SDA (PC2)
Curs 8
Liste / Grafuri / Arbori

Iulian Năstac

Recapitulare

- Se consideră tipul utilizator:

```
typedef struct nod
        { declarații;
          struct nod *urm;
        } NOD;
```

Lista circulară simplu înlănţuită
(Recapitulare)

Lista simplu înlănţuită conţine:
- un nod care nu are succesor (pointează ultim);
- un nod care nu are predecesor (pointează prim).

Se ştie că într-o listă obişnuită: **ultim -> urm = 0**;

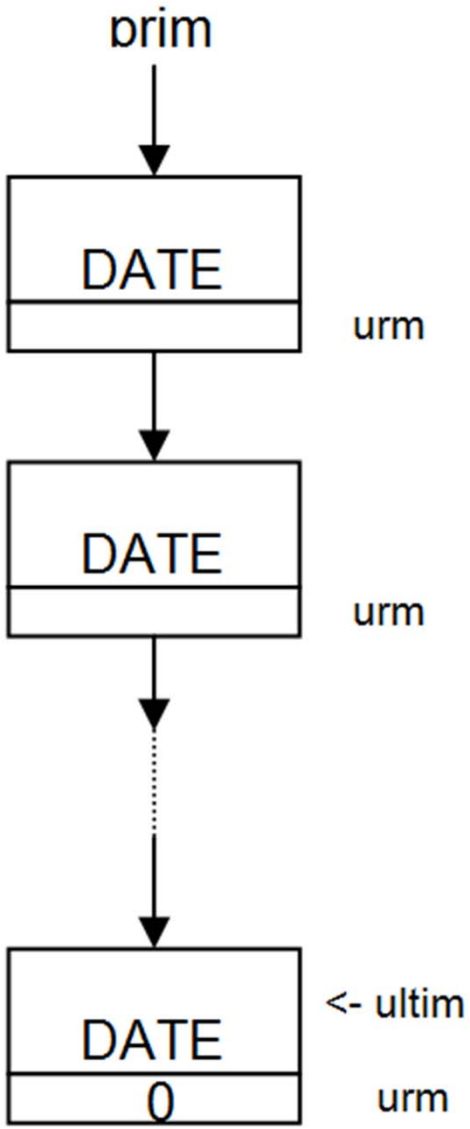
Dacă facem **ultim -> urm = prim**, atunci rezultă o **listă circulară simplu înlănţuită**.

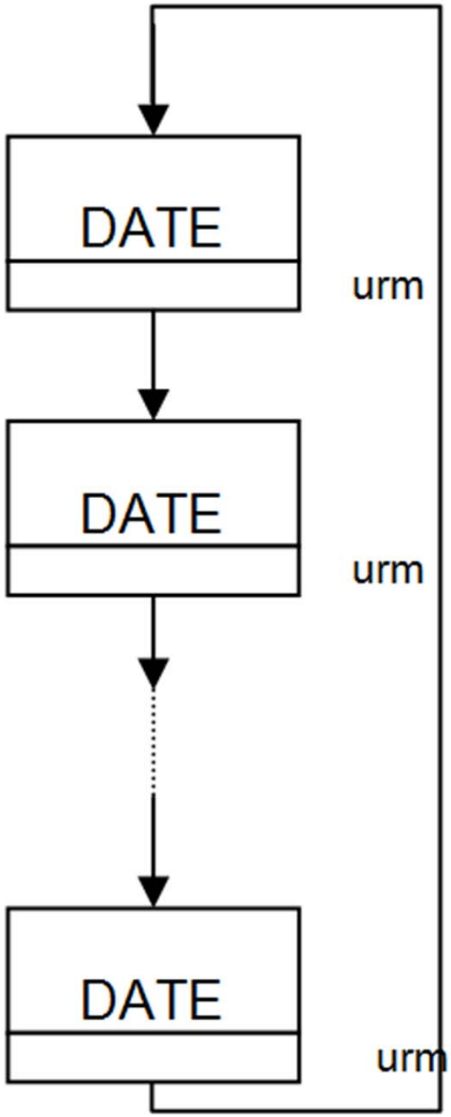
(Recapitulare)

Pentru gestiunea nodurilor, se foloseşte o variabilă globală care pointează spre un nod oarecare al listei *pcirc*:

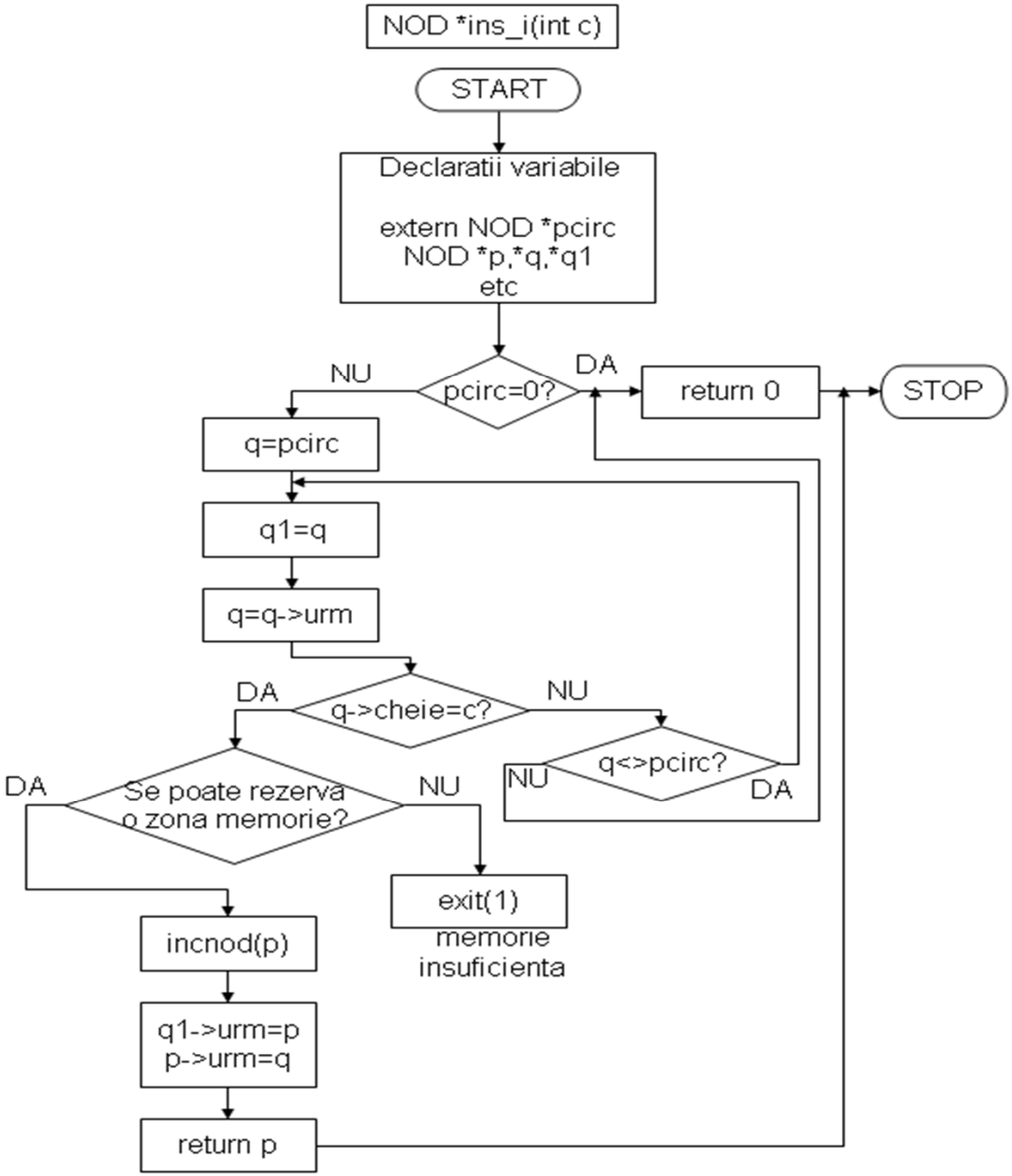
NOD *pcirc;

