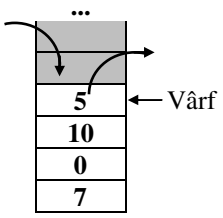
5. STIVA ŞI COADA

5.1. Stiva

Stiva este o structură de date abstractă 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 vârf.

Operaţii caracteristice



Singurele operaţii ce pot fi executate cu o stivă sunt:

- crearea unei stive vide;
- inserarea unui element în stivă (operaţie denumită în literatura de specialitate **PUSH**[1])
- extragerea unui element din stivă (operaţie denumită în termeni de specialitate **POP**[2])
- accesarea elementului de la vârf (operaţie denumită **TOP**[3]).

Ca să ne imaginăm mai bine funcţionarea unei stive, să ne gândim cum lucrăm cu un teanc de farfurii. Când dorim să punem o farfurie în teanc, o punem deasupra, când dorim să luăm o farfurie din teanc, o luăm tot pe cea de deasupra. Motivul este lesne de înţeles: nu ne-am propus să spargem farfuriile!

Acest mod de funcţionare face ca ultimul element inserat în stivă să fie primul extras. Din acest motiv, *stiva* este definită şi ca o structură de date care funcţionează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut – Ultimul Intrat Primul Ieşit).

Care este utilitatea stivelor?

În informatică stiva joacă un rol fundamental. Pentru a înţelege mecanisme fundamentale ale programării (de exemplu, funcţiile sau recursivitatea) este necesară cunoaşterea noţiunii de stivă. Pe scurt, stiva este utilă în situaţii în care este necesară memorarea unor informaţii şi regăsirea acestora într-o anumită

1. În traducere exactă, **PUSH** înseamnă „a împinge”. Termenul sugerează o imagine plastică: imaginându-ne stiva ca un sac, **PUSH** ar însemna că împingem înăuntru un element, prin capătul superior (nu prin mijloc sau pe la fundul sacului).
2. În traducere, **POP** înseamnă „a scoate cu zgomot (de exemplu dopul, etc), a descărca (pistolul, etc), a ieşi repede sau pe neaşteptate” (Levi̧chi, Leon; Bantaş, Andrei – *Dicţionar englez-român*). Imaginea este de asemeni sugestivă: **POP** este operaţia prin care primul element, cel de deasupra, „sare” din structură (sau din sac, ca să păstrăm analogia).
3. În traducere, **TOP** înseamnă „vârf”.

ordine, descrisă de principiul **LIFO**. Stiva este utilizată atunci când programul trebuie să amâne execuţia unor operaţii, pentru a le executa ulterior, în ordinea inversă a apariţiei lor. Operaţia curentă este cea corespunzătoare vârfului stivei, în stivă fiind reţinute toate informaţiile necesare programului pentru a executa operaţiile respective.

Cum implementăm o stivă?

Stiva este o structură de date abstractă, ce poate fi implementată în diferite moduri. De exemplu, putem implementa o stivă ca un vector în care reţinem elementele stivei. Pentru ca acest vector să funcţioneze ca o stivă, singurele operaţii permise sunt operaţiile caracteristice stivei.

Pentru exemplificare, să prezentăm declaraţiile necesare pentru implementarea unei stive cu elemente de tip `int`:

```
#define DimMax 100    //numarul maxim de elemente din stiva
typedef int Stiva[DimMax];
//tipul Stiva implementat ca vector
Stiva S;                    //stiva
int   vf;                   //varful stivei
```

Crearea unei stive vide

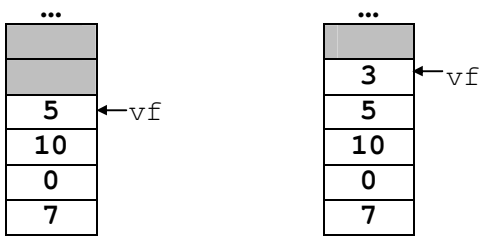
Pentru a crea o stivă vidă iniţializăm vârful stivei cu `-1` (vârful stivei indică întotdeauna poziţia ultimului element introdus în stivă; elementele sunt memorate în vector începând cu poziţia `0`):

```
vf=-1;
```

Inserarea unui element în stivă

Pentru a insera un element `x` în stiva `S` trebuie să verificăm în primul rând dacă „avem loc”, deci dacă stiva nu este plină. Dacă stiva este plină, inserarea nu se poate face, altfel vom mări vârful stivei şi vom plasa la vârf noul element.

