LOOPLESS GRAY CODE ALGORITHMS

T. A. Jenkyns
Department of Mathematics
Brock University, St. Catharines, Ontario
e-mail: tjenkyns@spartan.ac.brocku.ca

Abstract

All (non-empty) subsets of {1..n} may be represented as increasing sequences of varying length. In this form, they are easily generated in lexicographic order by elegant and compact algorithms. A second total order on the integer sequences, *alternating-lexocographic* order, is specified where each increasing sequence differs from its predecessor by the addition of a single new element or deletion of a single old element. Such minimal change listings are known as *Gray codes*. Using this order, new *loopless* algorithms for generating all subsets, all $\leq$k-subsets, all k-subsets, and all subsets of sizes min to max are given.

§1 Introduction

The elements of each non-empty subset of {1..n} may be written in order and the subset uniquely represented by this increasing sequence. Let U(a,b) denote the set of all such sequences S where $a \leq$ length(S) $\leq b$. This paper concerns algorithms for generating all members of U(a,b) and the three special cases of U(1,n), U(1,k) and U(k,k) in some order in a list.

In [1] Gideon Ehrlich formalized the notion of a *loopless* algorithm for this task as one where, after the initial sequence is generated, each succeeding sequence must be generated by at most a fixed number, K, of operations where K is independent of n.

If a generating algorithm is loopless, any succeeding sequence cannot differ from its predecessor in more than K entries. When the number of changes is consistently as small as possible, the list of sequences is known as a *Gray code* (see Wilf [5] p. 7).

The next section specifies two of the many orders in which the subsets may be generated. Section 3 contains algorithms for generating them in lexicographic order. Because this order is so natural, algorithms using it are reinvented frequently and variations have appeared in the literature (see [3] and [4]), perhaps beginning with [2]. Sections 4 and 5 discuss alternating-lexicographic order and the final section concludes with new, loopless algorithms for generating them in that order as Gray codes. All the algorithms given are quite simple, efficient and use no extra storage beyond the current sequence (S), its length (k), and sometimes, a single auxiliary index (j).

§2 Lexicographic Order and Alternating-Lexicographic Order

Let **X** be the set of non-empty, finite sequences of positive integers, and let

$$x = (x_1, x_2, \dots , x_m) \text{ and } y = (y_1, y_2, \dots , y_n)$$

be elements of **X**. Define two relations, α and β, on **X** by:

x α y if and only if

(a) there is an index $i \leq m,n$ where $x_i < y_i$ but if $1 \leq j < i$ then $x_j = y_j$

or (b) $x_i = y_i$ for $i=1,2, \dots ,m \leq n$;

$x \,\beta\, y$ if and only if

(c) there is an index $i \leq m,n$ where $x_i \neq y_i$ but if $1 \leq j < i$ then $x_j = y_j$,
and either (i) i is odd and $x_i < y_i$
or (ii) i is even and $x_i > y_i$,

or (d) $x_i = y_i$ for $i=1,2, \ldots ,m \leq n$ and m is odd,
or (e) $x_i = y_i$ for $i=1,2, \ldots ,n \leq m$ and n is even.

Both these relations are total (linear) orders on **X**, the first is the usual lexicographic order and we shall refer to the second as *alternating-lexicographic order.*

§3 Algorithms for subsets in lexicographic order

The algorithms, here and in §6, are given in pseudocode without parameters or declarations and, each contains calls in the form "Process(S[1]...S[k])" to a procedure which receives the subsets as they are generated for some unspecified purpose.

```
Algorithm  L1  (* generates non-empty subsets of {1..n} *)
               (* as increasing sequences S[1],...,S[k] *)
               (* in Lexicographic order                *)

  BEGIN    S[1]:=1;  k:=1;  Process( S[1] );
     WHILE S[1] < n DO
        IF S[k] = n  THEN  S[k-1]:=S[k-1]+1;  k:=k-1
                     ELSE  S[k+1]:=S[ k ]+1;  k:=k+1 END;
        Process( S[1],...,S[k] )
     END
  END Algorithm L1.
```

