Vector Dinamic

DYNAMIC ARRAY

Observaţii

1. Un tablou este static: nu pot fi inserate sau şterse celule.
2. Vector - tablou unidimensional
3. Reprezentarea vectorilor este **secvenţială**, adică elementele sunt memorate în locaţii succesive de memorie
   - Detaliu de implementare: spaţiul de memorare poate fi alocat *static* sau *dinamic* (în timpul execuţiei programului).
   - Accesul la elemente este **direct** (prin intermediul indicilor) $\Rightarrow$ complexitate-timp $\theta(1)$.
4. Tablourile sunt des folosite în programare şi pentru reprezentarea altor structuri de date.
5. Există aşa numitele ***tablouri dinamice - vectori dinamici*** (***"dynamic arrays"***), care sunt tablouri unidimensionale cu caracter dinamic, a căror lungime se modifică în timp, altfel spus în astfel de structuri se pot insera, respectiv şterge elemente.
   - În anumite abordări, adăugarea respectiv ştergerea elementelor se poate face doar la sfârşitul vectorului dinamic, nu pe orice poziţie în vector.

Caracteristici

1. Vectorul dinamic - reprezentare:
   a. capacitatea sa (*cp*) – numărul de locaţii alocat vectorului;
   b. dimensiunea sa (*n*) – lungimea efectivă a vectorului;
   c. elementele vectorului ($e = e_1, e_2, ..., e_n$).
      - în C, *e* este adresa la care se memorează primul element al vectorului
2. Dacă la adăugarea unui element în vector se depăşeşte capacitatea vectorului, atunci se **măreşte** capacitatea acestuia (de obicei se dublează, pentru generalitate se poate considera un anumit raport de creştere a capacităţii faţă de dimensiunea vectorului).
   - Ca detaliu de implementare, în cazul în care spaţiul de memorie necesar stocării elementelor vectorului se alocă dinamic, atunci se va face o **realocare** a acestui spaţiu (la noua capacitate), se copiază elementele din vechiul spaţiu în noul spaţiu, se adaugă/inserează elementul, după care se dealocă vechiul spaţiu.
   - Cu toate că această operaţie de redimensionare/realocare este costisitoare ($\theta(n)$), operaţia de adăugare a unui element la sfârşitul vectorului are, totuşi, *complexitatea-timp amortizată* $\theta(1)$
3. Implementări în bibliotecile existente în diferite limbaje
   - Java – clasa **Vector**
     - implementată încât să funcţioneze cu acces concurent
     - dacă nu se doreşte acces concurent $\Rightarrow$ **ArrayList** (lista reprezentată secvenţial pe tablou)
   - STL din C++ - clasa **Vector**
     - Considerat un container de tip *secvenţă* (acces prin rang şi poziţie)
     - implementat ca un *vector dinamic*

4. **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 notaţia $[n]=\{1,2,..,n\}$. De asemenea, vom considera următoarele:

- adăugarea şi ştergerea elementelor se poate face la atât la sfârşitul vectorului, cât şi pe orice poziţie în vector.
- la redimensionarea capacităţii vectorului, pp. că aceasta se dublează.

Vom da în continuare specificaţia Tipului Abstract de Date **VectorDinamic**.

TAD VectorDinamic

domeniu

$\mathbf{V}=\{\mathbf{v} \mid v=(cp,n,e_1e_2...e_n)$, $cp,n\in N, n\le cp$, e_i sunt de tip **TElement** $\}$

operaţii (**interfaţa** TAD-ului VectorDinamic)

creează(*v*, *cp*)
{constructor - se creează un vector cu lungime 0, având capacitatea *cp*}
pre: $cp\in$ **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) $\in$ **Natural**

element(*v*, *i*, *e*)
pre: $v\in \mathbf{V}$, $i\in$ **Natural,** $i\in[v.n]$
post: $e\in$ **TElement,** $e=v.e_i$ (elementul de pe poziţia *i* din vectorul *v*)
@ aruncă excepție dacă *i* e înafara intervalului [*v.n*]

modifică(*v*, *i*, *e*)
pre: $v\in \mathbf{V}$, $i\in$ **Natural,** $i\in[v.n]$, $e\in$ **TElement**
post: $v'\in \mathbf{V}$, $v'.e_i=e$ (al *i*-lea element din *v*' devine *e*)
@ aruncă excepție dacă *i* e înafara intervalului [*v.n*]

adaugaSfarsit(v, e)

