Examen Structuri de Date

-Rezolvare-

Varianta 1

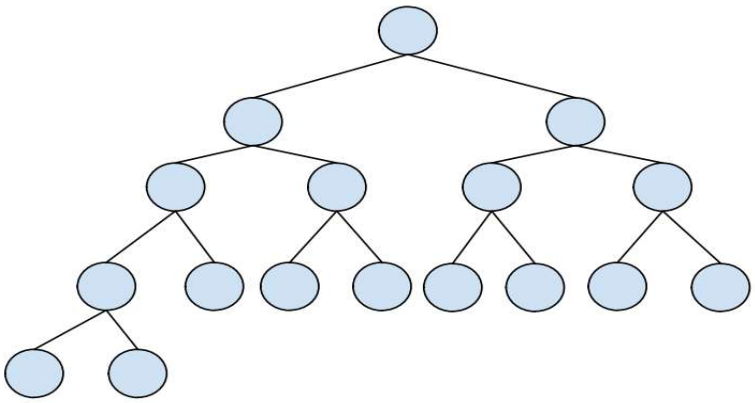
1) Ce înălțime poate sa aibă un arbore binar cu 17 elemente?

Desenați arborele de înălțime minimă și cel de înălțime maximă.

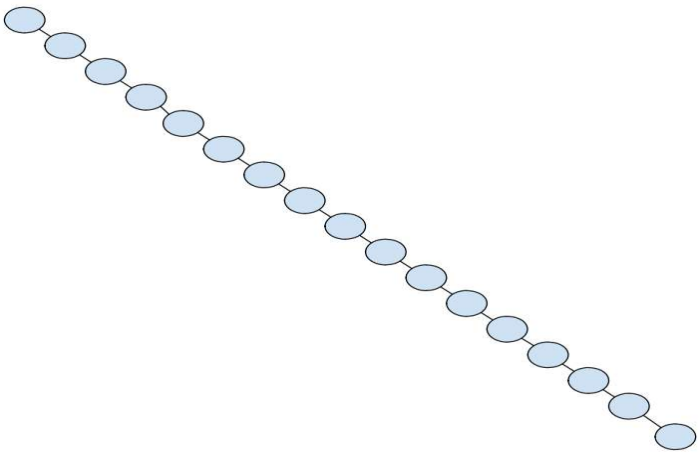
Rezolvare:

Un arbore binar cu 17 elemente poate avea înălțimea cuprinsă între 5 și 17.

Înălțimea minimă este 5



Înălțimea maximă este 17



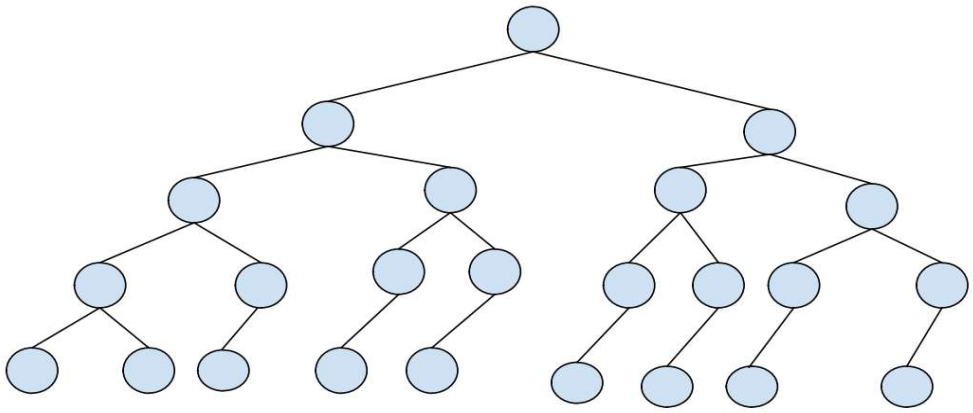
2) Dar unul echilibrat cu 24 de elemente?

Desenați arborele de înălțime minimă și cel de înălțime maximă.

