Kombinatorické algoritmy

- Generování n-tic
- Generování permutací
- Generování kombinací

Použití: řešení některých úloh vyžaduje prozkoumat „prostor“ všech možných řešení a ten bývá často vyjádřen kombinatoricky (nalezení všech podmnožin, permutací, n-prvkových kombinací apod).

Generování n-tic

Použití: generování podmnožin dané množiny, generování, n-řádových čísel apod.

Postup:

Pro množiny $S_b=\{0,1\}$ nebo $S_d=\{0,1,2,3,4,5,6,7,8,9\}$ je problém jednoduchý.

1. Začneme s n-ticí $a= (000..000)_2$ nebo $a=(000..000)_{10}$
2. Postupně přičítáme 1 k binárnímu číslu a (pro S_b) nebo k dekadickému číslu (pro S_d)
3. Ukončíme pokud $a=(111…111)=2^{n-1}$ (pro S_b) nebo $a=(999…999)= 2^{10-1}$ (pro S_d)

Algoritmus pro generování obecných n-tic v lexikografickém pořadí, které pro které platí následující podmínky:

$$0 \le a_j < m_j \qquad pro \quad 1 \le j \le n$$

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \\ m_1 & m_2 & \cdots & m_n \end{bmatrix}$$

Algorithm M (*Mixed-radix generation*). This algorithm visits all n-tuples that satisfy (1), by repeatedly adding 1 to the mixed-radix number in (2) until overflow occurs. Auxiliary variables a_0 and m_0 are introduced for convenience.

M1. [Initialize.] Set $a_j \leftarrow 0$ for $0 \le j \le n$, and set $m_0 \leftarrow 2$.

M2. [Visit.] Visit the n-tuple $(a_1, \ldots, a_n)$. (The program that wants to examine all n-tuples now does its thing.)

M3. [Prepare to add one.] Set $j \leftarrow n$.

M4. [Carry if necessary.] If $a_j = m_j - 1$, set $a_j \leftarrow 0$, $j \leftarrow j - 1$, and repeat this step.

M5. [Increase, unless done.] If $j = 0$, terminate the algorithm. Otherwise set $a_j \leftarrow a_j + 1$ and go back to step M2. ∎

Pokud je hodnota *n* dostatečně malá, lze algoritmus přepsat následovně: (n=4)

For $a_1 = 0, 1, \ldots, m_1 - 1$ (in this order) do the following:
  For $a_2 = 0, 1, \ldots, m_2 - 1$ (in this order) do the following:
    For $a_3 = 0, 1, \ldots, m_3 - 1$ (in this order) do the following:
      For $a_4 = 0, 1, \ldots, m_4 - 1$ (in this order) do the following:
        Visit (a_1, a_2, a_3, a_4).

Grayův binární kód

- V některých případech nevyhovuje lexikografické pořadí n-tic je potřeba volit jiné.
- Grayův kód vypisuje všech 2^n binárních n-tic v takovém pořadí, že mezi jednotlivými n-ticemi dochází k záměně pouze jediného bitu.

Př. Grayův kód pro n=4

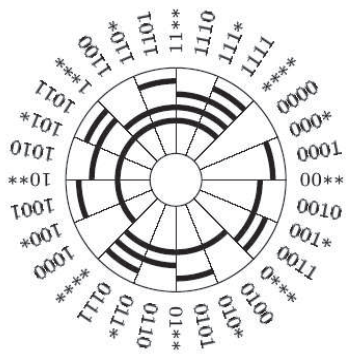
$$0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100,$$
$$1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000.$$

Použití Grayova kódu :

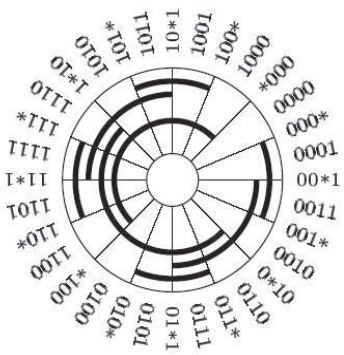
- přenos dat (umožňuje snadné zabezpečení paritou)
- Walshovy funkce, Walshova transformace (používá se při zpracování obrazů)

Snímání pozice natočení kotouče

a. Popis pozice lexikografickým binárním kódem
b. Grayovým kódem



(a) Lexicographic binary code.



(b) Gray binary code.

Fig. 10.

