Математика и информатика | *Volume 66, Number 6, 2023* | *Mathematics and Informatics*

https://doi.org/10.53656/math2023-6-1-mir

Science in Education
Научно-методически статии

MIRROR (LEFT-RECURSIVE) GRAY CODE

Dr. Valentin Bakoev, Assoc. Prof.
"St. Cyril and St. Methodius" University of Veliko Tarnovo (Bulgaria)

Abstract. Here we consider a version of the Binary Reflected Gray Code (BRGC, or Gray code for short), called the mirror Gray code – in parallel and in comparison to the Gray code. We discuss some sources where the two codes are not distinguished and the reasons why. We present our arguments for treating the two codes as distinct, starting with a definition of the mirror Gray code and showing its main properties. Among the most important of these is the relationship between the two codes – the codewords of the mirror Gray code are "left-right mirror images" of those in the Gray code and vice versa. Other arguments we show are the differences in the algorithms for generating the codewords. We also present another approach that involves representing the codewords by integers (their serial numbers), generating and examining the two codes by using their sequences of serial numbers. For this purpose, we use the bijections that relate the pairs of combinatorial objects: binary vectors in a certain order and the corresponding integer sequences and subsets. In this way, we get different algorithms, sequences, and functions for successor and predecessor, for ranking and unranking at the mirror Gray code. We believe that this article provides at least one more perspective, as well as expands and enriches the knowledge of the Gray code.

Keywords: Gray code; mirror Gray code; lexicographic order; binary vector; codeword; generation algorithm; integer sequence; predecessor; successor; rank; unrank function

1. Introduction

The ordering of all binary vectors (words, codewords) in a Gray code is a well-known concept that has many useful properties and applications. The most important property of the ordering in a Gray code is known as *minimal change*, which means that every two adjacent vectors in this sequence differ by exactly one coordinate. This property applies not only to the generation of binary vectors but also to other combinatorial objects – Frank Ruskey notes "The fastest known algorithms for generating certain objects are based on Gray codes..." (Ruskey 2003, p. 115). Other applications (and relations) of the Gray code are discussed in many sources. Almost all of them associate it with the n-dimensional Boolean cube (hypercube) and Hamiltonian cycles in the graph corresponding to the cube. Examples of a hypercube network with 2^n processors such that each processor is connected to other n processors are given in (Rosen 2012), (Koshy 2003), (Grimaldi 2004), (Anderson 2001), etc. In (Ruskey 2003), (Rosen 2012), (Knuth 2011), (Waren 2012) examples of a measuring device with a rotating pointer and 2^n sectors of its scale are given. When they are labeled by a Gray code, as opposed to their lexicographic labeling, the possibility of errors is minimized. Some historical notes and references on the French telegraph engineer Émile Baudot and his code (patented in 1874), the Bell Labs researcher Franc Gray and his "reflected binary code" (patented in 1947), the French magistrate Louis Gros and his solution to the classic toy "Chinese ring puzzle", as well as a solution of the "Towers of Hanoi problem" etc. are given in (Knuth 2011), (Ruskey 2003). Applications of Gray code in: (1) minimization of Boolean functions via Karnaugh maps are discussed in (Garnier & Taylor 2002), (Grimaldi 2004), (Ruskey 2003), (Manev 2012); (2) databases – in (Ruskey 2003); (3) addressing microprocessors, hashing algorithms, distributed systems, channel noise detection and correction – in (Bunder et al. 2008); (4) Algebraic coding theory – in (MacWilliams & Sloane 1978), (Suparta 2006), and others. So it is not surprising that dozens of different types of Gray codes have been invented and studied – see the surveys (Savage 1997) and especially the latter (Mütze 2022).

Here we consider another type of Gray code, which is closely related to the *Binary Reflected Gray Code* (BRGC), also called a *standard Gray code*, or *Gray code* for short. It is obtained by reflection of the internal order of the bits in the codewords of the Gray code. In (Knuth 2011), in Section 7.1.3. (Bit reversal, p. 144), Knuth writes "… let's change $x = (x_{63}x_{62} \dots x_1x_0)_2$ to its *left-right mirror image*, $x^R = (x_0x_1 \dots x_{62}x_{63})_2$" and that is why we call it *mirror Gray code*. We present our arguments regarding the distinction between the two codes and consider this code to be different from the Gray code. We use another approach that involves representing the codewords by integers (their serial numbers), generation and examining the two codes by using their sequences of serial numbers and performing simple operations on them. For this purpose, we use the bijections that relate the pairs of combinatorial objects: binary vectors in a certain ordering and their corresponding integer sequences and subsets. In this way, we get different algorithms, sequences, and functions for successor and predecessor, for ranking and unranking at the mirror Gray code. This article summarizes the main results of (Bakoev 2023a) and (Bakoev 2023b), which you can refer to for details.

2. Basic notions

The set of all n-dimensional binary vectors (words) is known as an *n-dimensional Boolean cube* (hypercube). It is defined as the Cartesian n-th power of the set $\{0,1\}$ which is $\{0,1\}^n = \{(x_1, x_2, \dots, x_n) | x_i \in \{0,1\}, \text{ for } i = 1, 2, \dots, n\}$. Let $\alpha = (a_1, a_2, \dots, a_n)$ and $\beta = (b_1, b_2, \dots, b_n)$ be arbitrary vectors of $\{0,1\}^n$. A *serial number* of the vector α is the non-negative integer $\#\alpha = \sum_{i=1}^{n} 2^{n-i} . a_i$, which is the decimal representation of the binary number $a_1, a_2, \dots, a_n$ formed by the coordinates of α. A (*Hamming*) *weight* of α is the non-negative integer $wt(\alpha)$ equal to the number of non-zero coordinates of α. Since the coordinates of α are zeros and ones, $wt(\alpha) = \sum_{i=1}^{n} a_i$. A *Hamming distance* between α and β is the non-negative integer $d(\alpha,\beta) = \sum_{i=1}^{n}|a_i - b_i| = \sum_{i=1}^{n}(a_i \oplus b_i)$, where $\oplus$ is the Boolean function eXclusive OR (XOR). So $d(\alpha,\beta)$ means the number of coordinates in which α and β differ. When $d(\alpha,\beta) = 1$ we call α and β *adjacent vectors*, or more precisely *adjacent in the ith coordinate*, if they differ only on it. When $d(\alpha,\beta) = n$ we call α and β *opposite*. We say that α *lexicographically precedes* β and write $\alpha \leq \beta$ if $\alpha = \beta$ or there exists an integer $i, 1 \leq i \leq n$, such that $a_i < b_i$, and $a_j = b_j$, for all $j < i$. The relation $R_{\leq}$ defined on $\{0,1\}^n \times \{0,1\}^n$ with $(\alpha,\beta) \in R_{\leq}$ if $\alpha \leq \beta$ is called a *lexicographic precedence* relation. It is reflexive, strongly antisymmetric and transitive and so $R_{\leq}$ is a *total order relation* on the vectors of the n-dimensional Boolean cube. The ordering determined by $R_{\leq}$ is called *lexicographic*.

