Tipuri structurate de date

LISTE

STIVE

COZI

Sumar

Competenţe generale

- *identificarea datelor care intervin într-o problemă şi a relaţiilor dintre acestea*
- *elaborarea algoritmilor de rezolvare a problemelor*
- *aplicarea algoritmilor fundamentali în prelucrarea datelor*
- *identificarea conexiunilor dintre informatică şi societate*

Competenţe specifice

- *evidenţierea necesităţii structurării datelor*
- *prelucrarea datelor structurate*
- *alegerea structurii de date adecvată rezolvării unei probleme*
- *elaborarea unui algoritm de rezolvare a unei probleme din aria currciculară a specialităţii*
- *alegerea unui algoritm eficient de rezolvare a unei probleme*
- *identificarea aplicaţiilor informaticii în viaţa socială*
- *elaborarea şi implementarea unor algoritmi de rezolvare a unor probleme cotidiene*

O ***structură de date*** este o colecţie de elemente asupra căreia se pot efectua anumite operaţii.

Clasificarea structurilor de date:

1. În funcţie de ***tipul datelor*** memorate în cadrul structurii, structurile de date se pot clasifica în:
• structuri de date ***omogene*** – toate datele componente sunt de acelaşi tip (de exemplu tabloul);
• structuri de date ***neomogene*** – pot conţine date de tipuri diferite (de exemplu înregistrarea).

2. În funcţie de ***modul de alocare a memoriei interne***, structurile de date sunt de două tipuri:
• structuri de date ***statice*** – ocupă o zonă de memorie de dimensiune fixă, alocată pe întreaga durată a execuţiei blocului în care este declarată (de exemplu tabloul, fişierul, lista, stiva, coada);
• structuri de date ***dinamice*** – ocupă o zonă de memorie de dimensiune variabilă, alocată pe parcursul execuţiei programului, la cererea explicită a programatorului (de exemplu lista, stiva, coada).

Lista este o structură de date de tip secvenţial, constituită dintr-o succesiune de elemente de acelaşi tip. Fiecare element din listă are un succesor (cu excepţia ultimului element al listei) şi un predecesor (cu excepţia primului element din listă).

- utilizarea listelor se impune ori de câte ori este necesară organizarea într-o formă secvenţială a unui ansamblu de informaţii;

Operaţii elementare care se pot efectua asupra unei liste:
- crearea unei liste vide;
- inserarea unui element în listă;
- eliminarea unui element din listă;
- accesarea unui element din listă;
- afişarea elementelor unei liste.

Exemplu

`14, 2, 6, 77`

LISTA	14	2	6	77

Reprezentarea listelor

Cel mai siplu mod de a **implementa o listă** constă în memorarea elementelor sale într-un vector.

Declararea listei:

```
<tip> <identificator>[<nr_elemente>];
```

Exemplu

```
int LISTA[11];
```

1. Crearea unei liste vide

- se iniţializează numărul de elemente din listă cu `0`;

Exemplu

```
int n=0;
```

2. Inserarea unui element în listă

- se deplasează cu o poziţie la dreapta toate elementele din listă;
- se inserează elementul;
- creşte cu o unitate numărul de elemente ale listei;

Exemplu

```
for(i=n;i>=poz;i--)
     LISTA[i+1]=LISTA[i];
LISTA[poz]=e;
n++;
```

Exerciţiu

- inserarea unui element la începutul listei;
- inserarea unui element la sfârşitul listei.

3. Eliminarea unui element din listă

- se deplasează cu o poziţie la stânga toate elementele din listă, începând cu poziţia elementului care se elimină;
- scade cu o unitate numărul de elemente ale listei;

Exemplu

```
e=LISTA[poz];
for(i=poz;i<n;i++)
   LISTA[i]=LISTA[i+1];
n--;
```

Exerciţiu

- ştergerea primului element din listă;
- ştergerea ultimului element din listă.

4. Accesarea unui element din listă

- se memorează un element al listei într-o variabilă;

Exemplu

```
e=LISTA[poz];
```

5. Afişarea elementelor din listă

- se parcurge lista element cu element;

Exemplu

```
for(i=1;i<=n;i++)
    cout<<LISTA[i]<<' ';
```

4. Structura de date de tip stivă

Stiva este un tip particular de listă, pentru care atât operaţia de inserare a unui element în structură, cât şi operaţia de extragere a unui element se realizează la un singur capăt, denumit ***vârful*** stivei.

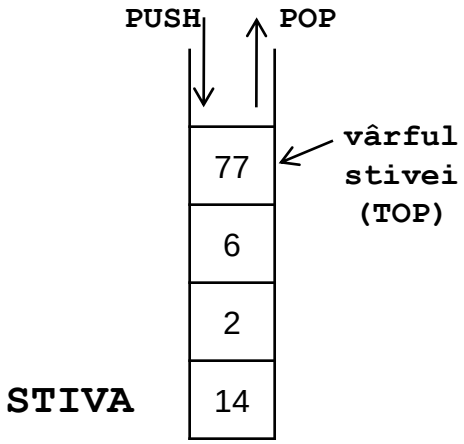
- singurul element din stivă la care avem acces direct este cel de la vârful stivei;
- trebuie cunoscut în permanenţă vârful stivei;
- stiva este utilizată atunci când programul trebuie să amâne execuţia unor operaţii, pentru a le executa ulterior, în ordinea inversă apariţiei lor;
- stiva funcţionează după principiul **`LIFO`** (**`Last In First Out`**);

Operaţii elementare care se pot efectua asupra unei stive:

- crearea unei stive vide;
- înserarea unui element în stivă (**PUSH**);
- extragerea unui element din stivă (**POP**);
- accesarea elementului de la vârful stivei (**TOP**).

