1 Laborator 5: Arbori binari de cautare

1.1 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.

1.2 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.*

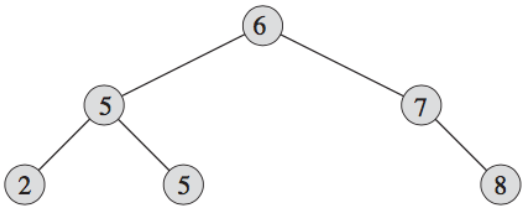


Figure 1.1: Arbore binar de căutare

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.

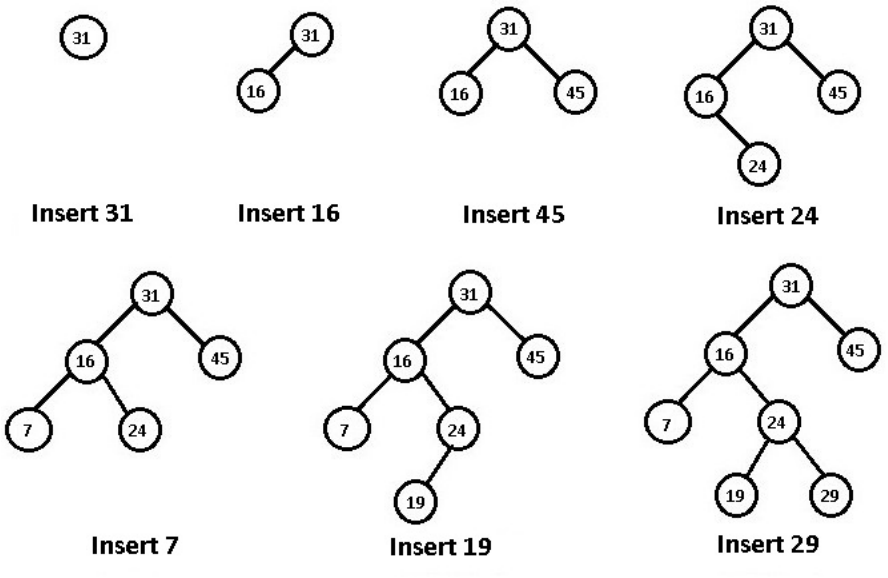


Figure 1.2: Inserarea nodurilor într-un arbore binar de căutare

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).

Algoritmul recursiv de inserare a unui nod într-un arbore binar de căutare:

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

    NodeT *p;

    /* Daca arborele este vid, se creeaza un nou nod care este radacina, cheia avand valoarea key,
        iar subarborii stang si drept fiind vizi (pointeri NULL catre nodul copil stang si cel drept
        ) */

    if ( root == NULL ) {
        p = ( NodeT *) malloc( sizeof( NodeT ));
        p->key = key;
        p->left = p->right = NULL;
        root = p;
    }
    else {
     /*  Daca cheia key este mai mica decat cheia radacinii, se reia algoritmul pentru subarborele
         stang */
    if ( key < root->key )
        root->left = insertNode( root ->left, key );
    else
     /*   Daca cheia key este mai mare decat cheia radacinii, se reia algoritmul pentru subarborele
         drept */
    if ( key > root->key )
        root->right = insertNode( root ->right, key );
    else /* cheia exista deja in arbore, nu se mai insereaza */
        printf( "\nNode of key = %d already exists\n", key );
    }

    return root;
}
```

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 înlocuieşte cu nodul cel mai din stânga al subarborelui drept sau cu nodul cel mai din dreapta al subarborelui stâng.

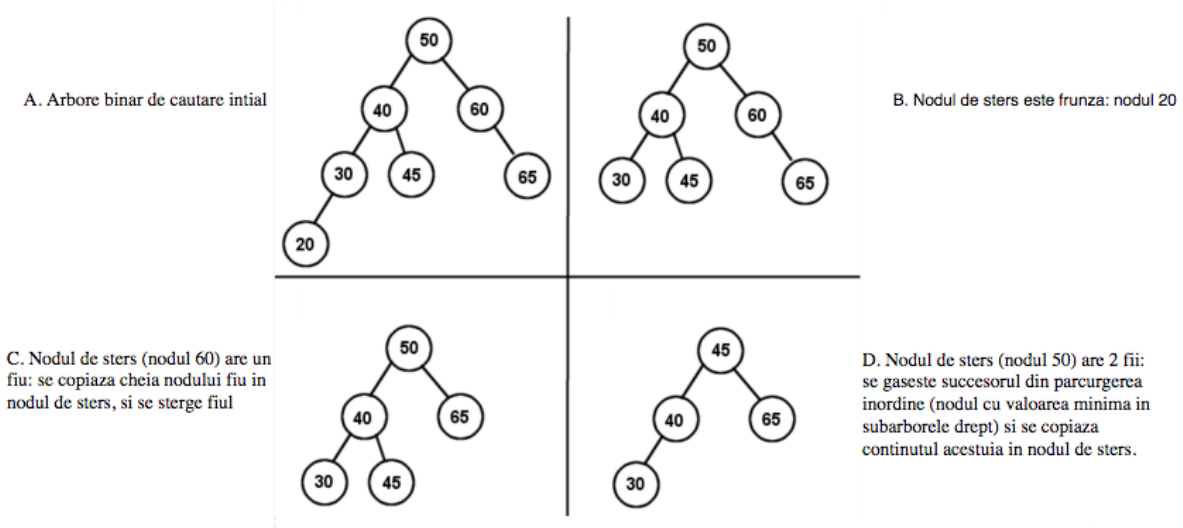


Figure 1.3: Ștergerea unui nod dintr-un arbore binar de căutare

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.

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;
}
```