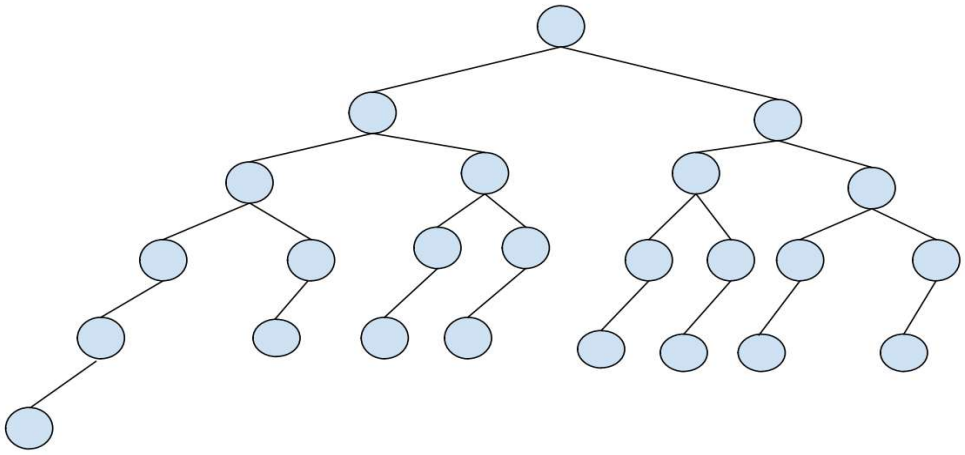
Rezolvare:

Un arbore binar echilibrat cu 24 elemente poate avea înălțimea cuprinsă între 5 și 6.

Înălțimea minimă este 5

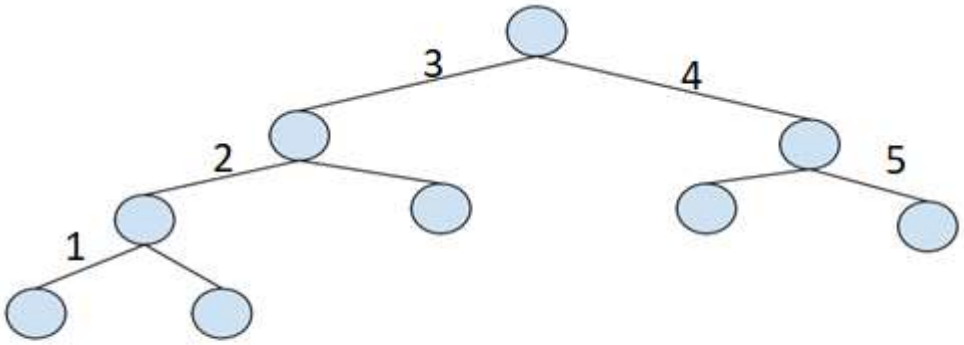


Înălțimea maximă este 6



3) Construiți un arbore binar cu 9 noduri si diametrul 5.

Rezolvare:



4) Exemplificați cum funcționează merge sort pe vectorul:

a)7 9 13 15 12 8 9 6

Rezolvare:

Împărțim vectorul în doi subvectori:

v1 = 7 9 13 15

v2 = 12 8 9 6

Împărțim cei doi subvectori în alți doi subvectori:

v11 = 7 9

v12 = 13 15

v21 = 12 8

v22 = 9 6

Am ajuns la 4 subvectori de lungime 2, așa că iterăm prin ei, în perechi de câte 2 și interclasăm valorile în ordine crescătoare, obținând 2 vectori sortați:

v112 = 7 9 13 15

v122 = 6 8 9 12

Apoi îi interclasăm pe aceștia și obținem vectorul sortat:

v = 6 7 8 9 9 12 13 15.

