ARBORI BINARI DE CĂUTARE⟹IMPLEMENTARE DINAMICĂ

CURS 9- 27.04.2021

Titular: Şef. Lucr. Dr. Mat. Cărbureanu Mădălina

Universitatea Petrol-Gaze din Ploieşti

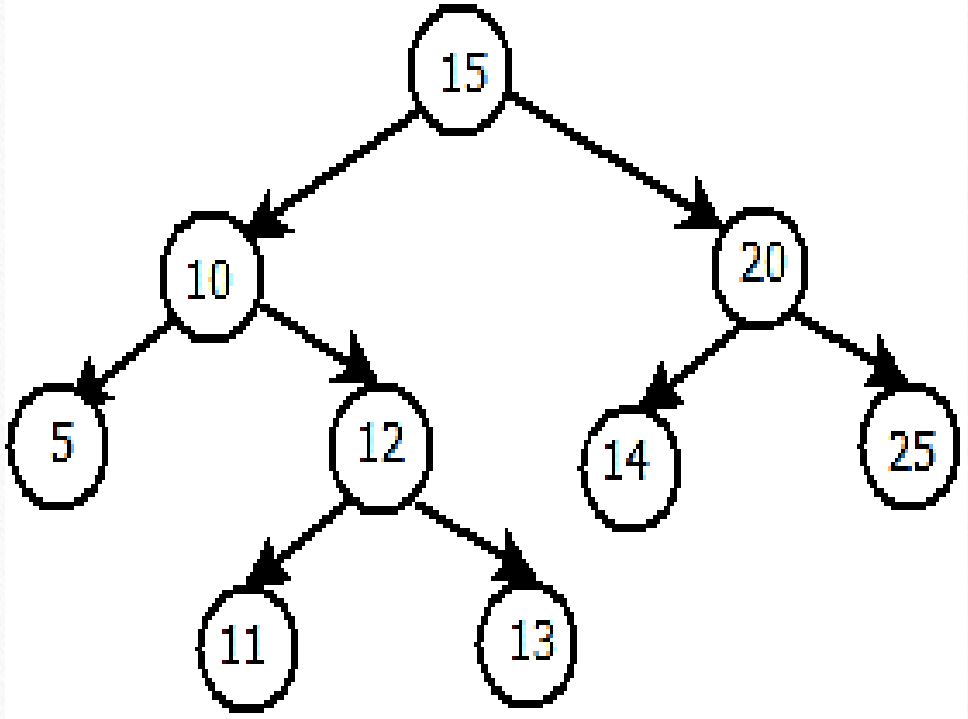
OBIECTIVE:

- **NOȚIUNE ARBORE BINAR DE CĂUTARE (Binary Search Tree-BST);**
- **EXEMPLU BST;**
- **DECLARARE BST;**
- **OPERAȚII DE BAZĂ BST;**
- **EXEMPLU APLICAȚIE;**
- **APLICAȚII PROPUSE.**

NOȚIUNE ARBORE BINAR DE CĂUTARE (Binary Search Tree-BST)

Alocare dinamică memorie

- **Arbore binar de căutare** ⇒ un **arbore de căutare (BST**) este un arbore binar, în care fiecare nod prezintă o așa numită cheie de identificare și următoarele proprietăți :
  - toate cheile nodurilor subarborelui stâng sunt mai mici ca și valoare decât cheia asociată nodului curent;
  - toate cheile nodurilor subarborelui drept sunt mai mari decât cheia asociată nodului curent.



DECLARARE BST

```
typedef struct arbore
{int inf;
struct arbore*st,*dr;
}Search_BTree;
```

- fiecare nod al arborelui binar de căutare prezintă trei câmpuri: *inf*, **st*, **dr*;
- câmpul *inf* reprezintă informația utilă a nodurilor din arbore;
- câmpul *st* reprezintă adresa la subarborele stâng, acesta fiind un pointer către tipul de date *struct arbore*;
- câmpul *dr* reprezintă adresa la subarborele drept, acesta fiind un pointer către tipul de date *struct arbore*.