unde NOD este tipul comun al nodurilor listei.

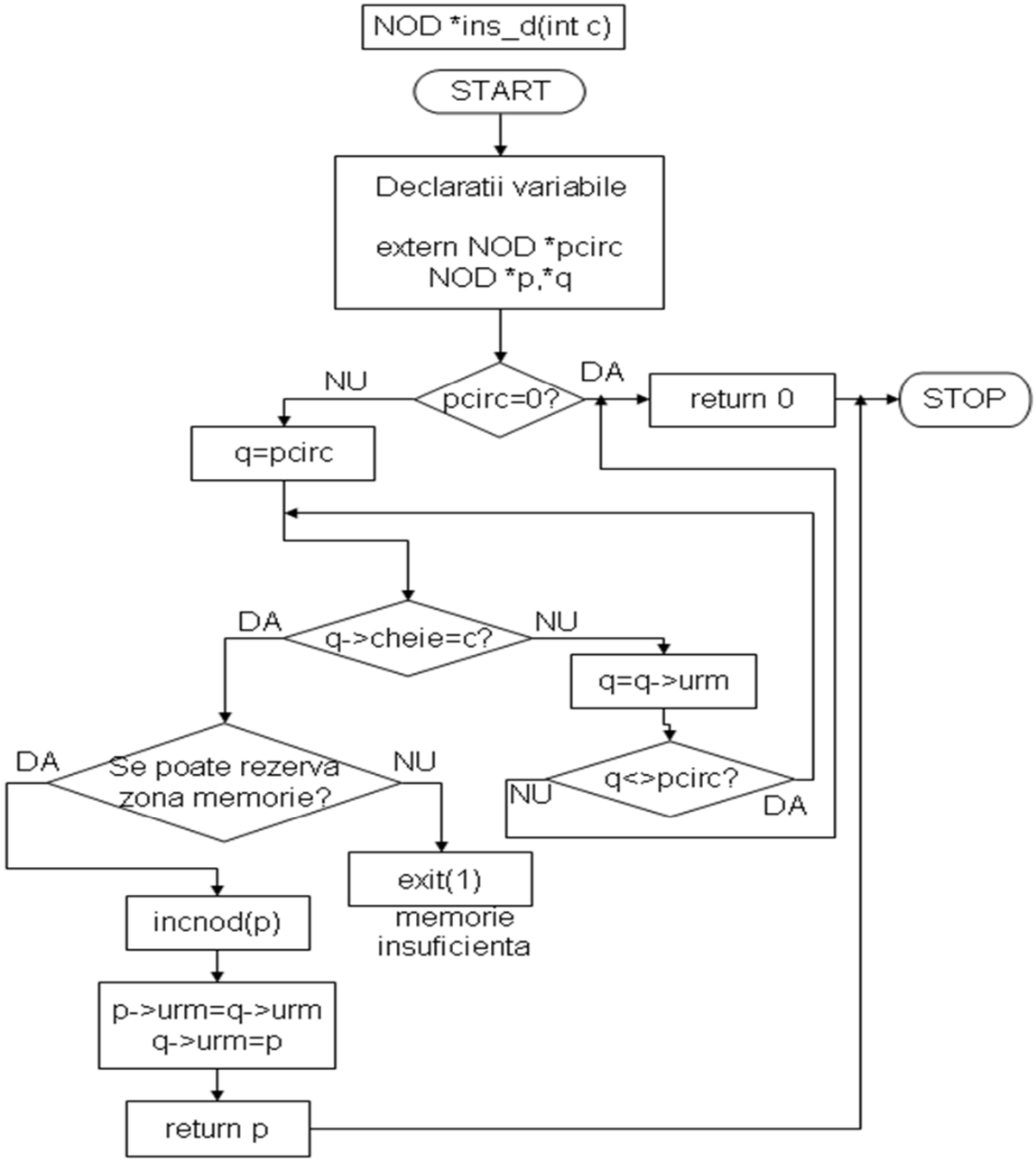

prim
DATE
urm
DATE
urm
DATE
0
<- ultim
urm


DATE
urm
DATE
urm
DATE
urm

Inserarea înaintea unui nod precizat printr-o cheie



Inserarea după un nod precizat printr-o cheie



+ Găsirea unui nod prin numărare!

Problema lui Josephus

(posibilă problemă de examen)

- Functia **eliminare** sterge din lista circulara indicata de pointerul **rep** cel de-al n-lea nod pornind de la **rep** si apoi intoarce noul pointer catre lista circulara care este precedentul celui sters.

- Pointerul **rep** joaca rolul lui **pcirc**.

