組み合わせを求めるプログラムについて

谷﨑 久志
神戸大学・経済学部

1994 年 5 月

Abstract: 組み合わせを求めるプログラムはかなり難しい。Harbison and Steele (1987) は C 言語特有のビット演算を用いて，組み合わせプログラムを作成した。本稿では，再帰を利用した組み合わせプログラムを紹介する。ビット演算を利用したものよりも，再帰を用いたものの方が計算速度やプログラムの単純性の面で著しく有利であることが示される。

Key Words: ビット演算，再帰，C 言語

1 はじめに

1 から $n1+n2$ までの $n1+n2$ 個の数字から $n1$ 個を取り出す組み合わせを求めるプログラムを考える。本稿では，2 種類の組み合わせを求めるプログラムを紹介し，さらに，2 つのプログラム (プログラム I, II) の計算時間を比較する。

プログラム I, II はどちらも，$n1$ と $n2$ を与えると，$n1+n2$ までの数字の中から $n1$ 個を取り出して，$n1$ 個の数字を画面に表示されるプログラムとなっている。プログラム I は 2 進数 (C 言語の用語によると，2 進数演算のことをビット演算と言う) を利用したものであり，プログラム II は，再帰を利用して，入れ子の形で直接組み合わせを求めるものである。

例えば，$n1=3$，$n2=3$ の場合を考える。すなわち，1 から 6 までの数字から 3 つを取り出す場合，プログラム I については表 1，プログラム II については表 2 のように，それぞれ出力結果が得られる。

表 1 では選び出される数字 (表 1 の左半分) が，1 によって表される桁 (表 1 の右半分) に対応している。1 行目では，2 進数 000111 が得られ，これは { 1,2,3 } を選ぶことを意味する。また，2 行目では，1 行目の 2 進数の次に大きい 2 進数 (すなわち，001011) を求め，これに対応する選び出される数字は { 1,2,4 } となる。同様に，3 行目では，2 行目の 2 進数の次に大きい 2 進数 (すなわち，001101) を求め，これに対応する選び出される数字は { 1,3,4 } となる。

一方，表 2 では，1 ～ 6 の数字の中で 3 つの数字を直接取り出している。右の列になるほど，また，各列で下になるほど，大きい数字を選び出すプログラムとなる。すなわち，1 行目から 4 行目までは，最初の 2 つの数字を 1,2 と固定したもとで，最後の数字 (3~6) を順番に割り当てるようになっている。

このように，プログラムによって，異なった出力結果が得られる。

2 組み合わせのプログラム I

$n1+n2$ 桁の 2 進数で表された数字の中で，$n1$ 個の 1 が含まれているものを取り出し，それぞれの桁に対応する数字を取り出す方法である。

Harbison and Steele (1987) の方法に従って，組み合わせの生成プログラムを紹介する (奥村 (1991) を参照せよ)。

例えば，$n1+n2=6$，$n1=3$ のとき，$\{3,4,6\}$ の組み合わせは 101100 として表される[1]。6 桁以内で表され，かつ，1 が 3 つ含まれている 2 進数の中で，101100 の次に大きい数字は次のようにして求められる。

[1]取り出される桁が 1 で表される。

表 1: プログラム I の出力結果

出力結果			対応する 2 進数					
1	2	3	0	0	0	1	1	1
1	2	4	0	0	1	0	1	1
1	3	4	0	0	1	1	0	1
2	3	4	0	0	1	1	1	0
1	2	5	0	1	0	0	1	1
1	3	5	0	1	0	1	0	1
2	3	5	0	1	0	1	1	0
1	4	5	0	1	1	0	0	1
2	4	5	0	1	1	0	1	0
3	4	5	0	1	1	1	0	0
1	2	6	1	0	0	0	1	1
1	3	6	1	0	0	1	0	1
2	3	6	1	0	0	1	1	0
1	4	6	1	0	1	0	0	1
2	4	6	1	0	1	0	1	0
3	4	6	1	0	1	1	0	0
1	5	6	1	1	0	0	0	1
2	5	6	1	1	0	0	1	0
3	5	6	1	1	0	1	0	0
4	5	6	1	1	1	0	0	0

