

# PROGRAMARE ȘI STRUCTURI DE DATE

## CURS 9

Lect. dr. Oneț-Marian Zsuzsanna

Facultatea de Matematică și Informatică UBB  
în colaborare cu NTT Data

# În cursul 7 și 8

- Complexități
  - Notăția  $O$ -mare
  - Notăția  $\Omega$ -mare
  - Notăția  $\Theta$ -mare
  - Caz favorabil, defavorabil, mediu
- Vector Dinamic

# Cuprins

1 Vector Dinamic

2 Ansamblu

# Vectorul dinamic

- *Vectorul dinamic* este un vector a cărui dimensiune poate fi modificată.
- Ideea de bază este simplă:
  - Definim un vector de  $n$  elemente.
  - Dacă la un moment dat vectorul este plin (toate pozițiile sunt ocupate), dar mai trebuie să adăugăm elemente, vom defini un alt vector, mai mare, și vom copia elementele din acest vector în cel nou. După ce am copiat elementele, vectorul nou devine vectorul nostru.

# Vectorul dinamic II

- Pentru a reprezenta un vector dinamic avem nevoie de 3 informații:
  - numărul maxim de elemente care încap în vector (capacitatea)
  - numărul de elemente care sunt stocate în vector
  - vectorul efectiv

## VectorDinamic:

cap: întreg

len: întreg

elemente: TElem[]

# Vector Dinamic - operația ștergePozitie

- Ce ar trebui să facă operația *ștergePozitie*?

# Vector Dinamic - operația ștergePozitie

- Ce ar trebui să facă operația *ștergePozitie*?

**funcție** *ștergePozitie* (poz: întreg) **este:**  
**dacă** *poz* < 0 **sau** *poz* >= *this.len* **atunci**  
    @aruncă o excepție  
**sf\_dacă**  
elem: TElem  
elem = *this.elemente*[*poz*]  
*i*: întreg // *mutăm elementele la stânga*  
**pentru** *i* = *poz*, *this.len*-1, 1 **execută**  
    *this.elemente*[*i*] = *this.elemente*[*i*+1]  
**sf\_pentru**  
*this.len* = *this.len* - 1  
**returnează** elem  
**sf\_funcție**

# Vector Dinamic - operația ștergePoziție

- Cât este complexitatea funcției *ștergePoziție*?
- Cazul favorabil (vrem să ștergem elementul de pe ultima poziție), complexitatea este  $\Theta(1)$
- Cazul defavorabil (vrem să ștergem elementul de pe prima poziție), complexitatea este  $\Theta(n)$
- Cazul mediu, complexitatea este  $\Theta(n)$
- Complexitatea totală:  $O(n)$
- **Obs:** Este posibil să modificăm dimensiunea vectorului și dacă este prea *gol* după o adăugare, putem crea un vector mai mic (de ex. jumătatea vectorului inițial) în care putem copia elementele.



# Vector dinamic ca structură de date

- Vectorul Dinamic poate fi folosit ca structură de date pentru a implementa containere (adică elementele containerelor vor fi stocate într-un Vector Dinamic).
- Când implementăm un container folosind un Vector Dinamic, trebuie să combinăm modul în care lucrăm cu Vectorul Dinamic cu caracteristicile containerului.
- În continuare vom discuta despre reprezentare și operația de adăugare pentru fiecare container, folosind vectorul dinamic pentru implementare.

# TAD Colecție pe VectorDinamic

- Dacă vrem să implementăm o Colecție folosind Vectorul Dinamic, cum ar trebui să reprezentăm Colecția?

# TAD Colecție pe VectorDinamic

- Dacă vrem să implementăm o Colecție folosind Vectorul Dinamic, cum ar trebui să reprezentăm Colecția?
- Există 2 posibilități de implementare în funcție de cât de multe elemente duplicate estimăm că vor fi în Colecție:
  - Punem elementele pur și simplu în Vector Dinamic, dacă avem elemente care se repetă, le punem de câte ori apar, de exemplu, dacă avem o Colecție cu elementele *ciocolată*, *salată*, *lapte*, *ciocolată*, *pâine*, *lapte*, vom avea un Vector Dinamic cu cele 6 elemente.
  - Dacă credem că vor fi multe duplicate, merită să reținem elemente unice, și pentru fiecare element frecvența lui. Pentru exemplul de mai sus, am avea un Vector Dinamic cu 4 elemente, *ciocolată* cu frecvența 2, *salată* cu frecvența 1, *lapte* cu frecvența 2, *pâine* cu frecvența 1.
- În continuare discutăm despre prima variantă (varianta a 2-a am discuta-o la Seminarul 4)

# TAD Colecție pe Vector Dinamic

- Cum reprezentăm o Colecție pe Vector Dinamic? Ce câmpuri ne trebuie?

# TAD Colecție pe Vector Dinamic

- Cum reprezentăm o Colecție pe Vector Dinamic? Ce câmpuri ne trebuie?

Colecție:

len: întreg

cap: întreg

elemente: TElem[]

# TAD Colecție pe Vector Dinamic

- Cum reprezentăm o Colecție pe Vector Dinamic? Ce câmpuri ne trebuie?

## Colecție:

len: întreg

cap: întreg

elemente: TElem[]

- Dacă ne uităm mai bine, câmpurile sunt la fel ca la Vector Dinamic, am schimbat doar numele. E important că dacă implementez TAD Colecție, structura să se numească Colecție.

# TAD Colecție pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?

# TAD Colecție pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?
- Într-o Colecție nu sunt poziții și pot exista duplicate, când adaug un element, pot să-l pun oriunde. Unde e cel mai ușor să adaug un element într-un Vector Dinamic?



# TAD Colecție pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?
- Într-o Colecție nu sunt poziții și pot exista duplicate, când adaug un element, pot să-l pun oriunde. Unde e cel mai ușor să adaug un element într-un Vector Dinamic?
  - La sfârșitul vectorului
- Fiind vector dinamic, trebuie să verific dacă mai am loc liber, dacă nu, trebuie să aloc (creez) un vector mai mare și să copiez elementele.

# TAD Colecție pe Vector Dinamic - adăugare

**subalgoritm** adaugă (e: TElem) **este:**

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

vectNou = @un vector cu this.cp\*2 elemente

i: întreg

**pentru** i = 0, this.len, 1 **execută** *//copiem elementele existente*

vectNou[i] = this.elemente[i]

**sf\_pentru**

this.elemente = vectNou *//înlocuim vectorul*

this.cp = this.cp \* 2

**sf\_dacă**

this.elemente[this.len] = e *//nu contează ordinea elementelor...*

this.len = this.len + 1 *//... punem la sfârșit*

**sf\_subalgoritm**

# TAD Colecție pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?

# TAD Colecție pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?
- Complexitatea este la fel ca pentru *adaugă* Sfârșit la VectorDinamic:
  - Caz favorabil:  $\Theta(1)$
  - Caz defavorabil:  $\Theta(n)$
  - Caz mediu:  $\Theta(1)$  - amortizat
  - Complexitatea totală:  $O(n)$

# TAD Mulțime pe Vector Dinamic

- Dacă vrem să implementăm o Mulțime folosind Vectorul Dinamic, cum ar trebui să reprezentăm Mulțimea?

# TAD Mulțime pe Vector Dinamic

- Dacă vrem să implementăm o Mulțime folosind Vectorul Dinamic, cum ar trebui să reprezentăm Mulțimea?