Let n be a positive integer, $U = \{u_1, u_2, \dots, u_n\}$ be a given set, and $X \subseteq U$. The vector $\alpha = (a_1, a_2, \dots, a_n) \in \{0,1\}^n$ defined as: $a_i = 0$, when $u_i \notin$ X, or $a_i = 1$, when $u_i \in$ X, for $i = 1, 2, \dots, n$, is called the *characteristic vector* of the subset X. A well-known theorem states that *if U is an n-element set, the function*

$$f: U \to \{0,1\}^n,$$

defined by the notion of a characteristic vector is a ***bijection***.

Many authors denote by G_n the sequence of all binary vectors of $\{0,1\}^n$ in a *Gray code* and define it recursively:

$$G_n = \begin{cases} (0),(1), & \text{if } n = 1, \\ 0G_{n-1}, 1G_{n-1}^r, & \text{if } n > 1, \end{cases} \tag{1}$$

where $0G_{n-1}$ denotes all vectors of $\{0,1\}^{n-1}$ in a Gray code, prefixed by 0, and $1G_{n-1}^r$ denotes all vectors of $\{0,1\}^{n-1}$ in a Gray code, taken in reversed order and prefixed by 1 (Ruskey 2003), (Kreher & Stinson 1999). Knuth uses a similar definition, starting from $n = 0$ and the empty word (Knuth 2011). The following inductive and constructive definition is more useful for us.

Definition 1. 1) The vectors (0) and (1) of $\{0,1\}^1$ are ordered in a *Gray code*.

2) Let $\alpha_0, \alpha_1, \dots, \alpha_{2^{n-1}-1}$ be the sequence of all vectors of $\{0,1\}^{n-1}$, ordered in a *Gray code*.

3) To obtain the vectors of $\{0,1\}^n$ in Gray code, we perform three steps. First, we prefix with 0 all vectors $\alpha_0, \alpha_1, \dots, \alpha_{2^{n-1}-1}$ of $\{0,1\}^{n-1}$ in a Gray code. Then, we take the same sequence $\alpha_0, \alpha_1, \dots, \alpha_{2^{n-1}-1}$ a second time and reverse the order of its vectors, i.e. we take its reflection $\alpha_{2^{n-1}-1}, \dots, \alpha_1, \alpha_0$ and then we prefix each of its vectors with 1. Third, we concatenate the two sequences, and the result

$$(0,\alpha_0),(0,\alpha_1),\dots,(0,\alpha_{2^{n-1}-1}),(1,\alpha_{2^{n-1}-1}),\dots,(1,\alpha_1),(1,\alpha_0)$$
contains all vectors of $\{0,1\}^n$ ordered in a *Gray code*.

Both definitions show why the Gray code is called *reflected* and *cyclic* – notice that the first and last vectors in these sequences are also adjacent.

Here we note that if we neglect the reflection in point 3) of Definition 1, we shall obtain the vectors of $\{0,1\}^n$ in lexicographic order. It is well known that *when the vectors of* $\{0,1\}^n$ *are in lexicographic order, the sequence of their serial numbers is:* $0,1,2,\dots,2^n-1$, *and the function that relates them is a bijection*.

Besides by this definition, the vectors of $\{0,1\}^n$ can be generated in a Gray code by using the so-called **transition sequence**. In accordance with the position of the new-coordinate expansion in Definition 1, we number the coordinates of the vectors of $\{0,1\}^n$ from right to left, i.e. $x=(x_n,x_{n-1},\dots,x_2,x_1)$.

Definition 2. Let $\alpha_0,\alpha_1,\dots,\alpha_{2^n-1}$ be the vectors of $\{0,1\}^n$ ordered in a Gray code. The sequence of integers: $T(n)=\ t_1,t_2,\dots,t_{2^n-1}$, where $t_i\in\{1,2,\dots,n\}$, for $i=1,2,\dots,2^n-1$, is defined as follows: the ith term t_i denotes the number of the coordinate in the serial vector α_{i-1} that must be inverted to obtain the next vector α_i, for $i=1,2,\dots,2^n-1$. $T(n)$ is called a *transition sequence*.

From the last two definitions it follows that $T(n)$ is a symmetric sequence with respect to its middle term, for $=1,2,3,\dots$ We can define $T(n)$ with $T(1)=1$, and $T(n)=T(n-1),n,T(n-1)$. Hence $T(n)$ is a *palindrome* of integers, exactly a *palindrome of palindromes* if we exclude the middle term. Another way to obtain $T(n)$ is given by the following theorem, which is little known.

Theorem 1. Let the vectors of $\{0,1\}^n$ be in lexicographic order. Then the sequence of Hamming distances between the pairs of adjacent vectors, i.e. $d(\alpha_{i-1},\alpha_i)$, for $i=1,2,\dots,2^n-1$, coincides with the transition sequence $T(n)$ of the Gray code.

More details about the transition sequence and its varieties can be found in, (Knuth 2011), (Gulliver et al. 1999), (Bakoev 2023a).

Table 1 illustrates the statement of Theorem 1, and Table 2 shows how the vectors of $\{0,1\}^n$ in a Gray code are obtained via the transitions sequence. The sequence denoted by $\#\alpha$ in Table 1 is A001477 in the OEIS[1]. The sequence denoted by $\#\beta$ in Table 2 is A003188, and the transition sequence is A001511. The tables also illustrate the bijection between the (serial) numbers in the left columns of the two tables (sequences A001477 and A003188) and the vectors of $\{0,1\}^n$ in the corresponding row. The aforementioned bijection between $\{0,1\}^n$ and the subsets of an n-element set (via the notion of a characteristic vector) defines an ordering of the subsets corresponding to the ordering of the vectors. So, **we explore the**

combinatorial objects: integer sequences, binary vectors, and subsets in a certain order through these bijections that relate each pair of them.