5) Câte inversiuni sunt în vectorul

a)16 4 9 2 11 3 14 1 (3 1 2 are 2 inversiuni 3 1 și 3 2)

Rezolvare:

16 -> 4, 9, 2, 11, 3, 14, 1

4 -> 2, 3, 1

9 -> 2, 3, 1

2 -> 1

11 -> 3, 1

3 -> 1

14 -> 1

=> în total vom avea 18 inversiuni

Se mai poate calcula numărul de inversiuni cu ajutorul algoritmului de sortare Merge Sort, într-o complexitate de O(nlogn).

6) Inserați pe rând într-un min-heap valorile

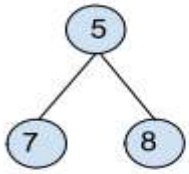
a)7 5 8 1 9 3 6 2

Rezolvare:

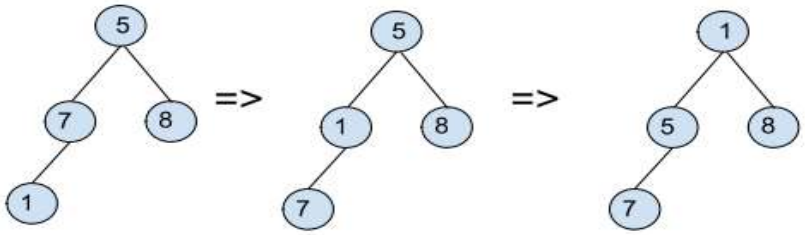
Inserez 7

Inserez 5

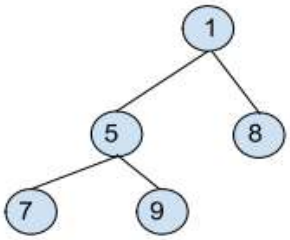
Inserez 8



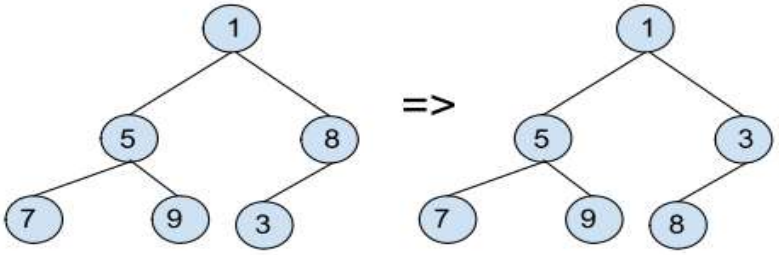
Inserez 1



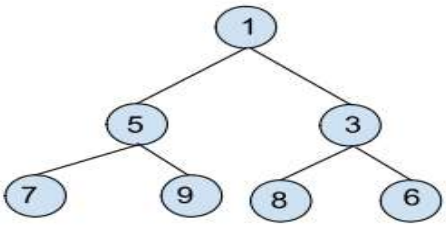
Inserez 9



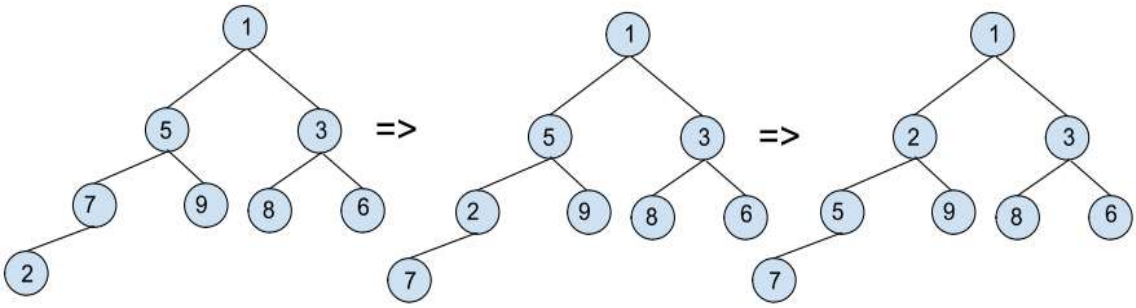
Inserez 3



Inserez 6



Inserez 2



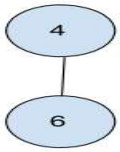
7) Inserați următoarele valori într-un heap binomial de minim

a)4 6 8 9 3 1 2 11

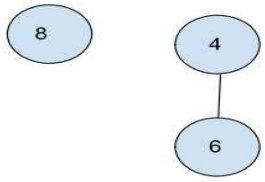
Rezolvare:

Inserez 4

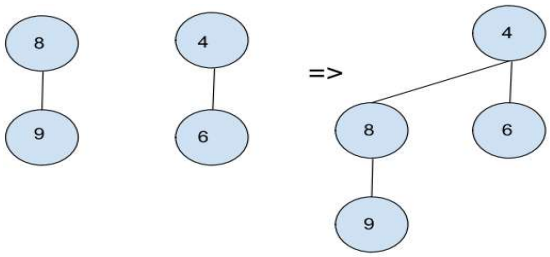
Inserez 6



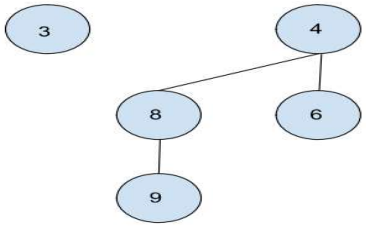
Inserez 8



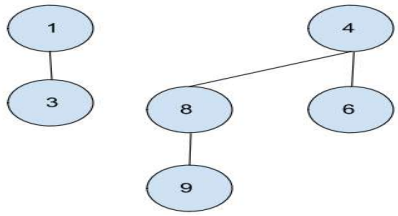
Inserez 9



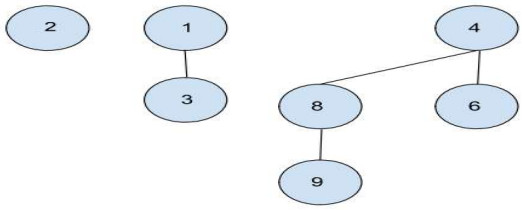
Inserez 3



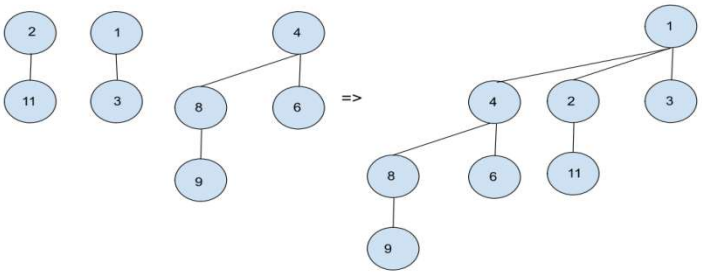
Inserez 1



Inserez 2



Inserez 11



8) Dacă vrem să sortăm 1000000 numere mai mici egale cu 50000 ce algoritm ar fi bine să folosim? De ce?

Rezolvare:

Counting Sort, deoarece având numere mici este simplu și rapid să implementăm un vector de frecvență în care să ținem evidența numerelor citite. Această soluție are o complexitate de O(n+max), unde max este cea mai mare valoare citită.

9) Dacă vrem să sortam 1.000.000 numere mai mici egale cu 50.000.000.000 ce algoritm ar fi bine sa folosim? De ce?

Rezolvare:

Radix Sort, deoareca având numere mari putem crea bucketuri de 2^8, pe care să le ordonăm rapid și efficient. Această sortare are o complexitate de O(nlogmax), unde max este cea mai mare valoare citită. Acest algoritm este mai efficient decât Quick Sort sau Merge Sort pentru ordonarea unor numere foarte mari.

10) Cât ne costă să găsim cel mai mic element dintr-o stivă? Cum îl găsim?

Rezolvare:

Găsirea celui mai mic element dintr-o stivă ne costă O(n), unde n reprezintă numărul de elemente din stivă, deoarece este necesar să iterăm în timp liniar prin stivă și să reținem întotdeauna minimul curent.

