2

Generating Elementary Combinatorial Objects

2.1 Combinatorial generation

Often it is necessary to find nice algorithms to solve problems such as generating all the subsets of a given set S of size n, say. Related problems include generating all the permutations of S, or all the k-subsets of S.

Among the generation algorithms we will study are those that generate the desired objects in a *lexicographic order*, and the so-called *minimal change algorithms*, in which each object is generated from the previous one by performing a very small change. Both of these types of *sequential generation* are often accomplished by means of a *successor* algorithm, which is used to find the next object following a given one, with respect to a given ordering.

As well as sequential generation, we will be interested in *ranking* and *unranking* algorithms. A ranking algorithm determines the position (or rank) of a combinatorial object among all the objects (with respect to a given order); an unranking algorithm finds the object having a specified rank. Thus, ranking and unranking can be considered as inverse operations.

Here are slightly more formal mathematical descriptions of these concepts. Suppose that $\mathcal{S}$ is a finite set and $N = |\mathcal{S}|$. A ranking function will be a bijection

$$\mathsf{rank} : \mathcal{S} \to \{0, \ldots, N-1\}.$$

A rank function defines a total ordering on the elements of $\mathcal{S}$, by the obvious rule

$$s < t \Leftrightarrow \mathsf{rank}(s) < \mathsf{rank}(t).$$

Conversely, there is a unique rank function associated with any total ordering defined on $\mathcal{S}$.

If rank is a ranking function defined on $\mathcal{S}$, then there is a unique unranking function associated with the function rank. This function unrank is also a bijection,

$$\mathsf{unrank} : \{0, \ldots, N-1\} \to \mathcal{S}.$$

unrank is the inverse function of the function rank, i.e., we have

$$\mathsf{rank}(s) = i \Leftrightarrow \mathsf{unrank}(i) = s,$$

for all $s \in \mathcal{S}$ and all $i \in \{0, \ldots, N-1\}$.

Efficient ranking and unranking algorithms have several potential uses. We mention a couple now. One application is the generation of random objects from a specified set $\mathcal{S}$. This can be done easily by generating a random integer $i \in \{0, \ldots, N-1\}$, where $N = |\mathcal{S}|$, and then unranking i. This algorithm ensures that every element of $\mathcal{S}$ is chosen with equal probability $1/N$, assuming that the random number generator being used is unbiased. For an example of the use of such an algorithm, see Algorithm 4.20.

Another use of ranking and unranking algorithms is in storing combinatorial objects in the computer. Instead of storing a combinatorial structure, which could be quite complicated, an alternative would be to simply store its rank, which of course is just an integer. If the structure is needed at any time, then it can be recovered by using the unranking algorithm.

Given a ranking function, rank, defined on $\mathcal{S}$, the successor function, which we name successor, satisfies the following rule:

$$\mathsf{successor}(s) = t \Leftrightarrow \mathsf{rank}(t) = \mathsf{rank}(s) + 1.$$

Thus, successor(s) is the next element following s in the total ordering. We will use the convention that successor(s) is undefined if $\mathsf{rank}(s) = N-1$ (i.e., if s is the last (largest) element in $\mathcal{S}$).

The function successor can easily be constructed from the functions rank and unrank, according to the following formula:

$$\mathsf{successor}(s) = \begin{cases} \mathsf{unrank}(\mathsf{rank}(s)+1) & \text{if } \mathsf{rank}(s) < N-1 \\ \text{undefined} & \text{if } \mathsf{rank}(s) = N-1. \end{cases}$$

However, for a given set $\mathcal{S}$ under consideration, it may be that there is a more efficient way to construct a successor function.

Once we have constructed a successor function, it is a simple matter to generate all the elements in $\mathcal{S}$. We would do this by beginning with the first element of $\mathcal{S}$, and applying the function successor $N-1$ times.

2.2 Subsets

2.2.1 Lexicographic ordering

Suppose that n is a positive integer, and $S = \{1, \ldots, n\}$. Define $\mathcal{S}$ to consist of the 2^n subsets of $S = \{1, \ldots, n\}$. We begin by describing how to generate the subsets in $\mathcal{S}$ in lexicographic order.

Given a subset $T \subseteq S$, let us define the *characteristic vector* of T to be the n-tuple

$$\chi(T) = [x_{n-1}, \ldots, x_0],$$

where

$$x_i = \begin{cases} 1 & \text{if } n-i \in T \\ 0 & \text{if } n-i \notin T. \end{cases}$$

This method of labeling the coordinates $x_{n-1}, \ldots, x_0$ is convenient for the purposes of the algorithms we are going to describe. It is essentially the same method that we used in Section 1.7.1, except that here we are taking the base set to be $\{1, \ldots, n\}$.