OPERAȚII DE BAZĂ BST

- **crearea arborelui binar de căutare** ⟹ inserarea repetată a unui nod în arbore;
- **parcurgerea arborelui binar de căutare:** parcurgerea în adâncime (RSD, SRD, SDR) și parcurgerea în lațime/pe niveluri (BFS) ⟺ parcurgere arbori binari;
- ștergerea unui nod din arborele binar de căutare;
- căutarea unei chei într-un arbore binar de căutare.

OPERAȚII DE BAZĂ BST ⇨ CREARE BST

- Presupune alegerea unei reprezentări sub care formă este memorat arborele BST ⇨ implementarea dinamică, prin utilizarea pointerilor și a recursivității directe;
- Constă în aplicarea unui număr de ori a operației de inserare a unei chei în arbore, ținând cont de proprietățile BST.

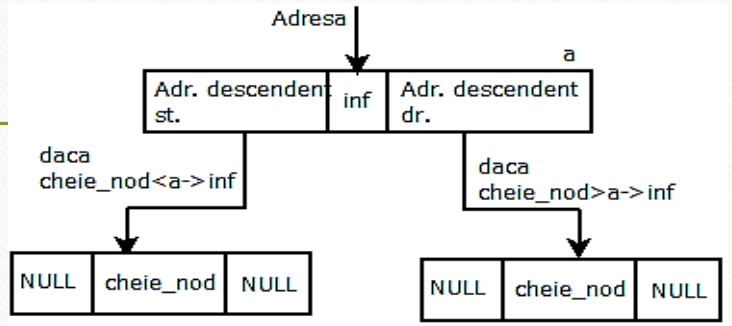
OPERAȚII DE BAZĂ BST ⟹ CREARE BST

- **Pași:**
- dacă nodul cu adresa primită (*Search_BTree*&a*) nu este nod frunză (*if(a!=NULL)*), adică nodul curent prezintă subarbore stâng și respectiv drept vid, atunci se compară valoarea *cheie_nod* de inserat cu cheia asociată nodului curent *a-→inf* și apar următoarele trei situații:
  - cheia nodului de inserat coincide cu cheia nodului curent (*cheie_nod==a-→inf*), deci nodul există deja în BST, caz în care se renunță la inserare;
  - cheia nodului de inserat este mai mare decât cheia nodului curent (*a→inf<cheie_nod*), caz în care cheia noului nod este inserat în subarborele drept al nodului curent (*inserare_nod(a→dr,cheie_nod);*);
  - cheia nodului de inserat este mai mică decât cheia nodului curent (*a→inf>cheie_nod*), caz în care cheia noului nod este inserat în subarborele stâng al nodului curent (*inserare_nod(a→st,cheie_nod);*);
- dacă nodul cu adresa primită (*Search_BTree*&a*) este nod frunză (*if(a==NULL)*) atunci:
  - se alocă memorie pentru acesta *nod= (Search_BTree*) malloc (sizeof(Search _BTree));*
  - se completează informația utilă a acestuia (*nod→inf=cheie_nod;*);
  - se completează subarborele stâng și drept al acestuia cu NULL (*nod→st = NULL; nod→dr=NULL;)* deoarece nodul curent este nod terminal;
  - nodul curent este nod terminal (*a=nod;*).

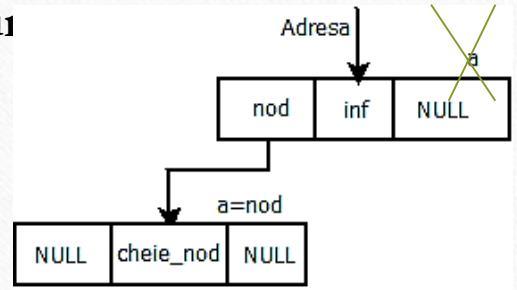
OPERAȚII DE BAZĂ BST ⇨ CREARE BST

```
void inserare_nod (Search_BTree * &a,
intcheie_nod)
{Search_BTree *nod;
 if(a!=NULL)
   if(cheie_nod==a-→inf)      cout<<"\n Nodul
exista deja in BST!";
      else if(a→inf<cheie_nod)
inserare_nod(a→dr,cheie_nod);
           else inserare_nod(a→st,cheie_nod);
 else if(a==NULL)
 {
nod=(Search_BTree*)malloc(sizeof(Search_BTre
e));
  nod→inf=cheie_nod;
  nod→st=NULL;
  nod→dr=NULL;
  a=nod;}
}
```