Algoritmus generování n-bitového Grayova kódu s paritou

Algorithm G (*Gray binary generation*). This algorithm visits all binary n-tuples $(a_{n-1}, \ldots, a_1, a_0)$ by starting with $(0, \ldots, 0, 0)$ and changing only one bit at a time, also maintaining a parity bit a_∞ such that

$$a_\infty = a_{n-1} \oplus \cdots \oplus a_1 \oplus a_0. \tag{14}$$

It successively complements bits $\rho(1)$, $\rho(2)$, $\rho(3)$, $\ldots$, $\rho(2^n - 1)$ and then stops.

G1. [Initialize.] Set $a_j \leftarrow 0$ for $0 \le j < n$; also set $a_\infty \leftarrow 0$.

G2. [Visit.] Visit the n-tuple $(a_{n-1}, \ldots, a_1, a_0)$.

G3. [Change parity.] Set $a_\infty \leftarrow 1 - a_\infty$.

G4. [Choose j.] If $a_\infty = 1$, set $j \leftarrow 0$. Otherwise let $j \ge 1$ be minimum such that $a_{j-1} = 1$. (After the kth time we have performed this step, $j = \rho(k)$.)

G5. [Complement coordinate j.] Terminate if $j = n$; otherwise set $a_j \leftarrow 1 - a_j$ and return to G2. ∎

Zrychlená varianta předchozího algoritmu – odstraňuje cyklus z kroku G4 a nahrazuje ho skupinou pointerů (f_n,…,f_0)

Algorithm L (*Loopless Gray binary generation*). This algorithm, like Algorithm G, visits all binary n-tuples $(a_{n-1}, \ldots, a_0)$ in the order of the Gray binary code. But instead of maintaining a parity bit, it uses an array of "focus pointers" $(f_n, \ldots, f_0)$, whose significance is discussed below.

L1. [Initialize.] Set $a_j \leftarrow 0$ and $f_j \leftarrow j$ for $0 \le j < n$; also set $f_n \leftarrow n$. (A loopless algorithm is allowed to have loops in its initialization step, as long as the initial setup is reasonably efficient; after all, every program needs to be loaded and launched.)

L2. [Visit.] Visit the n-tuple $(a_{n-1}, \ldots, a_1, a_0)$.

L3. [Choose j.] Set $j \leftarrow f_0$, $f_0 \leftarrow 0$. (If this is the kth time we are performing the present step, j is now equal to $\rho(k)$.) Terminate if $j = n$; otherwise set $f_j \leftarrow f_{j+1}$ and $f_{j+1} \leftarrow j + 1$.

L4. [Complement coordinate j.] Set $a_j \leftarrow 1 - a_j$ and return to L2. ∎

Grayův nebinární kód

V některých případech vyžadujeme obecný případ Grayova kódu tj. generování n-tic $(a_1, a_2, \ldots, a_n)$, kde $0 \le a_j \le m_j$, u kterých platí, že dvě následující n-tice liší pouze v jediné číslici

Př.: posloupnost trojic dekadických číslic v Grayově kódu (reflected Gray decimal)

$$000, 001, \ldots, 009, 019, 018, \ldots, 011, 010, 020, 021, \ldots, 091, 090, 190, 191, \ldots, 900,$$

Modulární Grayův kód

$$000, 001, \ldots, 009, 019, 010, \ldots, 017, 018, 028, 029, \ldots, 099, 090, 190, 191, \ldots, 900.$$

Generování Grayova nebinárního kódu

Algorithm H (*Loopless reflected mixed-radix Gray generation*). This algorithm visits all n-tuples $(a_{n-1}, \ldots, a_0)$ such that $0 \le a_j < m_j$ for $0 \le j < n$, changing only one coordinate by ± 1 at each step. It maintains an array of focus pointers $(f_n, \ldots, f_0)$ to control the actions as in Algorithm L, together with an array of directions $(d_{n-1}, \ldots, d_0)$. We assume that each radix m_j is ≥ 2.

H1. [Initialize.] Set $a_j \leftarrow 0$, $f_j \leftarrow j$, and $d_j \leftarrow 1$, for $0 \le j < n$; also set $f_n \leftarrow n$.

H2. [Visit.] Visit the n-tuple $(a_{n-1}, \ldots, a_1, a_0)$.

H3. [Choose j.] Set $j \leftarrow f_0$ and $f_0 \leftarrow 0$. (As in Algorithm L, j was the rightmost active coordinate; all elements to its right have now been activated.)

H4. [Change coordinate j.] Terminate if $j = n$; otherwise set $a_j \leftarrow a_j + d_j$.