Mulțime:

len: întreg

cap: întreg

elemente: TElem[]

# TAD Mulțime pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?

# TAD Mulțime pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?
- Nu există poziții într-o Mulțime, deci putem pune elementul unde vrem noi, și din nou, cel mai simplu este să-l punem la sfârșit.
- Fiind Mulțime, elementele trebuie să fie unice, deci înainte de a adăuga un element nou, trebuie să verificăm restul elementelor.
- Fiind Vector Dinamic, dacă elementul trebuie adăugat, trebuie să verificăm dacă mai avem loc liber, și dacă nu, alocăm un vector mai mare



# TAD Mulțime pe Vector Dinamic - adăugare

**subalgoritm** adaugă (e: TElem) **este:**

*//prima dată verificăm dacă mai există elementul*

i: întreg

gasit: boolean

i = 0

gasit = fals

**cât timp** i < this.len **ȘI** gasit == fals **execută**

**dacă** this.elemente[i] == e **atunci**

gasit = adevărat

**sf\_dacă**

i = i + 1

**sf\_cât timp**

**dacă** gasit == fals **atunci**

*//trebuie adăugat*

*//continuăm pe pagina următoare*

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

vectNou = @un vector cu this.cp\*2 elemente

i: întreg

**pentru** i = 0, this.len, 1 **execută**

vectNou[i] = this.elemente[i]

**sf\_pentru**

this.elemente = vectNou *//înlocuim vectorul*

this.cp = this.cp \* 2

**sf\_dacă**

this.elemente[this.len] = e

this.len = this.len + 1

**sf\_dacă**

**sf\_subalgorithm**

# TAD Mulțime pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?

# TAD Mulțime pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?
- Complexitatea depinde și de căutare
  - Caz favorabil:  $\Theta(1)$  (găsesc elementul pe prima poziție)
  - Caz defavorabil:  $\Theta(n)$  (elementul nu se găsește și trebuie adăgat)
  - Caz mediu:  $\Theta(n)$  (din cauza căutării nu mai am complexitate amortizată)
  - Complexitatea totală:  $O(n)$

# TAD Listă pe Vector Dinamic

- Dacă vrem să implementăm o Listă folosind Vectorul Dinamic, cum ar trebui să reprezentăm Lista?

# TAD Listă pe Vector Dinamic

- Dacă vrem să implementăm o Listă folosind Vectorul Dinamic, cum ar trebui să reprezentăm Lista?

Lista:

len: întreg

cap: întreg

elemente: TElem[]

# TAD Lista pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?

# TAD Lista pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?
- La Listă există poziții, și există operația de a adăuga un element pe o poziție. Deci nu putem pune elementul pur și simplu la final (exceptând cazul dacă poziția ne spune să-l punem acolo).
- În primul rând trebuie să verificăm să avem o poziție validă.
- Și pentru a elibera poziția respectivă, trebuie să mutăm elementele de după poziția respectivă cu o poziție mai la dreapta.
- Fiind Vector Dinamic, dacă elementul trebuie adăugat, trebuie să verificăm dacă mai avem loc liber, și dacă nu, alocăm un vector mai mare



# TAD Listă pe Vector Dinamic - adăugarePoziție

**subalgoritm** adaugăPoziție (poz: întreg, e: TElem) **este:**

*//prima dată verificăm poziția*

**dacă** poz < 0 **sau** poz > this.len **atunci**

    @aruncă excepție, poziție invalidă

**sf\_dacă**

i: întreg

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

    vectNou = @un vector cu this.cp\*2 elemente

**pentru** i = 0, this.len, 1 **execută**

        vectNou[i] = this.elemente[i]

**sf\_pentru**

    this.elemente = vectNou *//înlocuim vectorul*

    this.cp = this.cp \* 2

**sf\_dacă**

*//continuăm pe pagina următoare*

*//mutăm elementele la dreapta. Începem de la capăt*

**pentru**  $i = \text{this.len}-1$ ,  $\text{poz}-1$ ,  $-1$  **execută**

$\text{this.elemente}[i+1] = \text{this.elemente}[i]$

**sf\_pentru**

*//punem elementul pe pozitia poz*

$\text{this.elemente}[\text{poz}] = e$

$\text{this.len} = \text{this.len} + 1$

**sf\_subalgoritm**

# TAD Listă pe Vector Dinamic - *adăugarePoziție*

- Cât este complexitatea pentru *adaugăPoziție*?

# TAD Listă pe Vector Dinamic - adăugarePoziție

- Cât este complexitatea pentru *adaugăPoziție*?
- Complexitatea este:
  - Caz favorabil:  $\Theta(1)$  (adăugăm la sfârșit și nu trebuie realocat)
  - Caz defavorabil:  $\Theta(n)$  (adăugăm la început sau trebuie realocat)
  - Caz mediu:  $\Theta(n)$
  - Complexitatea totală:  $O(n)$

# TAD Dicționar pe Vector Dinamic

- Dacă vrem să implementăm un Dicționar folosind Vectorul Dinamic, cum ar trebui să reprezentăm Dicționarul?

# TAD Dicționar pe Vector Dinamic

- Dacă vrem să implementăm un Dicționar folosind Vectorul Dinamic, cum ar trebui să reprezentăm Dicționarul?
- Ne trebuie ceva similar cu ce am avut până acum, dar la un Dicționar avem perechi cheie-valoare și trebuie să ținem cont de acest lucru. Avem două variante:
  - Definim separat un tip *Pereche*, care este alcătuit dintr-o cheie și o valoare. În acest caz vom avea un singur vector, cu elemente de tip *Pereche*.
  - Nu definim tip *Pereche* și vom avea 2 vectori, unul cu chei și unul cu valori. Valoarea de pe poziția  $i$  aparține cheii de pe poziția  $i$ .
- Indiferent ce alegem, reprezentarea fiind un Vector Dinamic, ne trebuie lungime și capacitate.

# TAD Dicționar pe Vector Dinamic - cu Pereche

## Pereche:

cheie: TCheie // *TCheie e tipul cheilor*

valoarea: TValoare // *TValoare e tipul valorilor*

## Dicționar:

len: întreg

cap: întreg

elemente: Pereche[]

# TAD Dicționar pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?



# TAD Dicționar pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?
- Nu există poziții în Dicționar, putem pune pereche unde vrem noi.
- Trebuie să verificăm dacă mai există cheia în dicționar, dacă da, vom înlocui valoarea asociată.
- Fiind Vector Dinamic, dacă elementul trebuie adăugat, trebuie să verificăm dacă mai avem loc liber, și dacă nu, alocăm un vector mai mare

# TAD Dicționar pe Vector Dinamic - adăugare

**subalgoritm** adaugă (c: TCheie, v: TValoare) **este:**

*//prima dată verificăm dacă mai există cheia*

i: întreg

gasit: boolean

i = 0

gasit = fals

**cât timp** i < this.len **ȘI** gasit == fals **execută**

**dacă** this.elemente[i].cheie == c **atunci**

gasit = adevărat

*//înlocuim valoarea*

this.elemente[i].valoare = v

**sf\_dacă**

i = i + 1

**sf\_cât timp**

**dacă** gasit == fals **atunci**

*//trebuie să adăugăm. Continuăm pe pagina următoare*

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

vectNou = @un vector cu this.cp\*2 elemente

i: întreg

**pentru** i = 0, this.len, 1 **execută**

vectNou[i] = this.elemente[i]

