Discrete Applied Mathematics 36 (1992) 57-65
North-Holland

Generating permutations with given ups and downs

D. Roelants van Baronaigien and Frank Ruskey*

Department of Computer Science, University of Victoria, Victoria, B.C., Canada V8W 2Y2

Received 7 February 1989
Revised 6 December 1989

Abstract

Roelants van Baronaigien, D. and F. Ruskey, Generating permutations with given ups and downs, Discrete Applied Mathematics 36 (1992) 57-65.

A permutation with a signature $Q=(q_1,q_2,\dots,q_{n-1})$ where q_i is either 1 or -1, is a permutation, $P=\pi_1,\pi_2,\dots,\pi_n$, of the integers 1 to n such that $q_i(\pi_{i+1}-\pi_i)$ is positive for all $1\le i\le n-1$. Alternating permutations are an example of permutations with the signature $(1,-1,1,-1,\dots)$. A constant average time algorithm is developed for generating all permutations with a given signature. Permutations are represented by a variant of their inversion tables. Ranking and unranking algorithms are also discussed.

1. Introduction

Let $Q=(q_1,q_2,\dots,q_{n-1})$ be a sequence such that $q_i=+1$ or $q_i=-1$. A *permutation with signature Q* is a permutation $P=\pi_1,\pi_2,\dots,\pi_n$ of the integers $1,2,\dots,n$ such that $\pi_i<\pi_{i+1}$ if $q_i=+1$ and $\pi_i>\pi_{i+1}$ if $q_i=-1$ for all $i=1,2,\dots,n-1$. Alternating permutations are permutations with signature $Q=(+1,-1,+1,-1,\dots,(-1)^{i+1},\dots)$. Some work has been done on the problem of enumerating permutations with a given signature. See for example Niven [11], Foulkes [6], Abramson [1], and De Bruijn [3]. There are constant average time algorithms by Bauslaugh and Ruskey [2], Pruesse and Ruskey [12] and Ruskey [15] for generating alternating permutations. The problem of generating all permutations with a given signature is equivalent to the problem of generating all topological sortings of a poset whose Hasse diagram (regarded as an undirected graph) is a path. Kalvin and Varol [7] give a linear average time algorithm for generating all topological sortings. This paper presents the first constant average time algorithm for generating all permutations with a given signature.

* Research supported by the Natural Sciences and Engineering Research Council of Canada under grant A3379.

Consider a class of permutations that we wish to generate. Let N be the number of permutations of $1, 2, \ldots, n$ being generated. An algorithm for generating that class of permutations is said to run in *constant average time* if the total amount of computation (excluding output) is $O(N)$. Up to a constant factor no algorithm for generating those permutations is faster. An algorithm is said to run in *linear average time* if the total amount of computation is $O(nN)$.

Our motivation in studying this problem is to further extend the class of combinatorial objects that can be generated in constant average time. Furthermore, our algorithm can be used to generate other classes of permutations. A *run* in a permutation $\pi_1, \pi_2, \ldots, \pi_n$ is a maximal length subsequence $\pi_i \leq \pi_{i+1} \leq \cdots \leq \pi_j$. As noted in [8] the problem of generating all permutations with r runs can be solved by listing all permutations with signatures having $r-1$ negative ones. Thus our algorithm could be used to generate all permutations with a given number of runs in constant average time. There are a number of existing algorithms for generating all combinations of $r-1$ out of $n-1$ in constant average time [13].

The generation algorithm of [2] lists alternating permutations in lexicographic order. Many other generation algorithms list objects in lexicographic order. See Williamson [20], Reingold, Nievergelt and Deo [13] and Nijenhuis and Wilf [10] for many examples. Colexicographic (colex) order is a variant of lexicographic order in which the sequences are scanned from right-to-left, instead of left-to-right. Colex order (or reverse colex order) has been used by Nijenhuis and Wilf [10] to generate compositions, permutations, and Young Tableau with a specific shape (see also Wilf [18]).

The generating algorithm presented here lists a natural representation of the permutations in colex order.

