

# Vector Dinamic

## DYNAMIC ARRAY

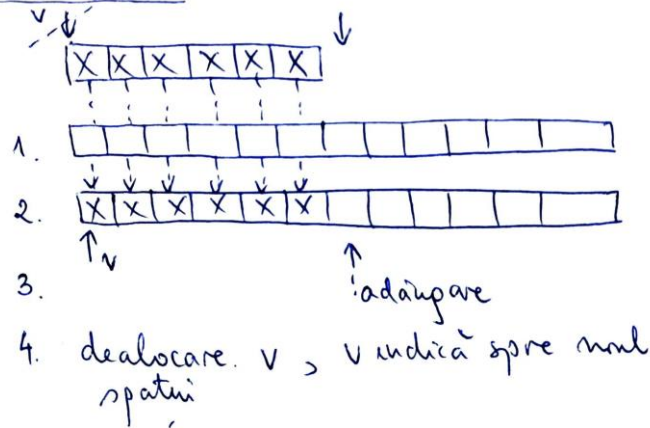
### Observații

1. Un tablou este static: nu pot fi inserate sau șterse celule.
2. Vector - tablou unidimensional
3. Reprezentarea vectorilor este **secvențială**, adică elementele sunt memorate în locații succesive de memorie
  - Detaliu de implementare: spațiul de memorare poate fi alocat *static* sau *dinamic* (în timpul execuției programului).
  - Accesul la elemente este **direct** (prin intermediul indicilor)  $\Rightarrow$  complexitate-timp  $\theta(1)$ .
4. Există așa numitele **tablouri dinamice** - **vectori dinamici** ("*dynamic arrays*"), care sunt tablouri unidimensionale cu caracter dinamic, a căror lungime se modifică în timp și în care se pot insera, respectiv șterge elemente.
  - spațiul de memorare alocat tabloului crește automat în momentul în care nu mai există spațiu suficient pentru inserarea unui nou element  $\Rightarrow$  alocare dinamică
6. Alegerea unei reprezentări bazate pe **alocarea dinamică** a memoriei prezintă avantajul amânării cunoașterii mărimii efective a vectorului până la momentul execuției. Un tablou alocat **static** impune specificarea mărimii tabloului ca o constantă.

### Caracteristici

1. Vectorul dinamic - reprezentare:
  - a. capacitatea sa ( $cp$ ) – numărul de locații alocat vectorului;
  - b. dimensiunea sa ( $n$ ) – lungimea efectivă a vectorului;
  - c. elementele vectorului ( $v = e_1, e_2, \dots, e_n$ ).
    - în C/C++,  $v$  este adresa la care se memorează primul element al vectorului
2. Dacă la adăugarea unui element în vector se depășește capacitatea vectorului, atunci se **mărește** capacitatea acestuia (de obicei se dublează, pentru generalitate se poate considera un anumit raport de creștere a capacității față de dimensiunea vectorului).
  - Ca detaliu de implementare, în cazul în care spațiul de memorie necesar stocării elementelor vectorului se alocă dinamic, atunci se va face o **realocare** a acestui spațiu (la noua capacitate), se copiază elementele din vechiul spațiu în noul spațiu, se adaugă/inserează elementul, după care se dealocă vechiul spațiu.

## Redimensionare



- Cu toate că această operație de redimensionare/reallocare este costisitoare ( $\theta(n)$ ), operația de adăugare a unui element la sfârșitul vectorului are, totuși, *complexitatea-timp amortizată*  $\theta(1)$

### 3. Implementări în bibliotecile existente în diferite limbaje

- Java – clasa **Vector**
  - implementată încât să funcționeze cu acces concurrent
  - dacă nu se dorește acces concurrent  $\Rightarrow$  **ArrayList** (lista reprezentată secvențial pe tablou)
- STL din C++ - clasa **Vector**
  - Considerat un container de tip *secvență* (acces poziție)
  - implementat ca un *vector dinamic*

