Pràctica 3. Arbres binaris

1. Introducció

***Objectiu**:* Familiaritzar-se amb les estructures de dades no lineals. En particular es treballarà amb diferents implementacions dels arbres binaris, analitzant les seves característiques i les seves diferències.

Temes de teoria relacionats amb la pràctica: Tema 4 Estructures no lineals: Arbres

2. Enunciat

L'aplicació que es vol implementar és una "base de dades" per guardar les puntuacions d'un conjunt de pel·lícules. Aquesta estructura haurà de mantenir informació sobre les dades de les pel·lícules i permetre accedir a aquesta informació de manera ràpida.

Per a cada pel·lícula es disposarà d'un ID únic, el títol i la seva puntuació (rating).

Un exemple de les dades podria ser:

1::Toy Story (1995)::3.87

2::Jumanji (1995)::3.40

3::Grumpier Old Men (1995)::3.16

4::Waiting to Exhale (1995)::2.38

5::Father of the Bride Part II (1995)::3.27

...

I també caldria oferir la possibilitat de poder consultar el títol i la puntuació d'una pel·lícula a partir d'un ID.

En aquesta pràctica es demana implementar aquesta funcionalitat utilitzant arbres de cerca binària.

Exercici 1. Arbre binari de cerca

Implementeu el TAD **`BinarySearchTree`** amb templates, per tal que representi un arbre binari de cerca amb una representació encadenada. L'especificació amb el **mínim** d'operacions necessàries al TAD **`BinarySearchTree`** és la següent:

- **`constructor`**: construeix l'arbre buit
- **`destructor`**: destrueix tots els elements de l'arbre binari. Atenció a la gestió de memòria dinàmica.
- **`size`**: retorna el nombre de nodes que hi ha l'arbre binari. Aquesta funció recorre els nodes per calcular quants nodes té l'arbre binari
- **`height:`** retorna un enter amb l'alçada de l'arbre binari de cerca
- **`empty`**: retorna cert si l'arbre binari està buit, fals en cas contrari
- **`root`**: retorna l'adreça del root

- **insert**: afegeix un nou node a l'arbre binari de cerca
- **search**: cerca un element a l'arbre binari de cerca, retorna cert si el troba, fals en cas contrari
- **printPreorder**: mostra el contingut l'arbre seguint un recorregut en preordre
- **printPostorder**: mostra el contingut l'arbre seguint un recorregut en postordre
- **printInorder**: mostra el contingut l'arbre seguint un recorregut en inordre

També haureu d'implementar amb templates el **TAD NodeTree,** corresponent a cada node de l'arbre. L'especificació mínima del **TAD NodeTree** és la següent:

- **constructor**: construeix el node de l'arbre binari
- **destructor**: destrueix el node
- **funció getData:** retorna l'element emmagatzemat en aquell node
- **funció setData**: permet canviar l'element emmagatzemat en un node
- **getRight**: retorna l'adreça del fill dret del node
- **getLeft**: retorna l'adreça del fill esquerra del node
- **getParent**: retorna l'adreça del pare d'un node
- **setRight**: modifica l'adreça del fill dret del node
- **setLeft**: modifica l'adreça del fill esquerra del node
- **setParent**: modifica l'adreça del pare del node
- **isRoot**: retorna cert si el node és el root de l'arbre, fals en cas contrari
- **isExternal**: retorna cert si el node és un node extern, fals en cas contrari
- **height**: retorna l'alçada d'aquell node en l'arbre
- **hasRight:** retorna cert si té fill dret.
- **hasLeft:** retorna cert si té fill esquerre.

A la memòria justifiqueu el cost computacional teòric de les funcions del TAD **BinarySearchTree** i del TAD **NodeTree**. Es poden implementar altres mètodes que siguin necessaris pel desenvolupament de la pràctica, tot i justificant a la memòria el seu ús i el seu cost computacional teòric.

Llenceu les excepcions oportunes a cada TAD. El codi s'ha de comentar obligatòriament.

Feu un mètode main que faci les operacions descrites al cas de prova 1, utilitzant l'arbre de cerca binària implementat en aquest exercici. De moment en aquest exercici provareu aquestes classes amb un Element de tipus enter (int).