表 2: プログラム II の出力結果

出力結果		
1	2	3
1	2	4
1	2	5
1	2	6
1	3	4
1	3	5
1	3	6
1	4	5
1	4	6
1	5	6
2	3	4
2	3	5
2	3	6
2	4	5
2	4	6
2	5	6
3	4	5
3	4	6
3	5	6
4	5	6

1. 101100 の最も右の 1 だけを残して，残りの桁を 0 にしたものを考える。すなわち，000100 である。
2. 元々の数字 101100 にステップ 1 で得られた 000100 を加える。得られた値は 110000 となる。
3. ステップ 2 で得られた数字 110000 の最も右の 1 を残して 0 にする。すなわち，010000 となる。
4. ステップ 3 で得られた数字 010000 をステップ 1 で得られた値 000100 で割る。このとき，000100 となる。
5. ステップ 4 の値 000100 をさらに右に 1 桁ずらす (000010 で割る)。すなわち，000010 である。
6. ステップ 5 の値から 1 を引く。$000010 - 000001 = 000001$
7. ステップ 6 で得られた数字 000001 をステップ 2 で得られた値 (110000) に加えると，次の数字が 110001 として与えられる。この 2 進数で表された数字は，{ 1, 5, 6 } の組み合わせに対応する。

最初の組み合わせは 000111 となり，これは { 1,2,3 } を取り出すことを意味する。ステップ 1 ～ 7 を繰り返すことによって，1 を 3 つ含むすべての 2 進数を求めることができる。

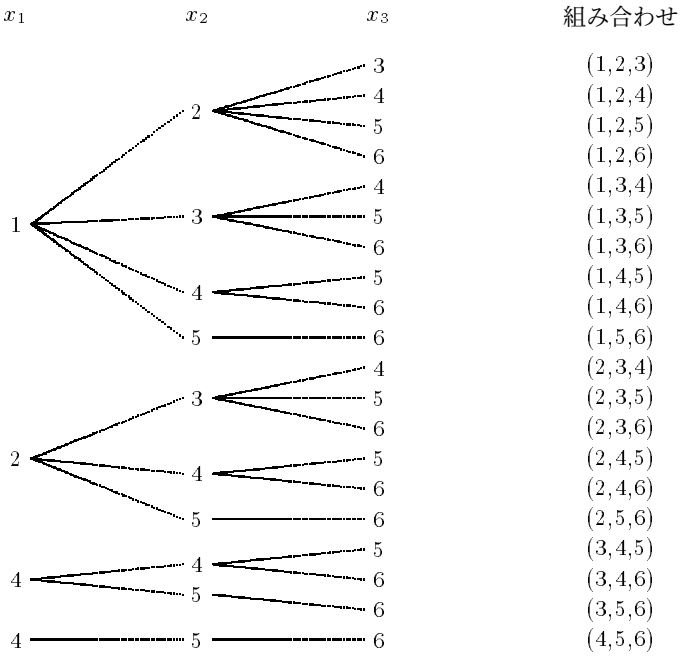
以上の方法は，任意の $n1$，$n2$ について，${}_{n1+n2}C_{n1}$ の組み合わせをすべて生成するのに非常に巧妙な方法である。C 言語では，ビット演算 (2 進数演算) が可能であり，2 進数の計算を直接行うことができる。そのプログラムは以下の通りである。

ステップ 1 ～ 7 で示された数値例との関連は以下の通りである。`x` にある値 (数値例では 101100) を入れる。このとき，`-x` は `x` の各桁を補数にしたものに 1 を加えたものになる。よって，`smallest` にステップ 1 の値 (すなわち，000100)，`ripple` にステップ 2 の値 (すなわち，110000)，`new_smallest` にステップ 3 の値 (すなわち，010000)，`new_smallest/smallest` にステップ 4 の値 (すなわち，000100)，`( new_smallest/smallest ) >> 1` にステップ 5 の値 (すなわち，000010)，`ones` にステップ 6 の値 (すなわち，000001)，`x` にステップ 7 の値 (すなわち，110001) をそれぞれ対応させている。

