Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Structuri de Date si Algoritmi

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2019 – 2020
Semestrul I
Seriile 21 + 25

Curs 5

29/10/2019

Curs 5 - Cuprins

3. Structuri arborescente

Arbori oarecari. Definitii, terminologie, reprezentari, parcurgeri.

Arbori binari. Reprezentari, parcurgeri.

Arbori binari stricti. Proprietati matematice. Aplicatii.

Arbori binari de cautare. Operatii: cautare, inserare, stergere.

Algoritmul de cautare binara si performanta lui.

Arbori binari echilibrati AVL. Performanta cautarii in arbori binari de cautare echilibrati AVL.

Arbori oarecare

Definitie ([1])

Fiind dată o mulţime M de elemente denumite noduri sau vârfuri, vom numi arbore (conform definiţiei date de Knuth) un set finit de noduri astfel încât:

a) există un nod cu destinaţie specială, numit rădăcina arborelui;

b) celelalte noduri sunt repartizate în m ≥ 0 seturi disjuncte A1 , A2 , ... Am , fiecare set Ai constituind la rândul său un arbore.

Elementele arborelui sunt nodurile şi legăturile dintre ele.

Nodul rădăcină r îl considerăm ca fiind pe nivelul 0.

[1] https://olidej.wikispaces.com/file/view/1009+Alocare+dinamica.pdf

Arbori oarecare

Definitie ([1])

A1 , A2 , ..., Am - arbori, fie r1 , r2 , ..., rm rădăcinile lor.

- r1 , r2 , ..., rm formează nivelul 1 al arborelui;

- Nodul r va avea câte o legătură cu fiecare dintre nodurile r1 , r2 , ..., rm;

Continuând acest procedeu, vom avea nivelurile 2, 3, ..., ale arborelui.

Dacă numărul nivelurilor este finit, atunci şi arborele este finit, în caz contrar se numeşte arbore infinit.

Numim **înălţime** (sau **adâncime**) a unui arbore nivelul maxim al nodurilor sale

[1] https://olidej.wikispaces.com/file/view/1009+Alocare+dinamica.pdf

Arbori oarecare

Definitie ([1])

Viziune ierarhică → nodurile sunt subordonate unele altora.

Fiecare nod este subordonat direct unui singur nod, excepţie constituie rădăcina care nu este subordonată nici unui nod.

Dacă un nod nu are nici un nod subordonat, atunci se numeşte frunză sau nod terminal.

[1] https://olidej.wikispaces.com/file/view/1009+Alocare+dinamica.pdf

Arbori oarecare

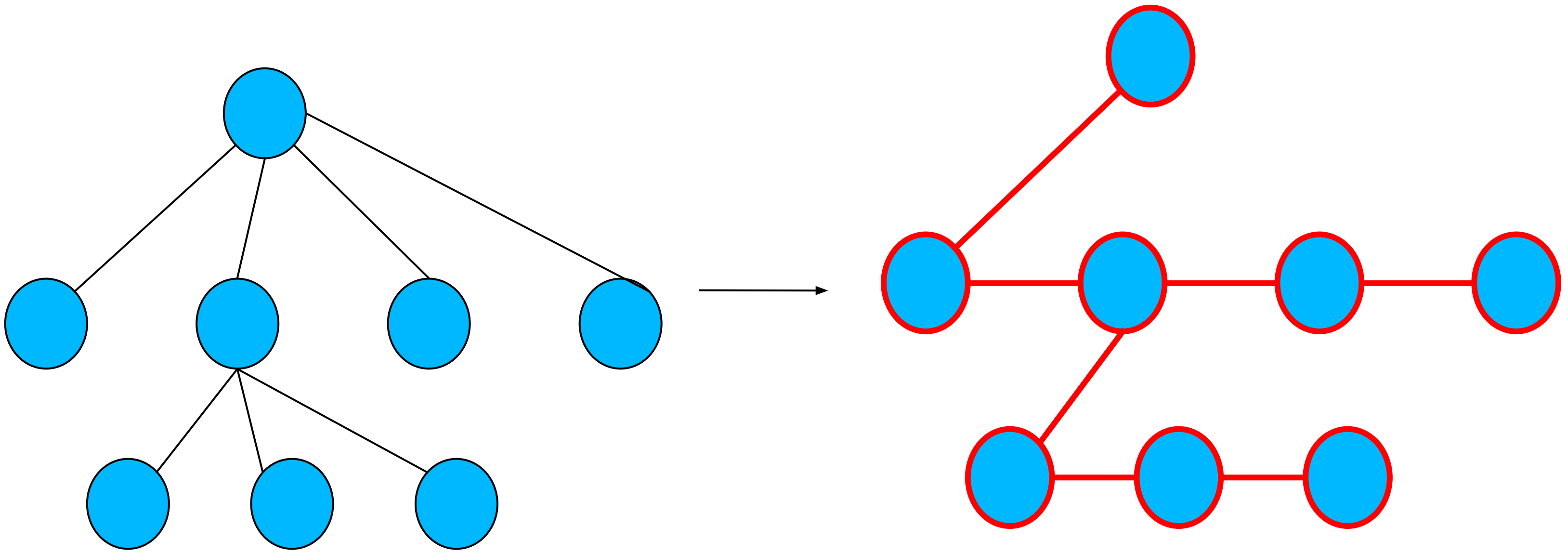
Terminologie

- *arborii ordonaţi*, arbori în care există o relaţie de ordine între descendenţii unui nod

- *arborii neordonaţi*, arbori în care nu există o relaţie de ordine între descendenţii unui nod