### 4. **VectorDinamic** - reprezentare secvențială a containerului **Lista**.

### 5. **VectorDinamic** se consideră potrivit pentru:

- Accesare element de pe o anumită poziție  $\theta(1)$  (complexitate-timp).
- Iterare elemente în orice ordine – timp liniar  $\theta(n)$
- Adăugare element la sfârșit – complexitate-timp **amortizată**  $\theta(1)$
- Ștergere element de la sfârșit - timp constant  $\theta(1)$

În continuare, pentru un număr natural  $n$  folosim notația  $[n] = \{1, 2, \dots, n\}$ . De asemenea, vom considera următoarele:

- adăugarea și ștergerea elementelor se poate face la atât la sfârșitul vectorului, cât și pe orice poziție în vector.
- la redimensionarea capacității vectorului, pp. că aceasta se dublează.

Deși este unanim acceptat faptul că reprezentarea tablourilor unidimensionale este *secvențială*, se poate abstractiza tipul de dată **VectorDinamic**.

## TAD VectorDinamic

### domeniu

$\mathbf{V} = \{ \mathbf{v} \mid v = (cp, n, e_1 e_2 \dots e_n), cp, n \in \mathbf{N}, n \leq cp, e_i \text{ sunt de tip } \mathbf{TElement} \}$

## operații (interfața)

*creează*( $v, cp$ )

{ constructor - se creează un vector cu lungime 0, având capacitatea  $cp$  }

**pre:**  $cp \in \mathbf{Natural}$

**post:**  $v \in \mathbf{V}, v.n=0, v.cp=cp$

@ aruncă excepție dacă  $cp$  e negativ

*dim*( $v$ )

**pre:**  $v \in \mathbf{V}$

**post:**  $dim$  = lungimea vectorului  $v$  (numărul de elemente)  $\in \mathbf{Natural}$

*element*( $v, i, e$ )

**pre:**  $v \in \mathbf{V}, i \in \mathbf{Natural}, i \in [v.n]$

**post:**  $e \in \mathbf{TElement}, e = v.e_i$  (elementul de pe poziția  $i$  din vectorul  $v$ )

@ aruncă excepție dacă  $i$  e în afara intervalului  $[v.n]$

*modifică*( $v, i, e$ )

**pre:**  $v \in \mathbf{V}, i \in \mathbf{Natural}, i \in [v.n], e \in \mathbf{TElement}$

**post:**  $v' \in \mathbf{V}, v'.e_i = e$  (al  $i$ -lea element din  $v'$  devine  $e$ )

@ aruncă excepție dacă  $i$  e în afara intervalului  $[v.n]$

*adaugaSfarsit*( $v, e$ )

{ se adaugă la sfârșitul vectorului elementul  $e$ ; dacă  $v.n=v.cp$  atunci crește capacitatea }

**pre:**  $v \in \mathbf{V}, e \in \mathbf{TElement}$

**post:**  $v' \in \mathbf{V}, v'.n = v.n + 1$

$(v.cp = v.n) \Rightarrow (v'.cp = v.cp * 2, v'.e[v'.n] = e)$

*adaugaPozitie*( $v, i, e$ )

{ se adaugă pe poziția  $i$  elementul  $e$ ; dacă  $v.n=v.cp$  atunci crește capacitatea }

**pre:**  $v \in \mathbf{V}, i \in \mathbf{Natural}, i \in [v.n] + 1, e \in \mathbf{TElement}$

**post:**  $v' \in \mathbf{V}, v'.n = v.n + 1$

$(v.cp = v.n) \Rightarrow (v'.cp = v.cp * 2, v'.e[j] = v.e[j-1] \forall j = v'.n, v'.n-1, \dots, i+1, v'.e[i] = e)$

@ aruncă excepție dacă  $i$  e în afara intervalului  $[v.n]$

*stergeSfarsit*( $v, e$ )

