

TAD CoadăCuPrioritati (PRIORITY QUEUE)

Observații:

1. O *coadă cu priorități* este un container în care fiecare element are asociat o anumită prioritate.
2. Principiul de gestionare a unei cozi cu priorități este "HPF - Highest Priority First".
3. **Accesul** într-o coadă cu priorități este *prespecificat* (se poate accesa doar elementul cel mai prioritar - prioritate maximă/minimă). Pentru generalitate, se poate considera o relație de ordine \mathcal{R} între priorități ($\mathcal{R} : TPrioritate \times TPrioritate$, \mathcal{R} relație de ordine). În cazul general, vom considera că se accesează elementul 'cel mai prioritar' în raport cu relația de ordine \mathcal{R} (de ex. dacă se dorește accesul la elementul de prioritate maximă, atunci $\mathcal{R} = \geq$). Dintr-o coadă cu priorități se **șterge** elementul 'cel mai prioritar' (în raport cu relația de ordine \mathcal{R}). **Inserarea** unui element se va face astfel încât să se păstreze relația de ordine între priorități.
4. Se poate considera și o capacitate inițială a cozii cu priorități (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 cu priorități plină*.
5. Cozile cu priorități sunt mai simple decât dicționarele (se restricționează accesul).
6. O coadă cu priorități în general nu se iterează.
7. Aplicații
 - simularea unor sisteme complexe (aeroport) - sunt mai multe evenimente care au loc la un anumit moment de timp t . Simularea se va face printr-o coadă cu priorități (t este prioritatea) - accesul este la elementul având prioritate *minimă*.
 - probleme de concurență a proceselor.

Tipul Abstract de Date COADA CU PRIORITATI:

domeniu:

$\mathcal{CP} = \{cp \mid cp \text{ este o coadă cu priorități cu elemente } (e, p) \text{ de tip } TElement \times TPrioritate\}$

operații:

- $creeaza(cp, \mathcal{R})$
 {creează o coadă cu priorități vidă}
 $pre : \mathcal{R} : TPrioritate \times TPrioritate, \mathcal{R} \text{ relație de ordine}$
 $post : cp \in \mathcal{CP}, cp = \Phi$
- $adauga(cp, e, p)$
 {se adaugă un element în coada cu priorități}
 $pre : cp \in \mathcal{CP}, e \in TElement, p \in TPrioritate$
 $post : cp' \in \mathcal{CP}, cp' = cp \oplus (e, p)$
- $sterge(cp, e, p)$
 {se șterge elementul 'cel mai prioritar'}
 $pre : cp \in \mathcal{CP}, cp \neq \Phi$
 $post : e \in TElement, p \in TPrioritate, e \text{ este 'cel mai prioritar' element din } cp$
 $p \text{ e prioritatea sa}, cp' \in \mathcal{CP}, cp' = cp \ominus (e, p)$

 @ aruncă excepție dacă coada cu priorități e vidă
- $element(cp, e, p)$
 {se accesează elementul 'cel mai prioritar'}
 $pre : cp \in \mathcal{CP}, cp \neq \Phi$
 $post : cp' = cp, e \in TElement, p \in TPrioritate$
 $e \text{ este 'cel mai prioritar' element din } cp, p \text{ e prioritatea sa}$

 @ aruncă excepție dacă coada cu priorități e vidă
- $vida(cp)$
 $pre : cp \in \mathcal{CP}$
 $post : vida = \begin{cases} adev, & \text{dacă } cp = \Phi \\ fals, & \text{dacă } cp \neq \Phi \end{cases}$
- $plina(cp)$
 $pre : cp \in \mathcal{CP}$
 $post : plina = \begin{cases} adev, & \text{dacă } cp \text{ e plină} \\ fals, & \text{contrar} \end{cases}$
- $distruge(cp)$
 {destructor}
 $pre : cp \in \mathcal{CP}$
 $post : cp \text{ a fost 'distrus'} \text{ (spațiul de memorie alocat a fost eliberat)}$

Observații

- CP nu este potrivită pentru aplicațiile care necesită traversarea ei (nu avem acces direct la elementele din interiorul cozii).
- Afișarea conținutului CP poate fi realizată (ca și la o **coadă** simplă) folosind o CP auxiliară. Complexitatea timp a subalgoritmului **tiparire** (descriș mai jos) este $\theta(n)$, n fiind numărul de elemente din CP.

Subalgoritm $tiparire(cp)$

{pre: cp este o CP}

{post: se tipăresc elementele din cp }

```

creeaza(cpAux) {se creează o CP auxiliară vidă}
{se șterg elementele din cp și se adaugă în cpAux}
CatTimp  $\neg$  vida(cp) executa
    sterge(cp, e)
    @ tipărește e
    adauga(cpAux, e)
SfCatTimp
{se șterg elementele din cpAux și se reface cp}
CatTimp  $\neg$  vida(cpAux) executa
    sterge(cpAux, e)
    adauga(cp, e)
SfCatTimp
SfSubalgoritm

```

Implementări ale cozilor cu priorități folosind

- o listă sortată (ordonată) în raport cu prioritatea elementelor
 - reprezentare secvențială pe tablou (vector dinamic) .
 - reprezentare înlănțuită (simplu/dublu, reprezentare înlănțuiri folosind alocare dinamică/alocare statică).
- **ansamblu (heap)** (va fi discutat în Cursul 14).

Observație:

- *Ansamblul* este cea mai potrivită SD pentru implementarea unei CP, deoarece oferă o complexitate $O(\log_2 n)$ pentru operațiile specifice (adăugare, ștergere) și $\theta(1)$ pentru operația de accesare.

Implementări ale TAD Coada cu Priorități în biblioteci

- **Java**
 - clasa PriorityQueue
- **STL**
 - priority_queue

Aplicație

Acordarea bursei pentru studenți. Să se simuleze procesul de acordare a bursei pentru studenți. Sunt n studenți, se acordă m burse. Bursele se acordă în ordine descrescătoare a mediilor.