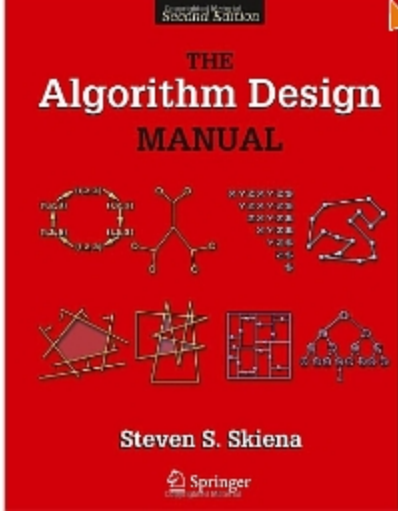
CURS 7:

Structuri liniare de date (I)

Motivație



S. Skiena – The Algorithm Design Manual

http://sist.sysu.edu.cn/~isslxm/DSA/textbook/Skiena.-.TheAlgorithmDesignManual.pdf

Stacks, Queues, and Lists

3-1. *[3]* A common problem for compilers and text editors is determining whether the parentheses in a string are balanced and properly nested. For example, the string (((())())()) contains properly nested pairs of parentheses, which the strings)()(and ()) do not. Give an algorithm that returns true if a string contains properly nested and balanced parentheses, and false if otherwise. For full credit, identify the position of the first offending parenthesis if the string is not properly nested and balanced.

De la mulțime la structură

- Set de date: $A=\{a_1,a_2,...,a_n\}$ (dacă elementele sunt distincte corespunde conceptului de mulțime din matematică)

- Structura de date: set de date + relații între elementele mulțimii

- Dpdv matematic:
  - O relație binară R este o submulțime a lui AxA; daca (a_i,a_j) aparține lui R spunem ca a_i se află în relația R cu a_j (se notează a_i R a_j)
  - O relație binară poate avea diferite proprietăți: reflexivitate (a_iRa_i), simetrie (a_iRa_j implică a_jRa_i), antisimetrie (a_iRa_j si a_jRa_i implica $a_i=a_j$), tranzitivitate (a_iRa_j, a_jRa_k implica a_iRa_k)
  - Cazuri particulare:
    - Relație de echivalență: reflexivă, simetrică și tranzitivă
    - Relație de ordine: reflexivă, antisimetrică și tranzitivă

De la mulțime la structură

- Set de date: $A=\{a_1,a_2,...,a_n\}$
- Exemple de relatii binare
  - „succesor” (RS): a_i RS a_j daca a_i se află înaintea lui a_j (a_i este succedat de a_j)
  - „predecesor” (RP): a_i RP a_j daca a_i se află după a_j (a_i este precedat de a_j)
  - „succesor direct” (RSD): a_i RSD a_j daca a_i RS a_j si nu exista a_k astfel incat a_i RS a_k si a_k RS a_j
  - „predecesor direct” (RPD): a_i RPD a_j daca a_i RP a_j si nu exista a_k astfel incat a_i RP a_k si a_k RP a_j

- Structura liniară = set de date + relație de tip succesor sau predecesor (sau ambele) cu proprietatea că fiecare element are un unic succesor direct (și/sau predecesor direct)
- Cazul cel mai simplu: extinderea relatiilor „predecesor”, „succesor” corespunzătoare mulțimii indicilor asupra datelor din set

De la mulțime la structură

- Exemplu: M=(3,1,4,2)
- Variante de reprezentare a elementelor secvenței M (în Python)
  - folosind 4 variabile: a=1, b=2, c=3, d=4
  - Folosind o singură variabilă: x=[3,1,4,2]
- In care dintre cele două variante putem considera că este definită o structură de date?
- Este vorba de o structură liniară?

Tipuri abstracte de date

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare

Exemple: lista (list), stiva (stack), coada (queue), movila (heap), dicționar (dictionary), tabel de dispersie (hash table), arbori de căutare (trees)

Reprezentare abstractă = modalitate de organizare a elementelor setului de date care permite efectuarea eficientă a operațiilor specifice

Nivel: proiectare

Exemple: structură liniară, structură arborescentă

Implementare = specificarea structurii de date folosind tipuri de date și construcții specifice unui limbaj de programare

Nivel: implementare

Exemple: tablouri (arrays), liste înlănțuite (linked lists)

Tipuri abstracte de date

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare

Exemple: lista (list), stiva (stack), coada (queue), movila (heap), dicționar (dictionary), tabel de dispersie (hash table), arbori de căutare (trees)

Reprezentare abstractă = modalitate de organizare a elementelor setului de date care permite efectuarea eficientă a operațiilor specifice

Nivel: proiectare

Exemple: structură liniară, structură arborescentă

Implementare = specificarea structurii de date folosind tipuri de date și construcții specifice unui limbaj de programare

Nivel: implementare

Exemple: tablouri (arrays), liste înlănțuite (linked lists)

Tip abstract de date: lista

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare Exemplu: lista

Operații specifice listelor: **interogare = căutarea unui element**

- După poziție
  - Elementul aflat pe o anumită poziție
  - Elementul următor/ anterior unui element specificat (element curent)
- După valoare
  - Elementul care conține o valoare specificată
  - Elementul care conține cea mai mică/ mare valoare
  - Elementul care conține o valoare specificată prin poziția relativă în raport cu alte valori (ex: al treilea element în ordine crescătoare, elementul median etc)

