



Programare procedurala

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2014 – 2015
Semestrul I

Saptamana 9



Curs 9 - Cuprins

0. Recursivitate – notiuni introductive

1. Structuri de date dinamice (II)

- arbori (oarecare, binari, binari de cautare)
- creare
- parcurgeri

2. (Di)grafuri



0. Recursivitate (1/5)

Recursivitate este proprietatea functiilor de a se autoapela.

Sintaxa

tip functie_recursiva (parametru formal)

{ ...

conditie de oprire

ramura de continuare

functie_recursiva (parametru formal modificat)

}

Toate instructiunile din subprogram se executa de cate ori este apelata functia.



0. Recursivitate (2/5)

Orice functie recursiva trebuie sa contina **o conditie de oprire** respectiv, de continuare.

La fiecare reapel al functiei se executa aceeas secventa de instructiuni.

La fiecare reapel, in zona de stiva a memoriei:

- se ocupa un nivel nou
- se memoreaza valoarea parametrilor formali transmisi prin valoare
- adresa parametrilor formali transmisi prin referinta
- adresa de revenire
- variabilele cu valorile din momentul respectiv



0. Recursivitate (3/5)

Obs:

Toate instructiunile din subprogram se executa pentru fiecare reapel

- se executa instructiunile din functie pana la instructiunea de reapel

- se executa din nou aceeaas secventa de instructiuni pana la conditia de oprire

- procedeul se reia pana la intalnirea conditiei de oprire

Pentru fiecare apel s-a salvat in stiva un nivel, apoi pentru fiecare dintre aceste apeluri se executa instructiunile ramase in functie cu valoarea

datelor din varful stivei **(atentie! vor fi in ordine inversa introducerii lor in stiva).**



0. Recursivitate (4/5)

Exemple

```
int fun1(int n)
{
    if (n == 0) return 0;
    else return n + fun1(n-1);
}

// varianta iterativa
int fun2(int n)
{
    int y = 0;
    while (n!=0)
    {
        y = y + n;
        n--;
    }
    return y;
}
```

```
int main()
{
    int n;
    scanf("%d",&n);
    printf("Rezultat functie recursiva = %d\n", fun1(n));
    printf("Rezultat functie iterativa = %d\n", fun2(n));

    return 0;
}
```

```
"C:\Users\Ank\Desktop\Curs 9\bin\Debug\Curs 9.exe"
5
Rezultat functie recursiva = 15
Rezultat functie iterativa = 15
```

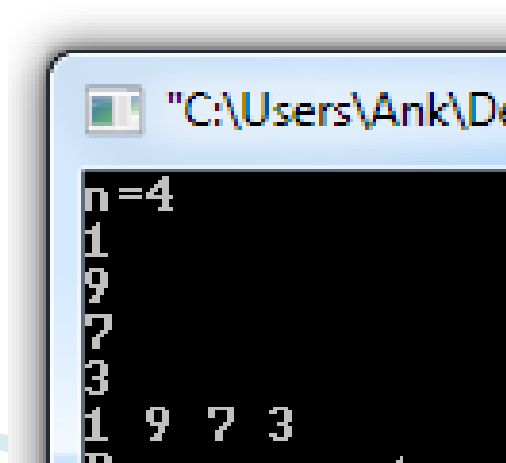


0. Recursivitate (5/5)

Exemple

Citirea si afisarea unui vector

```
int main()
{
    int a[20],n;
    printf("n="); scanf("%d",&n);
    citire(a,n);
    afisare(a,n);
}
```



```
void citire(int a[20],int n)
{
    scanf("%d",&a[n]);
    if(n>1) citire(a,n-1);
}

void afisare(int a[20],int n)
{
    if (n>=1)
    {
        printf("%d ",a[n]);
        afisare(a,n-1);
    }
}
```



1. Structuri dinamice de date (II) (1/16)

Avantaje ale alocarii dinamice

- memoria necesara este alocata (si / sau eliberata) in timpul executiei programului (cand e nevoie) si nu la compilarea programului
- un bloc de memorie alocat dinamic poate fi redimensionat dupa necesitati.

Functia malloc()

Returneaza adresa de inceput a unui bloc de memorie alocat in HEAP (daca exista suficient spațiu liber).

Pointerul în care păstrăm adresa returnată de malloc va fi plasat în zona de memorie statică.



1. Structuri dinamice de date (II) (2/16)

Funcția realloc() - Redimensionarea blocurilor alocate dinamic

Primește ca parametri adresa de memorie a unui bloc deja alocat și noua dimensiune și returnează noua lui adresa de memorie (daca exista suficient spatiu pentru realocare) sau NULL.

In caz de succes → blocul poate să fie mutat la o nouă locație de memorie, dar tot conținutul va fi păstrat.

Funcția calloc()

Este echivalenta cu funcția malloc(), dar, pe langa alocare de memorie pentru un bloc, realizeaza si inițializarea zonei alocate

Funcția free()

Elibereaza zona de memorie alocata in decursul executarii programului.



1. Structuri dinamice de date (II) (3/16)

Liste (simple, duble, circulare)

Stive, cozi simple, cozi speciale

Arbori (oarecare, binari, binari de cautare, etc.)

Grafuri (orientate/digrafuri, neorientate)

Operatii de baza

Traversarea

Cautarea

Inserarea

Stergerea



1. Structuri dinamice de date (II) (4/16)

Structuri arborescente

Arbori binari oarecare (recursiv)

```
nod* adaug()  
{  
    nod*t;  
    int x;  
    printf(" Dati un element:");  
    t = (nod *)malloc(sizeof(nod));  
    scanf("%d",&t->info);  
    t->st = t->dr = NULL;  
    printf("\nFiu stang pentru %d (d/n)? \n", t->info);  
    if(getche()!='n') t->st=adaug();  
    printf("\nFiu dreapta pentru %d (d/n)? \n", t->info);  
    if(getche()!='n') t->dr=adaug();  
    return t;  
}
```