Next, define the *lexicographic ordering* on the set of subsets of S to be that induced by the lexicographic ordering of the characteristic vectors. If we think of these characteristic vectors as being the binary representations of the integers from 0 to $2^n - 1$, then this ordering corresponds to the usual ordering of the integers. With respect to this ordering, the rank of a subset T, denoted $\mathsf{rank}(T)$, is just the integer whose binary representation is $\chi(T)$. That is,

$$\mathsf{rank}(T) = \sum_{i=0}^{n-1} x_i 2^i.$$

We illustrate by taking $n = 3$, and tabulating the eight subsets of $S = \{1, 2, 3\}$:

T	$\chi(T) = [x_2, x_1, x_0]$	$\mathsf{rank}(T)$
$\emptyset$	[0, 0, 0]	0
$\{3\}$	[0, 0, 1]	1
$\{2\}$	[0, 1, 0]	2
$\{2, 3\}$	[0, 1, 1]	3
$\{1\}$	[1, 0, 0]	4
$\{1, 3\}$	[1, 0, 1]	5
$\{1, 2\}$	[1, 1, 0]	6
$\{1, 2, 3\}$	[1, 1, 1]	7

We now present ranking and unranking algorithms for lexicographic generation of subsets. These are very simple. As mentioned above, ranking a subset $T \subseteq \{1, \ldots, n\}$ consists of computing the integer whose binary representation is $\chi(T)$. Unranking an integer r, where $0 \leq r \leq 2^n - 1$, requires the computation of the subset T having rank r. These algorithms are described below without reference to the characteristic vectors $\chi(T)$.

Algorithm 2.1: SUBSETLEXRANK (n, T)

$r \leftarrow 0$
for $i \leftarrow 1$ **to** n
  do { **if** $i \in T$
    then $r \leftarrow r + 2^{n-i}$
return (r)

Algorithm 2.2: SUBSETLEXUNRANK (n, r)

$T \leftarrow \emptyset$
for $i \leftarrow n$ **downto** 1
  do { **if** $r \bmod 2 = 1$
    then $T \leftarrow T \cup \{i\}$
    $r \leftarrow \lfloor \frac{r}{2} \rfloor$
return (T)

As an example, suppose that $n = 8$ and $T = \{1, 3, 4, 6\}$. Then Algorithm 2.1 computes

$$\begin{aligned} \mathsf{rank}(T) &= 2^7 + 2^5 + 2^4 + 2^2 \\ &= 128 + 32 + 16 + 4 \\ &= 180. \end{aligned}$$

Conversely, if we run Algorithm 2.2 with $n = 8$ and $r = 180$, then we obtain the following:

i	r	$r \bmod 2$	T
8	180	0	$\emptyset$
7	90	0	$\emptyset$
6	45	1	$\{6\}$
5	22	0	$\{6\}$
4	11	1	$\{4, 6\}$
3	5	1	$\{3, 4, 6\}$
2	2	0	$\{3, 4, 6\}$
1	1	1	$\{1, 3, 4, 6\}$.

This example illustrates that ranking and unranking are inverse operations, since the subset $\{1, 3, 4, 6\}$ has rank 180 and unranking 180 produces the set $\{1, 3, 4, 6\}$.

We have assumed in this section that our base set is $S = \{1, \ldots, n\}$. What would we do if we wanted to rank and unrank the subsets of some other n-element set, say S'? We could of course design algorithms for ranking and unranking

subsets of S', but a different approach is usually more convenient. It suffices to construct a bijection $\phi : S' \to S$. Now we can rank any subset $X \subseteq S'$, using a rank function for subsets of S, from the following formula:

$$\mathsf{rank}(X) = \mathsf{rank}(\phi(X)).$$

Similarly, we can unrank r to a subset of S', using the following formula:

$$\mathsf{rank}(r) = \phi^{-1}(\mathsf{unrank}(r)).$$

In the formula above, ϕ^{-1} is the inverse function of ϕ, i.e., $\phi(X) = Y$ if and only if $\phi^{-1}(Y) = X$, where $X \subseteq S'$ and $Y \subseteq S$.

