Asian Journal of Information Technology 6 (9): 956-957, 2007
ISSN: 1682-3915

Permutation Generation Algorithm

O.V. Viktorov
P.O.Box 1764 Amman 11821, Jordan

Abstract: A new permutation generation algorithm is presented in this study. The algorithm has the fastest software implementation because it uses exchanges of two elements, but certain software overhead is present due to analysis of generating numbers and m-module forward-backward counters.

Key words: Combinatorial algorithm, permutation, permutation generation algorithms, data structure, generating number, generating record

INTRODUCTION

Generation of *n!* permutations is used for encryption, matrix determinant calculation, random number generation, data encoding and optimization methods. Over thirty new algorithms have been presented for the last thirty years (Sedgewick, 1977; Roy, 1978; Kreher and Stinson,1998; Knuth, 1998; Iyer, 2001; Latif, 2004; Knuth, 2005). However, these algorithms differ too much in complexity and speed. Apparently, only fastest of them can be used in practice because the number of all possible permutation *n!* grows dramatically with n. The efficiency of permutation generation algorithm depends on the data structure that is used to represent permutations and simplicity of the operations used for permutation generation. Permutation generation algorithms based on exchange of two elements are the fastest ones, (Ives, 1976; Sedgewick, 1977; Roy,1978) but it is interesting to know is it possible to design an algorithm that uses only one exchange operation to generate each new permutation and how to choose two elements for it? The solution of this problem is presented in the study.

BACKGROUND

First of all the abstract data structures used in algorithms should be specified.

Data structure

Elements: Objects a_{i1} a_{i2}... a_{in}, distinguished from each other are located accordingly on positions 1, 2, 3..., n.

Operations: the exchange of two objects, whose positions are defined with the help of generating records and the use of modular recalculation.

Algorithm

1. Initialize the following variables: c(i) = 0, r(i) = 1, m(i) = n - i, i = 1,2..., n.
2. Initialize the following variables: i = 1; k = 1.
3. Generate the permutation a_1 a_2 a_3 ... a_n.
4. If c(i) = m(i) then go to the step 6.
5. Exchange the object located on the position c(i) + k with the object that is located on the positions c(i)+k+r (i); set c (i) = c(i)+r (i); go to the step 2.
6. If c(i + 1) = m(i + 1) then go to the step 8.
7. Exchange the object located on the position k with object that is located on positions n - k+1; set c(i) = n - i - m(i) and c(i+1) = c(i+1)+ r(i+1); go to the step 2.
8. Set r(i) = - r(i), m(i) = n - i - m(i); r(i + 1) = - r(i + 1); m (i + 1) = n - i - 1 - m(i + 1); i = i + 2; k = k + 1;
9. If i < n then go to the step 4.
10. The end

		Counter				Counter mode				Counter digit modules				Object position		Permutation			
L	k	C_1	C_2	C_3	C_4	r_1	r_2	r_3	r_4	m_1	m_2	m_3	m_4	x_1	x_2	C_1	C_2	C_3	C_4
1	1	0	0	0	0	1	1	1	1	3	2	1	0	1	2	1	2	3	4
2	1	1	0	0	0	1	1	1	1	3	2	1	0	2	3	2	1	3	4
3	1	2	0	0	0	1	1	1	1	3	2	1	0	3	4	2	3	1	4
4	1	3	0	0	0	1	1	1	1	3	2	1	0	1	4	2	3	4	1
5	1	0	1	0	0	1	1	1	1	3	2	1	0	1	2	1	3	4	2
6	1	1	1	0	0	1	1	1	1	3	2	1	0	2	3	3	4	2	1
7	1	2	1	0	0	1	1	1	1	3	2	1	0	3	4	3	4	1	2
8	1	3	1	0	0	1	1	1	1	3	2	1	0	1	4	3	4	2	1
9	1	0	2	0	0	1	1	1	1	3	2	1	0	1	2	1	4	2	3
10	1	1	2	0	0	1	1	1	1	3	2	1	0	2	3	4	1	2	3
11	1	2	2	0	0	1	1	1	1	3	2	1	0	3	4	4	2	1	3
12	1	3	2	0	0	-1	-1	1	1	0	0	1	0	2	3	4	2	3	1
13	1	3	2	1	0	-1	-1	1	1	0	0	1	0	4	3	4	3	2	1
14	1	2	2	1	0	-1	-1	1	1	0	0	1	0	3	2	4	3	1	2
15	1	1	2	1	0	-1	-1	1	1	0	0	1	0	2	1	4	1	3	2
16	1	0	2	1	0	-1	-1	1	1	0	0	1	0	1	4	1	4	3	2