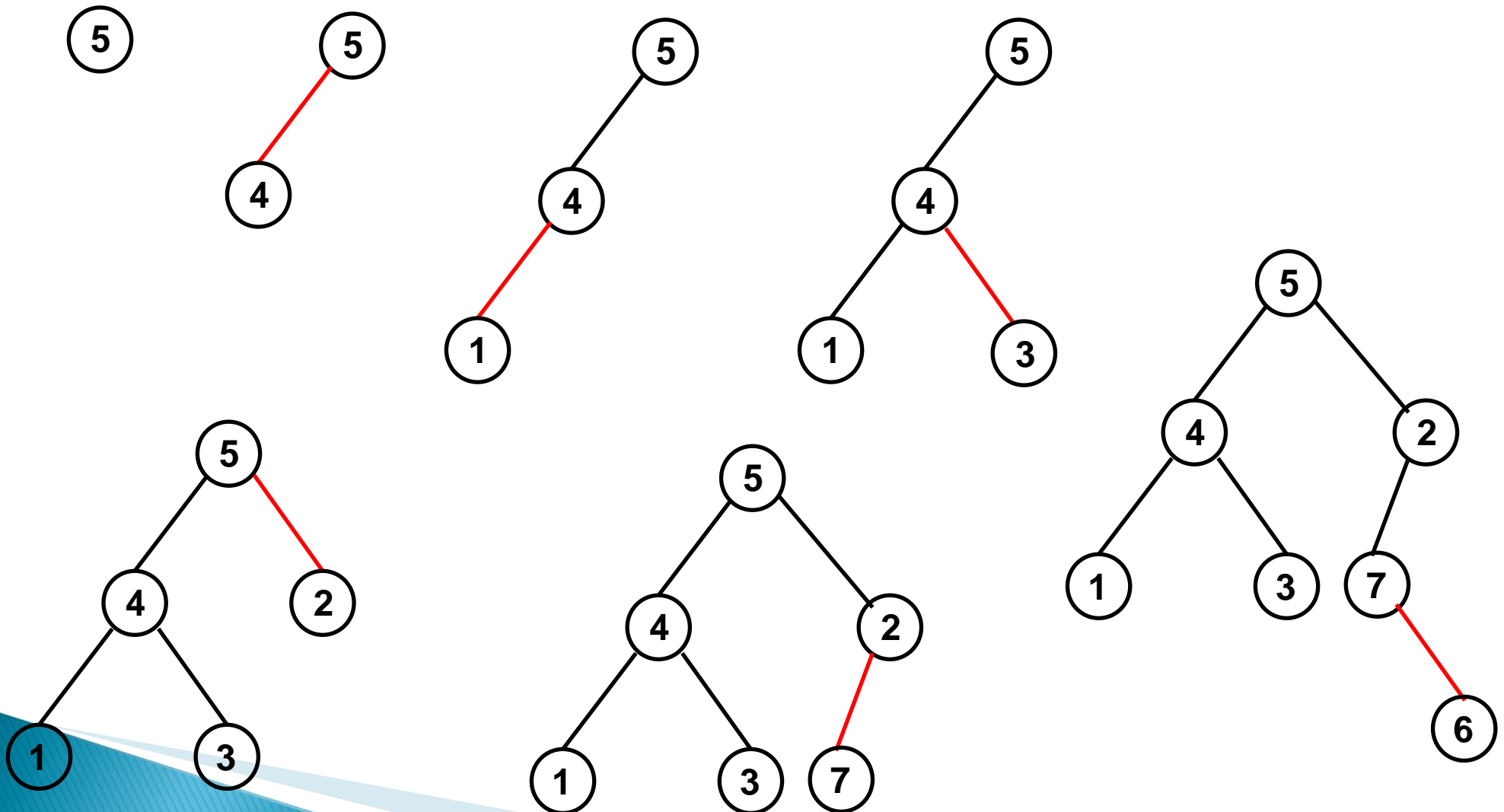
nod* rad = adaug();

```
"C:\Users\Ank\Desktop\Curs 9\bin\Debug  
Dati un element:5  
Fiu stang pentru 5 (d/n)?  
d Dati un element:  
4  
Fiu stang pentru 4 (d/n)?  
d Dati un element:  
1  
Fiu stang pentru 1 (d/n)?  
n  
Fiu dreapta pentru 1 (d/n)?  
n  
Fiu dreapta pentru 4 (d/n)?  
d Dati un element:  
3  
Fiu stang pentru 3 (d/n)?  
n  
Fiu dreapta pentru 3 (d/n)?  
n  
Fiu dreapta pentru 5 (d/n)?  
d Dati un element:  
2  
Fiu stang pentru 2 (d/n)?  
d Dati un element:  
7  
Fiu stang pentru 7 (d/n)?  
n  
Fiu dreapta pentru 7 (d/n)?  
Dati un element:6  
Fiu stang pentru 6 (d/n)?  
n  
Fiu dreapta pentru 6 (d/n)?  
n  
Fiu dreapta pentru 2 (d/n)?  
n
```



1. Structuri dinamice de date (II) (5/16)

Creare Arbore Binar Oarecare





1. Structuri dinamice de date (II) (6/16)

Structuri arborescente – Arbori binari oarecare

Parcurgeri (recursiv): RSD, SRD, SDR

```
void rsd(nod *rad)
{
    if(rad)
    {
        printf("%d ", rad->info);
        rsd(rad->st);
        rsd(rad->dr);
    }
}
```

```
void srd(nod *rad)
{
    if(rad)
    {
        srd(rad->st);
        printf("%d ", rad->info);
        srd(rad->dr);
    }
}
```

```
void sdr(nod *rad)
{
    if(rad)
    {
        sdr(rad->st);
        sdr(rad->dr);
        printf("%d ", rad->info);
    }
}
```

```
printf("\nParcurgerea in preordine: ");
rsd(rad);
printf("\nParcurgerea in inordine: ");
srd(rad);
printf("\nParcurgerea in postordine: ");
sdr(rad);
printf("\n");
```

```
Parcurgerea in preordine: 5 4 1 3 2 7 6
Parcurgerea in inordine: 1 4 3 5 7 6 2
Parcurgerea in postordine: 1 3 4 6 7 2 5
```



1. Structuri dinamice de date (II) (7/16)

Structuri arborescente – Arbori binari de cautare

Creare (recursiv)

```
void adaug(nod **rad, int x)
{
    if(!(*rad))
    {
        *rad = (nod *)malloc(sizeof(nod));
        (*rad)->info=x;
        (*rad)->st=NULL;
        (*rad)->dr=NULL;
    }
    else
    {
        if(x<(*rad)->info)
            adaug(&(*rad)->st,x);
        if(x>(*rad)->info)
            adaug(&(*rad)->dr,x);
    }
}
```

Sir initial:

