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

$$\mathrm{rank}\colon S \rightarrow \{0, \ldots, |S|-1\}$$

and unrank function is an inverse function to rank function.

- **definition:**

Given a ranking function rank , defined on S, the $\mathrm{successor}$ function satisfies the following rule:

$$\mathrm{successor}(s) = t \quad \Leftrightarrow \quad \mathrm{rank}(t) = \mathrm{rank}(s) + 1$$

- **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 M 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

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

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

Subsets

- **computing the subset rank over lexicographical ordering**

Function SUBSETLEXRANK(size n; set T) : rank
$r = 0$;
for $i = 1$ **to** n **do** {
    if $i \in T$ **then** $r = r + 2^{n-i}$;
}
return r ;

Function SUBSETLEXUNRANK(size n; rank r) : set
$T = \emptyset$;
for $i = n$ **downto** 1 **do** {
    if $r \bmod 2 = 1$ **then** $T = T \cup \{i\}$;
    $r = r \operatorname{div} 2$;
}
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 G_i^n , as follows:

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

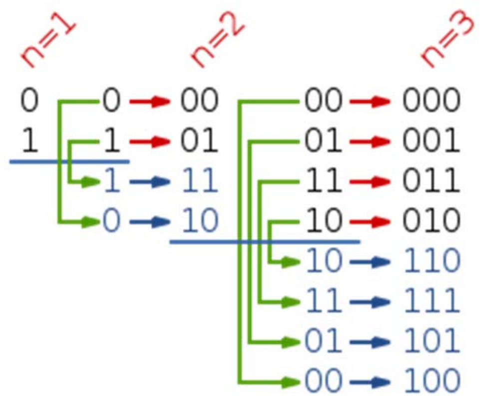
The codes G^n are defined recursively:

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

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

- **example:**

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

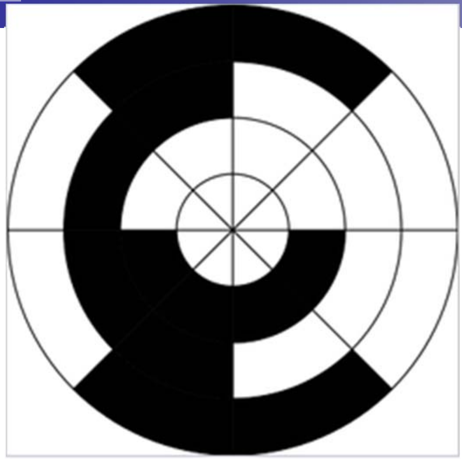


Gray Code

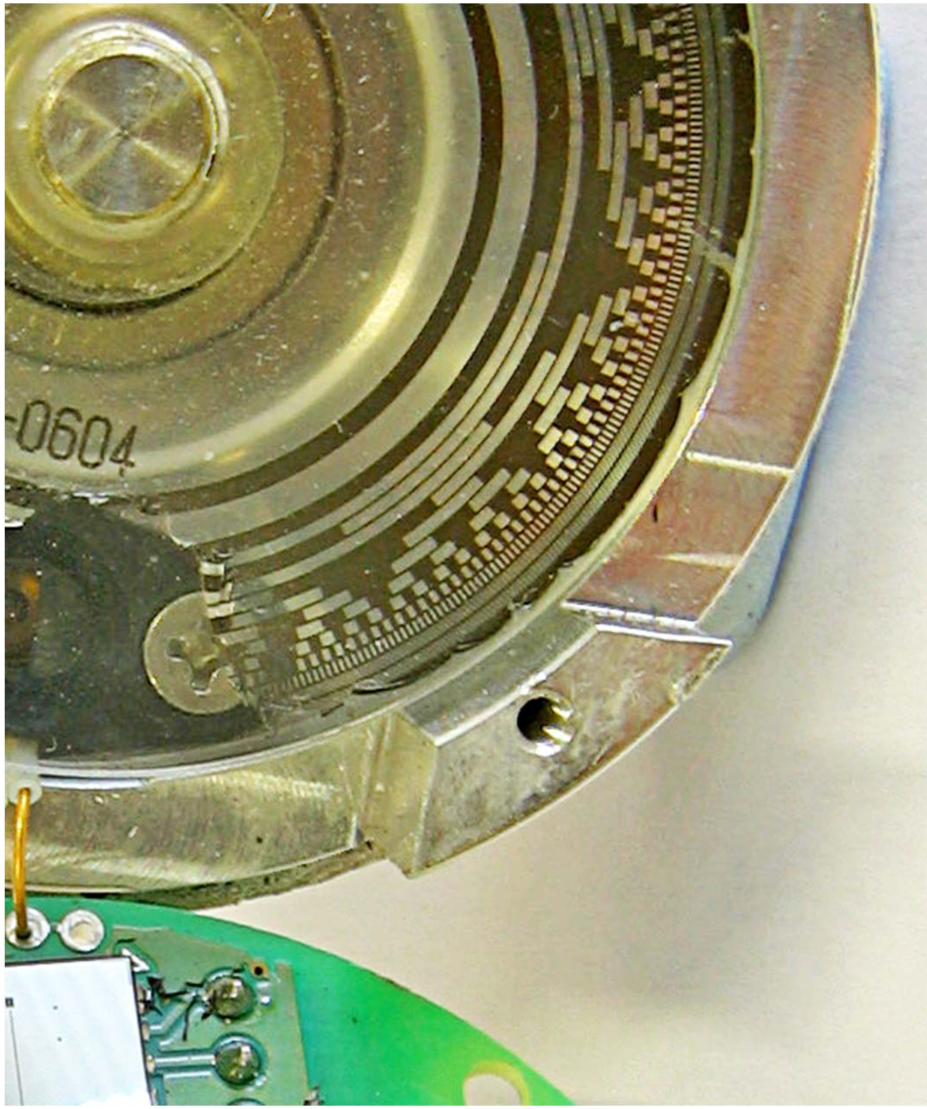
- Example:

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

Gray Code



Rotary position encoder



Regardless of the care in aligning the contacts, and accuracy of the pattern, a natural-binary code would have errors at specific disk positions, because it is impossible to make all bits change at exactly the same time as the disk rotates. (...) Rotary encoders benefit from the cyclic nature of Gray codes, because consecutive positions of the sequence differ by only one bit.

https://en.wikipedia.org/wiki/Gray_code

Gray Code

- **lemma 1**

Suppose

- $0 \le r \le 2^n - 1$
- $B = b_{n-1}, \ldots, b_0$ is a binary code of r
- $G = g_{n-1}, \ldots, g_0$ is a Gray code of r

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

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

- **proof**

By induction on n.

- **Note** We may suppose $b_n = g_n = 0$.

- Example:

G_{28}^{5}	**10010**
28	**11100**

$$\mathbf{0}_0 = \mathbf{0}_0 + \mathbf{0}_1$$
$$\mathbf{1}_1 = \mathbf{1}_1 + \mathbf{0}_2$$
$$\mathbf{0}_2 = \mathbf{1}_2 + \mathbf{1}_3$$
$$\mathbf{0}_3 = \mathbf{1}_3 + \mathbf{1}_4$$
$$\mathbf{1}_4 = \mathbf{1}_4 + \mathbf{0}_5$$

Gray Code

- **lemma 2**

Suppose

- $0 \le r \le 2^n - 1$
- $B = b_{n-1}, \dots, b_0$ is a binary code of r
- $G = g_{n-1}, \dots, g_0$ 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**

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

- Example:

G_{28}^{5} **10010**

28 **11100**

$$\mathbf{0}_0 = \mathbf{0}_0 + \mathbf{0}_1$$
$$\mathbf{0}_1 = \mathbf{1}_1 + \mathbf{1}_2$$
$$\mathbf{1}_2 = \mathbf{0}_2 + \mathbf{1}_3$$
$$\mathbf{1}_3 = \mathbf{0}_3 + \mathbf{1}_4$$
$$\mathbf{1}_4 = \mathbf{1}_4 + \mathbf{0}_5$$

Note We may suppose $b_n = g_n = 0$.

- **lemma 3**

Suppose

- $0 \le r \le 2^n - 1$
- $B = b_{n-1}, \ldots, b_0$ is a binary code of r
- $G = g_{n-1}, \ldots, g_0$ is a Gray code of r

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

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

- Example:

G^5_{28}	**10010**
28	**11100**

$$\mathbf{0}_0 = \mathbf{0}_0 + \mathbf{1}_1 + \mathbf{0}_2 + \mathbf{0}_3 + \mathbf{1}_4$$
$$\mathbf{0}_1 = \mathbf{1}_1 + \mathbf{0}_2 + \mathbf{0}_3 + \mathbf{1}_4$$
$$\mathbf{1}_2 = \mathbf{0}_2 + \mathbf{0}_3 + \mathbf{1}_4$$
$$\mathbf{1}_3 = \mathbf{0}_3 + \mathbf{1}_4$$
$$\mathbf{1}_4 = \mathbf{1}_4$$

- **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 1.