Arbori oarecare

Creare



Fiecare nod are trei câmpuri:

inf: pentru informaţia din noduri

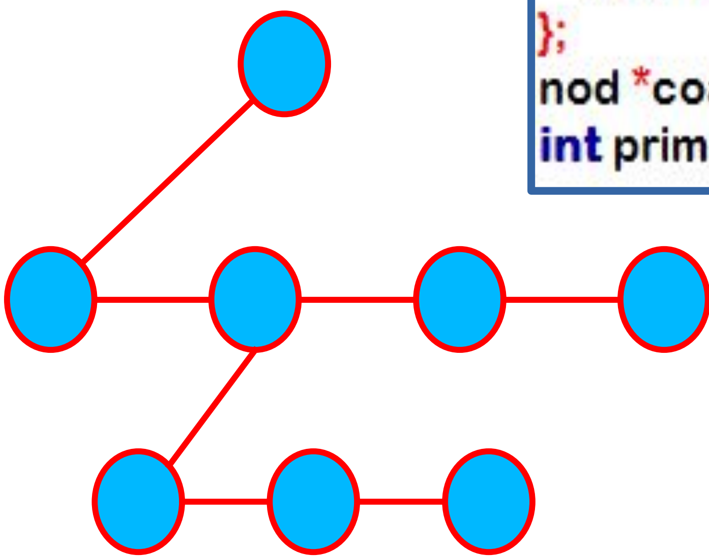
n: valoarea *NMAX* - numărul de fii ai nodului

leg: vector de dimensiune *NMAX*, cu componente pointeri, astfel încât, leg*[J]* este pointer către fiul *J* al nodului, iar *J= 1,2,…,NMAX.*

Arbori oarecare

Creare

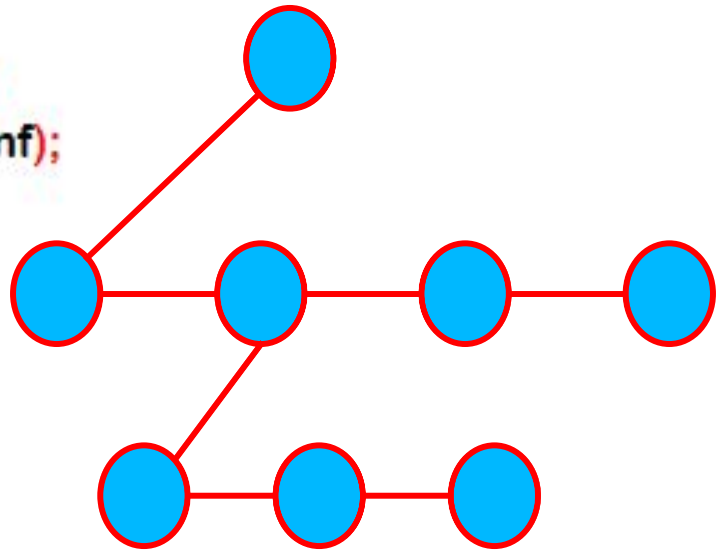
```
#define NMAX 30  //numarul maxim de descendenti ai unui nod
typedef struct nod nod;
struct nod{
   int inf;
   int n;          //numarul de descendenti
   nod *leg[NMAX]; //tabloul adreselor descendentilor
};
nod *coada[100]; //coada pentru parcurgerea pe nivele
int prim,ultim,fr[100],j=0;   //pentru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



Arbori oarecare

Creare

```
nod* Creare()  //creaza nodorele 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

}
```



Arbori oarecare

Traversare

Algoritmul de traversare:

1. Se porneşte de la rădăcină.

2. La fiecare nod curent:
(a) se procesează *inf*
(b) se introduc într-o structură ajutătoare fiii nodului curent în vederea procesării ulterioare.

3. Se extrage din structura ajutătoare un alt nod şi se reia de la punctul 2.

Arbori oarecare

Traversare

Algoritmul de traversare:

1. Se porneşte de la rădăcină.

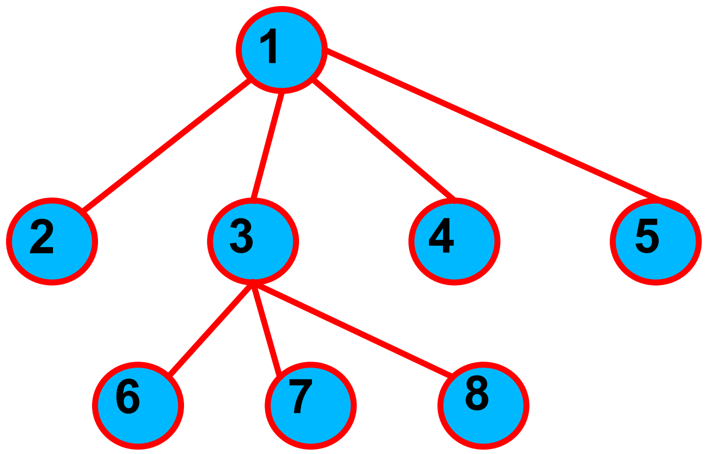
2. La fiecare nod curent:
(a) se procesează *inf*
(b) se introduc într-o structură ajutătoare fiii nodului curent în vederea procesării ulterioare.

3. Se extrage din structura ajutătoare un alt nod şi se reia de la punctul 2.

Obs. Structură ajutătoare - stiva / coada.
Dacă folosim o coadă obţinem *traversarea în lăţime (breadth first)* a arborelui.
Dacă folosim o stivă obţinem *traversarea în adâncime (depth first)* a arborelui.

Arbori oarecare

Traversarea in adancime (DF)

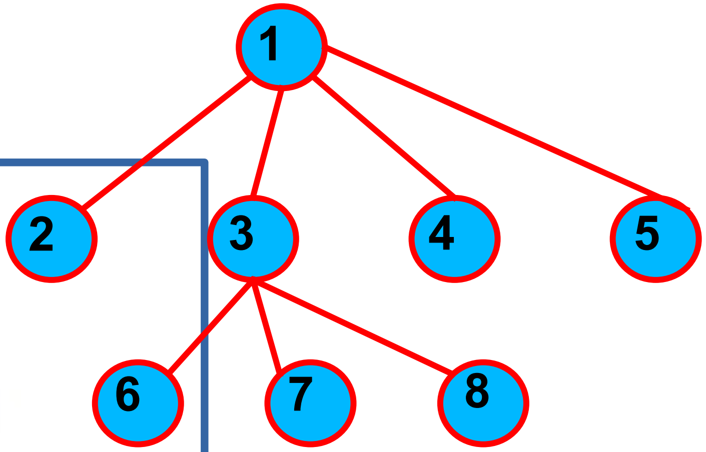


DF (Adancime):
1, 2, 3, 6, 7, 8, 4, 5

