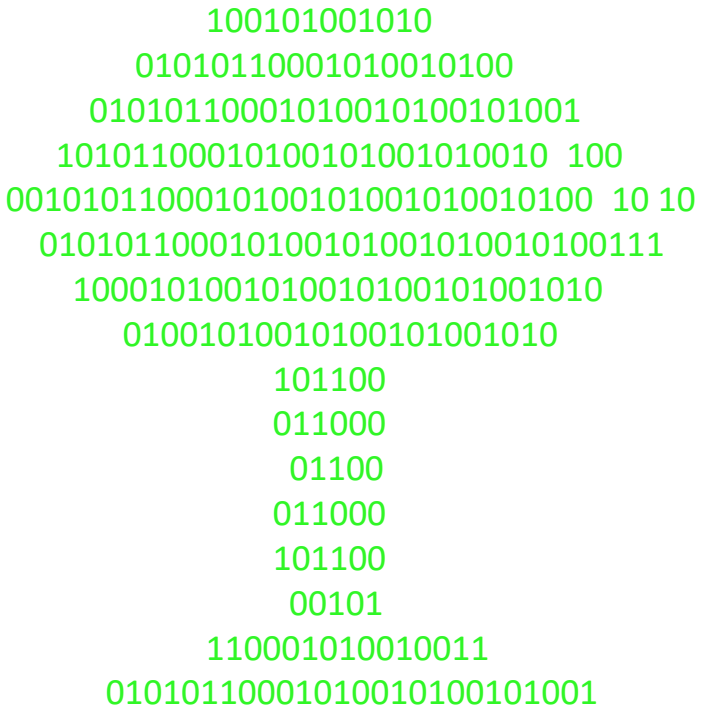
Structuri de date si algoritmi

Curs, AIA – An II



Arbori binari de cautare

BST – Binary Search Tree

Definitie

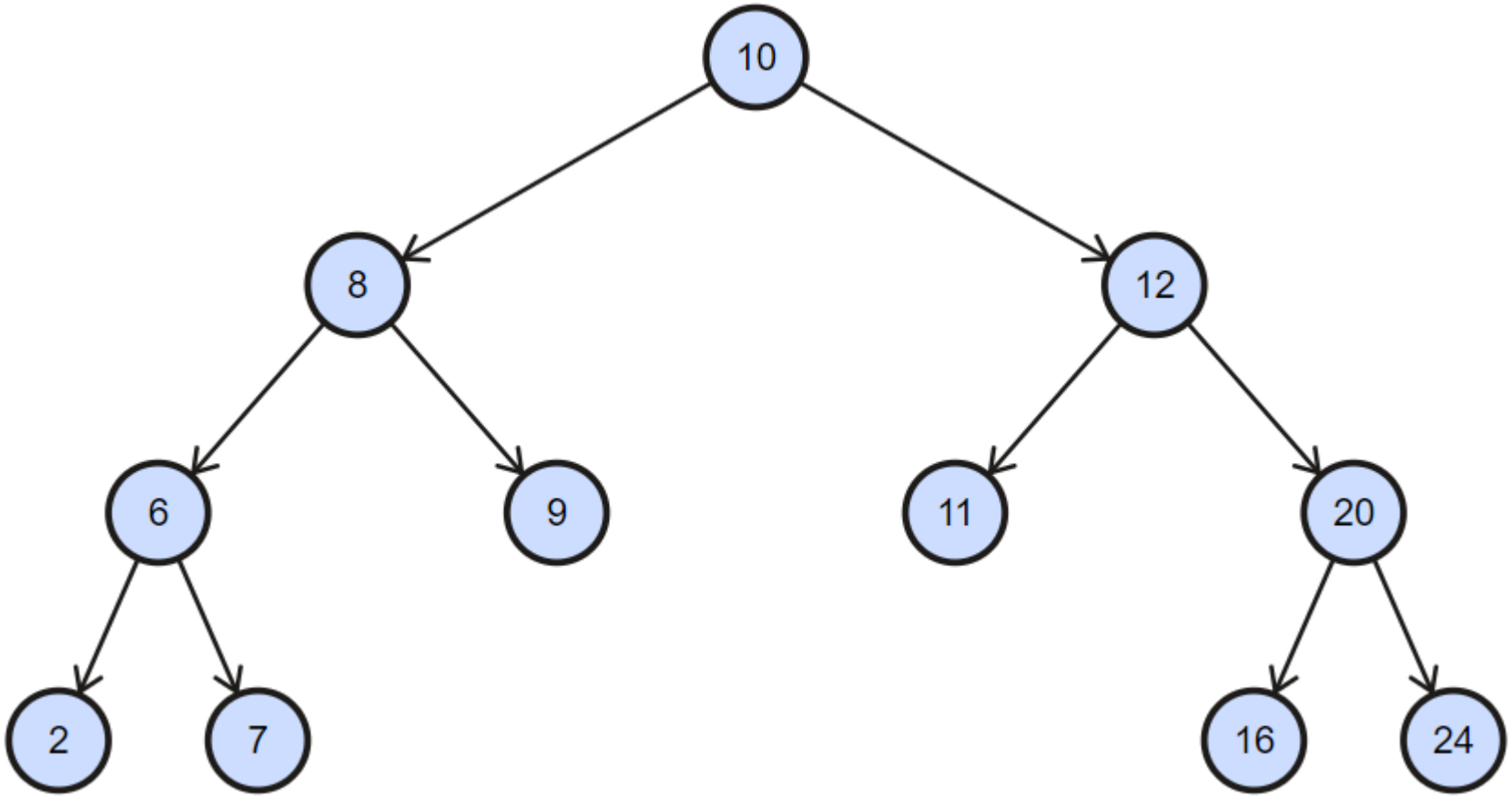
- Fiecare varf (nod) contine un camp cheie a.i. pe multimea constantelor tipului cheii sa existe o relatie de ordine

- Daca ***x*** este un nod oarecare:
  - orice cheie asociata unui nod al subarborelui stang este strict mai mica decat cheia asociata nodului ***x***
  - orice cheie asociata unui nod al subarborelui drept este strict mai mare decat cheia asociata nodului ***x***

