Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Structuri liniare

Liste. Stive. Cozi

- Inserare, cautare, stergere -

Lectii de pregatire pentru Admitere

03 / 03 / 2018

Structuri liniare (Liste. Stive. Cozi)

Cuprins

Liste (simple, duble, circulare)

Stive, Cozi (simple, speciale)

Subiectele vor fi abordate atat din perspectiva alocarii statice cat si a alocarii dinamice!

Structuri liniare (Liste. Stive. Cozi)

Structura liniara

- relatie de ordine totala pe multimea elementelor (fiecare element are un singur element **precedent** si un singur element **succesor**).

Exemple de structuri **liniare** – liste, stive, cozi

Exemple de structuri **neliniare**
- arbori
- elemente aflate in relatie de adiacenta data de o matrice

Structuri liniare (Liste. Stive. Cozi)

Clasificare

Dupa tipul de alocare:

- Structuri liniare în alocare statica (vectori)
- Structuri liniare în alocare dinamica (liste înlantuite)

Dupa modul de efectuare al operatiilor de intrare (inserarile) si de iesire (stergerile):

- Structuri liniare fara restrictii de intrare/iesire
- Structuri liniare cu restrictii de intrare/iesire (stive si cozi)

Structuri liniare - Liste

Operatii de baza

Traversarea - operatia care acceseaza fiecare element al structurii, o singura data, in vederea procesarii (*vizitarea* elementului).

Cautarea - se cauta un element cu cheie data in structura (*cu* sau *fara* succes) : consta dintr-o traversare - eventual incompleta a structurii, in care vizitarea revine la comparatia cu elementul cautat.

Inserarea - adaugarea unui nou element, cu pastrarea tipului structurii.

Stergerea - extragerea unui element al structurii (eventual in vederea unei procesari), cu pastrarea tipului structurii pe elementele ramase.

Liste liniare alocate secvential

Informatii de acelasi tip stocate in locatii de memorie contigue in ordinea indicilor (Nodurile se afla in pozitii succesive de memorie)

Avantaj: acces direct la orice nod

Dezavantaj: multe deplasari la operatiile de inserare si stergere

Liste liniare alocate secvential

Exemple

- lista de numere intregi

3	-12	10	7	1
0	1	2	3	4

- lista de numere reale

0.3	-1.2	10	5.7	8.7	0.2	-1.5	1

- lista de caractere

A	&	*	+	@	c	M	#

C / C++ | Pascal

Declarare

```
int a[20];
double b[30];
char c[23];
```

```
var a : array [1..20] of integer;
var b : array [1..30] of double;
var c : array [1..23] of char;
```

Liste liniare alocate secvential

C / C++

Pascal

Traversare (complexitate O(n))

```
for (i = 0; i<n; i++)
// viziteaza a[i];
```

```
for i:= 1 to n do
 { viziteaza a[i];}
```

Cautare (liniara – complexitate O(n))

```
int t = 0;
for (i = 0; i<n; i++)
if (a[i]==x) t = 1;
if (t==0) // cautare fara succes
```

```
var t : boolean;
t := false;
for i:= 1 to n do
    if (a[i] = x) then t := true;
if (t = false) then
write('cautare fara succes');
```

Liste liniare alocate secvential

Cautare liniara (componenta marcaj)

C / C++

```
int poz = 0, val;

a[n] = val;
while (a[poz] != val)
      {
      poz++;
      }
if (poz == n)
      // cautare fara succes
```

Pascal

```
var val, poz: integer;
poz := 1;

a[n+1] := val;
while (a[poz] <> val) do
      poz := poz + 1;

if (poz = n + 1) then
      { cautare fara succes}
```

Numarul de comparatii: n + 1 + 1

Liste liniare alocate secvential

Cautare binara (! pe vector ordonat) - $O(\log_2 n)$

C / C++

```
int l = 0, r = n-1, m, poz = -1;

m = (l+r) / 2;
while ((l <= r) && (val != a[m]))
  {
     if (val<a[m]) r = m-1;
                else l = m+1;
    m =(l+r) / 2;
   }

if (a[m]==val) poz = m;
```

Pascal

```
var l, r, m, poz: integer;
l := 1; r := n; poz:=0;

m := (l+r) div 2;
while (l <= r) and (val <> a[m]) do
  begin
     if (val<a[m]) then r := m-1
                else l := m+1;
    m:=(l+r) div 2;
   end;

if (a[m]=val) then poz:=m;
```

Liste liniare alocate secvential

Cautare binara (! pe vector ordonat) - $O(\log_2 n)$

Complexitate

Consideram cazul cel mai defavorabil (cautare fara succes)

Notatie: C(n) = numar de comparatii

- dupa o comparatie – cautarea se face pe un vector de lungime injumatatita

- in final avem un segment de un element

$$2^{C(n)} > n > 2^{C(n)-1} \Rightarrow C(n) < \log_2 n + 1 \Rightarrow \mathbf{C(n) = O(\log_2 n)}$$

Liste liniare alocate secvential

C / C++

Inserare (valoare **val** pe pozitia **poz**)

```
for (i = n-1; i >= poz; i--)
      a[i+1] = a[i];
a[poz] = val;
n++;
```

Stergere (valoare de pe pozitia **poz**)

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

Pascal

Inserare (valoare **val** pe pozitia **poz**)

```
for i:= n downto poz do
     a[i+1] := a[i];
a[poz]:=val;
n := n+1;
```

Stergere (valoare de pe pozitia **poz**)

```
for i := poz to n-1 do
a[i] := a[i+1];
n:=n-1;
```

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Stiva (LIFO)

- **LIFO (Last In First Out):** ultimul introdus este primul extras
- locul unic pt. ins./stergeri: varf ***(Top)***
- ***Push (Val)*** - inserarea valorii *Val* in stiva *(Stack)*
  - **Overflow (supradepasire)** - inserare in stiva plina
- ***Pop(X)*** - stergerea/extragerea din stiva *(Stack)* a unei valori care se depune in *X*
  - **Underflow (subdepasire) -** extragere din stiva goala

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Stiva (LIFO)

Exemplificarea operatiilor pe stiva (exercitiu dintr-o varianta de BAC)

Într-o stivă care memorează numere, o valoare **x** poate fi adăugată numai dacă în vârful stivei se află un element cu o valoare strict mai mare decât **x**; în caz contrar sunt eliminate toate elementele care nu îndeplinesc această condiție și apoi se adaugă valoarea **x**.

Exemplu: pentru stiva din **fig.1**, adăugarea elementului **11** este precedată de eliminarea elementelor ce conțin valorile **2** și **10**. După adăugare, stiva va avea conținutul din **fig.2**.

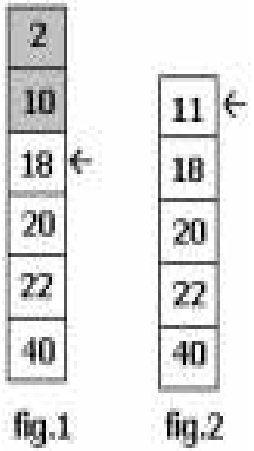


fig.1 fig.2

Dacă stiva este **inițial vidă**, care este numărul elementelor aflate în această stivă după adăugarea, respectând condițiile de mai sus, în ordine, a numerelor 20,5,16,9,3,7,5,4,8 ?

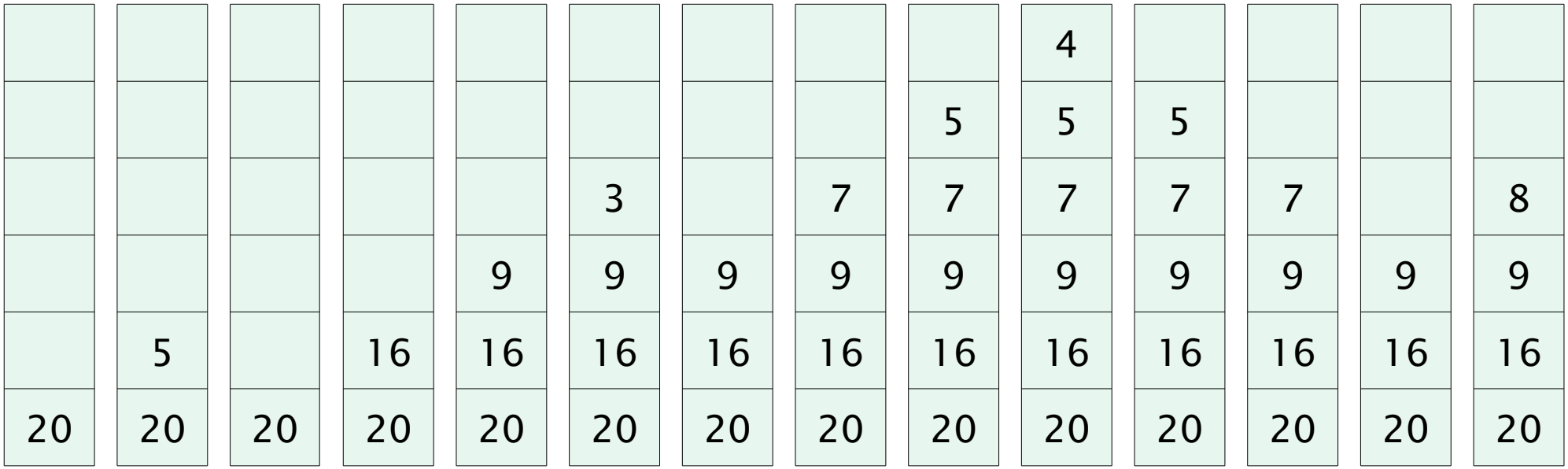
Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Stiva (LIFO)

Exemplificarea operatiilor pe stiva (exercitiu dintr-o varianta de BAC)

20,5,16,9,3,7,5,4,8

Stiva, dupa fiecare pas



Push (20) Pop () Push (9) Pop () Push (5) Pop () Pop () Pop ()
Push (5) Push (16) Push (3) Push (7) Push (4) Push (8)

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Stiva (LIFO)

Exemplificarea mecanismului RECURSIVITATII și ordinea efectuarii operatiilor

$$n! = \begin{cases} 1, \text{ dacă } n=0 \\ n*(n-1)!, \text{ dacă } n>=1 \end{cases}$$

```
int factorial(int n)
{
if (n==0) return 1; //conditia de oprire
return n*factorial(n-1);//recursivitate
}
```

4! = 4*3! = 4 * 6 = 24
3! = 3*2! = 3 * 2 = 6
2! = 2*1! = 2 * 1 = 2
1! = 1*0! = 1 * 1 = 1
0! = 1

Adâncimea recursivității

Ce se întâmplă în stivă pentru apelul t = factorial(4) ?

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Stiva (LIFO)

Exemplificarea mecanismului RECURSIVITATII și ordinea efectuarii operatiilor

Ce se întâmplă în stivă pentru apelul t = factorial(4) ?

