A permutation generation method

C. T. Fike

IBM Systems Research Institute, 219 East 42nd Street, New York, N.Y. 10017, USA

This paper describes a new method for generating permutation sequences. Timing experiments indicate that the method is competitive with the interchange methods of Wells, Johnson and Trotter. It is therefore among the fastest available. The method has the advantage that it generates permutations in what Lehmer calls an orderly listing.

(Received June 1973)

Introduction

The purpose of this note is to describe a new method for generating permutation sequences. A number of permutation generation methods already exist, and there is an extensive literature on the subject (see for example, Ord-Smith; 1970, 1971). There are two reasons for adding this method to the already long list of known methods. First, it is fast. Second, like any method, it may be specially advantageous for some applications, such as certain types of optimisation problems. The applications of permutation generation methods include problems like the travelling salesman problem, the assignment problem, and Latin square enumeration; see Ord-Smith (1970) and Wells (1971, Chapters 4 and 7).

Let n denote an integer such that $n \geq 2$. The problem is how to generate all the $n!$ permutations of n marks $123\ldots n$. Consider the set S of sequences $(d_2, d_3, \ldots, d_n)$, where d_k denotes an integer such that $1 \leq d_k \leq k$. The set S contains $n!$ sequences, and a computer program can easily generate them all. To invent a permutation generation method, it is only necessary to invent a one-to-one correspondence between the $n!$ sequences in S and the $n!$ permutations of the n marks $123\ldots n$. (This observation is the starting point for many permutation generation methods; see Lehmer; 1960).

Let the permutation corresponding to a sequence $(d_2, d_3, \ldots, d_n)$ be that one obtained with the following interchange rule:

> Starting with the original arrangement of the marks $123\ldots n$, interchange the mark in position k with the mark in position d_k, for $k = 2, 3, \ldots, n$. (When $d_k = k$, a mark is inter changed with itself, so no real rearrangement occurs.)

It can be verified readily that this correspondence between sequences and permutations is one to one.

Example 1:

Suppose $n = 5$. Find the permutation corresponding to the sequence (2, 1, 3, 2). Start with the initial arrangement 12345, and perform interchanges in positions 2-2, 3-1, 4-3, and 5-2. The resulting permutation is 35412. This process is reversible, and one can just as easily construct the sequence from the permutation.

Program implementation

To generate all the $n!$ permutations of n marks, generate the $n!$ sequences in S, and for each sequence construct the corresponding permutation. This process, which might appear to be inconvenient, can actually be performed in a backtrack program of great clarity and simplicity. A backtrack program is just a program whose purpose is a systematic search or enumeration and which is organised as a nest of iterations. Such a program is a fundamental tool of combinatorial computing; see Wells (1971, Chapter 4) for a thoroughgoing treatment of such programs. The PL/I backtrack program in **Fig. 1** generates the 4! permutations of the integers 1234. The program uses recursion, which is a natural device for backtrack programming. Since the purpose of the program is simply to illustrate the generation method, the program does nothing with the permutations except to print them.

The general idea of the program in Fig. 1 is this. The program generates all the sequences $(d_2, d_3, \ldots, d_n)$, varying d_n most rapidly. In the process of constructing a permutation it interchanges $P(K)$ and $P(D(K))$ in the way that the interchange rule prescribes. Later it interchanges them again to restore the status quo; this second interchange is the backtrack step in the

```
GO:   PROCEDURE OPTIONS (MAIN) REORDER;
      %DECLARE N FIXED; %N = 4;
      DEFAULT RANGE (*) FIXED BINARY;
      DECLARE P(N), D(2:N);
      DO I=1 TO N; P(I) = I; END;
      CALL PERMUTE (2);
      STOP;

PERMUTE:
      PROCEDURE (K) RECURSIVE;
      DECLARE TEMP;
      TEMP = P(K);
      DO D(K)=K TO 1 BY -1;
        P(K) = P(D(K));
        P(D(K)) = TEMP;
        IF K<N THEN CALL PERMUTE (K+1);
               ELSE PUT EDIT (P) (COL(1),(N)F(1));
        P(D(K)) = P(K);
        P(K) = TEMP;
        END;
      RETURN;
      END;
      END GO;
```

Fig. 1 A PL/I backtrack program that generates and prints the 4! permutations of the integers 1234

permutation	*sequence*	*permutation*	*sequence*
1234	(2, 3, 4)	2134	(1, 3, 4)
1243	(2, 3, 3)	2143	(1, 3, 3)
1432	(2, 3, 2)	2431	(1, 3, 2)
4231	(2, 3, 1)	4132	(1, 3, 1)
1324	(2, 2, 4)	2314	(1, 2, 4)
1342	(2, 2, 3)	2341	(1, 2, 3)
1423	(2, 2, 2)	2413	(1, 2, 2)
4321	(2, 2, 1)	4312	(1, 2, 1)
3214	(2, 1, 4)	3124	(1, 1, 4)
3241	(2, 1, 3)	3142	(1, 1, 3)
3412	(2, 1, 2)	3421	(1, 1, 2)
4213	(2, 1, 1)	4123	(1, 1, 1)