BST – Binary Search Tree

Exemple

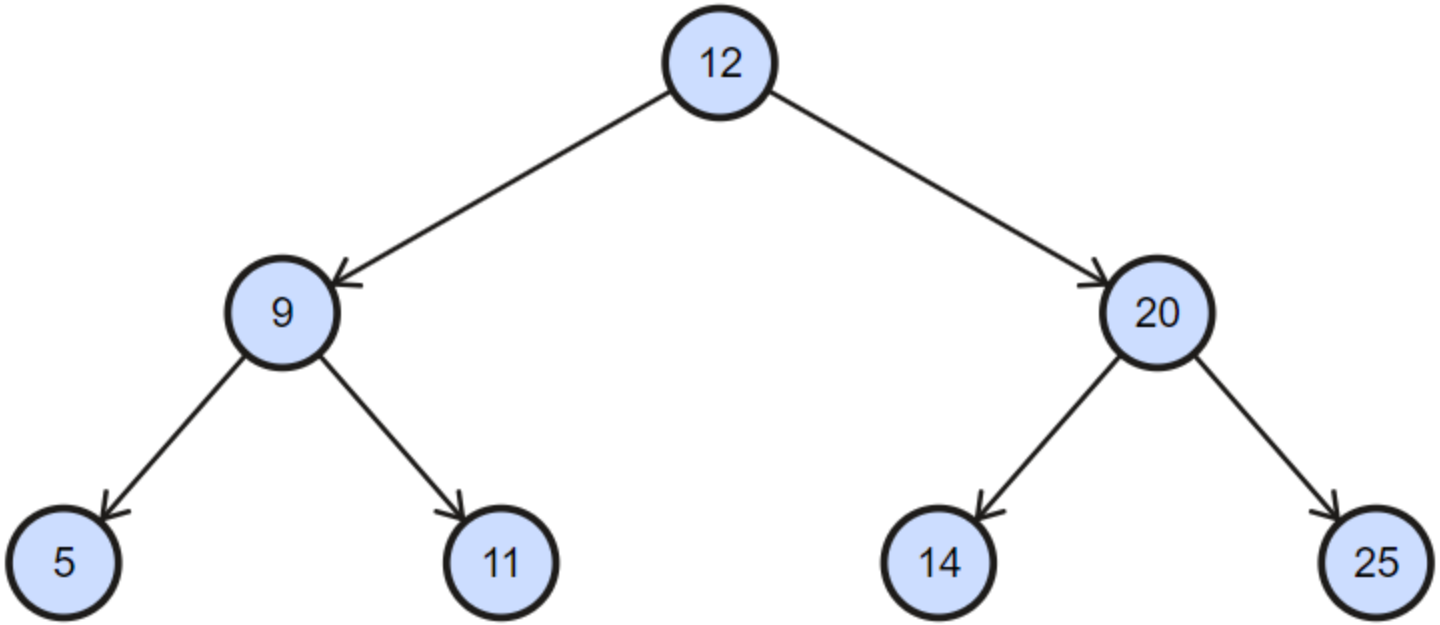
BST



BST – Binary Search Tree

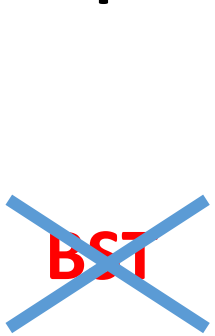
Exemple

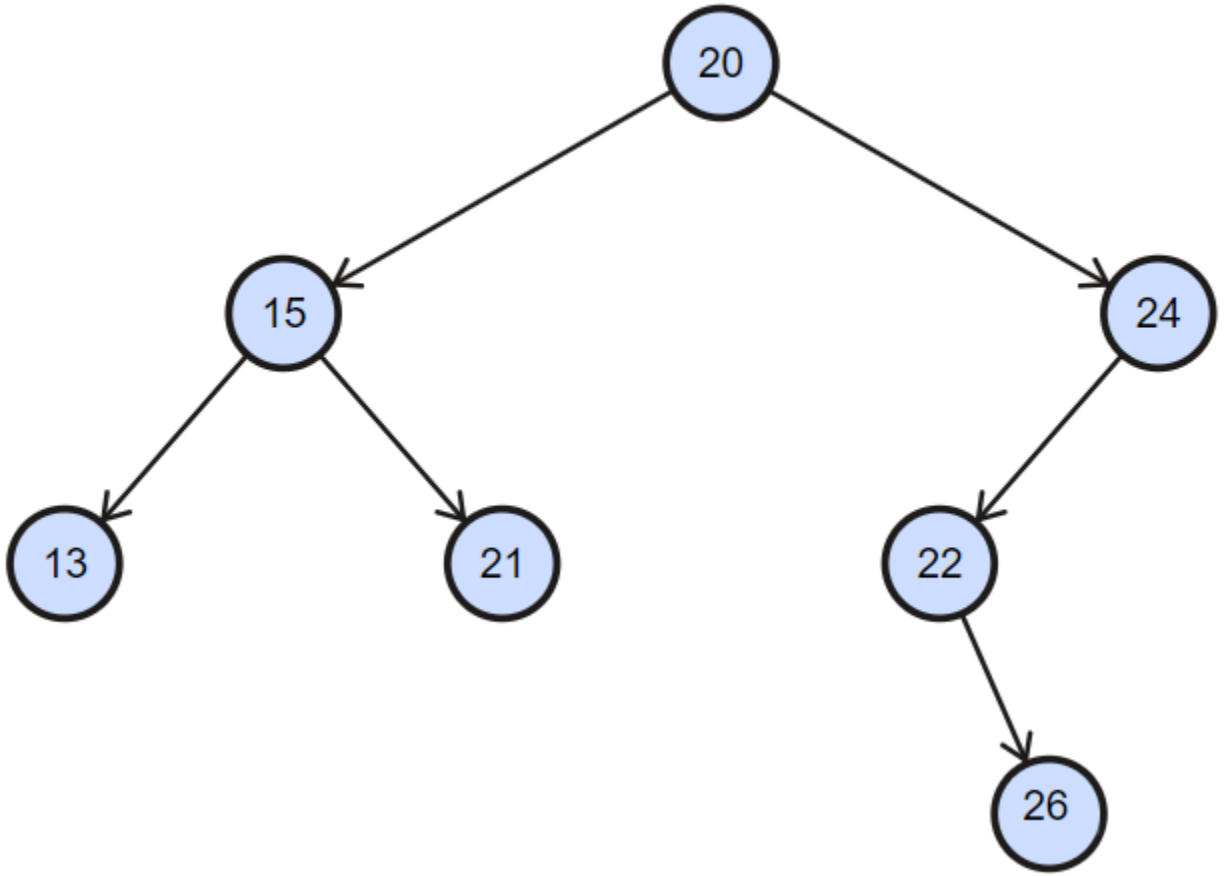
BST



BST – Binary Search Tree

Exemple





BST – Binary Search Tree

Observatii

- Cheile de identificare sunt distincte
- Parcurgerea in inordine → sortare crescatoare a cheilor
- Multimea cheilor – Dictionar (D)

BST – Binary Search Tree

Operatii peste multimea D:

Search(D,k)

- Cauta atomul cu cheia k in D si il returneaza sau determina daca acesta exista in dictionar
- Variatii: FindMin, FindMax

Insert(D,k)

- Insereaza atomul cu cheia k in dictionar, daca nu exista deja

Delete(D,k)

