Exhaustive generation for ballot sequences in lexicographic and Gray code order

Ahmad Sabri
Department of Informatics
Gunadarma University, Depok, Indonesia
sabri@staff.gunadarma.ac.id

Vincent Vajnovszki
LE2I
Université Bourgogne Franche-Comté, France
vvajnov@u-bourgogne.fr

Abstract

A generalized ballot sequence is a sequence over the set of non-negative integers where in any of its prefixes each positive integer i occurs at most as often as any integer less than i. We show that the Reflected Gray Code order induces a 3-adjacent Gray code on the set of fixed length generalized ballot sequences (that is, Gray code where consecutive sequences differ in at most 3 adjacent positions). Non-trivial efficient generating algorithms for generalized ballot sequences, in both lexicographic and Gray code order, are also presented.

1 Introduction

A *ballot sequence* is a sequence defined over the alphabet $\{0, 1\}$, where in any of its prefixes the number of occurrences of the symbol 1 is at most the same as that of symbol 0. In this paper we consider a more general notion, namely *generalized ballot sequence* (or *Yamanouchi word*, see *e.g.* [Sta99, Prop. 7.10.3]): a sequence $\boldsymbol{s}$ over the alphabet of non-negative integers is said to be a generalized ballot sequence if for every prefix $\boldsymbol{s}'$ of $\boldsymbol{s}$, and for every i, the number of occurrences of i in $\boldsymbol{s}'$ is greater than or equal to the number of occurrences of $i + 1$ in $\boldsymbol{s}'$. Such a length n sequence encodes a ballot counting scenario in an election involving n candidates, in which during the counting progress, the number of votes collected by i-th candidate is always greater than or equal to those collected by $(i + 1)$-th candidate [Ber87, Ren08, Sag01]. The cardinality of the set of generalized ballot sequences of length n follows the integer sequence A000085 in [OEI].

Originally Gray codes appeared in the study of signal processing [GrA53], and here we adopt its definition from [Wal03]: a *Gray code* is an infinite set of sequence-lists with unbounded sequence-length, one list for each sequence-length, such that the number of distinct symbols between any two consecutive sequences (*i.e.*, the Hamming distance) in any list is bounded independently of the sequence-length. A *d-Gray code* is a Gray code where the Hamming distance between any two consecutive sequences is upper bounded by d. If the positions where the successive sequences differ are adjacent, then we say that the list is a *d-adjacent* Gray code. In addition, if the last sequence differs from the first one in the same way, then the Gray code is *cyclic*.

The original binary Gray code in [GrA53] was naturally generalized to Reflected Gray Code for arbitrary k-ary sequences by Er in [Er84]. Here we show that the restriction of the Reflected Gray Code to the set of length n generalized ballot sequences induces a 3-adjacent Gray code. Similar techniques based on variations of the order relation induced by the Reflected Gray Code was used implicitly, for example in [Kin81, Wal00], and developed

In: L. Ferrari, M. Vamvakari (eds.): Proceedings of the GASCom 2018 Workshop, Athens, Greece, 18–20 June 2018, published at http://ceur-ws.org

systematically as a general method in [Vaj10, BV05, Vaj01, Vaj07, Vaj08, Vaj11, Sab15, BBP15, Sab17], and our Gray code is in the light of this direction. In the second part of this paper we give constant amortized time exhaustive generating algorithms for generalized ballot sequences for both, lexicographic order and the obtained Gray code.

2 Notation and definitions

Through this paper, we denote a sequence of length n by an n-tuple (for instance, $s_1 s_2 \cdots s_n$), or by an italicized boldface letter (for instance, $\boldsymbol{s}$ and $\boldsymbol{t}$). For a given sequence $\boldsymbol{s}$, the notation $|\boldsymbol{s}|_i$ refers to the number of occurrences of the symbol i in $\boldsymbol{s}$. For example, if $\boldsymbol{s} = 1121313$, then $|\boldsymbol{s}|_1 = 4$, $|\boldsymbol{s}|_2 = 1$, and $|\boldsymbol{s}|_3 = 2$.