For example if we want to rank and unrank subsets of $S' = \{0, \ldots, n-1\}$, then we can use the bijections ϕ and ϕ^{-1} defined by the following formulas:

$$\phi(X) = \{i + 1 : i \in X\}$$

and

$$\phi^{-1}(Y) = \{i - 1 : i \in Y\}.$$

2.2.2 Gray codes

The lexicographic ordering defined above makes ranking and unranking very simple, but the ordering is not well suited to the sequential generation of all 2^n subsets of an n-set. This is because subsets that are consecutive with respect to the ordering can be very "different." For example, in the case $n = 3$ considered above, $\mathsf{rank}(\{2,3\}) = 3$ and $\mathsf{rank}(\{1\}) = 4$. Hence, we have two consecutive subsets that are in fact complements of each other (so they are as different as they could possibly be).

Given two subsets $T_1, T_2 \subseteq S$, we define the *symmetric difference* of T_1 and T_2, denoted $T_1 \Delta T_2$, to be

$$T_1 \Delta T_2 = (T_1 \backslash T_2) \cup (T_2 \backslash T_1).$$

The *distance* between T_1 and T_2 is defined to be

$$\mathsf{dist}(T_1, T_2) = |T_1 \Delta T_2|.$$

Alternatively, $\mathsf{dist}(T_1, T_2)$ is equal to the number of coordinates in which $\chi(T_1)$ and $\chi(T_2)$ have different entries, which is called the *Hamming distance* between the vectors $\chi(T_1)$ and $\chi(T_2)$. The relevance of the distance between two subsets is that it represents the number of elements that need to be added to and/or deleted from one subset in order to obtain the other.

If we are going to generate all 2^n subsets sequentially, it might be desirable to do so in such a way that any two consecutive subsets have distance one (the smallest possible). This means that any subset can be obtained from the previous one

by either deleting a single element or adding a single element. Such an ordering of the 2^n subsets of an n-set will be called a *minimal change ordering*.

As an example in the case $n = 3$, the ordering

$$\emptyset, \{3\}, \{2,3\}, \{2\}, \{1,2\}, \{1,2,3\}, \{1,3\}, \{1\}$$

is a minimal change ordering.

The characteristic vectors of the subsets in a minimal change ordering form a structure that is known as a *Gray code*. Thus, a Gray code is an ordering of the 2^n binary vectors of length n in such a way that any two consecutive vectors have Hamming distance equal to one.

From the minimal change ordering presented above, the following Gray code is obtained:

$$000, 001, 011, 010, 110, 111, 101, 100.$$

There is another way to formulate the concept of minimal change coverings or Gray codes. Consider the n-dimensional unit cube, whose 2^n vertices are labeled by the 2^n binary vectors. The edges of this cube join vertices having Hamming distance equal to one. Thus, a Gray code is nothing more than a *Hamiltonian path* in the n-dimensional unit cube, i.e., a method traversing the edges of the cube so that each vertex is visited exactly once. Examples are given in Figure 2.1.

There has been a considerable amount of study done on different constructions for Gray codes. We will look at a particularly nice class of Gray codes called the *binary reflected Gray codes*. G^n will denote the binary reflected Gray code for the 2^n binary n-tuples, and it will be written as a list of 2^n vectors, as follows:

$$\mathsf{G}^n = [\mathsf{G}_0^n, \mathsf{G}_1^n, \ldots, \mathsf{G}_{2^n-1}^n].$$

The codes G^n are defined recursively. The first one, G^1, is defined to be

$$\mathsf{G}^1 = [0, 1].$$

Given G^{n-1}, the Gray code G^n is defined to be

$$\mathsf{G}^n = \left[0\mathsf{G}_0^{n-1}, \ldots, 0\mathsf{G}_{2^{n-1}-1}^{n-1}, 1\mathsf{G}_{2^{n-1}-1}^{n-1}, \ldots, 1\mathsf{G}_0^{n-1}\right].$$

Equivalently, we have that

$$\mathsf{G}_i^n = \begin{cases} 0\mathsf{G}_i^{n-1} & \text{if } 0 \le i \le 2^{n-1}-1 \\ 1\mathsf{G}_{2^n-1-i}^{n-1} & \text{if } 2^{n-1} \le i \le 2^n-1. \end{cases}$$

The code G^n is constructed from G^{n-1} in two steps. First, we take a copy of G^{n-1} with a "0" prepended to each vector. Then we take a copy of G^{n-1} in reverse order, with a "1" prepended to each vector. The fact that the second copy of G^{n-1} is in reverse order is the reason for the name "reflected."

