TAD Coada (QUEUE)

Observaţii:

1. În limbajul uzual cuvântul "coadă" se referă la o înşiruire de oameni, maşini, etc., aranjaţi în ordinea sosirii şi care aşteaptă un eveniment sau serviciu.
   - Noii sosiţi se poziţionează la sfârşitul cozii.
   - Pe măsură ce se face servirea, oamenii se mută câte o poziţie înainte, până când ajung în faţă şi sunt serviţi, asigurându-se astfel respectarea principiului "primul venit, primul servit".
   - Exemple de cozi sunt multiple: coada de la benzinării, coada pentru cumpărarea unui produs, etc. Tipul de date **Coadă** permite implementarea în aplicaţii a acestor situaţii din lumea reală.
2. O *coadă* este o structură liniară de tip listă care restricţionează adăugările la un capăt şi ştergerile la celălalt capăt (lista FIFO - *First In First Out*).
3. **Accesul** într-o coadă este *prespecificat* (se poate accesa doar elementul cel mai devreme introdus în coadă), nu se permite accesul la elemente pe baza poziţiei. Dintr-o coadă se poate **şterge** elementul CEL MAI DEVREME introdus (primul).
4. Se poate considera şi o capacitate iniţială a cozii (număr maxim de elemente pe care le poate include), caz în care dacă numărul efectiv de elemente atinge capacitatea maximă, spunem că avem o *coadă plină*.
   - adăugarea în coada plină se numeşte **depăşire superioară**.
5. O coadă fără elemente o vom numi *coadă vidă* şi o notăm Φ.
   - ştergerea din coada vidă se numeşte **depăşire inferioară**.
6. O coadă în general nu se iterează.
7. Cozile sunt frecvent utilizate în programare - crearea unei cozi de aşteptare a task-urilor într-un sistem de operare.
   - dacă task-urile nu au asociată o prioritate, ele sunt procesate în ordinea în care intră în sistem $\rightarrow$ **Coadă**.
   - dacă task-urile au asociate o prioritate şi trebuie procesate în ordinea priorităţii lor $\rightarrow$ **Coadă cu priorităţi**.

Tipul Abstract de Date COADA:

domeniu: $\mathcal{C} = \{c \mid c \text{ este o coadă cu elemente de tip } TElement\}$

operaţii:

- creeaza(c)
  {creează o coadă vidă}
  $pre:$ $true$
  $post:$ $c \in \mathcal{C}, c = \Phi$(coada vidă)

- adauga(c, e)
  {se adaugă un element la sfârşitul cozii}
  $pre:$ $c \in \mathcal{C}, e \in TElement, c$ nu e plină
  $post:$ $c' \in \mathcal{C}, c' = c \oplus e,$ e va fi cel mai recent element introdus în coadă

  @ aruncă excepţie dacă coada e plină

- sterge(c, e)
  {se şterge primul element introdus în coadă}
  $pre:$ $c \in \mathcal{C}, c \neq \Phi$
  $post:$ $e \in TElement,$ e este elementul cel mai devreme introdus în coadă, $c' \in \mathcal{C}, c' = c \ominus e$

  @ aruncă excepţie dacă coada e vidă

- element(c, e)
  {se accesează primul element introdus în coadă}
  $pre:$ $c \in \mathcal{C}, c \neq \Phi$
  $post:$ $c' = c, e \in TElement,$ e este elementul cel mai devreme introdus în coadă

  @ aruncă excepţie dacă coada e vidă

- vida (c)
  $pre:$ $c \in \mathcal{C}$
  $post:$ $vida = \begin{cases} adev, & \text{dacă } c = \Phi \\ fals, & \text{dacă } c \neq \Phi \end{cases}$

- plina (c)
  $pre:$ $c \in \mathcal{C}$
  $post:$ $plina = \begin{cases} adev, & \text{dacă } c \text{ e plină} \\ fals, & \text{contrar} \end{cases}$

- distruge(c)
  {destructor}
  $pre:$ $c \in \mathcal{C}$
  $post:$ c a fost 'distrusa' (spaţiul de memorie alocat a fost eliberat)