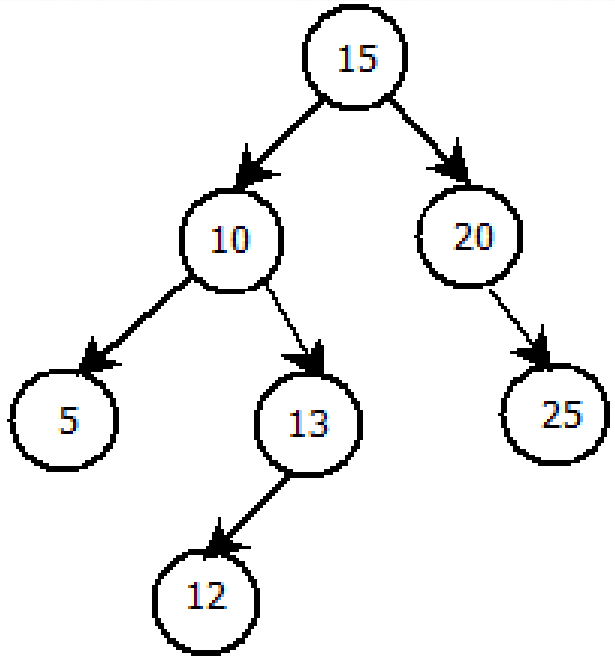
Dacă nodul cu adresa primită (*Search_BTree*&a*) nu este nod frunză (*if(a!=NULL)*):



Dacă nodul cu adresa primită (*Search_BTree*&a*) este nod frunză (*if(a==NULL)*) și dacă informația nodului de inserat (*nod*) este mai mică decât informația nodului terminal (*a*), atu



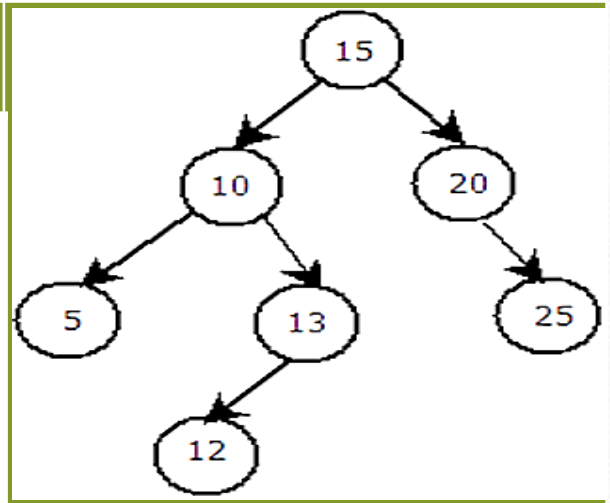
OPERAȚII DE BAZĂ BST ⟹ CREARE BST



Dati nr. de noduri pt. BST: 7
Cheia nodului de inserat este: 15
Cheia nodului de inserat este: 10
Cheia nodului de inserat este: 5
Cheia nodului de inserat este: 13
Cheia nodului de inserat este: 12
Cheia nodului de inserat este: 20
Cheia nodului de inserat este: 25

OPERAȚII DE BAZĂ BST ⟹ Parcurgere BST

RSD	SRD	SDR
*void RSD(Search_BTree*t)* *{if(t!=NULL)* *{cout<<t→inf<<",";* *RSD(t→st);* *RSD(t→dr);* *}* *}*	*void SRD(Search_BTree*t)* *{if(t!=NULL)* *{SRD(t→st);* *cout<<t→inf<<",";* *SRD(t→dr);* *}* *}*	*void SDR(Search_BTree*t)* *{if(t!=NULL)* *{SDR(t→st);* *SDR(t→dr);* *cout<<t→inf<<",";* *}* *}*



RSD: 15, 10, 5, 13, 12, 20, 25
SRD: 5, 10, 12, 13, 15, 20, 25.
SDR: 5, 12, 13, 10, 25, 20, 15.

