編入学試験問題 (情報編 再帰呼び出し)

山本昌志*

2005 年 7 月 28 日

1 順列

平成 15 年度横浜国立大学工学部編入試験に出題された問題である。

1.1 問題

N 個の要素を持つ文字形の配列 a[] が与えられているとする。このとき、a[0]〜a[N-1] の要素を並べ替えて得られる $N!$ 個の順列を全て印刷するプログラムについて以下の問に答えなさい。

なお、a[]〜a[N-1] に対する全ての順列を生成するという作業は、a[N-1] の要素を a[0]〜a[N-1] のいずれか一つの要素と交換した N 通りの状況に対して、それぞれ、a[0]〜a[N-2] の順序を全て生成することによりなされる点に注意すること。

[**問 1**] 下記プログラムが目標となるように、空欄 A 〜 E を埋めなさい。ただし、下記プログラムにおいて N の値は 3 である。

[**問 2**] 完成したプログラムの出力結果を示しなさい。ただし、複数行の出力が印刷される場合には、各行の時間順序も正しくないといけません。

*独立行政法人　秋田工業高等専門学校　電気情報工学科

```
#include <stdio.h>
#define N (3)

char a[N]={'a', 'b', 'c'};

main()
{
  perm(a, N);
}

perm(char a[], int n)
{
  int i;

  if(n <= 1)
    print_a(a, N);
  else
    for(i=0; i <n; i++){
      swap(a, i, n-1);
      perm(a,  [A] n-1);
      swap(a,  [B] , [C] );
    }
}

swap(char a[], int i, int j)
{
  char c;

  c=a[i];
  a[i]= [D] ;
  a[j]= [E] ;
}

print_a(char a[], int n)
{
  int i;

  printf("%c", a[0]);
  for(i=1; i<n; i++)
    printf(" %c", a[i]);
  printf("\n");
}
```

1.2 解答

このプログラムは、再帰呼び出しを用いて、順列を全て書き出している。プログラム中の関数 perm() の動作は、次のようになっている。

- 第二引数により与えられる値は、順列を求める範囲を示す。例えば、第二引数が n とすると、a[0]〜a[n-1] までの全ての順列を求め表示する。
- ただし、表示は 0〜a[n-1] までは全ての順列を表示するが、a[n]〜a[N-1] は変化せず、元のままである。

このような関数を再帰呼び出しを使って、a[0]〜a[N-1] までの順列を表示すればよい。そのためには、次のようにする。

- a[N-1] の要素を a[0]〜a[N-1] の全ての順列を書き出す。
  - そのためには、a[N-1] の要素を a[0]〜a[N-1] のいずれか一つの要素と交換しする。
  - a[0]〜a[N-2] の全ての順列を書き出す。ここで再帰呼び出しを使う。
  - a[N-1] の要素を戻し、次の要素に進む。