De exemplu, dacă dorim să inserăm elementul `x = 3` în stiva din figura următoare, obţinem:

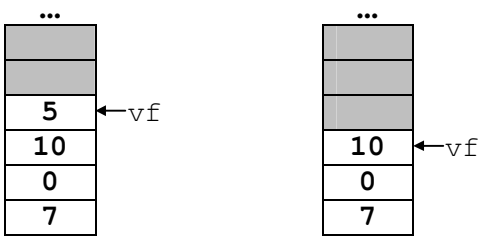


Stiva S înainte de inserare Stiva S după inserare

```
if (vf == DimMax-1)       //stiva este plina
    cout<<"Eroare - stiva este plina\n";
    else                  //inseram elementul x in stiva S
    S[++vf] = x;
```

Extragerea unui element din stivă

Pentru a extrage un element dintr-o stivă S trebuie să verificăm în primul rând dacă există elemente în stivă (deci dacă stiva nu este vidă). Dacă da, reținem elementul de la vârful stivei într-o variabilă (să o notăm x), după care micşorăm cu o unitate vârful stivei. De exemplu, dacă extragem un element din stiva din figura următoare, obținem:



Stiva S înainte de extragere Stiva S după extragere

```
if (vf<0)                    //stiva este vida
   cout<<"Eroare - stiva este vida\n";
   else                     //extragem elementul de la varf
   x = S[vf--];
```

Accesarea elementului de la vârf

Prin modul său restrictiv de funcționare, stiva permite numai accesarea elementului de la vârf. Dacă dorim să aflăm valoarea unui alt element al stivei, ar trebui să „golim" stiva (deci să extragem succesiv elemente) până la elementul dorit. Accesarea elementului de la vârf presupune determinarea valorii acestuia, valoare pe care noi o vom reține într-o variabilă denumită x.

```
x = S[vf];
```

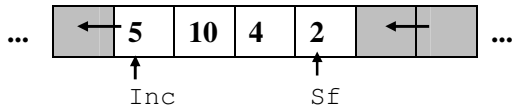
Observații

1. Dezavantajul implementării unei stive ca vector alocat static constă în faptul că indiferent de numărul de elemente existente în stivă, dimensiunea zonei de memorie alocată stivei este aceeaşi (DimMax).
2. Pentru a executa operații cu stiva alocată static este suficient să cunoaştem vârful stivei. Ca să reținem mai uşor modul de funcționare a stivei, ne imaginăm că la inserare vârful stivei urcă, iar la extragere vârful coboară.

5.2. Coada

Coada este o structură de date abstractă, pentru care operaţia de inserare a unui element se realizează la un capăt, în timp ce 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.



Operaţii caracteristice

Singurele operaţii ce pot fi executate cu o coadă sunt:
- crearea unei cozi vide;
- inserarea unui element în coadă;
- extragerea unui element din coadă;
- accesarea unui element.

Executarea acestor operaţii asupra unei cozi presupune cunoaşterea începutului cozii (să-l notăm `Inc`) şi a sfârşitului acesteia (să-l notăm `Sf`).

Modul de funcţionare a unei cozi este foarte uşor de intuit: toată lumea a „stat la coadă", măcar o dată. Orice situaţie în care sunt mai multe cereri de acces la o resursă unică (de exemplu, mai mulţi clienţi şi o singură vânzătoare; o singură pompă de benzină şi mai multe maşini, un singur pod şi mai multe capre, etc) necesită formarea unei „linii de aşteptare". Dacă nu apar alte priorităţi, cererile sunt satisfăcute în ordinea sosirii.

