Enumerated Permutations

Kurt Nalty

2011-03-16

Abstract

This note is a collection of algorithms for enumerated permutations. My generic reference is Knuth, The Art of Computer Programming, volume 2, for factorial numbers and swap permutations, mathworld.com for the lexical ordered permutations using inversion vectors, and Hugo Steinman via Martin Gardner and Scientific American for braid ordered permutations.

Mixed Radix Factorial Numbers

A first step in the following algorithims is to convert a number into a mixed radix factorial number representation, where the least significant digit is base 2, the next digit is base 3, the following base 4, and so on. We find the digits using modulo arithmetic and repeated division, as illustrated in the code fragment below.

```
// convert i to a four digit mixed radix number

a = (i % 2); // LSB
b = ((i/2) % 3) ;
c = ((i/6) % 4);
d = ((i/24) % 5);
printf("1234 with %1d%1d%1d becomes ",c,b,a);
```

For larger values, it is no problem to do the calculation in a loop.

Lexical Ordered Permutations

Given the mixed radix digits from above, we start with our most significant digit, and work toward our least significant digit. We take the digit position, increment by one to find the starting point for a right rotation in the target string, and rotate right the number of string digits given by the incremented value of our mixed radix digit. (A value of zero means no change occurs.) We then repeat this process for each digit in succession.

Example : Initial string is 1234. Permutation index is 10, which becomes 0r120 in mixed radix format. We first process the most significant digit (digit 2 with a value of 1). We increment our position for the starting position in our string: our target beginning is bit 3, which has a value of 1. Our string length is 1+1=2. Our first step is the change **12**34 → **21**34. We now process the next mixed radix 2. We begin at digit 3 (=2+1), and rotate 3 digits (being 2+1 again by value at digit 2) and find 2**134** → 2**413**.

Complicating the code just a bit, is the details of string storage in the C programming environment. The string "1234" is stored with 1 at the lowest memory location, and 4 at the highest location. In the code below, I use a character string for the input, and have thus mirrored the indices in the ROR (rotate right) routine.

This program Lexical.c prints out the 24 ordered permutations of 1234, followed by the 120 permutations of 12345.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void ROR(char *array, int size, int bitpos, int count)
{
// I assume a text string format, null terminated
// "1234" is [0]=0x31, [1]=0x32, [2]=0x33, [3]=0x34, [4]=0

    int start, finish, i;
    char storage;

    if (count==0) return;
    start = size - 2 - bitpos;
    finish =  start + count;
    storage = array[finish];
    for (i=finish;i>start;i--)
    {
        array[i] = array[i-1];
    }
    array[start] = storage;
}

int main(void)
{
    int i,j,k;
    int index,a,b,c,d;
    int perm[4];
    int size=4;
    char message[81];

    strcpy(message,"1234");

    printf("\n%s\n\n",message);

    for (i=0;i<24;i++) {
        // calculate factorial modulus numbers
        perm[0] = i%2;        // LSD
        perm[1] = (i/2)%3;
        perm[2] = (i/6)%4;  // MSD
 // create reference string
        strcpy(message,"1234");
        // Apply rotation rights in sequence
        ROR(message, size, 2, perm[2]);
        ROR(message, size, 1, perm[1]);
        ROR(message, size, 0, perm[0]);

        printf("%3d  %d%d%d  %s\n",i,
              perm[2],perm[1],perm[0],message);

    }
    size = 5;
    for (i=0;i<120;i++) {
        // calculate factorial modulus numbers
        perm[0] = i%2;        // LSD
        perm[1] = (i/2)%3;
        perm[2] = (i/6)%4;
        perm[3] = (i/24)%5;  // MSD
 // create reference string
        strcpy(message,"12345");
       // Apply rotation rights in sequence
        ROR(message, size, 3, perm[3]);
        ROR(message, size, 2, perm[2]);
        ROR(message, size, 1, perm[1]);
        ROR(message, size, 0, perm[0]);

        printf("%3d  %d%d%d%d  %s\n",i,perm[3],
                perm[2],perm[1],perm[0],message);