Table 1. Lex. order and distances between the consecutive vectors

$\#\alpha$	$\alpha \in \{0,1\}^n$ in lex. order	$d(\alpha_{i-1}, \alpha_i)$
0	$(0, \dots, 0,0,0,0)$	1
1	$(0, \dots, 0,0,0,1)$	2
2	$(0, \dots, 0,0,1,0)$	1
3	$(0, \dots, 0,0,1,1)$	3
4	$(0, \dots, 0,1,0,0)$	1
5	$(0, \dots, 0,1,0,1)$	2
6	$(0, \dots, 0,1,1,0)$	1
7	$(0, \dots, 0,1,1,1)$	4
8	$(0, \dots, 1,0,0,0)$	
$\vdots$	$\vdots$	$\vdots$

Table 2. Gray code order and the transition sequence

$\#\beta$	$\alpha \in \{0,1\}^n$ in Gray code	$T(n)$
0	$(0, \dots, 0,0,0,0)$	1
1	$(0, \dots, 0,0,0,1)$	2
3	$(0, \dots, 0,0,1,1)$	1
2	$(0, \dots, 0,0,1,0)$	3
6	$(0, \dots, 0,1,1,0)$	1
7	$(0, \dots, 0,1,1,1)$	2
5	$(0, \dots, 0,1,0,1)$	1
4	$(0, \dots, 0,1,0,0)$	4
12	$(0, \dots, 1,1,0,0)$	
$\vdots$	$\vdots$	$\vdots$

As we have seen, the generation of the Gray code for $\{0,1\}^n$ can be done by using formula (1), Definition 1, or by the transition sequence. This can be done through the predecessor and successor functions, through ranking and unranking functions (Ruskey 2003), (Kreher & Stinson 1999), (Knuth 2011), and others.

3. About mirror (left-recursive) Gray code

After an extensive search, we found 3 sources that refer to mirror (lefy-recursive) Gray code. In (Nijenhuis & Wilf 1978), the authors define Gray code recursively in relation to Hamiltonian walks in the Boolean cube and the generation of the subsets by minimal change. For a given set/subset S, $|S| = k$, the authors formulate and prove the **rule of succession**: "If k is even, then $j = 1$; if k is odd, then j is the index of the coordinate which follows the first "1" bit of S". Actually, the authors define, comment and use only the mirror Gray code.

In (Lipski 1988) an inductive definition of the Gray code is given, where the expansion of the codewords with a new coordinate is done on the right-hand side of the vectors (as in Definition 1). Lipski proposes Algorithm 1.13 which generates all subsets of a set of n elements ordered in a Gray code. In fact, the algorithm uses the "rule of succession" and successively performs integer divisions of the integers $0, 1, \dots, 2^n - 1$ to compute the coordinate number that must be inverted to obtain the new codeword. This corresponds to numbering the codeword coordinates (bits) from right to left (i.e. $x_n, x_{n-1}, \dots, x_1$), but the algorithm inverts the values (0 or 1) of an array whose elements are numbered in reverse, with $1, 2, \dots, n$ (Lipski 1988, p. 31, Fig. 1.6). So the algorithm generates the mirror Gray code. Lipski does not notice to this fact and does not distinguish between the two codes.

There are similar considerations in the third source (Ruskey 2003). Ruskey first defines recursively and considers binary reflected Gray code (BRGC) in relation to a Hamiltonian cycle on the Boolean cube and the Towers of Hanoi problem. Ruskey defines the transition sequence and considers the generation of this Gray code by it. The ranking and unranking functions for BRGC are also discussed. But in the next section "Recursive generation of BRGC", by formula (5.3), Ruskey gives a recursive definition of the mirror Gray code. Two recursive algorithms built on this formula are proposed: Algorithm 5.2 (*indirect*) and Algorithm 5.3 (*direct*). However, Ruskey does not distinguish between the two codes, possibly because the results of running Algorithm 5.1 or Algorithm 5.2 are not shown.

In (Suparta 2006) some equivalences are considered over the set of "all cyclic minimal-change N-ary Gray codes of length n". Following this classification, Gray code and mirror Gray code are equivalent because one of them can be obtained from the other by permutation of its columns, i.e. by permutation of its coordinates.

After all, the **main question** follows: Should we distinguish between Gray code and Gray mirror code, and is it even worth considering Gray mirror code separately? Our opinion is "yes" and we argue this view, presenting more arguments in the remainder of this article.

4. Mirror (left-recursive) Gray code

The following definition is analogous to Definition 1, the difference being in the place of expansion with a new coordinate, and we start with the case $n = 0$.

Definition 3. 1) If $n = 0$, then $\{0,1\}^0 = \{()\}^0$, i.e. it contains only the empty vector – an analog of the empty word. If $n = 1$, the vectors (0) and (1) of $\{0,1\}^1$ are ordered in a *mirror (left-recurcive) Gray code,* simply *mirror Gray code*.

2) Let $\alpha_0, \alpha_1, \dots, \alpha_{2^{n-1}-1}$ be the sequence of all vectors of $\{0,1\}^{n-1}$, ordered in a *mirror Gray code*.

3) We perform three steps. First, we append 0 at the end of all vectors $\alpha_0, \alpha_1, \dots, \alpha_{2^{n-1}-1}$ of $\{0,1\}^{n-1}$ in mirror Gray code. Second, we take the reflection $\alpha_{2^{n-1}-1}, \dots, \alpha_1, \alpha_0$ ofthe same sequence and then append 1 at the end of each of its vectors. Third, we concatenate the two sequences, and the result

$$(\alpha_0, 0), (\alpha_1, 0), \dots, (\alpha_{2^{n-1}-1}, 0), (\alpha_{2^{n-1}-1}, 1), \dots, (\alpha_1, 1), (\alpha_0, 1)$$

contains all vectors of $\{0,1\}^n$ ordered in a *mirror Gray code*.

The following statements follow from definitions 1 and 3. They can be proven rigorously by induction on n, following both definitions.

Theorem 2. The mirror Gray code is reflected and cyclic, as is the Gray code.

Theorem 3. (**Relationship between Gray code and mirror Gray code**) The mirror Gray code can be obtained from the Gray code and vice versa, by reflecting the internal order (i.e. the bits' order) of the vectors in the corresponding code.

We can supplement and clarify the last statement by defining the function (or operation) to *reflect* (or *reverse*) the coordinates of a vector. We denote it by φ and define $\varphi : \{0,1\}^{\mathrm{n}} \to \{0,1\}^{\mathrm{n}}$, for an arbitrary vector $\alpha = (a_1, a_2, \dots, a_n), \varphi(\alpha) = \alpha^R = (a_n, \dots, a_2, a_1)$. Thus the coordinates of α^R are a mirror image of those of α and vice versa. Obviously $\varphi(\alpha^R) = \varphi(\varphi(\alpha)) = \alpha$ and so φ is an *involution*.

