Facultatea de Matematica si Informatica
Universitatea Bucuresti

Structuri liniare

Liste. Stive. Cozi

- Inserare, cautare, stergere -

Lectii de pregatire pentru Admitere

04 / 03 / 2017

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.

Structuri liniare - Liste

Liste liniare alocate secvential

Nodurile in pozitii succesive de memorie

Avantaj: acces direct la orice nod

Dezavantaj: multe deplasari la operatiile de inserare si stergere

Structuri liniare - Liste

Liste liniare alocate secvential

Nodurile in pozitii succesive de memorie

Avantaj: acces direct la orice nod

Dezavantaj: multe deplasari la operatiile de inserare si stergere

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	#

Structuri liniare - Liste

Liste liniare alocate secvential

- 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 = true) 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;

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

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

Numarul de comparatii: n + 1 + 1

Liste liniare alocate secvential

Cautare binara (! pe vector ordonat)

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;
               else poz = -1;
```

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
              else poz:=0;
```

Liste liniare alocate secvential

Cautare binara (! pe vector ordonat)

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

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

Liste liniare alocate secvential

C / C++

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

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

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**)

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

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

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

Aplicatii

0. Aplicație standard – 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.

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

   i = k;          cout<<a[i]<<" ";
   for (int j = i; j<n; j++)   a[j] = a[j+1];   // eliminarea elementului de pe pozitia k
   n--;

   while (n>0)
   {
      i = i+ m-1;
      if (i%n==0) i = n;  // situatie speciala in cazul numerotarii 1..n
      else if (i > n) i = i % n;
      cout<<a[i]<<" ";
      for (int j = i; j<n; j++)   a[j] = a[j+1]; // eliminarea de pe pozitia i + m
      n--;
   }
```

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

Aplicatii 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.

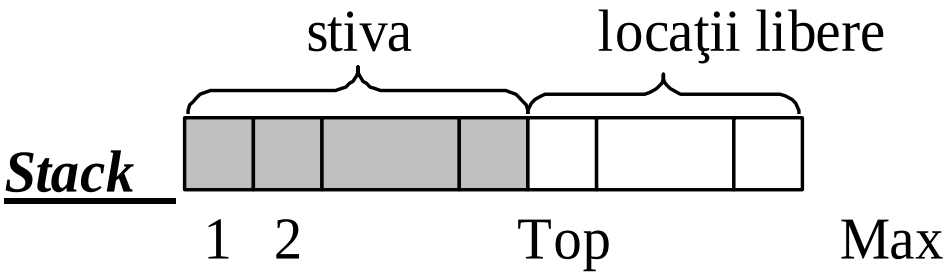
Structuri lineare cu restrictii la i/o: Stiva (LIFO)

- 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

Stiva in alocare statica

C / C++ | **Pascal**

Declarare



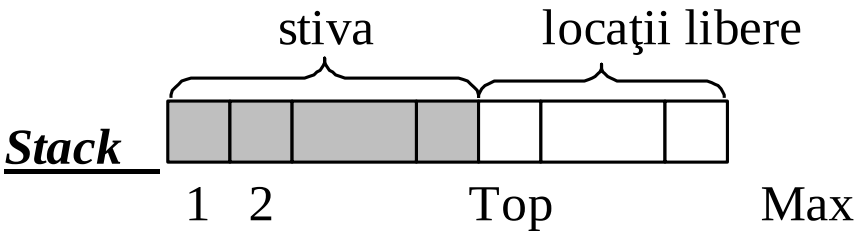

```
#define MAX 100

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

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

MAX := 100;
```

Stiva in alocare statica

C / C++ | **Pascal**

Inserare



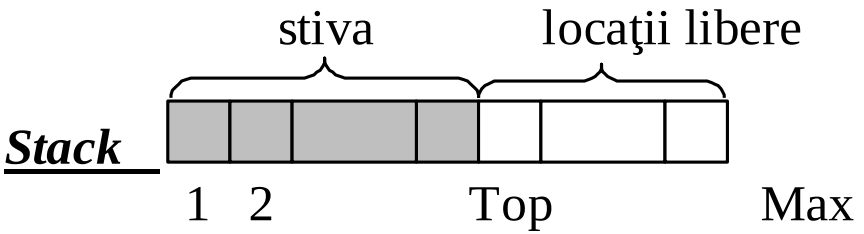

```
void Push (int Val)
{
     if (Top == Max)
                // Overflow
     else
          {
            Top++;
            Stack[Top] = Val;
          }
}
```

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

Stiva in alocare statica

C / C++ | **Pascal**

Stergere




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

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

Aplicatii

2. Exemplificarea mecanismului RECURSIVITATII și ordinea efectuarii operatiilor

