Estructura de Dades: Pràctica 3

Pràctica 3. Estructures de dades no lineals: Arbres binaris de cerca i Taules de Hash

Introducció

Objectius: Familiaritzar-se amb les estructures no lineals, concretament, amb els arbres binaris de cerca i les estructures de Hash, analitzant les seves característiques i diferències.
Temes de teoria relacionats amb la pràctica: Tema 4 Estructures no lineals: Arbres, Tema 6 Estructures de Hash

Enunciat

Continuant amb l'enunciat de la pràctica 2 i les bases de dades de pel·lícules, en aquesta pràctica farem servir dues estructures de dades no lineals per a fer cerques. En primer lloc programarem un arbre binari de cerca (BST) i una classe de nodes per a l'arbre, que posteriorment farem servir per organitzar pel·lícules per director a l'exercici 2. A l'exercici 3 programareu una Taula de Hash oberta, que posteriorment fareu servir a l'exercici 4. Finalment, a l'exercici 5 compararu les estructures i analitzareu el cost computacional de cadascuna d'elles.

Us recordem l'estructura de la base de dades de pel·lícules:

```
peliId|directorId|titol|durada|valoracio
838|1|American Graffiti|110|6.9
25757|2|Comic Book: The Movie|106|6
12|7|Finding Nemo|100|7.6
10681|7|WALL¬∑E|98|7.8
49529|7|John Carter|132|6.1
376549|21905|The Strange House|87|5.5
24631|68338|"Maria, ihm schmeckt's icht!"|92|5.5
274415|68338|The Pasta Detectives|95|6.2
25623|132305|House|88|7.1
129689|132305|"I Are You, You Am Me"|113|5
178736|132305|Chizuko's Younger Sister|155|0
```

Exercicis

Consideracions prèvies importants per a la realització dels exercicis:

- El **codi s'ha de comentar obligatòriament**:
  a) Als fitxers de declaració (.h), hi haureu d'especificar per cada operació del TAD que implementareu el cost computacional teòric en el pitjor dels casos. També heu d'incloure comentaris a les parts més importants de les funcions que implementeu per facilitar la lectura del codi.
  b) Als fitxers d'implementació (.cpp), hi haureu d'incloure els comentaris necessaris per facilitar la lectura del codi.
- **Controleu errors** quan ho considereu oportú, llençant les degudes excepcions i caçant-les a les seves corresponents clàusules try-catch.

Estructura de Dades: Pràctica 3

- La **implementació amb templates NO és opcional, és obligatori en aquesta pràctica**.
- Podeu implementar mètodes de suport addicionals, sempre que siguin necessaris pel desenvolupament de la pràctica, tot i justificant al codi el seu ús i el seu cost computacional teòric.

1. Arbre binari de cerca

Implementareu, el TAD **`BSTtree`** que representi l'arbre binari de cerca amb nodes encadenats. Els nodes de l'arbre seran cadascun un TAD **`NODEtree`**, que també haureu d'implementar i que representarà una clau i una llista d'elements. A continuació es presenten les definicions de les especificacions de cada classe. Més endavant teniu la descripció dels passos per implementar els TADs i les explicacions del que ha de fer cadascuna de les operacions de cada TAD:

BSTtree.h

```
template <class Key, class Value>
class BSTtree {
    public:
        BSTtree();
        BSTtree(const BSTtree<Key, Value>& orig);
        virtual ~BSTtree();

        bool empty() const;
        int size() const;
        int height() const;

        NODEtree<Key,Value>* insert(const Key& k, const Value& value);
        const vector<Value>& valuesOf(const Key& k) const;
        void printPreorder(const NODEtree<Key,Value>* n = nullptr) const;
        void printInorder(const NODEtree<Key,Value>* n = nullptr) const;
        void printPostorder(const NODEtree<Key,Value>* n = nullptr) const;
        void printSecondLargestKey() const;
        void mirrorTree();
        list<NODEtree<Key, Value>*> getLeafNodes() const;

    protected:
        NODEtree<Key,Value>* root;
        NODEtree<Key,Value>* search(const Key& k) const;

    private:
        int _size;
        /* Mètodes auxiliars definiu aquí els que necessiteu */
};
```

