Laborator 5: Arbori binari de căutare

1 Obiective

Scopul acestui laborator este de a familiariza studenții cu operații cu structuri de date de tip arbori binari de căutare. În lucrare sunt prezentate operațiile importante asupra arborilor binari de căutare: inserare, căutare, ștergere și traversare.

2 Noțiuni teoretice

Arborii binari de căutare, numiți și arbori ordonați sau sortați, sunt structuri de date care permit memorarea şi regăsirea rapidă a unor informaţii, pe baza unei chei. Fiecare nod al arborelui trebuie să conţină o cheie distinctă.

Cheile acestora sunt ordonate și pentru fiecare nod, subarborele stâng conține valori mai mici decât cea a nodului, iar cel drept conține valori mai mari decât cea a nodului. Cheile sortate permit folosirea unor algoritmi eficienți de căutare: traversând de la rădăcină la frunze, se realizează compararea valorilor cheilor memorate în noduri și se decide pe baza acestei comparații dacă căutarea va continua în subarborele drept sau subarborele stâng. În cazul arborilor echilibrați, la fiecare comparație se elimină aproximativ jumatate din nodurile rămase, așadar operația de căutare devine eficientă (de complexitate logaritmică).

Structura arborilor binari de căutare

Proprietatea arborilor binari de căutare: *Fie x un nod într-un arbore binar de căutare. Dacă y este un nod în subarborele stâng, atunci $y.cheie < x.cheie$. Dacă y este un nod în subarborele drept, atunci $y.cheie > x.cheie$.*

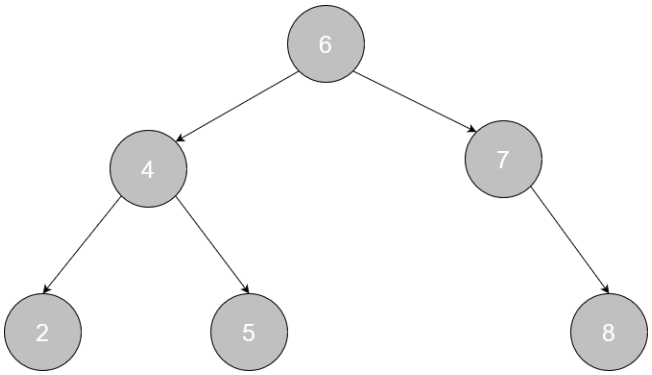


Figura 1: Arbore binar de căutare

Un arbore binar de căutare este organizat după cum îi spune numele într-un arbore binar. Acesta poate fi reprezentat printr-o structură de date înlănțuită, unde fiecare nod are o cheie și conține atributele stânga, dreapta. Acestea reprezintă pointeri către nodul fiului stâng, respectiv către nodul fiului drept. Daca un copil al unui nod lipsește, atunci atributul pentru acel copil va fi NULL.

Structura unui nod al unui arbore binar de căutare poate fi:

```
typedef struct _TREE_NODE
{
    int key;
    struct _TREE_NODE *left;
    struct _TREE_NODE *right;
} TREE_NODE;
```

Rădăcina arborelui poate fi declarată așa:

```
TREE_NODE *root = NULL;
```

Inserarea unui nod într-un arbore binar de căutare

Construcția unui arbore binar de căutare se realizează prin inserarea unor noduri noi în arbore. O nouă cheie este întotdeauna introdusă la nivelul frunzei. Se începe căutarea locului cheii începând de la rădăcină către frunze. Atunci când locul noului nod este găsit, noul nod se introduce ca fiu al unui nod frunză.

