Model Examen – Seria 13
Rezolvari

1. AVL Trees

Cerinta: Care este inaltimea maxima a unui AVL tree cu 4 noduri? Presupunem ca inaltimea unui arbore cu un nod este 0.

Inaltimea maxima este 2.

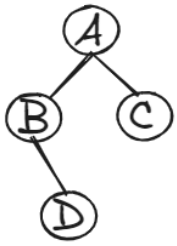

Figure 1: AVL Tree with 4 nodes

2. Tree-walks

Cerinta: Traversarea in *preordine* a unui arbore binar de cautare este **30, 20, 10, 15, 25, 23, 39, 35, 42**. Care este traversarea in *postordine*?

Ne folosim de traversarea in preordine ca sa construim arborele, iar apoi ii afisam traversarea in postordine. Aceasta va fi **15, 10, 23, 25, 20, 35, 42, 39, 30**.

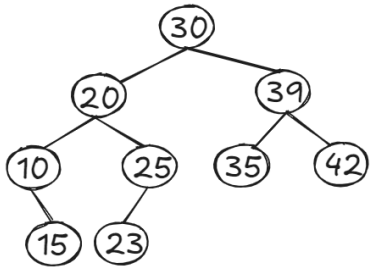

Figure 2: BST from a preorder tree-walk

3. Deletion in BST

Cerinta: pentru a efectua o stergere intr-un arbore binar de cautare asupra unui nod cu doi copii, trebuie sa ii gasim succesorul. Care din urmatoarele afirmatii sunt adevarate?

a. Succesorul este intotdeauna un nod frunza.

b. Succesorul este intotdeauna fie un nod frunza, fie un nod fara copil stang.

c. Succesorul poate fi un stramos al nodului.

d. Succesorul este intotdeauna fie un nod frunza, fie un nod fara copil drept.

Singura varianta corecta este **b**. Ca sa gasim succesorul unui nod, mergem in copilul drept si apoi incontinuu in stanga. Astfel, ne putem opri intr-o frunza, sau intr-un nod care are doar copil drept.

4. Insertion in BST

Cerinta: Urmatoarele valori sunt inserate succesiv intr-un arbore binar de cautare gol: **10, 1, 3, 5, 15, 12, 16**. Care este inaltimea arborelui la final? (inaltimea unui arbore cu un singur nod este 0).

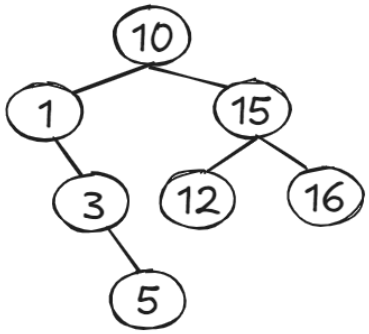

Figure 3: Insertion in a BST

Dupa cum se poate vedea in desen, inaltimea este 3.

5. Hash Tables

Cerinta: Sa consideram urmatoarele elemente **4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199** si functia hash $\mathbf{h(x) = x \bmod 10}$. Cum sunt mapate valorile?

Valorile sunt mapate in felul urmator:

- 2 : 4322
- 4 : 1334
- 1 : 1471 $\rightarrow$ 6171
- 9 : 9679 $\rightarrow$ 1989 $\rightarrow$ 4199
- 3 : 6173

6. Tree-walks

Cerinta: Care din urmatoarele secvente **nu** este una din traversarile in preordine, inordine, postordine ale arborelui din figura de mai jos?

a. A, B, C, D, E, F, G, H, I

b. F, B, D, A, C, E, G, H, I

c. F, B, G, A, D, I, C, E, H

d. Variantele de mai sus nu sunt corecte.