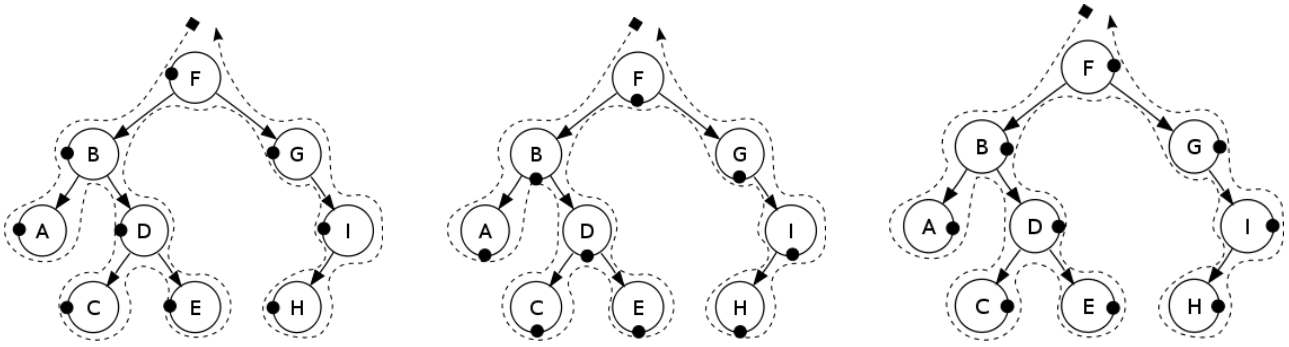
Ștergerea unui arbore binar de căutare

Ştergerea unui arbore binar de căutare constă în parcurgerea în postordine a arborelui şi ştergerea nod cu nod, conform funcţiei recursive următoare:

```
void delTree( NodeT *root )
{
  if ( root != NULL )
  {
    delTree( root ->left );
    delTree( root ->right );
    free( root );
  }
}
```

Traversarea unui arbore binar de căutare

Ca orice arbore binar, un arbore binar de căutare poate fi traversat în cele trei moduri: în preordine, în inordine şi în postordine.



(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 1.4: Traversarea arborilor binari de căutare

Funcțiile recursive de parcurgere a arborilor binari de căutare sunt:

```
void preorder( NodeT *p )
{
```

```
  if ( p != NULL )
  {
    /* code for info retrieval here */
    preorder( p->left );
    preorder( p->right );
  }
}
void inorder( NodeT *p )
{
  if ( p != NULL )
  {
    inorder( p->left );
    /* code for info retrieval here */
    inorder( p->right );
  }
}
void postorder( NodeT *p )
{
  if ( p != NULL )
  {
    postorder( p->left );
    postorder( p->right );
    /* code for info retrieval here */
  }
}
```

1.3 Exemplu de cod

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int key;
    struct node *left;
    struct node *right;
} NodeT;

NodeT* root;


void preOrder( NodeT *p) {

    if ( p != NULL ) {

        printf( "%d\n", p->key );
        preOrder( p->left);
        preOrder( p->right);
    }
}
void inOrder( NodeT *p) {

    if ( p != NULL ) {
        inOrder( p->left);
        printf( "%d\n", p->key );
        inOrder( p->right);
    }
}
void postOrder( NodeT *p) {

    if ( p != NULL ) {
        postOrder( p->left );
        postOrder( p->right);
        printf( "%d\n", p->key );
    }
}