NODEtree.h

```
template <class Key, class Value>
class NODEtree {
    public:
        NODEtree(const Key& key);
        NODEtree(const NODEtree<Key,Value>& orig);
        virtual ~NODEtree();
```

```
        /* Modificadors */
        // Declareu-hi aquí els modificadors (setters) dels atributs que manquen

        /* Consultors */
        const Key& getKey() const;
         const vector<Value>& getValue() const;
        // Declareu-hi aquí els consultors (getters) dels atributs que manquen

        /* Operacions */
        bool isRoot() const;
        bool hasLeft() const;
        bool hasRight() const;
        bool isExternal() const;

        void insertValue(const V& v);
        int depth() const;
        int height() const;
        bool operator==(const BSTNode<K,V>& node) const;

    private:
         Key key;
        vector<Value> values;
        // Afegiu-hi aquí els atributs que manquen
};
```

<u>Pas 1</u> Implementar el node de l'arbre binari de cerca

A continuació teniu la descripció de les operacions del TAD **`NODEtree`** (sense tenir en compte els consultors i modificadors dels atributs que falten per especificar):

Operació	Definició
`constructor`	Construeix un nou node de l'arbre binari, passant com a paràmetre una clau
`constructor còpia`	Construeix una còpia del node partir del node original rebut per paràmetre
`destructor`	Destrueix els nodes fills. *Atenció a la gestió de memòria dinàmica*
`getKey`	Retorna la clau del node (atribut `Key`)
`getValues`	Retorna el vector de valors (atribut `Values`)
`isRoot`	Retorna cert si el node és el root de l'arbre binari de cerca, fals altrament
`hasLeft`	Retorna cert si el node té un fill esquerre, fals altrament
`hasRight`	Retorna cert si el node té un fill dret, fals altrament
`isExternal`	Retorna cert si el node és una fulla de l'arbre binari de cerca, fals altrament
`insertValue`	Afegeix un valor al vector de valors.
`depth`	Retorna la profunditat del node en l'arbre binari de cerca. Convindrem que el root té sempre profunditat 0. **Aquesta funció s'ha d'implementar de forma RECURSIVA**

`height`	Retorna l'alçada del node en l'arbre de cerca binari. Convindrem que les fulles sempre tenen alçada 1. **Aquesta funció s'ha d'implementar de forma RECURSIVA**
`operator==`	(Sobrecarrega de l'operador d'igualtat) Retorna cert si dos nodes són iguals: tenen la mateixa clau i els mateixos valors

<u>Pas 2</u> Implementar l'arbre binari de cerca

La descripcions de les operacions del TAD **`BSTtree`** són les següents:

Operació	Definició
`constructor`	Construeix l'arbre buit.
`constructor còpia`	Construeix una còpia de l'arbre partir del node original rebut per paràmetre
`destructor`	Destrueix els nodes de l'arbre binari, començant pel root. *Atenció a la gestió de memòria dinàmica*
`empty`	Retorna cert si l'arbre binari està buit, fals en cas contrari
`size`	Retorna el nombre de nodes que hi ha l'arbre binari.
`height`	Retorna un enter amb l'alçada de l'arbre binari de cerca, és a dir, l'alçada del root. **Aquesta funció s'ha d'implementar de forma RECURSIVA.** Recordeu que l'alçada d'una fulla és 1.
`insert`	Afegeix un nou node a l'arbre binari de cerca. En el cas que la clau ja existeix al node, només afegeix el valor al node ja existent. Altrament, crea un nou node amb el valor inclòs.
`valuesOf`	Retorna el vector de valors (atribut `Values`) d'un node de l'arbre amb una certa `Key` passada per paràmetre
`printPreorder`	Mostra per consola les claus i el contingut de les *values* seguint un recorregut en preordre
`printInorder`	Mostra per consola les claus i el contingut de les *values* seguint un recorregut en inordre
`printPostorder`	Mostra per consola les claus i el contingut de les *values* seguint un recorregut en postordre
`getLeafNodes`	Retorna una llista amb tots els nodes fulla de l'arbre. **Aquesta funció s'ha d'implementar de forma RECURSIVA**
`printSecondLargestKey`	Retorna la segona Key més gran de l'arbre. **Aquesta funció s'ha d'implementar de forma RECURSIVA**

`mirrorTree`	Modifica l'arbre per tenir el seu arbre mirall. **Aquesta funció s'ha d'implementar de forma RECURSIVA**
`search` (protected)	Troba un element (indexat per la clau) de l'arbre binari de cerca i retorna el node si el troba, en cas contrari retorna nullptr. **Aquesta funció s'ha d'implementar de forma ITERATIVA.**

Pas 3 Programació d'un main amb un cas de prova

Per a comprovar la correcta implementació dels TADs, creeu finalment un `main.cpp` que realitzi les operacions del cas de prova que us deixem a continuació i compareu els resultats obtinguts pel vostre codi amb els resultats que us proporcionem en el cas. En aquesta prova, les claus i els valors seran de tipus enter (`int`).