STIVĂ

Se salvează un context de apel:
1.adresa de revenire
2.copii ale valorilor parametrilor efectivi
3.valorile variabilelor locale
4.copii ale regiștrilor
5.valoarea returnată

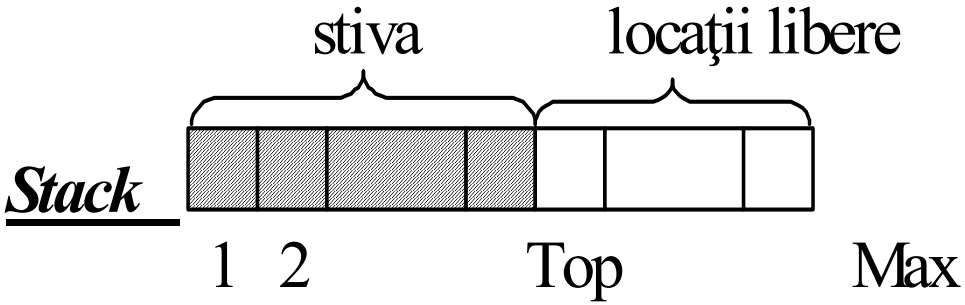
A_5	0	-	-	1
A_4	1	-	-	1
A_3	2	-	-	2
A_2	3	-	-	6
A_1	4	-	-	24

A1 = adresa de revenire pentru apelul factorial(4)

Sursa: Alexe B – Programare Procedurala (Note de curs 2015)

Stiva in alocare statica

Implementare



Declarare

C / C++

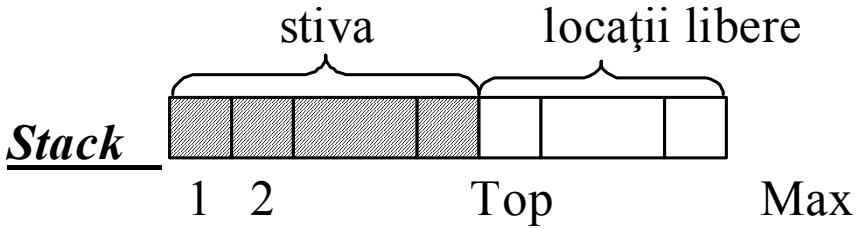
```
#define MAX 100

int Stack[MAX];
int Top = 0;
```

Pascal

```
var MAX: integer;
Stack : array [1..100] of integer;
Top:integer;
Top := 0;
MAX := 100;
```

Stiva in alocare statica

Implementare



Inserare

C / C++

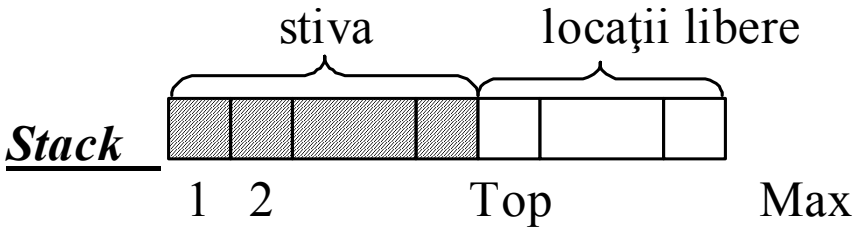
```
void Push (int Val)
{
if (Top == MAX)
                    // Overflow
else
      {
        Top++;
        Stack[Top] = Val;
      }
}
```

Pascal

```
procedure Push (Val : integer);
begin
if (Top = MAX) then
                    // Overflow
else
      begin
        Top := Top + 1;
        Stack[Top] := Val;
      end;
end;
```

Stiva in alocare statica

Implementare



Stergere

C / C++

```
void Pop (int &X)
{
if (Top == 0)
                    // Underflow
else
      {
        X = Stack[Top];
        Top--;
      }
}
```

Pascal

```
procedure Pop (var X:integer);
begin
if (Top = 0) then
                    // Underflow
else
      begin
        X := Stack[Top];
        Top := Top - 1;
      end;
end;
```

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Coada (FIFO)

- **FIFO (First In First Out):** primul introdus este primul extras
- capat pt. Inserari: sfirsit *(Rear)*
- capat pt. stergeri: inceput *(Front)*
- ***Push (Val)*** - inserarea
  - **Overflow (supradepasire)** - inserare in coada plina
- ***Pop(X)*** - stergerea/extragerea din coada *(Queue)* a unei valori care se depune in *X*
  - **Underflow (subdepasire) -** extragere din coada goala

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Coada (FIFO)

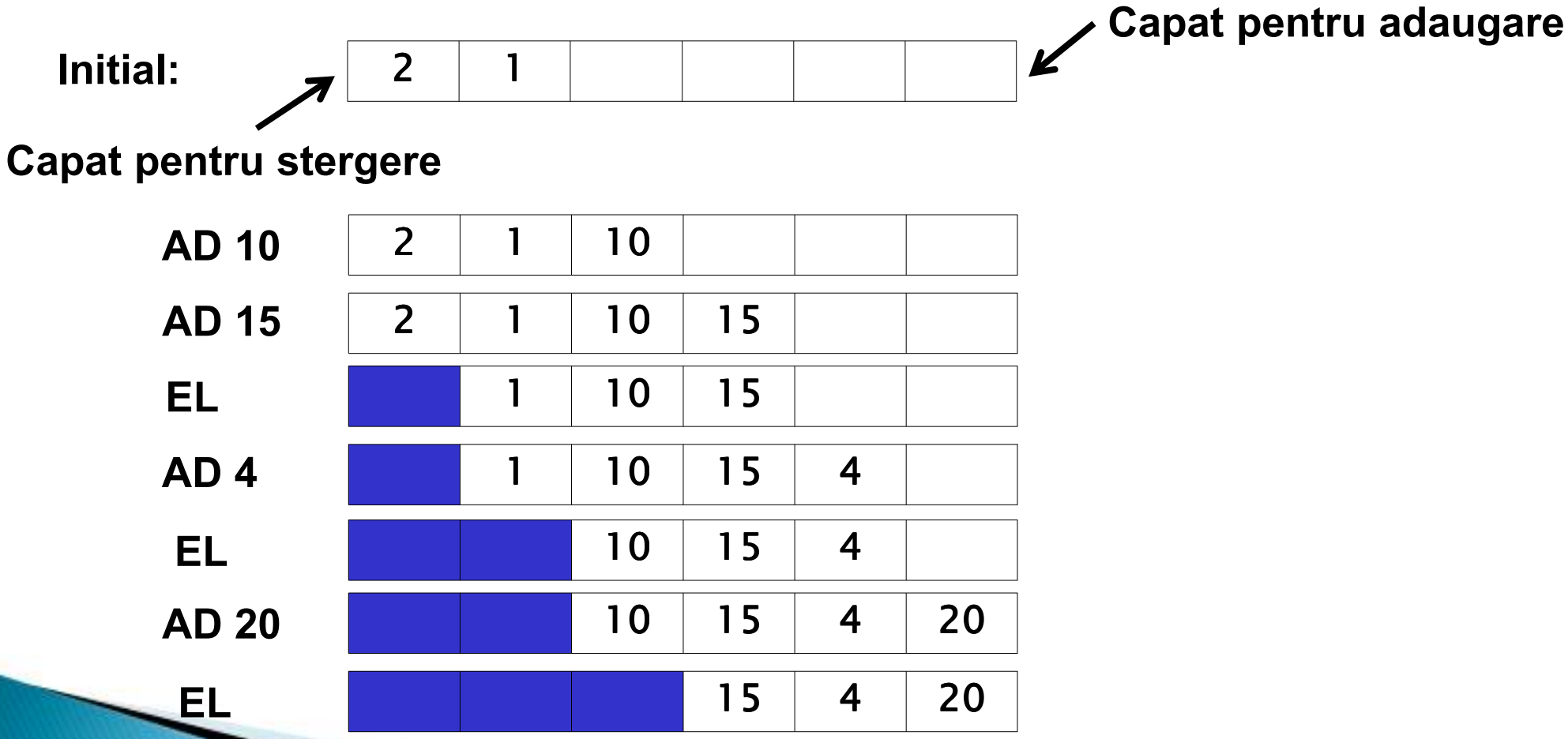
Exemplificarea operatiilor pe coada (exercitiu dintr-o varianta de BAC)

Se consideră o coadă, în care au fost introduse inițial, în această ordine, două numere: **2** şi **1**. Conținutul cozii este reprezentat în figura alăturată. Notăm cu **AD X** operația prin care se adaugă informația **X** în coadă şi cu **EL** operația prin care se elimină un element din coadă. Asupra cozii se efectuează, exact în această ordine, operațiile **AD 10**; **AD 15**; **EL**; **AD 4**; **EL**; **AD 20**; **EL**. Care este conținutul cozii după executarea operațiilor de mai sus? **(4p.)**

Liste liniare alocate secvential

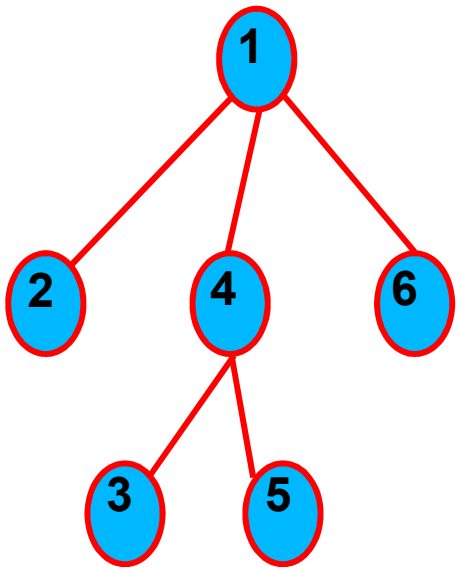
Structuri lineare cu restrictii la i/o: Coada (FIFO)

Exemplificarea operatiilor pe coada (exercitiu dintr-o varianta de BAC)

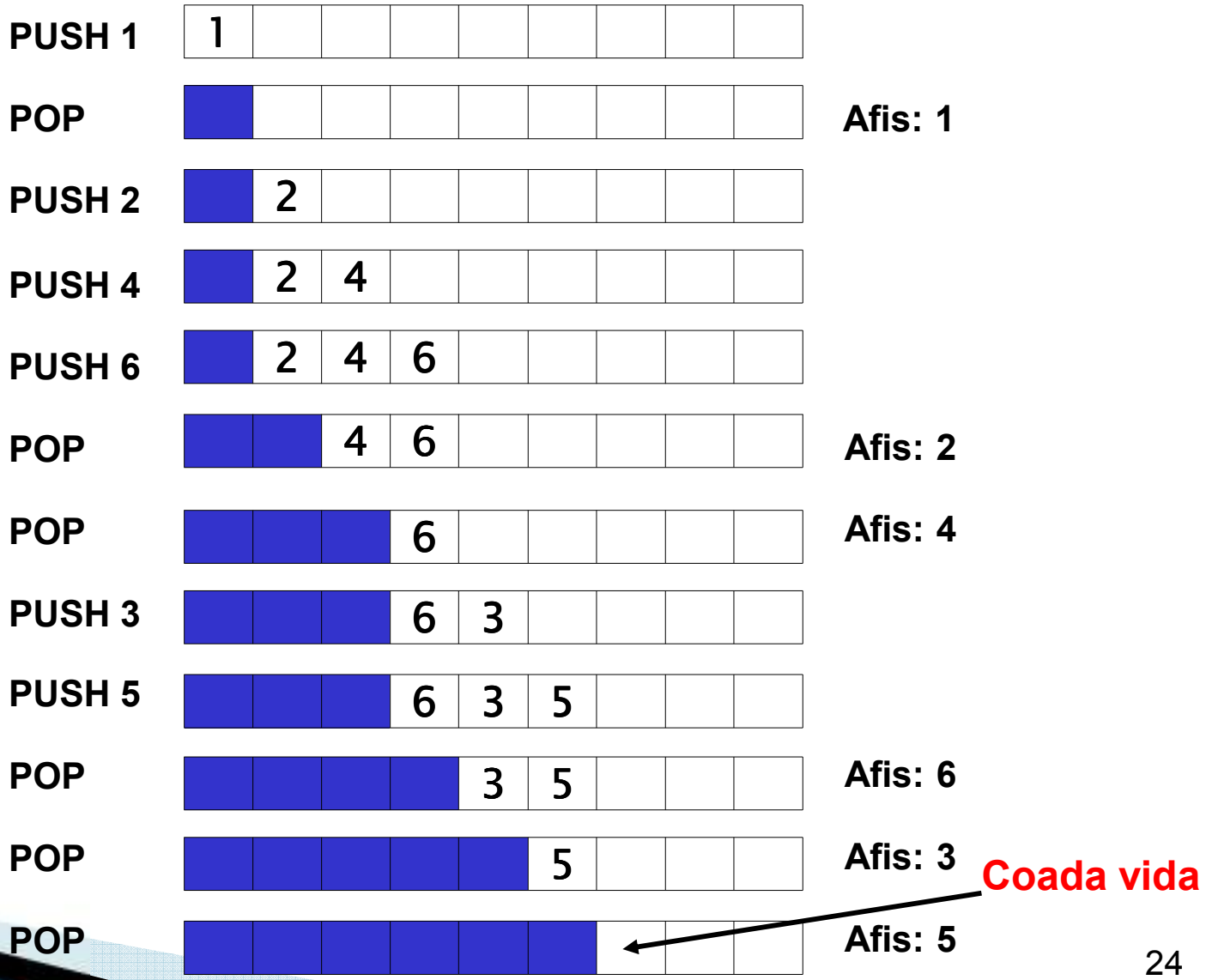


Structuri lineare cu restrictii la i/o: Coada (FIFO)