/* recursive version of insert */
NodeT *insertNode( NodeT *root, int key ) {

    NodeT *p;
```

```
    /* Daca arborele este vid, se creeaza un nou nod care este radacina, cheia avand valoarea key,
        iar subarborii stang si drept fiind vizi (pointeri NULL catre nodul copil stang si cel drept
        ) */

    if ( root == NULL ) {
        p = ( NodeT *) malloc( sizeof( NodeT ));
        p->key = key;
        p->left = p->right = NULL;
        root = p;
    }
    else {
     /*  Daca cheia key este mai mica decat cheia radacinii, se reia algoritmul pentru subarborele
         stang */
    if ( key < root->key )
        root->left = insertNode( root ->left, key );
    else
     /*   Daca cheia key este mai mare decat cheia radacinii, se reia algoritmul pentru subarborele
         drept */
    if ( key > root->key )
        root->right = insertNode( root ->right, key );
    else /* cheia exista deja in arbore, nu se mai insereaza */
        printf( "\nNode of key = %d already exists\n", key );
    }

    return root;
}

/* non-recursive function of find Node */

NodeT *findNode( NodeT *root, int key ) {

    NodeT *p;

    if ( root == NULL ) return NULL;
        p = root;
        while ( p != NULL ) {
            if ( p -> key == key )
                return p; /* found */
            else
            if ( key < p->key )
                p = p->left;
            else
                p = p->right;
            }
    return NULL; /* not found */
}

/* recursive function of find Node */

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);
}

/* non-recursive function of finding the node with the minimum value */

NodeT* findMinNode(NodeT* node)
{
    NodeT* p = node;

    while (p->left != NULL)
        p = p->left;

    return p;
```

```
}

/* recursive function of delete Node */

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;
}

/* recursive function of delete Tree */

void delTree( NodeT *root ) {
    if ( root != NULL ) {

        delTree( root->left );
        delTree( root->right );
        free( root );
    }
}


int main() {

    NodeT *p;
    int i, n, key;
    char ch;

    printf( "Number of nodes to insert= " );
    scanf( "%d", &n );
    root = NULL;

    for ( i = 0; i < n; i++ ){
        printf( "\nKey= " );
        scanf( "%d", &key );
        root = insertNode( root, key );
    }

    getchar();
    printf( "\nPreorder listing\n" );
    preOrder( root);
    printf( "Press Enter to continue." );
    while ('\n' != getc(stdin));
    printf( "\nInorder listing\n" );
    inOrder( root );
```

```
    printf( "Press Enter to continue." );
    while ('\n' != getc(stdin));
    printf( "\nPostorder listing\n" );
    postOrder( root);
    printf( "Press Enter to continue." );
    while ('\n' != getc(stdin));


    printf( "Continue with find (Y/N)? " );
    scanf( "%c", &ch );
    getchar();
    while ( ch == 'Y' || ch == 'y' ) {

        printf( "Key to find= " );
        scanf( "%d", &key );
        p = findNodeRec( root, key );
        if ( p != NULL )
            printf( "Node found\n" );
        else
            printf( "Node NOT found\n" );
        while ('\n' != getc(stdin));
        printf( "Continue with find (Y/N)? " );
        scanf( "%c", &ch );
        getchar();
    }

    printf( "Continue with delete (Y/N)? " );
    scanf( "%c", &ch );
    getchar();
    while ( ch == 'Y' || ch == 'y' ) {
        printf( "Key of node to delete= " );
        scanf( "%d", &key );
        root = delNode( root, key );
        inOrder( root);
        while ('\n' != getc(stdin));
        printf( "Continue with delete (Y/N)? " );
        scanf( "%c", &ch );
        getchar();
    }

    printf( "Delete the whole tree (Y/N)? " );
    scanf( "%c", &ch );
    getchar();
    if ( ch == 'Y' || ch == 'y' ) {
        delTree( root );
        root = NULL;
        printf(" Tree completely removed\n" );
    }

    printf( "Press Enter to exit program. \n" );
    while ('\n' != getc(stdin));


    return 0;

}
```

1.4 Mersul lucrarii

1.4.1 Probleme obligatorii

1.4.1. Să se scrie un program care ilustrează operații asupra unui arbore binar de căutare, care memorează litere mari ale alfabetului. Operațiile care trebuie sa fie implementate sunt:

- inserarea unui nod în arbore
- ștergerea unui nod în arbore
- găsirea unui nod în arbore
- traversarea inordine
- găsirea minimului
- găsirea maximului
- găsirea predecesorului unui nod în traversarea inordine
- găsirea succesorului unui nod în traversarea inordine.

1.4.2. Informaţiile pentru medicamentele unei farmacii sunt: nume medicament, preţ, cantitate, data primirii, data expirării. Evidenţa medicamentelor se ţine cu un program care are drept structură de date un arbore de căutare după nume medicament. Să se scrie programul care execută următoarele operaţii:

- creează arborele de căutare
- caută un nod după câmpul nume medicament şi actualizează câmpurile de informaţie;
- tipăreşte medicamentele în ordine lexicografică;
- elimină un nod identificat prin nume medicament;
- creează un arbore de căutare cu medicamentele care au data de expirare mai veche decât o dată specificată de la terminal.

1.4.2 Probleme opționale

1.4.3. Să se implementeze operaţia de interclasare a doi arbori de căutare.
1.4.4. Să se implementeze operația de determinare a adâncimii unui arbore binar de căutare.
1.4.5. Să se implementeze operația de găsire a unui drum dintre un nod într-un arbore binar de căutare către alt nod.
1.4.6. 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.