Definition 2.1. A length n *generalized ballot sequence* is an integer sequence $\boldsymbol{s} = s_1 s_2 \dots s_n$ over the set of non-negative integers with $s_1 = 0$ and $|\boldsymbol{s}'|_{i+1} \leq |\boldsymbol{s}'|_i$, for all i, $0 \leq i \leq n-1$, in any prefix $\boldsymbol{s}'$ of $\boldsymbol{s}$.

Notice that a non-empty prefix of a ballot sequence is still a (smaller length) ballot sequence and we denote by B_n the set of length n ballot sequences. See Table 1 for the set B_5.

Definition 2.2. Let $\boldsymbol{s} = s_1 s_2 \cdots s_n$ and $\boldsymbol{t} = t_1 t_2 \cdots t_n$ be two distinct integer sequences. Let k be the leftmost position where $\boldsymbol{s}, \boldsymbol{t}$ differ, and $u = \sum_{i=1}^{k-1} s_i = \sum_{i=1}^{k-1} t_i$. We say that $\boldsymbol{s}$ *precedes* $\boldsymbol{t}$ in *Reflected Gray Code order* (RGC order for short), denoted by $\boldsymbol{s} \prec \boldsymbol{t}$, if either

- u is even and $s_k < t_k$, or
- u is odd and $s_k > t_k$.

We denote by $\mathfrak{B}_n$ the list of all sequences in B_n with respect to RGC order. Actually, as it is noticed in [Vaj01], it is easy to see that for any $k \geq 2$ the set of unrestricted k-ary sequences of length n listed in RGC order yields a 1-Gray code (see again [Er84]).

Table 1: Generalized ballot sequences of length 5: (a) in lexicographic order, and (b) in RGC order together with the Hamming distance between consecutive sequences.

1	0 0 0 0 0	14	0 1 0 0 0
2	0 0 0 0 1	15	0 1 0 0 1
3	0 0 0 1 0	16	0 1 0 0 2
4	0 0 0 1 1	17	0 1 0 1 0
5	0 0 0 1 2	18	0 1 0 1 2
6	0 0 1 0 0	19	0 1 0 2 0
7	0 0 1 0 1	20	0 1 0 2 1
8	0 0 1 0 2	21	0 1 0 2 3
9	0 0 1 1 0	22	0 1 2 0 0
10	0 0 1 1 2	23	0 1 2 0 1
11	0 0 1 2 0	24	0 1 2 0 3
12	0 0 1 2 1	25	0 1 2 3 0
13	0 0 1 2 3	26	0 1 2 3 4

(a)

1	0 0 0 0 0		14	0 1 2 3 0	3
2	0 0 0 0 1	1	15	0 1 2 3 4	1
3	0 0 0 1 2	2	16	0 1 2 0 3	2
4	0 0 0 1 1	1	17	0 1 2 0 1	1
5	0 0 0 1 0	1	18	0 1 2 0 0	1
6	0 0 1 2 3	3	19	0 1 0 2 3	3
7	0 0 1 2 1	1	20	0 1 0 2 1	1
8	0 0 1 2 0	1	21	0 1 0 2 0	1
9	0 0 1 1 0	1	22	0 1 0 1 0	1
10	0 0 1 1 2	1	23	0 1 0 1 2	1
11	0 0 1 0 2	1	24	0 1 0 0 2	1
12	0 0 1 0 1	1	25	0 1 0 0 1	1
13	0 0 1 0 0	1	26	0 1 0 0 0	1

(b)

A recursive generating algorithm is said to run in *constant amortized time* (CAT) if it generates each object in $O(1)$ time, in amortized sense. Such an algorithm is also called a *CAT algorithm*. Ruskey [Rus03] shows that a recursive generating algorithm is a CAT one if it satisfies the following three properties:

1. Each recursive call generates at least one object (there is no dead-end recursive call);
2. The number of computations in each recursive call is proportional to the degree of the call (that is, the number of subsequent recursive calls produced by the current call). The call having zero degree is referred as *terminate call*.
3. The number of recursive calls having degree one (if any) is $O(N)$, where N is the number of generated objects.

3 The Gray code

Before proving the Graycodeness of $\mathfrak{B}_n$, first we need the three following lemmas. The proof of the first one is straightforward from Definition 2.2.