Caz particular: parcurgerea tuturor elementelor într-o anumită ordine

Tip abstract de date: lista

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare Exemplu: lista

Operații specifice listelor: **modificare**

- Inserarea unui element
  - La început
  - La sfârșit
  - Pe o poziție specificată relativ la alte elemente (înainte sau după un element dat)
- Eliminarea (ștergerea) unui element
  - De la început
  - De la sfârșit
  - De pe o poziție specificată relativ la alte elemente (înainte sau după un element dat)
- Rearanjare elemente = schimbarea ordinii elementelor după un anumit criteriu (ex: sortare)

Tip abstract de date: lista

Tip abstract de date = set de date asupra căruia se pot efectua operații specifice

Nivel: utilizare Exemplu: lista

Operații specifice listelor: **combinarea listelor**

- Concatenare = construirea unei liste prin adăugarea elementelor celei de a doua liste la sfârșitul primeia

- Interclasare = construirea unui liste cu elemente ordonate pornind de la două liste având elementele ordonate

Tip abstract de date: stiva

- Cum se poate extrage mingea pe care scrie „Wilson 1” din cutia alăturată?
- Accesul la mingi este limitat
- Corespunde unei structuri în care se poate accesa doar ultimul element introdus

Stiva = structură liniară pentru care operațiile de interogare și modificare se pot efectua doar la una dintre extremități (numită vârf).

Principiu: Last In First Out (LIFO)

Tip abstract de date: stiva

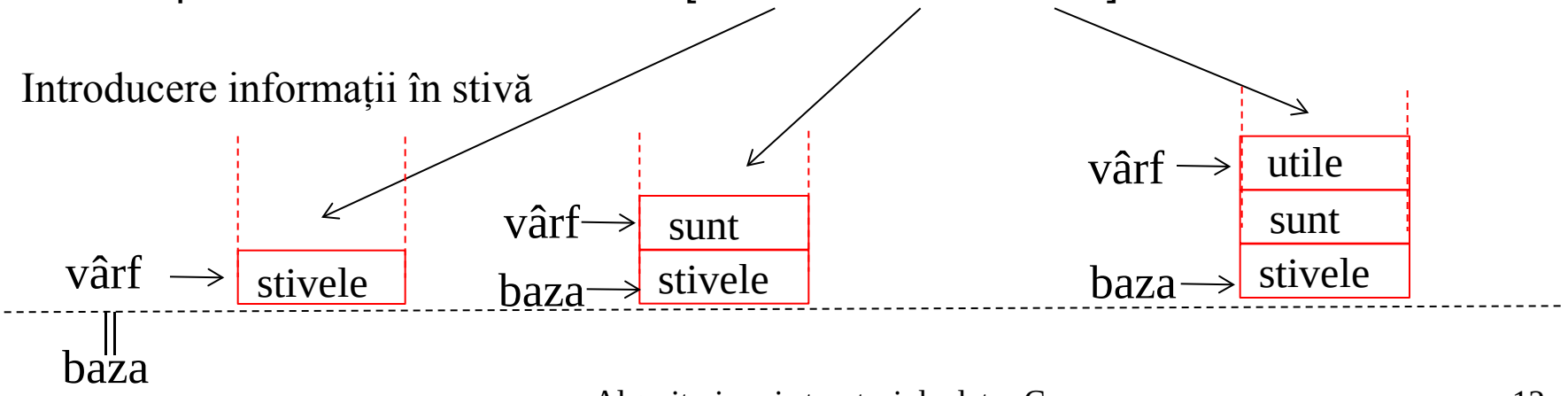
Stiva = structură liniară pentru care operațiile de interogare și modificare se pot efectua doar la una dintre extremități (numită vârf).

Principiu: Last In First Out (LIFO)

Operații de bază:

- Interogare (top): consultarea elementului din vârful stivei (fără extragerea lui)
- Modificare (pop): Extragerea elementului din vârful stivei
- Modificare (push): Introducerea unui element în stivă

Exemplu: „stivele sunt utile” -> [„stivele”, „sunt”, „utile”]