**sf\_pentru**

this.elemente = vectNou *//înlocuim vectorul*

this.cp = this.cp \* 2

**sf\_dacă**

p: Pereche

p.cheie = c

p.valoare = v

this.elemente[this.len] = p

this.len = this.len + 1

**sf\_dacă**

**sf\_subalgoritm**

# TAD Dictionar pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?

# TAD Dicționar pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?
- Complexitatea este:
  - Caz favorabil:  $\Theta(1)$  (cheia este pe prima poziție în vector)
  - Caz defavorabil:  $\Theta(n)$  (cheia nu se găsește sau trebuie realocat)
  - Caz mediu:  $\Theta(n)$
  - Complexitatea totală:  $O(n)$

# TAD Dicționar pe Vector Dinamic - fără Pereche

- Reprezentarea Dicționarului:

Dicționar:

len: întreg

cap: întreg

chei: TCheie[]

valori: TValoare[]

# TAD Dictionar pe Vector Dinamic - adăugare

**subalgoritm** adaugă (c: TCheie, v: TValoare) **este:**

*//prima dată verificăm dacă mai există cheia*

i: întreg

gasit: boolean

i = 0

gasit = fals

**cât timp** i < this.len **ȘI** gasit == fals **execută**

**dacă** this.chei[i] == c **atunci**

gasit = adevărat

*//înlocuim valoarea*

this.valori[i] = v

**sf\_dacă**

i = i + 1

**sf\_cât timp**

**dacă** gasit == fals **atunci**

*//trebuie să adăugăm. Continuăm pe pagina următoare*

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

cNou = @un vector cu this.cp\*2 elemente

vNou = @un vector cu this.cp\*2 elemente

i: întreg

**pentru** i = 0, this.len, 1 **execută**

cNou[i] = this.chei[i]

vNou[i] = this.valori[i]

**sf\_pentru**

this.chei = cNou *//înlocuim vectorul de chei*

this.valori = vNou *//înlocuim vectorul si de valori*

this.cp = this.cp \* 2

**sf\_dacă**

this.chei[this.len] = c

this.valori[this.len] = v

this.len = this.len + 1

**sf\_dacă**



# TAD Dictionar pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?

# TAD Dicționar pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?
- Complexitatea este:
  - Caz favorabil:  $\Theta(1)$  (cheia este pe prima poziție în vector)
  - Caz defavorabil:  $\Theta(n)$  (cheia nu se găsește sau trebuie realocat)
  - Caz mediu:  $\Theta(n)$
  - Complexitatea totală:  $O(n)$

# TAD MultiDicționar pe Vector Dinamic

- Dacă vrem să implementăm un MultiDicționar folosind Vectorul Dinamic, cum ar trebui să reprezentăm MultiDicționarul?

# TAD MultiDicționar pe Vector Dinamic

- Dacă vrem să implementăm un MultiDicționar folosind Vectorul Dinamic, cum ar trebui să reprezentăm MultiDicționarul?
- În general avem două variante pentru a reprezenta un MultiDicționar
  - Reținem perechi cheie - valoare (ca la un Dicționar) dar permitem mai multe perechi cu aceeași cheie, pentru situații când o cheie are mai multe valori
  - Reținem perechi cheie - *listă de valori*, în acest caz cheile sunt unice
    - *Lista de valori* - nu trebuie să fie neapărat TAD Listă, poate fi un Vector Dinamic sau altă structură de date.

# TAD MultiDicționar pe Vector Dinamic

- Varianta cu perechi cheie-valoare (prima variantă de pe pagina anterioară) seamănă foarte mult cu ce am făcut la Dicționar, dar nu verificăm dacă o cheie mai există.
  - Avem și aici varianta în care avem o structură separată pentru o Pereche (și un singur vector de Perechi), sau varianta în care avem doi vectori (unul pentru chei și unul pentru valori).
- Pentru varianta cu *lista de valori*, putem defini separat o structură Pereche (alcătuită dintr-o cheie și un vector dinamic de valori) sau putem lucra fără Pereche, cu un vector de chei și un vector dinamic de vectori dinamici (dar e mai complicată decât varianta anterioară).

# TAD MultiDicționar pe Vector Dinamic

- Dacă avem o structură Pereche formată dintr-o cheie și un Vector Dinamic de valori, cum reprezentăm MultiDicționarul?

# TAD MultiDicționar pe Vector Dinamic

- Dacă avem o structură Pereche formată dintr-o cheie și un Vector Dinamic de valori, cum reprezentăm MultiDicționarul?

## Pereche:

cheie: TCheie

valori: TValoare[]

len: întreg

cap: întreg

# TAD MultiDicționar pe Vector Dinamic

- Dacă avem o structură Pereche formată dintr-o cheie și un Vector Dinamic de valori, cum reprezentăm MultiDicționarul?

## Pereche:

cheie: TCheie

valori: TValoare[]

len: întreg

cap: întreg

## MultiDicționar:

elemente: Pereche[]

len: întreg

cap: întreg



# TAD MultiDicționar pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația adăugare?

# TAD MultiDicționar pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația adăugare?

**subalgoritm** adaugă (c: TCheie, v: TValoare) **este:**

*//prima dată verificăm dacă mai există cheia*

i: întreg

gasit: boolean

i = 0

gasit = fals

**cât timp** i < this.len **ȘI** gasit == fals **execută**

**dacă** this.elemente[i].cheie == c **atunci**

gasit = adevărat

*//trebuie să adăugăm valoarea*

*//valorile sunt reținute într-un VD, poate trebuie realocat*

*//continuăm pe pagina următoare*

```
dacă this.elemente[i].len == this.elemente[i].cap atunci  
    vNou = @un vector cu this.elemente[i].cap * 2 elemente  
    index: întreg  
    pentru index = 0, this.elemente[i].len, 1 executa  
        vNou[index] = this.elemente[i].valori[index]  
    sf_pentru  
    this.elemente[i].valori = vNou  
    this.elemente[i].cap = this.elemente[i].cap * 2  
sf_dacă  
    //punem valoarea in vectorul de valori  
    this.elemente[i].valori[this.elemente[i].len] = v  
    this.elemente[i].len = this.elemente[i].len + 1  
sf_dacă  
    i = i + 1  
sf_cât timp  
    //continuăm pe pagina următoare
```

**dacă** gasit == fals **atunci**

*//trebuie să adăugăm o pereche nouă. Verificăm dacă avem spațiu*

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

vectNou = @un vector cu this.cp\*2 elemente

index: întreg

**pentru** index = 0, this.len, 1 **execută**

vectNou[index] = this.elemente[index]

**sf\_pentru**

this.elemente = vectNou *//înlocuim vectorul*

this.cp = this.cp \* 2

**sf\_dacă**

p: Pereche

p.cheie = c

p.cap = 4 *//putem pune câte poziții vrem să avem inițial*

p.valori = @un vector de p.cap elemente

p.valori[0] = v *//punem valoarea in vector*

p.len = 1 *//am adăugat valoarea v*

*//continuăm pe pagina următoare*

```
this.elemente[this.len] = p  
this.len = this.len + 1  
sf_dacă
```

```
this.elemente[this.len] = p  
this.len = this.len + 1  
sf_dacă
```

- Cât este complexitatea algoritmului?

```
this.elemente[this.len] = p  
this.len = this.len + 1
```

**sf\_dacă**

- Cât este complexitatea algoritmului?
  - Avem caz favorabil:  $\Theta(1)$  - adaugăm o valoare la prima cheie și nu trebuie realocat.
  - Caz dezaforabil:  $\Theta(n + m)$ , unde  $n$  este numărul total de chei și  $m$  este numărul total de valori
    - Adunarea la complexități (de exemplu:  $n + m$ ) se folosește când avem 2 valori și nu știm care este maximul. Înseamnă "maximul dintre cele 2 valori".
  - Caz mediu:  $\Theta(n)$
  - Complexitate totală:  $O(n + m)$

# TAD Mulțime ordonată pe Vector Dinamic

- Dacă vrem să implementăm o Mulțime ordonată folosind Vectorul Dinamic, cum ar trebui să reprezentăm Mulțimea ordonată?