プログラム I: ビット演算の利用

```
1: #define first(n) ( (unsigned int)( ( 1U<<(n) ) - 1U ) )
2: void comb1(int n1,int n2)
3: {
4:     int i,j,k,m[101];
5:     unsigned int x,s;
6:     unsigned int smallest,ripple,new_smallest,ones;
7:
8:     i=1; x=first(n1);
9:     while( ! (x & ~first(n1+n2)) ) {
10:            s=x; k=1;
11:            for( j=1;j<=n1+n2;j++ ) {
12:                if( s & 1 ) {
13:                    m[k]=j;
14:                    k++;
15:                }
16:                s >>= 1;
17:            }
18:        for(k=1;k<=n1;k++) printf(" %2d",m[k]);
19:        printf("\n");
20:           smallest = x & -x;
21:           ripple = x + smallest;
22:           new_smallest = ripple & -ripple;
23:           ones = ( ( new_smallest/smallest ) >> 1 ) - 1;
24:           x= ripple | ones;
25:        i++;
26:    }
27: }
```

図 1: 樹系図 (6 個の数字の中で 3 個の異なる数字を取り出す方法)



プログラム II: 再帰の利用

```
1: void comb2(int n1,int n2)
2: {
3:     int  m[101];
4:     int  nest,column;
5:     void nest0(int nest,int column,int n1,int n2,int m[]);
6:
7:     nest=0; column=1;
8:     nest0(nest,column,n1,n2,m);
9: }
10: /* ------------------------------------------------------ */
11: void nest0(int nest,int column,int n1,int n2,int m[])
12: {
13:     int  i,k;
14:     void nest0();
15:
16:     for(i=nest+1;i<=n2+column;i++){
17:        m[column]=i;
18:        if( n1 != column )
19:           nest0(i,column+1,n1,n2,m);
20:        else {
21:           for(k=1;k<=n1;k++) printf(" %2d",m[k]);
22:           printf("\n");
23:        }
24:     }
25: }
```

3 組み合わせのプログラム II

プログラム I は 2 進数に基づいているが，プログラム II は $1 \sim n1+n2$ の数字を直接取り出すものである。同じ $n1+n2=6$，$n1=3$ の例，すなわち，1 から 6 までの 6 個の数字の中で 3 個の異なる数字を取り出す方法を考える。樹系図を考えると図 1 の通りになる。取り出される 1 から 6 までの数字を x_1，x_2，x_3 として割り当てる。このとき，$x_1 < x_2 < x_3$ を満たす 6 までの正の整数を求める。例えば，$x_1=1$，$x_2=2$ のとき，$x_3=3,\cdots,6$ の 4 通りの組み合わせとなり，$x_1=1$，$x_2=3$ のとき，$x_3=4,\cdots,6$ の 3 通りの組み合わせとなる。このような考え方に基づいて $n1+n2$ までの正の整数から $n1$ 個を取り出すプログラムを組むと，一般的に $n1$ 個の 繰り返し計算が，入れ子の形になる[2]。

再帰を用いて，このプログラムを作ることができる[3]。この場合の繰り返し回数は ${}_{n1+n2}C_{n1}$ 回となり，プログラム I よりも計算時間の面で改善される[4]。

プログラム II では，関数 `nest0()` から関数 `nest0()` を呼び出すという再帰形を用いる。7 行目で `column` に 1 を初期値として代入する。すなわち，`m[1]` に入る数字を選んでいることになる。関数 `nest0()` を呼ぶ度に `column` を一つづつ増やす。`n1` と `column` が等しくなるまで繰り返される。

このように，図 1 の樹系図に示された通りの考え方をそのままプログラムにしたものがプログラム II である。

---

[2] x_1 の数字を求めるのに $1 \sim 4$ の可能性があり，x_2 は $x_1+1 \sim 5$，x_3 は $x_2+1 \sim 6$，よって，$x_1,\cdots,x_{n1}$ のそれぞれの x_i について 繰り返し計算を必要とする。