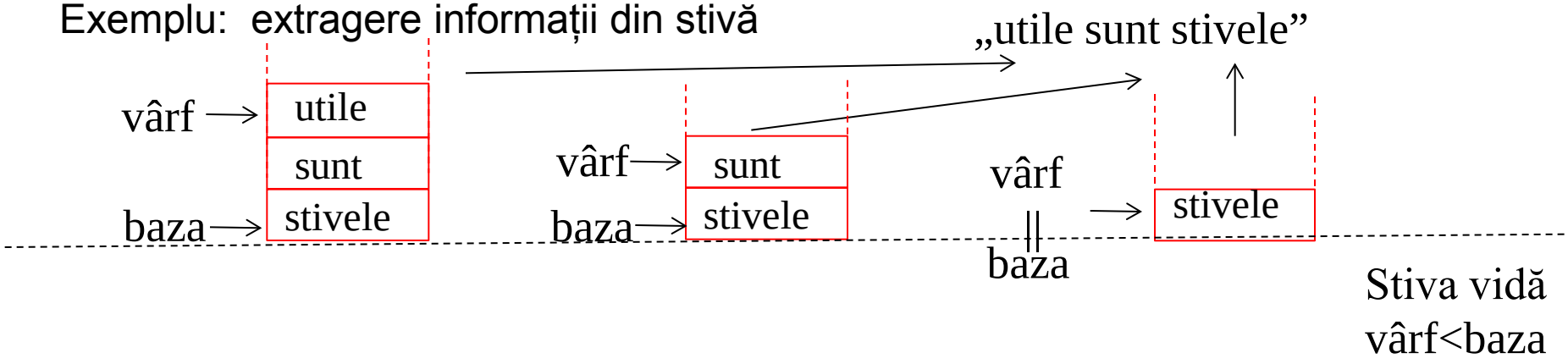
Tip abstract de date: stiva

Stiva = structură liniară pentru care operațiile de interogare și modificare se pot efectua doar la una dintre extremități (numită vârf).

Operații de bază:

- Interogare (top): consultarea elementului din vârful stivei (fără extragerea lui)
- Modificare (pop): Extragerea elementului din vârful stivei
- Modificare (push): Introducerea unui element în stivă

Exemplu: extragere informații din stivă



Tip abstract de date: stiva

Aplicații:

1. Este corectă (dpdv al plasării parantezelor) descrierea expresiei {a+[b-c*(a+2)]}/(b+c)? Dar în cazul expresiei {a+[b-c*(a+2])}/(b+c) ?

2. Este corectă sintactic secvența HTML?

```
<h3> <a name="course"></a>Course materials (in Romanian):</h3>
<p><b><font color="#cc0000">Curs 1-2 </font></b> Introducere in
rezolvarea algoritmica a problemelor.
(<a href="algoritmica_cap1.pdf">material curs</a>,
 <a href="ASD_introducere.pdf">introducere</a>,
 <a href="alg2016_folii1.pdf">prezentare curs 1</a>,
 <a href="alg2016_folii2.pdf">prezentare curs 2</a>)
```

Cum ar putea fi utilizate stivele într-un astfel de context?

Tip abstract de date: stiva

Aplicații:

1. Este corectă (dpdv al plasării parantezelor) descrierea expresiei {a+[b-c*(a+2)]}/(b+c)? Dar în cazul expresiei {a+[b-c*(a+2])}/(b+c) ?

Idee:

- ne interesează doar parantezele „{a+[b-c*(a+2)]}/(b+c)” -> {[()]}()
- Se parcurge secvența de paranteze:
  - La întâlnirea unei paranteze deschise aceasta se introduce in stivă
  - La întâlnirea unei paranteze închise se extrage și se analizează paranteza din vârful stivei: dacă paranteza din vârful stivei corespunde cu cea curentă („{” cu „}”, „[” cu „]”, „(” cu „)”) se continuă analiza; dacă nu corespund atunci s-a detectat o eroare și se abandonează parcurgerea.
  - Dacă se ajunge la o stivă vidă atunci expresia e corectă; dacă la epuizarea parantezelor din secvență stiva nu este vidă atunci expresia nu e corectă.

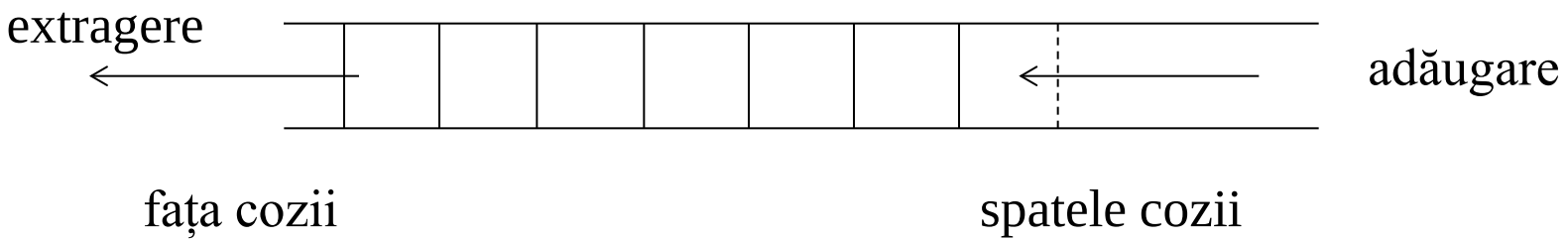
Tip abstract de date: coada

Coada (queue) = structură liniară pentru care operația de interogare/extragere se poate face la una dintre extremități (fața sau începutul cozii) iar adăugarea se poate face doar la cealaltă extremitate (spatele sau sfârșitul cozii).

