

Tema 5 - Arbori binar de căutare

1. Implementați un arbore binar de căutare cu chei numere întregi. Utilizați o structură NOD, care are un câmp de tip `int`, ce stochează cheia nodului și trei câmpuri de tip pointer la NOD pentru fiul stâng, fiul drept și părintele nodului. De asemenea structura NOD dispune de un constructor care setează câmpul `int` la o valoare transmisă prin parametru și câmpurile de tip pointer la NOD le inițializează cu `NULL`. Utilizați 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 subarboarele de rădăcină x (0.25 p ambele funcții)
- `SUCCESOR(NOD *x) / PREDECESOR(NOD *x)` - returnează nodul care este succesorul / predecesorul nodului x (0.25 p ambele funcții)
- `SEARCH(int val)` - returnează nodul cu valoarea val dacă există sau `NULL` altfel. (0.25 p)
- `DELETE(NOD *x)` - șterge din arbore nodul x (care a fost mai întâi identificat prin `SEARCH`) (0.5 p)
- `PRINT_TREE(int opt)` - afișază arborele în preordine (dacă `opt=1`), înordine (dacă `opt=2`), în postordine (dacă `opt=3`), pe niveluri (dacă `opt=4`). (0.5 p dintre care 0.25 pentru primele 3 afișări și 0.25 pentru a 4-a).
- `CONSTRUCT` - construiește un AB căutare pornind de la un vector de chei. (0.25p)
- `EMPTY()` - șterge toate nodurile din arbore (0.25 p)

Structura trebuie să dispună de un constructor care inițializează `RADACINA` cu `NULL`. În funcția *main* se declară o variabilă de tip `ARBORE_CAUT` și se folosește un *menu* implementat cu ajutorul unei instrucțiuni *switch*, prin care utilizatorul să poată selecta oricare dintre operațiile de inserție, căutare, ștergere, minim, maxim, succesor, predecesor, afișare în cele 4 moduri - la alegere. (1 p)

2. Implementați un arbore AVL cu chei numere întregi (pentru template 0.25 p suplimentar). Utilizați o structură NOD care dispune de un câmp informație, un câmp factor de balansare și câmpuri de tip pointer pentru fii stâng și drept și pentru părinte. De asemenea structura NOD trebuie să dispună de un constructor care inițializează câmpul informație cu valoarea transmisă prin parametru, câmpul factor de balansare cu 0 (sau câmpul înălțime cu 1) și câmpurile de tip pointer cu NULL. Utilizați o structură AVL care dispune de un membru de tip pointer la NOD, numit RADACINA. În plus dispune de funcțiile:

- INSERT - inserează un nou nod în arbore (0.25 p).
- INSERT_REPARA - reface balansare după inserție (1.25 p)
- MAXIM(NOD *x) / MINIM(NOD *x)- returnează nodul cu cheia maximă / minimă din subarboarele de rădăcină x (0.25 ambele funcții).
- SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p ambele funcții).
- SEARCH(int val) - returnează nodul cu valoarea val dacă există sau NULL altfel. (0.25 p).
- DELETE(NOD *x) - șterge din arbore nodul x (care a fost mai întâi identificat prin SEARCH) (0.5 p).
- DELETE_REPARA(NOD *x) - reface balansarea arborelui după ștergere - (1.25p)
- ROT_ST, ROT_DR - funcțiile de rotație - (0.25 p)
- PRINT_TREE(int opt) - afișază 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 afișări și 0.25 pentru a 4-a). Trebuie afișat și factorul de balansare pentru fiecare nod.
- EMPTY() - șterge toate nodurile din arbore (0.25 p)
- CONSTRUCT - construiește un arbore AVL pornind de la un vector de chei. (0.25 p)
- MERGE - reunește doi arbori AVL într-un al treilea în complexitate cât mai bună. Observație: dacă maximumul din $T1$ este mai mic decât minimumul din $T2$ sau minimumul din $T1$ este mai mare decât maximumul din $T2$ atunci problema poate fi rezolvată în complexitate logaritmică. (3p) Punctajul complet se dă la acest punct dacă există și o argumentare a complexității, atât în cele mai favorabile cazuri cât și în celelalte cazuri.