この再帰呼び出しが終了する条件は、呼び出しの第二引数 n が 1 の場合である。この場合は、一通りしかないので、画面に出力する。

ざっと、プログラムは、こんな感じである。以下に解答を示すので、よく理解せよ。

[**問 1**] 以下の通りにすれば、再帰呼び出しを用いた、順列の組み合わせを出力するプログラムになる。

A	n-1
B	n-1
C	i
D	a[j]
E	c

[**問 2**] プログラムの出力は以下の通り。

```
b c a
c b a
c a b
a c b
b a c
a b c
```

2　クイックソート

平成 16 年度横浜国立大学工学部編入試験に出題された問題である。

2.1　問題

N 個の要素を持つ整数型配列 a[] が与えられているとする。このとき、a[0]〜a[N-1] の要素を昇順に整列するプログラムについて以下の問いに答えなさい。

なお、/*1*/の行の割算は整数の割算 (小数点以下切捨て) である。

[**問 1**]　下記プログラムが目標どおり動作するように [A] 〜 [D] を埋めなさい。ただし、下記プログラムにおいて N の値は 8 である。

[**問 2**]　完成したプログラムの出力結果を示しなさい。ただし、複数行の出力が印刷される場合には、各行の時間順序も正しくないといけません。

[**問 3**]　このアルゴリズムでは配列 a[] の要素の比較回数 (/*2*/および/*3*/の関係演算子の実行回数) は、一般にどれくらいと見積もられるか。N を用いた式を用いて簡潔に説明しなさい。

```
#include <stdio.h>
#define N 8

int a[N]={3,1,5,8,2,6,7,4};

void sort(int [], int, int);
void swap(int [], int, int);

void main(void){
  sort(a, 0, N-1);
}


void sort(int a[], int L, int R){
  int i, j, part;

  i = L;
  j = R;

  part = a[(L+R)/2];

  do{
    while(a[i] < part) i++;
    while(part < a[j]) j--;

    if(i <= j){
      swap(a, i, j);
      i++;
      j--;
    }
  }while(i <= j);

  if(L < j)sort(a, [A] , [B] );
  if(i < R)sort(a, [C] , [D] );
}


void swap(int a[], int x, int y){
  int temp;

  temp = a[x];
  a[x] = a[y];
  a[y] = temp;

  printf("(%d, %d)\n", x, y);
}
```

2.2 解答

クイックソートに関する出題である。クイックソートが理解できていれば解けるはずである。問 3 は少し難しく、一度学習していないと解けないだろう。それでは、解答を以下に示す。

[**問 1**] クイックソートのプログラムにするためには、A〜D は以下の通りにする必要がある。

A `L`
B `j`
C `i`
D `R`

[**問 2**] プログラムの出力は以下の通り。

```
(3, 7)
(2, 4)
(3, 3)
(0, 1)
(1, 2)
(5, 5)
```

[**問 3**] `sort()` 関数が最初に呼び出されたときの比較 (`/*2*/`と`/*3*/`) の実行回数は、N あるいは $N+1$ 回である。平均で、$(2N+1)/2$ 回の比較が行われる。そして、再帰呼び出しにより、α 個と β 個の場合の `sort()` が呼び出されることになる。N 個の時のトータルの比較回数を a_N として、これらの関係は、

$$a_N = \frac{2N+1}{2} + a_\alpha + a_\beta \tag{1}$$

となる。もちろん、α と β は、1〜$N-1$ の値を取りうる。そして、`a[]` に格納されている値がランダムであれば、同じ確率で 1〜$N-1$ の値をとる。さらに、α と β の和は N になることはクイックソートのアルゴリズムから自明である。したがって、a_N の期待値は、

$$\begin{aligned} a_N &= \frac{2N+1}{2} + \frac{1}{N-1}\left(a_1 + a_2 + a_3 + \cdots + a_{N-1} + a_{N-1} + a_{N-2} + a_{N-3} + \cdots + a_1\right) \\ &= \frac{2N+1}{2} + \frac{2}{N-1}\sum_{k=1}^{N-1} a_k \end{aligned} \tag{2}$$

となる。この漸化式から、a_N を求めることになるが、計算は結構大変である。まず、式 (2) を

$$(N-1)a_N = \frac{(2N+1)(N-1)}{2} + 2\sum_{k=1}^{N-1} a_k \tag{3}$$

と変形しておく。つぎに、この式の N を $N-1$ にした場合の式

$$(N-2)a_{N-1} = \frac{(2N-1)(N-2)}{2} + 2\sum_{k=1}^{N-2} a_k \tag{4}$$

を利用する。式 (3) から式 (4) の辺々を差し引いて整理すると、

$$(N-1)a_N - (N-2)a_{N-1} = \frac{1}{2}\left[(2N+1)(N-1) - (2N-1)(N-2)\right] + 2a_{N-1} \tag{5}$$

となる。さらに、整理を進めると

$$2(N-1)a_N - 2Na_{N-1} = 4N-3 \tag{6}$$

となる。両辺を、$(N-1)N$ で割ると、

$$\begin{aligned}\frac{2a_N}{N} - \frac{2a_{N-1}}{N-1} &= \frac{4N-3}{(N-1)N} \\ &= \frac{3}{N} + \frac{1}{N-1}\end{aligned} \tag{7}$$

