Structuri de Date

Seminar 5

Recapitulare BST (Binary Search Trees/Arbori binari de căutare)

• fiecare nod are cel mult doi fii

• fiul stâng al unui nod X conține o valoare mai mică decât valoarea din nodul X, iar fiul drept o valoare mai mare decât cea din nodul X

• ca urmare a proprietății de mai sus, subarborele stâng al unui nod X va conține doar noduri cu valori mai mici decât valoarea din nodul X, iar subarborele drept va conține valori mai mari decât cea din nodul X

• inserare, căutare, ștergere în O(log N) average și O(N) worst-case

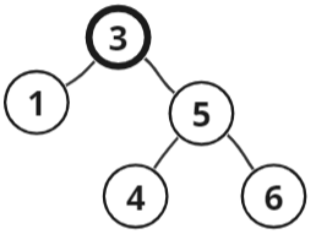
Problema 1:

Avem valorile 1, 4, 5, 6. În câte moduri distincte le putem insera într-un arbore binar de căutare cu rădăcina 3, astfel încât să aibă înălțimea minimă?

Soluţie:

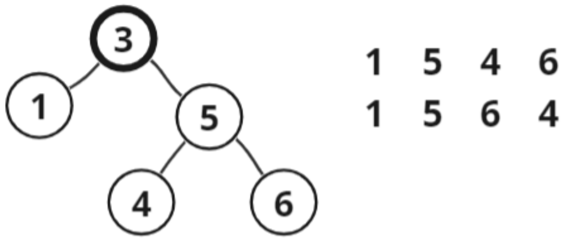
Răspuns: 8

- Inserăm 5 primul: 1, 4, 6 pot fi inserate în orice ordine → 3!=6 moduri



5	1	4	6
5	1	6	4
5	4	1	6
5	4	6	1
5	6	1	4
5	6	4	1

- Inserăm 1 și 5: 4 și 6 pot fi inserate în orice ordine → 2 moduri



Problema 2:

*Se dau o valoare **X** și un arbore binar de căutare. Găsiți cea mai apropiată valoare de **X** din arbore.*

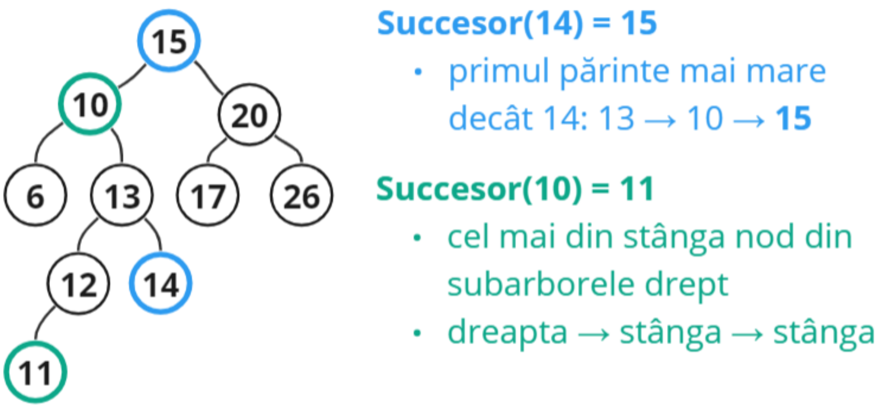
Soluție:

Să începem prin a defini noțiunile de succesor și predecesor într-un arbore binar de căutare.

- **Succesorul unui nod cu valoarea Y** = cel mai mic număr Z din arbore cu proprietatea că Z > Y; **cum îl găsim**: ne dorim un număr mai mare decât Y, așadar ne vom uita în subarborele drept. Ne dorim CEL MAI MIC număr mai mare decât Y, așadar cel mai mic număr din subarborele drept, anume <u>cel mai din stânga nod din subarborele drept</u>.