Exemplificarea operatiilor pe coada in parcurgerea unui arbore pe nivele (Breadth First)

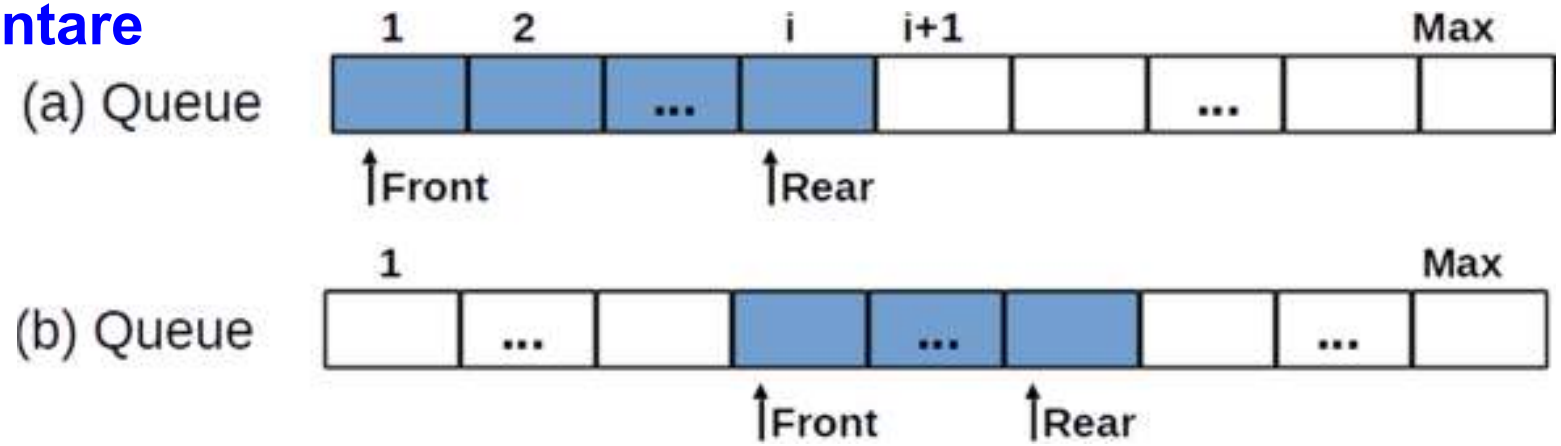


BF: 1, 2, 4, 6, 3, 5.



Coada in alocare statica

Implementare



Declarare

C / C++

```
#define MAX 100

int Queue[MAX];
int Front, Rear;
Front = Rear = 0;
```

Pascal

```
var MAX: integer;
Queue : array [1..100] of integer;
Front, Rear :integer;
MAX := 100;
Front := 0; Rear := 0;
```

Coada in alocare statica

Implementare

Inserare

C / C++

```
void Push (int Val)
{
if (Rear == MAX)
                  // Overflow
else
     {if (Rear == 0)
          //coada initial vida
          Front++;
      Rear++;
      Queue[Rear] = Val;}
}
```

Pascal

```
procedure Push (Val : integer);
begin
if (Rear = MAX) then
                  // Overflow
else
     begin
          if (Rear = 0) then
          // coada initial vida
               Front := Front + 1;
          Rear := Rear + 1;
       Queue[Rear] := Val;
     end;
end;
```

Coada in alocare statica

Implementare

Stergere

C / C++

```
void Pop (int &X)
{
if (Front == MAX)
                // Underflow
else {
     if (Front == 0 || Front > Rear)
                // Coada vida
     else
          {
                X = Queue[Front];
                Front++;}
     }
```

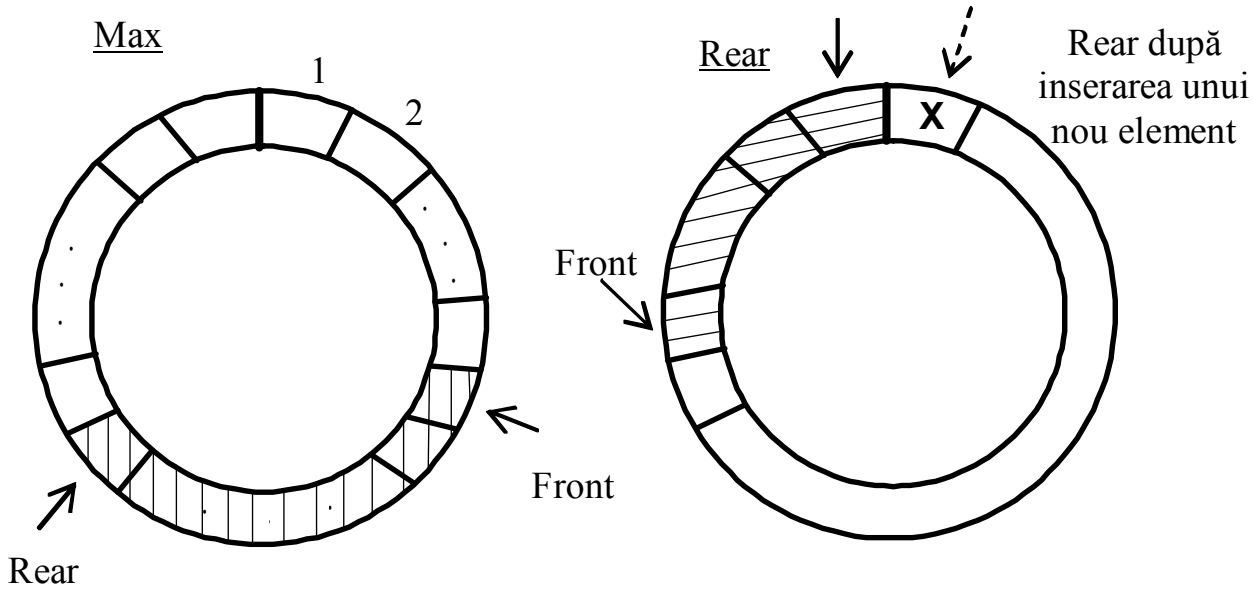
Pascal

```
procedure Pop (var X:integer);
begin
if (Front = MAX) then
                // Underflow
else
     begin
          if (Front = 0 OR Front> Rear)
            // Coada vida
               else begin
                    X := Queue[Front];
                    Front := Front + 1;
               end;
     end;
end;
```

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Alte tipuri de cozi

Coada circulara (in alocare statica)



Pe coada circulara: aritmetica (mod Max) la incrementarea indicilor
Coada vidă: *Front = Rear = 0.*
Coada plină (pe versiunea circulară): *Rear+1=Front (mod Max).*
Coada cu un singur element: *Rear = Front != 0.*

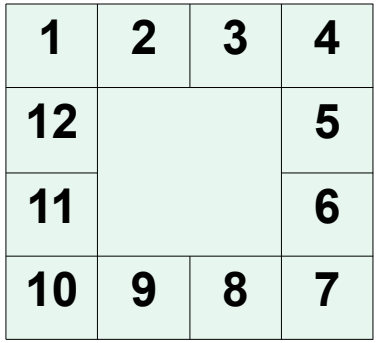
Liste liniare alocate secvential

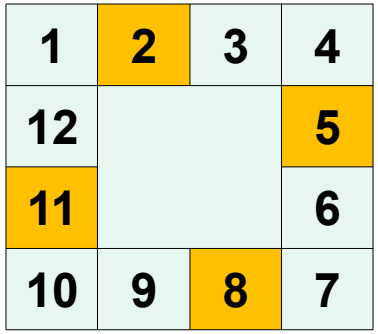
Structuri lineare cu restrictii la i/o: Alte tipuri de cozi

Exemplificare utilizarii unei cozi circulare – Problema Josephus

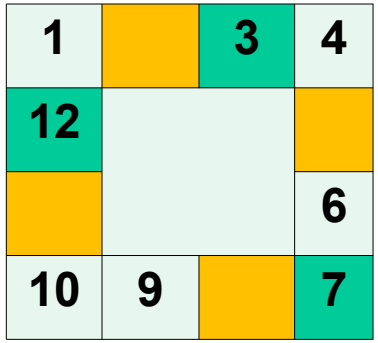
- n copii asezati in cerc sunt numarati din m in m plecand de la copilul k.
-fiecare al m – lea copil numarat iese din cerc.
-Afisare ordine iesire copii din cerc

n = 12
m = 3
k = 2;

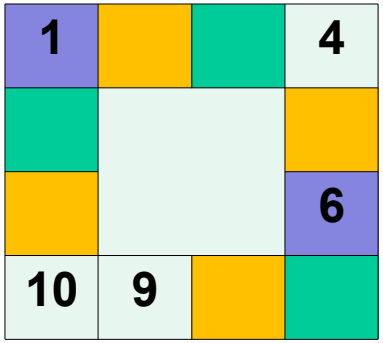




Afis: 2, 5, 8, 11

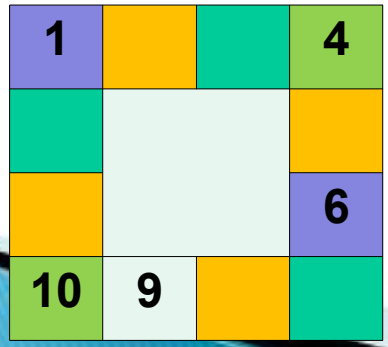


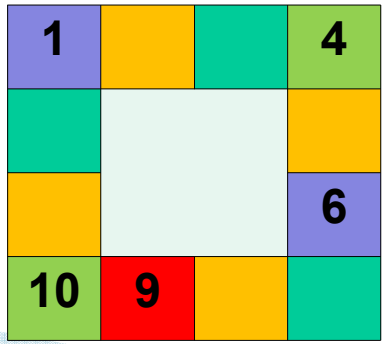
Afis: 3, 7, 12



Afis: 6, 1

Afis: 10, 4





Ordine: 2,5,8,11,3,7,12,6,1,10,4,9

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Alte tipuri de cozi

Coada cu priorităţi - Priority Queues

Elementele au, pe lângă cheie şi o prioritate:
- cea mai înaltă prioritate este 1, urmată de 2, etc.

Ordinea liniară este dată de regulile:

- elementele cu aceeaşi prioritate sunt extrase (şi procesate) în ordinea intrării;
- toate elementele cu prioritate *i* se află înaintea celor cu prioritate *i+1* (şi deci vor fi extrase înaintea lor).

Extragerile se fac dintr-un singur capăt.

Ca să se poată aplica regulile de mai sus la extragere, inserarea unui nou element cu prioritate *i* se va face la sfârşitul listei ce conţine toate elementele cu prioritate *i*.

Liste liniare alocate secvential

Structuri lineare cu restrictii la i/o: Alte tipuri de cozi

DEQUE - Double Ended Queue

- structură liniară în care inserările şi ştergerile se pot face la oricare din cele două capete, dar în nici un alt loc din coadă.

În anumite tipuri de aplicaţii sau în modelarea anumitor probleme pot apare structuri de cozi cu restricţii de tipul:

- inserările se pot face la un singur capăt şi extragerile la amândouă.