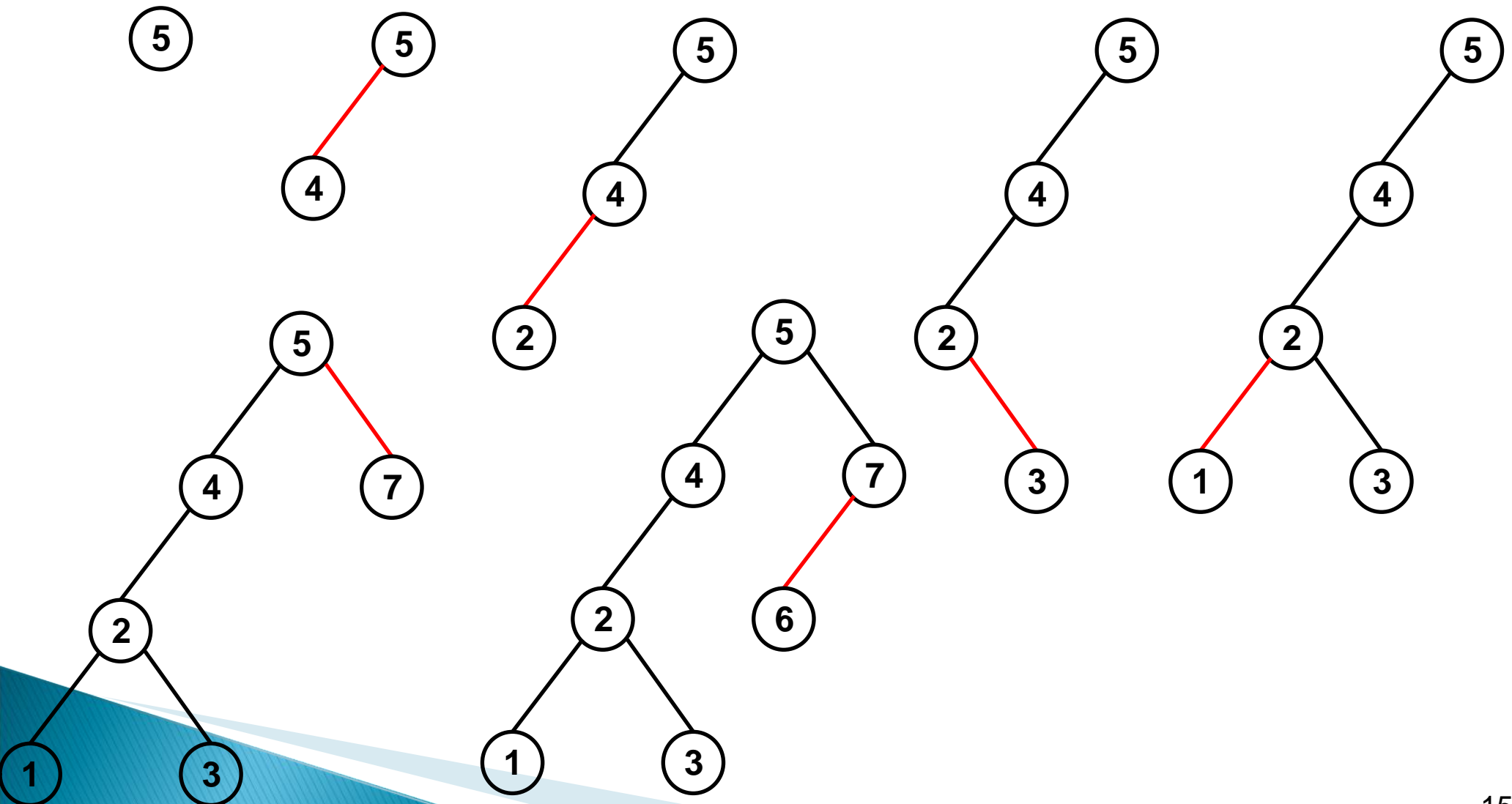
5, 4, 2, 3, 1, 7, 6, 0.

```
nod*rad = NULL;
int x;
printf("Introduce-ti elementele arborelui. Pen
printf("Dati un element:");scanf("%d",&x);
while(x)
{
    adaug(&rad,x);
    printf("Dati un element:");scanf("%d",&x);
}
```



1. Structuri dinamice de date (II) (8/16)

Creare ABC (recursiv) Sir initial: 5, 4, 2, 3, 1, 7, 6, 0.





1. Structuri dinamice de date (II) (9/16)

Structuri arborescente – Arbori binari de cautare

Parcurgeri (recursiv): RSD, SRD, SDR

```
void rsd(nod *rad)
{
    if(rad)
    {
        printf("%d ", rad->info);
        rsd(rad->st);
        rsd(rad->dr);
    }
}
```

```
void srd(nod *rad)
{
    if(rad)
    {
        srd(rad->st);
        printf("%d ", rad->info);
        srd(rad->dr);
    }
}
```

```
void sdr(nod *rad)
{
    if(rad)
    {
        sdr(rad->st);
        sdr(rad->dr);
        printf("%d ", rad->info);
    }
}
```

```
printf("\nParcurgerea in preordine: ");
rsd(rad);
printf("\nParcurgerea in inordine: ");
srd(rad);
printf("\nParcurgerea in postordine: ");
sdr(rad);
printf("\n");
```

```
"C:\Users\Ank\Desktop\Curs 9\bin\Debug\Curs 9.exe"
Introduce-ti elementele arborelui. Pentru
Dati un element:5
Dati un element:4
Dati un element:2
Dati un element:3
Dati un element:1
Dati un element:7
Dati un element:6
Dati un element:0

Parcurgerea in preordine: 5 4 2 1 3 7 6
Parcurgerea in inordine: 1 2 3 4 5 6 7
Parcurgerea in postordine: 1 3 2 4 6 7 5
```

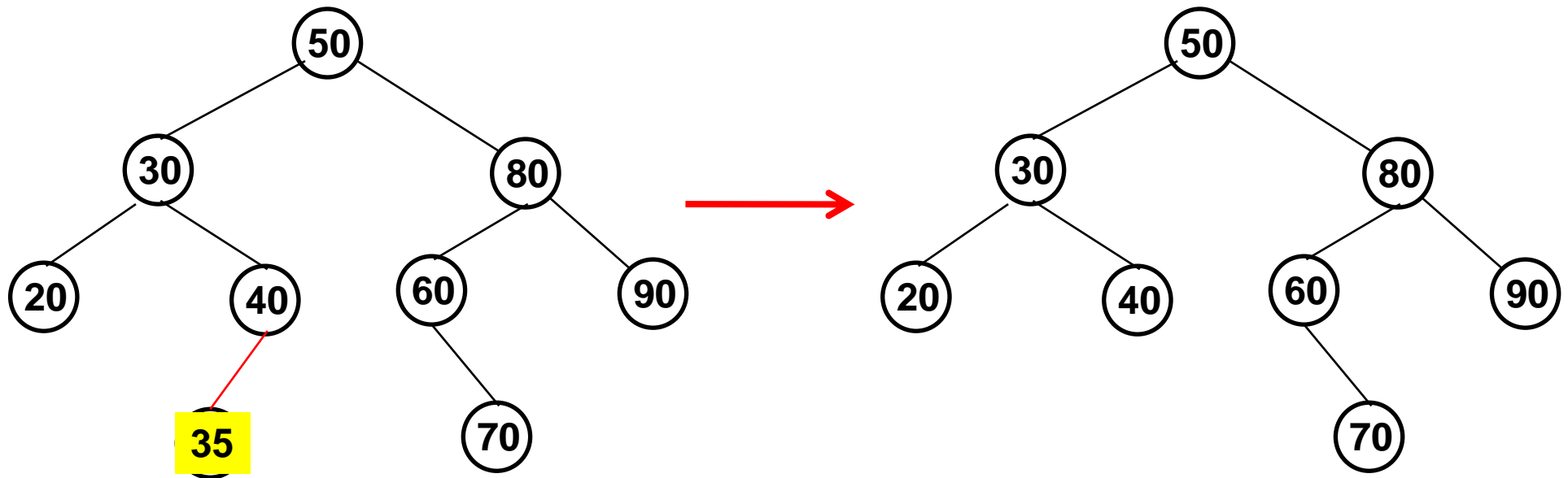



1. Structuri dinamice de date (II) (10/16)

Structuri arborescente – Arbori binari de cautare

Stergerea unei valori

a) Frunza (nu are descendenți)



