5 Arbori binari de cautare

5.1 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.

5.2 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 si 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 cum ar fi căutarea binară: 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 medie, cautarea binară permite saltul peste aproximativ jumatate din noduri, asadar operația de căutare devine mai rapidă.

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.*

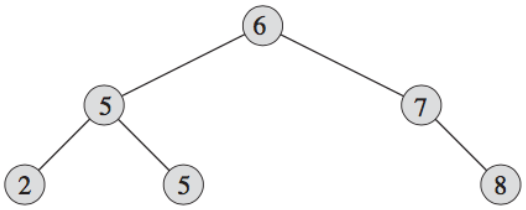


Figure 5.1: Arbore binar de căutare

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 int KeyType; // tipul cheii

typedef struct node {
    KeyType key;
    struct node *left;
    struct node *right;
} NodeT;
```

Rădăcina arborelui poate fi declarată variabilă globală:

```
NodeT *root;
```

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.

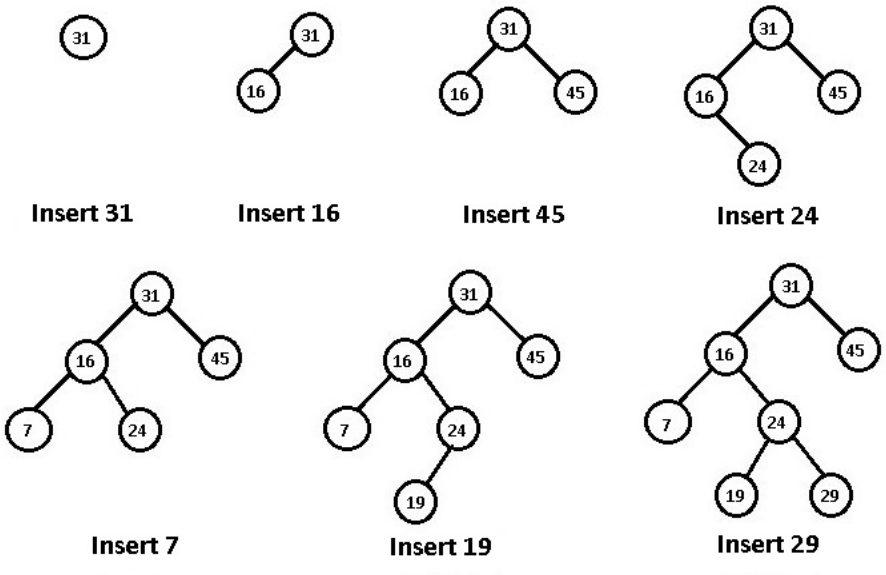


Figure 5.2: Inserarea nodurilor într-un arbore binar de căutare

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)
2. Dacă cheia rădăcinii este egală cu *key* atunci inserarea nu se poate face întrucât există deja un nod cu această cheie.
3. Dacă cheia *key* este mai mică decât cheia rădăcinii, se reia algoritmul pentru subarborele stâng (pasul 1).
4. Dacă cheia *key* este mai mare decât cheia rădăcinii, se reia algoritmul pentru subarborele drept (pasul 1).

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:

```
NodeT* findNodeRec(NodeT* 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 findNodeRec(root->right, key);

    /*  Daca cheia key este mai mica decat cheia radacinii, se reia algoritmul pentru subarborele
        stang */
    return findNodeRec(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 stang (maximul din subarborele stang).
- succesorul sau care este nodul cel mai din stanga al subarborelui drept (minimul din subarborele drept).