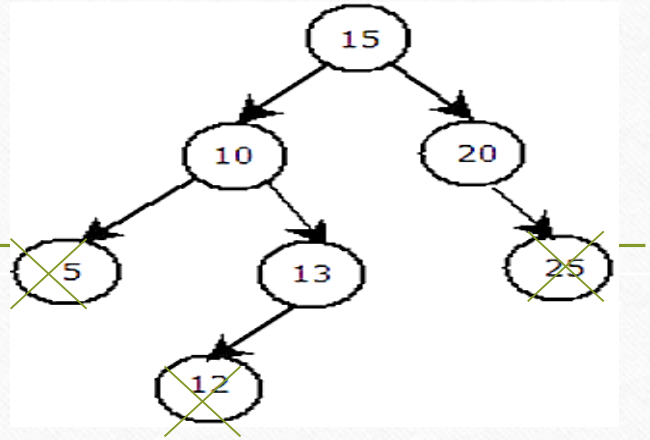
- **PARCURGEREA PE NIVELURI (BFS) ⟹ FIG. 7.36, PAG. 150 ⟹ EXEMPLU APLICAȚIE;**

OPERAȚII DE BAZĂ BST ⟹ ȘTERGEREA UNUI NOD DIN BST

- **Caz I: nodul ce se dorește a fi șters este nod terminal** (*(nod→st==0)&&(nod → dr==0)*) în BST, caz în care se eliberează zona de memorie ocupată de nodul terminal (*delete nod;)* și adresa acestuia se înlocuiește cu valoarea 0 (*nod=0;)* respectiv nodul părinte al nodului terminal va avea adresa 0 către acesta;
- **Caz II:nodul ce se dorește a fi șters prezintă doar subarbore drept** (*nod→st==0*), caz în care adresa nodului părinte către acest nod este înlocuită cu adresa subarborelui drept (*aux=nod→dr; delete nod;nod=aux;*) al nodului în cauză;
- **Caz III: nodul ce se dorește a fi șters prezintă doar subarbore stâng** (*nod→dr==0*), caz în care adresa nodului părinte către acest nod este înlocuită cu adresa subarborelui stâng (*aux=nod→st; delete nod;nod=aux;*) al nodului în cauză;
- **Caz IV: nodul ce se dorește a fi șters prezintă doi subarbori** (subarbore stâng și subarbore drept), caz în care în locul informației nodului șters ce va trece cea mai mare valoare din subarborele stâng sau subarborele drept al nodului ⟹ *Temă* .

OPERAȚII DE BAZĂ BST ⇨ ȘTERGEREA UNUI NOD DIN BST

Caz I: nodul ce se dorește a fi șters este nod terminal;




```
void sterg_nod_terminal
(Search_ BT ree *&nod,int
info)
{Search_BTree*aux;
if(nod!=NULL)
   {if(nod→inf==info)
      {if((nod→st==0)&&(nod-
→dr== 0))
        {delete nod;
         nod=0;
        }
      }
```

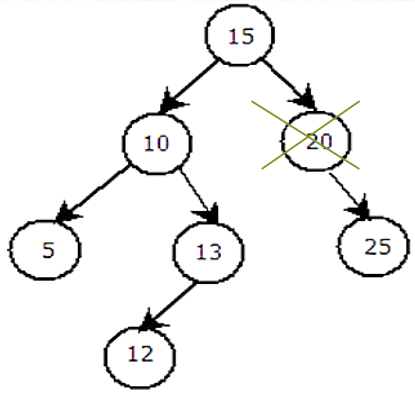
```
 else if(nod→inf<info)

sterg_nod_terminal(nod→dr,
info);
        else
sterg_nod_terminal(nod→st,
info);
   }
}
```

RSD: 15, 10, 5, 13, 12, 20, 25
Informația nodului terminal: 12
BST fără nod terminal 12:
15, 10, 5, 13, 20, 25
Informația nodului terminal: 5
BST fără nod terminal 5:
15, 10, 13, 12, 20, 25
Informația nodului terminal: 25
BST fără nod terminal 25:
15, 10, 5, 13, 12, 20

OPERAȚII DE BAZĂ BST ⇨ ȘTERGEREA UNUI NOD DIN BST

- Caz II:nodul ce se dorește a fi șters prezintă doar subarbore drept;

```
void sterg_nod_subdr (Search_BTree
*&nod,int info)
{Search_BTree*aux;
 if(nod!=NULL)
    {if(nod→inf==info)
          {if(nod→st==0)
             {aux=nod→dr;
              delete nod;
              nod=aux;}
          }
      else if(nod→inf<info)
         sterg_nod_subdr(nod→dr,info);
         else sterg_nod_subdr(nod→st, info);}}
```