  * ne deplasăm dreapta → stânga → stânga → ... → stânga

  * **dacă nodul cu valoarea Y nu are fiu drept:** succesorul este primul părinte cu valoare mai mare decât Y

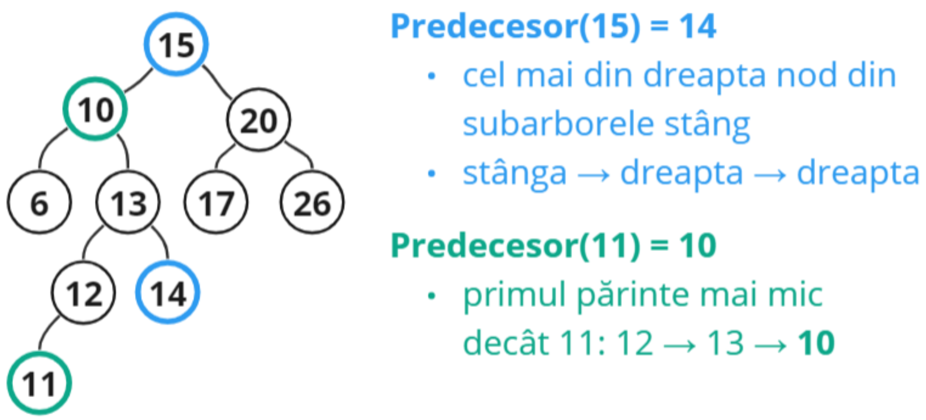


- **Predecesorul unui nod cu valoarea Y** = cel mai mare număr Z din arbore cu proprietatea că Z < Y; **cum îl găsim**: ne dorim un număr mai mic decât Y, așadar ne vom uita în subarborele stâng. Ne dorim CEL MAI MARE număr mai mic decât

Y, așadar cel mai mare număr din subarborele stâng, anume <u>cel mai din dreapta nod din subarborele stâng</u>.

* ne deplasăm stânga → dreapta → dreapta → ... → dreapta

* **dacă nodul cu valoarea Y nu are fiu stâng:** predecesorul este primul părinte cu valoare mai mare decât Y



Îl căutăm pe **X** să verificăm dacă se găsește fix el în arbore. În caz contrar, îl inserăm pe **X**, îi căutăm succesorul și predecesorul și vedem care este mai apropiat de **X**.

Complexitate timp: O(H) - unde H este înălțimea arborelui

Alternativ, îl putem căuta pe **X**, și în drum dinspre rădăcină să facem diferența dintre **X** și fiecare nod și să reținem diferența minimă. Dacă l-am insera pe **X**, inserția ar avea loc într-o frunză, deci atât succesorul, cât și predecesorul, s-ar găsi în lanțul de părinți ai lui **X**.

<u>Problema 3:</u>

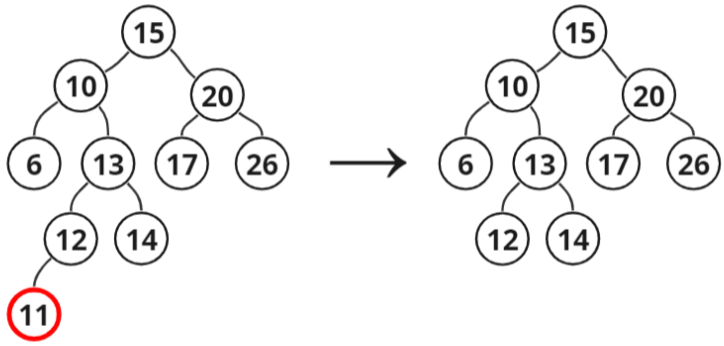
*Se dă un arbore binar de căutare și o valoarea **X**. Să se șteargă nodul cu valoarea **X** din arbore.*

Problema pe Leetcode: https://leetcode.com/problems/delete-node-in-a-bst/description/

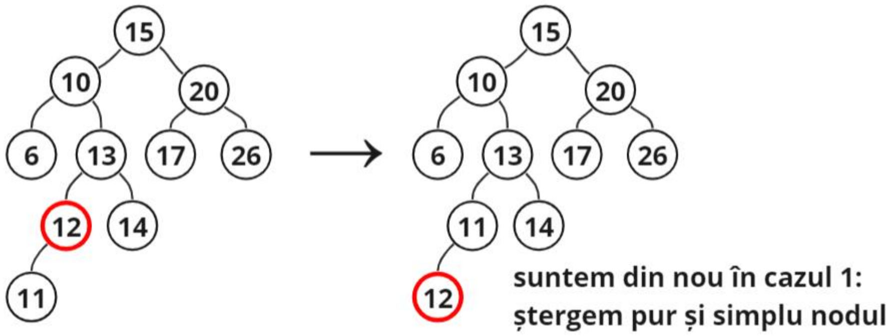
Soluție:

În primul rând, căutăm nodul cu valoarea **X** în arbore. Se disting trei cazuri (ultimele două sunt recursive, prin "ștergem nodul" ne referim la reluarea întregii operații):

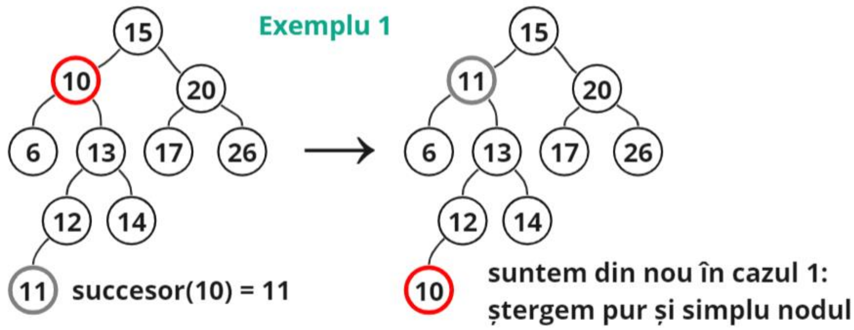
- Nodul este o frunză → îl ștergem pur și simplu

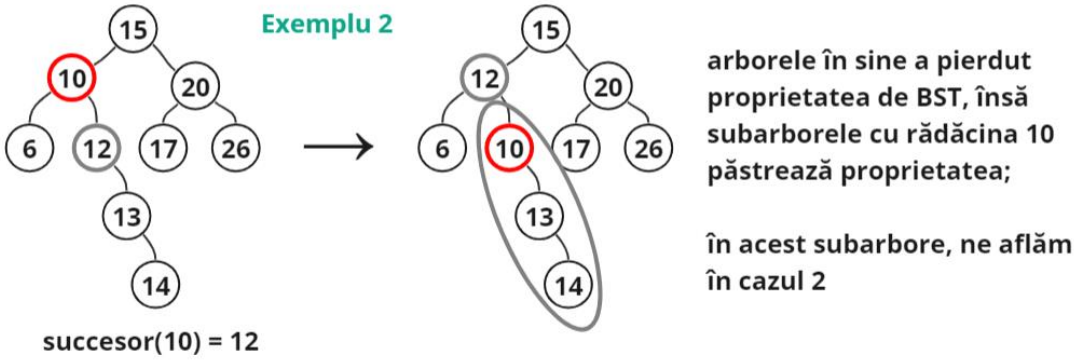


▪ Nodul are un singur copil → facem swap între nod și copil, apoi ștergem nodul



▪ Nodul are mai mulți copii → găsim succesorul, facem swap între nod și succesor, apoi ștergem nodul





Problema 4:

*Se dau un arbore binar de căutare și două valori **X** și **Y**. Găsiți toate valorile din intervalul **(X ; Y)** din arbore.*

Soluție:

Parcurgere inordine în care afișăm doar valorile din intervalul **(X ; Y)**. Alternativ, putem calcula **succesor(X)**, apoi **succesor(succesor(X))**, etc., până când ajungem la o valoare mai mare decât **Y**.

Deși ambele variante au **complexitate timp O(N)**, cea de-a doua ar putea merge mai repede atunci când intervalul e mic (deoarece simulează parcurgerea inordine doar pe intervalul **(X ; Y)**).

Problema 5:

*Dându-se un arbore binar de căutare și un număr **K**, să se determine dacă există o pereche de noduri în arbore a căror sumă este **K**.*

Problema pe Leetcode: https://leetcode.com/problems/two-sum-iv-input-is-a-bst/description/

Soluția 1:

Parcurgem pe rând elementele, iar pentru fiecare element **X** căutăm în arbore **(K-X)**.

Complexitate timp: O(N * H) - unde H este înălțimea arborelui

Soluția 2:

Parcurgerea inordine (inorder) ne dă valorile în ordine crescătoare! Așadar, putem în **O(N) timp** să reținem valorile **sortate** într-un vector. De aici, putem folosi tehnici precum cea cu doi pointeri, păstrând timpul de **O(N)** și memoria de **O(N)** cu care reținem vectorul.