Datorită faptului că întotdeauna este extras („servit") primul element din coadă, iar inserarea oricărui nou element se face la sfârşit („la coadă"), coada este definită ca o structură de date care funcţionează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut – Primul Intrat Primul Ieşit).

Care este utilitatea unei cozi?

Utilitatea structurii de tip coadă reiese din modul său de funcţionare – este necesară utilizarea unei cozi atunci când informaţiile trebuie prelucrate exact în ordinea în care „au sosit" şi ele sunt reţinute în coadă până când pot fi prelucrate. În informatică, cozile sunt utilizate frecvent. De exemplu, să considerăm că avem o reţea de calculatoare şi o singură imprimantă. Când utilizatorii reţelei vor da comenzi de tipărire, imprimanta nu poate răspunde tuturor comenzilor în acelaşi timp (imaginaţi-vă ce-ar ieşi!). Prin urmare comenzile de tipărire primite sunt înregistrate într-o coadă (*Print Queue* – Coadă de Tipărire). Imprimanta va tipări documentele pe rând, în ordinea în care au fost înregistrate în coadă. Un alt exemplu: pentru a mări viteza de execuţie, microprocesoarele Intel utilizează o coadă de instrucţiuni în care sunt memorate instrucţiunile care urmează a fi executate.

Cum implementăm o coadă?

Coada este o structură de date abstractă, care poate fi implementată în diferite moduri. Ca şi în cazul stivei, coada poate fi implementată static, reținând elementele sale într-un vector.

Să considerăm următoarele declarații care reprezintă o coadă cu elemente de tip `int` alocată static:

```
#define DimMax 100    //numarul maxim de elemente din coada
typedef int Coada[DimMax];
//tipul Coada implementat ca vector
Coada C;                      //coada
int  Inc, Sf;                 //inceputul si sfarsitul cozii
```

Elementele cozii sunt memorate în vector de la poziția `Inc` până la poziția `Sf`, deci numărul lor este `Sf-Inc+1`.

Crearea unei cozi vide

Pentru a crea o coadă vidă trebuie să inițializăm variabilele `Inc` şi `Sf` astfel:

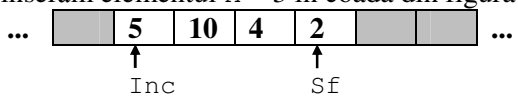
```
Inc = 0;
Sf = -1;
```

Observați că numărul de elemente din coadă este 0, iar poziția pe care va fi plasat primul element din coadă este 0.

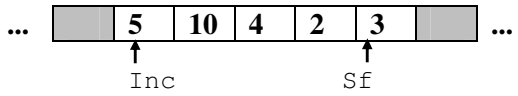
Inserarea unui element în coadă

Pentru a insera un element `x` în coada `C` trebuie să verificăm în primul rând dacă „avem loc”, deci dacă variabila `Sf` nu depăşeşte dimensiunea maximă a vectorului. Dacă inserarea se poate face, vom mări valoarea variabilei `Sf` cu o unitate (coada „creşte”) şi vom plasa la sfârşit noul element.

De exemplu, să inserăm elementul `x` = 3 în coada din figura următoare:



Coada `C` înainte de inserare



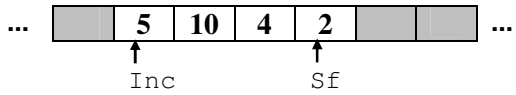
Coada `C` după inserare

```
if (Sf == DimMax-1)          //nu mai avem loc
   cout<<"Eroare\n";
   else                     //inseram elementul x in coada C
   C[++Sf] = x;
```

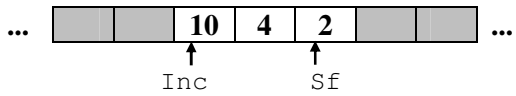
Extragerea unui element din coadă

Pentru a extrage un element dintr-o coadă `C` trebuie să verificăm în primul rând dacă numărul de elemente din coadă este diferit de `0` (coada nu este vidă). Dacă da, reținem elementul de la începutul cozii într-o variabilă (să o notăm `x`), după care mărim cu o unitate începutul cozii.

