Ord-Smith's pseudo-lexicographical permutation procedure is the Tompkins–Paige algorithm

Two of the classical permutation algorithms that produce the sequence in pseudo-lexicographical order are those of Tompkins and Paige, on the one hand, and Ord-Smith, on the other. The first uses only rotation, the second only reversing. This paper shows that they are closely related, in that a variant of the Tompkins–Paige algorithm which uses right-rotation, rather than the usual left-rotation, can be simply transformed into the Ord-Smith algorithm.

Received January 1990

Table 1. The permutation sequences for $n = 4$

ACM 68[a]				ACM 308[b]					Signature		
p_1	p_2	p_3	p_4	p_1	p_2	p_3	p_4	Rank	q_4	q_3	q_2
1	2	3	4	1	2	3	4	0	0	0	0
2	1	3	4	2	1	3	4	1	0	0	1
2	3	1	4	3	1	2	4	2	0	1	0
3	2	1	4	1	3	2	4	3	0	1	1
3	1	2	4	2	3	1	4	4	0	2	0
1	3	2	4	3	2	1	4	5	0	2	1
2	3	4	1	4	1	2	3	6	1	0	0
3	2	4	1	1	4	2	3	7	1	0	1
3	4	2	1	2	4	1	3	8	1	1	0
4	3	2	1	4	2	1	3	9	1	1	1
4	2	3	1	1	2	4	3	10	1	2	0
2	4	3	1	2	1	4	3	11	1	2	1
3	4	1	2	3	4	1	2	12	2	0	0
4	3	1	2	4	3	1	2	13	2	0	1
4	1	3	2	1	3	4	2	14	2	1	0
1	4	3	2	3	1	4	2	15	2	1	1
1	3	4	2	4	1	3	2	16	2	2	0
3	1	4	2	1	4	3	2	17	2	2	1
4	1	2	3	2	3	4	1	18	3	0	0
1	4	2	3	3	2	4	1	19	3	0	1
1	2	4	3	4	2	3	1	20	3	1	0
2	1	4	3	2	4	3	1	21	3	1	1
2	4	1	3	3	4	2	1	22	3	2	0
4	2	1	3	4	3	2	1	23	3	2	1

[a] The Tompkins–Paige algorithm, given by Peck and Schrank (ACM 86).
[b] The Ord-Smith algorithm (ACM 308).

1. Introduction

All of the permutation generation methods currently in use are quite efficient – they take, on average, a fixed (and small) time to generate each permutation of the sequence. Furthermore, they belong to one of two groups, all members of which can be speeded up by the application of the same techniques. Therefore, as Ord-Smith (1970) has observed, interest now concentrates on the properties of the sequences generated. Such properties include the reflection-free property, the lexicographical property, the adaptability to back-tracking, and the nature of the transformation from one permutation of a sequence to the next. Consequently the relationships between methods of permutation generation form an important area of study. In this paper we show that the pseudo-lexicographical algorithm of Ord-Smith is essentially equivalent to the Tompkins–Paige algorithm.

As is traditional, we assume that the permutations will take place *in situ* in an array, *p*, of marks. *P* is of type *perm* which, assuming the appropriate definition of *maxrange*, is defined:

range = 1...*maxrange*;
mark = *unspecified*;
perm = **array** [*range*] **of** *mark*;

Considering only the operations on the marks, as distinct from the operations needed for control purposes, all permutation methods use either:

- The swapping of two elements of *p*.
- The rotation of a contiguous part of *p*.
- The reversing of a contiguous part of *p*.
- Some combination of these.

The Tompkins–Paige algorithm uses rotation exclusively, while Ord-Smith's algorithm uses only reversing. It seems strange, at first sight, that these two should be intimately related. Indeed, from the output given in Table 1 from the ACM algorithms implementing these methods it is very difficult indeed to see any similarity, and it is easy to forgive the reviewers of permutation algorithms, Sedgewick[6] and Ord-Smith[2, 3] himself for not discovering it. (The other reviewer, Roy,[5] discussed neither method.) The table contains also the *rank* of each permutation (i.e. its position in the sequence) and its *signature*, a notion we will discuss presently. It is the purpose of this paper to show the relationship, and then to prove the equivalence of the methods. We take the view that the phrase 'the X permutation algorithm' applies not just to the text of the procedure which first presented the algorithm X, but to a whole class of procedures which use the same basic strategy. Within a given class, there are procedures which produce the marks in each permutation in reverse order; procedures which produce the sequence of permutations in reverse order; procedures which produce a sequence of inverse permutations; and those which combine a selection of these, and other variants. Our claim, then, is that the Tompkins–Paige and the Ord-Smith algorithms are in the same class, the Ord-Smith algorithm being essentially a transformation of one variant of the Tompkins–Paige algorithm.