```
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 descendentii
   }
}
```

Arbori oarecare

Traversarea in latime (BF)




```
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 descendentii nodului
      }
   }while(p);
   printf("\n");
```

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

Arbori binari oarecare

Definitie

Un arbore binar (2 - arbore ordonat) T este:

(1) fie un arbore vid.

(2) fie e nevid, şi atunci conţine un nod numit rădăcină, împreună cu doi subarbori binari disjuncţi numiţi subarborele stâng, respectiv subarborele drept.

```
struct nod{
    int info;
    nod *left, *right;
    };

nod * rad;
```

Arbori binari oarecare

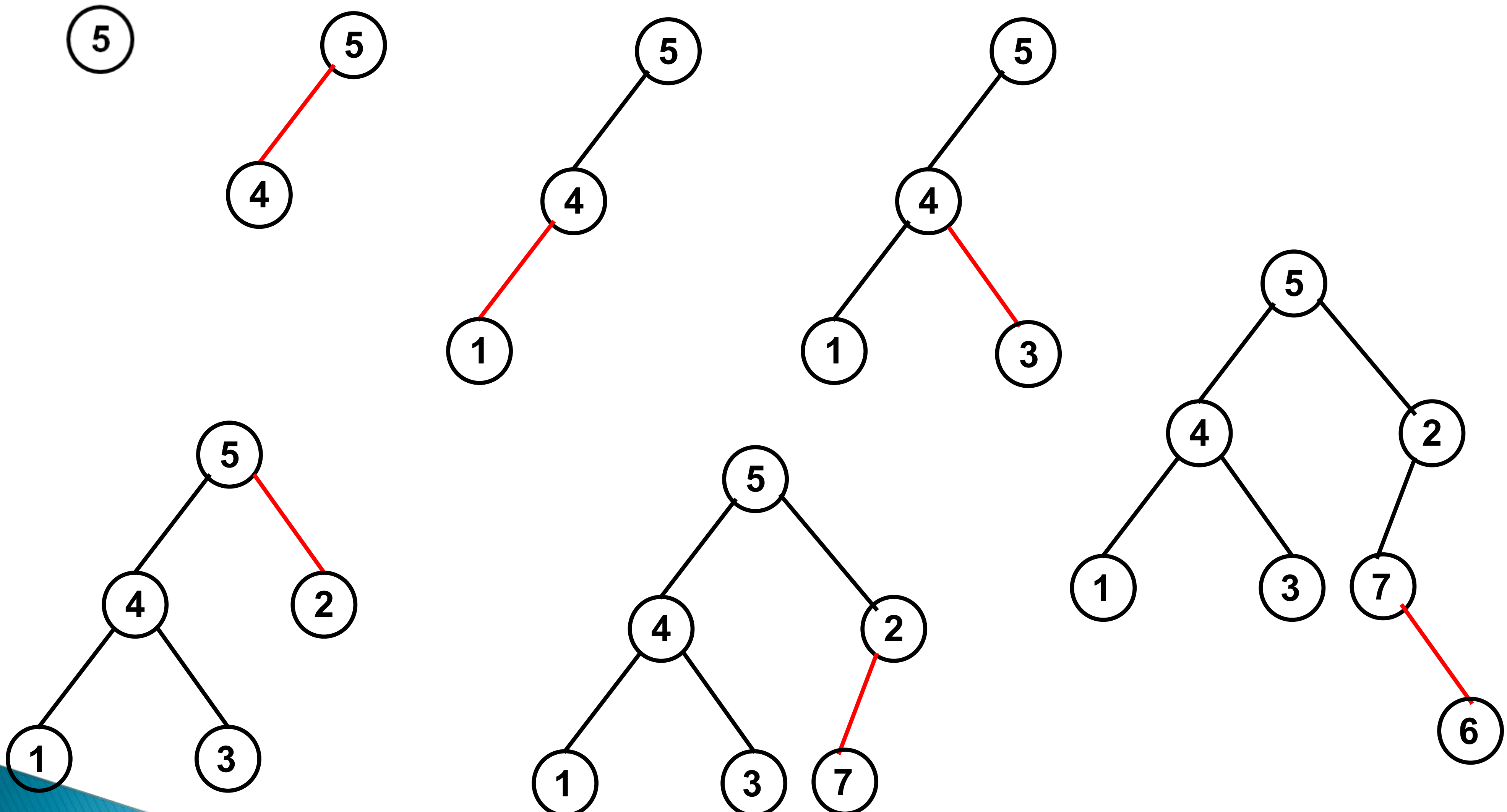
Creare (recursiv)

```c
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;
 }
```

```
"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
```

Arbori binari oarecare

Creare (recursiv)



Arbori binari oarecare

Parcurgeri (recursiv): RSD, SRD, SDR

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

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

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

