PROGRAMARE ȘI STRUCTURI DE DATE

CURS 14 - Arbore binar de căutare

Lect. dr. Oneț-Marian Zsuzsanna

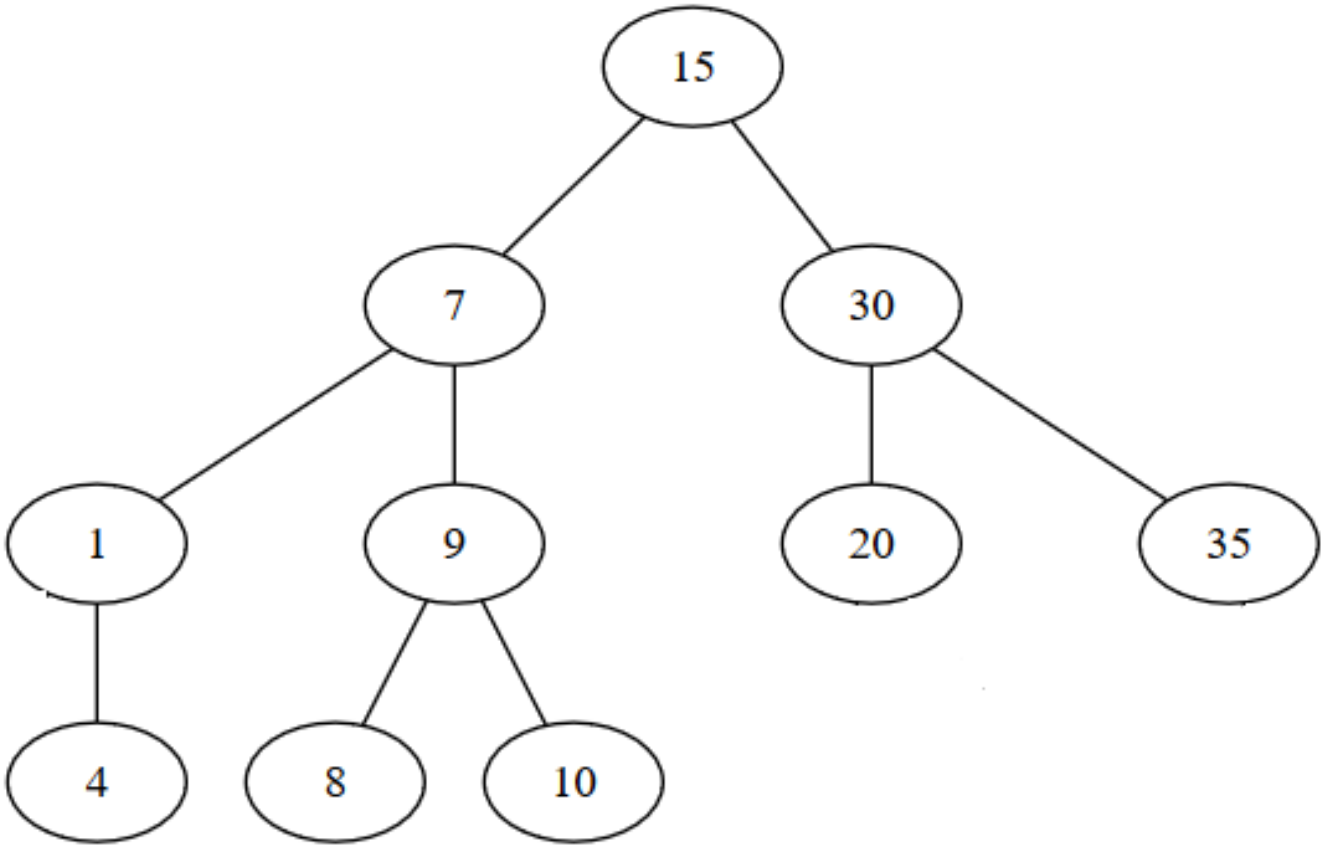
Facultatea de Matematică și Informatică UBB
în colaborare cu NTT Data

Cuprins

Arbore Binar de Căutare - ABC

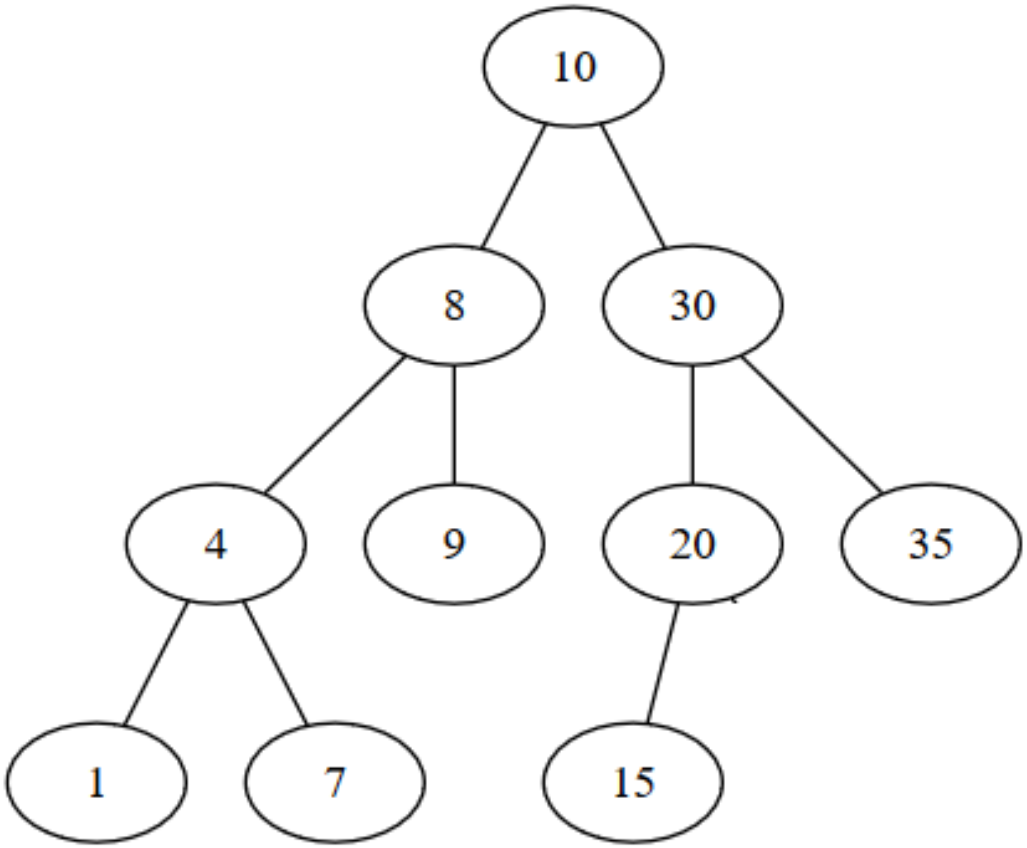
- Un *Arbore Binar de Căutare* (ABC) este un arbore binar în care pentru fiecare nod, nodurile din subarborele stâng conțin elemente mai mici (sau egale) decât el, și nodurile din subarborele drept conțin elemente mai mare decât el.

Exemplu de ABC



ABC

- Pot exista mai mulți ABC-uri cu exact aceleași elemente dar care au structură diferită.



ABC

- Dacă parcurgem un ABC în inordine (stânga, rădăcina, dreapta), vom avea toate elementele în ordine crescătoare.

ABC

- Pentru un ABC putem avea operații care nu au existat la arbori n-ari sau arbori binari:
  - adăugare
  - ștergere
  - căutare
- ABC poate fi folosit ca reprezentare pentru containere ordonate în care nu există poziții: mulțime ordonată, colecția ordonată, dicționar ordonat, etc.

ABC - reprezentare

- Reprezentarea pentru un ABC este similară cu reprezentarea unei arbori binari simple: avem nevoie de structura Nod, și avem nevoie de structura ABC.

Nod:
info: TElem
stang: ↑ Nod
drept: ↑ Nod

ABC:
rad: ↑ Nod

ABC - Căutare

- Cum putem verifica dacă un element apare sau nu în ABC (adică dacă am un nod care conține elementul respectiv)?

