



Combinatorial algorithms

computing subset rank and unrank, Gray codes,
 k -element subset rank and unrank,
computing permutation rank and unrank

Jiří Vyskočil, Radek Mařík

2012

Combinatorial Generation

■ definition:

Suppose that S is a finite set. A *ranking function* will be a bijection

$$\text{rank}: S \rightarrow \{0, \dots, |S| - 1\}$$

and unrank function is an inverse function to rank function.

■ definition:

Given a ranking function rank , defined on S , the successor function satisfies the following rule:

$$\text{successor}(s) = t \iff \text{rank}(t) = \text{rank}(s) + 1$$

■ potential uses:

- storing combinatorial objects in the computer instead of storing a combinatorial structure which could be quite complicated
- generation of random objects from S ensuring equal probability $1/|S|$

Subsets

- Suppose that n is a positive integer and $S = \{1, \dots, n\}$.
- Define \mathcal{S} to consist of the 2^n subsets of S .
- Given a subset $T \subseteq S$, let us define the *characteristic vector* of T to be the one-dimensional binary array

$$\chi(T) = [x_{n-1}, x_{n-2}, \dots, x_0]$$

where

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

Subsets

- Example of the lexicographic ordering on subsets of $S = \{1,2,3\}$:

T	$\chi(T) = [x_2, x_1, x_0]$	$rank(T)$
\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

■ computing the subset rank over lexicographical ordering

1) **Function** SUBSETLEXRANK(size n ; set T) : rank

2) $r = 0$;

3) **for** $i = 1$ **to** n **do** {

4) **if** $i \in T$ **then** $r = r + 2^{n-i}$;

5) }

6) **return** r

1) **Function** SUBSETLEXUNRANK(size n ; rank r) : set

2) $T = \emptyset$;

3) **for** $i = n$ **downto** 1 **do** {

4) **if** $r \bmod 2 = 1$ **then** $T = T \cup \{i\}$;

5) $r = \left\lfloor \frac{r}{2} \right\rfloor$;

6) }

7) **return** T ;

Gray Code

■ definition:

The *reflected binary code*, also known as *Gray code*, is a binary numeral system where two successive values differ in only one bit.

G^n will denote the reflected binary code for 2^n binary n -tuples, and it will be written as a list of 2^n vectors, as follows:

$$G^n = [G_0^n, G_1^n, \dots, G_{2^n-1}^n]$$

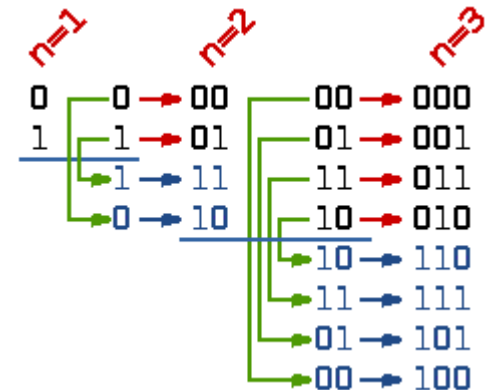
The codes G^n are defined recursively:

$$G^1 = [0, 1]$$

$$G^n = [0G_0^{n-1}, 0G_1^{n-1}, \dots, 0G_{2^{n-1}-1}^{n-1}, 1G_0^{n-1}, \dots, 1G_{2^{n-1}-1}^{n-1}]$$

■ example:

$$G^3 = [000, 001, 011, 010, 110, 111, 101, 100]$$



Gray Code

■ Example:

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

■ lemma 1

Suppose

- $0 \leq r \leq 2^n - 1$
- $B = b_n, \dots, b_0$ is a binary code of r
- $G = g_n, \dots, g_0$ is a Gray code of r

Then for every $j \in \{0, 1, \dots, n - 1\}$

$$g_j = (b_j + b_{j+1}) \bmod 2$$

■ proof

By induction on n .

■ lemma 2

Suppose

- $0 \leq r \leq 2^n - 1$
- $B = b_n, \dots, b_0$ is a binary code of r
- $G = g_n, \dots, g_0$ is a Gray code of r

Then for every $j \in \{0, 1, \dots, n-1\}$

$$b_j = (g_j + b_{j+1}) \bmod 2$$

■ proof

$$\begin{aligned} g_j &= (b_j + b_{j+1}) \bmod 2 \Rightarrow g_j \equiv (b_j + b_{j+1}) \pmod{2} \Rightarrow \\ b_j &\equiv (g_j + b_{j+1}) \pmod{2} \Rightarrow b_j = (g_j + b_{j+1}) \bmod 2 \end{aligned}$$

Gray Code

■ lemma 3

Suppose

- $0 \leq r \leq 2^n - 1$
- $B = b_n, \dots, b_0$ is a binary code of r
- $G = g_n, \dots, g_0$ is a Gray code of r

Then for every $j \in \{0, 1, \dots, n-1\}$

$$b_j = \left(\sum_{i=j}^{n-1} g_i \right) \bmod 2$$

■ proof

$$\left(\sum_{i=j}^{n-1} g_i \right) \bmod 2 = \left(\sum_{i=j}^{n-1} (b_i + b_{i+1}) \right) \bmod 2 = \left(b_j + b_n + 2 \sum_{i=j+1}^{n-1} b_i \right) \bmod 2 = (b_j + b_n) \bmod 2 = b_j$$

By lemma 2.

By the sum reordering.

By the property of modulo.

By the maximum range of r and the range of b_j .