Liste liniare inlantuite

- alocate static si dinamic

Nodul contine informatia si indicele (adresa) urmatorului nod

Avantaj: operatiile de adaugare sau stergere sunt rapide

Dezavantaj:

- Accesul la un nod se face prin parcurgerea nodurilor precedente

- Indicele (adresa) nodului urmator ocupa memorie suplimentara

Liste liniare inlantuite alocate static

Declarare

C / C++

```
struct nod {
        int inf, urm;
        };

nod a[100];
int n, prim, ultim;
int oc[100];
// 0 – liber, 1-ocupat
Prim = ultim = 0;
```

Pascal

```
nod = record
        inf: integer;
      urm: integer;
        end;
 var a: array[1..100] of nod;
      n, prim, ultim: integer;
oc: array[1..100] of integer;
{0 – liber, 1-ocupat}
prim := 0; ultim := 0;
```

n = 7
prim = 6
ultim = 4

a	10	11	22	40	65	38	77			
	3	7	5	0	2	1	4			
oc	1	1	1	1	1	1	1	0	0	0
	1	2	3	4	5	6	7	8	9	10

Ordine: a[6], a[1], a[3], a[5], a[2], a[7], a[4]

Liste liniare inlantuite alocate static

C / C++

Alocare

```
i = 0;
while (oc[i] != 0)  i++;
oc[i] = 1;
n++;
```

Existenta spatiu de memorare

```
if (n<100)
    //exista
else
    // nu exista
```

Eliberare

```
// eliberare pozitie x
    oc[x] = 0;
    n--;
```

Pascal

Alocare

```
i := 0;
while (oc[i]<>0) do i := i+1;
oc[i] := 1;
n := n+1:
```

Existenta spatiu de memorare

```
if (n<100) then
    { exista }
else
    { nu exista }
```

Eliberare

```
{eliberare pozitie x}
    oc[x]:=0;
     n:=n-1;
```

Liste liniare inlantuite alocate static

C / C++ | Pascal

Inserare

Inserarea valorii "val" la sfarsitul listei

ultim = 4

a	10	11	22	40	65	38	77			
	3	7	5	0	2	1	4			
oc	1	1	1	1	1	1	1	0	0	0
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Exemplu val = 100

nou = 8
a[8].inf = 100
a[8].urm = 0
ultim = 8

a	10	11	22	40	65	38	77	100		
	3	7	5	8	2	1	4	0		
oc	1	1	1	1	1	1	1	1	0	0
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

Liste liniare inlantuite alocate static

Inserare

Inserarea valorii "val" la sfarsitul listei

C / C++

```
int nou;
   if (!prim)
   {   alocare(prim);
       a[prim].inf = val;
       a[prim].urm = 0;
       ultim = prim; }
   else if (n<100)
   {   alocare(nou);
       a[ultim].urm = nou;
       a[nou].inf = val;
       a[nou].urm = 0;
       ultim = nou;    }
   else cout<<"lipsa spatiu";
```

Pascal

```
var nou: integer;
   if (prim=0) then begin
       alocare(prim);
       a[prim].inf := val;
       a[prim].urm := 0;
       ultim := prim
end
   else if (n<100) then
   begin
        alocare(nou);
       a[ultim].urm := nou;
       a[nou].inf := val;
       a[nou].urm := 0;
       ultim := nou
     end
   else write('lipsa spatiu');
```

Liste liniare inlantuite alocate static

C / C++ — Inserare — Pascal

Inserarea valorii "val_ins" dupa valoarea "val"

a[3].inf = 22
a[3].urm = 5

a	10	11	22	40	65	38	77			
	3	7	5	0	2	1	4			
oc	1	1	1	1	1	1	1	0	0	0

Exemplu val = 100 dupa valoarea 22 (care se gaseste pe pozitia 3)

p = 3
nou = 8
a[8].inf = 100
a[8].urm = 5
a[3].urm = 8

a	10	11	22	40	65	38	77	100		
	3	7	8	0	2	1	4	5		
oc	1	1	1	1	1	1	1	1	0	0

Liste liniare inlantuite alocate static

C / C++

Inserare

Pascal

Inserarea valorii "val_ins" dupa valoarea "val"

```
int p, nou;
    if (n<100)
    {   p = prim;
        while(a[p].inf != val)
            p = a[p].urm;
        alocare(nou);
        a[nou].inf = val_ins;
        a[nou].urm = a[p].urm;
        a[p].urm = nou;
        if (a[nou].urm == 0)
             ultim = nou;
    }
else cout<<"lipsa spatiu";
```

```
var p, nou: integer;
if (n<100) then
    begin
        p := prim;
        while(a[p].inf <> val)
            p = a[p].urm;
        alocare(nou);
        a[nou].inf := val_ins;
        a[nou].urm := a[p].urm;
        a[p].urm := nou;
        if (a[nou].urm = 0)
      ultim := nou;
      end
   else write('lipsa spatiu');
```

Liste liniare inlantuite alocate static

C / C++ | Pascal

Inserare

Inserarea valorii "val_ins" inaintea valorii "val"

a[3].inf = 22
a[3].urm = 5

a	10	11	22	40	65	38	77			
	3	7	5	0	2	1	4			
oc	1	1	1	1	1	1	1	0	0	0

Exemplu val = 100 inaintea valorii 22

Precedentul valorii 22 = pozitia p
p = 1
nou = 8
a[8].inf = 100
a[8].urm = 3
a[2].urm = 8

a	10	11	22	40	65	38	77	100		
	8	7	5	0	2	1	4	3		
oc	1	1	1	1	1	1	1	1	0	0

Liste liniare inlantuite alocate static

C / C++ — Inserare — Pascal

Inserarea valorii "val_ins" inaintea valorii "val"

```
int p, nou;
   if (n<100)
      if (a[prim].inf == val) {
         alocare(nou);
         a[nou].inf = val_ins;
         a[nou].urm = prim;
         prim = nou; }
      else  {
         p = prim;
         while(a[a[p].urm].inf != val)
            p = a[p].urm;
         alocare(nou);
         a[nou].inf = val_ins;
         a[nou].urm = a[p].urm;
         a[p].urm = nou;  }
else cout<<"lipsa spatiu";
```

```
var p, nou: integer;
if (n<100) then
      if (a[prim].inf = val)  then
begin
         alocare(nou);
         a[nou].inf := val_ins;
         a[nou].urm := prim;
         prim := nou
end
else   begin
       p := prim;
      while(a[a[p].urm].inf <> val)
         p = a[p].urm;
      alocare(nou);
      a[nou].inf := val_ins;
      a[nou].urm := a[p].urm;
      a[p].urm := nou;   end
   else write('lipsa spatiu');
```

Liste liniare inlantuite alocate static

C / C++ | Inserare | Pascal

Stergerea valorii "val" din lista

a[3].inf = 22
a[3].urm = 5

a	10	11	22	40	65	38	77			
	3	7	5	0	2	1	4			
oc	1	1	1	1	1	1	1	0	0	0

Exemplu val = 22

Precedentul valorii 22 = pozitia p
p = 1
aux = 3
a[8].inf = 100
a[8].urm = 2
a[2].urm = 8

a	10	11	22	40	65	38	77			
	5	7	5	0	2	1	4			
oc	1	1	0	1	1	1	1	0	0	0

aux 3

Liste liniare inlantuite alocate static

Inserare

Stergerea valorii "val" din lista

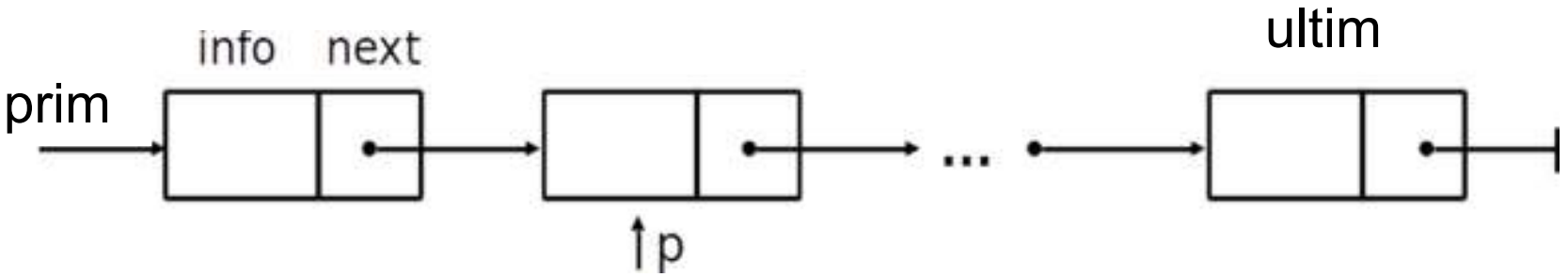
C / C++

```
int p, aux;
    if (a[prim].inf == val)
    {   aux = prim;
        prim = a[prim].urm;    }
    else
    {
        p = prim;
        while(a[a[p].urm].inf != val)
                    p = a[p].urm;
        aux = a[p].urm;
        a[p].urm = a[aux].urm;
        if (aux == ultim)
            ultim = p;
    }
    eliberare(aux);
```

Pascal

```
var p, aux: integer;
 if (a[prim].inf = val) then begin
        aux := prim;
        prim := a[prim].urm;
       end
    else begin
        p := prim;
        while(a[a[p].urm].inf <> val)
                    p := a[p].urm;
        aux := a[p].urm;
        a[p].urm := a[aux].urm;
        if (aux = ultim)
            ultim := p;
    }
    eliberare(aux);
```

Liste liniare inlantuite alocate dinamic



- ***prim*** retine adresa primului nod din lista, iar ***ultim*** retine adresa sfarsitului listei;
- fiecare nod conţine:

(1) un câmp, pe care se reprezintă un element al mulţimii;
în algoritmii care urmează putem presupune că elementul ocupă un singur câmp, *info*;

(2) un pointer către nodul următor, *urm*.

Liste simplu inlantuite

C / C++

Declarare

```
struct nod{
      int info;
      nod *urm;
};
nod *prim = NULL, *ultim;
```

Traversare

```
nod *p;
p = prim;
while (p != NULL)
        {
            // prelucrare p → info
            p = p → urm;
        }
```

Pascal

Declarare

```
type pnod = ^nod;
       nod = record
                inf :integer;
                urm :pnod;
                end;
var prim, ultim : pnod;
prim := nil;
```

Traversare

```
var p: pnod;
p := prim;
while (p <> nil) do
        begin
            {prelucrare p^.info}
            p := p^.urm;
        end
```

Liste simplu inlantuite

Cautare

C / C++

```
nod *p;
int x;

p = prim;
while (p != NULL  &&  x != p→info)
      p = p → urm;

if (p == NULL) // negasit;
else // gasit in p
```

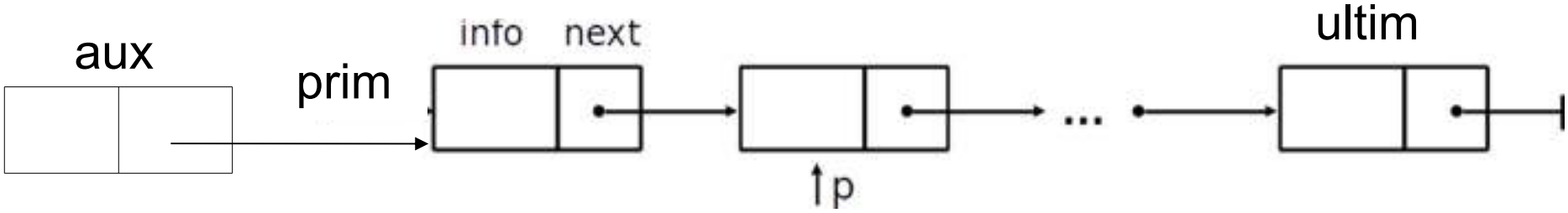
Pascal

```
var p: pnod;
int x;

p := prim;
while (p <> nil) and (x <> p^.info) do
        p := p^.urm;

if (p = nil) then {negasit}
else {gasit in p}
```

Liste simplu inlantuite

Inserare

Inserarea la inceputul listei



C / C++

```
// aux = nodul de inserat

nod* aux = new nod;
    // prelucrare aux->info
if (prim != NULL)
          aux->urm = prim;
else
     aux->urm = NULL;
prim = aux;
```