# TAD Mulțime ordonată pe Vector Dinamic

- Dacă vrem să implementăm o Mulțime ordonată folosind Vectorul Dinamic, cum ar trebui să reprezentăm Mulțimea ordonată?

## MulTimeOrdonată:

rel: Relație

len: întreg

cap: întreg

elemente: TElem[]

- Vom trata Relația ca o funcție cu 2 parametrii:
  - $rel(e_1, e_2) = 0$ , dacă  $e_1 == e_2$
  - $rel(e_1, e_2) = -1$ , dacă  $e_1 < e_2$
  - $rel(e_1, e_2) = 1$ , dacă  $e_1 > e_2$

# TAD Mulțime ordonată pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?

# TAD Mulțime ordonată pe Vector Dinamic - adăugare

- Ce ar trebui să facă operația de adăugare?
- Fiind Mulțime, elementele trebuie să fie unice, deci înainte de a adăuga un element nou, trebuie să verificăm restul elementelor. Verificarea e suficientă să o facem până la poziția unde elementul ar trebui să fie (elementele fiind ordonate).
- Fiind o Mulțime ordonată, dacă adăugăm elementul, trebuie să-l punem în așa fel, încât elementele să rămână ordonate (deci nu putem pune doar la capăt).
- Fiind Vector Dinamic, dacă elementul trebuie adăugat, trebuie să verificăm dacă mai avem loc liber, și dacă nu, alocăm un vector mai mare

# TAD Mulțime ordonată pe Vector Dinamic - adăugare

**subalgoritm** adaugă (e: TElem) **este:**

*//prima dată verificăm dacă mai există elementul*

i, index: întreg

i = 0

**cât timp** i < this.len **ȘI** this.rel(e, this.elemente[i]) == 1 **execută**

i = i + 1

**sf\_cât timp**

*//Acum ori i = this.len (am ieșit din vector) ori e = this.elemente[i]*

*//ori e < this.elemente[i]*

**dacă** (i < this.len **ȘI** this.elemente[i] ≠ e) **OR** i == this.len **atunci**

*//trebuie adăugat*

*//continuăm pe pagina următoare*

**dacă** this.len = this.cap **atunci**

*//nu mai avem loc liber.*

vectNou = @un vector cu this.cp\*2 elemente

**pentru** index = 0, this.len, 1 **execută**

vectNou[index] = this.elemente[index]

**sf\_pentru**

this.elemente = vectNou *//înlocuim vectorul*

this.cp = this.cp \* 2

**sf\_dacă**

*//mutăm elementele de la poziția i la dreapta*

**pentru** index = this.len-1, i-1, -1 **execută**

this.elemente[index+1] = this.elemente[index]

**sf\_pentru**

this.elemente[i] = e

this.len = this.len + 1

**sf\_dacă**

**sf\_subalgorithm**

# TAD Mulțime ordonată pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?

# TAD Mulțime ordonată pe Vector Dinamic - adăugare

- Cât este complexitatea pentru *adaugă*?
- Complexitatea depinde și de căutare
  - Caz favorabil:  $\Theta(1)$  (găsesc elementul pe prima poziție)
  - Caz defavorabil:  $\Theta(n)$  (elementul nu se găsește și trebuie adăgat)
  - Caz mediu:  $\Theta(n)$
  - Complexitatea totală:  $O(n)$

# Coadă cu Priorități pe Vector Dinamic

- La Mulțime ordonată am văzut că este necesar să reținem elementele ordonate, și acest lucru este adevărat și pentru restul containerelor ordonate care au iterator. Dar cum putem reține elementele pentru o Coadă cu Priorități?



# Coadă cu Priorități pe Vector Dinamic

- La Mulțime ordonată am văzut că este necesar să reținem elementele ordonate, și acest lucru este adevărat și pentru restul containerelor ordonate care au iterator. Dar cum putem reține elementele pentru o Coadă cu Priorități?
  - Putem reține elementele în ordinea în care au fost adăugate
    - La *adăugare* pur și simplu punem elementul unde este cel mai ușor de pus
    - La *ștergere* trebuie să căutăm elementul cu prioritatea maximă
    - La *element* trebuie să căutăm elementul cu prioritatea maximă.

# Coadă cu Priorități pe Vector Dinamic

- La Mulțime ordonată am văzut că este necesar să reținem elementele ordonate, și acest lucru este adevărat și pentru restul containerelor ordonate care au iterator. Dar cum putem reține elementele pentru o Coadă cu Priorități?
  - Putem reține elementele în ordinea în care au fost adăugate
    - La *adăugare* pur și simplu punem elementul unde este cel mai ușor de pus
    - La *ștergere* trebuie să căutăm elementul cu prioritatea maximă
    - La *element* trebuie să căutăm elementul cu prioritatea maximă.
  - Putem reține elementele ordonate după prioritate
    - La *adăugare* trebuie să punem elementul astfel încât structura să rămână ordonată.
    - La *ștergere* știm că elementul cu prioritatea maximă este la un capăt al cozii, nu trebuie să căutăm.
    - La *element* știm că elementul cu prioritatea maximă este la un capăt al cozii, nu trebuie să căutăm.

# CP - Reprezentare pe Vector Dinamic

- Reținem elementele în ordinea în care au fost adăugate. În care capăt al Vectorului adăugăm un element nou (număr mai mare înseamnă element mai prioritar)?

(Ion, 3)	(Elena, 2)	(Raul, 4)	(Oana, 5)	(Paul, 1)			
----------	------------	-----------	-----------	-----------	--	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

(Ion, 3)	(Elena, 2)	(Raul, 4)	(Oana, 5)	(Paul, 1)			
----------	------------	-----------	-----------	-----------	--	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

- Săgeata verde arată capătul unde inserăm.

# CP - Reprezentare pe Vector Dinamic

- Dacă vectorul nu este ordonat, la ștergere (și la operația element) trebuie să parcurgem vectorul să găsim prioritatea maximă, indiferent de capătul ales pentru adăugare.
- La adăugare e mai simplu să punem un element la finalul vectorului dinamic, de aceea vom alege varianta a 2-a.

(Ion, 3)	(Elena, 2)	(Raul, 4)	(Oana, 5)	(Paul, 1)			
----------	------------	-----------	-----------	-----------	--	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

# CP - Reprezentare pe Vector Dinamic

- Unde punem elementul (Radu, 4)?

# CP - Reprezentare pe Vector Dinamic

- Unde punem elementul (Radu, 4)?

(Ion, 3)	(Elena, 2)	(Raul, 4)	(Oana, 5)	(Paul, 1)	(Radu, 4)		
----------	------------	-----------	-----------	-----------	-----------	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 6

- Complexitate:

# CP - Reprezentare pe Vector Dinamic

- Unde punem elementul (Radu, 4)?

(Ion, 3)	(Elena, 2)	(Raul, 4)	(Oana, 5)	(Paul, 1)	(Radu, 4)		
----------	------------	-----------	-----------	-----------	-----------	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 6

- Complexitate:  $\Theta(1)$

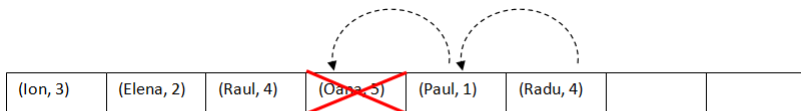
# CP - Reprezentare pe Vector Dinamic

- Care element va fi şters?