Lista dublu înlănțuită

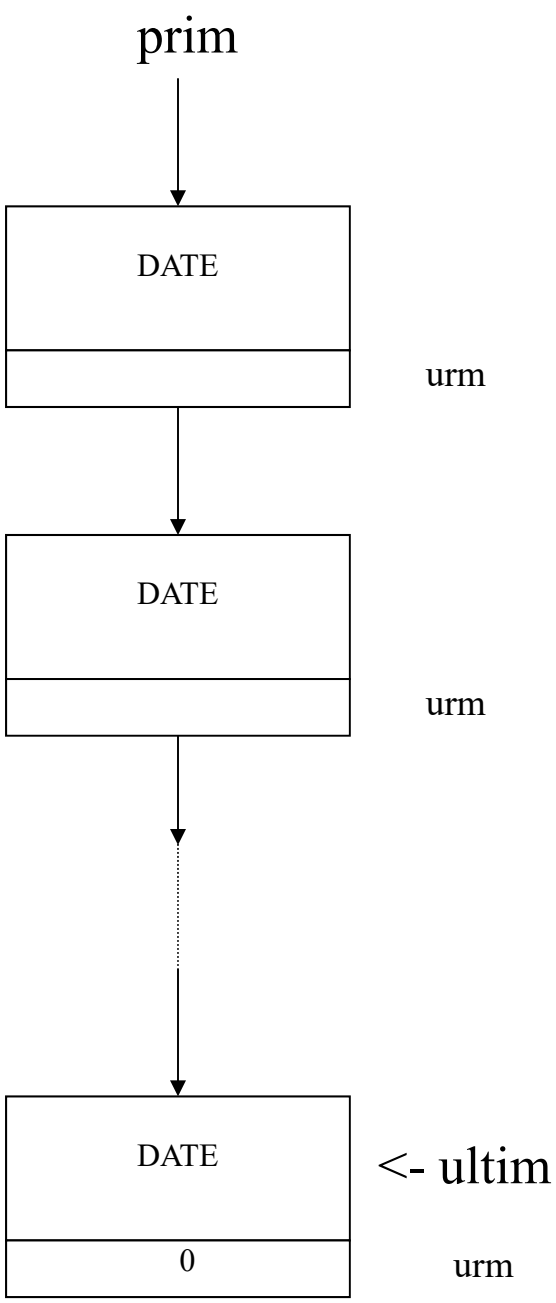
- Într-o astfel de listă fiecare nod conţine doi pointeri: unul spre nodul următor şi unul spre nodul precedent.

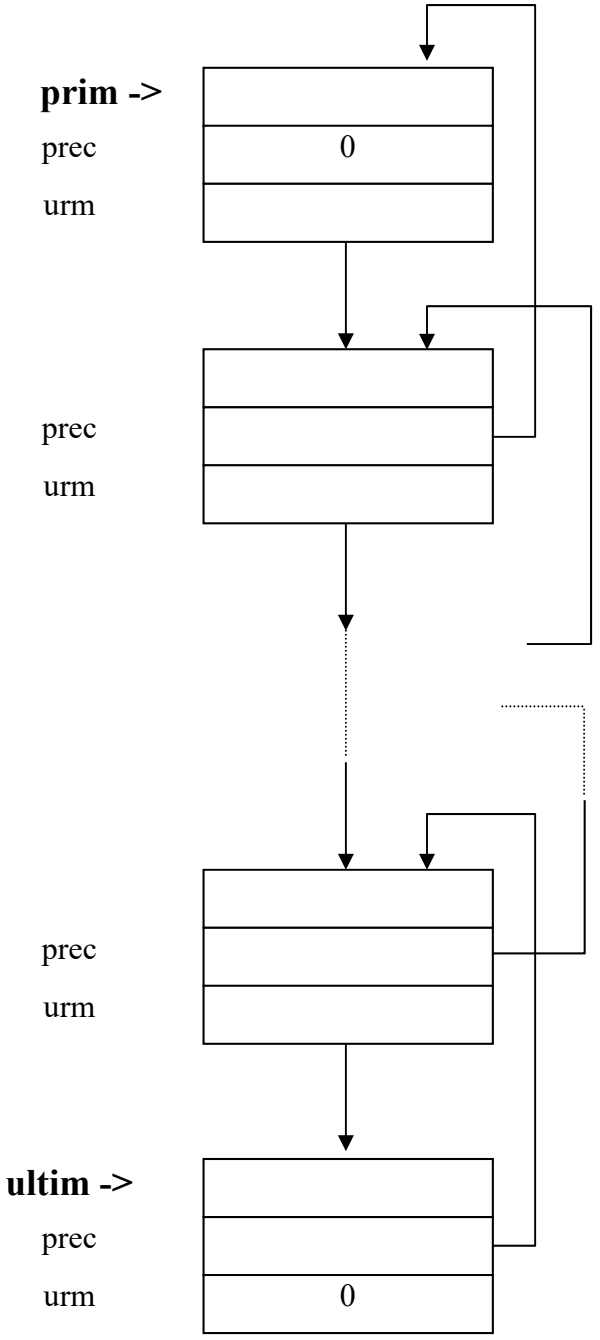
- Astfel fiecare nod al listei are un nod precedent definit prin pointerul **prec** şi un nod urmator definit prin pointerul **urm**.

Observaţii:

- Se consideră tipul utilizator:

```
typedef struct nod
        { declarații;
          struct nod *prec;
          struct nod *urm;
        } NOD;
```


prim
DATE
urm
DATE
urm
DATE
<- ultim
0
urm


prim ->
prec
0
urm
prec
urm
prec
urm
ultim ->
prec
urm
0

```
prim ->prec = 0;

ultim -> urm = 0;
```

Operaţii ce ţin de o listă dublu înlănţuită:

a) crearea listei înlănţuite;

b) accesul la un nod oarecare al listei;

c) inserarea unui nod într-o listă înlănţuită;

d) ştergerea unui nod dintr-o listă înlănţuită;

e) ştergerea unei liste înlănţuite.

Crearea unei liste dublu înlănțuite

1. La început lista este vidă:

prim = ultim = 0;

2. Se rezervă zonă de memorie (cu malloc) în memoria heap pentru nodul curent.

3. Dacă:

- Nu există date de încărcat:

elibnod(p);

- Există date de încărcat:

incnod(p);

4. Dacă lista:

- Era vidă:

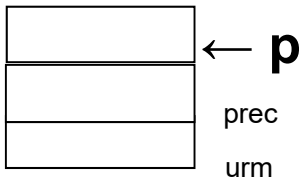
```
prim=ultim=p;
p->prec=p->urm=0;
```

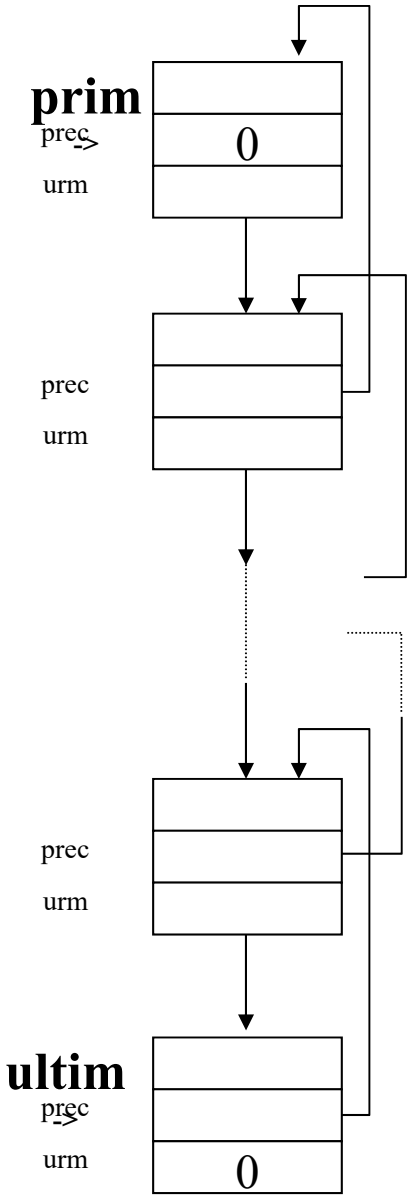
- Nu era vidă:

```
ultim->urm=p;
p->prec=ultim;
p->urm=0;
ultim=p;
```


