合、d,e は既に降順であり、次の順列を作る場合は c の値を d,e の内、c より大きいものの中で最小の値としなければならない。よって、次の順列の c の値は 3 となり、以降の d,e を昇順にして(2,4,3,1,5)が得られる。この流れを一般化して考えると次項のようになる。この操作を、順列全体が降順になるまで繰り返すと順列の生成は終了となる。

（１）順列の後ろ側から降順の大小関係が不成立となる部分（節点）を探す。

下図のように現在の順列の後ろ側から降順の関係が不成立となる部分を探す。

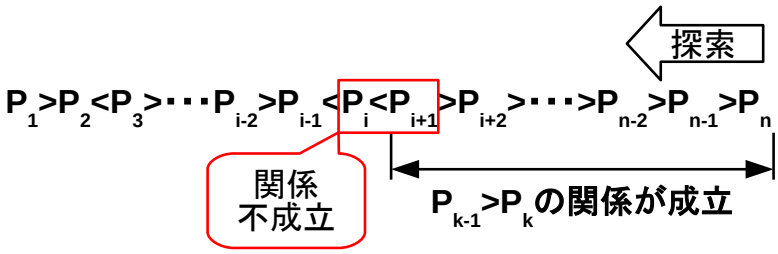

図２．２　節点の探索

（２）新たな節点の数を探す。

ここで、簡単のため次の集合を定義する。

・ $Q=\{P_k \ : \ i+1\leq k\leq n\}$

・ $R=\{P_k \ : \ (\forall P_k>P_i)\cap(\forall P_k\in Q)\}$

集合Qは現在の順列の節点より後ろ側の数の集合で、集合Rは集合Qの内、節点の数 P_i より大きな数の集合である。図２．３に示すように順列生成の都度、同じ位置（階層）の節点の数は、その節点より前の節点が変化しない限り、増加し続けなければならない。

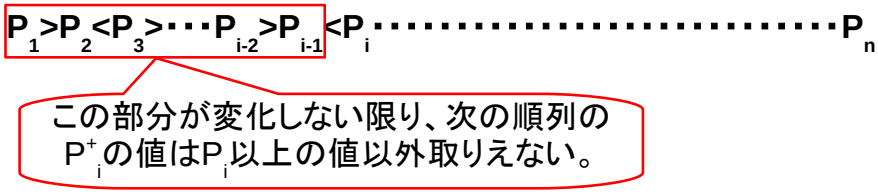

図２．３　次順列の節点の条件

従って、次の順列のi番目の数 P_i^+ は集合Rの最小値となる。ここで集合Rが空集合Φである場合を考える。この場合、集合Qの全ての数が P_i よりも小さいという事となり、$P_i>P_{i+1}$ でなければならないという事になる。これは（１）の"関係不成立 $P_i<P_{i+1}$ "に対する矛盾である。よって集合Rは空集合ではなく、集合Rの最小値 min R は存在する。

（３）順列の後ろ側を昇順に並べなおす。

（２）で次の順列のi番目の数 P_i^+ は min R と定まったので、これ以降の数を並べなおす。並べなおす数の集合は $Q-\{min\,R\}+\{P_i\}$ となり、これを昇順に並べる。

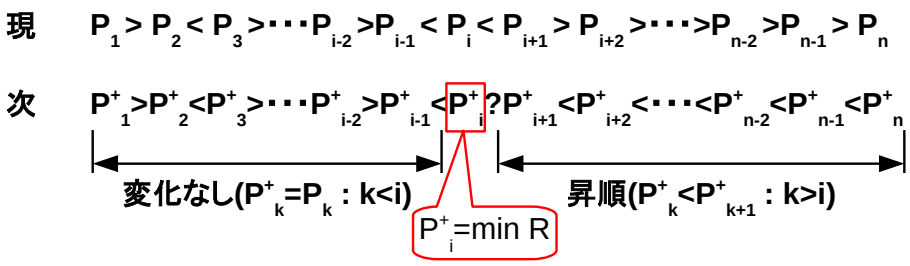

図２．４　次順列の作成

２．２．魔方陣の確認

通常、魔方陣はその形状が正方形で、行・列・斜めの和が全て等しいものをいう。これを総当たりで探索するには、前述の順列生成アルゴリズムによって作られる順列を順番に並べて正方形の行列を作り、これに行・列・斜めの数の和が条件を満たすかどうかの確認を繰り返すプログラムを作成すればよい。