# CP - Reprezentare pe Vector Dinamic

- Care element va fi șters?
- Trebuie să ștergem elementul cu prioritatea maximă, adică pe (Oana, 5).



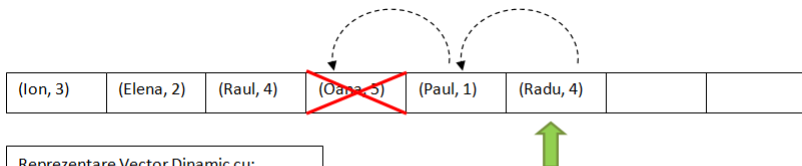
Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 6

- Complexitate:

# CP - Reprezentare pe Vector Dinamic

- Care element va fi șters?
- Trebuie să ștergem elementul cu prioritatea maximă, adică pe (Oana, 5).



- Complexitate:  $\Theta(n)$

# CP - Reprezentare pe Vector Dinamic

- Complexitatea este  $\Theta(n)$  pentru că pentru a găsi elementul cu prioritate maximă trebuie să parcurgem tot vectorul. Nu există caz favorabil, chiar dacă șterg ultimul element și nu am de mutat niciun element, pentru a afla că ultimul element are prioritatea maximă trebuie să parcurgem tot vectorul.
- După ștergere vom avea:

(Ion, 3)	(Elena, 2)	(Raul, 4)	(Paul, 1)	(Radu, 4)			
----------	------------	-----------	-----------	-----------	--	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

# CP - Reprezentare pe Vector Dinamic Ordonat

- Dacă folosesc un Vector Dinamic Ordonat pot să am 2 variante (ordonat crescător după prioritate, sau ordonat descrescător după prioritate):

(Paul, 1)	(Elena, 2)	(Ion, 3)	(Raul, 4)	(Oana, 5)			
-----------	------------	----------	-----------	-----------	--	--	--

Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5



(Oana, 5)	(Raul, 4)	(Ion, 3)	(Elena, 2)	(Paul, 1)			
-----------	-----------	----------	------------	-----------	--	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

- Care variantă e mai bună?

# CP - Reprezentare pe Vector Dinamic Ordonat

- La adăugare trebuie să inserăm elementul astfel încât vectorul să rămână ordonat, indiferent de varianta aleasă.
- La ștergere e mai simplu să ștergem de la finalul vectorului, deci vom alege prima variantă.

(Paul, 1)	(Elena, 2)	(Ion, 3)	(Raul, 4)	(Oana, 5)			
-----------	------------	----------	-----------	-----------	--	--	--

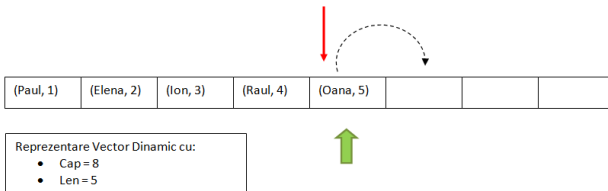


Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5


# CP - Reprezentare pe Vector Dinamic Ordonat

- Unde punem elementul (Radu, 4)?



# CP - Reprezentare pe Vector Dinamic Ordonat


- Unde punem elementul (Radu, 4)?



(Paul, 1)	(Elena, 2)	(Ion, 3)	(Raul, 4)	(Oana, 5)			
-----------	------------	----------	-----------	-----------	--	--	--

Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5



(Paul, 1)	(Elena, 2)	(Ion, 3)	(Raul, 4)	(Radu, 4)	(Oana, 5)		
-----------	------------	----------	-----------	-----------	-----------	--	--

Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 6

# CP - Reprezentare pe Vector Dinamic Ordonat

- Elementul (Radu, 4) poate fi pus și în fața elementului (Raul, 4) pentru că au aceeași prioritate.
- Complexitate:



# CP - Reprezentare pe Vector Dinamic Ordonat

- Elementul (Radu, 4) poate fi pus și în fața elementului (Raul, 4) pentru că au aceeași prioritate.
- Complexitate:  $O(n)$  - aici avem caz favorabil, poate elementul de adăugat are complexitate mai mare decât ultimul element, și atunci pur și simplu îl punem la final.

# CP - Reprezentare pe Vector Dinamic Ordonat

- Care element va fi șters?
- Ștergem din nou elementul cu prioritatea maximă, dar de data asta știm că este la finalul vectorului.

# CP - Reprezentare pe Vector Dinamic Ordonat

- Care element va fi șters?
- Ștergem din nou elementul cu prioritatea maximă, dar de data asta știm că este la finalul vectorului.

(Paul, 1)	(Elena, 2)	(Ion, 3)	(Raul, 4)	(Radu, 4)	<del>(Dana, 5)</del>		
-----------	------------	----------	-----------	-----------	----------------------	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

- Complexitate:

# CP - Reprezentare pe Vector Dinamic Ordonat

- Care element va fi șters?
- Ștergem din nou elementul cu prioritatea maximă, dar de data asta știm că este la finalul vectorului.

(Paul, 1)	(Elena, 2)	(Ion, 3)	(Raul, 4)	(Radu, 4)	<del>(Dana, 5)</del>		
-----------	------------	----------	-----------	-----------	----------------------	--	--



Reprezentare Vector Dinamic cu:

- Cap = 8
- Len = 5

- Complexitate:  $\Theta(1)$

# CP - Vector Dinamic - Rezumat

- Să vedem complexitatea operațiilor pentru cele 2 variante de reprezentare pe vector dinamic (ordonat sau neordonat)

Operație	VD Neordonat	VD Ordonat
adaugă	$\Theta(1)$	$O(n)$
șterge	$\Theta(n)$	$\Theta(1)$
element	$\Theta(n)$	$\Theta(1)$

- Pe baza complexităților putem concluda că este mai bine să reținem elementele într-o structură ordonată.

# CP - Rezumat

- Am văzut că putem să implementăm operația de adăugare având complexitate  $O(n)$  și ștergerea în  $\Theta(1)$ .
- Acest lucru înseamnă că dacă adăugăm  $n$  elemente într-o coadă cu priorități după care ștergem  $n$  elemente, complexitatea totală a operațiilor va fi:  $O(n^2)$  ( $n * O(n) + n * \Theta(1)$ ).
- Putem avea complexitate mai bună, dacă folosim pentru reprezentarea cozii o altă structură de date, numită *ansamblu* (*heap* în engleză).

# Ansamblu

- Ansamblul este o structură de date care este un fel de hibrid între
  - Vector Dinamic
  - Arbore Binar
- Ansamblul este folosit ca reprezentare doar pentru Coadă cu priorități
- Algoritmul de sortare *Heap-sort* este bazat pe un ansamblu.

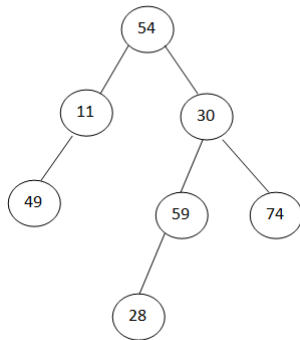
# Arbore binar

- Vectorul este o structură *liniară*: pentru fiecare element există un element anterior și unul următor (exceptând primul și ultimul element).
- Un arbore binar este o structură *neliniară*, alcătuită din *noduri*, fiecare nod are un *părinte* și 0, 1 sau 2 *fii* sau *descendenți*.
- Nodul care nu are părinte se numește *rădăcina* arborelui, iar nodurile care nu au descendenți se numesc *frunze*.