Any generation algorithm imposes an order on the objects being generated. For lexicographic generation, the order is quite natural. The *rank* of an object with respect to some ordering is the number of objects that come before it in the ordering. An *unranking* algorithm determines the object having a particular rank. Unranking algorithms are often used as a method of generating a random object. Ranking and unranking algorithms are normally quite straightforward for lexicographic order. Examples may be found in [10,19,13,21].

An algorithm for generating all permutations with a given signature is given in the second section. An analysis of the generation algorithm is given in the third section, and in the fourth section algorithms are discussed for ranking and unranking. Some open problems are mentioned in the final section.

2. The generation algorithm

In this section, we give an algorithm for the colex generation of a representation of permutations with a given signature. The representation that we use is related to the inversion table of the permutation (Knuth [9]).

Given a permutation π of $1, 2, \ldots, n$, we define the *P-sequence* associated with π, $P[\pi] = p_1, p_2, \ldots, p_n$ as follows: p_k is the number of elements in the sequence $\pi_k, \pi_{k+1}, \ldots, \pi_n$ that are less than or equal to π_k. For example $P[314652] = 312321$. P-sequences are useful for ranking permutations lexicographically [13]. Here, P-sequences are used to obtain the constant average time property of the generation algorithm.

Bauslaugh and Ruskey [2] noted that if $P[\pi'] > P[\pi]$ in lexicographic order, then $\pi' > \pi$ in lexicographic order. The same conclusion is not true for colex order; colex order of P-sequences does not correspond to colex order of permutations. The following lemma characterizes the P-sequences that represent permutations with a given signature.

Lemma 2.1. *Let $Q = (q_1, q_2, \ldots, q_{n-1})$ be a signature. The following conditions are necessary and sufficient for a P-sequence $p_1, p_2, \ldots, p_n$ to represent a permutation with signature Q:*

(a) $p_k > p_{k+1}$ *if* $q_k = -1$,
(b) $p_k \le p_{k+1}$ *if* $q_k = 1$.

Proof. If $q_k = -1$, then $\pi_k > \pi_{k+1}$, so clearly $p_k > p_{k+1}$. Also, if $q_k = 1$, then $\pi_k < \pi_{k+1}$, and so $p_k \le p_{k+1}$. The other direction of the proof is equally straightforward. □

We note that the largest value p_k can have is $n - k + 1$, the smallest is 1, and, by definition, $p_n = 1$. Lemma 2.1 can be used to determine the extreme values that p_k can have. The minima can be computed as follows:

$$l_k = \begin{cases} 1, & \text{if } k = n \text{ or } q_k = 1, \\ l_{k+1} + 1, & \text{otherwise,} \end{cases} \tag{1}$$

```
Procedure gen1(k, v: integer);
var i, lb, ub: integer;
begin
  P[k] := v;
  if k = 1 then PrintIt
  else begin
    if Q[k − 1] = 1 then begin
      lb := 1; ub := v;
    end else begin
      lb := v + 1; ub := n − k + 2;
    end;
    for i := lb to ub do gen1(k − 1, i);
  end;
end {of gen1};
```

Fig. 1. Pascal procedure to generate P-sequences in colex order.

and maxima can be computed as follows:

$$u_k = \begin{cases} n-k+1, & \text{if } k=n \text{ or } q_k=-1, \\ u_{k+1}, & \text{otherwise.} \end{cases} \tag{2}$$

These minima and maxima are used for initialization in the constant average time generation algorithm and in the ranking and unranking algorithms.

The generation algorithm presented here is recursive and depends on Lemma 2.1. It is given as a Pascal procedure $gen1(k, v)$ in Fig. 1. Array P holds the P-sequence and Q holds the signature. If $p_{k+1}, \ldots, p_n$ is a valid suffix of a P-sequence then $gen1(k, v)$ prints all P-sequences with suffix $p_k, p_{k+1}, \ldots, p_n$ where $p_k = v$. The call $gen1(n, 1)$ produces all valid P-sequences. It should be clear that algorithm $gen1$ will only generate valid P-sequences and that the generation is in colex order.