H5. [Reflect?] If $a_j = 0$ or $a_j = m_j - 1$, set $d_j \leftarrow -d_j$, $f_j \leftarrow f_{j+1}$, and $f_{j+1} \leftarrow j + 1$. (Coordinate j has thus become passive.) Return to H2. ∎

Generování permutací

- Permutace *n* prvků je skupina všech prvků, které jsou uspořádány v jakémkoliv možném pořadí, tzn. výběr prvků závisí na pořadí.
- Pokud se prvky ve výběru nemohou opakovat, pak počet všech možných výběrů je určen vztahem

  P(n)= n!
- Pokud se hovoří o permutacích prvků, jsou tím obvykle myšleny permutace bez opakování.

  Př. Mějme skupinu tří různých prvků *a*,*b*,*c*.

  Permutace těchto prvků představují skupiny

  abc, *acb*, *bac*, *bca*, *cab*, *cba*

Generování lexikograficky uspořádaných permutací

Algorithm L (*Lexicographic permutation generation*). Given a sequence of n elements $a_1 a_2 \ldots a_n$, initially sorted so that

$$a_1 \le a_2 \le \cdots \le a_n, \tag{1}$$

this algorithm generates all permutations of $\{a_1, a_2, \ldots, a_n\}$, visiting them in lexicographic order. (For example, the permutations of $\{1, 2, 2, 3\}$ are

1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212, 3221,

ordered lexicographically.) An auxiliary element a_0 is assumed to be present for convenience; a_0 must be strictly less than the largest element a_n.

L1. [Visit.] Visit the permutation $a_1 a_2 \dots a_n$.

L2. [Find j.] Set $j \leftarrow n-1$. If $a_j \geq a_{j+1}$, decrease j by 1 repeatedly until $a_j < a_{j+1}$. Terminate the algorithm if $j = 0$. (At this point j is the largest subscript such that we have already visited all permutations beginning with $a_1 \dots a_j$. Therefore the lexicographically next permutation will increase the value of a_j.)

L3. [Increase a_j.] Set $l \leftarrow n$. If $a_j \geq a_l$, decrease l by 1 repeatedly until $a_j < a_l$. Then interchange $a_j \leftrightarrow a_l$. (Since $a_{j+1} \geq \cdots \geq a_n$, element a_l is the smallest element greater than a_j that can legitimately follow $a_1 \dots a_{j-1}$ in a permutation. Before the interchange we had $a_{j+1} \geq \cdots \geq a_{l-1} \geq a_l > a_j \geq a_{l+1} \geq \cdots \geq a_n$; after the interchange, we have $a_{j+1} \geq \cdots \geq a_{l-1} \geq a_j > a_l \geq a_{l+1} \geq \cdots \geq a_n$.)

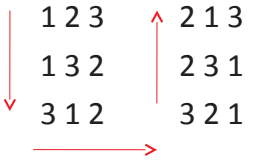
L4. [Reverse $a_{j+1} \dots a_n$.] Set $k \leftarrow j+1$ and $l \leftarrow n$. Then, if $k < l$, interchange $a_k \leftrightarrow a_l$, set $k \leftarrow k+1$, $l \leftarrow l-1$, and repeat until $k \geq l$. Return to L1. ∎

Základní myšlenka algoritmu:

1. Vezmeme posloupnost prvků {1,2, …, n-1}
2. Vložíme prvek n do každé permutace všemi možnými způsoby a uspořádáme do sloupců

Př. Vytváříme permutace nad množinou {1,2,3,4}

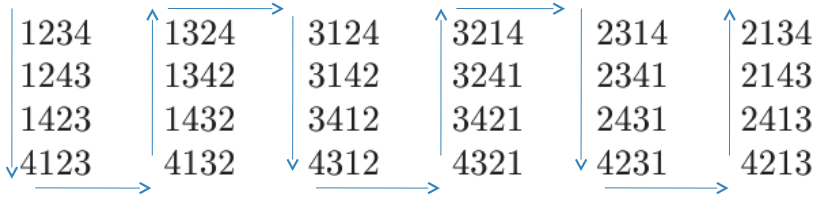
začneme pro n=2 → permutace {1 2, 2 1 } - permutace dáme do sloupců a přidáme 3 do všech možných pozic



A uspořádáme ve směru šipek

123, 132, 312, 321, 231, 213

Výsledky uložíme do sloupců a přidáme prvek 4



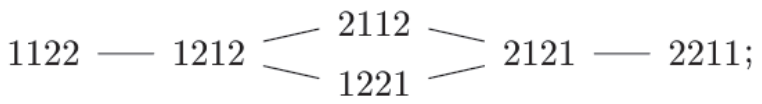
A podobně jako v předchozím případě uspořádáme a dostaneme výsledné permutace :

1234, 1243, 1423,4123,4132, 1432,1342, …, 2134, 2143,2413, 4213

Generování permutací ve kterých se mění pouze sousední elementy

Cíl: Podobně jako u Grayova kódu - vytvářet takové permutace, kde dochází ke změně pouze mezi sousedními prvky.