Corollary 1. (**Fixed points of transformation of Gray code to mirror Gray code**) The vectors that occupy the same positions and match in Gray code and mirror Gray code are the palindromes. Hence their number is $2^{\lceil n/2 \rceil}$.

Corollary 2. Let the coordinates of the vectors of the n-dimensional Boolean cube be numbered $1, 2, \dots, n$, i.e. from left to right. Then the sequence $T(n)$ given by Definition 2 is a transition sequence for the mirror Gray code.

Theorem 4. Let the vectors of $\{0,1\}^{\mathrm{n}}$ be obtained according to Definition 3. Then:

a) The sequence $s(n)$ of serial numbers of the vectors is defined recursively as:

$$s(n) = \begin{cases} 0, \text{ if } n = 0, \\ 2.s(n-1), 2.s^r(n-1) + 1, \text{ if } n > 1. \end{cases} \tag{2}$$

Thus $s(n)$ is the result of concatenating two sequences: (1) the terms of $s(n-1)$ multiplied by 2 and (2) $2.s^r(n-1) + 1$ which means that each term of the reversed sequence $s^r(n-1)$ is multiplied by 2 and increased by 1.

b) The sequence $w(n)$ of vector weights is defined recursively as:

$$w(n) = \begin{cases} 0, \text{ if } n = 0, \\ w(n-1), w^r(n-1) + 1, \text{ if } n > 1. \end{cases} \tag{3}$$

Similarly, $w(n)$ is the result of concatenating two sequences: $w(n-1)$ and $w^r(n-1) + 1$ – each of the terms of the reversed sequence is incremented by 1.

Proof. We will prove the statements of the theorem by induction on k.

1) For $k = 0$ and $k = 1$ both statements are obvious.

2) We assume that for arbitrary positive integer k the formulas (2) and (3) are valid, respectively for the sequences $s(k)$ and $w(k)$.

3) We consider the vectors of $\{0,1\}^{\mathrm{k+1}}$ in mirror Gray code obtained according to Definition 3.

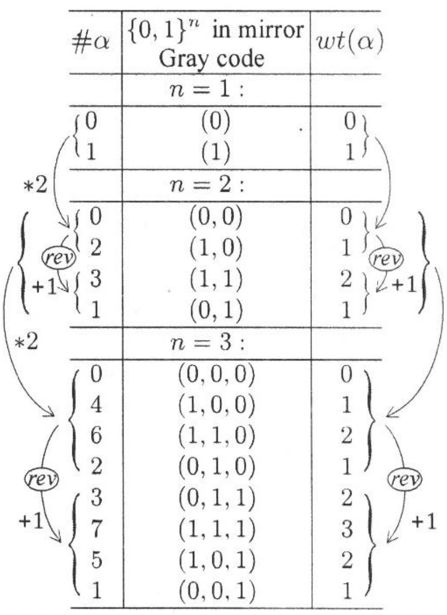
a) The first half of $\{0,1\}^{k+1}$ vectors are the vectors of $\{0,1\}^{k}$ expanded with a new coordinate 0 at the end of each of them. Thus, their serial numbers are obtained from the corresponding serial numbers of the vectors in $s(k)$ multiplied by 2. The second half of vectors of $\{0,1\}^{k+1}$ are obtained by reflection the outer order of the vectors of $\{0,1\}^{k}$ and then 1 is added at the end of each of them. Thus, their serial numbers are obtained from those of the reflected sequence $s^{r}(k)$, multiplied by 2 (due to the addition of a new coordinate at the end) and increased by one, since the new coordinate is 1. Therefore, formula (2) is also true for $k+1$.

b) The assertion about weights is proved analogously. In the first half of $\{0,1\}^{k+1}$the vectors remain the same as in $w(k)$ because all vectors of $\{0,1\}^{k}$ are expanded with a new coordinate 0. Analogously, the second half of the vectors of $\{0,1\}^{k+1}$ are obtained by incrementing the terms of the reflected sequence $w^{r}(k)$ by 1. Therefore, formula (3) is also true for $k+1$.

Therefore, the statements of the theorem are true for all positive integers n. □

Let us clarify formula (2). In fact $2.s^{r}(n-1)+1$ means that the resulting sequence $2.s(n-1)$ of (1) is taken a second time, reversed and all its terms are incremented by 1. Hence $2.s^{r}(n-1)+1=(2.s(n-1))^{r}+1$ and then $s(n)$ is a concatenation of $2.s(n-1)$, and $(2.s(n-1))^{r}+1$, when $n>0$. So we do not need to multiply the terms of $2.s(n-1)$ by 2 twice. Thus, the first half of $s(n)$ contains even integers, and the second half contains odd integers.

These notes and the statement of Theorem 4 are illustrated in Figure 1 for $n=1,2,3$. We note that the sequence $w(n)$ is the same for the Gray code and the mirror Gray code. The entry of $w(n)$ in the OEIS is A005811.



$\#\alpha$	$\{0,1\}^n$ in mirror Gray code	$wt(\alpha)$
	$n=1:$	
0	(0)	0
1	(1)	1
	$n=2:$	
0	(0,0)	0
2	(1,0)	1
3	(1,1)	2
1	(0,1)	1
	$n=3:$	
0	(0,0,0)	0
4	(1,0,0)	1
6	(1,1,0)	2
2	(0,1,0)	1
3	(0,1,1)	2
7	(1,1,1)	3
5	(1,0,1)	2
1	(0,0,1)	1

*2
rev
+1
*2
rev
+1
rev
+1
rev
+1

Figure 1. Illustration of Definition 2 and Theorem 4, for $n=1,2,3$

5. Generation of the mirror Gray code

There are several ways to generate mirror Gray code. We have already commented on the byte-wise (in an array) generation of the mirror Gray code by

Algorithm 1.13 in (Lipski 1988) and Algorithms 5.1 and 5.2 in (Ruskey 2003). The same type of generation can be done according to Corollary 2 – by using the transition sequence $T(n)$, in two ways. The first way is to generate the entire transition sequence in an array, following Definition 2 and the notes after it. The following simple function does this, assuming that the array `t` is suitably defined beforehand.

```
void gen_trans_seq (int n) {
  t[1] = 1;   // itnitalization for n = 1
  int mt = 1, len = 1;   // current max. term and current length
  while (mt < n) {
    mt++; len++;
    t[len] = mt;   // current center of symmetry
    for (int i = 1; i < len; i++)
      t[len+i] = t[len-i];  // or t[len+i] = t[i];
    len = 2 * len - 1;
  }
}
```

The second way is to compute the terms of $T(n)$ one after the other using Theorem 1 (see Table 1). This involves computing the weight of a binary vector (i.e. computer word) since $d(\alpha_i, \alpha_{i+1}) = wt(\alpha_i \oplus \alpha_{i+1})$. This can be done via the `__popcnt`, or via the `__builtin_popcount` function (for the GCC and Clang compilers), for example:

```
int next_term_of_TS (int k) {
  return __popcnt(k^(k-1));
  //or __builtin_popcount(k^(k-1));
}
```

In the first way, the sequence $T(n)$ is generated with time complexity $\Theta(2^n)$ – exponential concerning the size of the input but linear concerning the size of the output. Thus, one term of $T(n)$ is computed in constant time. The second way requires $O(\log_2 n)$ time, which is the running time of the `popcnt` function, and n is the size of the computer word that stores k. The transition sequence can be used to generate both a Gray code and a mirror Gray code, both byte-wise and bitwise.