2. The Ord-Smith algorithm

The Ord-Smith algorithm as given in ACM 308 was coded in Algol, the appropriate language of the time. In Figure 1 we give a version in Pascal. Its action is specified by its pre- and post-conditions. On the first entry *first* should be *true* and *x* should contain the initial permutation (whose rank is 0): on subsequent entries *first* should be *false*, *x* should contain a permutation and *q* contain its signature. (The function *count* to be defined shortly returns a natural number which is equal to the rank of the corresponding permutation.) On exit either the permutation provided in *x* was the last, in which case *first* is set *true*, or else *first* is set *false*, *x* is set to the next permutation and *q* adjusted accordingly. Note that *q* is quite redundant. It is just a particular representation of the rank, as described later. Provided that the implementation of natural numbers is generous enough, it could be replaced by *rank*. Note that in Figure 1 *Econoperm* has been subject to some correctness-preserving transformations:

- Structured constructs have been introduced, since Algol was not sympathetic to their use.

```
procedure Econoperm (var x: perm;
                     n: range;
                     var first: Boolean;
                     var q: signature);
{Pre: $first \wedge rank(x) = 0 \vee \sim first \wedge$
      $0 < count(q) = rank(x) \leqslant n! - 1$}
{Post: $first \wedge rank(x) = rank(X) = n! - 1 \vee$
      $\sim first \wedge 0 < count(q) = rank(x) =$
      $rank(X) + 1 \leqslant n! - 1$}
  var
    k: range;
begin
  if first then
    for $k := 2$ to $n+1$ do
      $q[k] := 0$;
  $k := 2$;
  while $q[k] = k - 1$ do
    begin
      $q[k] := 0$;
      $k := k + 1$
    end;
  $first := k = n + 1$;
  if not first then
    begin
      Reverse(x, 1, k);
      $q[k] := q[k] + 1$;
    end
end; {of procedure 'Econoperm'}
```

Figure 1. The Ord-Smith algorithm.

- The sequence for reversal has been abstracted into the procedure *Reverse*(p, k, n) which reverses the elements $p[k \ldots n]$.
- Identifiers have been changed to allow compatibility with Figure 2.
- The procedure has been fully parameterized, since Pascal does not include the notion of an own variable.
- The range of $c[k]$ has been transformed from $1 \ldots k$ to $0 \ldots k-1$, to enable the post-condition to be more simply expressed. Note that this does not imply a change of computation as $c[k]$ is used purely for counting. The type of c is:

signature = **array** [*range*] **of** *range*

These transformations preserve the fundamental character of the procedure, and indeed the procedure of Figure 1 was obtained by the author from a sequence of transformations from a version transliterated from the original Algol. As Ord-Smith did in his review we have extended the procedure to handle the case of $n = 1$.

We will not seek to prove the algorithm here, because it is much easier to do when it is expressed as a recursive procedure which generates all permutations in sequence. Instead, we content ourself with explaining its action. We will be considering the change to produce a permutation from its successor – and we will concentrate on the second of the pair. Notice first that every odd-ranked permutation is obtained from its predecessor by reversing the first two elements: the seventh permutation (1 4 2 3) is obtained from the sixth (4 1 2 3) by reversing the 4 and the 1. Now consider the even-ranked permutations. They are obtained by bigger reversals. To get the second and the fourth the first three elements are reversed. For the sixth the first four. (In this example all four!) Then for the eighth and tenth the reversal is again of length three, followed by one of length four for the twelfth. This continues until the twenty-fourth, which would require a reversal of length five. These numbers 1, 2, 6, 24 are the factorials of 1, 2, 3, 4. The signature q holds the information required to determine the length of shift required. If $q_2 = 1$ then the shift is of length two; if $q_2 = 0$ and $q_3 = 1$ or 2, then the shift is of length three; if $q_2 = q_3 = 0$ and $q_4 = 1$, 2 or 3, then the shift is of length four; and so on. The values attained by q_k range from 0 to $k-1$. The adjustment process involves finding the smallest k such that q_k has not reached its maximum, incrementing it and resetting to 0 all the elements of q whose subscript is less than k.

Notice that in Table 1 the elements of q have been written backwards, because it is instructive to read them as the digits of a number, that, rather than having weights of 1, 10, 100 and so on in the usual decimal system, have weights of 1, 2, 6, 24 – i.e. the factorial numbers. Thus (2 2 1) has the value $2*6+2*2+1*1 = 17$. Each signature has the same value as the rank of its associated permutation. (Thus the signature, as we noted earlier, is clearly redundant because it can be obtained from the rank by successive divisions by the factorial numbers.) This idea of having different weights for digits permeates the Imperial system of weights and measures. It has been called a *mixed-radix* system by Ord-Smith and *factorial counting* by Sedgewick. This counting is a feature of all pseudo-lexicographical permutation algorithms.

3. The Tompkins–Paige algorithm

The Tompkins–Paige algorithm is a little more complicated because the action required to convert one permutation into the next is not a single (reversing) operation, as is the case with Ord-Smith, but a sequence of (rotating) operations. Every odd-ranked permutation is obtained from its predecessor by rotating the first two elements: with two elements, rotating and reversing are equivalent! To get the second and the fourth permutations from their predecessors, the first two elements are rotated left and then the first three. For example, given that the third perm is (3 2 1 4), the fourth (3 1 2 4) is got by rotating the first two elements giving (2 3 1 4), and then the first three giving (3 1 2 4). For the sixth permutation, the first two, then the first three and lastly the first four elements are rotated. And so on. Within the algorithm this is achieved by rotating within the loop searching for k, as well as in the sequence for updating x. A Pascal procedure, based on the original algorithm given by Peck and Schrank, and transformed along the same lines as the Ord-Smith algorithm appears as Figure 2.

```
procedure Permute (var x: perm; n: range;
                   var first: Boolean;
                   var q: signature);
{Pre: first ∧ rank(x) = 0 ∨ ~first ∧
      0 < count(q) = rank(x) ≤ n! − 1}
{Post: first ∧ rank(x) = rank(X) = n! − 1 ∨
       ~first ∧ 0 < count(q) = rank(x) =
       rank(X) + 1 ≤ n! − 1}
  var k: range;
begin
  if first then
    for k := 2 to n + 1 do
      q[k] := 0;
  k := 2;
  while q[k] = k − 1 do
    begin
      Rotate(x, 1, k);
      q[k] := 0;
      k := k + 1;
    end;
  first := k = n + 1;
  if not first then
    begin
      Rotate(x, 1, k);
      q[k] := q[k] + 1
    end
end: {of procedure 'Permute'}
```

Figure 2. The Tompkins–Paige algorithm.

4. The relationship of these procedures

As we noted to start with the sequences generated by these two procedures are quite different. But, suppose that the direction of rotation in the Tompkins–Paige algorithm is changed to be a right shift. (We are expanding the class of Tompkins–Paige algorithms to include those with right-rotation and even those in which the direction of rotation varies with the length of rotation!) In this case the sequence generated is precisely that of the Ord-Smith procedure!

It is interesting to note that *all* descriptions of the Tompkins–Paige algorithms so far published have used the left shift automatically, as if there were no other way!

As the descriptions of Sections 2 and 3 showed, the only difference between the algorithms is that Tompkins–Paige requires a series of progressively longer rotations to take place on p, while the Ord-Smith requires a single reversal. Using the symbol ';' in a way analogous to the traditional use of Σ and π, to represent the concatenation of statements, the equivalence of the procedures can be expressed as the identity:

$$\mathop{;}_{i=2}^{k} \mathit{Rotate}(p, 1, i) = \mathit{Reverse}(p, 1, k)$$

which may be easily proved by induction on the length of the string being reversed ($= k-1$), provided that the *Rotate* represents a right rotation.

REFERENCES

1. R. J. Ord-Smith, Generation of permutations in pseudo-lexicographical order (Algorithm 308), *Comm. ACM* **10** (7), 452 (1967).
2. R. J. Ord-Smith, Generation of permutation sequences Part I, *Computer J.* **13** (3), 152–155 (1970).
3. R. J. Ord-Smith, Generation of permutation Sequences Part II, *Computer J.* **14** (2), 136–139 (1971).
4. J. E. L. Peck and G. F. Schrank, Permute (Algorithm 86), *Comm. ACM* **5** (4), 210–211 (1962).
5. M. K. Roy, Evaluation of permutation algorithms, *Computer J.* **21** (4), 296–301 (1978).
6. R. Sedgewick, Permutation Generation Methods, *Computer Surveys* **9** (2), 136–164 (1977).

J. S. ROHL[1]
Department of Computer Science,
The University of Western Australia,
Nedlands, Western Australia, 6009

[1] On sabbatical leave at Cornell University.