The next two Gray codes produced by this recipe are

$$\mathsf{G}^2 = [00, 01, 11, 10]$$

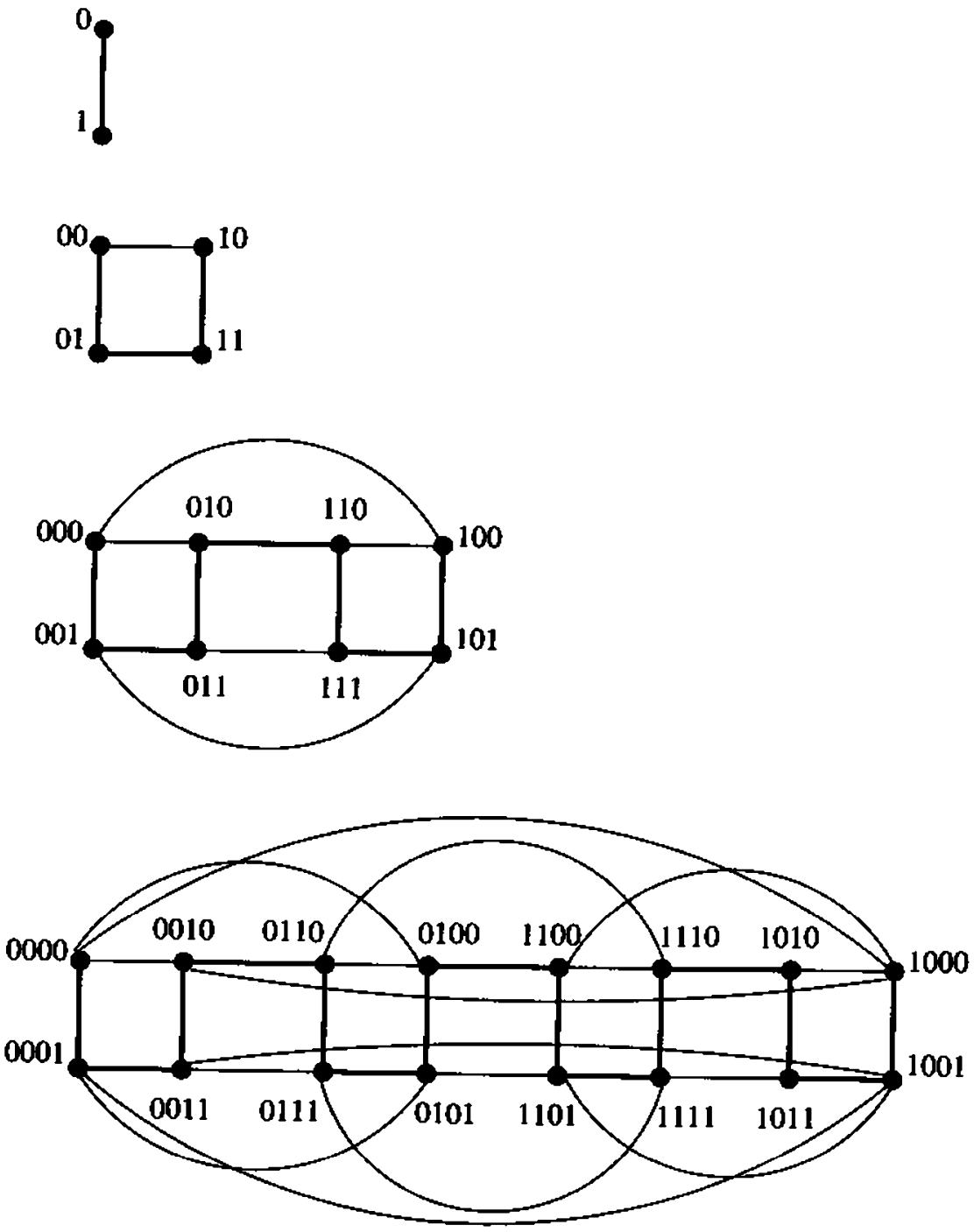

FIGURE 2.1
The evolution of the binary reflected Gray code.

and

$$\mathsf{G}^3 = [000, 001, 011, 010, 110, 111, 101, 100].$$

Figure 2.1 depicts the binary reflected Gray codes $\mathsf{G}^1, \ldots, \mathsf{G}^4$.

Our first result is to prove that any G^n is a Gray code. Since the codes G^n are defined recursively, it is most natural to prove the result by induction on n.