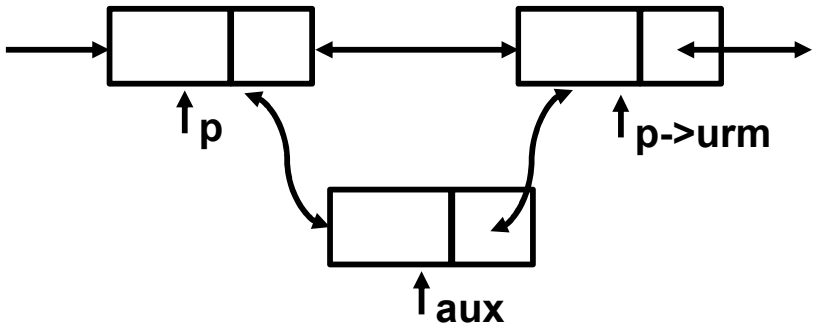
Pascal

```
{aux = nodul de inserat}

new (aux);
// prelucrare aux^.info;
if (prim <> nil) then
      aux^.urm := prim
else
      aux^.urm := nil;
prim := aux;
```

Liste simplu inlantuite

Inserare

Inserarea in interiorul listei



C / C++

```
nod *p, *aux;

aux = new nod;
// prelucrare aux → info;
aux → urm = p → urm;
p → urm = aux;
```

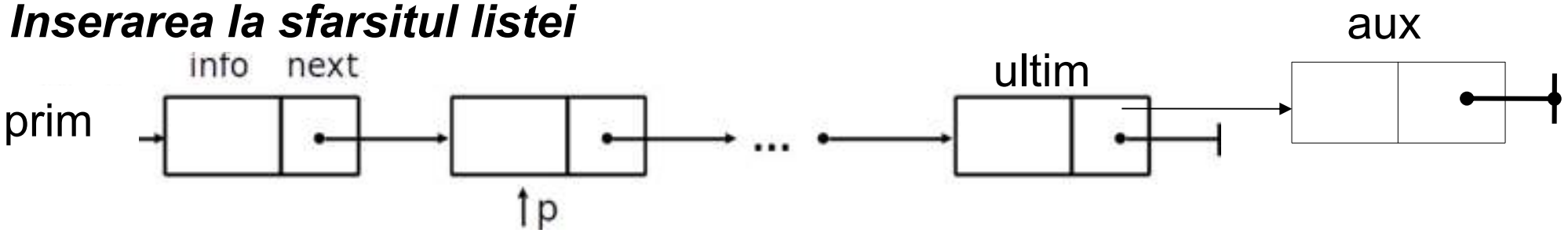
Pascal

```
var p, aux: pnod;

new (aux);
{ prelucrare aux^.info;}

aux^.urm := p^.urm;
p^.urm := aux;
```

Liste simplu inlantuite

Inserare

Inserarea la sfarsitul listei



C / C++

```
// aux = nodul de inserat
nod* aux = new nod;
    // prelucrare aux->info
      aux->urm = NULL;
if (prim != NULL)
          { aux->urm = ultim;
            ultim = aux;}
else {prim = aux;
      ultim = aux; }
```

Pascal

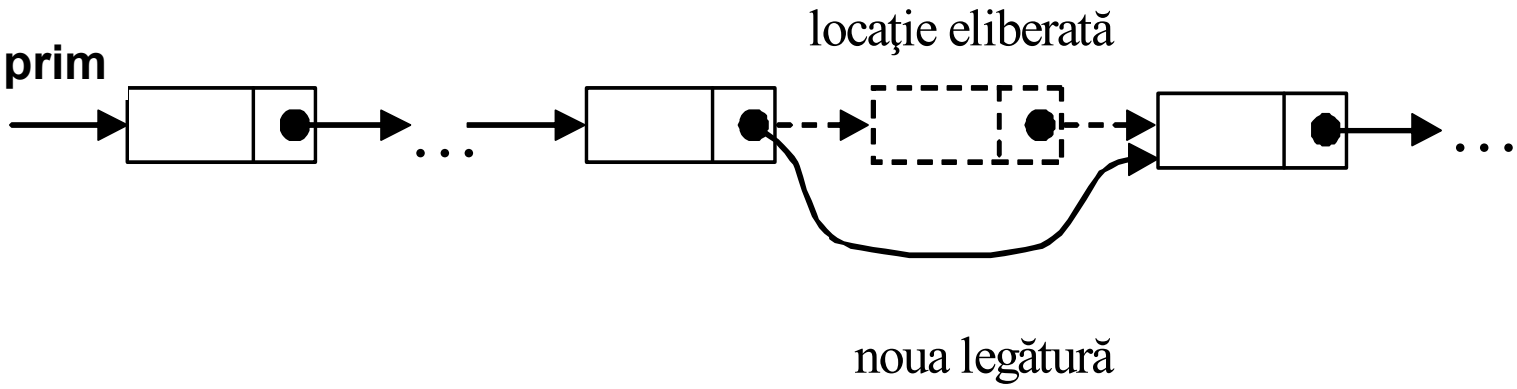
```
{aux = nodul de inserat}
new (aux);
{ prelucrare aux^.info;}
aux^.urm := nil;
if (prim <> nil) then begin
      aux^.urm := ultim;
      ultim := aux; end
else begin
       prim := aux;
      ultim := aux;
end;
```

Liste simplu inlantuite

C / C++

Pascal

Stergere



Refacerea structurii de lista simplu inlantuita pe nodurile ramase

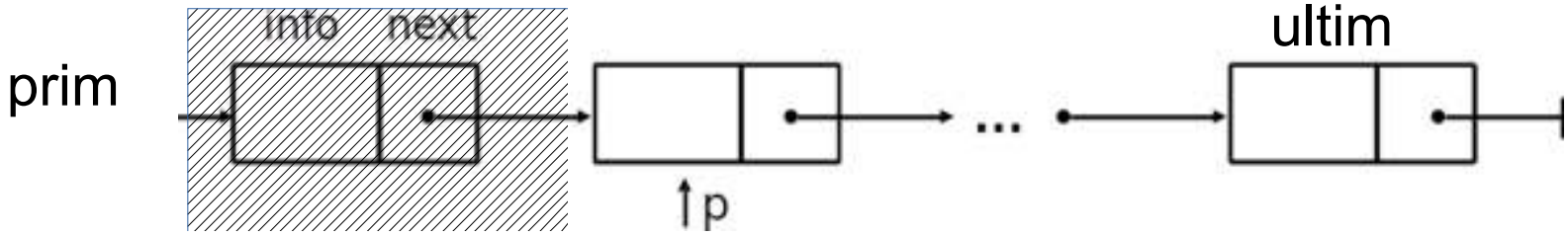
Eventual dezalocare de spatiu pentru nodul extras (sau alte operatii cu el)

Liste simplu inlantuite

Stergere

C / C++ | **Pascal**

Stergerea la inceputul listei




```
if (prim != NULL)
{
nod *temp = prim;
 prim = prim → urm;
delete temp;
}
```

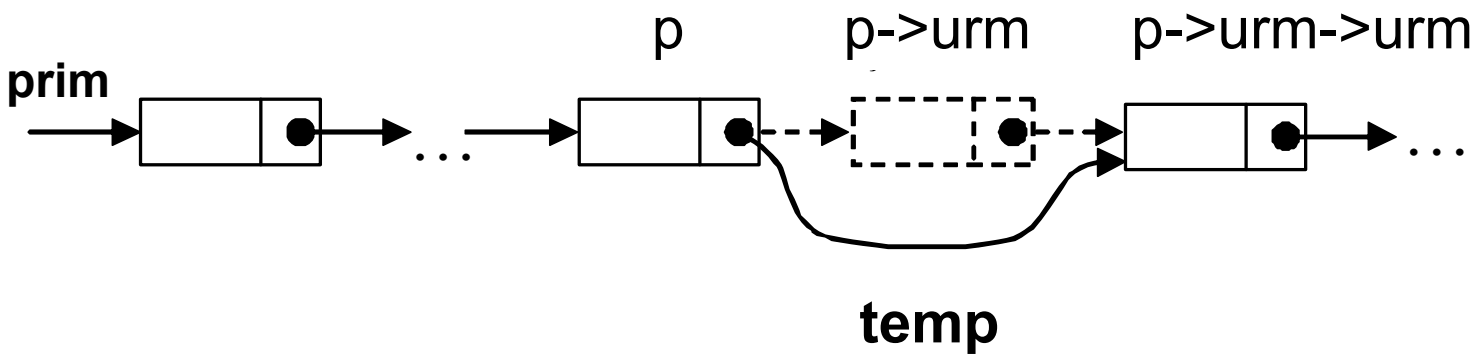
```
temp : pnod;

if (prim <> nil) then
      begin
      temp := prim;
       prim := prim ^.urm;
      dispose (temp);
      end
```

Liste simplu inlantuite

C / C++ | Pascal

Stergere

Stergerea in interiorul listei

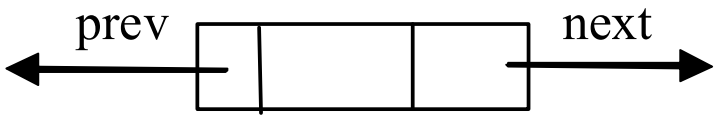



```
nod *temp = p → urm;
p → urm = p → urm → urm;
delete temp;
```

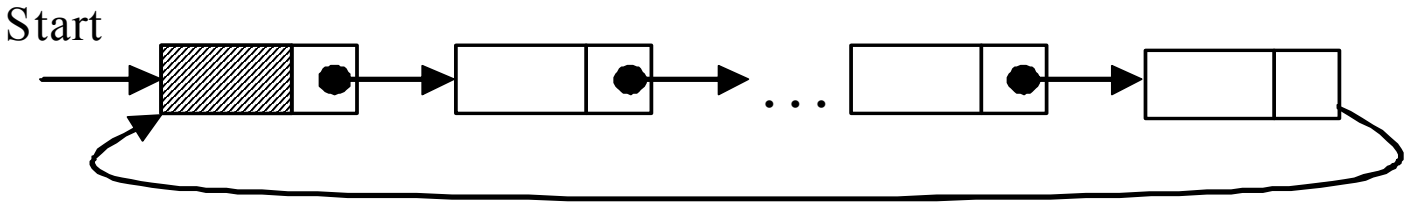
```
temp : pnod;
temp := p^.urm;
p^.urm := p^.urm^.urm;
dispose (temp);
```

Alte tipuri de liste

- **cu nod marcaj**
- **circulare**
- **dublu inlantuite**
- **alte inlantuiri (liste de liste, masive, etc.)**



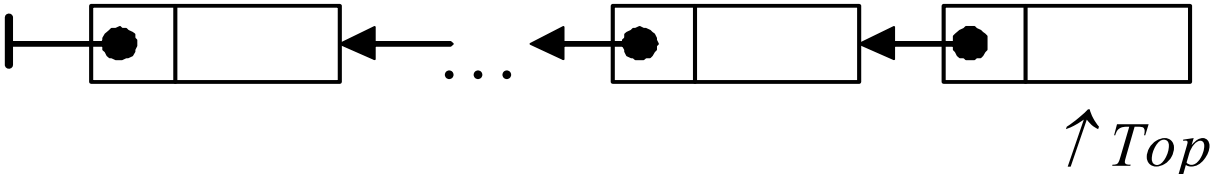
Nod într-o listă dublu înlănţuită.



Listă circulară cu nod marcaj.

Stiva in alocare dinamica

C / C++




```
struct nod {
      int info;
      nod *urm;
};

nod * Top = NULL;
```

Pascal

```
type pnod = ^nod;
      nod = record
            inf :integer;
            urm :pnod;
            end;

var Top : pnod;
Top := nil;
```

Se refac operatiile de adaugare si stergere de la liste simplu inlantuite, respectand restrictiile!

