[6] Generation and enumeration of combinatorial objects (subsets, k-element subsets, permutations), Gray codes

- [**Definice**] Předpokládejme, že S je konečná množina. Ohodnocení (*ranking function*) bude bijekce ***rank***: S → {0,…,|S|-1} a ***unrank*** bude funkce k ní inverzní.

- [**Definice**] S funkcí *rank* definovanou na S bude funkce ***successor*** splňovat následující podmínku: successor(s) = t ⇔ rank(t) = rank(s) + 1

- lze použít pro generování náhodných objektů z S s garancí stejné pravděpodobnosti 1/|S| nebo pro ukládání kombinatorických objektů v počítači (namísto ukládání složité kombinatorické struktury)

Podmnožiny (subsets)

- [**Definice**] Předpokládejme, že *n* je přirozené číslo a S = {1,…,*n*}. Nechť se množina M skládá z 2^n podmnožin S. Máme-li podmnožinu T množiny S, definujme charakteristický vector T jako jednodimenzionální binární pole $\chi(T) = [x_{n-1}, x_{n-2},\ldots,x_0]$, kde

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

- příklad lexikografického uspořádání na podmnožinách S = {1,2,3}:

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

- výpočet ranku podmnožiny lexikografického uspořádání:

```
1) Function SubsetLexRank( size n; set T ) : rank
2) r = 0 ;
3) for i = 1 to n do {
4)     if i ∈ T then r = r + 2^(n-i) ;
5) }
6) return r ;
```

```
1) Function SubsetLexUnrank( size n; rank r ) : set
2) T = ∅ ;
3) for i = n downto 1 do {
4)    if r mod 2 = 1 then T = T ∪ {i} ;
5)    r = r div 2 ;
6) }
7) return T ;
```

Grayův kód

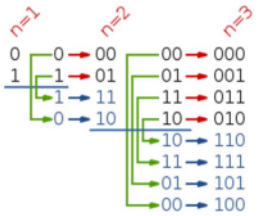
- [**Definice**] Zrcadlový binární kód (reflected bc), také Grayův kód, je binární číselný systém kde se dvě po sobě jdoucí hodnoty liší jen v jednom bitu.

- G^n je Grayův kód pro 2^n binárních n-tic, zapsaný jako seznam 2^n vektorů G_i^n následovně:
  $G^n=[G_0^n, G_1^n, \dots, G_{2^n-1}^n]$
- kódy G^n jsou definovány rekurzivně: $G^1=[0,1]$

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

- **příklad**:

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



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

- **lemma 1**

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

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

- **proof**

By induction on n.

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

- Example:

G_{28}^5 10010
28 11100

$0_0 = 0_0+0_1$
$1_1 = 1_1+0_2$
$0_2 = 1_2+1_3$
$0_3 = 1_3+1_4$
$1_4 = 1_4+0_5$

- **lemma 2**

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

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

- **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

$0_0 = 0_0 + 0_1$
$0_1 = 1_1 + 1_2$
$1_2 = 0_2 + 1_3$
$1_3 = 0_3 + 1_4$
$1_4 = 1_4 + 0_5$

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_{28}^5 10010
28 11100

$0_0 = 0_0 + 1_1 + 0_2 + 0_3 + 1_4$
$0_1 = 1_1 + 0_2 + 0_3 + 1_4$
$1_2 = 0_2 + 0_3 + 1_4$
$1_3 = 0_3 + 1_4$
$1_4 = 1_4$

- **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.

Algoritmy

converting to and from minimal change ordering (Gray code)

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

```
1) Function GrayToBinary(gray code rank G ) : binary code rank
2) B = 0;
3) n = (number of bits in G) – 1;
4)  for i=0 to n do {
5)   B = B << 1;
6)   B = B or (1 and ( (B >> 1) xor (G >> n) ) );
7)   G = G << 1;
8)  }
9)  return B ;
```

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

```
1)  Function GrayCodeRank( size n; subset T ) : rank
2)  r = 0 ;
3)  b = 0 ;
4)  for i = n − 1 downto 0 do {
5)      if n − i ∈ T then b = 1 − b ;
6)      if b = 1 then r = r + 2^i ;
7)  }
8)  return r ;
```

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

```
1)  Function GrayCodeUnrank( size n; rank r ) : set
2)  T = ∅ ;
3)  c = 0 ;
4)  for i = n − 1 downto 0 do {
5)      b = r div 2^i ;
6)      if b ≠ c then T = T ∪ {n − i} ;
7)      c = b ;
8)      r = r − b·2^i ;
9)  }
10) return T ;
```

k-prvkové podmnožiny

- Předpokládejme, že n je přirozené číslo a S = {1,…,n}.
- $\binom{S}{k}$ se skládá ze všech k-prvkových podmnožin S.
- k-prvkovou podmnožinu T ⊆ S lze vyjádřit jako setříděné jednodimenzionální pole:

$$\vec{T} = [t_1, t_2, \ldots, t_k] \text{ where } t_1 < t_2 < \cdots < t_k \ .$$

- příklad lexikografického uspořádání nad k-prvkovou podmnožinou:

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

Algoritmy

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 ≥ 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) }
```

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 ≤ T[i]−1 ) then {
7)        for j = T[i−1]+1 to T[i]−1 do r = r + C(n − j, k − i) ;
8)    }
9)  }
10) return r ;
```

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 ( C(n − x, k − i) ≤ r ) do {
6)        r = r − C(n − x, k − i) ;
7)        x = x + 1 ;
8)    }
9)    T[i] = x ;
10)   x = x + 1 ;
11) }
12) return T ;
```

Permutace

- Permutace je bijekce z množiny na sebe samu.
- Jedna z možných reprezentací permutace π: {1,…,n} → {1,…,n} je ukládání jejich hodnot do jednodimensionálního pole:

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

Algoritmy

computing the permutation rank over lexicographical ordering

```
1) Function PermLexRank( size n; permutation π ) : rank
2) r = 0 ;
3) ρ = π ;
4) for j = 1 to n do {
5)     r = r + (ρ[ j ] – 1)·(n – j)! ;
6)     for i = j +1 to n do if ρ[ i ] > ρ[ j ] then ρ[ i ] = ρ[ i ] – 1 ;
7) }
8) return r ;
```

computing the permutation unrank over lexicographical ordering

```
1) Function PermLexUnRank( size n; rank r ) : permutation
2) π[ n ] = 1 ;
3) for j = 1 to n – 1 do {
4)     d = (r mod (j+1)!) / j! ;
5)     r = r – d · j! ;
6)     π[ n – j ] = d + 1 ;
7)     for i = n – j + 1 to n do if π[ i ] > d then π[ i ] = π[ i ] + 1 ;
8) }
9) return π ;
```

Zdroj: https://cw.fel.cvut.cz/wiki/_media/courses/b4m33pal/pal06.pdf