Observaţii

- Coada nu este potrivită pentru aplicaţiile care necesită traversarea ei (nu avem acces direct la elementele din interiorul cozii).
- Afişarea conţinutului cozii poate fi realizată folosind o coadă auxiliară (scoatem valorile din coadă punându-le într-o coadă auxiliară, după care se reintroduc în coada iniţială). Complexitatea timp a sublgoritmului **tiparire** (descris mai jos) este $\theta(n)$, n fiind numărul de elemente din coadă.

```
Subalgoritm tiparire(c)
  {pre: c este o Coadă}
  {post: se tipăresc elementele din Coadă}
  creeaza(cAux) {se creează o coadă auxiliară vidă}
  {se şterg elementele din c şi se adaugă în cAux}
  CatTimp ¬ vida(c) executa
    sterge(c, e)
    @ tipăreşte e
    adauga(cAux, e)
  SfCatTimp
  {se şterg elementele din cAux şi se reface c}
  CatTimp ¬ vida(cAux) executa
    sterge(cAux, e)
    adauga(c, e)
  SfCatTimp
SfSubalgoritm
```

Implementări ale cozilor folosind

- tablouri - vectori (dinamici) - reprezentare circulară (Figura 1, Figura 2).
- liste înlănţuite.
- adăugarea ûn coadă, ştergerea din caoadă să se facă eficient ($\theta(1)$)

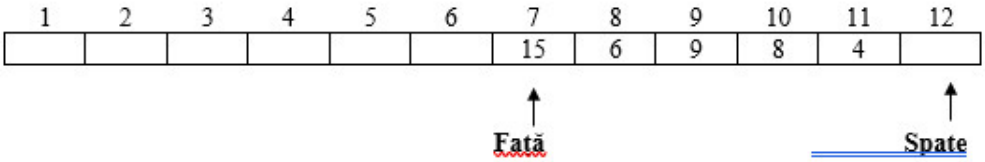
Generalizare a cozilor

- **Coada completă** (*Double ended queue* - **DEQUEUE**) - adăugări, ştergeri se pot face la ambele capete ale cozii.

Implementări ale TAD Coada în biblioteci

- **Java**
  - interfaţa **Queue**
    - sub-interfaţa **Deque**
  - clase care implementează **Queue**
    - ArrayDeque, ConcurrentLinkedQueue, PriorityQueue
- **STL**
  - **queue**, **deque**

Figura 1: Reprezentare circulară pe tablou



n = 12 (capacitatea tabloului)
Față = 7 - indicele unde e memorat primul element din coadă
Spate = 12 – primul indice liber din spatele cozii

Pp. că în vector memorăm elementele de la poziția 1

- Se adaugă în **Spate**, se șterge din **Față**
- Elementele cozii se află între indicii **Față, Față+1,...Spate-1**
- Inițializarea cozii: **Față=Spate=1** (dacă vectorul se memorează de la poziția 0, atunci **Față=Spate=0**)
- Depășire inferioară (coada vidă): **Față=Spate**
- Depășire superioară (coada plină): a) **Față= 1** și **Spate= n** sau b) **Față=Spate+1**

a) **Față= 1** și **Spate= 12**

1	2	3	4	5	6	7	8	9	10	11	12
2	8	9	3	5	7	15	6	9	8	4	

a) **Față= 7** și **Spate= 6**

1	2	3	4	5	6	7	8	9	10	11	12
2	8	9	3	5		15	6	9	8	4	5

Figura 2: Operaţii pe coada reprezentată circular pe tablou (unidimensional)

```
Reprezentare
        Coada
                cp: Intreg {capacitatea maximă de memorare}
                Față, Spate: Intreg {indicii Față, Spate}
                e: TElement[1..cp] {elementele memorate}

subalgoritm creează(c, cp) este { θ(1) }
        c.cp ← cp
        c.Față ← 1
        c.Spate ← 1
sfCreează

subalgoritm adaugă(c, e) este { θ(1) }
{nu se verifică depășirea superioară}
        c.e[c.Spate] ← e
        dacă c.Spate = c.cp atunci
                c.Spate ← 1
          altfel
                c.Spate ← c.Spate + 1
        sfdacă
sfAdaugă

subalgoritm șterge(c, e) este { θ(1) }
{nu se verifică depășirea inferioară}
        e ← c.e[c.Față]
        dacă c.Față = c.cp atunci
                c.Față ← 1
          altfel
                c.Față ← c.Față + 1
        sfdacă
sfAdaugă
```