11) Care sunt operațiile specifice unui deque?

Rezolvare:

push_back() – adaugă un element la finalul listei

pop_back() – elimină elementul de la finalul listei

push_front() - adaugă un element la începutul listei

pop_front() – elimină elementul de la începutul listei

back() – extrage fără să elimine elemental de la finalul listei

front() – extrage fără să elimine elementul de la începutul listei

12) Inserați într-un hash valorile 14, 23, 24, 42, 81, 67, 29, 34, 87 folosind funcția de dispersie x%10. Puteți alege cum rezolvați coliziunile, motivați decizia.

Rezolvare:

0
1 81
2 42
3 23
4 14 -> 24 -> 34
5
6
7 67 -> 87
8
9 29

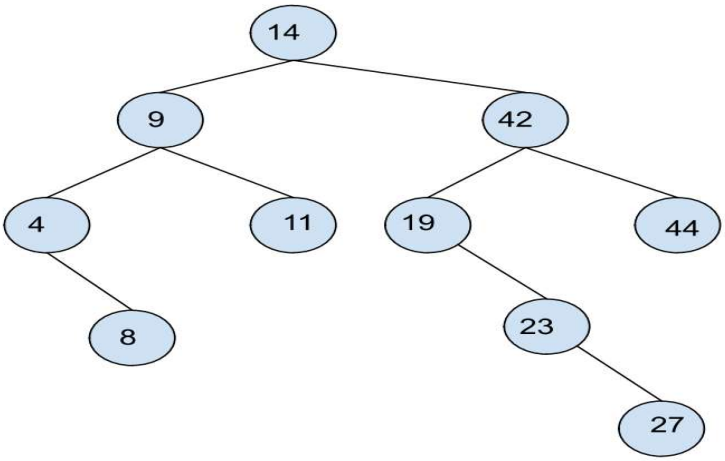
Coliziunile pot fi rezolvate prin implementarea unei liste înlănțuite, în cadrul căreia legăm valorile ce se suprapun. Astfel, pe un caz pur teoretic, toate cele n elemente ar putea fi repartizate în aceeași locatie, însă pe cazuri practice lungimea medie a celui mai lung lanț este de lg(n). Memorie: O(n). Accesare rapidă.

Coliziunile mai pot fi rezolvate și cu ajutorul căutării liniare, adica punem valoarea în următoarea căsuță liberă: pornim de la 0 în jos până găsim un spațiu gol.

Coliziunile mai pot fi rezolvate și cu ajutorul adresării directe, adică în momentul în care o căsuță din hash este deja ocupată vom aplica în mod repetat funcții diferite de hash până când vom găsi un spațiu neocupat. Memorie O(n). Accesare problematică.

13) Inserați într-un arbore binar de căutare următoarele valori: 14 42 19 9 4 11 8 23 44 27

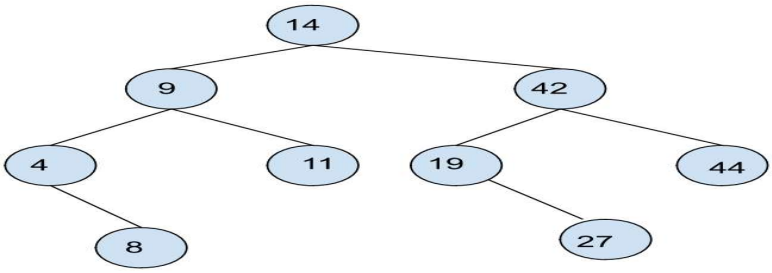
Rezolvare:



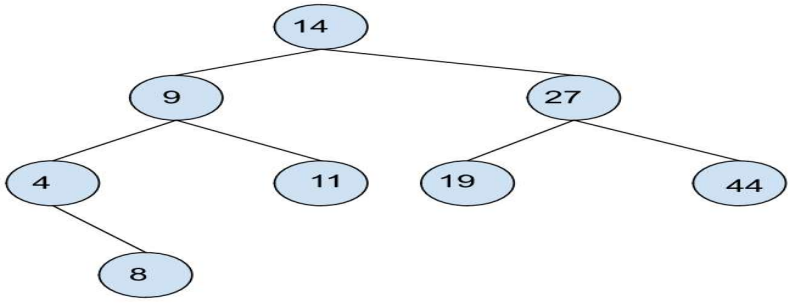
14) Șterge-ți pe rând 23 42 14 din acel arbore

Rezolvare:

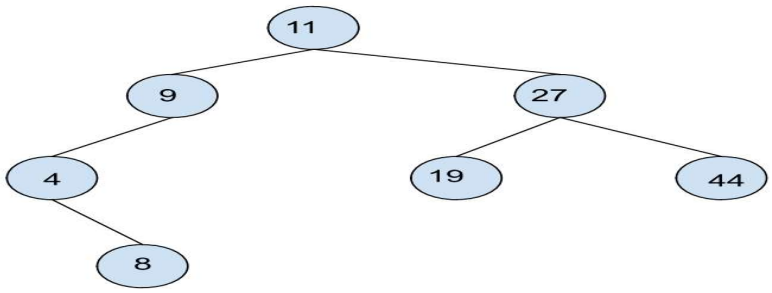
Șterg 23 (23 are un singur fiu, iar când îl ștergem, acesta îi ia locul)



Șterg 42 (42 are 2 fii, și trebuie să rebalansăm arborele, astfel încât să nu stricăm structura, prin urmare vom pune valoarea 27 în locul lui 42, fiind alegerea optimă deoarece este frunză și îl are ca fiu din dreapta pe 19 și fiu din stânga pe 44)

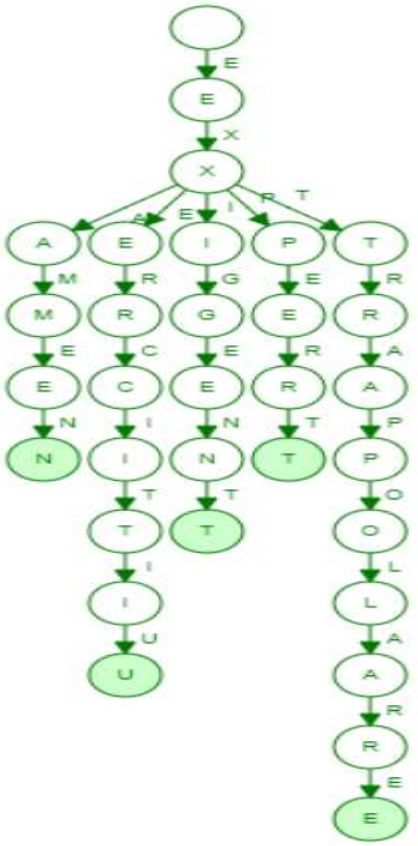


Șterg 14 (14 este rădăcina arborelui , ca sa șterg 14 și să nu modific structura arborelui, pun 11 în locul său, astfel încât 9 să rămână fiul din stânga și 27 fiul din dreapta, 11 este cea mai eficientă soluție deoarece este frunză și nu necesită modificări suplimentare)



15) Inserați într-un trie cuvintele : examen, exercițiu, extrapolare, exigent, expert

Rezolvare:



16) Demonstrați că orice algoritm care afișează elementele unui heap sortat are complexitatea Ω(n logn).