The successor of a sequence is obtained either by adding one new entry at the end, or increasing the second last entry by one <u>and</u> removing the last entry, n. The latter is not a minimal change so L1 is loopless but does not produce a Gray code.

```
Algorithm  L2  (*  generates all k-subsets of {1..n} *)
               (*  where 1 ≤ k ≤ max   in Lex Order  *)

  BEGIN    S[1]:=1;  k:=1;  Process( S[1] );
     WHILE S[1] < n DO
        IF S[k] = n THEN  S[k-1]:=S[k-1]+1;  k:=k-1
        ELSE
           IF k < max THEN  S[k+1]:=S[k]+1;  k:=k+1
                      ELSE  S[k]  :=S[k]+1 END;
        END;
        Process( S[1],...,S[k] )
     END
  END Algorithm L2.
```

The following two algorithms contain an unavoidable loop (the inner for-statement) but are included for comparison with later loopless algorithms, and because they avoid the inner for-statement whenever possible, an improvement on previously published versions.

```
Algorithm  L3  (*  generates all k-subsets of {1..n}      *)
               (*  as increasing sequences in Lex Order  *)
               (*  j=k or j=first index where S[j]=n-k+j *)

  BEGIN  FOR i:=1 TO k DO S[i]:=i END;
     Process( S[1],...,S[k] );
     j:=k;
     WHILE S[1] < n-k+1 DO
        IF S[k] < n THEN  S[k]:=S[k]+1
        ELSE  S[j-1]:=S[j-1]+1;
           IF S[j-1] = S[j]-1 THEN j:=j-1
           ELSE
             FOR i:=j TO k DO S[i]:=S[i-1]+1 END;
             j:=k
           END
        END;
        Process( S[1],...,S[k] )
     END
  END Algorithm L3.
```

```
Algorithm  L4  (*  generates all k-subsets of {1..n}        *)
               (*  where min ≤ k ≤ max in Lex Order         *)
               (*  j=min or first index where S[j]=n-min+j *)

  BEGIN  FOR i:=1 TO min DO S[i]:=i END;  k:=min;
     Process( S[1],...,S[k] );
     j:=min;
     WHILE S[1] < n-min+1 DO
        IF S[min] < n THEN
           IF S[k] = n THEN S[k-1]:=S[k-1]+1; k:=k-1
           ELSE
             IF k<max THEN S[k+1]:=S[k]+1;  k:=k+1
                      ELSE S[k]  :=S[k]+1 END
           END
        ELSE  S[j-1]:=S[j-1]+1;
           IF S[j-1] = S[j]-1 THEN j:=j-1
           ELSE
             FOR i:=j TO min DO S[i]:=S[i-1]+1 END;
             j:=min
           END
        END;
        Process( S[1],...,S[k] )
     END
  END Algorithm L4.
```

§4 Alternating-lexicographic order of subsets of {1..n}

For n = 4 the 15 sequences corresponding to non-empty subsets of {1..n} have the following alternating-lexicographic order.

```
 1.        1
 2.        1 4
 3.        1 3 4
 4.        1 3
 5.        1 2 3
 6.        1 2 3 4
 7.        1 2 4
 8.        1 2
 9.        2
10.        2 4
11.        2 3 4
12.        2 3
13.        3
14.        3 4
15.        4
```

The first sequence in the order is always (1) and the last is always (n).

If S = (S[1]...S[k-1],S[k]) is not the last sequence, its successor in the order, S', may be obtained as follows:

(a) If k is odd
  and if S[k] < n then S' = (S[1]...S[k-1],S[k], n)
  but if S[k] = n then S' = (S[1]...S[k-1]);
{Either n is added at the end of the sequence, or n is removed from the end.}

(b) If k is even
  and if S[k] = S[k-1]+1 then S' = (S[1]...S[k-2],S[k])
  but if S[k] > S[k-1]+1 then S' = (S[1]...S[k-2],S[k-1],S[k]-1,S[k]).
{Either the integer S[k]-1 is removed from position k-1 or inserted at position k.}

Thus the alternating-lexicographic order of these sequences is a Gray code.

§5 Alternating-lexicographic order of k-subsets of {1..n}

Each k-subset corresponds to an increasing k-sequence S such that when $1 \le i < i+j \le k$

$S[i+1] \ge S[i]+1$ and more generally $S[i+j] \ge S[i]+j$, so

$i = 1+(i-1) \le S[1]+(i-1) \le S[i] \le S[k]-(k-i) \le n-k+i$;

if $S[i+j] < n-k+i+j$ then $S[i] \le S[i+j]-j < n-k+i$, and

if $S[i] = n-k+i$ then $S[i+j] = n-k+i+j$.