Cas de Prova 1

Pas	Entrada	Sortida
1	Crear un arbre que emmagatzemi enters	Arbre binari creat
2	Inserir a l'arbre els valors d'un array anomenat testArray que contindrà : int testArray [] = {2, 0, 8, 45, 76, 5, 3, 40 };	Inserta a l'arbre element 2 Inserta a l'arbre element 0 Inserta a l'arbre element 8 Inserta a l'arbre element 45

		Inserta a l'arbre element 76 Inserta a l'arbre element 5 Inserta a l'arbre element 3 Inserta a l'arbre element 40
3	Imprimir en preordre l'array per pantalla	Preordre = {2, 0, 8, 5, 3, 45, 40, 76}
4	Imprimir en inordre l'array per pantalla	Inordre = { 0, 2, 3, 5, 8, 40, 45, 76}
5	Imprimir en postordre l'array per pantalla	Postordre = { 0, 3, 5, 40, 76, 45, 8, 2}
6	Fer el mirall de l'arbre	Arbre mirall creat
7	Imprimir en preordre el mirall de l'arbre	Preordre = {2, 8, 45, 76, 40, 5, 3, 0}
8	Eliminar l'arbre	Arbre binari destruït

Passos necessaris per dur a terme una correcta implementació:

Pas 1. Implementació del `TAD NodeTree` que serà utilitzat per l'arbre binari. A continuació es presenta la definició de l'especificació d'aquesta classe.

```
template <class Type>
class NodeTree {
        public:
                NodeTree(const Type& data);
                NodeTree(const NodeTree& orig);
                virtual ~NodeTree(); // destructor
                /*Consultors*/
                NodeTree<Type>* getright() const;
                NodeTree<Type>* getleft() const;
                NodeTree<Type>* getparent() const;
                bool hasRight() const;
                bool hasLeft() const;
                bool isRoot() const;
                bool isExternal() const;
                const Type& getElement() const;
                int getHeight() const;
                void setHeight(int h);
                /*Modificadors*/
                void setData(const Type& data);
                void setRight(NodeTree<Type>* newRight);
                void setLeft(NodeTree<Type>* newLeft);
                void setParent(NodeTree<Type>* newParent);
        private:
                NodeTree<Type>* pParent;
                NodeTree<Type>* pLeft;
                NodeTree<Type>* pRight;
                Type data;
                int height;
};
```

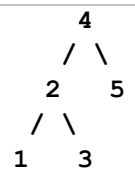
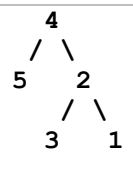
Per assegurar que la vostra implementació funcioni bé podeu implementar una breu funció al fitxer *main.cpp* que treballi amb la classe que heu implementat.

Pas 2. Implementació del TAD `BinarySearchTree`. A continuació es presenta la definició de l'especificació d'aquesta classe.

```
template <class Type>
class BinarySearchTree{
       public:
                /*Constructors i Destructors*/
                BinarySearchTree();
                BinarySearchTree(const BinarySearchTree& orig);
                virtual ~BinarySearchTree();
                /*Consultors*/
                int size() const;
                bool isEmpty() const;
                NodeTree<Type>* root();
                bool search(const Type& element);
                void printInorder() const;
                void printPreorder() const;
                void printPostorder() const;
                int getHeight();
                /*Modificadors*/
                void insert(const Type& element);
       private:
                void postDelete(NodeTree<Type>* p);
                int size(NodeTree<Type>* p) const;
                void printPreorder(NodeTree<Type>* p) const;
                void printPostorder(NodeTree<Type>* p) const;
                void printInorder(NodeTree<Type>* p) const;
                int getHeight(NodeTree<Type>* p);
                /*Atributs*/
                NodeTree<Type>* pRoot;
};
```

Tal i com podeu veure a la definició anterior es demana també que implementeu tres recorreguts: inOrder, preOrder i postOrder.

Pas 3. Per acabar la implementació d'aquest primer exercici heu d'afegir l'operació mirall sobre l'arbre. L'operació mirall consisteix en intercanviar els fills de cada node intern. De tal manera que el fil dret passi a ser fill esquerra i l'esquerra passi a ser fill dret. A continuació teniu un exemple, on l'arbre A, el seu arbre mirall serà l'arbre B.

Arbre A	Arbre B
	

Exercici 2. Cercador de pel·lícules amb arbres de cerca binària

En aquest exercici cal implementar un cercador de pel·lícules utilitzant un arbre de cerca binària. Per a resoldre el problema seguiu els passos següents:

Pas 1. Especificació del Cercador de pel·lícules

Implementeu el `TAD` **`Movie`** que contingui totes les dades necessàries per emmagatzemar la informació. L'especificació mínima del TAD és la següent:

- **`constructor`**: construeix una Movie amb totes les dades.
- **`funcions consultores:`** una o més funcions per consultar cada una de les dades associades a la Movie.
- **`funcions modificadores`**: una o més funcions per modificar cada una de les dades associades a la Movie.
- **`Funció 'toString'`**: retorna un string amb tot el contingut de la Movie.