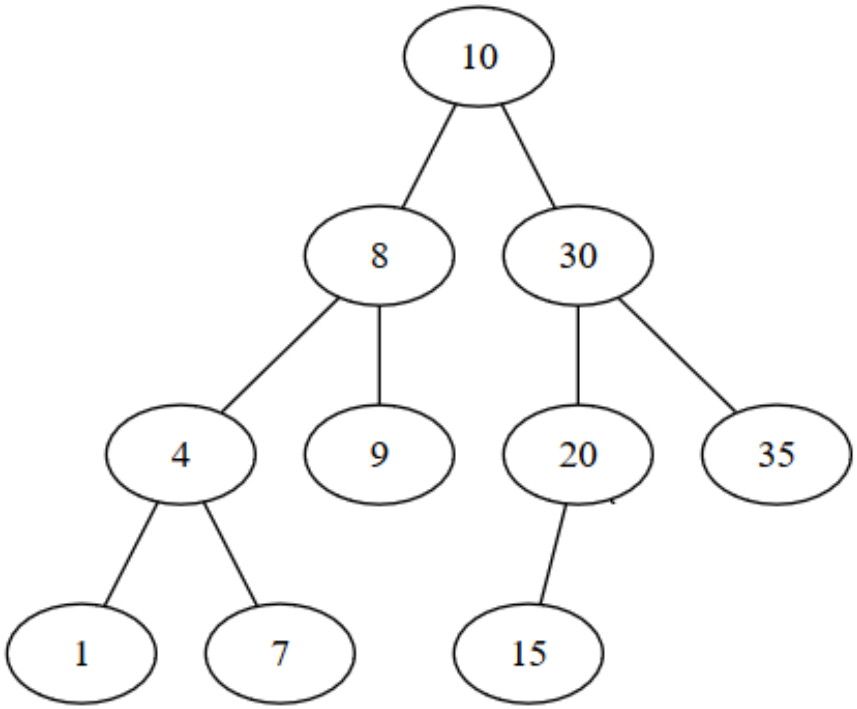
ABC - Căutare

- Cum putem verifica dacă un element apare sau nu în ABC (adică dacă am un nod care conține elementul respectiv)?

- Mecanismul este similar cu căutarea binară: pornesc de la rădăcină și continui ori pe subarborele stâng, ori pe cel drept, în funcție de cum este elementul căutat față de cel din nodul curent.

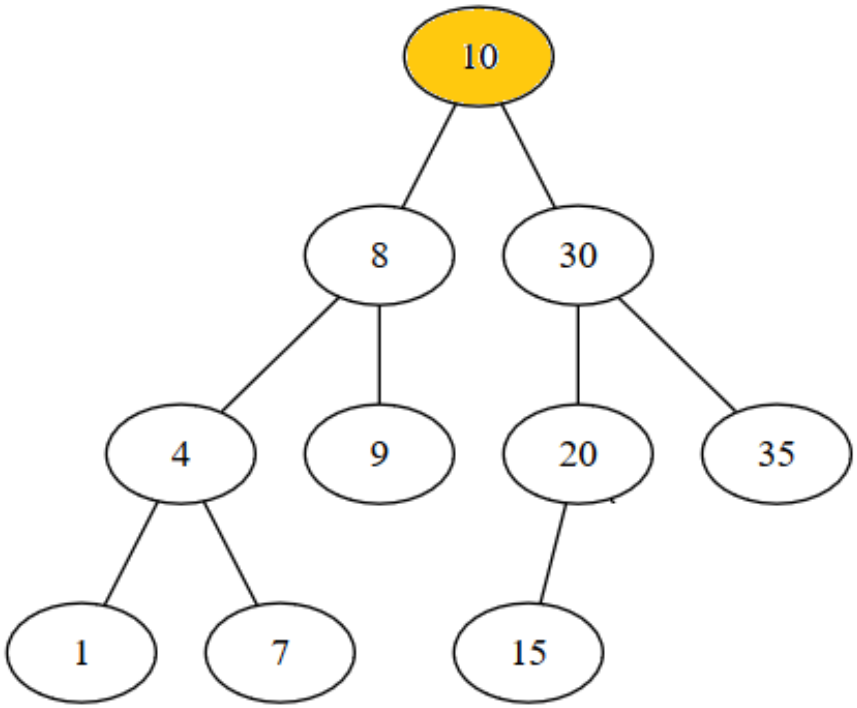
ABC - Căutare I

- Să căutăm elementul 15 în arborele de mai jos



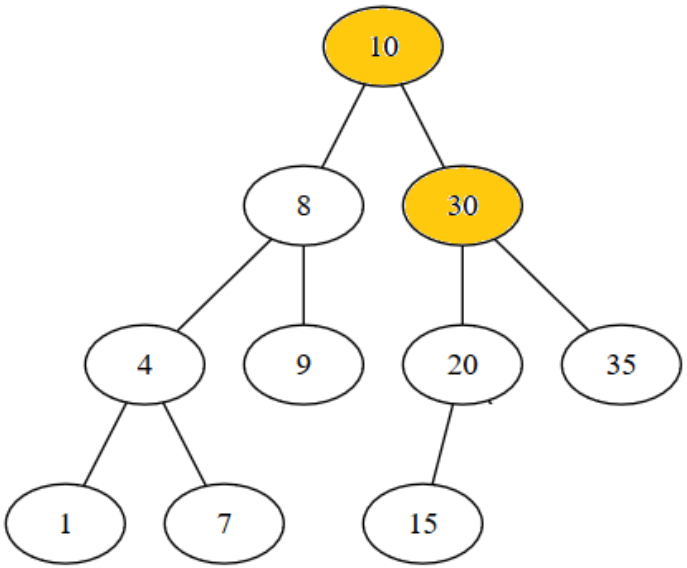
ABC - Căutare II

- Pornim cu un nod curent, care este rădăcina.



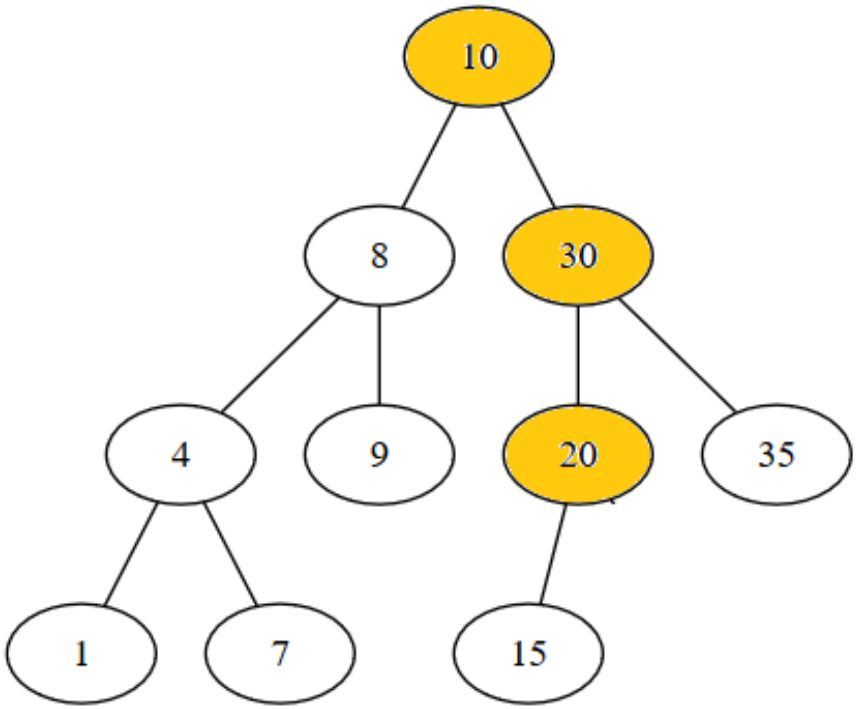
ABC - Căutare III

- Dacă nodul curent nu conține elementul căutat, comparăm elementul căutat cu elementul din nodul curent. Dacă elementul căutat este mai mic, continuăm în subarbore stâng (nod curent devine stângul lui nod curent), altfel continuăm în subarbore drept. Din moment ce $10 < 15$, mergem pe subarbore drept.