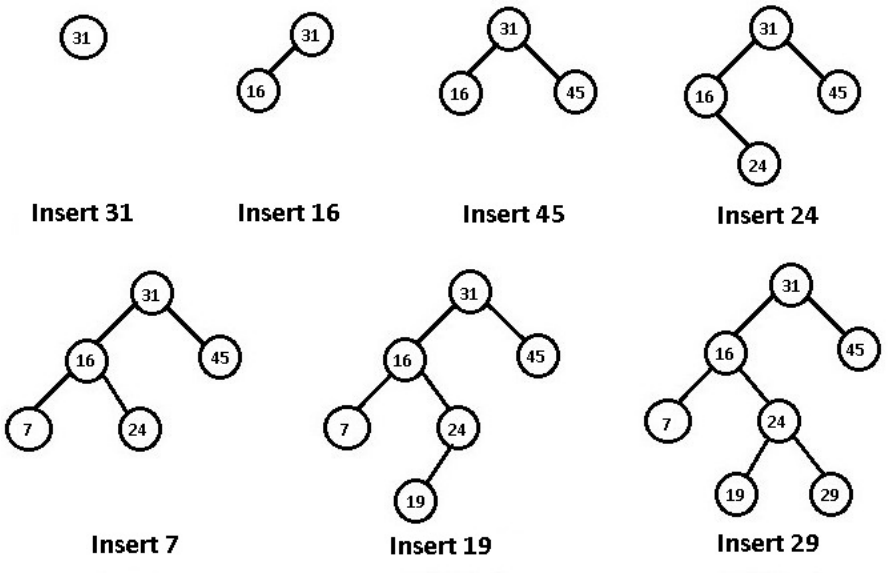


Figura 2: Inserarea nodurilor într-un arbore binar de căutare

Operația de inserare se poate realiza urmând pașii următori:

1. Dacă arborele este vid, se creează un nou nod care este rădăcina, cheia având valoarea *key*, iar subarborii stâng şi drept fiind vizi (pointeri NULL catre nodul copil stâng și cel drept). Se termină operația.
2. Dacă rădăcina nu este vidă atunci se caută poziția corectă a noului nod mergând pe legătura *left* dacă *key* este mai mic ca cheia nodului curent și respectiv pe *right* dacă *key* este mai mare ca cheia nodului curent. Pe parcurs se reține nodul părinte, adică ultimul nod nenul pe drum.
3. Se crează un nod nou cu cheia *key* și se schimbă copilul stâng sau drept al nodului părinte la noul nod.

Căutarea unui nod după cheie într-un arbore binar de căutare

Căutarea într-un arbore binar de căutare a unui nod de cheie dată se face după un algoritm asemănător cu cel de inserare. Cunoscând nodul rădăcină, se traversează arborele de la rădăcină către frunze și se compară cheile din arbore cu cel căutat. Ținând cont de proprietatea arborelui binar, se direcționează căutarea către

subarborele drept sau stâng.

Algoritmul recursiv de căutare este redat prin funcţia următoare:

```
TREE_NODE* searchKeyRec(TREE_NODE* root, int key)
{
    /* Arborele este vid sau cheia cautata key este in radacina, atunci returnam radacina */
    if (root == NULL || root->key == key)
        return root;
    /*   Daca cheia key este mai mare decat cheia radacinii, se reia algoritmul pentru
        subarborele drept */
    if (root->key < key)
        return searchKeyRec(root->right, key);

    /*  Daca cheia key este mai mica decat cheia radacinii, se reia algoritmul pentru subarborele
         stang */
    return searchKeyRec(root->left, key);
}
```

Ştergerea unui nod după cheie într-un arbore binar de căutare

În cazul ştergerii unui nod, arborele trebuie să-şi păstreze structura de arbore de căutare. La ştergerea unui nod de cheie dată intervin următoarele cazuri:

1. Nodul de şters este un nod frunză. În acest caz, în nodul tată, adresa nodului fiu de şters (stâng sau drept) devine NULL.
2. Nodul de şters este un nod cu un singur descendent. În acest caz, în nodul tată, adresa nodului fiu de şters se înlocuieşte cu adresa descendentului nodului fiu de şters.
3. Nodul de şters este un nod cu doi descendenţi. În acest caz, nodul de şters se poate inlocui fie cu:
   - predecesorul sau care este nodul cel mai din dreapta al subarborelui stâng (maximul din subarborele stang).
   - succesorul sau care este nodul cel mai din stânga al subarborelui drept (minimul din subarborele drept).