El TAD Movie serà, en aquest exercici, el contingut dels nodes de l'arbre de cerca binària.

Escriviu l'especificació d'una classe anomenada **`BSTMovieFinder`** que es cridarà des del main i ha de realitzar les següents operacions:

1. `appendMovies(filename)`: Aquest mètode rep el nom d'un fitxer i emmagatzema el seu contingut a una estructura de dades.
2. `insertMovie(movieID, title, rating)`: Aquest mètode rep les dades d'una pel·lícula i fa la inserció a l'estructura de dades corresponent.
3. `showMovie(movieID)`: Aquest mètode rep un identificador de pel·lícula i retorna un sol string amb les dades associades a la Movie.
4. `findMovie(movieID)`: Aquest mètode no ha de retornar un string, retorna l'objecte 'movie'.
5. `findRatingMovie(movieID)`: Aquest mètode ha de retornar el rating de la pel·lícula.

Pas 2. Implementació del Cercador de contactes amb un arbre

Implementeu l'especificació anterior, **`BSTMovieFinder`**, amb un **`BinarySearchTree`**. En aquest arbre els **`Nodes`** contindran elements consistents en una clau `(key)`, que servirà per ordenar-los dins l'arbre i un valor associat (`value`), corresponent a un atribut de tipus genèric. Si ho implementeu amb templates podeu donar per suposat que la `key` serà sempre un ID. En comptes de getElement(), definirem dos mètodes, el mètode `getKey()` per a que retorni la `Key` i el mètode `getValue()` que retornarà la Movie. De la mateixa manera substituirem el mètode `setElement` pels mètodes `setKey` i `setValue`. Aquest canvi potser us farà canviar algun altre mètode.

En el nostre exercici la clau serà el movieID, i el valor l'objecte Movie.

L'especificació usant templates d'aquests mètodes i atributs seria (noteu que aquesta especificació és incompleta i que falten els altres mètodes i atributs):

```
template <class Type>
class NodeTree {
public:
    /*Consultors*/
    const int getKey() const;
    const Type& getValue() const;

    /*Modificadors*/
```

```
    void setValue(const Type& newValue);
    void setKey(std::int newKey);
private:
    Type value;
    int key;
};
```

Pas 3. Programació del main

A l'algorisme principal s'han de realitzar les següents accions:

1. Demanar a l'usuari el nom del fitxer que volem utilitzar amb la pregunta: "Quin fitxer vols (P/G)?" (per petit o gran). Al campus virtual trobareu dos fitxers de text que podeu utilitzar per fer les proves: *movie_rating_small.txt* i *movie_rating.txt*. Aquests fitxers s'hauran d'incorporar a la carpeta del projecte i incloure'ls com a recursos en el projecte. Avaluar el temps d'inserció i mostrar-lo al final de tot. En aquest apartat, a més a més de demanar el nom del fitxer i llegir-lo, cal:
    - Crear un objecte `BSTMovieFinder`.
    - Inicialitzar el `BSTMovieFinder` a partir del contingut del fitxer.
    - Mostrar el temps de creació del `BSTMovieFinder`.

2. Mostrar l'arbre segons l'ID en ordre creixent. Fer un comptador que demani confirmació cada 40 pel·lícules per seguir mostrant l'arbre.
3. Llegir el fitxer *cercaPelicules.txt* que us proporcionem i per a cada movieID contingut a directori, fer una cerca en l'arbre emmagatzemat a `BSTMovieFinder`. Avaluar el temps de cerca de tots els elements de *cercaPelicules.txt* i comptar el nombre d'elements que estan en el `BSTMovieFinder`. Mostrar les dues dades per pantalla.
4. Visualitzar per pantalla la profunditat de l'arbre.

Implementeu aquestes funcionalitats en forma d'un menú al main que permeti realitzar les 4 accions descrites anteriorment i l'opció 5 serà sortir del menú.

Atenció: En aquest exercici sols es demana fer amb templates els TAD `NodeTree` i `BinarySearchTree`. El TAD `Movie` o `BSTMovieFinder` no s'han de fer amb templates.