Rezolvare:

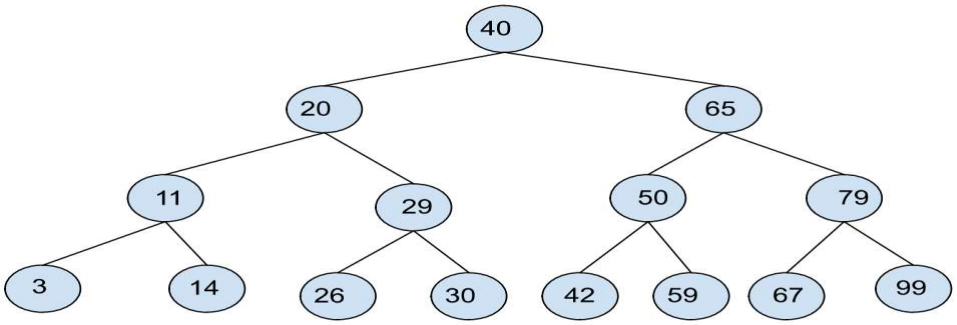
Un heap sortat va avea întotdeauna elementul minim în rădăcină. Pentru a extrage acel minim este necesar să rearanjăm heapul prin ReheapDown. Această operație va impinge rădăcina până când aceasta va ajunge frunză. Operația de ReheapDown costă O(logn), iar dacă vom avea un vector de n elemente, această operație va avea loc de n ori, așadar complexitatea este de O(nlogn).

Dacă această sortare ar putea avea loc în O(n), atunci ar fi mult mai eficientă o sortare prin comparații.

17) Desenați un arbore binar echilibrat cu 15 noduri (alegeți voi ce arbore binar echilibrat).

Rezolvare:

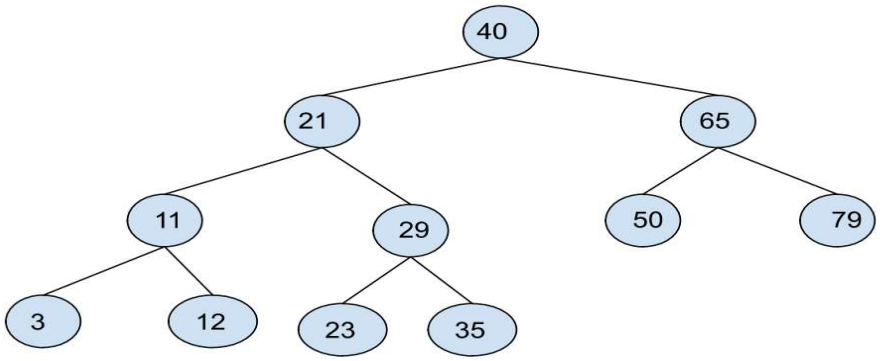
AVL



18) Explicați ce face LCA si arătați cum funcționează pe un arbore pe măcar 8 noduri.

Rezolvare:

LCA (Lowest Common Ancestor) este un algoritm care determină pentru două noduri dintr-un arbore cel mai mic strămoș comun. Acesta se poate implementa ca o matrice ce ține minte pentru fiecare nod nivelul pe care se află și strămoșul de pe nivelul k.



De exemplu:

LCA(11, 35) = 21

LCA(3, 50) = 40

LCA(23, 29) = 29

19) Câți arbori binari de căutare distincti cu 6 noduri se pot forma? Explicați cum ați obținut acest rezultat.

Rezolvare:

Se pot obține (2n)! / ((n+1)! * n!) arbori binari de căutare distincți cu n noduri.

Pentru n = 6 avem (12)! / (7! * 6!) = 132

https://www.geeksforgeeks.org/total-number-of-possible-binary-search-trees-with-n-keys/

20) Cum folosim un arbore de intervale să sortam un vector în O(n log n) ?

Rezolvare:

Memorăm într-un arbore de interval valorile din vectorul pe care vrem să îl sortăm, iar apoi, pentru fiecare pereche de copii reținem în tată valoarea minimă dintre aceștia. Vom ajunge cu minimul în rădăcină, de unde îl extragem de fiecare data, iar în locul valorii minime (în frunză) punem +infinit, astfel încât să nu mai selectăm niciodată valoarea acelui nod. Operația de aflare a minimului ne costă O(logn), iar această operație va avea loc de n ori, pentru cele n numere. Așadat complexitatea finală a algoritmului este de O(nlogn).