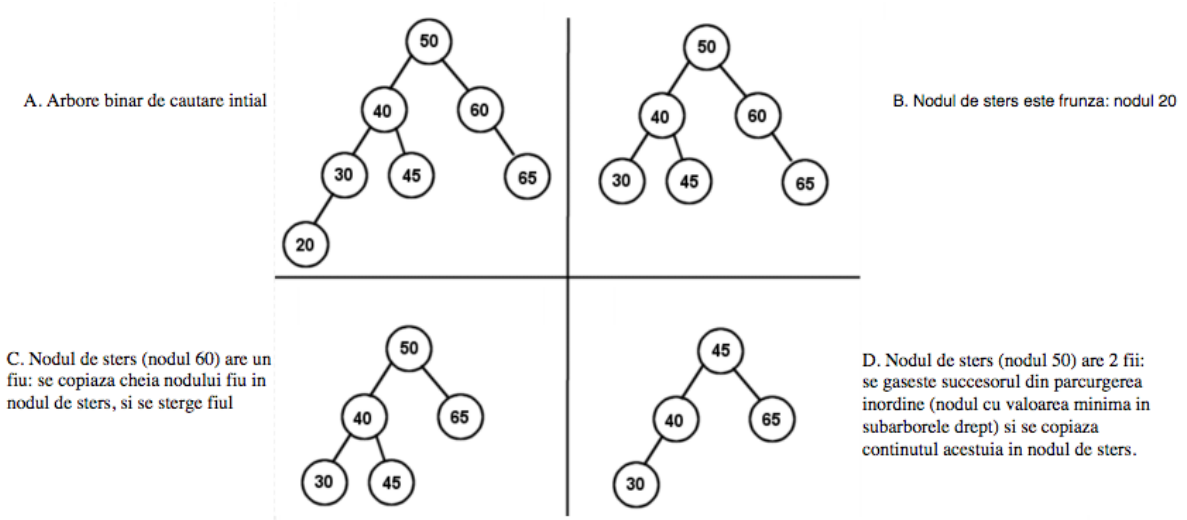
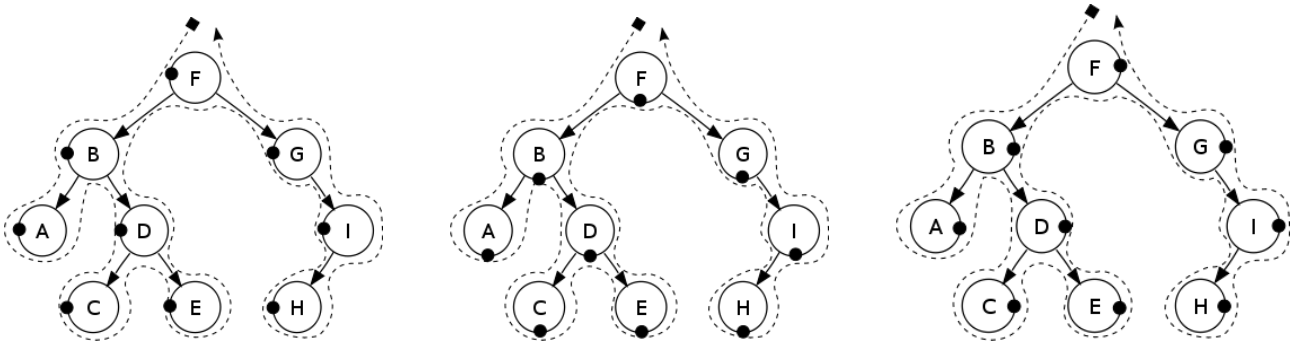