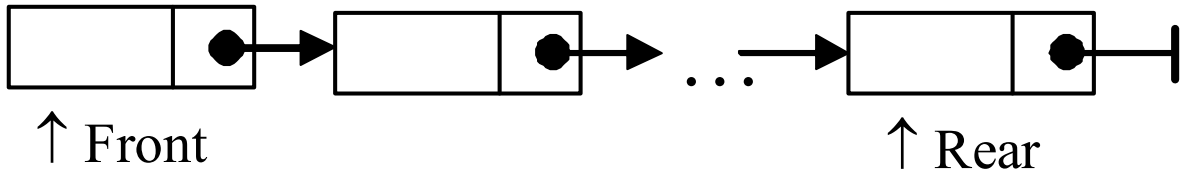
Stiva in alocare dinamica

Exemplificare operatiilor C++

```
void push(nod*& Top, int val)
{
    nod* aux = new nod;
    aux->info = val;
    aux->urm = NULL;
    if (Top == NULL)
        Top = aux;
    else
    {
        aux->urm = Top;
        Top = aux;
    }
}
```

```
void pop(nod*& Top)
{
    if(Top!=NULL)
    {
        cout<<Top->info;
        nod* aux = Top;
        Top = Top ->urm;
        delete aux;
    }
    else cout<<"Stiva vida\n";
}
```

Coada in alocare dinamica



Inserari – *Rear*

Stergeri - *Front*

Coada vidă: *Front = Rear = NULL*.

Coada cu un singur element: *Rear = Front != NULL*.

Se refac operatiile de adaugare si stergere de la liste simplu inlantuite, respectand restrictiile!

Coada in alocare dinamica

Exemplificare operatiilor C++

```
void push(nod*& Front, nod*& Rear, int val)
{
    nod* aux = new nod;
    aux->info = val;
    aux->urm = NULL;
    if (Front == NULL)
    {
        Front = aux;
        Rear = Front;
    }
    else
    {
      Rear ->urm = aux;
      Rear = aux;
    }
}
```

```
void pop(nod*& Front)
{
    if(Front!=NULL)
    {
        cout<<Front->info;
        nod* aux = Front;
        Front = Front ->urm;
        delete aux;
    }
    else cout<<"Coada vida\n";
}
```

Aplicatii

Stive si cozi

1. Exemplificare mecanisme

Se dau structurile: o stiva S si doua cozi C1 si C2, ce contin caractere. Cele trei structuri se considera de capacitate infinita, si initial vide. Se considera urmatoarele operatii:

'x' : se introduce caracterul x in S;
1 : daca S e nevida, se extrage un element si se introduce in C1, altfel nu se face nimic;
2 : daca C1 e nevida, se extrage un element si se introduce in C2, altfel nu se face nimic;
3 : daca C2 e nevida, se extrage un element si se introduce in S, altfel nu se face nimic.

(a) Sa se scrie continutul stivei S si al cozilor C1 si C2, dupa executarea urmatoarelor secvente de operatii: R 1 C 1 H 1 2 2 S E A R T 1 1 E E 2 2 2 1 1 2 2 3 3 3

(b) Sa se scrie o secventa de operatii in urma careia stiva S sa contina cuvantul BUBBLE, coada C1 sa fie vida, iar coada C2 sa contina cuvantul SORT.

Aplicatii

Stive si cozi

2. Parantezarea corecta

Dat un sir $s = s_1 s_2 \ldots s_n$ de caractere '(' si ')' sa se verifice daca acest sir este quasi - corect parantezat (i.e., pentru orice subsir $s_1 s_2 \ldots s_i$ avem ca numarul de caractere '(' este mai mare sau egal cu numarul de caractere ')').

In caz ca s nu este este quasi - corect parantezat, se va indica pozitia primei paranteze ')' care nu are corespondent.

Expl: ()(()) – corect
()(())(()(())) – corect
()(()))()))) – incorect (prima paranteza ‘)’ care nu are corespondent este pe pozitia 7

Aplicatii

Stive si cozi

2. Parantezarea corecta

C / C++

```
bool ok=true;
for(int i=0; i<strlen(s); i++)
   {if(empty(Stack))      // Stiva e vida
     { if(s[i]==')')
          { ok=false; break;}
     push(s[i],Stack);}
     else
          if(s[i] == peek(Stack))
               push(s[i],Stack);
          else
               pop(Stack);
}
if(ok) cout<<"Corect";
else cout<<"Incorect";
```

Pascal

```
var ok:boolean; ok:=true;
for i:=0 to length(s) do
begin
     if(empty(Stack)=true) then
     // Stiva vida
     begin
      if (s[i] = ')' ) then
          begin
           ok=false; break;
          end;
     push(s[i],Stack);
     end
     else
          if(s[i] = peek(Stack)) then
               push(s[i],Stack);
          else
               pop(Stack);
end;
```

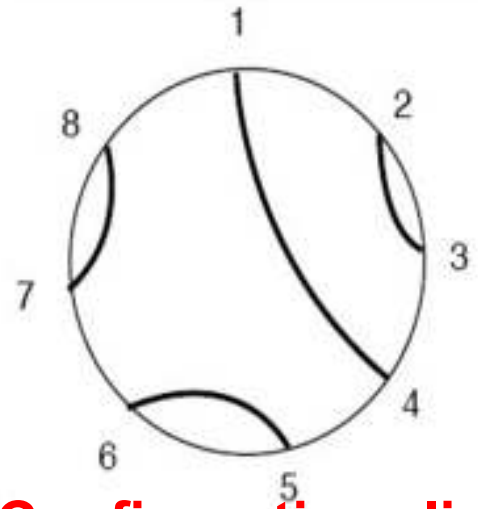
Aplicatii

Stive si cozi

3. Conectarea pinilor

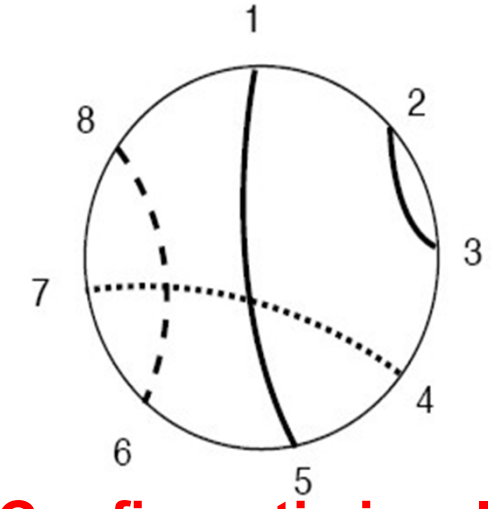
Se da o suprafata circulara cu un numar n de pini pe margini (numerotati de la 1 la n), impreuna cu o lista de perechi de pini ce trebuie conectati cu fire metalice.
Problema cere sa determinati in timp **O(n)** daca pentru o configuratie ca mai sus, pinii pereche pot fi conectati, fara ca acestea sa se intersecteze.

Pereche = {1,2,2,1,5,5,7,7}



Configuratie valida

Pereche = {1,2,2,4,1,6,4,6}



Configuratie invalida

Aplicatii

Stive si cozi

3. Conectarea pinilor

C / C++

```
// citire vector pereche

for(int i=0; i<n; i++)
   { if(empty(Stack))     // Stiva e vida
     push(pereche[i],Stack);
     else
          if(pereche[i] == peek(Stack))
               pop(Stack);
          else
               push(pereche[i],Stack);
}
if (empty(Stack))
      cout<<"Configuratie valida";
else cout<<"Configuratie invalida”;
```

Pascal

```
{ citire vector pereche}

for i:=1 to n do
begin
      if(empty(Stack)=true) then
      // Stiva vida
      push(pereche[i],Stack);
else
      if(pereche[i] = peek(Stack)) then
                pop(S);
           else
                push(pereche[i],Stack);
end;
if (empty(Stack) = true)
      write('Configuratie valida')
else
      write('Configuratie invalida');
```

Aplicatii

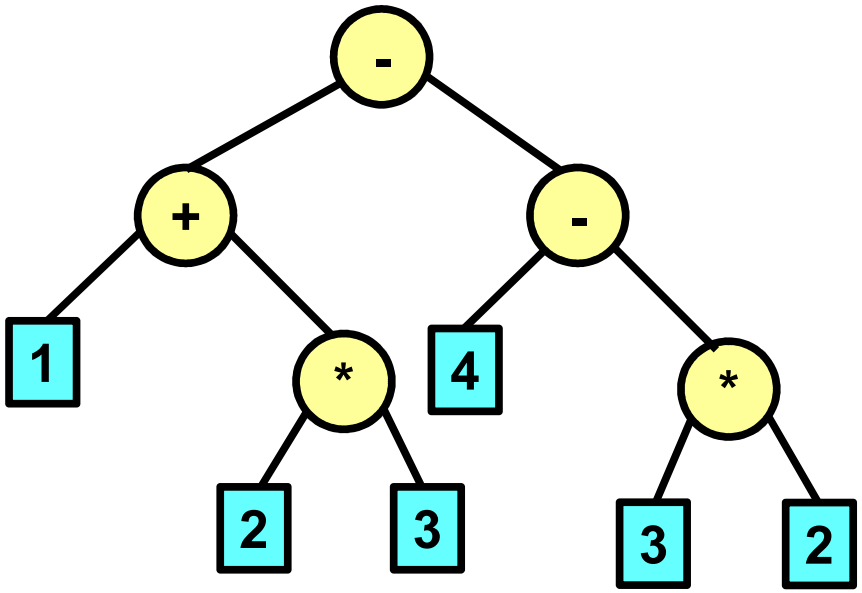
Stive si cozi

4. Evaluarea unei expresii în notatie postfixata

Exemplu

(1 + 2 * 3) - (4 - 3 * 2)
in notatie postfixata: 1 2 3 * + 4 3 2 * - -

Arborele binar asociat expresiei



Parcurgerea in preordine:
- + 1 * 2 3 - 4 * 3 2

Parcurgerea in inordine:
1 + 2 * 3 - 4 - 3 * 2

Parcurgerea in postordine:
1 2 3 * + 4 3 2 * - -

Aplicatii

Stive si cozi

4. Evaluarea unei expresii în notatie postfixata

Algoritm

Pas 1. – se citeste un sir de caractere, reprezentand expresia in **postfix**; **se considera diferentierea intre operanzi (/ operator) spatiul**; Stiva initial vida;
Pas 2. - se considera, pe rand, fiecare caracter.
Daca este “spatiu”, se trece la urmatorul;
Daca este operand → **Pas 3**;
Altfel → **Pas 4**;
Pas 3. - daca este operand, atunci:
- se extrag din stiva ultimele valori inserate, se aplica operandul si noua valoare se reintroduce in stiva

Pas 4. – se transforma caracterul in cifra si se adauga la numarul care va fi introdus in stiva
// numar = numar*10 + (caracter – '0') *
// cifra = cod ASCII caracter - 48 (codul caracterului '0')
Se repeta **Pas 2** pana la terminarea sirului de caractere introdus.
Pas ultim. Rezultatul este singura valoare aflata in stiva.

