Simple combinatorial Gray codes constructed by reversing sublists

Frank Ruskey[1]

Department of Computer Science, University of Victoria,
Victoria, B.C., V8W 3P6, Canada, e-mail:fruskey@csr.uvic.ca

Abstract. We present three related results about simple combinatorial Gray codes constructed recursively by reversing certain sublists. First, we show a bijection between the list of compositions of Knuth and the list of combinations of Eades and McKay. Secondly, we provide a short description of a list of combinations satisfying a more restrictive closeness criteria of Chase. Finally, we develop a new, simply described, Gray code list of the partitions of a set into a fixed number of blocks, as represented by restricted growth sequences. In each case the recursive definition of the list is easily translatable into an algorithm for generating the list in time proportional to the number of elements in the list; i.e., each object is produced in $O(1)$ amortized time by the algorithm.

1 Introduction

Frank Gray patented the Binary Reflected Gray Code (BRGC) in 1953 for use in "pulse code communication", but the underlying construction of the code existed for centuries as the solution of a puzzle that is known in the West as the "Chinese Rings". The BRGC has since found use in the minimization of logic circuits, in hypercube architectures, in the construction of multikey hashing functions, and has even been proposed for use in the organization of books on library shelves! For the purposes of this paper, we regard the BRGC as a way of listing all 2^n bitstrings of length n so that successive strings differ by a single bit or, equivalently, as a way of listing all subsets of an n-set so that successive subsets differ by a single element.

It is natural to take a more abstract view and look for listings of other combinatorial objects so that successive objects differ by a very small amount; such lists have come to be known as *combinatorial Gray codes.* In this paper, we shed new light on some previously known lists of combinations and compositions, and develop new lists for combinations and set partitions. In each case the list has a very simple recursive description. The simplicity of the recursion is mirrored in simple proofs and simple efficient algorithms for generating the lists. These algorithms generate the objects so that only a constant amount of computation is done between successive objects, in an amortized sense.

* Research supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant A3379.

There is a well-known simple recursive construction for the BRGC. This construction involves the concatenation of two recursively defined smaller sublists, one of which is reversed, as shown below.

$$\mathbf{B}(n) = \begin{cases} \emptyset & \text{if } n = 0 \\ \mathbf{B}(n-1)\cdot 0 \circ \overline{\mathbf{B}(n-1)}\cdot 1 & \text{if } n > 0 \end{cases}$$

This simple recursive construction is easily proven to be correct and can be implemented in an efficient manner to actually generate the list. In the construction of other combinatorial Gray codes, it is natural to try to invent similar recursive definitions. In general, the recursion defining the combinatorial Gray code should mirror some simple recursion for counting the combinatorial objects, in this case $a_n = 2a_{n-1}$ with $a_0 = 1$.

Very simple constructions have been obtained in the cases of combinations (e.g., Reingold, Nievergelt, and Deo [ReNiDe] or Eades and McKay [EaMc]), compositions (e.g., Wilf [Wi]), and well-formed parentheses (e.g., Ruskey and Proskurowski [RuPr]). Less simple recursive constructions, that are nevertheless based on reversing sublists, have been obtained for numerical partitions (Savage [Sa]), and set partitions (Fill and Reingold [FiRe], Ehrlich [Eh]).

We regard the objects being listed as strings over some alphabet. If L is a list of strings and x is a symbol, then $L \cdot x$ denotes the list of strings obtained by appending an x to each string of L. For example if $L = 01, 10$ the $L \cdot 1 = 011, 101$. If L and L' are lists then $L \circ L'$ denotes the concatenation of the two lists. For example, if $L = 01, 10$ and $L' = 11, 00$, then $L \circ L' = 01, 10, 11, 00$.

For a list L, let $first(L)$ denote the first element on the list and let $last(L)$ denote the last element on the list L. If L is a list $l_1, l_2, \ldots, l_n$, then $\overline{L}$ denotes the list obtained by listing the elements of L in reverse order; i.e., $\overline{L} = l_n, \ldots, l_2, l_1$. Note the obvious equations $first(\overline{L}) = last(L)$ and $last(\overline{L}) = first(L)$.