Figura 3: Ștergerea unui nod dintr-un arbore binar de căutare



(a) Preordine: F, B, A, D, C, E, G, I, H

(b) Inordine: A, B, C, D, E, F, G, H, I

(c) Postordine: A, C, E, D, B, H, I, G, F

Figura 4: Traversarea arborilor binari de căutare

Algoritmul de ştergere a unui nod presupune etapele:

1. se caută nodul q care are cheia *key* şi nodul p care este nodul tată corespunzător și se reține dacă nodul căutat este fiu drept sau fiu stâng al lui;
2. dacă nu se găsește cheia se termină funcția;
3. dacă nodul q nu are niciun fiu, se setează $tocopy = NULL$;
4. dacă nodul q are un singur fiu, se atribuie acest fiu la *tocopy*;
5. dacă nodul q are doi fii, atunci se caută succesorul, se copiază cheia lui în q, se șterge succesorul și se termină funcția;
6. dacă există nodul p tată, trebuie înlocuit fiul lui drept sau stâng să fie nodul *tocopy* și apoi trebuie dezalocat nodul q;
7. dacă nu există nodul p tată, înseamnă că trebuie schimbat pointerul de rădăcină să fie *tocopy*, care posibil este *NULL*.

Traversarea unui arbore binar de căutare

Ca orice arbore binar, un arbore binar de căutare poate fi traversat în cele trei moduri, vezi Figura 4:

- preordine: rădăcina, stânga, dreapta;
- inordine: stânga, rădăcina, dreapta;
- postordine: stânga, dreapta, rădăcina.

3 Mersul lucrării

Probleme obligatorii

Să se scrie un program care implementează următoarele operații pentru o structura de tip arbore binar de căutare. Nodurile din arbore memorează numere întregi.

Ex. 1 — Inserarea în arbore a unui nod cu cheia key. Folosiți o metodă iterativă. Rădăcina se trimite ca dublu pointer fiindcă ea se schimbă la prima inserare.

```
void insertKey(TREE_NODE** root, int key)
```

Inserați următoarele numere 15, 6, 18, 17, 20, 3, 7, 2, 4, 13, 9 pentru a obține arborele din figura 5.

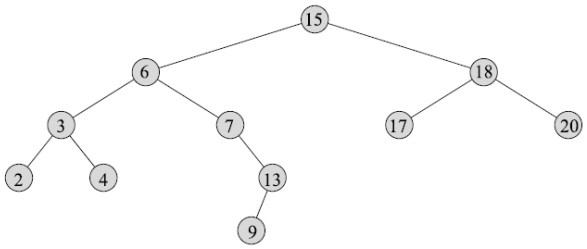


Figura 5: Arbore Binar de Cautare 2

Ex. 2 — Căutarea unui nod cu cheia key în arbore:

```
TREE_NODE* searchKey(TREE_NODE* root, int key)
```

Funcția returnează adresa nodului cu cheia key sau NULL dacă această cheie nu este în arbore.

Ex. 3 — Traversările unui arbore:

```
void preOrder(TREE_NODE* root)
void inOrder(TREE_NODE* root)
void postOrder(TREE_NODE* root)
```

Ex. 4 — Găsirea minimului din subarborele care are ca rădăcină un nod dat *root*:

```
TREE_NODE* findMin(TREE_NODE* root)
```

Funcția returnează adresa nodului cu valoarea minimă din subarborele care are ca rădăcină pe *root*.

De exemplu, pentru arborele din figura 5 minimul din subarborele care are ca rădăcină nodul 6 este 2, minimul pentru subarborele care are ca rădăcină nodul 15 este 2, minimul pentru subarborele care are ca rădăcină nodul 7 este 7.

Ex. 5 — Găsirea maximului din subarborele care are ca rădăcină un nod dat *root*:

```
TREE_NODE* findMax(TREE_NODE* root)
```

Funcția returnează adresa nodului cu valoarea maximă din subarborele care are ca rădăcină pe *root*.