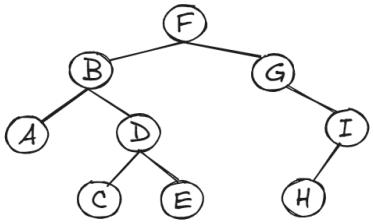

Figure 4: Tree (exercise 6)

Prima secventa reprezinta traversarea in inordine. A doua este o secventa aleatoare → punctul **b** este corect. A treia secventa este parcurgerea in latime → punctul **c** este corect.

7. Tree Rotations

Cerinta: Sa consideram arborele binar de cautare de mai jos. Daca rotim nodul 6 in jurul nodului 3:

a. Nodul 1 este copil al nodului 3.

b. Nodul 4 este copil al nodului 3.

c. Marimea subarborelui nu se schimba, dar radacina sa da.

d. Raspunsurile de mai sus nu sunt corecte.

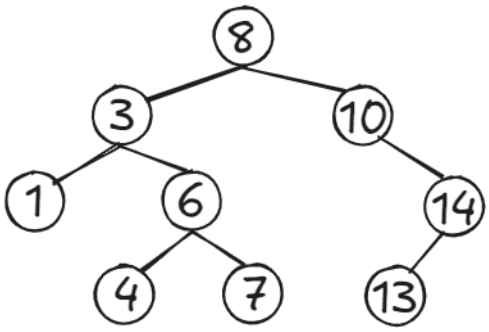

Figure 5: BST before rotation

Nodurile 1 si 4 sunt copiii nodului 3. De asemenea, marimea subarborelui nu se schimba, dar radacina devine 6. Asadar, punctele **a**, **b** si **c** sunt corecte.

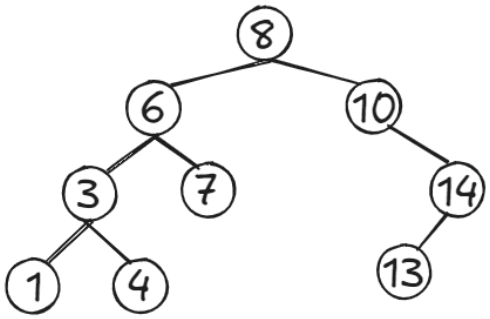

Figure 6: BST after rotation

8. Order-Statistics Trees

Cerinta: De ce adaugam campul **tree-size** la un arbore order-statistics?

Ca sa putem calcula in mod eficient interogari in stilul "al k-lea cel mai mic element" (operatiile devin $O(\log(n))$.

9. Skip Lists

Cerinta: Care din urmatoarele afirmatii sunt adevarate intr-un skip list?

a. Probabilitatea ca un nod sa aiba doi pointeri este exact $\frac{1}{4}$.

b. Elementele sunt sortate in ordine crescatoare.

c. Nivelele sunt spatiate in mod egal.

d. Raspunsurile nu sunt corecte.

Raspunsul corect este **b**. Prima varianta este gresita, deoarece probabilitatea este $\frac{1}{2}$. De asemenea, nivelele nu sunt spatiate in mod egal.

10. Stack

Cerinta: Care dintre urmatoarele secvente de operatii este **imposibila** intr-o stiva cu trei elemente?

a. PUSH, POP, POP, POP, POP, PUSH

b. PUSH, POP, POP, POP, PUSH, POP, POP

c. PUSH, POP, POP, POP, POP, POP, PUSH, POP

d. PUSH, PUSH, PUSH, PUSH, PUSH, PUSH

e. POP, POP, POP, POP, POP, POP, POP

Evident, variantele **c** si **e**.

11. Hash Tables

Cerinta: Cand numarul de slot-uri intr-o tabela de dispersie se dubleaza, ce se intampla cu incarcarea tabelei (load factor)?

a. Se dubleaza.

b. Se injumatateste.

c. Ramane la fel.

d. Raspunsurile nu sunt corecte.

Incarcarea tabelei este definita de relatia $\frac{m}{n}$, unde **m** este numarul de slot-uri ocupate, iar **n** este numarul total de slot-uri. Daca numarul de slot-uri se dubleaza, incarcarea tabelei va deveni $\frac{m}{2n}$, ceea ce inseamna ca se injumatateste $\rightarrow$ varianta **b**.