```
int f (int a)
{
    cout<<"In functie a = " << a <<" la adresa = " << &a <<endl;
    if (a == 1) return 1;
        else return a + f(a-1);
}

int main()
{   int a = 5;
    cout<<"In Main: &a = " << &a <<"; f(a) = "<<f(a);
}
```

Aplicatii

2. Exemplificarea mecanismului RECURSIVITATII și ordinea efectuarii operatiilor

La executie:

In functie a = 5 la adresa = 0x7fff2580e7bc
In functie a = 4 la adresa = 0x7fff2580e78c
In functie a = 3 la adresa = 0x7fff2580e75c
In functie a = 2 la adresa = 0x7fff2580e72c
In functie a = 1 la adresa = 0x7fff2580e6fc
In Main: a = 5 si f(a) = 15

Obs! Ordinea efectuarii apelurilor in afisarea pe ecran!

cout<<"In Main: &a = " << &a <<"; f(a) = "<<f(a);

Aplicatii

3. Parantezarea corecta

Dat un sir $s = s_1 s_2 \ldots s_n$ de caractere '(' si ')' sa se verifice daca acest sir este 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 parantezat corect, se va indica pozitia primei paranteze ')' care nu are corespondent.

Aplicatii

3. Parantezarea corecta

C / C++

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

Pascal

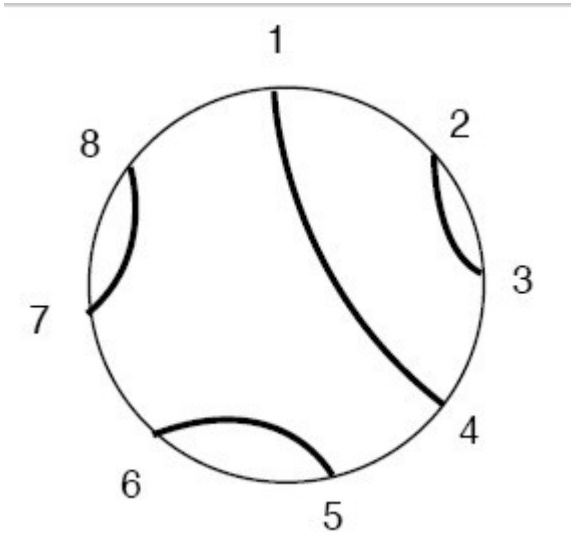
```
var ok:boolean; ok:=true;
for i:=0 to length(s) do
     begin
     if(empty(S)=true) then
          // Stiva vida
          begin
           if (v[i] = ')' ) then
                begin
                 ok=false; break;
                end;
          push(v[i],S);
          end
          else
   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

Aplicatii

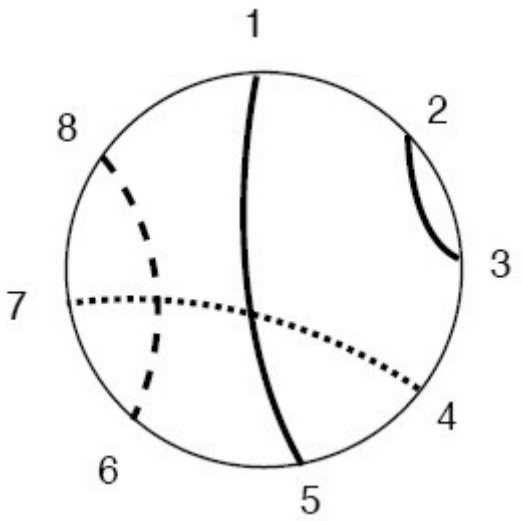
4. Conectarea pinilor

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.



Configuratie valida



Configuratie invalida

Aplicatii

C / C++

```
// citire vector pereche

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

Pascal

```
{ citire vector pereche}

for i:=0 to length(s) do
      begin
      if(empty(S)=true) then
            // Stiva vida
            push(pereche[i],S);
      else
            if(pereche[i] = peek(S)) then
                  pop(S);
            else
                  push(pereche[i],S);
      end;
if (empty(S) = true)
      write('Configuratie valida')
      else
            write('Configuratie invalida');
```

Aplicatii

5. Evaluarea unei expresii în notatie postfixata

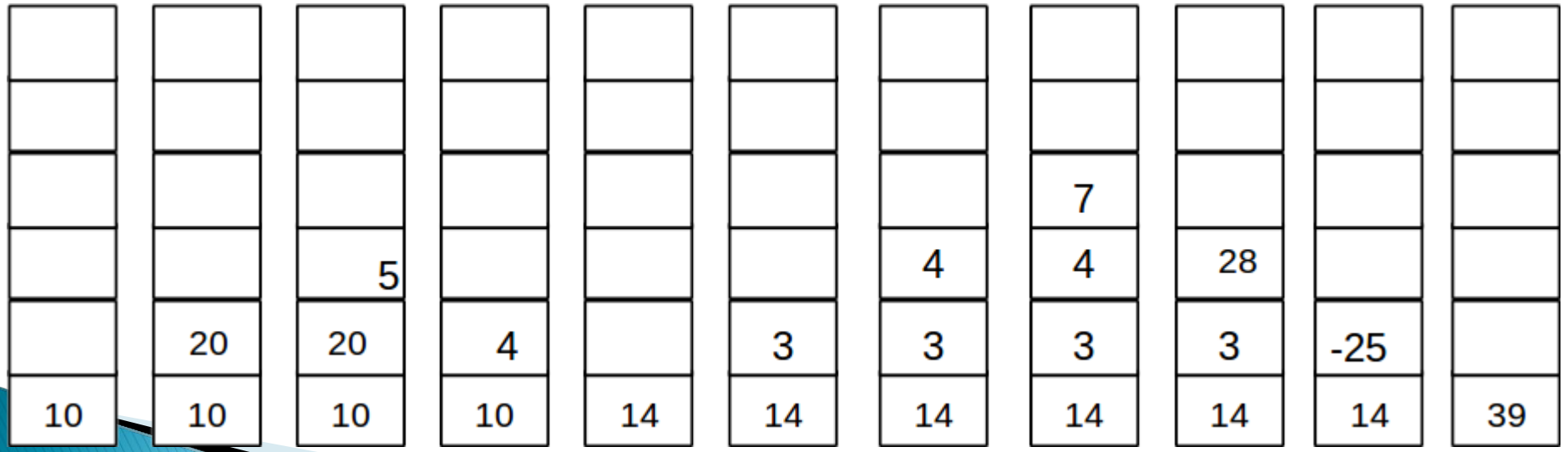
Exemplu

(10 + 20 / 5) - (3 - 4 * 7)

in notatie postfixata:

10 20 5 / + 3 4 7 * - -

Stiva, dupa fiecare pas



Aplicatii

Algoritm (rezolvare completa: fisier atasat)

Pas 0. - **postfix** - sirul de caractere introdus
Pas 1. - se considera adresa primului caracter // char *p = &postfix[0];
Pas 2. - se cauta primul caracter nenul (intre operanzi / operatori : ' ', sau '\t')