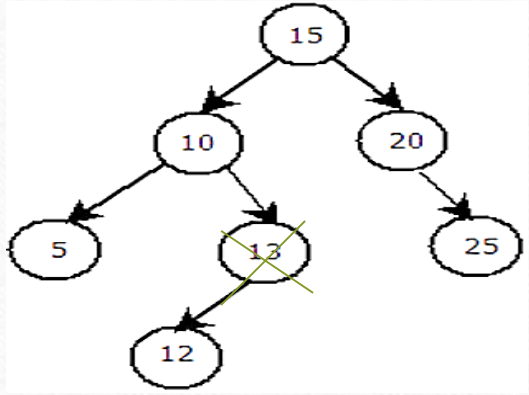


RSD: 15, 10, 5, 13, 12, 20, 25
Informația nodului ce prezintă doar subarbore drept: 20
BST fară nodul ce prezintă doar subarbore drept:
15, 10, 5, 13, 12, 25

OPERAȚII DE BAZĂ BST ⟹ȘTERGEREA UNUI NOD DIN BST

Caz III: nodul ce se dorește a fi șters prezintă doar subarbore stâng;

```
void sterg_nod_subst(Search_BTree* &nod,
int info)
{Search_BTree*aux;
 if(nod!=NULL)
    {if(nod→inf==info)
        {if(nod→dr==0)
          {aux=nod→st;
           delete nod;
           nod=aux;}
        }
       else if(nod→inf<info)
          sterg_nod_subst(nod→dr,info);
          else sterg_nod_subst(nod→st,info);}}
```



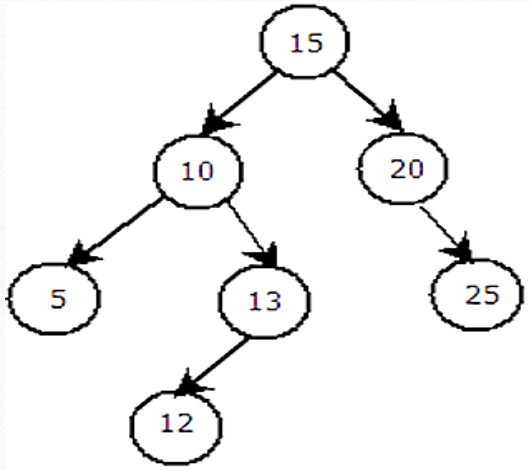
RSD: 15, 10, 5, 13, 12, 20, 25
Informația nodului ce prezintă doar subarbore stâng: 13
BST fară nodul ce prezintă doar subarbore stâng:
15, 10, 5, 12, 20, 25

OPERAȚII DE BAZĂ BST ⇨ căutarea unei chei într-un BST

- Operația de **căutarea unei chei într-un arbore binar de căutare** ⇨ una dintre cele mai frecvente operații care se pot aplica asupra unui BST.
- Constă în compararea cheii de căutat cu informația utilă a nodului rădăcină, iar dacă aceasta este mai mică ca și valoare numerică, se caută cheia în subarborele stâng, altfel (valoarea cheii este mai mare decât informația utilă a nodului rădăcină) se caută în subarborele drept.

OPERAȚII DE BAZĂ BST ⇨ căutarea unei chei într-un BST

```
void caut_cheie_ BST (Search_BTree *&nod,
int val_cheie)
{if(nod!=NULL)
  {if(nod→inf==val_cheie) cout<<"Cheia
cautata"<<val_cheie<<"este element al
BST!";
      else{if(nod→inf<val_cheie)     caut _
cheie_ BST(nod→dr,val_cheie);
            else
caut_cheie_BST(nod→st,val_cheie);}
  }
  else cout<<"Cheia cautata nu este element al
BST!";}
```



RSD: 15, 10, 5, 13, 12, 20, 25
Valoare cheie căutată: 13
Cheia cautată 13 este element al BST!
Valoare cheie căutată: 30
Cheia cautată nu este element al BST!