Continue Table

		Counter				Counter mode				Counter digit modules				Object position		Permutation			
L	k	C_1	C_2	C_3	C_4	r_1	r_2	r_3	r_4	m_1	m_2	m_3	m_4	x_1	x_2	C_1	C_2	C_3	C_4
17	1	3	1	1	0	-1	-1	1	1	0	0	1	0	4	3	2	4	3	1
18	1	2	1	1	0	-1	-1	1	1	0	0	1	0	3	2	2	4	1	3
19	1	1	1	1	0	-1	-1	1	1	0	0	1	0	2	1	2	1	4	3
20	1	0	1	1	0	-1	-1	1	1	0	0	1	0	1	4	1	2	4	3
21	1	3	0	1	0	-1	-1	1	1	0	0	1	0	4	3	3	2	4	1
22	1	2	0	1	0	-1	-1	1	1	0	0	1	0	3	2	3	2	1	4
23	1	1	0	1	0	-1	-1	1	1	0	0	1	0	2	1	3	1	2	4
24	1	0	0	1	0	1	1	-1	-1	3	2	0	0	-	-	1	3	2	4

Sequence of all n! (n = 4) permutations is shown in the Example 2:

The counter digits c(i) are independent and their digit modules m (i) and forward or backward counting r(i) (1 is increment, - 1 is decrement) are changing during algorithm implementation execution. Theorem.

n! different permutations can be generated using operation exchange of two elements that are located on any i and j positions (i = 1, 2, . . . , n and j = 1, 2, . . . , n) $i \neq j$.

Proof: It should be noted that exchange operation can be applied to the permutation if n > 1.

Basis step: Let us show, that the theorem is true when n = 2 . a_1a_2 ➔exchange➔ a_2a_1. 2! = 2, so a_1a_2 and a_2a_1 are all *n!* permutations of 2 elements.

Induction step: Let us assume that theorem is true for n = k. We will show now that theorem is true for n = k + 1. Let us locate new element a_{k+1} on the position i (i = 1, 2, . . . , k + 1) to get a new permutation $a_1 a_2 \ldots a_{i-1}\, a_{k+1}\, a_i \ldots a_k$. For each of the following permutations $a_1 a_2 \ldots a_k\, a_{k+1}$, $a_1 a_2 \ldots a_{k-1}\, a_{k+1}\, a_k$, . . . , $a_{k+1}\, a_1 a_2 \ldots a_k$ we can generate k! permutations of k + 1 elements using operation exchange of two elements. So, the total number of different (k+1) – permutations is (k+1) k! = (k + 1)!.

RESULTS

A new permutation generation algorithm is presented in this study. It has been proved that *n!* permutations can be generated using operation exchange of two elements that are located on any i and j positions (i = 1, 2, . . . , n and j = 1, 2, . . . , n) $i \neq j$.

CONCLUSION

The algorithm has the fastest software implementation because it uses one exchange of two elements to generate each new permutation from previous one, but certain software overhead is present due to analysis of generating numbers and m-module forward-backward counter states. Application of suggested algorithm allowed us to generate permutations with n more than 10 that was practically impossible to generate by algorithms that are known before.

REFERENCES

Ives, F.M., 1976. Permutation Enumeration: Four new algorithms, J. Assoc. Compu. Machinery, 19: 68-72.

Iyer, M.G., 2001. Permutation Generation Using Matrices, Dr. Dobb's Portal, 22: 26-38.

Knuth, D.E., 1998. The Art of Programming, Fundamental Algorithms, (3rd Edn.), Addison-Wesley Longman, Vol. 1.

Knuth, D.E., 2005.The Art of Programming, Volume 4, Fascicle 2, Generation All Tuples and Permutations Addison-Wesley Professional.

Kreher, D.L. and R. Stinson, 1998. Combinatorial Algorithms: Generation, Enumeration and Search, CRC.

Latif, U., 2004. Random Permutation Generation, TechUser. Net, 13:123-127.

Roy, M.K., 1978. Evaluation of Permutation Algorithms, The Comput. J., 21, 4: 296-301.

Sedgewick. R., 1977. Permutation Generation Methods, Computing Surveys, 9: 137-164.