By a *k-combination* of n we mean a bitstring of length n with exactly k ones. The set of all k-combinations of n is denoted $\mathbf{B}(n, k)$; i.e.,

$$\mathbf{B}(n,k) \;=\; \{b_1 b_2 \cdots b_n \mid b_i \in \{0,1\}, \Sigma b_i = k\}$$

By a *k-composition* of n we mean a solution $(x_1, x_2, \ldots, x_k)$, in non-negative integers, to the equation $x_1 + x_2 + \cdots + x_k = n$.

A *partition* of $[n] = \{1, 2, \ldots, n\}$ into k *blocks* is a collection of k non-empty disjoint sets $B_0, B_1, \ldots, B_{k-1}$ whose union is $[n]$. Assuming that the B_i are ordered by their smallest elements, a convenient representation of the partition is as a sequence $x_1 x_2 \cdots x_n$ where $x_i = j$ if $x_i \in B_j$. For example, the sequence corresponding to the partition $\{1,3,4\}, \{2,6\}, \{5,7\}$ is 0100212. Such sequences are known as *restricted growth sequences* (or RG sequences) in the combinatorial literature (e.g., Milne [Mi]) and are characterized by the following property, which explains the name. If $i = 1$ then $x_i = 0$, and if $i > 1$, then

$$0 \le x_i \le 1 + \max\{x_1, x_2, \ldots, x_{i-1}\}.$$

The number of block in the partition is one less than the maximum value that occurs in the RG sequence.

In this note we will omit all proofs. In each case they are straightforward inductive arguments. The real difficulties lie in coming up with the recursive definitions and the statements of the lemmata.

2 The equivalence of two well-known lists

The following recursively defined list of k-combinations of n is due to Eades and McKay [EaMc].

$$\mathbf{C}(n,k) = \begin{cases} 0^n & \text{if } k = 0 \\ 10^{n-1} \circ 010^{n-2} \circ \cdots \circ 0^{n-1}1 & \text{if } k = 1 \\ \mathbf{C}(n-1,k)\cdot 0 & \\ \quad \circ\overline{\mathbf{C}(n-2,k-1)}\cdot 01 & \\ \quad \circ\mathbf{C}(n-2,k-2)\cdot 11 & \text{if } 1 < k < n \\ 1^n & \text{if } k = n \end{cases} \tag{1}$$

This list has a number of interesting properties which are embodied in the next lemma.

Lemma 1. *The list* $\mathbf{C}(n,k)$ *satisfies the following properties.*

1. *Successive combinations differ by a transposition of two bits and all bits between those that are transposed are zeroes.*
2. $first(\mathbf{C}(n,k)) = 1^k 0^{n-k}$.
3. $last(\mathbf{C}(n,k)) = 0^{n-k} 1^k$.

Another list, this one of k-compositions of n, is attributed in Wilf [Wi] to Donald Knuth. Consider the list, $\mathbf{Comp}(n,k)$, of all k-compositions of n, defined as follows. If $k = 1$, then $\mathbf{Comp}(n,1) = n$ and if $k > 1$, then $\mathbf{Comp}(n,k)$ is given by list (a) below.

$$\begin{array}{cc} \mathbf{Comp}(n,k-1)\cdot 0\circ & \mathbf{Comb}(n,k-1)\cdot 01^0\circ \\ \overline{\mathbf{Comp}(n-1,k-1)}\cdot 1\circ & \overline{\mathbf{Comb}(n-1,k-1)}\cdot 01^1\circ \\ \mathbf{Comp}(n-2,k-1)\cdot 2\circ & \mathbf{Comb}(n-2,k-1)\cdot 01^2\circ \\ \overline{\mathbf{Comp}(n-3,k-1)}\cdot 3\circ & \overline{\mathbf{Comb}(n-3,k-1)}\cdot 01^3\circ \\ \vdots & \vdots \\ \mathbf{Comp}(0,k-1)\cdot n & \mathbf{Comb}(0,k-1)\cdot 01^n \\ \text{(a)} & \text{(b)} \end{array}$$

The basic properties of this list are embodied in the next lemma.

Lemma 2. *The list* $\mathbf{Comp}(n,k)$ *satisfies the following properties.*

1. *Successive compositions differ by one in exactly two positions.*
2. $first(\mathbf{Comp}(n,k)) = n0^{k-1}$.
3. $last(\mathbf{Comp}(n,k)) = 0^{k-1}n$.