```
    while(*p)
       {
          if(isdigit(*p)) → Pas 3.
          else → Pas 4.
          p++;    // se trece la urmatorul caracter
  }
```

Pas 5. - rezultatul este singura valoare aflata in stiva // result = pop(&X);
Pas 3. - daca este numar se introduce in stiva
// numar intreg = cod ASCII caracter - 48 (codul caracterului '0')
Pas 4. - daca este operand, atunci se extrag din stiva ultimele valori inserate, se aplica operandul si noua valoare se reintroduce in stiva.

```
    /*    op1 = pop(&X);  op2 = pop(&X);
         - se aplica operandul
         push(&X,newnode);   // se reintroduce in stiva rezultatul operatiei  */
```

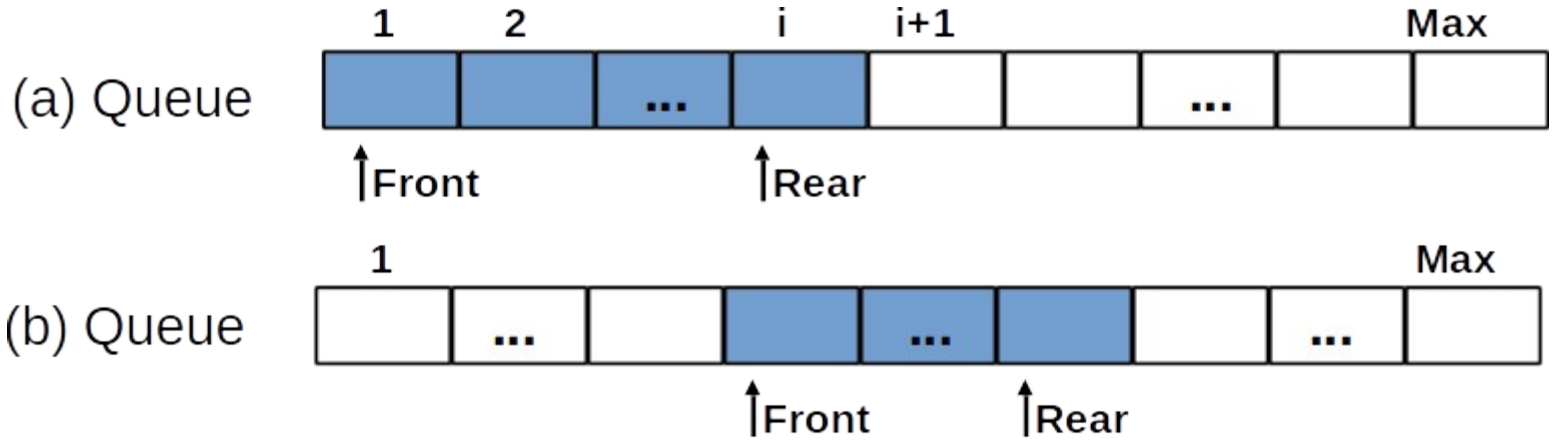
Structuri lineare cu restrictii la i/o: Coada (FIFO)

- FIFO (First In First Out): primul introdus este primul extras
- capat pt. Inserari: sfirsit (*Rear)*
- capat pt. stergeri: inceput (*Front)*
- *Insert (Val)* - inserarea
  - **Overflow (supradepasire)** - inserare in coada plina
- *Delete(X)* - stergerea/extragerea
  - **Underflow (subdepasire) -** extragere din coada goala

Coada in alocare statica

C / C++ | Pascal

Declarare