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)**

```
Function BinaryToGray( binary code rank B ) : gray code rank
return  B xor (B >> 1) ;
```

```
Function GrayToBinary(gray code rank G ) : binary code rank
B = 0;
n = (number of bits in G) – 1;
 for i=0 to n do {
   B = B << 1;
   B = B or (1 and ( (B >> 1) xor (G >> n) ) );
   G = G << 1;
 }
 return B ;
```

Subsets – Gray Code

- **computing the subset rank over minimal change ordering**
- Set: $\{1,2,\dots,n\}$, using relation $b_j = (g_j + b_{j+1}) \bmod 2$.

```
Function GRAYCODERANK( size n; subset T ) : rank
r = 0 ;
b = 0 ;
for  i = n – 1  downto  0  do  {
    if  n – i ∈ T  then b = 1 – b ;
    if  b = 1  then  r = r + 2^i ;
}
return  r ;
```

see also https://www.geeksforgeeks.org/gray-to-binary-and-binary-to-gray-conversion/

Subsets – Gray Code

- **computing the subset unrank over minimal change ordering**

```
Function GrayCodeUnrank( size n; rank r ) : set
T = ∅ ;
c = 0 ;
for i = n – 1 downto 0 do {
    b = r div 2^i ;
    if b ≠ c then T = T ∪ {n – i} ;
    c = b ;
    r = r – b·2^i ;
}
return T ;
```

k - Element subsets

- Suppose that n is a positive integer and $S = \{1, \ldots, n\}$.
- $\binom{S}{k}$ 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 $\vec{T} = [t_1, t_2, \ldots, t_k]$ where $t_1 < t_2 < \cdots < t_k$.

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**

```
Function KSubsetLexSuccesor(k-element subset as array T;
                            number n, k ) : k-element subset as array ;
U = T ;
i = k ;
while ( i ≥ 1 ) and ( T[i] = n – k + i )  do  i = i – 1 ;
if ( i = 0 ) then
    return "undefined" ;
else {
    for  j = i  to  k  do  U[j] = T[i] + 1 + j – i ;
    return  U ;
}
```

k - Element subsets

- **computing the k-element subset rank with lexicographic ordering**

Function KSUBSETLEXRANK(k-element subset as array T;
number n, k) : rank ;
$r = 0$;
$T[0] = 0$;
for $i = 1$ **to** k **do** {
  if ($T[i-1]+1 \le T[i]-1$) **then** {
    for $j = T[i-1]+1$ **to** $T[i]-1$ **do** $r = r + \binom{n-j}{k-i}$;
  }
}
return r ;

k - Element subsets

- **computing the k-element subset unrank with lexicographic ordering**

Function KSUBSETLEXUNRANK(rank r;
number n, k) : k-element subset as array ;
$x = 1$;
for $i = 1$ **to** k **do** {
    while ($\binom{n-x}{k-i} \leq r$) **do** {
        $r = r - \binom{n-x}{k-i}$;
        $x = x + 1$;
    }
    $T[i] = x$;
    $x = x + 1$;
}
return T ;

Permutations

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

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

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

index	1	2	…	n
value	$\boldsymbol{\pi[1]}$	$\boldsymbol{\pi[2]}$	**…**	$\boldsymbol{\pi[n]}$

Permutations

- **computing the permutation rank over lexicographical ordering**

$\textbf{Function}$ $\textsc{PermLexRank}$(size n; permutation π) : rank

$r = 0$;

$\rho = \pi$;

$\textbf{for}\ \ j = 1\ \ \textbf{to}\ \ n\ \ \textbf{do}\ \ \{$

$\quad r = r + (\rho[\,j\,] - 1)\cdot(n - j)!$;

$\quad \textbf{for}\ \ i = j + 1\ \ \textbf{to}\ \ n\ \ \textbf{do}\ \ \textbf{if}\ \ \rho[\,i\,] > \rho[\,j\,]\ \ \textbf{then}\ \ \rho[\,i\,] = \rho[\,i\,] - 1$;

$\}$

$\textbf{return}\ \ r$;

Permutations

- **computing the permutation unrank over lexicographical ordering**

Function PERMLEXUNRANK(size n; rank r) : permutation
$\pi[\,n\,] = 1$;
for $j = 1$ **to** $n - 1$ **do** {
$\quad d = \dfrac{r \textbf{ mod } (j+1)!}{j!}$;
$\quad r = r - d \cdot j!$;
$\quad \pi[\,n - j\,] = d + 1$;
$\quad$ **for** $i = n - j + 1$ **to** n **do** **if** $\pi[\,i\,] > d$ **then** $\pi[\,i\,] = \pi[\,i\,] + 1$;
}
return π ;

References

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