Lemma 3.1. *If $\boldsymbol{s} = s_1 s_2 \cdots s_k \in B_k$ and $M = \max\{s_1, s_2, \ldots, s_k\} + 1$, then 0 and M is, respectively, the smallest and the largest admissible value for s_{k+1} with respect to $\boldsymbol{s}$, such that $\boldsymbol{s}s_{k+1} \in B_{k+1}$.*

Lemma 3.2. *Let $\boldsymbol{s} = s_1 s_2 \cdots s_k \cdots s_n \in B_n$ be the last sequence in $\mathfrak{B}_n$, with respect to RGC order, having prefix $s_1 s_2 \cdots s_k$, for some $k \leq n$, and let $M = \max\{s_1, s_2, \ldots, s_k\} + 1$. Then, the sequence $\boldsymbol{s}$ is a length n prefix of the infinite sequence α defined below.*

- *If $\sum_{i=1}^{k} s_i$ is even, and*
  - *if M is even, then $\alpha = s_1 \cdots s_k M(M+1)00\cdots$, or*
  - *if M is odd, then $\alpha = s_1 \cdots s_k M00\cdots$.*
- *If $\sum_{i=1}^{k} s_i$ is odd, then $\alpha = s_1 \cdots s_k 00\cdots$.*

Proof. We begin the proof for the first claim where $\sum_{i=1}^{k} s_i$ is even. By Lemma 3.1, M is the largest admissible value for s_{k+1} with respect to $s_1 s_2 \cdots s_k$. Since $\boldsymbol{s}$ is the the last sequence in $\mathfrak{B}_n$ having prefix $s_1 s_2 \cdots s_k$ and $\sum_{i=1}^{k} s_i$ is even, it follows by definition of RGC order that $s_{k+1} = M$. Accordingly for the rest of the sequence, by Lemma 3.1 and definition of RGC order, the two following possibilities hold:

- if M is even, then $\sum_{i=1}^{k+1} s_i$ is even and this implies $s_{k+2} = (M+1)$. Since s_{k+2} is odd, then $\sum_{i=1}^{k+2} s_i$ is odd too, so that $s_{k+3} = 0$, which is the smallest admissible value with respect to $s_1 s_2 \cdots s_k s_{k+1} s_{k+2}$. Continuing in similar way for all succeeding positions, we have $0 = s_{k+4} = s_{k+5} = \ldots$.
- if M is odd, then $\sum_{i=1}^{k+1} s_i$ is odd, and as previously, $s_{k+2} = 0$. Continuing in similar way, we have $0 = s_{k+3} = s_{k+5} = \ldots$.

For the second claim, if $\sum_{i=1}^{k} s_i$ is odd, then as previously, s_{k+1} is the smallest admissible value with respect to $s_1 s_2 \cdots s_k$. Continuing in similar way, we have $0 = s_{k+2} = s_{k+3} = \cdots$. □

Lemma 3.3. *Let $\boldsymbol{t} = t_1 t_2 \cdots t_k \cdots t_n \in B_n$ be the first sequence in $\mathfrak{B}_n$, with respect to RGC order, having prefix $t_1 t_2 \cdots t_k$, for some $k \leq n$, and let $N = \max\{t_1, t_2, \ldots, t_k\} + 1$. Then, the sequence $\boldsymbol{t}$ is a length n prefix of the infinite sequence β defined below.*

- *If $\sum_{i=1}^{k} t_i$ is odd and*
  - *If N is even, then $\beta = t_1 \cdots t_k N(N+1)00\cdots$.*
  - *If N is odd, then $\beta = t_1 \cdots t_k N00\cdots$.*
- *If $\sum_{i=1}^{k} s_i$ is even, then $\beta = t_1 \cdots t_k 00\cdots$.*

Proof. The proof is similar with that of Lemma 3.2, by changing "even" with "odd" and vice versa, with the exception that the parity of N follows the parity of M. □

Combining Lemmata 3.2 and 3.3 we have the next theorem describing the Graycodeness of $\mathfrak{B}_n$.

