OPPA European Social Fund

Prague & EU: We invest in your future.

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

$$\mathrm{rank}\colon S \ \to \ \{0, \dots , |S| - 1\}$$

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 $S = \{1, \ldots, n\}$.

- Define 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

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

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

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 $T = T \cup \{i\}$ ;
    $r = \left\lfloor \frac{r}{2} \right\rfloor$ ;
}
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:

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

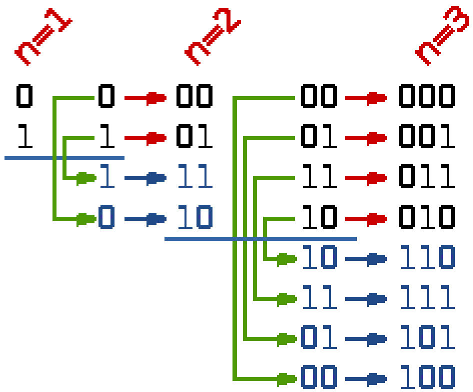
The codes G^n are defined recursively:

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

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

- **example:**

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



Gray Code

- Example:

G_r^3	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

Gray Code

- **lemma 1**

Suppose

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

Gray Code

- **lemma 2**

Suppose

- □ $0 \le r \le 2^n - 1$
- □ $B = b_n, \ldots, b_0$ is a binary code of r
- □ $G = g_n, \ldots, g_0$ 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**

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

Gray Code

lemma 3

Suppose

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

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

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

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)**

```
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**

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

Subsets – Gray Code

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

```
Function GrayCodeUnrank( size $n$; rank $r$ ) : set
$T = \emptyset$ ;
$c = 0$ ;
for $i = n - 1$ downto $0$ do {
    $b = \left\lfloor \frac{r}{2^i} \right\rfloor$ ;
    if $b \neq c$ then $T = T \cup \{n - i\}$ ;
    $c = b$ ;
    $r = r - b \cdot 2^i$ ;
}
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 ( $i \geq 1$ ) 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 ( $T[i\text{-}1]+1 \le T[i]-1$ ) then {
        for $j = T[i\text{-}1]+1$ 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

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

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**

```
Function PermLexRank( size $n$; permutation $\pi$ ) : rank
$r = 0$ ;
$\rho = \pi$ ;
for $j = 1$ to $n$ do {
    $r = r + (\rho[j] - 1)\cdot(n - j)!$ ;
    for $i = j + 1$ to $n$ do if $\rho[i] > \rho[j]$ then $\rho[i] = \rho[i] - 1$ ;
}
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 {
    $d = \frac{r \bmod (j+1)!}{j!}$ ;
    $r = r - d \cdot j!$ ;
    $\pi[ n - j ] = d + 1$ ;
    for $i = n - j + 1$ to $n$ do if $\pi[ i ] > d$ then $\pi[ i ] = \pi[ i ] + 1$ ;
}
return $\pi$ ;
```

References

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

OPPA European Social Fund

Prague & EU: We invest in your future.