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)
   - SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p)
   - 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) (0.5 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() - 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 0.5 p suplimentar). Utilizaţi o structură NOD care dispune de un câmp informaţie, un câmp factor de balansare 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 factor de balansare la 0 şi câmpurile de tip pointer la 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 (1.5 p)
   - MAXIM(NOD *x) / MINIM(NOD *x)- returnează nodul cu cheia maximă / minimă din subarborele de rădăcină x (0.25 p).
   - SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p).
   - 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) (0.5 p).
   - DELETE_REPARA(NOD *x) - reface balansarea arborelui după ştergere - (1.5)p
   - 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.
   - EMPTY() - şterge toate nodurile din arbore (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). (0.25 p)
Observaţie: Această problemă se poate rezolva prin adaptarea programului de la pb. 1.

3. Implementaţi un arbore roşu-negru. (pentru template 0.5 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 (0.5 p)
   - MAXIM(NOD *x) / MINIM(NOD *x)- returnează nodul cu cheia maximă / minimă din subarborele de rădăcină x (0.25 p)
   - SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p)
   - 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) (0.5 p)
   - DELETE_REPARA(NOD *x) - reface proprietăţile RN după ştergere (0.5p)
   - ROT_ST, ROT_DR - funcţiile de rotaţie - (0.25 p)
   - EMPTY() - şterge toate nodurile din arbore (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. (0.25 p)

4. **Îmbogăţirea arborilor binari de căutare**: Implementaţi un arbore pentru statistici de ordine pe baza unui ARN (pb 3) sau a unui AVL (pb 2), adăugând funcţiile: RANG şi SELECT (vezi curs) - fiecare valorează 0.25 p. Pentru refacerea proprietăţii *size* la inserţie, ştergere şi rotaţii - (1.5 p)

5. 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.) (2 p)
   - Implementarea unui iterator cu care să se poată itera prin structură (3p).
   - Funcţia CLEAR - goleşte structura (0.25 p)
   - Operatorul = pentru copierea structurii (1 p)
   - Funcţiile ISEMPTY( ) şi SIZE( ) (0.25 p)

   În plus vă asiguraţi de existenţa tuturor operatorilor necesari în structură. Structura sa fie template. Evident, la această problemă se adaugă şi punctajul aferent de la metodele structurii de arbore.

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!
- Cine implementează atât problema 2 (AVL) cât şi problema 5 (la 5 cu ARN) primeşte un punct suplimentar la nota finală de laborator.