De exemplu, să extragem un element din coada din figura următoare:



Coada `C` înainte de extragere



Coada `C` după extragere

```
if (Inc > Sf)             //coada este vida
   cout<<"Eroare \n";
   else                              //extragem primul element
   x = C[Inc++];
```

Accesarea unui element

Singurul element al unei cozi care poate fi accesat direct este primul. Dacă dorim să aflăm valoarea unui alt element, va trebui să extragem succesiv elemente până la cel dorit.

Accesarea primului element are ca scop determinarea valorii acestuia, valoare pe care o vom reține într-o variabilă denumită x.

```
x = C[Inc];
```

Observație

În acest mod de alocare statică a unei cozi observați că pe măsură ce inserăm şi extragem elemente, atât sfârşitul, cât şi începutul cozii „migrează" către limita maximă a vectorului. Dezavantajul este că în vector rămân poziții neutilizate (de la `0` până la `Inc-1`). De exemplu, ar putea să apară o situație în care coada conține un element, dar operația de inserare să nu fie posibilă pentru că `Inc=Sf=DimMax-1`. O soluție mai bună ar fi de a reutiliza circular spațiul de memorie alocat cozii. Pentru aceasta, următoarea poziție după poziția `i` poate fi:

- `i+1`, dacă `i<DimMax -1`
- `0`, dacă `i=DimMax-1`

Acest lucru se poate scrie concis: `(i+1)%DimMax`.

Caroiaj

Se consideră un caroiaj dreptunghiular cu `m` linii şi `n` coloane, în care pe anumite poziţii sunt plasate obstacole. În poziţia iniţială (x_0, y_0) se află plasat un mobil. Să se determine, pentru toate poziţiile în care mobilul poate ajunge, distanţa minimă de la poziţia iniţială a mobilului, măsurată în deplasări elementare. Prin deplasare elementară se înţelege deplasarea mobilului cu o poziţie stânga, dreapta, sus sau jos.

Labirint

Se dă un labirint dreptunghiular de dimensiuni `nxm` ($n, m \leq 100$). Poziţiile din stânga sus şi dreapta jos sunt marcate cu `0`, celelalte conţin unul dintre numerele `1`, `2`, `3`, `4`. Scopul este de a parcurge labirintul din colţul stânga–sus până la colţul din dreapta–jos pe un drum de lungime minimă, pe direcţii paralele cu laturile sale. Drumul urmat trebuie să plece din `0` în `1`, apoi din `1` în `2`, din `2` în `3` din `3` în `4`, din `4` în `1` etc. Se poate ajunge în poziţia finală din oricare poziţie vecină ei.

Din fişierul de intrare `INPUT.TXT` se vor citi de pe prima linie numerele întregi `n` şi `m`, care reprezintă dimensiunile labirintului, iar de pe următoarele `n` linii câte `m` numere întregi, separate prin spaţiu, reprezentând labirintul. În fişierul de ieşire `OUTPUT.TXT` se va afişa pe prima linie numărul minim de paşi, iar pe cea de a doua linie un şir de caractere, care reprezintă succesiunea de mişcări de pe cel mai scurt drum din labirint, folosind codificarea: `D` (jos), `U` (sus), `L` (stânga) respectiv `R` (dreapta).

Exemplu

```
INPUT.TXT                OUTPUT.TXT
5 4                      7
0 1 2 3                  RRRDDDD
3 2 1 4
4 1 2 1
1 4 3 2
2 3 4 0
```

(Balcaniada de Informatică, Cipru, 1996)

Romeo şi Julieta:
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=898

Alee
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=831

Paianjen
http://campion.edu.ro/arhiva/index.php?page=problem&action=view&id=682

Bibliografie

E. Cerchez, M. Şerban – „Programarea în limbajul C/C++”. Volumul I; Editura Polirom

Prof. E. Cerchez; prof. M. Şerban