THEOREM 2.1 *For any integer $n \geq 1$, G^n is a Gray code.*

PROOF The proof is by induction on n. For $n = 1$, the result is easily seen to be true. As an induction hypothesis, suppose that G^{n-1} is a Gray code, for some integer $n \geq 2$. We will prove that G^n is a Gray code.

First, it is clear that G^n contains all 2^n binary n-tuples. By induction, the 2^{n-1} n-tuples that begin with "0" are contained in the first half of G^n, and the 2^{n-1} n-tuples that begin with "1" are contained in the second half of G^n.

It remains to verify the minimal change property. Consider two consecutive n-tuples in G^n, say G^n_i and G^n_{i+1}. There are in fact three cases to consider, depending on the value of i.

First, if $0 \leq i \leq 2^{n-1} - 2$, then G^n_i and G^n_{i+1} are formed from two consecutive $(n-1)$-tuples in G^{n-1} by prepending a "0" to each of them. Therefore, by induction, G^n_i and G^n_{i+1} have Hamming distance equal to 1.

The second case is $2^{n-1} \leq i \leq 2^n - 2$. This is similar to the first case. This time, G^n_i and G^n_{i+1} are formed from two consecutive $(n-1)$-tuples in G^{n-1} by prepending a "1" to each of them. Therefore, by induction, G^n_i and G^n_{i+1} have Hamming distance equal to 1.

The final case is $i = 2^{n-1} - 1$. $\mathsf{G}^n_{2^{n-1}-1}$ and $\mathsf{G}^n_{2^{n-1}}$ are both formed from $\mathsf{G}^{n-1}_{2^{n-1}-1}$, by prepending a "0" and a "1" respectively. Therefore, $\mathsf{G}^n_{2^{n-1}-1}$ and $\mathsf{G}^n_{2^{n-1}}$ have Hamming distance equal to 1. (Note that this last case works out precisely because of the fact that the second copy of G^{n-1} in G^n is in reverse order.)

By induction on n, the result is true for all integers $n \geq 1$. ∎

We now present Algorithm 2.3, which computes the successor function for the binary reflected gray Code G^n. Suppose that the binary vector $A = [a_{n-1}, \ldots, a_0]$ represents the set $T \subseteq \{1, \ldots, n\}$. Then $a_i = 1$ if $n - i \in T$ and $a_i = 0$ otherwise. Let $\mathsf{w}(A)$ denote the *Hamming weight* of A (i.e., the number of "1"s in the vector A); note that $\mathsf{w}(A) = |T|$.

Algorithm 2.3 works as follows. If $\mathsf{w}(A)$ is even, then the last bit of A (namely, a_0) is flipped; if $\mathsf{w}(A)$ is odd, then we find the first "1" from the right, and flip the next bit (to the left). The last vector in G^n, which has no successor, is $[1, 0, \ldots, 0]$. This corresponds to the set $\{1\}$.

Algorithm 2.3 is described in terms of the set T. It could alternatively be presented in terms of the binary vector A, if desired. The operation "Δ", which denotes the symmetric difference of two sets, was defined earlier.

Algorithm 2.3: GRAYCODESUCCESSOR (n, T)

```
if |T| is even
  then U ← TΔ{n}
  else { j ← n
         while j ∉ T and j > 0
           do j ← j − 1
         if j = 1
           then return ("undefined")
         U ← TΔ{j − 1}
         return (U)
```

The following theorem can be proved by induction on n.

THEOREM 2.2 *Algorithm 2.3 computes the function* successor *for the Gray code* G^n.

We now proceed to develop ranking and unranking algorithms for the binary reflected Gray code. These algorithms depend on certain relationships between the binary representations of the integers $r = 0, \ldots, 2^n - 1$ and the corresponding vectors G_r^n. Let us begin by tabulating these in the case $n = 3$:

r	binary representation of r	G_r^3
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

For an integer r such that $0 \leq r \leq 2^n - 1$, suppose that its binary representation is written as

$$b_n b_{n-1} \ldots b_1 b_0.$$

In other words,

$$r = \sum_{i=0}^{n} b_i 2^i,$$

and $b_n = 0$ since $r \leq 2^n - 1$. Also, suppose we write the vector G_r^n in the form

$$\mathsf{G}_r^n = a_{n-1} \ldots a_1 a_0,$$

as in Algorithm 2.3.

The relations in the next lemma will form the basis of the ranking and unranking algorithms.