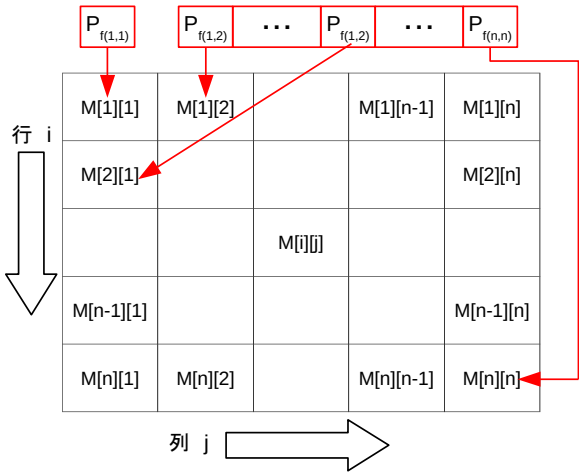

図２．５　順列と魔方陣行列の対応

行列の i 行 j 列目の場所に対応する順列 P の通し番号 f(i,j)は

$$f(i,j)=(i-1)n+j$$

と表されるので、i 行目の和を計算する場合は

$$\sum_{k=1}^{n} M[i][k]=\sum_{k=1}^{n} P_{f(i,k)}=\sum_{k=1}^{n} P_{(i-1)n+k}$$

j 列目の和を計算する場合は

$$\sum_{k=1}^{n} M[k][j]=\sum_{k=1}^{n} P_{f(k,j)}=\sum_{k=1}^{n} P_{(k-1)n+j}$$

となる。また左上(M[1][1])から右下(M[n][n])にかけての和は

$$\sum_{k=1}^{n} M[k][k]=\sum_{k=1}^{n} P_{f(k,k)}=\sum_{k=1}^{n} P_{k(n+1)-n}$$

右上(M[1][n])から左下(M[n][1])にかけての和は

$$\sum_{k=1}^{n} M[k][n+1-k]=\sum_{k=1}^{n} P_{f(k,n+1-k)}=\sum_{k=1}^{n} P_{k(n-1)+1}$$

となる。

３．方法(Method)

前章で示した計算方法をC言語のプログラムで具体的に作り上げる。

３．１．順列の生成

前章２．１に示した手続きの流れを具体的に記述すると次のようになる。なお、この関数は配列の1番目からn番目に設定された重複の無い数の列から次々と順列を生成するものである。

順列生成関数

```
int next_permutation(int fln,int bar[])          ※順列の要素数と配列のアドレスを引数とする。
{
        int i,buf0,num1,num2,num3;
        ※（１）順列の後ろ側から降順の大小関係が不成立となる部分（節点）を探す。
        num1=0;
        for(i=fln-1;i>=1;i--){
                if(bar[i]<bar[i+1]){
                        num1=i;
                        break;
                }
        }
        ※num1 が書換えられなかったという事は、この順列は既に降順になっている。
        if(num1==0) return 0;          ※次順列が存在しない場合は"0"を返却する。
        ※（２）新たな節点の数を探す。
        for(i=fln;i>num1;i--){
                if(bar[num1]<bar[i]){
                        num2=i;
                        break;
                }
        }
        ※現在の節点と次の節点を交換。
        buf0=bar[num1];
        bar[num1]=bar[num2];
        bar[num2]=buf0;
        ※（３）順列の後ろ側を昇順に並べなおす。
        num3=fln-num1;          ※並べなおす要素の数。
        num3=num3/2;          ※整数部のみを取り出す。
        for(i=1;i<=num3;i++){       ※既に降順であるので、前後を反転させて昇順にする。
                buf0=bar[num1+i];
                bar[num1+i]=bar[fln-i+1];
                bar[fln-i+1]=buf0;
        }
        return 1;          ※次順列が存在する場合は"1"を返却する。
}
```

３．２．魔方陣の確認

前章２．２の最後に示した行・列・斜めの和の計算を行う関数を記述すると次のようになる。行・列の関数に関しては 1 行目または 1 列目の和をそれぞれ num1 に格納し、2 行目または 2 列目以降の和を num2 に格納し、num1 と num2 を逐次比較する。全ての各行の和または各列の和が等しい場合はその和を返却し、一致しない場合は"0"を返却する。

行の計算関数

```
int line_calc(int fln,int bar[])
{
	int i,j,k,l,num1,num2;

	num1=0;
	for(i=1;i<=fln;i++) num1+=bar[i];

	for(i=2;i<=fln;i++){
		k=(i-1)*fln+1;
		l=i*fln;
		num2=0;
		for(j=k;j<=l;j++) num2+=bar[j];
		if(num1!=num2) return 0;
	}

	return num1;
}
```

列の計算関数

```
int column_calc(int fln,int bar[])
{
	int i,j,k,num1,num2;

	num1=0;
	j=(fln-1)*fln+1;
	for(i=1;i<=j;i+=fln) num1+=bar[i];

	for(i=2;i<=fln;i++){
		k=(fln-1)*fln+i;
		num2=0;
		for(j=i;j<=k;j+=fln) num2+=bar[j];
		if(num1!=num2) return 0;
	}

	return num1;
}
```