Exemplu

14, 2, 6, 77



Reprezentarea stivelor

Cel mai siplu mod de a **implementa o stivă** constă în memorarea elementelor sale într-un vector.

Declararea stivei:

```
<tip> <identificator>[<nr_elemente>];
```

- **Exemplu**

```
int STIVA[11];
```

1. Crearea unei stive vide

- se iniţializează numărul de elemente din stivă cu `0`;

Exemplu

```
int vârf=0;
```

2. Inserarea unui element în stivă

- se verifică dacă stiva nu este plină;
- se măreşte vârful stivei;
- se plasează la vârf noul element;

Exemplu

```
if(varf==10)
    cout<<"Stiva este plină";
 else
 {
     varf++;
     STIVA[varf]=e;
 }
```

3. Extragerea unui element din stivă

- se verifică dacă stiva nu este vidă;
- se reţine elementul din vârful stivei într-o variabilă;
- se micşorează cu o unitate vârful stivei;

Exemplu

```
if(varf==0)
        cout<<"Stiva este vidă";
 else
 {
     e=STIVA[varf];
     varf--;
 }
```

4. Accesarea elementului din vârful stivei

- se memorează elementul din vârful stivei într-o variabilă;

Exemplu

```
e=STIVA[varf];
```

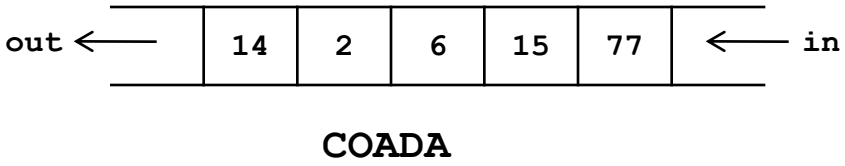
Coada este un tip particular de listă, pentru care operaţia de inserare a unui element se realizează la un capăt, iar operaţia de extragere a unui element se realizează la celălalt capăt.

- singurul element din coadă la care avem acces direct este cel de la început;
- trebuie cunoscut în permanenţă începutul cozii şi sfârşitul cozii;
- coada este utilizată atunci când informaţiile trebuie prelucrate exact în ordinea în care “au sosit” şi ele sunt reţinute în coadă până când pot fi prelucrate;
- coada funcţionează după principiul **`FIFO`** (**`First In First Out`**);

Operaţii elementare care se pot efectua asupra unei cozi:
- crearea unei cozi vide;
- înserarea unui element în coadă;
- eliminarea unui element din coadă;
- accesarea unui element din coadă.

Exemplu

`14, 2, 6, 15, 77`



COADA

Reprezentarea cozilor

Cel mai simplu mod de a **implementa o coadă** constă în memorarea elementelor sale într-un vector.

Declararea cozii:

```
<tip> <identificator>[<nr_elemente>];
```

- **Exemplu**

```
int COADA[11];
```

1. Crearea unei cozi vide

- se iniţializează numărul de elemente din coadă cu **0**, pentru aceasta se iniţializează variabilele **început** şi **sfârşit**;

Exemplu

```
int început=1, sfarsit=0;
```

2. Inserarea unui element în coadă

- se verifică dacă coada nu este plină;
- se măreşte sfârşitul cozii cu o unitate;
- se plasează la sfârşit noul element;

Exemplu

```
if(sfarsit==10)
        cout<<"Coada este plină";
 else
 {
      sfarsit++;
      COADA[sfarsit]=e;
 }
```

3. Eliminarea unui element din coadă

- se verifică dacă coada nu este vidă;
- se reţine elementul de la începutul cozii într-o variabilă;
- se măreşte cu o unitate începutul cozii;

Exemplu

```
if(inceptu>sfarsit)
    cout<<"Coada este vidă";
 else
 {
      e=COADA[inceput];
      inceput++;
 }
```

4. Accesarea unui element din coadă

- se memorează elementul de la începutul cozii într-o variabilă;

Exemplu

```
e=COADA[inceput];
```

Fişă de lucru

- Întrebări liste, stive, cozi
- Aplicaţii liste, stive, cozi

7. Bibliografie şi webografie

1. Cerchez E., Şerban M., *Informatică. Manual pentru clasa a X-a*, Editura Polirom, Iaşi, 2000
2. Mateescu G, Moraru P., *Informatica. Manual pentru calsa a X*, Editura Donaris, Sibiu, 2006
3. Popescu C., *Culegere de probleme de informatică*, Editura Donaris-Info, Sibiu, 2002
4. Ministerul Educaţiei, Cercetării şi Tineretului, Centrul Naţional pentru Curriculum şi Evaluare în Învăţământul Preuniversitar, *Proba scrisă la informatică. Examenul de bacalaureat – Variante (1-100)* , Bucureşti 2008
5. http://ro.wikipedia.org/wiki/Stiv%C4%83_(structur%C4%83_de_date)