```
#define MAX 100

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

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

Coada in alocare statica

C / C++

Inserare

```
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

C / C++

Stergere

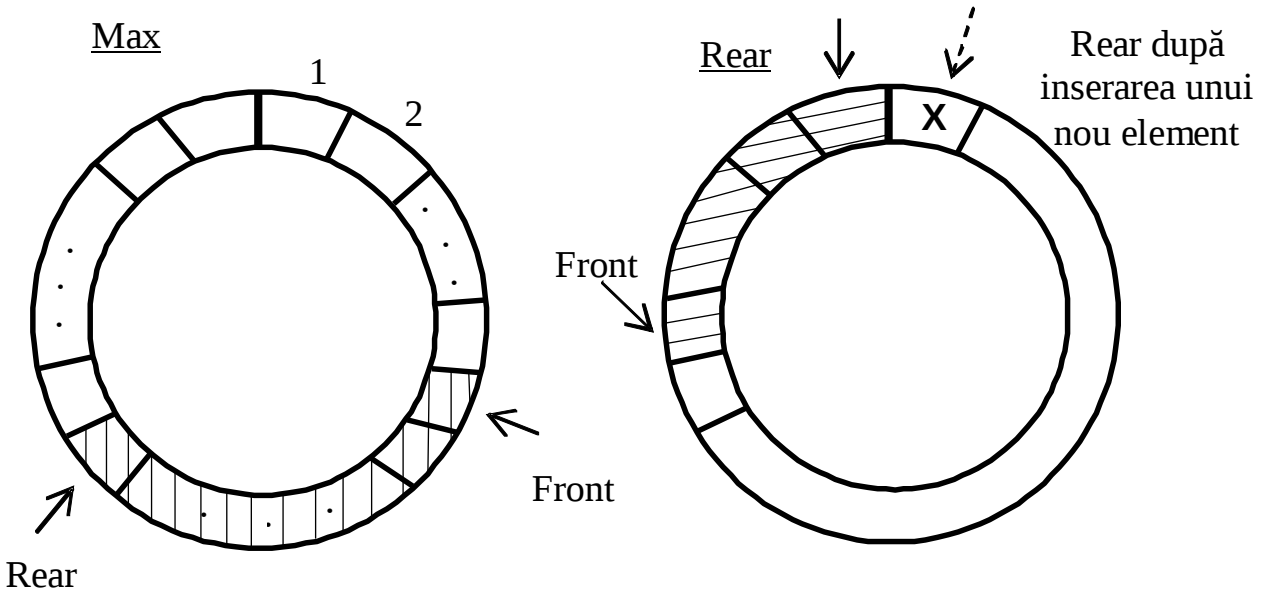
```
void Pop (int &X)
{
     if (Front == Max)
                  // Underflow
     else
          { if (Front == Rear)
                       Rear++;
               X = Queue[Front];
          Front++;}
}
```

Pascal

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

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.*

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*.

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:

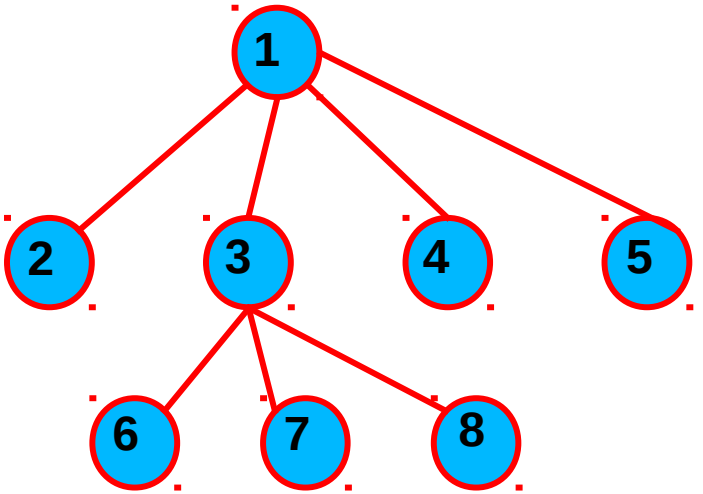
- inserările se pot face la un singur capăt şi extragerile la amândouă.

Aplicatii

(teoretic). MULTI - tasking, multi-user, ...

6. Parcurgerea unui arbore pe nivele (Breadth First)

BF (Latime):
1, 2, 3, 4, 5, 6, 7, 8



Aplicatii

6. Parcurgerea unui arbore pe nivele (Breadth First)

C / C++

```
int Front = 1,Rear = 1;    // Q[ ] - coada
// a – matricea de adiacenta
cin>>nod; // de inceput
Q[Front]=nod;
viz[nod]=1;

while(Front <= Rear)
   {
      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
       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

7. 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.

