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; Krencher 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_1, a_2,..., a_n, 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_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 c(i) = n - i - m(i) and c(i) = n - i - m(i); r(i+1) = r(i) - m(i+1); i = i + 2; k = k + 1;
9. If i < n then go to the step 4.
10. The end
### 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 \((i = 1, 2, \ldots, n) \) and \( j \neq i \).

### 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