Exercici 3. Arbres binaris balancejats

Implementeu el TAD **`BalancedBST`** que representi un arbre binari balancejat. La funcionalitat d'aquest TAD ha de ser exactament la mateixa que el TAD **`BinarySearchTree`** de l'exercici 1.

Un cop implementat el `BalancedBST` heu d'implementar les següents operacions:

- Implementació d'una funció recursiva que retorni el títol de pel·lícula més llarg del fitxer movie_rating (el que tingueu carregat).
- Implementació d'una funció recursiva que retorni el valor de puntuació més alt i més baix que s'ha donat a una pel·lícula i mostreu per consola la/les pel·lícules que tenen aquest valor més baix de valoració i la/les pel·lícules que tenen el valor de puntuació més alt.

Repetir els experiments duts a terme a l'exercici anterior. Expliqueu a la memòria les similituds i diferències en la implementació d'aquest TAD **`BalancedBST`** respecte al TAD **`BinarySearchTree`**. Detalleu quin és el cost computacional teòric de cadascuna de les operacions del TAD.

Exercici 4. Cercador de pel·lícules amb arbres binaris balancejats

Implementeu una nova versió del cercador de pel·lícules amb un arbre balancejat. En aquest cas la classe té un objecte de tipus `BalancedBST`. Penseu que algunes de les funcions associades a l'extracció d'informació dels fitxers les podreu reaprofitar; us animem a fer-ho ja que és una bona pràctica reusar codi, però us demanem que ho indiqueu en els comentaris del programa.

Exercici 5. Avaluació de les estructures

Feu una avaluació del rendiment experimentat de les dues implementacions anteriors (arbres de cerca binària i arbres binaris balancejats), i raoneu les diferències: compteu el temps d'accés (cerca) per dos fitxers de diferent grandària (al campus virtual trobareu un fitxer gran i un de petit); compteu el temps de generació de l'estructura per les dos llistes de pel·lícules de diferent grandària. Féu les dues proves en el mateix ordinador i en condicions similars. Raoneu els resultats de temps obtinguts.

Indiqueu quin és el cost computacional teòric de les operacions d'inserció i cerca en els dos arbres implementats (exercici 1 i exercici 3).

Memòria

La memòria ha de recollir tots els punts de valoració i raonament indicats anteriorment.

3. Lliurament

A partir de la descripció del problema, es demana:

- Escriure una memòria explicativa que inclogui els següents punts:
  - Portada amb: número de pràctica i títol complet, nom, cognom, NIUB de cada membre del grup, professor/a de pràctiques, número de parella assignat i grup de pràctiques al que assistiu (A, B, C, F)
  - Comentaris de cada exercici: observacions, decisions preses i resposta a les preguntes plantejades, si n'hi ha.

  La memòria s'entregarà en un document **memoriaPX.pdf** on X és el número de la pràctica.
- Implementar els exercicis en C++ . Lliurar el codi C++ corresponent als vostres exercicis en una carpeta anomenada codi, amb una subcarpeta per a cada exercici.

Com a màxim el dia del lliurament es penjarà en el campus virtual un fitxer comprimit **en format ZIP** amb el nom dels dos membres del grup, el nom del grup (A, B, C o F), el número de parella (ParX) i el numero de la pràctica com a nom de fitxer, **GrupA_Par2_BartSimpsonLisaSimpson_P3.zip**, on Par2 indica que és la "parella 2" i P3 indica que és la "pràctica 3". El fitxer ZIP inclourà: la memòria i la carpeta amb tot el 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 parelles implicades (tant la que ha copiat com la que ha deixat copiar).

4. Planificació

Lab4 i Lab 5. Del 16 d'abril al 6 de maig 2017

Per aquesta pràctica els professors proposen la següent planificació:

- **Setmana 1** *(Classe de Laboratori 4)*
  - Implementació de l'exercici 1 i comentar el codi
- **Setmana 2** *(No hi ha classe presencial)*
  - Implementació de l'exercici 2 i comentar el codi
- **Setmana 3** *(Classe de Laboratori 5)*
  - Implementació de l'exercici 3 i comentar el codi
- **Setmana 4** *(No hi ha classe presencial)*
  - Implementació de l'exercici 4 i comentar el codi
  - Realització de l'exercici 5
  - Realització de la memòria completa de la pràctica

La setmana 2 i la setmana 4 s'obrirà una tasca al campus virtual per dipositar la feina feta fins aquell moment. Recordeu que la setmana 4 s'ha de lliurar la totalitat de la pràctica (tots els exercicis)

Lliurament: dia 6 de maig de 2018