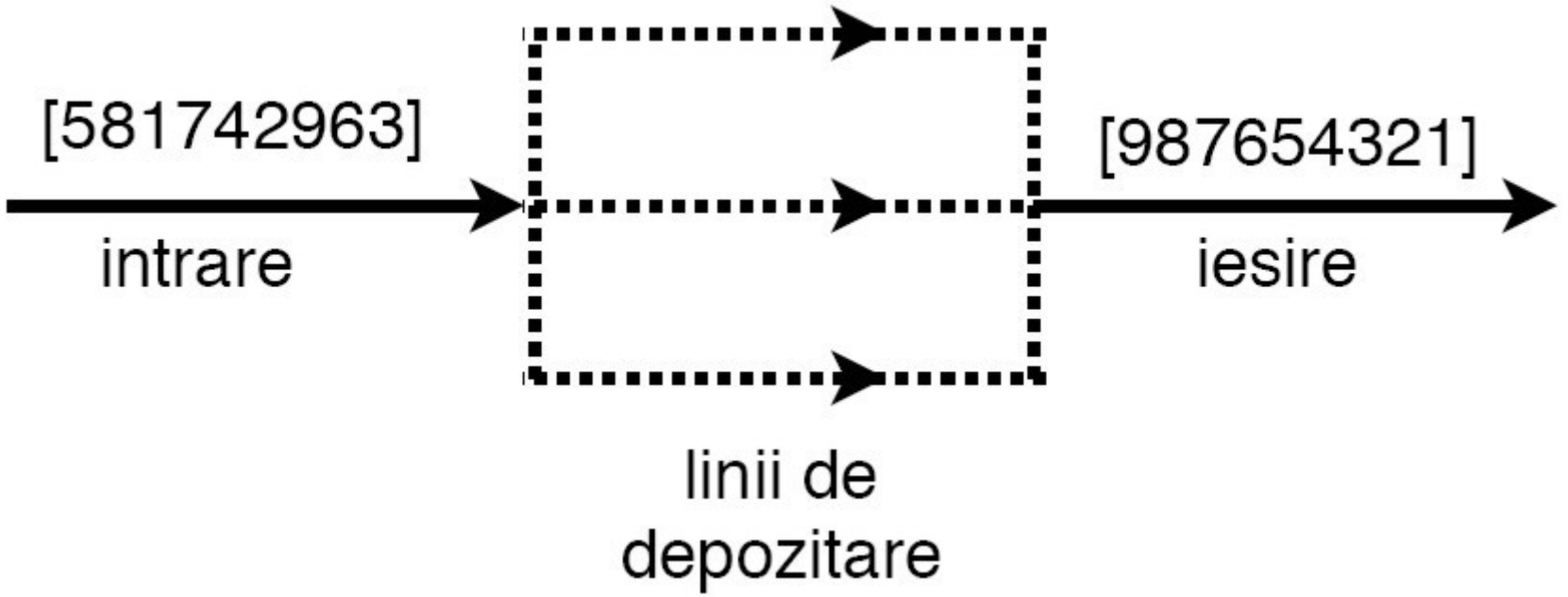
Sa se scrie un program care, dat un sir de vagoane pe linia de intrare (numerotate de la 1 la n si aranjate in orice ordine), descrie o strategie de a obtine pe linia de iesire sirul de vagoane n; n - 1;...; 2; 1, folosind liniile de depozitare.

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

Aplicatii

7. Depou feroviar

Exemplu de depou feroviar cu 3 linii de depozitare



Aplicatii

Algoritm (rezolvare partiala)

```
// Se extrage primul element din CI si se introduce pe prima linie auxiliara  CC[1]
j = 1; push(x,CC[j]);
for(i=2;i<=n;i++)
{     y = pop(CI)   // extrag elemente din CI
      k1 = 1;   // caut locul de inserat
      while(k1 <= j && CC[k1][Rear] > y) k1++; // am cozi ocupate si conditie
      if (k1 <= j) push(y,CC[k1]); // gasesc o coada nevida unde pot sa inserez y
      else {j++; push(y,CC[j]); // inserez y pe o coada noua
}
for (i=1; i <= n; i++)
   { int min,p,  k = 1;  // caut minimul varfurilor cozilor auxiliare nevide
     while (k<=j && CC[k] == NULL) k++;
     if (k <= j) {min = CC[k][val];
                  p = k;}    // se introduce in coada finala
     for(k1 = 1; k1 <= j; k1++)
      if (c[k1]!=NULL && CC[k1][val]<min)
            {min = CC[k1][val]; p = k1;}
     pop(CC[p]);
    }
```

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

C / C++

Declarare

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

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

Pascal

Declarare

```
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}
```

Liste liniare inlantuite alocate static

C / C++

Alocare

```
i = 0;
while ((oc[i] != 0) && (i<100))
        i++;

oc[i] = 1;
n++;
```

Eliberare

```
void elib (int x)
{
    oc[x] = 0;   n--;
}
```

Pascal

Alocare

```
i := 0;
while (oc[i]<>0) and (i<100) do
i := i+1;

oc[i] := 1;
n := n+1;
```

Eliberare

```
procedure elib(x:integer);
begin
    oc[x]:=0; n:=n-1;
end;
```

Liste liniare inlantuite alocate static

C / C++ | **Pascal**

Inserare

```
a[nou].urm = a[poz].urm;
a[poz].urm = nou;
a[nou].inf = val;
```

```
a[nou].urm := a[poz].urm;
a[poz].urm := nou;
a[nou].inf := val;
```

Inserare nod de indice nou dupa nodul de indice poz

Liste liniare inlantuite alocate static

C / C++ | **Pascal**

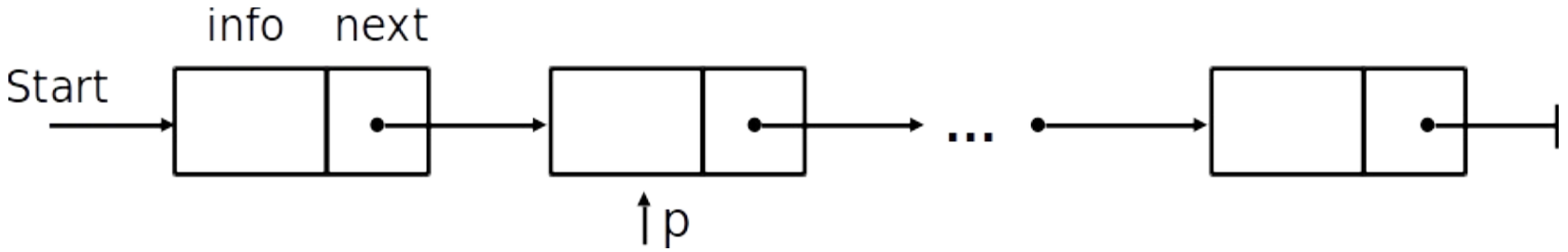
Stergere

```
n = a[poz].urm;
oc[n] = -1; // pozitie eliberata
a[poz].urm = a[n].urm;
```

```
n := a[poz].urm;
oc[n] := 0; {pozitie eliberata}
a[poz].urm := a[n].urm;
```