斜めの計算関数

```
int diagonal_calc(int fln,int bar[])
{
	int i,j,num1,num2;

	num1=0;
	for(i=1;i<=fln;i++){
		j=i*(fln+1)-fln;
		num1+=bar[j];
	}

	num2=0;
	for(i=1;i<=fln;i++){
		j=i*(fln-1)+1;
		num2+=bar[j];
	}

	if(num1!=num2) return 0;

	return num1;
}
```

４．結果(Results)

前章で作成したプログラムの実行結果を示す。なお計算環境は次の通りである。

CPU：Intel Core i5-4570 3.20GHz
MEM：DDR3 Dual Channel 32GB(8GB*4) 1333MHz
OS：Windows8 64bit

（１）3×3 の場合

3×3 の魔方陣については一瞬で演算が完了し、以下の 8 つの魔方陣が得られた。

魔方陣　(3×3)

2	7	6
9	5	1
4	3	8

2	9	4
7	5	3
6	1	8

4	3	8
9	5	1
2	7	6

4	9	2
3	5	7
8	1	6

6	1	8
7	5	3
2	9	4

6	7	2
1	5	9
8	3	4

8	1	6
3	5	7
4	9	2

8	3	4
1	5	9
6	7	2

（２）4×4 の場合

4×4 になると 3×3 の場合よりも桁違いに組み合わせの数が増加し、かなりの時間が掛かることから、演算を分割して魔方陣の探索を行った。（※$16! \fallingdotseq 2.09\times10^{13}$,$9! \fallingdotseq 3.63\times10^{5}$）分割方法は次に示す通りである。

プロセス 1：魔方陣の左上の角が 1 の場合の魔方陣の探索（※テストも兼ねて小規模で実施）
プロセス 2：魔方陣の左上の角が 2~6 の場合の魔方陣の探索
プロセス 3：魔方陣の左上の角が 7~11 の場合の魔方陣の探索
プロセス 4：魔方陣の左上の角が 12~16 の場合の魔方陣の探索

最終的に得られた魔方陣の数は 7040 個で、他の資料からも正しい結果が得られた事が分かった。また各プロセスでの演算についてそれぞれの結果を表４．１にまとめる。なお、計算の途中で計算機がスリープモードに入ってしまったため、正確な演算時間は不明である。

表４．１　計算結果

プロセス名	プロセス 1	プロセス 2	プロセス 3	プロセス 4
演算範囲	1（左上角値）	2~6（左上角値）	7~11（左上角値）	12~16（左上角値）
組み合わせ数	1.31×10^{12}	6.54×10^{12}	6.54×10^{12}	6.54×10^{12}
演算時間	18(h)31(m)4(s)	108(h)17(m)19(s)	108(h)10(m)19(s)	114(h)52(m)33(s)
魔方陣数	416	2168	2328	2128
備考	－	※スリープ時間有	※スリープ時間有	※スリープ時間有

プロセス 1 の結果から総演算数の 1/16 に掛かる時間は約 18 時間半であるので、総演算時間は約 296 時間となるはずである。上の表から総演算時間を求めると約 350 時間なので、18 時間程度計算機がスリープ状態になっていたと考えられる。またプロセス 4 の演算時間がプロセス 2,3 よりも長い点については、他の実行プロセスと共に計算資源を使用していたためと考えられる。

魔方陣　(4×4)

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

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

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

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

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

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

５．考察(Discussion)

順列の生成及び魔方陣の確認関数については問題なく作成できたが、魔方陣の探索には莫大な時間が掛かる事が分かった。魔方陣の探索を行う場合は効率的な方法を模索する必要がある。

６．結言(Summary)

個人的に再帰関数があまり好きではないという事で、再帰関数の例としてよく取り上げられる「順列の生成」を非再帰関数で実現してみた。今後何かのテーマで役立つかもしれないので、一応作ってみた。また、これまでは浮動小数点演算の多いものばかり作っていたので、たまには整数演算のみのテーマも扱ってみたかった。

７．文献(References)

［１］：https://ja.wikipedia.org/wiki/%E9%A0%86%E5%88%97
［２］：http://itakanaya9.hatenablog.com/entry/2014/02/17/121428
［３］：http://www.tbasic.org/reference/old/Permutation.html
［４］：「順列生成の開発Ⅰ」(pdf)
［５］：「順列生成の開発Ⅱ」(pdf)
［６］：https://ja.wikipedia.org/wiki/%E9%AD%94%E6%96%B9%E9%99%A3
［７］：http://www.guru.gr.jp/~issei/msqj/4houjin.htm

８．著者(Author)

氏名：志多　友史（工学修士）
略歴：
2011 年：下位国立大学　工学部電気系学科卒業
2013 年：同大学大学院　工学研究科修了
2013 年：研究開発機関へ就職
興味：物理・数学・コンピュータ・電気電子工作

９．備考(Notes)

特になし。