TAD Stiva (STACK)

Observaţii:

1. În limbajul uzual cuvântul "stivă" referă o "grămadă în care elementele constitutive sunt aşezate ordonat unele peste altele".

   - Un element nou se adaugă în stivă deasupra elementului cel mai recent adăugat în stivă.
   - Din stivă se poate accesa şi extrage doar elementul cel mai recent introdus.
   - Exemple de stive sunt multiple: stivă de farfurii, stivă de lemne, etc. Tipul de date **Stivă** permite implementarea în aplicaţii a acestor situaţii din lumea reală.

2. O *stivă* este o structură liniară de tip listă care restricţionează adăugările şi ştergerile la un singur capăt (listă LIFO - *Last In First Out*).

3. **Accesul** într-o stivă este *prespecificat* (se poate accesa doar elementul cel mai recent intodus în stivă - din vârful stivei), nu se permite accesul la elemente pe baza poziţiei. Dintr-o stivă se poate **şterge** elementul CEL MAI RECENT introdus în stivă - cel din vârful stivei.

4. Se poate considera şi o capacitate iniţială a stivei (număr maxim de elemente pe care le poate include), caz în care dacă numărul efectiv de elemente atinge capacitatea maximă, spunem că avem o *stivă plină*.

   - adăugarea în stiva plină se numeşte **depăşire superioară**.

5. O stivă fără elemente o vom numi *stivă vidă* şi o notăm Φ.

   - ştergerea din stiva vidă se numeşte **depăşire inferioară**.

6. O stivă în general nu se iterează.

7. Stivele sunt frecvent utilizate în programare - recursivitate, backtracking iterativ.

Tipul Abstract de Date STIVA:

domeniu:

$$\mathcal{S} = \{s \mid s \text{ este o stivă cu elemente de tip } TElement\}$$

operaţii (interfaţa):

- creeaza(s)
  {creează o stivă vidă}
  $pre:$ $true$
  $post:$ $s \in \mathcal{S}, s = \Phi$(stiva vidă)

- adauga(s, e)
  {se adaugă un element în vârful stivei}
  $pre:$ $s \in \mathcal{S}, e \in TElement, s$ nu e plină
  $post:$ $s' \in \mathcal{S}, s' = s \oplus e,$ e va fi cel mai recent element introdus în stivă

  @ aruncă excepţie dacă stiva e plină

- sterge(s, e)
  {se scoate un element din vârful stivei}
  $pre:$ $s \in \mathcal{S}, s \neq \Phi$
  $post:$ $e \in TElement,$ e este cel mai recent element introdus în stivă, $s' \in \mathcal{S}, s' = s \ominus e$

  @ aruncă excepţie dacă stiva e vidă

- element(s, e)
  {se accesează elementul din vârful stivei}
  $pre:$ $s \in \mathcal{S}, s \neq \Phi$
  $post:$ $s' = s, e \in TElement,$ e este cel mai recent element introdus în stivă

  @ aruncă excepţie dacă stiva e vidă

- vida (s)
  $pre:$ $s \in \mathcal{S}$
  $post:$ $vida = \begin{cases} adev, & \text{dacă } s = \Phi \\ fals, & \text{dacă } s \neq \Phi \end{cases}$

- plina (s)
  $pre:$ $s \in \mathcal{S}$
  $post:$ $plina = \begin{cases} adev, & \text{dacă } s \text{ e plină} \\ fals, & \text{contrar} \end{cases}$

- distruge(s)
  {destructor}
  $pre:$ $s \in \mathcal{S}$
  $post:$ s a fost 'distrusa' (spaţiul de memorie alocat a fost eliberat)

Observaţii

- Stiva nu este potrivită pentru aplicaţiile care necesită traversarea ei (nu avem acces direct la elementele din interiorul stivei).
- Afişarea conţinutului stivei poate fi realizată folosind o stivă auxiliară (scoatem valorile din stivă punându-le pe o stivă auxiliară, după care se repun pe stiva iniţială).

Implementări ale stivelor folosind

- tablouri - vectori (dinamici)
- liste înlănţuite.

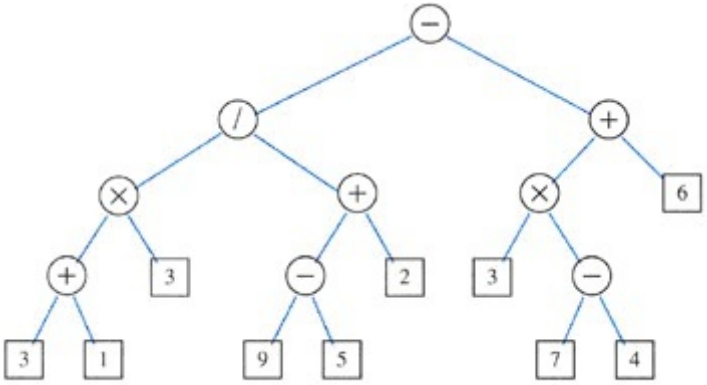

Figura 1: Arbore corespunzător expresiei ((((3+1)×3)/((9−5)+2))−((3×(7−4))+6)).