- Sterge atomul cu cheia k din D, daca exista in dictionar

BST – Binary Search Tree

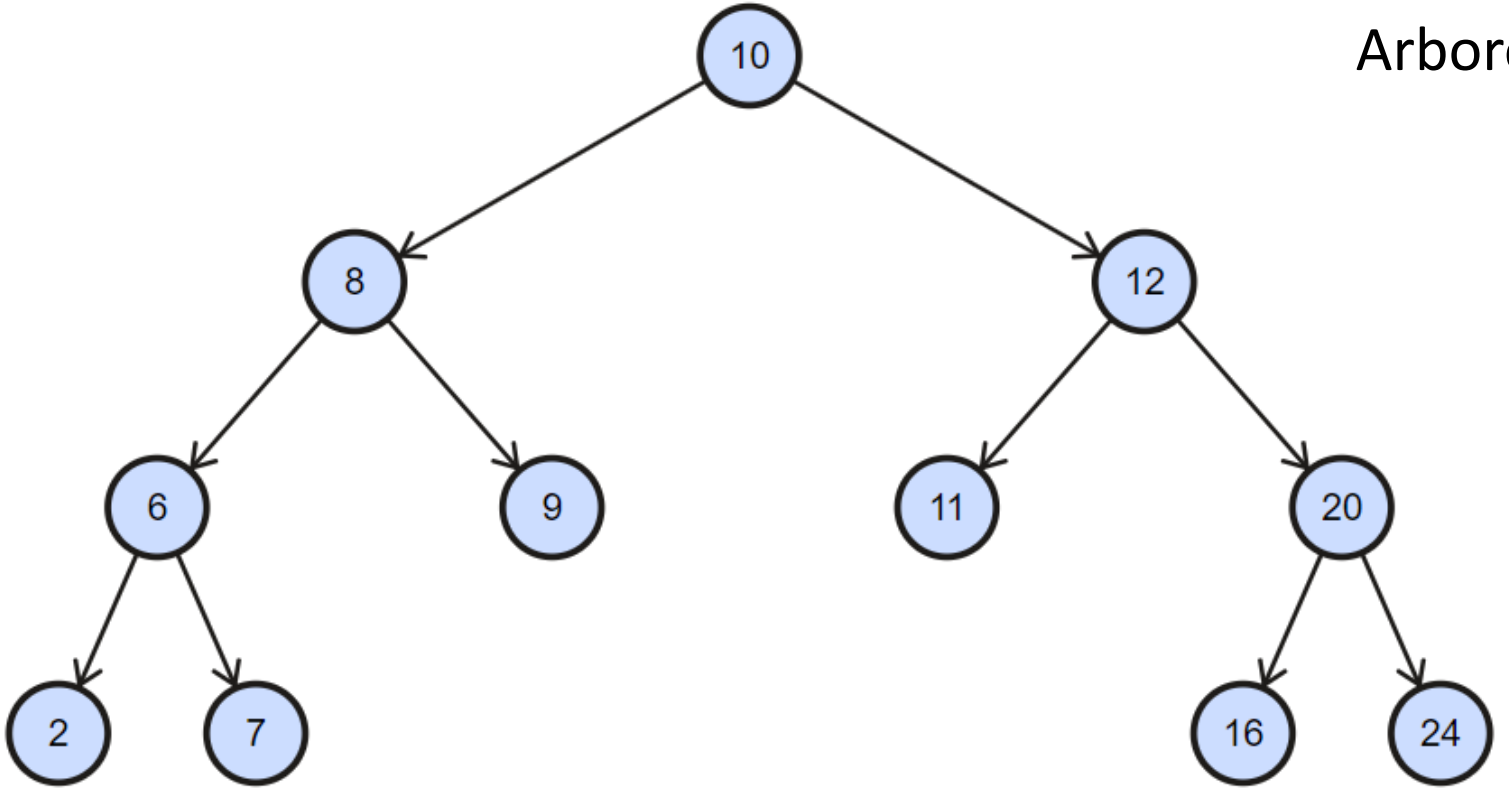
```
struct Nod{
    Atom data;
    Nod *stg, *drt;
};
```

- cheia poate fi
  - componenta a atomului
  - o valoare asociata conform unei formule de calcul

    functia `key(data(x))` – returneaza cheia asociata nodului x

BST – Binary Search Tree

Cautare



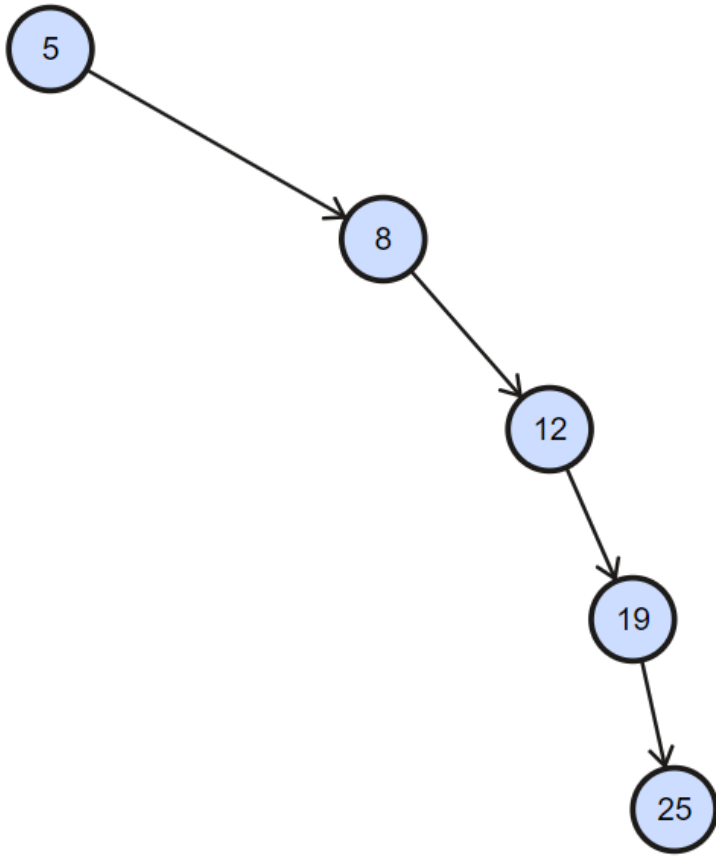
Complexitate: O(h)

Arbore total echilibrat cu ***n*** noduri:

h = log(***n***)

BST – Binary Search Tree

Cautare



Complexitate: O(h)

Arbore total echilibrat cu ***n*** noduri:

h = log(***n***)

n noduri introduse in ordine crescatoare/descrescatoare:

h = ***n*** !!!

BST – Binary Search Tree

Cautare - recursiv

```
Search(r, k)
        if( r=0 OR key(data(r) = k)
                return r
        endif
        if( k < key(data(r) )
                return Search(stg(r), k)
        else
                if( key(data(r) < k)
                        return Search(drt(r), k)
                endif
        endif
end
```

Complexitate: O(h)

BST – Binary Search Tree

Cautare - iterativ

```
Search(r, k)
        p := r
        while( p!=0 AND key(data(p))!=k) do
                if( k < key(data(p)))
                        p := stg(p)
                else
                        p := drt(p)
                endif
        endwhile
        return p
end
```

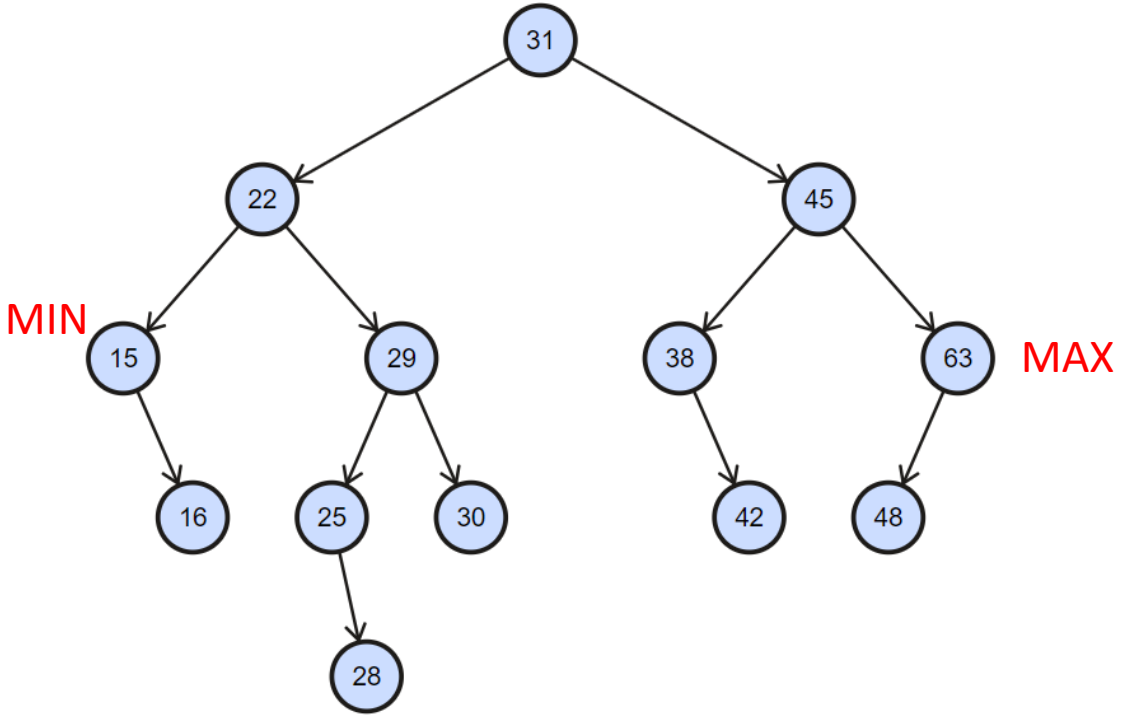
Complexitate: O(h)