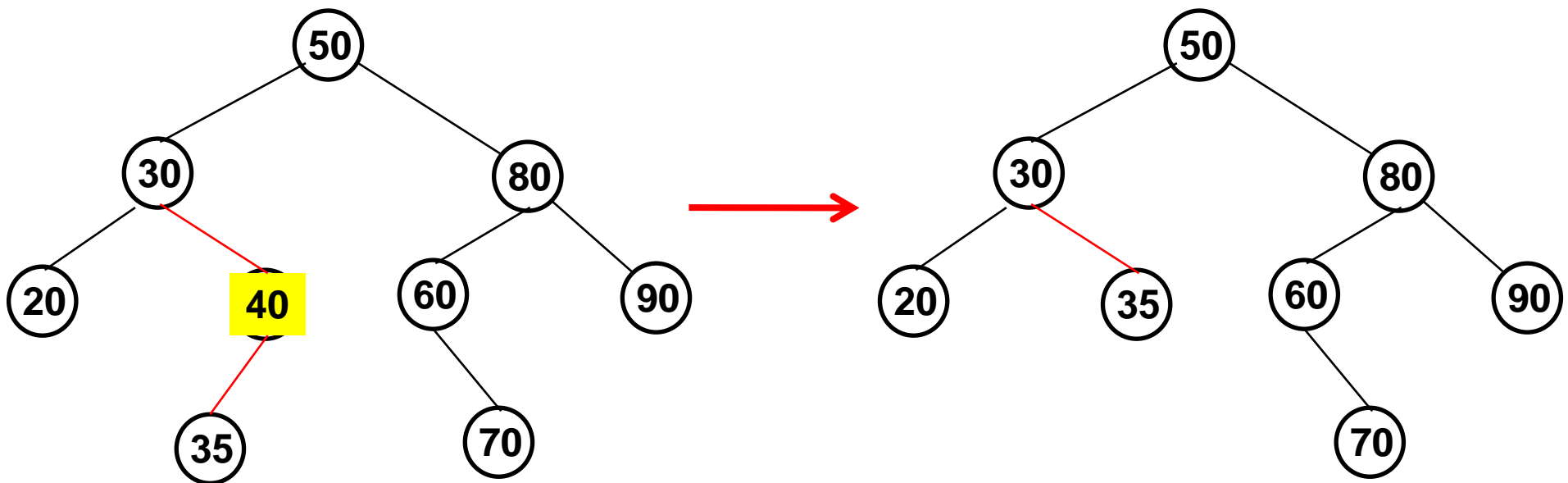


1. Structuri dinamice de date (II) (11/16)

Structuri arborescente – Arbori binari de cautare

Stergerea unei valori

b) Nodul are un singur descendent



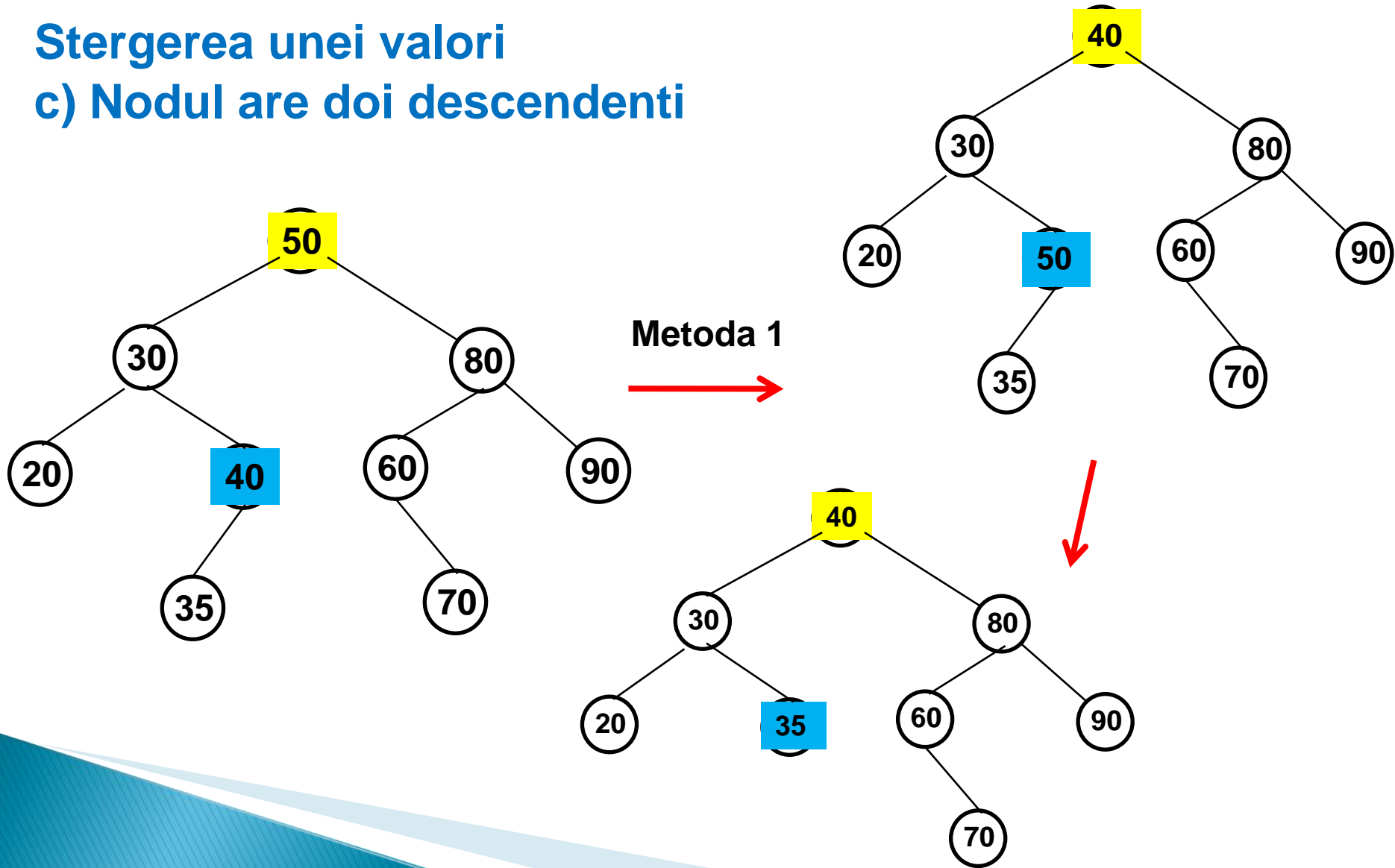


1. Structuri dinamice de date (II) (12/16)

Structuri arborescente – Arbori binari de cautare

Stergerea unei valori

c) Nodul are doi descendenți



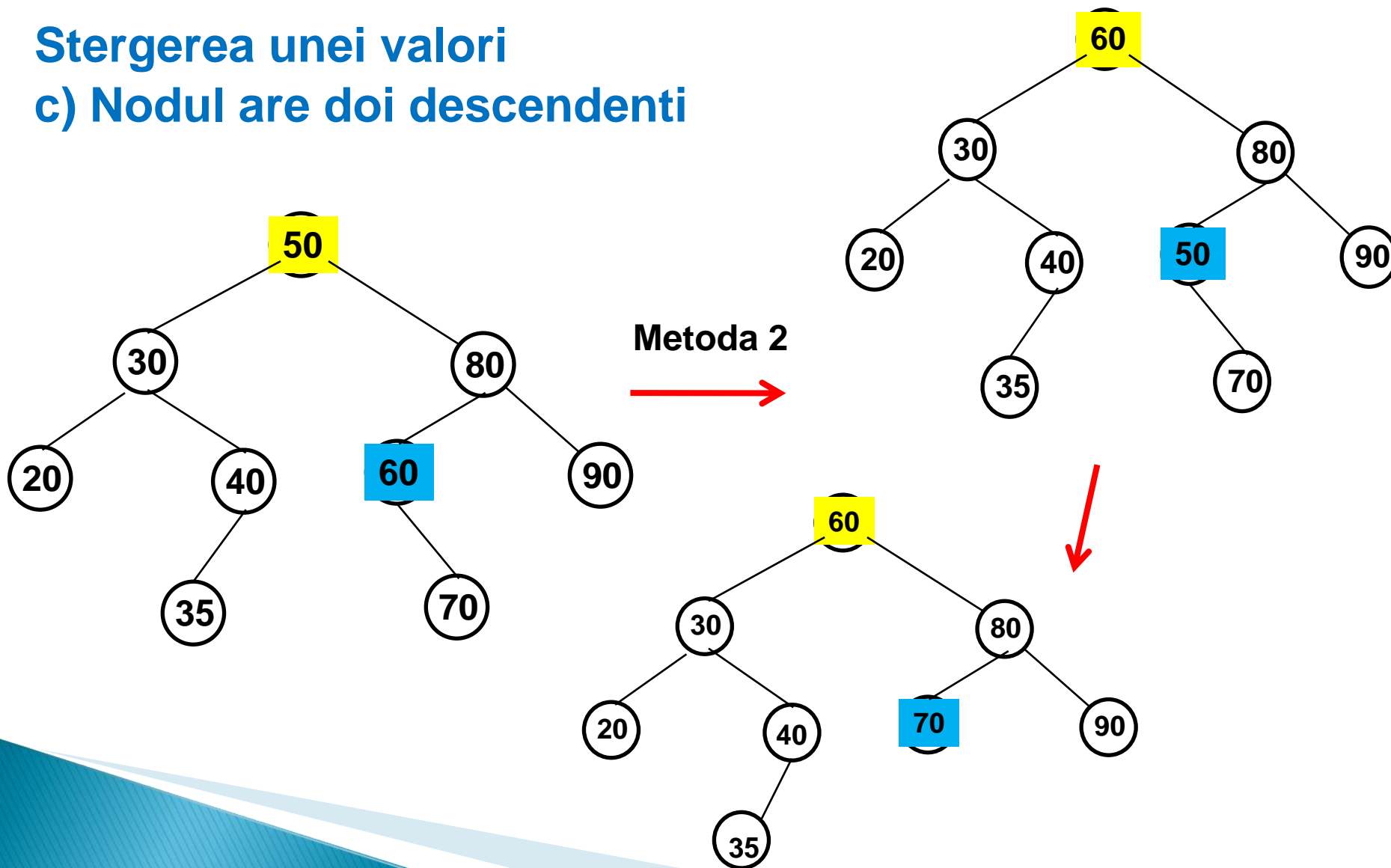


1. Structuri dinamice de date (II) (13/16)

Structuri arborescente – Arbori binari de cautare

Stergerea unei valori

c) Nodul are doi descendenți



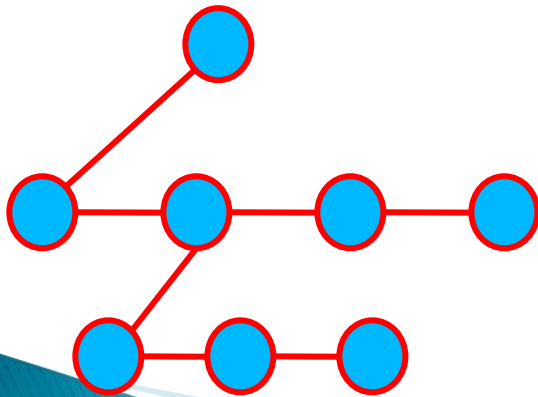
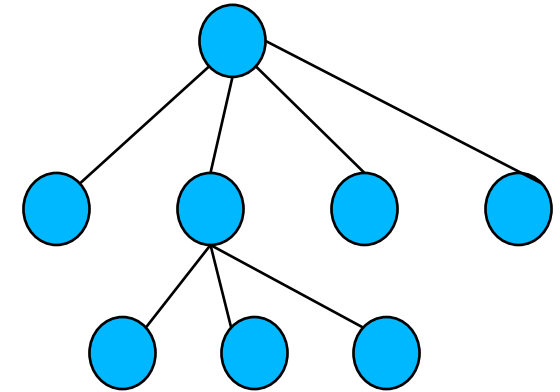


1. Structuri dinamice de date (II) (14/16)

Structuri arborescente – Arbori oarecare

Creare (recursiv)

```
#define NMAX 30 //numarul maxim de descendenti ai unui nod
typedef struct nod nod;
struct nod{
    int inf;
    int n; //numarul de descendenti
    nod *leg[NMAX]; //tabloul adreselor descendentilor
};
nod *coada[100]; //coada pentru parcurgerea pe nivele
int prim, ultim, fr[100], j=0; //pentru gestionarea cozii
```



```
nod* Creare() //creaza nodurile oarecare si returneaza adresa radacinii
{
    int i,q=0;
    nod *p = (nod*) malloc (sizeof(nod));
    printf("informatia nodului: "); scanf("%d",&p->inf);
    printf("numarul descendentilor pentru %d : ",p->inf);
    scanf("%d",&p->n);
    if(p->n==0)
    {fr[j]=p->inf;
    j++;}
    printf("\n");
    for(i=0;i<p->n;i++) p->leg[i]=Creare();
    return p; //radacina
}
```