5. Se reia procesul de la punctul 2

Adăugarea unui nou nod






```
ultim->urm = p;
p->prec = ultim;
p->urm = 0;
ultim = p;
```

Accesul la un nod al listei dublu înlănțuite

De preferat este cel printr-o cheie:

```
typedef struct nod
        {
                declaratii;
                int cheie;      /*practic orice tip*/
                declaratii;
                struct nod *prec;
                struct nod *urm;
        } NOD;
```

Inserarea unui nod într-o listă dublu înlănțuită

1. Inserarea înaintea primului nod al listei
2. Inserarea înaintea unui nod precizat printr-o cheie
3. Inserarea după un nod precizat printr-o cheie
4. Inserarea după ultimul nod al listei

Ştergerea unui nod dintr-o listă dublu înlănțuită

- Ștergerea primului nod
- Ștergerea unui nod precizat printr-o cheie
- Ștergerea ultimului nod

Aplicații ale listelor dublu înlănțuite

- Programe de mare securitate – refacerea rapidă a unei legături rupte (fară pierdere de noduri)
- Liste lungi în care căutarile pot porni de la ambele capete
- Programe bancare

Grafuri

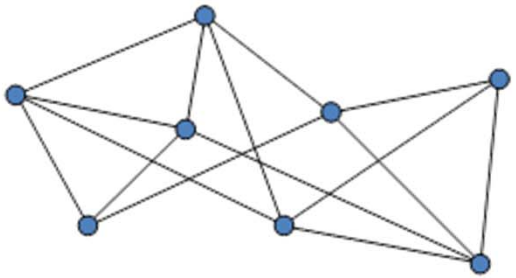
Definiție:

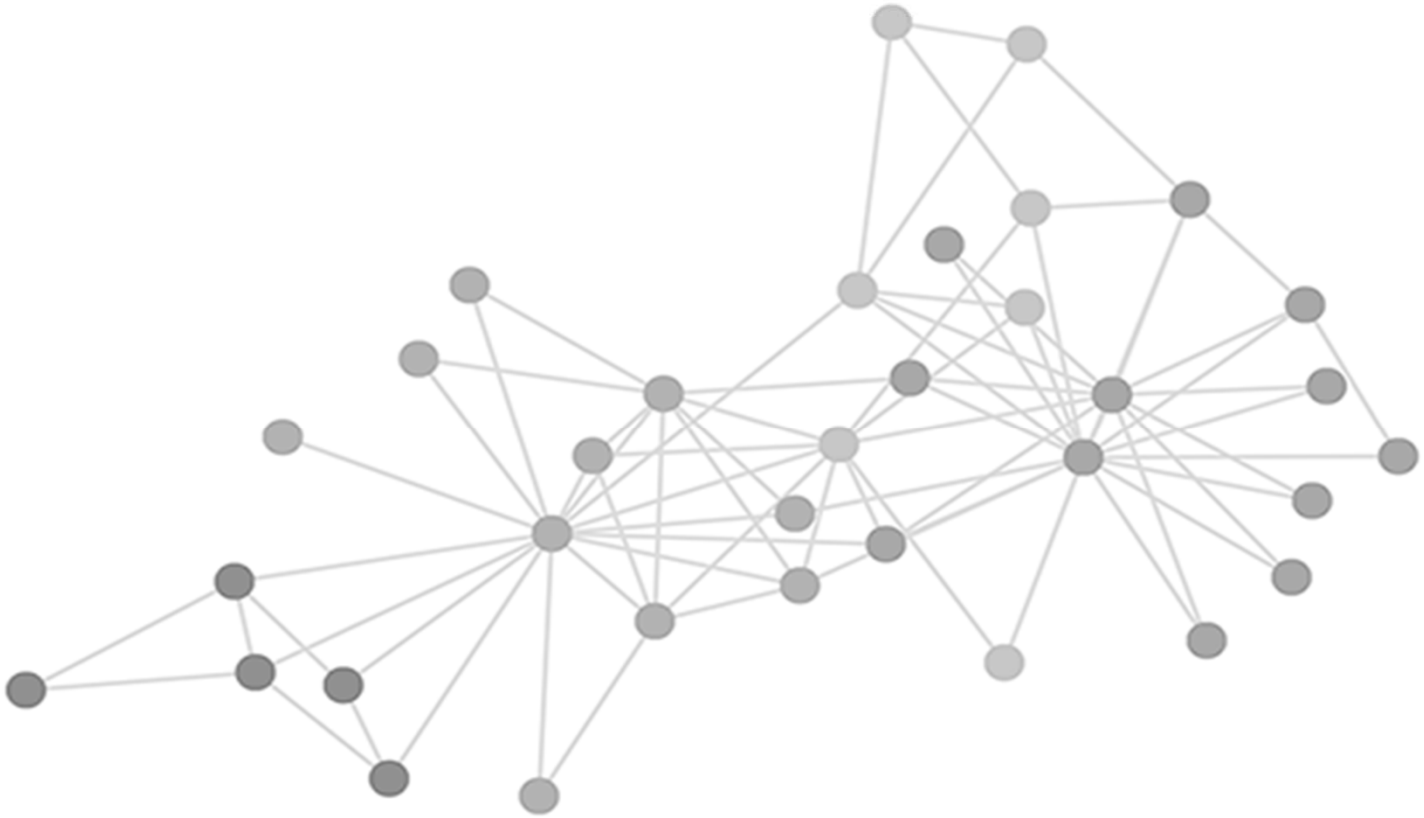
Un graf este o pereche

$$G = \langle V, M \rangle$$

unde V este o mulțime de vârfuri, iar $M \subseteq V \times V$ este o mulțime de muchii (laturi).

Exemple

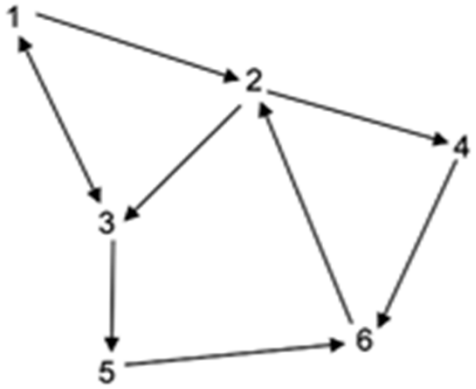




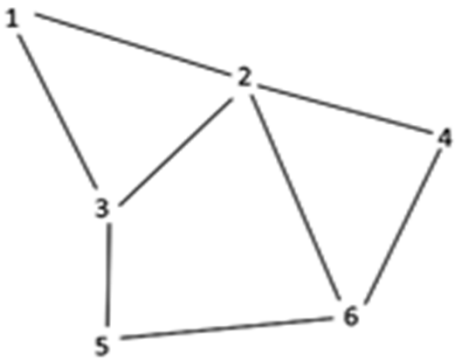
De obicei muchia de la vârful **a** la vârful **b** este notată cu:

- perechea ordonată **(a, b)** dacă graful este orientat;
- perechea neordonată **{a, b}** dacă graful este neorientat.