Stergerea nodului aflat dupa nodul de indice poz

Liste liniare inlantuite alocate dinamic



- 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, *next*.

Liste simplu inlantuite

C / C++

Declarare

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

Traversare

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

Pascal

Declarare

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

Traversare

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

Liste simplu inlantuite

Cautare

C / C++

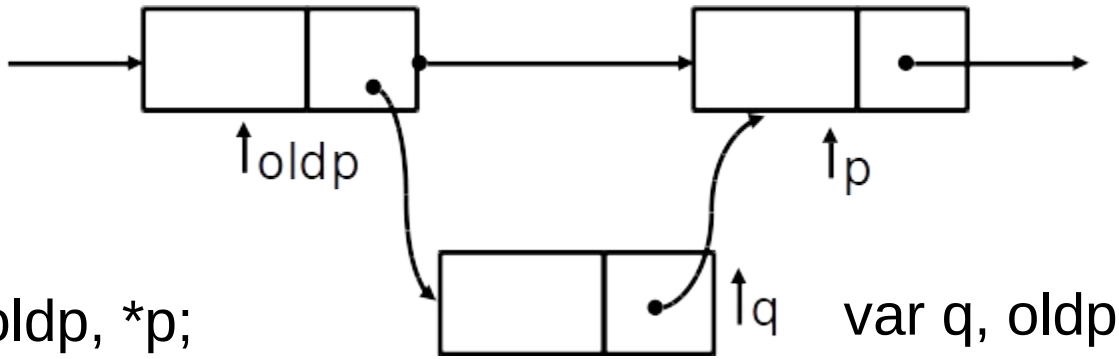
```
nod *p;
int x;

p = Start;
while (p != NULL  &&  x != p→info)
          p = p  →  next;

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

Pascal

```
var p: pnod;
int x;

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

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

Liste simplu inlantuite

Inserare



C / C++

```
nod *q, *oldp, *p;
q = new nod;
// prelucrare q → info;

q → next = p;

if (oldp != NULL)
     oldp → next = q;
else
     Start = q;
```

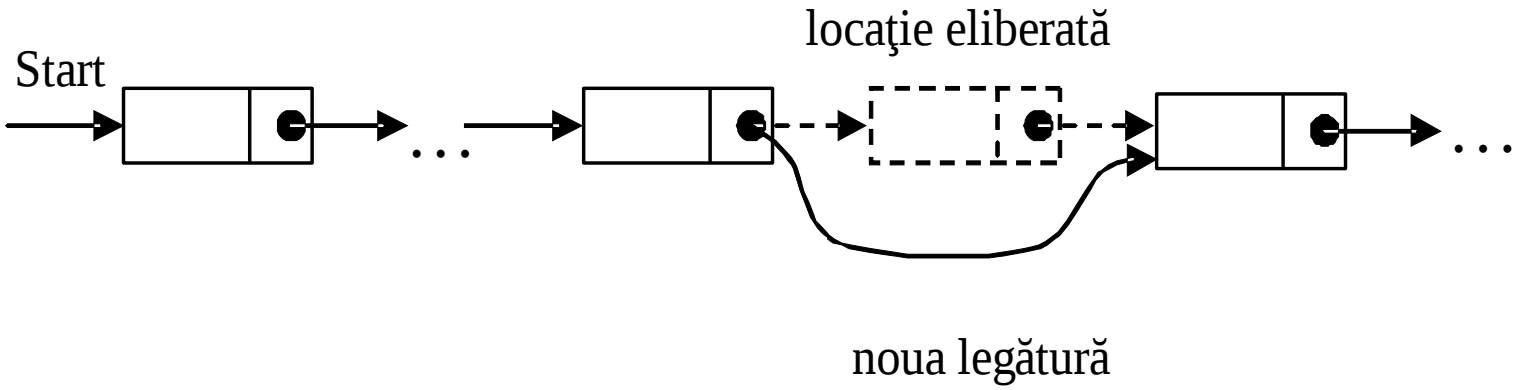
Pascal

```
var q, oldp, p: pnod;
new (q);
// prelucrare q^.info;

q^.next := p;

if (oldp <> nil) then
     oldp^.next := q
else
     Start := q;
```

Liste simplu inlantuite

C / C++ Pascal

Stergere



Refacerea structurii de lista simplu inlantuita pe nodurile ramase

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

Liste simplu inlantuite

Stergere

C / C++

```
nod *temp = p;

if (oldp != NULL)
      oldp → next = p → next;
else
      Start = p → next;
// prelucrare temp / temp → info

delete (temp);
```

Pascal

```
temp : pnod;
temp := p;

if (oldp <> nil) then
      oldp^.next := p^.next
else
      Start = p^.next;
{ prelucrare temp / temp^.info }

dispose (temp);
```

Aplicatii

8. Reprezentarea vectorilor rari

- are cel putin 80% dintre elemente egale cu0.