Aplicatii

Stive si cozi

4. Evaluarea unei expresii în notatie postfixata

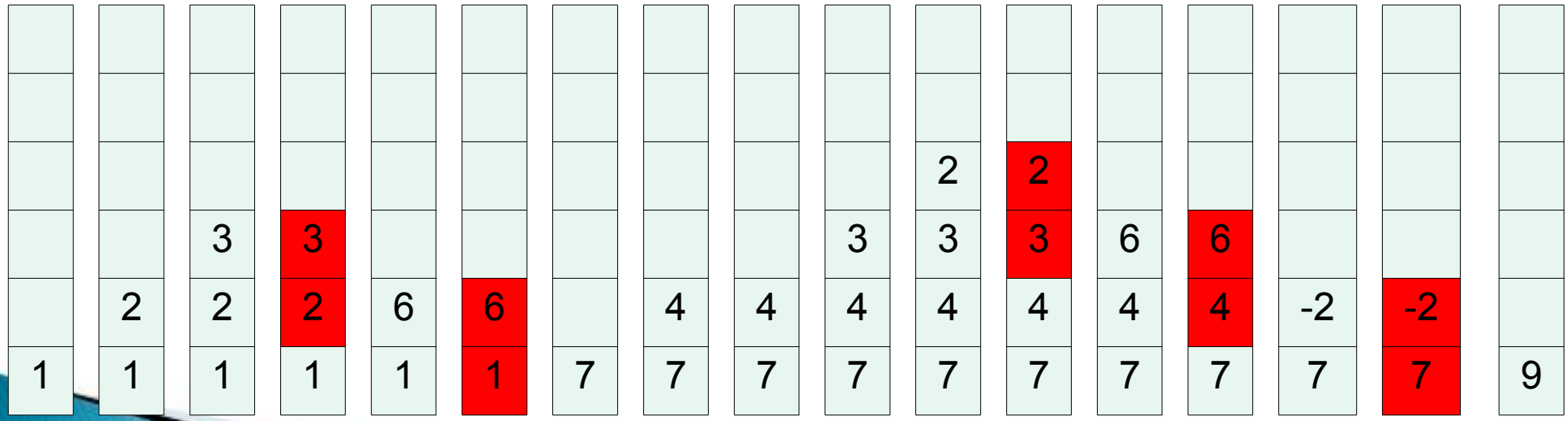
Exemplu

(1 + 2 * 3) - (4 - 3 * 2)

in notatie postfixata:

1 2 3 * + 4 3 2 * - -

Stiva, dupa fiecare pas



Aplicatii

Stive si cozi

4. Evaluarea unei expresii în notatie postfixata

Detalii de implementare (C++)

Pas 2. – parcurgerea sirului caracter cu caracter si verificarea acestuia (spatiu/ operator/ operand)

```
int evaluare(char *postfix) // postfix - sirul de caractere introdus
{

    int rez;

    for(int k = 0; k<strlen(postfix); k++)
    {
        if (postfix[k]==' ') continue;
        if (eOperator(postfix[k])==1) // daca acesta este operator
            {
            else
            {
            afis_fara_stergere();
    }
    pop(rez);
    return rez;
}
```

Aplicatii

Stive si cozi

4. Evaluarea unei expresii în notatie postfixata

Detalii de implementare (C++)

Pas 3. – caracterul este operator (+,-,*,/,%)

```
int eOperator(char x)
{
    if (x=='+' || x=='-' || x=='*' || x=='/' || x=='%') return 1;
    return 0;
}
```

```
if (eOperator(postfix[k])==1) // daca acesta este operator
    {
        int op1, op2;
        pop(op1);  // se scot ultimele 2 valori din stiva si se aplica operandul
        pop(op2);

        if(postfix[k]=='+') rez = op2+op1;
        else if (postfix[k]=='-') rez = op2-op1;
        else if(postfix[k]=='/') rez = op2/op1;
        else if (postfix[k]=='*') rez = op2*op1;
        else if (postfix[k]=='%') rez = op2%op1;

        push(rez); // se reintroduce in stiva rezultatul operatiei
    }
```

Aplicatii

Stive si cozi

4. Evaluarea unei expresii în notatie postfixata

Detalii de implementare (C++)

Pas 4. – caracterul este cifra

```
int eCifra(char x)
{
    if (x>='0' && x<='9') return 1;
    return 0;
}
```

```
else
{
   int numar=0; // daca acesta este cifra, se updateaza numarul care se introduce in stiva
   while(k<strlen(postfix) && eCifra(postfix[k])==1)
   {
      numar = numar * 10 + (postfix[k]-'0');
      k++;
   }
   push(numar); // numar intreg = cod ASCII caracter - 48 (codul caracterului '0')
}
```

Aplicatii

Stive si cozi

5. Parcurgerea unui arbore pe nivele (BF)

C / C++

```
Front = 1;Rear = 1;  // Q[ ] - coada
// a – matricea de adiacenta
cin>>nod; // de inceput
Q[Front]=nod;
viz[nod]=1;
while(Front <= Rear)
   {
     cout<<Q[Front;
     for(i=1;i<=n;i++)
       if( a[Q[Front]][i]==1 && viz[i]!=1 )
                 {   Rear++;
                     Q[Rear] = i;
                     viz[i] = 1; }
       Front++;
     }
```

Pascal

```
Front := 1; Rear := 1;
read(nod); // de inceput
Q[Front] := nod;
viz[nod] := 1;
while (Front <= Rear) do
    begin
      write(Q[Front],' ');
      for i := 1 to n do
       if (a[Q[Front]][i]=1) and (viz[i]!=1) then
                  begin
            Rear := Rear + 1;
                     Q[Rear] := i;
                     viz[i] := 1;
            end;
          Front := Front + 1;
        end;
```

Aplicatii

Stive si cozi

6. Depou feroviar

Un depou feroviar consta dintr-o linie ferata de intrare, k linii auxiliare de depozitare, si o linie de iesire. Fiecare linie opereaza pe un sistem de coada (FIFO). In plus, vagoanele se pot deplasa doar dinspre linia de intrare spre linia se iesire.

Sa se scrie un program care, dat un sir de vagoane pe linia de intrare (numerotate de la 1 la n si aranjate in orice ordine), descrie o strategie de a obtine pe linia de iesire sirul de vagoane n; n - 1;...; 2; 1, folosind liniile de depozitare.

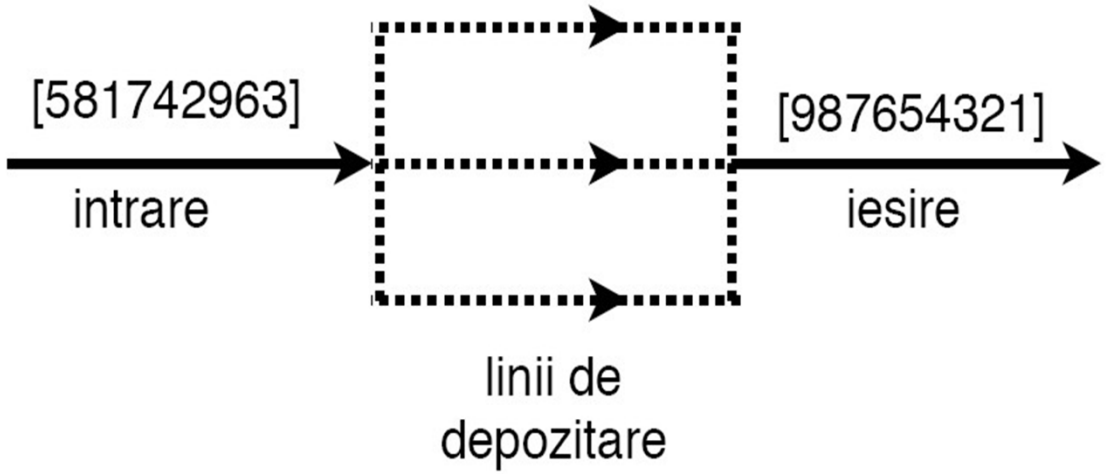
In caz ca nu exista o astfel de strategie, se va afisa acest lucru.

Aplicatii

Stive si cozi

6. Depou feroviar

Exemplu de depou feroviar cu 3 linii de depozitare



Rezolvare posibila:

Se extrag 3, 6, 9 din coada initiala si se adauga in prima coada auxiliara
Se extrag 2, 4, 7 din coada initiala si se adauga in a doua coada auxiliara
Se extrage 1 din coada initiala si se adauga in a treia coada auxiliara
Se extrage 8 din coada initiala si se adauga in a doua coada auxiliara
Se extrage 5 din coada initiala si se adauga in a treia coada auxiliara
Coada initiala este vida -> se extrag in ordine crescatoare elementele din cozile auxiliare si se introduc in coada finala.

Aplicatii

Stive si cozi

6. Depou feroviar

Algoritm

Pas 0. – se citesc elementele din coada initiala (CI)

Pas 1. - se extrage un element din CI. Daca elementul extras coincide cu pozitia sa in coada finala, atunci **Pas 2**, altfel **Pas 3**:

Pas 2. - se cauta prima coada auxiliara vida, prin care putem duce direct elementul extras in coada finala (CF). Daca nu exista o coada auxiliara vida, atunci STOP (Nu exista solutie pentru configuratia initiala si numarul auxiliar de cozi).

Pas 3. – se cauta locul de inserat pentru elementul extras la **Pas 1** (prima coada auxiliara pentru care ultima valoare este mai mica decat elementul extras).
Daca se respecta conditia, se revine la **Pas 1**, altfel STOP (Nu exista solutie pentru configuratia initiala si numarul auxiliar de cozi).

Pas 4. – Daca CI este vida si solutia este posibila, atunci se extrag valorile existente in cozile auxiliare in ordinea crescatoare a primelor elemente si se adauga in coafa finala (CF).

Aplicatii

Stive si cozi

Detalii de implementare in C++

6. Depou feroviar

Rutine ajutatoare

```
struct nod
{
    int info;
    nod *urm;
};

struct coada
{
    nod* Front, *End;
};
```

```
void initCoada(coada &c)
{  c.Front = c.End = NULL;  }
```

```
int isEmpty(coada c)
{
    if (c.Front == NULL ) return 1;
    return 0;
}
```

```
int prima_coada_vida(coada aux[], int k)
{
    for(int i=0; i<k; i++)
        if (isEmpty(aux[i])) return i;
    return -1;
}
```

Aplicatii

Stive si cozi

Detalii de implementare in C++

6. Depou feroviar

Rutine ajutatoare

```
int pozitie_minima(coada aux[], int k)
{
    int poz_min = 0;
    while(poz_min < k && isEmpty(aux[poz_min])) poz_min++;
    if (poz_min == k) return -1;

    for(int i = poz_min+1; i<k; i++)
        if ((!(isEmpty(aux[i]))) && aux[poz_min].Front->info > aux[i].Front->info)
            poz_min = i;
    return poz_min;
}
```

Aplicatii

Stive si cozi

Detalii de implementare in C++

6. Depou feroviar

Pas 2

```
cout<<"din coada initiala ";
        pop(Input,val);
        if (val == pozOutput)
        {
            int p = prima_coada_vida(aux,k);
            if (p!=-1)
            {
                cout<<"adaugata in coada "<<p<<"--->";
                push(aux[p],val);
                pop(aux[p],val);
                cout<<"adaugata in coada finala\n";
                push(Output,val);
                pozOutput++;
            }
            else posibil = 0;
        }
```

Aplicatii

Stive si cozi

Detalii de implementare in C++

6. Depou feroviar

Pas 3

```
else
   {
      for(i = 0; i<k; i++)
         if (isEmpty(aux[i]) || (!(isEmpty(aux[i])) && val>aux[i].End->info))
            {
               cout<<"adaugata in coada "<<i<<"\n";
               push(aux[i],val);
               break;
            }
       if (i==k) posibil = 0;
    }
```

Aplicatii

Stive si cozi

Detalii de implementare in C++

6. Depou feroviar

Pas 4

```
void golire_cozi(coada aux[],int k, coada& Output)
{
    while(pozitie_minima(aux,k)!=-1)
    {
        int x;
        cout<<"din coada auxiliara "<<pozitie_minima(aux,k)<<" ";
        pop(aux[pozitie_minima(aux,k)],x);
        cout<<" adaugata in coada finala\n";
        push(Output, x);
        pozOutput++;
    }
}
```

Aplicatii

Liste simplu inlantuite

7. Reprezentarea vectorilor rari

Un vector rar:

- are cel putin 80% dintre elemente = 0.

- reprezentare eficienta → liste simplu inlantuite alocate dinamic

- fiecare nod din lista retine:

- **valoarea**

- **indicele din vector**

Cerinte: adunarea, respectiv, produsul scalar a doi vectori rari.

