Tema 5 - Arbori binar de căutare

1. Implementaţi un arbore binar de căutare cu chei numere întregi. Utilizaţi o structură NOD, care are un câmp de tip int, ce stochează cheia nodului şi trei câmpuri de tip pointer la NOD pentru fiul stâng, fiul drept şi părintele nodului. De asemenea structura NOD dispune de un constructor care setează câmpul int la o valoare transmisă prin parametru şi câmpurile de tip pointer la NOD le iniţializează cu NULL. Utilizaţi apoi o structură de tip ARBORE_CAUT, care are ca membru RADACINA de tip pointer la NOD În plus structura trebuie să aibă metodele:

   - INSERT - inserează un nou nod în arbore (0.25 p)
   - MAXIM(NOD *x) / MINIM(NOD *x)- returnează nodul cu cheia maximă / minimă din subarborele de rădăcină x(0.25 p ambele funcţii)
   - SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p ambele funcţii)
   - SEARCH(int val) - returnează nodul cu valoarea *val* dacă există sau NULL altfel. (0.25 p)
   - DELETE(NOD *x) - şterge din arbore nodul x (care a fost mai întâi identificat prin SEARCH) (0.25 p)
   - PRINT_TREE(int opt) - afişază arborele în preordine (dacă opt=1), inordine (dacă opt=2), în postordine (dacă opt=3), pe niveluri (dacă opt=4). (0.5 p dintre care 0.25 pentru primele 3 afişări şi 0.25 pentru a 4-a).
   - CONSTRUCT - construieşte un AB căutare pornind de la un vector de chei. (0.25p)
   - EMPTY() - verifică dacă arborele este vid. (0.25 p)
   - CLEAR() - şterge toate nodurile din arbore (0.25 p)

   Structura trebuie să dispună de un constructor care iniţializează RADACINA cu NULL. În funcţia *main* se declară o variabilă de tip ARBORE_CAUT şi se foloseşte un *menu* implementat cu ajutorul unei instrucţiuni *switch*, prin

care utilizatorul să poată selecta oricare dintre operaţiile de inserţie, căutare, ştergere, minim, maxim, succesor, predecesor, afişare în cele 4 moduri - la alegere. (1 p)

2. Implementaţi un arbore AVL cu chei numere întregi (pentru template 0.25 p suplimentar). Utilizaţi o structură NOD care dispune de un câmp informaţie, un câmp înălţime şi câmpuri de tip pointer pentru fiii stâng şi drept şi pentru părinte. De asemenea structura NOD trebuie să dispună de un constructor care iniţializeă câmpul informaţie cu valoarea transmisă prin parametru, câmpul câmpul înăţime cu 1 şi câmpurile de tip pointer cu NULL. Utilizaţi o structură AVL care dispune de un membru de tip pointer la NOD, numit RADACINA. În plus dispune de funcţiile:

   - INSERT - inserează un nou nod în arbore (0.25 p).
   - INSERT_REPARA - reface balansare după inserţie (1 p)
   - MAXIM(NOD *x) / MINIM(NOD *x)- returnează nodul cu cheia maximă / minimă din subarborele de rădăcină x (0.25 ambele funcţii).
   - SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p ambele funcţii).
   - SEARCH(int val) - returnează nodul cu valoarea *val* dacă există sau NULL altfel. (0.25 p).
   - DELETE(NOD *x) - şterge din arbore nodul x (care a fost mai întâi identificat prin SEARCH) (0.25 p).
   - DELETE_REPARA(NOD *x) - reface balansarea arborelui după ştergere - (1.25p)
   - ROT_ST, ROT_DR - funcţiile de rotaţie - (0.25 p)
   - PRINT_TREE(int opt) - afişază arborele în preordine (dacă opt=1), inordine (dacă opt=2), în postordine (dacă opt=3), pe niveluri (dacă opt=4). (0.5 p dintre care 0.25 pentru primele 3 afişări şi 0.25 pentru a 4-a). Trebuie afişat şi factorul de balansare pentru fiecare nod.
   - CLEAR() - şterge toate nodurile din arbore (0.25 p)
   - EMPTY() - verifică dacă arborele este vid. (0.25 p)
   - CONSTRUCT - construieşte un arbore AVL pornind de la un vector de chei. (0.25 p)
   - MERGE - reuneşte doi arbori AVL într-un al treilea în complexitate cât mai bună. Observaţie: dacă maximul din $T1$ este mai mic decât minimul din $T2$ sau minimul din $T1$ este mai mare decât maximul din $T2$ atunci problema poate fi rezolvată în complexitate logaritmică. (3p) Punctajul

complet se dă la acest punct dacă există şi o argumentare a complexităţii, atât în cele mai favorabile cazuri cât şi în celelalte cazuri.