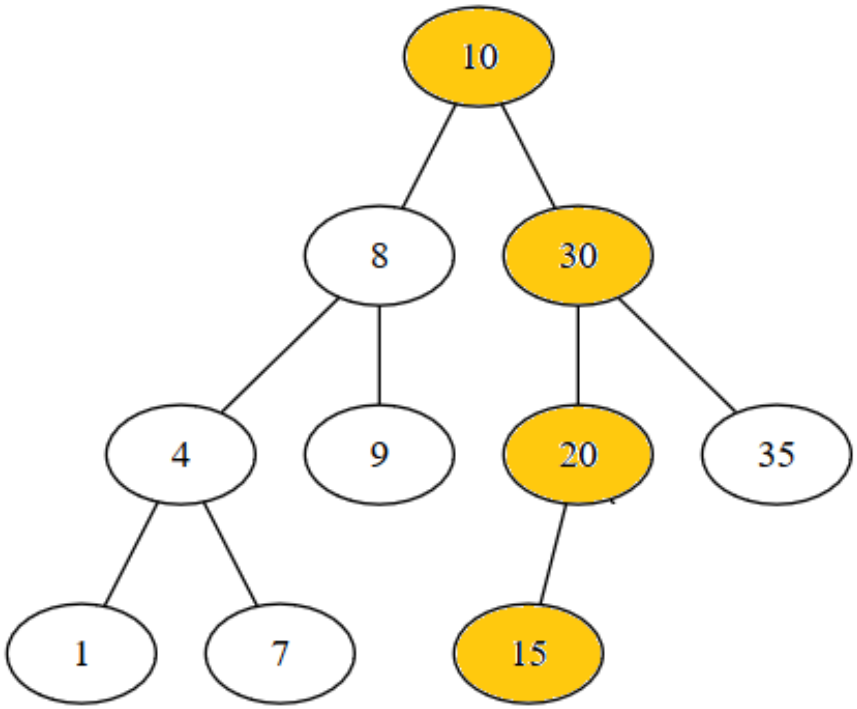
ABC - Căutare IV

- Continuăm pe subarborele stâng ($15 < 30$)



ABC - Căutare V

- Continuăm pe subarborele stâng ($15 < 20$)



ABC - Căutare VI

- Am găsit nodul, deci căutarea a fost cu succes.
- Dacă am fi căutat elementul 16, tot la fel am fi făcut ($16 > 10$, $16 < 30$, $16 < 20$), dar când nodul curent era 15, continuam pe subarborele drept, care nu există. Când nodul curent devine NIL, putem spune că elementul nu se găsește în arbore.

ABC - Căutare

```
funcție cautare(elem: TElem) este:
//elem este elementul căutat
    curent: ↑ Nod
    curent = this.rad //curent este ↑ Nod
    câttimp curent ≠ NIL SI [curent].info ≠ elem execută
        dacă elem < [curent].info atunci
            curent = [curent].stang
        altfel
            curent = [curent].drept
        sf_dacă
    sf_cattimp
    dacă curent == NIL atunci
        returnează Fals
    altfel
        returnează Adevarat
    sf_dacă
sf_funcție
```

ABC - Căutare

- Cât este complexitatea algoritmului de căutare?

ABC - Căutare

- Cât este complexitatea algoritmului de căutare?

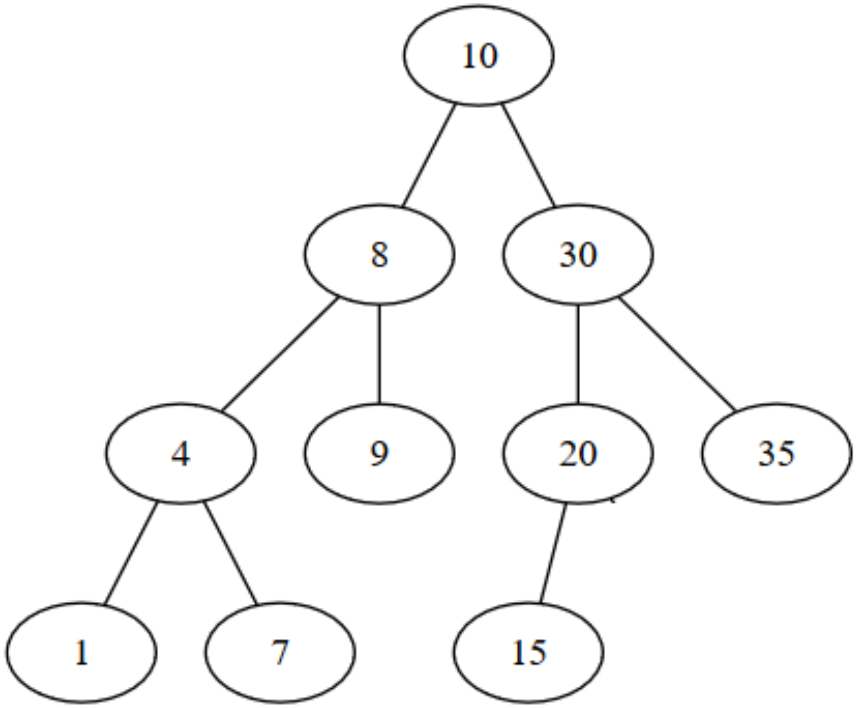
- Deși tindem să credem că la un ABC complexitatea va fi logaritmică, din moment ce putem avea și arbori degenerați (n noduri pe n nivele), complexitatea în caz defavorabil este de fapt $\Theta(n)$.
- În caz mediu, complexitatea este $\Theta(log_2 n)$

ABC - Adăugare I

- Cum adăugăm un element nou într-un ABC?

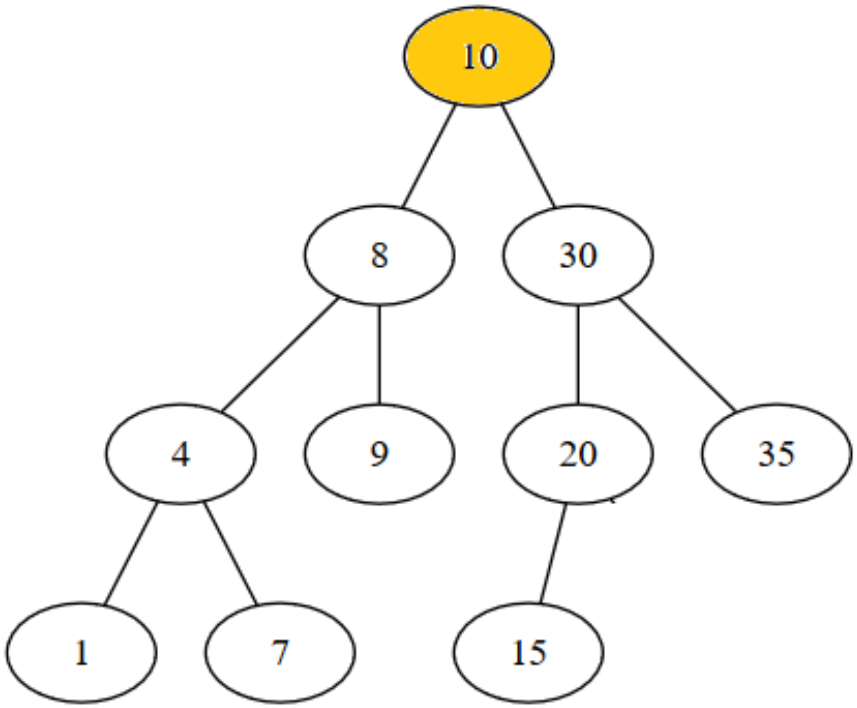
ABC - Adăugare II

- Să adăugăm elementul 6.