Principiu: First In First Out (FIFO)

Operații de bază:

- Extragere: extragerea elementului din fața cozii
- Adăugare: adăugarea unui element la sfârșitul cozii

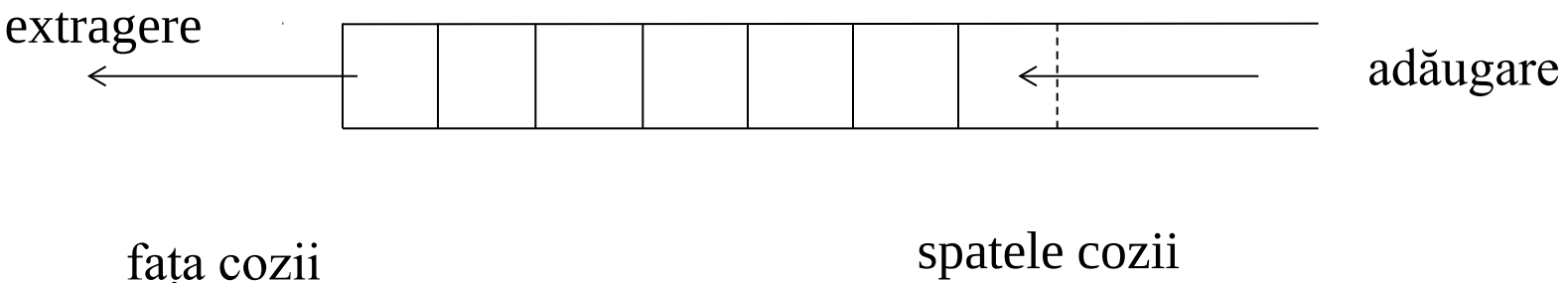


Tip abstract de date: coada

Exemplu: coada de servire

Caz simplificat: la fiecare moment de timp are loc un singur eveniment (fie sosește un nou client fie este servit unul dintre clienții aflați deja la coadă):

- T1: sosire C1 [C1]
- T2: sosire C2 [C1,C2]
- T3: servire C1 [C2]
- T4: sosire C3 [C2,C3]
- T5: sosire C4 [C2,C3,C4]
- T6: servire C2 [C3,C4]

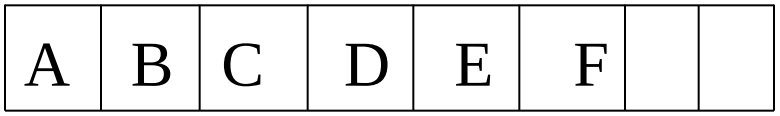


Implementarea structurii liniare

Variante de implementare:

- Folosind o zonă contiguă de memorie -> tablouri
- Folosind zone disparate „înlănțuite” prin specificarea unor informații de tip adresă (referință, pointer) -> liste înlănțuite

Tablou

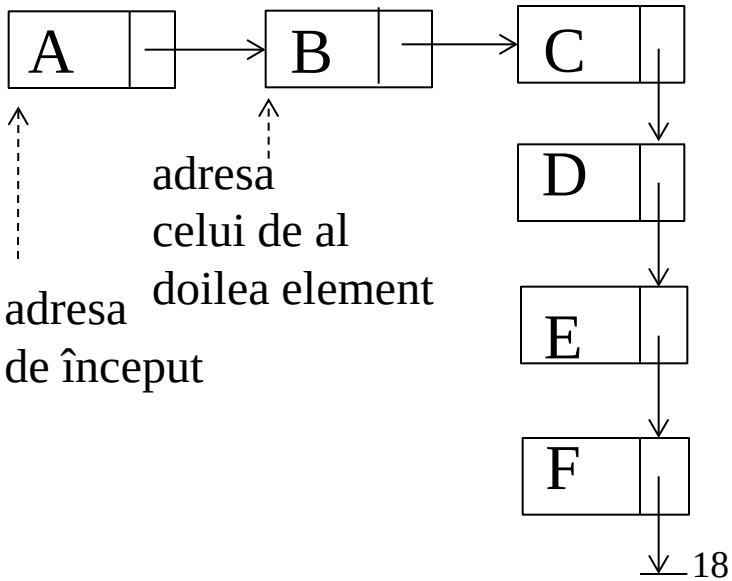


adresa unui element =
adresa de început + deplasament

adresa
de început

Obs:
deplasament = 0 pt primul element
= 1 pt al doilea
etc.

Structură înlănțuită



Implementarea structurii liniare

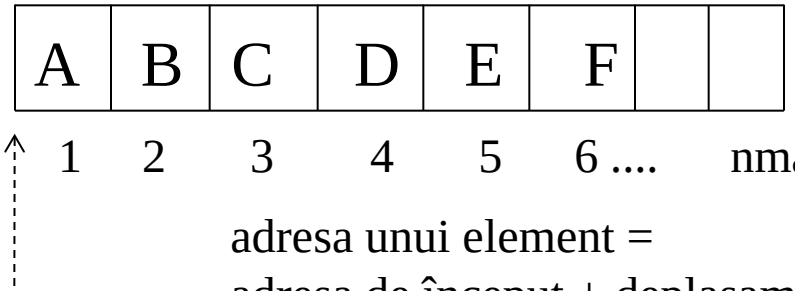
Tablou

Avantaje:

- Acces rapid pe baza indexului
- Se stochează doar valorile elementelor

Dezavantaje:

- Dimensiunea maximă este prestabilită



adresa unui element =
adresa de început + deplasament
(corelat cu valoarea indexului)
Obs:
deplasament = 0 pt primul element
= 1 pt al doilea
etc.