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

pre: $v \in \mathbf{V}$, $e \in$ **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 poziţia i elementul e; dacă $v.n$=$v.cp$ atunci creşte capacitatea}

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

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

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

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

stergeSfarsit(v, e)

{se şterge elementul de la sfârşitul vectorului }

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

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

ştergePozitie(v, i, e)

{se şterge pe poziţia i a vectorului }

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

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

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

iterator(v, i)

{se creează un iterator pe vectorul v}

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

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

distruge(v)

{destructor}

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

post: vectorul v a fost 'distrus' (spaţiul de memorie alocat a fost eliberat)

....alte operaţii....

Observaţii

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 $\theta(1)$
   - ***ştergePozitie*** 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 menţionam anterior, este posibil ca la redimensionarea capacităţii vectorului să se folosească un raport de creştere RC (a capacităţii faţă de numărul de elemente din vector) - în specificaţia anterioară am folosit $RC = 2$.
4. Pentru o gestionare mai eficientă a spaţiului de memorie alocat vectorului, pentru a evita situaţii în care numărul de elemente efectiv memorate în vector este mult mai mic decât capacitatea de memorare a acestuia, se poate folosi un raport maxim RM, care asigură faptul că nu se ajunge la o creştere a capacităţii prea mare faţă de numărul de elemente. Ceea ce înseamnă că la operaţia de ştergere, în cazul în care $\frac{v.cp}{v.n} > RM$, atunci se va micşora capacitatea vectorului (ceea ce va implica realocare/copiere elemente...)
5. Spre deosebire de operaţia ***adaugaSfarsit*,** a cărei complexitate-timp defavorabilă este $\theta(n)$, dar totuşi complexitatea-timp amortizată este $\theta(1)$, operaţia ***adaugaPozitie*** are complexitatea-timp amortizată $O(n)$ (ca şi cea defavorabilă).
6. În interfaţa TAD VectorDinamic pot fi adăugate şi alte operaţii, spre exemplu: verificarea dacă vectorul este sau nu vid (fără elemente), căutarea unui element în vector şi returnarea poziţiei pe care apare, transformarea în string (*toString*), transformarea în vector (*toArray*) etc.

   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 secvenţiale.

Ca urmare, tipărirea se poate face

```
1. subalgoritmul tipărire(v) este          {complexitate-timp θ(n) }
   {pre: v este un vector dinamic}
   {post: se tipăresc elementele vectorului}
        iterator(v,i)                      {vectorul îşi construieşte iteratorul}
        CâtTimp valid(i) execută           {cât timp iteratorul e valid}
            element(i, e)                  {se obţine elementul curent din iteraţie}
            @ tipăreşte e                  {se tipăreşte elementul curent}
            următor(i)                     {se deplasează iteratorul}
        SfCâtTimp
   sfTipărire
```

Ţinând cont de reprezentarea secvenţială, reprezentarea iteratorului şi implementarea operaţiilor iteratorului pe vector sunt similare celor discutate în Seminarul 1 (colecţia reprezentată ca un vector de elemente).

```
2. subalgoritmul tipărire(v) este          {complexitate-timp θ(n) }
   {pre: v este un vector dinamic}
   {post: se tipăresc elementele vectorului}
        pentru i←1,dim(v) execută
            element(v, i, e)               {se obţine elementul de poziţia}
            @ tipăreşte e                  {se tipăreşte elementul curent}
        sfPentru
   sfTipărire
```

Reprezentare secvențială – caracteristici

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