The problem with the algorithm presented in Fig. 1 is that its time complexity is dependent on the signature Q. If there are long runs where successive elements of Q are equal, then the algorithm is not constant average time. The easiest example is the signature $Q = (1, 1, 1, \ldots, 1, -1)$ for which the number of procedure calls is $\Omega(n^2)$ and there are only $n-1$ valid P-sequences. The problem arises because when $ub = lb$ the algorithm degenerates into a search for the next value of k where q_k is different from q_{k+1}. If we can some how speed up the search for k so that it takes O(1) time, then the algorithm given in Fig. 1 would become constant average time.

The algorithm $gen2$, presented in Fig. 2, eliminates the search for k. Because the elimination of the search also eliminates the initialization of part of the P-sequence,

```
Procedure gen2(k, lb, ub: integer);
var temp, v: integer;
begin
  if k < 1 then PrintIt
  else if lb = ub then
    if Q[ch[k]] = 1
      then gen2(ch[k], 1, P[ch[k] + 1])
      else gen2(ch[k], P[ch[k] + 1] + 1, n - ch[k] + 1)
    else begin
      temp := P[k];
      for v := lb to ub do begin
        P[k] := v;
        if Q[k - 1] = 1
          then gen2(k - 1, 1, v)
          else gen2(k - 1, v + 1, n - k + 2);
      end;
      P[k] := temp;
    end;
end {of gen2};
```

Fig. 2. Pascal procedure to generate P-sequences in colex order.

the P-sequence must be initialized to the values that it would contain when $lb=ub$ before the procedure *gen*2 is called. If $lb=ub$, then from (1) and (2) it follows that $p_k=1$ if $q_k=+1$ and $p_k=n-k+1$ if $q_k=-1$. These are the values we use to initialize p. Given that $p_{k+1},\dots,p_n$ is a valid suffix of a P-sequence, the call $gen2(k,lb,ub)$ generates all P-sequences with suffix $p_k,p_{k+1},\dots,p_n$ where $lb\le p_k\le ub$. The call $gen2(n,1,1)$ produces all valid P-sequences. An analysis of algorithm *gen*2 is given in the next section. The local variable *temp* is used to preserve the initialization. The array element $ch[k]$ is the greatest index $t<k$ such that $q_t\ne q_k$.

An example of the permutations and P-sequences for the signature $Q=(1,1,-1,1,-1)$ is given in Table 1. For this signature Q, the array $ch=(0,0,2,3,4)$.

3. Analysis

The proof that *gen*2 runs in constant average time uses the term call of degree n. By a *call of degree n* we mean that the procedure directly calls itself n times. As an example, the degree of a call to $gen2(k,lb,ub)$ is $ub-lb+1$ unless $k<1$. The analysis of the generation algorithm given in Fig. 2 is based on the results given in Ruskey and Roelants van Baronaigien [16] where they state that if (i) every call leads to the output of at least one object, (ii) all of the computation of a call not including additional recursive calls can be done in constant time, and (iii) the number of calls of the generation procedure that have degree one is on the order of the number of

Table 1. The P-sequences and permutations for $Q=(1,1,-1,1,-1)$.

Rank	P-sequence	Permutation	Rank	P-sequence	Permutation
0	112121	124365	18	444121	456132
1	122121	134265	19	113221	125463
2	222121	234165	20	123221	135462
3	113121	125364	21	223221	235461
4	123121	135264	22	133221	145261
5	223121	235164	23	233221	245361
6	133121	145263	24	333221	345261
7	233121	245163	25	114221	126453
8	333121	345162	26	124221	136452
9	114121	126354	27	224221	236451
10	124121	136254	28	134221	146251
11	224121	236154	29	234221	246351
12	134121	146253	30	334221	346251
13	234121	246153	31	144221	156342
14	334121	346152	32	244221	256341
15	144121	156243	33	344221	356241
16	244121	256143	34	444221	456231
17	344121	356142			

calls of higher degree, then the generation algorithm works in constant average time. Therefore, all we need to show is that the algorithm *gen*2 meets the conditions (i), (ii), (iii). All of the criteria except for (iii) are obvious from the algorithm. Note that in procedures *gen*1 and *gen*2 each call of degree zero causes the output of one P-sequence.