{ se șterge elementul de la sfârșitul vectorului }

**pre:**  $v \in \mathbf{V}, v.n > 0$

**post:**  $e \in \mathbf{TElement}, e = v.e[v.n], v' \in \mathbf{V}, v'.n = v.n - 1$

***ștergePozitie***( $v, i, e$ )

{se șterge pe poziția  $i$  a vectorului }

**pre:**  $v \in \mathbf{V}$ ,  $v.n > 0$ ,  $i \in \mathbf{Natural}$ ,  $i \in [v.n]$

**post:**  $e \in \mathbf{TElement}$ ,  $e = v.e[i]$ ,  $v' \in \mathbf{V}$ ,  $v'.n = v.n - 1$ ,  $v'.e[j] = v.e[j + 1] \forall j = i, i + 1, \dots, v'.n$   
@ aruncă excepție dacă  $i$  e în afara intervalului  $[v.n]$

***iterator***( $v, i$ )

{se creează un iterator pe vectorul  $v$ }

**pre:**  $v \in \mathbf{V}$

**post:**  $i \in \mathbf{I}$ ,  $i$  este iterator pe vectorul  $v$

***distruge***( $v$ )

{destructor}

**pre:**  $v \in \mathbf{V}$

**post:** vectorul  $v$  a fost 'distrus' (spațiul de memorie alocat a fost eliberat)

***....alte operații....***

Elementele unui vector dinamic pot fi tipărite în două moduri:

1. Prin iterator, ca orice container.
2. Folosind accesul la elemente prin indici, datorită reprezentării secvențiale.

Ca urmare, tipărirea se poate face

1. **subalgoritmul** *tipărire*( $v$ ) **este** {complexitate-timp  $\theta(n)$  }  
{**pre:**  $v$  este un vector dinamic}  
{**post:** se tipăresc elementele vectorului}  
    **iterator**( $v, i$ ) {vectorul își construiește iteratorul}  
    **CâtTimp** **valid**( $i$ ) **execută** {cât timp iteratorul e valid}  
        **element**( $i, e$ ) {se obține elementul curent din iterație}  
        **@ tipărește**  $e$  {se tipărește elementul curent}  
        **următor**( $i$ ) {se deplasează iteratorul}  
    **SfCâtTimp**  
    **sfTipărire**
2. **subalgoritmul** *tipărire*( $v$ ) **este** {complexitate-timp  $\theta(n)$  }  
{**pre:**  $v$  este un vector dinamic}  
{**post:** se tipăresc elementele vectorului}  
    **pentru**  $i \leftarrow 1, \dim(v)$  **execută**  
        **element**( $v, i, e$ ) {se obține elementul de poziția}  
        **@ tipărește**  $e$  {se tipărește elementul curent}  
    **sfPentru**  
    **sfTipărire**

## Operația adaugăSfârșit

### Reprezentare

#### VectorDinamic

cp: Integ {capacitatea maximă de memorare}

n: Integ {dimensiunea efectivă a vectorului – număr elemente memorate}

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

subalgoritm adaugăSfârșit (v, e) este  $\{O(n)\}$

```
dacă v.n = v.cp atunci
    { vectorul e plin }
    @ redimensionare (realocare, copiere, ...)
sf.dacă
    { dacă e caml, vectorul s-a redimensionat }
    v.e[v.n+1] ← e
    v.n ← v.n+1
```

### Complexitate amortizată

- timp de execuție mediu pentru o secvență de operații pentru care se consideră evaluarea în cel mai defavorabil caz
- diferă de evaluarea pentru cazul mediu, nu sunt folosite probabilități și se evaluează secvențe de operații, pentru care se face media

De ex. analiza amortizată pentru operația **adaugăSfârșit**

- dacă tabloul e plin, se dublează spațiul (realocare, copiere, eliberarea vechiului spațiu)
- dacă se consideră  $n$  operații de adăugare la sfârșit,  $n$  fiind dimensiunea tabloului  $\Rightarrow$  redimensionarea se va face cel mult o dată  $\Rightarrow$

$$\begin{array}{l} n \text{ operații } \underline{\text{adaugăSfârșit}} \\ \frac{\theta(1) + \theta(1) + \dots + \theta(1) + \theta(n)}{n-1 \text{ ori}} \\ \hline n \end{array} = \frac{\theta(n)}{n} = \theta(1)$$