# Arbore binar

- Un exemplu de arbore binar.
- Rădăcina: 54
- Frunze: 49, 28, 74
- Fii lui 54: 11, 30
- Fii lui 30: 59, 74
- Părintele lui 49: 11
- etc.



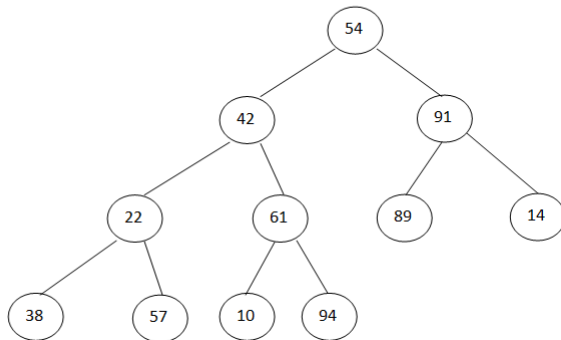
- Vom vorbi mai mult despre arbori în cursurile 13-14, deocamdată e important doar să înțelegeți ce este un arbore binar.

# Ansamblu

- Un *ansamblu* este de fapt un vector dinamic (elementele ansamblului sunt stocate într-un vector dinamic) care este privit/interpretat ca un arbore binar:
  - Rădăcina arborelui este primul element din vector.
  - Teoretic, pentru orice element de pe poziția  $i$ , descendenții elementului se găsesc în vector pe pozițiile  $2 * i$ ,  $2 * i + 1$
  - În practică, din moment ce prima poziție într-un vector dinamic e 0, descendenții pentru poziția  $i$  se găsesc pe pozițiile  $2 * i + 1$  și  $2 * i + 2$
  - Părintele unui element de pe poziția  $i$  este pe poziția  $(i - 1)/2$ .
    - Rădăcina: poziția 0.
    - Descendenții rădăcinei: pozițiile  $2*0+1 = 1$ ,  $2*0+2 = 2$
    - Descendenții elementului pe poziția 1:  $2*1+1 = 3$ ,  $2*1+2 = 4$
    - Descendenții elementului pe poziția 2:  $2*2+1 = 5$ ,  $2*2+2 = 6$
    - etc.

# Ansamblu Exemplu

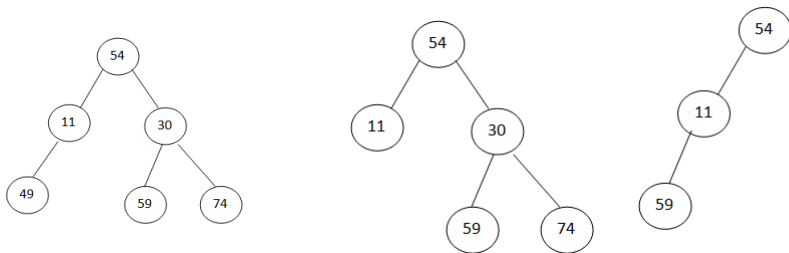
54	42	91	22	61	89	14	38	57	10	94
0	1	2	3	4	5	6	7	8	9	10



# Ansamblu

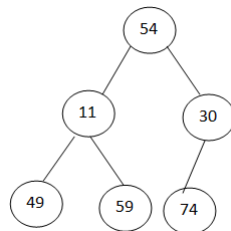
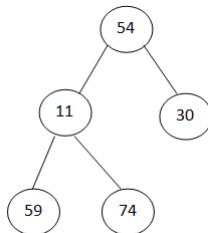
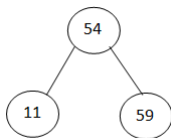
- Pentru a avea un ansamblu *valid* (nu orice arbore binar este un ansamblu) trebuie să respectăm *structura de ansamblu* și *proprietatea de ansamblu*.
- Structura de ansamblu: nu există “găuri” în arbore: fiecare nod are exact 2 descendenți, exceptând ultimele 2 niveluri. Pe ultimul nivel, descendenții sunt completați din stânga spre dreapta.

# Exemple de arbori binari care nu au structura de ansamblu



- Evident, sunt mult mai multe exemple de arbori binari care nu au structură de ansamblu.

# Exemple de arbori binari care au structura de ansamblu

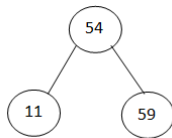


# Proprietatea de ansamblu

- Proprietatea de ansamblu cere ca pentru orice element
  - Elementul să fie mai mare (sau egal) decât cei doi descendenți (dacă există descendenți) - MAXHEAP
  - Elementul să fie mai mic (sau egal) decât cei doi descendenți (dacă există descendenți) - MINHEAP
- Mai formal: presupunând ca elementele ansamblului sunt reținute într-un vector dinamic  $v$  cu  $n$  elemente, proprietatea de ansamblu cere că pentru fiecare poziție  $i$ ,  $0 \leq i < n$ :
  - $v[i] \geq v[2 * i + 1]$  dacă  $2 * i + 1 < n$  și
  - $v[i] \geq v[2 * i + 2]$ , dacă  $2 * i + 2 < n$
- SAU
  - $v[i] \leq v[2 * i + 1]$  dacă  $2 * i + 1 < n$  și
  - $v[i] \leq v[2 * i + 2]$ , dacă  $2 * i + 2 < n$

# Exemple

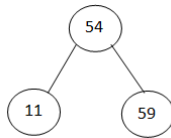
- Are proprietate de ansamblu?





# Exemple

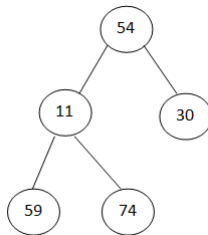
- Are proprietate de ansamblu?



- NU** are proprietate de ansamblu (54 este mai mare ca 11 și mai mic ca 59)

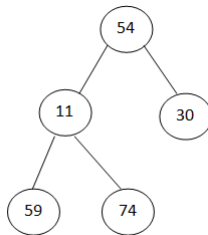
# Exemplu

- Are proprietate de ansamblu?



# Exemplu

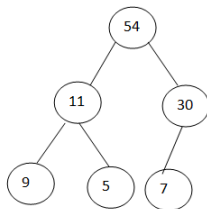
- Are proprietate de ansamblu?



- NU** are proprietatea de ansamblu (54 este mai mare ca descendenții, dar 11 nu este mai mare ca descendenții)

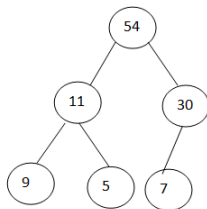
# Exemplu

- Are proprietate de ansamblu?



# Exemplu

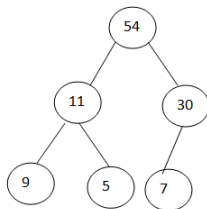
- Are proprietate de ansamblu?



- DA**, are proprietatea de ansamblu (54 mai mare ca 11 și 30, 11 mai mare ca 9 și 5, 30 mai mare ca 7). Pentru că relația dintre un nod și descendenții este  $\geq$ , spunem că avem un MAXHEAP. Oare de ce se numește MAXHEAP?

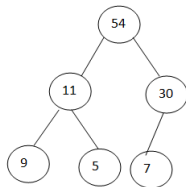
# Exemplu

- Are proprietate de ansamblu?



- DA**, are proprietatea de ansamblu (54 mai mare ca 11 și 30, 11 mai mare ca 9 și 5, 30 mai mare ca 7). Pentru că relația dintre un nod și descendenții este  $\geq$ , spunem că avem un MAXHEAP. Oare de ce se numește MAXHEAP?
- Într-un MAXHEAP rădăcina conține elementul maxim din ansamblu.