- 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

8. Reprezentarea vectorilor rari - Implementare

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

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

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

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

Aplicatii

8.Reprezentarea vectorilor rari - Implementare

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

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

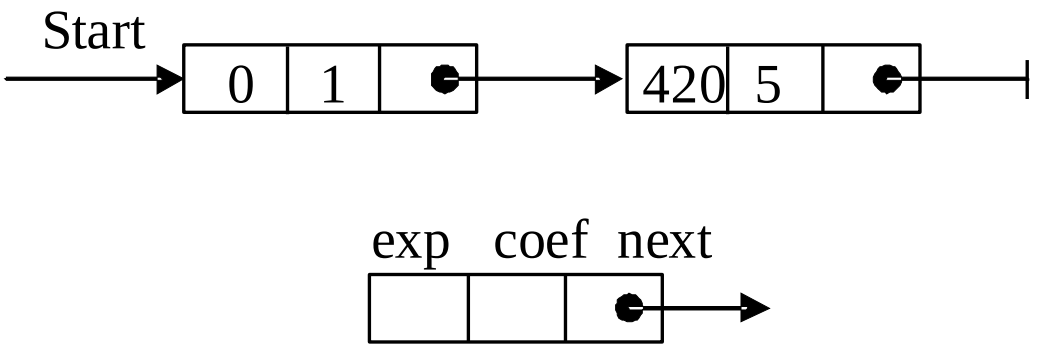
Aplicatii

8. Reprezentarea vectorilor rari - Implementare

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

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

Aplicatii

9. Reprezentarea polinoamelor rare



Cerinte: - evaluarea intr-un punct

- suma si produsul a doua polinoame.

Aplicatii

9. Reprezentarea polinoamelor rare - Implementare

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

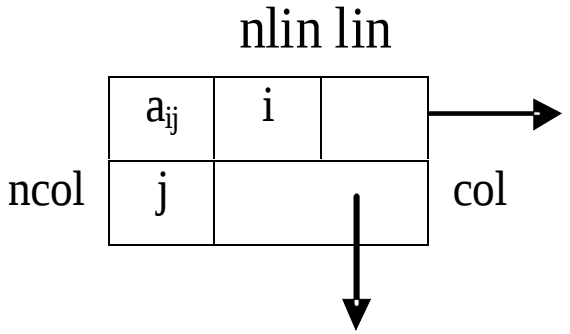
Aplicatii

9. Reprezentarea polinoamelor rare - Implementare

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

    for (nod *p = prim; p!= NULL; p = p -> next)
        prod += p2 -> coef * pow(x,p -> exp);
return prod;
}
```

Aplicatii

10. Reprezentarea matricelor rare

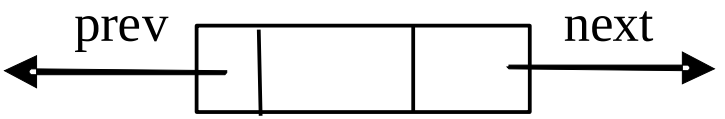


Cerinte: - suma a doua matrice

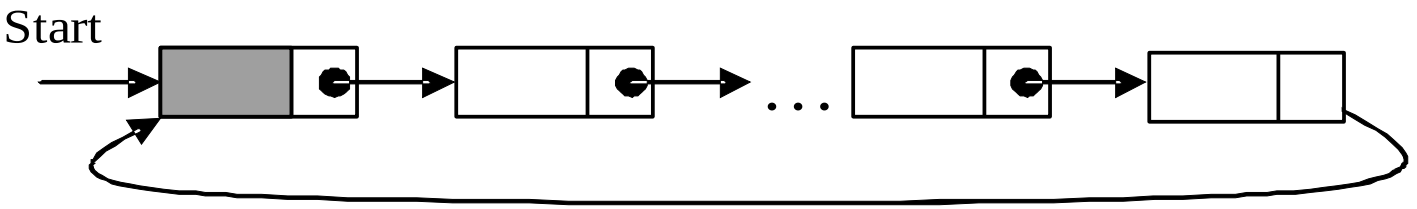
- determinantul si inversa unei matrice patratice

Alte tipuri de liste

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



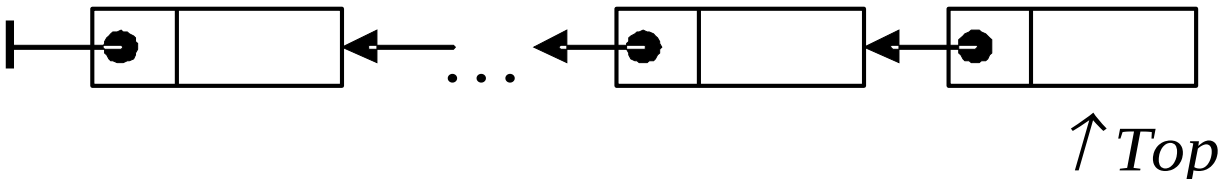
Nod într-o listă dublu înlănţuită.



Listă circulară cu nod marcaj.

Stiva in alocare dinamica

C / C++



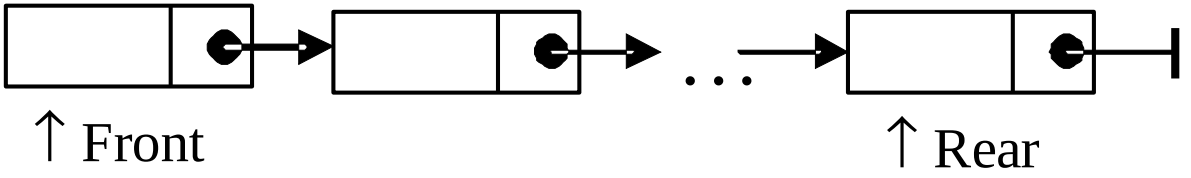