Theorem 3.4. *The list $\mathfrak{B}_n$ is a 3-adjacent Gray code.*

Proof. If $\boldsymbol{s}$ and $\boldsymbol{t}$ are consecutive in $\mathfrak{B}_n$, where $\boldsymbol{s}$ precedes $\boldsymbol{t}$, and k is the leftmost position where $\boldsymbol{s}$ and $\boldsymbol{t}$ differ, then $\boldsymbol{s}$ is the last sequence in $\mathfrak{B}_n$ having prefix $s_1 s_2 \cdots s_k$, and $\boldsymbol{t}$ is the first sequence in $\mathfrak{B}_n$ having prefix $t_1 t_2 \cdots t_k$. Besides at position k, by referring to Lemma 3.2 and 3.3, the difference possibly occurs at position $k+1$ and $k+2$, since in any case $s_i = t_i = 0$, for $i \geq k+3$.

The proof for the adjacency is by showing that if $s_{k+2} \neq t_{k+2}$, then $s_{k+1} \neq t_{k+1}$. By referring to Lemma 3.2 and 3.3, we have the following conditions:

- If $\sum_{i=1}^{k} s_i$ and $\sum_{i=1}^{k} t_i$ have the same parity, then $s_{k+2} \neq t_{k+2}$ happens when

$$\alpha = s_1 \cdots s_k M(M+1)00\cdots \text{ and } \beta = t_1 \cdots t_k 00\cdots,$$

or alternatively,

$$\alpha = s_1 \cdots s_k M00\cdots \text{ and } \beta = t_1 \cdots t_k N(N+1)00\cdots.$$

- If $\sum_{i=1}^{k} s_i$ and $\sum_{i=1}^{k} t_i$ have different parity, then $s_{k+2} \neq t_{k+2}$ happens according to the following:
  - if $M \neq N$ and M, N have the same parity, then

$$\alpha = s_1 \cdots s_k M(M+1)00\cdots \text{ and } \beta = t_1 \cdots t_k N(N+1)00\cdots.$$

    or
  - if $M \neq N$ and M, N have different parity, then,

$$\alpha = s_1 \cdots s_k M(M+1)00\cdots \text{ and } \beta = t_1 \cdots t_k N00\cdots,$$

    or alternatively,

$$\alpha = s_1 \cdots s_k M00\cdots \text{ and } \beta = t_1 \cdots t_k N(N+1)00\cdots.$$

Conditions above clarify that if $s_{k+2} \neq t_{k+2}$, then $s_{k+1} \neq t_{k+1}$, which proves the adjacency property. □

By [Er84], the length n sequences $000\cdots0$ and $010\cdots0$ are the first and last, respectively, length n ballot sequences listed with respect to RGC order. This implies that $\mathfrak{B}_n$ is also a *cyclic* Gray code.

4 Algorithmic considerations

In this section we give exhaustive generating algorithms for ballot sequences in both lexicographic and RGC order. They require some additional notions that we introduce below.

For $\boldsymbol{s} = s_1 s_2 \cdots s_n \in B_n$ we define $A(\boldsymbol{s})$, the *set of admissible values* with respect to $\boldsymbol{s}$, as the set of integers a such that $\boldsymbol{s}a \in B_{n+1}$. Recall from Lemma 3.1 that $0 \in A(\boldsymbol{s})$ and $\max\{s_1, s_2, \ldots, s_n\} + 1 \in A(\boldsymbol{s})$, for any such a sequence $\boldsymbol{s}$. For example, if $\boldsymbol{s} = 010213 \in B_6$, then $A(\boldsymbol{s}) = \{0, 2, 4\}$; and $A(\boldsymbol{s}0) = \{0, 1, 2, 4\}$, $A(\boldsymbol{s}2) = \{0, 3, 4\}$, and $A(\boldsymbol{s}4) = \{0, 2, 5\}$. The *Parikh vector* of $\boldsymbol{s}$ is the sequence $\boldsymbol{c} = c_0 c_1 \cdots c_{n-1}$ with $c_i = |\boldsymbol{s}|_i$, for $i = 0, 1, \ldots, n-1$.

Let $\boldsymbol{s} \in B_n$ be a ballot sequence and $\boldsymbol{c} = c_0 c_1 \cdots c_{n-1}$ its Parikh vector. For an $a \in A(\boldsymbol{s})$, the Parikh vector $\boldsymbol{c}'$ of the ballot sequence $\boldsymbol{s}a \in B_{n+1}$ is simply obtained from $\boldsymbol{c}$ and considering $c_n = 0$, and it is $\boldsymbol{c}' = c_0 \cdots c_{a-1}(c_a + 1)c_{a+1} \cdots c_n$. However, the set $A(\boldsymbol{s}a)$ of admissible values for $\boldsymbol{s}a$ is a little more complicated, and it is given by the next easy to see proposition.