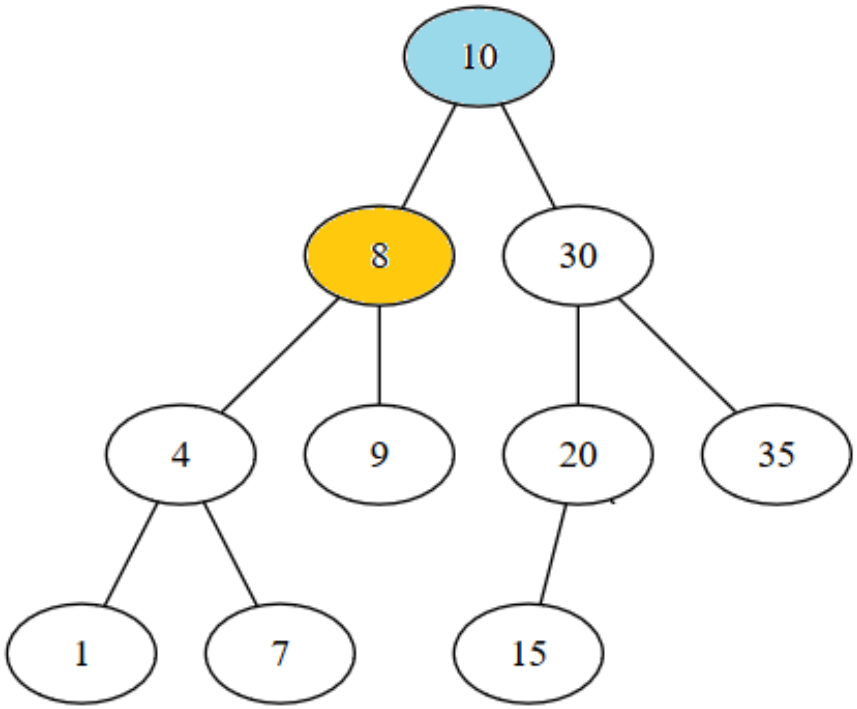
ABC - Adăugare III

- Pornim de la rădăcină. Vom folosi 2 noduri: nod curent (portocaliu) și părintele nodului curent (albastru). Inițial nodul curent este rădăcina și părintele de NIL.



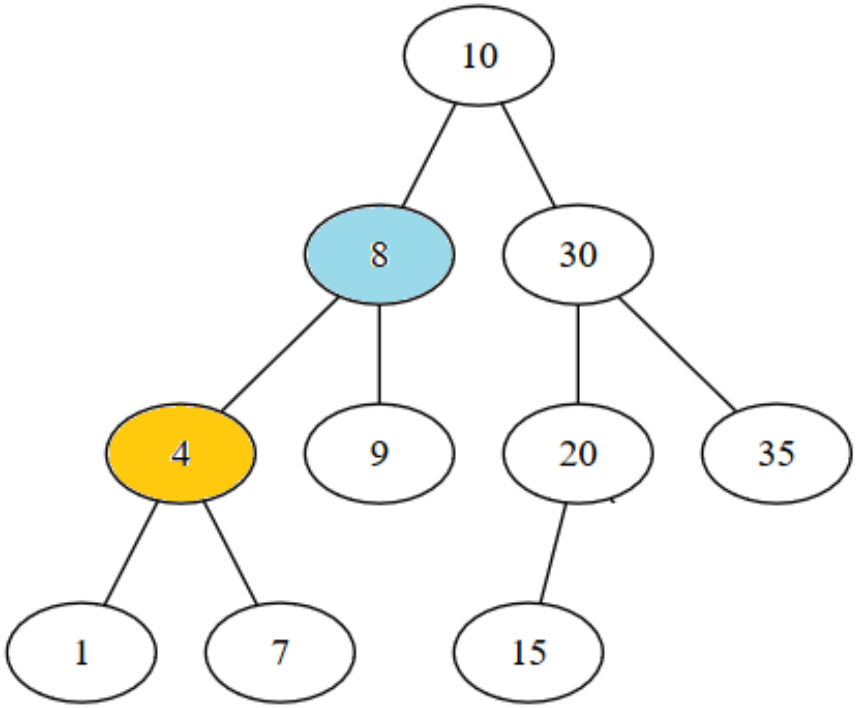
ABC - Adăugare IV

- La fiecare pas părintele ia valoarea nodului curent, iar nodul curent merge pe stânga sau dreapta în funcție de comparația elementului de adăugat cu elementul din nodul curent. Aici, din moment ce $6 < 10$ mergem pe stânga



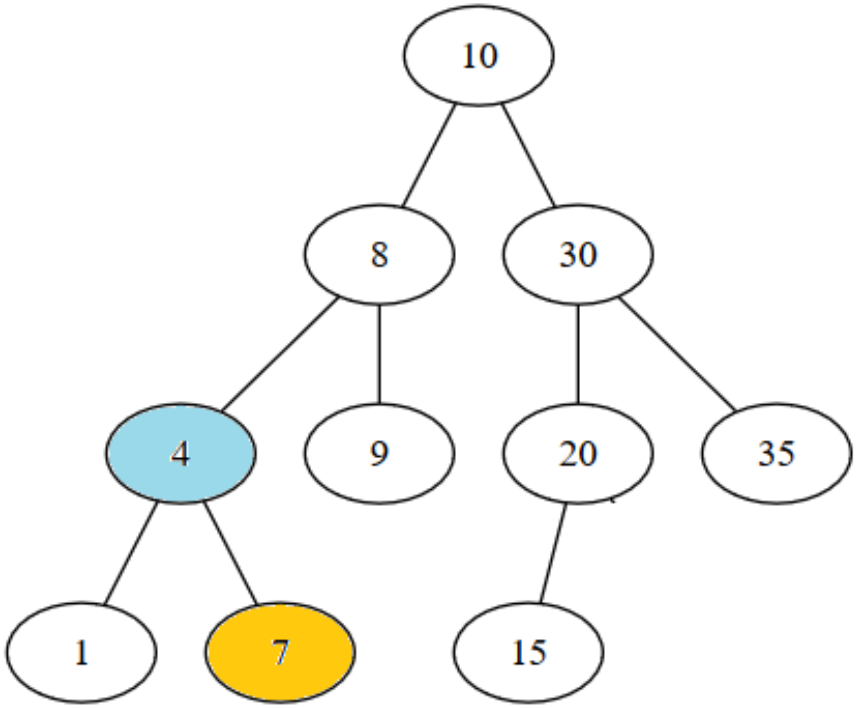
ABC - Adăugare V

- $6 < 8$, mergem pe stânga.



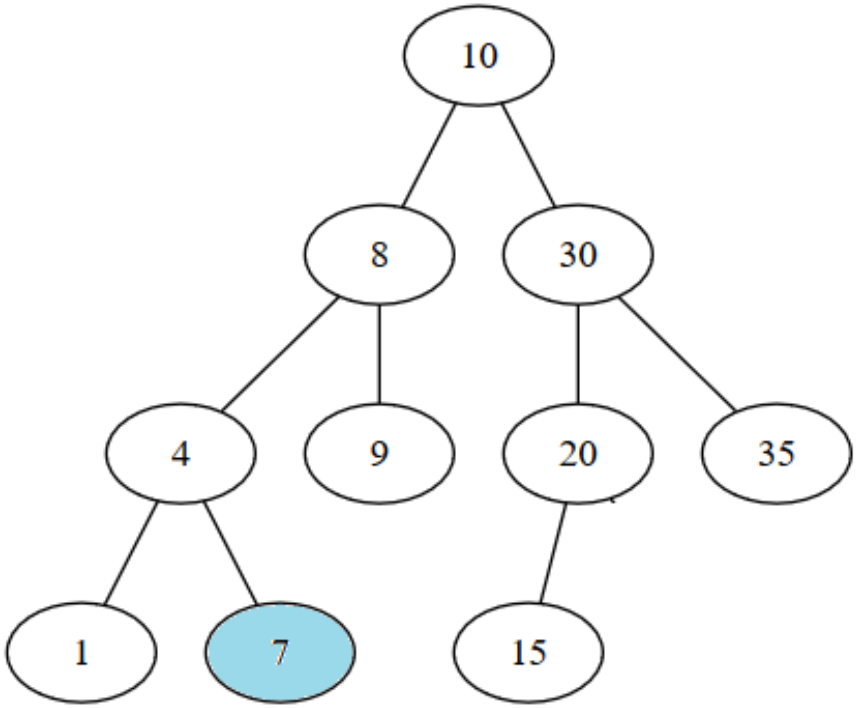
ABC - Adăugare VI

- $6 > 4$, mergem pe dreapta.