<u>Problema 6:</u>

*Dându-se un arbore binar de căutare și un număr **K**, să se determine al K-lea cel mai mic element din arbore.*

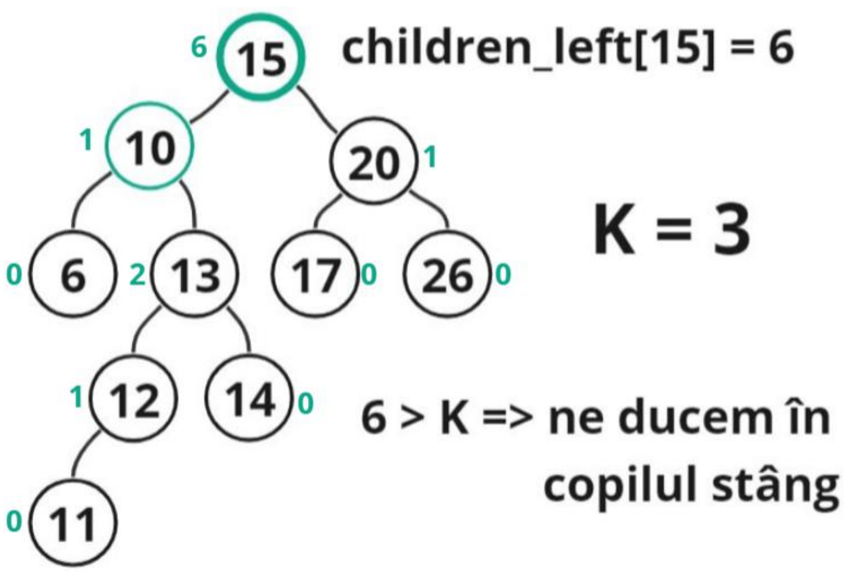
Soluția 1:

Parcurgem nodurile folosind parcurgere inordine, și deoarece aceasta parcurge nodurile în ordine crescătoare, ne putem opri după K pași și să returnăm ultimul nod parcurs.

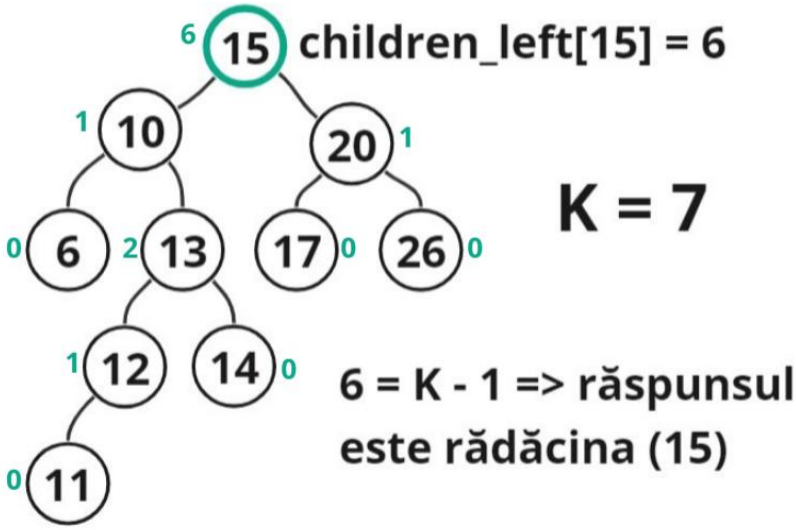
Soluția 2:

Reținem pentru fiecare nod câți copii are în subarborele stâng. Notăm cu **children_left[X]** numărul de copii din subarborele stâng al nodului **X**. Începem din rădăcină (X = root) și se disting cazurile:

- **children_left[X] >= K**: acest lucru înseamnă că există minim **K** numere mai mici decât **X**. În acest caz, ne dorim să căutăm în subarborele stâng (numerele mai MICI decât **X**).



- **children_left[X] = K – 1**: acest lucru înseamnă că **X** are exact **(K – 1)** numere mai mici decât el, așadar **X** este al **K**–lea număr.



- **children_left[X] < K – 1**: acest lucru înseamnă că pot să ignor primele **children_left[X]** numere plus nodul **X**. Adică pot merge în subarborele drept și să caut al **(K – children_left[X] – 1)** – lea număr. Acest lucru este mult mai simplu de vizualizat decât de citit: