Algorithm 122

*GENERATION OF PERMUTATIONS IN DIRECT LEXICOGRAPHIC ORDERING BY ARITHMETIC METHODS**

ENRICO SPOLETINI
Dipartimento di Fisica, Universitá di Milano (Italia)
* *This work is supported by Italian M.P.I.*

Received 21 March 1983; in final form 15 August 1984

Sedgewick in a survey paper[4] presents and compares various methods of generating permutations of the k marks 0, 1,..., $(k-1)$. Almost all these methods and other ones, more recently published, construct the permutations by means of exchanges and cycles or recursive procedures. Lehmer[3] and Howell[1] give an arithmetic method for constructing permutations in lexicographic ordering. Precisely, Howell shows that all permutations of the k marks 0, 1, ..., $(k-1)$ may be obtained by successively adding the number $(k-1)$ radix k to the permutation 0 1 ...$(k-1)$, regarded as a base k integer and rejecting all results which contain repeated digits. He gives also an algorithm[2], which turns out to be very slow, based on this result.

In Ref. 5 are proved two formulas for computing:

(i) the value v_1 to add radix k to the integer corresponding to the current permutation in order to obtain the next one;

(ii) the value v_2 to add radix k to the integer corresponding to the first permutation in order to obtain the mth permutation.

Starting from the above results two algorithms are given:

(1) nextlex(a, k) to give the next permutation P_{i+1} in the direct lexicographic ordering after $P_i \equiv a(k), \ldots, a(1)$;

(2) mthlex(k, m, a) to give the mth permutation P_m in the direct lexicographic ordering (where $P_1 \equiv 0, 1, \ldots, (k-1)$).

The algorithm 'nextlex' uses arithmetic only techniques of the type Howell's paper does; it also provides substantial improvement on the algorithm[2]. In fact starting from P_i 'nextlex' directly gives P_{i+1}, whereas in[2], P_{i+1} is obtained only after inspecting several sequences of marks; obviously 'nextlex', is less efficient than analogous algorithms which utilise exchanges (see, for instance[4]).

The algorithm 'mthlex' can be used in the case when k is large, and we are not interested in generating all the permutations, but it is enough to obtain a limited number of them (for instance in any choice problem). Between the sets $\{i|1 \leqslant i \leqslant k!\}$ and $\{P_i|1 \leqslant i \leqslant k!\}$ there is the one-to-one correspondence $i \sim P_i$ which preserves the ordering, for this reason 'mthlex' allows ordered subsets of permutations to be obtained starting from increasing sequences of integers.

Moreover, compared with the Lehmer and Howell algorithms, the ones supplied here enable the computation of v_1 and v_2 to be made in the radix most suitable for the utilised computer, using operation radix k in the final operations only.

```
procedure nextlex(a,k);
value a,k; integer k; integer array a;
comment if a[k], a[k-1],...,a[1] is a given permutation of the k marks 0,1,...,(k-1),
    the next permutation is generated by an arithmetic method. In case the given permutation is
    (k-1),...,1,0 the next permutation is taken to be 0,1,...,(k-1). The first step of the
    method is to compute the difference t between the next and the current permutation.
    The second step is to add t to a, radix k. We remark that it is sufficient to use the
    usual addition in the algorithm if the greatest available integer for our computer is
    larger than k**(k-1); in the opposite case a suitable routine for the computation of
    t and for the addition to a, radix k, in order to treat integers in multiple precision,
    must be used;

begin integer i,j,h,q,t,u;
    h:=k; u:=1;
    while u < k do
        if a[u] < a[u+1]
        then u:=u+1
        else begin h:=u+1; u:=k end;
    if h > k then
        for i:=1 step 1 until k do a[i]:=k-i
    else
        begin
            q:=k; i:=1; j:=1;
            while a[j] < a[h] do j:=j+1;
            t:=k**(h-j-1)*(1-k**j)*(a[h]-a[j]); i:=1;
            while 2*i < h do
                begin
                    t:=t+k**(i-1)*(1-k**(h-2*i))*(a[h-i]-a[i]);
                    i:=i+1
                end;
            i:=1;
            while t ≠ 0 do
                begin
                    q:=(t+a[i])÷k; a[i]:=(t+a[i])-q*k; t:=q; i:=i+1
                end
        end
end
```

```
procedure mthlex(k,m,a);
value k,m; integer k,m; integer array a;
comment The procedure generates the m^th permutation of the lexicographic ordering. The
    first step of the method is to produce the factorial representation of m-1, if m is
    greater than k! the procedure produces only zeros in a The second step is to compute
    the difference t between the m^th and the first permutation. The last step is to add t
    to P1 radix k. We remark that it is sufficient to use the usual addition in the
    algorithm if the greatest available integer in our computer is larger than k**k, otherwise
    a suitable routine must be used for the computation of t and for the addition of t to
    P1, radix k, in order to treat integers in multiple precision;

begin integer i,j,h,r,s,t,u,v,w; integer array b[1:k-1];
    t:=1;
    for i:=1 step 1 until k do t:=t*i;
    if t < m
        then
            begin
                for i:=1 step 1 until k do a[i]:=0
            end
        else
            begin
                m:=m-1; i:=1;
                while m ≠ 0 do
                    begin
                        b[i]:=m; m:=m÷(i+1); b[i]:=b[i]-m*(i+1); i:=i+1
                    end;
                h:=i-1; t:=0; i:=1;
                while i ⩽ h do
                    begin
                        t:=t+b[i]*k**i-(k*i-k**(i-b[i]))÷(k-1); i:=i+1
                    end;
                for u:=h-1 step -1 until 1 do
                    begin
                        v:=u+1;
                        a[u]:=b[u]-b[v]+1;
                        if a[u] > 0 then a[u]:=1 else a[u]:=0;
                        for i:=2 step 1 until u do
                            begin
                                s:=0; w:=v-i;
                                for r:=w+1 step 1 until u do s:=s+a[r];
                                a[w]:=b[w]-b[v]+i-s;
                                if a[w] > 0 then a[w]:=1 else a[w]:=0;
                            end;
                        for i:=h-u step 1 until h do
                            begin
                                w:=h-i; s:=v-b[v];
```

```
            for r:=w step 1 until u do s:=s-a[r];
            t:=t+a[w]*(k**w-k**s)
          end
      end;
    for i:=k-1 step -1 until 0 do
      begin
        s:=(i+t)÷k; a[k-i]:=i+t-s*k; t:=s
      end
  end
end
```

REFERENCES

1. J. R. Howell. Generation of permutations by addition. *Mathematics of Computation* **16** 243–244 (1962).
2. J. R. Howell, Algorithm 87: permutation generator, *Communications of the ACM* **5** (4) 209 (1962).
3. D. H. Lehmer, Teaching combinatorial tricks to a computer, *Proceedings of the Symposium on Applications of Mathematics to Combinatorial Analysis*, **10**, American Mathematical Society, Providence, R.I., 179–193 (1960).
4. R. Sedgewick, Permutation generation methods, *ACM Computing Surveys* **9**, 137–163 (1977).
5. E. Spoletini, Generation of permutations following Lehmer and Howell, *Mathematics of Computation* **43**, 565–572 (1984).