```c
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
```

Arbori binari de cautare

Definitie

Structura

- Arbore binar

- cu chei de un tip total ordonat

- pentru orice nod *u* al său avem relaţiile:

(1) *info[u]* > *info[v]*, pentru orice *v* in *left[u]*

(2) *info[u]* < *info[w]*, pentru orice *w* in *right[u]*.

Arbori binari de cautare

Definitie

Un arbore binar de cautare T este:

(1) fie un arbore vid
(2) fie e nevid,
şi atunci conţine un nod numit rădăcină, cu *info* de un tip totul ordonat împreună cu doi subarbori binari de cautare disjuncţi (numiţi subarborele stâng*, left*, respectiv subarborele drept*, right*) si astfel incat

(1') *info[root(T)] > info[root(left[T])]*
(2') *info[root(T)] < info[root(right[T])]*

Arbori binari de cautare

Chei multiple

Contorizare aparitii
Reprezentare efectiva a cheilor multiple:

Numim arbore binar de căutare ***nestrict la stânga*** un arbore binar *T* cu proprietatea că în fiecare nod *u* al său avem relaţiile:

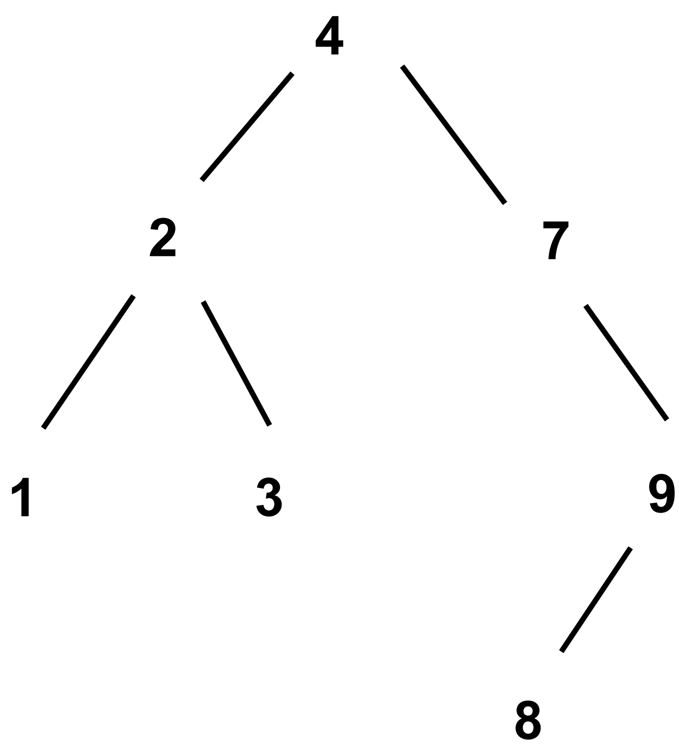
(3) *info[u]* **>=** *info[v],* pentru orice *v* in *left[u]*
(4) *info[u]* < *info[w],* pentru orice *w* in *right[u]*.

Analog, arbore binar de căutare ***nestrict la dreapta***, cu relaţiile:

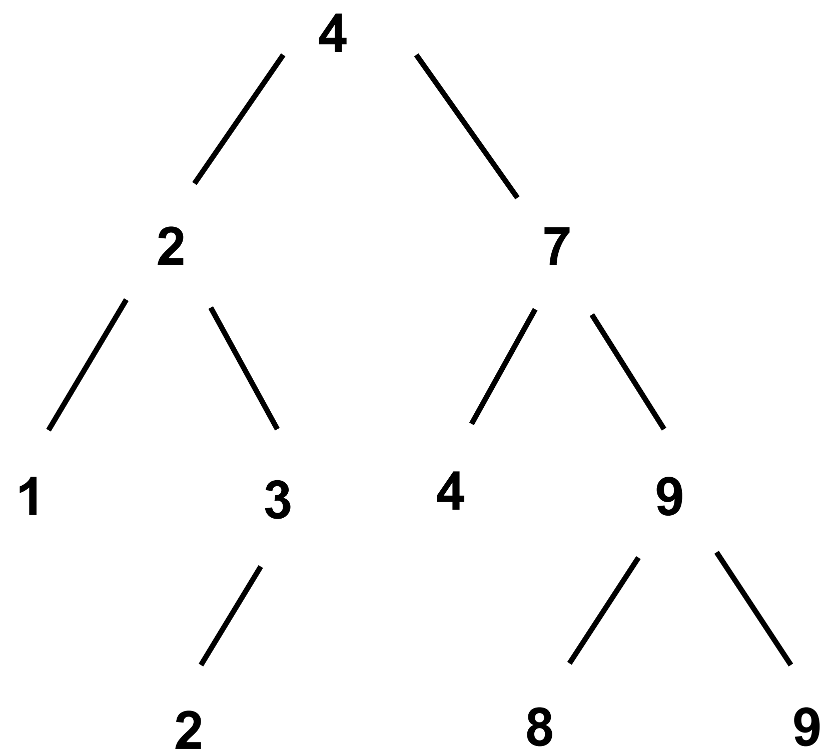
(5) *info[u]* > *info[v],* pentru orice *v* in *left[u]*
(6) *info[u]* <= *info[w],* pentru orice *w* in *right[u].*

Arbori binari de cautare

Chei multiple



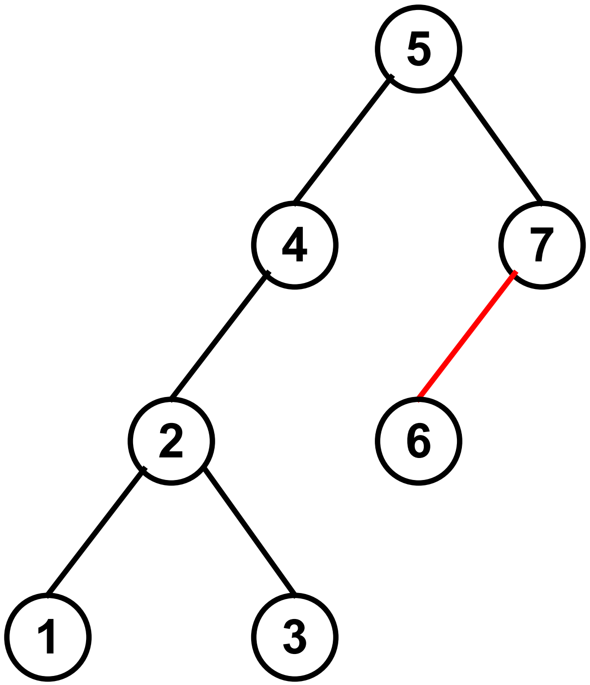
(a) Arbore binar de căutare strict.



(b) Arbore binar de căutare nestrict la dreapta. Cheile 22, 44 şi 99 sunt chei multiple.

Arbori binari de cautare

Creare (recursiv)



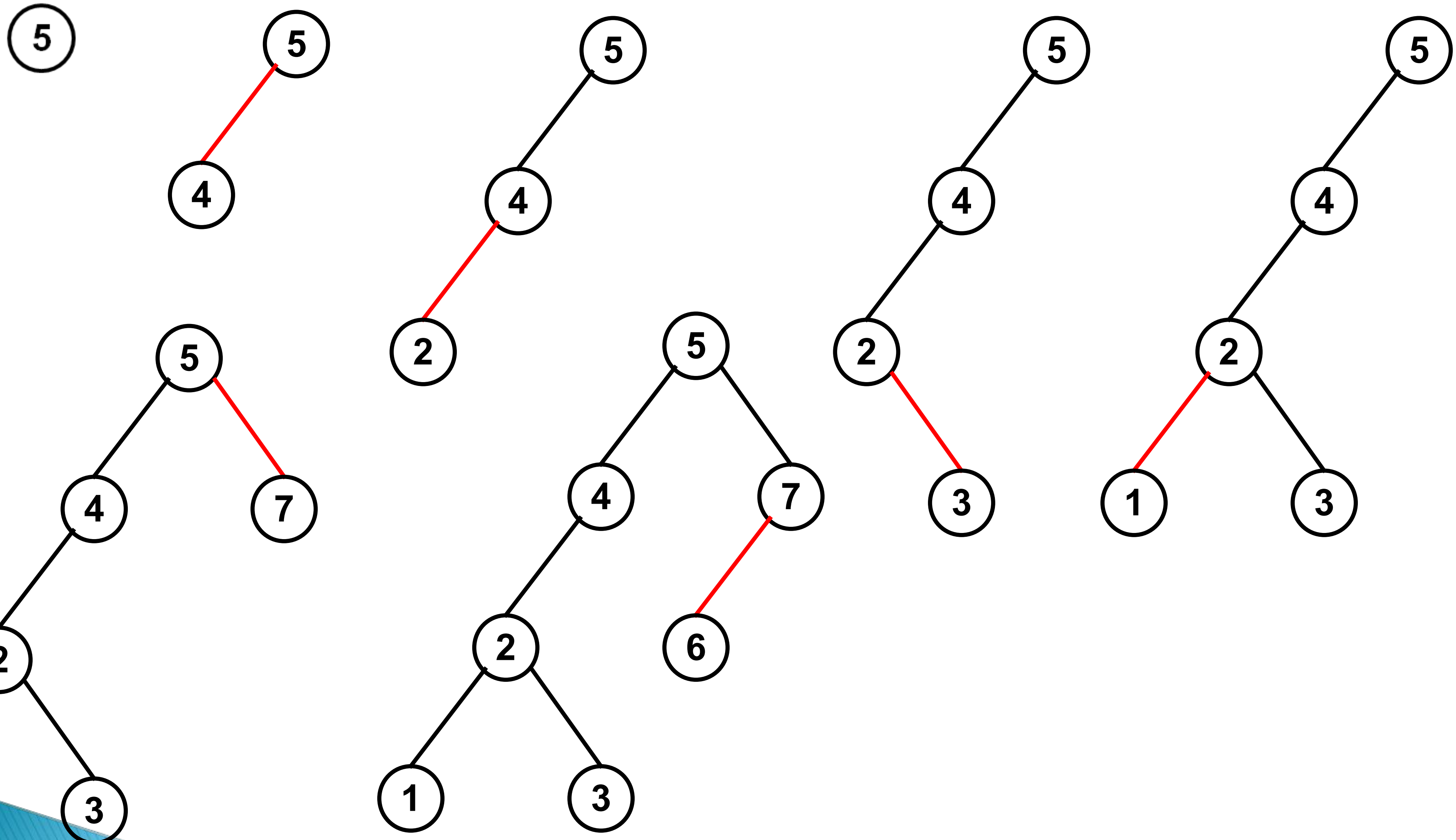
Sir initial:

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

