**Algoritmi de tip successor**

**Generare submultimi**

Fie n un număr natural nenul. Să se genereze toate submulțimile mulțimii { 1, …, n}.

*Soluție*

Vom reprezenta submulțimile prin **vector caracteristic**: un vector S cu n componente 0 sau 1, având semnificația S[i]=1, dacă i aparține submulțimii si 0 în caz contrar.

Problema se reduce astfel la generarea tuturor secvențelor binare de lungime n.

**Exemplu**: generam toate submultimile multimii { 1, 2, 3 }

|  |  |
| --- | --- |
| **Multimea** | **Vectorul characteristic** |
|  | |  |  |  | | --- | --- | --- | | 0 | 0 | 0 | | 1 | 2 | 3 | |
| 1 | |  |  |  | | --- | --- | --- | | 1 | 0 | 0 | | 1 | 2 | 3 | |
| 2 | |  |  |  | | --- | --- | --- | | 0 | 1 | 0 | | 1 | 2 | 3 | |
| 1 2 | |  |  |  | | --- | --- | --- | | 1 | 1 | 0 | | 1 | 2 | 3 | |
| 3 | |  |  |  | | --- | --- | --- | | 0 | 0 | 1 | | 1 | 2 | 3 | |
| 1 3 | |  |  |  | | --- | --- | --- | | 1 | 0 | 1 | | 1 | 2 | 3 | |
| 2 3 | |  |  |  | | --- | --- | --- | | 0 | 1 | 1 | | 1 | 2 | 3 | |
| 1 2 3 | |  |  |  | | --- | --- | --- | | 1 | 1 | 1 | | 1 | 2 | 3 | |

Vom utiliza un **algoritm de tip succesor**.

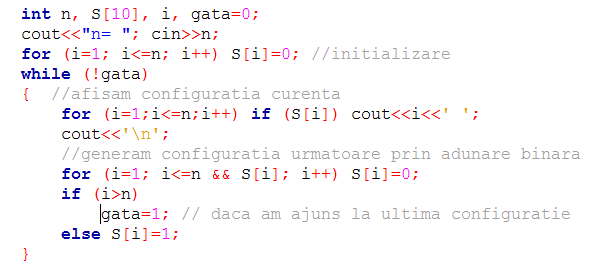
Mai exact, un astfel de algoritm functionează astfel:

**Pas 1**. Se initializează solutia cu cea mai mică configuratie posibilă (în cazul nostru, initializăm vectorul S cu 0, ceea ce corespunde multimii vide);

**Pas 2**. Cât timp este posibil (există succesor) se execută:

– se afisează configuratia curentă

– se generează configuratia următoare (în cazul nostru, vom considera că elementele vectorului reprezintă cifrele unui număr scris în baza 2, la care vom aduna 1).

****

**Generare elemente produs cartezian**

Fie n un număr natural (n>1) si L=(l1, l2, ..., ln) n numere naturale nenule. Să se determine în ordine lexicografică toate elementele produsului cartezian {1, 2, ..., l1}x{1, 2, ..., l2}x...x{1, 2, ..., ln}.

Ce înseamnă **ordine lexicografică**: Fie x=(x1, x2, ..., xn) si y=(y1, y2, ..., ym). Spunem că x precedă pe y din punct de vedere lexicografic dacă există k astfel încât xi=yi, pentru orice i<k si xk<yk sau k>n.

**Exemplu**: n=3 L=(2, 3, 2) elementele produsului cartezian {1,2}x{1,2,3}x{1,2} sunt:

**(1,1,1), (1,1,2), (1,2,1), (1,2,2), (1,3,1), (1,3,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2), (2,3,1), (2,3,2).**

produsul cartezian are l1\*l2\*...\*ln elemente.

*Soluție*

Vom reprezenta un element al produsului cartezian ca un vector E cu n elemente, unde E[i]Є{1, 2, ..., li}.

Pentru a genera toate elementele produsului cartezian în ordine lexicografică, vom aplica tot un algoritm de tip succesor:

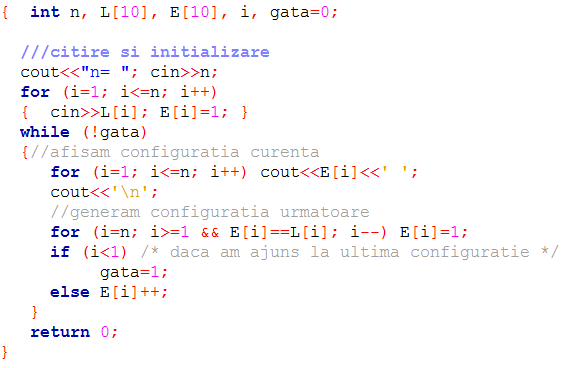
Pas 1. Initializăm vectorul E cu 1 (cel mai mic element al produsului cartezian, din punct de vedere lexicografic).

Pas 2. Cât timp este posibil (mai există succesor)

– afisăm elementul curent;

– generăm elementul următor; în acest scop căutăm prima componentă (începând din dreapta către stânga) care poate fi mărită (adică E[i]<L[i]); dacă găsim o astfel de componentă o mărim, si repunem pe 1 toate componentele

următoare; dacă nu găsim o astfel de componentă deducem că generarea s-a încheiat, acesta a fost cel mai mare element din punct de vedere lexicografic.



**Generare combinări**

Fie n şi k două numere naturale, 1≤k≤n≤20. Să se genereze în ordine lexicografică toate submultimile formate din k elemente ale multimii {1, 2, ..., n} (ordinea elementelor în submultime nu este semnificativă).

**Exemplu:** n=5 k=3 vom genera toate submultimile formate din 3 elemente ale multimii {1,2,3,4,5}

**{1,2,3}, {1,2,4}, {1,2,5}, {1,3,4}, {1,3,5}, {1,4,5}, {2,3,4}, {2,3,5}, {2,4,5}, {3,4,5}.**

*Soluție*

Vom reprezenta o submulţime ca un vector X cu k elemente ( 1<=X[i]<=n).

Pentru a nu genera de mai multe ori aceeaşi submulţime (ordinea elementelor într-o submulţime nu contează) vom conveni că plasăm elementele în submulţime în ordine crescătoare: X[i]<X[i+1], pentru orice 1≤i<k. Vom aplica tot un algoritm de tip succesor:

Cea mai mică submulţime (din punct de vedere lexicografic) este 1, 2, ..., k.

Cea mai mare submulţime este: n-k+1,..., n-1, n.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pozitie | 1 | … | i | … | k-1 | k |
| Valoare maxima | n-k+1 |  | n-k+i |  | n-1 | n |

Se poate observa că atunci când poziţia scade cu 1, valoarea maximă scade cu 1, deci diferenţa dintre Valoarea maximă şi Poziţie este constantă:

Valoare\_maximă-Poziţie=n-k=?-i.

Deducem că valoarea maximă care poate fi plasată pe poziţia i este n-k+i.

**Pas 1**. Iniţializăm vectorul X cu cea mai mica submultime dpv lexicographic (X[i]=i, pentru orice 1≤i≤k)

**Pas 2**. Cât timp mai există succesor

* afişăm submulţimea curentă;
* generăm submulţimea următoare; în acest scop căutăm prima componentă (începând din dreapta către stânga) care poate fi mărită (adică X[i]<n-k+i); dacă găsim o astfel de componentă o mărim, şi repunem pe cea mai mică valoare posibilă toate componentele următoare (X[j]=X[j-1]+1, i<j≤k); dacă nu găsim o astfel de componentă deducem că generarea s-a încheiat, acesta a fost cea mai mare submulţime din punct de vedere lexicografic.