となる。ここで、$2a_N/N$ を b_N と置くと、

$$b_N - b_{N-1} = \frac{3}{N} + \frac{1}{N-1} \tag{8}$$

となり、階差数列を用いて容易に計算できる。すなわち、

$$b_N = b_2 + \sum_{k=3}^{N}\left(\frac{3}{k} + \frac{1}{k-1}\right) \tag{9}$$

である。$a_2 = 2$ より、$b_2 = 2$ となるので、

$$\begin{aligned}b_N &= 2 + 3\sum_{k=3}^{N}\frac{1}{k} + \sum_{k=3}^{N}\frac{1}{k-1} \\ &= 2 + 3\left(\sum_{k=1}^{N}\frac{1}{k} - 1 - \frac{1}{2}\right) + \left(\sum_{k=1}^{N}\frac{1}{k} - 1 - \frac{1}{N}\right) \\ &= -\frac{7}{2} - \frac{1}{N} + 4\sum_{k=1}^{N}\frac{1}{k}\end{aligned} \tag{10}$$

となる。ここで、N が大きい場合、$\sum_{k=1}^{N}\frac{1}{k}$ は $\log_e N + \gamma$ に近づくことが知られている[1]。γ はオイラー数と言われる定数で、その値は $0.577\cdots$ である。したがって、N が大きい場合、

$$b_N \simeq -\frac{7}{2} - \frac{1}{N} + 4(\log_e N + \gamma) \tag{11}$$

となる。右辺の $\log_e N$ は他の項に比べて大きな値を取る[2]。したがって、

$$b_N \simeq 4\log_e N \tag{12}$$

[1]通常は、$1 + \frac{1}{2} + \frac{1}{3} + \cdots - \log_e N = \gamma$ という関係式を憶えている

[2]通常ソートが使われるのは大きなサンプルがあるときである。$N = 10000$ を考えれば、$\log_2 N$ の項が支配的であることはすぐに分かる

となる。b_N の定義から、

$$a_N \simeq 2N \log_e N \tag{13}$$

となる。

a_N は a[] の要素の比較回数を表す。したがって、N が大きい場合の比較回数は、$2N \log_e N$ となる。

3 ハノイの塔

平成 17 年度横浜国立大学工学部編入試験に出題された問題である。

3.1 問題

3 本の棒 (src, dst, work) が立っており、そのうち、左端の棒 (src) に大きさの異なる n 枚の円盤が、大きさの順に差し込まれている。円盤を一枚ずつ、棒から棒へ移動させ、最終的に src にある n 枚の円盤すべてを中心の棒 (dst) に移動させるアルゴリズムを考える。ただし、各円盤は、各棒において、常により小さい円盤がより大きな円盤の上に乗るようにしか移動できず、すべての円盤が移動した結果もまた、大きさの順に積まれていなければならないとする。この制約を満たす移動手順を求めるアルゴリズムは「ハノイの塔」と呼ばれている。

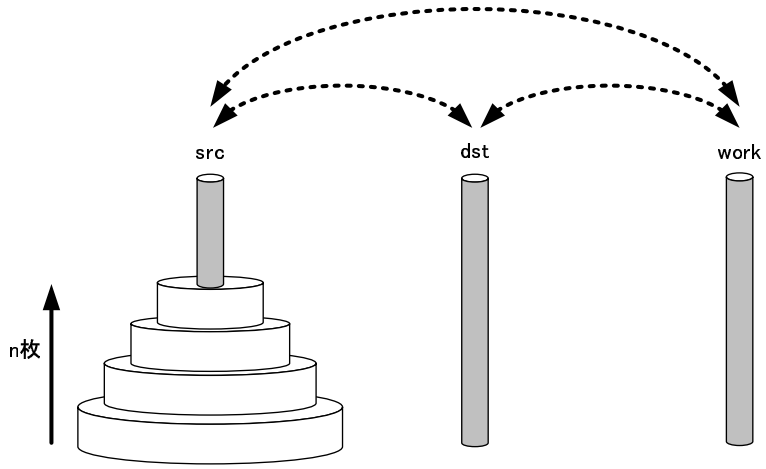

図 1: ハノイの塔

下記のプログラムは、「ハノイの塔」アルゴリズムを C 言語で実現した例である。ここで、例えば"src->work"という出力は、src にある一番上の円盤を、work の一番上に移動させることを意味する。また、任意の n を#define DISK_NUM n として指定している。