Tento postup není možné aplikovat pokud se v množině, nad kterou vytváříme permutace, opakují prvky.



Algoritmus generování permutací - dochází ke změnám pouze u sousedních prvků

Algorithm P (*Plain changes*). Given a sequence $a_1 a_2 \dots a_n$ of n distinct elements, this algorithm generates all of their permutations by repeatedly interchanging adjacent pairs. It uses an auxiliary array $c_1 c_2 \dots c_n$, which represents inversions as described above, running through all sequences of integers such that

$$0 \le c_j < j \qquad \text{for } 1 \le j \le n. \tag{5}$$

Another array $d_1 d_2 \dots d_n$ governs the directions by which the entries c_j change.

P1. [Initialize.] Set $c_j \leftarrow 0$ and $d_j \leftarrow 1$ for $1 \le j \le n$.

P2. [Visit.] Visit the permutation $a_1 a_2 \dots a_n$.

P3. [Prepare for change.] Set $j \leftarrow n$ and $s \leftarrow 0$. (The following steps determine the coordinate j for which c_j is about to change, preserving (5); variable s is the number of indices $k > j$ such that $c_k = k - 1$.)

P4. [Ready to change?] Set $q \leftarrow c_j + d_j$. If $q < 0$, go to P7; if $q = j$, go to P6.

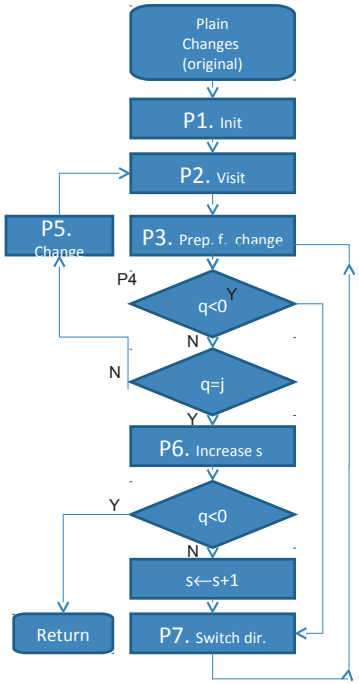
P5. [Change.] Interchange $a_{j-c_j+s} \leftrightarrow a_{j-q+s}$. Then set $c_j \leftarrow q$ and return to P2.

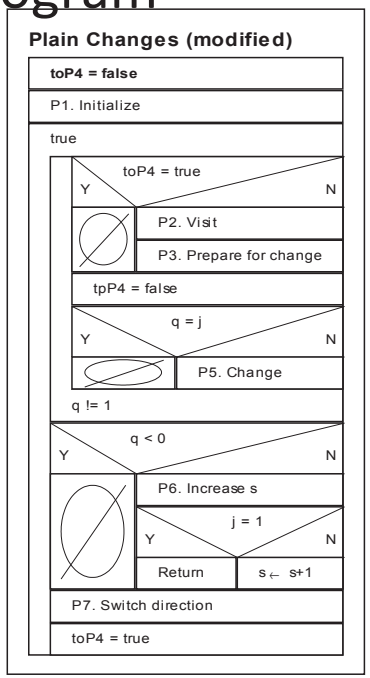
P6. [Increase s.] Terminate if $j = 1$; otherwise set $s \leftarrow s + 1$.

P7. [Switch direction.] Set $d_j \leftarrow -d_j$, $j \leftarrow j - 1$, and go back to P4. ∎

Generování kombinací

Struktogram





Kombinace *t*-té třídy z *n* prvků je skupina *t* prvků vybraných z celkového počtu *n* prvků, přičemž při výběru nezáleží na pořadí jednotlivých prvků.

Počet kombinací *t*-té třídy z *n* prvků bez opakování, tzn. žádný prvek výběru se nemůže opakovat, je

$$C_t(n) = \binom{n}{t} = \frac{n!}{t!(n-t)!}$$

- kde $\binom{n}{t}$ představuje kombinační číslo.

Uvažujme, že n=s+t .

V některé literatuře se kombinace uvádí ve tvaru (s,t)-kombinace.