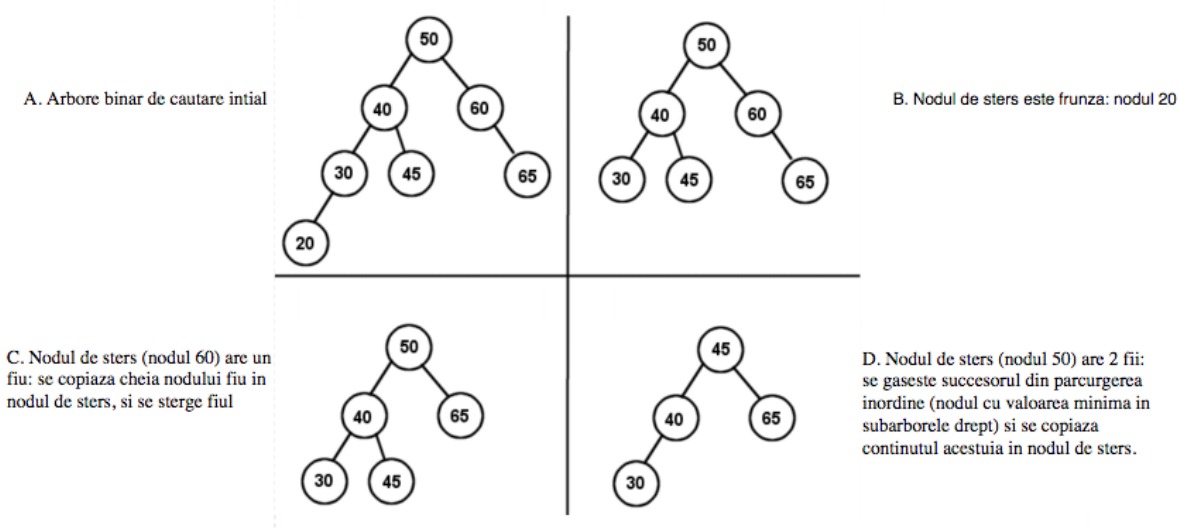


Figure 5.3: Ștergerea unui nod dintr-un arbore binar de căutare

Algoritmul de ştergere a unui nod conţine următoarele etape:

1. căutarea nodului de cheie *key* şi a nodului tată corespunzător.
2. determinarea cazului în care se situează nodul de şters si tratarea fiecarui caz in mod corespunzator.

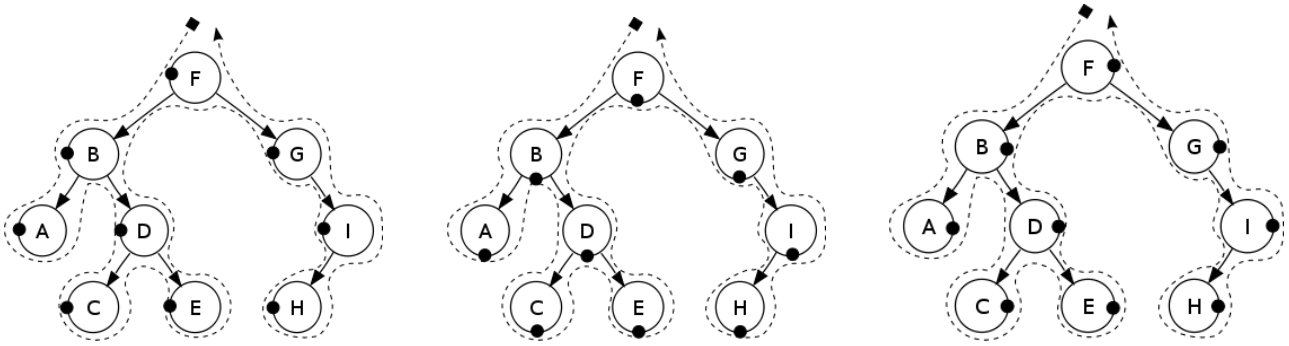
Funcția recursivă de ștergere este:

```
NodeT* delNode(NodeT* root, int key) {
    NodeT *p;
    /* arbore vid */
    if (root == NULL) return root;

    /* Daca cheia key este mai mica decat cheia radacinii, cautarea nodului key
        se face in subarborele stang */
    if (key < root->key)
        root->left = delNode(root->left, key);

    /* Daca cheia key este mai mare decat cheia radacinii, cautarea nodului key
        se face in subarborele drept */
    else if (key > root->key)
        root->right = delNode(root->right, key);
    /* cheia radacinii este egala cu key, acesta este nodul ce trebuie sters */
        else {
     /* Nodul are un singur fiu */
        if (root->left == NULL) {
            p = root->right;
            free(root);
            return p;
        }
        else if (root->right == NULL) {
            p = root->left;
            free(root);
            return p;
        }
        /* Nodul are 2 fii */
        p = findMinNode(root->right);
        root->key = p->key;
        root->right = delNode(root->right, p->key);
    }
    return root;
}
```

Traversarea unui arbore binar de căutare

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

- preordine: radacina, stanga, dreapta;
- inordine: stanga, radacina, dreapta;
- postordine: stanga, dreapta, radacina.



(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

Figure 5.4: Traversarea arborilor binari de căutare

5.3 Mersul lucrarii

Probleme obligatorii

Să se scrie un program care implementeaza urmatoarele operatii pentru o structura de tip arbore binar de căutare. Nodurile din arbore memoreaza numere intregi.

Ex. 1 — Inserarea în arbore a unui nod cu cheia key. Folositi o metoda nerecursiva.

```
NodeT *insertNode( NodeT *root, int key )
```

Inserati urmatoarele numere 15, 6, 18, 20, 17, 7, 13, 3, 2, 4, 9 pentru a obtine arborele din figura 5.5.

