Laborator 09 - Arbori Binari de Cautare

Responsabili:

- Octavian Rinciog (2014)
- Mihai Neacșu (2014)

Obiective

În urma parcurgerii laboratorului, studentul va fi capabil să:

- să înţeleagă structura şi proprietăţile unui arbore binar de căutare
- să construiască, în limbajul C++, un arbore binar de căutare
- să realizeze o parcurgere a structurii de date prin mai multe moduri
- să realizeze diferite operaţii folosind arborii binari de căutare

Noțiuni teoretice

Un arbore binar de căutare este un arbore binar care are în plus urmatoarele proprietăți:

- cheile stocate în noduri (informația utilă) aparțin unei mulțimi peste care există o relație de ordine
- cheia din oricare nod este mai mare decât cheile tuturor nodurilor din subarborele stâng
- cheia din oricare nod este mai mică decât cheile nodurilor ce compun subarborele drept

Arborii binari de căutare permit menţinerea datelor în ordine şi o căutare rapidă a unei chei, ceea ce îi recomandă pentru implementarea de mulţimi şi dicţionare ordonate.

O importantă carcteristica a arborilor de căutare, este aceea că parcurgerea în inordine produce o secventa ordonată crescător a cheilor din nodurile arborelui.

Valoarea maximă dintr-un arbore binar de căutare se află în **nodul din extremitatea dreaptă** şi se determină prin coborârea pe subarborele drept, iar **valoarea minimă** se află în **nodul din extremitatea stângă**.

Căutarea unei chei într-un arbore binar de căutare este asemănătoare căutării binare: cheia căutată este comparată cu cheia din nodul curent (iniţial nodul rădăcină). In funcţie de rezultatul comparaţiei apar trei cazuri:

- acestea coincid – elementul a fost găsit
- elementul căutat este mai mic decât cheia din nodul curent – atunci căutarea continuă în subarborele stâng
- elementul căutat este mai mare decât cheia din nodul curent - atunci căutarea continuă în subarborele drept

Inserarea unui nod se face, în funcţie de rezultatul comparaţiei cheilor, în subarborele stâng sau drept. Dacă arborele este vid, se creează un nod care devine nodul rădăcină al arborelui. În caz contrar, cheia se inserează ca fiu stâng sau fiu drept al unui nod din arbore.

Ștergerea unui nod este o operaţie mai complicată, întrucât presupune o rearanjare a nodurilor. Pentru eliminarea unui nod dintr-un arbore binar de căutare sunt posibile următoare cazuri:

- nodul de şters nu există, operaţia se consideră încheiată
- nodul de şters nu are succesori (este o frunză)
- nodul de şters are un singur successor
- nodul de şters are doi succesori

În cazul ştergerii unui nod frunză sau a unui nod având un singur successor,

Search

- Reguli generale și de notare
- Catalog
- Concursuri
- Calendar

Laboratoare

- Laborator 1 - Introducere in C++
- Laborator 2 - Noțiuni de C++
- Laborator 3 - Stive
- Laborator 4 - Cozi
- Laborator 5 - Liste generice
- Laborator 6 - HashTable
- Laborator 7 - Grafuri
- Laborator 8 - Arbori Binari
- Laborator 9 - Arbori Binari de Căutare
- Laborator 10 - Heap-uri
- Laborator 11 - Treap-uri
- Laborator 12 - Mulțimi Disjuncte

Teme

- Tema 1
- Tema 2
- Tema 3
- Tema 4

Resurse

- Debugging
- Data Structure Visualization

Table of Contents

- Laborator 09 - Arbori Binari de Cautare
  - Obiective
  - Noțiuni teoretice
  - Exemplu
  - Exercitii

legătura de la parintele nodului de şters este înlocuită prin legătura nodului de şters la succesorul său (care poate fi NULL).

Eliminarea unui nod cu doi succesori se face prin înlocuirea sa cu nodul care are cea mai apropiată valoare de nodul şters. Acesta poate fi nodul din extremitatea dreaptă a subarborelui stâng sau nodul din extremitatea stânga a subarborelui drept (este fie predecesorul, fie succesorul în ordine infixată). Acest nod are cel mult un successor.

Complexitatea operaţiilor (căutare, inserare, ştergere) într-un arbore binar de căutare este - *pe cazul mediu* - **O(logn)**.

Exemplu

BinarySearchTree.h

```
#ifndef __BINARY_SEARCH_TREE__H
#define __BINARY_SEARCH_TREE__H

template <typename T>
class BinarySearchTree
{
public:
    BinarySearchTree();
    ~BinarySearchTree();

    void insertKey(T x);
    void removeKey(T x);
    BinarySearchTree<T>* searchKey(T x);
    void inOrderDisplay();

private:
    BinarySearchTree<T> *leftNode;
    BinarySearchTree<T> *rightNode;
    BinarySearchTree<T> *parent;
    T *pData;
};

#endif // __BINARY_SEARCH_TREE_H
```

Exercitii

Acest laborator se va realiza pornind de la **lab09-tasks.zip**. Observații privind scheletul de cod:

- Scheletul citește N numere dintr-un fișier dat ca parametru în linia de comandă.
- Aceste N numere sunt introduse într-un arbore binar de căutare, funcționalitate pe care voi trebuie sa o codați
- Clasa 'BinarySearchTree' conține un membru de tip pointer către T, pentru a defini dacă nodul rădăcină este gol sau nu. Astfel putem verifica dacă un BinarySearchTree conține minim un element (vezi funcția `isEmpty`)
- Funcția 'removeKey' întoarce adresa noului nod rădăcină, dacă s-a șters vechea rădăcină.

1. [**3p**] Implementați următoarele funcționalități de bază ale unui arbore binar de căutare:
   a. [**0.5p**] constructor (TODO 1.1).
   b. [**0.5p**] destructor (TODO 1.2). Eliberați toată memoria alocată.
   c. [**0.5p**] adăugare elemente în arbore(TODO 1.3)
   d. [**0.5p**] căutare elemente în arbore (TODO 1.4)
   e. [**1p**] parcurgere inordine arbore (TODO 1.5)
2. [**2p**] Implementați următoarele funcționalități avansate ale unui arbore binar de căutare:
   a. [**0.5p**] funcții pentru returnare valoare minimă/maximă din arbore. Implementaţi eficient, ţinând cont de faptul ca arborele binar este unul de căutare (TODO 2.1).

```
T findMin ();
```

```
T findMax ();
```

b. [**0.5p**] calculeaza si returnează înalțimea unui arbore. Înălțimea unui arbore se calculează adunând 1 la înălțimea maximă a subarborelui său stâng și a celui drept. (TODO 2.2).

```
int findLevels();
```

c. [**1p**] funcție pentru afișarea cheilor (informațiilor utile) din nodurile situate pe un anumit nivel primit ca parametru (nivel = distanța de la un nod la rădăcina; nivelul rădăcinii este 0)(TODO 2.3).

```
void displayLevel(int level);
```

3. [**3p**] Implementați funcția de ștergere a unui element. **Atenție**. Trebuie să tratați și cazul în care se va șterge elementul din rădăcină. (TODO 3.1)