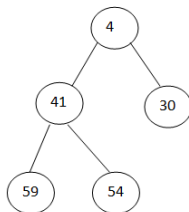
# Ansamblu



- Elementele ansamblului sunt stocate într-un vector dinamic, care este interpretat ca un arbore. În acest caz, vectorul conține elementele (în această ordine): 54, 11, 30, 9, 5, 7. Deși avem un MAXHEAP valid, vectorul nu e ordonat (nici nu trebuie să fie).

# Exemplu

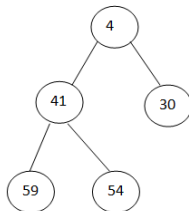
- Are proprietate de ansamblu?





# Exemplu

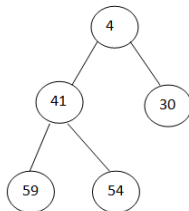
- Are proprietate de ansamblu?



- DA**, are proprietatea de ansamblu (4 mai mic ca 41 și 30, 41 mai mic ca 59 și 54). Pentru că relația dintre un nod și descendenții este  $\leq$ , spunem că avem un MINHEAP. Oare de ce se numește MINHEAP?

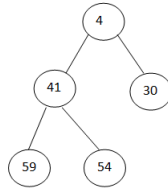
# Exemplu

- Are proprietate de ansamblu?



- DA**, are proprietatea de ansamblu (4 mai mic ca 41 și 30, 41 mai mic ca 59 și 54). Pentru că relația dintre un nod și descendenții este  $\leq$ , spunem că avem un MINHEAP. Oare de ce se numește MINHEAP?
- Într-un MINHEAP rădăcina conține elementul minim din ansamblu.

# Ansamblu



- Elementele ansamblului sunt stocate într-un vector dinamic, care este interpretat ca un arbore. În acest caz, vectorul conține elementele (în această ordine): 4, 41, 30, 59, 54. Deși avem un MINHEAP valid, vectorul nu e ordonat.

# Ansamblu - Operații

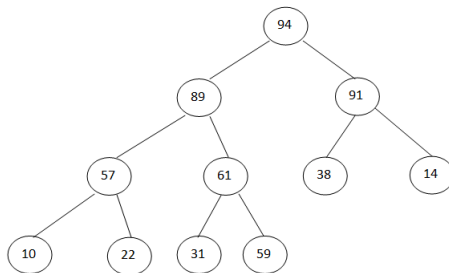
- Pentru un ansamblu avem doar 2 operații posibile:
  - Adăugare
  - Ștergere rădăcină
- Am discutat că ansamblu este folosit ca reprezentare pentru coadă cu priorități (imediat vom vedea cum exact) deci nici nu ne trebuie mai multe operații.
- Vom vedea cum sunt implementate aceste 2 operații și cum putem implementa o Coadă cu priorități folosind un ansamblu ca reprezentare. În exemplu vom discuta despre un MAXHEAP, MINHEAP-ul este implementat în mod similar.

# Adăugare în ansamblu

- La adăugare, elementul de adăugat vine neapărat pe prima poziție liberă din vector (altfel am strica structura de ansamblu).
- Dacă după adăugare, nu mai este respectată proprietatea de ansamblu, vom *urca* nodul adăugat: îl vom interschimba cu părintele lui, până părintele este mai mare, sau nodul ajunge rădăcină.

# Adăugare în ansamblu - exemplu

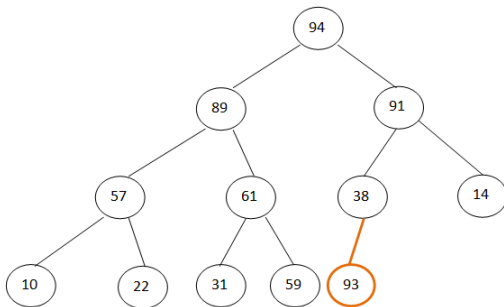
94	89	91	57	61	38	14	10	22	31	59		
0	1	2	3	4	5	6	7	8	9	10	11	12



- Să adăugăm elementul 93 în ansamblu.

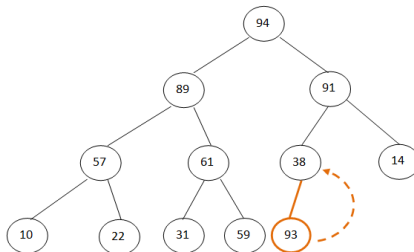
# Adăugare în ansamblu - exemplu

94	89	91	57	61	38	14	10	22	31	59	93	
----	----	----	----	----	----	----	----	----	----	----	----	--



# Adăugare în ansamblu - exemplu

94	89	91	57	61	38	14	10	22	31	59	93	
0	1	2	3	4	5	6	7	8	9	10	11	

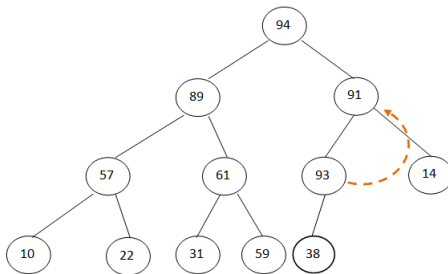


- Noul element a venit pe poziția 11 în vector. Părintele lui (cu care îl comparăm) este pe poziția  $11 - 1$  împărțită la 2, adică 5.



# Adăugare în ansamblu - exemplu

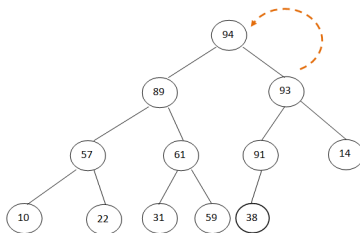
94	89	91	57	61	93	14	10	22	31	59	38	
0	1	2	3	4	5	6	7	8	9	10	11	



- După interschimbare, elementul nou este pe poziția 5. Îl comparăm cu părinte, care este pe poziția 2.

# Adăugare în ansamblu - exemplu

94	89	93	57	61	91	14	10	22	31	59	38	
0	1	2	3	4	5	6	7	8	9	10	11	



- După interschimbare, elementul nou este pe poziția 2. Îl comparăm cu părinte, care este pe poziția 0. Elementul de pe poziția 0 fiind mai mare, nu mai trebuie să facem interschimbare, am terminat adăugarea.

# Adăugare în ansamblu - implementare

- Să vedem cum se implementează operația de adăugare.
- Cum reprezentăm ansamblul? Ce câmpuri ar trebui să aibă structura ansamblu?
- Pentru simplitate, presupunem că elementele din ansamblu sunt numere. Putem avea și alt tip de date în ansamblu, trebuie numai să definim cum sunt elementele comparate.

# Adăugare în ansamblu - implementare

- Să vedem cum se implementează operația de adăugare.
- Cum reprezentăm ansamblul? Ce câmpuri ar trebui să aibă structura ansamblu?
- Pentru simplitate, presupunem că elementele din ansamblu sunt numere. Putem avea și alt tip de date în ansamblu, trebuie numai să definim cum sunt elementele comparate.