Exemplu

- O expresie aritmetică se poate reprezenta printr-un arbore binar ale cărui noduri terminale sunt asociate cu variabile sau constante şi ale cărui noduri interne sunt asociate cu unul dintre operatorii: **+**, **-**, **×**, şi **/**. (Vezi Figura 1.)
- Fiecare nod dintr-un asemenea arbore are o valoare asociată:
  - Dacă nodul este terminal valoarea sa este cea a variabilei sau constantei asociate.
  - Dacă nodul este neterminal valoarea sa este definită prin aplicarea operaţiei asociate asupra fiilor lui.

Aplicaţie

Evaluarea unei expresii aritmetice din forma postfixată. Fie $EPost$ o expresie aritmetică CORECTĂ în forma postfixată, conţinând operatorii binari: +, -, *, /, iar ca operanzi cifre 0-9 (ex: $EPost = 1\ 2\ +\ 3\ *\ 4\ /$). Se cere să se determine valoarea expresiei (ex: valoarea este 2.25).

Indicaţie: Se va folosi o stivă în care se se vor adăuga operanzii. În final, stiva va conţine valoarea expresiei.

Vom putea folosi următorul algoritm.

- Pentru fiecare $e \in EPost$ (în ordine de la stânga la dreapta)
  1. Dacă e este operand, atunci se adaugă în stivă.
  2. Dacă e este operator, atunci se scot din stivă doi operanzi (op_1 şi op_2), se efectuează operaţia e înte cei doi operanzi ($v = op_2\ e\ op_1$), după care se adaugă v în stivă. **Obs.** Aici s-ar putea depista dacă expresia nu ar fi corectă în forma postfixată (s-ar încerca extragerea unui element dintr-o stivă vidă).
- În presupunerea noastră că expresia în forma postfixată este validă, în final, stiva va conţine o singură valoare, care se va extrage din stivă. Această valoare reprezintă valoarea expresiei.

Temă

Se dă un text care conţine caractere incluzând paranteze rotunde, paranteze drepte şi acolade. Se cere să se verifice dacă în text parantezele se închid corect. De exemplu în textul a= (2 + b[3])*5; parantezele se închid corect; în textul a = (b[0) . 1]; parantezele nu se închid corect. **În aplicaţie, se vor folosi operaţiile din interfaţa TAD Stivă.**

TAD Coada (QUEUE)

Observaţii:

1. În limbajul uzual cuvântul "coadă" se referă la o înşiruire de oameni, maşini, etc., aranjaţi în ordinea sosirii şi care aşteaptă un eveniment sau serviciu.

   - Noii sosiţi se poziţionează la sfârşitul cozii.
   - Pe măsură ce se face servirea, oamenii se mută câte o poziţie înainte, până când ajung în faţă şi sunt serviţi, asigurându-se astfel respectarea principiului "primul venit, primul servit".
   - Exemple de cozi sunt multiple: coada de la benzinării, coada pentru cumpărarea unui produs, etc. Tipul de date **Coadă** permite implementarea în aplicaţii a acestor situaţii din lumea reală.

2. O *coadă* este o structură liniară de tip listă care restricţionează adăugările la un capăt şi ştergerile la celălalt capăt (lista FIFO - *First In First Out*).

3. **Accesul** într-o coadă este *prespecificat* (se poate accesa doar elementul cel mai devreme introdus în coadă), nu se permite accesul la elemente pe baza poziţiei. Dintr-o coadă se poate **şterge** elementul CEL MAI DEVREME introdus (primul).

4. Se poate considera şi o capacitate iniţială a cozii (număr maxim de elemente pe care le poate include), caz în care dacă numărul efectiv de elemente atinge capacitatea maximă, spunem că avem o *coadă plină*.

   - adăugarea în coada plină se numeşte **depăşire superioară**.

5. O coadă fără elemente o vom numi *coadă vidă* şi o notăm Φ.

   - ştergerea din coada vidă se numeşte **depăşire inferioară**.

6. O coadă în general nu se iterează.

7. Cozile sunt frecvent utilizate în programare - crearea unei cozi de aşteptare a task-urilor într-un sistem de operare.

   - dacă task-urile nu au asociată o prioritate, ele sunt procesate în ordinea în care intră în sistem $\rightarrow$ **Coadă**.
   - dacă task-urile au asociate o prioritate şi trebuie procesate în ordinea priorităţii lor $\rightarrow$ **Coadă cu priorităţi**.