```
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);
     }
  }
```

Arbori binari de cautare

Creare (recursiv)



Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Arbori binari de cautare

Parcurgere (recursiv)

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

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

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

```c
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
```

Arbori binari de cautare

Cautare (iterativ)

```
int search(nod *rad, int x)
{
    while(rad)
    {
        if(rad->info==x)
            return 1;
        if(rad->info<x)
            rad=rad->dr;
        if(rad->info>x)
            rad=rad->st;
    }
    return 0;
}
```

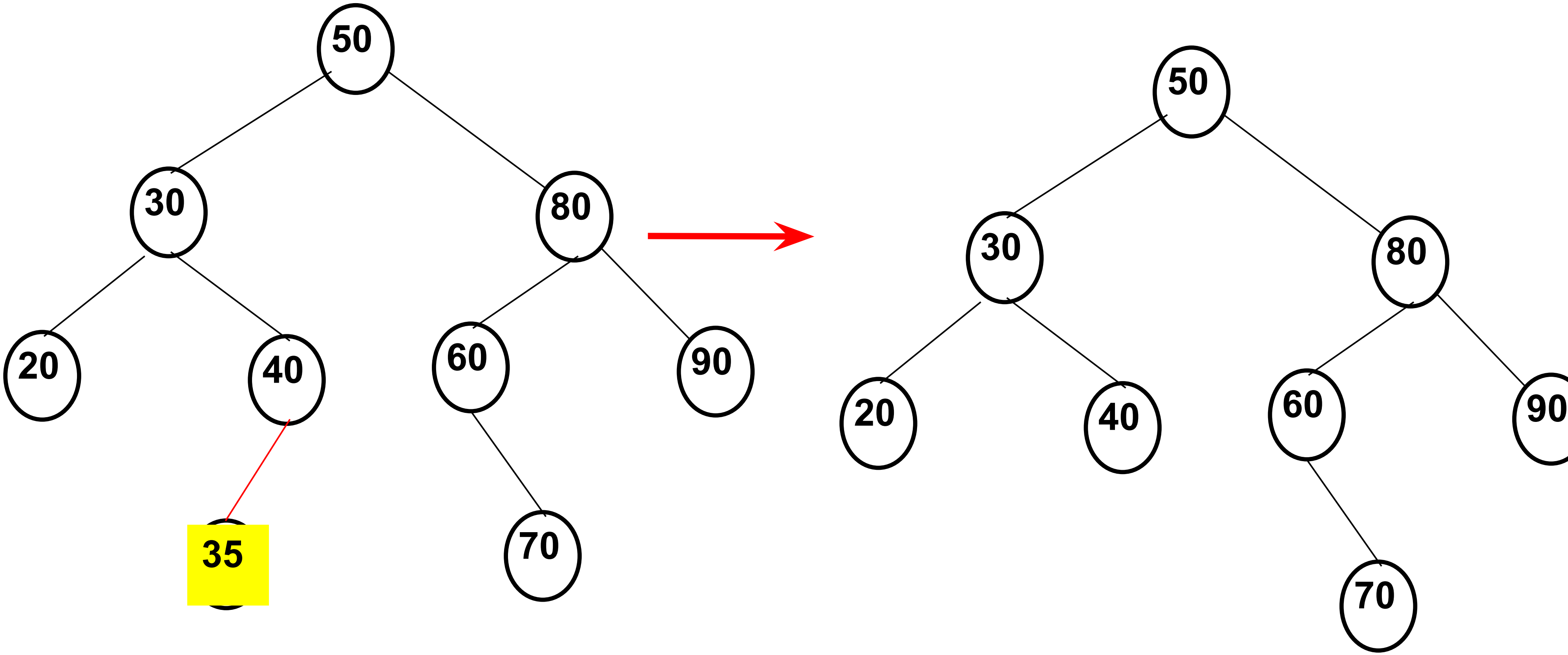
Arbori binari de cautare

Cautare (recursiv)

```
nod* SearchRec (nod* Root, int Val)
{
    if ((Root == NULL) || (Root → info == Val))
        return Root;