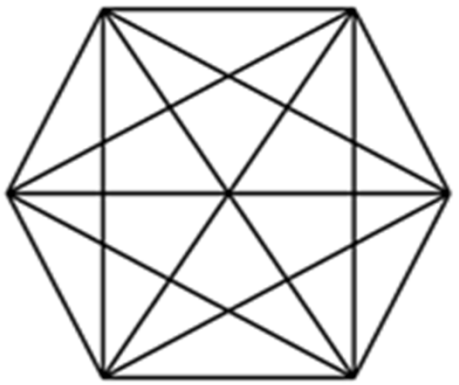
Observație: S-a presupus că ***a*** și ***b*** sunt diferite.



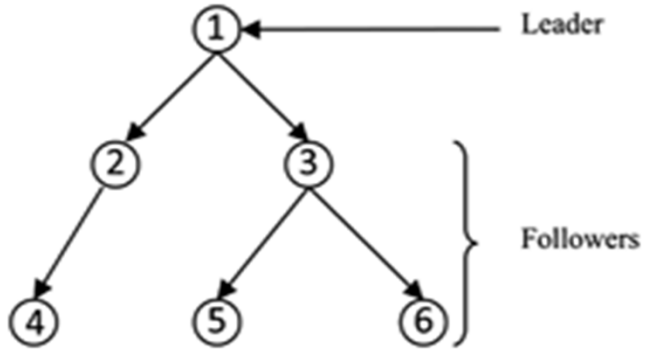
a Directed graph.



b Undirected graph

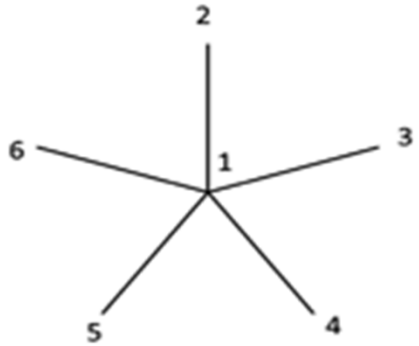


c Complete graph K^6



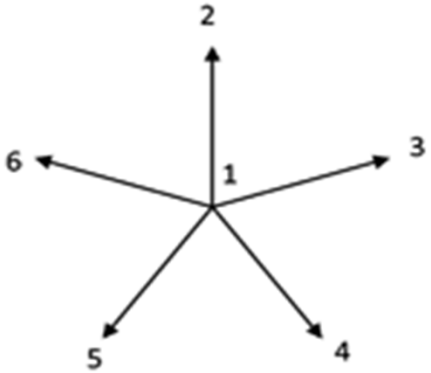
Directed tree graph (formation graph)

d



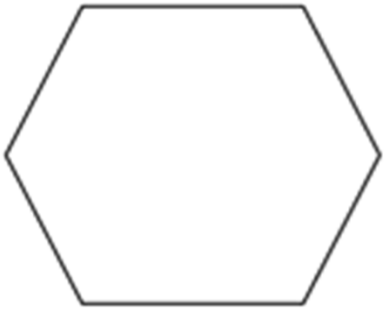
Undirected star $K_{1,5}$

e



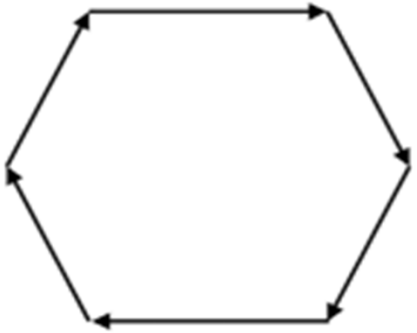
Directed star

f



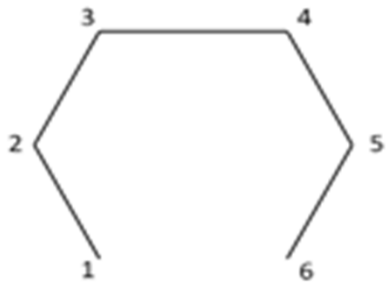
Undirected 6-cycle.
(2-regular graph)

g



Directed 6-cycle
(6-periodic graph)

h



Undirected path P^5

i

Un **drum** este o succesiune de muchii de forma:

- $(a_1, a_2), (a_2, a_3), (a_3, a_4), \ldots , (a_{n-1}, a_n)$ dacă graful este orientat

- $\{a_1, a_2\}, \{a_2, a_3\}, \{a_3, a_4\}, \ldots , \{a_{n-1}, a_n\}$ dacă graful este neorientat

Definiții

- Lungimea unui drum = numărul de muchii care îl constituie.
- Drum simplu = drum în care nici un vârf nu se repetă.
- Ciclu = drum simplu cu excepția primului și ultimului vârf care coincid.
- Graf aciclic = graf fără cicluri.

Se numește **subgraf** al lui **G** un graf

G’ = <V’, M’>

unde **V’**$\subseteq$ **V**, iar **M’** $\subseteq$ **M** (M’ este o mulțime de muchii din M care unesc vârfurile din V’).

Se numește **graf parțial** al lui **G** un graf

$$G'' = \langle V, M'' \rangle$$

unde $M'' \subseteq M$ (graful parțial păstrează același număr de vârfuri **V**).

Alte definiții

- Un graf (orientat sau neorientat) este **conex** dacă între oricare două vârfuri există un drum posibil.

- Un graf orientat este **tare conex** dacă între oricare două varfuri **i** și **j** există un drum de la **i** la **j** și un drum de la **j** la **i**.

- Pentru un graf neconex se pune problema determinării componentelor sale conexe.

- Se numește **componentă conexă** (subgraf conex maximal) al unui graf, un subgraf conex în care nici un vârf din subgraf nu este unit cu unul din afară printr-o muchia a grafului inițial.

Putem atașa informații pentru:

- vârfurile grafului (valori)

- muchii (lungimi sau costuri)

Moduri de reprezentare a unui graf:

a. **Matrice de adiacență** **A**, unde:
- A[i, j] = true , dacă i și j sunt adiacente
- A[i, j] = false , dacă i și j nu sunt adiacente

Observație: într-o altă modalitate de reprezentare a matricei de adiacență putem da valori unei muchii oarecare între nodurile i și j, iar dacă respectiva muchie nu există considerăm A[i, j] = ∞ (sau uneori A[i, j] = 0).