1. Structuri dinamice de date (II) (15/16)

Structuri arborescente – Arbori oarecare

Parcurgeri: Adancime si Latime

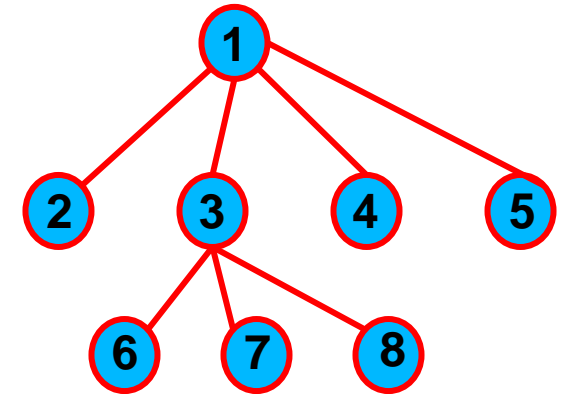
```
void Preordine(nod *p) //afiseaza nodurile in preordine(adancime)
{
    int i;
    if(p)
    {
        printf("%d ",p->inf); //afisez nodul tata
        for(i=0;i<p->n;i++) Preordine(p->leg[i]); //afisez descendenti
    }
}
```

DF (Adancime):

1, 2, 3, 6, 7, 8, 4, 5

BF (Latime):

1, 2, 3, 4, 5, 6, 7, 8



```
void Traversare_nivele(nod *rad)
{
    nod *p; int i,q=-1;
    prim=ultim=0;
    Adauga(rad); //in coada se introduce nodul radacina
    do{
        p=Extrage_nod(); //extrag un nod din coada
        if(p)
        {
            printf("%d ",p->inf); //afisez informatia nodului
            for(i=0;i<p->n;i++)
                Adauga(p->leg[i]); //adaug in coada descendenti nodului
        }
    }while(p);
    printf("\n");
}
```



1. Structuri dinamice de date (II) (16/16)

Structuri arborescente – Arbori oarecare

Alocare statica

Vectori de tata

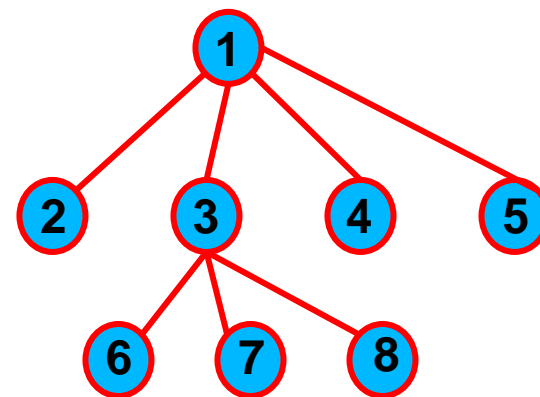
- Radacina are zero pe postul de tata
- Fiecare alt nod are un tata

2, 3, 4, 5 – au drept tata pe 1

6,7,8 – au drept tata pe 3

Vectorul de tata asociat arborelui:

0, 1, 1, 1, 1, 3, 3, 3



Legatura fii frate

- 1 – radacina
- 2 – fiu pentru radacina
- 3 – frate pentru 2
- 4 – frate pentru 2 (si implicit pentru 3)
- 5 – frate pentru 2 (...)
- 6 – fiu pentru 3
- 7 – frate pentru 6
- 8 – frate pentru 6 (si implicit pentru 7)

Matricea de adiacenta

$A[i][j] = 1$ / $[i,j]$ – muchie

$A[i][j] = 0$ / $[i,j]$ – nu este muchie



2. (Di)grafuri (1/8) ([1])

Def Se numeste graf sau graf neorientat o structura $G = (V, E)$, unde $V \neq \emptyset$ este o multime de varfuri si E este o submultime posibil vida a multimii perechilor neordonate cu componente distincte din V .

Obs:

1. Graful $G = (V, E)$ este graf finit, daca V este o multime finita.
2. Varfurile u, v sunt adiacente in G daca $[u, v] \in E$. Intr-un graf neorientat relatia de adiacenta este simetrica. Se spune si ca muchia $[u, v]$ este incidenta varfurilor u si v .
3. Intr-un graf neorientat $[u, v]$ si $[v, u]$ sunt considerate a fi aceeasi muchie.

Def Gradul unui varf al unui graf neorientat este nr muchiilor incidente ale acestuia.

Obs: Varful u este izolat daca gradul sau este 0.

Def Un graf orientat sau digraf G este o pereche (V, E) unde V - multimea varfurilor lui G este o multime finita, iar E este o relatie binara pe V . Elementele multimii E se numesc arce. Spunem ca arcul (u, v) este incident din sau pleaca din varful u si este incident in sau intra in varful v .



2. (Di)grafuri (2/8) ([1])

Reprezentarea grafurilor

Un graf $G = (V, E)$ (orientat sau neorientat) se reprezinta prin:

- Liste de adiacenta
- Matrice de adiacenta

Listele de adiacenta

In majoritatea algoritmilor, grafurile sunt introduse sub forma de liste de adiacenta.

Sunt indicate in special in reprezentarea grafurilor rare ($|E| \ll |V^2|$). In cazul grafurilor dense ($|E|$ este aproape de $|V^2|$), sau in cazul in care se doreste sa se cunoasca repede daca exista o muchie care conecteaza doua varfuri date, se prefera reprezentarea grafurilor prin matrice de adiacenta.



2. (Di)grafuri (3/8) ([1])

Reprezentarea grafurilor

Matricea de adiacenta

Se presupune ca varfurile sunt numerotate $1, 2, \dots, |V|$ intr-o maniera arbitrara. Matricea de adiacenta asociata grafului G este $A_{|V| \times |V|}$, $A = (a_{ij})$ unde:

$$a_{ij} = \begin{cases} 1, & \text{daca } [i, j] \in E, \\ 0, & \text{altfel.} \end{cases}$$

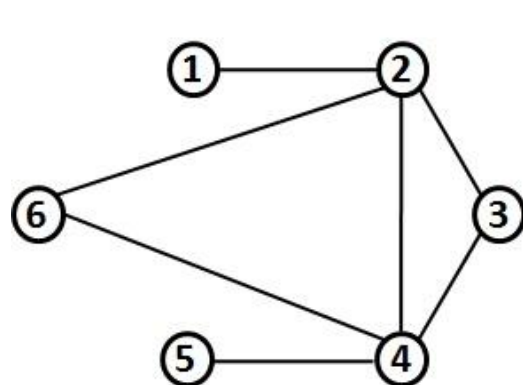
Proprietatile matricei de adiacenta A asociate unui graf neorientat G

1. elementele de pe diagonala principala sunt 0,
2. simetrie fata de diagonala principala ($A = A^T$),
3. linia corespunzatoare unui nod izolat contine numai 0,
4. suma elementelor de pe fiecare linie i indica gradul nodului i .