Structura trebuie să dispună de un constructor care iniţializează RADACINA cu NULL. În funcţia *main* se declară o variabilă de tip AVL şi se foloseşte un *menu* implementat cu ajutorul unei instrucţiuni *switch*, prin care utilizatorul să poată selecta oricare dintre operaţiile de inserţie, căutare, ştergere, minim, maxim, succesor, predecesor, afişare în cele 4 moduri - la alegere, Join (atunci trebuie doi arbori). (0.75 p)
Observaţie: Această problemă se poate rezolva prin adaptarea programului de la pb. 1.

3. Implementaţi un arbore roşu-negru. (pentru template 0.25 p suplimentar). Utilizaţi o structură NOD care dispune de un câmp informaţie, un câmp culoare (NU de tip şir de caractere!!!) şi câmpuri de tip pointer pentru fiii stâng şi drept şi pentru părinte. De asemenea structura NOD trebuie să dispună de un constructor care setează câmpul informaţie cu valoarea transmisă prin parametru, câmpul culoare la roşu şi câmpurile de tip pointerîn mod adecvat. Utilizaţi o structură ARN care dispune de un membru de tip pointer la NOD, numit RADACINA şi un câmp de tip pointer la NOD numit NIL, care este nodul santinelă. În plus dispune de funcţiile:

   - INSERT - inserează un nou nod în arbore (0.25 p)
   - INSERT_REPARA - reface proprietăţile RN după inserţie (0.75 p)
   - MAXIM(NOD *x) / MINIM(NOD *x)- returnează nodul cu cheia maximă / minimă din subarborele de rădăcină x (0.25 p ambele funcţii)
   - SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p ambele funcţii)
   - SEARCH(int val) - returnează nodul cu valoarea *val* dacă există sau NULL altfel. (0.25 p)
   - DELETE(NOD *x) - şterge din arbore nodul x (care a fost mai întâi identificat prin SEARCH) (0.25 p)
   - DELETE_REPARA(NOD *x) - reface proprietăţile RN după ştergere (1 p)
   - ROT_ST, ROT_DR - funcţiile de rotaţie - (0.25 p)
   - CLEAR() - şterge toate nodurile din arbore (0.25 p)
   - EMPTY() - verifică dacă arborele este vid. (0.25 p)
   - PRINT_TREE(int opt) - afişază arborele în preordine (dacă opt=1), inordine (dacă opt=2), în postordine (dacă opt=3), pe niveluri (dacă

opt=4). (0.5 p dintre care 0.25 pentru primele 3 afişări şi 0.25 pentru a 4-a). Trebuie afişată şi culoarea pentru fiecare nod.

- CONSTRUCT - construieşte un ARN pornind de la un vector de chei. (0.25p)

Structura trebuie să dispună de un constructor care iniţializează RADACINA şi santinela în modul prezentat la curs. În funcţia *main* se declară o variabilă de tip ARN şi se foloseşte un *menu* implementat cu ajutorul unei instrucţiuni *switch*, prin care utilizatorul să poată selecta oricare dintre operaţiile de inserţie, căutare, ştergere, minim, maxim, succesor, predecesor, afişare în cele 4 moduri - la alegere. (0.75 p)

4. **Arbori pentru statistici de ordine**: La un concurs de informatică elevii primesc punctaje între 1 şi 100. La acest concurs participă n elevi. Să se scrie un program care permite:

   a. afişarea elevilor în ordinea punctajului primit. Cei cu acelaşi punctaj vor fi plasaţi în ordine alfabetică.