LEMMA 2.3 *Suppose that $n \geq 1$ is an integer, $0 \leq r \leq 2^n - 1$, and suppose that $b_n, \ldots, b_0$ and $a_{n-1}, \ldots, a_0$ are as defined above. Then*

$$a_j \equiv (b_j + b_{j+1}) \bmod 2 \tag{2.1}$$

and

$$b_j \equiv \sum_{i=j}^{n-1} a_i \bmod 2, \tag{2.2}$$

for $j = 0, 1, \ldots, n-1$.

PROOF We begin by proving that Equation (2.1) is true for all $n \geq 1$ and $j = 0, 1, \ldots, n-1$. The proof is by induction on n. The induction can be started with $n = 1$, where Equation (2.1) can be verified easily.

For some integer $i \geq 2$ assume that Equation (2.1) is true for $n = i - 1$ and $0 \leq j \leq i - 2$. We now consider $n = i$ and $0 \leq j \leq i - 1$. Let r be an integer such that $0 \leq r \leq 2^i - 1$. We divide the proof into two cases, depending on the value of r.

The first case is when $0 \leq r \leq 2^{i-1} - 1$. In this case, we have $b_{i-1} = 0$ and $a_{i-1} = 0$. For $0 \leq j \leq i - 2$, Equation (2.1) is true by induction. For $j = i - 1$, we have

$$b_{i-1} + b_i \equiv 0 \bmod 2$$

and $a_{i-1} = 0$, so Equation (2.1) is true here as well.

Now, we proceed to the second case, $2^{i-1} \leq r \leq 2^i - 1$. In this case, we have that $a_{i-1} = 1, b_{i-1} = 1$,

$$G^{i-1}_{2^i-1-r} = a_{n-2} \ldots a_1 a_0,$$

and the binary representation of $2^i - 1 - r$ is

$$0(1 - b_{n-2}) \ldots (1 - b_0).$$

Since Equation (2.1) is true for $n = i - 1$ by induction, we have

$$a_j \equiv (1 - b_j) + (1 - b_{j+1}) \bmod 2$$

for $j = 0, 1 \ldots, i - 2$. Since

$$(1 - b_j) + (1 - b_{j+1}) \equiv (b_j + b_{j+1}) \bmod 2,$$

Equation (2.1) is true for $n = i$ and $j = 0, 1 \ldots, i - 2$. For $j = i - 1$, we have

$$b_{i-1} + b_i \equiv 1 \bmod 2$$

and $a_{i-1} = 1$, so Equation (2.1) is true here as well.

By induction, Equation (2.1) is true for $j = 0, 1, \ldots, n-1$, for all integers $n \geq 1$.

To complete the proof, we show that, for any $n \geq 1$, the truth of Equation (2.1) for $j = 0, 1, \ldots, n-1$ implies the truth of Equation (2.2) for $j = 0, 1, \ldots, n-1$. This is an easy computation:

$$\begin{aligned}\sum_{i=j}^{n-1} a_i &\equiv \sum_{i=j}^{n-1} (b_i + b_{i+1}) \bmod 2 \\ &\equiv (b_j + b_n) \bmod 2 \\ &\equiv b_j \bmod 2,\end{aligned}$$

since $b_n = 0$. ∎

The relations in Lemma 2.3 give rise to the ranking and unranking algorithms for the binary reflected Gray code which are presented as Algorithms 2.4 and 2.5. We provide brief explanations.

First, consider unranking. In iteration i of the **for** loop of Algorithm 2.5, b corresponds to b_{i+1} and b' corresponds to b_i. The algorithm successively computes $b_{n-1}, \ldots, b_0$, which are the bits in the binary representation of r. Recalling that $a_i \equiv b_i + b_{i+1} \bmod 2$, we see that

$$n - i \in T \Leftrightarrow a_i = 1 \Leftrightarrow b \neq b'.$$

Now we look at ranking. In iteration i of the **for** loop of Algorithm 2.4, b corresponds to b_i. Initially, $b = 0$ (corresponding to $b_n = 0$). Since

$$b_i = b_{i+i} + a_i \bmod 2,$$

we can update b during each iteration of the **for** loop by checking if $n - i \in T$ (since $a_i = 1$ if $n - i \in T$ and $a_i = 0$ if $n - i \notin T$). Whenever $b = 1$, we add 2^i to r, since $b = b_i$ is just bit i in the binary representation of r.

Algorithm 2.4: GRAYCODERANK (n, T)

$r \leftarrow 0$
$b \leftarrow 0$
for $i \leftarrow n-1$ **downto** 0
do { **if** $n - i \in T$
  then $b \leftarrow 1 - b$
if $b = 1$
  then $r \leftarrow r + 2^i$
return (r)

Algorithm 2.5: GRAYCODEUNRANK (n, r)

$T \leftarrow \emptyset$
$b' \leftarrow 0$
for $i \leftarrow n-1$ **downto** 0
do $\begin{cases} b \leftarrow \lfloor \frac{r}{2^i} \rfloor \\ \textbf{if } b \neq b' \\ \quad \textbf{then } T \leftarrow T \cup \{n-i\} \\ b' \leftarrow b \\ r \leftarrow r - b2^i \end{cases}$
return (T)

Let's work out a couple of examples to illustrate Algorithms 2.4 and 2.5. Suppose that $n = 8$ and $T = \{1,2,3,4,5,7,8\}$. We first use Algorithm 2.4 to compute rank(T).

i	2^i	$n-i \in T$?	b	r
7	128	yes	1	128
6	64	yes	0	128
5	32	yes	1	160
4	16	yes	0	160
3	8	yes	1	168
2	4	no	1	172
1	2	yes	0	172
0	1	yes	1	173

Thus, rank$(T) = 173$. The inverse algorithm, Algorithm 2.5, can be used to compute unrank(173). It executes as follows:

b'	r	i	2^i	b	T
0	173	7	128	1	$\{1\}$
1	45	6	64	0	$\{1,2\}$
0	45	5	32	1	$\{1,2,3\}$
1	13	4	16	0	$\{1,2,3,4\}$
0	13	3	8	1	$\{1,2,3,4,5\}$
1	5	2	4	1	$\{1,2,3,4,5\}$
1	1	1	2	0	$\{1,2,3,4,5,7\}$
0	1	0	1	1	$\{1,2,3,4,5,7,8\}$

Hence, unrank(173) $= \{1,2,3,4,5,7,8\}$.

2.3 k-Element subsets

2.3.1 Lexicographic ordering

Suppose that n is a positive integer, and $S = \{1, \ldots, n\}$. Define $\mathcal{S}$ to consist of the $\binom{n}{k}$ k-element subsets of S. We begin by describing how to generate the subsets in $\mathcal{S}$ in lexicographic order.

A k-element subset $T \subseteq S$ can be written in a natural way as a list

$$\vec{T} = [t_1, t_2, \ldots, t_k],$$

where

$$t_1 < t_2 < \cdots < t_k.$$

The lexicographic ordering on $\mathcal{S}$ is induced by the lexicographic ordering on the sequences $\vec{T}$ $(T \in \mathcal{S})$.

We illustrate with a small example. Let $n = 5$ and $k = 3$. The lexicographic ordering of the ten 3-element subsets $T \subseteq \{1, \ldots, 5\}$ is as follows:

T	$\vec{T}$	rank(T)
$\{1,2,3\}$	$[1,2,3]$	0
$\{1,2,4\}$	$[1,2,4]$	1
$\{1,2,5\}$	$[1,2,5]$	2
$\{1,3,4\}$	$[1,3,4]$	3
$\{1,3,5\}$	$[1,3,5]$	4
$\{1,4,5\}$	$[1,4,5]$	5
$\{2,3,4\}$	$[2,3,4]$	6
$\{2,3,5\}$	$[2,3,5]$	7
$\{2,4,5\}$	$[2,4,5]$	8
$\{3,4,5\}$	$[3,4,5]$	9

It is fairly straightforward to describe a successor algorithm for $\mathcal{S}$. This algorithm is presented in Algorithm 2.6.

Algorithm 2.6: KSUBSETLEXSUCCESSOR $(\vec{T}, k, n)$

$\vec{U} \leftarrow \vec{T}$
$i \leftarrow k$
while $(i \geq 1)$ **and** $(t_i = n - k + i)$
  do $i \leftarrow i - 1$
if $i = 0$
  then return ("undefined")
  else { **for** $j \leftarrow i$ **to** k
    do $u_j \leftarrow t_i + 1 + j - i$
    return $(\vec{U})$

To construct a ranking algorithm, we need to count the number of k-element subsets preceding a given set T in this ordering. Suppose that t_1 is an integer such that $1 \leq t_1 \leq n$. It is easy to see that there are exactly $\binom{n-t_1}{k-1}$ subsets $X \in \mathcal{S}$ such that $x_1 = t_1$, where $\vec{X} = [x_1, \ldots, x_k]$. More generally, for any $i \leq k$ integers $t_1, \ldots, t_i$ such that $1 \leq t_1 < \cdots < t_i \leq n$, there are exactly $\binom{n-t_i}{k-i}$ subsets $X \in \mathcal{S}$ such that $x_1 = t_1, \ldots,$ and $x_i = t_i$.

Now, suppose that $T \in \mathcal{S}$, and $\vec{T} = [t_1, t_2, \ldots, t_k]$ is defined as above. The k-element subsets X preceding T in lexicographic order are the following:

- The subsets X with $1 \leq x_1 \leq t_1 - 1$.
- The subsets X with $x_1 = t_1$ and $t_1 + 1 \leq x_2 \leq t_2 - 1$.
- The subsets X with $x_1 = t_1$, $x_2 = t_2$, and $t_2 + 1 \leq x_3 \leq t_3 - 1$.
- etc.
- The subsets X with $x_1 = t_1, x_2 = t_2, \ldots, x_{k-1} = t_{k-1}$ and $t_{k-1} + 1 \leq x_k \leq t_k - 1$.

From these facts, we can write down a formula for rank(T), where $\vec{T} = [t_1, t_2, \ldots, t_k]$. We get the following formula, where we define $t_0 = 0$ for convenience:

$$\text{rank}(T) = \sum_{i=1}^{k} \sum_{j=t_{i-1}+1}^{t_i - 1} \binom{n-j}{k-i}.$$

This formula immediately yields a ranking algorithm, which we present as Algorithm 2.7.

Algorithm 2.7: KSUBSETLEXRANK $(\vec{T}, k, n)$

$r \leftarrow 0$
$t_0 \leftarrow 0$
for $i \leftarrow 1$ **to** k
do { **if** $t_{i-1} + 1 \leq t_i - 1$
then { **for** $j \leftarrow t_{i-1} + 1$ **to** $t_i - 1$
do $r \leftarrow r + \binom{n-j}{k-i}$
return (r)

Now we unravel Algorithm 2.7 to obtain an unranking algorithm. Suppose that $0 \leq r \leq \binom{n}{k} - 1$, and suppose that $T = \text{unrank}(r)$ with $\vec{T} = [t_1, \ldots, t_k]$. The smallest element in T, t_1, can be determined by the observation that

$$t_1 = x \Leftrightarrow \sum_{j=1}^{x-1} \binom{n-j}{k-1} \leq r < \sum_{j=1}^{x} \binom{n-j}{k-1}.$$

Having determined t_1, we can compute t_2 in a similar way:

$$t_2 = x \Leftrightarrow \sum_{j=t_1+1}^{x-1} \binom{n-j}{k-2} \leq r - \sum_{j=1}^{t_1-1} \binom{n-j}{k-1} < \sum_{j=t_1+1}^{x} \binom{n-j}{k-2}.$$

The pattern continues, and the entire algorithm is presented as Algorithm 2.8.

Algorithm 2.8: KSUBSETLEXUNRANK (r, k, n)

$x \leftarrow 1$
for $i \leftarrow 1$ **to** k
do {
  while $\binom{n-x}{k-i} \leq r$
    do { $r \leftarrow r - \binom{n-x}{k-i}$; $x \leftarrow x + 1$ }
  $t_i \leftarrow x$
  $x \leftarrow x + 1$ }
return $(\overrightarrow{T})$

2.3.2 Co-lex ordering

There is a useful alternative to the lexicographic ordering for k-element subsets of an n-set. The ordering is called the *co-lex ordering*, and it is defined as follows. A k-element subset $T \subseteq S$ is written as a list

$$\overleftarrow{T} = [t_1, t_2, \ldots, t_k],$$

where

$$t_1 > t_2 > \cdots > t_k.$$

The co-lex ordering is induced by the lexicographic ordering on the sequences $\overleftarrow{T}$ $(T \in S)$.

We illustrate the co-lex ordering when $n = 5$ and $k = 3$. The co-lex ordering of the ten 3-element subsets of $\{1, \ldots, 5\}$ is as follows:

T	$\overleftarrow{T}$	rank(T)
$\{1,2,3\}$	$[3,2,1]$	0
$\{1,2,4\}$	$[4,2,1]$	1
$\{1,3,4\}$	$[4,3,1]$	2
$\{2,3,4\}$	$[4,3,2]$	3
$\{1,2,5\}$	$[5,2,1]$	4
$\{1,3,5\}$	$[5,3,1]$	5
$\{2,3,5\}$	$[5,3,2]$	6
$\{1,4,5\}$	$[5,4,1]$	7
$\{2,4,5\}$	$[5,4,2]$	8
$\{3,4,5\}$	$[5,4,3]$	9