

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

Es demana implementar una aplicació per guardar les muntanyes del món. Per a fer-ho, definirem una estructura de dades que haurà de mantenir informació sobre les dades de les muntanyes i permetre accedir a aquesta informació de manera ràpida.

Per a cada muntanya es disposarà d'un **ID** únic, el seu **nom**, i la seva **alçada** (m), cada camp separat amb els caràcters ::, vegeu a continuació un exemple de quines podrien ser les dades:

```
1::Makalu::8481
2::Everest::8848
3::Annapurna::8091
4::Nanga Parbat::8125
5::K2::8621
...
```

Caldrà poder consultar les seves dades (nom i alçada) 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 **Binary Search Tree (BST)** 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 **BST** és la següent:

Operació	Definició
constructor	Construeix l'arbre buit.
destructor	Destruïx tots els elements de l'arbre binari. <i>Atenció a la gestió de memòria dinàmica.</i>
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.
showPreorder	Mostra per consola el contingut l'arbre seguint un recorregut en preordre.
showPostorder	Mostra per consola el contingut l'arbre seguint un recorregut en postordre.
showInorder	Mostra per consola el contingut l'arbre seguint un recorregut en inordre.
showLeafNodes	Mostra per consola les fulles de l'arbre.

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

Estructura de Dades: Pràctica 3. Arbres binaris

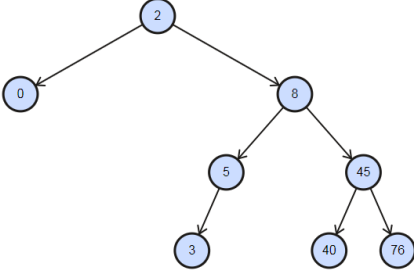
Operació	Definició
constructor	construeix el node de l'arbre binari
destructor	destrueix el node
getElement	retorna l'element emmagatzemat en aquell node
setElement	permet canviar l'element emmagatzemat en un node
right	retorna l'adreça del fill dret del node
left	retorna l'adreça del fill esquerra del node
parent	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

Al codi (en el fitxer BST.h i NodeTree.h) justifiqueu el cost computacional teòric de les funcions del TAD **BST** i del TAD **NodeTree**. Es poden implementar altres mètodes que siguin necessaris pel desenvolupament de la pràctica, tot i justificant al codi 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 elements de tipus enter (int).

Cas de Prova

Pas	Entrada	Sortida
1	Crear un arbre que emmagatzemi enters	Arbre binari creat
2	<p>Inserir a l'arbre els valors d'un array anomenat testArray que contindrà :</p> <pre>int testArray [] = {2, 0, 8, 45, 76, 5, 3, 40};</pre> 	<p>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</p>
3	Mostrar en preordre l'arbre per pantalla	Preordre = {2, 0, 8, 5, 3, 45, 40, 76}
4	Mostrar en inordre l'arbre per pantalla	Inordre = {0, 2, 3, 5, 8, 40, 45, 76}
5	Mostrar en postordre l'arbre per pantalla	Postordre = {0, 3, 5, 40, 76, 45, 8, 2}
6	Mostrar les fulles de l'arbre	Fulles = {0, 3, 40, 76}
7	Eliminar l'arbre	<p>Destruïnt arbre binari Eliminant node 0 Eliminant node 3 Eliminant node 5 Eliminant node 40, Arbre binari destruït</p>

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>* right() const;
    NodeTree<Type>* left() const;
    NodeTree<Type>* parent() const;
    bool hasRight() const;
    bool hasLeft() const;
    bool isRoot() const;
    bool isExternal() const;
    const Type& getElement() const;

    /*Modificadors*/
    void setElement(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 element;
};
```

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 **BST**. A continuació es presenta la definició de l'especificació d'aquesta classe.

```
template <class Type>
class BST{
public:
    /*Constructors i Destructors*/
    BST();
    BST(const BST& orig);
    virtual ~BST();
    /*Consultors*/
    int size() const;
    bool isEmpty() const;
    NodeTree<Type>* root();
    bool search(const Type& element);
    void showInorder() const;
    void showPreorder() const;
    void showPostorder() const;
    void showLeafNodes() const;
    /*Modificadors*/
    void insert(const Type& element);

private:
    int size(NodeTree<Type>* p) const;
    void showPreorder(NodeTree<Type>* p) const;
    void showPostorder(NodeTree<Type>* p) const;
    void showInorder(NodeTree<Type>* p) const;