Memoria necesară păstrării unei matrice de adiacență ~ $\mathbf{n^2}$

b. **Listă de adiacență**

- în acest caz se atașează fiecărui vârf ***i*** câte o listă de vârfuri adiacente lui ***i*** (pentru un graf orientat este necesar ca muchia să plece din ***i***).

Observație: Pentru un graf cu ***m*** muchii, suma lungimilor listelor de adiacență este:

- **2m** pentru un graf neorientat
- **m** pentru un graf orientat

c. **Listă de muchii**

- aceasta este o reprezentare eficientă când avem de examinat toate muchiile grafului

Arbori

- Definiția 1:

  Arborele este un graf orientat, aciclic și simplu conex.

- Definiția 2:

  Un arbore este un ansamblu de structuri de date de natură recursivă şi dinamică.

- Definiția 3:

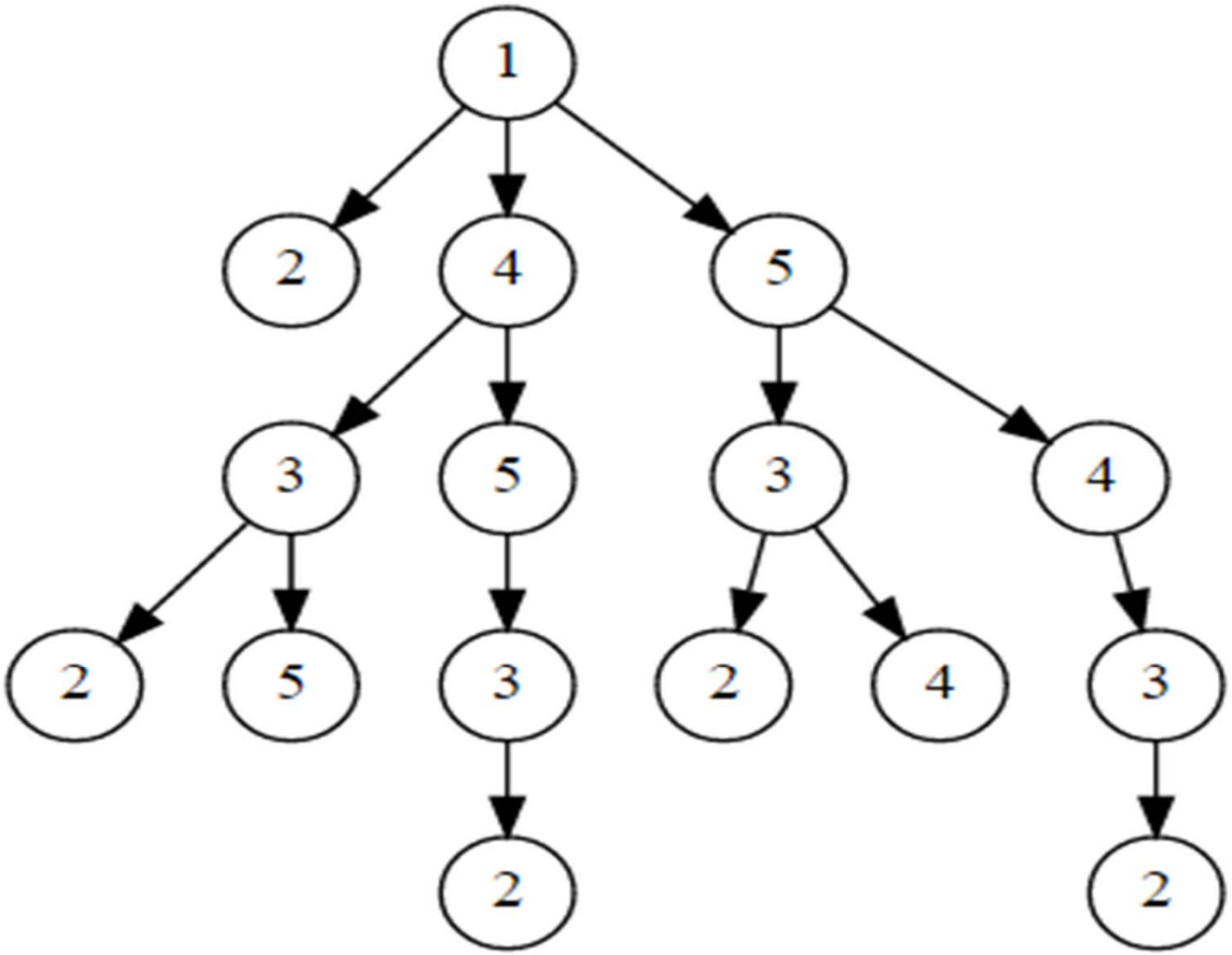
  Prin **arbore** înţelegem o mulţime finită şi nevidă de elemente numite noduri:

  ARB = { A1, A2, A3, ..., An } , unde n > 0 ,

care are următoarele proprietăţi:

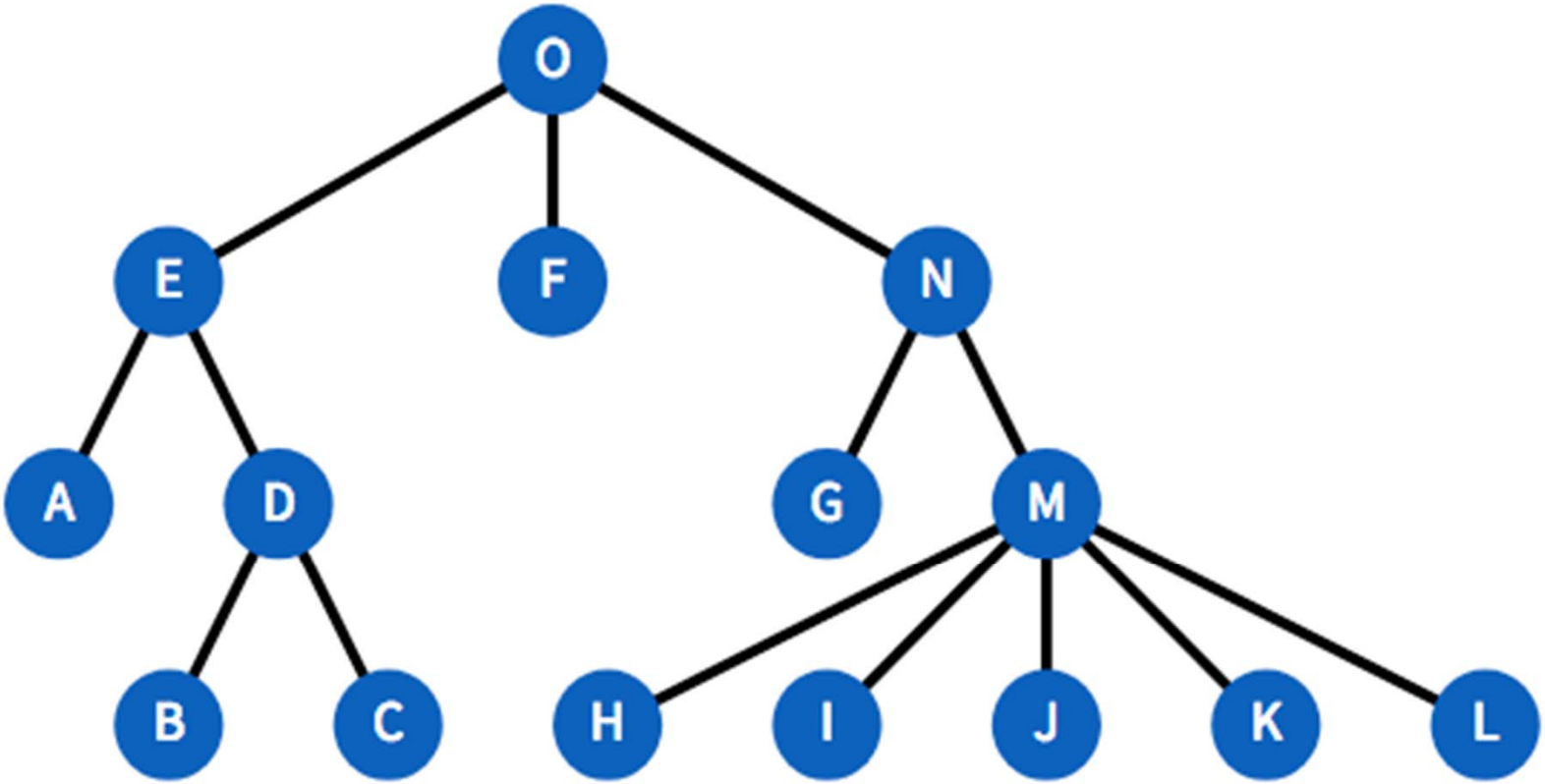
- Există un nod şi numai unul care se numeşte **rădăcina** arborelui.
- Celelalte noduri formează submulţimi disjuncte ale lui **ARB**, care formează fiecare câte un arbore. Arborii respectivi se numesc **subarbori** ai rădăcinii.

Exemplu de arbore (grafic particular)



Nodurile unui arbore

- Într-un arbore există noduri cărora nu le mai corespund subarbori. Un astfel de nod se numeşte **terminal** sau **frunză**.
- În legătură cu arborii s-a stabilit un limbaj conform căruia un ***nod rădacină*** se spune că este un ***nod tată***, iar subarborii rădăcinii sunt ***descendenţii*** acestuia.
- ***Rădăcinile descendenţilor*** unui ***nod tată*** sunt ***fiii*** lui.


O
E
F
N
A
D
G
M
B
C
H
I
J
K
L

Nivel

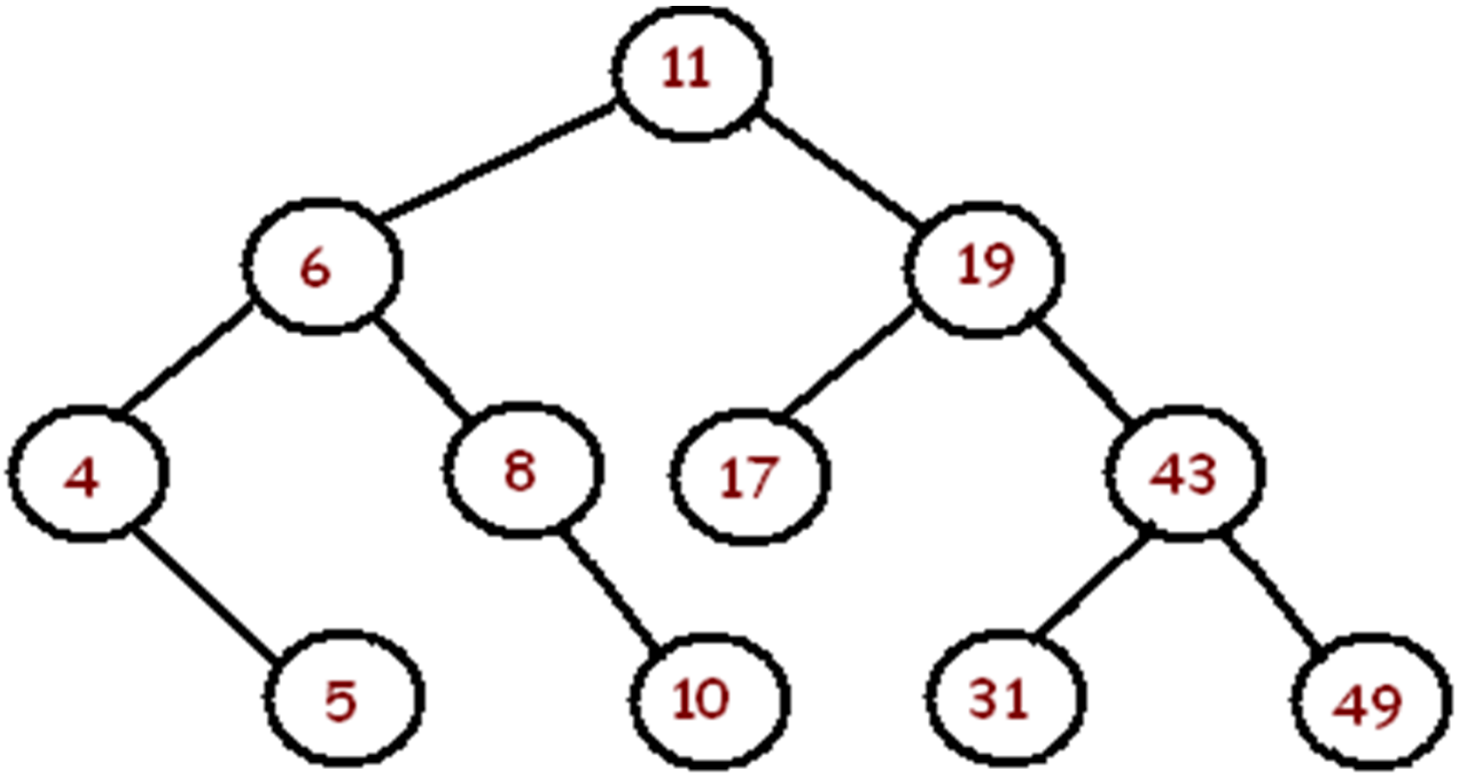
- Rădăcina unui arbore are nivel 1 (sau 0 în alte implementări)

- Dacă un nod se găsește la nivelul **n**, atunci fii lui se găsesc în nivelul **n+1**.

Relația de ordine

- Dacă există o relație de ordine între subarborii oricărui nod al arborelui atunci arborele este **ordonat**.