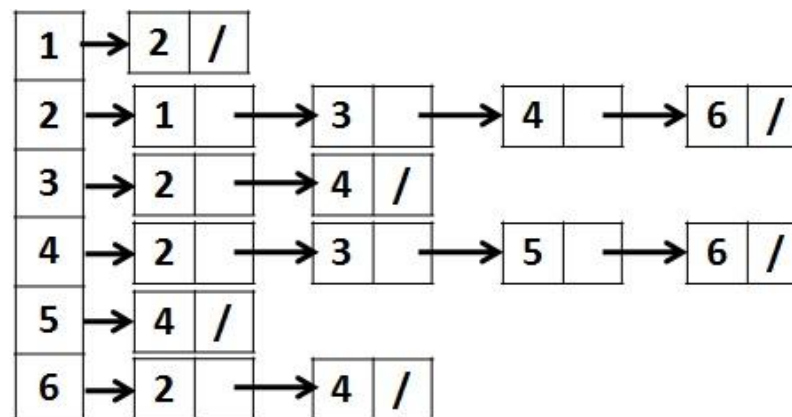


2. (Di)grafuri (4/8) ([1])

Reprezentarea grafurilor



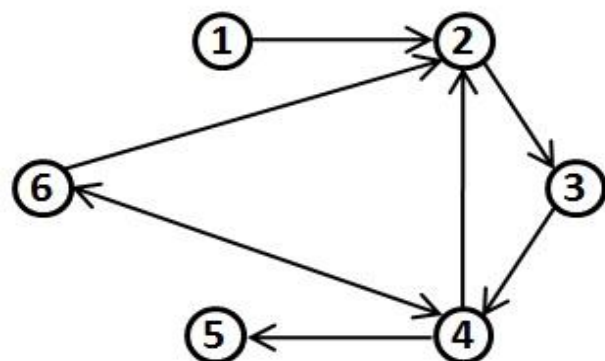
a)



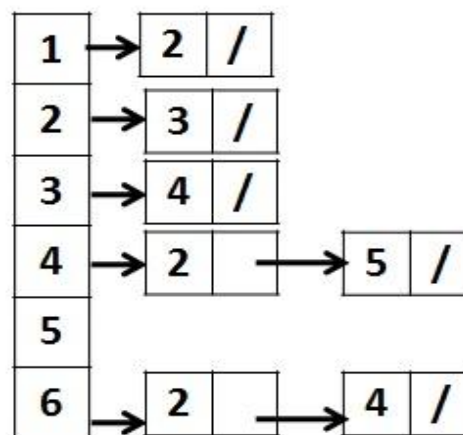
b)

$$\begin{bmatrix}
 0 & 1 & 0 & 0 & 0 & 0 \\
 1 & 0 & 1 & 1 & 0 & 1 \\
 0 & 1 & 0 & 1 & 0 & 0 \\
 0 & 1 & 1 & 0 & 1 & 1 \\
 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & 1 & 0 & 1 & 0 & 0
 \end{bmatrix}$$

c)



a)



b)

$$\begin{bmatrix}
 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & 1 & 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 1 & 0 & 0
 \end{bmatrix}$$

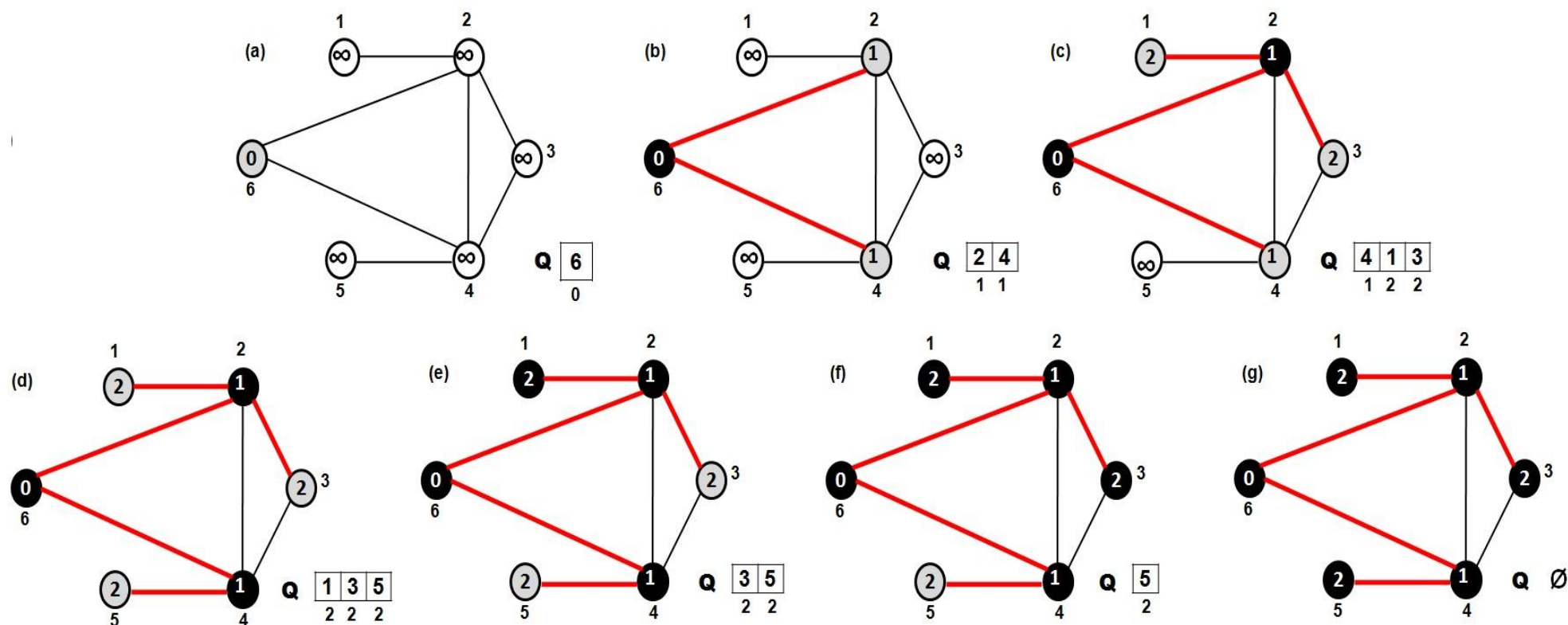
c)

Reprezentare (di)grafuri: a) grafic, b) liste de adiacenta, c) matrice de adiacenta



2. (Di)grafuri (5/8) ([1])