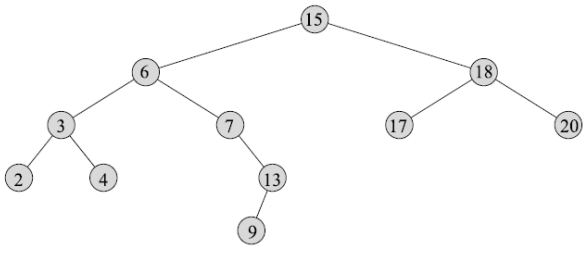


Figure 5.5: Arbore Binar de Cautare 2

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

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

Functia returneaza adresa nodului cu cheia key sau NULL daca aceasta cheie nu este in arbore.

Ex. 3 — Traversarile unui arbore:

```
void inOrder( NodeT *p)
void preOrder( NodeT *p)
void postOrder( NodeT *p)
```

Faceti o functie de afisare frumoasa !

Ex. 4 — Găsirea minimului din subarborele care are ca radacina un nod dat, $node$**.**

```
NodeT* findMin(NodeT* node)
```

Functia returneaza adresa nodului cu valoarea minima din subarborele care are ca radacina pe node.

De exemplu pentru arborele din figura 5.5 minimul din subarborele care are ca radacina nodul 6 este 2, minimul pentru subarborele care are ca radacina nodul 15 este 2, minimul pentru subarborele care are ca radacina nodul 7 este 7.

Ex. 5 — Găsirea maximului din subarborele care are ca radacina un nod dat, $node$**.**

```
NodeT* findMax(NodeT* node)
```

Functia returneaza adresa nodului cu valoarea maxima din subarborele care are ca radacina pe node.

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

Ex. 6 — Găsirea succesorului unui nod. Cunoscând un nod dintr-un arbore binar de cautare, este important câteodata sa putem determina succesorul sau în ordinea de sortare determinata de traversarea în inordine a arborelui. Daca toate cheile sunt distincte, succesorul unui nod $node$ este nodul având cea mai mica cheie mai mare decât $cheie[node]$. Functia returneaza succesorul nodului $node$.

```
NodeT* succesor(NodeT *node)
```

La implementare este necesar sa se trateze doua cazuri:

- Daca subarborele drept al lui node nu este vid atunci succesorul nodului este cel mai din stanga nod din subarborele drept determinat cu $findMin(node->right)$.
- Daca subarborele drept al lui node este vid si $node$ are un succesor y, atunci y este cel mai de jos stramos al lui $node$ al carui fiu din stânga este de asemenea stramos al lui $node$.

Pentru arborele din figura 5.5 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. Functia returneaza predecesorul nodului $node$, adica adresa nodului cu valoarea maxima din subarborele stang al nodului $node$.

```
NodeT* predecesor(NodeT *node)
```

La implementare este necesar sa se trateze doua cazuri:

- Daca subarborele stang al nodului nu este vid atunci predecesorul nodului este cel mai din dreapta nod din subarborele stang determinat cu $findMax(node->left)$.
- Daca subarborele stang al nodului este vid si $node$ are un predecesor, y, atunci y este cel mai de jos stramos al lui $node$ al cariu fiu din dreapta este de asemenea stramos al lui $node$.

In figura 5.6 predecesorul nodului cu valoarea 60 este nodul cu cheia 30 !

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

```
NodeT* deleteNode(NodeT* root, int key)
```

Functia returneaza arborele rezultat dupa stergere.

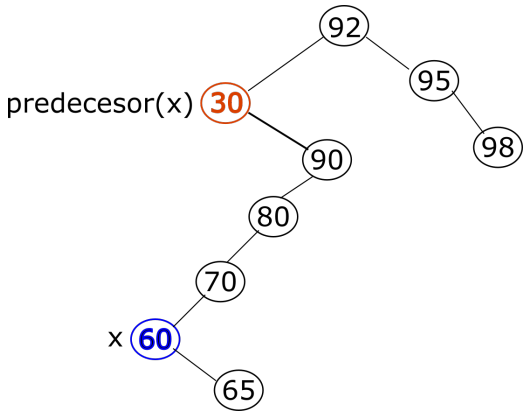


Figure 5.6: Predecesor pentru cazul cand subarborle stang al lui x este vid.

5.3.1 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 5.7 arata doi arbori in orglinda.

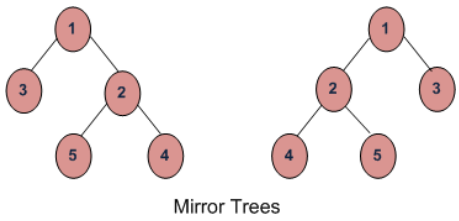


Figure 5.7: Arbori in oglinda

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:

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

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

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

Analizati eficienta in fiecare din cele 3 cazuri.