```
struct nod {
      int info;
      nod *next;
      };

nod * Top = NULL;
```

Pascal

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

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

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

Coada in alocare dinamica



Inserari – *Rear*

Stergeri - *Front*

Coada vidă: *Front = Rear = NULL*.

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

11. Aplicație standard pe cozi circulare – 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.

Aplicatii

11. Aplicație standard pe cozi circulare – Josephus

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

Declarare si citire lista circulara

```
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

11. Aplicație standard pe cozi circulare – Josephus

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

Afisarea si stergerea elementului de pe pozitia de inceput

```
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

11. Aplicație standard pe cozi circulare – 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;
        }
```

Aplicatii

12. Aplicație Cozi – subiect DL Info 2013 (subpunct b)

IV. Informatică. Fie n un număr natural nenul şi $m = 2^n$. Se dă vectorul $0, 1, 2, 3, \ldots, m, m+1$ şi p, cu $1 \leq p \leq m$. În acest vector, marcăm numerele 0, p şi $m+1$ ca fiind şterse. *Exemplu*: Pentru $n = 3$ şi $p = 5$, avem vectorul $X, 1, 2, 3, 4, X, 6, 7, 8, X$ unde elementele 0, 5 şi 9 sunt marcate cu X ca fiind şterse.

(a) Scrieţi un program care să şteargă toate elementele vectorului, în n paşi, în aşa fel încât la pasul k să se şteargă 2^{k-1} elemente, dintre cele neşterse până la pasul respectiv. Programul va afişa $m-1$ perechi de forma (k, q) unde q este unul dintre elementele vectorului, diferit de p, iar k este pasul la care a fost şters q. Programul scris trebuie să aibă complexitatea timp liniară în funcţie de m, adică numărul de instrucţiuni ale programului să fie aproximativ egal cu dimensiunea vectorului.

(b) Scrieţi un program similar cu cel de la punctul (a), dar cu următoarea condiţie suplimentară: după pasul k, între oricare două elemente deja şterse consecutive să nu fie o distanţă mai mare de 2^{n-k}, unde prin distanţa dintre i şi j se înţelege $|j - i|$. Calculaţi complexitatea timp în funcţie de n a programului pe care l-aţi scris. *Exemplu*: Considerăm vectorul $X, 1, 2, 3, 4, X, 6, 7, 8, X$. Printr-o posibilă strategie de ştergere, conţinutul vectorului după fiecare pas k este: $X, 1, X, 3, 4, X, 6, 7, 8, X$ (după pasul 1), $X, 1, X, 3, X, X, 6, X, 8, X$ (după pasul 2), respectiv $X, X, X, X, X, X, X, X, X, X$ (după pasul 3). Rezultatul afişat de program în acest caz este secvenţa (1,2),(2,4),(2,7),(3,1),(3,3),(3,6),(3,8).

Notă: Programele vor fi scrise într-unul dintre limbajele de programare studiate în liceu (Pascal,C,C++). Pentru fiecare soluţie se vor descrie informal detaliile algoritmului folosit şi ale implementării sub formă de program: semnificaţia variabilelor, a structurilor de date, a structurilor repetitive, a instrucţiunilor condiţionale.

Aplicatii

12. Aplicație Cozi – subiect DL Info 2013 (subpunct b)

```
16      //punctul b
17      int stanga[m],dreapta[m];
18      stanga[0] = 1;
19      dreapta[0] = m-1;
20      int startQ=0;
21      int finalQ=0;
22      for(k=1;k<=n;k++)
23      {
24          int elementeDeSters = (int) pow(2,k-1);
25          for(i=1;i<=elementeDeSters;i++)
26          {
27              //extrage din coada
28              int mijloc = (dreapta[startQ] + stanga[startQ])/2;
29              printf("(%d,%d)",k,mijloc+(mijloc>=p));
30              v[mijloc+(mijloc>=p)] = -1;
31              if (k<n)
32              {
33                  //insereaza in coada
34                  finalQ++;
35                  stanga[finalQ] = stanga[startQ];
36                  dreapta[finalQ] = mijloc-1;
37                  finalQ++;
38                  stanga[finalQ] = mijloc+1;
39                  dreapta[finalQ] = dreapta[startQ];
40              }
41                  startQ++;
42          }
43          printf("\n");
44          //afiseaza v
45          for(i=0;i<m+2;i++)
46           printf("%d ",v[i]);
47          printf("\n");
```