[3] 再帰プログラムもまた，ビット演算と同様に，C 言語特有のものである。しかし，最近は再帰の可能な Fortran コンパイラもいくつかある (例えば，WATOCOM F77/386 Ver. 9.01)。本稿では，C 言語を用いてプログラムを組む。

[4] プログラム I では，11 行 $\sim$ 17 行，20 行 $\sim$ 24 行の計算が ${}_{n1+n2}C_{n1}$ 回のすべての組み合わせについて実行されなければならない。そのため，プログラム II より計算時間がかかる。

表 3: $1 \sim n1+n2$ の数字から $n1$ 個を取り出す組み合わせプログラムの計算時間

		プログラム I					プログラム II				
$n1$	$n2$	8	9	10	11	12	8	9	10	11	12
8		.051	.101	.188	.336	.580	.014	.025	.041	.065	.102
9		.102	.212	.417	.783	1.415	.030	.055	.095	.161	.262
10		.193	.422	.873	1.721	3.250	.060	.114	.210	.370	.633
11		.350	.804	1.741	3.590	7.085	.113	.228	.438	.808	1.441
12		.610	1.468	3.329	7.167	14.745	.205	.433	.870	1.680	3.120

1) 表の数字は 400 回の平均時間 (単位は秒) を示す。

2) プログラム I, II の画面出力の部分を削除した場合の計算結果である。すなわち，プログラム I については 18, 19 行を，プログラム II については 21, 22 行を削除する。

3) この表は，CPU 486DX2/66Mz のコンピュータ，WATCOM C/C++32/386 Ver.9.5 の $C++$ コンパイラを使って計算した結果である。

4 各プログラムの計算時間

以上に紹介された組み合わせのプログラム I, II を計算時間の面で比較する。プログラム I, II にメイン・プログラムを加えて比較を行う。その結果は表 3 に示される。

表 3 から，$n1, n2$ が大きくなると計算時間も飛躍的に増大する。一般に，$n1 = n2 = n$ のとき (表 3 の対角要素)，${}_{2n}C_n$ と ${}_{2n-2}C_{n-1}$ との計算量の違いは，$\dfrac{{}_{2n}C_n}{{}_{2n-2}C_{n-1}} = 4 - \dfrac{2}{n}$ であるので，$n = 2$ のとき 3 倍，n が大きくなるにつれて約 4 倍の計算量の増大につながる。また，$n2$ を固定したとき，${}_{n1+n2}C_{n1}$ と ${}_{n1+n2+1}C_{n1+1}$ とは，$\dfrac{{}_{n1+n2+1}C_{n1+1}}{{}_{n1+n2}C_{n1}} = 1 + \dfrac{n2}{n1+1}$ となり，$n1$ に比べて $n2$ が大きくなるにつれて，計算量も増大する。

明らかに，プログラム I よりもプログラム II の方がまとまった形になっている。さらに，表 3 の計算時間の比較によると，プログラム II の方がプログラム I よりもかなり計算時間が短い。 計算時間とプログラムの単純さの両面から判断すると，プログラム II が実用的であるように思われる。

5 おわりに

Harbison and Steele (1987) はビット演算を利用して，組み合わせのプログラムを示した。本稿では，再帰を用いて，組み合わせを求めるプログラムを提示した。そして，さらに 2 つのプログラムの計算時間を比較した。その結果，本稿で示されたプログラムは，Harbison and Steele (1987) のものよりも，煩雑さはなく，しかも計算時間は 3~5 倍程度も短縮され，より実用的であることが明らかにされた。

参考文献

Harbison, S.P. and G.L. Steele Jr. (1987) *C: A Reference Manual* (second edition), Prentice-Hall.

奥村晴彦 (1991) 『C 言語による 最新 アルゴリズム辞典』 (ソフトウエアテクノロジー 13) 技術評論社。