Lemma 3.1. *The number of calls of gen2 of degree one is less than the number of calls of degree zero or degree greater than one.*

Proof. Clearly, if $lb \neq ub$, then the degree of the call is greater than 1. Let $lb = ub$. We must consider the cases $Q[k]=1$ and $Q[k]=-1$. Let $Q[k]=-1$. There are two subcases to consider; either there is a sign change between positions 1 and k or there is not. If there is no change, then the string is output by a call of degree zero. If there is a change, then the next call is $gen2(ch[k], 1, P[ch[k]+1])$ which has degree $P[ch[k]+1]$. Since $Q[ch[k]+1]=Q[k]=-1$, it follows that $P[ch[k]+1]= n-(ch[k]+1)+1=n-ch[k]>1$. A similar argument can be given for the case where $Q[k]=1$. □

4. Ranking and unranking

In this section, an algorithm is presented for ranking of the P-sequences of permutations with a given signature. Unranking is also discussed. The method for ranking combinatorial objects in lexicographic order is well known. See Wilf [17], Bauslaugh and Ruskey [2], Zaks and Richards [21], and Williamson [19].

Because we list P-sequences in colex order, the ranking algorithm requires that we compute the number of P-sequences that end with a given subsequence instead of computing the number of P-sequences that start with a given subsequence. These numbers can be easily computed but they are dependent on the signature Q. Throughout this section we assume that Q is fixed.

Definition 4.1. For $l_k \leq v \leq u_k$ we define $C(k,v)$ to be the number of different sequences $p_1, p_2, \ldots, p_{k-1}, v$ that are a prefix of a P-sequence of a permutation with signature Q.

Lemma 4.2. *The numbers $C(k,v)$ satisfy the boundary conditions $C(k,v)=0$ if $v<l_k$ or $v>u_k$, $C(k,v)=1$ if $k=1$, and the following recurrence relation:*

$$C(k,v) = \begin{cases} \sum_{j=1}^{v} C(k-1,j), & \text{if } q_{k-1}=+1, \\ \sum_{j=v+1}^{n-k+1} C(k-1,j), & \text{if } q_{k-1}=-1. \end{cases} \tag{3}$$

Table 2. The numbers $C(k,v)$ for $Q=(1,1,-1,1,-1)$.

	1	2	3	4	5	6
1	1	1	1	1	0	0
2	1	2	3	4	0	
3	0	3	6	10		
4	19	16	0			
5	0	35				
6	35					

Proof. The proof follows directly from Lemma 2.1 and the definitions of P-sequence, l_k, u_k and $C(k,v)$. □

Considering the definition of the $C(k,v)$ numbers, it is clear that the rank of a P-sequence $p_1,p_2,\dots,p_n$ is given by the following equation:

$$rank(p_1,p_2,\dots,p_n) = \sum_{k=1}^{n-1} r(k),$$

where $r(k)$ is defined to be

$$r(k) = \begin{cases} \sum_{j=1}^{p_k-1} C(k,j), & \text{if } q_k=+1, \\ \sum_{j=p_{k+1}+1}^{p_k-1} C(k,j), & \text{if } q_k=-1. \end{cases}$$

If we consider the recurrence relation (3) for the $C(k,v)$ numbers the formula for $r(k)$ can be rewritten as follows:

$$r(k) = \begin{cases} C(k+1,p_{k+1})-C(k,p_k), & \text{if } q_k=-1,\ p_k+k>n \\ & \text{and } p_k>p_{k+1}+1, \\ C(k+1,p_{k+1})-C(k+1,p_k-1), & \text{if } q_k=-1,\ p_k+k\le n \\ & \text{and } p_k>p_{k+1}+1, \\ C(k+1,p_k-1), & \text{if } q_k=1 \text{ and } p_k>1, \\ 0, & \text{otherwise.} \end{cases}$$

As an example, we compute the rank of the P-sequence $P=344121$ where $Q=(1,1,-1,1,-1)$ (see Table 2). The values for $r(k)$ are as follows: $r(1)=C(2,2)=2$, $r(2)=C(3,3)=6$, $r(3)=C(4,1)-C(3,4)=9$, $r(4)=0$ and $r(5)=0$ so the rank of P with signature Q is 17.

