Mulţimea (SET)

O **mulţime** ("set") este un container cu ajutorul căruia se poate reprezenta o colecţie finită de elemente distincte. Altfel spus, într-o mulţime elementele nu se pot repeta, există o singură instanţă a unui element. O altă caracteristică a mulţimii este faptul că într-o mulţime nu contează ordinea elementelor.

Mulţimea are toate operaţiile specifice **Colecţiei**, cu observaţia că operaţia adăugare într-o mulţime are specificaţie diferită faţă de operaţia de adăugare într-o colecţie (într-o mulţime elementele trebuie să fie distincte).

Tipul elementelor din mulţime, **TElement**, ca şi într-o colecţie, suportă cel puţin operaţiile de: atribuire (←) şi testarea egalităţii (=).

Spre exemplu, o mulţime de numere întregi ar putea fi: ***m***={1, 2, 3, 5, 4}.

Caracterul finit al unei mulţimi ne permite (totuşi) indexarea elementelor sale, ceea ce face ca la nivelul reprezentării interne, mulţimea **M** să poată fi asimilată cu un vector m_1, m_2,..., m_n (chiar dacă ordinea elementelor dintr-o mulţime nu este esenţială).

Pentru a putea preciza modul în care se vor efectua operaţiile pe mulţimi, vom defini structura de ***submulţime*** . Aceasta se poate realiza cu ajutorul unui vector format din valorile funcţiei caracteristice asociate submulţimii.

Dacă **M** este o mulţime, atunci submulţimea **S⊆M**   va avea asociat vectorul $\mathbf{V_S}$=(s_1, s_2,..., s_n) unde

$$s_i = \begin{cases} 1, & daca\ m_i \in S \\ 0, & daca\ m_i \notin S \end{cases}$$

De exemplu, dacă **M** ={110, 200, 318, 400}, atunci submulțimea **S**={200, 400} a lui **M** se va reprezenta sub forma vectorului **(0, 1, 0, 1)** sau **(false, true, false, true).**

Operaţiile pe submulţimi pot fi acum definite prin intremediul operaţiilor pe vectorii caracteristici asociaţi.

Fie **S1, S2⊆M** două submulţimi ale mulţimii **M**. Atunci:

a) **S1 ∪ S2 (reuniunea celor două submulţimi)** va fi caracterizată de vectorul **V** obţinut din $\mathbf{V_{S1}}$ şi $\mathbf{V_{S2}}$ efectuând operaţia logică "∨" (sau) element cu element

∨	0	1
0	0	1
1	1	1

b) **S1 ∩ S2 (intersecţia celor două submulţimi)** va fi caracterizată de vectorul **V** obţinut din $\mathbf{V_{S1}}$ şi $\mathbf{V_{S2}}$ efectuând operaţia logică "∧" (sau) element cu element

∧	0	1
0	0	0
1	0	1

Ca urmare, orice alte operaţii cu submulţimile unei mulţimi pot fi imaginate ca operaţii logice asupra vectorilor ataşaţi.

În continuare, vom prezenta specificaţia Tipul Abstract de Date **Mulţime.**

domeniu

$\mathcal{M}$={**m | m** este o mulţime cu elemente de tip **TElement**}

operaţii (**interfaţa** TAD-ului Mulţime)

creează(m)

pre: -

post: m∈$\mathcal{M}$, m este mulţimea vidă (fără elemente)

adaugă(m, e)

pre: m∈$\mathcal{M}$, e∈**TElement**

post: m'∈$\mathcal{M}$, m'=m∪{e}

{e se "reuneşte" la mulţime, adică se va adăuga numai dacă e nu mai apare în mulţime.}
{se poate returna adevărat dacă elementul a fost adăugat}

şterge(m, e)

pre: m∈$\mathcal{M}$, e∈**TElement**

post: m'∈$\mathcal{M}$, m'=m-{e}

{se şterge e din m}
{se poate returna adevărat dacă elementul a fost șters}

caută(m, e)

pre: m∈$\mathcal{M}$, e∈**TElement**

post: ***cauta***= adevărat dacă e∈m
fals în caz contrar

dim(m)

pre: m∈$\mathcal{M}$

post: ***dim***= dimensiunea mulţimii m (numărul de elemente) ∈$\mathcal{N}$

vidă(m)

pre: m∈$\mathcal{M}$

post: ***vida***= adevărat în cazul în care m e mulţimea vidă
fals în caz contrar

iterator(m, i)

pre: m∈$\mathcal{M}$

post: i∈$\mathcal{I}$, i este un iterator pe mulţimea m

distruge(m)

pre: m∈$\mathcal{M}$
post: mulţimea m a fost 'distrusă' (spaţiul de memorie alocat a fost eliberat)

Accesarea elementelor mulţimii se va face în aceeaşi manieră ca la colecţie, folosind iteratorul pe care-l oferă mulţimea.

<u>Observație.</u> În cazul în care elementele din mulțime sunt de tip **TComparabil**, elementele pot fi memorate în ordine (în raport cu o anumită relație de ordine, de ex. ≤), pentru a reduce complexitatea timp a unor operații.

Modalităţi de implementare ale mulţimilor:

- tablouri (dinamice);
- vectori booleeni (de biţi);
- liste înlănţuite;
- tabele de dispersie;
- arbori binari (de căutare echilibrați).

<u>Implementări în biblioteci predefinite</u>

- **Java**
  - interfața **Set**
    - clase care implementează interfața
      - HashSet – implementare cu o tabelă de dispersie
      - TreeSet – implementare cu un arbore echilibrat (roșu-negru)
      - ....
- **STL**
  - **unordered_set**
    - implementare - tabelă de dispersie (complexitate medie O(1))