12. Median of a BST

Cerinta: Presupunand ca arborele de mai jos este unul de cautare, unde se afla elementul median?

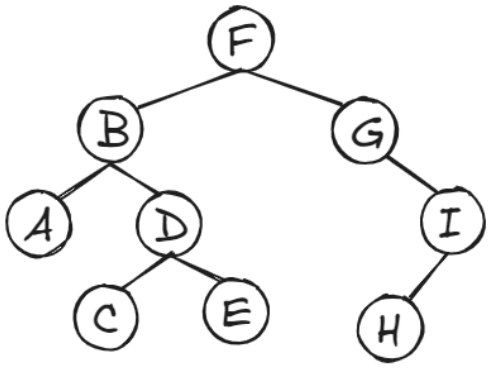

Figure 7: BST (exercise 12)

Luand traversarea in inordine, obtinem secventa **A, B, C, D, E, F, G, H, I**. Deoarece lungimea secventei este impara, elementul median este cel din mijloc $\rightarrow$ elementul **E**.

13. Convex Hulls

Cerinta: Cand calculam infasuratoarea convexa a n puncte:

a. Folosim o coada.

b. Sortam punctele dupa coordonata O_y.

c. Sortam punctele dupa coordonata O_x.

d. Raspunsurile nu sunt corecte.

Pentru a calcula infasuratoarea convexa, cunoastem doi algoritmi: **Graham's Scan**, care nu foloseste o coada (ci o stiva) si sorteaza dupa unghiul polar format cu punctul de baza al poligonului (nu dupa axe); **Jarvis' March**, care nu se foloseste de cozi sau stive si nu face sortari. Asadar, raspunsurile nu sunt corecte

14. Tree-walks

Cerinta: Sa presupunem ca modificam algoritmul de parcurgere in latime (BFS) a unui arbore binar in felul urmator: in loc de o coada, folosim o coada dubla; cand scoatem primul nod din coada, mai intai il vizitam, iar vecinii nodului scos ii adaugam in varful cozii duble. Modificarea astfel descrisa cu ce fel de parcurgere este echivalenta?

Modificarea, de fapt, simuleaza o stiva. Ultimul nod adaugat in varful cozii duble va fi primul nod scos si afisat. Asadar, depinde de ordinea in care adaugam elementele in deque! Exista doua cazuri:

- Afisam nodul curent, adaugam copilul drept si apoi copilul stang. Asta inseamna ca urmatorul nod scos si afisat va fi copilul stang. Parcurgerea este echivalenta cu $print(node.val) \rightarrow f(node.left) \rightarrow f(node.right)$, care este, de fapt, parcurgerea in **preordine**.
- Afisam nodul curent, adaugam copilul stang si apoi copilul drept. Urmatorul nod scos si afisat va fi copilul drept. Parcurgerea este echivalenta cu $print(node.val) \rightarrow print(node.right) \rightarrow print(node.left)$ (care nu cred ca are un nume).

15. Dynamic Sets

Cerinta: Vrem sa implementam urmatoarea operatie pe o multime dinamica de elemente: **query(a,b)** returneaza numarul de elemente ale lui S in intervalul $[a, b]$. Vrem ca operatia sa aiba complexitate $O(log(n))$. De la ce structura de date ar fi cel mai bine sa pornim?

a. Vector.

b. Hash Table.

c. Splay Tree.

d. Order-Statistics Tree.

e. Raspunsurile nu sunt corecte.

Hash Table-urile nu au treaba cu asemenea operatii. Vectorii sunt nefolositori. Splay tree-urile au un alt scop. **Order Statistics Tree** este varianta perfecta pentru o asemenea operatie.