De exemplu, maximul din subarborele care are ca rădăcină nodul 6 este 13, maximul din subarborele care are ca rădăcină nodul 15 este 20, maximul din subarborele care are ca rădăcină nodul 7 este 13.

Ex. 6 — Găsirea succesorului unui nod:

```
TREE_NODE* succesor(TREE_NODE* root, TREE_NODE* node)
```

Cunoscând un nod dintr-un arbore binar de cautare, este important câteodata să putem determina succesorul său în ordinea de sortare determinată de traversarea în inordine a arborelui. Dacă toate cheile sunt distincte, succesorul unui nod *node* este nodul având cea mai mica cheie mai mare decât $node->key$. Funcția returnează succesorul nodului *node* în arborele cu rădăcina *root*.
La implementare este necesar să se trateze două cazuri:

- Dacă subarborele drept al lui *node* nu este vid atunci succesorul nodului este cel mai din stânga nod din subarborele drept determinat cu $findMin(node->right)$.
- Dacă subarborele drept al lui *node* este vid, atunci succesorul este cel mai de jos nod cu cheie mai mare ca $node->key$ pe drumul de la rădăcină pâna la *node*. Este posibil să nu existe un astfel de nod, cazul în care nu există succesor.

Pentru verificare, să apelați în mod succesiv funcția de succesor pentru a parcurge nodurile din arbore în ordine crescătoare a cheii.

Pentru arborele din figura 5 succesorul nodului având cheia 15 este nodul având cheia 17, deoarece aceasta cheie este cheia minima din subarborele drept al nodului având cheia 15. Nodul având cheia 13 nu are subarbore drept, prin urmare succesorul sau este cel mai de jos nod stramos al sau al carui fiu stâng este de asemenea stramos pentru 13. În cazul nodului având cheia 13, succesorul este nodul având cheia 15.

Ex. 7 — Găsirea predecesorului unui nod:

```
TREE_NODE* predecesor(TREE_NODE* root, TREE_NODE* node)
```

Funcția returnează predecesorul nodului *node* din arborele cu rădăcina *root*.
La implementare este necesar să se trateze două cazuri:

- Dacă subarborele stâng al nodului nu este vid atunci predecesorul nodului este cel mai din dreapta nod din subarborele stâng determinat cu $findMax(node->left)$.
- Dacă subarborele stâng al nodului este vid, atunci predecesorul este cel mai de jos nod cu cheie mai mică ca $node->key$ pe drumul de la rădăcină pâna la *node*. Este posibil să nu existe un astfel de nod, cazul în care nu există predecesor.

In figura 6 predecesorul nodului cu valoarea 60 este nodul cu cheia 30.

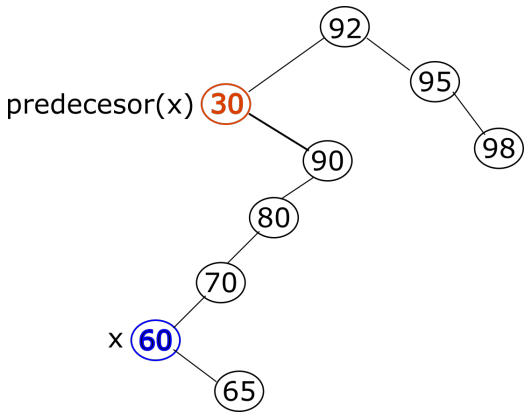


Figura 6: Predecesor pentru cazul când subarborle stâng al nodului este vid.

Ex. 8 — Ștergerea din arbore a unui nod care are cheia *key*:

```
void deleteKey(TREE_NODE** root, int key)
```

Rădăcina se trimite ca dublu pointer pentru că în unele cazuri ea se modifică în urma ștergerii.

Testați funcția prin ștergerea tuturor nodurilor.