## Ansamblu:

cap: Întreg

len: Întreg

elem: Întreg[]

# Adăugare în ansamblu - implementare

**subalgoritm** adauga(e: întreg) **este:**

**dacă** this.len == this.cap **atunci**

*//vectorul dinamic folosit pentru ansamblu este plin*

*@ alocăm un vector mai mare și copiem elementele*

**sf\_dacă**

this.elem[this.len] = e *//punem elementul la final*

this.len = this.len + 1

*//începem procesul de urcare*

pozE, pozP, temp: întreg

pozE = this.len-1 *//poziția elementului*

pozP = (pozE-1) / 2 *//poziția părintelui*

*//continuăm pe pagina următoare*

# Adăugare în ansamblu - implementare

**cât timp**  $\text{pozE} > 0$  **ȘI**  $\text{this.elem}[\text{pozP}] < \text{this.elem}[\text{pozE}]$  **execută**

temp = this.elem[pozP]

this.elem[pozP] = this.elem[pozE]

this.elem[pozE] = temp

pozE = pozP *//resetăm poziția elementului*

pozP = (pozE-1) / 2 *//recalculăm poziția părintelui*

**sf\_cât timp**

**sf\_subalgoritm**

- Complexitate:

# Adăugare în ansamblu - implementare

**cât timp**  $\text{pozE} > 0$  **ȘI**  $\text{this.elem}[\text{pozP}] < \text{this.elem}[\text{pozE}]$  **execută**

$\text{temp} = \text{this.elem}[\text{pozP}]$

$\text{this.elem}[\text{pozP}] = \text{this.elem}[\text{pozE}]$

$\text{this.elem}[\text{pozE}] = \text{temp}$

$\text{pozE} = \text{pozP}$  *//resetăm poziția elementului*

$\text{pozP} = (\text{pozE}-1) / 2$  *//recalculăm poziția părintelui*

**sf\_cât timp**

**sf\_subalgoritm**

- Complexitate:  $O(\log_2 n)$  - unde  $n$  este numărul de elemente din ansamblu.

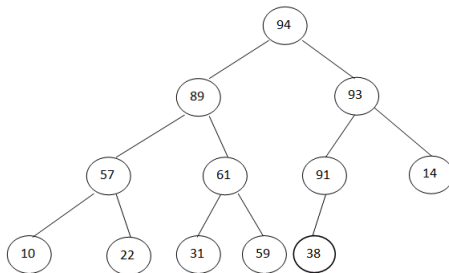
# Ștergere din ansamblu

- Dintr-un ansamblu putem șterge un singur element: rădăcina.
- Nu există ștergere de pe o poziție, nu există ștergerea unui element dat, doar rădăcina poate fi ștearsă.
- Când se șterge rădăcina, în primul pas se copiază ultimul element din vector în locul rădăcinii. Această mutare păstrează structura de ansamblu, dar poate strica proprietatea de ansamblu (rădăcina poate nu mai este mai mare ca fii).
- Pentru a restabili proprietatea de ansamblu, folosim un proces de *coborâre*. Elementul curent (care inițial este rădăcina) se compară cu maximul dintre fii, dacă maximul este mai mare, elementul curent coboară (schimbă locul cu fiu). Procesul se repetă până elementul curent este mai mare ca fii sau ajunge să fie frunză.



# Ștergere din ansamblu - Exemplu

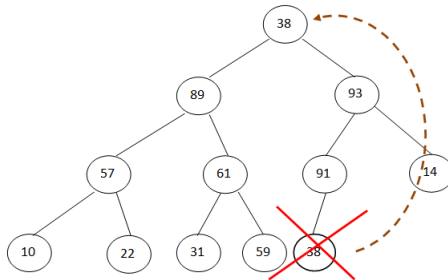
94	89	93	57	61	91	14	10	22	31	59	38	
0	1	2	3	4	5	6	7	8	9	10	11	



- Doar rădăcina poate fi ștearsă.

# Ștergere din ansamblu - Exemplu

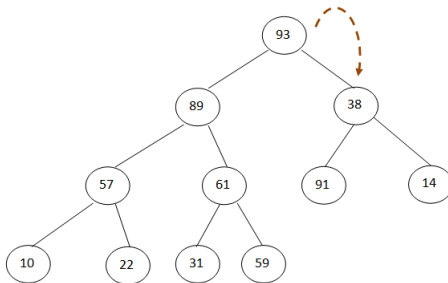
38	89	93	57	61	91	14	10	22	31	59		
0	1	2	3	4	5	6	7	8	9	10		



- Mutăm 38 în locul lui 94. Comparăm elementul 38 cu maximul dintre fii și dacă trebuie îl coborâm.

# Ștergere din ansamblu - Exemplu

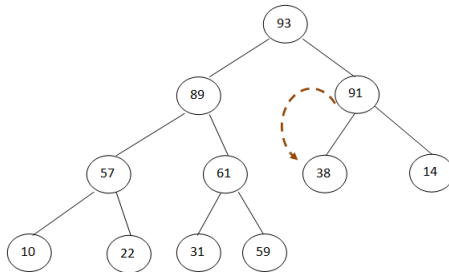
93	89	38	57	61	91	14	10	22	31	59		
0	1	2	3	4	5	6	7	8	9	10		



- Am schimbat 93 cu 38. În continuare comparăm elementul 38 cu maximul dintre fii și dacă trebuie îl coborâm.

# Ștergere din ansamblu - Exemplu

93	89	91	57	61	38	14	10	22	31	59		
0	1	2	3	4	5	6	7	8	9	10		



- 38 e frunză. Nu mai avem cu ce să comparăm, a terminat ștergerea (mai trebuie doar să returnăm elementul șters, 94).

# Ștergere din ansamblu - Implementare

**funcție** `sterge()` **este:**

```
//presupunem că ansamblul are măcar un element  
elemSters, pozE, fiuS, fiuD, fiuMax, temp: întreg  
elemSters = this.elem[0] //elementul care va fi sters  
this.elem[0] = this.elem[this.len-1] //mutăm ultimul element  
this.len = this.len - 1  
pozE = 0  
cât timp pozE < this.len execută  
    fiuS = pozE * 2 + 1  
    fiuD = pozE * 2 + 2  
    fiuMax = fiuS //presupunem că fiu stâng e mai mare  
//continuăm pe pagina următoare
```

```
dacă fiuD < this.len atunci //are 2 fii  
    dacă this.elem[fiuD] > this.elem[fiuS] atunci  
        fiuMax = fiuD  
    sf_dacă  
sf_dacă  
dacă fiuS < this.len SI this.elem[fiuMax] > this.elem[pozE] atunci  
    temp = this.elem[fiuMax]  
    this.elem[fiuMax] = this.elem[pozE]  
    this.elem[pozE] = temp  
    pozE = fiuMax  
altfel //fiuMax este mai mic sau nu are fii. Ne oprim  
    pozE = this.len  
sf_dacă  
sf_cât timp  
    returnează elemSters  
sf_funcție
```

# Ștergere din ansamblu - Implementare

- Complexitate:

# Ștergere din ansamblu - Implementare

- Complexitate:  $O(\log_2 n)$



# Coadă cu Priorități pe ansamblu

- Ansamblu poate fi folosit ca reprezentare pentru o coadă cu priorități, unde rădăcina ansamblului este elementul cu prioritate maximă.
- Operații și complexități:
  - adaugă - adăugăm elementul cu prioritate în ansamblu.  $O(\log_2 n)$
  - sterge - ștergem un element din ansamblu (care va fi cel cu prioritate maximă).  $O(\log_2 n)$
  - element - returnă elementul cu prioritate maximă (de pe poziția 0).  $\Theta(1)$
  - vidă - verificăm dacă vectorul e vid.  $\Theta(1)$
- Folosind un ansamblu, o secvență de  $n$  adăugări în coada cu priorități, urmat de  $n$  ștergeri are complexitate  $O(n \log_2 n)$