ABC - Adăugare VII

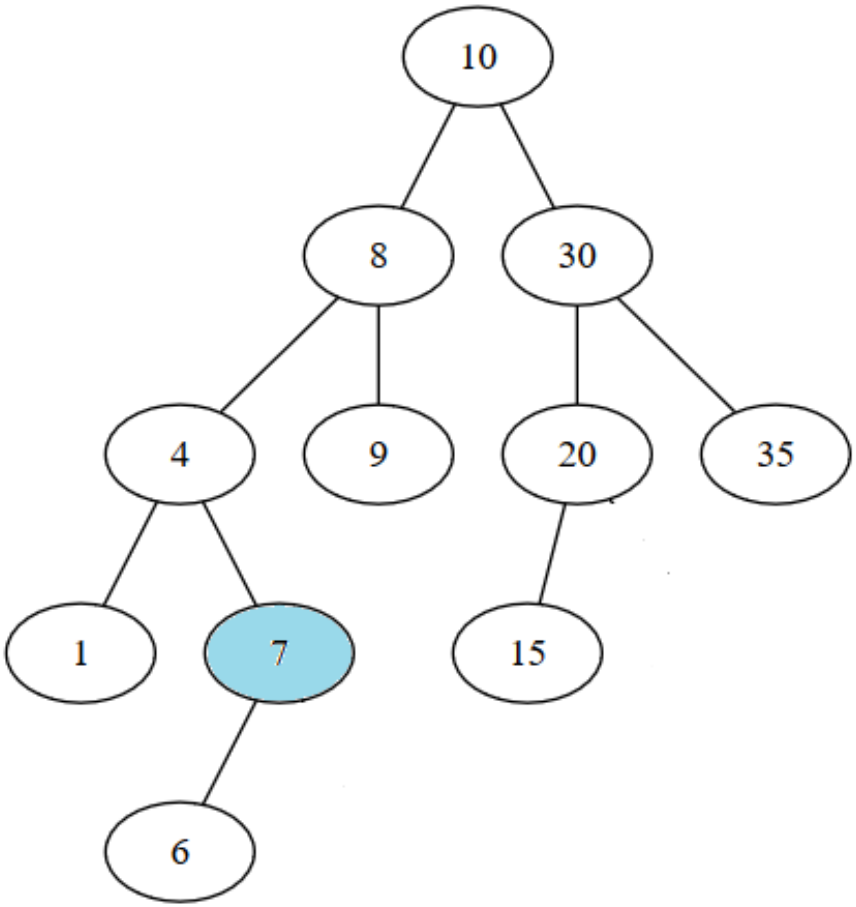
- $6 < 7$, mergem pe stânga.



ABC - Adăugare VIII

- Acum nodul curent este NIL, ceea ce înseamnă că știm că la nodul părinte (cercul albastru) trebuie să adăugăm descendentul. Comparăm valoarea din părinte cu elementul de adăugat ca să știm dacă îi adăugăm descendent stâng sau drept (stâng la noi).

ABC - Adăugare IX



ABC - Adăugare

```
subalgoritm adauga(elem: TElem) este:
//elem este elementul de adăugat
    nodNou, curent, parinte: ↑ Nod
    [nodNou].info = elem
    [nodNou].stang = NIL
    [nodNou].drept = NIL
    dacă this.rad == NIL atunci
        this.rad = nodNou
    altfel
        curent = this.rad
        parinte = NIL //curent si parinte sunt ↑ Nod
        cattimp curent ≠ NIL executa
            parinte = curent
            dacă elem ≤ [curent].info atunci
                curent = [curent].stang
            altfel
                curent = [curent].drept
            sf_dacă
        sf_cattimp
        //continuăm pe pagina următoare
```

ABC - Adăugare

```
        dacă elem ≤ [parinte].info atunci
            [parinte].stang = nodNou
        altfel
            [parinte].drept = nodNou
        sf_dacă
    sf_dacă
sf_subalgoritm
```

- Complexitate:

ABC - Adăugare

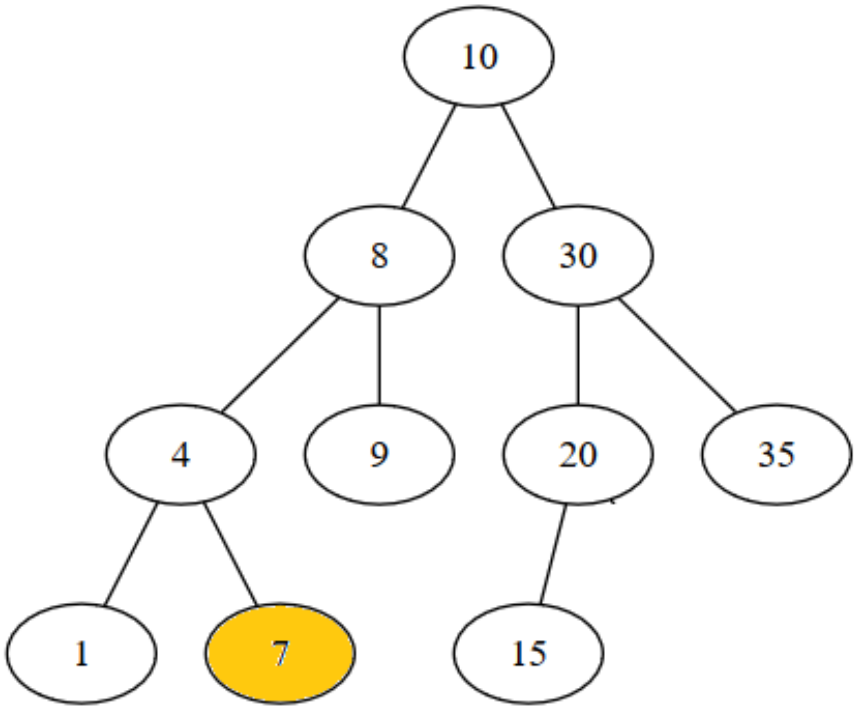
```
        dacă elem ≤ [parinte].info atunci
            [parinte].stang = nodNou
        altfel
            [parinte].drept = nodNou
        sf_dacă
    sf_dacă
sf_subalgoritm
```

- Complexitate: $O(n)$

ABC - Ștergere

- Ștergerea este o operație un pic mai complicată, pentru că trebuie să ne asigurăm că nu stricăm arborele.
- La ștergere prima dată căutăm nodul pe care vrem să-l ștergem și avem 3 cazuri:
  - Nodul nu are descendenți deloc
  - Nodul are un singur descendent
  - Nodul are 2 descendenți