BST – Binary Search Tree

Cautare

FindMin / FindMax

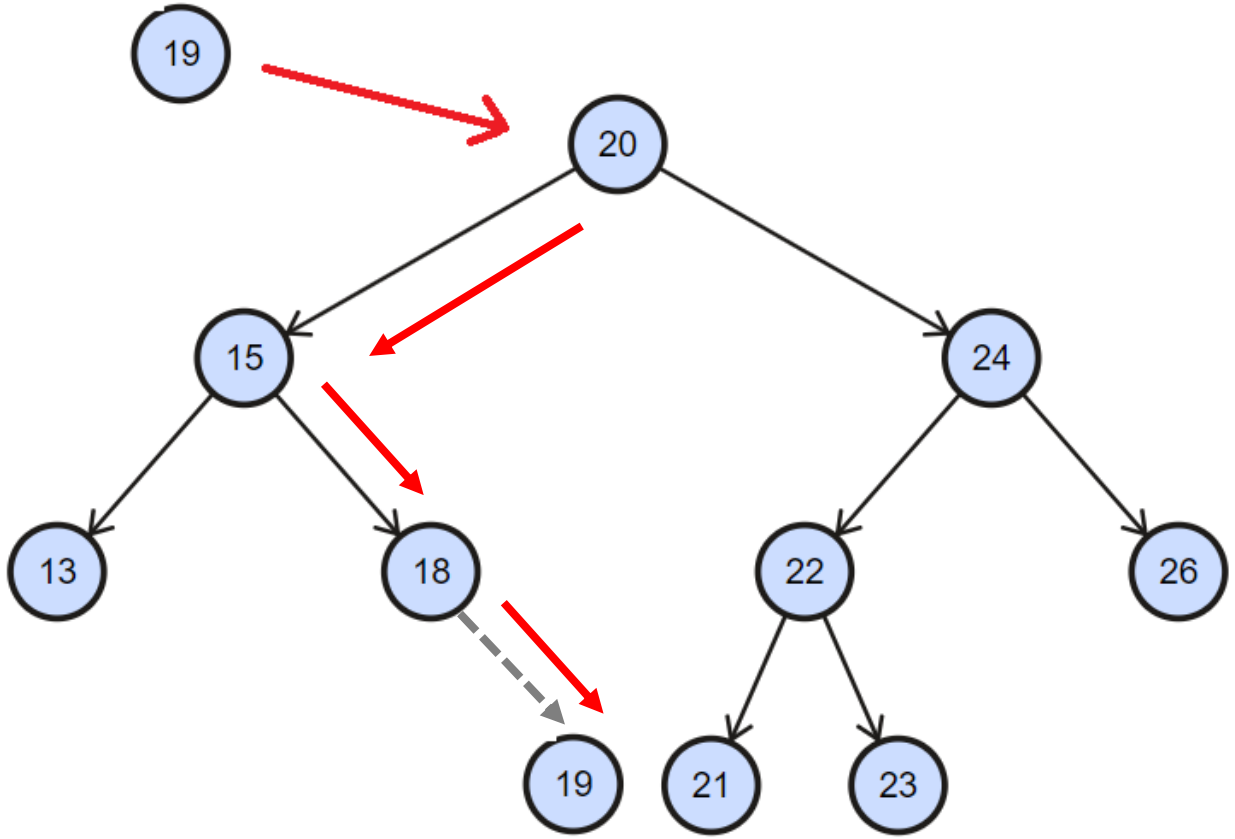
Se parcurge arborele pe ramura stanga / dreapta pana la ultimul nod



O(h)

BST – Binary Search Tree

Inserare



Complexitate: O(h)

Arbore total echilibrat cu ***n*** noduri:

h = log(***n***)

```
MakeNod(a)
        p := get_space()
        data(p) := a
        stg(p) :=0
        drt(p) :=0
        return p
end
```

BST – Binary Search Tree

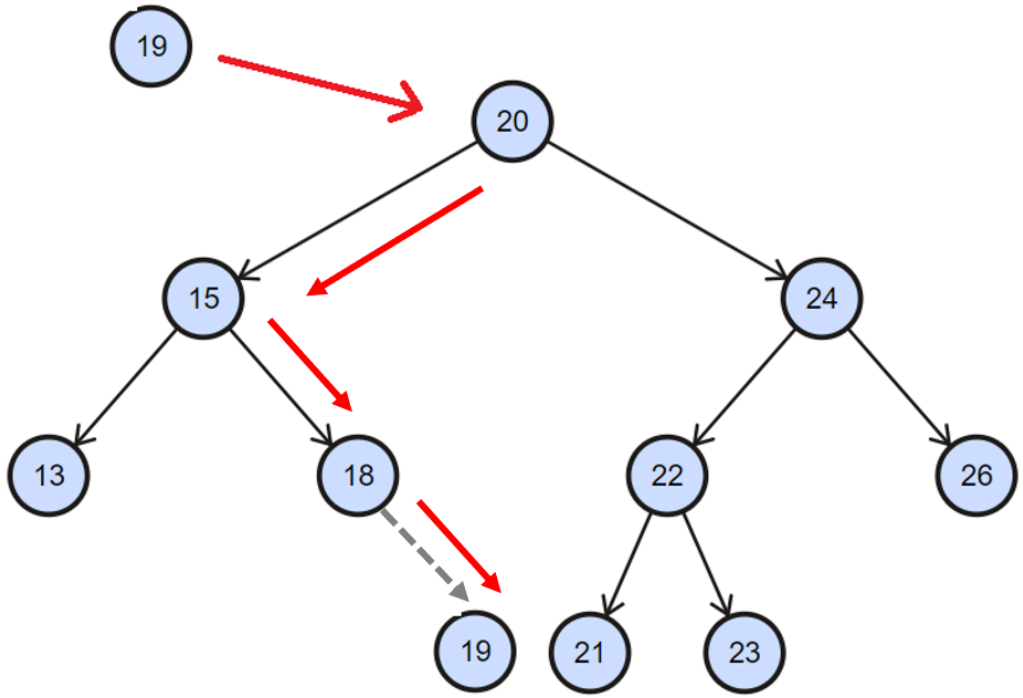
Inserare - recursiv

```
Insert(r, a)
        if( r=0 )
                r := MakeNod(a)
                return
        endif
        if( key(a) < key(data(r) )
                Insert(stg(r), a)
        else
                if( key(data(r) < key(a))
                        Insert(drt(r), a)
                endif
        endif
end
```

Complexitate: O(h)

BST – Binary Search Tree

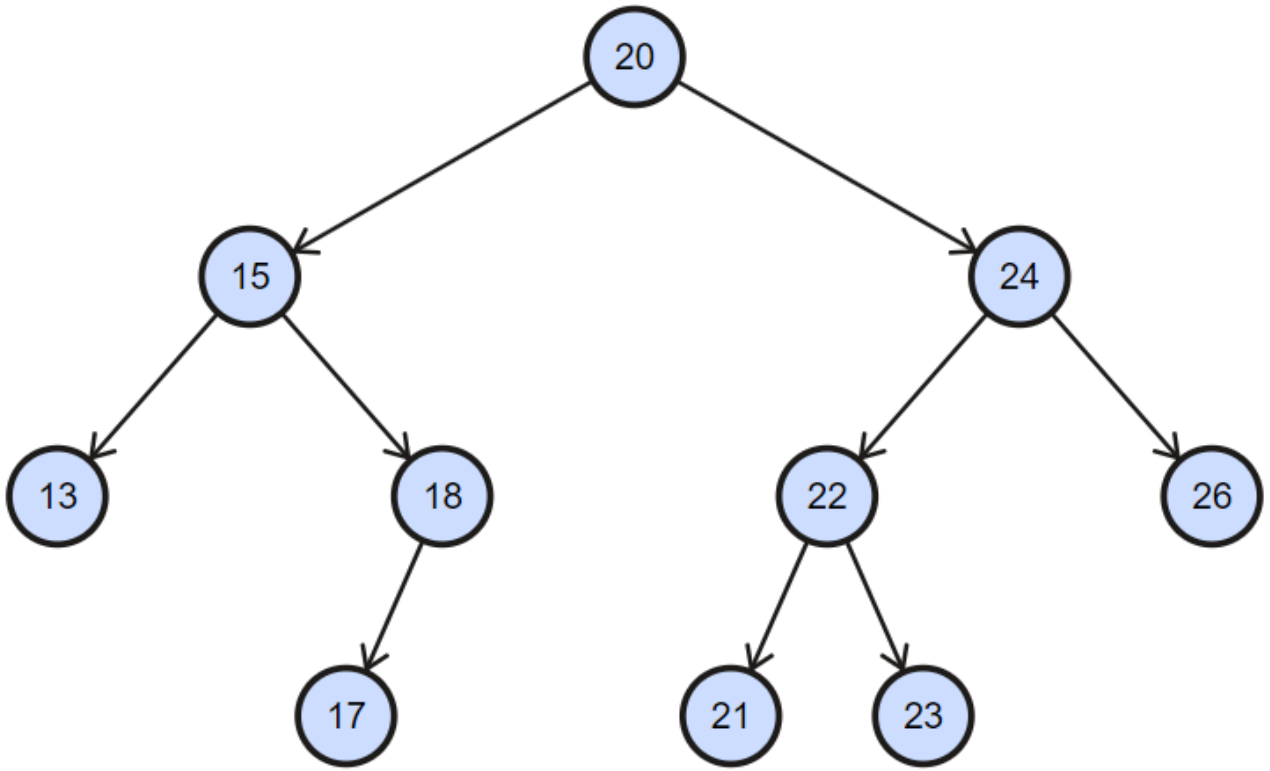
Inserare - iterativ




```
Insert(r, a)
        if( r=0 )
                r := MakeNod(a)
                return
        endif
        p := r
        while ( p!= 0) do
                p1 := p
                if( key(a) < key(data(p)))
                        p := stg(p)
                else if( key(a) > key(data(p)))
                                p := drt(p)
                     endif
                endif
        endwhile
        if( key(a) < key(data(p1)))
                stg(p1) := MakeNod(a)
        else
                drt(p1) := MakeNod(a)
        endif
end
```

BST – Binary Search Tree

Stergerea



Algoritmul Hibbard
- pastrare structura BST
- evitare degenerare arbore

3 cazuri de stergere
- nod frunza
- nod cu un singur descendent
- nod cu doi descendenti

BST – Binary Search Tree

Stergere

```
Delete(r, k)
        if( r=0 )
                //error
                return
        endif
        if( k < key(data(r) )
                Delete(stg(r), k)
        else
                if( key(data(r) < k)
                        Delete(drt(r), k)
                else
                        DeleteRoot(r)
                endif
        endif
end
```

Complexitate: O(h)

BST – Binary Search Tree

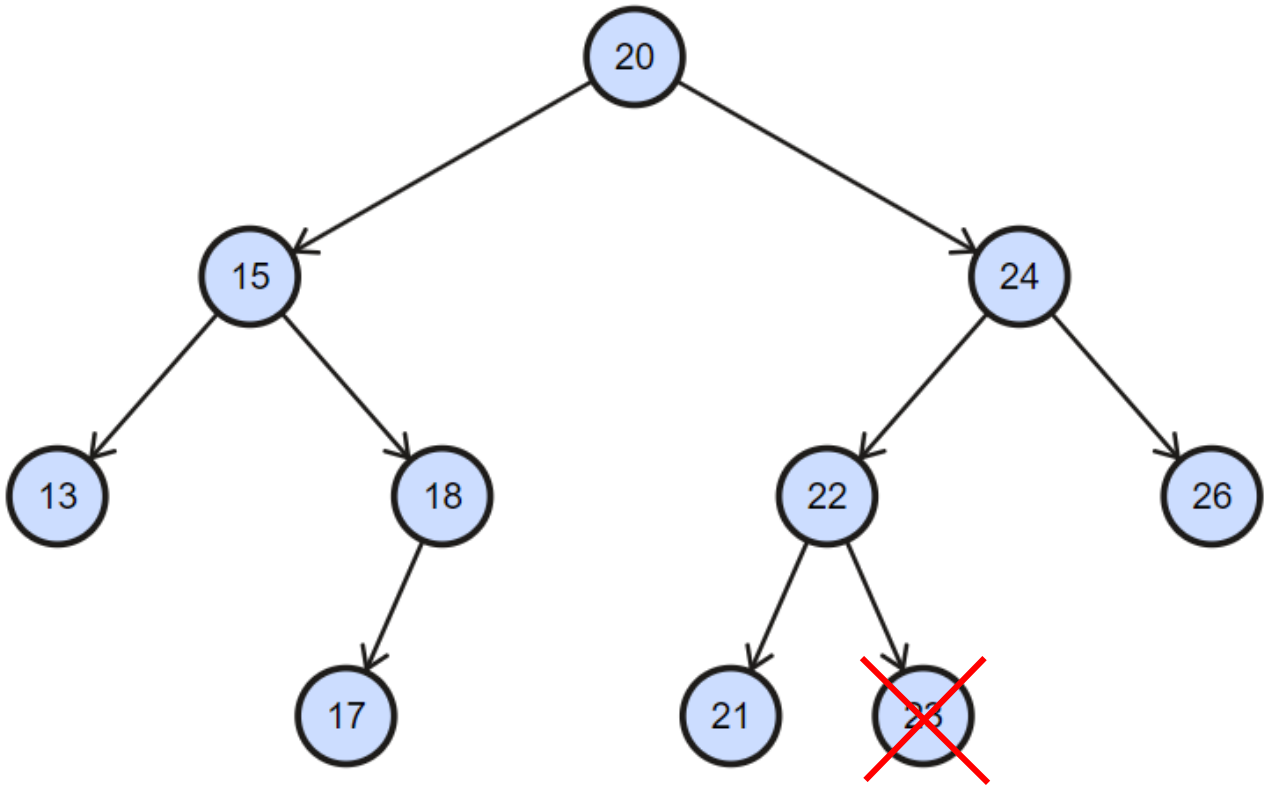
Stergere

```
Delete(r, k)
        if( r=0 )
                //error
                return
        endif
        if( k < key(data(r) )
                Delete(stg(r), k)
        else
                if( key(data(r) < k)
                        Delete(drt(r), k)
                else
                        DeleteRoot(r)
                endif
        endif
end
```

functia **DeleteRoot** realizeaza stergerea efectiva si trebuie sa trateze cele 3 cazuri

BST – Binary Search Tree

Stergere

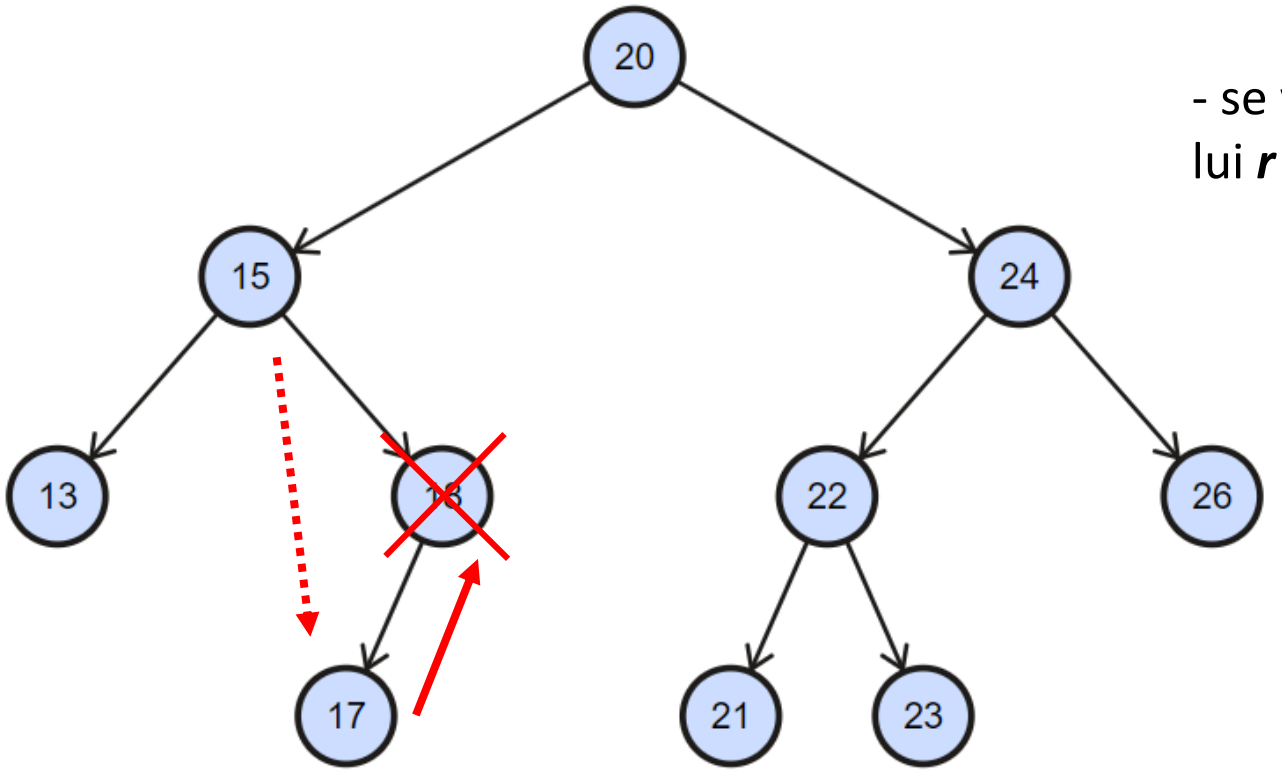


`DeleteRoot`

stergere nod frunza
- direct

BST – Binary Search Tree

Stergere



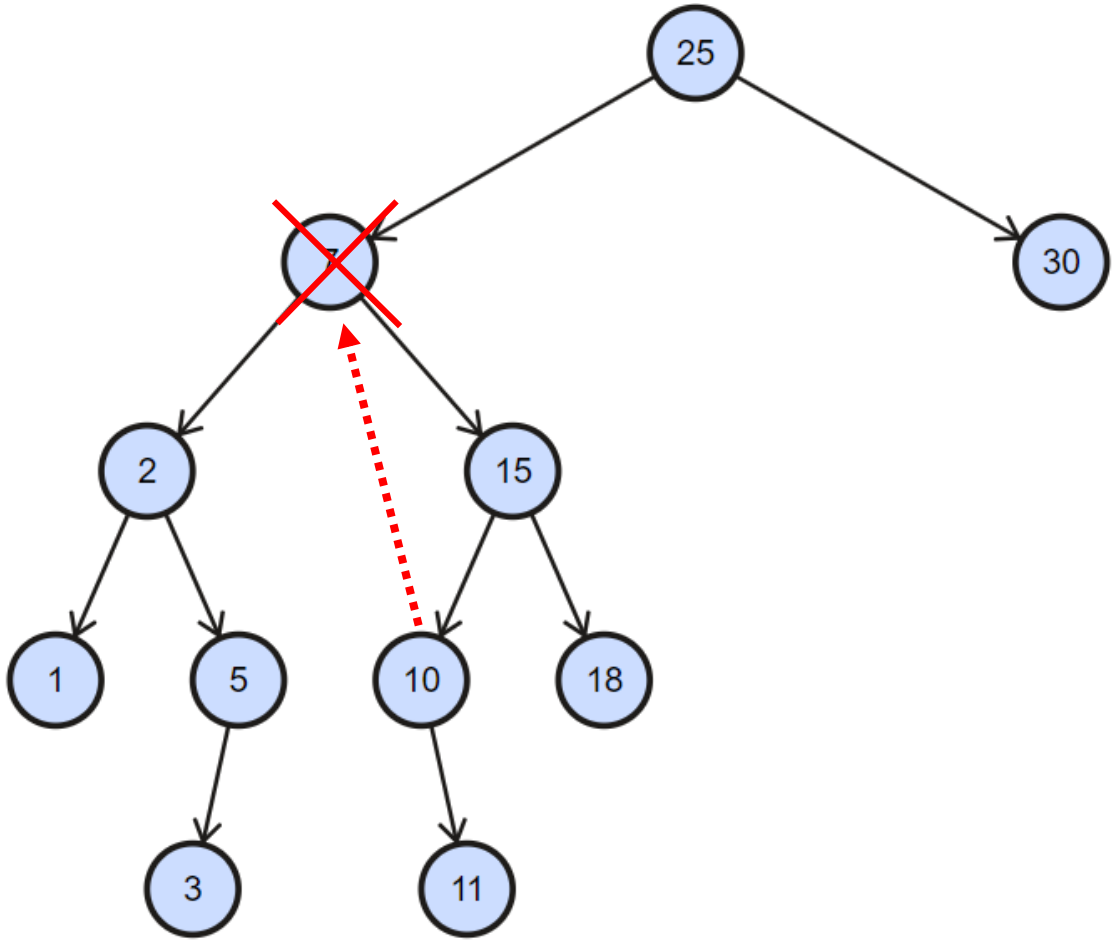
`DeleteRoot`

stergere nod ***r*** cu 1 descendent

- se va crea o legatura intre parintele lui ***r*** si descendentul lui ***r*** (nodul ***r*** este inlocuit cu fiul sau)