     if (Val < Root → info)
            return SearchRec (Root → left, Val);
    else    //Val > Root → info
            return SearchRec (Root → right, Val);
}
```

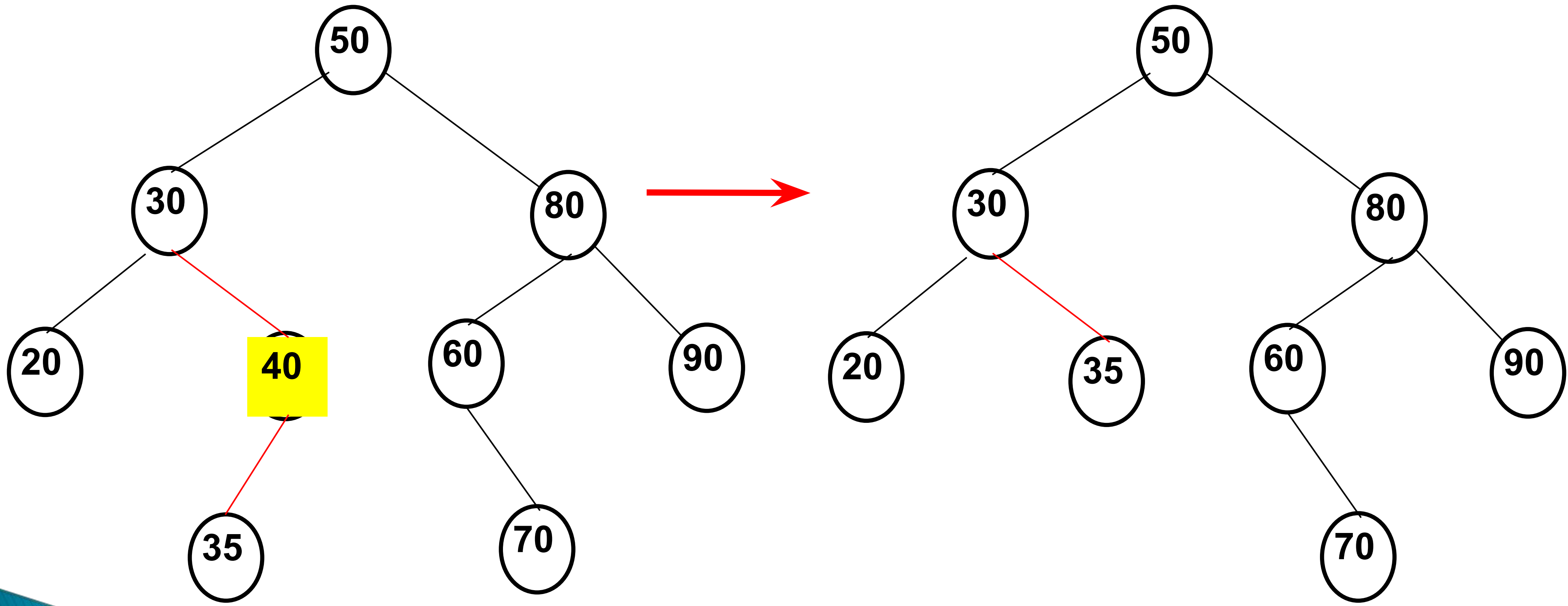
Arbori binari de cautare

Stergerea unei valori
a) Frunza (nu are descendenti)



Arbori binari de cautare

Stergerea unei valori
b) Nodul are un singur descendent



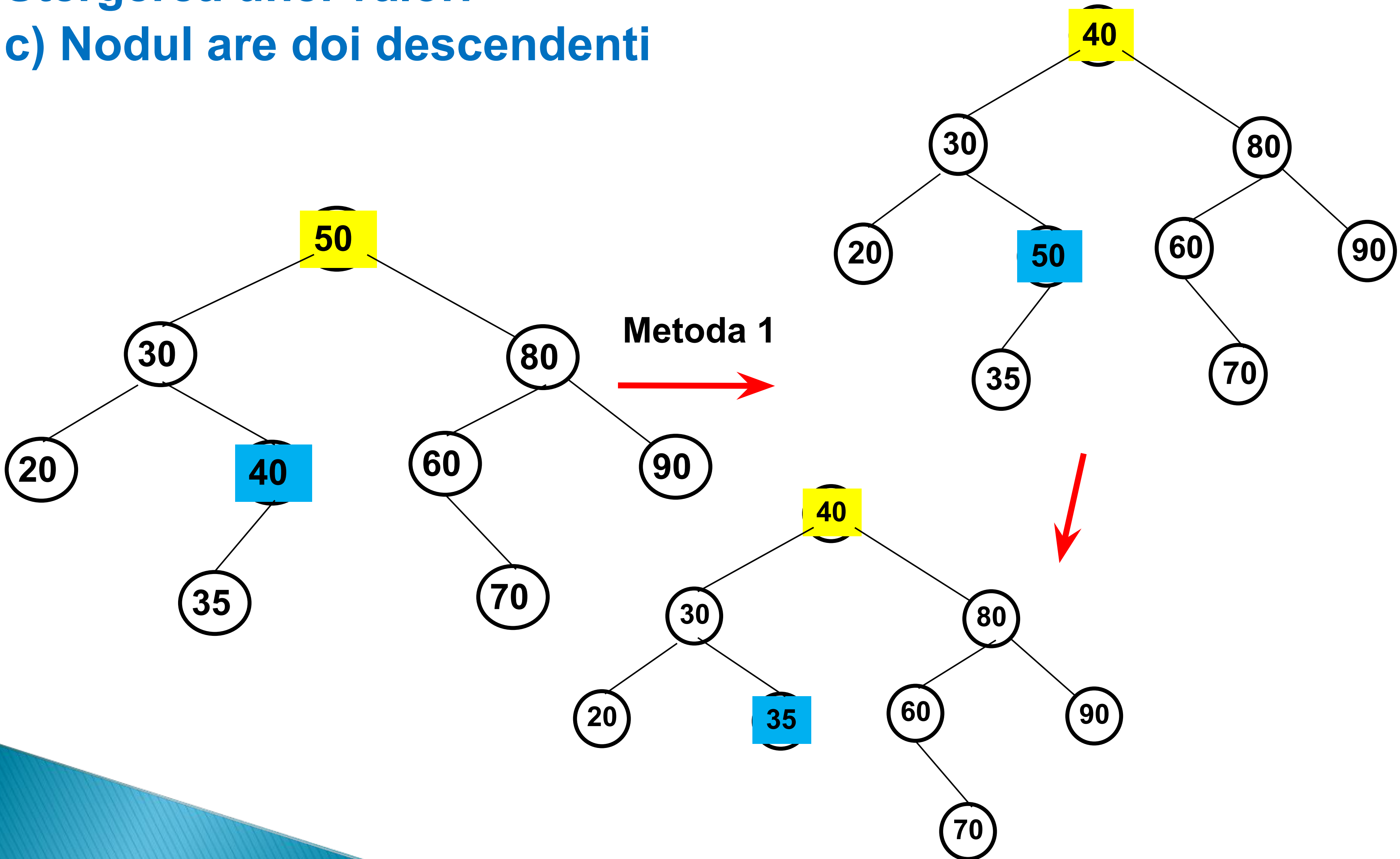
Arbori binari de cautare

Stergerea unei valori
Nodul are cel mult un descendent