Structura trebuie să dispună de un constructor care inițializează RADACINA cu NULL. În funcția *main* se declară o variabilă de tip AVL și se folosește un

menu implementat cu ajutorul unei instrucțiuni *switch*, prin care utilizatorul să poată selecta oricare dintre operațiile de inserție, căutare, ștergere, minim, maxim, succesor, predecesor, afișare în cele 4 moduri - la alegere, Join (atunci trebuie doi arbori). (0.25 p)

Observație: Această problemă se poate rezolva prin adaptarea programului de la pb. 1.

3. Implementați un arbore roșu-negru. (pentru template 0.25 p suplimentar). Utilizați o structură NOD care dispune de un câmp informație, un câmp culoare (NU de tip șir de caractere!!!) și câmpuri de tip pointer pentru fiii stâng și drept și pentru părinte. De asemenea structura NOD trebuie să dispună de un constructor care setează câmpul informație cu valoarea transmisă prin parametru, câmpul culoare la roșu și câmpurile de tip pointer în mod adecvat. Utilizați o structură ARN care dispune de un membru de tip pointer la NOD, numit RADACINA și un câmp de tip pointer la NOD numit NIL, care este nodul santinelă. În plus dispune de funcțiile:

- INSERT - inserează un nou nod în arbore (0.25 p)
- INSERT_REPARA - reface proprietățile RN după inserție (0.75 p)
- MAXIM(NOD *x) / MINIM(NOD *x) - returnează nodul cu cheia maximă / minimă din subarborele de rădăcină x (0.25 p ambele funcții)
- SUCCESOR(NOD *x) / PREDECESOR(NOD *x) - returnează nodul care este succesorul / predecesorul nodului x (0.25 p ambele funcții)
- SEARCH(int val) - returnează nodul cu valoarea val dacă există sau NULL altfel. (0.25 p)
- DELETE(NOD *x) - șterge din arbore nodul x (care a fost mai întâi identificat prin SEARCH) (0.5 p)
- DELETE_REPARA(NOD *x) - reface proprietățile RN după ștergere (1 p)
- ROT_ST, ROT_DR - funcțiile de rotație - (0.25 p)
- EMPTY() - șterge toate nodurile din arbore (0.25 p)
- PRINT_TREE(int opt) - afișază 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 afișări și 0.25 pentru a 4-a). Trebuie afișată și culoarea pentru fiecare nod.
- CONSTRUCT - construiește un ARN pornind de la un vector de chei. (0.25p)

Structura trebuie să dispună de un constructor care inițializează RADACINA și santinela în modul prezentat la curs. În funcția *main* se declară o variabilă de tip ARN și se folosește un *menu* implementat cu ajutorul unei instrucțiuni *switch*, prin care utilizatorul să poată selecta oricare dintre operațiile de inserție, căutare, ștergere, minim, maxim, succesor, predecesor, afișare în cele 4 moduri - la alegere. (0.25 p)

4. **Îmbogățirea arborilor binari de căutare:** Implementați un arbore pentru statistici de ordine pe baza unui ARN (pb 3) sau a unui AVL (pb 2), adăugând funcțiile: RANG și SELECT (vezi curs) - fiecare valorează 0.25 p. Pentru refacerea proprietății *size* la inserție, ștergere și rotații - (1.5 p)
5. Implementați 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 față de funcțiile 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 și valoarea 7.) (2 p)
 - Implementarea unui iterator cu care să se poată itera prin structură (3p).
 - Funcția CLEAR - golește structura (0.25 p)
 - Operatorul = pentru copierea structurii (1 p)
 - Funcțiile ISEMPY() și SIZE() (0.25 p)

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

Observații:

- Funcțiile INSERT, SEARCH, DELETE, EMPTY, PRINT_TREE, SUCCESSOR, PREDECESSOR, MINIM, MAXIM, CONSTRUCT, EMPTY, funcțiile de rotație se punctează doar o dată, oricâte dintre probleme au fost rezolvate!
- Pentru cod copiat de pe net nota finala nu poate depăși 4