BST – Binary Search Tree

Stergere



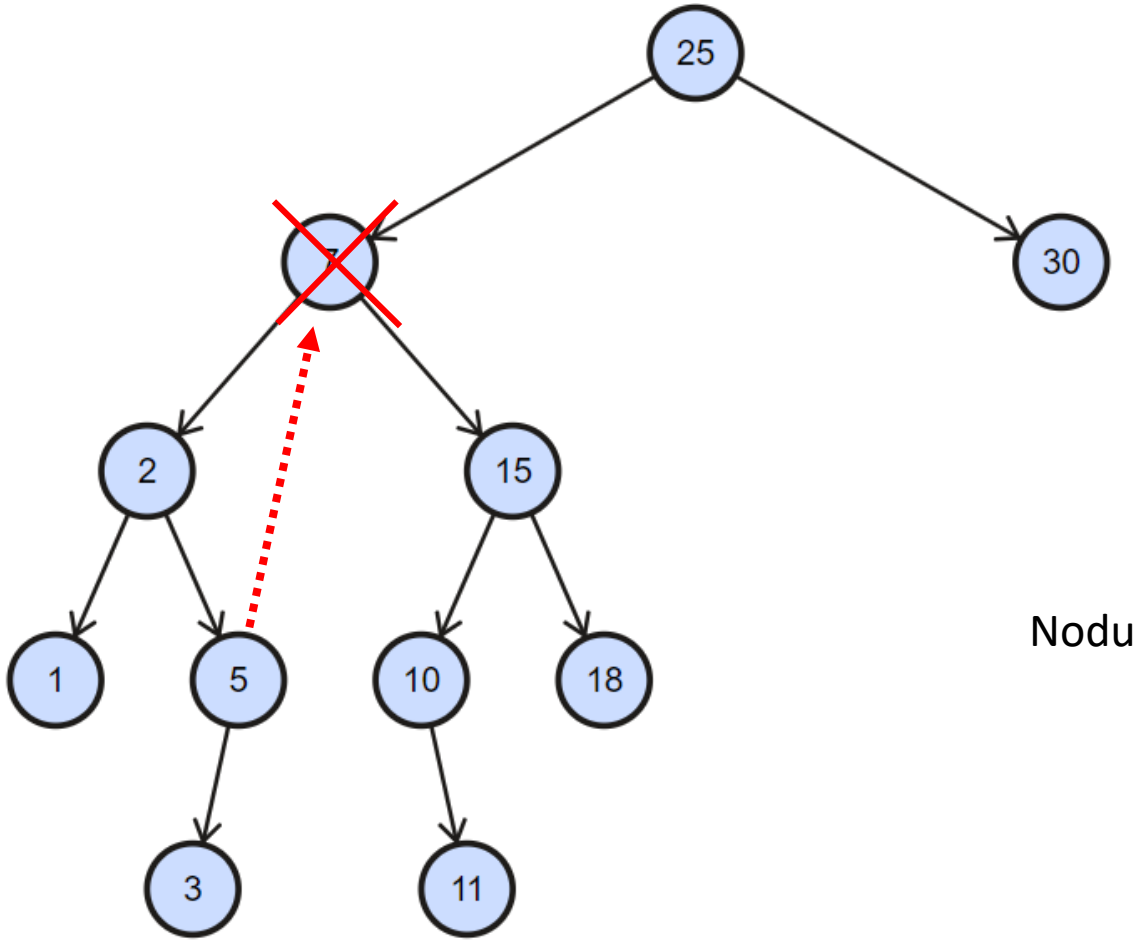
`DeleteRoot`

stergere nod ***r*** cu 2 descendenti

- Se incearca inlocuirea sa cu
  1. Nodul de cheie minima din subarborele drept SAU

BST – Binary Search Tree

Stergere



`DeleteRoot`

stergere nod ***r*** cu 2 descendenti

- Se incearca inlocuirea sa cu
  1. Nodul de cheie minima din subarborele drept SAU
  2. Nodul de cheie maxima din subarborele stang

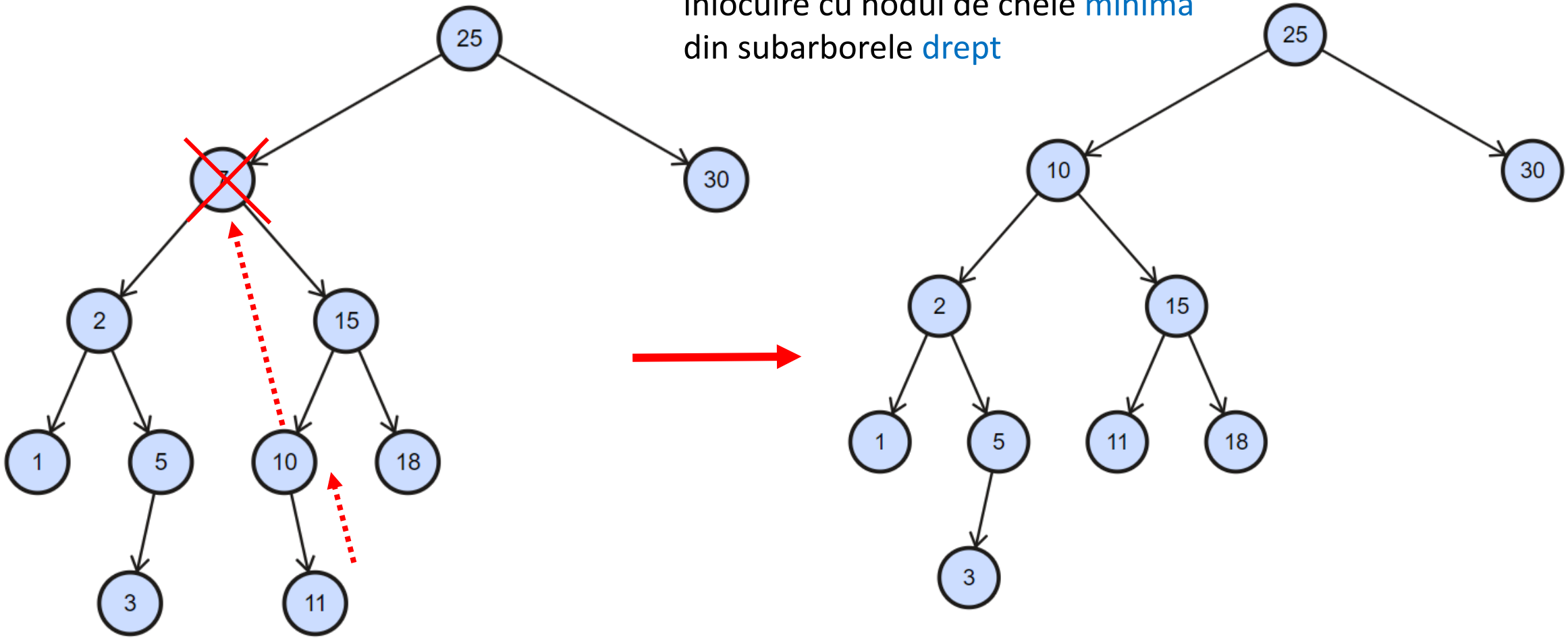
Nodul cu care se inlocuieste poate avea maxim 1 fiu!!