5.1. Generating the sequence of the serial numbers of the vectors in mirror Gray code

Here we consider two algorithms that generate the sequence of serial numbers $s(n)$ of the vectors in the mirror Gray code. The **first algorithm**, called

Gen_MGC_Seq, is based on Theorem 4 and the remarks after it that determine its correctness. It sequentially generates the sequences $s(1), s(2), \dots, s(n)$ in the one-dimensional array `seq` with 2^n elements, predefined according to a suitable maximum value of n, so that memory can store them. Here is its code.

```
const int max_n = 26; // maximum length of the vectors
const int seq_max_len = 1 << max_n;
int seq[seq_max_len]; // to store s(n)
void gen_mirror_Gray_code_seq (int n) {// 0 < n ≤ max_n
  seq[0] = 0;  seq[1] = 1;   // initialization: s(1)
  int len = 2; // length of the current sequence
  for (int i = 2; i <= n; i++) {   // generates s(i)
    for (int j = 0; j < len; j++)// filling the left half
      seq[j] *= 2;
    for (int j = 0; j < len; j++) // filling the right half
      seq[j + len] = seq[len - j - 1] + 1;
    len *= 2; // doubling the length
  }
}
```

When $n \leq 32$ and there is enough memory (static or dynamic) to hold the array `seq`, then 32-bit (or 16-bit) computer words can be used to represent the elements of the sequence $s(n)$. The complexity of the algorithm is easily calculated. The first nested loop performs a total of $\Theta(2 + 4 + \cdots + 2^{n-1}) = \Theta(2^n)$ operations. The second nested loop also performs$\Theta(2^n)$ operations. The time complexity of the algorithm is therefore $\Theta(2^n)$ and so is the space complexity since it uses an array of 2^n integers. So the average cost of computing an arbitrary element of $s(n)$ is constant, i.e. Gen_MGC_Seq is a CAT-algorithm (CAT stands for Constant Amortized Time (Ruskey 2003)).

The **second algorithm** is called **Mirror algorithm G** and it is a modification of the well-known **Algorithm G** given in (Knuth 2011). Algorithm G is similar to Algorithm 1.13 (Lipski 1988) and also uses the “rule of succession”. But it maintains a parity check variable that reverses its value after each subsequent codeword is generated. In (Knuth 2011) the bits are numbered $n - 1, \dots, 1, 0$, the algorithm is described in words and it is suitable for byte-wise and bitwise implementations. In our bitwise version of Algorithm G, all codewords are sequentially received in the integer `c_v`, which stands for “**c**urrent **v**ector”, the binary representation of that integer. Briefly, the algorithm starts with the zero vector whose parity is 0. If at the current step, the parity is 0, the rightmost (the least significant) bit of c_v is flipped. Otherwise, the algorithm looks for the bit with the least significant 1 in c_v, i.e. the first bit with value 1 from right to left. If the number of this bit i is equal to $n-1$, the algorithm terminates, otherwise it reverses the value of the $(i+1)$th bit. Each time a new integer is received, the value of the parity variable is flipped.

We reverse these rules to obtain the Mirror algorithm G: (1) if the parity is 0, the $(n-1)$th bit is flipped, or (2) we look for the most significant bit that is 1 to flip the value of the bit to its right if its number j is ≥ 0. In the following code we use the vector $(1,0,0,\dots,0)$, i.e. 1, followed with $(n-1)$ zeros (which is 2^{n-1}) as a mask – this is the variable m.

```
void mirror_alg_G (int n) { // for n > 0
  int c_v = 0, // current vector – the zero vector
  m = 1 << (n-1); // mask for the most significant bit
  int parity = 0; // for parity check, 0 for c_v
  int j; // bit number to flip
  do {
    cout << c_v << ", "; // or use (c_v), or visit (c_v)
    if (0 == parity) c_v ^= m; // flips the most significant bit
    else {
      j = m;
      while (!(c_v & j)) j >>= 1;
      // instead of while(0 == (c_v & (1<<j)))
      j--; if (j >= 0) c_v ^= j; // flips the jth bit
    }
    parity ^= 1; // or parity = 1 - parity;
  } while (j > 0);
  cout << endl;
}
```

Lipski proves the correctness of Algorithm 1.13 and notes that the average number of operations to generate the next codeword is constant. Therefore, this applies to both Algorithm G and Mirror algorithm G – they are CAT-algorithms.

5.2. The sequence A362160 in the OEIS

The results of running the Gen_MGC_Seq algorithm were used to create the A362160 sequence in the OEIS (the Mirror algorithm G gives the same results). The first of its terms can be seen in Figure 1, but they are not enough. The terms of $s(n)$ are generated for $n = 0, 1, \dots, 15$ and available via the section LINKS in the description of A362160. The sequences $s(0), s(1), \ldots, s(5)$ form the rows of an irregular triangle denoted by $T(n,k)$ – see section EXAMPLE of A362160. The first 5 rows of $T(n,k)$ are shown in Table 3.

Table 3. Results obtained by Algorithm Gen_MGC_Seq, for $n = 0, 1, \ldots, 4$

$n =$	$T(n,k),\ for\ k =$ 0, 1, 2, 3, 4, 5, 6, 7, …
0	0,
1	0, 1
2	0, 2, 3, 1
3	0, 4, 6, 2, 3, 7, 5, 1
4	0, 8, 12, 4, 6, 14, 10, 2, 3, 11, 15, 7, 5, 13, 9, 1

We note that A362160 does not contain only the terms of $s(n)$, even for some sufficiently large n. It is a concatenation of the rows of $T(n,k)$ and so starts with:

$$0, 0, 1, 0, 2, 3, 1, 0, 4, 6, 2, 3, 7, 5, 1, 0, 8, 12, 4, 6, 14, 10, 2, 3, 11, 15, 7, 5, \ldots$$

Representing the results by $T(n,k)$ allows one to notice more properties about $s(n)$. For example:

- The zero column is A000004 in the OEIS is named “The zero sequence”. The last term of each row is 1, the sequence of the last terms is A000012 in the OEIS: “The simplest sequence of positive numbers: the all 1's sequence”.
- Row n is a permutation of the numbers $0, 1, \ldots, 2^n - 1$ and so their sum is $2^{n-1}.(2^n - 1)$. The numbers of $T(n,k)$ taken for $n = 0, 1, 2, \ldots$, form A006516.
- The numbers in column 1 (as well as columns 3, 7, etc.) form the sequence of powers of 2, which is A000079, and so on.

