

# TAD Coadă (QUEUE)

## Observații:

1. În limbajul uzual cuvântul “coadă” se referă la o înșiruire de oameni, mașini, etc., aranjați în ordinea sosirii și care așteaptă un eveniment sau serviciu.
  - Noii sosiți se poziționează la sfârșitul cozii.
  - Pe măsură ce se face servirea, oamenii se mută câte o poziție înainte, până când ajung în față și sunt serviți, 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 aplicații a acestor situații din lumea reală.
2. O *coadă* este o structură liniară de tip listă care restricționează adăugările la un capăt și ștergerile 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 poziției. Dintr-o coadă se poate **șterge** elementul CEL MAI DEVREME introdus (primul).
4. Se poate considera și o capacitate inițială 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 numește **depășire superioară**.
5. O coadă fără elemente o vom numi *coadă vidă* și o notăm  $\Phi$ .
  - ștergerea din coada vidă se numește **depășire inferioară**.
6. O coadă în general nu se iterează.
7. Cozile sunt frecvent utilizate în programare - crearea unei cozi de așteptare 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 → **Coadă**.
  - dacă task-urile au asociate o prioritate și trebuie procesate în ordinea priorității lor → **Coadă cu priorități**.

## Tipul Abstract de Date COADA:

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

## operații:

- $creeaza(c)$   
 {creează o coadă vidă}  
 $pre : true$   
 $post : c \in \mathcal{C}, c = \Phi(\text{coada vidă})$
- $adauga(c, e)$   
 {se adaugă un element la sfârșitul cozii}  
 $pre : c \in \mathcal{C}, e \in TElement, c \text{ nu e plină}$   
 $post : c' \in \mathcal{C}, c' = c \oplus e, e \text{ va fi cel mai recent element introdus în coadă}$   
 @ aruncă excepție dacă coada e plină
- $sterge(c, e)$   
 {se șterge primul element introdus în coadă}  
 $pre : c \in \mathcal{C}, c \neq \Phi$   
 $post : e \in TElement, e \text{ este elementul cel mai devreme introdus în coadă}, c' \in \mathcal{C}, c' = c \ominus e$   
 @ aruncă excepție dacă coada e vidă
- $element(c, e)$   
 {se accesează primul element introdus în coadă}  
 $pre : c \in \mathcal{C}, c \neq \Phi$   
 $post : c' = c, e \in TElement, e \text{ este elementul cel mai devreme introdus în coadă}$   
 @ aruncă excepție 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}  
 $pre : c \in \mathcal{C}$   
 $post : c \text{ a fost 'distrusa' (spațiul de memorie alocat a fost eliberat)}$

## Observații

- Coadă nu este potrivită pentru aplicațiile care necesită traversarea ei (nu avem acces direct la elementele din interiorul cozii).
- Afișarea conținutului cozii poate fi realizată folosind o coadă auxiliară (scoatem valorile din coadă punându-le într-o coadă auxiliară, după care se reintroduc în coada inițială). Complexitatea timp a subalgoritmului **tiparire** (descriș 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 șterg elementele din c și se adaugă în cAux}
CatTimp  $\neg$  vida(c) executa
    sterge(c, e)
    @ tipărește e
    adauga(cAux, e)
SfCatTimp
{se șterg elementele din cAux și se reface c}
CatTimp  $\neg$  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 înlănțuite.

## Generalizare a cozilor

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

## Implementări ale TAD Coadă în biblioteci

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

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

↑  
Față
↑  
Spate

$n = 12$  (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 1: Reprezentare circulară pe tablou

## Reprezentare

### Cooda

$cp$ : Intreg {capacitatea maximă de memorare}

Fată, Spate: Intreg {indicii Fată, Spate}

$e$ : TElement[1..cp] {elementele memorate}

subalgoritm *creează*( $c$ ,  $cp$ ) este  $\{ \theta(1) \}$

$c.cp \leftarrow cp$

$c.Fată \leftarrow 1$

$c.Spate \leftarrow 1$

**sfCreează**

subalgoritm *adaugă*( $c$ ,  $e$ ) este  $\{ \theta(1) \}$

{nu se verifică depășirea superioară}

$c.e[c.Spate] \leftarrow e$

**dacă**  $c.Spate = c.cp$  **atunci**

$c.Spate \leftarrow 1$

**altfel**

$c.Spate \leftarrow c.Spate + 1$

**sfdacă**

**sfAdaugă**

subalgoritm *șterge*( $c$ ,  $e$ ) este  $\{ \theta(1) \}$

{nu se verifică depășirea inferioară}

$e \leftarrow c.e[c.Fată]$

**dacă**  $c.Fată = c.cp$  **atunci**

$c.Fată \leftarrow 1$

**altfel**

$c.Fată \leftarrow c.Fată + 1$

**sfdacă**

**sfAdaugă**

Figura 2: Operații pe coada reprezentată cirucular pe tablou