- Pentru un nod al unui arbore ordonat se notează rădăcina primului subarbore ca fiind fiul cel mai în vârstă, iar rădăcina ultimului subarbore drept fiul cel mai tânăr.


11
6
19
4
8
17
43
5
10
31
49

Uneori un **arbore ordonat** este echivalent unui **arbore genealogic** în care rădăcina arborelui este cel mai vechi strămoș cunoscut.

Arbori binari

Un **arbore binar** este o mulţime finită de elemente care sau este **vidă**, sau conţine un element numit **rădăcina**, iar celelalte elemente (dacă există) se împart în două submulţimi disjuncte, care fiecare la rândul ei, este un arbore binar.

Observații:

- Una din submulţimi este numită **subarborele stâng** al rădăcinii, iar cealaltă **subarborele drept**.
- Arborele binar este ordonat, deoarece în fiecare nod, subarborele stâng se consideră că precede subarborele drept.
- De aici rezultă că un nod al unui arbore binar are cel mult doi fii şi că unul este **fiul stâng**, iar celalalt este **fiul drept**.

Observații (continuare)

- Fiul stâng este considerat *mai în vârstă* decât cel drept.

- Un nod al unui arbore binar poate să aibă numai un singur descendent. Acesta poate fi subarborele stâng sau subarborele drept.

Observații (continuare)

- Cele două posibilităţi anterior menționate se consideră distincte.

- Cu alte cuvinte, dacă doi arbori binari diferă numai prin aceea că nodul **V** dintr-un arbore are ca descendent numai fiul stâng, iar acelaşi nod echivalent, din celălalt arbore, are ca descendent numai fiul drept, cei doi arbori se consideră distincţi.

Particularități

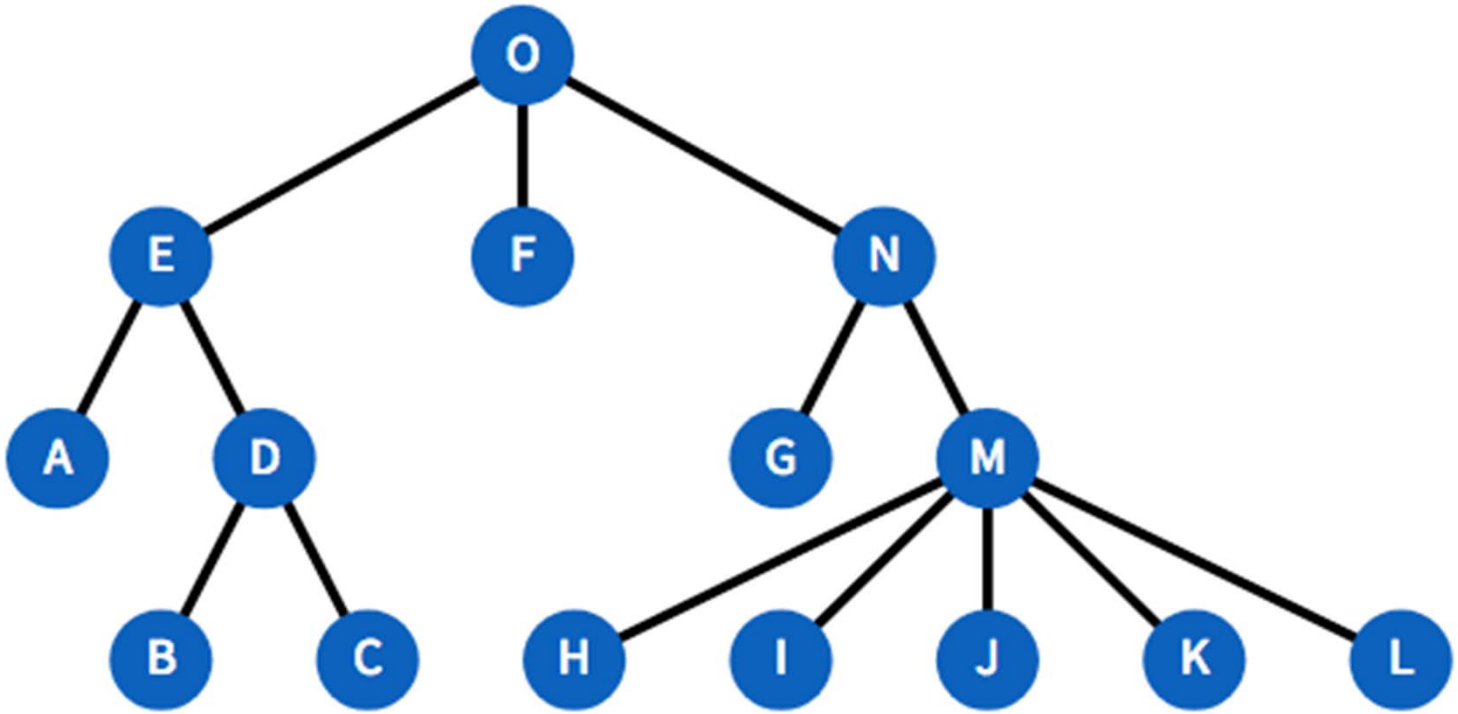
- Un arbore binar nu se defineşte ca un caz particular de arbore ordonat. Astfel, un arbore nu este niciodată vid, spre deosebire de un arbore binar care poate fi şi vid.

- Orice arbore ordonat poate fi întotdeauna reprezentat printr-un arbore binar.

Modalitatea de transformare a unui arbore ordonat într-un arbore binar

1. Se leagă între ei frații descendenți ai aceluiași nod tată și se suprimă legăturile lor cu nodul tată, cu excepția primului fiu.
2. Fostul prim fiu devine fiul stâng al nodului tată, iar ceilalți foști frați formează, în mod consecutiv, subarbori drepți. Fiecare dintre foștii frați devine descendent drept (fiu drept) al fostului frate mai mare.

Încercați să transformați acest arbore într-un arbore binar



Utilizarea structurilor pentru realizarea unui arbore binar

Nodul unui arbore binar poate fi reprezentat ca o dată structurală de tipul NOD care se defineşte în felul următor:

```
typedef struct nod
    {
        declaratii ;
        struct nod *st;
        struct nod *dr;
    } NOD;
```

unde:

- st - este pointerul spre fiul stâng al nodului curent;
- dr - este pointerul spre fiul drept al aceluiaşi nod.

Asupra arborilor binari se pot defini mai multe operaţii:

1. inserarea unui nod frunză într-un arbore binar;

2. accesul la un nod al unui arbore;

3. parcurgerea unui arbore;

4. ştergerea unui arbore.

- Operaţiile de **inserare** şi **acces** la un nod, au la bază un **criteriu** care să definească locul în arbore al nodului în cauză.

- Acest **criteriu** este dependent de problema concretă, la care se aplică arborii binari, pentru a fi rezolvată.