For n=5 and k=3 the 10 sequences corresponding to k-subsets of {1..n} have the following alternating-lexicographic order.

```
 1.        1 4 5
 2.        1 3 4
 3.        1 3 5
 4.        1 2 3
 5.        1 2 4
 6.        1 2 5
 7.        2 4 5
 8.        2 3 4
 9.        2 3 5
10.        3 4 5
```

In general, S is the last k-sequence in the order if and only if S[1] = n-k+1.

Suppose S is not the last increasing k-sequence. Let p be the largest index where S[p] < n-k+p. We shall associate with S the integer j where: if p is even then j = p, but if p is odd then j = p+1; so j is always even and may equal k+1 when k is odd.

If S is not the last k-sequence, its successor in the order, S', with its j-value, j', may be obtained as follows:

<u>Case 1.</u> Suppose j=k+1. Then k is odd and S[k] < n, so S[k] may be increased and hence

S' = (S[1]...S[k-1], S[k]+1).

If the new S[k] < n then j'=j but if the new S[k] = n then j' = k-1 = j-2.
(If j' is zero then k=1 and S' is the last in the order.)

<u>Case 2.</u> Suppose j=k. Then k is even, k $\geq$ 2 and S[k] < n.

If S[k] > S[k-1]+1 then S[k] may be decreased and hence

S' = (S[1]...S[k-1], S[k]-1) and j'=k=j.

If S[k] = S[k-1]+1 then S[k] cannot be decreased and hence

S' = (S[1]...S[k-2], S[k], n)

since S[k-1] may be increased (by 1 to S[k]) and then the following *even* entry reset to the 'earliest' possible value it might take.
If the new S[k-1] < n-1 then j'=k=j but if the new S[k-1] = n-1 then j' = k-2 = j-2.
(If j' is zero then k=2 and S' is the last in the order.)

Case 3. Suppose $j < k$. Then for $j < i \leq k$, $S[i] = n-k+i$ and the *odd* entry $S[j+1]$ cannot be increased. In fact, no such entry $S[i]$ can be altered unless some entry at or before position j is changed.

If $S[j] > S[j-1]+1$ then $S[j]$ may be decreased (by 1) and the following *odd* entry, $S[j+1]$, reset to the 'earliest' possible value it might take, namely $S[j]$. All following entries, beginning with the *even* entry $S[j+2]$, are already at their 'earliest' possible values. Hence

$$S' = (\ S[1]...S[j-1],\ \ S[j]-1,\ \ S[j],\ \ S[j+2]...S[k]\ ) \quad \text{and } j'=j+2.$$

If $S[j] = S[j-1]+1$ then $S[j]$ cannot be decreased but the *odd* entry $S[j-1]$ may be increased (by 1 to $S[j]$) and then the following *even* entry reset to the 'earliest' possible value it might take. Again all following entries, beginning with $S[j+1]$, are already at their 'earliest' possible value. Hence

$$S' = (\ S[1]...S[j-2],\ \ S[j],\ \ n-k+j,\ \ S[j+1]...S[k]\ ).$$

If the new $S[j-1] < n-k+j-1$ then $j'=j$ but if the new $S[j-1] = n-k+j-1$ then $j' = j-2$. (If j' is zero then S' is the last in the order.) .

In every case S' is obtained from S by removing one entry and inserting one entry so the alternating lexicographic order is a *minimal change* list; i.e., a Gray code. In fact, either the last entry is changed up or down by one, or two adjacent entries are changed.

§6 Loopless Algorithms for subsets in alternating-lexicographic order

```
Algorithm  A1  (* generates non-empty subsets of {1..n}  *)
               (* as increasing sequences in a Gray code *)

  BEGIN  S[1]:=1;  k:=1;  Process( S[1] );
     WHILE S[1] < n DO
        IF S[k] = n THEN   k:=k-1
        ELSE   S[k+1]:=n;  k:=k+1  END;
        Process( S[1],...,S[k] );
        IF S[k-1] = S[k]-1 THEN S[k-1]:=S[k];  k:=k-1
        ELSE     S[k+1]:=S[k];  S[k]:=S[k]-1;  k:=k+1 END;
        Process( S[1],...,S[k] )
     END
  END Algorithm A1.
```

The cases of odd and even k alternate and both have been put into the body of the loop.