```
    //Delete1 sterge nod cu cel mult un fiu nevid,

void Delete1(nod *p)
{   // Şterge nodul p cu cel mult un succesor
    if (p → left == NULL)
        p = p→right;
    else p = p→left;
} // Delete1
```

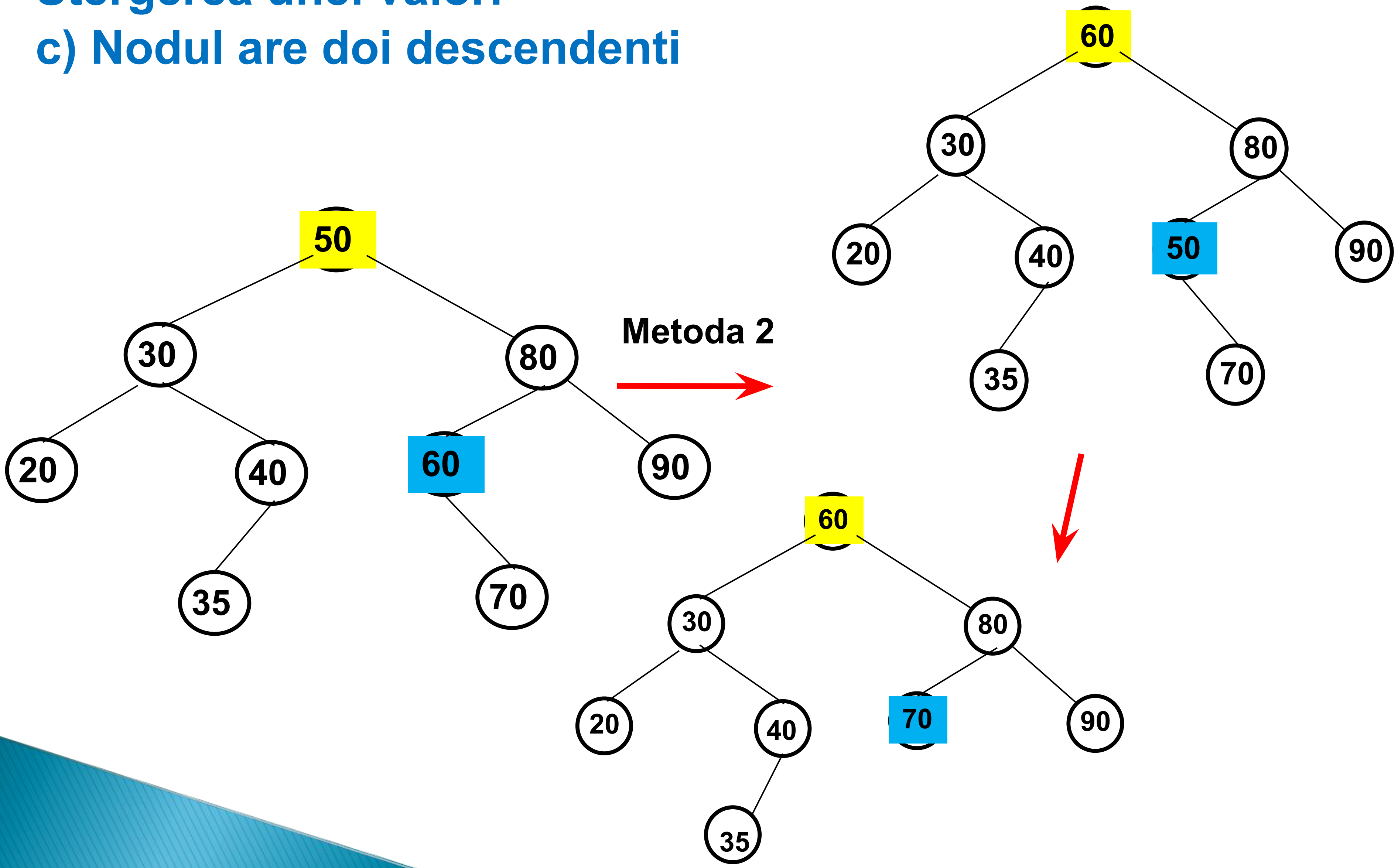
Arbori binari de cautare

Stergerea unei valori
c) Nodul are doi descendenti



Arbori binari de cautare

Stergerea unei valori
c) Nodul are doi descendenti



Stergerea unei valori
Nodul are doi descendenti

void Delete2 (arbore *p)
{   // şterge nodul p cu doi succesori
/* caută predecesorul în inordine al lui p → info mergând un pas la stânga, apoi la dreapta cât se poate .

Parcurgerea se face cu r şi q = tatăl lui r */

arbore *q, *r;   /* d1 = -1 <=> r = q → left */

int d1;   /* d1 = 1 <=> r = q → right */

Arbori binari de cautare

Stergerea unei valori
Nodul are doi descendenti

```
  // (a)
  q = p;
  r  = p → left;
  d1 = -1;

while (r → right != NULL)
  { q = r;
        r  = r → right;
        d1 = 1;   }
```

Arbori binari de cautare

Stergerea unei valori
Nodul are doi descendenti

```
    // (b)
    p → info = r → info;      // Se copiază în p valorile din r
      p → contor = r → contor;


      // (c)  Se leagă de tată, q, subarborele stâng al lui r
    if (d1< 0)
          q → left = r → left;
          else q → right = r → left;
} // Delete 2
```

Arbori binari de cautare

Stergerea unei valori
Nodul are doi descendenti

```
void  Search Del (int x, nod *Root)
{     nod *p1, *p2, *falseroot; int found;
      falseroot = new arbore;
      falseroot → right = Root;   // adăugăm nod marcaj
      p1 = Root; p2 = falseroot;
      d = 1; found = 0;
      while ((p1!= NULL) && (!not found))
            {  p2 = p1;
               if (x < p1 → info)
                  {p2 = p1; p1 = p1 → left; d = -1; }
                       else if (x > p1 → info)
                     {p2 = p1; p1 = p1 → left; d = 1;}
                  else found = 1;}
```

Arbori binari de cautare

Stergerea unei valori
Nodul are doi descendenti

```
    if (!found) // cautare fara succes
    else //found = 1 şi trebuie să şterg nodul p1
        { if ((p1 → left == NULL ) || (p1 → right == NULL))
        Delete1 (p1)  // ştergere caz 1
           else Delete 2 (p1);  // ştergere caz 2
    // legarea noului nod p1 de tatăl său p2
        if (d > 0) p2 → right = p1;
                    else p2 → left = p1;}
} //SearchDel
```

Arbori binari de cautare

Complexitatea operaţiilor la arborele binar de căutare

Operaţiile de inserare şi ştergere de noduri într-un arbore binar de căutare depind în mod esenţial de operaţia de căutare.

Căutarea revine la parcurgerea, eventual incompletă, a unei ramuri, de la rădăcină până la un nod interior în cazul căutării cu succes, sau până la primul fiu vid întîlnit în cazul căutării fără succes (şi al inserării).

Performanţa căutării depinde de lungimea ramurilor pe care se caută;
lungimea medie a ramurilor,
lungime maximă = adâncimea arborelui.

Arbori binari de cautare

Complexitatea operaţiilor la arborele binar de căutare

Forma arborelui, deci şi adâncimea depind, cu algoritmii daţi, de ordinea introducerii cheilor

cazul cel mai nefavorabil, în care adâncimea arborelui este *n,* numărul nodurilor din arbore, adică performanţa căutării rezultă *O(n)*.

Avem (vom demonstra) **o limită inferioară pentru adâncime** de ordinul lui $log_2 n$, ceea ce însemnă că performanţa operaţiei de căutare nu poate coborâ sub ea.