16. Skip Lists

Cerinta: Sa presupunem ca modificam un skip list ca sa putem face salturi inainte si inapoi, folosind liste dublu inlantuite pe fiecare nivel. Ne vom limita la a implementa o lista cu patru nivele de pointeri. Nivelul unui nod va fi ales exact ca la un skip list obisnuit. Numarul total mediu de pointeri in varianta noastra este:

a. $\Theta(n)$.

b. $\Theta(n \log(n))$.

c. $\Theta(n^2)$.

d. Raspunsurile nu sunt corecte.

Daca desenam primul nivel, vom observa ca pentru n noduri vom avea $2 * (n + 1)$ pointeri. De fiecare data cand urcam in nivel, ar trebui sa se injumatateasca numarul de noduri $\rightarrow$ pe al doilea nivel vom avea nevoie de $n + 1$ pointeri, pe al treilea nivel de $\frac{n+1}{2}$ pointeri si pe ultimul nivel de $\frac{n+1}{4}$ pointeri.

Facem suma pointerilor de pe fiecare nivel si obtinem:

$$2 * (n + 1) + n + 1 + \frac{n + 1}{2} + \frac{n + 1}{4} = \frac{8n + 8 + 4n + 4 + 2n + 2 + n + 1}{4} = \frac{15n + 15}{4} = \frac{15}{4} + \frac{15}{4} * n \in \Theta(n)$$

17. Hash Tables

Cerinta: Sa consideram o schema de double hashing care mapeaza elementele unui univers U pe multimea $0, 1, \ldots, m-1$ via functia $h(x,i) = (h_1(x) + i * h_2(x)) \mod m$, unde **m** este marimea tabelei, iar $h_1(x)$ si $h_2(x)$ sunt functii hash "simple". De asemenea, sa consideram functiile $H_1(x,i) = (h_2(x) + i * h_1(x)) \mod m$ si $H_2(x,i) = (h_2(x) - 1 + i * (h_1(x) + 1)) \mod m$. Care dintre functiile H_1 si H_2 sunt potrivite, in principiu, in loc de h pentru double hashing?

a. H_1 dar nu si H_2.

b. H_2 dar nu si H_1.

c. si H_1 si H_2.

d. nici H_1 si nici H_2.

Cel mai sigur este sa spunem ca H_1 si H_2 nu sunt potrivite, deoarece:

- Pentru H_1, ar trebui ca $h_1(x)$ sa fie nenul si coprim cu m ca sa putem accesa toate sloturile din tabel.
- Pentru H_2, exista problema ca sarim peste anumite sloturi.

18. Red-Black Trees

Cerinta: Vrem sa reprezentam multimea **1, 2, 3** cu un red-black tree. In cate moduri putem face acest lucru? Doi arbori pot diferi fie prin forma, fie prin culoare.

Exista doar doua moduri.

Figure 8: Red-Black Trees with 3 elements

19. AVL Trees

Cerinta: Vrem sa reprezentam multimea **1, 2, 3** cu un AVL tree. In cate moduri putem face acest lucru?

Exista un singur mod.

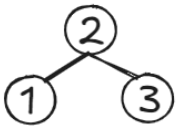

Figure 9: AVL Tree with 3 elements

20. Skip Lists

Cerinta: O lista skip reprezinta multimea de elemente **1, 2, 3, 5, 8, 13, 21, 44**. In al catelea nod vom gasi elementul 5?

Intr-un skip list, elementele sunt sortate crescator → valoarea 5 se gaseste in al patrulea nod.

21. B-Trees

Cerinta: Fie x o cheie intr-un B-Tree. Unde se poate afla succesorul lui x (presupunand ca exista)?

a. In acelasi nod cu cheia originala.

b. Intr-un descendent al nodului in care se afla cheia.

c. Intr-un nod stramos al nodului in care se afla cheia.

d. Mai exista si alte alternative neluate in calcul.

Se poate afla si in acelasi nod, si intr-un stramos, si intr-un descendent.