Aplicatii

Liste simplu inlantuite

7. Reprezentarea vectorilor rari

V1 si V2 – vectori rari

V1

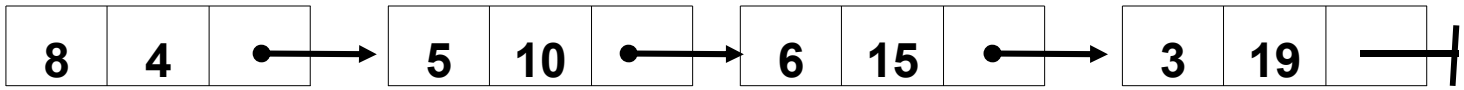
0	0	0	8	0	0	0	0	0	5	0	0	0	0	6	0	0	0	3	0

V2

0	0	0	0	0	0	7	0	0	4	0	0	0	0	0	0	0	0	2	0

Transformarea in liste simplu inlantuite

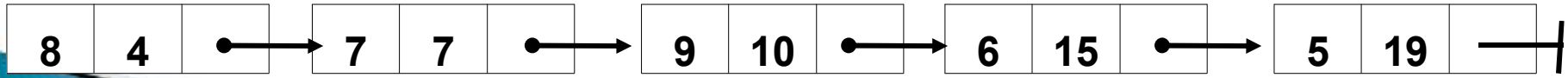
L1



L2

Produsul scalar = 5 x 4 + 3 x 2 = 26

L1+L2



Aplicatii

Liste simplu inlantuite

7. Reprezentarea vectorilor rari

Crearea unui vector rar

```
void inserare(nod *&prim, nod *&ultim, int a, int b)
{   nod *q = new nod;
    q->val=a;    q->poz=b; q->urm=NULL;

if(prim==NULL)
{     prim = q;
      ultim = prim;}
else
{     ultim -> urm = q;
      ultim = q; }
}
```

```
struct nod
{
    int poz, val;
nod*urm;
};
```

```
void creare_vector(int &n, nod *&p, nod *&u)
{ int i,a,b;
cin>>n;
for(i=1;i<=n;i++)
    {cin>>a>>b;
    inserare(p, u, a, b);
    }
}
```

Aplicatii

Liste simplu inlantuite

7. Reprezentarea vectorilor rari

Suma a doi vectori rari

```
void suma (nod *prim1, nod *prim2, nod *&prim3, nod *&ultim3)
{
    nod *p1, *p2;
    for (p1 = prim1; p1!= NULL; p1 = p1 -> urm)
        inserare(prim3, ultim3, p1 -> val, p1 -> poz);

    for (p2 = prim2; p2!= NULL; p2 = p2 -> urm)
    {   int ok = 0;
        for (p1 = prim3; p1!= NULL; p1 = p1 -> urm)
            if (p2 -> poz == p1 -> poz) {p1 -> val += p2 -> val; ok = 1;}
        if (ok == 0) inserare(prim3, ultim3, p2 -> val, p2 -> poz);
        }
}
```

Aplicatii

Liste simplu inlantuite

7. Reprezentarea vectorilor rari

Produsul scalar a 2 vectori rari

```
int prod_scalar(nod *prim1, nod *prim2)
{   int prod = 0; nod *p1, *p2;

    for (p2 = prim2; p2!= NULL; p2 = p2 -> urm)
       for (p1 = prim1; p1!= NULL; p1 = p1 -> urm)
             if (p2 -> poz == p1 -> poz) prod += p1 -> val * p2 -> val;
return prod;
}
```

Aplicatii

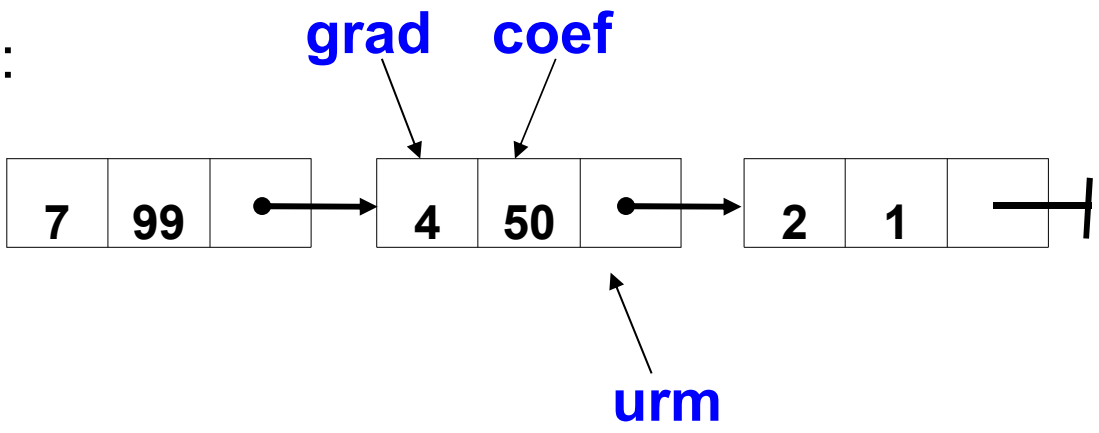
Liste simplu inlantuite

8. Reprezentarea polinoamelor rare

- fiecare nod din lista retine:

- **gradul**

- **coeficientul**



$7x^{99} + 4x^{50} + 2x$

Cerinte: - evaluarea intr-un punct

- produsul a doua polinoame rare.

Reprezentare eficienta → liste simplu inlantuite alocate dinamic

Aplicatii

Liste simplu inlantuite

8. Reprezentarea polinoamelor rare

Inserarea unui nou monom

```
struct nod
{
      float coef;
      int exp;
nod*urm;
};
```

```
float eval(nod *prim, int x)
{
// evaluarea unui polinom intr-un punct
    float prod = 0;
nod *p;

    for (p = prim; p!= NULL; p = p -> urm)
        prod += p -> coef * pow(x,p -> exp);
return prod;
}
```

Aplicatii

Liste simplu inlantuite

8. Reprezentarea polinoamelor rare

Crearea unui polinom rar

```
void inserare(nod *&prim, nod *&ultim, int e, float c)
{   nod *q = new nod;
    q->exp = e; q->coef = c;     q->urm=NULL;

if(prim==NULL)
{     prim = q;
      ultim = prim;}
else
{     ultim -> urm = q;
      ultim = q; }
}
```

```
struct nod
{
      float coef;
      int exp;
nod*urm;
};
```

```
void citire(nod* &prim, nod* &ultim, int &n)
{   cout<<"Dati numarul de monoame: ";
    cin>>n;
    for(int i = 0; i<n; i++)
    {   cout<<"Dati coeficientul si gradul monomului: ";
        int e; float c;  cin>>c>>e;
        inserare(prim,ultim,e,c);
    }
}
```

Aplicatii

Liste simplu inlantuite

8. Reprezentarea polinoamelor rare

Evaluarea unui polinom intr-un punct

```
struct nod
{
      float coef;
      int exp;
nod*urm;
};
```

```
float eval(nod *prim, int x)
{
// evaluarea unui polinom intr-un punct
    float prod = 0;
nod *p;

    for (p = prim; p!= NULL; p = p -> urm)
        prod += p -> coef * pow(x,p -> exp);
return prod;
}
```

Aplicatii

Liste simplu inlantuite

8. Reprezentarea polinoamelor rare

Produsul a 2 polinoame

```
void produs(nod *prim1, nod *prim2, nod *&prim3, nod *&ultim3)
{   nod *p1, *p2;
    for (p1 = prim1; p1!= NULL; p1 = p1 -> urm)
        for (p2 = prim2; p2!= NULL; p2 = p2 -> urm)
        {   int a = p1 -> exp + p2 -> exp;
            float b = p1 -> coef * p2 -> coef;
            int ok = 0;
            for (nod* p3 = prim3; p3!= NULL; p3 = p3 -> urm)
                if (p3 -> exp == a)  { p3 -> coef += b; ok =1; }
            if (ok == 0)
                inserare(prim3, ultim3, a,b);
        }
}
```

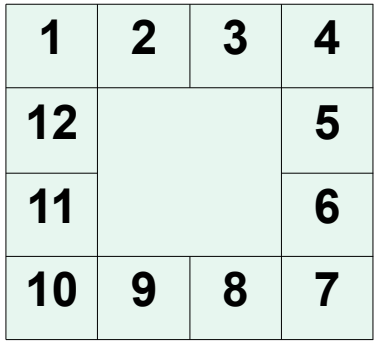
Aplicatii

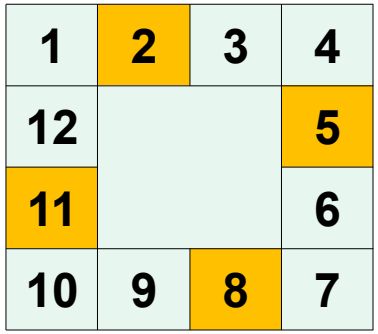
Cozi circulare

9. Josephus

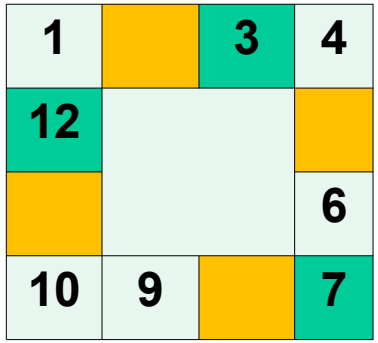
- n copii asezati in cerc sunt numarati din m in m plecand de la copilul k.
-fiecare al m – lea copil numarat iese din cerc.
-Afisare ordine iesire copii din cerc

n = 12
m = 3
k = 2;

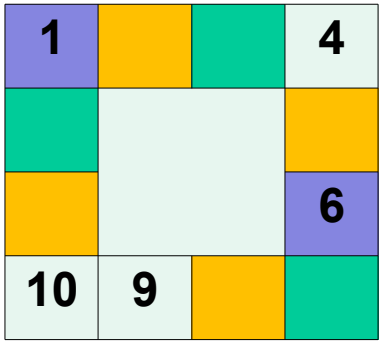




Afis: 2, 5, 8, 11

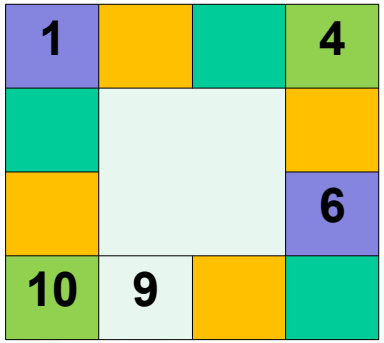


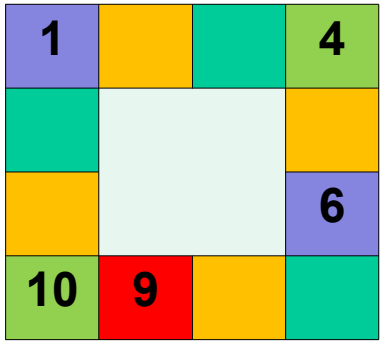
Afis: 3, 7, 12



Afis: 6, 1

Afis: 10, 4





Ordine: 2,5,8,11,3,7,12,6,1,10,4,9

Aplicatii

Cozi circulare

9. Josephus

Declarare si crearea configuratiei initiale

```
struct nod
{
    int info;
    nod *urm;
};
```

```
void inserare(nod *&prim, nod *&ultim, int x)
{
nod * p = new nod;  p -> info = x;  p -> urm = NULL;

    if (prim == NULL)     {  prim = ultim = p;  }
    else  {  ultim -> urm = p;      ultim = p;    }
}

// creare
for(i=1; i<=n; i++)
        inserare(prim,ultim,i);
    ultim->urm = prim;
```

Aplicatii

Cozi circulare

9. Josephus

Afisarea si stergerea elementului de pe pozitia de start

```
if (k == 1)
   {
      cout<<prim->info<<" ";
      nod *t = prim;
      prim = prim->urm;
      ultim->urm = prim;
      delete t;
      p = prim;
   }
```

```
else
   {
      p = prim;
      for (i = 1; i <= k-1; i++)
         { q = p; p = p->urm; }

      q->urm = p->urm;
      cout<<p->info<<" ";
      delete p;
      p = q->urm;
   }
```

Aplicatii

Cozi circulare

9. Josephus

Afisarea informatiei si stergerea elementelor din m in m

```
while (p->urm != p)
        {
            for (i = 1; i < m; i++)
            {
                q = p;
                p = p->urm;
            }
            q->urm = p->urm;
            cout<<p->info<<" ";
            delete p;
            p = q->urm;
        }
```

Structuri liniare (Liste. Stive. Cozi)

Concluzii

S-au recapitulat notiunile urmatoare:

- Structuri liniare în alocare statica (vectori) si dinamica (liste înlantuite)
- Structuri liniare fara restrictii de intrare/iesire
- Structuri liniare cu restrictii de intrare/iesire (stive si cozi)

Succes!