3.1 Probleme aplicative

Ex. 9 — Sa se verifice daca doi arbori binari de cautare sunt in oglinda. Pentru ca doi arbori a si b sa fie in oglinda este nevoie sa fie satisfacute urmatoarele conditii:

- Radacina lor sa fie aceeasi.
- Subarborele stang al radacinii arborelui a si subarborele drept al radacinii arborelui b sunt in oglinda.
- Subarborele drept al radacinii arborelui a si subarborele stang al radacinii arborelui b sunt in oglinda.

Figura 7 arata doi arbori in orglinda.

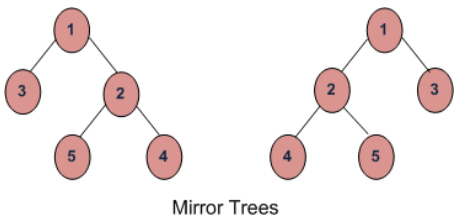


Figura 7: Arbori in oglinda

Ex. 10 — Să se implementeze operaţia de interclasare a doi arbori binari de căutare. Se va impune conditia ca inaltimea arborelui rezultat sa fie mai mica sau egala cu suma inaltimilor arborilor initiali.

Ex. 11 — Să se implementeze operația de găsire a unui drum dintre un nod într-un arbore binar de căutare către alt nod.

Ex. 12 — Se citesc dintr-un fisier n cuvinte. Sa se formeze un arbore binar de cautare din cele n cuvinte citite. Afisati arborele in preordine. Cititi un cuvant cuv de la tastatura. Daca acest cuvant este in arbore stergeti cuvantul si afisati arborele rezultat.

Ex. 13 — Se citesc n numere reale, de exemplu $9.23, 8.35, 10.28, -98.01, 3.14, 2.19$. Sa se formeze un arbore binar de cautare din numerele citite. Afisati arborele in inordine. Din arborele creat formati un arbore binar de cautare nou care sa contina partea intreaga a numerelor reale, in cazul de fata va contine $9, 8, 10, -98, 3, 2$. Afisati arborele in inordine. Atentie
Din arborele creat initial formati un arbore binar de cautare nou care sa contina partea zecimala a numerelor reale, in cazul de fata va contine $0.23, 0.35, 0.28, -0.01, 0.14, 0.19$. Afisati arborele in inordine.

Ex. 14 — Se dă un arbore binar de căutare și o valoare x. Să se găsească doua noduri în arborele binar de căutare ale căror sumă este egala cu x. Nu se permite modificarea arborelui binar de căutare.

Ex. 15 — Se da un vector ordonat de chei, sa se genereze **arborele binar de cautare perfect echilibrat** folosind cheile din vector. Un arbore binar de cautare este perfect echilibrat daca diferenta intre numarul de noduri din subarborele stang si drept este cel mult 1, la orice nod din arbore.
Ideea echilibrarii arborelui de cautare se datoreaza lui Adel'son-Vel'skii si Landis, care au introdus o clasa de arbori de cautare echilibrati numiti "arbori AVL". În arborii AVL, echilibrarea este mentinuta prin rotatii.

Ex. 16 — Sa se verifice daca un arbore binar este arbore AVL. Realizati operatia de verificare si de calculare a inaltimii simultan. Cat va fi eficienta algoritmului in acest caz?
Care va fi eficienta alrogitmului daca verificarea conditiei AVL si calcularea inaltimii se fac separat ?

Ex. 17 — (*)Fiind dat un arbore binar de cautare care contine numere intregi. Sa se determina daca in arbore exista siruri de k noduri a caror suma este egala cu m. Valorile pentru k si m se citesc de la tastatura. Considerati urmatoarele situatii:

1.$k = 2$ si $m = 0$

2.$k = 3$ si $m = 0$

3.$k = 3$ si $m > 0$

Analizati eficienta in fiecare din cele 3 cazuri.