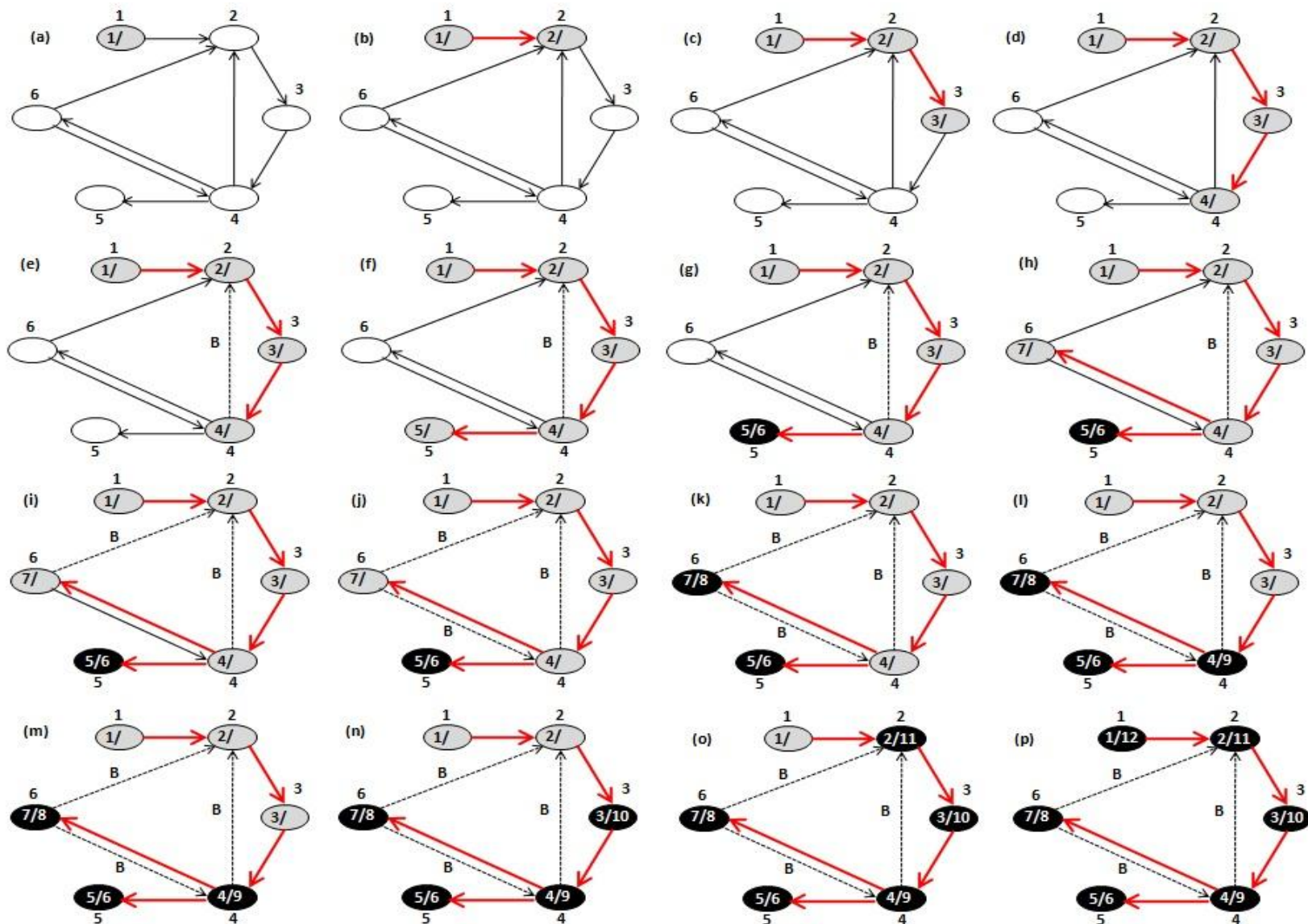
Parcurgerea (di) grafurilor in Latime (BF)





2. (Di)grafuri (6/8) ([1])

Parcurgerea (di) grafurilor in Adancime (DF)





2. (Di)grafuri (7/8) ([1])

Aplicatie grafuri: Problema celebrității ([2])

Se dă un grup format din n persoane, care se cunosc sau nu între ele. De la tastatură se citesc: n , nr.de persoane, m nr.de muchii și **extremitățile celor m muchii**, cu semnificația (x,y) persoana x cunoaște persoana y .

Relația de cunoștință nu este neapărat reciprocă.

Numim **celebritate**, o persoană care este cunoscută de către toate celelalte persoane, dar ea nu cunoaște pe niciun alt membru al grupului.

Să se determine dacă în grup există sau nu o astfel de celebritate.

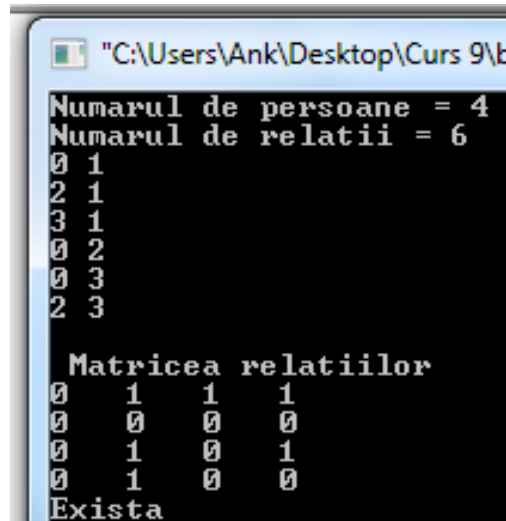


2. (Di)grafuri (8/8) ([1])

```
int main()
{
    int n,m,x,y,i,j,a[20][20],celebritate=0;
    printf("Numarul de persoane = ");
    scanf("%d", &n);
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            a[i][j]=0;

    printf("Numarul de relatii = ");
    scanf("%d", &m);
    for(i=0;i<m;i++)
    {
        scanf("%d%d", &x,&y);
        a[x][y]=1;
    }

    printf("\n Matricea relatiilor \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
}
```



```
"C:\Users\Ank\Desktop\Curs 9\l
Numarul de persoane = 4
Numarul de relatii = 6
0 1
2 1
3 1
0 2
0 3
2 3

Matricea relatiilor
0 1 1 1
0 0 0 0
0 1 0 1
0 1 0 0
Exista
```

```
int suma_linie(int x, int a[20][20], int n)
{
    int s=0,i;
    for(i=0;i<n;i++)
        s+=a[x][i];
    return s;
}

int suma_coloana(int x, int a[20][20], int n)
{
    int s=0,i;
    for(i=0;i<n;i++)
        s+=a[i][x];
    return s;
}
```

```
for(i=0;i<n;i++)
    if ((suma_coloana(i,a,n)==0) && (suma_linie(i,a,n)==n-1))
        celebritate++;
if (celebritate!=0)
    printf("Exista");
else
    printf("Nu exista");
```



Concluzii

Functii recursive

Structuri de date dinamice (II)

- **arbori (oarecare, binari, binari de cautare)**
 - **creare (iterativa / recursiva)**
 - **parcurgeri: BF, DF, RSD, SRD, SDR**

(Di)grafuri

- **reprezentare**
- **parcurgeri**
- **aplicatii**



Perspective

Cursul 10:

- 1. Operatii de intrare – iesire (inclusiv fisiere) si aplicatii
privind sortarea datelor stocate extern**
- 2. Pointeri la functii si functii cu numar variabil de
argumente**