22. Sorting Algorithms

Cerinta: Cand rulam algoritmul **heap-sort**, numarul maxim de swap-uri de elemente se atinge cand:

a. Secventa este sortata crescator.

b. Secventa este sortata descrescator.

c. Secventa este aleatoare.

d. Raspunsurile nu sunt corecte.

Algoritmul **heap-sort** utilizeaza un **max-heap**: fiecare nod are valoarea mai mare decat copiii sai. Cand secventa este sortata **descrescator**, proprietatea de max-heap este indeplinita; mai ramane doar sa extragem incontinuu valoarea maxima, punand-o la indexul corespunzator. Cand secventa este sortata **crescator**, proprietatea de max-heap este total incalcata; o sa existe mult mai multe comparatii in apelul functiei **BuildHeap()**. In concluzie, numarul maxim de swap-uri de elemente se atinge cand secventa este sortata crescator.hat d

23. Heaps

Cerinta: In cate moduri putem pune elementele **1, 2, 3, 4** intr-un vector, astfel incat vectorul rezultant sa fie vazut drept un **min-heap**?

Exista 3 moduri.

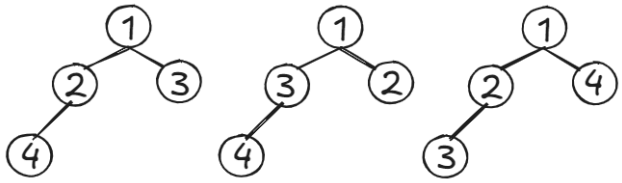

Figure 10: Min-Heap

24. Code

Cerinta: Ce face urmatorul cod?

Algorithm 1: Functie recursiva

```
1 void f (struct Node* head){
2     if head == NULL then
3         return;
4     f(head->next);
5     cout << head->data;
```

Codul afiseaza valorile dintr-o lista inlantuita, in ordine descrescatoare.

25. Red-Black Trees

Cerinta: Daca arborele de mai jos ar fi un Red-Black Tree, care noduri ar fi colorate in mod sigur cu **negru**?

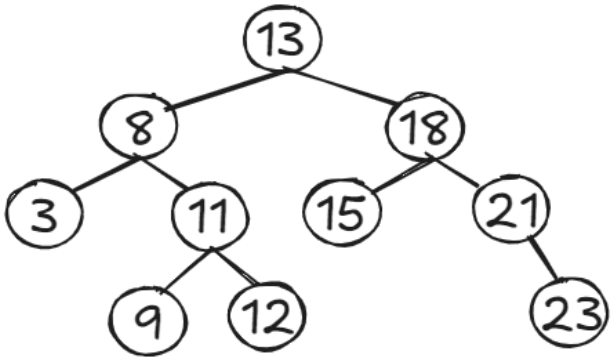

Figure 11: Uncolored Red-Black Tree

Nodul 13 (deoarece este radacina) si 3, 15, 21. Cea mai simpla rezolvare este realizarea desenului.

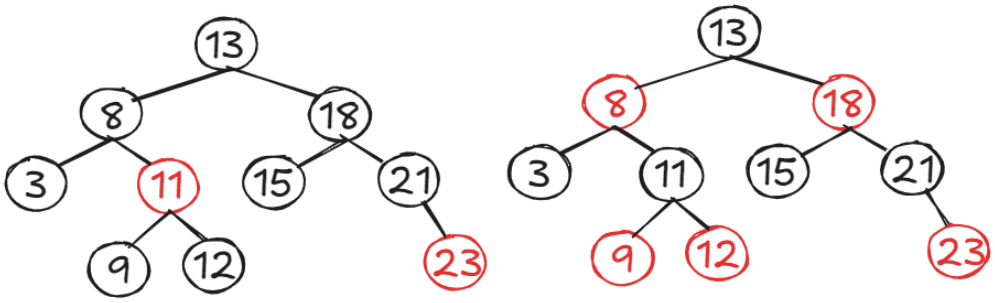

Figure 12: Solution (exercise 25)