Algorithm Gen_MGC_Seq and its analogous version for the Gray code (Bakoev 2023a) are representatives of algorithms that generate the sequences of integers – the serial numbers of the vectors in a certain ordering. The applications, advantages and disadvantages of this approach are discussed in (Bakoev 2023b).

The relationships between the binary vectors of $\{0,1\}^3$ in these two orderings, the corresponding serial numbers, and the subsets $A \subseteq U = \{a, b, c\}$ are illustrated in Table 4 and Table 5.

5.3. The function to reflect the coordinates of a vector

Recall Theorem 3 about the relationship between Gray code and mirror Gray code by the operation (function) φ to reflect (or reverse) the coordinates of a vector. We can use this relationship in several ways:

- It can be applied sequentially to each vector of one code to obtain the corresponding vector of the other code, and so any algorithm that generates one code can generate the other code.

Table 4. The vectors of $\{0,1\}^3$ in a Gray code, their corresponding serial numbers and subsets

$\#\alpha$	$\alpha \in \{0,1\}^3$ in a Gray code	$A \subseteq U$
0	(0,0,0)	$\emptyset$
1	(0,0,1)	$\{c\}$
3	(0,1,1)	$\{b, c\}$
2	(0,1,0)	$\{b\}$
6	(1,1,0)	$\{a, b\}$
7	(1,1,1)	$\{a, b, c\}$
5	(1,0,1)	$\{a, c\}$
4	(1,0,0)	$\{a\}$

Table 5. The vectors of $\{0,1\}^3$ in a mirror Gray code, their corresponding serial numbers and subsets

$\#\beta$	$\beta \in \{0,1\}^3$ in a mirror Gray code	$A \subseteq U$
0	(0,0,0)	$\emptyset$
4	(1,0,0)	$\{a\}$
6	(1,1,0)	$\{a, b\}$
2	(0,1,0)	$\{b\}$
3	(0,1,1)	$\{b, c\}$
7	(1,1,1)	$\{a, b, c\}$
5	(1,0,1)	$\{a, c\}$
1	(0,0,1)	$\{c\}$

- It can be incorporated into the algorithm itself so that it generates the codewords of the corresponding code (without the need to reverse each one), as we have shown through Algorithm G and Mirror algorithm G.
- The difference between the ways of numbering the array elements and the bits of the binary vectors, as in Algorithm 1.13 (Lipski 1988). Also, when the transition sequence $T(n)$ is applied to an array, the mirror Gray code is obtained, and when applied to binary vectors, the Gray code is obtained. But if we reverse the $(n-k)$th coordinate or index, instead of the kth (where k is the serial term of $T(n)$), this swaps the two resulting codes.

The implementation of the operation φ depends on the representation of codewords. At the byte-wise one, reversing the first n elements in the array `a` is simple and needs $\Theta(n)$ operations. The corresponding bitwise approach, which reflects the least significant n bits in the binary representation of the number a, looks like this:

```
int reverse_bits (int a, int n) {
  int b = a & 1; // the last bit of b is the first bit of a
  n --;   // one bit of a is already reversed
  while (n) { // loop on the remaining bits
    a >>= 1; b <<= 1; n--;
    if (a & 1) b ^= 1;
    }
    return b;  // use the least significant n bits of b
}
```

This function also performs $\Theta(n)$ operations. More efficient implementations (with $O(n)$ operations) of the bit reflect function can be seen in (Knuth 2011), (Warren 2013), (Arndt 2011), (Anderson S.2005), etc. But the reader should be careful with them, because they require more knowledge and experience.

6. The four basic functions for the mirror Gray code

Apart trough the algorithms under consideration, the mirror Gray code (as well as the Gray code) can be obtained by the functions for successor (next) and predecessor (previous), for ranking and unranking, etc., as we will see here. They are related as it is shown in Figure 2.

The successor and predecessor functions are used to generate combinatorial objects sequentially, one after the other, in straight or reverse order. Ruskey notes that "Ranking (and unranking) is generally only possible for some of the elementary combinatorial objects." (Ruskey 2003). These four functions for the Gray code are well-known, they can be found in most of the books on Combinatorial algorithms cited here. Using them, the function that reverses bits and the relations in Figure 2, we can easily create such functions for the mirror Gray code. When the sequence $s(n)$ is generated, the predecessor and successor functions are trivial. Therefore, we will not consider these ways of obtaining them. Instead, we will derive the four basic functions for the sequence $s(n)$ and its terms.

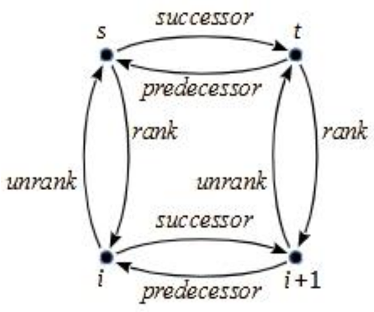


Figure 2. Four basic functions: successor, predecessor, rank and unrank

6.1. *The successor and predecessor functions for mirror Gray code*

We recall the "rule of succession" again, which is implemented in the main loop of Mirror algorithm G and so we only need to extract it. When an arbitrary term t of $s(n)$ is given, its parity is unknown and must first be computed. This can be done via the `__builtin_parity` function (for the GCC and Clang compilers), or via the `__popcnt` fuction to compute the weight of t and then check its rightmost bit. If $t = 1$, then it is the last member of $s(n)$ and has no successor. So the following function returns -1 as such indicator, and its second parameter is n.

```
int MGC_successor (int t, int n) {
    if (1 == t) return -1;
    int j= 1 << (n-1); // the most significant bit in t
    if (0 == (1 & __popcnt(t))) return t ^ j;
    while (!(t & j)) j >>= 1; j >>= 1; t ^= j;
    return t; // the next term of s(n)
}
```

As for the **predecessor** function, we simply look at the changes made to the successor function and reverse them. If t has an odd weight, then its last bit has been flipped and we flip it again to restore its previous value. Otherwise, the bit to the right of the most significant 1 is inverted. We invert this bit again and get the previous vector. Similarly, the zero vector (the number 0) is first in $s(n)$ and has no predecessor – the function again returns -1 to indicate this. Here is its code.

```
int MAG_predecessor (int t, int n) {
if (0 == t) return -1;
    int j = 1 << (n-1); //the most significant bit in t
    if (1 & __popcnt(t)) return t ^ j;
    //or if (__builtin_parity(t)) return t ^ j;
    while (!(t & j)) j >>= 1;
    j >>= 1; t ^= j;
    return t;// the preivious term of s(n)
}
```

Let us consider the time complexity of the last two algorithms. The `__popcnt` function runs in $O(\log_2 m)$ time (the same as `__builtin_parity`), where $m \in \{8, 16, 32, 64\}$ is the size of the computer word that stores n. The `while` loop runs in $O(n)$ time and hence the time complexity of these algorithms is $O(\log_2 m) + O(n) = O(n)$ for these values of m – it is best to be the smallest integer, such that $2^m > n$.

Obviously, only minimal changes to the two algorithms are needed to obtain their respective algorithms for the Gray code. Also, their byte-wise realizations for both codes are simple and easy.

6.2. The rank and unrank functions for mirror Gray code

These functions for the Gray code can be found in many sources, in some of them they have been proved. For example, formulas (9) and (10) in (Knuth 2011, p. 284), Lemma 2.3 in (Kreher & Stinson 1999, p. 40), Lemma 5.1 and Algorithm 5.1 in (Ruskey 2003, p. 119), etc. They have very efficient implementations. It is important to note that ranking the codewords of a Gray code actually means converting that order of binary vectors to their lexicographic order, and vice versa for the unrank function. So we can use these functions for the Gray code, the bijective relation between the Gray code and mirror Gray code stated by Theorem 3, and the function φ (implemented by the function `reverse_bits)`. Instead, we apply another approach to derive these functions using the recursive formula (2) for the sequence $s(n)$, explanations after it and Algorithm Gen_MGC_Seq.

Let t be any term of $s(n)$, $0 \le t \le 2^n - 1$. We want to compute its **rank**, i.e. which place (in order) it occupies in $s(n)$. So, if the n-bit binary representation of t is the vector α, we want to compute its rank – the place that occupies α in the mirror Gray code. Let us denote this place by m. So, if $s(n,m) = t$, we seek $m, 0 \le m \le 2^n - 1$, as a function of t and n. We recall that the left half of the terms of $s(n)$ are even numbers, which are the doubled terms of $s(n-1)$. The terms of the right half of $s(n)$ are odd numbers which are the reflected terms of the left half and each of them is increased by 1. Reflection also means central symmetry about an imaginary point in the middle of the sequence $s(n)$. Thus the element with number k, $0 \le k \le 2^n - 1$, in one half corresponds to the element with number $2^n - 1 - k$ in the other half. We reason like this:

- If t is an **even number**, it is the result of the “filling the left half” step in the code of Algorithm Gen_MGC_Seq. First, we check if $t = 0$ and if “yes”, its serial number is $m = 0$. Otherwise, $t = 2k > 0$ and element k has the same number m in the sequence $s(n-1)$, i.e. $t = s(n,m) = 2k = 2s(n-1,m)$. Therefore $m(n,t) = m(n-1,t/2)$.
- If t is an **odd number**, t is the result of the the “filling the right half” step in the code of Algorithm Gen_MGC_Seq. This means that t is obtained by reflecting the number $t-1$ from the left half of $s(n)$ then and adding 1. Let t occupies the place with number m in the right half of $s(n)$, i.e. $t = s(n,m)$, where $2^{n-1} \le m \le 2^n - 1$. Since t is obtained by adding 1 to the number $t-1$ whose serial number in $s(n)$ is $2^n - 1 - m$, i.e. $t = s(n, 2^n - 1 - m) + 1 = s(n,m)$, it follows that $m(n,t) = 2^n - 1 - m(n, t-1)$.But now the number $t-1$ is even, and for $m(n,t-1)$ we can apply the inference from the first case, which associates it with the sequence$s\ (n-1)$. Thus we save one recursive step and finally get:

$$m(n,t) = 2^n - 1 - m(n,t-1) = 2^n - 1 - m(n-1,(t-1)/2).$$

Therefore, the recursive definition of m by the parameters t and n is:

$$m(n,t) = \begin{cases} 0, \text{ if } t = 0, \\ m(n-1,t/2), \text{ if } t \text{ is even} \\ 2^n - 1 - m(n-1,(t-1)/2), \text{ if } t \text{ is odd.} \end{cases} \quad (4)$$

We propose the code of two functions – recursive and non-recursive, which implement formula (4). Replacing recursion with iteration is not so simple. When `t` is an odd number, the result is formed in the reverse step of the recursion. This is why the non-recursive function uses a stack represented by the `st` array. The `while` loop fills the stack with the required values and then the final value is computed by the `for` loop.

```
int rank_term_rec (int n, int t) {
  if (t& 1)
    return (1<<n)-1-rank_term_rec(n-1,(t-1)/2);
  else {
    if (0 == t) return 0;
    else return rank_term_rec(n-1, t/2);
  }
}
int rank_term (int n, int t) {
  if (0 == t) return 0;
  int len= 1 << n; // the length of s(n): 2 to the n-th power
  int st[n], i= 0;
  while (t) {
    if (t& 1) { st[i++]= len - 1;t--; }
    len >>= 1; t >>= 1;
  }
  for (int j= i-1; j > 0; j--) st[j-1] -= st[j];
  return st[0];
}
```

Formula (4) and the arguments in its derivation determine the correctness of these ranking algorithms. Their time complexity is of type $O(n)$ since the first function executes at most $n+1$ recursive calls, the same applies to the `while` loop and the `for` loop executions in the second function.

The function **unrank** must compute the term $t$that occupies the m-th consecutive position in $s(n)$, for given integers n and $m, 0 \leq m \leq 2^n - 1$. The derivation of the unrank function is analogous to that of the rank function.

1) If $m < 2^{n-1}$, then t is in the **left half** of $s(n)$. First, we check if $m = 0$; if "yes", then t is the zero term of $s(n)$, i.e. $t = 0$. Otherwise, $t = 2k > 0$ and the element k occupies position with the number m in $s(n-1)$, the same as t in $s(n)$. Then $t = s(n, m)$ and $k = s(n-1, m)$ and therefore $t = 2k = 2t(n-1, m)$.

2) If $m \geq 2^{n-1}$, then t is in the **right half** of $s(n)$. Then t is obtained by reflecting the number $t - 1$ from the left half of $s(n)$ and adding 1. The number

$t - 1$ occupies a position with the number $2^n - 1 - m$ in the left half of $s(n)$ and therefore $t = s(n,m) = s(n, 2^n - 1 - m) + 1$. Also, the number $t - 1$ is even, and then $t - 1 = 2k = s(n, 2^n - 1 - m)$. Then $t - 1$ and k the occupy the same position – with number $2^n - 1 - m$ in the sequences $s(n)$ and $s(n-1)$, respectively. Since $k < 2^{n-1}$ we save one recursive step using the first case, i.e. $k = t(n-1, 2^n - 1 - m)$. Then we have $t - 1 = 2t(n-1, 2^n - 1 - m)$, whence $t = 2t(n-1, 2^n - 1 - m) + 1$.

Thus we get the recursive formula:

$$t(n,m) = \begin{cases} 0, \text{ if } m = 0, \\ 2t(n-1,m), \text{ if } m < 2^{n-1} \\ 2t(n-1,2^n-1-m)+1, \text{ if } m \geq 2^{n-1}. \end{cases} \tag{5}$$

We propose the code of two functions – recursive and non-recursive, which implement formula (5). Their parameters and explanations are the same as for the rank function. The same is valid for their correctness and time complexity.

```
int unrank_num_rec (int n, int m) {
  if (0 == m) return 0;
  int len = 1 << n; // the length of s(n): 2 to the n-th power
  if (m < len/2)
    return 2*unrank_num_rec(n-1, m);
  else return 1 + 2*unrank_num_rec(n-1, len-1-m);
}
int unrank_number (int n, int m) {
  if (0 == m) return 0;
  int st[max_n]; // represents the stack
  int i = 0, t = 0,
      len = 1 << n;
  while (m) {
    if (m < len/2)  st[i++]= 0;
    else { st[i++] = 1; m = len - 1 - m; }
    len >>= 1;
  }
  for (int j= i -1; j >= 0; j--)
    t= 2*t + st[j];
  return t;
}
```

7. Conclusions

Here we considered the mirror Gray code, comparing it to the Gray code. We have examined and shown that the algorithms for their generation are different, and the serial number sequences of the corresponding codewords, the corresponding subsets generated by them, and their four basic functions are also different.

Finally, whether readers continue to think that both codes are the same or not, we hope that they have at least looked at it from a different perspective and gained a deeper knowledge of the Gray code.

Acknowledgments

This work was supported in part by the Bulgarian National Science Fund under Contract No KP-06-H62/2-2022.

NOTES

1. *The On-line Encyclopedia of Integer Sequences*. https://oeis.org/

REFERENCES

ANDERSON, J., 2001. *Discrete Mathematics with Combinatorics*, New Jersey: Prentice-Hall.

ANDERSON, S., *Bit Twiddling Hacks*. [viewed 26 Nov. 2023]. Available from: https://graphics.stanford.edu/~seander/bithacks.html

ARNDT, J., 2011. *Matters Computational: Ideas, Algorithms, Source Code,* Springer.

BAKOEV, V., 2023a. *Generating sets in lexicographic order and with minimal change*, "St. Cyril and St. Methodius" University Publishing House, Veliko Tarnovo (In Bulgarian, БАКОЕВ, В., *Генериране на множества в лексикографска наредба и чрез минимално изменение*, Университетско издателство „Св. св. Кирил и Методий", В. Търново, 2023).

BAKOEV, V., 2023b. *Binary vectors: orderings, integer sequences, and generating of sets*, "St. Cyril and St. Methodius" University Publishing House, Veliko Tarnovo (In Bulgarian, БАКОЕВ, В., *Двоични вектори: наредби, числови редици и генериране на множества*, Университетско издателство „Св. св. Кирил и Методий", В. Търново, 2023).

GARNIER, R., TAYLOR, J., 2002. *Discrete Mathematics for New Technology*, 2nd ed. IOP Publishing Ltd.

GRIMALDI, R.,2004. *Discrete and Combinatorial Mathematics. An Applied Introduction*, 5th ed., Addison-Wesley.

GULLIVER, T.A., BHARGAVA, V.K., STEIN, J.M., 1999. Q-ary Gray codes and weight distributions, *Applied Mathematics and Computation*, no. 103, pp. 97 – 109.

KNUTH, D., 2011. *The art of computer programming, Volume 4A: Combinatorial Algorithms, Part 1*, Addison-Wesley.

KOSHY, T., 2003. *Discrete Mathematics with Applications*, Academic Press.

KREHER, D., STINSON, D., 1999. *Combinatorial algorithms: generation, enumeration and search*, CRC Press.

LIPSKI, W., *Combinatorics for programmers*, Mir, Moscow (In Russian, ЛИПСКИЙ, В., 1988. *Комбинаторика для программистов*, Мир, Москва).

MACWILLIAMS, F.J., SLOANE, N.J.A., 1978. *The Theory of Error-Correcting Codes*, Amsterdam: North-Holland.

MANEV, K., 2012. *Introduction to discrete mathematics*, 5th ed., KLMN, Sofia (МАНЕВ, К., 2012. *Увод в дискретната математика*, V изд., КЛМН, София)

MÜUTZE, T., 2022. *Combinatorial Gray Codes – An Updated Survey*, [viewed 26 Nov. 2023]. Available from: https://arxiv.org/abs/2202.01280

NIJENHUIS, A., WILF, H., 1978. *Combinatorial Algorithms for Computers and Calculators*, (1st ed., 1975), 2nd ed., Academic Press.

ROSEN, K., 2012. *Discrete Mathematics and its Applications*, 7th edition, McGraw-Hill.

RUSKEY, F., 2003. *Combinatorial Generation. Working Version* (1j-CSC 425/ 520), [viewed 26 Nov. 2023]. Available from: http://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf

SAVAGE, C., 1997. A Survey of Combinatorial Gray Codes, *SIAM Review*, vol. 39, no. 4, pp. 605 – 629.

SUPARTA, I.N., 2006. *Counting sequences, Gray codes and Lexicodes*, Dissertation at Delft University of Technology. [viewed 26 Nov. 2023]. Available from: https://theses.eurasip.org/theses/113/counting-sequences-gray-codes-and-lexicodes/download/

WARREN, H.S.Jr., 2013. *Hacker's Delight*, 2nd Ed., Addison-Wesley.

☒ **Dr. Valentin Bakoev, Assoc. Prof.**
ORCID iD: 0000-0003-2503-5325
"St. Cyril and St. Methodius" University
Veliko Tarnovo, Bulgaria
E-mail: v.bakoev@ts.uni-vt.bg