    }
}
```

Swap Permutations

If lexical ordering is not required, we can dispense with the ROR, and simply swap pairs based upon the factorial number digits being interpreted as swap indices.

The program below stdperm.c prints out the swap permutations for 1234, followed by the same for 123.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i,j,k,l;
    int a,b,c,d;
    int source[4]={4,3,2,1};
    int dest[4];
    int holder;
    int raw[24];
    int ray[4][4];

    for (i=0;i<24;i++) {
        a = (i % 2); // LSB
        b = ((i/2) % 3) ;
        c = ((i/6) % 4);
        d = ((i/24) % 5);
        printf("1234 with %1d%1d%1d becomes ",c,b,a);

        dest[0] = 4;
        dest[1] = 3;
        dest[2] = 2;
        dest[3] = 1;

        {
            holder = dest[3];
            dest[3] = dest[3-c];
            dest[3-c] = holder;
        }
        for (j=0;j<4;j++) printf("%1d",dest[3-j]);
        printf("  ");

        {
            holder = dest[2];
            dest[2] = dest[2-b];
            dest[2-b] = holder;
        }
        for (j=0;j<4;j++) printf("%1d",dest[3-j]);
        printf("  ");

         {
            holder = dest[1];
            dest[1] = dest[1-a];
            dest[1-a] = holder;
        }

        for (j=0;j<4;j++) printf("%1d",dest[3-j]);
        printf("\n");

    }

    for (i=0;i<6;i++) {
        a = (i % 2); // LSB
        b = ((i/2) % 3) ;
        printf("123 with %1d%1d becomes ",b,a);

        dest[0] = 3;
        dest[1] = 2;
        dest[2] = 1;


        {
            holder = dest[2];
            dest[2] = dest[2-b];
            dest[2-b] = holder;
        }
        for (j=0;j<3;j++) printf("%1d",dest[2-j]);
        printf("  ");

         {
            holder = dest[1];
            dest[1] = dest[1-a];
            dest[1-a] = holder;
        }

        for (j=0;j<3;j++) printf("%1d",dest[2-j]);
        printf("\n");
    }
}
```

Braid Ordered Permutations

Nice reference: http://www.usna.edu/Users/math/wdj/book/node156.html.

The braid ordered permuatations are very similar to a binary Gray code, where only one pair of symbols swaps at a time during the sequence. This is a natural sequence, preferable to the lexical sequence, in that it easily reveals even/odd permutation membership.

The basic algorithm is illustrated by example below.

```
The initial permutation of a single element is

 1

Duplicate twice

 1
 1

Braid in the new strand (we only get 1/2 cycle)

 1 2
 2 1

For three strands, duplicate each line above three times.

 1 2
 1 2
 1 2

 2 1
 2 1
 2 1

Now fill in the new strand. I start from far right,
moving left, then bounce back right.(1 cycle)
```

```
 123
 132
 312

 321
 231
 213
```

For S4, duplicate each line above four times, then place the bouncing new element. (3 cycles)

```
 1234
 1243
 1423
 4123

 4132
 1432
 1342
 1324

 3124
 3142
 3412
 4312

 4321
 3421
 3241
 3214

 2314
 2341
 2431
 4231
 4213
 2413
 2143
 2134

For S5, duplicate each line above five times,
and place the new element. (12 cycles)

 12345
 12354
 12534
 15234
 51234

 51243
 15243
 12543
 12453
 12435

 14235
 14253
 14523
 15423
 51423

 54123
 45123
 41523
 41253
 41235

 41325
 41352
 41532
 45132
 54132
```

```
51432
15432
14532
14352
14325

13425
13452
13542
15342
51342

51324
15324
13524
13254
13245

31245
31254
31524
35124
53124

53142
35142
31542
31452
31425

34125
34152
34512
35412
53412

54312
45312
```

```
43512
43152
43125

43215
43251
43521
45321
54321

53421
35421
34521
34251
34215

32415
32451
32541
35241
53241

53214
35214
32514
32154
32145

23145
23154
23514
25314
52314

52341
25341
23541
23451
```

```
23415

24315
24351
24531
25431
52431

54231
45231
42531
42351
42315

42135
42153
42513
45213
54213

52413
25413
24513
24153
24135

21435
21453
21543
25143
52143

52134
25134
21534
21354
21345
```