Using the second formula for $r(k)$ we have an $O(n)$ algorithm for computing the rank of the P-sequence of a permutation with a given signature (given that the numbers $C(k,v)$ have been precomputed). The algorithm for unranking is similar to unranking algorithms given for other combinatorial objects [14,17]. For this reason the unranking algorithm is left to the reader.

5. Final remarks

We have given a constant average time algorithm for generating permutations with a given signature Q. Unlike most generation algorithms, this one skips across parts of the sequence that have been initialized to the values that would be put there during a scanning phase. The idea of skipping over redundant processing is a natural one and has been used before in connection with algorithms for generating combinatorial objects. An example may be found in Dershowitz [4]. In our case the algorithm is improved from having a time complexity that is dependent on the signature Q to being a constant average time algorithm.

Some open problems remain. It is not known whether the permutations, as opposed to the P-sequences, with a given signature can be generated in constant average time or, if either the *P-sequences* or the permutations can be generated by a loopless algorithm. Loopless algorithms for permutations, combinations and partitions are presented in Ehrlich [5].

References

[1] M. Abramson, A simple solution of Simon Newcomb's problem, J. Combin. Theory Ser. A 18 (1975) 223-225.
[2] B. Bauslaugh and F. Ruskey, Generating alternating permutations lexicographically, BIT, to appear.
[3] N.G. De Bruijn, Permutations with given ups and downs, Nieuw Arch. Wisk. 18 (1970) 61-65.
[4] N. Dershowitz, A simplified loop-free algorithm for generating permutations, BIT 15 (1975) 158-164.
[5] G. Ehrlich, Loopless algorithms for generating permutations, combinations, and other combinatorial configurations, J. ACM 20 (1973) 500-513.
[6] H.O. Foulkes, Enumeration of permutations with prescribed up-down and inversion sequences, Discrete Math. 15 (1976) 235-252.
[7] A.D. Kalvin and Y.L. Varol, On the generation of all topological sortings, J. Algorithms 4 (1983) 150-162.
[8] A.D. Kalvin and Y.L. Varol, On two problems reducible to topological sorting, Comput. J. 27 (1984) 176-177.
[9] D.E. Knuth, Sorting and Searching (Addison-Wesley, Reading, MA, 1973).
[10] A. Nijenhuis and H.S. Wilf, Combinatorial Algorithms (Academic Press, New York, 2nd ed., 1978).
[11] I. Niven, A combinatorial problem on finite sequences, Nieuw Arch. Wisk 16 (1968) 116-123.
[12] G. Pruesse and F. Ruskey, Generating the linear extensions of certain posets by transpositions, SIAM J. Discrete Math. 4 (1991) 413-422.
[13] E.M. Reingold, J. Nievergelt and N. Deo, Combinatorial Algorithms: Theory and Practice (Prentice-Hall, Englewood Cliffs, NJ, 1977).

[14] D. Roelants van Baronaigien and F. Ruskey, Generating t-ary trees in A-order, Inform. Process. Lett. 27 (1988) 205–213.

[15] F. Ruskey, Transposition generation of alternating permutations, Order 6 (1989) 227–233.

[16] F. Ruskey and D. Roelants van Baronaigien, Fast recursive algorithms for generating combinatorial objects, Congr. Numer. 41 (1984) 53–62.

[17] H.S. Wilf, A unified setting for sequencing, ranking and selection algorithms for combinatorial objects, Adv. in Math. 24 (1977) 281–291.

[18] H.S. Wilf, A unified setting for selection algorithms (II), in: Annals of Discrete Mathematics 2 (North-Holland, Amsterdam, 1978) 135–148.

[19] S.G. Williamson, On the ordering, ranking and random generation of basic combinatorial sets, in: Lecture Notes in Mathematics 579 (Springer, Berlin, 1976) 311–339.

[20] S.G. Williamson, Combinatorics for Computer Science (Computer Science Press, Rockville, MD, 1985).

[21] S. Zaks and D. Richards, Generating trees and other combinatorial objects lexicographically, SIAM J. Comput. 8 (1979) 73–81.