    /*Atributs*/
    NodeTree<Type>* pRoot;
};
```

Tal i com podeu veure a la definició anterior es demana també que implementeu tres recorreguts: *showInorder*, *showPreorder* i *showPostorder*. Decidiu vosaltres la capçalera de la funció privada per calcular *showLeafNodes*.

Exercici 2. Cercador de muntanyes amb arbres de cerca binària

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

Pas 1. Especificació del Cercador de muntanyes

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

Operació	Definició
constructor	construeix una Mountain amb totes les dades.
funcions consultores	una o més funcions per consultar cada una de les dades associades a la Mountain.
funcions modificadores	una o més funcions per modificar cada una de les dades associades a la Mountain.
toString	retorna un string amb tot el contingut de la Mountain.

El TAD **Mountain** serà, en aquest exercici, el contingut dels nodes de l'arbre de cerca binària. Escriviu l'especificació d'una classe anomenada **BSTMountainFinder** que es cridarà des del main i ha de realitzar les següents operacions:

Operació	Definició
appendMountains(filename)	Aquest mètode rep el nom d'un fitxer i emmagatzema el seu contingut a una estructura de dades.
insertMountain(mountainID, name, height)	Aquest mètode rep les dades d'una muntanya i fa la inserció a l'estructura de dades corresponent.
showMountain(mountainID)	Aquest mètode rep un identificador de muntanya i retorna un sol string amb les dades associades a la muntanya.
findMountain(mountainID)	Rep un identificador de muntanya i retorna l'objecte 'Mountain'. Compte!, no ha de retornar un string.

Pas 2. Implementació del Cercador de muntanyes amb un arbre

Implementeu l'especificació anterior, **BSTMountainFinder**, amb un **BST**. 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 *Mountain*. De la mateixa manera substituïrem 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 `mountainID`, i el valor l'objecte *Mountain*.

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: *Mountain_list_small.txt* i *Mountain_list.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 `BSTMountainFinder`.
 - Inicialitzar el `BSTMountainFinder` a partir del contingut del fitxer.
 - Mostrar el temps de creació del `BSTMountainFinder`.
2. Mostrar l'arbre segons l'ID en ordre creixent. Fer un comptador que demani confirmació cada 40 muntanyes per seguir mostrant l'arbre.
3. Llegir el fitxer *cercaMuntanyes.txt* que us proporcionem i per a cada `MountainID` contingut a directori, fer una cerca en l'arbre emmagatzemat a `BSTMountainFinder`. Avaluar el temps de cerca de tots els elements de *cercaMuntanyes.txt* i comptar el nombre d'elements que estan en el `BSTMountainFinder`. 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 `BST`. El `TAD Mountain` o `BSTMountainFinder` 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 **BST** de l'exercici 1.

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

- Implementació d'una funció recursiva que retorni el nom de muntanya més llarg del fitxer *Mountain_list* (el que tingueu carregat).
- Implementació d'una funció recursiva que retorni el valor d'alçada màxim i mínim i els mostri per consola.

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

Exercici 4. Cercador de muntanyes amb arbres binaris balancejats

Implementeu una nova versió del cercador de muntanyes 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 reutilitzar 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 muntanyes de diferent grandària. Féu les dues proves en el mateix ordinador i en condicions similars. Raoneu els resultats de temps obtinguts i poseu una taula com la de l'exemple al main de l'exercici 4.

	Inserció Fitxer SMALL	Inserció Fitxer LLARG	Cerca Fitxer SMALL	Cerca Fitxer LLARG
BST				
BalancedBST				

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

3. LLIURAMENT

A partir de la descripció del problema, es demana:

- Implementar els exercicis en NetBeans 8.2 i C++ versió 11. Lliurar el codi C++ corresponent als vostres exercicis en una carpeta anomenada codi, amb un projecte NetBeans 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 i cognoms de l'alumne, el nom del grup (A, B, C 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 amb tots els projectes Netbeans.

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).

4. PLANIFICACIÓ

Lab6 i Lab 7. Del 8 d'abril al 5 de maig 2019

Per aquesta pràctica els professors proposen la següent planificació:

- **Setmana 1** (*Classe de Laboratori 6*)
 - Implementació de l'exercici 1, exercici 2 i comentar el codi
- **Setmana 2** (*Classe de Laboratori 7*)
 - Implementació de l'exercici 3, exercici 4, exercici 5 i comentar el codi

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

Lliurament: dia 5 de maig de 2019