EXEMPLU APLICAȚIE → PARCURGEREA BFS A UNUI BST

```
#include<iostream.h>
typedef struct arbore
{int inf;
 struct arbore*st,*dr;
}Search_BTree;
Search_BTree *bst;
int i,n,x;
void inserare_nod (Search_Btree *&a,int cheie_nod)
{Search_BTree *nod;
 if(a!=NULL)
   if(cheie_nod==a→inf) cout<<"\n Nodul exista deja in BST!";
      else if(a→inf<cheie_nod) inserare_nod(a→dr, cheie_nod);
         else inserare_nod(a→st,cheie_nod);
 else if(a==NULL)
 {nod=(Search_BTree*)malloc(sizeof(Search_BTree));
  nod→inf=cheie_nod; nod→st=NULL; nod→dr=NULL; a=nod;} }
```

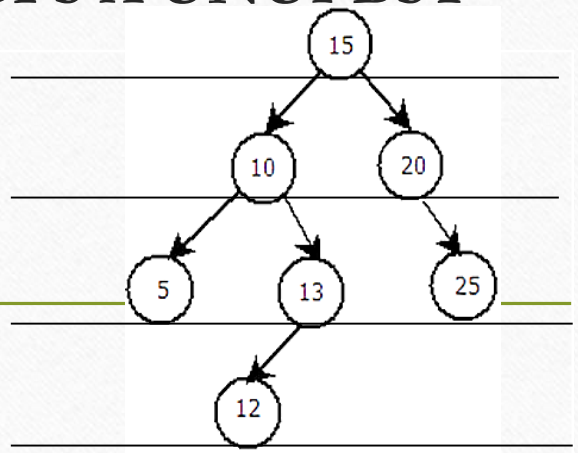
```
void RSD(Search_BTree*t)
 {if(t!=NULL)
    {cout<<t→inf<<",";
     RSD(t→st);
     RSD(t→dr);}
 }
int  inaltime(Search_Btree *t)
{if(t!=NULL)
 {if(inaltime  (t→st)<inaltime(t  → c
 ))
       return inaltime(t→dr)+1;
       else return inaltime(t→st)+1;
     }
  return 0;
 }
```

```
void  noduri_nivel  (Search_Btree
* rad, int niv)
{if(rad!=NULL)
    {if(niv==1)
      cout<<rad→inf<<",";
     else if(niv>1)
       {noduri_nivel(rad→st,niv-1);
        noduri_nivel(rad→dr,niv-1)}
     }
}
```

```
void afisare_nivele_BST (Search_
Btree *rad)
{int h=inaltime(rad);
 for(int i=1;i<=h;i++)
  noduri_nivel(rad,i);
}
```

EXEMPLU APLICAȚIE ⇨ PARCURGEREA BFS A UNUI BST

```
void main()
{clrscr();
int nivel;
cout<<"\n Dati nr. de noduri pt. BST:";
cin>>n;
for(i=1;i<=n;i++)
{cout<<"\n Cheia nodului de inserat este:";
 cin>>x;
 inserare_nod(bst,x);}
cout<<"\n BST parcurs RSD este:";
RSD(bst);
cout<<"\n Inaltimea BST este:"<<inaltime(bst);
cout<<"\n Nivel:";cin>>nivel;
cout<<"\n Nodurile de pe nivelul "<<nivel<<"sunt:";
noduri_nivel(bst,nivel);
cout<<"\n BST parcurs pe nivele-BFS este:";
afisare_nivele_BST(bst);
getch();
```



BST parcurs RSD este:

15, 10, 5, 13, 12, 20, 25

Inalțimea BST este: 4

Nivel: 3

Nodurile de pe nivelul 3 sunt:

5, 13, 25

BST parcurs pe nivele-BFS este:

15, 10, 20, 5, 13, 25, 12

APLICAȚII PROPUSE

- Testarea operaţiilor de bază BST;
- Testarea aplicaţiei exemplu: BFS a unui BST;
- Aplicaţii propuse: lucrare de laborator nr. 11, pag. 161;
- Obs: se va utiliza cartea "*Elemente de proiectarea algoritmilor. Ghid teoretic și practic*", Șef. lucr. dr. mat. Cărbureanu Mădălina, Editura Universităţii Petrol-Gaze din Ploieşti, 2021.

Să vă fie de folos și spor la lucru!