リスト 1: ハノイの塔のプログラム

```
1  #include <stdio.h>
2  #define DISK_NUM 3
3
4  void hanoi(int n, char* a, char* b, char* c){
5
6    if(0==n)return;
7
8    hanoi(n-1, a, c, b);
9    printf("%4s -> %4s \n", a, b);
10   hanoi(n-1, c, b, a);
11 }
12
13 int main(){
14   int n = DISK_NUM;
15   hanoi(n, "src", "dst", "work");
16   return 0;
17 }
```

[問 1] 関数 `hanoi()` 中では、`hanoi()` が 2 回呼ばれている。このように、関数の中で自分自身を関数として呼び出すことを「_(a)_呼び出し」という。(a) に当てはまる言葉を答えなさい。

[問 2] このプログラムを実行すると下記の出力が得られた。(b), (c) の行に相当する結果を答えなさい。ただし、空白 (スペース) は明記する必要はない。

```
 src -> dst
 src -> work
_____(b)_____
 src -> dst
work -> src
_____(c)_____
 src -> dst
```

[問 3] `#define disk_num 4` とした場合に、出力される行数は_(d)_行である。(d) に相当する数を答えなさい。

[問 4] 任意の n について、初期状態から dst へ円盤がすべて移動するまでに要する `hanoi()` の呼び出し総数 (e) と、ディスクの移動回数 (f) を、それぞれ n を用いてできるだけ簡潔な式で表しなさい。

3.2 解答

再帰呼び出しを使って、ハノイの塔のパズルを解くためには、以下の手順を踏めばよい。

1. src にある n 枚の円盤のうち、上の n-1 枚を work に移す。
2. src に残っている一番下の円盤を dest に移動させる。

3. 以上で、一番下の処理が終わり、処理の終わっていない n-1 枚の円盤は wark にある。これは、円盤の数が 1 枚減った最初の状態と同じである。したがって、最初から同じことを繰り返す。

問題で与えられたプログラムは、この手順を踏むようになっている。プログラムの内容が理解できれば、以下の解答は分かるはずである。

[**問 1**] 再帰

[**問 2**]
```
dst -> work
work -> dst
```

[**問 3**] 15

[**問 4**] 円盤が n 枚の時の関数 `hanoi()` の呼び出し総数 e_n は、

$$\begin{aligned} e_n &= 1 + e_{n-1} + en - 1 \\ &= 1 + 2e_{n-1} \end{aligned} \tag{14}$$

となる。n 枚の時、1 回呼び出した後、n_1 枚で 2 度呼び出しているからである。この数列の一般項を求めることになるが、そのために、

$$e_n + 1 = 2(e_{n-1} + 1) \tag{15}$$

と変形する。これは等比級数なので、簡単に計算できる。$e_1 = 3$ なので、

$$\begin{aligned} e_n + 1 &= 4 \times 2^{n-1} \\ &= 2^{n+1} \end{aligned} \tag{16}$$

となる。したがって、関数 `hanoi()` の呼び出し総数 e_n は、

$$e_n = 2^{n+1} - 1 \tag{17}$$

となる。

n 枚の時の円盤の移動回数 f_n は、

$$f_n = 1 + f_{n-1} + f_{n-1} \tag{18}$$

となる。関数 `hanoi()` を 1 回呼び出すと、必ず円盤の移動が一回発生する。加えて、$n-1$ 枚の呼び出しを 2 回行っているからである。この式を変形すると、

$$f_n + 1 = 2(f_{n-1} + 1) \tag{19}$$

となる。これは等比数列なので、一般項は簡単に求められる。$f_1 = 1$ なので、

$$f_n + 1 = 2 \times 2^{n-1} \tag{20}$$

となる。したがって、円盤の移動回数 f_n は、

$$f_n = 2^n - 1 \tag{21}$$

と表すことができる。