Tipul Abstract de Date COADA:

domeniu: $\mathcal{C} = \{c \mid c \text{ este o coadă cu elemente de tip } TElement\}$

operaţii:

- creeaza(c)
  {creează o coadă vidă}
  $pre:$ $true$
  $post:$ $c \in \mathcal{C}, c = \Phi$(coada vidă)

- adauga(c, e)
  {se adaugă un element la sfârşitul cozii}
  $pre:$ $c \in \mathcal{C}, e \in TElement, c$ nu e plină
  $post:$ $c' \in \mathcal{C}, c' = c \oplus e,$ e va fi cel mai recent element introdus în coadă

  @ aruncă excepţie dacă coada e plină

- sterge(c, e)
  {se şterge primul element introdus în coadă}
  $pre:$ $c \in \mathcal{C}, c \neq \Phi$
  $post:$ $e \in TElement,$ e este elementul cel mai devreme introdus în coadă, $c' \in \mathcal{C}, c' = c \ominus e$

  @ aruncă excepţie dacă coada e vidă

- element(c, e)
  {se accesează primul element introdus în coadă}
  $pre:$ $c \in \mathcal{C}, c \neq \Phi$
  $post:$ $c' = c, e \in TElement,$ e este elementul cel mai devreme introdus în coadă

  @ aruncă excepţie dacă coada e vidă

- vida (c)
  $pre:$ $c \in \mathcal{C}$
  $post:$ $vida = \begin{cases} adev, & \text{dacă } c = \Phi \\ fals, & \text{dacă } c \neq \Phi \end{cases}$

- plina (c)
  $pre:$ $c \in \mathcal{C}$
  $post:$ $plina = \begin{cases} adev, & \text{dacă } c \text{ e plină} \\ fals, & \text{contrar} \end{cases}$

- distruge(c)
  {destructor}
  $pre:$ $c \in \mathcal{C}$
  $post:$ c a fost 'distrusa' (spaţiul de memorie alocat a fost eliberat)

Observaţii

- Coada nu este potrivită pentru aplicaţiile care necesită traversarea ei (nu avem acces direct la elementele din interiorul cozii).
- Afişarea conţinutului cozii poate fi realizată folosind o coadă auxiliară (scoatem valorile din coadă punându-le într-o coadă auxiliară, după care se reintroduc în coada iniţială).

Implementări ale cozilor folosind

- tablouri - vectori (dinamici) - reprezentare circulară (Figura 2).
- liste înlănţuite.

1	2	3	4	5	6	7	8	9	10	11	12
						15	6	9	8	4	

Față **Spate**

n = 12 (capacitatea tabloului)
Față = 7 - indicele unde e memorat primul element din coadă
Spate = 12 – primul indice liber din spatele cozii

- Se adaugă în **Spate**, se șterge din **Față**
- Elementele cozii se află între indicii **Față, Față+1,...Spate-1**
- Initializarea cozii: **Față=Spate=1**
- Depășire inferioară (coada vidă): **Față=Spate**
- Depășire superioară (coada plină): a) **Față= 1** și **Spate= n** sau b) **Față=Spate+1**

a) **Față= 1** și **Spate= 12**

1	2	3	4	5	6	7	8	9	10	11	12
2	8	9	3	5	7	15	6	9	8	4	

a) **Față= 7** și **Spate= 6**

1	2	3	4	5	6	7	8	9	10	11	12
2	8	9	3	5		15	6	9	8	4	5

Generalizare a cozilor

- **Coada completă** (*Double ended queue* - **DEQUEUE**) - adăugări, ştergeri se pot face la ambele capete ale cozii.

Aplicaţie

Translatarea unei expresii aritmetice din forma infixată în forma postfixată. Fie E o expresie aritmetică CORECTĂ în forma infixată, fără paranteze, conţinând operatorii binari: +, -, *, /, iar ca operanzi cifre 0-9 (ex: $E = 1\ +\ 2\ *\ 3$). Se cere să se determine forma postfixată $EPost$ a expresiei (ex: $EPost = 1\ 2\ 3\ *\ +$).

Indicaţie: Se va folosi o stivă în care se se vor adăuga operatorii şi o coadă $EPost$ care va conţine în final forma postfixată a expresiei.

Vom putea folosi următorul algoritm.

- Pentru fiecare $e \in E$ (în ordine de la stânga la dreapta)
  1. Dacă e este operand, atunci se adaugă în coada $EPost$.
  2. Dacă e este operator, atunci se scot din stivă operatorii având prioritatea de evaluare mai mare sau egală decât a lui e şi se adaugă în coada $EPost$, după care se adaugă e în stivă.
- Se scot din stivă operatorii rămaşi şi se adaugă în coada $EPost$.
- În final, $EPost$ va conţine forma postfixată a expresiei.

Temă.

Trataţi cazul în care expresia în forma infixată conţine paranteze (ex: $E = (1 + 2) * 3$).

Indicaţie: Ideea/algoritmul de bază este acelaşi ca şi în cazul în care expresia nu ar conţine paranteze. Paranteza deschisă "("se va adăuga în stivă. Va trebui să identificaţi paşii care vor trebui efectuaţi la întâlnirea unei paranteze închise ")".