The bit-strings corresponding to the subsets generated here are precisely the 'standard reflected (binary) Gray code' (see [5]) produced in reverse order, if the all zero string is added at the end.

Algorithm A2 (* generates all k-subsets of {1..n} *)
(* where 1 ≤ k ≤ max in a Gray code *)

```
 BEGIN S[1]:=1;  k:=1;  Process( S[1] );
    WHILE S[1] < n DO
       IF ((k MOD 2) > 0 THEN           (* k is odd  *)
          IF S[k] = n THEN k:=k-1
          ELSE IF k < max THEN S[k+1]:=n;  k:=k+1
               ELSE S[k]:=S[k]+1 END
          END
       ELSE                             (* k is even *)
          IF S[k-1] = S[k]-1 THEN S[k-1]:=S[k];  k:=k-1
          ELSE S[k]:=S[k]-1;
             IF k < max THEN S[k+1]:=S[k]+1; k:=k+1 END
          END
       END;
       Process( S[1],...,S[k] )
    END
END Algorithm A2.
```

Algorithm A3 (* generates all k-subsets of {1..n} *)
(* as increasing sequences in a Gray code *)

```
 BEGIN S[1]:=1;  FOR i:=2 TO k DO S[i]:=n-k+i END;
    Process( S[1],...,S[k] );   j:=2;
    WHILE S[1] < n-k+1 DO
       IF j > k THEN S[k]:=S[k]+1;
          IF S[k] = n THEN j:=j-2 END
       ELSE
          IF S[j-1] = S[j]-1 THEN
             S[j-1]:= S[j];  S[j]:=n-k+j;
             IF S[j-1]=S[j]-1 THEN j:=j-2 END
          ELSE S[j]:=S[j]-1;
             IF j < k THEN S[j+1]:=S[j]+1;  j:=j+2 END
          END
       END;
       Process( S[1],...,S[k] )
    END
END Algorithm A3.
```

```
Algorithm  A4  (*  to generate all k-subsets of {1..n}  *)
               (*  where min ≤ k ≤ max  in a Gray code  *)

  BEGIN S[1]:=1;  k:=min;
     IF min=1 THEN j:=1
     ELSE FOR i:=2 TO min DO S[i]:=n-min+i END; j:=2
     END;
     Process( S[1],...,S[k] );
     WHILE S[1] < n-min+1 DO
        IF ((j MOD 2) > 0 THEN           (* j is odd  *)
           IF S[j] = n THEN j:=j-1
           ELSE IF j < max THEN  S[j+1]:=n;  j:=j+1
                ELSE S[j]:=S[j]+1;
                     IF S[min]=n THEN j:=j-1 END
                END
           END
        ELSE                             (* j is even *)
           IF S[j-1] = S[j]-1 THEN S[j-1]:= S[j];
              IF j > min THEN
                 IF S[min] = n THEN j:=j-2 ELSE j:=j-1 END
              ELSE S[j]:=n-min+j;
                 IF S[j-1]=S[j]-1 THEN j:=j-2 END
              END
           ELSE S[j]:=S[j]-1;
              IF j < max THEN S[j+1]:=S[j]+1;
                 IF j >= min-1 THEN j:=j+1 ELSE j:=j+2 END
              END
           END
        END;
        IF j<min THEN k:=min ELSE k:=j END;
        Process( S[1],...,S[k] )
     END
 END Algorithm A4.
```

This algorithm can be somewhat simplified when 1<min and/or min<max.

References

[1] Ehrlich, G., "Loopless Algorithms for Generating Permutations, Combinations and Other Combinatorial Configurations", *J. ACM* 20, 3 (July 1973), 500-513.

[2] Mifsud, C.J., "Algorithm 154. Combination in Lexicographical Order", *Comm. ACM* 6, (Mar 1963), 103.

[3] Nijenhuis, A. and Wilf, H. S., *Combinatorial Algorithms* , Academic Press, 1978.

[4] Reingold, E. M., Nievergelt, J. and Deo, N.,*Combinatorial Algorithms*, Prentice-Hall, 1977.

[5] Wilf, H. S., *Combinatorial Algorithms: An Update,* SIAM CBMS-55, 1989.