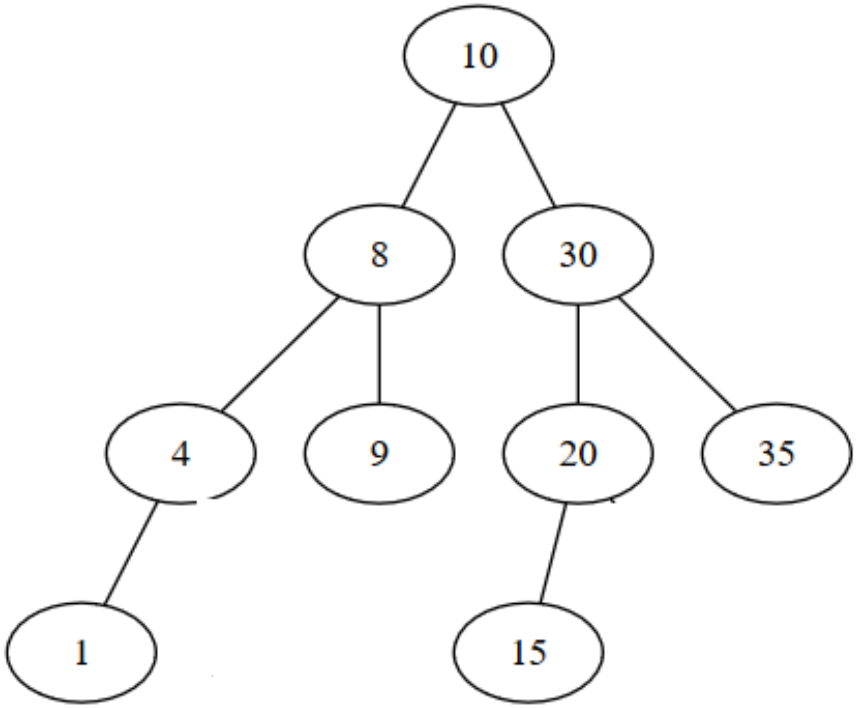
ABC - Ștergere I



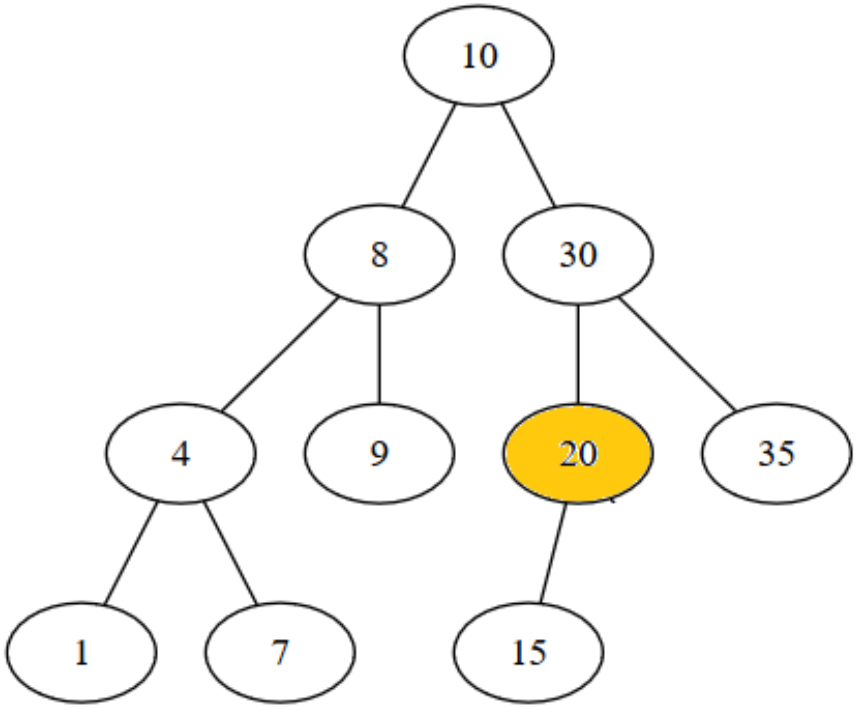
ABC - Ștergere II

- Dacă nodul de șters nu are descendenți, îl putem șterge fără probleme.
- De exemplu dacă vrem să ștergem nodul 7.

ABC - Ștergere III



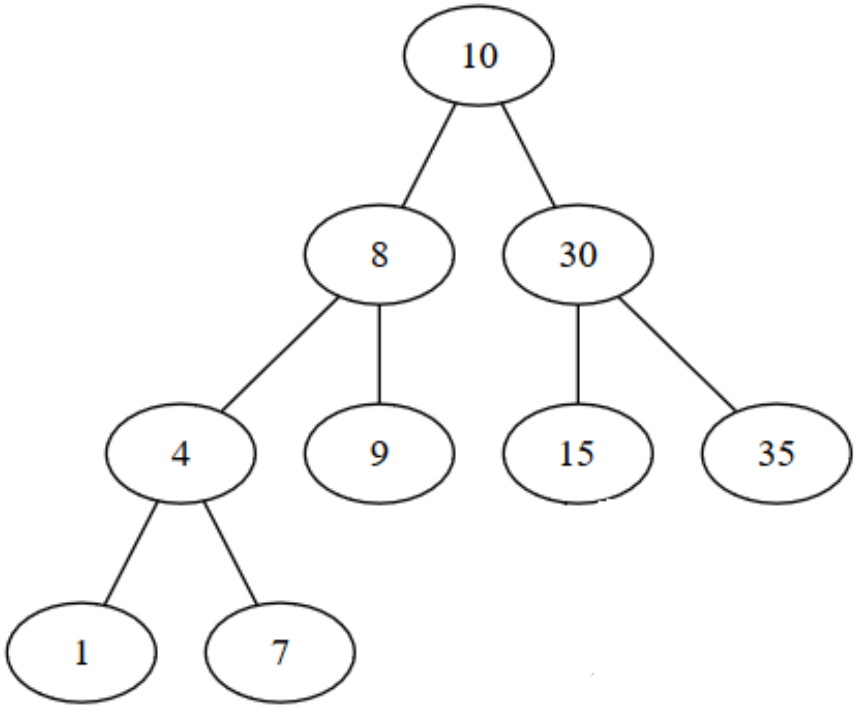
ABC - Ștergere IV



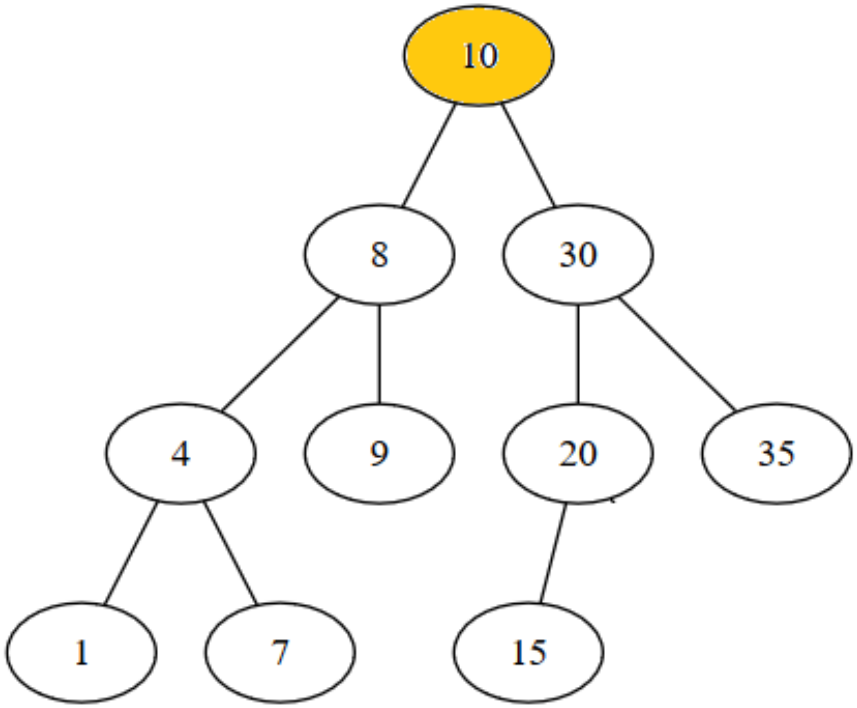
ABC - Ștergere V

- Dacă nodul de șters are un singur descendent, atunci legăm acest descendent de părintele nodului.
- De exemplu dacă vrem să ștergem nodul 20.

ABC - Ștergere VI



ABC - Ștergere VII

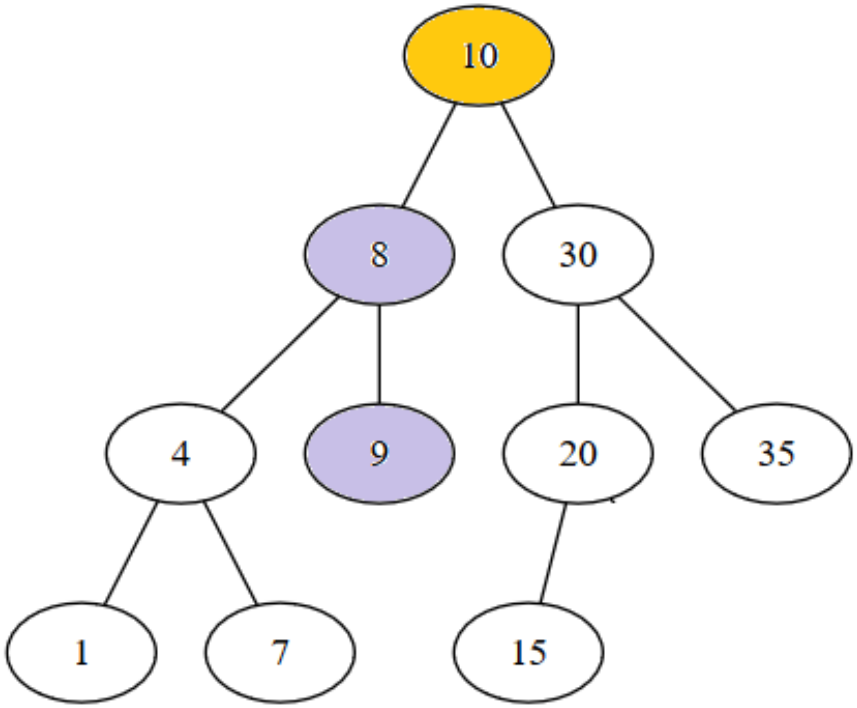


ABC - Ștergere VIII

- Dacă vrem să ștergem un nod cu 2 descendenți avem 2 opțiuni (ambele sunt corecte, oricare poate fi folosită):
  - Căutăm elementul maxim din subarbore stâng, înlocuim conținutul nodului de șters cu acest maxim, și ștergem nodul cu maxim.
  - Căutăm elementul minim din subarbore drept, înlocuim conținutul nodului de șters cu acest minim, și ștergem nodul cu minim.
- Să ștergem nodul 10.