Cas de Prova

```
void mainExercici1(){
    BSTtree<int, int> tree1;
    int testKeys[] = {2, 0, 8, 45, 76, 5, 3, 40};
    int testValues[] = {5, 5, 1, 88, 99, 12, 9, 11};

    for (int i = 0; i < 8 ; i++) {
        cout << "Inserta a l'arbre la key " << testKeys[i] << " amb valor " <<
testValues[i] << endl;
        tree1.insert(testKeys[i], testValues[i]);
    }

    cout << "Preorder = [";
    tree1.printPreorder();
    cout << "]" << endl;
    cout << "Inorder = [";
    tree1.printInorder();
    cout << "]" << endl;
    cout << "Postorder = [";
    tree1.printPostorder();
    cout << "]" << endl;

    tree1.printSecondLargestKey();
    cout << "FULLES De l'arbre = ";
    list<NODEtree<int,int> *> fulles = tree1.getLeafNodes();
    cout << "{ " ;
    for(NODEtree<int, int>* n: fulles){
         cout << n->getKey() << " ";
    }
    cout << "}" << endl<< endl;

    BSTtree<int, int> tree2(tree1);
```

```
    tree1.mirrorTree();
    cout << "Inorder = [";
    tree1.printInorder();
    cout << "]" << endl;
}
```

OUTPUT:

```
Inserta a l'arbre la key 2 amb valor 5
Inserta a l'arbre la key 0 amb valor 5
Inserta a l'arbre la key 8 amb valor 1
Inserta a l'arbre la key 45 amb valor 88
Inserta a l'arbre la key 76 amb valor 99
Inserta a l'arbre la key 5 amb valor 12
Inserta a l'arbre la key 3 amb valor 9
Inserta a l'arbre la key 40 amb valor 11
Preorder = [2, 0, 8, 5, 3, 45, 40, 76]
Inorder = [0, 2, 3, 5, 8, 40, 45, 76]
Postorder = [0, 3, 5, 40, 76, 45, 8, 2]
 --> Second largest key is...  45
FULLES De l'arbre = {0 3 40 76 }
Inorder = [76, 45, 40, 8, 5, 3, 2, 0]
```

Lliurament

A partir de la descripció del problema, es demana:

- Implementar els exercicis en **Visual Studio Code i C++ versió 11**. Lliurar el codi C++ corresponent als vostres exercicis en una carpeta anomenada codi, amb tot el codi dels TADs.
- No us oblideu d'afegir a la carpeta principal els fitxers de les proves: pelis-cas_de_prova.csv, pelis-gran.csv, pelis-petit.csv.

Com a màxim el dia del lliurament es penjarà en el campus virtual un fitxer comprimit **en format ZIP** amb el nom i cognoms de l'alumne, el nom del grup (A, B, C, D, E o F) i el numero de la pràctica com a nom de fitxer, **GrupA_BartSimpson_P3.zip**, on A indica grup de pràctiques A i P3 indica que és la "pràctica 3". El fitxer ZIP inclourà la carpeta codi.

Els criteris per acceptar la pràctica són:

- La pràctica ha de funcionar en la seva totalitat.

- La pràctica ha de ser orientada a objectes.
- El codi ha d'estar comentat.

IMPORTANT: La còpia de la implementació d'una pràctica implica **un zero a la nota global de pràctiques** de l'assignatura per les persones implicades (tant la que ha copiat com la que ha deixat copiar).

Planificació

Del 24 d'abril al 25 de maig 2025. Per aquesta pràctica els professors proposen la següent planificació:

- **Setmana 1** *(Classe de Laboratori 9)*
  - Implementació de la classe Peli
  - Fer una funció al main per llegir el fitxer de dades i que guardi cada línia a un objecte Peli
  - Implementació del **BSTtree**, el main amb el cas de prova inclòs, i comentar el codi
- **Setmana 2** *(Classe de Laboratori 10)*
  - Implementació de la classe **MubiesflixBST** i del main.cpp amb el menú, i comentar el codi
- **Setmana 3** *(Classe de Laboratori 11)*
  - Implementació del **HashMap**, el main, i comentar el codi
- **Setmana 4** *(Classe de Laboratori 12)*
  - Implementació de la classe **MubiesflixHASH** i del main.cpp amb el menú, i comentar el codi
- **Setmana 5** *(Classe de Laboratori 13)*
  - **PROVA DE LA PRÀCTICA 3 al laboratori.**

Recordeu que la setmana 4 s'ha de lliurar la totalitat de la pràctica (tots els exercicis al ZIP).

Lliurament: dia **25 de maig de 2025**