Every k-composition of n corresponds to a element of $\mathbf{B}(n+k-1, n)$. This correspondence is obtained by turning each x_i into a run of x_i 1's and separating the k runs of 1's by $k-1$ 0's. For example $300123 \in \mathbf{Comp}(9,6)$ corresponds to $11100010110111 \in B(14,9)$. Applying this correspondence to the list definition (a) yields the list $\mathbf{Comb}(n,k)$ of elements of $\mathbf{B}(n+k-1,n)$ as shown in (b). The following lemma states that the Eades-McKay and Knuth lists are essentially identical.

Lemma 3. $\mathbf{Comb}(n,k) = \mathbf{C}(n+k-1,n)$.

3 A more restrictive closeness condition

In the Eades-McKay list the two bits that are transposed could be quite far apart. The elements of $\mathbf{B}(n,k)$ cannot be listed so that successive strings differ by the transposition of two *adjacent* bits, unless n is even and k is odd, or $k \in \{0, 1, n-1, n\}$ (e.g., Buck and Wiedemann [BuWi], Eades, Hickey and Read [EaHiRe]). Let us say that two distinct bitstrings are *two-close* if they differ by a transposition of two bits that are either adjacent or have a single 0 between them. Chase [Ch] showed that the elements of $\mathbf{B}(n,k)$ could always be listed so that successive strings are two-close. He also presented a loopfree algorithm for producing such a listing. However, his construction takes far more explanation than the one we will now describe.

The two-close condition can be handled by the same list-reversing technique that we used previously, but we add an additional boolean parameter. Let $\mathbf{R}(n,k,1)$ denote a listing of the bitstrings in $\mathbf{B}(n,k)$ that starts at $0^{n-k-1}1^k0$ and ends at $0^{n-k}1^k$ and satisfying the two-close condition. Let $\mathbf{R}(n,k,0)$ be a similar list, except starting at $0^{n-k-2}1^k00$. The list $\mathbf{R}(n,k,1)$ exists only if $0 \le k < n$ and the list $\mathbf{R}(n,k,0)$ exists only if $0 \le k < n-1$. These lists may be defined recursively as shown below.

$$\mathbf{R}(n,k,p) = \begin{cases} 0^n & \text{if } k=0 \\ 1^{n-1}0 \circ \mathbf{R}(n-1,k-1,1) \cdot 1 & \text{if } p=1 \text{ and } k=n-1 \\ \overline{\mathbf{R}(n-1,k,1)} \cdot 0 & \\ \quad \circ \mathbf{R}(n-1,k-1,0) \cdot 1 & \text{if } p=1 \text{ and } 0<k<n-1 \\ \mathbf{R}(n-1,k,1) \cdot 0 & \\ \quad \circ \mathbf{R}(n-1,k-1,1) \cdot 1 & \text{if } p=0 \text{ and } 0<k<n-1 \end{cases}$$

These lists have a number of interesting properties which are embodied in the next lemma.

Lemma 4. *The lists* $\mathbf{R}(n,k,0)$ *and* $\mathbf{R}(n,k,1)$ *satisfy the following properties.*

1. *Successive bitstrings are two-close.*
2. $first(\mathbf{R}(n,k,0) = 0^{n-k-2}1^k00$
3. $first(\mathbf{R}(n,k,1) = 0^{n-k-1}1^k0$
4. $last(\mathbf{R}(n,k,0)) = last(\mathbf{R}(n,k,1)) = 0^{n-k}1^k$

4 Set partitions

Previous works have defined two set partitions to be *close* if they differ by taking an element from one block and moving it into another block. Gray codes under this criteria have been developed by Knuth (see Wilf [Wi]) for the set of all partitions, and by Ehrlich [Eh] and Hansche (see Fill and Reingold [FiRe]) for partitions into a fixed number of blocks. It is easy to see that two RG sequences that differ in one position have corresponding partitions that are close, but not vice-versa. We will show how to list RG sequences representing partitions into a fixed number of blocks so that successive sequences differ in exactly one position. Previous lists for partitions into a fixed number of blocks did not have this property, nor were they as simple as what we present here.

We define two lists of RG sequences representing partitions of $[n]$ into k blocks, denoted $\mathbf{S}(n,k,0)$ and $\mathbf{S}(n,k,1)$. These definitions have similarities to the $\mathbf{R}(n,k,\cdot)$ lists defined earlier. Successive RG sequences on these lists have the property that they differ in exactly one position.

Recall the basic recurrence relation for the Stirling numbers of the second kind $\left\{{n \atop k}\right\}$, which count partitions of $[n]$ into exactly k blocks.

$$\left\{{n \atop k}\right\} = \left\{{n-1 \atop k-1}\right\} + k\left\{{n-1 \atop k}\right\}$$

This recurrence relation may be proven in terms of RG sequences $x_1x_2\cdots x_n$ by classifying them according to the value of x_n. If $x_n = k-1$ then $x_1x_2\cdots x_{n-1}$ has largest element equal to $k-2$ or $k-1$. If $0 \le x_n < k-1$, then $x_1x_2\cdots x_{n-1}$ must have largest value equal to $k-1$.

The construction of the lists depends upon the parity of k as shown in the tables below, first for k even and then for k odd. If $k=1$ then $\mathbf{S}(n,1,0) = \mathbf{S}(n,1,1) = 0^n$ and otherwise use the tables below. Here's the lists for k even.

$\mathbf{S}(n,k,0)$ even k	$\mathbf{S}(n,k,1)$ even k
$\mathbf{S}(n-1,k-1,0)\cdot(k-1)\circ$	$\mathbf{S}(n-1,k-1,1)\cdot(k-1)\circ$
$\mathbf{S}(n-1,k,1)\cdot(k-1)\circ$	$\overline{\mathbf{S}(n-1,k,1)}\cdot(k-1)\circ$
$\overline{\mathbf{S}(n-1,k,1)}\cdot(k-2)\circ$	$\mathbf{S}(n-1,k,1)\cdot(k-2)\circ$
$\vdots$	$\vdots$
$\mathbf{S}(n-1,k,1)\cdot 1\circ$	$\overline{\mathbf{S}(n-1,k,1)}\cdot 1\circ$
$\overline{\mathbf{S}(n-1,k,1)}\cdot 0$	$\mathbf{S}(n-1,k,1)\cdot 0$

Here's the lists for k odd.

$\mathbf{S}(n,k,0)$ odd k	$\mathbf{S}(n,k,1)$ odd k
$\mathbf{S}(n-1,k-1,1)\cdot(k-1)\circ$	$\mathbf{S}(n-1,k-1,0)\cdot(k-1)\circ$
$\overline{\mathbf{S}(n-1,k,1)}\cdot(k-1)\circ$	$\mathbf{S}(n-1,k,1)\cdot(k-1)\circ$
$\mathbf{S}(n-1,k,1)\cdot(k-2)\circ$	$\overline{\mathbf{S}(n-1,k,1)}\cdot(k-2)\circ$
$\vdots$	$\vdots$
$\mathbf{S}(n-1,k,1)\cdot 1\circ$	$\overline{\mathbf{S}(n-1,k,1)}\cdot 1\circ$
$\overline{\mathbf{S}(n-1,k,1)}\cdot 0$	$\mathbf{S}(n-1,k,1)\cdot 0$

These lists have a number of interesting properties which are embodied in the following lemma.

Lemma 5. *The lists $\mathbf{S}(n,k,0)$ and $\mathbf{S}(n,k,1)$ satisfy the following properties.*

1. *Successive RG sequences differ in exactly one position.*
2. $first(\mathbf{S}(n,k,0)) = first(\mathbf{S}(n,k,1)) = 0^{n-k}012\cdots(k-1)$
3. $last(\mathbf{S}(n,k,0)) = 0^{n-k}12\cdots(k-1)0$
4. $last(\mathbf{S}(n,k,1)) = 012\cdots(k-1)0^{n-k}$

5 Algorithmic Considerations

We will now illustrate how the recursive definitions given earlier can be translated into efficient procedures for generating the objects. The proof of Lemma 4 leads to the consideration of the following table.

$$
\begin{array}{cc}
p=1 & p=0 \\
\hline
\end{array}
$$

$$
\left.\begin{array}{cc}
0^{n-k-2}01^{k-1}1\ \mathbf{0} & 0^{n-k-2}1^{k-1}10\ \mathbf{0} \\
\vdots & \vdots \\
0^{n-k-2}1^{k-1}\underline{1}0\ \underline{\mathbf{0}} & 0^{n-k-2}01^{k-1}\underline{1}\ \underline{\mathbf{0}}
\end{array}\right\}\binom{n-1}{k}
$$

$$
\left.\begin{array}{cc}
0^{n-k-2}1^{k-1}00\ \mathbf{1} & 0^{n-k-2}01^{k-1}0\ \mathbf{1} \\
\vdots & \vdots \\
0^{n-k-2}001^{k-1}\mathbf{1} & 0^{n-k-2}001^{k-1}\mathbf{1}
\end{array}\right\}\binom{n-1}{k-1}
$$

This table reveals that the transposition between the two sublists occurs in positions $n-2$ and n if $p=1$ and is in positions $n-1$ and n if $p=0$. This revelation in turn leads directly to the Pascal procedure `gen(n,k,p)` of Algorithm 1 which generates the list $\mathbf{C}(n,k,p)$. There is also a symmetric procedure `neg(n,k,p)` which generates the reversed list $\overline{\mathbf{C}(n,k,p)}$. The procedure `swap(i,j)` swaps the bits in positions i and j (and does whatever processing of the bitstring is required).

What is the running time of `gen(n,k,p)`? We need only observe the following.

```
procedure gen ( n,k,p : integer );
begin
  if p = 1 then begin
     if k = n-1 then begin
        for i := n-1 downto 1 do swap( i,i+1 );
     end else
     if (0 < k) and (k < n-1) then begin
        neg( n-1, k, 1 );  swap( n-2, n );  gen( n-1, k-1, 0 );
     end
  end else {p = 0} begin
     if (0 < k) and (k < n-1) then begin
        gen( n-1, k, 1 );  swap( n-1, n );  gen( n-1, k-1, 1 );
     end
  end;
end {of gen};
```

Fig. 1. Two-close generation of Combinations.

- Each recursive call to `gen` or `neg` can be charged to a call to `swap` so that any call to `swap` is charged at most twice.
- The total number of calls to `swap` is one less than $\binom{n}{k}$.
- The for loop makes no recursive calls, and each iteration of the for loop produces one new combination.

Thus total amount of computation is proportional to the number of recursive calls, which is $O(\binom{n}{k})$.

Analogous algorithmic development may be carried out for the other lists of this paper. In each case the proof reveals the change that occurs between successive sublists, and the costs of the recursive calls can be charged to the strings so that no string gets charged more than a constant amount.

References

[BuWi] M. Buck and D. Wiedemann, *Gray Codes with Restricted Density*, Discrete Mathematics, 48 (1984) 163-171.

[Ch] P.J. Chase, *Combination Generation and Graylex Ordering*, Congressus Numerantium, 69 (1989) 215-242.

[EaHiRe] P. Eades, M. Hickey and R. Read, *Some Hamilton Paths and a Minimal Change Algorithm*, Journal of the ACM, 31 (1984) 19-29.

[EaMc] P. Eades and B. McKay, *An Algorithm for Generating Subsets of Fixed Size with a Strong Minimal Change Property*, Information Processing Letters, 19 (1984) 131-133.

[Eh] G. Ehrlich, *Loopless Algorithms for Generating Permutations, Combinations and Other Combinatorial Configurations*, Journal of the ACM, 20 (1973) 500-513.

[FiRe] J.A. Fill and E.M. Reingold, *Solutions Manual for Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, 1977.

[Mi] S.C. Milne, *Restricted Growth Functions, Rank Row Matchings of Partition Lattices, and q-Stirling Numbers*, Advances in Mathematics, 43 (1982) 173-196.

[ReNiDe] E.M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.

[RuPr] F. Ruskey and A. Proskurowski, *Generating Binary Trees by Transpositions*, J. Algorithms, 11 (1990) 68-84.

[Sa] C.S. Savage, *Gray Code Sequences of Partitions*, J. Algorithms, 10 (1989) 577-595.

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