Proposition 4.1. *If $\boldsymbol{s} \in B_n$, $a \in A(\boldsymbol{s})$ and $\boldsymbol{c}$ is the Parikh vector of $\boldsymbol{s}$, then*

$$A(\boldsymbol{s}a) = \begin{cases} A(\boldsymbol{s}) \cup \{a+1\} & \text{if } c_{a-1} > c_a - 1, \\ A(\boldsymbol{s}) \cup \{a+1\} \setminus \{a\} & \text{otherwise.} \end{cases}$$

Lexicographic generation

Every non-empty prefix of a ballot sequence is a smaller size ballot sequence, and our generating algorithm expands recursively each length k ballot sequence into length $k+1$ ones, until the desired size is obtained, and the sequence is printed out by procedure PRINT.

At each generated prefix $\boldsymbol{t}$, our algorithm needs the set $A(\boldsymbol{t})$, and this is implemented by two linked lists *succ* and *pred* defined as follows. For an $a \in A(\boldsymbol{t})$:

- $succ[a]$ is the smallest value in $A(\boldsymbol{t})$ larger than a, if it exists; and is $succ[a] = n$ otherwise.
- $pred[a]$ is the largest value in $A(\boldsymbol{t})$ smaller than a, if it exists; and $pred[a] = -1$ otherwise.

Before the first recursive call of our generating algorithm, the variables are initialized as follows:

- the current sequence $\boldsymbol{s}$ is $0 \in B_1$, the unique generalized ballot sequence of length one,
- $\boldsymbol{c}$, the Parikh vector of the current generated sequence is the length n array $100\cdots0$,

```
procedure UPDATE(a: integer)
   c[a] := c[a] + 1;
   if succ[a] ≠ a + 1 then
      succ[a + 1] := succ[a]; succ[a] := a + 1;
      pred[succ[a + 1]] := a + 1; pred[a + 1] := a;
   if a ≠ 0 and c[a] = c[a − 1] then
      succ[pred[a]] := succ[a];
      pred[succ[a]] := pred[a];
end procedure
```

```
procedure RESTORE(a, before, after: integer)
   c[a] := c[a] − 1;
   succ[before] := a; pred[after] := a;
   pred[a] := before; succ[a] := after;
end procedure
```

Figure 1: Procedures UPDATE and RESTORE.

```
procedure GEN_LEX(k: integer)
   local a, before, after: integer;
   if k = n + 1 then PRINT;
   else a := 0;
        while a < n
            s[k] := a;
            before := pred[a]; after := succ[a];
            UPDATE(a);
            GEN_LEX(k + 1);
            RESTORE(a, before, after);
            a := succ[a];
        end while
end procedure
```

Figure 2: Procedure GEN_LEX.

- $succ[0] = 1$ and $succ[1] = n$,
- $pred[n] = 1$, $pred[1] = 0$ and $pred[0] = -1$.

For each value $a \in A(\boldsymbol{s})$, the lists *succ* and *pred* are updated, and after the corresponding recursive call, *succ* and *pred* are restored. The obtained lexicographic generating algorithm for B_n is GEN_LEX in Figure 2, the main call is GEN_LEX(2), and n is a global variable.

Gray code generation

Adapting the algorithm GEN_LEX according to the considerations in Section 3 we obtain the algorithm GEN_GRAY in Figure 3 which generates the set B_n in RGC order, that is the list $\mathfrak{B}_n$. The main call is GEN_GRAY$(2, 0)$.

Theorem 4.2. *Algorithm* GEN_LEX *and* GEN_GRAY *satisfy the CAT desiderata.*

Proof. Since $\{0, M\} \subset A(\boldsymbol{s})$, letting $M = \max\{s_1, s_2, \cdots, s_n\}+1$, it follows that each recursive call GEN_LEX$(k+1)$ or GEN_GRAY$(k + 1,\ sum)$ generates at least two ballot sequences $s_1 s_2 \cdots s_k 0$ and $s_1 s_2 \cdots s_k M$, so that there is no call of degree one. Each recursive call produces several subsequent recursive calls doing the similar computations with different parameters. This means that the number of computations in each recursive call is proportional to the degree of call. So the algorithm satisfies the CAT desiderata presented at the end of Section 2. □

As a result, Theorem 4.2 confirms the CAT complexity for both algorithms.