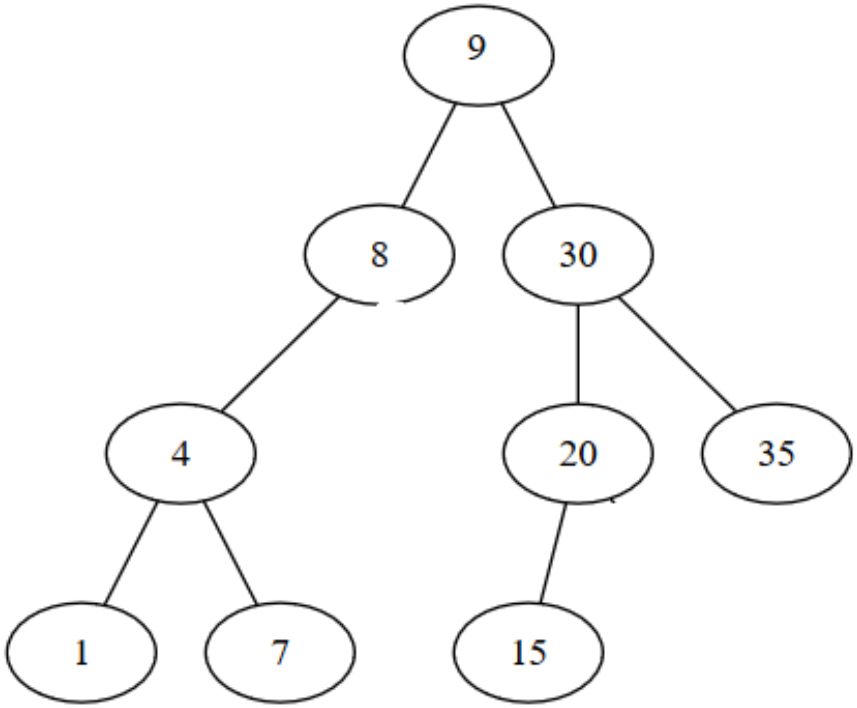
ABC - Ștergere IX

- Maximul din stânga, este elementul cel mai la dreapta din subarborele stâng (pornesc pe subarborele stâng și merg pe ramura dreapta până dau de un nod fără descendent drept)



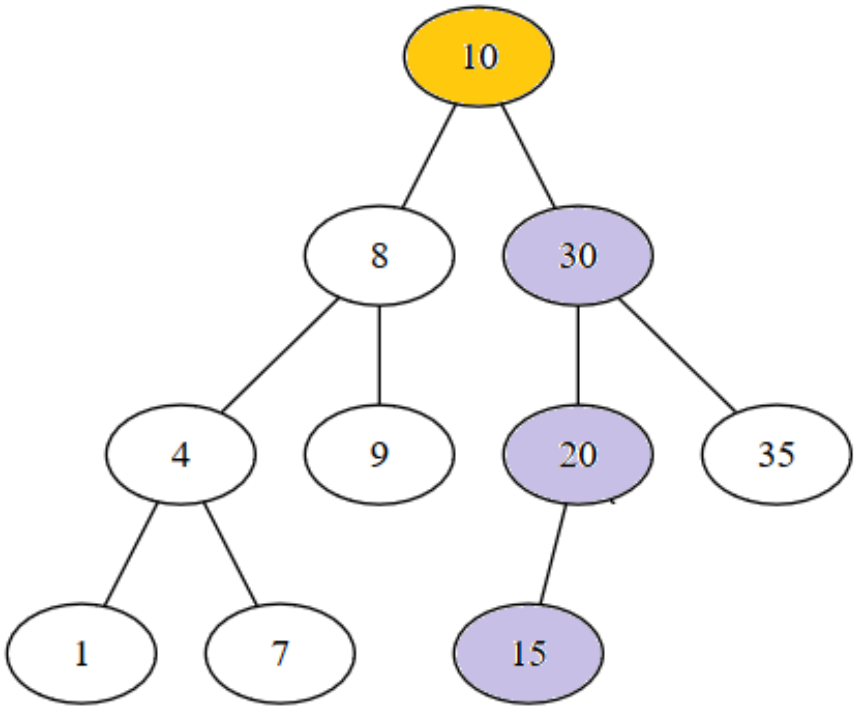
ABC - Ștergere X

- Mutăm 9 în rădăcină, și ștergem nodul cu 9.



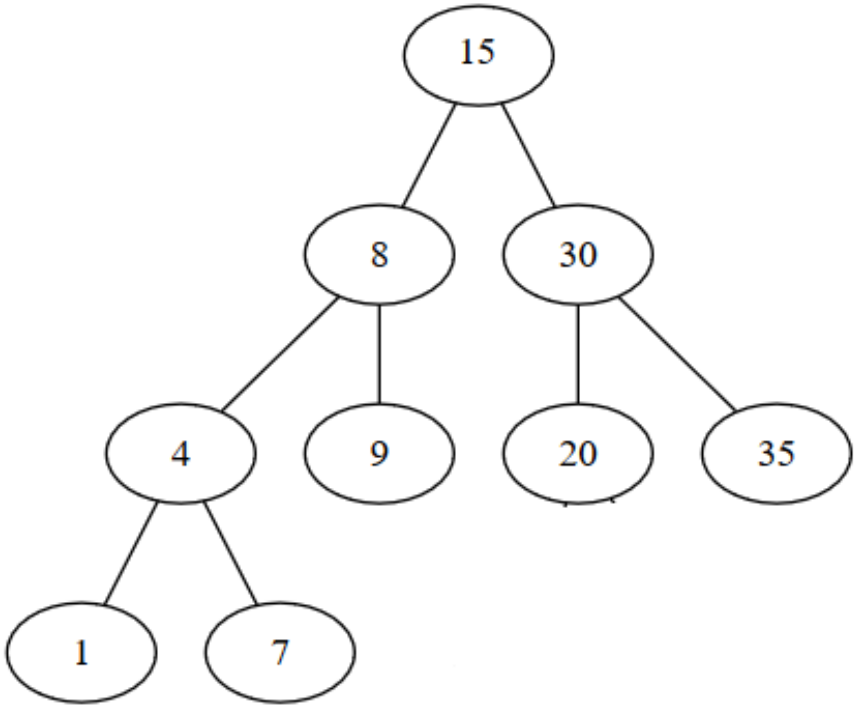
ABC - Ștergere XI

- Cealaltă variantă:
- Minimul din dreapta, este elementul cel mai la stâga din subarborele drept (pornesc pe subarborele drept și merg pe ramura stânga până dau de un nod fără descendent stâng)



ABC - Ștergere XII

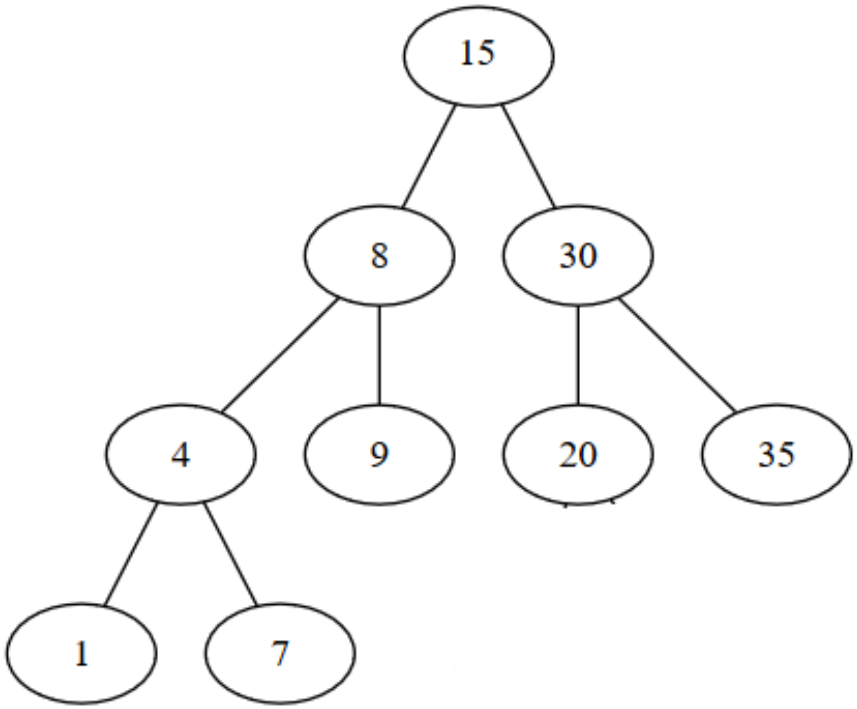
- Mutăm 15 în rădăcină, și ștergem nodul cu 15.