   b. să se poată determina în mod eficient poziţia în clasament a unui elev pe baza numelui şi a punctajului obţinut.

   c. să se verifice cine se află pe o anumită poziţie din clasament.

   d. să se adauge / elimine persoane.

   e. să de determine în mod eficient nr de persoanele cu punctajul aflat într-un anumit interval

   Puncatj: punctele a-d cumulează 1.5 puncte, punctul e - 1.5 puncte

5. **Arbore de intervale. Problema vaccinării**: Mihai doreşte să se vaccineze împotriva COVID-19. El este însă foarte ocupat. În oraşul său se organizează vaccinări în diferite zile, iar pentru fiecare zi se cunoaşte intervalul orar în care centrul de vaccinare este deschis. Pentru Mihai ziua nu este importantă, dar el poate doar într-un anumit interval orar. Scrieţi un program eficient, care să îl ajute pe Mihai să găsească un o zi şi un interval orar în care se poate vaccina. (trebuie ca intervalul orar în care poate Mihai să se intersecteze cu intervalul orar propus de program). **Atenţie:** Este problemă suplimentară la ARN. (1p).

   Observaţii:

   - implementaţi pentru aceasta un arbore de intervale (având la bază un ARN).

- se permite adăugarea de intervale de vacccinare noi, precum şi eliminarea altora (care au trecut de exemplu)
- trebuie scrisă o funcţie, care are ca parametru intervalul orar dorit şi returnează o propunere corespunzătoare.

6. *B-secvenţă*: Numim B-secvenţă un şi de n numere $a_1, a_2, \ldots, a_n$ cu următoarele proprietăţi:

   - $a_1, a_2 < \ldots < a_j$ şi $a_j > a_{j+1} > \ldots > a_n$
   - fiecare element, cu excepţia maximului apare de cel mult 2 ori în şir: o dată în partea crescătoare, o dată în partea descrescătoare a şirului
   - toate elementele din partea descrescătoare se regăsesc şi în partea crescătoare.

   Se citeşte o astfel de secvenţă S dintr-un fişier. Apoi se realizează K operaţii în modul următor: - pentru fiecare operaţie se citeşe o valoare *val*. Această valoare se inserează în secvenţa S, numai dacă după inserţie se păstrează proprietăţile definite mai sus. De asemenea după fiecare operaţie se primeşte un mesaj, care indică dacă operaţia a putut fi efectuată şi se afişează noua secvenţă. Folosiţi *set* din STL. (2p)

7. *Dicţionar*: Se citeşte dintr-un fişier un text în care cuvintele sunt separate prin spaţii. Pot exista şi semne de punctuaţie. Se cere la final afişarea în ordine alfabetică a cuvintelor. Fiecare cuvânt se afişază o singură dată, având alături numărul de apariţii în text. Utilizaţi **map** din STL. Semnele de punctuaţie se ignoră. (1p)

8. Implementaţi o structură de tip MAP asemănătoare celei din STL, utilizând un ARN sau un AVL. Elementele inserate vor fi de tip perche (cheie, valoare). Structura trebuie să dispună în plus faţă de funcţiile specifice ARN / AVL de:

   - operatorul [ ] - prin care să poată fi accesat un element cu o anumită cheie /respectiv inserat în cazul în care nu există (de ex: MyMap m; m[3]=7 - inserează elementul cu cheia 3 şi valoarea 7.)
   - Implementarea unui iterator cu care să se poată itera prin structură.
   - Funcţia CLEAR - goleşte structura
   - Operatorul = pentru copierea structurii
   - Funcţiile EMPTY( ) şi SIZE( )

   În plus vă asiguraţi de existenţa tuturor operatorilor necesari în structură. Structura sa fie template (pe tipul de date + comparator).

Observaţii:

- Funcţiile INSERT, SEARCH, DELETE, EMPTY, PRINT_TREE, SUCCESOR, PREDECESOR, MINIM , MAXIM, CONSTRUCT, EMPTY, funcţiile de rotaţie se punctează doar o dată, oricâte dintre probleme au fost rezolvate!
- Pentru cod copiat de pe net nota finala nu poate depăsi 4
- Pentru ultima problemă se oferă 1 punct suplimentar la examen (dacă este completă).