Corollary 4.3. *Algorithm* GEN_LEX *and* GEN_GRAY *are CAT generating algorithms.*

```
procedure GEN_GRAY(k, sum: integer)
  local a, before, after: integer;
  if k = n + 1 then PRINT;
  else if sum mod 2 = 0 then
        a := 0;
        while a < n
            s[k] := a;
            before := pred[a]; after := succ[a];
            UPDATE(a);
            GEN_GRAY(k + 1, sum + a);
            RESTORE(a, before, after);
            a := succ[a];
        end while
      else
        a := pred[n];
        while a ≥ 0
            s[k] := a;
            before := pred[a]; after := succ[a];
            UPDATE(a);
            GEN_GRAY(k + 1, sum + a);
            RESTORE(a, before, after);
            a := pred[a];
        end while
end procedure
```

Figure 3: Procedure GEN_GRAY.

References

[BV05] J. L Baril, V. Vajnovszki. Minimal change list for Lucas strings and some graph theoretic consequences. *Theoretical Computer Science*, 346:189-199, 2005.

[BBP15] A. Bernini, S. Bilotta, R. Pinzani, A. Sabri, V. Vajnovszki. Reflected Gray codes for q-ary words avoiding a given factor. *Acta Informatica*, 52(7):573-592, 2015.

[Ber87] J. Bertrand. Solution d'un probléme. *Comptes Rendus de l'Académie des Sciences*, 105, p. 369, 1887.

[Er84] M. C. Er. On generating the N-ary reflected Gray code. *IEEE Transaction on Computers*, 33(8):739–741, 1984.

[GrA53] F. Gray. Pulse code communication. *U.S. Patent 2632058*, 1953.

[Kin81] P. Klingsberg. A Gray code for compositions. *Journal of Algorithms*, 3(1):41-44, 1981.

[OEI] *The On-line Encyclopedia of Integer Sequences (OEIS)*. `https://oeis.org/`.

[Ren08] M. Renault. Lost (and found) in translation: André's actual method and its application to the generalized ballot problem. *The American Mathematical Monthly*, 115(4):358-363, 2008.

[Rus03] F. Ruskey. *Combinatorial Generation*, book in preparation, 2003.

[Sab15] A. Sabri and V. Vajnovszki. Two Reflected Gray code based orders on some restricted growth sequences. *The Computer Journal*, 58(5):1099-1111, 2015.

[Sab17] A. Sabri, V. Vajnovszki. More restricted growth functions: Gray codes and exhaustive generations. *Graphs and Combinatorics*, 33(3):573-582, 2017.

[Sag01] B. Sagan, The Symmetric Group: Representations, Combinatorial Algorithms, and Symmetric Functions, Springer-Verlag, New York, 2001.

[Sta99] R. P. Stanley. *Enumerative combinatorics-Volume 2.* Cambridge Studies in Advanced Mathematics, 62. Cambridge University Press, Cambridge, 1999.

[Vaj01] V. Vajnovszki, A loopless generation of bitstrings without p consecutive ones, *Discrete Mathematics and Theoretical Computer Science*, Springer, 227-240, 2001.

[Vaj07] V. Vajnovszki, Gray code order for Lyndon words, Discrete Mathematics and Theoretical Computer Science, 9(2):145-152, 2007.

[Vaj08] V. Vajnovszki, More restrictive Gray codes for necklaces and Lyndon words, *Information Processing Letters*, 106, 96-99, 2008.

[Vaj10] V. Vajnovszki. Generating involutions, derangements, and relatives by ECO. *Discrete Mathematics and Theoretical Computer Science*, 12(1):109-122, 2010.

[Vaj11] V. Vajnovszki, R. Vernay. Restricted compositions and permutations: from old to new Gray codes. *Information Processing Letters*, 111:650-655, 2011.

[Wal00] T. Walsh. Loop-free sequencing of bounded integer compositions. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 33:323-345, 2000.

[Wal03] T. Walsh. Generating Gray codes in $O(1)$ worst-case time per word. *Lecture Notes in Computer Science*, 2731:71-88, 2003.