Arbori echilibrați

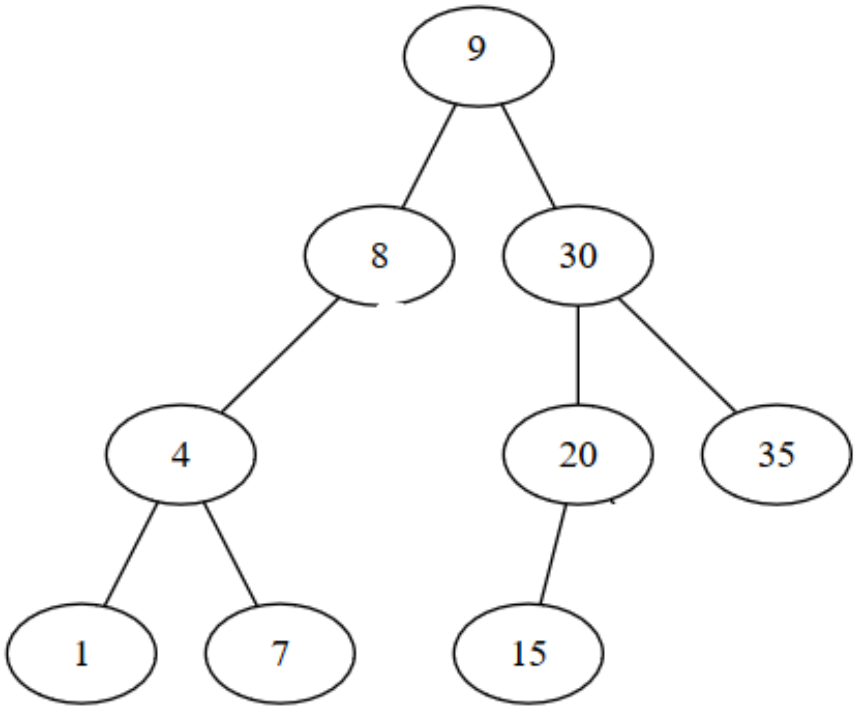
- Am văzut că la ABC avem complexitate $O(n)$ pentru operații.
- Dacă vrem să asigurăm complexitate $O(log_2 n)$ pentru operații trebuie să ne asigurăm că ABC-ul este echilibrat:
  - Diferența de înălțime dintre subarborele stâng și cel drept este 1, 0 sau -1.

Arbori echilibrați



- Este echilibrat.

Arbori echilibrați

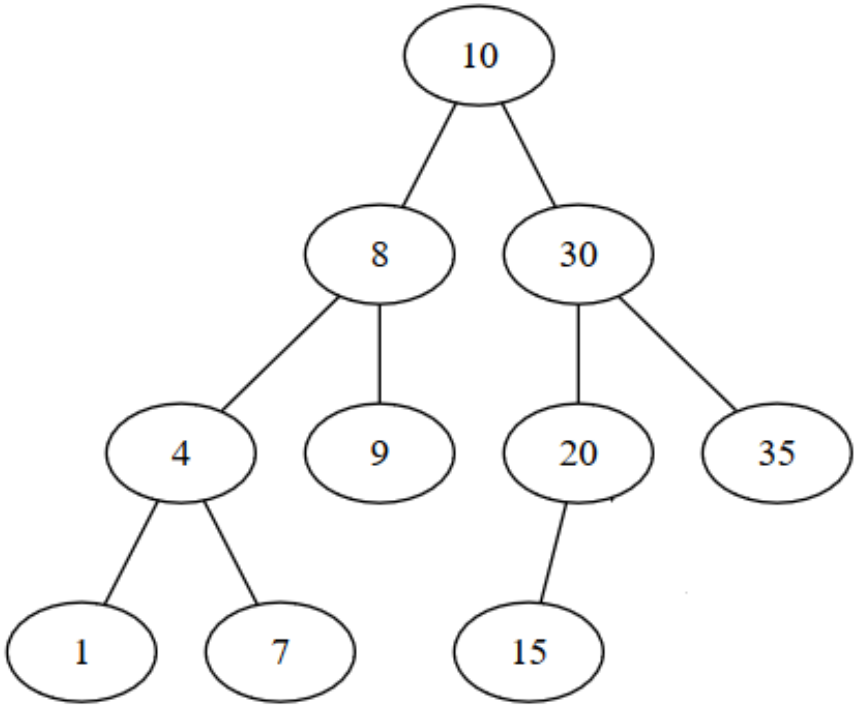


- Nu este echilibrat (8 are adâncime 2 la stânga și 0 la dreapta).

Arbori echilibrați I

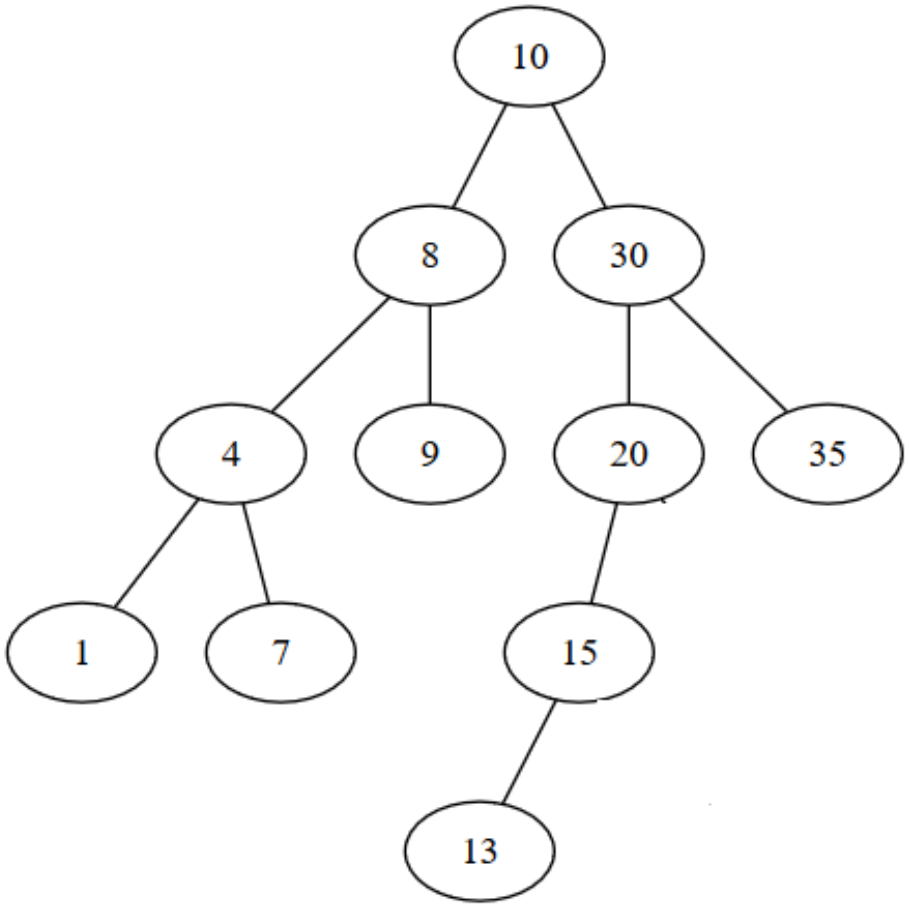
- Pentru a avea arbori echilibrați, după operații de adăugare și ștergere, dacă se strică echilibrul, aplicăm rotații.

- De exemplu:



Arbori echilibrați II

- Adăugăm 13 și nu mai este echilibrat



Arbori echilibrați III

- Aplicăm o rotație la nodul 20.