Gray Code

■ converting to and from minimal change ordering (Gray code)

- 1) **Function** BINARYTOGRAY(binary code rank B) : gray code rank
- 2) **return** $B \text{ xor } (B \gg 1)$;

- 1) **Function** GRAYTOBINARY(gray code rank G) : binary code rank
- 2) $B = 0$;
- 3) $n = (\text{number of bits in } G) - 1$;
- 4) **for** $i=0$ **to** n **do** {
- 5) $B = B \ll 1$;
- 6) $B = B \text{ or } (1 \text{ and } ((B \gg 1) \text{ xor } (G \gg n)))$;
- 7) $G = G \ll 1$;
- 8) }
- 9) **return** B ;

Subsets – Gray Code

■ computing the subset rank over minimal change ordering

```
1) Function GRAYCODERANK( size  $n$ ; set  $T$  ) : rank
2)    $r = 0$  ;
3)    $b = 0$  ;
4)   for  $i = n - 1$  downto  $0$  do {
5)       if  $n - i \in T$  then  $b = 1 - b$  ;
6)       if  $b = 1$  then  $r = r + 2^i$  ;
7)   }
8)   return  $r$  ;
```

Subsets – Gray Code

■ computing the subset unrank over minimal change ordering

```
1) Function GRAYCODEUNRANK( size  $n$ ; rank  $r$  ) : set
2)    $T = \emptyset$  ;
3)    $c = 0$  ;
4)   for  $i = n - 1$  downto 0 do {
5)      $b = \left\lfloor \frac{r}{2^i} \right\rfloor$  ;
6)     if  $b \neq c$  then  $T = T \cup \{n - i\}$  ;
7)      $c = b$  ;
8)      $r = r - b \cdot 2^i$  ;
9)   }
10) return  $T$  ;
```

k - Element subsets

- Suppose that n is a positive integer and $S = \{1, \dots, n\}$.
- $\binom{S}{k}$ consists of all k -element subsets of S .
- A k -element subset $T \subseteq S$ can be represented in a natural way as a sorted one-dimensional array $\vec{T} = [t_1, t_2, \dots, t_k]$ where $t_1 < t_2 < \dots < t_k$.

k - Element subsets

- Example of the lexicographic ordering on k -element subsets:

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

k - Element subsets

■ computing the k -element subset successor with lexicographic ordering

```
1) Function KSUBSETLEXSUCCESOR( $k$ -element subset as array  $T$ ;  
2)                               number  $n, k$ ) :  $k$ -element subset as array ;  
3)   $U = T$  ;  
4)   $i = k$  ;  
5)  while (  $i \geq 1$  ) and (  $T[i] = n - k + i$  ) do  $i = i - 1$  ;  
6)  if (  $i = 0$  ) then  
7)      return "undefined" ;  
8)  else {  
9)      for  $j = i$  to  $k$  do  $U[j] = T[i] + 1 + j - i$  ;  
10)     return  $U$  ;  
11) }
```


k - Element subsets

■ computing the k -element subset rank with lexicographic ordering

```
1) Function KSUBSETLEXRANK( $k$ -element subset as array  $T$ ;  
2)                      number  $n, k$ ) : rank ;  
3)   $r = 0$  ;  
4)   $T[0] = 0$  ;  
5)  for  $i = 1$  to  $k$  do {  
6)      if (  $T[i-1]+1 \leq T[i]-1$  ) then {  
7)          for  $j = T[i-1]+1$  to  $T[i]-1$  do  $r = r + \binom{n-j}{k-i}$  ;  
8)      }  
9)  }  
10) return  $r$  ;
```

k - Element subsets

■ computing the k -element subset unrank with lexicographic ordering

```
1)  Function KSUBSETLEXUNRANK(rank  $r$ ;  
2)                                number  $n, k$ ) :  $k$ -element subset as array ;  
3)   $x = 1$  ;  
4)  for  $i = 1$  to  $k$  do {  
5)      while (  $\binom{n-x}{k-i} \leq r$  ) do {  
6)           $r = r - \binom{n-x}{k-i}$  ;  
7)           $x = x + 1$  ;  
8)      }  
9)       $T[i] = x$  ;  
10)    $x = x + 1$  ;  
11) }  
12) return  $T$  ;
```

Permutations

- A *permutation* is a bijection from a set to itself.
- one possible representation of a permutation

$$\pi: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$$

is by storing its values in a one-dimensional array as follows:

index	1	2	...	n
value	$\pi[1]$	$\pi[2]$...	$\pi[n]$

Permutations

■ computing the permutation rank over lexicographical ordering

```
1) Function PERMLEXRANK( size  $n$ ; permutation  $\pi$  ) : rank
2)    $r = 0$  ;
3)    $\rho = \pi$  ;
4)   for  $j = 1$  to  $n$  do {
5)        $r = r + (\rho[j] - 1) \cdot (n - j)!$  ;
6)       for  $i = j + 1$  to  $n$  do if  $\rho[i] > \rho[j]$  then  $\rho[i] = \rho[i] - 1$  ;
7)   }
8)   return  $r$  ;
```

Permutations

■ computing the permutation unrank over lexicographical ordering

```
1) Function PERMLEXUNRANK( size  $n$ ; rank  $r$  ) : permutation
2)  $\pi[ n ] = 1$  ;
3) for  $j = 1$  to  $n - 1$  do {
4)    $d = \frac{r \bmod (j+1)!}{j!}$  ;
5)    $r = r - d \cdot j!$  ;
6)    $\pi[ n - j ] = d + 1$  ;
7)   for  $i = n - j + 1$  to  $n$  do if  $\pi[ i ] > d$  then  $\pi[ i ] = \pi[ i ] + 1$  ;
8) }
9) return  $\pi$  ;
```



References

- D.L. Kreher and D.R. Stinson , *Combinatorial Algorithms: Generation, Enumeration and Search* , CRC press LTC , Boca Raton, Florida, 1998.