Fig. 2 The 4! permutations of the integers 1234 in the order they are produced by the program in Fig. 1, together with the corresponding sequences in set S

```
GO:  PROCEDURE OPTIONS (MAIN) REORDER;
     %DECLARE N FIXED; %N = 4;
     DEFAULT RANGE (*) FIXED BINARY;
     DECLARE P(N), D(2:N), MORE BIT (1) ALIGNED;
     DO I=1 TO N; P(I) = I; END;
     PUT EDIT (P) (COL(1),(N)F(1));
     DO I=2 TO N; D(I) = I; END;
     MORE = '1'B;
     DO WHILE (MORE);
       DO K=N TO 2 BY -1 WHILE (D(K)=1);
         TEMP = P(K);
         P(K) = P(1);
         P(1) = TEMP;
         D(K) = K;
         END;
       IF K=1 THEN MORE = '0'B;
              ELSE DO;
                   TEMP = P(D(K));
                   P(D(K)) = P(K);
                   P(K) = P(D(K)-1);
                   P(D(K)-1) = TEMP;
                   D(K) = D(K) - 1;
                   PUT EDIT (P) (COL(1),(N)F(1));
                   END;
       END;
     STOP;
     END GO;
```

Fig. 3 A nonrecursive PL/I program that generates and prints the 4! permutations of the integers 1234

program. The table in **Fig. 2** shows (1) the permutations in the order they are listed by the program, and (2) the corresponding sequences in S.

One can, of course, also program the generation method without recursion. The program in **Fig. 3** does exactly the same thing as that in Fig. 1, but without using recursion. The generation method is, however, not so obvious in this second program.

Timing experiments indicate that this method compares favourably with the interchange methods of Wells (1961), Johnson (1963), and Trotter (1962). In my tests, programs similar to those in Figs. 1 and 3 generated the 10! permutations of the integers 1 to 10 in less time than comparable programs based on the Wells, Johnson and Trotter methods*. Such a comparison of methods is not definitive, because it depends on programming skill, compiler performance, and machine performance as much as it does on permutation methods, but it does indicate that the new method deserves to be classified with the fastest methods previously available. In a way this is surprising, for the following reason. The program in Fig. 1 performs over $2n!$ interchanges in the process of generating $n!$ permutations. On the other hand, each of the three methods mentioned above performs only $n! - 1$ interchanges. The reason why the new method performs well in comparison to them seems to be its simplicity. It does perform more interchanges but, on the other hand, does hardly anything else.

Serial numbers

Let the $n!$ permutations of n marks be numbered from 0 to $n! - 1$ in the order they would be produced by the two programs in Figs. 1 and 3. The serial number of a permutation is its number in the list. This method generates permutations in what Lehmer (1964) calls an *orderly listing*. This means two things. First, one may find the serial number of a permutation without generating any other permutations. Second, one may obtain the pth permutation in the list directly from its serial number p.

For this method these manipulations with serial numbers involve representation of an integer p such that $0 \leq p \leq n! - 1$ in the form

$$p = n!\left(\frac{d_2'}{2!} + \frac{d_3'}{3!} + \cdots + \frac{d_n'}{n!}\right), \qquad (1)$$

where d_k' denotes an integer such that $0 \leq d_k' \leq k - 1$. Such a representation is unique for each p. This number representation, which is related to the more familiar factorial representation of integers (Lehmer, 1964), is not new and was, in fact, used by Johnson (1963) in his description of his interchange method.

The serial number of a given permutation may be obtained as follows. Find the sequence $(d_2, d_3, \ldots, d_n)$ corresponding to the permutation, and let $d_k' = k - d_k$ for $k = 2, 3, \ldots, n$. Then the serial number p for the permutation is given by (1).

Example 2:
Suppose $n = 5$. Find the serial number of the permutation 35412. The corresponding sequence is (2, 1, 3, 2). (See Example 1.) Therefore, the serial number is

$$5!(0/2! + 2/3! + 1/4! + 3/5!) = 48 .$$

The permutation having a given serial number p may be formed as follows. Construct the representation (1) for p. Let $d_k = k - d_k'$ for $k = 2, 3, \ldots, n$, and form the sequence $(d_2, d_3, \ldots, d_n)$. Then the permutation sought is the one corresponding to this sequence.

Example 3:
Suppose $n = 5$. Find the permutation whose serial number is 109. Since $109 = 5!(1/2! + 2/3! + 1/4! + 4/5!)$, the sequence corresponding to the permutation is (1, 1, 3, 1), and the permutation is 51423.

*Actual timings for the new method are these. A recursive test program with no print statements generated 10! permutations in 9·25 minutes on an IBM System/360 Model 50; a nonrecursive program, in 10·76 minutes.

References

JOHNSON, S. M. (1963). Generation of permutations by adjacent transposition. *Mathematics of Computation*, Vol. 17, pp. 282-285.

LEHMER, D. H. (1960). Teaching combinatorial tricks to a computer. Chapter 15 in Bellman, R., and Hall, M., Jr. (Eds.), *Combinatorial Analysis, Proc. Sympos. Applied Math.* 10, American Mathematical Society, Providence, R. I., pp. 179-193.

LEHMER, D. H. (1964). The machine tools of combinatorics. Chapter 1 in Bechenbach, E. F. (Ed.), *Applied Combinatorial Mathematics*, Wiley, New York, pp. 5-31.

ORD-SMITH, R. J. (1970). Generation of permutation sequences: part 1. *The Computer Journal*, Vol. 13, pp. 152-155.

ORD-SMITH, R. J. (1971). Generation of permutation sequences: part 2. *The Computer Journal*, Vol. 14, pp. 136-139.

TROTTER, H. F. (1962). Algorithm 115, Perm. *CACM*, Vol. 5, p. 434.

WELLS, M. B. (1971). *Elements of Combinatorial Computing*. Pergamon Press, New York.

WELLS, M. B. (1961). Generation of permutations by transposition. *Mathematics of Computation*, Vol. 15, pp. 192-195.