BST – Binary Search Tree

Stergere

`DeleteRoot`

inlocuire cu nodul de cheie minima
din subarborele drept

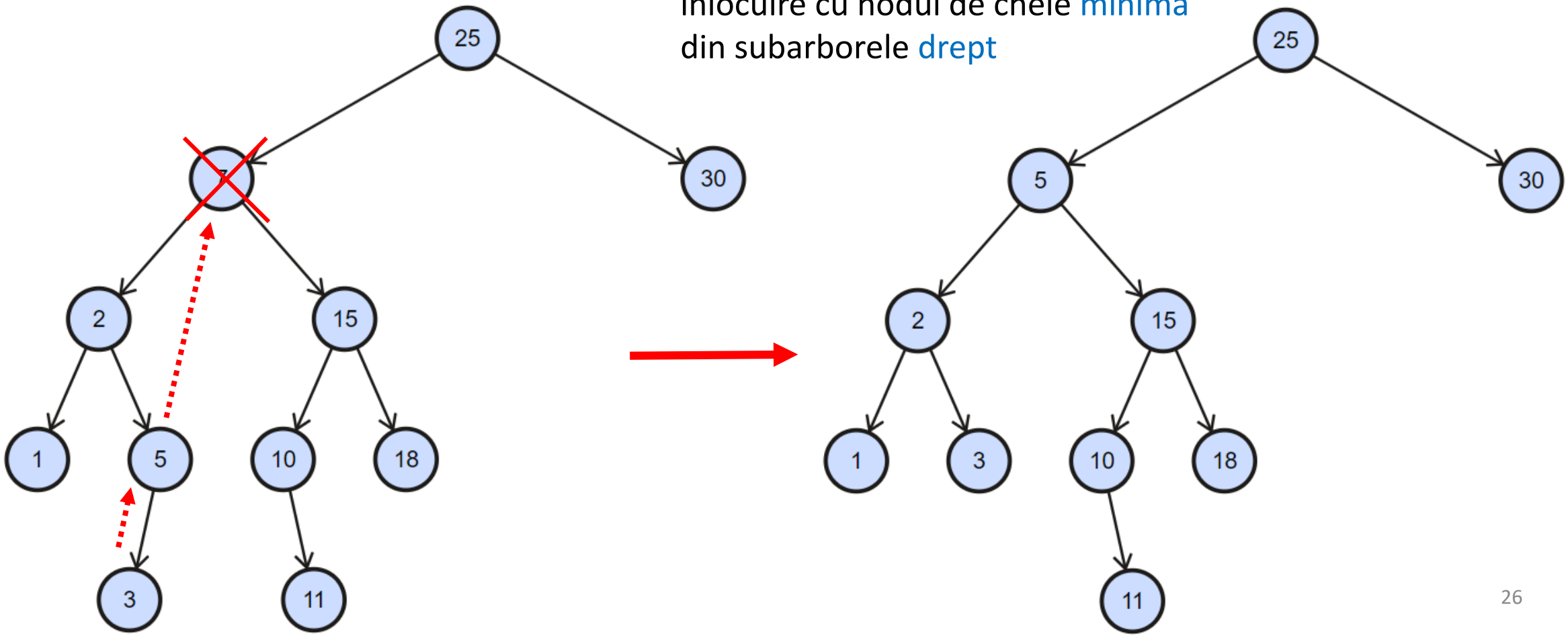


BST – Binary Search Tree

Stergere

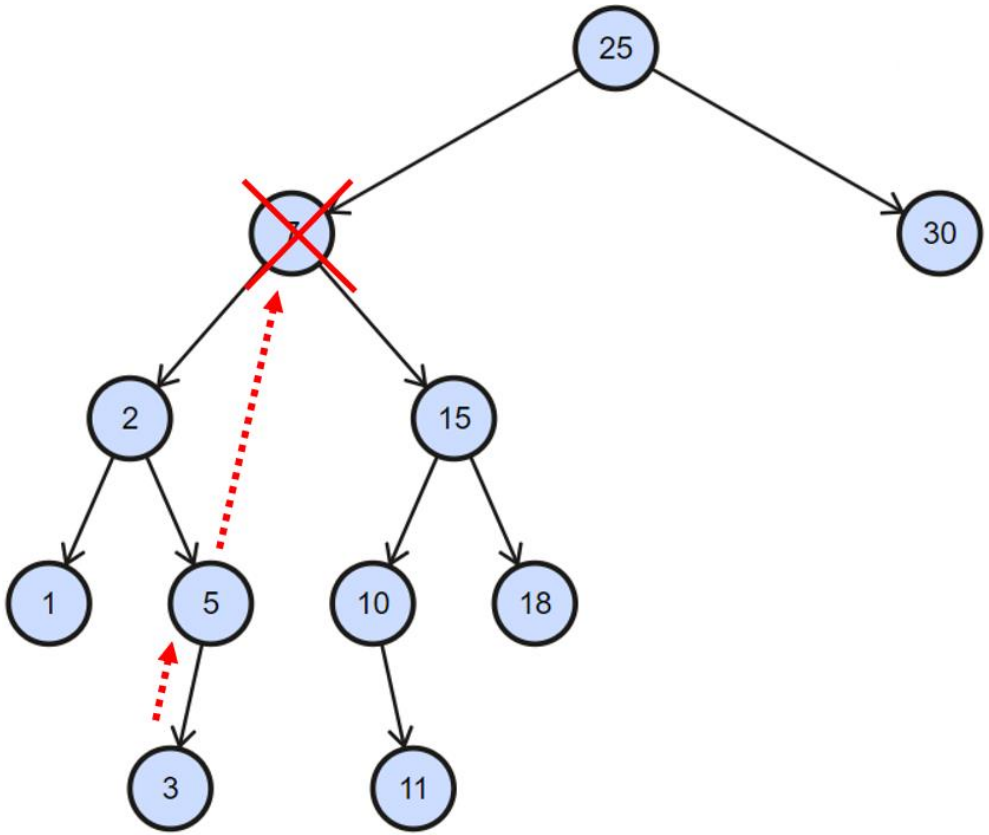
`DeleteRoot`

inlocuire cu nodul de cheie minima
din subarborele drept



BST – Binary Search Tree

Stergere




```
DeleteRoot(r)
        if( stg(r)=0 ) //daca r are fiu drt sau este frunza
                p := drt(r)
                delete r
                r := p
        else
                if( drt(r)=0 ) //daca r are doar fiu stg
                        p := stg(r)
                        delete r
                        r := p
                else //r are 2 fii
                        p := RemoveGreatest(stg(r)) //var. 2
                        stg(p) := stg(r)
                        drt(p) := drt(r)
                        delete r
                        r := p
                endif
        endif
end
```

BST – Binary Search Tree

Stergere

```
RemoveGreatest(r)
        if( drt(r)=0)
                p := r
                r := stg(r)
                return p
        else
                return RemoveGreatest(drt(r))
        endif
end
```

```
DeleteRoot(r)
        if( stg(r)=0 ) //daca r are fiu drt sau este frunza
                p := drt(r)
                delete r
                r := p
        else
                if( drt(r)=0 ) //daca r are doar fiu stg
                        p := stg(r)
                        delete r
                        r := p
                else //r are 2 fii
                        p := RemoveGreatest(stg(r)) //var. 2
                        stg(p) := stg(r)
                        drt(p) := drt(r)
                        delete r
                        r := p
                endif
        endif
end
```

BST – Binary Search Tree

- Exercitii
  - Se introduc numerele **31, 22, 29, 25, 28, 45, 38, 42, 63, 15, 16, 30** intr-un BST
    - Care este valoarea radacinii?
  - Se sterg elementele **22** si **45**
    - Care este parcurgerea in in/pre/postordine a arborelui rezultat?