Způsoby reprezentace (s,t)-kombinací:

1) Binárním řetězcem $a_{n-1}, \ldots, a_1, a_0$, pro který platí
   $a_{n-1} + \ldots a_1 + a_0 = t$. Prvky a_i nabývají hodnot
   0 … pokud prvek nebyl vybrán, nebo
   1 … pokud vybrán byl
2) Nebo vektorem $c_t, \ldots c_2\ c_1$ ve kterém jsou uloženy pozice vybraných prvků

Generování lexikografických kombinací

- Pro malé velikosti t , lze kombinace generovat následující sekvencí příkazů:

For $c_3 = 2, 3, \ldots, n-1$ (in this order) do the following:
  For $c_2 = 1, 2, \ldots, c_3 - 1$ (in this order) do the following:
    For $c_1 = 0, 1, \ldots, c_2 - 1$ (in this order) do the following:
      Visit the combination $c_3 c_2 c_1$. (15)

Table 1
THE (3, 3)-COMBINATIONS AND THEIR EQUIVALENTS

$a_5a_4a_3a_2a_1a_0$	$b_3b_2b_1$	$c_3c_2c_1$	$d_3d_2d_1$	$e_3e_2e_1$	$p_3p_2p_1p_0$	$q_3q_2q_1q_0$	path
000111	543	210	000	210	4111	3000	
001011	542	310	100	310	3211	2100	
001101	541	320	110	320	3121	2010	
001110	540	321	111	321	3112	2001	
010011	532	410	200	010	2311	1200	
010101	531	420	210	020	2221	1110	
010110	530	421	211	121	2212	1101	
011001	521	430	220	030	2131	1020	
011010	520	431	221	131	2122	1011	
011100	510	432	222	232	2113	1002	
100011	432	510	300	110	1411	0300	
100101	431	520	310	220	1321	0210	
100110	430	521	311	221	1312	0201	
101001	421	530	320	330	1231	0120	
101010	420	531	321	331	1222	0111	
101100	410	532	322	332	1213	0102	
110001	321	540	330	000	1141	0030	
110010	320	541	331	111	1132	0021	
110100	310	542	332	222	1123	0012	
111000	210	543	333	333	1114	0003	

- pro velká t lze použít následující algoritmus.

Algorithm L (*Lexicographic combinations*). This algorithm generates all t-combinations $c_t \ldots c_2 c_1$ of the n numbers $\{0, 1, \ldots, n-1\}$, given $n \geq t \geq 0$. Additional variables c_{t+1} and c_{t+2} are used as sentinels.

L1. [Initialize.] Set $c_j \leftarrow j - 1$ for $1 \leq j \leq t$; also set $c_{t+1} \leftarrow n$ and $c_{t+2} \leftarrow 0$.

L2. [Visit.] Visit the combination $c_t \ldots c_2 c_1$.

L3. [Find j.] Set $j \leftarrow 1$. Then, while $c_j + 1 = c_{j+1}$, set $c_j \leftarrow j - 1$ and $j \leftarrow j + 1$; repeat until $c_j + 1 \neq c_{j+1}$.

L4. [Done?] Terminate the algorithm if $j > t$.

L5. [Increase c_j.] Set $c_j \leftarrow c_j + 1$ and return to L2. ∎

The running time of this algorithm is not difficult to analyze. Step L3 sets $c_j \leftarrow j - 1$ just after visiting a combination for which $c_{j+1} = c_1 + j$, and the number of such combinations is the number of solutions to the inequalities

$$n > c_t > \cdots > c_{j+1} \geq j; \qquad (16)$$

Nebo rychlejší verze tohoto algoritmu – viz následující slide.

Algorithm T (*Lexicographic combinations*). This algorithm is like Algorithm L, but faster. It also assumes, for convenience, that $t < n$.

T1. [Initialize.] Set $c_j \leftarrow j - 1$ for $1 \le j \le t$; then set $c_{t+1} \leftarrow n$, $c_{t+2} \leftarrow 0$, and $j \leftarrow t$.

T2. [Visit.] (At this point j is the smallest index such that $c_{j+1} > j$.) Visit the combination $c_t \ldots c_2 c_1$. Then, if $j > 0$, set $x \leftarrow j$ and go to step T6.

T3. [Easy case?] If $c_1 + 1 < c_2$, set $c_1 \leftarrow c_1 + 1$ and return to T2. Otherwise set $j \leftarrow 2$.

T4. [Find j.] Set $c_{j-1} \leftarrow j - 2$ and $x \leftarrow c_j + 1$. If $x = c_{j+1}$, set $j \leftarrow j + 1$ and repeat this step until $x < c_j$.

T5. [Done?] Terminate the algorithm if $j > t$.

T6. [Increase c_j.] Set $c_j \leftarrow x$, $j \leftarrow j - 1$, and return to T2. ∎

Struktogram