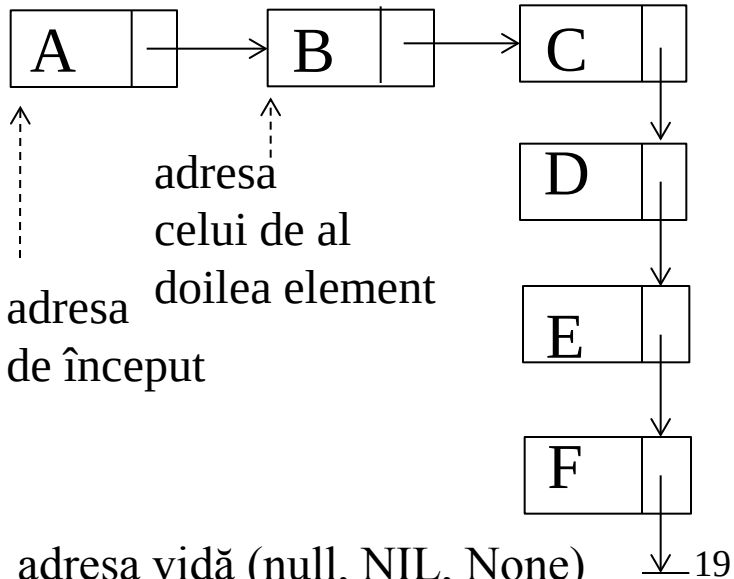
Structură (listă) înlănțuită

Avantaje:

- Dimensiune flexibilă
- Cost mic la inserare/ eliminare

Dezavantaje:

- Necesită stocarea unor informații adiționale (ex: adresa elem. următor)
- Accesul aleator nu este facil



Implementare utilizând tablouri

Tablou: x[1..nmax] - zona maximă alocată pentru stocarea structurii

n – numărul efectiv de elemente

Structura: s are două componente (câmpuri):

s.x (tabloul ce conține valorile)

s.n (număr efectiv de elemente)

Complexitatea operațiilor:

- Interogare după poziție
  - Elementul aflat pe o anumită poziție:

    s.x[i] cost:$\Theta(1)$
  - Elementul următor/ anterior unui element specificat (element curent):

    s.x[i-1] sau s.x[i+1] cost: $\Theta(1)$

Implementare utilizând tablouri

Tablou: x[1..nmax] - zona maximă alocată pentru stocarea structurii

n – numărul efectiv de elemente

Structura: s are două componente (câmpuri):

s.x (tabloul ce conține valorile)

s.n (numărul elementelor)

Complexitatea operațiilor: Interogare după valoare

- Elementul care conține o valoare specificată

  căutare secvențială cost:O(n)

- Elementul care conține cea mai mică/ mare valoare

  determinare minim/maxim cost:$\Theta(n)$

- Elementul care conține o valoare specificată prin poziția relativă în raport cu alte valori (ex: al treilea element în ordine crescătoare, elementul median etc)

  selecția celui de al k-lea element în ordine crescătoare/ descrescătoare:

  - varianta bazată pe sortare parțială prin metoda selecției: $\Theta(kn)$
  - varianta bazată pe ideea de la quicksort : O(n) în medie

Implementare utilizând tablouri

Tablou: x[1..nmax] - zona maximă alocată pentru stocarea structurii

n – numărul efectiv de elemente

Structura: s are două componente (câmpuri):

s.x (tabloul ce conține valorile)

s.n (numărul elementelor)

Inserarea unui element

- La început (cost: $\Theta(n)$)
  - Necesită deplasarea tuturor elementelor, începând de la ultimul, cu o poziție la dreapta
- La sfârșit (cost:$\Theta(1)$)
  - s.n=s.n+1; s.x[s.n]=e
- Pe poziția i (cost:O(n))
  - Necesită deplasarea elementelor cu indicii n, n-1,..., i cu o poziție la dreapta (obs: similar prelucrării folosite la sortarea prin inserție)

Implementare utilizând tablouri

Tablou: x[1..nmax] - zona maximă alocată pentru stocarea structurii

n – numărul efectiv de elemente

Structura: s are două componente (câmpuri):

s.x (tabloul ce conține valorile)

s.n (numărul elementelor)

Eliminarea (ștergerea, suprimarea) unui element

- De la început (cost:$\Theta(n)$)
  - Deplasarea tuturor elementelor, începând cu primul, cu o poziție la stânga
- De la sfârșit (cost:$\Theta(1)$)
  - s.n=s.n-1
- De pe poziția i (cost:$O(n)$)
  - Deplasarea tuturor elementelor având indicii (i+1), (i+2),..., n cu o poziție la stânga

Implementare stive

Stiva: x[1..nmax] - zona maximă alocată pentru stocarea structurii
v – indicele elementului din vârful stivei

Structura: s are două componente (câmpuri):
s.x (tabloul ce conține valorile)
s.v (indicele elementului din vârful stivei)

Inițializarea stivei: s.v ← 0 (inițial stiva este vidă)

Consultarea stivei (fără extragere element): s.x[s.v]

Adăugarea unui element la stivă:

```
push(s,element)
    if s.v<nmax then
        s.v ←s.v+1
        s.x[s.v] ← element
        return s
      else „stiva plina”
   endif
```

Extragerea unui element din stivă:

```
pop(s)
    if s.v!=0 then
        s.v ←s.v-1
        return s.x[s.v+1]
     else „stiva vida”
    endif
```

cost:$\Theta(1)$

Implementare stive

Exemplu de implementare în Python

```
class stiva:
    def __init__(self,nmax):
        self.data=[None]*nmax
         self.v=-1
        self.nmax=nmax
    def isEmpty(self):
        return self.v==-1
    def push(self,e):
        if self.v<self.nmax-1:
          self.v=self.v+1
          self.data[self.v]=e
    def top(self):
        if self.isEmpty()!=True:
            return self.data[self.v]
    def pop(self):
        if self.isEmpty()!=True:
           elem=self.data[self.v]
           self.v=self.v-1
           return elem
```

Obs:

- Clasele reprezintă în Python (si in alte limbaje de programare orientate obiect) suportul pentru implementarea tipurilor abstracte de date: conțin date și metode corespunzătoare operațiilor asupra datelor
- Obiectele sunt instanțe ale claselor
- Parametrul self se referă la instanța clasei asupra căreia se aplică prelucrarea (cea pentru care este invocată metoda)
- Parametrul self este special intrucât se specifică doar la definirea metodei (funcției) nu și la invocarea (apelul) ei

Implementare stive

Exemplu de implementare în Python

```
class stiva:
    def __init__(self,nmax):
        self.data=[None]*nmax
         self.v=-1
        self.nmax=nmax
    def isEmpty(self):
        return self.v==-1
    def push(self,e):
        if self.v<self.nmax-1:
          self.v=self.v+1
          self.data[self.v]=e
    def top(self):
        if self.isEmpty()!=True:
            return self.data[self.v]
    def pop(self):
        if self.isEmpty()!=True:
           elem=self.data[self.v]
           self.v=self.v-1
           return elem
```

Varianta ce folosește clasa standard pt liste

```
class stiva:
    def __init__(self):
        self.data=[]
    def __len__(self):
        return len(self.data)
    def isEmpty(self):
        return len(self.data)==0
    def push(self,e):
        self.data.append(e)
    def top(self):
        if self.isEmpty()!=True:
          return self.data[-1]
    def pop(self):
        if self.isEmpty()!=True:
          return self.data.pop()
```

Implementare stive

Exemplu de implementare în Python

```
def validare(text):
    s=stiva(100)          # crearea unui obiect de tip stiva
#   s=stiva()             # pt varianta a doua de implementare
    for par in text:
        if (par=="(") or (par=="[")or (par=="{"):
            s.push(par)  # includerea simbolului in stiva
        elif (par==")"or (par=="]")or (par=="}")):
            e=s.pop()    # extragerea simbolului din stiva
            if ((par==")") and (e!="("))or((par=="]") and
                 (e!="["))or((par=="}") and (e!="{")):
                return False  # s-a detectat o nepotrivire
    return s.isEmpty()   # daca secventa e corecta stiva e vida

text=["{","[","(","(","(",")","]","}"]
print(validare(text))
```

Implementare cozi

Coada: x[1..nmax] - zona maximă alocată pentru stocarea structurii

f – indicele elementului din fața cozii

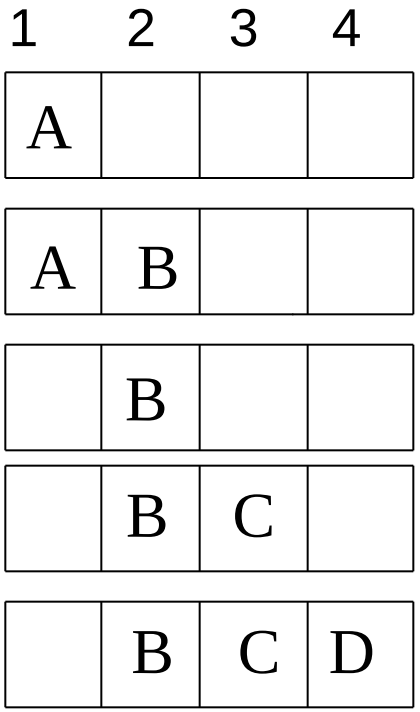
s – indicele elementului din spatele cozii

Structura: c are trei componente (câmpuri):

c.x (tabloul ce conține valorile), c.f, c.s

Coada vidă: c.s==0

Coada plină: c.s-c.f==nmax-1



Initializare: c.f=1 c.s=0 (nmax=4)

Adăugare „A”: c.f=1, c.s=1

Adăugare „B”: c.f=1, c.s=2

Extragere „A”: c.f=2, c.s=2

Adăugare „C”: c.f=2, c.s=3

Adăugare „D”: c.f=2, c.s=4

Extragere:

```
extrage(c)
  if c.f<=c.s then
     elem ←c.x[c.f]
     c.f ←c.f+1
     return elem
  else
     „coada vida”
  endif
```

Cost prelucrari: Θ(1)

Implementare cozi

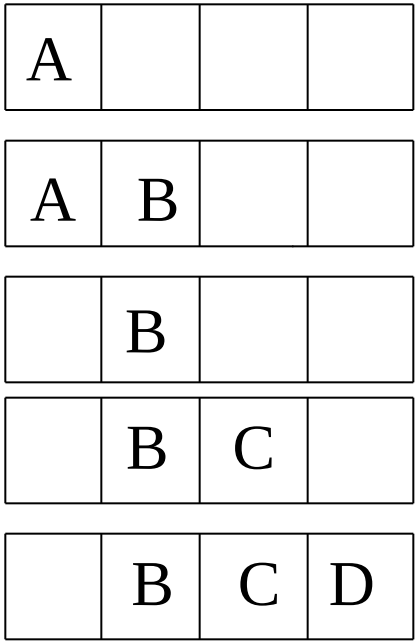
Coada: x[1..nmax] - zona maximă alocată pentru stocarea structurii

f – indicele elementului din fața cozii

s – indicele elementului din spatele cozii

Structura: c are trei componente (câmpuri):

c.x (tabloul ce conține valorile), c.f, c.s



Initializare: c.f=1 c.s=0 (nmax=4)

Adăugare „A”: c.f=1, c.s=1

Adăugare „B”: c.f=1, c.s=2

Extragere „A”: c.f=2, c.s=2

Adăugare „C”: c.f=2, c.s=3

Adăugare „D”: c.f=2, c.s=4

Coada vidă: c.s==0

Coada plină: c.s==nmax

Adăugare :

```
adauga(c,elem)
 if c.s<nmax then
     c.s ←c.s+1
     c.x[c.s] ←elem
else
     „coada plina”
endif
```

Cost prelucrari: Θ(1)

Problema: adăugarea e blocată chiar dacă există locuri disponibile în față

Implementare cozi

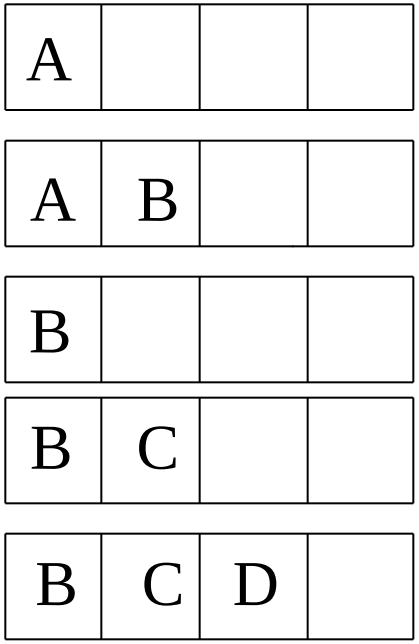
Coada: x[1..nmax] - zona maximă alocată pentru stocarea structurii

f – indicele elementului din fața cozii

s – indicele elementului din spatele cozii

Structura: c are trei componente (câmpuri):

c.x (tabloul ce conține valorile), c.f, c.s



Initializare: c.f=1 c.s=0 (nmax=4)

Adăugare „A”: c.f=1, c.s=1

Adăugare „B”: c.f=1, c.s=2

Extragere „A”: c.f=1, c.s=2

Adăugare „C”: c.f=1, c.s=3

Adăugare „D”: c.f=1, c.s=4

Coada vidă: c.s==0

Coada plină: c.s==nmax

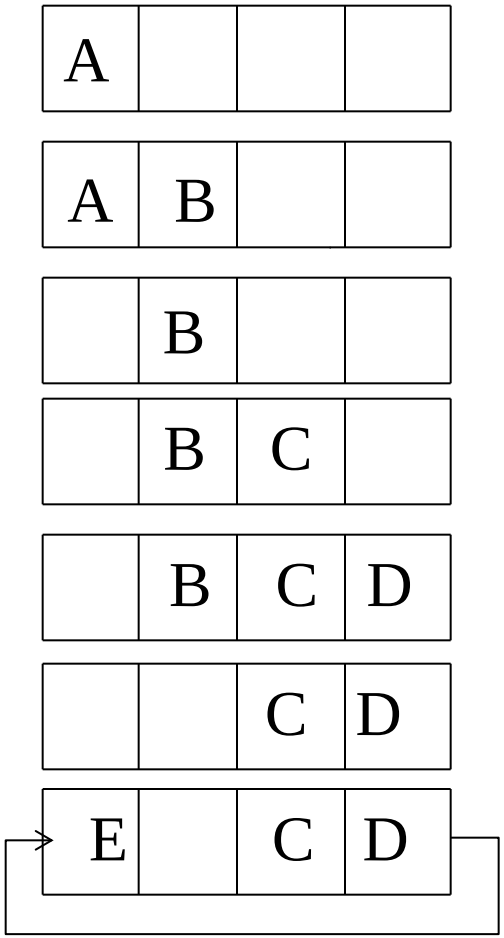
Solutie: deplasarea elementelor cu o poziție la stânga după fiecare extragere (c.f rămâne pe 1)

extrage(c) (cost: O(nmax)):

```
if c.s!=0
   elem ←c.x[c.f]
   for i ←2,c.s do
       c.x[i-1] ←c.x[i]
   endfor
   c.s ← c.s-1
   return elem
else
   „coada vidă”
```

Implementare cozi

Structura: c.x (tabloul ce conține valorile), c.f (indice față), c.s (indice spate), c.nre (numărul efectiv de elemente din coadă)



Initializare: c.f=1,c.s=0, c.nre=0

Adăugare „A”: c.f=1, c.s=1

Adăugare „B”: c.f=1, c.s=2

Extragere „A”: c.f=2, c.s=2

Adăugare „C”: c.f=2, c.s=3

Adăugare „D”: c.f=2, c.s=4

Extragere „B”: c.f=3, c.s=4

Adăugare „E”: c.f=3, c.s=1

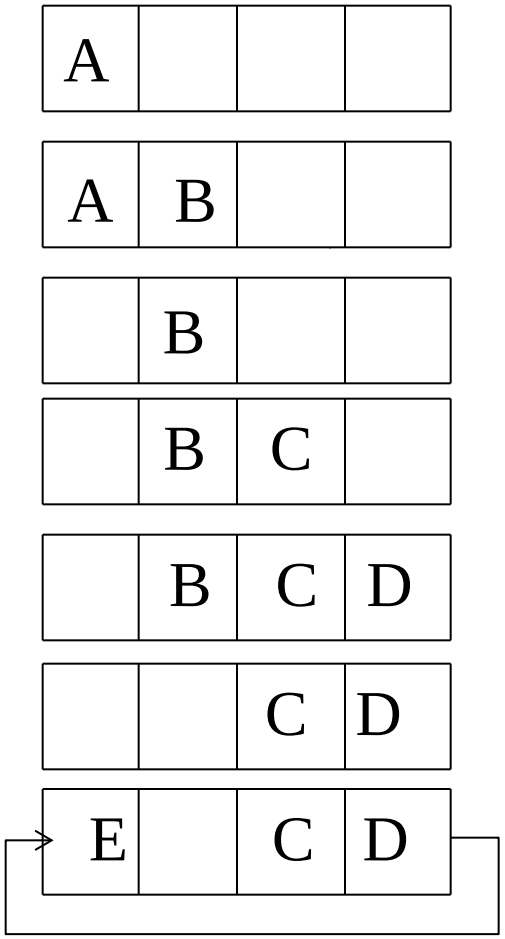
Coada circulară:

```
adauga (c,elem)
  if c.nre!=nmax then
    c.s ←c.s+1
    if c.s>nmax then c.s=1 endif
    c.x[c.s] ←elem
    c.nre ← c.nre+1
  else „coada plina”
  endif
```

Complexitate: $\Theta(1)$

Implementare cozi

Structura: c.x (tabloul ce conține valorile), c.f (indice față), c.s (indice spate), c.nre (numărul efectiv de elemente din coadă)



Initializare: c.f=1,c.s=0, c.nre=0

Adăugare „A”: c.f=1, c.s=1

Adăugare „B”: c.f=1, c.s=2

Extragere „A”: c.f=2, c.s=2

Adăugare „C”: c.f=2, c.s=3

Adăugare „D”: c.f=2, c.s=4

Extragere „B”: c.f=3, c.s=4

Adăugare „E”: c.f=3, c.s=1

Coada circulară:

```
extrage(c)
  if c.nre!=0 then
    elem ←c.x[c.f]
    c.f ←c.f+1
    if c.f>nmax then c.f=1 endif
    c.nre←c.nre-1
  else
    „coada vida”
  endif
```

Complexitate: $\Theta(1)$

Implementare cozi

```
# clasa pentru coada
# circulara (cu gestiunea
# explicita a nr de elemente)
class coada:
    def __init__(self,nmax):
        self.data=[None]*nmax
        self.f=0
        self.s=-1
        self.nmax=nmax
        self.nre=0
```

```
def adauga(self,e):
    if(self.nre!= self.nmax):
       self.s=self.s+1
       if self.s>=self.nmax:
                self.s=0
       self.data[self.s]=e
       self.nre=self.nre+1
    else:
       print(„Eroare:coada plina")
```

Observatie:

- acesta este doar un exemplu simplu de implementare.
- o implementare eficientă evită alocarea intregului tablou si gestiunea nr de elemente din coada

Implementare cozi

```
def extrage(self):
  if self.nre!=0:
      elem=self.data[self.f]
      self.f=self.f+1
      if self.f>=self.nmax:
        self.f=0
      self.nre=self.nre-1
   else:
    print(„Eroare:coada vida")
```

```
def vizualizare(self):
  if self.nre!=0:
    i=self.f;
    nr=1;
    while (nr<=self.nre):
      print(self.data[i])
      i=i+1
      if i>=self.nmax:
         i=0
      nr=nr+1
   else:
     print(„Eroare: coada vida")
```

Obs: vizualizarea cozii se poate realiza într-o manieră mai elegantă folosind particularități ale implementării claselor în Python (vezi lab 9)

Sumar

- Tip abstract de date: set de date + operații specifice de interogare și modificare
- Structura liniară: set de date + relație de precedență/succesiune (fiecare element are maxim un predecesor/succesor direct)
- Cazuri particulare (acces limitat la elementele structurii):
  - Stive
  - Cozi
- Variante de implementare:
  - Tablouri (zona contiguă => acces pe bază de indice)
  - Liste înlănțuite (zone disparate => acces pe bază de referință)

Cursul următor va fi despre…

… implementarea structurilor liniare prin liste înlănțuite

Intrebare de final

Se consideră următoarele operații aplicate asupra unei stive vide:

push(s,3)
push(s,1)
el←pop(s)
push(s,2)
push(s,4)
el ← pop(s)
el ← pop(s)

Ce va conține la final stiva și variabila el?

Variante de răspuns:

a) s=[1,3] și el=4
b) s=[3] și el=2
c) s=[1] și el=4
d) s=[1] și el =2
e) alt răspuns – precizați care