#### IteratorVectorDinamic

v: VectorDinamic {ca implementare, va fi o referință către container}

curent: Integ {poziția unui element din vector}

subalgoritm creează(i, v) este  $\{ \theta(1) \}$

{ se creează iteratorul  $i$  pe vectorul dinamic  $v$  }

$i.v \leftarrow v$

$i.curent \leftarrow 1$

**sfCreează**

**funcția**  $valid(i)$  **este**  $\{ \theta(1) \}$

{ verifică dacă iteratorul este valid }

$valid \leftarrow (i.curent \leq i.v.n)$

**sfValid**

**subalgoritm**  $element(i, e)$  **este**  $\{ \theta(1) \}$

{  $e$  este elementul curent referit de iterator }

$e \leftarrow i.v[i.curent]$

**sfElement**

**subalgoritm**  $urmator(i)$  **este**  $\{ \theta(1) \}$

{ se deplasează referința curent în container }

$i.curent \leftarrow i.curent + 1$

**sfUrmator**

## Observații

1. Complexitatea operațiilor unui TAD poate fi determinată după ce s-au luat decizii legate de reprezentarea și modul de implementare a TAD-ului. Considerând reprezentarea secvențială a unui vector dinamic, operațiile de bază din interfața **TAD VectorDinamic** au complexitățile:

- ***dim*** complexitate-timp  $\theta(1)$
- ***element*** complexitate-timp  $\theta(1)$
- ***modifică*** complexitate-timp  $\theta(1)$
- ***adaugaSfarsit*** complexitate-timp amortizată  $\theta(1)$
- ***adaugaPozitie*** complexitate-timp  $O(n)$
- ***stergeSfarsit*** complexitate-timp  $\theta(1)$   
!!! dacă se folosește redimensionare, atunci complexitatea-timp amortizată e  $\theta(1)$
- ***stergePozitie*** complexitate-timp  $O(n)$
- ***iterator*** complexitate-timp  $\theta(1)$

2. Deoarece se permite modificarea capacității vectorului, se impune pentru implementare folosirea **alocării dinamice a memoriei**.
3. După cum menționam anterior, este posibil ca la redimensionarea capacității vectorului să se folosească un raport de creștere  $RC$  (a capacității față de numărul de elemente din vector) - în specificația anterioară am folosit  $RC = 2$ .
4. Pentru o gestionare mai eficientă a spațiului de memorie alocat vectorului, pentru a evita situații în care numărul de elemente efectiv memorate în vector este mult mai mic decât capacitatea de memorare a

acestui, se poate folosi un raport maxim  $RM$ , care asigură faptul că nu se ajunge la o creștere a capacității prea mare față de numărul de elemente. Ceea ce înseamnă că la operația de ștergere, în cazul în care  $\frac{v.cp}{v.n} > RM$ , atunci se va micșora capacitatea vectorului (ceea ce va implica realocare/copiere elemente...)

5. Spre deosebire de operația *adaugaSfarsit*, a cărei complexitate-timp defavorabilă este  $\theta(n)$ , dar totuși complexitatea-timp amortizată este  $\theta(1)$ , operația *adaugaPozitie* are complexitatea-timp amortizată  $O(n)$  (ca și cea defavorabilă).
6. În interfața TAD VectorDinamic pot fi adăugate și alte operații, spre exemplu: verificarea dacă vectorul este sau nu vid (fără elemente), căutarea unui element în vector și returnarea poziției pe care apare, transformarea în string (*toString*), transformarea în vector (*toArray*) etc.

### **Reprezentare secvențială – caracteristici**

- inserări, ștergeri  $O(n)$
- gestionare inefficientă a spațiului de memorare
- accesul la elemente este **direct**  $\theta(1)$
- adăugare la sfârșit  $\theta(1)$  amortizat
- ștergere la sfârșit  $\theta(1)$  (amortizat, dacă se face redimensionare)